Space Domain Based Algorithms for Automatic Modeling and Inversion of Gravity Anomalies of Sedimentary Basins by Means of Growing Bodies and Exponential Density Function and A Semiautomatic Method for Gravity Modeling of Complex Geologic Structures

A Thesis submitted during August 2014 to the University of Hyderabad in partial fulfillment of the award of a Ph.D. degree in University Centre for Earth and Space Sciences

by

Ramamma Batta

University Centre for Earth and Space Sciences
School of Physics

University of Hyderabad
(P.O.) Central University, Gachibowli
Hyderabad – 500 046
India

CERTIFICATE

This is to certify that the thesis entitled "Space Domain Based Algorithms for Automatic Modeling and Inversion of Gravity Anomalies of Sedimentary Basins by Means of Growing Bodies and Exponential Density Function and a Semiautomatic Method for Gravity Modeling of Complex Geologic Structures" submitted by Ms. Ramamma Batta, bearing regd. No 12ESPE03, in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in Geophysics is a bonafide work carried out by her under my supervision and guidance which is a plagiarism free thesis.

The thesis has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Date: August 05, 2014

(V. Chakravarthi)

Supervisor

Head of the Centre

Dr. V. Chakravarthi
Head
Centre for Earth and Space Sciences
University of Hyderabad
Hyderabad - 500 046.

Dean of the School

सांकाय अध्यक्ष / Dean
भौतिकी संकाय / School of Physics
हैदराबाद विश्वविद्यालय
UNIVERSITY OF HYDERABAD
हैदराबाद / HYDERABAD-500 046. भारत / INDIA.

Declaration

I, Ramamma Batta, hereby declare that this thesis entitled "Space Domain Based Algorithms for Automatic Modeling and Inversion of Gravity Anomalies of Sedimentary Basins by Means of Growing Bodies and Exponential Density Function and a Semiautomatic Method for Gravity Modeling of Complex Geologic Structures" submitted by me under the guidance of Dr. V. Chakravarthi, Associate Professor is a bonafide research work which is also free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma. I hereby agree that my thesis can be deposited in Shodganga/INFLIBNET.

Date: August 05, 2014

Name: Ramamma Batta

B. Ramamma

Signature of the student

Regd. No: 12ESPE03

To my beloved parents….

Acknowledgements

It is with a deep sense of pleasure and indebtedness that I thank my research supervisor Dr. V. Chakravarthi, Associate Professor and Head, University Centre for Earth and Space Sciences (UCESS), University of Hyderabad for his constant encouragement and expert guidance throughout the progress of my research work. My close association with him for the last over a couple of years is one of the unforgettable memories in my life.

I express my deep sense of gratitude to all Doctoral Research Committee members, especially Professor B. V. S. Murthy, for giving valuable advices and many scientific inputs throughout my research work.

It is ethical in my part to express my sincere gratitude to the University of Hyderabad for providing me this golden opportunity to pursue research work leading to the award of Ph.D Degree. University Grants Commission (UGC), New Delhi is profusely acknowledged for extending financial support in the form of Rajiv Gandhi National Fellowship (RGNF).

I express my heartfelt respects to Professor S. Chaturvedi, Dean, School of Physics; Dr. P. S. Roy, DST Chair Professor; Dr. S. Sri Lakshmi, Assistant Professor; Sri T. Suryanarayana of UCESS for their unstinted support and encouragement. I also take this opportunity to thank previous director, UCESS, for providing the facilities to carry out my research work.

I would like to acknowledge Sri V. Kameshwara Rao; Dr. Rajitha, Sravanthi, Sandeep, Anjaiah, Pawan Kumar Gautam, Shiva Kumar,

Bhavani, Reshma, Pavani, Sudha Rani and Sunitha for their benevolent attitude and cooperation during the course of my research work.

All supporting staff, UCESS, have helped me in many ways. I thank one and all.

Mr. Pamod Kumar, fellow research scholar, has helped me in the final compilation of my thesis for which I am thankful to him.

I owe my affection to my family friends Lakshmi, Sasi, and Sowjanya with whom I shared all my happiness and sorrows. Their association is indeed a fabulous gift given to me by the Almighty.

It is the happiest moment for me to express my deep sense of respects and regards to my parents who gave me many things. Words fall insufficient to express my feelings. All my family members have extended their cordial support and encouragement all through my academic career, for which I am ever grateful.

***

Contents

Preface

The economic growth of a nation is largely dependent on its mineral wealth. Geophysical methods particularly when integrated play an indispensable and important role in the exploration of such precious natural resources. A wide spectrum of geophysical methods is in vogue for exploration depending on the nature and type of the resource. The contrast in physical properties such as the density, magnetic susceptibility, electrical resistivity etc. between the source of interest and the surroundings pave the way for measurement of the corresponding anomalous/physical fields. The measured physical fields are subsequently corrected for the unwarranted effects that arise due to the earth's topography, the nature of the subsurface etc. before being subjected to any meaningful interpretation.

The utility or success of any geophysical method largely depends on a reasonable and reliable interpretation of the data acquired in the field. Even though, several mathematical and statistical tools and elegant software packages for processing and interpretation of geophysical data are in vogue, they depend on many approximations or assumptions.

In this direction, it may be emphasized in unequivocal terms that the gravity method, which is the subject of this thesis, depends exclusively on dimensions and depth of the target and also on considerable density contrast between the source/sources of interest and

surrounding formations, which generates a favorable gravity anomaly. It is well known that the density of sedimentary rocks varies with depth in the subsurface of the earth due to factors such as compaction, stratigraphic layering, facies change, cementation, diagenesis, tectonic history etc. Although, it is often possible to simulate the density variation of sedimentary rocks with depth by an exponential density function, it becomes an uphill task to derive closed form analytical expressions for gravity anomalies of geophysical geometries in space domain. Thus it is imperative to develop appropriate mathematical tools in the space domain not only to realize forward gravity modeling of geological sources but also to estimate the source parameters making use of the exponential density function.

In the thesis, both analytical and numerical methods have been used in the space domain to derive expressions to calculate the gravity anomalies of geophysical geometries using Exponential Density Function (EDF). Based on this, modeling and inversion techniques are developed to interpret gravity anomalies due to 2-D and 2.5-D sedimentary basins besides relevant GUI based software. Furthermore, a new technique and a code in JAVA are presented to model the gravity anomalies of multiple strike-limited geologic sources having arbitrary density contrast depth variations. The validity and applications of the techniques developed are illustrated with synthetic and real field data analysis. The results are highly encouraging and thereby substantiating the validity of proposed techniques.

The thesis under study is organized into five chapters as detailed here under.

In chapter-I, general introduction, a brief account of the earlier work on modeling and inversion techniques are discussed. Although the terms “automatic modeling” and “inversion” are used more or less synonymously to refer to various interpretation strategies of gravity anomalies, criteria has been formulated and followed to design mathematical tools for automatic modeling and inversion of gravity anomalies. In all the inversion schemes as presented in the thesis, the ridge regression algorithm is used to estimate the source parameters.

The principles of automatic modeling and inversion are used to develop two independent techniques in the space domain in Chapter II to interpret gravity anomalies due to 2-D sedimentary basins, in which the density contrast varies exponentially with depth. Expression for the gravity anomaly of a building block is derived in the space domain using the exponential density function by means of performing analytical integration of the gravity contribution of a 2-D element within the bounds of the block in xy-plane and then numerical integration over its depth extent. In both automatic modeling and inversion, initial depth estimates of a sedimentary basin are computed automatically and subsequently improved in an iterative approach within the specified convergence criteria. Based on the algorithms of automatic modeling and inversion, a GUI based software, MOD2DSD, coded in JAVA is developed. This code, works on Model-View-Controller (MVC) pattern, reads residual gravity anomalies of a sedimentary basin and estimates basement depths at plurality of observations on the profile based on the principles of either modeling or inversion as specified by the user. Besides generating output in both ASCII and graphical forms, the software displays (i) the changes in

the depth structure, (ii) nature of fit between the observed and modeled gravity anomalies, (iii) changes in misfit, and (iv) variation of density contrast with depth against iteration in animated forms. The efficiency of modeling and inversion is illustrated in each case with the gravity anomalies of a synthetic model of a sedimentary basin using both Exponential Density Function (EDF) and Uniform Density Function (UDF). The automatic modeling and inversion techniques when implemented on the gravity anomalies of the San Jacinto graben, California with the derived EDF yield a geologically plausible model, while rejecting the UDF.

Two interpretation methods are developed in the space domain in Chapter- III based on the principles of automatic modeling and inversion along with a GUI based JAVA code, MODTOHAFSD, to analyze the gravity anomalies of strike limited sedimentary basins (2.5-D) using a prescribed EDF. Forward gravity modeling of sedimentary basins is realized in the space domain with appropriate analytical and numerical approaches. The effect of profile offset on the magnitude of gravity anomaly of a 2.5-D strike sedimentary basin is discussed in detail and concluded that the offset parameter plays a vital role in the interpretation of gravity anomalies. The MVC based software, MODTOHAFSD, which is developed based on the algorithms of automatic modeling and inversion reads the Bouguer gravity anomalies, constructs/modifies regional gravity background in an interactive approach, estimates residual gravity anomalies and performs automatic modeling or inversion based on user specification for basement topography. In addition to generating output in ASCII and graphical forms, the software displays the animated versions of model space improvement and corresponding changes in model gravity

response with the iteration number. The efficiency of modeling and inversion are demonstrated with the gravity anomalies of a synthetic model of a 2.5-D strike sedimentary basin. The modeling and inversion of gravity anomalies over the Chintalpudi sub-basin in India with the simulated EDF result reliable interpretations while rejecting UDF.

In chapter-IV, a semiautomatic modeling technique coupled with a GUI based JAVA code, FORMODTOHAF, is developed to model the gravity anomalies of multiple geological sources having arbitrary strike lengths, thicknesses and density contrast variations (both with depth and lateral position). Forward modeling is realized in the space domain with a suitable expression for the gravity anomaly. The effects of profile azimuth and strike length of the structure on the magnitude of gravity anomaly due to a 2.5-D strike prismatic structure are demonstrated. The significance of profile azimuth on the characteristics of gravity anomaly is studied in detail over a synthetic model of 2.5-D sedimentary basin filled with different density formations having different thicknesses. It is concluded that the profile length should be appropriately chosen in case the profile along which the interpretation needs to be performed fails to run transverse to the strike of the anomalous source/s. In such a case the profile length must be chosen such that the projected component of the profile length on any profile that runs parallel to the x-axis should cover the lateral dimensions of the target/s. For the real field example, the gravity anomalies are modeled for a deep crustal structure of the Churchill and Superior provinces in the Canadian Shield by assigning large strike lengths to the Churchill and Superior blocks and finite strike lengths to the Cape Smith fold belt associated with a deep seated suture

zone. Modeling of the field anomalies by the present method reveals that i) the Churchill block is much thicker and ii) the western boundary of the base of the fold belt within the Superior block is shallower than reported in previous studies.

Based on the automatic modeling and inversion of gravity anomalies of 2-D and 2.5-D sedimentary basins with EDF, it is concluded that the results are highly reliable in comparison with the assumed parameters in case of synthetic examples and with the available/drilling depths in the case of field examples. On the other hand, almost in all cases, when modeling and inversion implemented with UDF results inaccurate depth structures implying that UDF is inappropriate in such applications. Furthermore, the advantage of the semiautomatic modeling technique is that it can be used to model the gravity anomalies of a variety of geological sources irrespective of their size, shape and density contrast.

The novelty of the 2.5-D automatic modeling and inversion techniques and the semiautomatic modeling is that these algorithms are fairly and equally applicable even when the profile along which the interpretation is intended fails to bisect the strike lengths of the anomalous sources, which is yet another significant contribution from the thesis.

Chapter-V summarizes the overall conclusions from the thesis and scope for future research.

List of Figures

		(b) Modeled deep crustal structure by the present method (Chakravarthi and Ramamma, 2013d). Density contrasts (g/cm^3) assigned to different blocks of the two provinces by Mukhopadhyay and Gibb (1981) are shown along with their interpreted model.	

List of Tables

List of research publications (SCI)

Chakravarthi. V., **Ramamma, B.**, and Venkat Reddy, T., 2013a, Gravity anomaly modeling of sedimentary basins by means of multiple structures and exponential density contrast-depth variations: A space domain approach, Journal of the Geological Society of India, 82, 561-569.

Chakravarthi. V., **Ramamma, B.**, 2013b, A space domain based gravity inversion to trace discontinuous basement interfaces above which the density contrast decreases exponentially with depth, Acta Geophysica (in press).

Chakravarthi. V., Rajeswara Sastry, S., and **Ramamma, B.**, 2013c, MODTOHAFSD – A GUI based JAVA code for gravity analysis of strike limited sedimentary basins by means of growing bodies with exponential density contrast – depth variation: A space domain approach, Computers & Geosciences, 56, 131-141.

Chakravarthi. V., and **Ramamma, B.**, 2013d, Gravity anomaly modelling of multiple geological sources having differing strike lengths and arbitrary density contrast variations, Near Surface Geophysics, 11, 363-370.

CHAPTER
ONE

Introduction

1.1 General

The enhanced expansion in the demand for oil and gas and other natural resources of all kinds led to the development of many geophysical techniques of ever increasing sensitivity for the detection and mapping of unseen deposits and structures. In this direction, some of the geophysical methods like gravity, magnetic, electrical, electromagnetic, seismic etc. play an important and indispensable role in comprehending the various intricacies of the subsurface of the earth. Further, the advent and advancement of computer hard and softwares besides a sea change in geophysical instrumentation, the task of data acquisition, processing and interpretation become elegant and efficient besides being effective.

Despite the fact that the geophysical industry witnessed amazingly an all round development, spatial location of subsurface targets and geological translation of geophysical measurements/interpretation poses a formidable challenge to the practicing geophysicists and at times continues to be elusive. Deducing some aspects of the earth's subsurface structure on the basis of measurements taken at the surface is invariably an inverse

problem. Generally, such inverse problems suffer from an inherent ambiguity or non-uniqueness either at interpretation stage or in the conclusions. The degree of uncertainty in geophysical interpretation can often be reduced to an acceptable level by integrating other geophysical field measurements to solve such problems; however, the problem of inherent ambiguity cannot be circumvented (Chakravarthi et al., 2007). It is true that significant differences from an actual subsurface geological situation may give rise to insignificant or immeasurably small differences in the quantities actually measured during a geophysical survey. Thus, ambiguity arises because many different geological configurations could reproduce the observed measurements. This basic limitation results from the inevitable fact that geophysics attempts to solve a cumbersome inverse problems.

It is to be realized that experimentally derived quantities are never exactly determined and experimental error adds a further degree of indeterminacy to that caused by the incompleteness of the field data and the ambiguity associated with the inverse problem.

Generally, a unique solution can't be found from a set of field measurements, geophysical interpretation is concerned either to determine properties of the subsurface that all possible solutions share or to introduce assumptions to restrict the number of admissible solutions.

In spite of the inherent problems, however, geophysical methods are inevitable tools for the investigation of subsurface geology and occupy a key role in exploration programs for geological resources. The gravity

method plays a significant role in mineral and hydrocarbon explorations and some mathematical concepts of gravity field are briefed hereunder.

A body rotating with the earth experiences the gravitational force of the masses of the earth, other celestial bodies, as well as the centrifugal force due to the earth's rotation. The resultant is the force of gravity. If the effects of centrifugal force and that of celestial bodies are removed from gravity, the left out component is gravitation. Newton's law of gravitation states that the force of attraction between two point masses m_1 and m_2 is directly proportional to the product of their masses and inversely to the square of the distance between the centers of respective masses and is given as

$$F = \frac{-Gm_1m_2}{r^2}\frac{k}{r}, \tag{1.1}$$

where, F is the force on m_2, r is the distance between m_1 and m_2 and G is the universal gravitational constant. The minus sign represents that the force is always attractive. By setting the mass at the attracted point to unity, the above equation transforms into the gravitational acceleration

$$g = \frac{-Gm}{r^2}\frac{k}{r}. \tag{1.2}$$

The vector k may be expressed by the position vectors r and r' in the Cartesian coordinate system as

$$k = r - r', r^T = (X, Y, Z), r'^T = (X', Y', Z'), \tag{1.3}$$

with the magnitude of

$$k = \sqrt{(X - X')^2 + (Y - Y')^2 + (Z - Z')^2}. \quad (1.4)$$

However, the earth is composed of an infinite number of differential mass elements, dm. The gravitation on the unit mass results from the integral over the individual contributions. The equation for gravitational acceleration then takes the form (Moritz, 1980; Chakravarthi, 2011)

$$g = -G \iiint \frac{r - r'}{|r - r'|^3} dm. \quad (1.5)$$

The mass element dm can be expressed as

$$dm = \rho du, \quad (1.6)$$

where, ρ is the density of volume element, du.

The representation of gravitational acceleration and related computations are simplified if the scalar quantity (potential) is used. Because the gravitational field is invariant to rotations

$$curl\ g = 0. \quad (1.7)$$

The vector g may be represented as the gradient of a potential U (Sigl, 1985),

$$g = -grad\ U. \quad (1.8)$$

For a point m,

$$U = \frac{GM}{r}, with\ \lim_{r \to \infty} U. \quad (1.9)$$

For the earth,

$$U = G\iiint \frac{dm}{r} = G\iiint \frac{\rho}{r}\,du\,,\lim_{r\to\infty} U. \tag{1.10}$$

The potential indicates the work that must be done by gravitation in order to move the unit mass from infinity to the point under consideration. From the above relations, it is clear that once the density function ρ was known for the earth, the gravitation can be calculated as a function of position.

Gravity potential, U, satisfies Laplace's equation (Murthy,1998), i.e.,

$$\nabla^2 U = 0, \tag{1.11}$$

at all points unoccupied by matter. All the space derivatives of this potential also satisfy Laplace's equation. However, at points occupied by matter, gravity potential satisfy Poisson's equation

$$\nabla^2 U = -4\pi G d, \tag{1.12}$$

where, d is the density at the point under consideration. Green's theorem connecting volume and surface integrations of potential fields has many important applications. If U and U' are two potential functions, whose first derivatives are finite, continuous, and single valued throughout a volume, v, bounded by a closed surface, s', then

$$\int_v (U\nabla^2 U' - U'\nabla^2 U)dv = \int_{s'} \left[U\frac{\partial U'}{\partial n} - U'\frac{\partial U}{\partial n}\right] ds', \tag{1.13}$$

where, n is the direction of the outward drawn normal to the surface. The theorem is independent of the size or shape of the surface or the nature of

distribution of masses causing U and U'. If U' is assumed to be constant everywhere, and U is the Newtonian potential due to masses distributed both inside and outside the closed surface, s', then

$$\nabla^2 U' = 0, \quad \frac{\partial U'}{\partial n} = 0. \tag{1.14}$$

Equation (1.13) then takes the form

$$\int_v \nabla^2 U dv = \int_{s'} \frac{\partial U}{\partial n} ds'. \tag{1.15}$$

If U_{in} and U_{out} are, respectively, the potentials due to masses inside and outsides, then U_{out} obeys Laplace's equation, while U_{in} satisfies Poisson's equation, i.e,

$$\int_{s'} \frac{\partial U}{\partial n} ds' = -4\pi GM. \tag{1.16}$$

This is known as Gauss' law, which states that the total normal gravitational flux over a closed surface is $-4\pi G$ times the enclosed mass. This law holds good irrespective of the type of mass distribution in s'. Basic relationships can therefore be established between the observations in the gravity field and the parameters describing the surface.

The acceleration of gravity in the direction of z-axis (that is the vertical which is the only direction in which g can be measured directly) can be derived from equation (1.9) with reference to the origin of the Cartesian coordinate system as

$$g_z = -\frac{\partial U}{\partial z} = G\rho \iiint \frac{z}{r^3} dxdydz, \tag{1.17}$$

where, $r^2 = x^2 + y^2 + z^2$.

On the other hand, if the mass is very long in the y-direction and has a uniform cross-section in the xz-plane, the gravity attraction can be derived from a logarithmic or 2-D potential and is given as

$$U = 2G\rho \iint \log\left(\frac{1}{r}\right) dxdz\,. \tag{1.18}$$

Thus the gravity effect of a 2-D structure can be expressed as

$$g_z = -\frac{\partial U}{\partial z} = 2G\sigma \iint \frac{zdxdz}{r^2}, \tag{1.19}$$

where, $r^2 = x^2 + z^2$.

The relation derived above for the gravity effect of a 2-D and 3-D structures are under the assumption that the density is constant all through the volume.

The contrast in density between the source of interest and the surroundings is exploited to measure the corresponding gravity field either or both on the surface of the earth as well as in boreholes. The measured gravity field is subsequently corrected for the unwarranted effects that arise due to the earth's topography, the nature of the subsurface etc. before attempting any meaningful interpretation followed by its geological translation.

It may be emphasized in unequivocal terms that the success of gravity method largely depends on the existence of significant density contrast, which could generate favorable geophysical response due to the source of interest, the size and depth of the source and subsequent interpretation.

From the measurements of gravity fields, it could be possible to identify large objects having a change in density even when the objects are buried beneath the earth's surface. Furthermore, with the advent of gravity measurements derived from satellite altimetry, the density of gravity measurements in the oceans took a quantum leap forward. Many details of the geometry of tectonic plates, particularly in the southern hemisphere, became clear for the first time. A brief history of the development of the gravity method was elucidated well in classic papers by Eckhardt (1948) and Nabighian et al. (2005).

Gravity method plays a decisive and definite role in oil and natural gas exploration. Geologic structures in which hydrocarbons are entrapped exhibit such large contrasts in density with respect to the surrounding formations that gravity data also can be used to decide on drilling locations. Currently gravity surveys are carried out in the search for oil is designed for reconnaissance of large unexplored areas. In oil exploration, the gravity method is particularly applicable in salt provinces, over thrust and foothills belts, and targets of interest that underlie high velocity zones. Gravity models are routinely used as a constraint for seismic imaging of the bottom of allocthonous salt bodies. The gravity data are used in

conjunction with prestack depth migration of the seismic data in an iterative way to build a better velocity cube, there by leading to clearer images of the base of the salt (Huston et al., 2004).

In the absence of required geological information in a region, gravity method can provide vital information about the size and thickness of a sedimentary basin to justify further investigation. Generally, sedimentary rocks have densities lower than basement rocks, in which case, the density contrast makes it possible to map the boundaries and determine the approximate depth distribution of the basins. Furthermore, a priori knowledge of the region from gravity data reduces the risk and cost involved in seismic surveys and aids in properly locating seismic lines. Although, the gravity method appears to have limited applicability in mineral prospecting, some ores such as high-density chromites and sulphides can be located by detailed gravity surveys. In addition, this method is used to directly calculate massive ore reserves. Buried channels, which might contain gold and uranium minerals, can frequently be located by gravity method. Regional studies make it possible to detail major structural features such as faults or lineaments, where mineral accumulations are likely. Further, gravity surveys are frequently employed in base metal exploration. There is also a modest increase in the use of gravity method in archeological investigations (Lakshmanan and Montlucon, 1987; Deletie et al., 1988; Brissaud et al., 1989). Thus, gravity method being economical serves as an important and indispensable tool particularly when the contrast in density between the targets of interest and their host rocks is significant.

1.2 Gravity corrections

The measured vertical component of gravity field on the surface of rotating earth is not uniform owing to its uneven topography besides its departure of shape from that of a sphere and its centrifugal force. Corrections can be worked out and applied to the measured gravity fields to offset the aforesaid effects. The left out residues are known as the Bouguer gravity anomalies, which are related to the lateral variation in density of the materials at different depths below the ground surface. The deduced subsurface density distributions are finally translated in terms of the sources of geological interest. The end product of a gravity survey yields a set of gravimeter readings and a set of station elevations measured with respect to the mean sea level or geoidal surface. After properly accounting for the drift of the instrument and temporal variations of the earth's gravity field, absolute gravity at each observation is established by connecting the field gravity meter readings to the base station. These absolute gravity values represent the cumulative effects of the sources being looked for, in addition to the earth's topography, its non-sphericity of shape and its centrifugal force. In principle, normal gravity is calculated on the spheroidal surface of the earth, projecting it to the level of observation and adding to it the gravity effect of the masses between the spheroid and the ground surface. This can be realized by applying a series of corrections viz., normal or latitude correction, free-air correction, Bouguer correction and topographic correction to the measured gravity data. A detailed account on the application of gravity corrections is given by Rao and Murthy (1978).

The corrected gravity fields are known as the Bouguer gravity anomalies. However, it is to be noted that in free-air and Bouguer gravity corrections, the elevation, h, has to be measured from the spheroid. In reality, elevations are measured with reference to the mean sea level or geoidal surface. Therefore, the Bouguer gravity anomalies are invariably associated with an indispensable error, which is being equal to $(0.3086 - 2\pi G d_B)h_N$, where, d_B is the Bouguer density and h_N being the elevation of the geoidal surface above the spheroid. However this effect will be automatically compensated during the process of regional and residual anomaly separation (Murthy, 1998). When the elevation differences between stations are small, the anomalies are sought to have been made on a mean horizontal plane coinciding with the topography. Finally, the Bouguer gravity anomalies are presented in the form of contour maps and as stacked profiles and then subjected to interpretation.

1.3 Interpretation

Interpretation of gravity anomalies is a four-prong strategy viz., i) qualitative interpretation of Bouguer anomaly maps, ii) regional and residual anomaly separation, which is often supplemented with derivative calculations and continuation, iii) quantitative interpretation and iv) geological translation of geophysical interpretations.

1.3.1 Qualitative interpretation

In qualitative interpretation, gravity highs and lows are identified on a contour map and their axes are correlated with known surface or

subsurface geology and the correlations are extrapolated to poorly mapped areas. Further, the nature and characteristics of the anomalies are studied to identify the source geometry, its orientation besides its approximate depth of occurrence.

Two-dimensional bodies of large strike lengths are represented by anomaly contours elongated roughly parallel to the strike of anomalous bodies; conversely the direction of elongation of contours or their axis is a measure of the strike of the anomalous body. Three-dimensional bodies with little or no strike-length produce elliptical or closed contours. Bodies occurring at shallow depths generate sharp anomalies, while those at depth will be responsible for broad anomalies only.

When the anomalies are presented in the form of profiles, typical characteristics of individual profiles are identified and correlated with those on neighboring profiles. Characteristic features persisting on neighboring profiles are indications of anomalous bodies of large strike length, whose direction can be determined. Fault bodies or their terminations create dislocation or complete absence of characteristics on neighboring profiles. Widths of individual anomalies and their sharpness decide the order of depth or horizontal dimensions of the body.

1.3.2 Regional and residual anomaly separation

The measured Bouguer gravity anomalies are the cumulative gravity effects of the sources distributed both laterally and vertically. The contribution of gravity field due to the source/sources of interest are

known as the residual gravity anomalies, which are high frequency in nature, while that of the sources situated at deeper levels and the sources far away from the source/sources correspond to low frequency and known as regional gravity anomalies. The gravity field due to sources above the object of interest is known as noise and can be identified and removed from the Bouguer anomalies owing to its conspicuous and random nature of occurrence. Deeper/far away sources generate broad wavelength gravity anomalies in comparison with the residuals and can be separated from the Bouguer anomalies. This process of separating the regional gravity anomalies from residuals is known as the regional and residual anomaly separation.

Generally, known geology plays a decisive role in deciphering the best regional trend on a Bouguer anomaly map. Further, the process of regional and residual separation is highly biased in the absence of *apriori* information about the geology (Chakravarthi and Sundararajan, 2007). Some of the elementary methods such as graphical and smoothing techniques, empirical gridding methods, second derivative etc. (Telford et al., 1990) are commonly employed for regional anomaly separation. Many other methods based on upward continuation (Jacobsen, 1987; Pacino and Introcaso, 1988), minimum curvature (Mickus et al., 1991), orthogonal polynomial fitting (Grant, 1957; Forsythe, 1957; Spitz, 1966; Merriam and Cocke, 1967; Beltrao et al., 1991; Agarwal and Sivagi, 1992), linear filtering (Strakhov, 1964; Strakhov and Lapina, 1967, Naidu, 1966; Naidu, 1967; Spector and Grant, 1970; Byerly, 1965; Sax, 1966; Lavin and Devane, 1970), nonlinear filtering (Naudy and Dreyer, 1968), Band

pass filtering (Blakely, 1995), Wiener filtering (Pawlowski and Hansen, 1990), Green's equivalent layer concept (Pawlowski, 1994), fractals (Chapin, 1996), finite elements (Mallick and Sharma, 1999; Agarwal and Shalivahan, 2010), Hartley transform (Kadirov, 2000), multiscale edge and iterative lateral continuation and subtraction analysis (Boschetti et al., 2004), eigenimage extraction technique (Ganguly and Dimri, 2013), and higher order polynomial fitting (Chakravarthi et al., 2013c) are available for separating regional anomaly from the Bouguer gravity anomalies.

1.3.3 Quantitative interpretation

Quantitative interpretation of gravity anomalies aims at estimating the parameters of geophysical models, which in turn explain the geology. However, the interpretation of gravity anomalies is non-unique in the sense that the measured gravity anomalies on the plane of observation can be explained by a variety of density distributions as explained by Roy (1962) and Blakely (1995). The ambiguity in gravity interpretation is often tackled by assigning a mathematical geometry to the anomalous mass with a known density and then to invert the anomalies for the unknown parameters such that the modeled gravity anomalies mimic the observed ones and the estimated structure be geologically reasonable (Murthy, 1998; Chakravarthi et al., 2013c). Using the information derived from drilling/other geophysical data in the model space would further reduce the degree of uncertainty in the interpretation.

The nature and characteristics of the anomalies very often reveal that whether the anomalous source is a 2-D, 2.5-D and/or 3-D structure

(Rao and Murthy, 1978; Murthy, 1998). Suitable interpretation strategies can therefore be applied to interpret gravity data.

Although, interpretation techniques based on the thumb rules, curve matching etc. are in vogue, the advent and advancement of digital computers coupled with sophisticated software revolutionized the art of interpretation leading into the development of highly powerful algorithms, which almost surpass the existing old techniques.

1.4 Review of existing methods

Techniques developed in the past, based on the Fourier transforms (Sharma and Geldart, 1968; Collins et al., 1974; Mohan, 1978; Murthy and Rao, 1980; Chacko and Battacharya, 1980; Mareschal, 1985; Annecchione et al., 2001), Singular Value Decomposition (Lines and Treitel, 1984), Backus-Gilbert approach (Green, 1975), Hilbert transforms (Sundararajan et al., 1983), Hartley transforms (Sundararajan and Brahmam, 1998), tunneling algorithms (Mohan et al., 1986), Monte Carlo methods (Mosegaard and Tarantola, 1995), neural networks (Guang et al., 1998; Leite and Filho, 2009), Sundararajan transform (Sundararajan et al., 2000), simulated annealing (Nagihara and Hall, 2001; Jingxin et al., 2013), wavelet transforms (Marlet et al., 2001; Oruç, 2014), genetic algorithms (Boschetti et al., 1997; Yao et al., 2003), analytic signals (Beiki, 2010), Eigenvector analysis (Beiki and Pedersen, 2010), Particle swarm optimization (Toushmalani, 2013) are available to interpret the gravity anomalies assuming uniform density for the source/sources. However, such an approximation of uniform density for the anomalous source/

sources is seldom valid particularly when reference is made to the structures associated with the sedimentary rocks.

There are two specific approaches namely linear and nonlinear for the inversion of gravity anomalies is available. In the case of linear inversion, the source volume is divided into a number of blocks with simple geometry and the density of each block is determined (Li and Oldenburg, 1998) while in nonlinear inversion, a known density contrast is assumed and nonlinear operators are designed to determine the geometry of the source of the anomaly. The linear approach is easily applicable to the subsalt structures in hydrocarbon exploration, wherein the density contrast between the salt and surrounding sediments varies with depth. In such strategies, to constrain the density of each block in the linear method, ridge regression algorithm is generally used. Fedi and Rapolla (1999) have suggested an inversion technique of gravity and magnetic data with depth resolution. In their method, which is natural and objective, involves least squares of both the horizontal and vertical variation of the field. Boulanger and Chouteau (2001) have developed a flexible code for the inversion of gravity data using a selection of constraints such as minimum curvature, flatness and compactness. Jessell (2001) studied in detail the different ways of potential field modeling systems that represent the 3-D structure of the earth's crust. On the other hand, Bosch and McGaughey (2001) discussed the effect of joint inversion of gravity and magnetic data under lithologic constraints. Nagihara and Hall (2001) introduced a robust technique for gravity inversion, which dramatically reduces the ambiguity of the final model

associated with the problem of non-uniqueness. This inversion algorithm of Nagihara and Hall (2001) makes use of Simulated Annealing Global Optimization (SAGO), which has advantages over the conventional methods as this algorithm converges rapidly with minimum computation memory besides being efficient. Further, Mendonca (2004) present an inversion of gravity field inclination to map the basement of sedimentary basins that does not require the knowledge of the basin density contrast, however, determine the contrast under the assumption that the basin is homogeneous.

In inverse problem a set of geological parameters is sought to define mathematical models which reproduce the observed data in a satisfactory fashion. Normally, such an inversion is dealt with local optimization techniques such as steepest descent (Paul, 1972), conjugate gradients or simultaneous iterative reconstruction method (Commer, 2011) etc. However, there are a few constraints, which may limit the effectiveness of these methods. In recent years, global optimization techniques such as genetic algorithms (Boschetti et al., 1997; Roy et al., 2002), simulated annealing (Roy et al., 2002; De Vicente et al., 2003), ant colony optimization (Dorigo and Blum, 2005; Gupta et al., 2011; Srivastava et al., 2013), and particle swarm optimization (Eberhart and Kennedy, 1995; Shalivahan and Agarwal, 2010; Toushmalani, 2013) etc. which can overcome the limitations associated with local optimization are found in literature. These methods usually start with an initial set of variables and then evolve to obtain the global minimum or maximum of the objective function.

Genetic Algorithms (GA) are search methods modeled on the evolutionary behavior of biological systems, capable of solving complex nonlinear problems. Simulated Annealing (SA) simulates annealing process in which a substance is heated above its melting temperature and then gradually cools to produce the crystalline lattice, which minimizes its energy probability distribution. Ant Colony Optimization (ACO) is another evolutionary optimization algorithm which is inspired by the pheromone trail laying behavior of real ant colonies. On the other hand, Particle Swarm Optimization (PSO) is a population based stochastic optimization technique inspired by social behavior of bird flocking or fish schooling. PSO shares many similarities with GA. The system is initialized with a population of random solutions and searches for optima by updating generations. However; unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. These algorithms are increasingly finding application in geophysical problems including potential field inverse problems, particularly in simultaneous inversion of gravity and magnetic data in order to reconstruct the shape of buried geological bodies; however, they are far more expensive. The main advantage of such a procedure is that the ambiguity inherent in the inversion of potential field data is greatly reduced when the two data sets are combined.

Thus, the real goal of inverse problems is to determine model parameters from observations; hence inverse problems are an important area of geophysical research and industrial application. A major task of

geophysics is to make quantitative analysis about the subsurface of the earth, based on the measurements made on the surface of the earth. Although, a solution that satisfies the observed data can easily be found, its non-uniqueness in potential field method is caused by the nature of the physics and under determination of the problem. To deal with non-uniqueness using models composed of a large number of contiguous cells of known density, a *priori* information can be added to constrain the solution.

The aim of the present work as presented in this thesis is to develop appropriate 2-D and 2.5-D automatic modeling and inversion strategies coupled with relevant GUI based softwares in JAVA to analyze the gravity anomalies of sedimentary basins using prescribed Exponential Density Function (EDF). Further, a semiautomatic 2.5-D modeling with relevant software coded in JAVA is also developed in the space domain to analyze the gravity anomalies of multiple geologic sources having differing strike lengths, density contrasts and thickness parameters. In all the inversion strategies presented in Chapters II and III, the ridge regression algorithm (Marquardt, 1963; Marquardt, 1970) is used to estimate the source parameters.

1.4.1 Exponential density function (EDF)

It is well known that the density of sedimentary rocks varies with depth (Athy, 1930; Hedberg, 1936; Hughes and Cooke, 1953; Rubey and Hubbert, 1956; Weller, 1959; Crowe and Redmond. 1962; Ham, 1966; Foster and Whalen, 1966; Teodorovich and Chernov, 1968; Gardner et al.,

1974; Nettleton, 1976; Maxant, 1980; Sclater and Christie, 1980; Prieto et al., 1985; Hermes,1986; Peirce and Lipkov, 1988; Ferguson et al., 1988; Abdoh et al., 1990; Jachens and Moring, 1990; Garcia-Abdeslem, 1992; Ruotoistenmaki, 1992; Abbott and Louie, 2000; Moral et al., 2000; Kadirov, 2000; Nagihara and Hall, 2001; Adriasyah and McMechan, 2002; Ursin et al., 2003; Hinze, 2003; Chakravarthi, 2003; Gȯmez-Ortiz et al., 2005; Garcia-Abdeslem, 2005; Gimenez et al., 2009; Azeglio et al., 2010; Gu et al., 2014; Martinez et al., 2014) and hence the use of variable density functions in the analysis of gravity anomalies is more preferable than uniform density model.

Furthermore, the density-depth curves constructed for different stratigraphic units in many sedimentary basins at their centers and along margins from the measured well density logs exhibit a range of densities but the mean sediment density clearly increases with depth with the highest rate of increase in the top few kilometers. More often, the sediments in the deepest part of a basin have a density approaching that of the basement rocks whereas near the basin margins, the sediments in contact with the basement have a relatively lower density (Cowie and Karner, 1990).

The variation of density of sedimentary rocks with depth can be described more effectively by an exponential density function (Cordell, 1973; Chai and Hinze, 1988; Chappell and Kusznir, 2008; Chakravarthi et al., 2013a, Chakravarthi et al., 2013b, Chakravarthi et al., 2013c), if differential compaction is assumed to be the most important diagenetic

process in the evolution of sedimentary basins. Hence, the use of exponential density function in the interpretation of gravity anomalies of sedimentary basins often ensures more reliable results.

The exponential density function is defined as (Cordell, 1973)

$$\Delta\rho(z) = \Delta\rho_0 e^{-\lambda z}, \tag{1.20}$$

where, $\Delta\rho(z)$ is the density contrast of sediments at any depth, z, $\Delta\rho_0$ is the density contrast at the surface/topography and λ is a decay factor expressed in reciprocal length units. The values of $\Delta\rho_0$ and λ can be obtained by fitting equation (1.20) with the knowledge of at least one of the following

i) known subsurface geological information,

or

ii) pre/post stack inversion of seismic data,

or

iii) seismic tomography.

However, the quandary associated with the exponential density function is that no closed form analytical gravity expressions can be derivable in space domain even for simple geophysical geometries (Chakravarthi and Sundararajan, 2004). In this thesis, a combination of analytical and numerical approaches is used to realize forward gravity modeling of geologic structures in the space domain using this density function. From the known density-depth information, the Exponential

Density Function (EDF) and Uniform Density Function (UDF) are constructed for the field data presented in chapters II and III.

1.4.2 Automatic modeling and inversion

In literature, the terms 'forward modeling', 'automatic modeling' and 'inversion' are used more or less synonymously to refer to various interpretation strategies of gravity anomalies. However, in the present thesis, these terms are distinguished from each other following the criteria suggested by Murthy (1998), Chakravarthi et al. (2013c).

Accordingly, forward modeling (indirect methods) refers to a mathematical process of computing theoretical gravity response of a given geophysical model with the known size and shape parameters. Therefore, forward modeling is essentially an inbuilt part of the automatic modeling and inversion strategies (direct methods). Automatic modeling schemes refer to those strategies, which interpret the gravity anomalies of the open-ended body/bodies that undulate over a mean/undisturbed depth. Inversion strategies are capable of interpreting the gravity anomalies of both open ended and closed geophysical models. Sometimes, modeling is identified with forward modeling of anomalies, or the classical trial and error method of interpretation, where the model parameters are modified by the interpreter based on a comparison between the observed and modeled gravity anomalies. There exist a few typical cases, where the gravity anomalies are modeled automatically by computing machines without an active interaction of the interpreter.

In automatic modeling schemes error between the observed and modeled gravity anomalies at each observation is used to modify selectively one particular parameter of the model such as depth to the density interface only at that point, while in inversion error between the observed and modeled gravity anomalies at all stations are used to estimate the changes of a given geophysical geometry (Murthy, 1998; Chakravarthi et al., 2013c). A detailed account of the principles of inversion is described by Al-Chalabi (1970, 1972), Oldenburg (1974), Pedersen (1977), Murthy (1998), Scales and Snieder (2000), Chakravarthi (2003).

In automatic modeling and inversion strategies, approximate parameters of a given geophysical model are generally assumed based on known geology and subsequently improved in an iterative approach until the observed gravity anomalies fit closely the theoretical anomalies. Gravity anomaly due to a geophysical model can be expressed as a product of shape and size parameters. The size parameter contains the information on the physical property contrast and shape parameter possesses the information of the dimension of the geophysical model under consideration. Modeling schemes solve the shape parameters of a geophysical model, such as the depth to a density interface, whereas inversion strategies can be formulated to determine either or both size and shape parameters, provided the number of unknowns to be solved is less than or equal to the number of measurements.

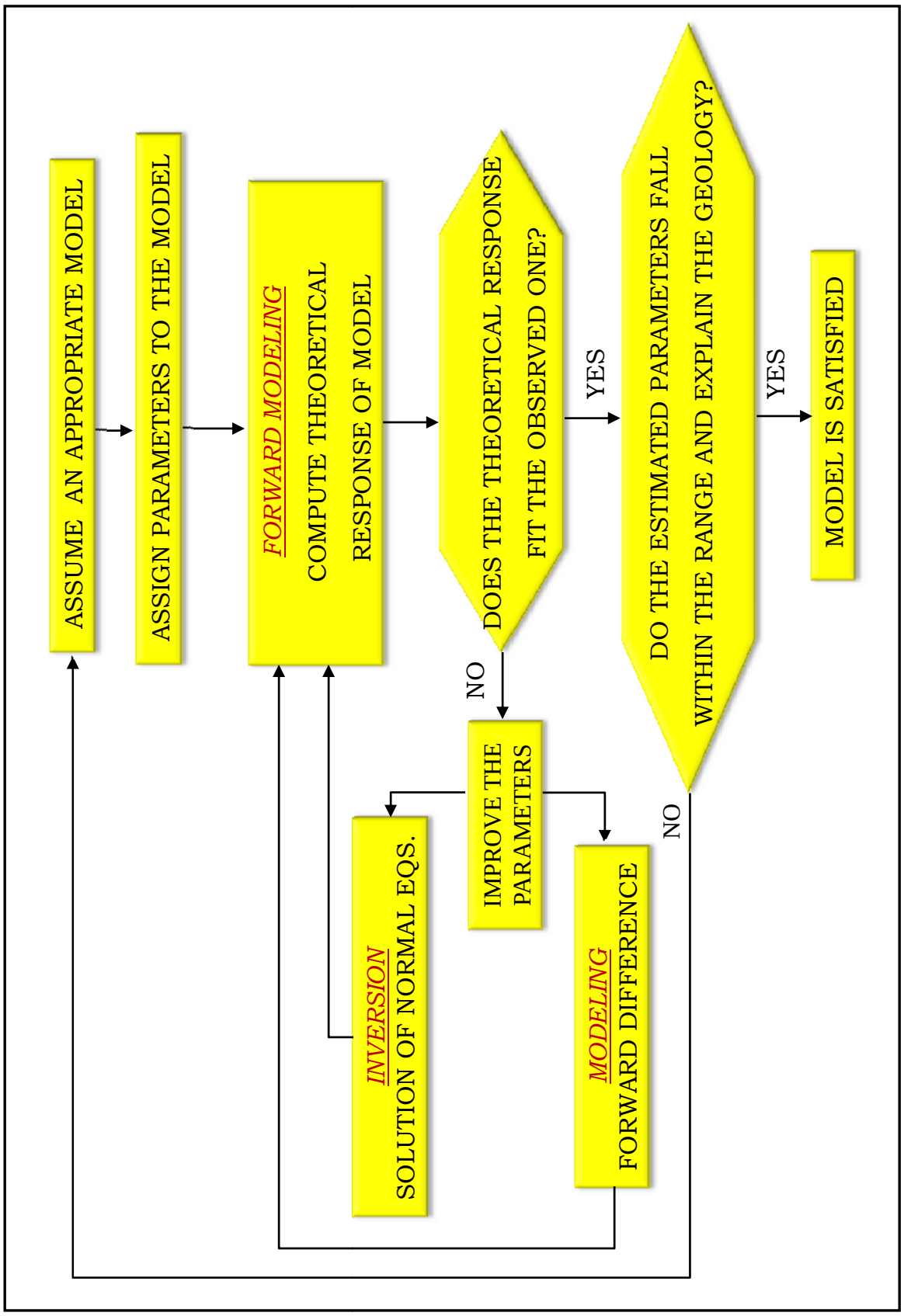

Figure 1.1 Generalized flow chart of automatic modeling and inversion

The generalized flow chart of automatic modeling and inversion strategies is given in Figure 1.1. In both the schemes, an appropriate geophysical model is assumed with a set of shape and size parameters and a suitable forward modeling routine computes the theoretical anomalies. If the model response compares well with the observed one and the interpreted parameters fall within the range to explain the geology, then the interpreted model is considered to be satisfactory. In case, the modeled anomalies do not fit well the observed ones then the model parameters are modified in an iterative approach within the specified convergence criteria. It is to be noted that in automatic modeling, the parameters are improved based on simple forward difference approximation (Chakravarthi et al., 2013a) whereas in inversion, normal equations are to be constructed and solved for the increments/decrements of the parameters (Murthy, 1998; Chakravarthi et al., 2013b; Chakravarthi et al., 2013c).

1.4.3 *Principles of ridge regression algorithm*

The principles of ridge regression inversion (Marquardt, 1963) can briefly be explained as under. Let $\hat{P}_{\hat{k}}$, $\hat{k}=1,2,...,N$ are the unknown parameters of a geophysical model to be solved from N_{obs} observations. Further, let $P'_{\hat{k}}$, $\hat{k}=1,2,...,N$ be the initial parameters for which the model gravity anomalies, $g_{mod(k)}$, $k=1,2,\ldots,N_{obs}$ are obtained by a relevant forward modeling routine, $g(x,y,z,\hat{P}_1,\hat{P}_2,....)$. Since the initial parameters are only approximate, the model gravity anomalies, $g_{mod(k)}$, deviate from the observed anomalies, $g_{obs(k)}$. The difference between observed and

modeled gravity anomalies is quantified by a misfit function, J, (Murthy, 1998; Chakravarthi and Sundararajan, 2004) as

$$J = \sum_{k=1}^{N_{obs}} \left[g_{obs(k)} - g_{mod(k)} \right]^2. \tag{1.21}$$

For J to be minimum, the partial derivatives of J with respect to each parameter to be solved, $\hat{P}_{\hat{k}}$, $\hat{k}=1,2,...,N$ is set to zero i.e.,

$$\frac{\partial J}{\partial \hat{P}_{j'}} = 0, for\ j' = 1, 2, \dots, N. \tag{1.22}$$

In gradient search methods, the differences between optimum and assumed parameters, $dP_{\hat{k}}$, $\hat{k}=1,\ 2,\ ..\ N$ are solved in an iterative approach. The expression for gravity anomaly of a geophysical model with optimum parameters can be expressed in a truncated Taylor's expansion as

$$g(P_1' + dP_1, P_2' + dP_2, \dots., P_N' + dP_N) = g(P_1', P_2', \dots, P_N') \pm \sum_{\hat{k}=1}^{N} \frac{\partial g}{\partial \hat{P}_{\hat{k}}} d\hat{P}_{\hat{k}}. \tag{1.23}$$

Substituting equation (1.23) in equation (1.21), the misfit function, J, can be expressed as

$$J = \sum_{k=1}^{N_{obs}} \left[g_{obs(k)} - g_{mod(k)}(P_1', P_2', \dots, P_N') \pm \sum_{\hat{k}=1}^{N} \frac{\partial g}{\partial \hat{P}_{\hat{k}}} d\hat{P}_{\hat{k}} \right]^2. \tag{1.24}$$

Upon substitution of equation (1.24), equation (1.22) takes the form

$$\sum_{k=1}^{N_{obs}} \sum_{\hat{k}=1}^{N} \frac{\partial g}{\partial \hat{P}_{j'}} \frac{\partial g}{\partial \hat{P}_{\hat{k}}} d\hat{P}_{\hat{k}}$$

$$= \sum_{k=1}^{N_{obs}} \left[g_{obs(k)} - g_{mod(k)}(P'_1, P'_2, \dots, P'_N) \right] \frac{\partial g}{\partial \hat{P}_{j'}}, j' = 1, 2, .., N. \qquad (1.25)$$

A set of normal equations equal to the number of unknown parameters are framed and solved for the increments/decrements of the N unknown parameters, $d\hat{P}_{\hat{k}}, \hat{k} = 1, 2, .., N$. When the approximate initial parameters, $P'_{\hat{k}}, \hat{k} = 1, 2, .., N$ of the modeled anomalies are significantly deviated from the actual parameters, $\hat{P}_{\hat{k}}$, then the increments/ decrements, $d\hat{P}_{\hat{k}}, \hat{k} = 1, 2, .., N$ obtained from equation (1.25) tend to be spurious resulting in the divergence of the solution. To overcome such a difficulty a damping factor, δ, can be incorporated in equation (1.25) as

$$\sum_{k=1}^{N_{obs}} \sum_{\hat{k}=1}^{N} \frac{\partial g}{\partial \hat{P}_{j'}} \frac{\partial g}{\partial \hat{P}_{\hat{k}}} (1 + \vartheta\delta) d\hat{P}_{\hat{k}}$$

$$= \sum_{k=1}^{N_{obs}} \left[g_{obs(k)} - g_{mod(k)}(P'_1, P'_2, \dots, P'_N) \right] \frac{\partial g}{\partial \hat{P}_{j'}}, j' = 1, 2, .., N, \qquad (1.26)$$

where,

$$\vartheta = \begin{cases} 1 \; for \; j' = \hat{k} \\ 0 \; for \; j' \neq \hat{k} \end{cases}.$$

Partial derivatives required in the above system of equation (1.26) can be computed either numerically or analytically. Initially the value of δ is set to 0.5 and equation (1.26) is solved for the increments/decrements, $d\hat{P}_{\hat{k}}, \hat{k} = 1, 2, .., N$, and are subsequently added to/subtracted from, $P'_{\hat{k}}$,

$\hat{k} = 1, 2, .., N$, to obtain the improved parameters, $P'_{mod\hat{k}}$, $\hat{k} = 1, 2, .., N$. If the current value of the misfit function, J_{mod}, obtained with $P'_{mod\hat{k}}$, $\hat{k} = 1, 2, .., N$ is less than its previous value, J, then J_{mod} is assigned to J and $P'_{mod\hat{k}}$ to $P'_{\hat{k}}$, $\hat{k} = 1, 2, .., N$, and the damping factor, δ, is decreased by a factor of 2. In case, J_{mod} is greater than J, then the value of δ increases by a factor of 2. The inversion algorithm automatically gets terminated in case

i) the specified number of iterations completed

or

ii) the misfit function falls below a predefined allowable error

or

iii) the damping factor assumes an unusually large value.

1.5 Aim and scope of the thesis

The aim and objective of the thesis is to investigate the application of ridge regression algorithm in inversion of gravity anomalies due to 2-D, 2.5-D sedimentary basins, besides developing automatic and semiautomatic modeling schemes coupled with relevant GUI based softwares coded in JAVA. Accordingly, the thesis under study is organized into five chapters as detailed here under.

Chapter-I deals with general introduction, a brief review of the earlier work on automatic modeling and inversion besides a detailed discussion on the mathematical analysis of ridge regression algorithm.

In Chapter-II, automatic modeling and inversion algorithms coupled with a GUI based software to analyze the gravity anomalies due to 2-D

sedimentary basins with exponential density function are dealt with. The analysis is supported in each case by a theoretical model and substantiated with real field data pertaining to the San Jacinto graben, California.

Automatic modeling and inversion of gravity anomalies due to 2.5-D sedimentary basins with exponential density function are dealt with in Chapter-III along with a relevant GUI based software. The effect of the offset of the profile on the magnitude of gravity anomaly is discussed in depth. Both with a theoretical model and real field data of Chintalpudi sub-basin, India, the modeling and inversion are demonstrated.

In Chapter-IV, a semiautomatic modeling algorithm and a GUI based code to analyze the gravity anomalies of multiple geologic sources having differing strike lengths and arbitrary density contrast depth variations is dealt with. The effects of azimuth of the profile and strike length of the structure on the magnitude of the gravity anomaly are demonstrated in detail. The analysis is supported by a synthetic model and real field data across the suture zone between the Superior and Churchill structural provinces in the Canadian Shield.

In all the theoretical models and field data analysis presented in chapters II and III the modeling and inversion carried out with Exponential Density Function (EDF) is compared with that of Uniform Density Function (UDF).

A comprehensive conclusion of the entire work presented in the thesis and scope for future research are given in Chapter-V.

CHAPTER
TWO

Automatic Modeling* and Inversion+ of Gravity Anomalies due to 2-D Sedimentary Basins with Exponential Density Function

2.1 General:

Calculation of basement depths where the density contrast varies significantly is an important geological study of the gravity method in regional geophysical surveys and hydrocarbon exploration. Negative gravity anomalies are generally observed over sedimentary basins having a large thickness of low-density rocks. The interpretation of gravity anomalies due to such sedimentary basins mimics a mathematical process of trying to fit a suitable geometry to low-density sedimentary load over the basement and identifying the model parameters within the permissible limits such that the modeled gravity anomalies fit the observed ones. Stacked prism model of Bott (1960) and polygon model of Talwani et al. (1959) are widely used mathematical geometries for modeling sedimentary basins. In Bott's (1960) method of interpretation,

---

* *Published in Journal of the Geological Society of India (2013, 82, 561-569), Springer.*
+ *In press with Acta Geophysica, Springer.*

the cross-section of a sedimentary basin was viewed as a series of outcropping juxtaposed prisms, while in the method of Talwani et al. (1959); the outline of the source was approximated by an N-sided polygon. The necessary condition involved in Bott's method of interpretation is that the observations/measurements must be available at equal station interval along a profile; which however is not mandatory in Talwani's method.

Although many forward modeling schemes (e.g., Paul, 1974; Barnett, 1976; Okabe, 1979; Golizdra, 1981; Strakhov et al., 1986; Won and Bevis, 1987; Gotze and Lahmeyer, 1988; Pohánka, 1988; Pohánka, 1990; Holstein and Ketteridge, 1996; Singh and Guptasarma, 2001) to compute the gravity anomalies with uniform density are available, they are of limited application in analyzing the gravity anomalies of sedimentary basins because i) the parameters of such basins are not known in advance, and ii) density of sedimentary rocks varies with depth. Further, in the absence of additional information it becomes an uphill task to fit the model gravity response of a sedimentary basin with the observed one by adjusting several depths to the interface in an interactive mode (Chakravarthi et al., 2013a). In this context, Morgan and Grant (1963), Oldenburg (1974), Bhattacharya and Navolio (1975), Murthy et al. (1988), Murthy and Rao (1989), Leão et al. (1996), Barbosa et al. (1997), Barbosa et al. (1999), Annecchione et al. (2001), Silva Dias et al. (2007) have developed techniques to interpret gravity anomalies due to 2-D sedimentary basins presuming uniform density for the sedimentary rocks, which is seldom valid.

On the other hand, Ruotoistenmäki (1992) devised a scheme to calculate the gravity effects of 2-D sources of arbitrary cross-sections using variable density contrast functions. García-Abdeslem (1996) developed a code in Fortran to compute the gravity response of a vertical prism in which the density varies as a function depth. Martín Atienza and García-Abdeslem (1999) developed a technique to calculate gravity anomalies of 2-D sources with an analytically defined geometry and density contrast defined by a second order polynomial function of depth and distance along the profile. Hansen (1999) derived a closed form analytical expression for the gravity field of a polyhedron with linearly varying density. Zhang et al. (2001) proposed a technique for computing the gravity anomalies of 2-D bodies having irregular shape with density contrasts varying according to a polynomial function with depth and lateral position. Holstein (2003) presented formulas for the gravity potential, field, and field gradient tensor for a polyhedral target body of a spatially linear density medium, García-Abdeslem (2005) employed a cubic polynomial to describe the variation of density of sedimentary rocks with depth. However, the linear density function is more appropriate to describe the density variation of sediments at larger depths than at shallower depths, whereas cubic polynomial fails to represent the actual density at larger depths as demonstrated by Chakravarthi (2009). Further, the enlisted forward modeling schemes find limited practical application in analyzing the gravity anomalies of sedimentary basins.

Litinsky (1989) used a hyperbolic density function to calculate the thickness of sedimentary basins using the Bouguer slab formula.

However, this method invariably incurs a considerable amount of error that is inversely proportional to the width of the basins (Chakravarthi and Sundararajan, 2004). The interpretation methods developed by Rao (1986), Rao (1990), Gallardo-Delgado et al. (2003) using a quadratic density function would pose problems in automatic modeling and inversion of gravity anomalies because this density function deviates both in sign and magnitude from true density contrast at larger depths (Chakravarthi, 2009). Zhou (2008) and Zhou (2009) derived generalized equations for the gravity anomalies of irregular 2D masses with a depth dependent density and horizontally and vertically dependent densities.

On the other hand, a few direct modeling methods are in vogue to analyze the gravity anomalies of sedimentary basins using the Exponential Density Function (EDF). Because of the fact that no closed form analytical equations can be derivable in the space domain for the gravity anomalies with an EDF, these algorithms perform forward modeling in the frequency domain and then transform the anomalies back to the space domain for further analysis. For e.g., Cordell (1973) developed a recursive method using both gravity field and its vertical derivative (determined by convolution in discrete Fourier series) to solve the structure of a sedimentary basin from the observed gravity anomalies. Granser (1987) had calculated the gravity effect of a structure based on series expansion, the numerical evaluation of which was performed by fast Fourier transform. Chai and Hinze (1988) proposed methods to analyze both profile and 2-D gravity data, where the forward modeling was realized in the wave number domain followed by its conversion to the

space domain by a shift-sampling technique. Rao et al. (1993) derived Fourier transforms of gravity anomalies of some simple geometric models with exponential density contrast and used them in the analysis of the gravity anomalies of sedimentary basins; however, these strategies yield unreliable interpretations when the sediment-basement interface has major undulations (Chakravarthi and Ramamma, 2013b). The method developed by Rao and Rao (1999) also involved the calculation of the gravity effect in the frequency domain and its subsequent transformation to the space domain by Filon's (1928) method. In recent past, Chappel and Kusznir (2008) extended the method of Granser (1987) to calculate the gravity anomaly as a function of the Fourier transforms of the bounding surfaces of a basin with irregular top and bottom surfaces. Nonetheless, the enlisted methods incur truncation errors when the modeled anomalies transform from frequency domain to the space domain (Chakravarthi and Sundararajan, 2007; Chakravarthi et al., 2013a).

On the other hand, a few space domain based algorithms are available to model the gravity anomalies, where the exponential density variation was accommodated in the interpretation by alternative means. For e.g., Murthy and Rao (1979) proposed the subdivision of each side of a 2-D polygon into a number of segments along each of which the density contrast was assumed to vary linearly with depth, whereas Guspí (1990) described the exponential density function by a series approximation to compute the gravity response. However, the method of Murthy and Rao (1979) consumes significant amount of time even for forward modeling (Visweswara Rao et al., 1994), whereas the method proposed by Guspí

(1990) requires the knowledge of the degree of the polynomial, which is generally not known a priori.

Thus, not many methods are in vogue in the space domain to overcome the drawbacks associated with the existing aforesaid techniques to interpret gravity anomalies of 2-D sedimentary basins using an exponential density function, although many geological situations mimic such density-depth variation.

This chapter deals with two space domain based interpretation strategies and relevant GUI based software that are developed based on the principles of automatic modeling and inversion to analyze gravity anomalies of 2-D sedimentary basins, among which the density contrast decreases exponentially with depth. The Bott's (1960) method of profile gravity interpretation is extended to develop these strategies. Applicability of these techniques is illustrated with a synthetic model using both exponential and uniform density functions (EDF and UDF) in each case and also is exemplified with the interpretation of gravity anomalies of the San Jacinto graben, California.

2.2 Automatic modeling

Automatic modeling of gravity anomalies of a sedimentary basin is tantamount to a mathematical exercise of estimating depth values of the floor of a sedimentary basin within the permissible limits such that the modeled gravity anomalies mimic the observed anomalies (Chakravarthi et al., 2013a).

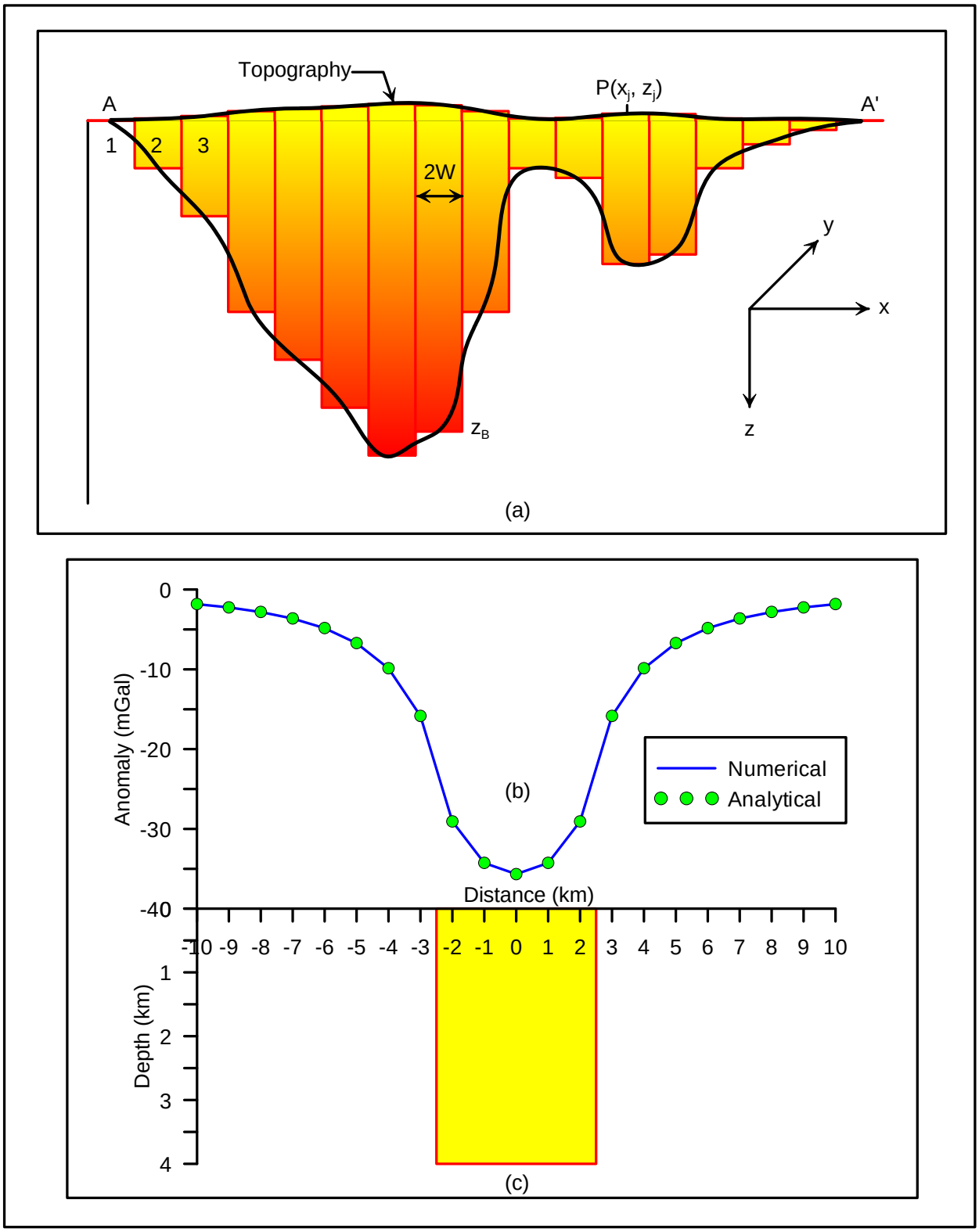

Figure 2.1 (a) Approximation of the cross-section of a 2-D sedimentary basin (solid line in black) by an ensemble of vertical prisms (solid lines in red). The shade within the basin from yellow to red indicates increase in density of sedimentary rocks with depth. (b) Gravity response calculated by numerical and analytical methods using UDF. (c) Geometry of a 2-D vertical prism.

2.2.1 Forward modeling - theoretical considerations

Figure 2.1a shows the cross-section of a typical 2-D sedimentary basin in xz-plane of the Cartesian coordinate system. Let the profile, AA', runs along the x-axis transverse to the strike of the basin. A collage of vertical prisms having equal widths, whose thicknesses are to be estimated, describes the geometry of the sedimentary basin (Figure 2.1a). In this case, the number of prisms is equal to the number of observations on the profile and the width of each prism is equal to the station spacing. Let $2W$ be the width of one such prism and z_B represents the thickness of the prism along the z-axis positive vertically downwards (Figure 2.1a). Let the density contrast along the prism varies vertically with depth according equation (1.20). The gravity anomaly of such a prism, $\Delta g_{prism}(x_j, z_j)$, on the profile, AA', outside the source region can be expressed as (Chakravarthi et al., 2013a; Chakravarthi and Ramamma, 2013b)

$$\Delta g_{prism}(x_j, z_j) = 2G \int_S \Delta\rho(z) \frac{(z - z_j)dxdz}{\left[\overline{x - x_j}^2 + \overline{z - z_j}^2\right]}. \tag{2.1}$$

Here, G is universal gravitational constant, (x, z) are source coordinates of an element within the prism, and (x_j, z_j) are observer's coordinates. Substituting equation (1.20) for $\Delta\rho(z)$ and upon integration with respect to x, equation (2.1) takes the form (Chakravarthi and Ramamma, 2013b)

$$\Delta g_{prism}(x_j, z_j) = 2G\Delta\rho_0 \int_0^{z_B} e^{-\lambda z} \left[tan^{-1} \frac{\overline{x - x_j} + W}{\overline{z - z_j}} - tan^{-1} \frac{\overline{x - x_j} - W}{\overline{z - z_j}} \right] dz. \tag{2.2}$$

Equation (2.2) needs to be solved numerically because no closed form solution exists in the space domain (Chakravarthi et al., 2013a; Chakravarthi and Ramamma, 2013b). Generally, numerical integration of a set of digitized values at equal intervals is convenient with the Simpson's rule. In case, the sampled values show an unduly large variation between any pair of consecutive points then the Simpson's rule tends to be unreliable and in such a case it is convenient to assume an exponential variation in the digitized values between two consecutive points (Murthy, 1998; Chakravarthi et al., 2013a; Chakravarthi et al., 2013b). This approach has been adopted in the present case to obtain the numerical solution of equation (2.2). The gravity anomaly with uniform density function (UDF) can also be realized by letting $\lambda = 0$ in equation (2.2).

To evaluate the accuracy of the proposed numerical method, the gravity effect produced by a homogeneous vertical prism having dimensions z_B = 4 km, $2W$ = 5 km (Figure 2.1c) with a density contrast of -0.35 g/cm^3 was obtained from both numerical integration of equation (2.2) and the analytic solution of equation (2.2) by Murthy (1998) and shown in Figure 2.1b. It was found that the maximum difference between the two anomalies is hardly exceeds 10^{-4} mGal, which in turn reveals the efficacy of the proposed numerical technique. Further, it is to be realized that equation (2.2) is strictly valid for the profile, AA', which runs transverse to the strike of the prism. In case the profile runs at an angle, α, with the x-axis then x_j in equation (2.2) needs to be replaced by $x_j cos\alpha$ (Chakravarthi and Ramamma, 2013b).

The total gravity anomaly produced by a sedimentary basin at any observation can be obtained as

$$g_{basin}(x_j, z_j) = \sum_{i=1}^{N} \Delta g_{prism}(x_i, z_i), \tag{2.3}$$

where, N stands for the number of prisms/observations on the profile.

2.2.2 Characteristics of gravity anomalies with EDF and UDF

To demonstrate the nature of gravity anomalies produced by a geologic structure with exponential and uniform density functions (EDF and UDF); two gravity anomaly profiles were generated (Figure 2.2a) in the interval x_j ∈ [-10 km, 10 km] over a prismatic structure whose geometry is shown in Figure 2.1c.

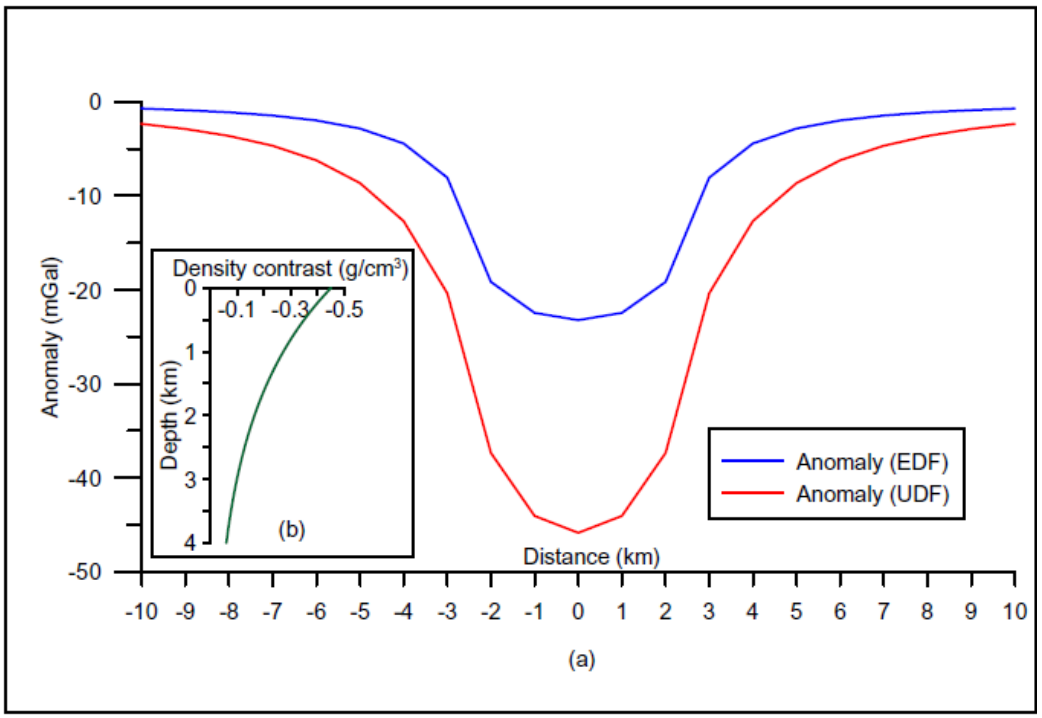

Figure 2.2 (a) Gravity anomalies with EDF and UDF over a prismatic structure. Geometry of the prism is shown in Figure 2.1c. (b) Prescribed EDF.

Equation (1.20) described with the constants $\Delta\rho_0$ = -0.45 g/cm^3 and λ = 0.5 km^{-1} is used to simulate the density contrast variation within the prism (Figure 2.2b). The anomalies of the structure with UDF are realized by setting λ = 0.

It can be noticed from Figure 2.2a that the maximum anomaly (absolute) produced by the prismatic structure with EDF hardly exceeds half the maximum anomaly produced by the same structure with UDF. It implies that sedimentary basins among which the density contrast varies exponentially with depth would generate only moderate gravity anomalies, and the fact that the present algorithm is also dependent on the magnitude of the anomalous field necessitates considering the exponential density function as a significant parameter in the interpretation for reliable results (Chakravarthi et al., 2013a).

2.2.3. Automatic modeling of gravity anomalies

The method of interpretation starts by initializing a sedimentary basin. It is presumed that at each observation, an infinite horizontal slab (Bouguer slab) in which, the density contrast varies according to equation (1.20) would generate the corresponding observed gravity anomaly. The thickness of Bouguer slab with EDF is given by (Cordell, 1973),

$$Z_B(x_j, z_j) = \frac{-1}{\lambda} log\left(1 - \frac{\lambda g_{obs}(x_j, z_j)}{2\pi G \Delta\rho_0}\right), \tag{2.4}$$

where, $g_{obs}(x_j, z_j)$, is the observed residual gravity anomaly at the station (x_j, z_j). In case of UDF, the thickness of Bouguer slab can be estimated as

$$Z_B(x_j, z_j) = \frac{g_{obs}(x_j, z_j)}{2\pi G \Delta\rho_0}, \tag{2.5}$$

Using the initial depth estimates as obtained from equation (2.4) in case of EDF and equation (2.5) in case of UDF, the gravity anomalies of the basin are calculated using equation (2.3). Since the initial depth estimates of a basin obtained from either equation (2.4) or equation (2.5) are only approximate, the model gravity anomalies, $g_{basin}(x_j, z_j)$, from equation (2.3) obviously deviate from the measured ones. The difference between the observed and modeled gravity anomalies can be explained by a misfit function, J, (equation 1.21). The difference between the observed and model gravity anomalies are used to improve the depth estimates of the prisms based on the expression given by Chakravarthi et al. (2013a) as,

$$Z_B^{m+1}(x_j, z_j) = Z_B^m(x_j, z_j) + \frac{g_{obs}(x_j, z_j) - g_{basin}(x_j, z_j)}{2\pi G \Delta\rho(z)}, j = 1, 2, \ldots, N, \tag{2.6}$$

where, m stands for the number of iterations. The modified depth estimates are again used to compute the model gravity anomalies of the basin using equation (2.3) and a new misfit, J_{mod}, is estimated. If the new misfit, J_{mod}, is less than its previous value, J, then the value of J_{mod} will be assigned to J and the process is repeated until one of the following conditions is satisfied.

i) the specified number of iterations completed

or

ii) the current value of misfit, J, falls below a predefined allowable error

or

iii) the current value of misfit is larger than its previous value.

2.3 Inversion

The objective of gravity inversion is to fit the modeled gravity anomalies to the observed ones by adjusting the thickness parameters of the basin in least squares approach such that the modeled gravity response of the optimum depth structure mimics the observed anomaly within the specified convergence criteria (Chakravarthi and Ramamma, 2013b).

The process of inversion begins by calculating the gravity response, $g_{basin}(x_j, z_j)$, using the initial the depth estimates obtained from equation (2.4) in case of EDF and equation (2.5) in case of UDF. The difference between these two anomalies at any observation at the end of the k^{th} iteration can be expressed as the cumulative of the products of vertical gradients of the anomaly and corresponding depth improvements of prisms as (Chakravarthi and Ramamma, 2013b)

$$g_{obs}(x_j, z_j) - g_{basin}(x_j, z_j) = \sum_{i=1}^{N} \left[\frac{\partial \Delta g_{prism}(x_i, z_i)}{\partial z} \right]_{z_k} dz_i. \qquad (2.7)$$

Equation (2.7) is constructed for each observation and N normal equations are framed and solved for the improvements in N depth parameters of prisms by minimizing the misfit function, J, defined by equation (1.21).

The system of normal equations (equation 1.26) can also be expressed in a matrix form as described by Chakravarthi and Ramamma (2013b)

$$(A + \delta I)X = B, \tag{2.8}$$

where, A is a nxn matrix whose elements $A_{nj'}$are given by

$$A_{nj'} = \sum_{n=1}^{N}\sum_{m=1}^{N} \frac{\partial \Delta g_{prism}(x_m, z_m)}{\partial a_{j'}} \frac{\partial \Delta g_{prism}(x_m, z_m)}{\partial a_n}, \qquad j' = 1, 2, \dots, N. \tag{2.9}$$

$$X = da_n, \tag{2.10}$$

$$B = \sum_{m=1}^{N} [g_{obs}(x_m, z_m) - g_{basin}(x_m, z_m)] \frac{\partial \Delta g_{prism}(x_m, z_m)}{\partial a_{j'}}, j' = 1, 2, \dots, N. \tag{2.11}$$

Here, $a_n, n = 1, 2, \dots, N$ are depth parameters of prisms and da_n represents corresponding depth improvements. I is a diagonal matrix containing the diagonal elements of the matrix A.

The partial derivatives required in equations (2.9) and (2.11) are evaluated numerically, which involves the calculation of the rate of change of the gravity anomaly with respect to the thickness of each prism (Chakravarthi and Ramamma, 2013b). The improvements, da_n, solved from equation (2.8) are added to/or subtracted from existing depth parameters of the prisms and the process repeats until one of the conditions of its termination is fulfilled as described under section 1.4.3 in Chapter-I.

2.4 Description of software – MOD2DSD

Based on the algorithms of automatic modeling and inversion (sections 2.2.3 and 2.3), a GUI based software, MOD2DSD, coded in JAVA is developed in the space domain to analyze the gravity anomalies of 2-D sedimentary basins using EDF (Appendix 2.1). The software provides an option to the user to choose either automatic modeling or inversion to interpret the anomalies. This software is platform independent and works on any GUI based operating system with at least jdk 1.6 version installed. The software follows Model-View-Controller (MVC) pattern according to the structural relationship shown in Figure 2.3.

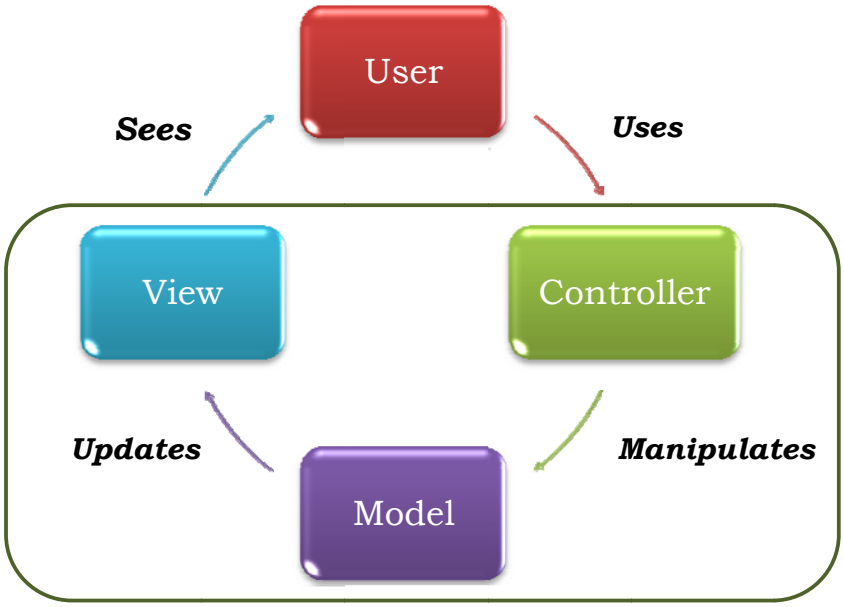

Figure 2.3 Structural relationships between Model, View and Controller (MVC)

The module pertaining to 'Model' computes the model gravity anomalies, and performs the business logic of respective algorithm (either automatic modeling or inversion as specified by user) to recover the basement structure. The 'View' module reads input parameters and shows

the output subsequent to the analysis. The 'Controller' passes the required actions to the model and view modules.

The advantage of the present code is that besides generating the output in both ASCII and graphical forms, it also shows i) the changes in depth structure, ii) fit between the observed and modeled gravity anomalies, iii) changes in misfit, and iv) variation of density contrast with depth against the iteration number in animated forms.

Upon execution, the view module of the code appears on the monitor as shown in Figure 2.4.

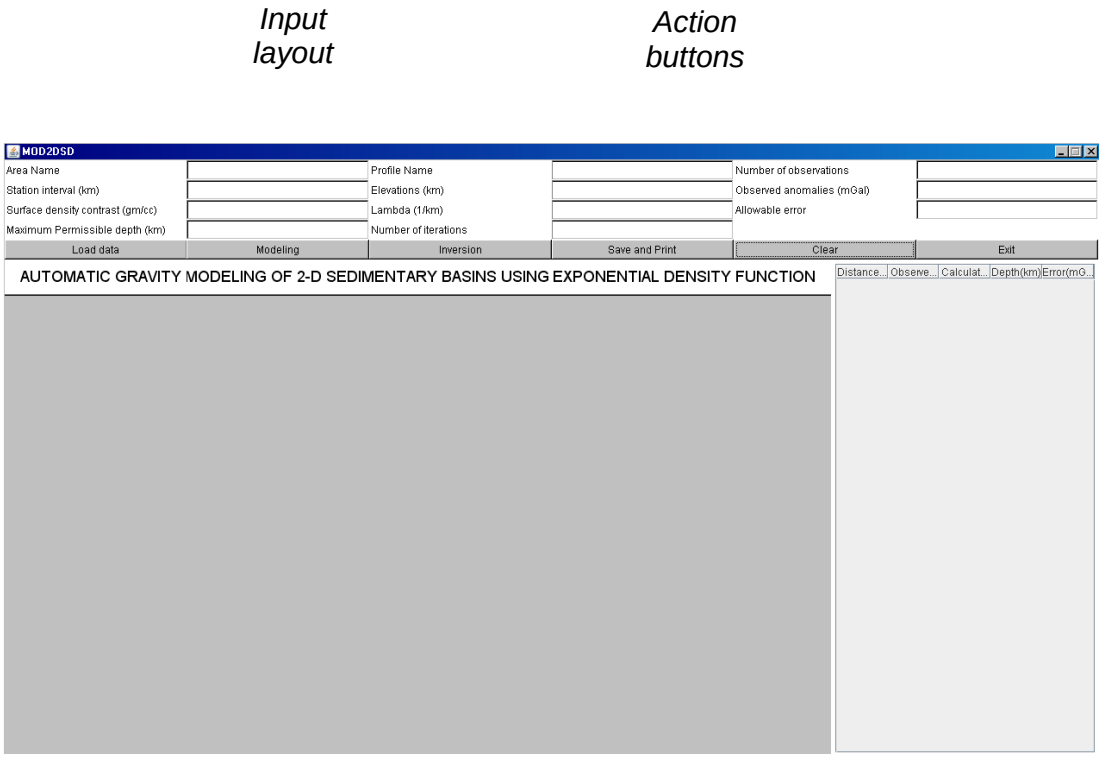

Figure 2.4 View module of MOD2DSD

The view module is structured into input layout, graphical layout and ASCII layout (Figure 2.4). The input layout consists of eleven fields

and six action buttons. The graphical layout, which serves as an interface between the user and the software, is further divided into the anomaly panel on top, structure panel in bottom, misfit panel and density-depth panel in the right. The ASCII layout displays the interpreted results in ASCII form.

The user enters the information pertaining to the area name, profile name, number of observations, station interval (km), station elevations (km), observed gravity anomalies (mGal), constants of exponential density function namely $\Delta\rho_0$ (g/cm^3) and λ (km^{-1}), allowable error, maximum permissible depth and number of iterations to be performed in respective fields in the input layout (Figure 2.4) and opts for either modeling or inversion for the analysis. Alternatively, the user can enter and save the input data in a Microsoft Office Excel sheet and invoke the same to the software by means of 'Load data' action button (Figure 2.4). 'Save and Print' action button enables the user to save the interpreted results and allows for printing.

2.5 Applications

Reliability of automatic modeling and inversion techniques discussed above is demonstrated by interpreting gravity anomalies due to a synthetic model of a sedimentary basin using both EDF and UDF. Further, interpreting the gravity anomalies over the San Jacinto graben, California, shows the practical utility of modeling and inversion. In both examples, the observer is on top of the topography at $z_j = 0$.

2.5.1 Automatic modeling

(a) Synthetic example – interpretation of noisy gravity anomalies

Figure 2.5a shows 40 equispaced noisy gravity anomalies in the interval $x_j \in$ [0 km, 40 km] produced by a synthetic model of a sedimentary basin, whose geometry is shown in Figure 2.5b. The structure resembles to a typical block faulted intracratonic rift basin filled with thick sectioned sediments. The anomalies of the structure are generated assuming that the density contrast of sediments within the basin decreases with depth according to equation (1.20) defined by $\Delta\rho_0 =$ -0.45 g/cm^3 and λ = 0.39 km^{-1} (Figure 2.5c). In this case, pseudorandom noise was Gaussian with zero mean and a standard deviation of 0.12 mGal. Further, the gravity profile runs transverse to the strike of the basin and extends farther away to stations resting on the basement. The anomaly shows asymmetric nature across the strike of the structure with large gradient observed over the western margin of the basin (Figure 2.5a). Negative gravity anomaly of the order of -26 mGal is observed at the 19th km on the profile (Figure 2.5a) over the depocentre.

The observed anomalies are subjected to analysis using the automatic modeling algorithm for which the software, MOD2DSD, had performed 139 iterations before it got terminated. The misfit function (equation 1.21) had reduced drastically from its initial value of 101.3 to 0.15 at the end of the 5th iteration and then progressively reached to almost zero at the end of the 139th iteration (Figure 2.5d). The modeled

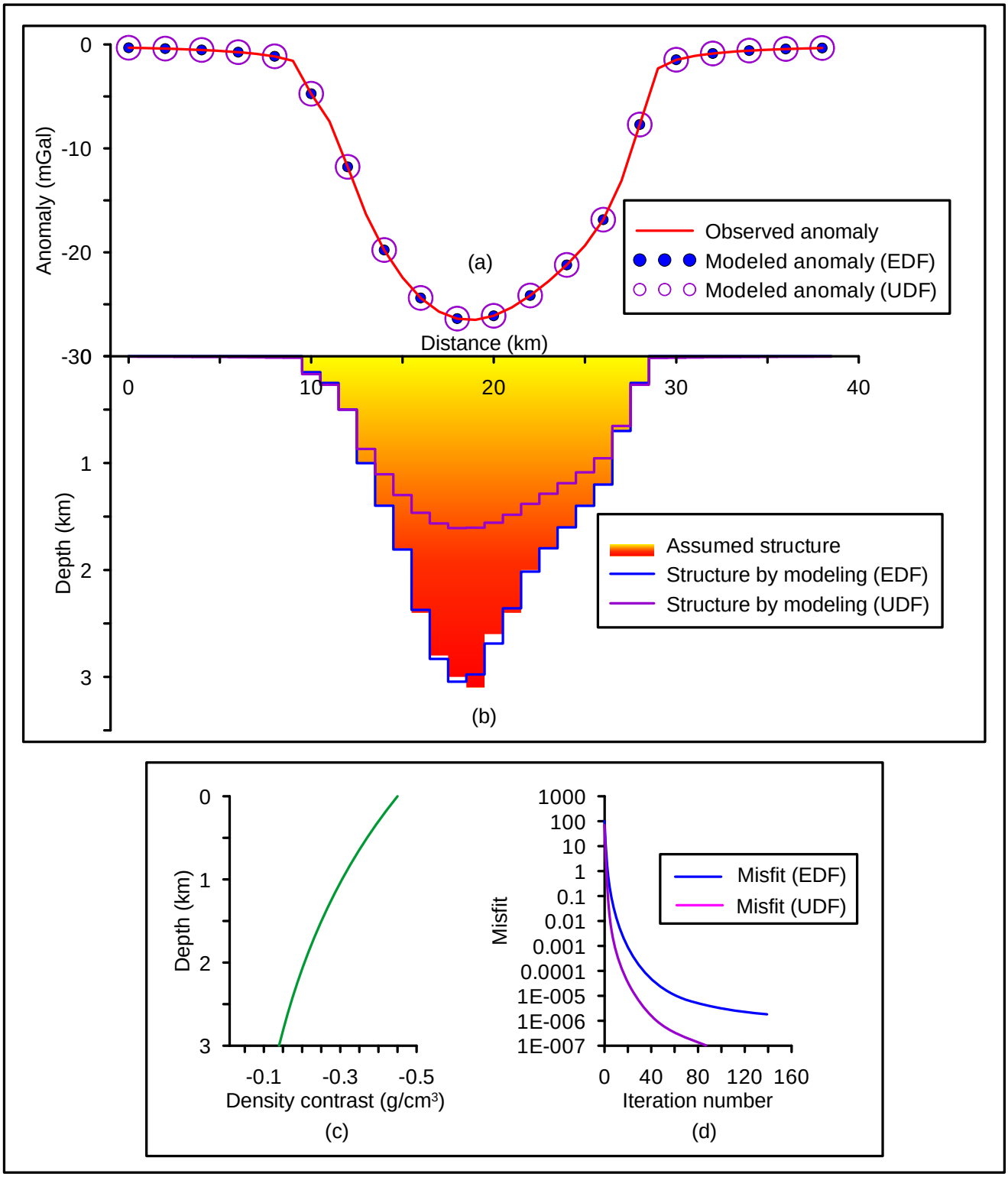

Figure 2.5 (a) Observed noisy gravity anomalies and modeled anomalies in case of EDF and UDF. (b) Assumed and estimated depth structures by automatic modeling. The tonal variation from yellow to red within structure indicates decrease in density contrast (absolute magnitude) with depth in case of EDF. (c) Prescribed EDF. (d) Variation of misfit with iteration number.

anomalies subsequent to interpretation (Figure 2.5a) closely mimic the observed anomalies. Figure 2.5d shows the variation of misfit (log scale) against the iteration number. By and large, the estimated structure (Figure 2.5b) after the analysis closely matches with the assumed one over the length of the profile except at the centre, where the estimated depth modestly deviated from the assumed depth. The maximum error in the depth between the assumed and estimated structure observed at the 19th km is 3.8%, which is negligible considering the presence of pseudorandom noise in the anomaly.

Furthermore, the same anomaly shown in Figure 2.5a is also interpreted assuming uniform density contrast (-0.45 g/cm^3) between the sedimentary infill and the basement rocks. In this case, the algorithm had performed 87 iterations before it got terminated as the misfit attained a value of almost zero from its initial value of 56.3 (Figure 2.5d). The modeled gravity anomaly and the inferred depth structure at the end of the 87th iteration are shown in Figures 2.5a and 2.5b respectively. In this case, the inferred structure is grossly underestimated to the tune of ~50% (Figure 2.5b). The maximum thickness of the basin estimated from the analysis is only 1.61 km against the assumed depth of 3.1 km.

Further, it can be noticed from Figure 2.5a that the modeled gravity anomaly of the structure obtained with EDF and UDF equally fit the observed anomaly, however, the inferred structure with EDF more or less mimics the actual one while it is not so with UDF.

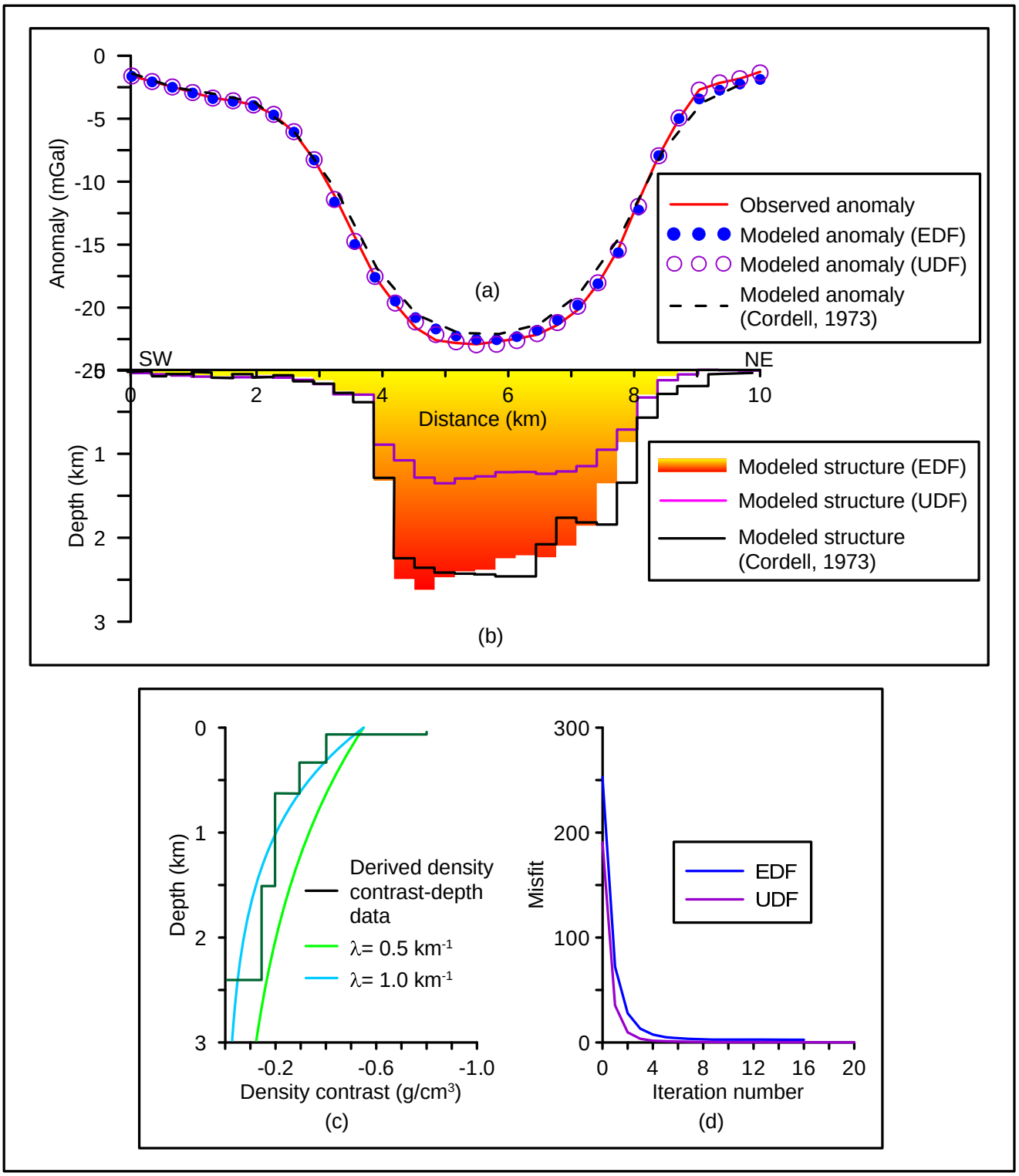

Figure 2.6 (a) Observed and modeled gravity anomalies in case of EDF and UDF, San Jacinto graben, California. Modeled gravity anomaly by Cordell (1973) is also shown. (b) Estimated depth structures by automatic modeling. The description for color gradation from yellow to red within structure is given in Figure 2.5b. Depth structure inferred by Cordell (1973) is also shown for comparison. (c) Derived density contrast-depth data (Fett, 1968) and fitted EDFs (Cordell, 1973). (d) Variation of misfit with iteration number.

(b) Field example – San Jacinto graben, California

The San Jacinto graben is bounded by two parallel branches of the San Jacinto fault, and has a northwesterly trend (Cordell, 1973). Country rock is a basement complex of pre-Tertiary schist and gneiss together with Cretaceous intrusive tonalities and granodiorites. Pliocene and Pleistocene detrital sedimentary rocks and Pleistocene and Holocene alluvium fill the graben, and the relatively lower density of this material accounts for the observed negative gravity anomaly (Figure 2.6a). On the basis of seismic refraction data, Fett (1968) determined the depth of the basement in the center of the graben as 2.4 km. Cordell (1973) had used two EDFs based on equation (1.20), one with $\Delta\rho_0$= -0.55 g/cm^3 and λ = 0.5 km^{-1} and the other with $\Delta\rho_0$ = -0.55 g/cm^3 and λ = 1.0 km^{-1} (Figure 2.6c) to describe the density contrast-depth data of the graben (step line in Figure 2.6c) derived from seismic refraction surveys. He demonstrated that the use of the former density model in the gravity analysis had yielded structural solution that was consistent with seismically derived information.

For the present study, the digitized observed gravity data of the graben is analyzed using both EDF and UDF following the principles of automatic modeling. For such a modeling the algorithm had performed 16 iterations in case of EDF and 20 iterations in case of UDF (Figure 2.6d). When EDF is used in modeling, the misfit showed a value of 252.8 for the initial structure and got reduced to 3.88 at the end of the 6th iteration and then gradually reached to a value of 2.6 (Figure 2.6d) at the end of the

16th iteration. The algorithm got terminated after the 16th iteration as the value of new misfit is more than its preceding value.

On the other hand, when UDF is used in modeling the misfit had reduced from 190.7 for the starting model to 0.75 at the end of the 6th iteration and finally reached to a value of 0.17 (Figure 2.6d) at the end of the 20th iteration. The new misfit beyond the 20th iteration attained a value more than its preceding value and hence the algorithm got terminated.

The modeled gravity anomaly in each case is shown in Figure 2.6a and corresponding estimated depth structures in Figure 2.6b respectively. It can be noticed from Figure 2.6a that the modeled gravity anomalies realized by both EDF and UDF equally explain the observed anomaly. The maximum depth to the basement obtained from the present method using EDF is 2.61 km, which compares reasonably well with the estimated depth of 2.44 km by Cordell (1973). On the other hand, the estimated maximum depth of 1.35 km obtained from automatic modeling using UDF is not consistent with known geologic information of the graben. The interpreted anomaly and the inferred structure of the graben by Cordell (1973) are also shown in Figures 2.6a and 2.6b for comparison.

2.5.2 Inversion

(a) Synthetic example – interpretation of noisy gravity anomalies

The validity of the proposed inversion scheme is demonstrated by interpreting the noisy gravity anomalies shown in Figure 2.7a using both

EDF and UDF. It is to be noted that the same anomaly is interpreted in the previous section 2.4.1a by automatic modeling to decipher the basement configuration.

In this case, the inversion algorithm of the software took 29 iterations before it got terminated as the resulting damping factor, δ (equation 1.26), attained an unusually large value (more than 12). The misfit function, J, which attained a maximum value of 101.3 for the initial model had reduced drastically to 0.005 at the end of the 3rd iteration and then reached to 1E-05 at the end of the 29th iteration (Figure 2.7c). The fit between the observed and modeled gravity anomalies at the end of the concluding iteration is satisfactory (Figure 2.7a). The estimated structure subsequent to inversion is shown in Figure 2.7b along with the assumed one. By and large, the algorithm had recovered the structure, however, with a few exceptions in and around the depocentre (Figure 2.7b). A maximum error of 3.7% between the assumed depth (2.6 km) and estimated depth (2.698 km) is observed at the 20th km on the profile (Figure 2.7b). Such an error is insignificant considering the presence of noise in the residual signal of the structure.

The same anomaly when analyzed using UDF the algorithm had performed only 6 iterations. In this case, the misfit had reduced from 69.5 for the starting model to 0.09 at the end of the 3rd iteration and finally reached to a value less than the predefined allowable error (1E-06) at the end of the 6th iteration (Figure 2.7c). The modeled gravity anomalies and

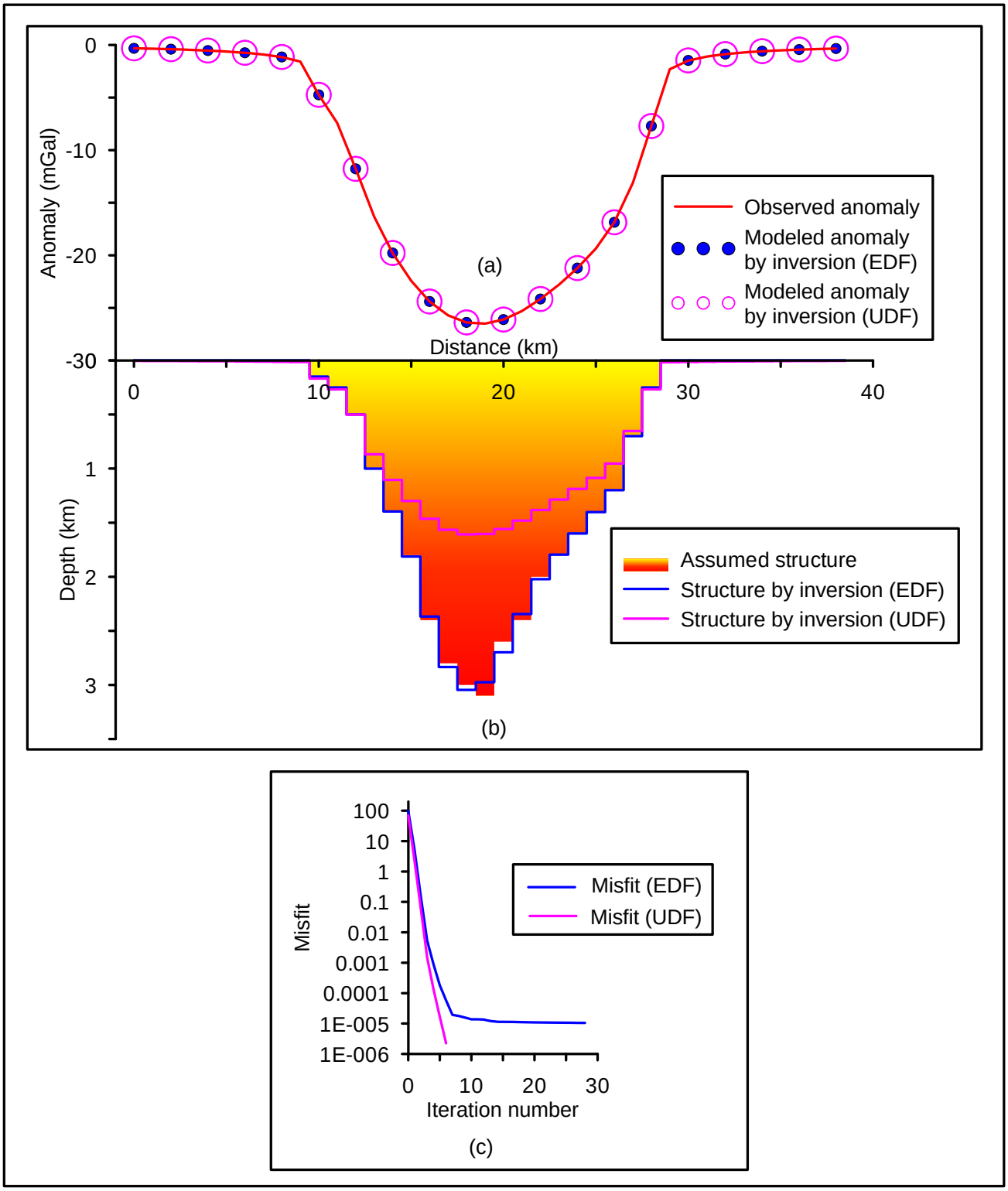

Figure 2.7 (a) Observed noisy gravity anomalies and modeled anomalies by inversion in case of EDF and UDF. (b) Assumed and estimated depth structures by inversion. The description for tonal variation from yellow to red within structure is given in Figure 2.5b. (c) Variation of misfit with iteration number.

the estimated depth structure subsequent to inversion are shown in Figures 2.7a and 2.7b, respectively. The maximum thickness of the basin estimated in this case is only 1.606 km. Although, the modeled gravity anomalies with UDF showed good correlation with the observed anomalies as in the case of EDF (Figure 2.7a), the inferred structure with UDF is foully underestimated as much as 50% (Figure 2.7b).

(b) Field example – San Jacinto graben, California

Figure 2.8 shows the results of gravity interpretation of the San Jacinto graben, California using EDF and UDF based on the principles of inversion. The inversion algorithm took 35 iterations in case of EDF and 16 iterations in case of UDF respectively. The damping factor, δ, attained a large value subsequent to respective concluding iterations, which had resulted in the termination of the algorithm. The variation of misfit with iteration in each case is shown in Figure 2.8c. The modeled gravity anomalies after the inversion in either case are shown in Figure 2.8a and the corresponding estimated structures in Figure 2.8b, respectively. A satisfactory fit between the observed and modeled gravity anomalies is noticed in case of EDF, the same is also repeated with UDF. However, the maximum thickness of the graben estimated from the inversion using EDF is 2.59 km, which is consistent with seismically derived depth (Fett, 1968), whereas the use of UDF in the analysis had resulted 1.36 km for the maximum thickness of the graben, which is not acceptable. The modeled anomalies and corresponding estimated structure by Cordell (1973) are also shown in Figure 2.8a and 2.8b for comparison.

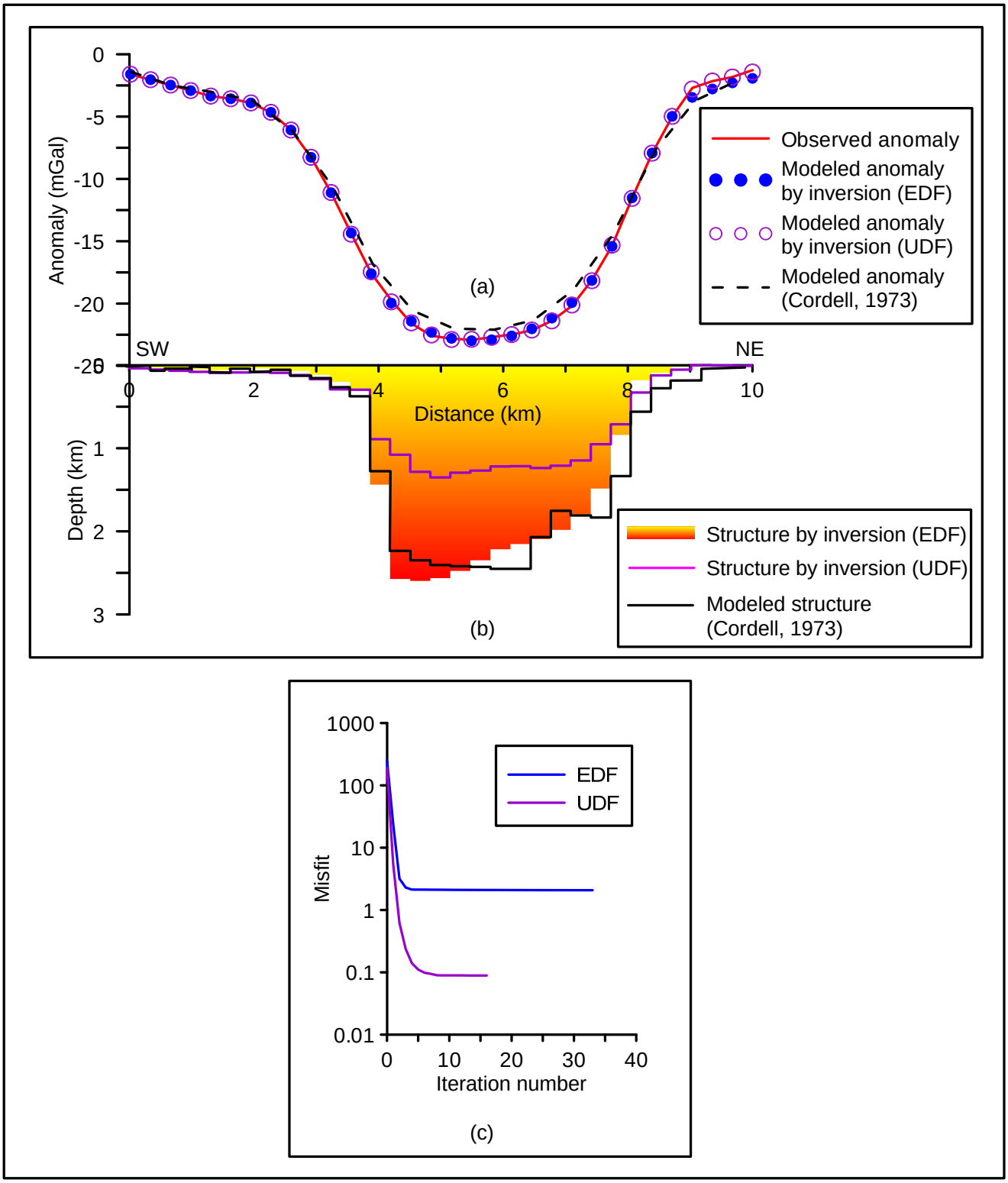

Figure 2.8 (a) Observed and modeled gravity anomalies by inversion using EDF and UDF, San Jacinto graben, California. Modeled anomalies by Cordell (1973) are also shown. (b) Estimated depth structures by inversion. The description of color gradation from yellow to red within the structure is given in Figure 2.5b. Interpreted model by Cordell (1973) is also shown for comparison. (c) Variation of misfit with iteration number.

2.5 Results and Discussion

The fully automated 2-D automatic modeling and inversion of gravity anomalies due to sedimentary basins on synthetic and field examples yield highly reliable results. In the case of synthetic example, modeling and inversion with EDF results the depth structure reasonably similar to the assumed one even in the presence of pseudorandom noise, whereas with UDF a distinct deviation in depth structure is noticed.

In modeling the field data of San Jacinto graben, California, using EDF the estimated depth structure agrees well with the information derived from seismic surveys. Further, the same field data of San Jacinto graben when subjected to inversion using EDF yielded more or less similar depth structure as that of automatic modeling. The inferred structural models (from both automatic modeling and inversion) of the graben using EDF and the one reported by Cordell (1973) showed more or less similar morphological features over the shoulders of the graben (Figures 2.6b and 2.8b). For e.g., all the three models divulge that the graben is bounded by steeply dipping fault system towards southwest and by gently dipping fault system towards northeast. However, the estimated models of the graben from present analysis (by automatic modeling and inversion) using EDF reveal progressive deepening of the basement towards the southwest, which however was not repeated in Cordell's (1973) interpretation (Figures 2.6b and 2.8b). The basement configuration derived from the present analysis appears to be more reliable than the one proposed by Cordell (1973) because in all the cases the modeled gravity

anomalies obtained with EDF showed remarkable correlation with the observed anomalies (Figures 2.6a and 2.8a) than the ones realized by Cordell (1973). On the other hand, when UDF is implemented in automatic modeling and inversion, it had resulted in inaccurate depth structures.

"MOD2DSD : A GUI based JAVA code for gravity analysis of 2-D sedimentary basins with exponential density function".

The system requirements to run the program are:-

Java Development Kit (jdk) 1.6 & above
RAM : 512 MB
Disk space : 70 MB

```
package com.mod2dsd.view;

import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class MOD2DSD_MainView extends Frame{

    /**

     */
    private static final long serialVersionUID = 1L;

    public static void main(String s[])
    {
        MOD2DSD_MainView cm = new MOD2DSD_MainView();
        cm.setSize(1280, 768);
        cm.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                JFrame frame = null;
                int r = JOptionPane.showConfirmDialog(
                        frame,
                        "Exit MOD2DSD ?",
                        "Confirm Exit ",
                        JOptionPane.YES_NO_OPTION);
                if(r == JOptionPane.YES_OPTION )
                    System.exit(0);
            }
        });
        cm.setTitle("MOD2DSD");
        cm.setResizable(false);
        cm.add(new MOD2DSD_MainPanel());
        cm.setVisible(true);

    }

}
```

```
package com.mod2dsd.view;

import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import javax.swing.JFileChooser;

import jxl.Cell;
import jxl.CellType;
import jxl.Sheet;
import jxl.Workbook;
import jxl.read.biff.BiffException;
```

```
public class MOD2DSD_MainPanel extends Panel {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    Panel p_North, p_West;
    public static Panel p_East;
    static Panel p_South;
    public static Panel p_Center;
    static TextField inputValues [] = new TextField[13];
    public static TextArea img = new TextArea(36,140);
    Button actionButton[] = new Button[6];
    Object rowdata[][]={};

    /**Field Area Name*/
    final static int AREA_FE = 0;
    /**Profile Name*/
    final static int PROFILE_NAME = 1;
    /**Number of observations*/
    final static int N_OBS = 2 ;
    /**Distance(km)*/
    final static int X_KM = 3;
    /**Elevations(km)*/
    final static int ELE_KM = 4;
    /**observed anomalies*/
    final static int NOB_GOB = 5;
    /** Surface density contrast (gm/cc) */
    final static int SD_POLY = 6;
    /**ALPHA(km)*/
    final static int ALPHA_ST = 7 ;
    /**Allowable Error*/
    final static int AL_ERR =   8;
    /**Maximum Depth*/
    final static int DEP_ZMAX = 9;
    /**Number of iteration values*/
    final static int NOB_ITER = 10;
    public MOD2DSD_MainPanel(){

        this.setLayout(new BorderLayout());
        p_North = new Panel();
        p_West = new Panel();
        p_East = new Panel ();
        p_South = new Panel();
        p_Center = new Panel();

        Label graphLabel = new Label("AUTOMATIC GRAVITY MODELING OF 2-D SEDIMENTARY BASINS USING
        EXPONENTIAL DENSITY FUNCTION", Label.CENTER);
        graphLabel.setFont(new Font("Arial", 10, 18));
        p_Center.add(graphLabel);
        for(int i = 0; i < 12; i++){
            inputValues[i] = new TextField();
        }

        actionButton[0] = new Button("Load data");
        actionButton[1] = new Button("Modeling");
        actionButton[2] = new Button("Inversion");
        actionButton[3] = new Button("Save and Print");
        actionButton[4] = new Button("Clear");
        actionButton[5] = new Button("Exit");

        this.populateNorthPanel();
        MOD2DSD_TableData.populateEastPanel(rowdata);
        this.add(p_North, BorderLayout.NORTH);
        p_Center.setSize(1000, 760);
        this.add(p_Center, BorderLayout.CENTER);
        img.setEditable(false);
        p_Center.add(img);
        this.add(p_East, BorderLayout.EAST);
        this.setVisible(true);
    }

    public void populateNorthPanel(){
        p_North.setLayout(new GridLayout(5,6));
```

```
        p_North.add(new Label("Area Name"));
        p_North.add(inputValues[0]);
        p_North.add(new Label("Profile Name"));
        p_North.add(inputValues[1]);
        p_North.add(new Label("Number of observations"));
        p_North.add(inputValues[2]);
        p_North.add(new Label("Station interval (km)"));
        p_North.add(inputValues[3]);
        p_North.add(new Label("Elevations (km)"));
        p_North.add(inputValues[4]);
        p_North.add(new Label("Observed anomalies (mGal)"));
        p_North.add(inputValues[5]);
        p_North.add(new Label("Surface density contrast (gm/cc)"));
        p_North.add(inputValues[6]);
        p_North.add(new Label("Lambda (1/km)"));
        p_North.add(inputValues[7]);
        p_North.add(new Label("Allowable error"));
        p_North.add(inputValues[8]);
        p_North.add(new Label("Maximum Permissible depth (km)"));
        p_North.add(inputValues[9]);
        p_North.add(new Label("Number of iterations"));
        p_North.add(inputValues[10]);


        p_North.add(new Label(""));
        p_North.add(new Label(""));
        p_North.add(actionButton[0]);
        p_North.add(actionButton[1]);
        p_North.add(actionButton[2]);
        p_North.add(actionButton[3]);
        p_North.add(actionButton[4]);
        p_North.add(actionButton[5]);

        actionButton[0].addActionListener(new com.mod2dsd.control.MOD2DSD_Control());
        actionButton[1].addActionListener(new com.mod2dsd.control.MOD2DSD_Control());
        actionButton[2].addActionListener(new com.mod2dsd.control.MOD2DSD_Control());
        actionButton[3].addActionListener(new com.mod2dsd.control.MOD2DSD_Control());
        actionButton[4].addActionListener(new com.mod2dsd.control.MOD2DSD_Control());
        actionButton[5].addActionListener(new com.mod2dsd.control.MOD2DSD_Control());
    }

    public static HashMap captureValues(){

        HashMap h_Map = new HashMap();
        try {

            h_Map.put("N_OBS", inputValues[N_OBS].getText());
            h_Map.put("SD_POLY", inputValues[SD_POLY].getText());
            h_Map.put("ALPHA_ST",inputValues[ALPHA_ST].getText());
            h_Map.put("X_KM", inputValues[X_KM].getText());
            h_Map.put("ELE_KM", inputValues[ELE_KM].getText());
            h_Map.put("NOB_GOB", inputValues[NOB_GOB].getText());
            h_Map.put("AL_ERR", inputValues[AL_ERR].getText());
            h_Map.put("DEP_ZMAX", inputValues[DEP_ZMAX].getText());
            h_Map.put("NOB_ITER", inputValues[NOB_ITER].getText());
            h_Map.put("AREA_FE",inputValues[AREA_FE].getText());
            h_Map.put("PROFILE_NAME",inputValues[PROFILE_NAME].getText());
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        return h_Map;

    }

    public static void clearPanel(TextArea p) {

        Graphics g = p.getGraphics();
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 1280, 650);
    }
```

```
    public static void loadData()throws IOException  {
        try{

            String current = System.getProperty("user.dir");
            JFileChooser chooser=new  JFileChooser(current);
            int returnVal = chooser.showOpenDialog(null);
            String  ele[],gobs[];
            String eleval="",gobsval="";
            Workbook w;

            if(returnVal == JFileChooser.APPROVE_OPTION) {
                File f = chooser.getSelectedFile();
                w = Workbook.getWorkbook(f);
                Sheet sheet = w.getSheet(0);
                ele = new String[sheet.getRows()+1];
                gobs = new String[sheet.getRows()+1];

                for (int j = 0; j < sheet.getColumns(); j++) {
                    for (int i = 1; i < sheet.getRows(); i++) {
                        Cell cell = sheet.getCell(j, i);
                        CellType type = cell.getType();
                        if (type == CellType.LABEL) {
                            //  System.out.println("I got a label "
                            //    + cell.getContents());
                            MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.AREA_FE].setText(cell.
getContents());

                        }

                        if (type == CellType.NUMBER) {
                            if (j==1){

                                MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.PROFILE_NAME].setT
ext(cell.getContents());

                            }
                            if (j==2){

                                MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.N_OBS].setText(cel
l.getContents());

                            }

                            if (j==3){
                                MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.X_KM].setText(cell
.getContents());
                            }

                            if (j==4){
                                ele[i] = cell.getContents()+",";
                                eleval = eleval+ele[i];
                            }

                            if (j==5){

                                gobs[i] = cell.getContents()+",";
                                gobsval = gobsval+gobs[i];
                            }

                            if (j==6){

                                MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.SD_POLY].setText(c
ell.getContents());

                            }

                            if (j==7){

                                MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.ALPHA_ST].setText(
cell.getContents());

                            }
                            if (j==8){

                                MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.AL_ERR].setText(ce
```

```
ll.getContents());
                                }
                                if (j==9){
                                    MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.DEP_ZMAX].setText(
cell.getContents());

                                }
                                if (j==10){
                                    MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.NOB_ITER].setText(
cell.getContents());

                                }

                                //System.out.println("I got a number "
                                //  + cell.getContents());
                            }

                        }
                    }


                    MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.ELE_KM].setText(""+eleval);
                    MOD2DSD_MainPanel.inputValues[MOD2DSD_MainPanel.NOB_GOB].setText(""+gobsval);

                }
            }
            catch (BiffException e) {
                e.printStackTrace();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }



        }

        public static void clearDefaultValues(){

            inputValues[N_OBS].setText("");
            inputValues[SD_POLY].setText("");
            inputValues[ALPHA_ST].setText("");
            inputValues[X_KM].setText("");
            inputValues[ELE_KM].setText("");
            inputValues[NOB_GOB].setText("");
            inputValues[AL_ERR].setText("");
            inputValues[DEP_ZMAX].setText("");
            inputValues[NOB_ITER].setText("");
            inputValues[AREA_FE].setText("");
            inputValues[PROFILE_NAME].setText("");
        }
}
```

```
package com.mod2dsd.view;

import java.applet.Applet;
import java.awt.*;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import java.text.DecimalFormat;

import com.mod2dsd.model.MOD2DSD_CalculateValues;
import com.mod2dsd.util.MOD2DSD_Utility;

public class MOD2DSD_Graph extends Applet {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    float maxX,maxY,diff;
    float maxZ;
```

```
double obs[];
public void drawGraph(Graphics2D g2) {

    g2.setFont(new Font("Arial", 20, 12));
    g2.setColor(Color.BLACK);

    g2.drawLine(70, 35, 1040, 35);
    g2.drawString("DISTANCE(km)", 385, 315);
    String []a={"A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
    String []b={"D", "E", "P", "T", "H", "(k", "m)"};

    for (int i = 0; i < a.length; i++) {

        g2.drawString(""+a[i], 120, 20 + 60 + ( i * 20 ) );
    }
    for (int i = 0; i < b.length; i++) {

        g2.drawString(""+b[i], 120, 20 + 350 + ( i * 20 ) );
    }

}


/**Plotting Axis Values*/
public void plot(Graphics2D g)
{
    g.setColor(Color.black);
    maxX = (float) MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.x);
    maxY = MOD2DSD_Utility.findMaximumNumber(MOD2DSD_CalculateValues.input_nob_gob);
    double inidep = MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.o_dep);
    maxZ = (float) inidep;
    diff = (float) ( MOD2DSD_CalculateValues.input_ddx_km / 2);
    diff = (float) (450 * diff / maxX);
    g.draw(new Line2D.Float(200-diff, 70,200-diff , 550 + 20));
    DecimalFormat df = new DecimalFormat("0.#");
    g.draw(new Line2D.Float(200, 320, 650, 320 ));
    g.drawString("0",194-diff,80);
    g.drawString(""+(int)maxY  ,165-diff,320);
    g.drawString("0", 175,330);
    g.drawString("|", 650, 318);
    g.drawString(""+df.format(MOD2DSD_CalculateValues.x[MOD2DSD_CalculateValues.input_n_obs]),
    650,305);
    float points = maxX/5;
    int yInterval=50;
    int zInterval=50;
    float xInterval=(float)
    (MOD2DSD_CalculateValues.x[MOD2DSD_CalculateValues.input_n_obs]/5);
    float xplot=0;

    for (float x = xInterval, j =1; x < 600; x+=xInterval)
    {
        xplot=xplot+xInterval;
        if(j>4)
            break;
        g.drawString("|",(float) (200+(450*x/maxX)), 318);
        g.drawString("" + df.format(xplot), (float) (200+(450*x/maxX))-3, 305);
        j++;
    }

    points = maxY/5;
    for (int x = yInterval, j =1; x < 250; x+=yInterval)
    {
        g.drawString("-",196-diff,50+x+20);
        g.drawString("" + (int)(points*j), 165-diff,50+x+20);
        j++;
    }

    float point =maxZ/5 ;
    for(int x = zInterval+250,j =1; x < 550; x+=zInterval)
    {
        g.drawString("-",196-diff,50+x+20);
        g.drawString("" +df.format(point*j),165-diff,50+x+20);
        j++;
    }
}
```

```
public void plotXYCoordinates (Graphics2D g2)
{
    float prevx = 200;
    float prevy = (float)( 50 + 20 + ( 250 * MOD2DSD_CalculateValues.o_GC[1] / maxY ) );
    float xpoint = 0;
    float ypoint = 0;
    float gypoint = 0;

    for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {

        xpoint = (float)( 450 * MOD2DSD_CalculateValues.x[k] / maxX);
        ypoint = (float)( ( 250 * MOD2DSD_CalculateValues.o_GC[k] / maxY ) );
        gypoint = (float)( (250 * MOD2DSD_CalculateValues.input_nob_gob[k] / maxY ) );
        g2.setFont(new Font("Arial", 20, 20));
        g2.setColor(Color.BLACK);
        g2.draw(new Line2D.Float(prevx, prevy, 200 + xpoint, 50 + ypoint + 20 ));
        g2.setColor(Color.BLUE);
        g2.setFont(new Font("Arial", 20, 40));
        g2.drawString(".", 200 + xpoint -6 , 50 + gypoint + 23);

        prevx = 200 + xpoint;
        prevy = 50 + ypoint + 20;

    }
}


public void plotPOS(Graphics2D g){


    g.setFont(new Font("Arial", 20, 12));
    g.setColor(Color.black);
    maxX = (float) MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.x);
    diff = (float) ( MOD2DSD_CalculateValues.input_ddx_km / 2);
    diff = (float) (450 * diff / maxX);
    g.draw(new Line2D.Float(200-diff, 35,200-diff , 550 + 20));
    double inidep = MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.o_dep);
    double pos[] = new double[MOD2DSD_CalculateValues.input_n_obs];
    double neg[] = new double[MOD2DSD_CalculateValues.input_n_obs];
    int m=1,n = 1;
    for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {

        if(MOD2DSD_CalculateValues.input_nob_gob[k]>0){
            pos[m] = MOD2DSD_CalculateValues.input_nob_gob[k];

            m=m+1;
        }
        else{
            neg[n] = MOD2DSD_CalculateValues.input_nob_gob[k];
            n=n+1;
        }
    }
    maxY = MOD2DSD_Utility.findMaximumNumber(neg);
    double maxY1 = (float) MOD2DSD_Utility.findMaximumNumber1(pos);
    maxZ = (float) inidep;
    DecimalFormat df = new DecimalFormat("0.#");
    g.drawString("0",170-diff,80);
    g.drawString("0", 170-diff,330);
    g.drawString("|", 650, 318);
    g.drawString(""+df.format(MOD2DSD_CalculateValues.x[MOD2DSD_CalculateValues.input_n_obs])
    ,650 ,305);

    int zInterval=50;
    float xInterval=(float)
    (MOD2DSD_CalculateValues.x[MOD2DSD_CalculateValues.input_n_obs]/5);
    float xplot=0;
    for (float x = xInterval, j =1; x < 600; x+=xInterval)
    {
        xplot=xplot+xInterval;
        if(j>4)
            break;
        g.drawString("|",(float) (200+(450*x/maxX)), 318);
        g.drawString("" + df.format(xplot), (float) (200+(450*x/maxX))-3, 305);
        j++;
```

```
        }

        g.drawString(""+df.format(maxY1),165-diff, 45);
        float point =maxZ/5 ;
        for(int x = zInterval+250,j =1; x < 550; x+=zInterval)
        {
            g.drawString("-",196-diff,50+x+20);
            g.drawString("" +df.format(point*j),165-diff,50+x+20);
            j++;
        }
    }



    public void plotXYCoordinatesPos (Graphics2D g2){

        float points = maxY/5;
        int yInterval = 50;
        g2.drawString(""+(int)maxY  ,165-diff,320);
        for (int x = yInterval, j =1; x < 250; x+=yInterval)
        {
            g2.drawString("-",196-diff,50+x+20);
            g2.drawString("" + (int)(points*j), 165-diff,50+x+20);
            j++;
        }

        double pos[] = new double[MOD2DSD_CalculateValues.input_n_obs];
        double neg[] = new double[MOD2DSD_CalculateValues.input_n_obs];
        int m=1,n = 1;
        for (int k = 1; k <=MOD2DSD_CalculateValues.input_n_obs; k++) {

            if(MOD2DSD_CalculateValues.input_nob_gob[k]>0){
                pos[m] = MOD2DSD_CalculateValues.input_nob_gob[k];

                m=m+1;
            }
            else{
                neg[n] = MOD2DSD_CalculateValues.input_nob_gob[k];
                n=n+1;
            }
        }

        double maxY1 = (float) MOD2DSD_Utility.findMaximumNumber1(pos);

        float prevx = 200;
        float prevy = 0;
        float xpoint = 0;
        float ypoint = 0;
        float gypoint = 0;
        if (MOD2DSD_CalculateValues.o_GC[1] < 0){
            prevy = 70 + (float)( ( 250 * (MOD2DSD_CalculateValues.input_nob_gob[1]) / maxY ) );
        }
        else{

            prevy = 70 - (float)( ( 35 * (MOD2DSD_CalculateValues.input_nob_gob[1]) / maxY1 ) );
        }

        for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {

            xpoint = (float)( 450 * MOD2DSD_CalculateValues.x[k] / maxX);

            if (MOD2DSD_CalculateValues.input_nob_gob[k] < 0){

                gypoint = 70 + (float)( (250 * MOD2DSD_CalculateValues.input_nob_gob[k] / maxY )
);
            }
            else{
                gypoint = 70 -(float)( (35 * MOD2DSD_CalculateValues.input_nob_gob[k] / maxY1 ) );
            }
            if(MOD2DSD_CalculateValues.o_GC[k] < 0){
                ypoint = 70 + (float)( ( 250 * (MOD2DSD_CalculateValues.o_GC[k]) / maxY ) );
            }
            else{
                ypoint = 70 - (float)( ( 35 * (MOD2DSD_CalculateValues.o_GC[k]) / maxY1 ) );
            }
```

```
            g2.setFont(new Font("Arial", 20, 20));
            g2.setColor(Color.BLACK);
            g2.draw(new Line2D.Float(prevx, prevy, 200 + xpoint,  ypoint  ));
            g2.setColor(Color.BLUE);
            g2.setFont(new Font("Arial", 20, 40));
            g2.drawString(".", 200 + xpoint -6 ,  gypoint +3);

            prevx = 200 + xpoint;
            prevy = ypoint ;

        }
    }


    public void plotZCoordinates (Graphics2D g2) {

        g2.setFont(new Font("Arial", 20, 12));
        float xpoint = 0;
        float xpoint1 = 0;
        float zpoint = 0;
        diff = (float) ( MOD2DSD_CalculateValues.input_ddx_km / 2);
        diff = (float) (450 * diff / maxX);
        maxY = MOD2DSD_Utility.findMaximumNumber(MOD2DSD_CalculateValues.input_nob_gob);
        g2.setColor(Color.RED);
        g2.fill(new Rectangle2D.Float(200-diff, 321, 450+diff+diff , 250 ));
        Color col[] =
        {Color.YELLOW,Color.PINK,Color.ORANGE,Color.PINK,Color.YELLOW,Color.ORANGE,Color.WHITE,Col
        or.PINK,Color.YELLOW,Color.ORANGE,Color.WHITE};
        Color col1[] =
        {Color.MAGENTA,Color.GREEN,Color.BLUE,Color.BLUE,Color.CYAN,Color.BLUE,Color.GREEN,Color.C
        YAN,Color.GREEN,Color.DARK_GRAY,Color.MAGENTA};
        for (int i = 0; i <1 ; i++){
            for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {

                if(i>11){
                    col[i] = Color.YELLOW;
                    col1[i] = Color.MAGENTA;
                }
                diff = (float) ( MOD2DSD_CalculateValues.input_ddx_km / 2);
                diff = (float) (450 * diff / maxX);
                xpoint = (float) (450 * MOD2DSD_CalculateValues.x[k] / maxX);
                if (k == 1){
                    GradientPaint gradient = new GradientPaint(10, 10, Color.YELLOW, 30, 200,
Color.MAGENTA, true);
                    g2.setPaint(gradient);
                    zpoint =(float) (250 * MOD2DSD_CalculateValues.o_dep[k] / maxZ);
                    g2.fill(new Rectangle2D.Float(200-diff, 320,diff+diff, zpoint ));
                }
                else{
                    if(k<MOD2DSD_CalculateValues.input_n_obs) {
                        xpoint1 = (float) (450 * MOD2DSD_CalculateValues.x[k+1] / maxX);
                    }
                    zpoint =(float) (250 * MOD2DSD_CalculateValues.o_dep[k] / maxZ);

                    GradientPaint gradient = new GradientPaint(10, 10, col[i], 30, 200, col1[i],
true);
                    g2.setPaint(gradient);
                    g2.fill(new Rectangle2D.Float(200+xpoint-diff ,320,xpoint1-xpoint, zpoint ));
                }
                if(k==MOD2DSD_CalculateValues.input_n_obs){
                    zpoint =(float) (250 * MOD2DSD_CalculateValues.o_dep[k]/ maxZ);
                    g2.fill(new Rectangle2D.Float(200+xpoint-diff, 320,diff+diff, zpoint ));

                }

            }
        }
        g2.setColor(Color.BLACK);
        g2.draw(new Line2D.Float(200-diff, 320, 650+diff, 320 ));
    }

    public void plotZCoordinatesele (Graphics2D g2) {

        g2.setFont(new Font("Arial", 20, 12));
```

```
        DecimalFormat f = new DecimalFormat("0.#");

        float maxEle = (float)
        MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.input_ele_km);
        maxEle = Math.abs(maxEle);
        float xpoint = 0;
        float prevx = 200;
        float elepoint[] = new float[MOD2DSD_CalculateValues.input_n_obs+1];
        float preelepoint[] = new float[MOD2DSD_CalculateValues.input_n_obs+1];
        float dis[] = new float[MOD2DSD_CalculateValues.input_n_obs+1];
        float predis[] = new float[MOD2DSD_CalculateValues.input_n_obs+1];
        float preele = 320 - (float)( ( 25 * Math.abs(MOD2DSD_CalculateValues.input_ele_km[1]) /
        maxEle ) );
        float xpoint1 = 0;
        float zpoint = 0;

        g2.setColor(Color.RED);
        g2.fill(new Rectangle2D.Float(200-diff, 295, 450+diff+diff , 275 ));
        Color col = Color.YELLOW;
        Color col1 = Color.MAGENTA;

        for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {

            diff = (float) ( MOD2DSD_CalculateValues.input_ddx_km / 2);
            diff = (float) (450 * diff / maxX);
            xpoint = (float) (450 * MOD2DSD_CalculateValues.x[k] / maxX);

            if (k == 1){
                GradientPaint gradient = new GradientPaint(10, 10, col, 30, 190, col1, true);
                g2.setPaint(gradient);
                zpoint =(float) (275 * MOD2DSD_CalculateValues.o_dep[k] / maxZ);
                g2.fill(new Rectangle2D.Float(200-diff, 295,diff+diff, zpoint ));
            }
            else{
                if(k<MOD2DSD_CalculateValues.input_n_obs) {
                    xpoint1 = (float) (450 * MOD2DSD_CalculateValues.x[k+1] / maxX);
                }
                zpoint =(float) (275 * MOD2DSD_CalculateValues.o_dep[k] / maxZ);
                GradientPaint gradient = new GradientPaint(10, 10, col, 30, 190, col1, true);
                g2.setPaint(gradient);
                g2.fill(new Rectangle2D.Float(200+xpoint-diff ,295,xpoint1-xpoint,zpoint ));
            }
            if(k==MOD2DSD_CalculateValues.input_n_obs){
                zpoint =(float) (275 * MOD2DSD_CalculateValues.o_dep[k]/ maxZ);
                g2.fill(new Rectangle2D.Float(150+xpoint-diff, 295,diff+diff, zpoint ));

            }

        }

        for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {
            g2.setColor(Color.BLACK);
            xpoint = (float) (450 * MOD2DSD_CalculateValues.x[k] / maxX);
            dis[k] = xpoint;
            elepoint[k] = 320 - (float) (25 * Math.abs(MOD2DSD_CalculateValues.input_ele_km[k]) /
maxEle);
            preelepoint[k]= preele;
            predis[k]= prevx;
            g2.draw(new Line2D.Float(prevx, preele, (float)200 + xpoint, elepoint[k]));
            prevx = 200+xpoint;
            preele = elepoint[k];
        }
        for(float i = (float)1.0; i<=25;){
            for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {
                g2.setColor(Color.WHITE);
                g2.draw(new Line2D.Float(predis[k]-diff, preelepoint[k]-i, (float)200+dis[k]+diff,
elepoint[k]-i));
            }
            i=(float) (i+0.5);

        }

        g2.setColor(Color.BLACK);
        float xInterval=(float)
        (MOD2DSD_CalculateValues.x[MOD2DSD_CalculateValues.input_n_obs]/5);
```

```
    float xplot=0;

    for (float x = xInterval, j =1; x < 600; x+=xInterval)
    {
        xplot=xplot+xInterval;
        if(j>4)
            break;
        g2.drawString("|",(float) (200+(450*x/maxX)), 318);
        g2.drawString("" + f.format(xplot), (float) (200+(450*x/maxX))-3, 305);
        j++;
    }
    g2.drawString("|", 650, 318);
    g2.drawString(""+f.format(MOD2DSD_CalculateValues.x[MOD2DSD_CalculateValues.input_n_obs])
    ,650 ,305);
    g2.setColor(Color.BLACK);
    g2.drawString("DISTANCE(km)", 385, 315);
}


public void drawOBJ(Graphics2D g2) {

    g2.setFont(new Font("Arial", 20, 12));
    g2.setColor(Color.BLACK);
    g2.drawLine(850, 35, 850, 580);
    g2.drawLine(70, 580, 1040, 580);
    g2.drawLine(1040, 35, 1040, 580);
    g2.drawLine(70, 35, 70, 580);

    g2.drawLine(890, 70, 890, 160);
    g2.drawLine(890, 160, 980, 160);
    g2.drawString("J", 870, 90);

    double maxOb = MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.o_funct);
    int ini = MOD2DSD_Utility.findMaximumNumber(MOD2DSD_CalculateValues.o_iter);
    if(ini == 5)
        ini = ini + 1;
    int maxiter = ( ini / 3 * 5 ) * 2;
    int point;
    int xInterval = 22;
    point = ( ( ini ) / 3 * 5 ) / 5;
    for (int x = xInterval, j = 1; x < 90 ; x += xInterval) {
        if(ini>1000){
            g2.drawString("'", 891 + x, 170);
            g2.drawString("" + (point * j), 890 + x-20 +(5*j) , 175);
        }
        else{
            g2.drawString("'", 891 + x, 170);
            g2.drawString("" + (point * j), 890 + x - 3, 175);
        }
        j++;
    }
    float prevx = 890;
    float prevy = 70;
    float xpoint = 0;
    float ypoint = 0;
    DecimalFormat d1= new DecimalFormat("0.#####");
    for (int i = 1; i <= MOD2DSD_CalculateValues.o_iter; i++) {

        xpoint = (float)( 250 * i / maxiter );
        ypoint = 70 + (float) - ( ( 90 * (MOD2DSD_CalculateValues.o_funct[i]) / maxOb ) );
        if(i == MOD2DSD_CalculateValues.o_iter){
            g2.draw(new Line2D.Float(prevx, prevy, 890 + xpoint - 4, 90 + ypoint));
        }
        else {
            g2.draw(new Line2D.Float(prevx, prevy, 890 + xpoint, 90 + ypoint));
        }
        prevx = 890 + xpoint;
        prevy = 90 + ypoint;

    }
    DecimalFormat d= new DecimalFormat("0.#");
    g2.drawString(" "+d.format(MOD2DSD_CalculateValues.o_funct[1]), 850, 70);
    g2.drawString("
    "+d1.format(MOD2DSD_CalculateValues.o_funct[MOD2DSD_CalculateValues.o_iter]), 890 +
    xpoint, 90 + ypoint);
```

```
    g2.setFont(new Font("Arial", 40,11));
    g2.drawString ("Iterations",920,186);
}

public void drawSd(Graphics2D g2) {
    g2.setColor(Color.black);
    g2.drawLine(850, 200, 1040, 200);
    g2.setColor(Color.red);
    g2.setFont(new Font("Arial", 20, 12));
    DecimalFormat d= new DecimalFormat("0.#");
    DecimalFormat d1= new DecimalFormat("0.###");
    double inidep = MOD2DSD_Utility .findMaximumNumber1(MOD2DSD_CalculateValues.o_dep);
    g2.draw(new Line2D.Float(890, 320, 890,  (float)(320+ (250*inidep/maxZ))));
    g2.drawString(""+d.format(inidep),870,(float)(320+ 250*inidep/maxZ));
    g2.drawString("-",890,(float)(320+ 250*inidep/maxZ)+2);
    g2.drawString("0",877 ,320);
    g2.drawLine(890, 320, 980, 320);
    double maxOb1 = MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.vsd);
    DecimalFormat df = new DecimalFormat("0.#");
    float points =maxZ/5 ;
    int zInterval=50;
    for(int x = zInterval+250,j =1; x < 550; x+=zInterval)
    {
        if(j>4)
            break;

        g2.drawString("-",890,50+x+22);
        g2.drawString("" +df.format(points*j),870,50+x+20);
        j++;

    }

    float prevx = 890+ (float) ( ( 90 * ( Math.abs(MOD2DSD_CalculateValues.vsd[1] )) / maxOb1
    ) );
    float prevy = 320;
    float xpoint = 0;
    float ypoint = 0;

    for (int i = 1; i <= MOD2DSD_CalculateValues.count; i++) {
        xpoint = (float)( 90 * Math.abs(MOD2DSD_CalculateValues.vsd[i]) / maxOb1 );
        ypoint = (float)( 250 * MOD2DSD_CalculateValues.dep[i] / maxZ );
        g2.setColor(Color.blue);
        g2.draw(new Line2D.Float(prevx, prevy, 890 + xpoint, 320 + ypoint));
        prevx = 890 + xpoint;
        prevy = 320 + ypoint;
    }
    g2.drawString(""+d.format(MOD2DSD_CalculateValues.vsd[1] ),875+ (float) ( ( 90 * (
    Math.abs(MOD2DSD_CalculateValues.vsd[1] )) / maxOb1 ) ) ,320 );
    g2.drawString(""+d1.format(MOD2DSD_CalculateValues.vsd[MOD2DSD_CalculateValues.count]
    ),890+ (float) ( ( 90 * (
    Math.abs(MOD2DSD_CalculateValues.vsd[MOD2DSD_CalculateValues.count] )) / maxOb1 ) ),
    320+(float)( 250 * inidep / maxZ ) );
    g2.setFont(new Font("Arial", 40,11));
    g2.drawString ("Density contrast",900,285);
    g2.drawString ("(gm/cc)",913,295);
    g2.drawString("Z(km)", 860,(float)( 320+((250*inidep/maxZ))/2));
}

public void idex(Graphics2D g)
{
    g.setColor(Color.BLUE);
    g.setFont(new Font("Arial", 20, 50));
    g.drawString(" ...    ",665,70);
    g.setFont(new Font("Arial", 20, 12));
    g.drawString("Observed anomalies",720,70);
    g.setColor(Color.BLACK);
    g.drawString("____:",685,87);
    g.drawString("Calculated anomalies",720,90);
    g.setColor(Color.black);
    GradientPaint gradient = new GradientPaint(10, 10, Color.yellow, 30, 50, Color.MAGENTA,
    true);
    g.setPaint(gradient);
    g.fillRect(685, 100, 35, 10);
    g.setColor(Color.BLACK);
    g.drawString(": Estimated Depth", 720, 110);
```

```
        g.drawString("Structure", 730, 130);

        g.drawLine(675, 55, 850, 55) ;
        g.drawLine(675, 55, 675, 135) ;
        g.drawLine(675, 135, 850, 135) ;
    }
}
```

```
package com.mod2dsd.view;

import java.awt.*;
import javax.swing.JScrollPane;
import javax.swing.JTable;


public class MOD2DSD_TableData extends Panel{
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public static  void populateEastPanel(Object rowData[][]) {
        com.mod2dsd.view.MOD2DSD_MainPanel.p_East.removeAll();
        Object columnNames[] = {"Distance(km)", "Observed anamolies (mGal)", "Calculated anamolies
        (mGal)", "Depth(km)","Error(mGal)"};
        JTable table = new JTable(rowData, columnNames);
        table.setPreferredScrollableViewportSize(new Dimension(300,550));
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);
        com.mod2dsd.view.MOD2DSD_MainPanel.p_East.add(scrollPane);
        com.mod2dsd.view.MOD2DSD_MainPanel.p_East.validate();
        com.mod2dsd.view.MOD2DSD_MainPanel.p_East.setVisible(true);
    }
}
```

```
package com.mod2dsd.model;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileOutputStream;
import java.text.DecimalFormat;
import java.util.HashMap;
import javax.imageio.ImageIO;

import com.mod2dsd.util.MOD2DSD_Utility;
import com.mod2dsd.view.MOD2DSD_MainPanel;

public class MOD2DSD_CalculateValues {


    public static int input_n_obs = 0;
    public static double input_ddx_km;
    public static double []input_nob_gob ;
    double input_sd_poly=0;
    double input_alambda=0;
    int input_nob_iter =0;
    double input_al_err = 0 ,input_zmax_km =0,input_zmin_km=0;
    double funct1;
    public static double []x;
    public static double []input_ele_km;
    public static Object obj[][] = null;
    public static double []vsd ;
    public static double []dep ;
    public static double []err ;
    public static double []o_GC ;
    public static double []o_dep ;
    public static double []o_funct ;
```

```
    static BufferedImage image;
    public static int o_iter,count;
    public static String input_area_name="";
    public static String input_profile_name="";
    double  PI = 22.0 / 7.0;
    double  GK = 13.3333;


    public void getAnamolyValues(HashMap h_Map) {

        try {

            input_n_obs = MOD2DSD_Utility.convertInteger((String)h_Map.get("N_OBS"));
            input_ddx_km = MOD2DSD_Utility.convertDouble((String)h_Map.get("X_KM"));
            input_ele_km = MOD2DSD_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
            input_nob_gob = MOD2DSD_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
            input_sd_poly = MOD2DSD_Utility.convertDouble((String)h_Map.get("SD_POLY"));
            input_alambda = MOD2DSD_Utility.convertDouble((String)h_Map.get("ALPHA_ST"));
            input_al_err =  MOD2DSD_Utility.convertDouble((String)h_Map.get("AL_ERR"));
            input_zmax_km =  MOD2DSD_Utility.convertDouble((String)h_Map.get("DEP_ZMAX"));
            input_nob_iter = MOD2DSD_Utility.convertInteger((String)h_Map.get("NOB_ITER"));
            input_area_name = MOD2DSD_Utility.convertString((String)h_Map.get("AREA_FE"));
            input_profile_name = MOD2DSD_Utility.convertString((String)h_Map.get("PROFILE_NAME"));
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    public void getForCal(){
        o_GC  = new double[input_n_obs+1];
        o_dep = new double[input_n_obs+1];
        x = new double[input_n_obs+1];
        o_funct  = new double[input_nob_iter+1];
        input_zmin_km=0;
        err = new double[input_n_obs+1];
        double dcz[] = new double[input_n_obs+1];

        for (int  KL=1; KL<= input_n_obs; KL++) {
            x[KL] = (KL-1) * input_ddx_km;
        }
        double  T=(x[2]-x[1])/2.0;
        for(int KK=1 ;KK <= input_n_obs;KK++) {
            if(input_alambda==0){
                o_dep[KK] = input_nob_gob[KK]/(GK*PI*input_sd_poly) ;
            }
            else{
                o_dep[KK] = (1 / input_alambda) * Math.log(1 + ((input_alambda *
input_nob_gob[KK]) / (GK * PI * input_sd_poly)));
            }

        }
        getGbasin(input_n_obs,x,input_ele_km,o_dep,input_sd_poly,input_alambda,input_ddx_km,T,o_GC
        );
        funct1=0;
        for(int  K=1 ;K<=input_n_obs;K++) {
            err[K] = input_nob_gob[K]-o_GC[K];
            funct1=funct1+Math.pow((input_nob_gob[K]-o_GC[K]),2);
        }

        for(int iter=1 ; iter<=input_nob_iter;iter++)  {

            setGraphValues(input_n_obs,iter, x, input_nob_gob, o_GC,o_dep,
funct1,err,input_area_name);
            for(int KK=1;KK<=input_n_obs;KK++) {
                err[KK] = input_nob_gob[KK]-o_GC[KK];
                if(input_alambda==0){
                    o_dep[KK] = o_dep[KK]+(err[KK]/(GK*PI*input_sd_poly));
                }
                else{
                    dcz[KK] = (input_sd_poly * Math.exp(input_alambda * o_dep[KK]));
                    o_dep[KK] = o_dep[KK] + (err[KK] / (GK * PI * dcz[KK]));
                }

                if(o_dep[KK] <= input_zmin_km)
```

```
                    o_dep[KK] = input_zmin_km;

                if(o_dep[KK] > input_zmax_km )
                    o_dep[KK] = input_zmax_km;
            }
            getGbasin(input_n_obs,x,input_ele_km,o_dep,input_sd_poly,input_alambda,input_ddx_km,T,
o_GC);
            double  funct2 = 0.0;
            for(int LI = 1 ; LI <= input_n_obs ; LI++) {
                funct2=funct2+Math.pow(input_nob_gob[LI] - o_GC[LI],2);
            }
            o_iter = iter;
            o_funct[iter] = funct1;

            denCal(input_sd_poly,input_alambda);
            drawGraph();

            if (funct2 - funct1 < 0 || funct2 - funct1 == 0) {

                if (funct2 - input_al_err <= 0) {

                    break;
                }

                else {
                    funct1 = funct2;
                }
            }

            else if (funct2 - funct1 > 0 ) {
                break;
            }
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }


    public void invcal(){

        double dpar = 0.1;
        double  ALER = input_al_err;
        o_GC  = new double[input_n_obs + 1];
        o_dep = new double[input_n_obs + 1];
        o_funct  = new double[input_nob_iter +5];
        x = new double[input_n_obs+1];
        err = new double[input_n_obs + 1];
        double err1[] = new double[input_n_obs + 1];
        for (int i = 0; i <= input_n_obs; i++){

            o_dep[i]=0;
            err[i]=0;
            err1[i]=0;
        }

        double []par = new double[input_n_obs + 1];
        double []par1 = new double[input_n_obs + 1];
        double []par2 = new double[input_n_obs + 1];
        double []dupar = new double[input_n_obs + 1];
        double []g1 = new double[input_n_obs + 1];
        double []g2 = new double[input_n_obs + 1];
        double [][]st = new double[input_n_obs + 1][input_n_obs + 1];
        double []actz = new double[input_n_obs + 1];
        double []b = new double[input_n_obs + 1];
        double []KS = new double[2];
        double LAMBDA = 0.5;
        int np1 = input_n_obs + 1;
        double [][]p = new double[input_n_obs+1][np1+1];
        for (int  KL=1; KL<= input_n_obs; KL++) {
            x[KL] = (KL-1) * input_ddx_km;
        }
        double  t = (x[2] - x[1]) / 2.0;
```

```
        for (int KK = 1 ; KK <= input_n_obs; KK++) {

            if(input_alambda==0){
                par[KK] = input_nob_gob[KK]/(GK*PI*input_sd_poly) ;
            }
            else
            {
                par[KK] = (1 / input_alambda) * Math.log(1 + ((input_alambda * input_nob_gob[KK])
/ (GK * PI * input_sd_poly)));
            }
        }
        getGbasin(input_n_obs,x,input_ele_km,par,input_sd_poly,input_alambda,input_ddx_km,t,o_GC);
        funct1=0.0;
        for(int K = 1 ; K <= input_n_obs; K++) {
            err[K] = input_nob_gob[K] - o_GC[K];
            funct1 = funct1 + Math.pow((input_nob_gob[K] - o_GC[K]), 2);

        }

        int iter  = 0;
        double funct2 = 0.0;
        while(iter< input_nob_iter)     {
            iter = iter + 1;
            setGraphValues(input_n_obs,iter, x, input_nob_gob, o_GC,o_dep,  funct1,err,
input_area_name);
            for(int  KJ = 1; KJ <= input_n_obs; KJ++) {
                actz[KJ] = par[KJ];
            }

            for(int K = 1;K <= input_n_obs; K++) {
                par1[K] = par[K];
            }
            for (int  I = 1; I <= input_n_obs; I++) {
                par1[I] = par[I] + dpar/2.0;
                getGbasin(input_n_obs,x,input_ele_km,par1,input_sd_poly,input_alambda,input_ddx_km
,t,g1);
                par1[I] = par[I] - dpar/2.0;
                getGbasin(input_n_obs,x,input_ele_km,par1,input_sd_poly,input_alambda,input_ddx_km
,t,g2);
                for(int K=1;K<=input_n_obs;K++) {
                    st[I][K] = (g1[K] - g2[K]) /dpar;
                }
            }
            for (int  J=1; J <= np1; J++) {
                for (int I=1; I <= input_n_obs;I++) {
                    p[I][J]=0.0;
                }
            }
            for(int J = 1;J <= input_n_obs;J++)  {
                for(int I = 1;I <= input_n_obs;I++) {
                    for(int K = 1;K <= input_n_obs;K++) {
                        p[I][J] = p[I][J] + st[I][K] * st[J][K];
                    }
                }
            }
            for (int J = 1; J <= input_n_obs;J++) {
                for (int K = 1; K <= input_n_obs;K++) {
                    p[J][np1] = p[J][np1] + err[K] * st[J][K];
                }
            }

            do {
                double  CON = LAMBDA + 1.0;
                for(int I = 1; I <= input_n_obs;I++) {
                    dupar[I] = par[I];
                }
                for (int L = 1;L <= input_n_obs; L++) {

                    for (int J = 1;J <= input_n_obs; J++) {

                        if (L - J == 0) {
                            p[L][J] = p[L][J] * CON;
                        }
                    }
```

```
                }
                SIMEQ(p,b,input_n_obs,KS);
                for (int I=1;  I <= input_n_obs; I++) {
                    par2[I] = dupar[I] + b[I];
                }
                for (int KK = 1; KK <= input_n_obs; KK++) {
                    if(par2[KK] <= input_zmin_km)
                        par2[KK] = input_zmin_km;
                    if(par2[KK] > input_zmax_km)
                        par2[KK] = input_zmax_km;
                }

                getGbasin(input_n_obs,x,input_ele_km,par2,input_sd_poly,input_alambda,input_ddx_km
,t,o_GC);
                funct2 = 0.0;
                for (int K = 1; K <= input_n_obs; K++) {
                    err[K] = input_nob_gob[K] - o_GC[K];
                    funct2 = funct2 + Math.pow(err[K],2);
                }
                if (funct1 - funct2 < 0) {

                    LAMBDA = LAMBDA * 2;

                    if (LAMBDA - 13 < 0) {

                        for (int I = 1; I <= input_n_obs; I++) {

                            for (int J = 1; J <= input_n_obs; J++) {

                                if(I - J == 0)
                                    p[I][J] = p[I][J] / CON;
                            }
                        }
                    }
                    else
                        break;
                }

            }while (funct1 <= funct2);

            o_iter = iter;
            o_funct[iter] = funct1;
            funct1 = funct2;

            for(int I = 1; I <= input_n_obs; I++) {
                par[I] = par2[I];
            }
            for(int I = 1; I <= input_n_obs; I++) {
                o_dep[I] = par2[I];
            }
            LAMBDA = LAMBDA / 2.0;
            denCal(input_sd_poly,input_alambda);
            drawGraph();

            if (funct2 - funct1 <= 0 ) {
                if (funct2 - ALER <= 0) {
                    break;
                }
                else  {
                    funct1 = funct2;
                }
            }
            else
                break;

        }
    }

    public static void denCal(double sd,double alpha){
        int i = 1;
        double z1=0;
        double z2 = MOD2DSD_Utility .findMaximumNumber1(MOD2DSD_CalculateValues.o_dep);
        vsd = new double[(int) Math.pow(input_n_obs,2)];
        dep = new double[(int) Math.pow(input_n_obs,2)];
        while(z1 <= z2){
```

```
        double dc = (sd * Math.exp(alpha * z1));
        vsd[i] = dc;
        dep[i] = z1;

        z1 = z1+0.1;
        i++;
    }
    count = i;
    vsd[count] = (sd * Math.exp(alpha * z2));
    dep[count] = z2;

}


public double [] getGbasin(int n,double []x,double []ele,double []z,double sd,double al,double
ddx,double t,double []gc) {
    double xx = 0,z2 = 0,gff,ggc=0;
    double z1 = 0.0;
    double dz = 0,dc = 0;
    double GK = 13.3333;
    for(int JJ = 1 ; JJ <= n ;JJ++){
        gc[JJ]=0.0;
    }
    for(int II = 1 ; II <= n ; II++){
        for(int LL = 1 ; LL <= n; LL++) {
            xx = x[II] - x[LL];
            z2 = z[LL];

            double dx = ddx/10;
            double zb = z2 - z1;

            double []zk = new double[1000];
            double []gs = new double[1000];

            int nd = (int)(zb / dx) + 1;
            int n1 = nd / 2;
            if (nd - (2 * n1 ) < 0 || nd - ( 2 * n1 ) > 0) {
                nd = nd + 1;
            }
            dz = zb / nd;
            int n2 = nd + 1;
            for (int JZ = 1 ; JZ <= n2; JZ++) {
                zk[JZ] = z1 + dz * (JZ - 1);
            }

            for (int JZ = 1; JZ <= n2; JZ++) {
                dc = (sd * Math.exp(al * zk[JZ]));
                double t1 = GK * dc;
                double t2 = Math.atan(((xx + t)) / (zk[JZ]-ele[II]));
                double t3 = Math.atan(((xx - t)) / (zk[JZ]-ele[II]));
                gs[JZ] = t1 * (t2 - t3);//(gg2[1] + gg2[2]) / 2.0;
            }
            gff = SIMP(gs,zk,n2,ggc);
            gc[II] = gc[II] + gff;

        }
    }
    return gc;
}

public double SIMP(double []gs,double []z,int n,double ggc) {
    double dz = z[2]-z[1];
    double sum1 = 0.0;
    double sum2 = 0.0;
    int n1 = n / 2;
    int n4 = n1 - 1;
    for(int I = 1; I <= n1; I++) {
        int n2 = 2 * I;
        sum1 = sum1 + gs[n2];
    }
    for(int I = 1; I <= n4; I++) {
        int n3 = 2 * I +1;
        sum2 = sum2 + gs[n3];
    }
    ggc = gs[1]+4*sum1+2*sum2+gs[n];
```

```
    ggc = ggc * dz / 3.0;
    return ggc;
}


public static double []SIMEQ(double p[][], double b[], int n, double KS[]) {

    int I = n + 1;
    double []a = new double[n*n+1];

    for (int I1 = 1; I1 <= n; I1++) {

        for (int I2 = 1; I2 <= n; I2++) {
            int I3 = (I1 - 1) * n + I2;
            a[I3] = p[I2][I1];
        }
    }

    for (int I4 = 1;I4 <= n; I4++) {
        b[I4] = p[I4][I];
    }
    double tol = 0;
    KS[0] = 0;
    int JJ = - n;
    int IT;
    int NY = 0;
    for (int J = 1;J <= n; J++) {
        int JY = J + 1;
        JJ = JJ + n + 1;
        double biga = 0;
        IT = JJ - J;
        int imax = 0;
        for (int i = J; i <= n; i++) {
            int IJ = IT + i;
            if (Math.abs(biga) - Math.abs(a[IJ]) < 0) {

                biga = a[IJ];
                imax = i;
            }
        }
        int I1 = 0;
        if (Math.abs(biga) - tol <= 0) {
            KS[1] = 1;
            return KS;
        }
        else {
            I1 = J + n * (J - 2);
            IT = imax - J;
        }

        double save;
        for (int K = J;K <= n; K++) {
            I1 = I1 + n;
            int I2 = I1 + IT;
            save = a[I1];
            a[I1] = a[I2];
            a[I2] = save;
            a[I1] = a[I1] / biga;
        }
        save = b[imax];
        b[imax] = b[J];
        b[J] = save / biga;
        int IQS = 0;

        if (J - n < 0 || J - n > 0) {

            IQS = n * (J - 1);
            for (int IX = JY;IX <= n; IX++) {
                int IXJ = IQS + IX;
                IT = J - IX;
                for (int JX = JY;JX <= n; JX++) {
                    int IXJX = n * (JX - 1) + IX;
                    int JJX = IXJX + IT;
                    a[IXJX] = a[IXJX] - (a[IXJ] * a[JJX]);
                }
```

```
                b[IX] = b[IX] - (b[J] * a[IXJ]);
            }
        }
    }
    NY = n - 1;
    IT = n * n;

    for (int J = 1;J <= NY; J++) {
        int ia = IT - J;
        int ib = n - J;
        int ic = n;
        for (int K = 1;K <= J; K++) {
            b[ib] = b[ib] - a[ia] * b[ic];
            ia = ia - n;
            ic = ic - 1;
        }
    }
    return b;
}



public static void  setGraphValues(int i_no_obs,int ite,double []dis,double []GOBS,double
[]GCAL,double []Dep, double funct,double []ERR,String Area_fe) {

    obj = new Object[i_no_obs+21][5];

    DecimalFormat df =new DecimalFormat("0.###");
    for(int K=1;K<=i_no_obs;K++){

        obj[K][0]= "" + dis[K];
        obj[K][1]= "" + df.format(GOBS[K]);
        obj[K][2]= "" + df.format(GCAL[K]);
        obj[K][3]= "" + df.format(Dep[K]);
        obj[K][4]= "" + df.format(ERR[K]);
    }

    obj[0][0] ="ITERATION";
    obj[0][1] = "=" +" "+ite;
}
public static void drawGraph(){

    final com.mod2dsd.view.MOD2DSD_Graph dg = new com.mod2dsd.view.MOD2DSD_Graph();
    double pos[] = new double[MOD2DSD_CalculateValues.input_n_obs];
    int m=1;
    for (int k = 1; k <= MOD2DSD_CalculateValues.input_n_obs; k++) {

        if(MOD2DSD_CalculateValues.input_nob_gob[k]>0){
            pos[m] = MOD2DSD_CalculateValues.input_nob_gob[k];

            m=m+1;
        }

    }

    try
    {
        int width = 1080;
        int height = 650;
        BufferedImage buffer = new BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);

        Graphics g1= buffer.createGraphics();
        g1.setColor(Color.WHITE);
        g1.fillRect(0,0,width,height);
        Graphics2D g2 = (Graphics2D)g1 ;

        if(pos[1]>0){
            dg.plotPOS(g2);
            dg.plotXYCoordinatesPos(g2);
        }
        else{
            dg.plot(g2);
            dg.plotXYCoordinates(g2);
        }
```

```
            dg.drawGraph(g2);
            float maxEle = (float)
MOD2DSD_Utility.findMaximumNumber1(MOD2DSD_CalculateValues.input_ele_km);
            if(maxEle==0)
                dg.plotZCoordinates(g2);
            else
                dg.plotZCoordinatesele(g2);

            dg.drawOBJ(g2);
            dg.idex(g2);
            dg.drawSd(g2);

            FileOutputStream os = new FileOutputStream( MOD2DSD_CalculateValues.input_area_name
+".jpg");
            ImageIO.write(buffer, "jpg", os);
            os.close();

            String path =  MOD2DSD_CalculateValues.input_area_name  +".jpg";
            image = ImageIO.read(new File(path));

            Graphics g_image = MOD2DSD_MainPanel.img.getGraphics();
            g_image.drawImage(image, -60,-30,image.getWidth(), image.getHeight(),dg);

            MouseListener ml3 = new MouseAdapter(){
                public void mouseClicked(MouseEvent e){
                    Graphics g_image = MOD2DSD_MainPanel.img.getGraphics();
                    g_image.drawImage(image, -60,-30,image.getWidth(), image.getHeight(),dg);
                }
            };
            MOD2DSD_MainPanel.img.addMouseListener(ml3);
        }
        catch (Exception e2) {

            e2.printStackTrace();
        }
    }
}
```

```
package com.mod2dsd.control;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import com.mod2dsd.model.MOD2DSD_CalculateValues;
import com.mod2dsd.view.MOD2DSD_MainPanel;

public class MOD2DSD_Control implements ActionListener{

    com.mod2dsd.model.MOD2DSD_CalculateValues cv = new
    com.mod2dsd.model.MOD2DSD_CalculateValues();
    public static boolean success = false;
    Object rowdata[][]={};
    public void actionPerformed(ActionEvent ae) {

        if(ae.getActionCommand().equals("Modeling")) {
            com.mod2dsd.view.MOD2DSD_MainPanel.clearPanel(MOD2DSD_MainPanel.img);
            cv.getAnamolyValues(com.mod2dsd.view.MOD2DSD_MainPanel.captureValues());
            cv.getForCal();
            com.mod2dsd.view.MOD2DSD_TableData.populateEastPanel(MOD2DSD_CalculateValues.obj);
            com.mod2dsd.view.MOD2DSD_MainPanel.p_East.repaint();
            com.mod2dsd.view.MOD2DSD_MainView mv = new com.mod2dsd.view.MOD2DSD_MainView();
            mv.setResizable(true);
        }else if(ae.getActionCommand().equals("Inversion")) {
            com.mod2dsd.view.MOD2DSD_MainPanel.clearPanel(MOD2DSD_MainPanel.img);
            cv.getAnamolyValues(com.mod2dsd.view.MOD2DSD_MainPanel.captureValues());
            cv.invcal();
```

```
            com.mod2dsd.view.MOD2DSD_TableData.populateEastPanel(MOD2DSD_CalculateValues.obj);
            com.mod2dsd.view.MOD2DSD_MainPanel.p_East.repaint();
            com.mod2dsd.view.MOD2DSD_MainView mv = new com.mod2dsd.view.MOD2DSD_MainView();
            mv.setResizable(true);
        }else if(ae.getActionCommand().equals("Save and Print")){

            try
            {
                MOD2DSD_PrintValues.printGraphValues();

            }
            catch(Exception e1) {

                e1.printStackTrace();
            }
        }else if(ae.getActionCommand().equals("Load data")){

            try {
                MOD2DSD_MainPanel.loadData();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

        }else if(ae.getActionCommand().equals("Clear")){

            MOD2DSD_MainPanel.clearDefaultValues();
            MOD2DSD_MainPanel.clearPanel(MOD2DSD_MainPanel.img);
            com.mod2dsd.view.MOD2DSD_TableData.populateEastPanel(rowdata);
        }
        else if(ae.getActionCommand().equals("Exit")){
            JFrame frame = null;

            int r = JOptionPane.showConfirmDialog(
                    frame,
                    "Exit MOD2DSD ?",
                    "Confirm Exit ",
                    JOptionPane.YES_NO_OPTION);
            if(r == JOptionPane.YES_OPTION ){
                if(success==false){
                    String fileName = MOD2DSD_CalculateValues.input_area_name+".jpg";
                    File f = new File(fileName);
                    f.delete();
                }
                System.exit(0);
            }
        }
    }


}
```

```
package com.mod2dsd.control;

import java.io.File;
import java.io.FileWriter;
import java.text.DecimalFormat;
import javax.swing.JFileChooser;
import com.mod2dsd.model.MOD2DSD_CalculateValues;

public class MOD2DSD_PrintValues {

    public static void printGraphValues() throws Exception {

        try{
            String current = System.getProperty("user.dir");
            File img_file = new File(MOD2DSD_CalculateValues.input_area_name+".jpg");
            JFileChooser saveFile = new JFileChooser(current);
            File OutFile = saveFile.getSelectedFile();
            FileWriter myWriter = null;
            if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
            {
                OutFile = saveFile.getSelectedFile();
```

```
                if (OutFile.canWrite() || !OutFile.exists())
                {
                    File dir = new File(OutFile.getParent());
                    MOD2DSD_Control.success = img_file.renameTo(new File(dir,img_file.getName()));
                    System.out.println("save success"+MOD2DSD_Control.success);
                    myWriter = new FileWriter(OutFile+".html");
                    myWriter.write(" </table> </td> <td> <img src = '"+
MOD2DSD_CalculateValues.input_area_name +".jpg'></td></tr></table>");
                    myWriter.write("<html> <Body onLoad = \"window.print()\"><table> <tr> <td>" +
                            "<table border = 1> <tr> <th colspan = 4>LOCATION:-
"+MOD2DSD_CalculateValues.input_area_name+"</th> </tr>");
                    myWriter.write(" <tr><th colspan = 4> PROFILE:-"+"
"+MOD2DSD_CalculateValues.input_profile_name+" </th></tr>");
                    DecimalFormat df =new DecimalFormat("0.###");
                    myWriter.write(" <tr><th colspan = 4> ITERATION"+"
"+MOD2DSD_CalculateValues.o_iter+" </th></tr>");

                    myWriter.write("<tr > <th>Distance (km) </th> <th> Observed anamolies (mGal)
</th> <th> Calculated anamolies (mGal) </th> <th> Depth (km) </th><th>
ERROR(mGal) </th></tr>");

                    for ( int K = 1; K <= MOD2DSD_CalculateValues.input_n_obs; K++){

                        myWriter.write("<tr> <td>" + MOD2DSD_CalculateValues.x[K]+"</td>
<td>"+df.format(MOD2DSD_CalculateValues.input_nob_gob[K])+"</td>
<td>"+df.format(MOD2DSD_CalculateValues.o_GC[K])+"</td>
<td>"+df.format(MOD2DSD_CalculateValues.o_dep[K])+"</td><td>"+df.format(MO
D2DSD_CalculateValues.err[K])+"</td></tr>");
                    }
                    myWriter.close();
                }
            }
            else
            {
                //pops up error message
            }

        }
        catch(Exception e1) {

            e1.printStackTrace();
        }
    }
}
```

```
package com.mod2dsd.util;


import com.mod2dsd.model.MOD2DSD_CalculateValues;

public class MOD2DSD_Utility {

    public static double convertDouble(String str) throws Exception {

        Double temp = new Double(str.trim());
        return temp.doubleValue();
    }

    public static String convertString(String str) throws Exception {

        String temp = new String(str.trim());
        return temp;
    }

    public static int convertInteger(String str) throws Exception {

        Integer temp = new Integer(str.trim());
        return temp.intValue();
    }

    public static int findMaximumNumber( double observe[]) {
```

```
    double max = 0.0d;
    for (int i = 0; i < observe.length; i++) {

        if (Math.abs(observe[i]) > Math.abs(max)) {

            max = observe[i];
        }
    }

    int maxVal = (int) max/3*5;
    return maxVal;
}

public static int findMaximumNumber( double observe) {

    double max = 0.0d;
    int maxVal=0;
    max = observe;

    if (max < 5) {

        maxVal = 5;
    }
    else if (max >= 5 && max <= 10) {

        maxVal = 10;
    }
    else if ( max > 10 && max <= 15) {

        maxVal = 15;
    }
    else if (max > 15 && max <= 20) {

        maxVal = 20;
    }
    else
    {
        maxVal = MOD2DSD_CalculateValues.o_iter;
    }

    return maxVal;
}

public static double findMaximumNumber1( double observe[]) {

    double max = 0.0d;
    for (int i = 1; i < observe.length; i++) {

        if (Math.abs(observe[i]) > Math.abs(max)) {

            max =Math.abs(observe[i]);
        }
    }

    double maxVal =  max;
    return maxVal;
}
public static double[] convertDoubleArray(String str) throws Exception {

    java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
    String temp = "";
    java.util.ArrayList arr = new java.util.ArrayList();

    while(st.hasMoreTokens()) {

        temp = st.nextToken();
        arr.add(temp);
    }
    double d_array[] = new double[arr.size() + 1];

    for(int i = 0; i <= arr.size(); i++) {

        if (i == 0)
            d_array[i] = 0.0;
```

```
            else
                d_array[i] = convertDouble( arr.get(i-1).toString() );
        }
        return d_array;
    }
}
```

CHAPTER
THREE

Automatic Modeling* and Inversion* of Gravity Anomalies due to 2.5-D Sedimentary Basins by Means of Growing Prismatic Bodies and Exponential Density Function

3.1 General

In regional studies and hydrocarbon exploration, residual gravity fields are generally attributed to bodies having a flat top/bottom or undulating over a mean depth (Rao and Murthy, 1978). Many sedimentary basins on the continental platforms often possess limited strike lengths (Peirce and Lipkov, 1988), and therefore, approximations of such sedimentary basins to finite strike geometries are often justified in the quantitative interpretation of gravity anomalies. Since 3D models generally involve more data and computational requirements than 2D, use of two and one-half dimensionality (2.5D) in the interpretation is acceptable. Rasmussen and Pedersen (1979), Cady (1980) derived equations for the vertical gravity field due to a homogeneous body with polygonal cross-section and finite strike length. Dobrin and Savit (1988)

---

* *Published in Computers & Geosciences (2013, 56, 131-141), Elsevier Ltd., USA.*

had separated the gravity expression of a 2.5-D polygonal body into the 2D terms of Talwani et al. (1959) and the exact terms for the contributions of the ends of the prism, which were referred as end corrections. Nagy (1966), Banerjee and Gupta (1977) derived expressions due to a homogeneous 3-D parallelepiped. Takin and Talwani (1966) developed a technique using uniform density to compute the gravity attraction of 3D sources, where the topographic contours of the body were represented by several polygons and the vertical component of the gravitational attraction was calculated by evaluating the cone formula for a number of vertical sections of the topography. The practical applicability of these forward modeling schemes is rather limited in automatic gravity modeling studies of sedimentary basins for the reasons elucidated in section 2.1 of Chapter-II.

On the other hand, Mickus and Peeples (1992) developed a technique to interpret the gravity and magnetic data for the lower surface of a 2.5-D sedimentary basin using the inverse theory of Backus and Gilbert (1967, 1968, 1970). Again this technique presumes uniform density for the sedimentary rocks, which is not valid. Furthermore, sedimentary basins on the continental regions often possess finite (Peirce and Lipkov, 1988) but variable strike lengths, such as the one shown in Figure 3.1a. To analyze the gravity anomalies of such a basin along a profile, AA', across its strike, the basin needs to be approximated by an ensemble of variable but finite strike multiple models rather than single 2.5-D structure.

Therefore, a need exists to develop a new interpretation strategy using EDF to analyze the gravity anomalies of sedimentary basins treating the source as an ensemble of variable but finite strike multiple models.

This chapter deals with two interpretation strategies namely automatic modeling and inversion and related GUI based software, MODTOHAFSD, to analyze the gravity anomalies of variable but finite strike (2.5-D) sedimentary basins. Unlike the case of many existing methods, the Bouguer gravity anomalies can be supplied to the software as a part of input. Based on the known geologic and other geophysical information, the interpreter can construct/edit regional gravity field in an interactive approach. The estimated residual gravity anomalies are then automatically analyzed for the basement structures either by using the principles of automatic modeling or inversion (user specific).

Applicability of these techniques is illustrated with a synthetic model using both EDF and UDF and also is exemplified with the interpretation of gravity anomalies of the Chintalpudi sub-basin, India.

3.2 Automatic modeling

Modeling of gravity anomalies of strike limited sedimentary basins is equivalent to a mathematical process of estimating the depths of a sedimentary basin from the observed gravity anomalies by means of adjusting the thickness parameters of strike limited prisms within the permissible limits based on the differences between the observed and modeled gravity anomalies (Chakravarthi et al., 2013c).

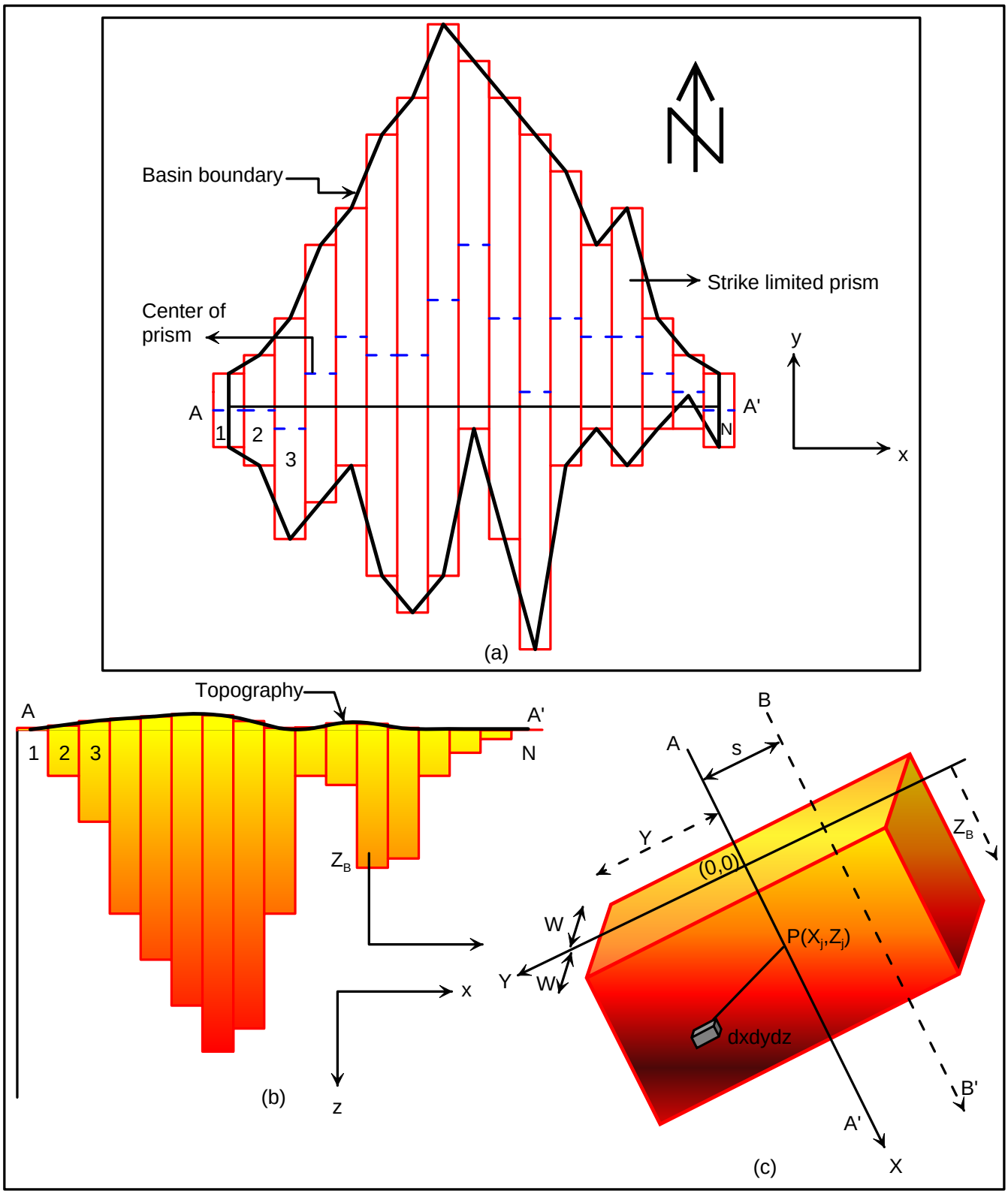

Figure 3.1 (a) Plan view of a synthetic model of a strike limited sedimentary basin and its approximation by an ensemble of variable but finite strike vertical prisms. Note that the limited strike length prevents the structure to represent as a 2-D source. (b) Depth structure of the basin. (c) Geometry of a 2.5-D prism. Note that s is the offset of the profile, BB', from the origin, (0,0).

3.2.1 Forward modeling - theoretical considerations

In Cartesian co-ordinate system, let the z-axis is positive vertically downwards and the x-axis runs transverse to the strike of a sedimentary basin (Figure 3.1b). The plan view and depth structure of a sedimentary basin are shown in Figure 3.1a and Figure 3.1b, respectively. In this case, a stack of N vertical prisms (equal to the number of observations) all having equal widths, but each one possesses its own limited strike length (Figure 3.1a) and thickness (Figure 3.1b) describes the dimensions of the sedimentary basin. Let $2Y$, $2W$ and z_B represent the strike length, width and thickness of one such a prism (Figure 3.1c). The gravity anomaly at any point, $P(x_j, z_j)$, on the profile, AA', which runs along the x-axis and bisects the strike length of the prism can be expressed as (Chakravarthi et al., 2013c)

$$\Delta g_{prism}(x_j, z_j) = 2G \int_v \Delta\rho(z) \frac{(z - z_j)dxdydz}{\left[\overline{x - x_j}^2 + \overline{z - z_j}^2 + y^2\right]^{3/2}}, \tag{3.1}$$

where, G is universal gravitational constant, (x, y, z) are source co-ordinates of an element volume within the prism, and $\Delta\rho(z)$ is the density contrast of sediments represented equation (1.20). Performing analytical integration with respect to y and x, equation (3.1) takes the form (Chakravarthi et al., 2013c)

$$\Delta g_{prism}(x_j, z_j) = 2G\Delta\rho_0 \int_0^{z_B} e^{-\lambda z} \left[\begin{array}{c} tan^{-1} \dfrac{Y\,\overline{x_j + W} + T}{\overline{z - z_j}\sqrt{\overline{x_j + W}^2 + Y^2 + \overline{z - z_j}^2}} \\ -tan^{-1} \dfrac{Y\,\overline{x_j - W} + T}{\overline{z - z_j}\sqrt{\overline{x_j - W}^2 + Y^2 + \overline{z - z_j}^2}} \end{array} \right] dz. \tag{3.2}$$

Equation (3.2) needs to be solved numerically because no closed form solution exists in the space domain. Further, it is to be realized that equation (3.2) is strictly valid for the profile which runs across and bisects the strike length of the prism (such as AA' in Figure 3.1c). In case, the profile fails to bisect the strike length but runs at an offset of s across the strike (such as profile, BB', in Figure 3.1c), then the anomalous field at any point on the profile outside the source region can be calculated as the average of equation (3.2) by substituting, $Y - s$ and $Y + s$ for Y (Chakravarthi et al., 2013c). The effect of offset parameter, s, on the magnitude of the gravity anomaly of a strike limited vertical prism is shown in Figure 3.2a.

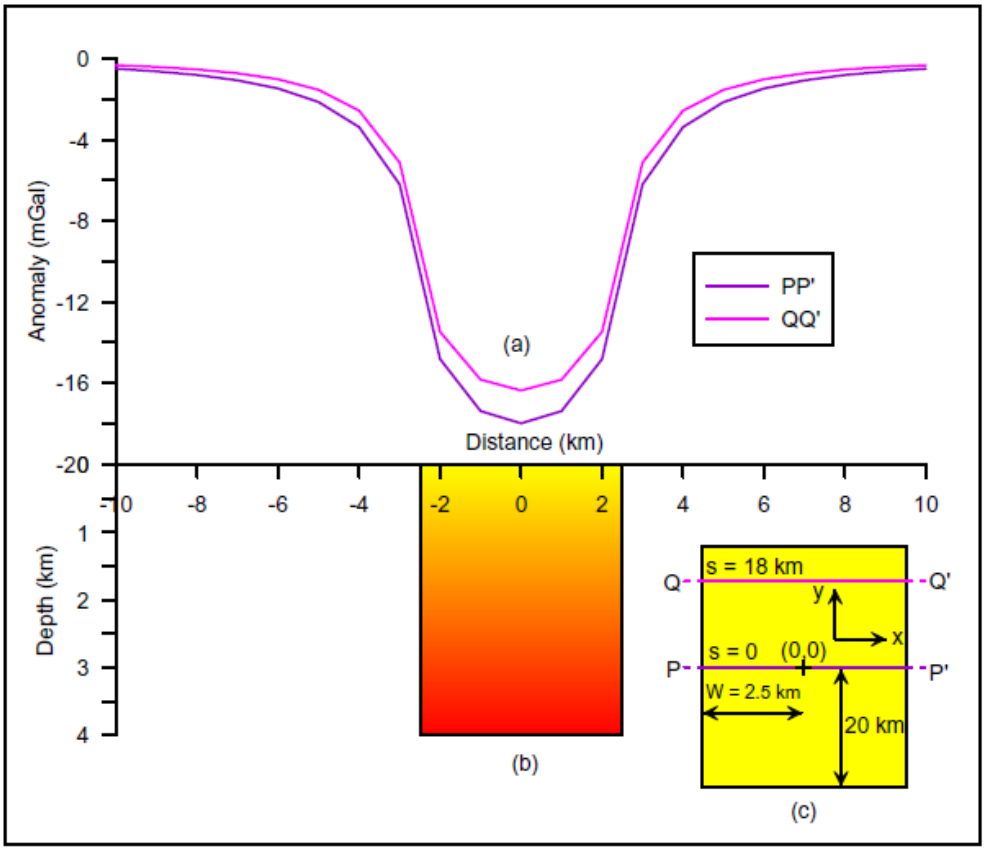

Figure 3.2 (a) Gravity anomalies at zero and 18 km offset. (b) Geometry of a 2.5-D vertical prism in xz-plane. (c) Plan view of the 2.5-D prism in xy-plane with the locations of two selected profiles, PP' and QQ'.

The gravity anomalies produced by a 2.5-D prism in the interval x_j Є [-10 km, 10 km] along two selected profiles, PP' and QQ' (locations are shown in Figure 3.2c), where one profile runs at zero offset and the other at 18 km offset (Figure 3.2b) are shown in Figure 3.2a. The assumed parameters of the prism are: z_B = 4 km, W = 2.5 km and Y = 20 km (Figures 3.2b and 3.2c) with $\Delta\rho_0$ = -0.35 g/cm^3 and λ = 0.5 km^{-1}. The colour variation from yellow to red within the prism (Figure 3.2b) represents the decrease in density contrast (absolute magnitude) with depth. One can notice from Figure 3.2a that the gravity response of the prism at 18 km offset portrays larger magnitude than the corresponding one observed at the zero offset.

In the presence of regional background, the gravity effect of a sedimentary basin, $g_{joint}(x_j, z_j)$, can be expressed as (Chakravarthi et al., 2013c),

$$g_{joint}(x_j, z_j) = g_{basin}(x_j, z_j) + \psi(x_j), \tag{3.3}$$

where,

$$g_{basin}(x_j, z_j) = \sum_{j=1}^{N} \Delta g_{prism}(x_j, z_j), \tag{3.4}$$

and

$$\psi(x_j) = \sum_{m=0}^{N1} f_m x_j^m. \tag{3.5}$$

Here, $N1$ is the degree of polynomial and b_m represent a set of coefficients. The gravity anomaly of a prism with UDF can also be realized by letting λ = 0 in equation (3.2).

3.2.2 Gravity signatures of a 2.5-D sedimentary basin at different offsets

Figure 3.3a shows the plan view of a north south striking sedimentary basin, whose depth structure is shown in Figure 3.3c. The nature of gravity anomalies produced by the sedimentary basin at three different offsets along three selected profiles CC', DD' and EE' (locations are shown in Figure 3.3a) are shown in Figure 3.3b. The profile CC', which runs from west to east covers the basin fill completely, whereas the profiles DD' and EE' fail to do so (Figure 3.3a). The basin is having variable but finite strike length; hence, a stack of 16 vertical prisms with different strike lengths (Figure 3.3a) and thicknesses (Figure 3.3c) is used to describe the dimension of the basin. Furthermore, all three selected profiles (Figure 3.3a) do not bisect the strike lengths of the prisms (except the profile CC', which bisects the 1st and 2nd prisms) but run at different offsets. Hence, in each case the offset of the profile from the centre of each prism is measured and used in equations (3.2) and (3.4) to realize forward modeling. The density contrast of sediments within the basin is assumed to vary according to equation (1.20), where $\Delta\rho_0$ = -0.5 g/cm^3 and λ = 0.55 km^{-1} (Figure 3.3d).

The gravity anomalies calculated at 16 equispaced observations in the interval $x_j \in$ [0 km, 15 km] along each selected profile are shown in Figure 3.3b. A gravity low of about -26 mGal is observed along the profile, CC', at the basin's depocentre, where the thickness of sediments attains its maximum (4.3km) and a relative high of -22 mGal over the basement ridge towards the east. In the case of profile, DD', the magnitude of the

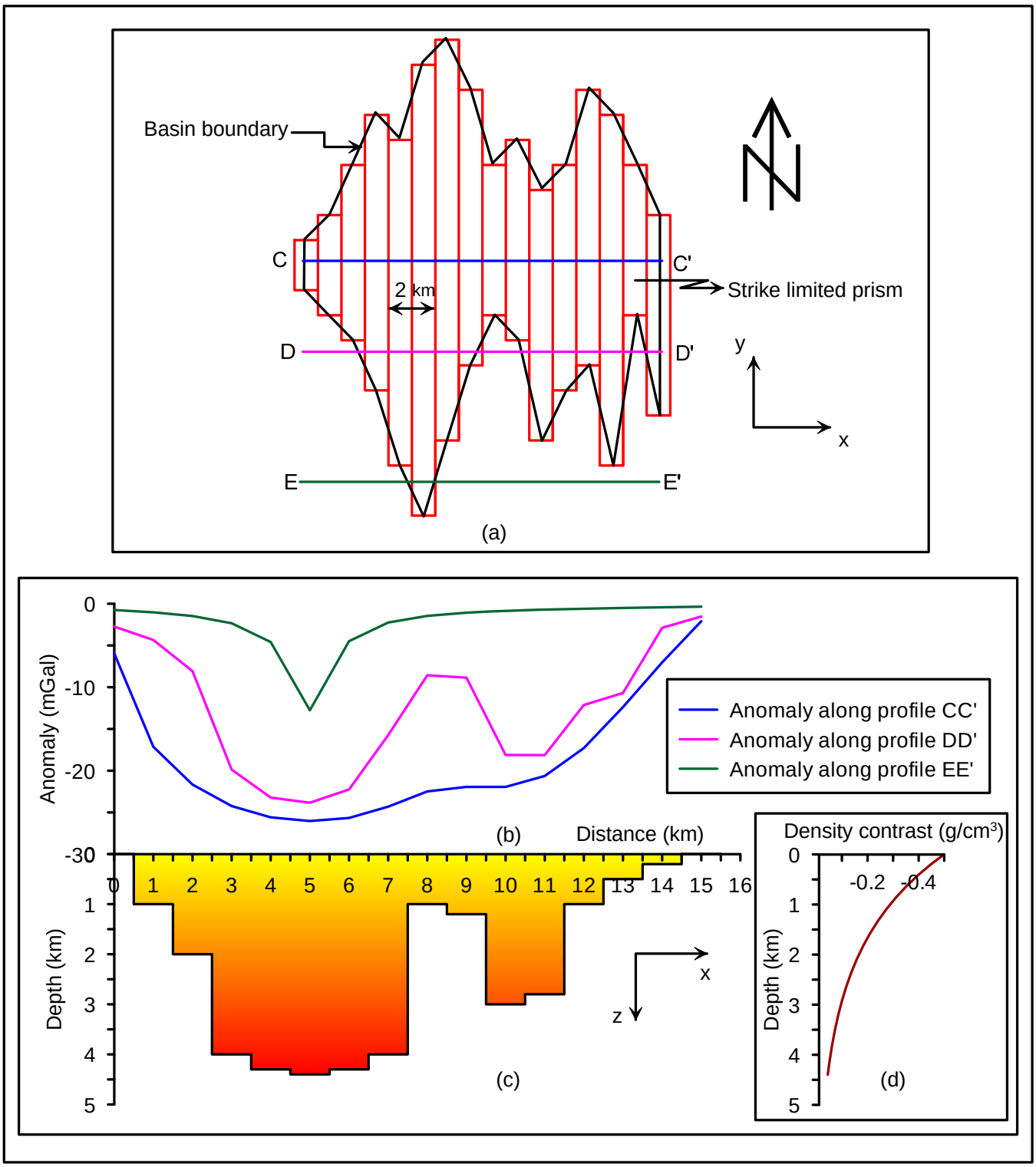

Figure 3.3 (a) Plan view of a north south striking 2.5-D sedimentary basin with the locations of three selected profiles (CC', DD', EE') at different offsets. (b) Gravity anomalies along the three profiles CC', DD', EE'. (c) Depth structure of the basin. The description for tonal variation from yellow to red within structure is given in Figure 2.5b. (d) Prescribed EDF.

anomaly increases to -23 mGal over the basin's depocentre and towards the east a relative gravity high of about -8.5 to -12 mGal is observed over the basement ridge. On the other hand, the gravity anomaly along the profile, EE', shows a low of -12.8 mGal at the 5th km on the profile bordered on either side by marginal highs.

Therefore, the magnitude of gravity anomaly across a geological source varies as a function of offset and hence, it is essential to consider this parameter in the analysis of gravity anomalies for reliable interpretations (Chakravarthi et al., 2013c).

3.2.3 Automatic modeling of gravity anomalies

It is well-known that known geology plays a crucial role in deciphering the best regional trend on a Bouguer gravity anomaly map. However, the process of regional and residual anomaly separation is highly subjective in the absence of a *priori* information about the geology. Although, Chakravarthi and Sundararajan (2004, 2007) used linear and quadratic polynomials to describe regional gravity field in analyzing gravity anomalies, elsewhere, regional anomalies do not lend themselves to be exactly simulated with linear/quadratic polynomials. The proposed algorithm allows one to choose any degree of polynomial to describe complex regional trends (Chakravarthi et al., 2013c).

The interpretation starts by constructing regional gravity background, which needs to be separated from the observed Bouguer gravity anomalies prior to the analysis. Based on the known geologic and

other geophysical information, the interpreter selects a few control points, by means of mouse clicks, in the anomaly panel (Figure 3.4) to decipher appropriate regional trend. The algorithm automatically reads the coordinates of the selected points, fits a polynomial of a predefined degree and estimates the coefficients of the polynomial. These coefficients and observer locations along the profile are then used by the algorithm to construct and display an analytically defined regional. The algorithm then estimates residual gravity anomalies by subtracting the analytically defined regional from the Bouguer gravity anomalies and displays the residual anomalies in the anomaly panel. The advantage of the algorithm is that it provides a means to modify/edit the regional trend. In this case the interpreter needs to select the control points again (by means of mouse clicks) and edit them using simple drag and drop mouse operations. The coefficients of the polynomial are then automatically updated and a modified regional will be constructed and displayed in the anomaly panel along with the modified residual. The estimated residual anomalies are then consequentially modeled for basement configuration (Chakravarthi et al., 2013c).

The initial depth estimates, $Z_B(x_j, z_j)$, of a basin are calculated at each observation on the profile using equation (2.4) in case of EDF and equation (2.5) in case of UDF. These initial depth estimates are used in equation (3.4) to calculate the model gravity response of the basin at all observations on the profile. The depth estimates of the basin are automatically updated based on equation (2.6) and the process of

modeling repeats until one of the conditions of its termination is fulfilled as described under section 2.2.3 of Chapter-II.

3.3 Inversion

Most of the existing 2.5-D inversion strategies (e.g., Mickus and Peeples, 1992) yield unreliable interpretations not only when the density contrast portrays variation with depth but also when the profile (such as the profile, DD', in Figure 3.3) along which the inversion is intended fails to bisect the strike length of the structure. Therefore, it is necessary to develop a new inversion technique using EDF to overcome the drawbacks associated with existing algorithms to analyze the gravity anomalies of 2.5-D sedimentary basins as presented in the following section.

In this case, the N unknown depth parameters to be determined from N observations form a system of $\hat{P}_{\hat{m}}$, where $\hat{P}_{\hat{m}} = Z_{\hat{m}}$, for $\hat{m} = 1, 2, \ldots, N$ are depth estimates of the prisms.. The error between observed and modeled gravity anomalies at all locations are used to improve N depth estimates of the basin (Chakravarthi et al., 2013c). The following system of normal equations is to be solved for the increments/decrements of the unknown parameters, N, namely $dP_{\hat{k}}$, $\hat{k} = 1, 2, \ldots, N$ based on the principles of inversion as described in Chapter-I as

$$\sum_{k=1}^{N}\sum_{\hat{k}=1}^{N}\frac{\partial \Delta g_{prism}(x_k, z_k)}{\partial \hat{P}_{j'}}\frac{\partial \Delta g_{prism}(x_k, z_k)}{\partial \hat{P}_{\hat{k}}}\left(1 + \vartheta\delta_{j'\hat{k}}\right) d\hat{P}_{\hat{k}}$$

$$= \sum_{k=1}^{N}[g_r(x_k, z_k) - g_{basin}(x_k, z_k)]\frac{\partial \Delta g_{prism}(x_k, z_k)}{\partial \hat{P}_{j'}}, j' = 1, 2, .., N. \qquad (3.6)$$

Partial derivatives required in the above system of equation (3.6) are calculated by a numerical approach, which involves the calculation of the rate of change of gravity anomaly with respect to each unknown parameter to be solved (Chakravarthi et al., 2013c). The estimated increments/decrements, $dP_{\hat{k}}$, $\hat{k} = 1, 2, \ldots, N$ are used to update the existing depth parameters, $\hat{P}_{\hat{m}}$, and the inversion process continues until one of its termination conditions is fulfilled as described in Chapter-I.

3.4 Description of software - MODTOHAFSD

Based on the algorithms of automatic modeling and inversion as described in sections 3.2.3 and 3.3, a GUI based software, MODTOHAFSD, coded in JAVA is developed in the space domain to analyze the gravity anomalies of 2.5-D strike sedimentary basins using EDF (Appendix 3.1). This software, being platform independent, works on MVC pattern (Figure 2.3). The user has the option to choose either automatic modeling or inversion to analyze the gravity anomalies for basement structures. The complete details of the software including its installation and operation are available at https://github.com/chakravarthi/MODTOHAF/tree/master/program/modtohafsd (Chakravarthi et al., 2013c).

Upon execution, the view module of the software appears on the monitor as shown in Figure 3.4. The input layout consists of ten fields and four menus. The user needs to specify the information pertaining to Area/Profile, number of observations, station elevations (km), station interval (km), observed Bouguer gravity anomalies (mGal), minimum and

maximum anomalies (mGal), maximum permissible depth (km), allowable error, and the number of iterations to be performed in the input layout, and the parameters of model space i.e., half strike length of each prism (km), offset of the profile from the center of each prism (km), constants of EDF namely, $\Delta\rho_0$ (g/cm^3) and λ(km^{-1}), in appropriate fields under the menu 'Model information'.

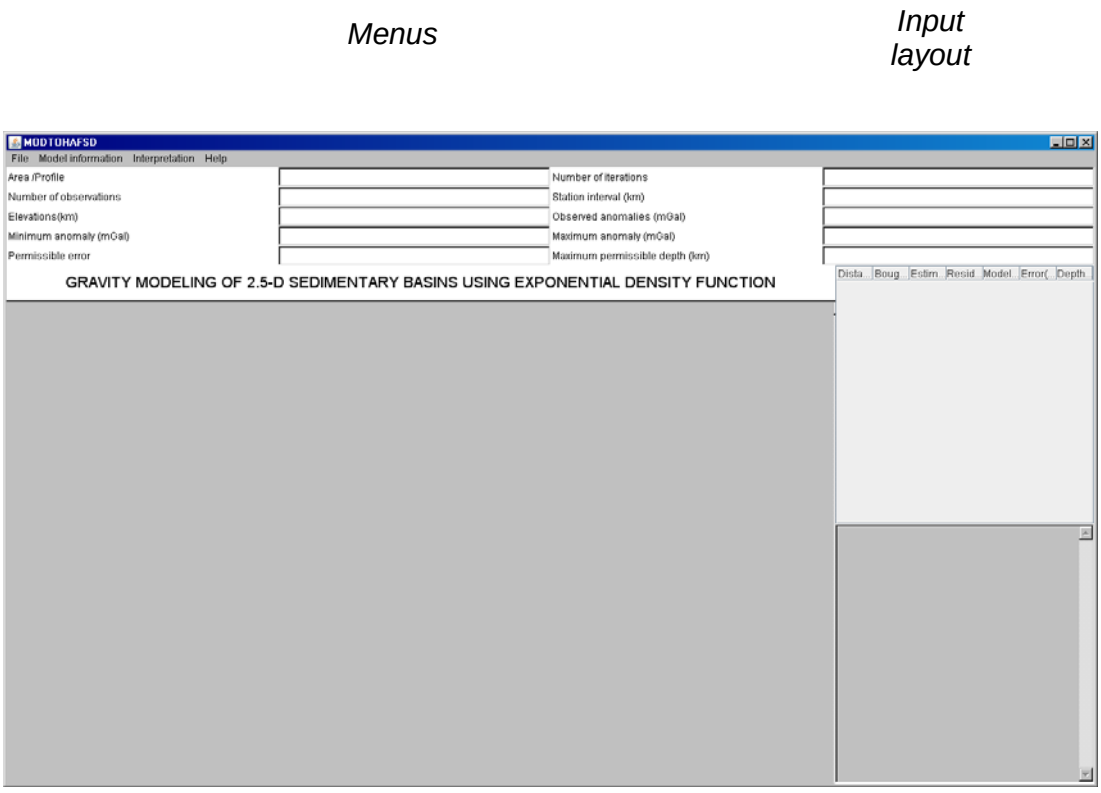

Figure 3.4 View module of MODTOHAFSD

The software also provides an option to the user to choose either a constant regional or a variable regional by making use of the 'Interpretation' menu. Upon selection of variable regional the software prompts the user to specify the degree of polynomial. Regional field can be constructed/edited following the procedure described in the text under

the section 3.2.3. Finally, the user opts either 'Automatic Modeling' or 'Inversion' under the 'Analysis' submenu of 'Interpretation' to analyze the gravity anomalies for the basement configuration. The parameters that are specified in the input layout remain unchanged during the process of analysis, while the thicknesses of the prisms at plurality of locations are automatically initiated and updated in an iterative approach within the specified convergence criteria. The software shows the animated versions in model growth, nature of fit between the observed and modeled gravity anomalies and corresponding changes in misfit with iteration and variation of density contrast with depth.

3.5 Applications

Reliability of automatic modeling and inversion schemes discussed in the text is demonstrated by interpreting the gravity anomalies due to a synthetic model of a 2.5-D sedimentary basin using both EDF and UDF. Further, the interpretation real field gravity anomalies over the Chintalpudi sub-basin in India, shows the practical utility of the algorithms.

3.5.1 Automatic modeling

(a) Synthetic example – interpretation of noisy gravity anomalies

Figure 3.5a shows a set of noisy Bouguer gravity anomalies in the interval, $x_j \in$ [0 km, 32 km] produced by a synthetic model of a sedimentary basin, whose plan view is shown in Figure 3.3a and its depth structure in Figure 3.5b respectively. The model characterizes the features

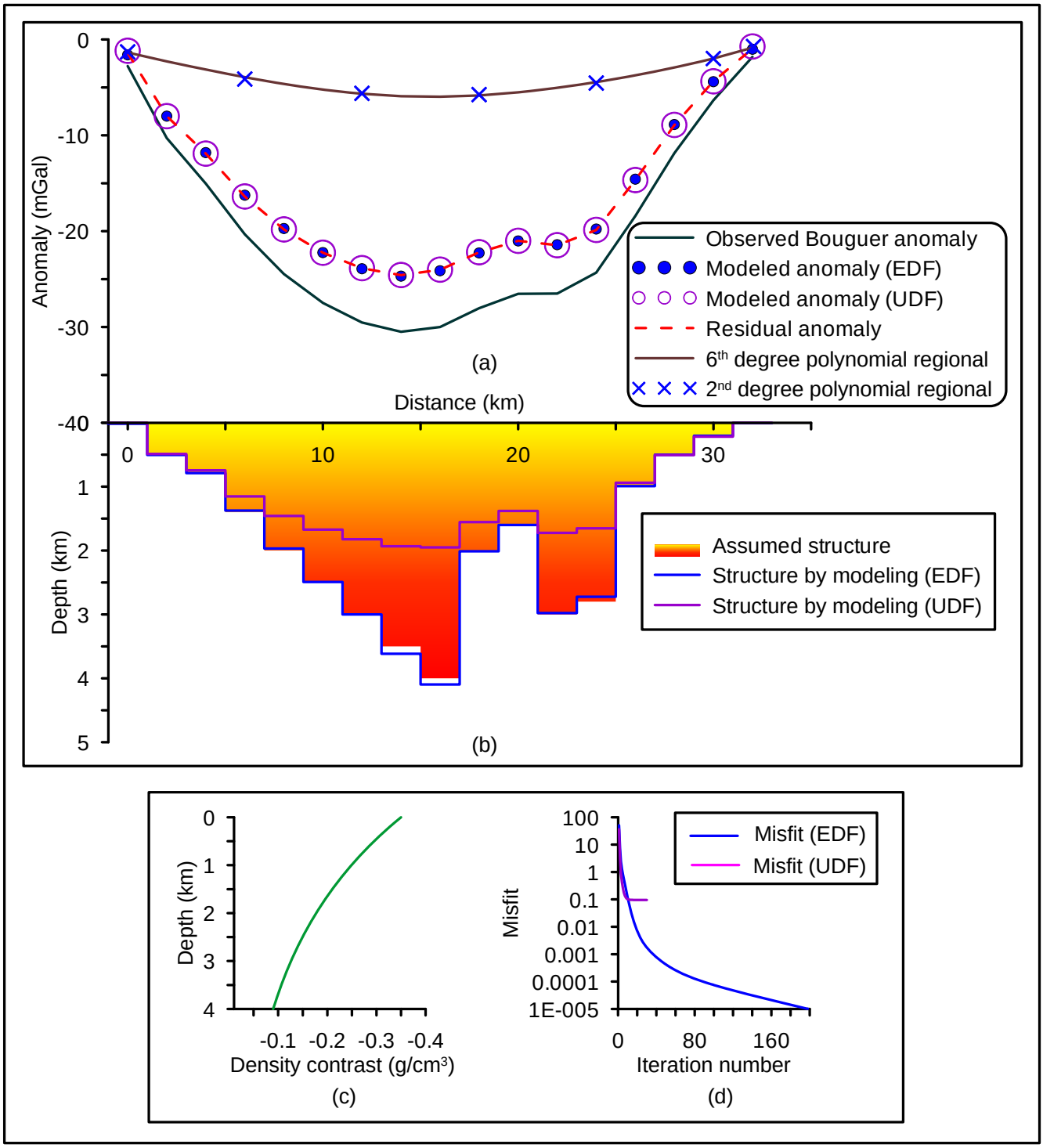

Figure 3.5 (a) Noisy Bouguer gravity anomaly, assumed and estimated regional gravity backgrounds by 6th and 2nd degree polynomials, estimated and modeled residual gravity anomalies using EDF and UDF. (b) Assumed and estimated depth structures by automatic modeling. The description for tonal variation from yellow to red within structure is given in Figure 2.5b. (c) Prescribed EDF. (d) Variation of misfit with iteration number.

of a typical graben structure. Further, a basement high bounded by steep normal faults divides the basin into two parts. The density contrast of sediments within the basin varies according to equation (1.20) defined with $\Delta\rho_0$= -0.35 g/cm^3 and λ = 0.33 km^{-1} (Figure 3.5c). In this case, the pseudorandom noise was Gaussian, with zero mean and a standard deviation of 0.21 mGal. The regional gravity background present in the Bouguer gravity anomaly varies according to a 6th degree polynomial, $\sum_{m=0}^{6} f_m x^m$. The coefficients of the polynomial are given in Table 3.1.

Table 3.1

Assumed coefficients of the 6th degree polynomial regional background

Coefficient	Assumed value
f_0	-1.364880845
f_1	-0.4765145366
f_2	0.01075816477
f_3	-0.001127570838
f_4	0.0001382314667
f_5	-4.892878729E-006
f_6	5.673502197E-008

This interference term exhibits large wavelength compared to the one produced by the residual source, namely the sedimentary basin. Further, it can be noticed from Figure 3.5a that the magnitude of regional background varies considerably over the length of the profile: from about -1.5 mGal over the left part to as much as -6 mGal over the central part and then reaches to about -1 mGal towards the right.

However; in reality, the degree of polynomial to be chosen to simulate regional background is always uncertain in the absence of

known geology (Chakravarthi and Sundararajan, 2004, Chakravarthi et al., 2013c). For this reason, although a 6th degree polynomial is used to describe the regional background in generating the noisy Bouguer gravity anomaly of the structure (Figure 3.5a); a 2nd degree polynomial is chosen in the analysis to study its effect in the interpretation, if any.

Accordingly, a total of seven anomalies are selected (shown as '**x**' in Figure 3.5a) keeping in view the general characteristics of regional anomaly and a 2nd degree polynomial was fitted to construct an analytically defined regional trend (Figure 3.5a). The coefficients of 2nd degree polynomial regional background estimated by the algorithm are given in Table 3.2.

Table 3.2

Estimated coefficients of the 2nd degree polynomial regional background

Coefficient	**Estimated value**
f_0	-1.154740095
f_1	-0.5997758819
f_2	0.01910476074

The estimated residual gravity anomalies were analyzed for the basement structure by the automatic modeling module of MODTOHAFSD.

The parameters $\Delta\rho_0$ and λ of EDF to analyze the gravity anomalies are obtained from the known density-depth information (Figure 3.5c). Further, the half strike lengths of prisms and offset distances of the profile from each bisector of the prisms are assumed. In this case, the code had performed 198 before it got terminated as the misfit (equation 1.21) fell

below an allowable error of 1E-05 mGal. The misfit between observed and modeled gravity anomalies had reduced drastically from 50.74 for the starting model to about 0.001 at the end of the 37th iteration (Figure 3.5d), and then slowly reached to 1E-05 at the end of 198th iteration. The depth parameters of the structure remain more or less unchanged beyond the concluding iteration and the fit between the observed and modeled gravity anomalies is satisfactory (Figure 3.5a). The estimated structure of the basin compares reasonably well with the assumed one (Figure 3.5b). The inferred maximum thickness of the basin (4.096 km) from modeling at the 16th km on the profile was marginally over estimated by 2.4% (the assumed maximum thickness of the basin is 4km). Such a discrepancy between the assumed and estimated structure is acceptable considering the fact that significant level of noise present in the Bouguer gravity anomaly.

The anomaly shown in Figure 3.5a is also interpreted using UDF by letting λ = 0, for which the modeling took 30 iterations before it got terminated as the new misfit exceeds its preceding value. In this case, the misfit had reduced from its initial value of 36.8 to 0.09 at the end of the 30th iteration (Figure 3.5d). The modeled gravity anomaly and the corresponding depth structure of the basin are also shown in Figures 3.5a and 3.5b. The maximum thickness of the basin estimated from modeling using UDF is only 1.951 km against the assumed depth of 4 km.

In short, it can be seen from Figure 3.5a that the modeled gravity anomaly realized by both EDF and UDF equally explain the observed

anomaly, however, the depth structure of the basin obtained with UDF is very much underestimated.

(b) Field example – Chintalpudi subbasin, India

The Chintalpudi subbasin, one of the major coal producing Gondwana basins of India, represents the southeasterly continuation of the Kothagudem subbasin of the Pranhita-Godavari valley and covers an area of about 2500 sq. km. The basin on either side is bounded by well-defined faults, which gives rise to a graben structure within the basin. In the subbasin, Archaean gneisses form the basement for the Gondwana sequence, and the basin is of a younger generation, as evidenced by the absence of Barren Measure Formations over a major part of the area. The basin is about 65 km long and 38 km wide and the topography of the entire area is flat or gently undulating (Rao, 1982).

The details of the gravity survey including the distribution of observations, application of various corrections to the measured data and the accuracy of Bouguer anomalies are discussed by Mishra et al. (1987). Further, Kaila et al. (1990) have carried out Deep Seismic Sounding (DSS) investigations along a profile across the basin and estimated the maximum thickness of sediments as 2.8 km. The Oil and Natural Gas Corporation Ltd (ONGC), India has drilled a deep borehole in the basin at its depocentre and encountered the Archaean basement at a depth of 2.935 km (Agarwal, 1995). The constants of EDF obtained by fitting equation (1.20) to the measured density contrast-depth data from this borehole (Figure 3.6c) are given as $\Delta\rho_0$ = -0.4692 g/cm^3 and λ = 0.4078

km^{-1}, respectively (Rao and Rao, 1999). Although the basin possesses limited and variable strike length (Rao, 1982), Rao and Rao (1999) and Zhou (2013) modeled the gravity anomalies of the basin treating the basin as a 2D structure. Rao and Rao (1999) analyzed the anomalies of the basin along a profile using a combination of both frequency and space domain methods, whereas Zhou (2013) makes use of the line integral (LI) and maximum difference reduction (MDR) methods.

For the present study, the same anomaly (Figure 3.6a) is interpreted by the proposed algorithm using the principles of automatic modeling. For such a study, the structure is described with a stack of 16 vertical prisms (equal to the number of observations), each one with its own limited strike length. Further, the offset of the profile from the centre of each prism is measured and supplied to the code as a part of input. The interpretation converges in 33 iterations. The change in misfit against the iteration number is shown in Figure 3.6d. It is to be noted that the magnitude of residuals observed in the proposed modeling are below ± 0.23 mGals, whereas the residuals reported in the methods of Rao and Rao (1999) and Zhou (2013) are ± 0.25 mGals, 0.5 mGal respectively (Chakravarthi et al., 2013c). The modeled anomaly is shown in Figure 3.6a and corresponding inverted bedrock depths in Figure 3.6b respectively. By and large, the modeled anomalies mimic the observed anomalies (Figure 3.6a). The thickness of the basin estimated from present modeling (2.913 km) at the existing borehole compares excellently well with the drilling depth of 2.935 km. The interpreted depth structure of the basin by Rao and Rao (1999) and Zhou (2013) are also shown in Figure 3.6b for comparison. In this

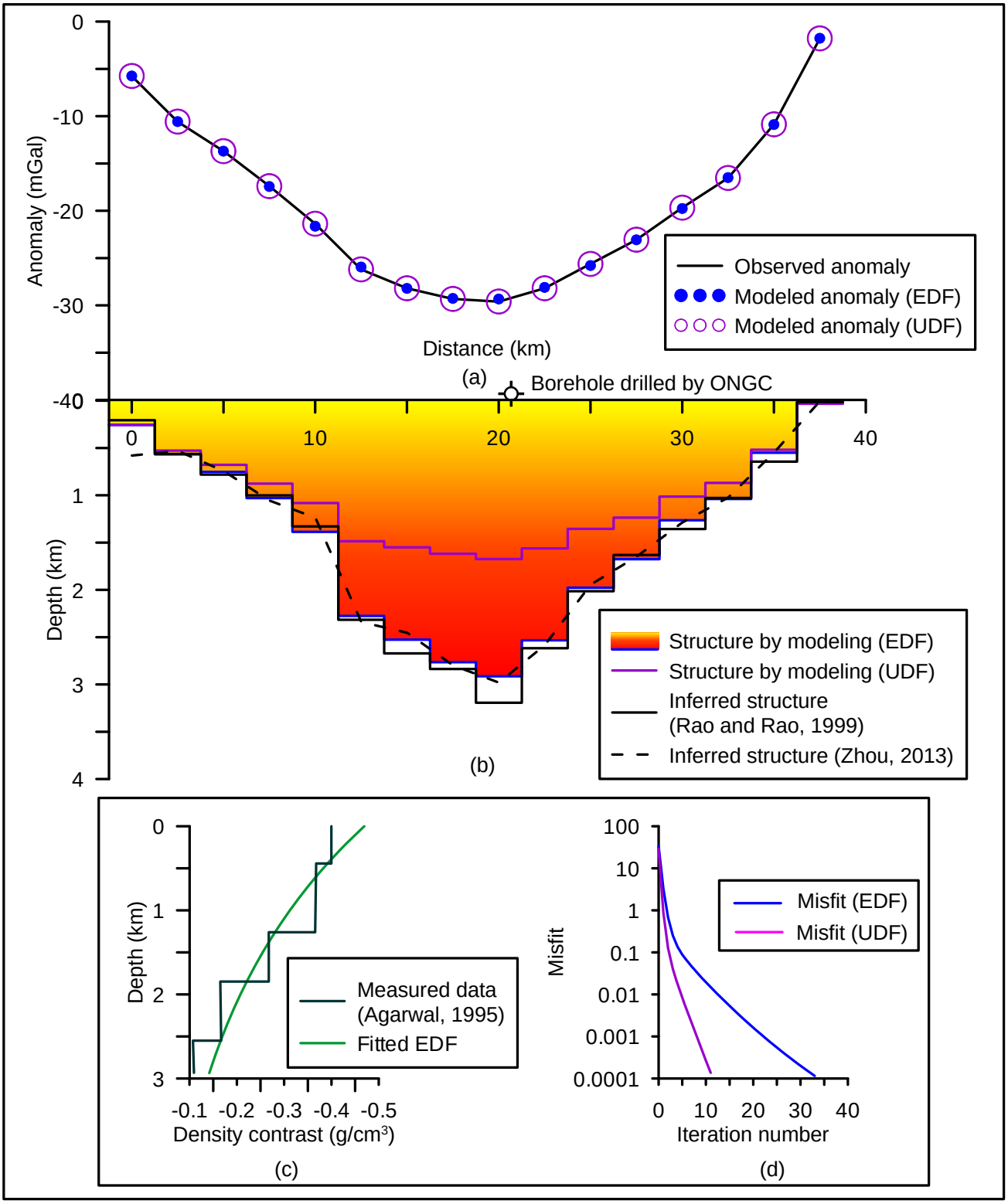

Figure 3.6 (a) Observed and modeled gravity anomalies using EDF and UDF, Chintalpudi sub-basin, India. (b) Estimated depth structures by modeling. The description for tonal variation from yellow to red within the structure is given in Figure 2.5b. Interpreted depth structure by Rao and Rao (1999) and Zhou (2013) are also shown. (c) Measured density contrast-depth data (Agarwal, 1995) and fitted EDF (Rao and Rao, 1999). (d) Variation of misfit with iteration number.

case, the interpreted structure of the basin from automatic modeling closely mimics the structure inferred by Zhou (2013) rather than the one inferred by Rao and Rao (1999).

The gravity anomaly in Figure 3.6a is also interpreted using UDF by letting λ = 0. In this case, the algorithm took 11 iterations before it got terminated as the resulting misfit attained a value more than its preceding value. The modeled gravity anomaly with UDF is shown in Figure 3.6a and the corresponding depth structure in Figure 3.6b. The changes in misfit are shown in Figure 3.6d. In this case, the estimated thickness of the basin at the borehole location is too small (1.67 km) to account for the total thickness of sediments revealed from drilling and hence the UDF is inappropriate in such modeling.

3.5.2 Inversion

(a) Synthetic example – interpretation of noisy gravity anomalies

The applicability of the inversion is illustrated with the same gravity anomaly, which was interpreted in the previous section 3.5.1a by automatic modeling. For such an inversion, the algorithm had performed 1589 iterations before it got terminated as the misfit (equation 1.21) reached to a predefined allowable error of 1E-05 from its initial value of 50.74. The changes in misfit with iteration are shown graphically in Figure 3.7c. In this case, the misfit drastically reduced from 50.74 for the initial model to 0.001 at the end of the 37th iteration (Figure 3.7c), beyond which the convergence rate is relatively slow in comparison with the automatic modeling (Figures 3.5d and 3.7c).

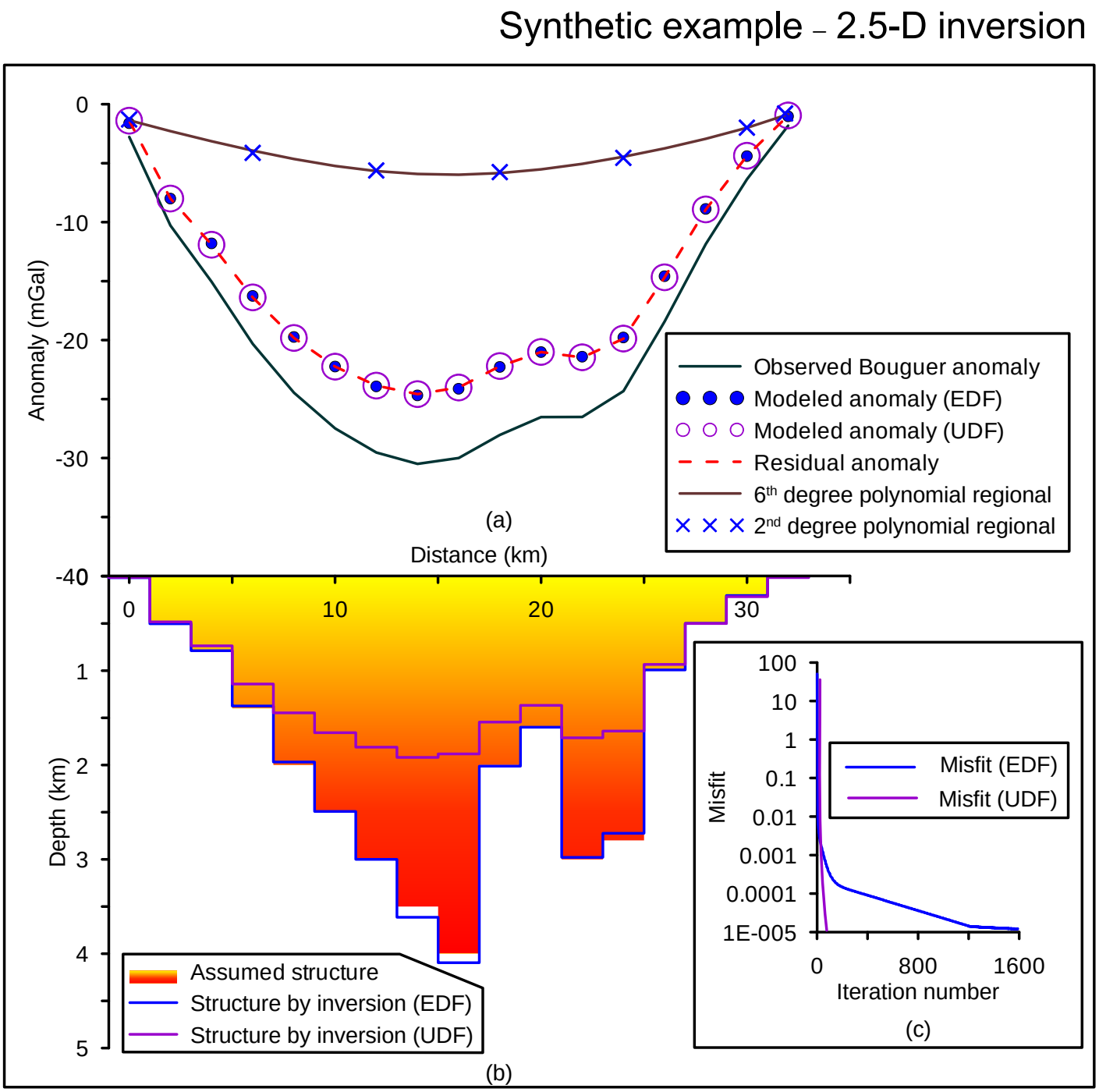

Figure 3.7 (a) Noisy residual and modeled gravity anomalies by inversion in case of EDF and UDF. (b) Assumed and estimated depth structures by inversion. The tonal variation from yellow to red within structure indicates the decrease in density contrast as in the case of Figure 2.5b. (c) Variation of misfit with iteration number.

The modeled gravity anomaly subsequent to inversion and the estimated structure of the basin corresponding to the obtained minimum misfit are shown in Figures 3.7a and 3.7b along with the observed gravity anomaly and the assumed structure. The inferred depth estimates of the structure and the nature of fit between the observed and modeled gravity anomalies do not show appreciable changes subsequent to the concluding iteration. It is to be noticed from Figure 3.7b that the estimated structure closely mimics the assumed structure even in the presence of significant level of pseudorandom noise in the anomaly. The maximum thickness of the basin estimated from present inversion (4.096 km) precisely coincides with the maximum thickness inferred from modeling and also compares sensibly well with the assumed depth of 4.0 km. In short, the gravity anomaly of the synthetic model analyzed both by automatic modeling and inversion yields exactly the same results.

Furthermore, the gravity anomaly shown in Figure 3.7a is also inverted with UDF (-0.35 g/cm^3) by setting λ to zero. In this case, the algorithm took 19 iterations before it got terminated as the misfit reached almost to zero from its initial value of 34.3. The modeled gravity anomaly and the derived depth structure of the basin are shown in Figures 3.7a and 3.7b for comparison.

It can be observed from Figure 3.7a that the modeled gravity anomaly of both EDF and UDF fit equally the observed anomaly; however, the inverted structure with EDF closely mimics the actual one while with CDF it doesn't. Further, structures inverted both with EDF and UDF

coincide with each other at the extremities of the profile, while at the center of the basin; the structure with UDF is grossly underestimated. Hence, UDF is not appropriate in such application.

(b) Field example – Chintalpudi subbasin, India

The inversion is illustrated with the field gravity anomaly of the Chintalpudi subbasin in India (Figure 3.8a) using both EDF and UDF and the interpretations are shown in Figure 3.8. The same anomaly is interpreted in the previous section by automatic modeling. When the gravity anomaly is subjected to inversion with EDF, no significant changes in the depth estimates of the basin are observed beyond the 82nd iteration. The algorithm got terminated after the 82nd iteration as the misfit function attained an allowable error of 1E-05. The modeled gravity anomaly subsequent to inversion is shown in Figure 3.8a and the inverted depth structure of the basin in Figure 3.8b, respectively. The magnitude of residuals is below ± 0.23 mGals as observed in the case of modeling (Chakravarthi et al., 2013c). The inversion yields the maximum thickness of the basin as 2.965 km exactly corresponding to the 20th km along the profile and compares excellently well with the drilling depth of 2.935 km. Further, the shallow basement of 0.55 km estimated around 36th km from both automatic modeling and inversion agrees well with several bores drilled within the area for groundwater exploitation.

On the other hand, when UDF is used in the inversion, the algorithm performed only 12 iterations as the new misfit corresponding to the structure at the end of the 12th iteration had exceeded its previous

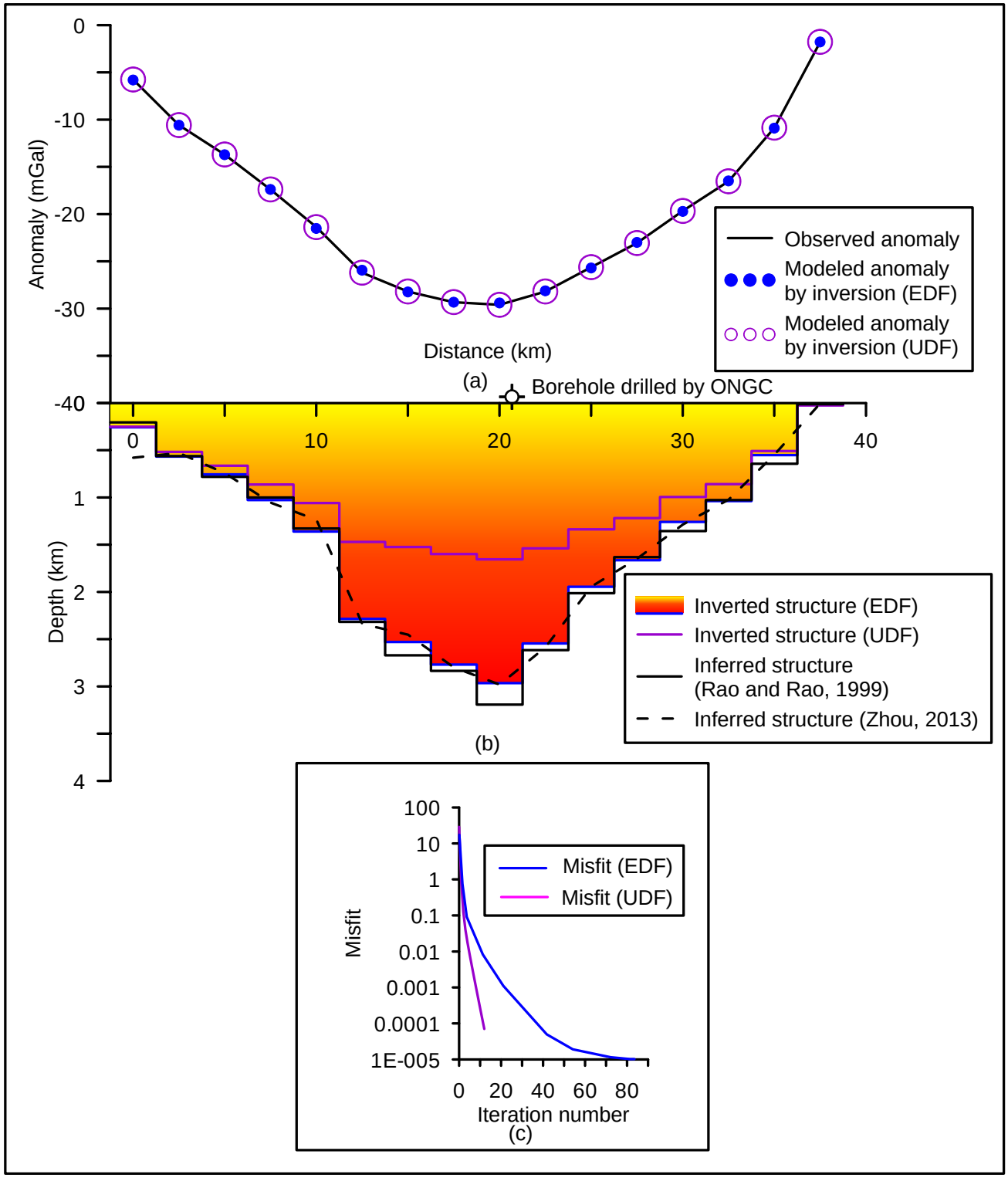

Figure 3.8 (a) Observed (after Rao and Rao, 1999) and modeled gravity anomalies by inversion in case of EDF and UDF, Chintalpudi sub-basin, India. (b) Estimated depth structures by inversion using EDF and UDF . Inferred configurations of the basin by Rao and Rao (1999) and Zhou (2013) are also shown for comparison. The description for tonal variation from yellow to red within the structure is given in Figure 2.5b. (c) Variation of misfit with iteration number.

value. The changes in misfit with iteration number are shown in Figure 3.8c; whereas the modeled gravity anomaly after the inversion in Figure 3.8a and the inferred depth structure in Figure 3.8b, respectively. The inferred thickness of 1.65 km for the basin at the existing borehole location is unacceptable, and hence the UDF is inappropriate.

Thus the gravity anomaly analyzed both by automatic modeling and inversion yield more or less the same results. That is in the case of EDF; modeling results a thickness of the basin as 2.913 km, while inversion 2.965 km against the drilling depth of 2.935 km. On the other hand, with UDF the depth obtained from modeling is 1.67 km and inversion 1.65 km, which is more or less the same.

3.6 Results and Discussion

Automatic modeling and inversion when implemented with EDF on synthetic examples, these schemes successfully recovered the structure with modest deviation even in the presence of pseudorandom noise, whereas with UDF, a distinct deviation in depth structure is noticed.

The field gravity anomaly of Chintalpudi sub-basin, India when subjected to both automatic modeling and inversion with EDF results a similar depth structure in line with available drilling information, however, a disturbed picture of depth structure in case of UDF. Modeling and inversion of both synthetic and real field anomalies reveal that automatic modeling takes less time for analysis than inversion. The salient feature of both automatic modeling and inversion is that they are

fairly and equally applicable even the profile does not bisect the strike length of the sedimentary basins.

"MODTOHAFSD : A GUI based JAVA code for gravity analysis of strike limited sedimentary basins with exponential density function".

The system requirements to run the program are:-

Java Development kit(jdk)1.6 & above
RAM : 512 MB.
Disk space: 70 MB.

```
package com.modtohafsd.view;


import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import com.modtohafsd.control.MODTOHAFSD_Controller;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;

public class MODTOHAFSD_MainView extends Frame{


    private static final long serialVersionUID = 1L;

    public static void main(String s[])
    {
        MODTOHAFSD_MainView cm = new MODTOHAFSD_MainView();
        cm.setSize(1280, 768);
        cm.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                JFrame frame = null;
                int r = JOptionPane.showConfirmDialog(
                        frame,
                        "Exit MODTOHAFSD ?",
                        "Confirm Exit ",
                        JOptionPane.YES_NO_OPTION);
                if(r == JOptionPane.YES_OPTION ){
                    if(MODTOHAFSD_Controller.success == false){
                        String fileName = MODTOHAFSD_CalculateValues.input_area_name+".jpg";
                        File f = new File(fileName);
                        f.delete();
                    }
                    System.exit(0);

                }
            }
        });
        cm.setTitle("MODTOHAFSD");
        cm.setResizable(true);
        cm.add(new MODTOHAFSD_MainPanel(cm));
        cm.setVisible(true);

    }

}
```

```
package com.modtohafsd.view;

import java.awt.*;
import java.io.*;
import java.util.HashMap;
import javax.swing.JFileChooser;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
```

```
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.constant.*;
import com.modtohafsd.view.event.MODTOHAFSD_PlotReg;



public class MODTOHAFSD_MainPanel extends Panel {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    static Panel p_North;
    public static Panel p_East;
    public static Panel p_Center;
    public static TextField inputValues [] = new TextField[15];
    public static TextArea img = new TextArea(46,140);
    public static TextArea im = new TextArea(46,140);
    public static TextArea im1 = new TextArea(46,140);
    public static TextArea im2 = new TextArea(46,140);
    public static Object lamb[][];
    public static double arr[];
    public static Label graphLabel;
    Object rowdata[][]={};

    /**Field Area Name*/
    final static int AREA_FE = 0;
    /**Iterations*/
    final static int ITE_RAT = 1;
    /**Number of observations*/
    final static int N_OBS = 2 ;
    /**Distance(km)*/
    final static int X_KM = 3;
    /**Elevations(km)*/
    final static int ELE_KM = 4;
    /**observed anomalies*/
    final static int NOB_GOB = 5;
    /**Minimum anomaly*/
    final static int MIN_ANO = 6;
    /**Maximum anomaly*/
    final static int MAX_ANO =  7;
    /**Allowable Error*/
    final static int AL_ERR =   8;
    /**Maximum Depth*/
    final static int DEP_ZMAX = 9;


    public MODTOHAFSD_MainPanel(MODTOHAFSD_MainView cm){

        this.setLayout(new BorderLayout());

        p_East = new Panel ();
        p_Center = new Panel();
        p_North = new Panel ();
        graphLabel = new Label("GRAVITY MODELING OF 2.5-D SEDIMENTARY BASINS USING EXPONENTIAL
        DENSITY FUNCTION", Label.CENTER);
        graphLabel.setFont(new Font("Arial", 10, 18));
        p_Center.add(graphLabel);

        for(int i = 0; i < 10; i++){
            inputValues[i] = new TextField();
        }
        MODTOHAFSD_MainPanel.populateNorthPanel();
        MODTOHAFSD_TableView.populateEastPanel(rowdata);
        this.add(p_North, BorderLayout.NORTH);
        p_Center.setSize(1000, 760);
        this.add(p_Center, BorderLayout.CENTER);
        img.setEditable(false);
        p_Center.add(img);
        this.add(p_East, BorderLayout.EAST);
        this.setVisible(true);
        MenuBar mb = new MenuBar();
        cm.setMenuBar(mb);
        Menu file = new Menu("File");
        Menu file2 = new Menu("Model information");
```

```
    Menu file3 = new Menu("Interpretation");
    MenuItem i1,i2,i3,i5,i6,i7,i8,i21,i23,i10;
    file.add(i1 = new MenuItem("New"));
    file.add(i2 = new MenuItem("Load file"));
    file.add(i3 = new MenuItem("Save file"));
    file.add(i5 = new MenuItem("Clear"));
    file.add(i6 = new MenuItem("Exit"));
    Menu off  = new Menu("Offset");
    MenuItem off1,off2;
    off.add(off1 = new MenuItem("Constant offset"));
    off.add(off2 = new MenuItem("Variable offset"));
    file2.add(off);
    Menu half  = new Menu("Half strike");
    MenuItem h1,h2;
    half.add(h1 = new MenuItem("Constant half strike"));
    half.add(h2 = new MenuItem("Variable half strike"));
    file2.add(half);
    file2.addSeparator();
    file2.add(i7 = new MenuItem("Surface density & Lambda"));
    Menu reg  = new Menu("Regional");
    MenuItem r1,r2;
    reg.add(r1 = new MenuItem("Constant regional"));
    reg.add(r2 = new MenuItem("Variable regional"));
    file3.add(reg);
    file2.addSeparator();
    file3.add(i23 = new MenuItem("Draw/Edit polynomial"));
    file2.addSeparator();
    Menu mod  = new Menu("Analysis");
    MenuItem m1,m2;
    mod.add(m1 = new MenuItem("Modeling"));
    mod.add(m2 = new MenuItem("Inversion"));
    file3.add(mod);
    file3.add(i21 = new MenuItem("Save and Print"));
    Menu help  = new Menu("Help");
    help.add(i10 = new MenuItem("Flow module"));
    help.add(i8 = new MenuItem("Readme"));
    mb.add(file);
    mb.add(file2);
    mb.add(file3);
    mb.add(help);
    i1.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i2.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i3.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i5.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i6.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i7.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    off1.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    off2.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    h1.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    h2.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    r1.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    r2.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    m1.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    m2.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i21.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i23.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i10.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
    i8.addActionListener(new com.modtohafsd.control.MODTOHAFSD_Controller());
}

public static void populateNorthPanel(){

    p_North.setLayout(new GridLayout(5,3));

    p_North.add(new Label("Area /Profile"));
    p_North.add(inputValues[0]);
    p_North.add(new Label("Number of iterations"));
    p_North.add(inputValues[1]);
    p_North.add(new Label("Number of observations"));
    p_North.add(inputValues[2]);
    p_North.add(new Label("Station interval (km)"));
    p_North.add(inputValues[3]);
    p_North.add(new Label("Elevations(km)"));
    p_North.add(inputValues[4]);
    p_North.add(new Label("Observed anomalies (mGal)"));
```

```
        p_North.add(inputValues[5]);
        p_North.add(new Label("Minimum anomaly (mGal)"));
        p_North.add(inputValues[6]);
        p_North.add(new Label("Maximum anomaly (mGal)"));
        p_North.add(inputValues[7]);
        p_North.add(new Label("Permissible error"));
        p_North.add(inputValues[8]);
        p_North.add(new Label("Maximum permissible depth (km)"));
        p_North.add(inputValues[9]);
    }

    public static HashMap captureValues(){

        HashMap h_Map = new HashMap();

        try {

            h_Map.put("N_OBS", inputValues[N_OBS].getText());
            h_Map.put("X_KM", inputValues[X_KM].getText());
            h_Map.put("ELE_KM", inputValues[ELE_KM].getText());
            h_Map.put("NOB_GOB", inputValues[NOB_GOB].getText());
            h_Map.put("MIN_ANO", inputValues[MIN_ANO].getText());
            h_Map.put("MAX_ANO", inputValues[MAX_ANO].getText());
            h_Map.put("AL_ERR", inputValues[AL_ERR].getText());
            h_Map.put("DEP_ZMAX", inputValues[DEP_ZMAX].getText());
            h_Map.put("AREA_FE",inputValues[AREA_FE].getText());
            h_Map.put("ITE_RAT",inputValues[ITE_RAT].getText());
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        return h_Map;

    }

    public static void clearPanel(TextArea p) {

        Graphics g = p.getGraphics();
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 1280, 650);
    }

    public static void loadData(){
        try{
            String current = System.getProperty("user.dir");
            JFileChooser chooser=new  JFileChooser(current);
            int returnVal = chooser.showOpenDialog(null);
            int count = 0;
            if(returnVal == JFileChooser.APPROVE_OPTION) {
                File f = chooser.getSelectedFile();
                BufferedReader br=new BufferedReader(new FileReader(f));
                String st;
                st = br.readLine();
                count++;

                while((st)!=null){

                    try {

                        if (count==1){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.AREA_FE].setText
(""+st);
                        }
                        if (count==2){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.ITE_RAT].setText
(""+st);
                        }
                        if (count==3){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.N_OBS].setText("
"+st);
                        }
                        if (count==4){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.X_KM].setText(""
+st);
```

```
                        }
                        if (count==5){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.ELE_KM].setText(
""+st);
                        }
                        if (count==6){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.NOB_GOB].setText
(""+st);
                        }
                        if (count==7){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.MIN_ANO].setText
(""+st);
                        }
                        if (count==8){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.MAX_ANO].setText
(""+st);
                        }
                        if (count==9){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.AL_ERR].setText(
""+st);
                        }
                        if (count==10){
                            MODTOHAFSD_MainPanel.inputValues[MODTOHAFSD_MainPanel.DEP_ZMAX].setTex
t(""+st);
                        }
                        if (count==11){
                            MODTOHAFSD_LambInt.sdval = MODTOHAFSD_Utility.convertDouble(st);
                        }
                        if (count==12){
                            MODTOHAFSD_LambInt.lamval = MODTOHAFSD_Utility.convertDouble(st);
                        }
                        if (count==13){
                            MODTOHAFSD_GetConstOff.input_y_km =
MODTOHAFSD_Utility.convertDoubleArray(st);
                        }
                        if (count==14){
                            MODTOHAFSD_GetConstHaf.input_strike_km =
MODTOHAFSD_Utility.convertDoubleArray(st);
                        }
                        if (count==15){
                            MODTOHAFSD_CalculateValues.len =
MODTOHAFSD_Utility.convertInteger(st);
                        }
                        if (count==16){
                            MODTOHAFSD_PlotReg.val = MODTOHAFSD_Utility.convertDoubleArray(st);
                        }
                        if (count==17){
                            MODTOHAFSD_PlotReg.val1 = MODTOHAFSD_Utility.convertDoubleArray(st);
                        }
                        if (count==18){
                            MODTOHAFSD_GetDegPoly.input_deg_poly =
MODTOHAFSD_Utility.convertInteger(st);
                        }
                        if (count==19){
                            MODTOHAFSD_CalculateValues.d_cftnt_arr =
MODTOHAFSD_Utility.convertDoubleArray(st);
                        }
                        if (count==20){
                            MODTOHAFSD_GetConstReg.input_reg_val =
MODTOHAFSD_Utility.convertDoubleArray(st);
                        }
                    }
                    catch(Exception e) {

                        e.printStackTrace();
                    }
                    st = br.readLine();
                    count++;

                }
            }
        }
        catch(Exception e){

        }
```

```
    }


    public static void clearDefaultValues(){
        inputValues[N_OBS].setText("");
        inputValues[X_KM].setText("");
        inputValues[ELE_KM].setText("");
        inputValues[NOB_GOB].setText("");
        inputValues[MIN_ANO].setText("");
        inputValues[MAX_ANO].setText("");
        inputValues[AL_ERR].setText("");
        inputValues[DEP_ZMAX].setText("");
        inputValues[AREA_FE].setText("");
        inputValues[ITE_RAT].setText("");
    }
}
-----------------------------------------------------------------------------------------------------

package com.modtohafsd.view;

import java.applet.Applet;
import java.awt.*;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import java.text.DecimalFormat;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.constant.MODTOHAFSD_GetConstReg;


public class MODTOHAFSD_DrawGraph extends Applet {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    float maxZ,maxX,maxY,diff;
    public static double[] fc;
    double obs[];

    public void drawGraph(Graphics2D g2) {

        g2.setFont(new Font("Arial", 20, 12));
        g2.setColor(Color.BLACK);
        maxX = (float) MODTOHAFSD_CalculateValues.x[MODTOHAFSD_CalculateValues.input_n_obs];
        diff = (float) ( MODTOHAFSD_CalculateValues.input_ddx_km / 2);
        diff = (float) (450 * diff / maxX);
        g2.draw(new Line2D.Float(150-diff, 50 + 20,150-diff , 550 + 20));
        g2.drawLine(70, 45, 1040, 45);

        g2.drawString("DISTANCE(km)", 490, 315);
        String []a={"A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
        String []b={"D", "E", "P", "T", "H", "(k", "m)"};
        for (int i = 0; i < a.length; i++) {

            g2.drawString(""+a[i], 80, 20 + 60 + ( i * 20 ) );
        }

        for (int i = 0; i < b.length; i++) {

            g2.drawString(""+b[i], 80, 20 + 350 + ( i * 20 ) );
        }

    }

    public void plot(Graphics2D g) {

        g.setFont(new Font("Arial", 20, 12));
        g.setColor(Color.black);
        maxX = (float) MODTOHAFSD_CalculateValues.x[MODTOHAFSD_CalculateValues.input_n_obs];
        maxY = (float) MODTOHAFSD_CalculateValues.input_min_ano;
        double inidep = MODTOHAFSD_Utility.findMaximumNumber1(MODTOHAFSD_CalculateValues.o_dep);
        maxZ = (float) inidep;
        DecimalFormat df = new DecimalFormat("0.#");
```

```
    diff = (float) ( MODTOHAFSD_CalculateValues.input_ddx_km / 2);
    diff = (float) (450 * diff / maxX);
    g.drawString("0",120-diff,120);
    g.drawString("-",146-diff,124);
    g.drawString(""+MODTOHAFSD_CalculateValues.input_max_ano  ,115-diff,70);
    g.drawString("-",146-diff,74);
    g.drawString("|", 600, 330);
    g.drawString(""+df.format(MODTOHAFSD_CalculateValues.x[MODTOHAFSD_CalculateValues.input_n_
    obs]),597,350);
    float points = maxX / 5;
    int zInterval=50;
    DecimalFormat f = new DecimalFormat("0.#");
    int xInterval=90;
    g.drawString("|",150 , 330);
    g.drawString("0",147, 350);
    for (int x = xInterval, j = 1; x < 600; x+=xInterval) {

        if(j > 4)
            break;
        g.drawString("|",150 + x, 330);
        g.drawString("" + f.format(points*j), 147 + x, 350);
        j++;
    }


    float point = maxZ / 5 ;
    for (int x = zInterval + 250,j = 1; x < 550; x+=zInterval) {

        g.drawString("-", 146-diff, 50 + x + 23);
        g.drawString("" +df.format(point * j), 115-diff, 50 + x +20);
        j++;
    }
}


public void plotXYCoordinates (Graphics2D g2){

    g2.setFont(new Font("Arial", 20, 12));
    float maxY1;
    float prevx = 150;
    float prevy = 0;
    float prevy1 = 0;
    if (MODTOHAFSD_CalculateValues.o_gc[1] < 0){
        maxY1 = (float) MODTOHAFSD_CalculateValues.input_min_ano;
        prevy = 120 + (float)( ( 200 * (MODTOHAFSD_CalculateValues.in_gob[1]) / maxY1 ) );
    }
    else{
        maxY1 = (float) MODTOHAFSD_CalculateValues.input_max_ano;
        prevy = 120 - (float)( ( 50 * (MODTOHAFSD_CalculateValues.in_gob[1]) / maxY1 ) );
    }
    if (MODTOHAFSD_CalculateValues.input_nob_gob[1] < 0){
        maxY1 = (float) MODTOHAFSD_CalculateValues.input_min_ano;
        prevy1 = 120 + (float)( ( 200 * (MODTOHAFSD_CalculateValues.input_nob_gob[1]) / maxY1
        ) );
    }
    else{
        maxY1 = (float) MODTOHAFSD_CalculateValues.input_max_ano;
        prevy1 = 120 - (float)( ( 50 * (MODTOHAFSD_CalculateValues.input_nob_gob[1]) / maxY1 )
        );
    }

    float xpoint = 0;
    float ypoint = 0;
    float gypoint = 0;
    float gpoint = 0;

    for (int k = 1; k <= MODTOHAFSD_CalculateValues.input_n_obs; k++) {

        xpoint = (float)( 450 * MODTOHAFSD_CalculateValues.x[k] / maxX);
        //ypoint = (float)( ( 200 * MODTOHAFSD_CalculateValues.o_gc[k] / maxY ) );
        //gypoint = (float)( (200 * MODTOHAFSD_CalculateValues.in_gob[k] / maxY ) );

        if (MODTOHAFSD_CalculateValues.input_nob_gob[k] < 0){
            maxY1 = (float) MODTOHAFSD_CalculateValues.input_min_ano;
            gpoint = 120 + (float)( ( 200 * (MODTOHAFSD_CalculateValues.input_nob_gob[k]) /
```

```
maxY1 ) );
            }
            else{
                maxY1 = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                gpoint = 120 - (float)( ( 50 * (MODTOHAFSD_CalculateValues.input_nob_gob[k]) /
maxY1 ) );
            }
            if (MODTOHAFSD_CalculateValues.o_gc[k] < 0){
                maxY1 = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                ypoint = 120 + (float)( ( 200 * (MODTOHAFSD_CalculateValues.o_gc[k]) / maxY1 ) );
            }
            else{
                maxY1 = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                ypoint = 120 - (float)( ( 50 * (MODTOHAFSD_CalculateValues.o_gc[k]) / maxY1 ) );
            }
            if (MODTOHAFSD_CalculateValues.in_gob[k] < 0){
                maxY1 = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                gypoint = 120 + (float)( ( 200 * (MODTOHAFSD_CalculateValues.in_gob[k]) / maxY1 )
);
            }
            else{
                maxY1 = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                gypoint = 120 - (float)( ( 50 * (MODTOHAFSD_CalculateValues.in_gob[k]) / maxY1 )
);
            }

            g2.setFont(new Font("Arial", 20, 20));
            g2.setColor(Color.DARK_GRAY);
            g2.draw(new Line2D.Float(prevx, prevy, 150 + xpoint,  gypoint  ));
            g2.setColor(Color.RED);
            g2.draw(new Line2D.Float(prevx, prevy1, 150 + xpoint, gpoint ));

            g2.setColor(Color.BLUE);
            g2.setFont(new Font("Arial", 20, 40));
            g2.drawString(".", 150 + xpoint -6 ,  ypoint + 3);

            prevx = 150 + xpoint;
            prevy = gypoint;
            prevy1 = gpoint;

        }
    }


    public void plotXPoly (Graphics2D g2,int diff,int mul,int mul1) {

        int i_no_obs = MODTOHAFSD_CalculateValues.input_n_obs;
        obs = new double[i_no_obs + 1];
        for (int i = 1; i <= i_no_obs; i++) {
            obs[i] = MODTOHAFSD_CalculateValues.x[i];
        }
        maxX = (float) obs[i_no_obs];
        float s = 0;
        fc = new double[i_no_obs + 1];
        float spoint = 150;
        float fcpoint ;
        if (MODTOHAFSD_CalculateValues.d_cftnt_arr[1]<0){
            maxY = (float)(MODTOHAFSD_CalculateValues.input_min_ano);
            fcpoint = diff + (float)( mul * MODTOHAFSD_CalculateValues.d_cftnt_arr[1] / maxY);
        }
        else{
            maxY = (float)Math.abs(MODTOHAFSD_CalculateValues.input_max_ano);
            fcpoint = diff - (float)( mul1 * MODTOHAFSD_CalculateValues.d_cftnt_arr[1] / maxY);
        }
        float xpoint = 0;
        float ypoint = 0;
        for (int j = 1;j <= i_no_obs; j++) {
            fc[j] = 0;
        }
        for (int j = 1; j <= i_no_obs; j++) {

            s = (float) MODTOHAFSD_CalculateValues.x[j];

            for (int i = 1; i<=MODTOHAFSD_CalculateValues.i_d_poly + 1; i++){
                fc[j] = (float) (fc[j] + MODTOHAFSD_CalculateValues.d_cftnt_arr[i] * Math.pow(s, i
```

```
- 1));
                MODTOHAFSD_GetConstReg.input_reg_val[j] = fc[j];
            }
            xpoint = (float)( 450 * s / maxX);
            if(fc[j]<0){
                maxY = (float)(MODTOHAFSD_CalculateValues.input_min_ano);
                ypoint = diff + (float)( mul * fc[j] / maxY);
            }
            else{
                maxY = (float)Math.abs(MODTOHAFSD_CalculateValues.input_max_ano);
                ypoint = diff - (float)( mul1 * fc[j] / maxY);
            }

            g2.setColor(Color.GREEN);
            g2.draw(new Line2D.Float(spoint, fcpoint, (float)150 + xpoint, ypoint));
            spoint = (float) 150 + xpoint;
            fcpoint =  ypoint;
        }
    }


    public void plotZCoordinates (Graphics2D g2) {

        g2.setFont(new Font("Arial", 20, 12));
        float xpoint = 0;
        float xpoint1 = 0;
        float zpoint = 0;
        g2.setColor(Color.BLACK);
        g2.drawString("0", 105,330);
        DecimalFormat f = new DecimalFormat("0.#");
        int yInterval=50;
        float points = maxY / 4;
        for (int x = yInterval, j = 1; x < 250; x+=yInterval){

            g2.drawString("-", 146-diff, 100 + x + 24);
            g2.drawString("" + f.format(points * j), 115-diff, 100 + x + 20);
            j++;
        }
        g2.setColor(Color.RED);
        g2.fill(new Rectangle2D.Float(150-diff, 321, 450+diff+diff , 250 ));
        Color col[] =
        {Color.YELLOW,Color.PINK,Color.ORANGE,Color.PINK,Color.YELLOW,Color.ORANGE,Color.WHITE,Col
        or.PINK,Color.YELLOW,Color.ORANGE,Color.WHITE};
        Color col1[] =
        {Color.MAGENTA,Color.GREEN,Color.BLUE,Color.BLUE,Color.CYAN,Color.BLUE,Color.GREEN,Color.C
        YAN,Color.GREEN,Color.DARK_GRAY,Color.MAGENTA};
        for (int i = 0; i < MODTOHAFSD_CalculateValues.input_lambda_st; i++){
            for (int k = 1; k <= MODTOHAFSD_CalculateValues.input_n_obs; k++) {

                if(i>11){
                    col[i] = Color.YELLOW;
                    col1[i] = Color.MAGENTA;
                }
                diff = (float) ( MODTOHAFSD_CalculateValues.input_ddx_km / 2);
                diff = (float) (450 * diff / maxX);
                xpoint = (float) (450 * MODTOHAFSD_CalculateValues.x[k] / maxX);
                if (k == 1){
                    GradientPaint gradient = new GradientPaint(10, 10, Color.YELLOW, 30, 200,
Color.MAGENTA, true);
                    g2.setPaint(gradient);
                    zpoint =(float) (250 * MODTOHAFSD_CalculateValues.o_dep[k] / maxZ);
                    g2.fill(new Rectangle2D.Float(150-diff, 320,diff+diff, zpoint ));
                    //g2.fill(new Rectangle2D.Float(150+xpoint, 320, diff, zpoint ));
                }
                else{
                    if(k<MODTOHAFSD_CalculateValues.input_n_obs) {
                        xpoint1 = (float) (450 * MODTOHAFSD_CalculateValues.x[k+1] / maxX);
                    }
                    zpoint =(float) (250 * MODTOHAFSD_CalculateValues.o_dep[k] / maxZ);

                    GradientPaint gradient = new GradientPaint(10, 10, col[i], 30, 200, col1[i],
true);
                    g2.setPaint(gradient);
                    g2.fill(new Rectangle2D.Float(150+xpoint-diff ,320,xpoint1-xpoint, zpoint ));
                }
```

```
            if(k==MODTOHAFSD_CalculateValues.input_n_obs){
                zpoint =(float) (250 * MODTOHAFSD_CalculateValues.o_dep[k]/ maxZ);
                g2.fill(new Rectangle2D.Float(150+xpoint-diff, 320,diff+diff, zpoint ));
                //g2.fill(new Rectangle2D.Float(150+xpoint-diff ,320,diff, zpoint ));
            }

        }
    }
    g2.setColor(Color.BLACK);
    g2.draw(new Line2D.Float(150-diff, 320, 600+diff, 320 ));
}

public void plotZCoordinatesele (Graphics2D g2) {

    g2.setFont(new Font("Arial", 20, 12));
    DecimalFormat f = new DecimalFormat("0.#");
    int yInterval=50;
    float points = maxY / 4;
    for (int x = yInterval, j = 1; x < 250; x+=yInterval){
        if(j>3)
            break;
        g2.drawString("-", 146-diff, 100 + x + 24);
        g2.drawString("" + f.format(points * j), 115-diff, 100 + x + 20);
        j++;
    }

    float maxEle = (float)
    MODTOHAFSD_Utility.findMaximumNumber1(MODTOHAFSD_CalculateValues.input_ele_km);
    maxEle = Math.abs(maxEle);
    float xpoint = 0;
    float prevx = 150;
    float elepoint[] = new float[MODTOHAFSD_CalculateValues.input_n_obs+1];
    float preelepoint[] = new float[MODTOHAFSD_CalculateValues.input_n_obs+1];
    float dis[] = new float[MODTOHAFSD_CalculateValues.input_n_obs+1];
    float predis[] = new float[MODTOHAFSD_CalculateValues.input_n_obs+1];
    float preele = 320 - (float)( ( 25 * Math.abs(MODTOHAFSD_CalculateValues.input_ele_km[1])
    / maxEle ) );
    float xpoint1 = 0;
    float zpoint = 0;

    g2.setColor(Color.RED);
    g2.fill(new Rectangle2D.Float(150-diff, 295, 450+diff+diff , 275 ));
    Color col = Color.YELLOW;
    Color col1 = Color.MAGENTA;

    for (int k = 1; k <= MODTOHAFSD_CalculateValues.input_n_obs; k++) {


        diff = (float) ( MODTOHAFSD_CalculateValues.input_ddx_km / 2);
        diff = (float) (450 * diff / maxX);
        xpoint = (float) (450 * MODTOHAFSD_CalculateValues.x[k] / maxX);
        //elepoint[k] = (float) (25 * Math.abs(MODTOHAFSD_CalculateValues.input_ele_km[k]) /
        maxEle);
        if (k == 1){
            GradientPaint gradient = new GradientPaint(10, 10, col, 30, 190, col1, true);
            g2.setPaint(gradient);
            zpoint =(float) (275 * MODTOHAFSD_CalculateValues.o_dep[k] / maxZ);
            g2.fill(new Rectangle2D.Float(150-diff, 295,diff+diff, zpoint ));
            //g2.fill(new Rectangle2D.Float(150+xpoint, 295, diff, zpoint ));
        }
        else{
            if(k<MODTOHAFSD_CalculateValues.input_n_obs) {
                xpoint1 = (float) (450 * MODTOHAFSD_CalculateValues.x[k+1] / maxX);
            }
            zpoint =(float) (275 * MODTOHAFSD_CalculateValues.o_dep[k] / maxZ);
            GradientPaint gradient = new GradientPaint(10, 10, col, 30, 190, col1, true);
            g2.setPaint(gradient);
            g2.fill(new Rectangle2D.Float(150+xpoint-diff ,295,xpoint1-xpoint,zpoint ));
        }
        if(k==MODTOHAFSD_CalculateValues.input_n_obs){
            zpoint =(float) (275 * MODTOHAFSD_CalculateValues.o_dep[k]/ maxZ);
            g2.fill(new Rectangle2D.Float(150+xpoint-diff, 295,diff+diff, zpoint ));
            //g2.fill(new Rectangle2D.Float(150+xpoint-diff ,295,diff, zpoint ));
        }
```

```
        }

        for (int k = 1; k <= MODTOHAFSD_CalculateValues.input_n_obs; k++) {
            g2.setColor(Color.BLACK);
            xpoint = (float) (450 * MODTOHAFSD_CalculateValues.x[k] / maxX);
            dis[k] = xpoint;
            elepoint[k] = 320 - (float) (25 * Math.abs(MODTOHAFSD_CalculateValues.input_ele_km[k])
            / maxEle);
            preelepoint[k]= preele;
            predis[k]= prevx;
            g2.draw(new Line2D.Float(prevx, preele, (float)150 + xpoint, elepoint[k]));
            prevx = 150+xpoint;
            preele = elepoint[k];
        }
        for(float i = (float)1.0; i<=25;){
            for (int k = 1; k <= MODTOHAFSD_CalculateValues.input_n_obs; k++) {
                g2.setColor(Color.WHITE);
                g2.draw(new Line2D.Float(predis[k]-diff, preelepoint[k]-i, (float)150+dis[k]+diff,
elepoint[k]-i));
            }
            i=(float) (i+0.5);

        }
        g2.setColor(Color.BLACK);
        //g2.drawLine(150, 50 + 20, 150, 550 + 20);
        g2.drawString("DISTANCE(km)", 333, 315);
    }

    public void drawOBJ(Graphics2D g2) {

        g2.setFont(new Font("Arial", 20, 12));
        g2.setColor(Color.BLACK);
        g2.drawLine(780, 45, 780, 580);
        g2.drawLine(70, 580, 1040, 580);
        g2.drawLine(1040, 45, 1040, 580);
        g2.drawLine(70, 45, 70, 580);

        g2.drawLine(820, 70, 820, 160);
        g2.drawLine(820, 160, 910, 160);
        g2.drawString("J", 800, 90);

        double maxOb = MODTOHAFSD_Utility.findMaximumNumber1(MODTOHAFSD_CalculateValues.o_funct);
        int ini = MODTOHAFSD_Utility.findMaximumNumber(MODTOHAFSD_CalculateValues.o_iter);
        if(ini == 5)
            ini = ini + 1;
        int maxiter = ( ini / 3 * 5 ) * 2;
        int point;
        int xInterval = 22;
        point = ( ( ini ) / 3 * 5 ) / 5;
        for (int x = xInterval, j = 1; x < 90 ; x += xInterval) {
            if(ini>1000){
                g2.drawString("'", 821 + x, 170);
                g2.drawString("" + (point * j), 820 + x-20 +(5*j) , 175);
            }
            else{
                g2.drawString("'", 821 + x, 170);
                g2.drawString("" + (point * j), 820 + x - 3, 175);
            }
            j++;
        }
        float prevx = 820;
        float prevy = 70;
        float xpoint = 0;
        float ypoint = 0;
        DecimalFormat d1= new DecimalFormat("0.#####");
        for (int i = 1; i <= MODTOHAFSD_CalculateValues.o_iter; i++) {

            xpoint = (float)( 250 * i / maxiter );
            ypoint = 70 + (float) - ( ( 90 * (MODTOHAFSD_CalculateValues.o_funct[i]) / maxOb ) );
            if(i == MODTOHAFSD_CalculateValues.o_iter){
                g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint - 4, 90 + ypoint));
            }
            else {
                g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 90 + ypoint));
            }
```

```
            prevx = 820 + xpoint;
            prevy = 90 + ypoint;

        }
        DecimalFormat d= new DecimalFormat("0.#");
        g2.drawString(" "+d.format(MODTOHAFSD_CalculateValues.o_funct[1]), 780, 70);
        g2.drawString("
        "+d1.format(MODTOHAFSD_CalculateValues.o_funct[MODTOHAFSD_CalculateValues.o_iter]), 820 +
        xpoint, 90 + ypoint);
        g2.setFont(new Font("Arial", 40,11));
        g2.drawString ("Iterations",850,186);
    }

    public void drawSd(Graphics2D g2,Color col,int n) {

        g2.setFont(new Font("Arial", 20, 12));
        g2.setColor(Color.black);
        g2.drawLine(780, 200, 1040, 200);
        g2.setColor(Color.red);
        g2.setFont(new Font("Arial", 20, 12));
        DecimalFormat d= new DecimalFormat("0.#");
        DecimalFormat d1= new DecimalFormat("0.###");
        double inidep = MODTOHAFSD_Utility .findMaximumNumber1(MODTOHAFSD_CalculateValues.o_dep);
        maxZ = (float) inidep;
        g2.draw(new Line2D.Float(820, 320, 820,  570));

        g2.drawString(""+d.format(inidep), 800, (float)(320 +  250 * inidep / maxZ));
        g2.drawString("-",820,(float)(320 +  250 * inidep / maxZ) + 2);
        g2.drawString("0",807 , 320);
        g2.drawLine(820, 320, 910, 320);

        double maxOb1 =
        Math.abs(MODTOHAFSD_Utility.findMaximumNumber1(MODTOHAFSD_CalculateValues.input_sd_val));

        DecimalFormat df = new DecimalFormat("0.#");
        float points = maxZ / 5 ;
        int zInterval = 50;
        for (int x = zInterval+250,j = 1; x < 550; x+=zInterval) {

            if(j>4)
                break;

            g2.drawString("-", 820, 50 + x + 22);
            g2.drawString("" +df.format(points * j), 800,50 + x + 20);
            j++;

        }

        float prevx = 820+ (float) ( ( 90 * ( Math.abs(MODTOHAFSD_CalculateValues.vsd[1] )) /
        maxOb1 ) );
        float prevy = 320;
        float xpoint = 0;
        float ypoint = 0;
        for (int i = 1; i <= MODTOHAFSD_CalculateValues.count; i++) {
            xpoint = (float)( 90 * Math.abs(MODTOHAFSD_CalculateValues.vsd[i]) / maxOb1 );
            ypoint = (float)( 250 * MODTOHAFSD_CalculateValues.dep[i] / maxZ );
            g2.setColor(col);
            g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 320 + ypoint));
            prevx = 820 + xpoint;
            prevy = 320 + ypoint;
        }

        //g2.drawString(""+MODTOHAFSD_GetLambdaVal.start[n]+"-"+MODTOHAFSD_GetLambdaVal.end[n],940
        ,335+n*15);
        //g2.drawString(""+d1.format(MODTOHAFSD_GetLambdaVal.lambda[n] ),990,335+n*15);
        g2.setColor(Color.black);
        g2.drawString(""+d1.format(-maxOb1 ),900 ,320 );
        g2.setFont(new Font("Arial", 20,15));
        g2.drawString("Variation of density contrast " , 800,235);
        g2.drawString("with depth" , 850,255);
        g2.setFont(new Font("Arial", 40,11));
        g2.drawString ("Density contrast",830,285);
        g2.drawString ("(gm/cc)",843,295);
        g2.drawString("Z(km)", 790,(float)( 320+((250*inidep/maxZ))/2));
```

```
    }

    public void idex(Graphics2D g){

        g.setColor(Color.BLUE);
        g.setFont(new Font("Arial", 20, 50));
        g.drawString(" ...    ",605,190);
        g.setFont(new Font("Arial", 20, 12));
        g.drawString("Modeled anomalies",660,190);
        g.setColor(Color.RED);
        g.drawString("____:",625,120);
        g.drawString("Bouguer anomalies",660,120);
        g.setColor(Color.DARK_GRAY);
        g.drawString("____:",625,170);
        g.drawString("Residual anomalies",660,170);
        g.setColor(Color.GREEN);
        g.drawString("____:",625,140);
        g.drawString("Assumed/Estimated",660,140);
        g.drawString(" regional",670,150);
        GradientPaint gradient = new GradientPaint(10, 10, Color.yellow, 30, 100, Color.MAGENTA,
        true);
        g.setPaint(gradient);
        g.fillRect(645, 395, 30, 20);
        g.setColor(Color.black);
        g.drawString("Estimated depth ",680,410);
        g.drawString(" structure",685,420);
        g.setColor(Color.red);
        g.fillRect(645, 425, 30, 20);
        g.setColor(Color.black);
        g.drawString("Basement ",680,440);
    }

}

-----------------------------------------------------------------------------------------------

package com.modtohafsd.view;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.view.getvalues.MODTOHAFSD_GetHalfstrikeVal;

public class MODTOHAFSD_HalfstrikeVal implements ActionListener {
    public static JTable table;
    Button bt  = new Button("Submit");
    public static Object haf[][];
    public  void val() {
        haf  = new Object[MODTOHAFSD_CalculateValues.input_n_obs+1][2];
        com.modtohafsd.view.MODTOHAFSD_MainPanel.p_Center.removeAll();
        com.modtohafsd.view.MODTOHAFSD_MainPanel.p_Center.repaint();
        Label label = new Label("Column A: Prism number ", Label.CENTER);
        label.setFont(new Font("Arial", 10, 18));
        Label label1 = new Label("Column B: Halfstrike  ", Label.CENTER);
        label1.setFont(new Font("Arial", 10, 18));
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);
        MODTOHAFSD_MainPanel.p_Center.add(label);
        MODTOHAFSD_MainPanel.p_Center.add(label1);

        table = new JTable(MODTOHAFSD_CalculateValues.input_n_obs, 2);
        table.setPreferredScrollableViewportSize(new Dimension(300,300));

        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);

        MODTOHAFSD_MainPanel.p_Center.add(scrollPane);
        MODTOHAFSD_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        MODTOHAFSD_MainPanel.p_Center.validate();
        com.modtohafsd.view.MODTOHAFSD_MainPanel.p_Center.setVisible(true);

    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
```

```
            MODTOHAFSD_GetHalfstrikeVal ghv = new MODTOHAFSD_GetHalfstrikeVal();
            ghv.getVal();
        }
    }
}
```

```
package com.modtohafsd.view;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.view.getvalues.MODTOHAFSD_GetOffsetVal;

public class MODTOHAFSD_OffsetVal implements ActionListener {
    public static JTable table;
    Button bt  = new Button("Submit");
    public static Object off[][];
    public  void val() {
        off  = new Object[MODTOHAFSD_CalculateValues.input_n_obs+1][2];
        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.repaint();
        Label label = new Label("Column A: Prism number ", Label.CENTER);
        label.setFont(new Font("Arial", 10, 18));
        Label label1 = new Label("Column B: Offset value ", Label.CENTER);
        label1.setFont(new Font("Arial", 10, 18));
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);
        MODTOHAFSD_MainPanel.p_Center.add(label);
        MODTOHAFSD_MainPanel.p_Center.add(label1);
        table = new JTable(MODTOHAFSD_CalculateValues.input_n_obs, 2);
        table.setPreferredScrollableViewportSize(new Dimension(300,300));
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);
        MODTOHAFSD_MainPanel.p_Center.add(scrollPane);
        MODTOHAFSD_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        MODTOHAFSD_MainPanel.p_Center.validate();
        MODTOHAFSD_MainPanel.p_Center.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            MODTOHAFSD_GetOffsetVal gfv = new MODTOHAFSD_GetOffsetVal();
            gfv.getVal();
        }
    }
}
```

```
package com.modtohafsd.view;

import java.awt.*;
import javax.swing.JScrollPane;
import javax.swing.JTable;


public class MODTOHAFSD_TableView extends Panel{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public static TextArea val = new TextArea(5,5);


    public static  void populateEastPanel(Object rowData[][]) {
        MODTOHAFSD_MainPanel.p_East.removeAll();
        MODTOHAFSD_MainPanel.p_East.setLayout(new GridLayout(2,1));
        Object columnNames[] = {"Distance(km)", "Bouguer anamolies (mGal)","Estimated regional
        (mGal)", "Residual anamolies (mGal)", "Modeled anamolies
        (mGal)","Error(mGal)","Depth(km)"};
        JTable table = new JTable(rowData, columnNames);
        table.setPreferredScrollableViewportSize(new Dimension(300,550));
        JScrollPane scrollPane = new JScrollPane(table);
```

```
        scrollPane.setAutoscrolls(true);
        MODTOHAFSD_MainPanel.p_East.add(scrollPane);
        val.setEditable(false);
        MODTOHAFSD_MainPanel.p_East.add(val);
        MODTOHAFSD_MainPanel.p_East.validate();
        MODTOHAFSD_MainPanel.p_East.setVisible(true);
    }
    public static  void populateEastPanel1(Object rowData[][]) {
        MODTOHAFSD_MainPanel.p_East.removeAll();
        MODTOHAFSD_MainPanel.p_East.setLayout(new GridLayout(2,1));
        Object columnNames[] = {"Distance(km)", "Bouguer anamolies (mGal)","Estimated regional
        (mGal)", "Residual anamolies (mGal)"};
        JTable table = new JTable(rowData, columnNames);
        table.setPreferredScrollableViewportSize(new Dimension(300,550));
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);
        MODTOHAFSD_MainPanel.p_East.add(scrollPane);
        val.setEditable(false);
        MODTOHAFSD_MainPanel.p_East.add(val);
        MODTOHAFSD_MainPanel.p_East.validate();
        MODTOHAFSD_MainPanel.p_East.setVisible(true);
    }

}
```

```
package com.modtohafsd.view.constant;

import java.awt.*;
import java.awt.event.*;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;

public class MODTOHAFSD_GetConstHaf implements ActionListener{
    public int hafval=0;
    TextField haf = new TextField(12);
    Button bt = new Button("Submit");
    public static double input_strike_km[] = new double[MODTOHAFSD_CalculateValues.input_n_obs+1];
    public void constHaf(){
        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.validate();
        MODTOHAFSD_MainPanel.p_Center.repaint();
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);

        MODTOHAFSD_MainPanel.p_Center.add(new Label("Halfstrike(km):",Label.LEFT));
        MODTOHAFSD_MainPanel.p_Center.add(haf);
        MODTOHAFSD_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        MODTOHAFSD_MainPanel.p_Center.validate();
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            try {
                hafval = MODTOHAFSD_Utility.convertInteger(haf.getText());
            } catch (Exception e) {

                e.printStackTrace();
            }
            for(int i =1;i<= MODTOHAFSD_CalculateValues.input_n_obs;i++){

                input_strike_km[i]= hafval;

            }

        }
    }
}
```

```
package com.modtohafsd.view.constant;

import java.awt.*;
import java.awt.event.*;
```

```
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;

public class MODTOHAFSD_GetConstOff implements ActionListener {
    public double offval=0;
    TextField offset = new TextField(12);
    public static double input_y_km[]= new double[MODTOHAFSD_CalculateValues.input_n_obs+1];
    Button bt = new Button("Submit");
    public void constOff(){
        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.validate();
        com.modtohafsd.view.MODTOHAFSD_MainPanel.p_Center.repaint();
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);
        MODTOHAFSD_MainPanel.p_Center.add(new Label("Offset(km):",Label.LEFT));
        MODTOHAFSD_MainPanel.p_Center.add(offset);
        MODTOHAFSD_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        MODTOHAFSD_MainPanel.p_Center.validate();
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            try {
                offval = MODTOHAFSD_Utility.convertDouble(offset.getText());
            } catch (Exception e) {

                e.printStackTrace();
            }
            for(int i =1;i<= MODTOHAFSD_CalculateValues.input_n_obs;i++){

                input_y_km[i]= offval;

            }
        }
    }
}
-----------------------------------------------------------------------------------------------

package com.modtohafsd.view.constant;

import java.awt.*;
import java.awt.event.*;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;

public class MODTOHAFSD_GetConstReg implements ActionListener {
    public static double regval=0;
    TextField reg = new TextField(12);
    Button bt = new Button("Submit");
    public static double input_reg_val[] = null;//new
    double[MODTOHAFSD_CalculateValues.input_n_obs+1];
    public void constReg(){
        input_reg_val=new double[MODTOHAFSD_CalculateValues.input_n_obs+1];
        for(int i =1;i<= MODTOHAFSD_CalculateValues.input_n_obs;i++){
            input_reg_val[i]= 0;
        }
        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.validate();
        com.modtohafsd.view.MODTOHAFSD_MainPanel.p_Center.repaint();
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);

        MODTOHAFSD_MainPanel.p_Center.add(new Label("Regional(mGal):",Label.LEFT));
        MODTOHAFSD_MainPanel.p_Center.add(reg);
        MODTOHAFSD_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        MODTOHAFSD_MainPanel.p_Center.validate();
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            try {
                regval = MODTOHAFSD_Utility.convertDouble(reg.getText());
            } catch (Exception e) {

                e.printStackTrace();
```

```
            }
            for(int i =1;i<= MODTOHAFSD_CalculateValues.input_n_obs;i++){
                input_reg_val[i]= regval;
            }
            MODTOHAFSD_GetDegPoly.input_deg_poly=0;
            //MODTOHAFSD_CalculateValues.d_cftnt_arr[1] = regval;
        }
    }
}
```

```
package com.modtohafsd.view.constant;

import java.awt.*;
import java.awt.event.*;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;
import com.modtohafsd.view.event.MODTOHAFSD_PlotReg;

public class MODTOHAFSD_GetDegPoly implements ActionListener{
    public int polyval=0;
    TextField p = new TextField(12);
    Button bt = new Button("Submit");
    public static int input_deg_poly=0;
    public void constHaf(){
        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.validate();
        com.modtohafsd.view.MODTOHAFSD_MainPanel.p_Center.repaint();
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);

        MODTOHAFSD_MainPanel.p_Center.add(new Label("Degree of polynomial:",Label.LEFT));
        MODTOHAFSD_MainPanel.p_Center.add(p);
        MODTOHAFSD_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);

        MODTOHAFSD_MainPanel.p_Center.validate();
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            try {
                polyval = MODTOHAFSD_Utility.convertInteger(p.getText());
            } catch (Exception e) {

                e.printStackTrace();
            }
            input_deg_poly  = polyval;

            MODTOHAFSD_PlotReg pf = new MODTOHAFSD_PlotReg();
            Graphics g = MODTOHAFSD_MainPanel.p_Center.getGraphics();
            pf.paint(g);

        }
    }
}
```

```
package com.modtohafsd.view.constant;

import java.awt.*;
import java.awt.event.*;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.*;


public class MODTOHAFSD_LambInt implements ActionListener {
    public static double lamval=0;
    public static double sdval=0;
    public static TextField lam = new TextField(12);
    public static TextField sd = new TextField(12);
    Button bt = new Button("Submit");
    public void constLamSd(){
        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.validate();
        MODTOHAFSD_MainPanel.p_Center.repaint();
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);
```

```
        MODTOHAFSD_MainPanel.p_Center.add(new Label("Surface density
        contrast(gm/cc):",Label.LEFT));
        MODTOHAFSD_MainPanel.p_Center.add(sd);
        MODTOHAFSD_MainPanel.p_Center.add(new Label("Lambda(1/km):",Label.LEFT));
        MODTOHAFSD_MainPanel.p_Center.add(lam);
        MODTOHAFSD_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        MODTOHAFSD_MainPanel.p_Center.validate();
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            try {
                lamval = MODTOHAFSD_Utility.convertDouble(lam.getText());
                sdval = MODTOHAFSD_Utility.convertDouble(sd.getText());
            } catch (Exception e) {

                e.printStackTrace();
            }


        }
    }
}
-----------------------------------------------------------------------------------------------

package com.modtohafsd.view.event;

import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_HandleException;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;
import com.modtohafsd.view.MODTOHAFSD_TableView;
import com.modtohafsd.view.event.MODTOHAFSD_PlotReg;
import com.modtohafsd.view.graph.MODTOHAFSD_PlainGraph;

public class MODTOHAFSD_EditReg {
    public static float zpoint = 0;
    public static int count = 0;
    public static MouseListener ml1;
    com.modtohafsd.view.MODTOHAFSD_DrawGraph dg = new com.modtohafsd.view.MODTOHAFSD_DrawGraph();
    MODTOHAFSD_PlainGraph pg = new MODTOHAFSD_PlainGraph();
    com.modtohafsd.model.MODTOHAFSD_CalculateValues cv = new
    com.modtohafsd.model.MODTOHAFSD_CalculateValues();

    public void paint(Graphics g){
        final DecimalFormat f = new DecimalFormat("0.##");
        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.im2);
        MODTOHAFSD_MainPanel.im2.setEditable(false);
        MODTOHAFSD_MainPanel.im2.setBackground(Color.WHITE);
        MODTOHAFSD_MainPanel.p_Center.validate();
        cv.getAnamolyValues(com.modtohafsd.view.MODTOHAFSD_MainPanel.captureValues());
        try {
            cv.getCoefficients();
        } catch (MODTOHAFSD_HandleException e2) {

        }

        Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im2.getGraphics();
        g2.setFont(new Font("Arial", 20, 12));
        pg.drawPlainGraph(g2);
        dg.plotXPoly(g2,300,250,250);
        pg.newAno(g2);
        pg.drawPlainGraph(g2);
        g2.setColor(Color.black);
        float ypoint,maxY;
        for(int i = 1;i<= MODTOHAFSD_CalculateValues.len;i++){
            float xpoint  = (float)
            (450*MODTOHAFSD_CalculateValues.d_x_km_arr[i]/MODTOHAFSD_PlainGraph.maxX1);
            if(MODTOHAFSD_CalculateValues.d_z_km_arr[i]<0){

                maxY = (float)(MODTOHAFSD_CalculateValues.input_min_ano);
```

```
                ypoint = 300+(float)( 250 *  MODTOHAFSD_CalculateValues.d_z_km_arr[i]/ maxY);
            }
            else{

                maxY = (float)Math.abs(MODTOHAFSD_CalculateValues.input_max_ano);
                ypoint = 300-(float)( 250 * MODTOHAFSD_CalculateValues.d_z_km_arr[i] / maxY);

            }

            g2.setFont(new Font("Arial", 20, 12));
            g2.drawString("("+f.format(MODTOHAFSD_CalculateValues.d_z_km_arr[i])+")",
            150+xpoint-8, ypoint-10);
            g2.drawString("X", 150+xpoint-3, ypoint+3);
        }

        ml1 = new MouseAdapter(){

            public void mouseDragged(MouseEvent e1) {
                DecimalFormat f = new DecimalFormat("0.##");

                float xer = e1.getX();
                float yer = e1.getY();
                float maxAno,ypoint ;
                if(yer>300){
                    maxAno = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                    ypoint = (float) (maxAno * (yer - 300) / 250);
                }
                else{

                    maxAno = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                    ypoint = (float) Math.abs(maxAno * (300 - yer) / 250);
                }

                float xpoint =(float) Math.abs((MODTOHAFSD_PlainGraph.maxX1 * (150 - xer)) / 450);
                Graphics2D g = (Graphics2D)MODTOHAFSD_MainPanel.im2.getGraphics();
                if (yer >= 50 && yer <= 550 && xer >= 150 && xer <= 600){

                    g.setColor(Color.black);
                    g.drawString("X-coordinates            Y-coordinates", 780, 320);
                    g.setColor(Color.red);
                    g.fillRect(780, 320, 180, 20);
                    g.setColor(Color.black);
                    g.drawString(""+f.format(xpoint)+"
"+f.format(ypoint)+"", 780, 330);
                }
                pg.drawPlainGraph(g);
                dg.plotXPoly(g,300,250,250);
                pg.newAno(g);
                pg.drawPlainGraph(g);
            }


            public void mousePressed(MouseEvent e1) {

                float x1 = e1.getX();
                float y1 = e1.getY();
                float diff = (float)
(MODTOHAFSD_CalculateValues.x[MODTOHAFSD_CalculateValues.input_n_obs]/100);
                float xer = (float) Math.abs((MODTOHAFSD_PlainGraph.maxX1 * (150 - (x1-3))) /
450);
                Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im2.getGraphics();
                if (y1 > 50 && y1 < 550 && x1 <= 600){
                    for (int kk = 1; kk <= MODTOHAFSD_CalculateValues.len; kk++){
                        if (Math.abs((float)MODTOHAFSD_CalculateValues.d_x_km_arr[kk] - xer )<=
diff  ){
                            count = kk;
                        }
                        g2.setColor(Color.white);
                        g2.drawString(""+count, 750, 380);
                    }
                }
                pg.drawPlainGraph(g2);
                dg.plotXPoly(g2,300,250,250);
                pg.newAno(g2);
                pg.drawPlainGraph(g2);
```

```
            }
            public void mouseMoved(MouseEvent e){

                DecimalFormat f = new DecimalFormat("0.##");
                float xer = e.getX();
                float yer = e.getY();
                float maxAno,ypoint ;
                if(yer>300){
                    maxAno = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                    ypoint = (float) (maxAno * (yer - 300) / 250);
                }
                else{
                    maxAno = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                    ypoint = (float) Math.abs(maxAno * (300 - yer) / 250);
                }

                float xpoint =(float) Math.abs((MODTOHAFSD_PlainGraph.maxX1 * (150 - xer)) / 450);
                Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im2.getGraphics();
                if (yer >= 50 && yer <= 550 && xer >= 150 && xer <= 600){

                    g2.setColor(Color.black);
                    g2.drawString("X-coordinates            Y-coordinates", 780, 320);
                    g2.setColor(Color.red);
                    g2.fillRect(780, 320, 180, 20);
                    g2.setColor(Color.black);
                    g2.drawString(""+f.format(xpoint)+"
"+f.format(ypoint)+"", 780, 330);
                }
                pg.drawPlainGraph(g2);
                dg.plotXPoly(g2,300,250,250);
                pg.newAno(g2);
                pg.drawPlainGraph(g2);
            }
            public void mouseReleased(MouseEvent e1) {

                float x = e1.getX();
                float y = e1.getY();
                float maxAno,ypoint ,maxY;
                if(y>300){
                    maxAno = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                    ypoint = (float) (maxAno * (y - 300) / 250);
                }
                else{

                    maxAno = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                    ypoint = (float) Math.abs(maxAno * (300 - y) / 250);
                }

                Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im2.getGraphics();
                MODTOHAFSD_PlainGraph pg = new MODTOHAFSD_PlainGraph();
                if (y > 50 && y <= 550 && x <= 600){

                    MODTOHAFSD_PlotReg.val1[count]= ypoint;
                    MODTOHAFSD_CalculateValues.d_z_km_arr[count] = ypoint;

                }
                try {
                    cv.getCoefficients();
                } catch (MODTOHAFSD_HandleException e) {

                }
                MODTOHAFSD_MainPanel.clearPanel(MODTOHAFSD_MainPanel.im2);
                DecimalFormat d = new DecimalFormat("0.##");
                DecimalFormat df1 = new DecimalFormat("0.####");
                Graphics2D g1 = (Graphics2D)MODTOHAFSD_MainPanel.im2.getGraphics();
                pg.drawPlainGraph(g1);
                dg.plotXPoly(g1,300,250,250);
                pg.newAno(g1);
                com.modtohafsd.view.MODTOHAFSD_TableView.populateEastPanel1(MODTOHAFSD_PlainGraph.
obj1);
                MODTOHAFSD_TableView.val.setText("");
                MODTOHAFSD_TableView.val.appendText("\n");
                MODTOHAFSD_TableView.val.append("Polynomial coefficients of regional
background:-\n");
                MODTOHAFSD_TableView.val.appendText("----------------------------------------\n
```

```
");
                MODTOHAFSD_TableView.val.appendText("\n");

                for (int i = 1; i <= MODTOHAFSD_CalculateValues.i_d_poly+1; i++){
                    MODTOHAFSD_TableView.val.appendText("f"+(i - 1)+" = "
+df1.format(MODTOHAFSD_CalculateValues.d_cftnt_arr[i])+ "\n");
                }

                g1.setColor(Color.black);
                for(int i = 1;i<= MODTOHAFSD_CalculateValues.len;i++){
                    float xpoint  = (float)
(450*MODTOHAFSD_CalculateValues.d_x_km_arr[i]/MODTOHAFSD_PlainGraph.maxX1);
                    if(MODTOHAFSD_CalculateValues.d_z_km_arr[i]<0){

                        maxY = (float)(MODTOHAFSD_CalculateValues.input_min_ano);
                        ypoint = 300+(float)( 250 *  MODTOHAFSD_CalculateValues.d_z_km_arr[i]/
maxY);

                    }
                    else{

                        maxY = (float)Math.abs(MODTOHAFSD_CalculateValues.input_max_ano);
                        ypoint = 300-(float)( 250 * MODTOHAFSD_CalculateValues.d_z_km_arr[i] /
maxY);

                    }

                    g2.setFont(new Font("Arial", 20, 12));
                    g2.drawString("("+f.format(MODTOHAFSD_CalculateValues.d_z_km_arr[i])+")",
150+xpoint-8, ypoint-10);
                    g2.drawString("X", 150+xpoint-3, ypoint+3);
                }
            }

        };

        MODTOHAFSD_MainPanel.im2.addMouseListener(ml1);
        MODTOHAFSD_MainPanel.im2.addMouseMotionListener((MouseMotionListener)ml1);
    }
}
-----------------------------------------------------------------------------------------------

package com.modtohafsd.view.event;


import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;
import com.modtohafsd.view.graph.MODTOHAFSD_PlainGraph;


public class MODTOHAFSD_PlotReg extends Applet{

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public static double val[] =null;
    public static double val1[]=null;
    public static int pos=0;
    public static MouseListener ml;
    com.modtohafsd.model.MODTOHAFSD_CalculateValues cv = new
    com.modtohafsd.model.MODTOHAFSD_CalculateValues();
    MODTOHAFSD_PlainGraph pg = new MODTOHAFSD_PlainGraph();

    public void paint(Graphics g){

        MODTOHAFSD_MainPanel.p_Center.removeAll();
        MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.im);
        MODTOHAFSD_CalculateValues.len=0;
        val = new double[50] ;
        val1 = new double[50] ;
```

```
        Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im.getGraphics();
        g2.setFont(new Font("Arial", 40,12));
        g2.setColor(Color.red);
        MODTOHAFSD_MainPanel.im.setEditable(false);
        MODTOHAFSD_MainPanel.im.setBackground(Color.WHITE);
        MODTOHAFSD_MainPanel.p_Center.validate();

        for (int j = 0; j < val.length; j++){
            val[j] = 0;
            val1[j] = 0;
        }
        pos = 1;
        cv.getAnamolyValues(com.modtohafsd.view.MODTOHAFSD_MainPanel.captureValues());
        Graphics2D g1 = (Graphics2D)MODTOHAFSD_MainPanel.im.getGraphics();
        pg.drawPlainGraph(g1);

        ml = new MouseAdapter(){
            public void mouseClicked(MouseEvent e) {
                DecimalFormat f = new DecimalFormat("0.##");
                float polyx = e.getX();
                float polyy = e.getY();
                float x[] = new float[50];
                float z[] = new float[50];
                int get = pos;
                float maxAno;

                MODTOHAFSD_CalculateValues.len = get;
                Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im.getGraphics();
                if (polyy >= 50 && polyy <= 550 && polyx >= 150 && polyx <= 600){

                    MODTOHAFSD_PlotReg.val[get] = (float) Math.abs((MODTOHAFSD_PlainGraph.maxX1 *
(150 - polyx)) / 450);
                    if(polyy >300){
                        maxAno = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                        MODTOHAFSD_PlotReg.val1[get] = (float) (maxAno * (polyy - 300) / 250);
                    }
                    else{
                        maxAno = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                        MODTOHAFSD_PlotReg.val1[get] = (float) Math.abs(maxAno * (300 - polyy) /
250);
                    }

                    x[get] = polyx;
                    z[get] = polyy;
                    g2.setColor(Color.white);
                    g2.fillRect(730, 160, 250, 40);

                    g2.setColor(Color.BLACK);
                    g2.setFont(new Font("Arial", 40, 12));
                    g2.drawString("x", polyx, polyy);
                    g2.drawString("("+f.format(MODTOHAFSD_PlotReg.val1[get])+")", polyx, polyy);
                    g2.setFont(new Font("Arial", 40, 20));
                    if(get<=MODTOHAFSD_CalculateValues.i_d_poly)
                        g2.setColor(Color.red);
                    else
                        g2.setColor(Color.blue);

                    g2.drawString("Number of points "+get, 740, 180);
                    g2.setFont(new Font("Arial", 40, 12));
                    MODTOHAFSD_PlotReg.pos++;
                }
                pg.drawPlainGraph(g2);
                repaint();
            }
            public void mouseDragged(MouseEvent e){
                Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im.getGraphics();
                pg.drawPlainGraph(g2);
            }
            public void mouseMoved(MouseEvent e){

                DecimalFormat f = new DecimalFormat("0.##");
                float xer = e.getX();
                float yer = e.getY();
                float maxAno,ypoint ;
                if(yer>300){
```

```
                    maxAno = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                    ypoint = (float) (maxAno * (yer - 300) / 250);
                }
                else{
                    maxAno = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                    ypoint = (float) Math.abs(maxAno * (300 - yer) / 250);
                }

                float xpoint =(float) Math.abs((MODTOHAFSD_PlainGraph.maxX1 * (150 - xer)) / 450);

                if (yer >= 50 && yer <= 550 && xer >= 150 && xer <= 600){
                    Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im.getGraphics();
                    g2.setColor(Color.black);
                    g2.drawString("X-coordinates            Y-coordinates", 780, 320);
                    g2.setColor(Color.red);
                    g2.fillRect(780, 320, 180, 20);
                    g2.setColor(Color.black);
                    g2.drawString(""+f.format(xpoint)+"
"+f.format(ypoint)+"", 780, 330);
                }
                Graphics2D g2 = (Graphics2D)MODTOHAFSD_MainPanel.im.getGraphics();
                pg.drawPlainGraph(g2);
            }

        };

        MODTOHAFSD_MainPanel.im.addMouseListener(ml);
        MODTOHAFSD_MainPanel.im.addMouseMotionListener((MouseMotionListener)ml);
        g2.setFont(new Font("Arial", 40, 20));
        g2.setColor(Color.black);

    }

}
```

```
package com.modtohafsd.view.getvalues;

import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.MODTOHAFSD_HalfstrikeVal;
import com.modtohafsd.view.constant.MODTOHAFSD_GetConstHaf;

public class MODTOHAFSD_GetHalfstrikeVal {

    public void getVal(){

        for(int i=0;i< MODTOHAFSD_CalculateValues.input_n_obs;i++){
            for(int j=0;j<2;j++){

                MODTOHAFSD_HalfstrikeVal.haf[i][j]=MODTOHAFSD_HalfstrikeVal.table.getValueAt(i,
j);

            }
        }

        for(int i=1;i<=MODTOHAFSD_CalculateValues.input_n_obs;i++){

            String val1 = (String)MODTOHAFSD_HalfstrikeVal.haf[i-1][0];
            String val2 =   (String)MODTOHAFSD_HalfstrikeVal.haf[i-1][1];

            try{
                MODTOHAFSD_GetConstHaf.input_strike_km[i]=MODTOHAFSD_Utility.convertDouble(val2);

            }
            catch(Exception e){

            }


        }
    }
}
```

```
package com.modtohafsd.view.getvalues;

import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.MODTOHAFSD_OffsetVal;
import com.modtohafsd.view.constant.MODTOHAFSD_GetConstOff;

public class MODTOHAFSD_GetOffsetVal {

    public void getVal(){

        for(int i=0;i< MODTOHAFSD_CalculateValues.input_n_obs;i++){
            for(int j=0;j<2;j++){
                MODTOHAFSD_OffsetVal.off[i][j]=MODTOHAFSD_OffsetVal.table.getValueAt(i, j);
            }

        }

        for(int i=1;i<=MODTOHAFSD_CalculateValues.input_n_obs;i++){

            String val1 = (String)MODTOHAFSD_OffsetVal.off[i-1][0];
            String val2 =   (String)MODTOHAFSD_OffsetVal.off[i-1][1];

            try{
                MODTOHAFSD_GetConstOff.input_y_km[i]=MODTOHAFSD_Utility.convertDouble(val2);

            }
            catch(Exception e){

            }

        }
    }
}
-----------------------------------------------------------------------------------------------

package com.modtohafsd.view.graph;

import java.awt.*;
import java.awt.geom.Line2D;
import java.text.DecimalFormat;
import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.view.MODTOHAFSD_DrawGraph;
import com.modtohafsd.view.constant.MODTOHAFSD_GetConstReg;

public class MODTOHAFSD_PlainGraph {

    int i_no_obs=0;
    public static float maxX1=0;
    public static Object obj1[][]=null;
    public void drawPlainGraph(Graphics2D g2){
        g2.setColor(Color.RED);
        g2.setFont(new Font("Arial", 20, 12));
        g2.drawString("____:",625,70);
        g2.drawString("Bouguer anomalies",660,70);
        g2.setColor(Color.BLACK);
        g2.setFont(new Font("Arial", 20, 12));
        i_no_obs = MODTOHAFSD_CalculateValues.input_n_obs;
        double obser[] = new double[i_no_obs + 1];
        for (int i = 1; i <= i_no_obs; i++) {

            obser[i] = MODTOHAFSD_CalculateValues.x[i];
        }
        maxX1 = (float) MODTOHAFSD_CalculateValues.x[i_no_obs];
        float points = maxX1 / 5;
        DecimalFormat f = new DecimalFormat("0.#");
        g2.drawLine(150,50,150,300);
        g2.drawString("DISTANCE(km)", 315, 540);
        String []a={"A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
        for (int i = 0; i < a.length; i++) {

            g2.drawString(""+a[i], 80,  160 + ( i * 20 ) );
        }
        g2.drawString("|",(float) 598, 550);
        g2.drawString(""+f.format(MODTOHAFSD_CalculateValues.x[i_no_obs]),600, 570);
```

```
        int xInterval=90;
        for (int x = xInterval, j =1; x < 600; x+=xInterval)
        {
            if(j > 4)
                break;
            g2.drawString("|",150+x,550);
            g2.drawString("" + f.format(points*j), 147+x, 570);
            j++;
        }
        points = (float) (MODTOHAFSD_CalculateValues.input_max_ano / 5) ;
        g2.drawString(""+MODTOHAFSD_CalculateValues.input_max_ano, 115, 50);
        int yInterval = 50;
        for (int x = 50, j = 4; x < 250; x += yInterval){

            g2.drawString("-", 148, 55 + x);
            g2.drawString("" + (points * j), 115, 55 + x);
            j--;
        }
        g2.draw(new Line2D.Float(150, 550, 600, 550));
        g2.draw(new Line2D.Float(150, 300, 150, 550));
        float maxY = (float) MODTOHAFSD_CalculateValues.input_min_ano;
        g2.drawString("0", 135, 305);
        points = (float) (MODTOHAFSD_CalculateValues.input_min_ano / 5) ;
        g2.drawString(""+MODTOHAFSD_CalculateValues.input_min_ano, 115, 550);
        int yInter = 50;
        for (int x = yInter, j = 1; x < 250; x += yInter){
            if(j>4)
                break;
            g2.drawString("-", 148, 305 + x);
            g2.drawString("" + (points * j), 115, 305 + x);
            j++;
        }

        float xpoint = 0;
        float gypoint = 0;
        float prevx = 150;
        float prevy ;
        if(MODTOHAFSD_CalculateValues.input_nob_gob[1]<0){
            maxY = (float) MODTOHAFSD_CalculateValues.input_min_ano;
            prevy = 300+(float)( ( 250 * (MODTOHAFSD_CalculateValues.input_nob_gob[1]) / maxY ) );
        }
        else{
            maxY = (float) MODTOHAFSD_CalculateValues.input_max_ano;
            prevy = 300-(float)( ( 250 * (MODTOHAFSD_CalculateValues.input_nob_gob[1]) / maxY ) );
        }
        for (int k = 1; k <= i_no_obs; k++) {

            xpoint = (float)( 450 * obser[k] / maxX1);
            if(MODTOHAFSD_CalculateValues.input_nob_gob[k]<0){
                maxY = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                gypoint = 300+(float)( ( 250 * (MODTOHAFSD_CalculateValues.input_nob_gob[k]) /
maxY ) );
            }
            else{
                maxY = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                gypoint = 300-(float)( ( 250 * (MODTOHAFSD_CalculateValues.input_nob_gob[k]) /
maxY ) );
            }

            g2.setColor(Color.RED);
            g2.draw(new Line2D.Float(prevx, prevy, 150 + xpoint, gypoint  ));

            prevx = 150 + xpoint;
            prevy = gypoint ;
        }

    }

    public void newAno(Graphics2D g){

        g.setFont(new Font("Arial", 20, 12));
        g.setColor(Color.DARK_GRAY);
        g.drawString("____:",625,110);
        g.drawString("Residual anomalies",660,110);
        g.setColor(Color.GREEN);
```

```
        g.drawString("____:",625,90);
        g.drawString("Assumed/Estimated regional",660,90);
        float maxY,iniano ;
        double ano;
        obj1 = new Object[i_no_obs+1][4];
        float prevx = 150;
        float prevy;
        iniano = (float)(MODTOHAFSD_CalculateValues.input_nob_gob[1]-MODTOHAFSD_DrawGraph.fc[1]);
        if(iniano < 0){
            maxY = (float) MODTOHAFSD_CalculateValues.input_min_ano;
            prevy = (float)( 300 + ( 250 *(
            MODTOHAFSD_CalculateValues.input_nob_gob[1]-MODTOHAFSD_DrawGraph.fc[1]) / maxY )
            );//starting point for the y-axis
        }
        else{
            maxY = (float) MODTOHAFSD_CalculateValues.input_max_ano;
            prevy = (float)( 300 - ( 250 *(
            MODTOHAFSD_CalculateValues.input_nob_gob[1]-MODTOHAFSD_DrawGraph.fc[1]) / maxY ) );
        }
        float xpoint = 0;
        float ypoint = 0;


        for (int k = 1; k <= MODTOHAFSD_CalculateValues.input_n_obs; k++) {

            xpoint = (float)( 450 * MODTOHAFSD_CalculateValues.x[k] / maxX1);
            ano =
            (float)MODTOHAFSD_CalculateValues.input_nob_gob[k]-MODTOHAFSD_GetConstReg.input_reg_va
            l[k];
            MODTOHAFSD_CalculateValues.in_gob[k] = ano;
            if(ano<0){
                maxY = (float) MODTOHAFSD_CalculateValues.input_min_ano;
                ypoint = 300+(float)( ( 250 * (ano) / maxY ) );
            }
            else{
                maxY = (float) MODTOHAFSD_CalculateValues.input_max_ano;
                ypoint = 300-(float)( ( 250 * (ano) / maxY ) );
            }

            g.setFont(new Font("Arial", 20, 20));
            g.setColor(Color.DARK_GRAY);
            g.draw(new Line2D.Float(prevx, prevy, 150 + xpoint,  ypoint  ));

            prevx = 150 + xpoint;
            prevy =  ypoint ;
            DecimalFormat df =new DecimalFormat("0.###");
            obj1[k][0]= "" + MODTOHAFSD_CalculateValues.x[k];
            obj1[k][1]= "" + df.format(MODTOHAFSD_CalculateValues.input_nob_gob[k]);
            obj1[k][2]= "" + df.format(MODTOHAFSD_GetConstReg.input_reg_val[k]);
            obj1[k][3]= "" + df.format(ano);
        }
    }
}
```

```
package com.modtohafsd.model;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileOutputStream;
import java.text.DecimalFormat;
import java.util.HashMap;
import javax.imageio.ImageIO;
import com.modtohafsd.util.MODTOHAFSD_HandleException;
import com.modtohafsd.util.MODTOHAFSD_Utility;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;
import com.modtohafsd.view.MODTOHAFSD_TableView;
import com.modtohafsd.view.constant.*;
import com.modtohafsd.view.event.MODTOHAFSD_PlotReg;


public class MODTOHAFSD_CalculateValues {
```

```
public static int input_n_obs,len,i_d_poly,input_ite_num,input_lambda_st = 0;
public static double input_ddx_km,input_min_ano,input_max_ano=0;
public double input_lambda_val[]=null;
public static double
input_sd_val[],input_nob_gob[],input_reg_val[],x[],input_ele_km[],o_dep[]= null;
public static double reg[],err[],err1[],input_strike_km
[],input_y_km[],d_cftnt_arr[],vsd[],dep[],o_funct[]=null;
public static Object obj[][] = null;
public static double d_x_km_arr[], d_z_km_arr[],input_zmax_km = 0;
public double input_zmin_km=0;
public static double input_al_err=0;
public static double []o_gc ,in_gob=null;
static BufferedImage image;
public static int o_iter,count=0;
public static String input_area_name="";
public static int np=0;
double  PI = 22.0 / 7.0;
double  GK = 13.3333;
public void getAnamolyValues(HashMap h_Map) {

    try {
        input_n_obs = MODTOHAFSD_Utility.convertInteger((String)h_Map.get("N_OBS"));
        input_ddx_km = MODTOHAFSD_Utility.convertDouble((String)h_Map.get("X_KM"));
        input_ele_km = MODTOHAFSD_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
        input_nob_gob = MODTOHAFSD_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
        input_min_ano = MODTOHAFSD_Utility.convertDouble((String)h_Map.get("MIN_ANO"));
        input_max_ano =  MODTOHAFSD_Utility.convertDouble((String)h_Map.get("MAX_ANO"));
        input_al_err =  MODTOHAFSD_Utility.convertDouble((String)h_Map.get("AL_ERR"));
        input_zmax_km =  MODTOHAFSD_Utility.convertDouble((String)h_Map.get("DEP_ZMAX"));
        input_area_name = MODTOHAFSD_Utility.convertString((String)h_Map.get("AREA_FE"));
        input_ite_num = MODTOHAFSD_Utility.convertInteger((String)h_Map.get("ITE_RAT"));
    }
    catch(Exception e) {

    }

    x = new double[input_n_obs+1];
    in_gob = new double[input_n_obs+1];
    for (int j = 1; j <= input_n_obs; j++){
        x[j]=0;
        in_gob[j]=0;
    }
    input_y_km = MODTOHAFSD_GetConstOff.input_y_km;
    input_strike_km = MODTOHAFSD_GetConstHaf.input_strike_km;
    input_reg_val = MODTOHAFSD_GetConstReg.input_reg_val;
    i_d_poly = MODTOHAFSD_GetDegPoly.input_deg_poly;
    d_cftnt_arr = new double[i_d_poly + 2];
    input_lambda_st = 1;
    for (int  KL = 1; KL <= input_n_obs; KL++) {
        x[KL] = (KL - 1) * input_ddx_km;
    }
}

public void cal(){

    double funct1 = 0.0;
    double  ALER = input_al_err;
    o_gc  = new double[input_n_obs + 1];
    o_dep = new double[input_n_obs + 1];
    input_lambda_val = new double[input_n_obs + 1];
    input_sd_val = new double[input_n_obs + 1];
    o_funct  = new double[input_ite_num +5];
    err = new double[input_n_obs + 1];
    err1 = new double[input_n_obs + 1];
    for (int i = 0; i <= input_n_obs; i++){
        o_gc[i]=0;
        o_dep[i]=0;
        input_lambda_val[i]=0;
        input_sd_val[i]=0;
        err[i]=0;
        err1[i]=0;
    }

    for (int j = 1; j <= input_n_obs; j++){
```

```
            input_lambda_val[j] = MODTOHAFSD_LambInt.lamval;
            input_sd_val[j] =  MODTOHAFSD_LambInt.sdval;
        }

        double dcz[] = new double[input_n_obs + 1];
        for (int KL = 1; KL <= input_n_obs; KL++) {
            in_gob[KL] = input_nob_gob[KL] - input_reg_val[KL];
        }
        double  b = (x[2] - x[1]) / 2.0;

        for (int KK = 1 ; KK <= input_n_obs; KK++) {
            o_dep[KK] = (1 / input_lambda_val[KK]) * Math.log(1 + ((input_lambda_val[KK] *
            in_gob[KK]) / (GK * PI * input_sd_val[KK])));
        }
        GF(input_n_obs,x,input_ele_km,o_dep,input_sd_val,input_lambda_val,input_strike_km,input_y_
        km,input_ddx_km,b,o_gc);

        for(int K = 1 ; K <= input_n_obs; K++) {
            err[K] = in_gob[K] - o_gc[K];
            funct1 = funct1 + Math.pow((in_gob[K] - o_gc[K]), 2);
        }

        int iter  = 0;
        double funct2 = 0.0;

        while(iter<input_ite_num)     {
            iter = iter + 1;

            for (int ll = 1; ll <= input_n_obs; ll++){
                err1[ll] = in_gob[ll] - o_gc[ll];
            }
            for (int KK = 1; KK <= input_n_obs; KK++) {
                err[KK] = in_gob[KK] - o_gc[KK];
                dcz[KK] = (input_sd_val[KK] * Math.exp(input_lambda_val[KK] * o_dep[KK]));
                o_dep[KK] = o_dep[KK] + ((err[KK] / (GK * PI * dcz[KK])));
                if(o_dep[KK] <= input_zmin_km)
                    o_dep[KK] = input_zmin_km;

                if(o_dep[KK] > input_zmax_km )
                    o_dep[KK] = input_zmax_km;
            }
            GF(input_n_obs, x, input_ele_km,o_dep, input_sd_val, input_lambda_val,
            input_strike_km, input_y_km, input_ddx_km, b, o_gc);
            funct2 = 0.0;
            for (int LI = 1 ; LI <= input_n_obs ; LI++) {
                funct2 = funct2 + Math.pow(in_gob[LI] - o_gc[LI], 2);
            }
            o_iter = iter;
            o_funct[iter] = funct1;

            setGraphValues(input_n_obs, iter, x, input_nob_gob, in_gob, o_gc, o_dep, funct1, err1,
            input_area_name);
            drawGraph();

            if (funct2 - funct1 <= 0 ) {
                if (funct2 - ALER <= 0) {
                    break;
                }
                else  {
                    funct1 = funct2;
                }
            }
            else
                break;

        }
    }

    public void invcal(){

        double dpar = 0.1;
        double funct1 = 0.0;
        double  ALER = input_al_err;
        o_gc  = new double[input_n_obs + 1];
        o_dep = new double[input_n_obs + 1];
```

```
        input_lambda_val = new double[input_n_obs + 1];
        input_sd_val = new double[input_n_obs + 1];
        o_funct  = new double[input_ite_num +5];
        err = new double[input_n_obs + 1];
        err1 = new double[input_n_obs + 1];
        for (int i = 0; i <= input_n_obs; i++){
            o_gc[i]=0;
            o_dep[i]=0;
            input_lambda_val[i]=0;
            input_sd_val[i]=0;
            err[i]=0;
            err1[i]=0;
        }
        for (int j = 1; j <= input_n_obs; j++){
            input_lambda_val[j] = MODTOHAFSD_LambInt.lamval;
            input_sd_val[j] =  MODTOHAFSD_LambInt.sdval;
        }

        double []par = new double[input_n_obs + 1];
        double []par1 = new double[input_n_obs + 1];
        double []par2 = new double[input_n_obs + 1];
        double []dupar = new double[input_n_obs + 1];
        double []g1 = new double[input_n_obs + 1];
        double []g2 = new double[input_n_obs + 1];
        double [][]st = new double[input_n_obs + 1][input_n_obs + 1];
        double []actz = new double[input_n_obs + 1];
        double []b = new double[input_n_obs + 1];
        double []KS = new double[2];
        double LAMBDA = 0.5;
        int np1 = input_n_obs + 1;
        double [][]p = new double[input_n_obs+1][np1+1];
        for (int KL = 1; KL <= input_n_obs; KL++) {
            in_gob[KL] = input_nob_gob[KL]-input_reg_val[KL];
        }
        double  t = (x[2] - x[1]) / 2.0;

        for (int KK = 1 ; KK <= input_n_obs; KK++) {
            par[KK] = (1 / input_lambda_val[KK]) * Math.log(1 + ((input_lambda_val[KK] *
            in_gob[KK]) / (GK * PI * input_sd_val[KK])));
        }
        GF(input_n_obs,x,input_ele_km,par,input_sd_val,input_lambda_val,input_strike_km,input_y_km
        ,input_ddx_km,t,o_gc);
        funct1=0.0;
        for(int K = 1 ; K <= input_n_obs; K++) {
            err[K] = in_gob[K] - o_gc[K];
            funct1 = funct1 + Math.pow((in_gob[K] - o_gc[K]), 2);
        }

        int iter  = 0;
        double funct2 = 0.0;
        while(iter< input_ite_num)     {
            iter = iter + 1;
            for(int  KJ = 1; KJ <= input_n_obs; KJ++) {
                actz[KJ] = par[KJ];
            }

            for(int K = 1;K <= input_n_obs; K++) {
                par1[K] = par[K];
            }
            for (int  I = 1; I <= input_n_obs; I++) {
                par1[I] = par[I] + dpar/2.0;
                GF(input_n_obs,x,input_ele_km,par1,input_sd_val,input_lambda_val,input_strike_km,i
nput_y_km,input_ddx_km,t,g1);
                par1[I] = par[I] - dpar/2.0;
                GF(input_n_obs,x,input_ele_km,par1,input_sd_val,input_lambda_val,input_strike_km,i
nput_y_km,input_ddx_km,t,g2);
                for(int K=1;K<=input_n_obs;K++) {
                    st[I][K] = (g1[K] - g2[K]) /dpar;
                }
            }
            for (int  J=1; J <= np1; J++) {
                for (int I=1; I <= input_n_obs;I++) {
                    p[I][J]=0.0;
                }
            }
```

```
            for(int J = 1;J <= input_n_obs;J++)  {
                for(int I = 1;I <= input_n_obs;I++) {
                    for(int K = 1;K <= input_n_obs;K++) {
                        p[I][J] = p[I][J] + st[I][K] * st[J][K];
                    }
                }
            }
            for (int J = 1; J <= input_n_obs;J++) {
                for (int K = 1; K <= input_n_obs;K++) {
                    p[J][np1] = p[J][np1] + err[K] * st[J][K];
                }
            }

            do {
                double  CON = LAMBDA + 1.0;
                for(int I = 1; I <= input_n_obs;I++) {
                    dupar[I] = par[I];
                }
                for (int L = 1;L <= input_n_obs; L++) {

                    for (int J = 1;J <= input_n_obs; J++) {

                        if (L - J == 0) {
                            p[L][J] = p[L][J] * CON;
                        }
                    }
                }
                SIMEQ(p,b,input_n_obs,KS);
                for (int I=1;  I <= input_n_obs; I++) {
                    par2[I] = dupar[I] + b[I];
                }
                for (int KK = 1; KK <= input_n_obs; KK++) {
                    if(par2[KK] <= input_zmin_km)
                        par2[KK] = input_zmin_km;
                    if(par2[KK] > input_zmax_km)
                        par2[KK] = input_zmax_km;
                }

                GF(input_n_obs,x,input_ele_km,par2,input_sd_val,input_lambda_val,input_strike_km,i
nput_y_km,input_ddx_km,t,o_gc);
                funct2 = 0.0;
                for (int K = 1; K <= input_n_obs; K++) {
                    err[K] = in_gob[K] - o_gc[K];
                    funct2 = funct2 + Math.pow(err[K],2);
                }
                if (funct1 - funct2 < 0) {

                    LAMBDA = LAMBDA * 2;

                    if (LAMBDA - 13 < 0) {

                        for (int I = 1; I <= input_n_obs; I++) {

                            for (int J = 1; J <= input_n_obs; J++) {

                                if(I - J == 0)
                                    p[I][J] = p[I][J] / CON;
                            }
                        }
                    }
                    else
                        break;
                }

            }while (funct1 <= funct2);

            o_iter = iter;
            o_funct[iter] = funct1;
            funct1 = funct2;

            for(int I = 1; I <= input_n_obs; I++) {
                par[I] = par2[I];
            }
            for(int I = 1; I <= input_n_obs; I++) {
                o_dep[I] = par2[I];
```

```
            }
            LAMBDA = LAMBDA / 2.0;

            setGraphValues(input_n_obs, iter, x, input_nob_gob, in_gob, o_gc, o_dep,  funct1, err,
            input_area_name);
            drawGraph();

            if (funct2 - funct1 <= 0 ) {
                if (funct2 - ALER <= 0) {
                    break;
                }
                else  {
                    funct1 = funct2;
                }
            }
            else
                break;

        }
    }

    public double [] GF(int n,double []x,double ele[],double []z,double []sd,double []la,double
    []yy,double []ydist,double ddx,double t,double []gc) {

        double xx = 0,z2 = 0,y1 = 0,gff,ggc=0;
        double []effy = new double [3];
        double []gg2 = new double [3];
        double z1 = 0.0;
        double GK = 13.3333;
        for(int JJ = 1 ; JJ <= n ;JJ++){
            gc[JJ]=0.0;
        }
        for(int II = 1 ; II <= n ; II++){
            for(int LL = 1 ; LL <= n; LL++) {
                xx = x[II] - x[LL];
                z2 = z[LL];
                effy[1] = yy[LL] - ydist[LL];
                effy[2] = yy[LL] + ydist[LL];
                double dx = ddx/10;
                double zb = z2 - z1;
                double dz = 0,dc = 0;
                double []zk = new double[1000];
                double []gs = new double[1000];

                int nd = (int)(zb / dx) + 1;
                int n1 = nd / 2;
                if (nd - (2 * n1 ) < 0 || nd - ( 2 * n1 ) > 0) {
                    nd = nd + 1;
                }
                dz = zb / nd;
                int n2 = nd + 1;
                for (int JZ = 1 ; JZ <= n2; JZ++) {
                    zk[JZ] = z1 + dz * (JZ - 1);
                }
                for (int JZ = 1; JZ <= n2; JZ++) {
                    dc = (sd[LL] * Math.exp(la[LL] * zk[JZ]));
                    for (int LK = 1 ; LK <= 2; LK++) {
                        y1 = effy[LK];
                        double t1 = GK * dc;
                        double t2 = Math.atan((y1 * (xx + t)) / (Math.sqrt(Math.pow((xx + t), 2) +
Math.pow(y1, 2) + Math.pow(zk[JZ]-ele[II], 2)) * (zk[JZ]-ele[II])));
                        double t3 = Math.atan((y1 * (xx - t)) / (Math.sqrt(Math.pow((xx - t), 2) +
Math.pow(y1, 2) + Math.pow(zk[JZ]-ele[II], 2))* (zk[JZ]-ele[II])));
                        gg2[LK] = t1 * (t2 - t3);
                    }
                    gs[JZ] = (gg2[1] + gg2[2]) / 2.0;
                }
                gff = SIMP(gs,zk,n2,ggc);
                gc[II] = gc[II] + gff;

            }
        }
        return gc;
    }
```

```
public double SIMP(double []gs,double []z,int n,double ggc) {

    double dz = z[2]-z[1];
    double sum1 = 0.0;
    double sum2 = 0.0;
    int n1 = n / 2;
    int n4 = n1 - 1;
    for(int I = 1; I <= n1; I++) {
        int n2 = 2 * I;
        sum1 = sum1 + gs[n2];
    }
    for(int I = 1; I <= n4; I++) {
        int n3 = 2 * I +1;
        sum2 = sum2 + gs[n3];
    }
    ggc = gs[1]+4*sum1+2*sum2+gs[n];
    ggc = ggc * dz / 3.0;
    return ggc;
}


public static double []SIMEQ(double p[][], double b[], int n, double KS[]) {

    int I = n + 1;
    double []a = new double[n*n+1];

    for (int I1 = 1; I1 <= n; I1++) {

        for (int I2 = 1; I2 <= n; I2++) {
            int I3 = (I1 - 1) * n + I2;
            a[I3] = p[I2][I1];
        }
    }

    for (int I4 = 1;I4 <= n; I4++) {
        b[I4] = p[I4][I];
    }
    double tol = 0;
    KS[0] = 0;
    int JJ = - n;
    int IT;
    int NY = 0;
    for (int J = 1;J <= n; J++) {
        int JY = J + 1;
        JJ = JJ + n + 1;
        double biga = 0;
        IT = JJ - J;
        int imax = 0;
        for (int i = J; i <= n; i++) {
            int IJ = IT + i;
            if (Math.abs(biga) - Math.abs(a[IJ]) < 0) {

                biga = a[IJ];
                imax = i;
            }
        }
        int I1 = 0;
        if (Math.abs(biga) - tol <= 0) {
            KS[1] = 1;
            return KS;
        }
        else {
            I1 = J + n * (J - 2);
            IT = imax - J;
        }

        double save;
        for (int K = J;K <= n; K++) {
            I1 = I1 + n;
            int I2 = I1 + IT;
            save = a[I1];
            a[I1] = a[I2];
            a[I2] = save;
            a[I1] = a[I1] / biga;
        }
```

```
            save = b[imax];
            b[imax] = b[J];
            b[J] = save / biga;
            int IQS = 0;

            if (J - n < 0 || J - n > 0) {

                IQS = n * (J - 1);
                for (int IX = JY;IX <= n; IX++) {
                    int IXJ = IQS + IX;
                    IT = J - IX;
                    for (int JX = JY;JX <= n; JX++) {
                        int IXJX = n * (JX - 1) + IX;
                        int JJX = IXJX + IT;
                        a[IXJX] = a[IXJX] - (a[IXJ] * a[JJX]);
                    }
                    b[IX] = b[IX] - (b[J] * a[IXJ]);
                }
            }
        }
        NY = n - 1;
        IT = n * n;

        for (int J = 1;J <= NY; J++) {
            int ia = IT - J;
            int ib = n - J;
            int ic = n;
            for (int K = 1;K <= J; K++) {
                b[ib] = b[ib] - a[ia] * b[ic];
                ia = ia - n;
                ic = ic - 1;
            }
        }
        return b;
    }

    public void getCoefficients() throws MODTOHAFSD_HandleException {

        d_x_km_arr = new double[len + 1];
        d_z_km_arr = new double[len + 1];
        for (int i = 0; i <= len; i++){
            if ( i == 0){
                d_x_km_arr[i] = 0;
                d_z_km_arr[i] = 0;
            }
            else{

                d_x_km_arr[i] = MODTOHAFSD_PlotReg.val[i];
                d_z_km_arr[i] = (MODTOHAFSD_PlotReg.val1[i]);

            }
        }

        double d_coeff_values[] = new double[i_d_poly + 1];
        double d_coeff_xzval_arr[] = new double[i_d_poly + 1];
        double d_coeff_zval_arr[][] = new double[i_d_poly + 1][i_d_poly + 1] ;
        double d_sum_lmatrix = 0;
        double d_sum_rmatrix = 0;

        for (int l = 0; l < i_d_poly + 1; l++){
            d_sum_lmatrix = 0;
            for(int i = 0; i < d_x_km_arr.length; i++){
                if (l == 0){
                    d_sum_lmatrix = d_sum_lmatrix + d_z_km_arr[i];
                }
                else{
                    double d_sum_xz = d_z_km_arr[i] * Math.pow(d_x_km_arr[i], l);
                    d_sum_lmatrix = d_sum_lmatrix+d_sum_xz;
                }
            }
            d_coeff_xzval_arr[l] = d_sum_lmatrix;
        }

        double d_rmatrix_arr[] = new double[2 * i_d_poly + 1];
        for (int l = 0; l < d_rmatrix_arr.length; l++){
```

```
            d_sum_rmatrix = 0;
            for (int i = 0; i < d_x_km_arr.length; i++){
                if (l == 0){
                    d_sum_rmatrix = d_sum_rmatrix + d_x_km_arr[i];
                    d_rmatrix_arr[l] = d_sum_rmatrix;
                }
                else{
                    double d_sum_zpow = Math.pow(d_x_km_arr[i], l);
                    d_sum_rmatrix = d_sum_rmatrix + d_sum_zpow;
                    d_rmatrix_arr[l] = d_sum_rmatrix;
                }
            }
        }
        for (int i = 0; i < i_d_poly + 1; i++){

            for (int j = 0; j < i_d_poly + 1; j++){
                if ( i == 0 && j == 0){
                    d_coeff_zval_arr[i][j] = d_x_km_arr.length - 1;
                }
                else{
                    d_coeff_zval_arr[i][j] = d_rmatrix_arr[i + j];
                }
            }
        }

        int index[] = new int[i_d_poly + 1];
        d_coeff_values = solve(d_coeff_zval_arr, d_coeff_xzval_arr, index);

        //d_cftnt_arr = new double[i_d_poly + 2];
        d_cftnt_arr[0] = 0.0;

        for (int i = 1; i <= i_d_poly + 1; i++){
            if(d_coeff_values[i - 1]<=0||d_coeff_values[i - 1]>0)
                d_cftnt_arr[i] = d_coeff_values[i - 1];
            else
                d_cftnt_arr[i] = MODTOHAFSD_GetConstReg.regval;
        }
    }

    public  double[] solve(double a[][],double b[], int index[]) {

        int n = b.length;
        double x[] = new double[n];
        gaussian(a, index);

        for (int i = 0; i < n - 1; ++i) {
            for (int j = i + 1; j < n; ++j) {
                b[index[j]] -= a[index[j]][i] * b[index[i]];
            }
        }
        x[n - 1] = b[index[n - 1]] / a[index[n - 1]][n - 1];
        for (int i = n - 2; i >= 0; --i) {
            x[i] = b[index[i]];
            for (int j = i + 1; j < n; ++j) {
                x[i] -= a[index[i]][j] * x[j];
            }
            x[i] /= a[index[i]][i];
        }
        return x;
    }

    public  void gaussian(double a[][],int index[]) {

        int n = index.length;
        double c[] = new double[n];

        for (int i = 0; i < n; ++i) index[i] = i;
        for (int i = 0; i < n; ++i) {
            double c1 = 0;
            for (int j = 0; j < n; ++j) {
                double c0 = Math.abs(a[i][j]);
                if (c0 > c1) c1 = c0;
            }
            c[i] = c1;
        }
```

```
    int k = 0;
    for (int j = 0; j < n - 1; ++j) {
        double pi1 = 0;
        for (int i = j; i < n; ++i) {
            double pi0 = Math.abs(a[index[i]][j]);
            pi0 /= c[index[i]];
            if (pi0 > pi1) {
                pi1 = pi0;
                k = i;
            }
        }
        int itmp = index[j];
        index[j] = index[k];
        index[k] = itmp;
        for (int i = j + 1; i < n; ++i) {
            double pj = a[index[i]][j] / a[index[j]][j];
            a[index[i]][j] = pj;
            for (int l=j+1; l<n; ++l)
                a[index[i]][l] -= pj*a[index[j]][l];

        }
    }
}

public static void denCal(int n){

    int i = 1;
    double Z1 = 0;
    double []depth = new double[input_n_obs+1];
    for (int k = 1; k <= input_n_obs; k++){
        depth[k] = o_dep[k];
    }
    double Z2 = MODTOHAFSD_Utility .findMaximumNumber1(depth);
    vsd = new double[(int) Math.pow(input_n_obs,2)];
    dep = new double[(int) Math.pow(input_n_obs,2)];
    while(Z1 <= Z2){
        double dc = MODTOHAFSD_LambInt.sdval*Math.exp(MODTOHAFSD_LambInt.lamval*Z1);
        vsd[i] = dc;
        dep[i] = Z1;
        Z1 = Z1 + 0.1;
        i++;
    }
    count = i;
    vsd[count] = MODTOHAFSD_LambInt.sdval*Math.exp(MODTOHAFSD_LambInt.lamval*Z2);
    dep[count] = Z2;

}

public static void  setGraphValues(int i_no_obs,int ite,double []dis,double []GOBS,double
[]Boug,double []GCAL,double []Dep, double funct,double []Err,String Area_fe) {

    obj = new Object[i_no_obs+1][7];
    DecimalFormat df1 = new DecimalFormat("0.####");
    DecimalFormat d = new DecimalFormat("0.##");
    DecimalFormat df =new DecimalFormat("0.###");
    for (int K = 1; K <= i_no_obs; K++){

        obj[K][0] = "" + dis[K];
        obj[K][1] = "" + df.format(GOBS[K]);
        obj[K][2] = "" + df.format(input_reg_val[K]);
        obj[K][3] = "" + df.format(Boug[K]);
        obj[K][4] = "" + df.format(GCAL[K]);
        obj[K][5] = "" + df.format(Err[K]);
        obj[K][6] = "" + df.format(Dep[K]);
    }

    obj[0][0] ="ITERATION";
    obj[0][1] = "=" +" "+ite;
    MODTOHAFSD_TableView.val.setText("");

    MODTOHAFSD_TableView.val.appendText("\n");
    MODTOHAFSD_TableView.val.append("Polynomial coefficients of regional background:-\n");
    MODTOHAFSD_TableView.val.appendText("-----------------------------------------\n");
    MODTOHAFSD_TableView.val.appendText("\n");
```

```
        for (int i = 1; i <= i_d_poly+1; i++){
            MODTOHAFSD_TableView.val.appendText("f"+(i - 1)+" = " +df1.format(d_cftnt_arr[i])+
            "\n");
        }

        MODTOHAFSD_TableView.val.appendText("\n");

    }

    public static void drawGraph(){

        final com.modtohafsd.view.MODTOHAFSD_DrawGraph dg = new
        com.modtohafsd.view.MODTOHAFSD_DrawGraph();

        try {
            int width = 1050;
            int height = 650;
            BufferedImage buffer = new BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);

            Graphics g1 = buffer.createGraphics();
            g1.setColor(Color.WHITE);
            g1.fillRect(0,0,width,height);
            Graphics2D g2 = (Graphics2D)g1 ;
            dg.plot(g2);
            dg.drawGraph(g2);
            float maxEle = (float)
            MODTOHAFSD_Utility.findMaximumNumber1(MODTOHAFSD_CalculateValues.input_ele_km);
            if(maxEle==0)
                dg.plotZCoordinates(g2);
            else
                dg.plotZCoordinatesele(g2);
            dg.plotXYCoordinates(g2);
            dg.plotXPoly(g2,120,200,50);
            dg.plot(g2);
            dg.drawOBJ(g2);
            int i=1;
            Color c = Color.MAGENTA;
            denCal(i);
            dg.drawSd(g2,c,i);
            dg.idex(g2);

            FileOutputStream os = new FileOutputStream( MODTOHAFSD_CalculateValues.input_area_name
            +".jpg");
            ImageIO.write(buffer, "jpg", os);
            os.close();

            String path =  MODTOHAFSD_CalculateValues.input_area_name  +".jpg";
            image = ImageIO.read(new File(path));

            Graphics g_image = MODTOHAFSD_MainPanel.img.getGraphics();
            g_image.drawImage(image, -60,-30,image.getWidth(), image.getHeight(),dg);

            MouseListener ml3 = new MouseAdapter(){
                public void mouseClicked(MouseEvent e){
                    Graphics g_image = MODTOHAFSD_MainPanel.img.getGraphics();
                    g_image.drawImage(image, -60,-30,image.getWidth(), image.getHeight(),dg);
                }
            };
            MODTOHAFSD_MainPanel.img.addMouseListener(ml3);
        }
        catch (Exception e2) {

            e2.printStackTrace();
        }
    }
}
```

```
package com.modtohafsd.control;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.text.DecimalFormat;
```

```
import javax.swing.*;

import com.modtohafsd.model.MODTOHAFSD_CalculateValues;
import com.modtohafsd.util.MODTOHAFSD_HandleException;
import com.modtohafsd.view.*;
import com.modtohafsd.view.constant.*;
import com.modtohafsd.view.event.*;
import com.modtohafsd.view.graph.MODTOHAFSD_PlainGraph;

public class MODTOHAFSD_Controller implements ActionListener{

    com.modtohafsd.model.MODTOHAFSD_CalculateValues cv = new
    com.modtohafsd.model.MODTOHAFSD_CalculateValues();
    Object rowdata[][]={};
    public static boolean success = false;

    public void actionPerformed(ActionEvent ae) {

        if(ae.getActionCommand().equals("Variable regional")) {

            cv.getAnamolyValues(com.modtohafsd.view.MODTOHAFSD_MainPanel.captureValues());
            MODTOHAFSD_MainPanel.im.removeMouseListener(MODTOHAFSD_PlotReg.ml);
            MODTOHAFSD_MainPanel.im.removeMouseMotionListener((MouseMotionListener)MODTOHAFSD_Plot
            Reg.ml);
            MODTOHAFSD_GetDegPoly gdp = new MODTOHAFSD_GetDegPoly();
            gdp.constHaf();

        } else if(ae.getActionCommand().equals("Draw/Edit polynomial")){
            cv.getAnamolyValues(com.modtohafsd.view.MODTOHAFSD_MainPanel.captureValues());
            if(MODTOHAFSD_CalculateValues.i_d_poly!=0){
                try{
                    if (MODTOHAFSD_CalculateValues.len <= MODTOHAFSD_CalculateValues.i_d_poly){
                        JFrame frame = null;
                        JOptionPane.showMessageDialog(frame,
                                "The polynomial fit requires\n"
                                +"degree+1 data points.\n"
                                +"Use additional data points",
                                "Error!",
                                JOptionPane.ERROR_MESSAGE);
                        throw new MODTOHAFSD_HandleException();
                    }
                    cv.getCoefficients();
                    Graphics g1 =  MODTOHAFSD_MainPanel.img.getGraphics();
                    MODTOHAFSD_EditReg er = new MODTOHAFSD_EditReg();

                    er.paint(g1);
                    com.modtohafsd.view.MODTOHAFSD_TableView.populateEastPanel1(MODTOHAFSD_PlainGr
aph.obj1);
                }
                catch(Exception e){
                    e.printStackTrace();
                }
            }

        }else if(ae.getActionCommand().equals("Surface density & Lambda")){

            MODTOHAFSD_TableView.populateEastPanel(rowdata);
            cv.getAnamolyValues(MODTOHAFSD_MainPanel.captureValues());
            DecimalFormat df = new DecimalFormat("0.####");
            MODTOHAFSD_LambInt li = new MODTOHAFSD_LambInt();
            String lamval = df.format(MODTOHAFSD_LambInt.lamval);
            MODTOHAFSD_LambInt.lam.setText(lamval);
            String sdval = df.format(MODTOHAFSD_LambInt.sdval);
            MODTOHAFSD_LambInt.sd.setText(sdval);
            li.constLamSd();
        }else if(ae.getActionCommand().equals("Modeling")) {

            MODTOHAFSD_MainPanel.p_Center.removeAll();
            MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);
            MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.img);
            MODTOHAFSD_MainPanel.img.setEditable(false);
            MODTOHAFSD_MainPanel.img.setBackground(Color.WHITE);
            MODTOHAFSD_MainPanel.p_Center.validate();
            MODTOHAFSD_MainPanel.p_East.removeAll();
            MODTOHAFSD_MainPanel.p_East.validate();
```

```
            MODTOHAFSD_TableView.populateEastPanel(rowdata);
            MODTOHAFSD_MainPanel.p_East.validate();
            com.modtohafsd.view.MODTOHAFSD_MainPanel.clearPanel(MODTOHAFSD_MainPanel.img);
            cv.getAnamolyValues(com.modtohafsd.view.MODTOHAFSD_MainPanel.captureValues());
            cv.cal();
            com.modtohafsd.view.MODTOHAFSD_TableView.populateEastPanel(MODTOHAFSD_CalculateValues.
            obj);
            com.modtohafsd.view.MODTOHAFSD_MainPanel.p_East.repaint();
            com.modtohafsd.view.MODTOHAFSD_MainView mv = new
            com.modtohafsd.view.MODTOHAFSD_MainView();

            mv.setResizable(true);

        }else if(ae.getActionCommand().equals("Inversion")) {

            MODTOHAFSD_CalculateValues.obj = null;
            MODTOHAFSD_TableView.populateEastPanel(rowdata);
            MODTOHAFSD_MainPanel.p_Center.removeAll();
            MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.graphLabel);
            MODTOHAFSD_MainPanel.p_Center.add(MODTOHAFSD_MainPanel.img);
            MODTOHAFSD_MainPanel.img.setEditable(false);
            MODTOHAFSD_MainPanel.img.setBackground(Color.WHITE);
            MODTOHAFSD_MainPanel.p_Center.validate();
            MODTOHAFSD_MainPanel.p_East.removeAll();
            MODTOHAFSD_MainPanel.p_East.validate();
            MODTOHAFSD_TableView.populateEastPanel(rowdata);
            MODTOHAFSD_MainPanel.p_East.validate();
            com.modtohafsd.view.MODTOHAFSD_MainPanel.clearPanel(MODTOHAFSD_MainPanel.img);
            cv.getAnamolyValues(com.modtohafsd.view.MODTOHAFSD_MainPanel.captureValues());
            cv.invcal();
            com.modtohafsd.view.MODTOHAFSD_TableView.populateEastPanel(MODTOHAFSD_CalculateValues.
            obj);
            com.modtohafsd.view.MODTOHAFSD_MainPanel.p_East.repaint();
            com.modtohafsd.view.MODTOHAFSD_MainView mv = new
            com.modtohafsd.view.MODTOHAFSD_MainView();

            mv.setResizable(true);

        }else if(ae.getActionCommand().equals("Save and Print")){

            try{
                String current = System.getProperty("user.dir");
                File img_file = new File(MODTOHAFSD_CalculateValues.input_area_name+".jpg");
                JFileChooser saveFile = new JFileChooser(current);
                File OutFile = saveFile.getSelectedFile();
                FileWriter myWriter = null;
                if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION){

                    OutFile = saveFile.getSelectedFile();

                    if (OutFile.canWrite() || !OutFile.exists()){

                        File dir = new File(OutFile.getParent());
                        success = img_file.renameTo(new File(dir,img_file.getName()));
                        System.out.println("Save successful"+success);
                        myWriter = new FileWriter(OutFile+".html");
                        myWriter.write(" </table> </td> <td> <img src = '"+
MODTOHAFSD_CalculateValues.input_area_name +".jpg'></td></tr></table>");
                        myWriter.write("<html> <Body onLoad = \"window.print()\"><table> <tr>
<td>" +
                                "<table border = 1> <tr> <th colspan = 4>LOCATION:-
"+MODTOHAFSD_CalculateValues.input_area_name+"</th> </tr>");
                        DecimalFormat df =new DecimalFormat("0.###");
                        DecimalFormat d = new DecimalFormat("0.##");
                        //myWriter.write(" <tr><th colspan = 4> PROFILE NUMBER:-"+"
"+MODTOHAFSD_CalculateValues.input_profile_num+" </th></tr>");
                        myWriter.write(" <tr><th colspan = 4> ITERATION NUMBER:-"+"
"+MODTOHAFSD_CalculateValues.o_iter+" </th></tr>");
                        myWriter.write("<tr > <th>Distance (km) </th><th> Bouguer anamolies (mGal)
</th> <th>  Assumed/Estimated regional (mGal) </th><th> Residual
anomalies(mGal) </th><th>Modeled anomalies(mGal) </th><th>Error(mGal)
</th><th>Depth(km) </th> </tr>");

                        for ( int K = 1; K <= MODTOHAFSD_CalculateValues.input_n_obs; K++){
```

```
                            myWriter.write("<tr> <td>" +
d.format(MODTOHAFSD_CalculateValues.x[K])+"</td><td>"+df.format(MODTOH
AFSD_CalculateValues.input_nob_gob[K])+"</td>
<td>"+df.format(MODTOHAFSD_DrawGraph.fc[K])+"</td><td>"+df.format(MODT
OHAFSD_CalculateValues.in_gob[K])+"</td><td>"+df.format(MODTOHAFSD_Cal
culateValues.o_gc[K])+"</td><td>"+df.format(MODTOHAFSD_CalculateValues
.err[K])+"</td><td>"+df.format(MODTOHAFSD_CalculateValues.o_dep[K])+"<
/td></tr>");
                        }
                        myWriter.write("</table>");

                        DecimalFormat d1 =new DecimalFormat("0.####");
                        myWriter.write("<BR>");
                        myWriter.write("Polynomial coefficients of regional background:"+"<BR>");
                        myWriter.write("-----------------------------------------<BR>");
                        for ( int i = 1; i <= MODTOHAFSD_CalculateValues.i_d_poly + 1; i++) {

                            myWriter.write("f"+ ( i - 1 ) +" =
"+d1.format(MODTOHAFSD_CalculateValues. d_cftnt_arr[i])+"<BR>");
                        }
                        myWriter.write("<BR>");

                        myWriter.close();
                    }
                }
                else
                {
                    //pops up error message

                }

            }
            catch(Exception e1) {

                e1.printStackTrace();
            }
        }else if(ae.getActionCommand().equals("Load file")){

            MODTOHAFSD_MainPanel.loadData();
        }else if(ae.getActionCommand().equals("Clear")||ae.getActionCommand().equals("New")){

            MODTOHAFSD_MainPanel.clearDefaultValues();
            MODTOHAFSD_MainPanel.clearPanel(MODTOHAFSD_MainPanel.img);
            com.modtohafsd.view.MODTOHAFSD_TableView.populateEastPanel(rowdata);
            com.modtohafsd.view.MODTOHAFSD_TableView.val.setText("");
            MODTOHAFSD_CalculateValues.len = 0;
            setZero();
        }else if(ae.getActionCommand().equals("Save file")){

            try{
                String current = System.getProperty("user.dir");
                JFileChooser saveFile = new JFileChooser(current);
                File OutFile = saveFile.getSelectedFile();
                FileWriter myWriter = null;
                if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION){

                    OutFile = saveFile.getSelectedFile();

                    if (OutFile.canWrite() || !OutFile.exists()){

                        myWriter = new FileWriter(OutFile+".txt");
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_area_name);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_ite_num);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_n_obs);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_ddx_km);
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.input_n_obs; i++){
                            myWriter.write(""+MODTOHAFSD_CalculateValues.input_ele_km[i]+",") ;
                        }
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.input_n_obs; i++){
                            myWriter.write(""+MODTOHAFSD_CalculateValues.input_nob_gob[i]+",") ;
```

```
                        }
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_min_ano);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_max_ano);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_al_err);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.input_zmax_km);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_LambInt.sdval);
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_LambInt.lamval);
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.input_n_obs; i++){
                            myWriter.write(""+(MODTOHAFSD_CalculateValues.input_y_km[i])+",") ;
                        }
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.input_n_obs; i++){
                            myWriter.write(""+(MODTOHAFSD_CalculateValues.input_strike_km[i])+",")
;
                        }

                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.len);
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.len; i++){
                            myWriter.write(MODTOHAFSD_CalculateValues.d_x_km_arr[i]+",");
                        }
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.len; i++){
                            myWriter.write(MODTOHAFSD_CalculateValues.d_z_km_arr[i]+",");
                        }
                        myWriter.append(System.getProperty("line.separator"));
                        myWriter.write(""+MODTOHAFSD_CalculateValues.i_d_poly);
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.i_d_poly+1; i++){
                            myWriter.write(MODTOHAFSD_CalculateValues.d_cftnt_arr[i]+",");
                        }
                        myWriter.append(System.getProperty("line.separator"));
                        for (int i = 1; i <= MODTOHAFSD_CalculateValues.input_n_obs; i++){
                            myWriter.write(""+(MODTOHAFSD_GetConstReg.input_reg_val[i])+",") ;
                        }

                        myWriter.close();
                    }
                }
                else
                {
                    //pops up error message

                }

            }
            catch(Exception e1) {

                e1.printStackTrace();
            }
        }else if(ae.getActionCommand().equals("Constant offset")){
            cv.getAnamolyValues(MODTOHAFSD_MainPanel.captureValues());
            MODTOHAFSD_GetConstOff gc = new MODTOHAFSD_GetConstOff();
            gc.constOff();

        }else if(ae.getActionCommand().equals("Constant half strike")){

            cv.getAnamolyValues(MODTOHAFSD_MainPanel.captureValues());
            MODTOHAFSD_GetConstHaf gch = new MODTOHAFSD_GetConstHaf();
            gch.constHaf();
        }else if(ae.getActionCommand().equals("Constant regional")){
            cv.getAnamolyValues(MODTOHAFSD_MainPanel.captureValues());
            try {
                cv.getCoefficients();
            } catch (MODTOHAFSD_HandleException e) {

                e.printStackTrace();
```

```
            }

                        MODTOHAFSD_GetConstReg gcr = new MODTOHAFSD_GetConstReg();
            gcr.constReg();
            for(int i=1;i<=MODTOHAFSD_CalculateValues.input_n_obs;i++){
                MODTOHAFSD_CalculateValues.len=0;
            }
        }else if(ae.getActionCommand().equals("Variable offset")){

            try{
                MODTOHAFSD_TableView.populateEastPanel(rowdata);
                cv.getAnamolyValues(MODTOHAFSD_MainPanel.captureValues());
                MODTOHAFSD_OffsetVal ov = new MODTOHAFSD_OffsetVal();
                ov.val();
                DecimalFormat df = new DecimalFormat("0.##");

                for (int i = 1;i <= MODTOHAFSD_CalculateValues.input_n_obs; i++){
                    for (int j = 0;j < 2; j++){
                        MODTOHAFSD_OffsetVal.off[i-1][1]=
df.format(MODTOHAFSD_GetConstOff.input_y_km[i]);
                        MODTOHAFSD_OffsetVal.off[i-1][0]= df.format(i);
                        MODTOHAFSD_OffsetVal.table.setValueAt(MODTOHAFSD_OffsetVal.off[i-1][j],i-1
, j);
                    }
                }
            }
            catch(Exception e){

            }

        }else if(ae.getActionCommand().equals("Variable half strike")){

            try{
                MODTOHAFSD_TableView.populateEastPanel(rowdata);
                cv.getAnamolyValues(MODTOHAFSD_MainPanel.captureValues());
                MODTOHAFSD_HalfstrikeVal sv = new MODTOHAFSD_HalfstrikeVal();
                sv.val();
                DecimalFormat df = new DecimalFormat("0.##");

                for (int i = 1; i <= MODTOHAFSD_CalculateValues.input_n_obs; i++){
                    for (int j = 0; j < 2; j++){
                        MODTOHAFSD_HalfstrikeVal.haf[i-1][1]=
df.format(MODTOHAFSD_GetConstHaf.input_strike_km[i]);
                        MODTOHAFSD_HalfstrikeVal.haf[i-1][0]= df.format(i);
                        MODTOHAFSD_HalfstrikeVal.table.setValueAt(MODTOHAFSD_HalfstrikeVal.haf[i-1
][j],i-1, j);
                    }
                }
            }
            catch(Exception e){

            }
        }else if(ae.getActionCommand().equals("Flow module")){

            try {
                String path =  "Flow module.txt";
                File pptFile = new File(path);
                if (pptFile.exists()) {
                    if (Desktop.isDesktopSupported()) {
                        Desktop.getDesktop().open(pptFile);
                    } else {
                        System.out.println("Desktop is not supported!");
                    }

                } else {
                    System.out.println("File does not exists!");
                }

                System.out.println("Done");

            } catch (Exception ex) {
                JFrame frame = null;
                JOptionPane.showMessageDialog(frame,
                        "System cannot open the file\n",
                        "Error!",
```

```
                    JOptionPane.ERROR_MESSAGE);

        }
    }else if(ae.getActionCommand().equals("Readme")){

        try {
            String path =  "Readme.txt";
            File pptFile = new File(path);
            if (pptFile.exists()) {
                if (Desktop.isDesktopSupported()) {
                    Desktop.getDesktop().open(pptFile);
                } else {
                    System.out.println("Desktop is not supported!");
                }

            } else {
                System.out.println("File does not exists!");
            }

            System.out.println("Done");

        } catch (Exception ex) {
            JFrame frame = null;
            JOptionPane.showMessageDialog(frame,
                    "System cannot open the file\n",
                    "Error!",
                    JOptionPane.ERROR_MESSAGE);

        }
    }else if(ae.getActionCommand().equals("Exit")){
        JFrame frame = null;

        int r = JOptionPane.showConfirmDialog(
                frame,
                "Exit MODTOHAFSD ?",
                "Confirm Exit ",
                JOptionPane.YES_NO_OPTION);
        if(r == JOptionPane.YES_OPTION ){
            if(success==false){
                String fileName = MODTOHAFSD_CalculateValues.input_area_name+".jpg";
                File f = new File(fileName);
                f.delete();
            }
            System.exit(0);
        }
    }
}
public void setZero(){
    MODTOHAFSD_CalculateValues.input_area_name = "";
    MODTOHAFSD_CalculateValues.count=0;
    MODTOHAFSD_CalculateValues.d_cftnt_arr=null;
    MODTOHAFSD_CalculateValues.d_x_km_arr = null;
    MODTOHAFSD_CalculateValues.d_z_km_arr = null;
    MODTOHAFSD_CalculateValues.dep=null;
    MODTOHAFSD_CalculateValues.err=null;
    MODTOHAFSD_CalculateValues.err1=null;
    MODTOHAFSD_CalculateValues.i_d_poly = 0;
    MODTOHAFSD_CalculateValues.in_gob=null;
    MODTOHAFSD_CalculateValues.input_al_err=0;
    MODTOHAFSD_CalculateValues.input_ddx_km=0;
    MODTOHAFSD_CalculateValues.input_ele_km=null;
    MODTOHAFSD_CalculateValues.input_ite_num=0;
    MODTOHAFSD_CalculateValues.input_lambda_st=0;
    MODTOHAFSD_CalculateValues.input_max_ano=0;
    MODTOHAFSD_CalculateValues.input_min_ano=0;
    MODTOHAFSD_CalculateValues.input_n_obs=0;
    MODTOHAFSD_CalculateValues.input_nob_gob=null;
    MODTOHAFSD_CalculateValues.input_reg_val=null;
    MODTOHAFSD_CalculateValues.input_sd_val=null;
    MODTOHAFSD_CalculateValues.input_strike_km=null;
    MODTOHAFSD_CalculateValues.input_y_km=null;
    MODTOHAFSD_CalculateValues.input_zmax_km=0;
    MODTOHAFSD_CalculateValues.len=0;
    MODTOHAFSD_CalculateValues.np=0;
    MODTOHAFSD_CalculateValues.o_dep=null;
```

```
        MODTOHAFSD_CalculateValues.o_funct=null;
        MODTOHAFSD_CalculateValues.o_gc=null;
        MODTOHAFSD_CalculateValues.o_iter=0;
        MODTOHAFSD_CalculateValues.obj=null;
        MODTOHAFSD_CalculateValues.vsd=null;
        MODTOHAFSD_CalculateValues.x=null;


    }
}
```

```
package com.modtohafsd.util;

import com.modtohafsd.model.MODTOHAFSD_CalculateValues;

public class MODTOHAFSD_Utility {

    public static double convertDouble(String str) throws Exception {

        Double temp = new Double(str.trim());
        return temp.doubleValue();
    }

    public static String convertString(String str) throws Exception {

        String temp = new String(str.trim());
        return temp;
    }

    public static int convertInteger(String str) throws Exception {

        Integer temp = new Integer(str.trim());
        return temp.intValue();
    }

    public static int findMaximumNumber( double observe[]) {

        double max = 0.0d;
        for (int i = 0; i < observe.length; i++) {

            if (Math.abs(observe[i]) > Math.abs(max)) {

                max = observe[i];
            }
        }

        int maxVal = (int) max/3*5;
        return maxVal;
    }

    public static int findMaximumNumber( double observe) {

        double max = 0.0d;
        int maxVal=0;
        max = observe;

        if (max < 5) {

            maxVal = 5;
        }
        else if (max >= 5 && max <= 10) {

            maxVal = 10;
        }
        else if ( max > 10 && max <= 15) {

            maxVal = 15;
        }
        else if (max > 15 && max <= 20) {

            maxVal = 20;
        }
        else
```

```
    {
        maxVal = MODTOHAFSD_CalculateValues.o_iter;
    }

    return maxVal;
}

public static double findMaximumNumber1( double observe[]) {

    double max = 0.0d;
    for (int i = 1; i < observe.length; i++) {

        if (Math.abs(observe[i]) > Math.abs(max)) {

            max =(observe[i]);
        }
    }

    double maxVal =  max;
    return maxVal;
}
public static double findMinimumNumber1( double observe[]) {

    double max = 0.0d;
    for (int i = 1; i < observe.length; i++) {

        if ((observe[i]) < (max)) {

            max = (observe[i]);
        }
    }

    double maxVal =  max;
    return maxVal;
}

public static double findMaximumNumber( double observe[], double anoVal) {

    double max = anoVal;
    for (int i = 1; i < observe.length; i++) {

        if ((observe[i]) > (max)) {

            max = (observe[i]);
        }
    }

    double maxVal =  max;
    return maxVal;
}


public static double[] convertDoubleArray(String str) throws Exception {

    java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
    String temp = "";
    java.util.ArrayList arr = new java.util.ArrayList();

    while(st.hasMoreTokens()) {

        temp = st.nextToken();
        arr.add(temp);
    }
    double d_array[] = new double[arr.size() + 1];

    for(int i = 0; i <= arr.size(); i++) {

        if (i == 0)
            d_array[i] = 0.0;
        else
            d_array[i] = convertDouble( arr.get(i-1).toString() );
    }
    return d_array;
}
```

```
    public static double[] DoubleArray(String str) throws Exception {

        java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
        String temp = "";
        java.util.ArrayList arr = new java.util.ArrayList();

        while(st.hasMoreTokens()) {

            temp = st.nextToken();
            arr.add(temp);
        }
        double d_array[] = new double[arr.size() + 1];

        for(int i = 0; i < arr.size(); i++) {
            d_array[i] = convertDouble( arr.get(i).toString() );
        }
        return d_array;
    }

    public static int[] convertIntegerArray(String str) throws Exception {

        java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
        String temp = "";
        java.util.ArrayList arr = new java.util.ArrayList();

        while(st.hasMoreTokens()) {

            temp = st.nextToken();
            arr.add(temp);
        }
        int d_array[] = new int[arr.size() + 1];

        for(int i = 0; i < arr.size(); i++) {

            d_array[i] = convertInteger( arr.get(i).toString() );

        }
        return d_array;
    }
}
-----------------------------------------------------------------------------------------------

package com.modtohafsd.util;

import java.awt.*;
import com.modtohafsd.view.MODTOHAFSD_MainPanel;

public class MODTOHAFSD_HandleException extends Exception{
    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public MODTOHAFSD_HandleException(){
        Graphics g = MODTOHAFSD_MainPanel.img.getGraphics();
        g.setColor(Color.white);
        g.fillRect(0, 0, 1000, 600);
        g.setColor(Color.black);
        g.setFont(new Font("Arial", 20, 40));
        g.drawString("ERROR...", 400, 325);
    }
}
```

CHAPTER
FOUR

Gravity Anomaly Modeling of Multiple Geologic Sources having Differing Strike Lengths and Arbitrary Density Contrast Depth Variations*

4.1 General

The deduced Bouguer gravity anomalies after a gravity survey are considered to have been measured on topographic elevations below which are included several anomalous mass distributions. Modeling and inversion techniques are being routinely used to trace the outlines of anomalous bodies from the observed gravity anomalies.

In gravity anomaly calculations, most studies consider the density contrast as a constant (Pohánka, 1988, Ferguson et al., 1988). Owing to the fact that variable density functions yield more reliable interpretations, many forward modeling schemes are being developed using variable density contrast functions to compute the gravity anomalies of geologic sources, describing their geometries by a variety of geophysical models as briefed under section 2.1 of Chapter-II. However, these density functions

---

* *Published in Near Surface Geophysics (2013, 11, 363-370), European Association of Geoscientists & Engineers (EAGE), The Netherlands.*

are efficient only when the subsurface density distribution follows some empirical relationship either with depth or lateral position or a combination of both (Chakravarthi and Ramamma, 2013d).

More often, vertically homogeneous rocks can be metamorphosed leading to horizontal changes in density between different rock types. It is also not uncommon that the subsurface density contrast may vary significantly within short distances because of structural disturbances as well as the differing lithological character of the rocks forming various parts of the subsurface geology. Also the subsurface density distribution becomes too complicated to be simulated by any one of the existing density functions in regions where the depositional history, tectonics, regional metamorphism and geochemical processes play vital roles in the evolutionary processes of geological settings. Therefore, when attempt is made to analyze the gravity anomalies of the subsurface, the anomalous mass(es) must be described with appropriate geometries taking into account their strike lengths and appropriate density contrasts (Chakravarthi and Ramamma, 2013d).

This chapter deals with a semiautomatic modeling technique and related GUI based software, FORMODTOHAF, in JAVA to interactively model the gravity anomalies of multiple geologic sources having differing strike lengths, thicknesses and density contrast variations. The applicability of the technique is demonstrated with both synthetic and real field examples.

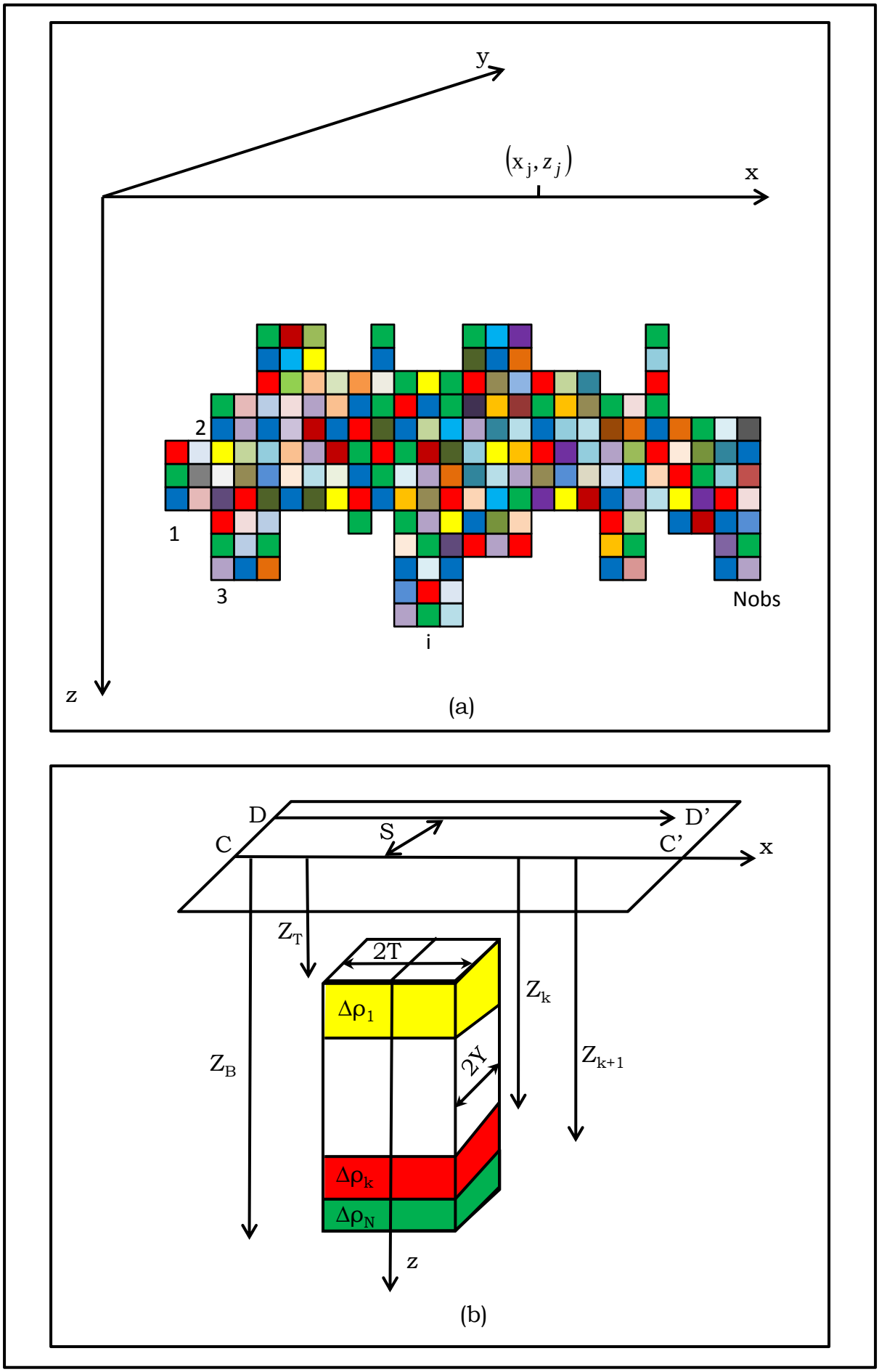

Figure 4.1 (a) Schematic representation of an anomalous mass in xz-plane by a stack of 2.5-D vertical prisms. Different colours within the source represent different density contrasts. (b) Geometry of a 2.5-D vertical prism with N formations.

4.2 Forward modeling - theoretical considerations

In a Cartesian-coordinate system, let the z-axis be positive vertically downwards with the x-axis transverse to the strike of the anomalous mass and the density contrast within the structure vary arbitrarily both with depth and lateral position (Figure 4.1a). In the xz-plane, the source is approximated with a series of vertical prisms, N_{obs} in number, all having the same width, $2T$, but of different strike lengths, $2Y_i$, $i = 1, 2, .., N_{obs}$. Further, each prism may consist in any number of formations and the density contrasts of these formations may vary arbitrarily in the vertical and horizontal directions but for each formation the density contrast is assumed as constant and remains unchanged along its strike. Figure 4.1b shows the geometry of a prism at the i^{th} observation on the profile. Let this prism consist in N formations and be contained in the vertical direction between the limits z_T and z_B. Each one of the formations of the prism has a different density contrast, $\Delta\rho_l$, l = 1, 2,…, N (Figure 4.1b). The gravity anomaly of the structure, Δg_s, at any point, (x_j, z_j), outside the source region (Figure 4.1a) can be expressed as (Chakravarthi and Ramamma, 2013d)

$$\Delta g_s = \sum_{i=1}^{N_{obs}} \Delta g_i^{prm}, \tag{4.1}$$

where,

$$\Delta g_i^{prm} = \sum_{k=1}^{N} \Delta g_k^{mod}. \tag{4.2}$$

Here, Δg_k^{mod}, is the gravity effect of the k^{th} formation within the prism represented by,

$$\Delta g_k^{mod} = -G \int_v \Delta\rho_k \frac{\partial}{\partial z} \frac{1}{\sqrt{(u-x_j)^2 + v^2 + (w-z_j)^2}}\, dudvdw. \tag{4.3}$$

Here, (u, v, w) are the coordinates of an element volume within the k^{th} formation and $\Delta\rho_k$ is the density contrast of the corresponding layer.

Equation (4.3) can be expressed as

$$\Delta g_k^{mod} = G\Delta\rho_k \int_{u=-T}^{T} \int_{v=-Y}^{Y} \int_{w=z_k}^{z_{k+1}} \frac{(w-z_j)\, dudvdw}{\left[(u-x_j)^2 + v^2 + (w-z_j)^2\right]^{\frac{3}{2}}}, \tag{4.4}$$

where, z_k and z_{k+1} represent the depths to the top and bottom of the k^{th} formation respectively. Upon integration equation (4.4) becomes (Rama Rao et al., 1999)

$$\Delta g_k^{mod} = 2G\Delta\rho_k \left\{ B\, tan^{-1}\frac{YA}{BR} - \frac{Y}{2}\ln\frac{R+A}{R-A} - \frac{A}{2} ln\frac{R+Y}{R-Y} \right\}_{\substack{u=T \\ u=-T}}^{\substack{w=z_{k+1} \\ w=z_k}}, \tag{4.5}$$

where, $A = (u-x_j)$, $B = (w-z_j)$, $R = \sqrt{A^2 + B^2 + Y^2}$. This equation is valid only for the observations located on the profile, CC' (Figure 4.1b). The anomalous field at any offset, s, (such as on the profile, DD', Figure 4.1b) can be computed based on the procedure described in section 3.2.1 of Chapter-III.

4.2.1 Effects of profile azimuth and strike length on the magnitude of gravity anomaly

Figure 4.2a shows the nature of gravity anomalies produced by a 2.5-D strike vertical prism along two selected profiles, out of which one profile (FF') runs transverse to the strike of the prism while the other (HH') at 45° (Figure 4.2c). In this case, the prism is presumed to consist in three geologic formations with each one possessing 10 km width (Figure 4.2b) and 20 km strike length (Figure 4.2c).

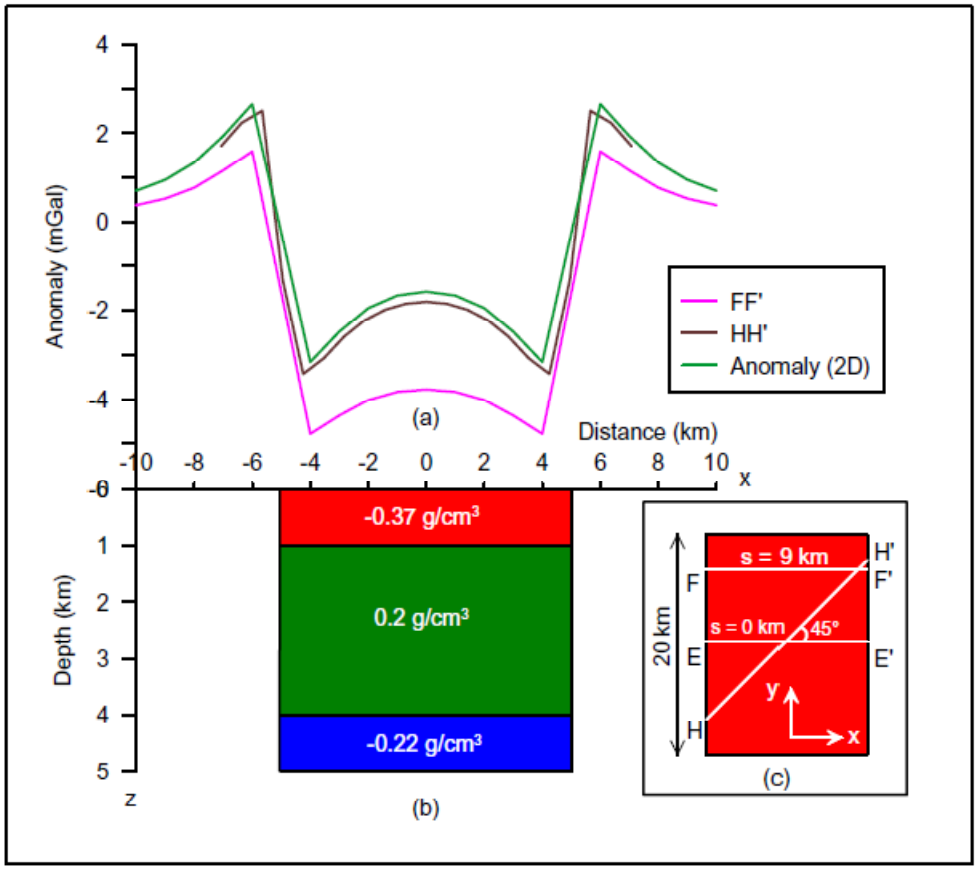

Figure 4.2 (a) Cumulative gravity effect of a 2.5-D prism along two selected profiles, FF' and HH'. Anomaly produced by the same prism with infinite strike length (2-D) is also shown for comparison. (b) Geometry of a vertical prism containing three formations in xz- plane. (c) Plan view of a vertical prism (schematic) in xy- plane with the orientation of selected profiles.

The assumed thicknesses of the three formations from top to bottom are 1.0 km, 3.0 km and 1.0 km with corresponding density contrasts -0.37 g/cm^3, 0.2 g/cm^3, and -0.22 g/cm^3 respectively (Figure 4.2b). The assumed values of -0.37 g/cm^3 and -0.22 g/cm^3 typically represent the density contrasts of sedimentary rocks at shallower and larger depths, and 0.2 g/cm^3 represents the density contrast of flood basalt (presuming 2.67 g/cm^3 as the density of basement complex).

It must be noted that equation (4.5) cannot be used to compute the gravity anomalies of the prism along the profile, HH', because the profile runs at an angle to the strike of the prism (Figure 4.2c). In such a case, the observer locations along this profile must be projected on to a profile that runs transverse to the strike of the prism. This could be done by multiplying the distance of each observation on the profile, HH', with $cos\alpha$, where α is the angle between a profile that runs perpendicular to the strike of the prism and the profile HH' (Chakravarthi and Ramamma, 2013d). In this case, the observer locations along the profile, HH', in the interval, $x_j \in$ [-10 km, 10 km], are projected on to the profile, EE', before using equation (4.5) to compute the gravity response (Figure 4.2a) of the structure. The anomaly produced by the same structure along a profile, FF', at 9 km offset in the interval $x_j \in$ [-10 km, 10 km] is also shown in Figure 4.2a for comparison. One can notice from Figure 4.2a that the anomalies calculated on the profile, FF', and those by projecting the observations along the profile, HH', onto the profile, EE', show significant deviations from each other.

Further, to study the effect of strike length on the anomaly; a large value has been assigned to the strike of the prism (2-D) while keeping the other parameters intact. The corresponding anomalous field calculated using equation (4.5) is shown in Figure 4.2a. The magnitude of the anomaly in this case differs from the one calculated by presuming the structure as 2.5-D.

Therefore, the magnitude of gravity anomaly over a geologic structure is dependent on its strike length, offset of the profile, and also on the azimuth of the profile. The fact that the present algorithm is also dependent on the magnitude of the anomalous filed consider these parameters as vital in gravity modeling studies (Chakravarthi and Ramamma, 2013d).

4.3 Description of software - FORMODTOHAF

Based on the methodology presented in section 4.2, a GUI based JAVA software, FORMODTOHAF, is developed to model the gravity anomalies of multiple subsurface geologic sources in an interactive mode (Appendix 4.1). The code is user friendly in the sense that it allows model construction and modification, the display of model geometry, depth and the densities of various sub-surface formations, and real-time computation of the gravity anomalies arising from the model. The input layout consists of six fields and four menus. The graphical layout consists of two panels one each for the anomaly and structure as shown in Figure 4.3.

The information pertaining to Area/Profile, number of observations, station interval (km), observed anomaly (mGal), angle of the profile (degrees) and station elevations (km) are to be entered in appropriate fields of the input layout. The parameters pertaining to half strike length (km), offset of the profile (km), number of formations that each prism consists in, and their corresponding density contrasts and thicknesses need to be specified under the menu 'Model information'.

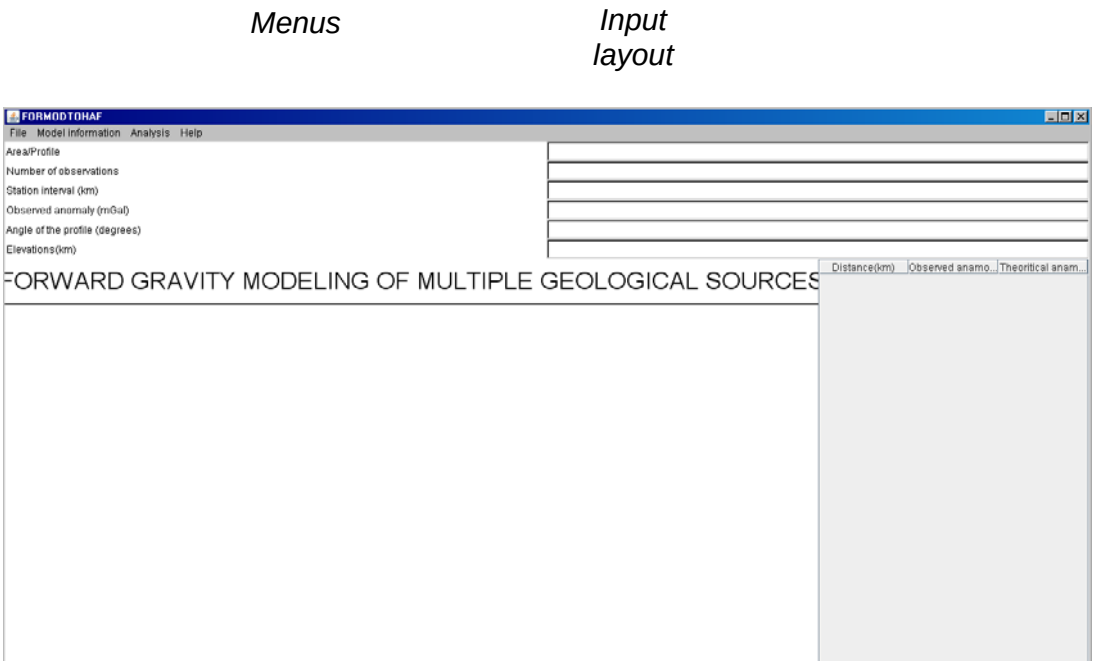

Figure 4.3 View module of FORMODTOHAF

The forward modeling option under the analysis menu computes the gravity field of the model space and displays the response in the anomaly panel along with the observed gravity anomaly. The corresponding model space will also be displayed in the structure panel. The difference between the observed and modeled gravity anomalies can be minimized in an

interactive approach by modifying either density contrasts or thickness parameters or a combination of both.

4.4 Applications

The applicability of the method is demonstrated with two examples, one synthetic and a real. The synthetic model corresponds to a local scale structure, where the depth of exploration is about 5 km, whereas the real field example signifies a crustal scale structure associated with a suture zone with the depth of investigation in excess of 40 km. In either case the observer is on top of the topography, at z_j = 0 km.

(a) Synthetic example

The plan view of a north-south striking sedimentary basin is shown in Figure 4.4a with its depth structure in Figure 4.4c. The strike length of the basin is finite but varies considerably across its width (Figure 4.4a). The structure resembles to that of a typical block faulted rifted sedimentary basin, where the post depositional tectonic activity might have resulted in the formation of two prominent depressions in the basement separated by a basement high (Figure 4.4c). In this case, low density sediments are exposed at the surface in the vicinity of the boundary fault towards the east and concealed under a thick pile of flood basalt towards the west (Figure 4.4c). The profile, PP', runs transverse and covers the lateral dimensions of the basin completely (Figure 4.4a). The basement is exposed at the surface on either ends of the profile at 1st and 34th km. The thickness of the basalt varies considerably along the profile, about 0.5 km towards the east to about 3.0 km towards the west. The

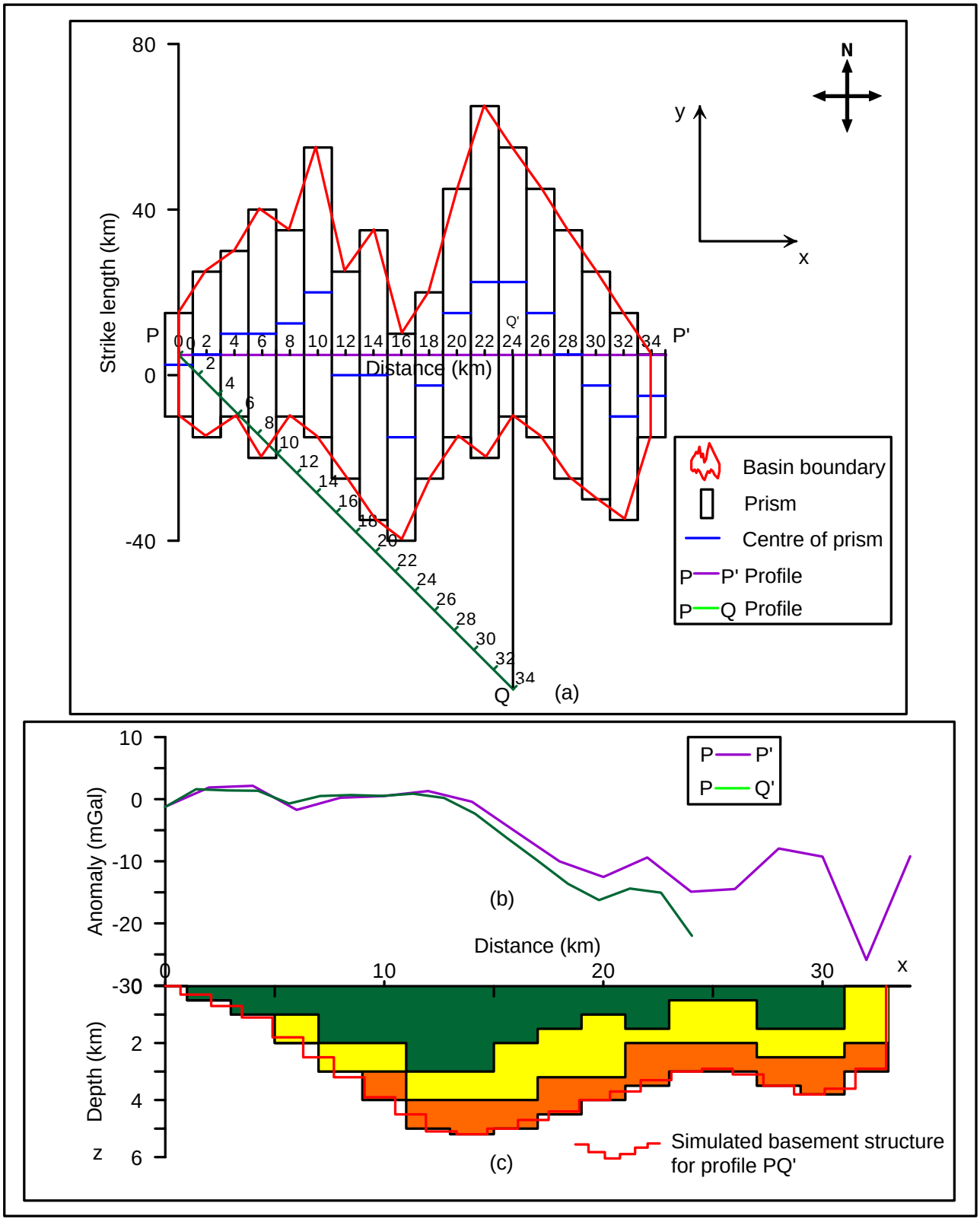

Figure 4.4 (a) Plan view of a sedimentary basin with the locations of selected profiles, PP' and PQ. (b) Gravity responses along PP' and PQ. (c) 3-layered depth structure of a basin. Note that the density contrast-depth data given in Table 1 is used to calculate the gravity anomalies of the structure along the profile, PP', and Table 2 for PQ' respectively.

assumed density contrast distribution of the basin is given in Table 4.1. One can notice from Table 4.1 that the density contrast (absolute) of the sediments at larger depths in the vicinity of the basement ridge shows lesser magnitude, because of compaction due to post-depositional tectonic activity. By and large the density contrast of the basalt remains unchanged, whereas the density contrasts of concealed sediments vary significantly both with depth and lateral position thus making it difficult to simulate it with any of the existing density functions.

To realize forward gravity modeling of such a structure along a selected profile, PP', the subsurface is approximated with a stack of 18 juxtaposed vertical prisms having variable but finite strike lengths (Figure 4.4a and Figure 4.4c). Because the profile does not bisect the strike lengths of the prisms except at 12 and 14 km, the offset of the profile from the centre of each prism is measured and supplied to the algorithm as part of the input.

The gravity response of the structure calculated at 18 observations in the interval $x_j \in$ [0 km, 34 km] along the profile, PP' is shown in Figure 4.4b. One can see from Figure 4.4b that the magnitude of the anomaly hardly exceeds -2 mGal between 0–14 km on the profile although low-density sediments of about 2 km thick exist below the basalt cover in the western part. In the central part the cumulative gravity effect of the low-density sediments exceeds the gravity effect of the overlying basalt thereby yielding an anomaly of about -20 mGal. Towards the east the anomaly attains its maximum (-25.9 mGal) over the exposed sediments.

Table 4.1 Assumed density contrast-depth data to compute the gravity anomalies of the structure along the profile, PP', synthetic example

PRM/ NFM	1/1	2/1	3/1	4/2	5/2	6/3	7/3	8/3	9/3	10/3	11/3	12/3	13/3	14/3	15/3	16/3	17/2	18/1
FM-1	0.0 (0)	0.2 (0.5)	0.2 (1.0)	0.2 (1.0)	0.2 (1.0)	0.2 (2.0)	0.2 (3.0)	0.2 (3.0)	0.2 (2.0	0.2 (1.5)	0.2 (1.0)	0.2 (1.5)	0.2 (0.5)	0.2 (0.5)	0.2 (1.5)	0.2 (1.5)	-0.37 (2.0)	0.0 (0)
FM-2				-0.32 (2.0)	-0.31 (3.0)	-0.31 (3.0)	-0.30 (4.0)	-0.31 (4.0)	-0.30 (4.0)	-0.29 (3.2)	-0.27 (3.2)	-0.24 (2.0)	-0.23 (2.0)	-0.22 (2.0)	-0.27 (2.5)	-0.30 (2.5)	-0.32 (3.0)	
FM-3						-0.27 (4.0)	-0.27 (5.0)	-0.27 (5.2)	-0.24 (5.0)	-0.2 (4.5)	-0.18 (4.0)	-0.16 (3.5)	-0.14 (3.0)	-0.13 (3.0)	-0.2 (3.5)	-0.27 (3.8)		

PRM/NFM = Prism/number of formations
FM stands for formation
Open numbers are density contrasts (g/cm^3)
Numbers in brackets are depth(s) of formation(s) in km

On the other hand, if the anomalies are intended to be calculated using equation (4.1) along the profile, PQ, (Figure 4.4a), which runs at 45° to the strike of the structure, then the observer locations, $x_j \in$ [0 km, 34 km], along this profile must be projected onto a profile that runs transverse to the strike of the source, such as PP' (Figure 4.4a). It is to be noted that the projected component of the profile PQ on the profile PP' (shown as PQ' in Figure 4.4a) fails to cover the lateral dimensions of the structure completely. Furthermore, the station interval in this case becomes 1.414 km and accordingly the subsurface needs to be approximated by a stack of prisms with the width of each prism as 1.414 km. As a result, the subsurface is described with a set of 25 vertical prisms having variable but finite strike lengths (the simulated structure of the basement in this case is shown in Figure 4.4c). The density contrast-depth data for this simulation of the structure are given in Table 4.2. The gravity anomaly of the structure calculated over the length of the profile, PQ', is shown in Figure 4.4b. One can notice from Figure 4.4b that the magnitude of the gravity anomaly in this case differs from the one calculated along the profile, PP'.

Therefore, interpretation schemes that are valid to analyze the gravity anomalies along profiles transverse to the strike of the anomalous mass should not be used directly to analyze the gravity anomalies if the profile fails to run perpendicular to the strike of the anomalous source (Chakravarthi and Ramamma, 2013d).

Table 4.2 Assumed density contrast-depth data to compute gravity anomalies of the structure along the projected profile, PQ', Synthetic example

PRM/ NFM	1/1	2/1	3/1	4/1	5/2	6/2	7/2	8/3	9/3	10/3	11/3	12/3	13/3
FM1	0.0 (0)	0.2 (0.5)	0.2 (0.5)	0.2 (1.0)	0.2 (1.0)	0.2 (2.0)	0.2 (2.0)	0.2 (2.0)	0.2 (3.0)	0.2 (3.0)	0.2 (3.0)	0.2 (2.0)	0.2 (2.0)
FM2					-0.32 (2.0)	-0.32 (3.0)	-0.32 (3.0)	-0.32 (3.0)	-0.32 (4.0)	-0.32 (4.0)	-0.32 (4.0)	-0.32 (4.0)	-0.32 (4.0)
FM3								-0.27 (4.0)	-0.27 (5.0)	-0.27 (5.0)	-0.27 (5.2)	-0.27 (5.0)	-0.27 (5.0)

PRM/ NFM	14/3	15/3	16/3	17/3	18/3	19/3	20/3	21/3	22/3	23/3	24/2	25/1
FM1	0.2 (1.5)	0.2 (1.0)	0.2 (1.5)	0.2 (1.5)	0.2 (0.5)	0.2 (0.5)	0.2 (0.5)	0.2 (1.5)	0.2 (1.5)	0.2 (1.5)	0.37 (2.0)	0.0 (0)
FM2	-0.32 (3.2)	-0.32 (3.2)	-0.32 (2.0)	-0.32 (2.0)	-0.32 (2.0)	-0.32 (2.0)	-0.32 (2.0)	-0.32 (2.5)	-0.32 (2.5)	-0.32 (2.5)	-0.32 (3.0)	
FM3	-0.27 (4.5)	-0.27 (4.0)	-0.27 (3.5)	-0.27 (3.5)	-0.27 (3.0)	-0.27 (3.0)	-0.27 (3.0)	-0.27 (3.5)	-0.27 (3.8)	-0.27 (3.8)		

PRM/NFM = Prism/number of formations
FM stands for formation
Open numbers are density contrasts (g/cm^3)
Numbers in brackets are depth(s) of formation(s) in km

(b) Field example - Superior and Churchill structural provinces, Canadian Shield

The Superior and Churchill structural provinces of the Canadian Shield meet across a major Proterozoic fold belt known as the Circum-Ungava geosyncline, which was interpreted by Wilson (1968) as a Precambrian suture. Based on the structural features, the geosyncline is segmented into the Belcher fold belt, the Cape Smith fold belt and the Labrador trough. The Cape Smith fold belt forms a part of the suture zone between the Churchill and Superior structural provinces and runs east-northeast across the northern part of the Ungava craton over a distance of 350 km. Both mafic and ultramafic intrusive rocks occur throughout the belt in the form of concordant sills and sedimentary rocks occur toward the base of the fold belt succession. Granodioritic gneiss, paragneiss and a granulite suite of rocks form the Churchill province whereas Archean granodiorite and granodiorite gneiss located to the south belong to the Superior province (Bergeron, 1957; Stam, 1961; Baragar 1974). A major thrust fault with a northward dip of 45–65° cuts the fold belt.

The gravity data collected in and around the two structural provinces referred above include static gravity measurements on land and on an ice-surface and dynamic gravity measurements on the sea-surface. The details of the gravity survey including gravity corrections, data accuracy etc. were described in detail by Mukhopadhyay and Gibb (1981). They have analyzed the gravity anomalies of the region using 2D modeling along a few selected profiles in terms of the deep structure of the proposed

suture and the structure of the fold belt. One such profile cuts across the fold belt and parts of the two bordering cratons is shown in Figure 4.5a. The positive gravity anomaly observed over the Cape Smith fold belt is flanked on either side by negative gravity anomalies. The density contrasts and thicknesses assigned to various blocks of the two provinces including the fold belt in 2-D gravity modeling by Mukhopadhyay and Gibb (1981) are shown in Figure 4.5b. The corresponding modeled gravity anomaly is shown in Figure 4.5a. Their modeling had yielded a maximum thickness of 55 km for the Churchill block, 13 km for the fold belt and 46 km for the Superior crust in the neighborhood of the inter-province boundary.

In the present study, the estimated crustal structure by Mukhopadhyay and Gibb (1981) is used as a starting model for analyzing the observed anomalies by the present method. Unlike the case with 2-D, the present method allows one in assigning different strike lengths to different parts of the crustal blocks and the fold belt. Because the strike length of the Cape Smith fold belt is 350 km (Mukhopadhyay and Gibb, 1981), the subsurface structure was simulated by 98 vertical prisms (80 prisms to describe the two bordering cratons and 18 for the fold belt) having different strike lengths. Large values are assigned to the strike lengths of the prisms that are used to describe the Churchill and Superior provinces whereas a strike length of 350 km was assigned to each of the prisms describing the fold belt. The gravity anomalies calculated at 98 observations in the interval $x_j \in$ [0 km, 400 km] are shown as solid dots in Figure 4.5a. One can see from Figure 4.5a that the observed gravity anomalies and the ones calculated by the present method using the

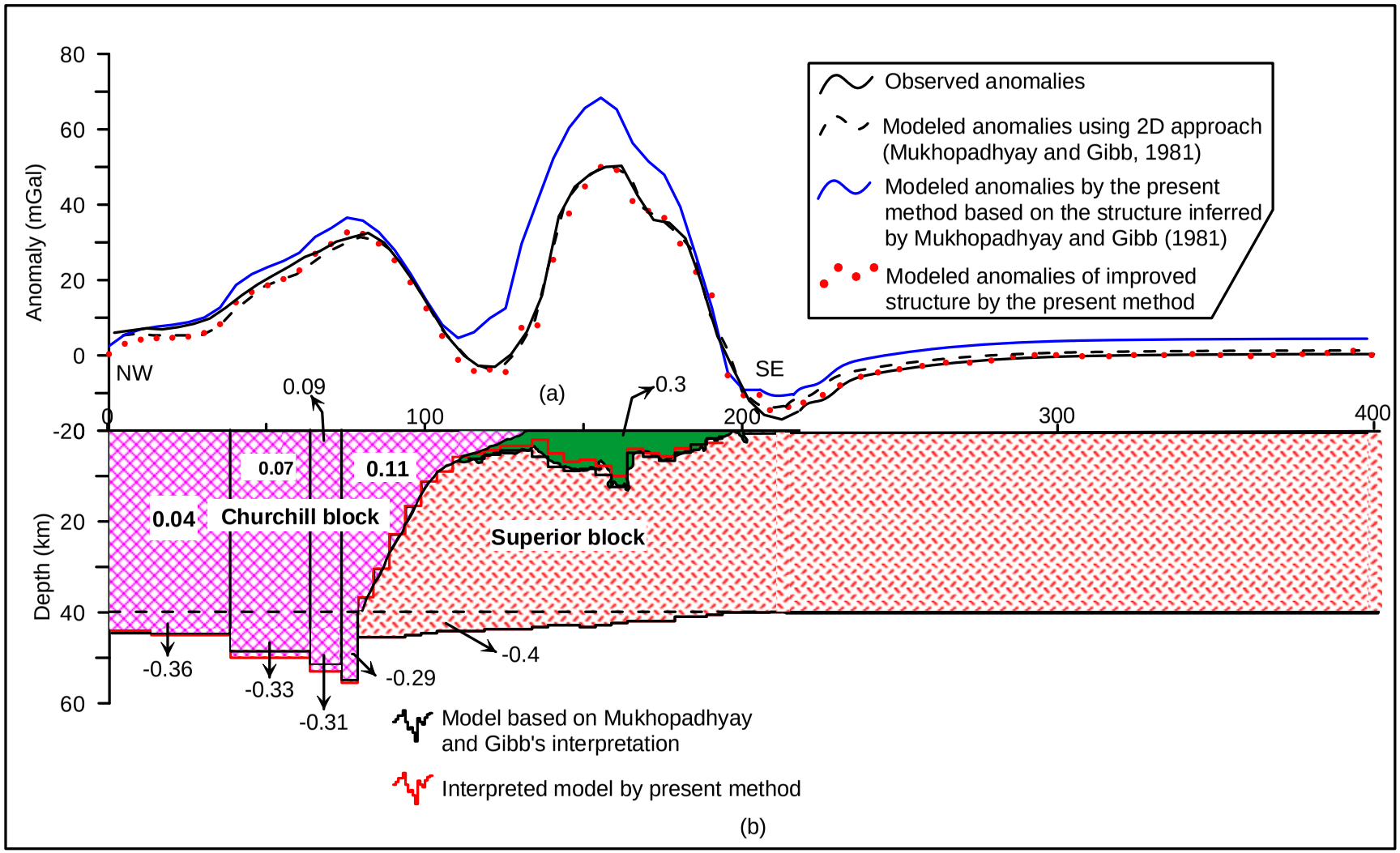

Figure 4.5 (a) Observed and modeled gravity anomalies along a NW-SE profile, Churchill and Superior provinces, Canadian Shield. (b) Modeled deep crustal structure by the present method (Chakravarthi and Ramamma, 2013d). Density contrasts (g/cm^3) assigned to different blocks of the two provinces by Mukhopadhyay and Gibb (1981) are shown along with their interpreted model.

estimated depth structure by Mukhopadhyay and Gibb (1981) display similar characteristics but they differ in magnitude. Attributing such a large difference in magnitude (~10 to 20 mGal) between these anomalies to digitization errors is ruled out (Chakravarthi and Ramamma, 2013d).

The difference between these two anomalies can be minimized either by adjusting the thicknesses of the various blocks used in the interpretational model or the density contrasts assigned to them or a combination of both. Since the density contrasts assigned to the various blocks in the interpretation are determined based on the density measurements of surface rock samples (Mukhopadhyay and Gibb 1981) no attempt was made to modify the density contrasts. Instead, the thicknesses of the crustal blocks are adjusted together with the fold belt until the cumulative gravity effect of the structure mimics the observed anomalies. The refined subsurface structure is shown in Figure 4.5b and the corresponding modeled anomalies in Figure 4.5a. By and large the fit between the modeled anomalies and the observed ones is satisfactory (Figure 4.5a).

One can see from Figure 4.5b that the refined structure from present modeling deviates modestly from the one interpreted by Mukhopadhyay and Gibb (1981). For example, i) the maximum thickness of the Cape Smith fold belt obtained from the present method is 10 km compared to the estimated thickness of 13 km by Mukhopadhyay and Gibb (1981), ii) the western boundary of the base of the fold belt is shallower than the one interpreted by Mukhopadhyay and Gibb (1981)

and iii) the Churchill block is much thicker than the one estimated by Mukhopadhyay and Gibb (1981).

4.4 Results and Discussion

The effects of strike length and azimuth of the profile on the magnitude of the gravity anomaly are discussed in detail. It is demonstrated that if the profile along which modeling is intended runs at an angle to the strike of the anomalous source, then the observer locations need to be projected onto a profile that runs transverse to the strike of the source. Such a transformation would lead to shortening the profile length. Hence, the profile length must be chosen such that its projected component on a profile transverse to the strike of the anomalous source covers the lateral dimensions of the structure completely (Chakravarthi and Ramamma, 2013d).

In case of synthetic example, gravity anomalies are calculated over a typical rifted sedimentary basin along two selected profiles where one runs transverse to the strike of the structure and the other at an angle. It was found that the magnitudes of the anomalies in each case differ from each other even though the subsurface geology remains the same.

In case of real field example pertaining to the Churchill and Superior provinces in the Canadian Shield, the interpreted subsurface model from the present method shows modest deviations from the one reported previously by Mukhopadhyay and Gibb (1981).

The proposed method has the advantage that it can be used to model the gravity anomalies of variety of subsurface geological structures because it does not prevent one in choosing the number of prisms/ formations to be considered in the interpretation model.

"FORMODTOHAF : A GUI based JAVA code for gravity modeling of multiple strike limited geological sources".

The system requirements to run the program are:-

Java Development kit(jdk)1.6 & above
RAM : 512 MB.
Disk space: 70 MB.

```
package com.formodtohaf.view;


import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import com.formodtohaf.control.FORMODTOHAF_Controller;
import com.formodtohaf.model.FORMODTOHAF_CalculateValues;

public class FORMODTOHAF_MainView extends Frame{


    private static final long serialVersionUID = 1L;

    public static void main(String s[])
    {
        FORMODTOHAF_MainView cm = new FORMODTOHAF_MainView();
        cm.setSize(1280, 768);
        cm.addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                JFrame frame = null;
                int r = JOptionPane.showConfirmDialog(
                        frame,
                        "Exit FORMODTOHAF ?",
                        "Confirm Exit ",
                        JOptionPane.YES_NO_OPTION);
                if(r == JOptionPane.YES_OPTION ){
                    if(FORMODTOHAF_Controller.success == false){
                        String fileName = FORMODTOHAF_CalculateValues.input_area_name+".jpg";
                        File f = new File(fileName);
                        f.delete();
                    }
                    System.exit(0);

                }
            }
        });
        cm.setTitle("FORMODTOHAF");
        cm.setResizable(true);
        cm.add(new FORMODTOHAF_MainPanel(cm));
        cm.setVisible(true);

    }

}
```

```
package com.formodtohaf.view;

import java.awt.*;
import java.io.*;
import java.util.HashMap;
import javax.swing.JFileChooser;

import jxl.Cell;
```

```
import jxl.CellType;
import jxl.Sheet;
import jxl.Workbook;
import jxl.read.biff.BiffException;

import com.formodtohaf.util.FORMODTOHAF_Utility;
import com.formodtohaf.view.constant.FORMODTOHAF_GetConstHaf;
import com.formodtohaf.view.constant.FORMODTOHAF_GetConstOff;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetDenDep;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetStrVal;



public class FORMODTOHAF_MainPanel extends Panel {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    static Panel p_North;
    public static Panel p_East;
    public static Panel p_Center;
    public static TextField inputValues [] = new TextField[15];
    public static TextArea img = new TextArea(46,140);
    public static Label graphLabel;
    Object rowdata[][]={};

    /**Field Area Name*/
    final static int AREA_FE = 0;
    /**Number of observations*/
    final static int N_OBS = 1 ;
    /**Distance(km)*/
    final static int X_KM = 2;
    /**Observed Anomaly*/
    final static int OBS_ANO = 3;
    /** Angle of the profile */
    final static int ANG_PRO = 4;
    /**Elevations(km)*/
    final static int ELE_KM = 5;



    public FORMODTOHAF_MainPanel(FORMODTOHAF_MainView cm){

        this.setLayout(new BorderLayout());

        p_East = new Panel ();
        p_Center = new Panel();
        p_North = new Panel ();
        graphLabel = new Label("FORWARD GRAVITY MODELING OF MULTIPLE GEOLOGICAL SOURCES",
        Label.CENTER);
        graphLabel.setFont(new Font("Arial", 10, 28));
        p_Center.add(graphLabel);

        for(int i = 0; i < 10; i++){
            inputValues[i] = new TextField();
        }
        FORMODTOHAF_MainPanel.populateNorthPanel();
        FORMODTOHAF_TableView.populateEastPanel(rowdata);
        this.add(p_North, BorderLayout.NORTH);
        p_Center.setSize(1000, 760);
        this.add(p_Center, BorderLayout.CENTER);
        img.setEditable(false);
        p_Center.add(img);
        this.add(p_East, BorderLayout.EAST);
        this.setVisible(true);
        MenuBar mb = new MenuBar();
        cm.setMenuBar(mb);
        Menu file = new Menu("File");
        Menu file2 = new Menu("Model information");
        Menu file3 = new Menu("Analysis");
        MenuItem i1,i2,i3,i5,i6,i7,i8,i9,i11,i21,i10;
        file.add(i1 = new MenuItem("New"));
        file.add(i2 = new MenuItem("Load file"));
```

```
        file.add(i3 = new MenuItem("Save file"));
        file.add(i5 = new MenuItem("Clear"));
        file.add(i6 = new MenuItem("Exit"));
        Menu off  = new Menu("Offset");
        MenuItem off1,off2;
        off.add(off1 = new MenuItem("Constant offset"));
        off.add(off2 = new MenuItem("Variable offset"));
        file2.add(off);
        Menu half  = new Menu("Half strike");
        MenuItem h1,h2;
        half.add(h1 = new MenuItem("Constant half strike"));
        half.add(h2 = new MenuItem("Variable half strike"));
        file2.add(half);
        file2.addSeparator();
        file2.add(i7 = new MenuItem("Formation info"));
        file2.add(i11 = new MenuItem("Density contrast & Depth"));
        file3.add(i9 = new MenuItem("Forward modeling"));
        file3.add(i21 = new MenuItem("Save and Print"));
        Menu help  = new Menu("Help");
        help.add(i10 = new MenuItem("Flow module"));
        help.add(i8 = new MenuItem("Readme"));
        mb.add(file);
        mb.add(file2);
        mb.add(file3);
        mb.add(help);
        i1.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i2.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i3.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i5.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i6.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i7.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i11.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        off1.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        off2.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        h1.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        h2.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i9.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i21.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i10.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
        i8.addActionListener(new com.formodtohaf.control.FORMODTOHAF_Controller());
    }

    public static void populateNorthPanel(){

        p_North.setLayout(new GridLayout(6,3));

        p_North.add(new Label("Area/Profile"));
        p_North.add(inputValues[0]);
        p_North.add(new Label("Number of observations"));
        p_North.add(inputValues[1]);
        p_North.add(new Label("Station interval (km)"));
        p_North.add(inputValues[2]);
        p_North.add(new Label("Observed anomaly (mGal)"));
        p_North.add(inputValues[3]);
        p_North.add(new Label("Angle of the profile (degrees)"));
        p_North.add(inputValues[4]);
        p_North.add(new Label("Elevations(km)"));
        p_North.add(inputValues[5]);

    }

    public static HashMap captureValues(){

        HashMap h_Map = new HashMap();

        try {

            h_Map.put("N_OBS", inputValues[N_OBS].getText());
            h_Map.put("X_KM", inputValues[X_KM].getText());
            h_Map.put("ELE_KM", inputValues[ELE_KM].getText());
            h_Map.put("AREA_FE",inputValues[AREA_FE].getText());
            h_Map.put("ANG_PRO",inputValues[ANG_PRO].getText());
            h_Map.put("OBS_ANO",inputValues[OBS_ANO].getText());
        }
        catch (Exception e) {
```

```
            e.printStackTrace();
        }

        return h_Map;

    }

    public static void clearPanel(TextArea p) {

        Graphics g = p.getGraphics();
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, 1280, 650);
    }


    public static void loadData1()throws IOException  {
        try{

            String current = System.getProperty("user.dir");
            JFileChooser chooser=new  JFileChooser(current);
            int returnVal = chooser.showOpenDialog(null);
            String  ele[],off[],haf[],nfm[],den[],dep[],obs[];
            String eleval="",offval="",hafval="",nfmval="",denval="",depval="",obsval="";
            Workbook w;

            if(returnVal == JFileChooser.APPROVE_OPTION) {
                File f = chooser.getSelectedFile();
                w = Workbook.getWorkbook(f);
                Sheet sheet = w.getSheet(0);
                ele = new String[sheet.getRows()+1];
                off = new String[sheet.getRows()+1];
                haf = new String[sheet.getRows()+1];
                nfm = new String[sheet.getRows()+1];
                den = new String[sheet.getRows()+1];
                dep = new String[sheet.getRows()+1];
                obs = new String[sheet.getRows()+1];

                for (int j = 0; j < sheet.getColumns(); j++) {
                    for (int i = 1; i < sheet.getRows(); i++) {
                        Cell cell = sheet.getCell(j, i);
                        CellType type = cell.getType();
                        if (type == CellType.LABEL) {

                            FORMODTOHAF_MainPanel.inputValues[FORMODTOHAF_MainPanel.AREA_FE].setTe
xt(cell.getContents());

                        }

                        if (type == CellType.NUMBER) {

                            if (j==1){
                                FORMODTOHAF_MainPanel.inputValues[FORMODTOHAF_MainPanel.N_OBS].set
Text(cell.getContents());
                            }

                            if (j==2){
                                FORMODTOHAF_MainPanel.inputValues[FORMODTOHAF_MainPanel.X_KM].setT
ext(cell.getContents());
                            }
                            if (j==3){
                                obs[i] = cell.getContents()+",";
                                obsval = obsval+obs[i];
                            }
                            if (j==4){
                                FORMODTOHAF_MainPanel.inputValues[FORMODTOHAF_MainPanel.ANG_PRO].s
etText(cell.getContents());
                            }

                            if (j==5){
                                ele[i] = cell.getContents()+",";
                                eleval = eleval+ele[i];
                            }

                            if (j==6){
                                haf[i] = cell.getContents()+",";
```

```
                                hafval = hafval+haf[i];;
                            }

                            if (j==7){
                                off[i] = cell.getContents()+",";
                                offval = offval+off[i];;
                            }
                            if (j==8){
                                nfm[i] = cell.getContents()+",";
                                nfmval = nfmval+nfm[i];;
                            }
                            if (j==9){
                                den[i] = cell.getContents()+",";
                                denval = denval+den[i];;
                            }
                            if (j==10){
                                dep[i] = cell.getContents()+",";
                                depval = depval+dep[i];;
                            }


                        }

                    }
                }


                FORMODTOHAF_MainPanel.inputValues[FORMODTOHAF_MainPanel.ELE_KM].setText(""+eleval)
;
                FORMODTOHAF_MainPanel.inputValues[FORMODTOHAF_MainPanel.OBS_ANO].setText(""+obsval
);
                FORMODTOHAF_GetConstHaf.input_strike_km =
FORMODTOHAF_Utility.convertDoubleArray(hafval);
                FORMODTOHAF_GetConstOff.input_y_km =
FORMODTOHAF_Utility.convertDoubleArray(offval);
                FORMODTOHAF_GetStrVal.nfm = FORMODTOHAF_Utility.convertDoubleArray(nfmval);
                FORMODTOHAF_GetDenDep.den = FORMODTOHAF_Utility.convertDoubleArray(denval);
                FORMODTOHAF_GetDenDep.dep = FORMODTOHAF_Utility.convertDoubleArray(depval);

            }
        }
        catch (BiffException e) {
            e.printStackTrace();
        } catch (Exception e) {

            e.printStackTrace();
        }

    }

    public static void clearDefaultValues(){
        inputValues[N_OBS].setText("");
        inputValues[X_KM].setText("");
        inputValues[ELE_KM].setText("");
        inputValues[ANG_PRO].setText("");
        inputValues[AREA_FE].setText("");
        inputValues[OBS_ANO].setText("");
    }
}
```

```
package com.formodtohaf.view;

import java.awt.Button;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Label;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JScrollPane;
import javax.swing.JTable;

import com.formodtohaf.model.FORMODTOHAF_CalculateValues;
```

```
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetDenDep;


public class FORMODTOHAF_DenDep implements ActionListener{
    public static JTable table;
    Button bt  = new Button("Submit");
    public static Object off[][];
    public  void val() {
        off  = new Object[FORMODTOHAF_CalculateValues.input_num_for+1][4];
        FORMODTOHAF_MainPanel.p_Center.removeAll();
        FORMODTOHAF_MainPanel.p_Center.repaint();
        Label label2 = new Label("Column A: Density contrast(gm/cc) ", Label.CENTER);
        label2.setFont(new Font("Arial", 10, 18));
        Label label3 = new Label("Column B: Depth(km) ", Label.CENTER);
        label3.setFont(new Font("Arial", 10, 18));
        FORMODTOHAF_MainPanel.p_Center.add(FORMODTOHAF_MainPanel.graphLabel);
        FORMODTOHAF_MainPanel.p_Center.add(label2);
        FORMODTOHAF_MainPanel.p_Center.add(label3);
        table = new JTable(FORMODTOHAF_CalculateValues.input_num_for, 2);
        table.setPreferredScrollableViewportSize(new Dimension(300,300));
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);
        FORMODTOHAF_MainPanel.p_Center.add(scrollPane);
        FORMODTOHAF_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        FORMODTOHAF_MainPanel.p_Center.validate();
        FORMODTOHAF_MainPanel.p_Center.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            FORMODTOHAF_GetDenDep gfv = new FORMODTOHAF_GetDenDep();
            gfv.getVal();
        }
    }
}
```

```
package com.formodtohaf.view;

import java.applet.Applet;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import java.text.DecimalFormat;


import com.formodtohaf.model.FORMODTOHAF_CalculateValues;
import com.formodtohaf.util.FORMODTOHAF_Utility;

public class FORMODTOHAF_DrawGraph extends Applet {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    float maxX,maxY,maxY1;
    float maxZ,diff;
    double obs[];
    int sep;
    public void drawGraph(Graphics2D g2) {

        g2.setFont(new Font("Arial", 20, 12));
        g2.setColor(Color.BLACK);
        maxX = (float) FORMODTOHAF_CalculateValues.x[FORMODTOHAF_CalculateValues.input_n_obs];
        diff = (float) ( (FORMODTOHAF_CalculateValues.x[2]-FORMODTOHAF_CalculateValues.x[1]) / 2);
        diff = (float) (450 * diff / maxX);
        g2.draw(new Line2D.Float(200-diff, 35,200-diff , 550 + 20));
        g2.drawLine(70, 20, 1040, 20);
        g2.drawLine(1040, 20,1040,600);
        g2.drawLine(70, 20, 70, 600);
        g2.drawLine(70, 600, 1040, 600);
```

```
        g2.drawString("DISTANCE(km)", 383, 318);
        String []a={"A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
        String []b={"D", "E", "P", "T", "H", "(k", "m)"};
        for (int i = 0; i < a.length; i++) {

            g2.drawString(""+a[i], 120, 20 + 60 + ( i * 20 ) );
        }

        for (int i = 0; i < b.length; i++) {

            g2.drawString(""+b[i], 120, 20 + 350 + ( i * 20 ) );
        }

    }

    public void plot(Graphics2D g){
        g.setFont(new Font("Arial", 20, 12));
        g.setColor(Color.black);
        maxX = (float) FORMODTOHAF_Utility.findMaximumNumber1(FORMODTOHAF_CalculateValues.x);
        diff = (float) ( (FORMODTOHAF_CalculateValues.x[2]-FORMODTOHAF_CalculateValues.x[1]) / 2);
        diff = (float) (450 * diff / maxX);
        double inidep = FORMODTOHAF_Utility.findMaximumNumber1(FORMODTOHAF_CalculateValues.dep);
        double pos[] = new double[FORMODTOHAF_CalculateValues.input_n_obs];
        double neg[] = new double[FORMODTOHAF_CalculateValues.input_n_obs];
        double posob[] = new double[FORMODTOHAF_CalculateValues.input_n_obs];
        double negob[] = new double[FORMODTOHAF_CalculateValues.input_n_obs];
        int m=1,n = 1,u=1,v=1;
        for (int k = 1; k <= FORMODTOHAF_CalculateValues.input_n_obs; k++) {

            if(FORMODTOHAF_CalculateValues.o_gc[k] > 0){
                pos[m] = FORMODTOHAF_CalculateValues.o_gc[k];

                m = m + 1;
            }
            else{
                neg[n] = FORMODTOHAF_CalculateValues.o_gc[k];
                n = n + 1;
            }
            if(FORMODTOHAF_CalculateValues.input_obs[k] > 0){
                posob[u] = FORMODTOHAF_CalculateValues.input_obs[k];

                u = u + 1;
            }
            else{
                negob[v] = FORMODTOHAF_CalculateValues.input_obs[k];
                v = v + 1;
            }
        }
        float ycal = FORMODTOHAF_Utility.findMaximumNumber(neg);
        float yobs = FORMODTOHAF_Utility.findMaximumNumber(negob);
        if(Math.abs(ycal)>Math.abs(yobs))
            maxY = ycal;
        else
            maxY = yobs;

        float posy = (float) FORMODTOHAF_Utility.findMaximumNumber1(pos);
        float posmaxy = (float) FORMODTOHAF_Utility.findMaximumNumber1(posob);
        if(posy>posmaxy)
            maxY1 = posy;
        else
            maxY1 = posmaxy;
        maxZ = (float) inidep;
        DecimalFormat df = new DecimalFormat("0.#");
        g.drawString("0",170-diff,80);
        g.drawString("0", 170-diff,330);
        g.drawString("|", 650, 318);
        g.drawString(""+df.format(FORMODTOHAF_CalculateValues.x[FORMODTOHAF_CalculateValues.input_
        n_obs]) ,650 ,305);

        int zInterval=50;
        float xInterval=(float)
        (FORMODTOHAF_CalculateValues.x[FORMODTOHAF_CalculateValues.input_n_obs]/5);
        float xplot=0;
        for (float x = xInterval, j =1; x < 600; x+=xInterval)
        {
```

```
            xplot=xplot+xInterval;
            if(j>4)
                break;
            g.drawString("|",(float) (200+(450*x/maxX)), 318);
            g.drawString("" + df.format(xplot), (float) (200+(450*x/maxX))-3, 305);
            j++;
        }

        DecimalFormat f = new DecimalFormat("0.##");
        g.drawString(""+f.format(maxY1),165-diff, 45);
        float point =maxZ/5 ;
        for(int x = zInterval+250,j =1; x < 550; x+=zInterval)
        {
            g.drawString("-",196-diff,50+x+20);
            g.drawString("" +df.format(point*j),165-diff,50+x+20);
            j++;
        }
    }


    public void plotXYCoordinates (Graphics2D g2){

        float prevx = 200;
        float prevy = 0;
        float xpoint = 0;
        float ypoint = 0;
        float gypoint = 0;

        if (FORMODTOHAF_CalculateValues.o_gc[1] < 0){
            prevy = 70 + (float)( ( 250 * (FORMODTOHAF_CalculateValues.o_gc[1]) / maxY ) );
        }
        else{

            prevy = 70 - (float)( ( 35 * (FORMODTOHAF_CalculateValues.o_gc[1]) / maxY1 ) );
        }

        for (int k = 1; k <= FORMODTOHAF_CalculateValues.input_n_obs; k++) {

            xpoint = (float)( 450 * FORMODTOHAF_CalculateValues.x[k] / maxX);

            if (FORMODTOHAF_CalculateValues.o_gc[k] < 0){

                ypoint = 70 + (float)( ( 250 * (FORMODTOHAF_CalculateValues.o_gc[k]) / maxY ) );
            }
            else{

                ypoint = 70 - (float)( ( 35 * (FORMODTOHAF_CalculateValues.o_gc[k]) / maxY1 ) );
            }
            if (FORMODTOHAF_CalculateValues.input_obs[k] < 0){

                gypoint = 70 + (float)( ( 250 * (FORMODTOHAF_CalculateValues.input_obs[k]) / maxY
) );
            }
            else{

                gypoint = 70 - (float)( ( 35 * (FORMODTOHAF_CalculateValues.input_obs[k]) / maxY1
) );
            }


            g2.setFont(new Font("Arial", 20, 20));
            g2.setColor(Color.BLACK);
            g2.draw(new Line2D.Float(prevx, prevy, 200 + xpoint,  ypoint  ));
            g2.setColor(Color.BLUE);
            g2.setFont(new Font("Arial", 20, 40));
            g2.drawString(".", 200 + xpoint -6 ,  gypoint + 3);

            prevx = 200 + xpoint;
            prevy = ypoint ;

        }
    }

    public void plotZCoordinates (Graphics2D g2){
```

```
        float xpoint=0;
        float xpoint1=0;
        float zpoint=0;
        int u =1;
        float points = maxY/5;
        int yInterval = 50;
        g2.drawString(""+(int)maxY  ,165-diff,320);
        for (int x = yInterval, j =1; x < 250; x+=yInterval)
        {
            g2.drawString("-",196-diff,50+x+20);
            g2.drawString("" + (int)(points*j), 165-diff,50+x+20);
            j++;
        }

        for (int k = 1; k <=FORMODTOHAF_CalculateValues.input_n_obs; k++)
        {
            if(k<FORMODTOHAF_CalculateValues.input_n_obs){
                diff = (float) (
(FORMODTOHAF_CalculateValues.x[k+1]-FORMODTOHAF_CalculateValues.x[k]) / 2);
                diff = (float) (450 * diff / maxX); }
            for(int j = 1;j<=FORMODTOHAF_CalculateValues.nfm[k];j++){
                xpoint = (float) (450 * FORMODTOHAF_CalculateValues.x[k]/ maxX);
                if (k == 1){
                    xpoint1 = (float) (450 * FORMODTOHAF_CalculateValues.x[k+1]/ maxX);
                    g2.setColor(Color.black);
                    zpoint =(float) (250 * FORMODTOHAF_CalculateValues.dep[u] / maxZ);
                    g2.draw(new Rectangle2D.Float(200-diff, 320,xpoint1-xpoint, zpoint ));
                    g2.setFont(new Font("Arial", 20,9 ));
                    g2.drawString(""+FORMODTOHAF_CalculateValues.den[u],200+xpoint-diff,320+zpoint
);
                }
                else{
                    if(k<FORMODTOHAF_CalculateValues.input_n_obs) {
                        xpoint1 = (float) (450 * FORMODTOHAF_CalculateValues.x[k+1]/ maxX);
                    }
                    zpoint =(float) (250 * FORMODTOHAF_CalculateValues.dep[u]/ maxZ);

                    g2.setColor(Color.black);
                    g2.draw(new Rectangle2D.Float(200+xpoint-diff ,320,xpoint1-xpoint, zpoint ));
                    g2.setFont(new Font("Arial", 20,9 ));
                    g2.drawString(""+FORMODTOHAF_CalculateValues.den[u],200+xpoint-diff,320+zpoint
);
                }
                if(k==FORMODTOHAF_CalculateValues.input_n_obs){
                    xpoint1 = (float) (450 * FORMODTOHAF_CalculateValues.x[k-1]/ maxX);
                    zpoint =(float) (250 * FORMODTOHAF_CalculateValues.dep[u]/ maxZ);
                    g2.draw(new Rectangle2D.Float(200+xpoint-diff, 320,xpoint-xpoint1, zpoint ));

                }

                u = u+1;

            }
        }
        g2.setColor(Color.BLACK);

    }

    public void plotZCoordinatesele (Graphics2D g2) {

        g2.setFont(new Font("Arial", 20, 12));
        DecimalFormat f = new DecimalFormat("0.#");
        int yInterval=50;
        float points = maxY / 5;
        for (int x = yInterval, j = 1; x < 250; x+=yInterval){

            g2.drawString("-", 196-diff, 50 + x + 24);
            g2.drawString("" + f.format(points * j), 165-diff, 50 + x + 20);
            j++;
        }

        float maxEle = (float)
        FORMODTOHAF_Utility.findMaximumNumber1(FORMODTOHAF_CalculateValues.input_ele_km);
        maxEle = Math.abs(maxEle);
        float xpoint = 0;
```

```
        float prevx = 200-diff;
        float elepoint[] = new float[FORMODTOHAF_CalculateValues.input_n_obs+1];
        float preelepoint[] = new float[FORMODTOHAF_CalculateValues.input_n_obs+1];
        float dis[] = new float[FORMODTOHAF_CalculateValues.input_n_obs+1];
        float predis[] = new float[FORMODTOHAF_CalculateValues.input_n_obs+1];
        float preele = 320 - (float)( ( 25 * Math.abs(FORMODTOHAF_CalculateValues.input_ele_km[1])
        / maxEle ) );
        float xpoint1 = 0;
        float zpoint = 0;
        int u = 1;

        for (int k = 1; k <= FORMODTOHAF_CalculateValues.input_n_obs; k++) {
            for(int j = 1;j<=FORMODTOHAF_CalculateValues.nfm[k];j++){

                diff = (float) ( FORMODTOHAF_CalculateValues.input_ddx_km / 2);
                diff = (float) (450 * diff / maxX);
                xpoint = (float) (450 * FORMODTOHAF_CalculateValues.x[k] / maxX);
                if (k == 1){
                    xpoint1 = (float) (450 * FORMODTOHAF_CalculateValues.x[k+1]/ maxX);
                    g2.setColor(Color.black);
                    zpoint =(float) (275 * FORMODTOHAF_CalculateValues.dep[u] / maxZ);
                    g2.draw(new Rectangle2D.Float(200-diff, preele,xpoint1-xpoint, zpoint ));
                    g2.setFont(new Font("Arial", 20,9 ));
                    g2.drawString(""+FORMODTOHAF_CalculateValues.den[u],200+xpoint-diff,295+zpoint
);
                }
                else{
                    if(k<FORMODTOHAF_CalculateValues.input_n_obs) {
                        xpoint1 = (float) (450 * FORMODTOHAF_CalculateValues.x[k+1] / maxX);
                    }
                    zpoint =(float) (275 * FORMODTOHAF_CalculateValues.dep[u] / maxZ);
                    g2.draw(new Rectangle2D.Float(200+xpoint-diff ,295,xpoint1-xpoint, zpoint ));
                    g2.setFont(new Font("Arial", 20,9 ));
                    g2.drawString(""+FORMODTOHAF_CalculateValues.den[u],200+xpoint-diff,295+zpoint
);
                }
                if(k==FORMODTOHAF_CalculateValues.input_n_obs){
                    xpoint1 = (float) (450 * FORMODTOHAF_CalculateValues.x[k-1]/ maxX);
                    zpoint =(float) (275 * FORMODTOHAF_CalculateValues.dep[u]/ maxZ);
                    g2.draw(new Rectangle2D.Float(200+xpoint-diff, 295,xpoint-xpoint1, zpoint ));
                    g2.setFont(new Font("Arial", 20,9 ));
                    g2.drawString(""+FORMODTOHAF_CalculateValues.den[u],200+xpoint-diff,295+zpoint
);

                }

                u = u+1;
            }

        }
        for (int k = 1; k <= FORMODTOHAF_CalculateValues.input_n_obs; k++) {
            g2.setColor(Color.BLACK);
            xpoint = (float) (450 * FORMODTOHAF_CalculateValues.x[k] / maxX);
            dis[k] = xpoint;
            elepoint[k] = 320 - (float) (25 *
            Math.abs(FORMODTOHAF_CalculateValues.input_ele_km[k]) / maxEle);
            preelepoint[k]= preele;
            predis[k]= prevx;
            g2.draw(new Line2D.Float(prevx, preele, (float)200 + xpoint, elepoint[k]));
            prevx = 200+xpoint;
            preele = elepoint[k];
        }
        for(float i = (float)1.0; i<=25;){
            for (int k = 1; k <= FORMODTOHAF_CalculateValues.input_n_obs; k++) {
                g2.setColor(Color.WHITE);
                g2.draw(new Line2D.Float(predis[k], preelepoint[k]-i, (float)200+dis[k],
elepoint[k]-i));
            }
            i=(float) (i+0.5);

        }
        g2.setColor(Color.BLACK);

    }
```

```
    /** Index of Graph*/
    public void idex(Graphics2D g)
    {
        g.setFont(new Font("Arial", 20, 40));
        g.setColor(Color.BLUE);
        g.drawString("....",730,87);
        g.setFont(new Font("Arial", 20, 20));
        g.drawString(": Observed anomalies",780,90);
        g.setColor(Color.BLACK);
        g.drawString("____:",730,127);
        g.drawString("Theoretical anomalies",780,130);
        g.drawRect(730, 160, 35, 10);
        g.setColor(Color.BLACK);
        g.drawString(":Depth Structure", 780, 170);
    }
}
```

```
package com.formodtohaf.view;


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import com.formodtohaf.model.FORMODTOHAF_CalculateValues;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetHalfstrikeVal;

public class FORMODTOHAF_HalfstrikeVal implements ActionListener {
    public static JTable table;
    Button bt  = new Button("Submit");
    public static Object haf[][];
    public  void val() {
        haf  = new Object[FORMODTOHAF_CalculateValues.input_n_obs+1][2];
        com.formodtohaf.view.FORMODTOHAF_MainPanel.p_Center.removeAll();
        com.formodtohaf.view.FORMODTOHAF_MainPanel.p_Center.repaint();
        Label label = new Label("Column A: Prism number ", Label.CENTER);
        label.setFont(new Font("Arial", 10, 18));
        Label label1 = new Label("Column B: Halfstrike  ", Label.CENTER);
        label1.setFont(new Font("Arial", 10, 18));
        FORMODTOHAF_MainPanel.p_Center.add(FORMODTOHAF_MainPanel.graphLabel);
        FORMODTOHAF_MainPanel.p_Center.add(label);
        FORMODTOHAF_MainPanel.p_Center.add(label1);

        table = new JTable(FORMODTOHAF_CalculateValues.input_n_obs, 2);
        table.setPreferredScrollableViewportSize(new Dimension(300,300));

        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);

        FORMODTOHAF_MainPanel.p_Center.add(scrollPane);
        FORMODTOHAF_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        FORMODTOHAF_MainPanel.p_Center.validate();
        com.formodtohaf.view.FORMODTOHAF_MainPanel.p_Center.setVisible(true);

    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            FORMODTOHAF_GetHalfstrikeVal ghv = new FORMODTOHAF_GetHalfstrikeVal();
            ghv.getVal();
        }
    }
}
```

```
package com.formodtohaf.view;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JScrollPane;
```

```
import javax.swing.JTable;

import com.formodtohaf.model.FORMODTOHAF_CalculateValues;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetOffsetVal;

public class FORMODTOHAF_OffsetVal implements ActionListener {
    public static JTable table;
    Button bt  = new Button("Submit");
    public static Object off[][];
    public  void val() {
        off  = new Object[FORMODTOHAF_CalculateValues.input_n_obs+1][2];
        FORMODTOHAF_MainPanel.p_Center.removeAll();
        FORMODTOHAF_MainPanel.p_Center.repaint();
        Label label = new Label("Column A: Prism number ", Label.CENTER);
        label.setFont(new Font("Arial", 10, 18));
        Label label1 = new Label("Column B: Offset value ", Label.CENTER);
        label1.setFont(new Font("Arial", 10, 18));
        FORMODTOHAF_MainPanel.p_Center.add(FORMODTOHAF_MainPanel.graphLabel);
        FORMODTOHAF_MainPanel.p_Center.add(label);
        FORMODTOHAF_MainPanel.p_Center.add(label1);
        table = new JTable(FORMODTOHAF_CalculateValues.input_n_obs, 2);
        table.setPreferredScrollableViewportSize(new Dimension(300,300));
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);
        FORMODTOHAF_MainPanel.p_Center.add(scrollPane);
        FORMODTOHAF_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
        FORMODTOHAF_MainPanel.p_Center.validate();
        FORMODTOHAF_MainPanel.p_Center.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            FORMODTOHAF_GetOffsetVal gfv = new FORMODTOHAF_GetOffsetVal();
            gfv.getVal();
        }
    }
}
-----------------------------------------------------------------------------------------------------

package com.formodtohaf.view;

import java.awt.Button;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Label;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import com.formodtohaf.model.FORMODTOHAF_CalculateValues;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetStrVal;

public class FORMODTOHAF_StrVal implements ActionListener {
    public static JTable table;
    Button bt  = new Button("Submit");
    public static Object off[][];
    public  void val() {
        off  = new Object[FORMODTOHAF_CalculateValues.input_num_for+1][4];
        FORMODTOHAF_MainPanel.p_Center.removeAll();
        FORMODTOHAF_MainPanel.p_Center.repaint();
        Label label = new Label("Column A: Prism number ", Label.CENTER);
        label.setFont(new Font("Arial", 10, 18));
        Label label1 = new Label("Column B: Number of formations in each prism            ",
        Label.CENTER);
        label1.setFont(new Font("Arial", 10, 18));
        FORMODTOHAF_MainPanel.p_Center.add(FORMODTOHAF_MainPanel.graphLabel);
        FORMODTOHAF_MainPanel.p_Center.add(label);
        FORMODTOHAF_MainPanel.p_Center.add(label1);
        table = new JTable(FORMODTOHAF_CalculateValues.input_n_obs, 2);
        table.setPreferredScrollableViewportSize(new Dimension(300,300));
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);
        FORMODTOHAF_MainPanel.p_Center.add(scrollPane);
        FORMODTOHAF_MainPanel.p_Center.add(bt);
        bt.addActionListener(this);
```

```
        FORMODTOHAF_MainPanel.p_Center.validate();
        FORMODTOHAF_MainPanel.p_Center.setVisible(true);
    }
    public void actionPerformed(ActionEvent ae){
        if(ae.getActionCommand().equals("Submit")){
            FORMODTOHAF_GetStrVal gfv = new FORMODTOHAF_GetStrVal();
            gfv.getVal();
        }
    }
}
```

```
package com.formodtohaf.view;

import java.awt.*;
import javax.swing.JScrollPane;
import javax.swing.JTable;


public class FORMODTOHAF_TableView extends Panel{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public static  void populateEastPanel(Object rowData[][]) {
        FORMODTOHAF_MainPanel.p_East.removeAll();
        FORMODTOHAF_MainPanel.p_East.setLayout(new GridLayout(1,1));
        Object columnNames[] = {"Distance(km)","Observed anamolies (mGal)","Theoritical anamolies
        (mGal)"};
        JTable table = new JTable(rowData, columnNames);
        table.setPreferredScrollableViewportSize(new Dimension(300,550));
        JScrollPane scrollPane = new JScrollPane(table);
        scrollPane.setAutoscrolls(true);
        FORMODTOHAF_MainPanel.p_East.add(scrollPane);
        FORMODTOHAF_MainPanel.p_East.validate();
        FORMODTOHAF_MainPanel.p_East.setVisible(true);
    }


}
```

```
package com.formodtohaf.model;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileOutputStream;
import java.text.DecimalFormat;
import java.util.HashMap;
import javax.imageio.ImageIO;

import com.formodtohaf.util.FORMODTOHAF_Utility;
import com.formodtohaf.view.FORMODTOHAF_MainPanel;
import com.formodtohaf.view.constant.FORMODTOHAF_GetConstHaf;
import com.formodtohaf.view.constant.FORMODTOHAF_GetConstOff;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetDenDep;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetStrVal;

public class FORMODTOHAF_CalculateValues {
    public static int input_n_obs,input_num_for = 0;
    public static double input_ddx_km,input_ang_pro=0;
    public static double x[],input_ele_km[],o_dep[],input_obs[]= null;
    public static double input_strike_km [],input_y_km[]=null;
    public static Object obj[][] = null;
    public static double nfm[],den[],dep[] ;
    public static double []o_gc ,in_num[];
    static BufferedImage image;
    public static String input_area_name="";
```

```
public void getAnamolyValues(HashMap h_Map) {

    try {
        input_n_obs = FORMODTOHAF_Utility.convertInteger((String)h_Map.get("N_OBS"));
        input_ddx_km = FORMODTOHAF_Utility.convertDouble((String)h_Map.get("X_KM"));
        input_ele_km = FORMODTOHAF_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
        input_area_name = FORMODTOHAF_Utility.convertString((String)h_Map.get("AREA_FE"));
        input_obs = FORMODTOHAF_Utility.convertDoubleArray((String)h_Map.get("OBS_ANO"));
        input_ang_pro = FORMODTOHAF_Utility.convertDouble((String)h_Map.get("ANG_PRO"));
    }
    catch(Exception e) {

    }
    den = FORMODTOHAF_GetDenDep.den;
    dep = FORMODTOHAF_GetDenDep.dep;
    input_y_km = FORMODTOHAF_GetConstOff.input_y_km;
    input_strike_km = FORMODTOHAF_GetConstHaf.input_strike_km;
    nfm =  FORMODTOHAF_GetStrVal.nfm;
    int s=0;
    int q=0;
    for(int i =1;i <= input_n_obs;i++){
        s = (int) nfm[i];
        q = q+s;

    }
    input_num_for =q;
}
public void cal(){
    int v =1;
    double hafthc,ggr=0;
    double yy[] =  new double[3];
    double dnn[] =  new double[input_num_for+1];
    double zrr[] =  new double[input_num_for+1];
    double dc[][] = new double[input_num_for+1][input_num_for+1];
    double zr[][] = new double[input_num_for+1][input_num_for+1];
    x = new double[input_n_obs+1];
    o_gc  = new double[input_n_obs + 1];

    for(int i = 1;i <= input_n_obs; i++){

        for(int j =1;j<=nfm[i];j++){
            dc[i][j] = den[v];
            zr[i][j] = dep[v];
            v=v+1;
        }

    }
    input_ang_pro = input_ang_pro* 3.14159265/180;
    for(int i =1;i <= input_n_obs;i++){
        x[i]= (i-1)* input_ddx_km *Math.cos(input_ang_pro);
    }
    hafthc = (x[2]-x[1])/2;
    for(int i =1;i <= input_n_obs;i++){
        for(int j =1;j <= input_n_obs;j++){
            double xx = x[i]- x[j];
            yy[1]= input_strike_km[j]+input_y_km [j];
            yy[2]= input_strike_km[j]-input_y_km [j];
            for(int k =1;k<=nfm[j];k++){
                dnn[k] = dc[j][k];
                zrr[k] = zr[j][k];
            }
            double nfmm = nfm[j];
            double ele = input_ele_km[j];
            ggr = gprism(xx,yy,zrr,ele,input_ddx_km,hafthc,dnn,nfmm);
            o_gc[i] = o_gc[i]+ggr;
        }

    }

    setGraphValues(x,input_obs,o_gc);
    drawGraph();
}
public static double gprism(double xx,double yy[],double []zrr,double ele,double ddx,double
```

```
    hafthc,double []dnn,double nfmm) {

        double gprm =0;
        int effd = 0;
        double []z;;
        double []gs;
        double []gg = new double[5];
        int kk=1;
        double zt = 0.000000001;
        do{
            double dx = ddx / 10;
            double zb = zrr[kk];
            effd = effd+1;
            double zdif = zb - zt;
            int nd = (int)(zdif/dx)+1;
            int n1 = nd / 2;
            if (nd - (2 * n1 ) < 0 || nd - ( 2 * n1 ) > 0) {

                nd = nd + 1;
            }
            double dz = zdif / nd;
            int N2 = nd + 1;
            z = new double[N2+1];
            gs = new double[N2+1];

            for (int JZ = 1; JZ <= N2; JZ++) {
                z[JZ] = zt + dz*(JZ-1);
            }

            for (int JZ = 1; JZ <= N2; JZ++) {

                double  den = dnn[effd];
                double t1 = 13.3333 * den;
                for (int kl = 1; kl <=2; kl++) {

                    double effy = yy[kl];
                    double c1 = xx-hafthc;
                    double ter1 = Math.atan((effy*c1)/((z[JZ]-ele)*Math.sqrt(Math.pow(c1,2) +
Math.pow(effy,2) + Math.pow(z[JZ]-ele,2))));
                    double c2 = xx+hafthc;
                    double ter2 =
Math.atan((effy*c2)/((z[JZ]-ele)*Math.sqrt(Math.pow(c2,2)+Math.pow(effy,2)+Mat
h.pow(z[JZ]-ele,2))));
                    gg[kl] = t1*(ter2-ter1);
                }
                gs[JZ]=(gg[1]+gg[2])/2;
            }
            for (int  kl = 1; kl <= N2-1; kl++) {
                double g1 = gs[kl];
                double g2 = gs[kl+1];
                double dy3 = z[kl+1] - z[kl];
                gprm = gprm + EXPNT(g1,g2,dy3);
            }
            kk=kk+1;
            if(kk-nfmm<=0)
                zt=zrr[kk-1];
            else
                break;
        }while(kk-nfmm<=0);
        return gprm;
    }


    public static double  EXPNT(double g1,double g2,double dx){

        double c;
        double expnt;
        if (g1 == 0)
            g1 = 0.0001;
        if(g2==0)
            g2=0.0001;
        if( g2 / g1 < 0) {
            double x = Math.abs(g1)/(Math.abs(g1)+Math.abs(g2));
            double  g = 0.00001 * g1/ Math.abs(g1);
            c = Math.log(g/g1);
```

```
            double f1 = ( g - g1) / c;
            g = 0.00001 * g2/Math.abs(g2);
            c = Math.log(g2/g);
            double f2 = (g2-g)/c;
            expnt = f1*x+f2*(1-x);
        }
        else {
            c = Math.log(g2/g1);
        }

        if(Math.abs(c) <= 0.0001){
            expnt=g1;
        }
        else {
            expnt=(g2-g1)/c;
        }
        expnt = dx*expnt;
        return expnt;

    }
    public static void  setGraphValues(double []dis,double []gobs ,double []gcal) {

        obj = new Object[input_n_obs+21][4];
        DecimalFormat df =new DecimalFormat("0.###");
        for (int K = 1; K <= input_n_obs; K++){
            obj[K][0] = "" + df.format(dis[K]);
            obj[K][1] = "" + df.format(gobs[K]);
            obj[K][2] = "" + df.format(gcal[K]);

        }

    }
    public static void drawGraph(){
        final com.formodtohaf.view.FORMODTOHAF_DrawGraph dg = new
        com.formodtohaf.view.FORMODTOHAF_DrawGraph();
        try
        {
            int width = 980;
            int height = 650;
            BufferedImage buffer = new BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);

            Graphics g1= buffer.createGraphics();
            g1.setColor(Color.WHITE);
            g1.fillRect(0,0,width,height);
            Graphics2D g2 = (Graphics2D)g1 ;
            dg.plot(g2);
            dg.plotXYCoordinates(g2);
            dg.drawGraph(g2);
            float maxEle = (float)
            FORMODTOHAF_Utility.findMaximumNumber1(FORMODTOHAF_CalculateValues.input_ele_km);
            if(maxEle==0)
                dg.plotZCoordinates(g2);
            else
                dg.plotZCoordinatesele(g2);
            dg.plot(g2);
            dg.idex(g2);

            FileOutputStream os = new FileOutputStream(
            FORMODTOHAF_CalculateValues.input_area_name +".jpg");
            ImageIO.write(buffer, "jpg", os);
            os.close();

            String path =  FORMODTOHAF_CalculateValues.input_area_name  +".jpg";
            image = ImageIO.read(new File(path));

            Graphics g_image = FORMODTOHAF_MainPanel.img.getGraphics();
            g_image.drawImage(image, -70,-20,image.getWidth(), image.getHeight(),dg);
            MouseListener ml3 = new MouseAdapter(){
                public void mouseClicked(MouseEvent e){
                    Graphics g_image = FORMODTOHAF_MainPanel.img.getGraphics();
                    g_image.drawImage(image, -70,-20,image.getWidth(), image.getHeight(),dg);
                }
            };
            FORMODTOHAF_MainPanel.img.addMouseListener(ml3);
```

```
        }
        catch (Exception e2) {

            e2.printStackTrace();
        }

    }
}
```

```
package com.formodtohaf.control;

import java.awt.Color;
import java.awt.Desktop;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

import jxl.Workbook;
import jxl.format.Colour;
import jxl.write.Label;
import jxl.write.Number;
import jxl.write.WritableCell;
import jxl.write.WritableCellFormat;
import jxl.write.WritableFont;
import jxl.write.WritableSheet;
import jxl.write.WritableWorkbook;
import jxl.write.WriteException;
import jxl.write.biff.RowsExceededException;

import com.formodtohaf.model.FORMODTOHAF_CalculateValues;
import com.formodtohaf.view.FORMODTOHAF_DenDep;
import com.formodtohaf.view.FORMODTOHAF_HalfstrikeVal;
import com.formodtohaf.view.FORMODTOHAF_MainPanel;
import com.formodtohaf.view.FORMODTOHAF_OffsetVal;
import com.formodtohaf.view.FORMODTOHAF_StrVal;
import com.formodtohaf.view.FORMODTOHAF_TableView;
import com.formodtohaf.view.constant.FORMODTOHAF_GetConstHaf;
import com.formodtohaf.view.constant.FORMODTOHAF_GetConstOff;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetDenDep;
import com.formodtohaf.view.getvalues.FORMODTOHAF_GetStrVal;


public class FORMODTOHAF_Controller implements ActionListener{


    com.formodtohaf.model.FORMODTOHAF_CalculateValues c = new
    com.formodtohaf.model.FORMODTOHAF_CalculateValues();
    Object rowdata[][]={};
    public static boolean success = false;

    public void actionPerformed(ActionEvent ae) {



        if(ae.getActionCommand().equals("Formation info")){

            try{
                FORMODTOHAF_TableView.populateEastPanel(rowdata);
                c.getAnamolyValues(FORMODTOHAF_MainPanel.captureValues());
                FORMODTOHAF_StrVal ov = new FORMODTOHAF_StrVal();
                ov.val();
                DecimalFormat df = new DecimalFormat("0.##");

                for (int i = 1;i <= FORMODTOHAF_CalculateValues.input_n_obs; i++){
                    for (int j = 0;j < 2; j++){
                        FORMODTOHAF_StrVal.off[i-1][1]= df.format(FORMODTOHAF_GetStrVal.nfm[i]);
                        FORMODTOHAF_StrVal.off[i-1][0]= df.format(i);
```

```
                        FORMODTOHAF_StrVal.table.setValueAt(FORMODTOHAF_StrVal.off[i-1][j],i-1,
j);
                    }
                }


            }
            catch(Exception e){

            }
        }else if(ae.getActionCommand().equals("Density contrast & Depth")){

            try{
                FORMODTOHAF_TableView.populateEastPanel(rowdata);
                c.getAnamolyValues(FORMODTOHAF_MainPanel.captureValues());
                FORMODTOHAF_DenDep ov = new FORMODTOHAF_DenDep();
                ov.val();
                DecimalFormat df = new DecimalFormat("0.##");


                for (int i = 1;i <= FORMODTOHAF_CalculateValues.input_num_for; i++){
                    for (int j = 0;j < 2; j++){
                        FORMODTOHAF_DenDep.off[i-1][0]= df.format(FORMODTOHAF_GetDenDep.den[i]);
                        FORMODTOHAF_DenDep.off[i-1][1]= df.format(FORMODTOHAF_GetDenDep.dep[i]);
                        FORMODTOHAF_DenDep.table.setValueAt(FORMODTOHAF_DenDep.off[i-1][j],i-1,
j);
                    }
                }

            }
            catch(Exception e){

            }
        }else if(ae.getActionCommand().equals("Forward modeling")) {

            FORMODTOHAF_MainPanel.p_Center.removeAll();
            FORMODTOHAF_MainPanel.p_Center.add(FORMODTOHAF_MainPanel.graphLabel);
            FORMODTOHAF_MainPanel.p_Center.add(FORMODTOHAF_MainPanel.img);
            FORMODTOHAF_MainPanel.img.setEditable(false);
            FORMODTOHAF_MainPanel.img.setBackground(Color.WHITE);
            FORMODTOHAF_MainPanel.p_Center.validate();
            c.getAnamolyValues(com.formodtohaf.view.FORMODTOHAF_MainPanel.captureValues());
            c.cal();
            com.formodtohaf.view.FORMODTOHAF_TableView.populateEastPanel(FORMODTOHAF_CalculateValu
            es.obj);
            com.formodtohaf.view.FORMODTOHAF_MainPanel.p_East.repaint();
            com.formodtohaf.view.FORMODTOHAF_MainView mv = new
            com.formodtohaf.view.FORMODTOHAF_MainView();

            mv.setResizable(true);

        }else if(ae.getActionCommand().equals("Save and Print")){

            try{
                String current = System.getProperty("user.dir");
                File img_file = new File(FORMODTOHAF_CalculateValues.input_area_name+".jpg");
                JFileChooser saveFile = new JFileChooser(current);
                File OutFile = saveFile.getSelectedFile();
                FileWriter myWriter = null;
                if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION){

                    OutFile = saveFile.getSelectedFile();

                    if (OutFile.canWrite() || !OutFile.exists()){

                        File dir = new File(OutFile.getParent());
                        success = img_file.renameTo(new File(dir,img_file.getName()));
                        System.out.println("Save successful"+success);
                        myWriter = new FileWriter(OutFile+".html");
                        myWriter.write(" </table> </td> <td> <img src = '"+
FORMODTOHAF_CalculateValues.input_area_name +".jpg'></td></tr></table>");
                        myWriter.write("<html> <Body onLoad = \"window.print()\"><table> <tr>
<td>" +
                                "<table border = 1> <tr> <th colspan = 4>LOCATION:-
"+FORMODTOHAF_CalculateValues.input_area_name+"</th> </tr>");
```

```
                        DecimalFormat d = new DecimalFormat("0.###");
                        myWriter.write("<tr > <th>Distance (km) </th><th>Observed anomalies
</th><th> Theoretical anamolies (mGal) </th>  </tr>");

                        for ( int K = 1; K <= FORMODTOHAF_CalculateValues.input_n_obs; K++){

                            myWriter.write("<tr> <td>" +
d.format(FORMODTOHAF_CalculateValues.x[K])+"<td>"+d.format(FORMODTOHAF
_CalculateValues.input_obs[K])+"<td>"+d.format(FORMODTOHAF_CalculateVa
lues.o_gc[K])+"</td></tr>");
                        }
                        myWriter.write("</table>");

                        myWriter.close();
                    }
                }
                else
                {
                    //pops up error message

                }

            }
            catch(Exception e1) {

                e1.printStackTrace();
            }
        }else if(ae.getActionCommand().equals("Load file")){

            try {
                FORMODTOHAF_MainPanel.loadData1();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }else if(ae.getActionCommand().equals("Clear")||ae.getActionCommand().equals("New")){

            FORMODTOHAF_MainPanel.clearDefaultValues();
            FORMODTOHAF_MainPanel.clearPanel(FORMODTOHAF_MainPanel.img);
            com.formodtohaf.view.FORMODTOHAF_TableView.populateEastPanel(rowdata);

        }else if(ae.getActionCommand().equals("Save file")){

            String current = System.getProperty("user.dir");
            JFileChooser saveFile = new JFileChooser(current);
            File OutFile = saveFile.getSelectedFile();

            if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION){

                OutFile = saveFile.getSelectedFile();

                if (OutFile.canWrite() || !OutFile.exists()){


                    try {

                        WritableWorkbook workbook = Workbook.createWorkbook(new
File(OutFile+".xls"));
                        WritableSheet sheet = workbook.createSheet("Page1", 0);

                        String Header[] = new String[20];
                        Header[0] = "Area Name ";
                        Header[1] = "Number of Observations";
                        Header[2] = "Station spacing";
                        Header[3] = "Observed anomalies";
                        Header[4] = "Angle of Profile";
                        Header[5] = "Elevations";
                        Header[6] = "Half-strike";
                        Header[7] = "Offset";
                        Header[8] = "Number of formations in each prism";
                        Header[9] = "Densities";
                        Header[10] = "Depths";

                        WritableCellFormat cellFormat = new WritableCellFormat();
                        WritableFont font = new WritableFont(WritableFont.ARIAL);
```

```
                font.setColour(Colour.BLACK);
                cellFormat.setFont(font);

                for (int i = 0; i < Header.length; i++) {
                    Label label = new Label(i, 0, Header[i]);
                    sheet.addCell(label);
                    WritableCell cell = sheet.getWritableCell(i, 0);
                    cell.setCellFormat(cellFormat);

                }

                Label label = new Label(0,
1,FORMODTOHAF_CalculateValues.input_area_name.toString() );
                sheet.addCell(label);

                double nobs = FORMODTOHAF_CalculateValues.input_n_obs;
                Number label2 = new Number(1, 1,nobs );
                sheet.addCell(label2);

                double dis = FORMODTOHAF_CalculateValues.input_ddx_km;
                Number label3 = new Number(2, 1,dis );
                sheet.addCell(label3);

                for (int i = 1; i <= FORMODTOHAF_CalculateValues.input_n_obs; i++){
                    double obs = FORMODTOHAF_CalculateValues.input_obs[i];
                    Number label1 = new Number(3, i,obs);
                    sheet.addCell(label1);
                }

                double ang = FORMODTOHAF_CalculateValues.input_ang_pro;
                Number label4 = new Number(4, 1,Math.toDegrees(ang));
                sheet.addCell(label4);


                for (int i = 1; i <= FORMODTOHAF_CalculateValues.input_n_obs; i++){

                    double ele = FORMODTOHAF_CalculateValues.input_ele_km[i];
                    double haf = FORMODTOHAF_CalculateValues.input_strike_km[i];
                    double off = FORMODTOHAF_CalculateValues.input_y_km[i];
                    double nfm = FORMODTOHAF_CalculateValues.nfm[i];

                    Number label5 = new Number(5, i,ele );
                    Number label6 = new Number(6, i,haf );
                    Number label7 = new Number(7, i,off );
                    Number label8 = new Number(8, i,nfm );
                    sheet.addCell(label5);
                    sheet.addCell(label6);
                    sheet.addCell(label7);
                    sheet.addCell(label8);
                }

                for (int i = 1; i <= FORMODTOHAF_CalculateValues.input_num_for; i++){

                    double den = FORMODTOHAF_CalculateValues.den[i];
                    double dep = FORMODTOHAF_CalculateValues.dep[i];
                    Number label12 = new Number(9, i,den );
                    Number label13 = new Number(10, i,dep );
                    sheet.addCell(label12);
                    sheet.addCell(label13);
                }


                workbook.write();
                workbook.close();

            }
            catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (RowsExceededException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (WriteException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
```

```
                    }
                }
            }


        }
        else if(ae.getActionCommand().equals("Constant offset")){
            c.getAnamolyValues(FORMODTOHAF_MainPanel.captureValues());
            FORMODTOHAF_GetConstOff gc = new FORMODTOHAF_GetConstOff();
            gc.constOff();

        }else if(ae.getActionCommand().equals("Constant half strike")){

            c.getAnamolyValues(FORMODTOHAF_MainPanel.captureValues());
            FORMODTOHAF_GetConstHaf gch = new FORMODTOHAF_GetConstHaf();
            gch.constHaf();
        }else if(ae.getActionCommand().equals("Variable offset")){

            try{
                FORMODTOHAF_TableView.populateEastPanel(rowdata);
                c.getAnamolyValues(FORMODTOHAF_MainPanel.captureValues());
                FORMODTOHAF_OffsetVal ov = new FORMODTOHAF_OffsetVal();
                ov.val();
                DecimalFormat df = new DecimalFormat("0.##");

                for (int i = 1;i <= FORMODTOHAF_CalculateValues.input_n_obs; i++){
                    for (int j = 0;j < 2; j++){
                        FORMODTOHAF_OffsetVal.off[i-1][1]=
df.format(FORMODTOHAF_GetConstOff.input_y_km[i]);
                        FORMODTOHAF_OffsetVal.off[i-1][0]= df.format(i);
                        FORMODTOHAF_OffsetVal.table.setValueAt(FORMODTOHAF_OffsetVal.off[i-1][j],i
-1, j);
                    }
                }
            }
            catch(Exception e){

            }

        }else if(ae.getActionCommand().equals("Variable half strike")){

            try{
                FORMODTOHAF_TableView.populateEastPanel(rowdata);
                c.getAnamolyValues(FORMODTOHAF_MainPanel.captureValues());
                FORMODTOHAF_HalfstrikeVal sv = new FORMODTOHAF_HalfstrikeVal();
                sv.val();
                DecimalFormat df = new DecimalFormat("0.##");

                for (int i = 1; i <= FORMODTOHAF_CalculateValues.input_n_obs; i++){
                    for (int j = 0; j < 2; j++){
                        FORMODTOHAF_HalfstrikeVal.haf[i-1][1]=
df.format(FORMODTOHAF_GetConstHaf.input_strike_km[i]);
                        FORMODTOHAF_HalfstrikeVal.haf[i-1][0]= df.format(i);
                        FORMODTOHAF_HalfstrikeVal.table.setValueAt(FORMODTOHAF_HalfstrikeVal.haf[i
-1][j],i-1, j);
                    }
                }
            }
            catch(Exception e){

            }
        }else if(ae.getActionCommand().equals("Flow module")){

            try {
                String path =  "Flow module.txt";
                File pptFile = new File(path);
                if (pptFile.exists()) {
                    if (Desktop.isDesktopSupported()) {
                        Desktop.getDesktop().open(pptFile);
                    } else {
                        System.out.println("Desktop is not supported!");
                    }

                } else {
                    System.out.println("File does not exists!");
```

```
                }

                System.out.println("Done");

            } catch (Exception ex) {
                JFrame frame = null;
                JOptionPane.showMessageDialog(frame,
                        "System cannot open the file\n",
                        "Error!",
                        JOptionPane.ERROR_MESSAGE);

            }
        }else if(ae.getActionCommand().equals("Readme")){

            try {
                String path =  "Readme.txt";
                File pptFile = new File(path);
                if (pptFile.exists()) {
                    if (Desktop.isDesktopSupported()) {
                        Desktop.getDesktop().open(pptFile);
                    } else {
                        System.out.println("Desktop is not supported!");
                    }

                } else {
                    System.out.println("File does not exists!");
                }

                System.out.println("Done");

            } catch (Exception ex) {
                JFrame frame = null;
                JOptionPane.showMessageDialog(frame,
                        "System cannot open the file\n",
                        "Error!",
                        JOptionPane.ERROR_MESSAGE);

            }
        }else if(ae.getActionCommand().equals("Exit")){
            JFrame frame = null;

            int r = JOptionPane.showConfirmDialog(
                    frame,
                    "Exit FORMODTOHAF ?",
                    "Confirm Exit ",
                    JOptionPane.YES_NO_OPTION);
            if(r == JOptionPane.YES_OPTION ){
                if(success==false){
                    String fileName = FORMODTOHAF_CalculateValues.input_area_name+".jpg";
                    File f = new File(fileName);
                    f.delete();
                }
                System.exit(0);
            }
        }

    }
}
```

```
package com.formodtohaf.util;


public class FORMODTOHAF_Utility {

    public static double convertDouble(String str) throws Exception {

        Double temp = new Double(str.trim());
        return temp.doubleValue();
    }

    public static String convertString(String str) throws Exception {
```

```
    String temp = new String(str.trim());
    return temp;
}

public static int convertInteger(String str) throws Exception {

    Integer temp = new Integer(str.trim());
    return temp.intValue();
}

public static float findMaximumNumber( double observe[]) {

    double max = 0.0d;
    for (int i = 0; i < observe.length; i++) {

        if (Math.abs(observe[i]) > Math.abs(max)) {

            max = observe[i];
        }
    }

    float maxVal =  (float) (max/3*5);
    return maxVal;
}

public static int findMaximumNumber( double observe) {

    double max = 0.0d;
    int maxVal=0;
    max = observe;

    if (max < 5) {

        maxVal = 5;
    }
    else if (max >= 5 && max <= 10) {

        maxVal = 10;
    }
    else if ( max > 10 && max <= 15) {

        maxVal = 15;
    }
    else if (max > 15 && max <= 20) {

        maxVal = 20;
    }
    else
    {
        //maxVal = THEOGRANO_CalculateValues.o_iter;
    }

    return maxVal;
}

public static double findMaximumNumber1( double observe[]) {

    double max = 0.0d;
    for (int i = 1; i < observe.length; i++) {

        if (Math.abs(observe[i]) > Math.abs(max)) {

            max =Math.abs(observe[i]);
        }
    }

    double maxVal =  max;
    return maxVal;
}

public static double[] convertDoubleArray(String str) throws Exception {

    java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
    String temp = "";
    java.util.ArrayList arr = new java.util.ArrayList();
```

```
        while(st.hasMoreTokens()) {

            temp = st.nextToken();
            arr.add(temp);
        }
        double d_array[] = new double[arr.size() + 1];

        for(int i = 0; i <= arr.size(); i++) {

            if (i == 0)
                d_array[i] = 0.0;
            else
                d_array[i] = convertDouble( arr.get(i-1).toString() );
        }
        return d_array;
    }
}
```

CHAPTER
FIVE

Conclusions

The major conclusions and salient features of the work presented in chapters II, III and IV are mainly on the following lines

1. Novelty in the methodology,
2. Efficiency of 2-D and 2.5-D automatic modeling and inversion and related software,
3. Efficacy of 2.5-D semiautomatic modeling and related software.

Two interpretation methods based on the principles of automatic modeling and inversion are developed in each case in the space domain to analyze gravity anomalies of 2-D and 2.5-D sedimentary basins by means of multiple growing bodies using an exponential density function. Although, it is well known that no closed form solutions exist in the space domain for the gravity anomalies of geophysical geometries using an exponential density function, new strategies have been formulated in the space domain to realize forward modeling by judiciously combining both analytical and numerical approaches. These algorithms are fully automatic in the sense that they generate initial structures of sedimentary

basins from the residual gravity anomalies, and improve the structures iteratively based on the differences between the observed and modeled gravity anomalies until the modeled anomalies mimic the observed ones. The corresponding GUI based software, MOD2DSD, in case of 2-D and MODTOHAFSD in case of 2.5-D are developed using JAVA. The advantage of each software is that in addition to generate the output in both ASCII and graphical forms each one shows the changes in the structure, fit between the observed and modeled gravity anomalies, changes in the misfit and variation of density contrast with depth in animated forms. Further, the semiautomatic modeling technique and related JAVA software, FORMODTOHAF, allows one to compute the gravity anomalies of a variety of strike limited multiple geologic sources and subsequent modeling in an interactive mode. The novelty of 2.5-D algorithms is that these are fairly applicable to interpret the gravity anomalies even when the profile fails to bisect and runs transverse to the strike of the anomalous source(s).

The interpretation of gravity anomalies of 2-D and 2.5-D sedimentary basins with EDF remarkably agree well with the assumed parameters in the case of synthetic examples even in the presence of pseudorandom noise and with the available/drilling depths in case of field examples. On the other hand, almost in all cases, when automatic modeling and inversion implemented with UDF had resulted disturbed depth structures implying that UDF is inappropriate in such applications. Thus, in view of the reliable results coupled with other aforesaid features, it is suggested that for a meaningful interpretation of field gravity

anomalies of sedimentary basins, automatic modeling and inversion need to be implemented with EDF.

In case the subsurface density distribution depicts complex behavior, it is suggested that for reliable interpretation the gravity anomalies need to be analyzed by properly taking into account the strike length(s) of subsurface structure(s), azimuth and offset parameters of the profile besides appropriate selection of the profile length. The applicability of the semiautomatic modeling is demonstrated on both synthetic and real world gravity anomalies.

In short, it is concluded that the proposed algorithms and related software in the thesis are simple, elegant besides being effective.

Scope for Future Research

An important and still unsatisfactorily solved problem in gravity interpretation is the difficulty caused by a vertical superposition of sources in a sedimentary basin. Inversion algorithms of gravity anomalies of basement reliefs must take into account the presence of sources within the sedimentary pack and vice-versa. The methodology elucidated in Chapter-IV can be used to design an appropriate inversion strategy to address the enlisted problem.

An equally intrinsic solved problem in gravity interpretation is to estimate the source parameters and regional gravity background simultaneously from a set of observed Bouguer gravity anomalies. Although a few techniques are in vogue in this direction, each one has its

own merits and demerits in its application. Therefore, it is anticipated that future research will contemplate the more sophisticated regional-residual separation techniques, possibly integrated with the inversion processes described in the thesis, to cater more lucid interpretations of gravity anomalies.

*****

References

Abbott, R. E., and Louie, J. N., 2000, Depth to bed rock using gravimetry in the Reno and Carson City, Nevada, area basins: Geophysics, 65, 340 - 350.

Abdoh, A., Cowan D., and Pilkington, M., 1990, 3D gravity inversion of the Cheshire basin: Geophysical Prospecting, 38, 999 -1011.

Adriasyah, A., and McMechan, A. G., 2002, Analysis and interpretation of seismic data from thin reservoirs: Northwest Java basin, Indonesia: Geophysics, 67, 14-26.

Agarwal, B. N. P., and Sivagi, C., 1992, Separation of regional and residual anomalies by least squares orthogonal polynomial and relaxation techniques: a performance evaluation: Geophysical Prospecting, 40, 143-156.

Agarwal, B. N. P., and Shalivahan, S., 2010, A FORTRAN code to implement the finite element method to compute regional and residual anomalies from the gravity data: Computers & Geosciences, 36, 848-852.

Agarwal, B. P., 1995, Hydrocarbon prospects of the Pranhita - Godavari graben, India: Proceedings of Petrotech. 95, 115-121.

Al-Chalabi, M., 1970, Interpretation of two-dimensional magnetic profiles by non-linear optimization: Bollettino Di Geofisica Teorica Ed Applicata, 12, 3-20.

Al-Chalabi, M., 1972, Interpretation of gravity anomalies by non-linear optimization: Geophysical Prospecting, 10, 1-15.

Annecchione, M. A., Chouteau, M., and Keating, P., 2001, Gravity interpretation of bedrock topography: the case of the Oak Ridges Moraine, southern Ontario, Canada: Journal of Applied Geophysics, 47, 63-81.

Athy, L. F., 1930, Density, porosity and compaction of sedimentary rocks: Bulletin of the American Association of Petroleum Geology, 14, 1-24.

Azeglio, E. A., Gimenez, M. E., and Introcaso, A., 2010, Interpretation of Las Salinas sedimentary basin - Argentina, based on integration of geological and geophysical data: Geofísica Internacional, 49, 225-244.

Backus, G. E., and Gilbert, F. J., 1967, Numerical applications of a formalism for geophysical inverse problems: Geophysical Journal of the Royal Astronomical Society, 13, 247-276.

Backus, G. E., and Gilbert, F. J., 1968, The resolving power of gross earth data: Geophysical Journal of the Royal Astronomical Society, 16, 169-205.

Backus, G. E., and Gilbert, F. J., 1970, Uniqueness in the inversion of inaccurate gross earth data: Philosophical Transactions of the Royal of London, A226, 123-192.

Banerjee, B., and Gupta, S. P. D., 1977, Gravitational attraction of rectangular parallelopiped: Geophysics, 42, 1053-1055.

Baragar, W.R.A., 1974, Volcanic studies in the Cape Smith-Wakeham Bay belt, New Quebec: Geological Survey of Canada, Paper 74-1, Part. A: 155-157.

Barbosa, V. C., Silva, J. B. C., and Medeiros, W., 1997, Gravity inversion of basement relief using approximate equality constraints on depths: Geophysics, 62, 1745-1757.

Barbosa, V. C., Silva, J. B. C., and Medeiros, W., 1999, Gravity inversion of a discontinuous relief stabilized by weighted smoothness constraints on depth: Geophysics, 64, 1429-1437.

Barnett, C. T., 1976, Theoretical modeling of the magnetic and gravitational fields of an arbitrarily shaped three-dimensional body: Geophysics, 41, 1353-1364.

Bergeron, R., 1957, Preliminary report on Cape Smith-Wakeham Bay belt, New Quebec: Quebec Department of Mines, Preliminary Report, 355.

Beiki, M., 2010, Analytic signals of gravity gradient tensor and their application to estimate source location: Geophysics, 75, I59–I74.

Beiki, M. and Pedersen, L., 2010, Eigenvector analysis of gravity gradient tensor to locate geologic bodies: Geophysics, 75, I37–I49.

Beltrao, J. F., Silva, J. B. C., and Costa, J. C., 1991, Robust polynomial fitting method for regional gravity estimation: Geophysics, 56, 80-89.

Bhattacharya, B. K., and Navolio, M. E., 1975, Digital convolution for computing gravity and magnetic anomalies due to arbitrary bodies: Geophysics, 40, 981-992.

Blakely, R. J., 1995, Potential Theory in Gravity and Magnetic Applications: Cambridge University Press, 464 pp.

Bosch, M., and McGaughey, J., 2001, Joint inversion of gravity and magnetic data under lithologic constraints: The Leading Edge, 20, 877-881.

Boschetti, F., Dentith, M., and List, R., 1997, Inversion of potential field data by genetic algorithms: Geophysical Prospecting, 45, 461-478.

Boschetti, F., Therond, V., and Hornby, P., 2004, Feature removal and isolation in potential field data: Geophysical Journal International, 159, 833-841.

Bott, M. H. P., 1960, The use of rapid digital computing methods for direct gravity interpretation of sedimentary basins. Geophysical Journal of Royal Astronomical Society, 3, 63-67.

Boulanger, O., and Chouteau, M., 2001, Constraints in 3D gravity inversion: Geophysical Prospecting, 49, 265-280.

Brissaud, P., Deletie, P., Lakshmanan, J., Lemoine, Y., and Montlucon, J., 1989, Site of Tanis (Egypt): Geophysical investigations and their archaeological follow up: 59th Annual International Meeting, SEG, Expanded Abstracts, 292–293.

Byerly, P. E., 1965, Convolution filtering of gravity and magnetic maps: Geophysics, 30, 281-283.

Cady, J. W., 1980, Calculation of gravity and magnetic anomalies of finite-length right polygonal prisms: Geophysics, 45, 1507–1512.

Chacko, S., and Battacharya, B., 1980, A method for analyzing gravity anomalies due to a geologic contact by Fourier transform: Geoexploration, 18, 43-50.

Chai, Y., and Hinze, W. J., 1988, Gravity inversion of an interface above which the density contrast varies exponentially with depth: Geophysics, 53, 837-845.

Chakravarthi, V., 2003, Digitally implemented method for automatic optimization of gravity fields obtained from three-dimensional density interfaces using depth dependent density: US Patent 6,615,139.

Chakravarthi, V., and Sundararajan, N., 2004, Ridge regression algorithm for gravity inversion of fault structures with variable density: Geophysics, 69, 1394-1404.

Chakravarthi, V., and Sundararajan, N., 2007, 3D gravity inversion of basement relief – A depth dependent density approach: Geophysics, 72, I23- I32.

Chakravarthi. V., Shankar, G. B. K., Muralidharan, D., Harinarayana, T., and Sundararajan, N., 2007, An integrated geophysical approach for imaging subbasalt sedimentary basins: Case study of Jam River basin, India: Geophysics, 72, B141–B147.

Chakravarthi, V., 2009, Automatic gravity inversion for simultaneous estimation of model parameters and regional gravity background: an application to 2D pull-apart basins: Current Science, 96, 1349-1360.

Chakravarthi, V., 2011, Geodesy, Physical: Encyclopedia of Solid Earth Geophysics, Springer, 331-335.

Chakravarthi. V., Ramamma, B., and Venkat Reddy, T., 2013a, Gravity anomaly modeling of sedimentary basins by means of multiple structures and exponential density contrast-depth variations: A space domain approach: Journal of the Geological Society of India, 82, 561-569.

Chakravarthi. V., Ramamma, B., 2013b, A space domain based gravity inversion to trace discontinuous basement interfaces above which the density contrast decreases exponentially with depth: Acta Geophysica (in press).

Chakravarthi. V., Rajeswara Sastry, S., and Ramamma, B., 2013c, MODTOHAFSD – A GUI based JAVA code for gravity analysis of strike limited sedimentary basins by means of growing bodies with exponential density contrast – depth variation: A space domain approach: Computers & Geosciences, 56, 131-141.

Chakravarthi. V., and Ramamma, B., 2013d, Gravity anomaly modelling of multiple geological sources having differing strike lengths and arbitrary density contrast variations: Near Surface Geophysics, 11, 363-370.

Chapin, D. A., 1996, A deterministic approach toward isostatic gravity residuals; a case study from South America: Geophysics, 61, 1022 - 1033.

Chappell, A., and Kusznir, N., 2008, An algorithm to calculate the gravity anomaly of sedimentary basins with exponential density-depth relationships: Geophysical Prospecting, 56, 249–258.

Collins, S. J., Dodds, A. R., and Johnson, B. D., 1974, Gravity profile interpretation using Fourier transform: Geophysics, 39, 862-866.

Commer, M., 2011, Three-dimensional gravity modelling and focusing inversion using rectangular meshes: Geophysical Prospecting, 59, 966–979.

Cordell, L., 1973, Gravity anomalies using an exponential density-depth function-San Jacinto Graben, California: Geophysics, 38, 684-690.

Cowie, P. A., Karner, G. D., 1990, Gravity effect of sediment compaction: examples from the North Sea and Rhine Graben: Earth and Planetary science Letters, 99 (1-2), 141-153.

Crowe, C., and Redmond, J. C., 1962, Some effects of pressure on porosity, permeability, and resistivity of sandstones: Twenty-third technical conference on petroleum production: Mineral Industries Experiment Station Circ. 63, University Park, Pa. The Pennsylvania State University.

Deletie, P., Lemoine, Y., Montlucon, J., and Lakshmanan, J., 1988, Discovery of two unknown pyramids at Saqqarah, Egypt by a multi method geophysical survey: 58th Annual International Meeting, SEG, Expanded Abstracts, 335–337.

De Vicente, J., Lanchares, J., and Hermida, R., 2003, Placement by thermodynamic simulated annealing: Physics Letters, A317, 415-423.

Dobrin, M. B., and Savit, C. H., 1988, Introduction to Geophysical Prospecting: McGraw-Hill Book Co., 867 pp.

Dorigo, M., and Blum, C., 2005, Ant colony optimization theory: a survey: Theoretical Computational Science, 344, 243-278.

Eberhart, R. C., and Kennedy, J., 1995, A new optimizer using particle swarm theory: Proceedings of sixth International symposium on micro machine and human science, IEEE service centre, Piscataway, Nagoya, Japan, 39-43.

Eckhardt, E. A., 1948, A brief history of the gravity method of prospecting for oil: Geophysical case histories, Society of Exploration Geophysicists, 1, 21-32.

Fedi, M., and Rapolla, A., 1999, 3-D inversion of gravity and magnetic data with depth resolution: Geophysics, 64, 452-460.

Ferguson, J. F., Felch, R. N., Aiken, C. L. V., Oldow, J. S., and Dockery, H., 1988, Models of the Bouguer gravity and geologic structure at Yucca Flat, Nevada: Geophysics, 53, 231-244.

Fett, J. D., 1968, Geophysical investigation of the San Jacinto Valley, Riverside County, California: Unpublished thesis, University of California at Riverside, 87 pp.

Filon, L. N. G., 1928, On a quadrature formula for trigonometric integrals: Proceedings of the Royal Society of Edinburgh, 49, 38-47.

Forsythe, G. E., 1957, Generation and use of orthogonal polynomials for data filtering with a digital computer: Journal Society Indian Applied Mathematics, 5, 75-88.

Foster, J. B., and Whalen, H. E., 1966, Estimation of formation pressures from electrical surveys - offshore Louisiana: Journal of Petroleum Technology, 17, 165-171.

Ganguly, S. S., and Dimri, V. P., 2013, Interpretation of gravity data using eigenimage with Indian case study: A SVD approach: Journal of Applied Geophysics, 95, 23-35.

García-Abdeslem, J., 1992, Gravitational attraction of a rectangular prism with depth-dependent density: Geophysics, 57, 470-473.

García-Abdeslem, J., 1996, GL2D: A Fortran program to compute the gravity anomaly of a 2-d prism where density varies as a function of depth: Computers & Geosciences, 22, 823-826.

García -Abdeslem, J., 2005, The gravitational attraction of a right rectangular prism with density varying with depth following a cubic polynomial: Geophysics, 70, j39-j42.

Gardner, G. H. F., Gardner, L. W., and Gregory, A. R., 1974, Formation velocity and density - The diagnostic basis for stratigraphic traps: Geophysics, 39, 770-780.

Gimenez, M. E., Martinez, M. P., Jordan, T., Ruíz, F., and Lince Klinger, F., 2009, Gravity characterization of the La Rioja Valley basin, Argentina: Geophysics, 74, B83–B94.

Golizdra, G. Y., 1981, Calculation of the gravitational field of a polyhedron: Izvestiya, Earth physics, 17, 625-628.

Gallardo-Delgado, L. A., Perez-Flores, M. A., and Gomez-Trevino, E., 2003, A versatile algorithm for joint inversion of gravity and magnetic data: Geophysics, 68, 949–959.

Gómez-Ortiz, D., Tejero-López, R., Babín-Vich, R., and Rivas-Ponce, A., 2005, Crustal density structure in the Spanish Central System derived from gravity data analysis (Central Spain): Tectonophysics, 403, 131–149.

Gotze, H. J., and Lahmeyer, B., 1988, Application of three-dimensional interactive modeling in gravity and magnetics: Geophysics, 53, 1096 - 1108.

Granser, H., 1987, Three-dimensional interpretation of gravity data from sedimentary basins using an exponential density-depth function: Geophysical Prospecting, 35, 1030-1041.

Grant, F. S., 1957, A problem in the analysis of geophysical data: Geophysics, 22, 309-344.

Green, W. R., 1975, Inversion of gravity profile by uses of Backus-Gilbert approach: Geophysics, 40, 763–772.

Gu, X., Tenzer, R., and Gladkikh, V., 2014, Empirical models of the ocean-sediment and marine sediment-bedrock density contrasts: Geosciences Journal (in press), DOI 10.1007/s12303-014-0015-9.

Guang, Z., Hou, J., Huang, L., and Yao, C., 1998, Inversion of gravity and magnetic anomalies using pseudo-BP neural network method and its application: Chinese Journal of Geophysics, 41, 139-149.

Gupta, D. K., Gupta, J. P., Arora, Y., and Singh, U. K., 2011, Recursive Ant Colony Global Optimization: a new technique for the inversion of geophysical data: American Geophysical Union, Fall Meeting, abstract #H13B-1196.

Guspí, F., 1990, General 2D inversion with density contrast varying with depth: Geoexploration, 26, 253-265.

Ham, H. H., 1966, New charts help estimate formation pressure: Oil Gas Journal, 65, 58-63.

Hansen, R. O., 1999, An analytical expression for the gravity field of a polyhedral body with linearly varying density: Geophysics, 64, 75–77.

Hedberg, H. D., 1936, The gravitational compaction of clays and shales: American Journal of Science, 31, 241-287.

Hermes, H. J., 1986, Calculation of pre-Zechstein Bouguer anomaly in northwest Germany: First Break, 4, 13–22.

Hinze, W. J., 2003, Bouguer reduction density. Why 2.67? : Geophysics, 68, 1559-1560.

Holstein, H., and Ketteridge, B., 1996, Gravimetric analysis of uniform polyhedra: Geophysics, 61, 357-364.

Holstein, H., 2003, Gravimagnetic anomaly formulas for polyhedra of spatially linear media: Geophysics, 68, 157-167.

Hughes, D. S., and Cooke, C. E., 1953, The effect of pressure on the reduction of pore volume of consolidated sandstones: Geophysics, 18, 298-309.

Huston, D. C., Huston, H. H., and Johnson, E., 2004, Geostatistical integration of velocity cube and log data to constrain 3D gravity modeling, deepwater Gulf of Mexico: The Leading Edge, 23, 842–846.

Jachens, R.C., and Moring, B. C., 1990, Maps of thickness of Cenozoic deposits and isostatic residual gravity over basement in Nevada: U.S. Geological Survey Open-File Report, 90–404.

Jacobsen, B. H., 1987, A case for upward continuation as a standard separation filter for potential field maps: Geophysics, 52, 1138-1148.

Jessell, M., 2001, Three-dimensional geological modelling of potential-field data: Computers & Geosciences, 27, 455-465.

Jingxin, Q., Ziqi, G., Jianying, L., and Yanchao, Q., 2013, Research in Modified Simulated Annealing Interface Inversion Method: Near Surface Geophysics Asia Pacific Conference, Beijing, China, 657-659.

Kadirov, F. A., 2000, Application of the Hartley transform for interpretation of gravity anomalies in the Shamakhy–Gobustan and Absheron oil-and gas-bearing regions, Azerbaijan: Journal of Applied Geophysics, 45, 49-61.

Kaila, K. L., Murthy, P. R. K., Rao, V. K. , and Venkateswarlu, N., 1990, Deep seismic sounding in the Godavari Graben and Godavari (Coastal) Basin, India: Tectonophysics, 173, 307–317.

Lakshmanan, J., and Montlucon, J., 1987, Microgravity probes the Great Pyramid: The Leading Edge, 6, 10–17.

Lavin, P. M., and Devane, J. F., 1970, Direct design of two-dimensional digital wave number filters: Geophysics, 35, 1073-1078.

Leão, J. W. D., Menezes, P. T. L., Beltrão, J. F., and Silva, J. B. C., 1996, Gravity inversion of basement relief constrained by the knowledge of depth at isolated points: Geophysics, 61, 1702–1714.

Leite, E. P., and Filho, C. R. de. S., 2009, Artificial neural networks applied to mineral potential mapping for copper-gold mineralizations in the Carajás Mineral Province, Brazil: Geophysical Prospecting 57, 1049-1065.

Li, Y., and Oldenburg, D. W., 1998, 3-D gravity inversion of gravity data: Geophysics, 63, 109-119.

Lines, L. R., and Treitel, S., 1984, Tutorial a review of least-squares inversion and its application to geophysical problems: Geophysical Prospecting, 32, 159-186.

Litinsky, V. A., 1989, Concept of effective density: key to gravity depth determinations for sedimentary basins: Geophysics, 54, 1474-1482.

Mallick, K., and Sharma, K. K., 1999, A finite element method for computation of the regional gravity anomaly: Geophysics, 64, 461-469.

Mareschal, J. C., 1985, Inversion of potential field data in Fourier transform domain: Geophysics, 50, 685-691.

Marlet, G., Sailhac, P., Moreau, F., and Diament, M., 2001, Characterization of geological boundaries using 1-D wavelet transform on gravity data; theory and application to the Himalayas: Geophysics, 66, 1116-1129.

Marquardt, D. W., 1963, An algorithm for least squares estimation of nonlinear parameters. Journal Society Indian Applied Mathematics, 11, 431-441.

Marquardt D. W., 1970, Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation: Technometrics, 12, 591-612.

Martín Atienza, B., and García-Abdeslem, J., 1999, 2-D gravity modeling with analytically defined geometry and quadratic polynomial density functions: Geophysics 64, 1730–1734.

Martinez, P., Gimenez, M., Folguera, A., and Klinger, F. L., 2014, Integrated seismic and gravimetric model of Jocolí Basin, Argentina: Interpretation, 2, T57–T68.

Maxant, J., 1980, Variation of density with rock type, depth, and formation in the Western Canada basin from density logs: Geophysics, 45, 1061-1076.

Mendonca, C. A., 2004, Inversion of gravity-field inclination to map the basement telief of sedimentary basins: Geophysics, 69, 1240-1251.

Merriam, D. F., and Cocke, N. C., 1967, Colloquium on trend analysis: Kansas State Geological survey Comput. Contr., 12, P.62.

Mickus, K. L., Aiken, C. L. V., and Kennedy, W. D., 1991, Regional-residual gravity anomaly separation using minimum curvature technique: Geophysics, 56, 279-283.

Mickus, K. L., and Peeples, W. J., 1992, Inversion of gravity and magnetic data for the lower surface of a two and one-half dimensional sedimentary basin: Geophysical Prospecting, 40, 171-193.

Mishra, D. C. , Gupta, S. B., Rao, M. B. S. V., Venkatarayudu, M., and Laxman, G., 1987, Godavaribasin - a geophysical study: Journal of the Geological Society of India, 30, 469–476.

Mohan, N. L., 1978, Interpretation techniques in geophysical exploration using Fourier Transforms: Ph. D thesis, Osmania University, Hyderabad, India (unpublished).

Mohan, C. R., Sastry, R. G. S., Moharir, P. S., 1986, Inversion of gravity anomalies using Tunneling algorithm: In twelfth annual convention and seminar on Exploration Geophysics, A7-A8.

Moral, C. R., Gȯmez Ortiz, D., and Tejero, R., 2000, Spectral analysis and gravity modelling of the Almazán basin (Central Spain): Journal of the Geological Society of Spain, 13, 131-142.

Morgan, N. A., and Grant, F. S., 1963, High-speed calculation of gravity and magnetic profiles across two-dimensional bodies having an arbitrary cross-section: Geophysical Prospecting, 11, 10 -15.

Moritz, H.,1980.Advanced Physical Geodesy: Karlsruhe:Wichmann, 500 pp.

Mosegaard, K., and Tarantola, A., 1995, Monte Carlo sampling of solutions to inverse problems: Journal of Geophysical Research, 100, 12431-12447.

Mukhopadhyay, M., and Gibb, R. A., 1981, Gravity anomalies and deep structure of Eastern Hudson Bay: Tectonophysics, 72, 43-60.

Murthy, I. V. R., and Rao, D. B., 1979, Gravity anomalies of two-dimensional bodies of irregular cross-section with density contrast varying with depth: Geophysics, 44, 1525-1530.

Murthy, I. V. R., and Rao, D. B., 1980, Interpretation of gravity anomalies over faults and dykes by Fourier transforms, employing end correction: Geophysical Research Bulletin, 18, 95-110.

Murthy, I. V. R., 1998, Gravity and Magnetic Interpretation in Exploration Geophysics: Memoir 40, Geological Society of India, 363 pp.

Murthy, I. V. R., Krishna, P. R., and Rao, S. J., 1988, A generalized computer program for two-dimensional gravity modeling of bodies with a flat top or a flat bottom or undulating over a mean depth: Journal of Association of Exploration Geophysicists, 9, 93-103.

Murthy, I. V. R., and Rao, S. J., 1989, A Fortran 77 program for inverting gravity anomalies of two-dimensional basement structures: Computers & Geosciences, 15, 1149-1156.

Nabighian, M. N., Ander, M. E., Grauch, V. J. S., Hansen, R. O., LaFehr, T. R., Li, Y., Pearson, W. C., Peirce, J. W., Phillips, J. D., and Ruder, M.

E., 2005, Historical development of the gravity method in exploration: Geophysics, 70, 63ND–89ND.

Nagihara, S., and Hall, S. A., 2001, Three-dimensional gravity inversion using simulated annealing, constraints on the diapiric roots of allochthonous salt structures: Geophysics, 66, 1438-1449.

Nagy, D., 1966, The gravitational attraction of a right rectangular prism: Geophysics, 31, 362-371.

Naidu, P. S., 1966, Extraction of a potential field signal from a background of random noise by Strakhov's method: Journal of Geophysical Research, 71, 5987-5995.

Naidu, P. S., 1967, Two-dimensional Strakhov's filter for extraction of a potential field signal: Geophysical Prospecting, 15, 135-150.

Naudy, H., and Dreyer, H., 1968, Essai de filtrage non-lineaire appliqué aux profiles aeromagnetiques: Geophysical Prospecting, 16, 171-178.

Nettleton, L. L., 1976, Gravity and Magnetics in oil prospecting: McGraw - Hill Press Inc.

Oldenburg, D. W., 1974, The inversion and interpretation of gravity data: Geophysics, 39, 526-536.

Oruç, B., 2014, Structural interpretation of south part of western Anatolian using analytic signal of the second order gravity gradients and discrete wavelet transform analysis: Journal of Applied Geophysics (In Press, Accepted Manuscript), Available online 24 January 2014.

Okabe, M., 1979, Analytical expressions for gravity anomalies due to homogenous polyhedral bodies and translations into magnetic anomalies: Geophysics, 44, 730-741.

Pacino, M. C., and Introcaso, A., 1988, Regional anomaly determination using the upwards – continuation method: Bolúmen 1 Geofisica Teorica Applicata, XXIX (114), 113- 122.

Paul, M. K., 1972, Interpretation of the gravity anomaly over a causative body with circular symmetry: Geophysical Prospecting, 20, 118–129.

Paul, M. K., 1974, The gravity effect of homogeneous polyhedron for three-dimensional interpretation: Pure and Applied Geophysics, 112, 553-561.

Pawlowski, R. S., 1994, Green's equivalent-layer concept in gravity band-pass filter design: Geophysics, 59, 69-76.

Pawlowski, R. S., and Hansen, R. O., 1990, Gravity anomaly separation by Wiener filtering: Geophysics, 55, 539–548.

Pedersen, L. B., 1977, Interpretation of potential data - A generalized inverse approach: Geophysical Prospecting, 25, 199-230.

Peirce, J. W., and Lipkov, L., 1988, Structural interpretation of the Rukwa Rift, Tanzania: Geophysics, 53, 824-836.

Pohánka, V., 1988, Optimum expression for computation of the gravity field of a homogeneous polyhedral body: Geophysical Prospecting, 36, 733–751.

Pohánka, V., 1990, Reply to comment by M. Ivan: Geophysical Prospecting, 38, 333-335.

Prieto, C., Perkins, C., and Berkman, E., 1985, Columbia River Basalt Plateau - An integrated approach to interpretation of basalt-covered areas: Geophysics, 50, 2709-2719.

Rama Rao, P., Swamy, K. V., and Murthy, I. V. R., 1999, Inversion of gravity anomalies of three-dimensional density interfaces: Computers and Geosciences, 25, 887–896.

Rasmussen, R., and Pedersen, L. B., 1979, End corrections in potential field modeling: Geophysical Prospecting, 27, 749-760.

Rao, B. S. R., and Murthy, I. V. R., 1978, Gravity and Magnetic methods of prospecting: Arnold-Heinemann publishers (India) Pvt. Ltd., 390 pp.

Rao, C. S. R., 1982, Coal resources of Tamilnadu, Andhra Pradesh, Orissa and Maharashtra: Bulletin of the Geological Survey of India, 2, 1-103.

Rao, D. B., 1986, Modeling of sedimentary basins from gravity anomalies with variable density contrast: Geophysical Journal of the Royal Astronomical Society, 84, 207-212.

Rao, D. B., 1990, Analysis of gravity anomalies of sedimentary basins by an asymmetrical trapezoidal model with quadratic density function: Geophysics, 55, 226-231.

Rao, D. B., Prakash, M. J., and Babu, N. R., 1993, Gravity interpretation using Fourier transforms and simple geometrical models with exponential density contrast: Geophysics, 58, 1074-1083.

Rao, D. B., Rao, C. P. V. N. J. M., 1999, Two-dimensional interpretation of gravity anomalies over sedimentary basins with an exponential decrease in density contrast with depth: Proceedings of Indian Academy of Sciences, 108, 99-106.

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

Roy, L., Shaw, R., and Agarwal, B. N. P., 2002, Inversion of gravity anomalies over sedimentary basins; applications of genetic algorithm and simulated annealing: SEG Annual Meeting Expanded Technical Program Abstracts with Biographies, 72, 747-750.

Rubey, W. W., and Hubbert, M. K., 1956, Role of fluid pressure in mechanics of overthrust faulting, II, overthrust belt in geosynclinal area of western Wyoming in light of fluid-pressure hypothesis: Geological Society of America Bulletin, 70, 167-206.

Ruotoistenmäki, T., 1992, The gravity anomalies of two-dimensional sources with continuous density distribution bounded by continuous surfaces: Geophysics, 57, 623-628.

Sax, R. L., 1966, Application of filter theory and information theory to the interpretation of gravity measurements: Geophysics, 31, 570-575.

Scales, J. A., and Snieder, R., 2000, The anatomy of inverse problems: Geophysics, 65, 1708-1710.

Sclater, J. G., and Christie, P. A. F., 1980, Continental stretching: An explanation of the post mid-Cretaceous subsidence of the Central North Sea Basin: Journal of Geophysical Research, 85, 3711-3739.

Shalivahan, S., and Agarwal, B. N. P., 2010, Inversion of the amplitude of the analytic signal of the magnetic anomaly by particle swarm optimization technique: Geophysical Journal International, 182, 652-662.

Sharma, B., and Geldart, L. P., 1968, Analysis of gravity anomalies using Fourier transforms: Geophysical Prospecting, 16, 77-93.

Sigl, R., 1985, Introduction to Potential Theory: Cambridge, Abacus Press, 229 pp.

Silva Dias, F. J. S., Barbosa, V. C. F., and Silva, J. B. C., 2007, 2D gravity inversion of a complex interface in the presence of interfering sources: Geophysics, 72, I13- I22.

Singh, B., and Guptasarma, D., 2001, New method for fast computation of gravity and magnetic anomalies from arbitrary polyhedra: Geophysics, 66, 521-526.

Spector, A., and Grant, F. S., 1970, Statistical models for interpreting aeromagnetic data: Geophysics, 35, 293-302.

Spitz, O. T., 1966, Generation of orthogonal polynomials for trend surfacing with a digital computer: Proceedings of symposium on operations research, In Mineral Ind., Pennsylvania State Univ., 3, 2-7.

Srivastava, S., Datta, D., Agarwal, B. N. P., and Mehta, S., 2013, Applications of Ant Colony Optimization in determination of source

parameters from total gradient of potential fields: Near Surface Geophysics (Early Online), DOI: 10.3997/1873-0604.2013054.

Stam, J.C., 1961, On the geology and petrology of the Cape Smith - Wakeham Bay belt, Ungava, Quebec: Geologie en Mijnbouw, 40, 412-421.

Strakhov, V. N., 1964, The smoothing of observed strength of potential fields: Akad. Nauk. SSSR. Izv. Fizika Zemli, Part I, 897-904.

Strakhov, V. N., and Lapina, M. I., 1967, A method of smoothing potential fields: Akad. Nauk. SSSR. Izv. Fizika Zemli, 40-57.

Strakhov, V. N., Lapina, M. I., and Yefimov, A. B., 1986, A solution to forward problem in gravity and magnetism with new analytical expressions for the field elements of standard approximating bodies I: Izvestiya, Earth Physics, 22, 471-482.

Sundararajan, N., Mohan, N. L., and Rao, S. V. S., 1983, Gravity interpretation of 2D fault structures using Hilbert transforms: Journal of Geophysics, 53, 34-41.

Sundararajan, N., and Brahmam, G. R., 1998, Spectral analysis of gravity anomalies caused by slab-like structures: A Harley transform technique: Journal of Applied Geophysics, 39, 53-61.

Sundararajan, N., Srinivas, Y., and Rao, T. L., 2000, Sundararajan Transform - a tool to interpret potential field anomalies: Exploration Geophysics, 31, 622 – 628.

Takin, M., and Talwani, M., 1966, Rapid computation of the gravitation attraction of topography on a spherical Earth: Geophysical Prospecting, 14,119–142.

Talwani, M., Worzel, J., and Ladisman, M., 1959, Rapid gravity computations for two dimensional bodies with application to the Mendocino submarine fracture zone: Journal of Geophysical Research, 64, 49 - 59.

Telford, W. M., Geldert, L. P., and Sheriff, R. E., 1990, Applied Geophysics, Cambridge University Press, 744pp.

Teodorovich, G. J., and Chernov, A. A., 1968, Changes with depth in the productive series of the Apsheron petroliferous region: Sovetskaya Geologiya, 4, 83-93.

Toushmalani, R., 2013, Gravity inversion of a fault by Particle swarm optimization (PSO): SpringerPlus, 2:315.

Ursin, B., Bauer, C., Zhao, H., and Fichler, C., 2003, Combined seismic and gravity modeling of a shallow anomaly in the southern Barents Sea: Geophysics, 68, 1140 - 1149.

Visweswara Rao, C., Chakravarthi, V., and Raju, M. L., 1994, Forward Modelling: Gravity anomalies of two-dimensional bodies of arbitrary shape with hyperbolic and parabolic density functions: Computers and Geosciences, 20, 873-880.

Weller, J. M., 1959, Compaction of sediments: Bulletin of the American Association of Petroleum Geology, 43, 273-310.

Wilson, J. T., 1968, Comparison of Hudson Bay arc with some other features: In History and Hudson Bay (ed. C.S. Beals and D. A. Shenstone), 1015-1033, Department of Energy, Mines and Resources.

Won, I. J., and Bevis, M., 1987, Computing the gravitational and magnetic anomalies due to a polygon: Algorithms and Fortran subroutines: Geophysics, 52, 232-238.

Yao, C., Hao, T., Guan, Z., and Zhang, Y., 2003, High-speed computation and efficient storage in 3-D gravity and magnetic inversion based on genetic algorithms: Acta Geophysica Sinica, 46, 252-258.

Zhang, J., Zhong, B., Zhou, X. and Dai, Y., 2001, Gravity anomalies of 2-D bodies with variable density contrast: Geophysics, 66, 809-813.

Zhou, X., 2008, Two-dimensional (2D) vector gravity potential and general line integrals for gravity anomaly due to 2D masses of depth-dependent density contrast: Geophysics, 73, I43–I50.

Zhou, X., 2009, General line integrals for gravity anomalies of irregular two-dimensional (2D) masses with horizontally- and vertically-dependent density contrast: Geophysics, 74, I1–I7.

Zhou, X., 2013, Gravity inversion of 2D bedrock topography for heterogeneous sedimentary basins based on line integral and maximum difference reduction methods: Geophysical Prospecting, 61, 220–234.