Space domain-based algorithms and related software to analyse magnetic anomalies of 2D Listric fault structures

A Thesis submitted during the year 2024 to the University of Hyderabad in partial fulfilment of the award of a Ph.D. degree in the Centre for Earth, Ocean and Atmospheric Sciences, School of Physics

by

V. Ani Nibisha

Centre for Earth, Ocean and Atmospheric Sciences
School of Physics

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

CERTIFICATE

This is to certify that the thesis entitled *Space domain-based algorithms and related software to analyze magnetic anomalies of 2D Listric fault structures* submitted by *V. Ani Nibisha,* bearing registration number *17ESPE04,* in partial fulfilment of the requirements for award of Doctor of Philosophy in the *Centre for Earth, Ocean and Atmospheric Sciences, School of Physics* is a bonafide work carried out by her under my supervision and guidance. This thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for award of any degree or diploma. Further, the student has the following publication(s) before submission of the thesis for adjudication and has produced evidence for the same in the form of reprints in the relevant area of her research

1. Ani Nibisha V, Ramamma B, Sastry R S and Chakravarthi V 2021 Forward modelling: Magnetic anomalies of arbitrarily magnetised 2D fault sources with analytically defined fault planes; J. Earth Syst. Sci. 130 130, https://doi.org/10.1007/s12040-021-01634-x. Chapter of dissertation where this publication appears 2 & 3,
2. Ani Nibisha V., Ramamma B, and Chakravarthi V 2022 Automatic inversion of magnetic anomalies caused by 2D listric fault sources with arbitrary magnetisation; J. Earth Syst. Sci. (2022) 131:265, https://doi.org/10.1007/s12040-022-02012-x. Chapter of dissertation where this publication appears 4

and has made presentations in the following conferences:

1. Ani Nibisha, V., Mallesh, K., Ramamma, B., Chakravarthi, V., Magnetic anomalies due to 2D fault sources bounded by non-planar fault planes, International Seminar on Advances in Geosciences for exploration of minerals, oil and gas, Department of Exploration Geophysics, Osmania University, 14-15 March, 2019.
2. Ani Nibisha, V., Deepak, M., Ramamma, B., Chakravarthi, V., Automatic inversion of magnetic anomalies caused by listric fault sources with arbitrary magnetization, Oil & Gas Exploration: Challenges and Opportunities, CEOAS, 20 October, 2022.

Further, the student has passed the following courses towards fulfilment of the coursework requirement for Ph.D.

Sl No.	Course Code	Course Title	Credits	Pass/Fail
1	ES 801	Earth System Science	4	Pass
2	ES 805	Research Methodology	3	Pass
3	ES 806	Mathematics for Earth Sciences	4	Pass
4	ES 851	Interdisciplinary course	3	Pass
5	AP 811 – 830	Special Paper on Specified Research Topic	2	Pass

Supervisor

Dr. V. Chakravarthi
Professor
Centre for Earth, Ocean & Atmospheric Sciences
University of Hyderabad
Hyderabad-500 046, Telangana, INDIA.

10/5/2024

Head of the Centre

Head
Centre for Earth, Ocean & Atmospheric Sciences
University of Hyderabad
Hyderabad-500 046, INDIA.

10/5/24

Dean of School

DEAN
School of Physics
University of Hyderabad
HYDERABAD - 500 046

Declaration

Centre for Earth, Ocean and Atmospheric Sciences
School of Physics
University of Hyderabad
Hyderabad – 500 046

I hereby declare that, this thesis entitled "*Space domain-based algorithms and related software to analyze magnetic anomalies of 2D Listric fault structures*" submitted by me under the guidance and supervision of Prof. V. Chakravarthi, is a bonafide research work. I also declare that it has not been submitted previously, in part or in full to this University or any other University or Institute, for the award of any degree or diploma.

Date: Name: V. Ani Nibisha

Signature of the student

Reg. No: 17ESPE04

Acknowledgements

It is with a deep sense of indebtedness and respect that I thank my research supervisor Prof. V. Chakravarthi, Centre for Earth, Ocean and Atmospheric Sciences (CEOAS), University of Hyderabad for his expert guidance and constant encouragement throughout the progress of my research work.

I owe special thanks to my Research Advisory Committee members, Prof. M. Jayananda and Dr. S. Srilakshmi for their useful suggestions during the course of my research work.

Head, CEOAS is acknowledged for providing necessary facilities to carry out my research at the Centre. I am thankful to my fellow research scholars, and the supporting staff of CEOAS for their benevolent attitude.

Words fall short to express my heartfelt gratitude to my parents, husband, and other family members for their ever-loving support and encouragement all through my career.

***

List of Figures

List of Tables

Table of contents

CHAPTER

ONE

Introduction

1.1 General

Geophysical methods facilitate to untangle the earth's subsurface geology and earth's internal architecture by measuring the geophysical signals produced by them on various measuring platforms. The increasing demand for oil, gas and other natural resources has led to the advancement of geophysical technology in terms of sophisticated instruments, survey designs, and state-of-the art processing and interpretational techniques to unravel the intricacies of the subsurface. Certainly, the existence of significant physical property contrast between the target that is being looked for and the surrounding rocks is essential for generating detectable anomalous signals at the measuring stations. Adding to this the source depth and its dimensions also would influence the nature of the anomalous field.

Broadly, geophysical methods can be categorized into static, dynamic, relaxation, and integrated effect methods. In static geophysical methods, like gravity and magnetic, the distortions in static physical fields are measured and quantified to explore the subsurface causative sources. The governing equations for gravity anomalous field contain the density contrast, and in case of magnetics the anomalous field involves magnetization contrast. In dynamic geophysical methods, the energy is generated and sent into the ground, and the returning signals are picked up at several measuring stations. In this category, the equations governing the physical field contain the time factor like the arrival time in case of seismic

methods, and phase or frequency difference in case of electromagnetic methods. In relaxation methods (like induced polarization), the time required for an energized medium to restore back to its ground state is studied, hence, the time factor enters into the corresponding governing equations. In integrated effect methods, the measured signals become the statistical average within a given volume or over a given area.

Generally, deducing concealed geological sources/structures from the signals measured at the surface comes under geophysical inverse problems. Geophysical exploration continues to be a grave encounter due to the fact that the physical property contrast needed to detect the subsurface features can be caused by different rock types having the same physical properties, which makes the process of finding a definite solution for the subsurface target elusive. The ambiguity or non-uniqueness in any geophysical interpretation can be reduced to a tolerable degree by integrating with other geophysical methods or making use of the available information as inputs/constraints in the model space to be mapped (Pilkington, 2009). Nonetheless, the interpretation is generally concerned to introduce assumptions or a definite geometry to the structure to restrict the number of plausible solutions.

The magnetic method, which is the subject matter of the thesis, is a versatile and perhaps the cheapest geophysical method to deploy in many geologic explorations. Magnetic surveys are carried out on the surface, in air, and also in oceans. Airborne magnetic surveys are essentially of reconnaissance nature, although some structures capable of holding hydrocarbons and large magnetite deposits have been located directly by the magnetic surveys. The airborne surveys have an advantage over the ground surveys in terms of quick coverage of large areas, surveying in inaccessible regions like forests, water logged areas, rugged terrains etc. Furthermore, the airborne surveys serve in demarcating areas of interest, where further ground geophysical work can be planned and thus, they help in saving large amounts of expenditure which would have been incurred if detailed ground magnetic work were to be undertaken in non-potential regions.

Marine magnetic surveys are also carried out extensively on both continental shelves and deeper portions of the seas for diversified geologic applications. Among several applications, the magnetic method is used extensively in the exploration of geothermal resources (Soengkono et al., 1992; Xu et al., 2007; Pauta et al., 2021), hydrocarbon and mineral resources (Campbell and Mason, 1979; Paterson and Reeves, 1985; Sharma, 1987), identification of paleochannels, concealed basic intrusives, landfill boundaries, archaeological investigations etc.

1.2 Earth's magnetic field and magnetic elements

Large part of the earth's magnetic field is produced by the convection currents in the outer core. In addition, currents in the ionosphere, currents induced in the earth from external magnetic fields, steady-state induced magnetizations and remanent magnetizations induced in crustal rocks further add to the overall magnetic field of the earth. The features of the earth's main magnetic field (attributed to the convection system of currents in the outer core) closely resembles that of a magnetic dipole deemed to have been placed at the earth's center, which is inclined to the rotation axis of the earth by about $11^{1/2^{\circ}}$. Incidentally, the geomagnetic poles (i.e., the points where the magnetic axis of the imaginary dipole meets the earth's surface) do not coincide with the magnetic dip poles (i.e., the locations where the magnetic dip attains 90°).

The direction of the main magnetic field, F, at any observation on the surface of the earth is defined by a tangent drawn to the magnetic line of force at the point. The total field, F, is resolved into the vertical (F_V) and horizontal (F_H) components directed vertically downwards and towards the magnetic north respectively (Fig. 1.1). The magnetic dip, i, at any location is defined as the angle between the horizontal and the total field vectors. In the northern hemisphere the magnetic field dips downwards, whereas in the southern hemisphere it dips upwards. By convention, the dip of the magnetic field, i, is considered positive in the northern hemisphere, and negative in the southern hemisphere. The horizontal component, F_H, is again

resolved into two components, F_{Hx} and F_{Hy}, one directed parallel and the other perpendicular to the geographic north. The vertical plane containing the total field vector, F, and two of its resolved components, F_{Hx} and F_{Hy} is defined as the magnetic meridian.

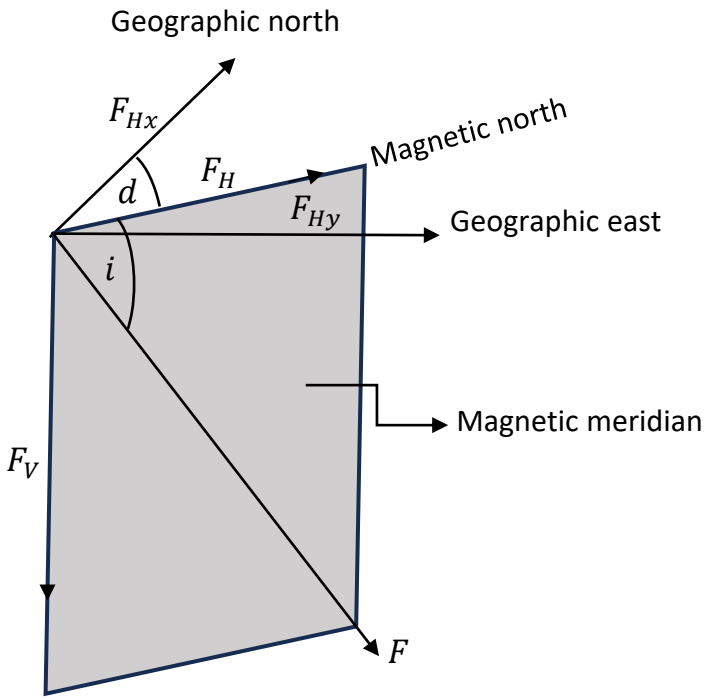


Fig. 1.1: Schematic representation of earth's magnetic field and its components in the northern hemisphere

Because the magnetic field experiences temporal variation (and hence the magnetic meridian) its declination, d, with respect to a fixed direction – the geographic north must be known to define the magnetic field at any given instance. The declination, angle between the geographic north and geomagnetic north, is considered positive if F_H strikes to the east of the geographic north and it is treated as negative if strikes west of the geographic north.

The following relations exist between the components of the earth's magnetic field

$$F_V = Fsini \tag{1.1}$$

$$F_H = Fcosi \tag{1.2}$$

$$F^2 = F_V^2 + F_H^2 \tag{1.3}$$

$$tani = \frac{F_V}{F_H} \tag{1.4}$$

$$F_{Hx} = F_H cosd = Fcosi\ cosd \tag{1.5}$$

$$F_{Hy} = F_H sind = Fcosi\ sind \tag{1.6}$$

$$F_H^2 = F_{Hx}^2 + F_{Hy}^2 \tag{1.7}$$

$$tand = \frac{F_{Hy}}{F_{Hx}} \tag{1.8}$$

Though lateral variations in magnetization of crustal rocks produce distortions in all the magnetic elements, variations in the total field (F), and two of its resolved components (F_V and F_H) are generally opted and mapped in most of the exploration activities.

1.3 Magnetic force and magnetic potential

The magnetic repulsion force between two identical poles m_1 and m_2 is given by Coulomb's law as,

$$F = \left(\frac{m_1 m_2}{\mu x^2}\right)\widehat{r_1}, \tag{1.9}$$

where, μ stands for magnetic permeability, x denotes the distance between the two poles and $\widehat{r_1}$ is the unit vector directed from m_1 to m_2.

Magnetic potential, W, is defined as the amount of work done in moving a unit north pole against the magnetic field from an infinitely long distance in non-magnetic medium ($\mu = 1$),

$$W = -\int_{\infty}^{r} \frac{m}{x^2} dx = \frac{m}{r}. \tag{1.10}$$

since monopoles do not exist, the magnetic potential due to a magnetic dipole (Fig. 1.2) can be expressed as

$$W = \frac{m}{s_1} - \frac{m}{s_2}, \tag{1.11}$$

where, m`is pole strength, s_1 and s_2 are the lengths of the radial vectors connecting two poles of the dipole to the observation, $P(s, \theta)$.

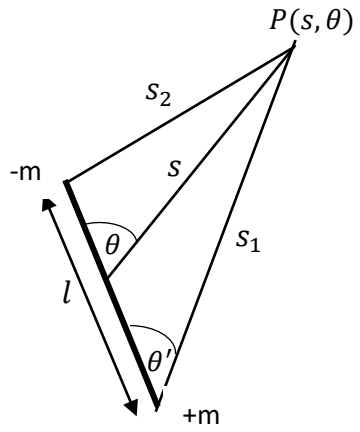


Fig. 1.2: Orientation of magnetic dipole with reference to observation, $P(s, \theta)$.

Expressing s_1 and s_2 in terms of s,

$$s_1 = s + \frac{l}{2} cos\theta', \tag{1.12}$$

$$s_2 = s - \frac{l}{2} cos\theta. \tag{1.13}$$

Here, θ' is the angle made by the radial vector, s_1, with the magnetic dipole. Substituting eqs (1.12) and (1.13) in eq (1.11), magnetic potential becomes,

$$W = -\frac{ml}{2}\left[\frac{\cos\theta + \cos\theta'}{\left(s + \frac{l}{2}\cos\theta'\right)\left(s + \frac{l}{2}\cos\theta\right)}\right]. \tag{1.14}$$

When $r \to \infty$, eq (1.14) takes the form

$$W = -\frac{ml\cos\theta}{s^2} = -\frac{M\cos\theta}{s^2}, \tag{1.15}$$

where, M stands for magnetic moment of the dipole.

A volume of magnetic material contains a mixture of magnetic dipoles resulting from

individual atoms and dipoles. They are aligned either to exhibit residual magnetism owing to its magnetic history or by induction in the presence of an external field. Both the cases, however, result in continuous distribution of dipoles with vector dipole moment per unit volume of magnitude M.

The scalar magnetic potential at P, caused by the magnetic dipole is given by,

$$W = -M\frac{\partial}{\partial\varphi}\left(\frac{1}{s}\right). \tag{1.16}$$

Here, φ denotes the direction of magnetization.

For an infinitesimal volume, dv, the magnetic moment, dm, becomes

$$dm = Jdv. \tag{1.17}$$

Here, J is the intensity of magnetization. Thus, the magnetic potential, dW, of an elementary volume is given by,

$$dW = -Jdv\frac{\partial}{\partial\varphi}\left(\frac{1}{s}\right). \tag{1.18}$$

The gravitational potential in moving a unit mass, dm_s, and density ρ to theoretically infinite distance against the force of attraction is given by,

$$dU = \frac{Gdm_s}{s}, \tag{1.19}$$

where, G is the universal gravitational constant. Since $dm_s = \rho dv$,

$$dU = \frac{G\rho dv}{s}. \tag{1.20}$$

Substituting dv obtained from eq (1.20) in eq (1.18),

$$dW = -J\frac{sdU}{G\rho}\frac{\partial}{\partial\varphi}\left(\frac{1}{s}\right). \tag{1.21}$$

The total magnetic potential, W, due to the source volume can be obtained as,

$$W = \int dW = -\frac{J}{G\rho}\int \frac{\partial}{\partial \varphi} dU = -\frac{J}{G\rho}\frac{\partial U}{\partial \varphi}. \quad (1.22)$$

Eq (1.22) is popularly known as Poisson's relation, which connects the gravity and magnetic potentials.

1.4 Magnetic anomalies

Let $\overrightarrow{F_A}$ represent the anomalous magnetic field produced by a concealed magnetic source by virtue of its presence in the earth's magnetic field, $\vec{F}$. Let θ be the angle subtended by the anomalous field vector, $\overrightarrow{F_A}$, with the earth's magnetic field vector, $\vec{F}$. Then, at any observation on the earth's surface the total field magnetometer measures the resultant magnetic vector, $\vec{T} = \vec{F} + \vec{F}_A$ (Fig. 1.3).

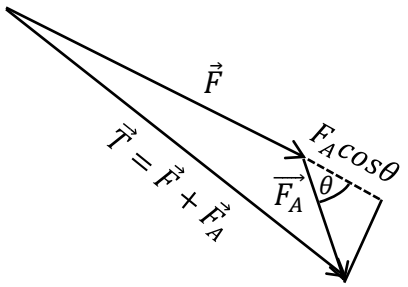


Fig. 1.3: Schematic representation of the earth's ambient magnetic field vector, and the anomalous magnetic field vector

Magnetic anomaly in total field, ΔT, becomes

$$\Delta T = \left|\vec{T}\right| - \left|\vec{F}\right|. \quad (1.23)$$

Obviously, $\Delta T \neq \overrightarrow{F_A}$.

From Fig. 1.3,

$$\vec{T}^2 = \vec{F}^2 + \overrightarrow{F_a}^2 + 2\vec{F}\,\overrightarrow{F_A} cos\,\theta. \quad (1.24)$$

If $\overrightarrow{F_A} << \overrightarrow{F}$, eq (1.24) can be expressed as

$$\vec{T}^2 \approx \vec{F}.\vec{F} + 2\overrightarrow{F}.\overrightarrow{F_A} \tag{1.25}$$

From eq (1.23), ΔT can be written as

$$\Delta T = \left|\left[\vec{F}.\vec{F} + 2\overrightarrow{F}.\overrightarrow{F_A}\right]^{1/2}\right| - \left|\vec{F}\right| \tag{1.26}$$

Applying Taylor series on eq (1.26)

$$\Delta T \approx \left|\left[\vec{F}.\vec{F}\right]^{1/2}\left[1 + \frac{\overrightarrow{F}.\overrightarrow{F_A}}{\vec{F}.\vec{F}}\right]\right| - \left|\vec{F}\right| \tag{1.27}$$

Clearly, $\left[\vec{F}.\vec{F}\right]^{1/2}$=$\left|\vec{F}\right|$.

$$\Delta T \approx \left|\vec{F}\right| + \left|\frac{\overrightarrow{F}.\overrightarrow{F_A}}{\vec{F}}\right| - \left|\vec{F}\right| \tag{1.28}$$

$$\Delta T \approx \hat{F}.F_A, \tag{1.29}$$

where, $\hat{F}$ is the unit vector along $\vec{F}$. Eq (1.29) suggests that the total field anomaly measured by a total field magnetometer is in fact the component of the anomalous field projected on to the direction of the ambient field, $\vec{F}$. Similarly, the vertical magnetic anomaly, ΔV, is defined as the projected component of the anomalous field in z direction (vertically downwards), and horizontal magnetic anomaly is the projected component of the anomalous field in x direction.

The vertical (ΔV) and the horizontal magnetic anomalies (ΔH) are related to the total field anomaly, ΔT, as

$$\Delta T = \Delta V sini + \Delta H cosi. \tag{1.30}$$

Here, i is the magnetic dip.

1.5 Magnetic surveys

In ground magnetic surveys, two types of station layouts are generally employed. In the first type, the measurements are carried out along straight lines, called profiles consisting of specific

number of stations usually placed equidistant from each other. A profile is laid such that it is more or less perpendicular to the expected strike of the structure under investigation so that maximum change in the physical field (magnetization) could be recorded along the profile. The stations on the profile are selected so as to cross the lateral dimensions of the body completely, and a sizeable number of stations are laid over less perturbed regions on either side of the body. Sometimes, the area is investigated by means of several parallel profiles crossing the body. The distance between these profiles is usually kept constant during the survey, and stations are fixed on each profile so that they form a rectangular or square grid of points on the topography. In some cases, like rugged or uneven topography, data collection in the form of grid surveys becomes difficult. In such cases, the measurements are taken at convenient locations free from cultural noise. Optimum separation for the profiles is decided on the basis of the expected linear extent of the target. Generally, it would be ideal to opt the profile interval 2 to 5 times the station interval in grid surveys. In airborne surveys, the flight altitude and separation of flight lines among which the data is to be collected are determined by the objective of survey.

1.5.1 Magnetic data corrections

Magnetic data collected from field observations require a few corrections to remove the contributions from sources other than the target of interest. The magnetometer readings are influenced by the earth's main magnetic field, magnetic field variations arise due to extraneous disturbances besides variations in geology. In order extract the anomalous magnetic field attributed to variations in geology the other two components need to be removed from the magnetometer readings.

1.5.1.1 Diurnal correction

In ground magnetic surveys a base station is first occupied and a reading is taken. The field stations are then covered sequentially, returning the magnetometer to the base station in

between at convenient intervals of time to realise the diurnal correction. The magnetometer reading corresponding to the time of field station reading is interpolated and subtracted from the respective field reading. However, reoccupation of a base/a secondary base several times a day during the field operations invariably consumes lot of time and money. Such a difficulty is overcome by continuous recording of the magnetic field at the base/secondary base by an additional magnetometer. In this case, the field magnetometer reading at the base at the start of the work is subtracted from all the other field station readings and the left outs are corrected for the variation recorded by the base magnetometer. For aeromagnetic surveys, tie lines are run perpendicular to the main flight lines. From among the main flight lines, a few control lines are also selected. At the intersections of the control and tie lines two magnetometer readings are obviously obtained, which differ due to the diurnal variation. The closure errors in the loops formed by the base and control lines are distributed along the control lines.

1.5.1.2 Normal correction

The earth's main magnetic field calculated with the International Geomagnetic Reference Field (IGRF) using latitude, longitude, date and station elevation values is subtracted from the field magnetometer reading to leave the magnetic anomaly. If the dimensions of the survey area are very small, then the N-S and E-W gradients of the normal field are calculated and multiplied with the N-S and E-W distances of the field stations measured from the base station, and added to the total change of the main field between the base and field stations.

The magnetic effects due to station elevation and the terrain are usually not considered in magnetic data reduction.

1.6 Interpretation

The magnetic data after being subjected to all the corrections are displayed either as profiles or

as contour maps. The anomalous values of the field obtained from a field survey are always complex owing to the fact that they are the cumulative magnetic effects of different sources at different depths. Geological knowledge on the study area prior to the survey and the use of other geophysical methods to investigate the same area can aid in restraining few interpretations to the observed anomalies.

Generally, interpretation of magnetic anomalies is carried out following a four-prong strategy viz., i) qualitative interpretation, ii) regional and residual anomaly separation, iii) quantitative interpretation and iv) geological translation of geophysical interpretation.

1.6.1 Qualitative interpretation

Magnetic anomalies presented in the form of profiles and contour maps yield important information about the sub-surface bodies generating the magnetic field. The anomaly maps are categorized into different magnetic character and correlated with the known geology, and the inferred correlations are extrapolated into poorly mapped regions. Magnetic anomaly of an object varies sharply if the object is shallow, whereas it will be of smoothly varying character for objects located at deeper depths. Also, the width of the anomaly depends not only on the depth but also on the lateral extent of the target being looked for. Information on major structural trends is derived from the characteristics of the anomaly axes. Flexures and termination of anomaly contours on maps reveal the presence of fault margins in the survey area. Parallel contours on an anomaly map indicate two-dimensionality of the anomalous sources, whereas closed contours reflect 3D bodies.

1.6.2 Regional-residual anomaly separation

The observed magnetic anomalies are the cumulative effects of the magnetic fields generated by sources distributed underground. The high frequency magnetic signal received from the

target of interest at shallower depths are embedded in the magnetic response of sources that are deeper or located far away, corresponding to lower frequencies. The latter field is known as regional magnetic anomaly from which the observed data is removed to give the former field known as the residual magnetic anomaly. Deep, large and far away sources produce broader wavelength anomalies and the smaller anomalies which are mostly overhead the source of interest are called noise, which are removable. The process of removing regional field from the residual anomalies is called regional and residual anomaly separation. Clearly, the authenticity and dependability of interpretation relies on the effectiveness of the regional-residual anomaly separation.

Among many other methods available, isolation and enhancement techniques become more popular to separate regional trends from the observed anomalies. Amidst them the simplest one is the graphical method ("Nettleton, 1954'; "Hinze, 1990"). Another approach which is analogous to the graphical method is the trend-surface analysis by least-squares fitting ("Agocs, 1951; Oldham and Sutherland, 1955; Skeels, 1967"). Techniques based on digital filtering ("Peters, 1949; "Griffin, 1949"; Henderson, 1960; Byerly, 1965; Fraser et al., 1966; Fuller, 1967; Grant, 1972; Gupta and Ramani, 1980"), Fourier transform ("Dean, 1958; Bhattacharyya, 1965"), reduction to pole filter ("Baranov, 1957; Baranov and Naudy, 1964"), wavelength filter ("Zurflueh, 1967; Lidiak et al., 1985"), shaded relief maps ("Chandler, 1985; Dods et al., 1985; Broome, 1986"), derivative filter ("Peters, 1949; Mesko, 1966"), continuation filter ("Peters, 1949; Henderson and Zietz, 1949; Dean, 1958; Negi, 1967; Keller et al., 1985; Yarger, 1985"), strike sensitive filter ("Chandler, 1985; Lidiak et al., 1985"), band-pass filter ("Blakely, 1996"), Wiener filter ("Pawlowski and Hansen, 1990"), stripping ("Hammer, 1963; Li and Oldenburg, 1998"), 3D magnetic inversion (Li and Oldenburg, 1996) and higher order polynomial fitting ("Chakravarthi et al., 2013") are also available to separate regional anomaly trends. Though plethora of methods are available for regional anomaly separation, *apriori* knowledge

on geology plays a crucial role in extracting meaningful residual signals from the observed anomalies.

1.6.3 Quantitative interpretation

In quantitative interpretation, the dimensions of causative sources and/or their physical properties are estimated by analyzing the anomalies based on mathematical strategies. In general, the magnetic anomaly of a geophysical geometry can be expressed as the product of size and shape parameters. The size parameter contains the information on the intensity of magnetization, whereas the shape parameter encompasses information on model parameters including the direction of magnetization. Two approaches are popular by which quantification of magnetic anomalies is realized. In the first approach, the subsurface is discretized into numerous elementary units, and the susceptibilities of such units are estimated from magnetic anomalies (Li and Oldenburg, 1996; Abedi et al., 2015). In the second category of methods, appropriate geometry is assigned to the anomalous source and its dimensions are worked out (Murthy, 1998; Ani Nibisha et al., 2022). The practical utility of many methods come under first category falls short because of the fact that these techniques consider induced magnetism alone is responsible for generating the anomalies. Over and above, the number of unknowns (susceptibilities) to be estimated from the anomalies far exceeds the number of observations, thereby, and uncertainty in the analysis also increases (Cai et al., 2018). The ambiguity in interpretation can be effectively tackled by assigning a known geometry to the subsurface target of interest. The anomalies generated from the geologic source are then inverted to parametrize the assigned model such that the model anomalies replicate the observed anomalies ("Murthy 1998; Chakravarthi et al., 2013b"). The use of information obtained from drilling and/or other geophysical data in the model space would further reduce the uncertainty to a greater extent.

1.7 Review of existing methods

Several techniques have been developed in the past to interpret the magnetic anomalies of geological sources. Some of them are based on the use of logarithmic curves ("Hutchison 1958"), harmonic analysis ("Bhattacharya, 1965"), standard curves ("Gay, 1963, 1965; Haigh and Smith, 1975; Rao and Babu, 1983"), characteristic curves ("Bruckshaw and Kunaratnam, 1963; Grant and West, 1965; Koulomzine et.al, 1970; Rao and Murthy, 1978; Telford et.al, 1990"), equivalent source techniques ("Dampney, 1969; Radhakrishna Murthy and Pradeep Kumar, 1983"), derivative based approaches like steepest descent ("Murthy et al., 1980"), Gauss-Newton approach ("Won, 1981"), Hilbert transform ("Mohan et al., 1982"), Marquardt-Levenberg ("Khurana et al., 1981; Murthy, 1990"), Werner deconvolution ("Hartman et al., 1971; Ku and Sharp, 1983"), midpoint method ("Murthy, 1985"), filtering techniques (" Stavrev, 2006; Furness, 2007; Khalil, 2014"), Euler deconvolution ("Reid et al., 1990; Gerovska and Araúzo-Bravo, 2003; Salem and Ravat, 2003"; Khalil, 2016"), analytic signal derivatives ("Salem, 2005"), genetic algorithm ("Montesinos et al., 2005"), fair function minimization ("Tlas and Asfahani, 2011"), Singular value decomposition ("Ulugergerli and Meju, 1997; Tavakoli et al., 2016"), simulated annealing ("Biswas, 2016"), particle swarm optimization ("Liu et al., 2018") etc.

Interpretation of magnetic anomalies combining linear and non-linear optimization procedures using SVD has been developed by Mirzaei and Bredewout (1996) and Tavakoli et.al. (2016). Lelievre and Oldenburg (2006) solved the forward and inverse problems by discretizing the causative body into prismatic cells of finite volume and fixed susceptibilities. Fregoso and Gallardo (2009), Wang et al. (2012) combined SVD with other methods while Bosch and McGaughey (2001) discussed the inversion technique under lithologic constraints to interpret both gravity and magnetic anomalies. Fedi and Rapolla (1999) have suggested an inversion technique using least squares method for both horizontal and vertical fields of gravity and

magnetic data through depth resolution.

Generally, inverse problem deals with the process of deriving parameters of a specified geological model from the observed geophysical signal affected by the varying properties of the earth that can reproduce the anomaly signal to an acceptable level. Local optimization techniques were used in the past to solve inversion of anomalies like steepest descent ("Paul, 1972"), conjugate gradients ("Commer, 2011; Wang and Lilley, 1999; Pilkington, 1997") etc. Owing to the fact that there are certain constraints like high nonlinear mathematical formulation ("Montesinos et al., 2016") which may limit the effectiveness of these methods, global optimization techniques have been used to overcome the limitations associated with it. Genetic algorithms ("Currenti et al., 2005, 2007; Montesinos et al., 2005; Montesinos et al., 2016"), simulated annealing ("Biswas 2016; Biswas and Acharya 2016; Biswas, 2018"), higher order derivatives ("Ekinci, 2016"), particle swarm optimization ("Desmarais and Spiteri, 2017; Godio and Santilano, 2018; Liu et al., 2018") and ant colony optimization ("Teimouri and Baseri, 2014; Yu et al., 2021") are among the few global optimization techniques available that begins with a set of initial values and then progress to find a global minimum or maximum of an objective function.

Genetic algorithm (GA) was developed based on biological evolutions like mutations and crossover where initial parameters are selected at random and then improved for fitness ("Sen and Stoffa, 2013"). Simulated annealing (SA) deals with finding global minimum or maximum of a function (cost function) that has a large number of independent variables. It is based on physical annealing process to minimalize the energy distribution that befalls when a substance is subjected to heating and the particles get randomly distributed in the liquid phase, followed by cooling to produce the crystalline lattice. Ant Colony Optimization (ACO) is a probabilistic optimization technique inspired by the pheromone trail behaviour of real ants where the artificial ants indicate multi-agent methods. ACO has slow convergence rate and

improvisations are being given to the algorithm for better results. Particle Swarm Optimization (PSO) is another progressive optimization algorithm built on the idea of social behaviour of birds and fish swarms. PSO, much like GA, starts with random solutions and optimizes by updating iterations. PSO makes use of particles (potential models) to retain memory from its neighbour to generate a new trail. PSO's have been used in parametric inversion mainly which focuses on estimating the small number of unknown parameters from specified geological models with simple geometries. This leads to large errors if PSO is used in physical property inversion. Although, few researchers have overcome this limitation and used PSO in physical property inversion (Liu et al., 2018). These algorithms are used extensively to solve inversion of gravity, magnetic and other potential methods to parametrize the geometry of buried geological bodies, greatly reducing the inherent ambiguity associated with it.

The aim of the present work as presented in the thesis is to develop i) a generalized forward modelling scheme to compute the magnetic anomalies in any component (vertical, horizontal, total) due to two-dimensional (2D) listric fault morphologies that are magnetized by arbitrary magnetisation, ii) a generalised interactive modelling to analyse the magnetic anomalies of 2D listric fault sources, and iii) an automatic inversion technique to recover the listric fault structures from the observed magnetic anomalies. The presented modelling and inversion techniques are efficient in the sense that the anomalous field measured in any component can be used to recover the fault structures. The relevant software is developed using object oriented and class-based JAVA programming language. For inversion, the ridge regression algorithm (Marquardt, 1970) is used to estimate the source parameters.

1.8 Aim and scope of the thesis

The aim of the present thesis is to develop new algorithms and related software with built-in GUI capabilities to i) compute the anomalous magnetic fields produced by arbitrarily

magnetized 2D listric fault morphologies in any component using a unified equation, ii) interactively model the anomalous magnetic fields in any component to recover the fault structures, and iii) automatically invert the magnetic anomalies to restore the fault morphologies. Accordingly, the thesis is organized into five chapters as detailed below.

Chapter 1 presents the basic principles of the magnetic method. It covers earth's magnetic field and its elements, fundamental relations between the magnetic potential and magnetic field, Poisson's relation connecting gravity and magnetic potentials etc. This chapter discusses the concept of magnetic anomaly, and the relation between total field, vertical and horizontal magnetic anomalies. Description on planning and execution of magnetic surveys, and the corrections to the magnetic observations are discussed. Discussion on the interpretation of magnetic anomalies covering both qualitative and quantitative, and review on the existing interpretation techniques are discussed in length in this chapter. This chapter ends with a section highlighting the aim and scope of the thesis.

In Chapter-2, a new generalised equation is derived and relevant software is developed to realise forward modelling of arbitrarily magnetised 2D listric fault sources. Polynomial functions of specific degree describe the non-planar nature of the fault ramps of listric fault sources. The present forward modelling operator has the advantage that the same equation can be used to compute the anomalous magnetic field of the structure in any component. The software computes and display the model response and structure geometry in user-friendly manner. The reliability of the present operator is justified by comparing the anomalous fields obtained over a 2D vertical fault structure in the vertical, horizontal and the total field with the corresponding anomalies derived using an analytic equation (Murthy et al., 2001). In addition, it is demonstrated with a theoretical model of a listric fault structure at mid-crustal level that the anomalous magnetic fields produced by such non-planar structures would deviate significantly from those fault structures with planar fault ramps. It is concluded that the use of

fault models with planar fault ramps to analyse the magnetic anomalies produced by listric fault sources should be discouraged.

Chapter-3 deals with a methodology for interactive modelling of magnetic anomalies to restore the model space geometry. Based on the methodology, a user-friendly GUI software coded in JAVA is developed and presented. This software allows one to construct the non-planar fault ramp analytically by means of a few control points. The co-ordinates of the control points are automatically assigned using which the polynomial coefficients are estimated to construct the fault plane. The model response of the structure is computed and the deviations of the theoretical response with the observed ones are minimized by modifying the model space using simple drag and drop mouse operations. The presented software is simple in operation as it enables the user to navigate through the model space to update the required changes. The applicability of the method and code are demonstrated on the synthetic magnetic anomalies in horizontal component attributed to a theoretical listric fault source.

In chapter-4, an automatic inversion technique based on the principles of optimization is presented along with the software to recover 2D fault sources having nonplanar fault planes. This technique provides a means to analyse the magnetic anomalies of the structure measured in any component (i.e., vertical, horizontal or total). The present technique uses the polynomial functions to describe the fault plane, the coefficients of which become the unknown parameters to estimate from the magnetic anomalies along with the other shape parameters, namely, the depths to the top and bottom of the fault structure, the location of the fault edge, besides intensity and direction of magnetisation. This technique initialises the model space based on some characteristic anomalies and their positions on the anomaly profile and subsequently updates them iteratively following the predefined convergence conditions. During the process of interpretation, the software displays the convergence between the observed and theoretical magnetic anomalies, model growth, and misfit decay with iteration in animated form. The

closeness of fit between the recovered and assumed theoretical models of a fault structure from the analysis of noise-added vertical magnetic anomalies justifies the strength and applicability of the proposed technique. The observed total magnetic anomalies along one EW profile across the western margin of the Perth Basin, Australia are interpreted and the results are compared with the available/reported information.

Chapter-5 epitomizes a comprehensive summary of the research findings presented in this thesis and presents scope for future research.

CHAPTER

TWO

Magnetic anomalies of 2D Listric fault sources

2.1 General

Fault is a planar or gently curved fracture, where tensional or compressional forces trigger relative displacement of rocks on either side of the fracture. Listric faults are typical normal faults, which are characterized by curved or concave-upward geometry along the fault ramp. The curvature of the fault plane arises due to diminishing fault dips against depth caused by varying amounts of extensional strain along the fault ramp (McKenzie, 1978; Smith and Bruhn, 1984; Jackson, 1987; Middleton et al. 1993; Chakravarthi, 2011). Listric faults are mostly associated with the extensional tectonic settings, where the Earth's crust is being stretched. The rotation and dipping of beds reflect the deformation style associated with extensional forces. As the beds rotate and dip, the hanging wall subsides, creating a depression in the form of basin that accumulates sediments, contributing to the overall tectonic and sedimentary history of the region. Large scale listric faults occur on passive continental margins during rifting and drifting and as sedimentary faults in deformed basins of deltaic strata. These faults also manifest in the aftermath of orogeny (late and post orogenic), occurring after the initial formation of fold belts. The formation of this type faults is attributed to the conditions involving increase in ductility in sedimentary prisms and the crust. After the formation, the faults are deformed due to various factors such as compaction of shale in footwalls, arching during upliftment of rocks beneath the fault and an enhanced tilting of the entire upthrown fault block (Roux, 1979; Shelton, 1984).

The rotation and tilting of beds indicate the distribution of strain within the hanging wall which is required to maintain the geometric compatibility of the rock layers with the evolving fault geometry. This rotation is a result of the differential movement along the fault at various depths. Near the fault surface, the movement is more vertical, while at greater depths, it becomes more or less horizontal (Shelton, 1984; Spahic et al., 2011; Zhao et al., 2020). As a consequence, the beds in the hanging wall undergo rotation to accommodate this change in displacement direction. This kind of rotation is known as antithetic and the faults are therefore named as antithetic faults (Patalakha, 1986; Erickson et al., 2001). Another characteristic feature of listric faults is that the beds in the hanging wall tend to dip towards the fault plane.

Listric faults have been extensively investigated on diversified geologic terrains across the globe for exploration of a wide range of mineral resources like copper, lead, zinc, silver, gold (Keith et al., 1983; Spencer and Reynolds, 1989; Spencer and Welty, 1986), Uranium deposits (Singh, 2013), geothermal resources (Tarits et al., 2019; Duwiquet et al., 2019) etc. The knowledge on the geometries of listric fault is important for evaluating commercially viable natural resources since movement along these faults create notable structural traps for hydrocarbons (Hardman and Booth, 1991; Bhattacharaya and Davies 2001; Lane, 2002; Goussav et al., 2006; Cong et al., 2020). Besides, listric faults also play a crucial role in seismotectonic studies. For e.g., Smith and Bruhn (1984) showed that large magnitude earthquakes nucleate near the bottom of listric faults at the changeover of brittle nature of the crust to ductile. Torizin et al. (2009) argue that the study of the nature of fault dips with increasing depth could improve the precision of seismic source characterization.

Due to the non-planar nature of fault planes, mere surface observations of very small vertical extent do not easily unveil the listric nature of fault morphology (McKenzie, 1978; Patalakha, 1986). So also, it is indeed difficult to accurately estimate the major extension and throw of

faults from surface observations on fault dips alone (McKenzie 1978; Chakravarthi 2011). In such cases, the existence of magnetization contrast(s) between the displaced/detached rock masses on either side of fault planes could generate measurable magnetic anomalies, which can be mapped and parameterized to quantify the listric fault morphologies (Ani Nibisha et al., 2021; Ani Nibisha et al., 2022). Forward modelling of magnetic anomalies is a powerful tool in geophysics which has numerous applications in mineral exploration, resource evaluation, and geological mapping. This technique involves simulating the magnetic field observed at the surface of the earth from a known distribution of magnetic sources. One of the applications of forward modelling of magnetic anomalies is in the identification and exploration of mineral deposits. Magnetic minerals such as magnetite and pyrrhotite are commonly associated with ore deposits, and their distribution in the subsurface can be mapped using magnetic surveys. By modelling the magnetic field produced by these minerals, it is possible to identify areas of high magnetic intensity caused by mineral deposits. Another application of forward modelling is in geological mapping. The distribution of magnetic sources in the subsurface can reveal information about the geological history of a region, such as the location and extent of igneous rocks, fault zones, and other geological structures (Blakley, 1996). By modelling the magnetic field produced by these sources, it is feasible to prepare detailed maps of subsurface geology to look into tectonic history of a region (Oldenburg and Li, 2005).

In this chapter, a generalized equation to compute the magnetic anomaly due to a 2D listric fault morphology in any component (i.e., horizontal, vertical, and total field) is derived (forward modelling) in the space domain. A software, FORMAG, coded in JAVA with built-in GUI interface is developed. The use of the forward modelling operator and the software are exemplified with two synthetic listric fault models. Sample input and output data files of the software are presented.

2.2 Magnetic anomaly of an arbitrary magnetized 2D listric fault source

In Cartesian co-ordinate system, let the z-axis is positive vertically downwards and x-axis traverse to the strike of a listric fault source whose geometry is shown in Fig. 2.1.

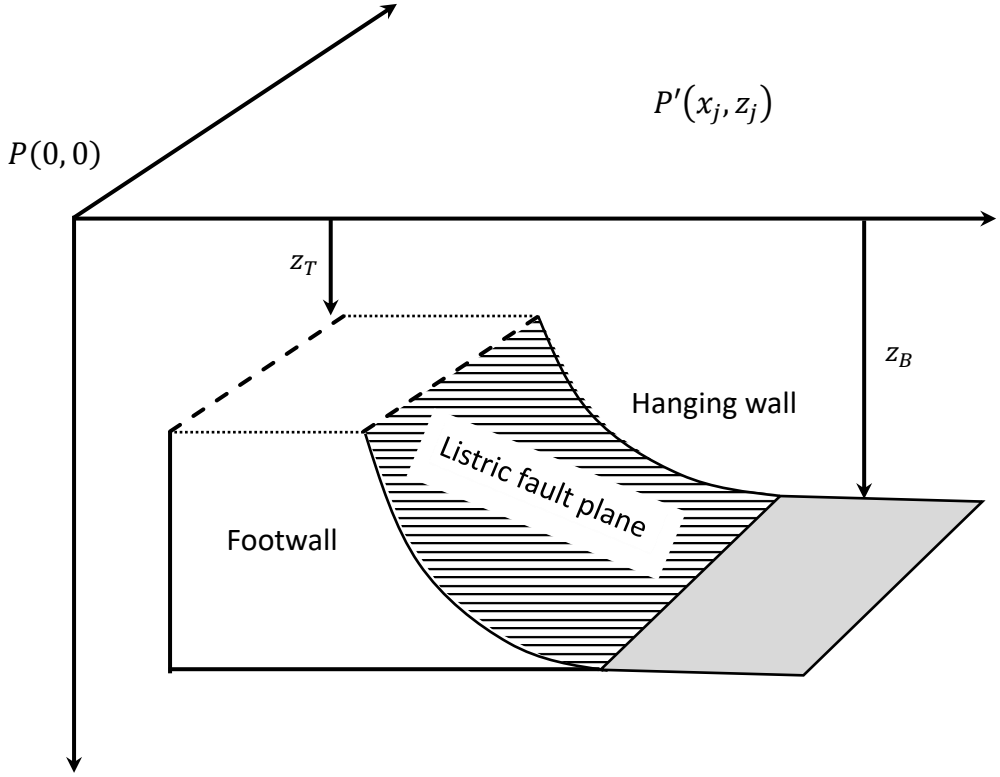


Fig. 2.1: Schematic diagram showing a conceptual geometry of a typical listric fault source, and the fault plane described by a polynomial of specific degree.

The 2D fault structure is confined in the vertical plane between the depth limits z_T and z_B ($z_B > z_T$) along the z-axis. The sedimentary infill within the hanging wall is presumed magnetically transparent, whereas the magnetic interface (fault plane) between the hanging wall and the footwall is only responsible for generating the magnetic anomalies. Further, the structure is bounded on the left by a non-planar surface defined by the function $f(z) = \sum_{i=0}^{n} f_i z^i$, and towards the right it is extending to infinity. Here, f_i represents a set of coefficients, and n stands for the degree of polynomial. Because of the fact that both induced and remanent magnetizations are responsible for generating magnetic anomalies over a geologic structure,

we presume that the structure is magnetized in an unknown direction along the resultant of both induced and remanent magnetic vectors. For a 2D source, the resultant magnetic field vector, J, can be resolved spatially into three mutually orthogonal components namely, $Jsin\theta$, $Jcos\theta cos\delta$, and $Jcos\theta sin\delta$ along the vertical, parallel to the strike of the source, and perpendicular to the strike of the source, respectively. Here, θ and δ denote the resultant magnetic dip and declination of resultant magnetic field vector measured with reference to strike of the source. Because the component resolved along the strike of the body fails to generate magnetic anomalies, the effective magnetization which is responsible for producing the anomalies can be expressed (Murthy, 1998; Ani Nibisha et al., 2021) as

$$J_{ef} = J\sqrt{(1 - cos^2\theta cos^2\delta)}. \tag{2.1}$$

The dip of the effective magnetization vector is given by

$$\theta_{ef} = tan^{-1}(\tan\theta\ cosec\delta). \tag{2.2}$$

The effective magnetization of the source causing the anomalies lies in the vertical plane perpendicular to the strike of the source.

Now considering d_c as the density contrast of the structure with reference to the undisturbed footwall, the magnetic potential, W, due to the structure at any point $P(0, 0)$ outside the source region can be expressed using the Poisson's relation (eq 1.22) as

$$W = -\frac{J_{ef}}{Gd_c}\left\{\frac{\partial U}{\partial x}cos\theta_{ef} + \frac{\partial U}{\partial z}sin\theta_{ef}\right\}, \tag{2.3}$$

where, G is universal gravitational constant, and U represents the gravity potential.

The vertical magnetic anomaly ΔV outside the source region can be expressed as

$$\Delta V = \frac{J_{ef}}{Gd_c}\left[\frac{\partial^2 U}{\partial x \partial z}cos\theta_{ef} + \frac{\partial^2 U}{\partial z^2}sin\theta_{ef}\right]. \tag{2.4}$$

The gravity potential U due to the 2D source at the point $P(0,0)$ is given by

$$U = -Gd_c \int_s \ln(x^2 + z^2)\, ds, \tag{2.5}$$

where, s is the cross-sectional area of the structure, and (x, z) stands for the source coordinates of an element within the structure. Substituting the partial derivatives of U obtained from eq (2.5) in eq (2.4), the vertical magnetic anomaly can be expressed as (Ani Nibisha et al., 2021)

$$\Delta V = 2J_{ef} \int_s \frac{(z^2 - x^2)\sin\theta_{ef} + 2xz\, cos\theta_{ef}}{(x^2 + z^2)^2} ds. \tag{2.6}$$

Applying Stokes' theorem, eq (2.6) can be rewritten as

$$\Delta V = 2J_{ef} \int_{z=z_T}^{z_B} \left[\int_{x=f(z)}^{\infty} \frac{(z^2 - x^2)\sin\theta_{ef} + 2xz\, cos\theta_{ef}}{(x^2 + z^2)^2} dx \right] dz. \tag{2.7}$$

Upon simplification, eq (2.7) takes the form

$$\Delta V = 2J_{ef} \int_{z=z_T}^{z_B} \frac{z\, cos\theta_{ef} - f(z)\sin\theta_{ef}}{f(z)^2 + z^2} dz. \tag{2.8}$$

The vertical magnetic anomaly due to the structure at any point $P'(x_j,\, z_j)$ on the topography along the principal profile can be obtained (Ani Nibisha et al., 2021) as

$$\Delta V(x_j,\, z_j) = 2J_{ef} \int_{z=z_T}^{z_B} \frac{(z - z_j)cos\theta_{ef} - (f(z) - x_j)\sin\theta_{ef}}{(f(z) - x_j)^2 + (z - z_j)^2} dz. \tag{2.9}$$

Further, the anomaly in horizontal component ΔH at the point $P(0,0)$ can be obtained from eq (2.3) (Ani Nibisha et al., 2021) as

$$\Delta H = \frac{J_{ef}\, sin\vartheta}{Gd_c} \left[\frac{\partial^2 U}{\partial x \partial z} \cos\left(\theta_{ef} - \frac{\pi}{2}\right) + \frac{\partial^2 U}{\partial z^2} \sin\left(\theta_{ef} - \frac{\pi}{2}\right) \right], \tag{2.10}$$

where, ϑ is strike of the body. The horizontal magnetic anomaly due to the structure at any point $P'(x_j,\ z_j)$ on the principal profile outside the source region can be expressed using eq (2.5) (Ani Nibisha et al., 2021) as

$$\Delta H(x_{j,}\ z_j) = 2J_{ef}\ sin\vartheta \int_{z=z_T}^{z_B} \frac{(z-z_j)cos(\theta_{ef}-\frac{\pi}{2}) - (f(z)-x_j)\sin(\theta_{ef}-\frac{\pi}{2})}{(f(z)-x_j)^2+(z-z_j)^2}\,dz. \tag{2.11}$$

From eqs (2.9) and (2.11) one can realize that the vertical magnetic anomaly produced by a 2D listric fault source is similar in form to the horizontal magnetic anomaly produced by the same structure but with a different amplitude and a phase.

The generalized equation for magnetic anomaly in any component due to a listric fault source at an observer location at $P'(x_j,\ z_j)$ can be finally expressed (Ani Nibisha et al., 2021) as

$$\Delta T(x_{j,}\ z_j) = 2J' \int_{z=z_T}^{z_B} \frac{(z-z_j)\cos\theta' - (f(z)-x_j)\sin\theta'}{(f(z)-x_j)^2+(z-z_j)^2}\,dz, \tag{2.12}$$

where, $J' = J_{ef}\sqrt{(1-cos^2\vartheta cos^2\alpha)}$

and

$$\theta' = \theta_{ef} - tan^{-1}(sin\vartheta/tan\alpha).$$

Eq (2.12) is a standard form to calculate the magnetic anomaly of a 2D listric fault source in any specific component. For e.g., setting α to 90°, 0°, and i (magnetic dip) in eq (2.12) magnetic anomalies of the fault can be realized in the vertical, horizontal, and total field components, respectively. Also, it is more appropriate to solve eq (2.12) by a numerical approach rather than an analytical approach because of the simple fact that the polynomial, $f(z)$, in the integrand may take any degree (Chakravarthi, 2010a; Chakravarthi, 2011, Ani Nibisha et al., 2021).

The validity of eq (2.12) is illustrated by comparing the anomalies (in each component) obtained from the present method against the corresponding anomalies realized from an analytical equation (Murthy et al., 2001) over a 2D vertical fault structure (Fig. 2.2b). In this case the assumed parameters of the source are z_T = 0 km, z_B= 4.0 km, θ = 30°, and ϑ = 40°. The anomalies in each component are calculated along a profile in the interval $x_j \epsilon$ (0, 40 km) on the observational plane, z_j = 0, and shown in Figs. 2.2a, 2.3a, and 2.4a respectively. The differences between the anomalies obtained from the present method and the analytic equation in each component are shown in Figs. 2.2c, 2.3b, and 2.4b.

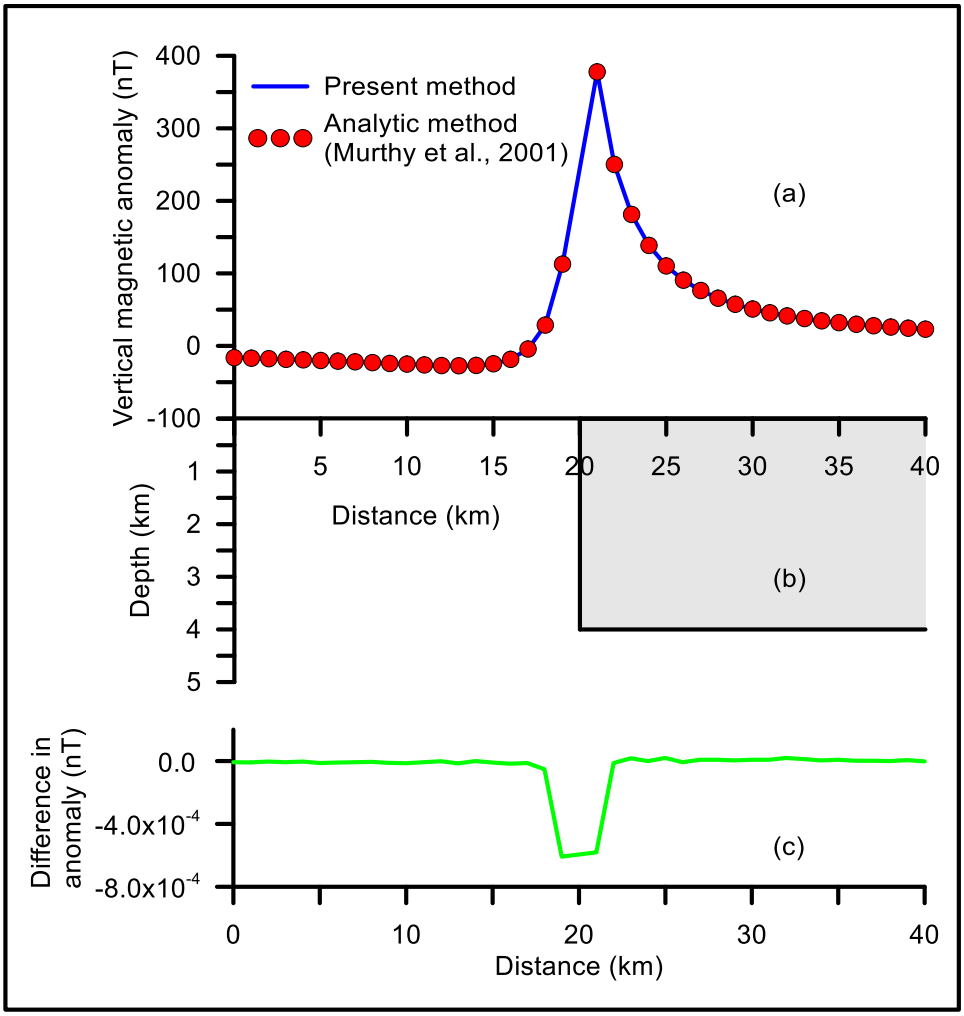


Fig. 2.2: (a) Comparison of vertical magnetic anomalies obtained from the present method (Ani Nibisha et al., 2021) and the analytic equation (Murthy et al., 2001), (b) geometry of vertical fault, (c) differences between the anomalies from the two methods.

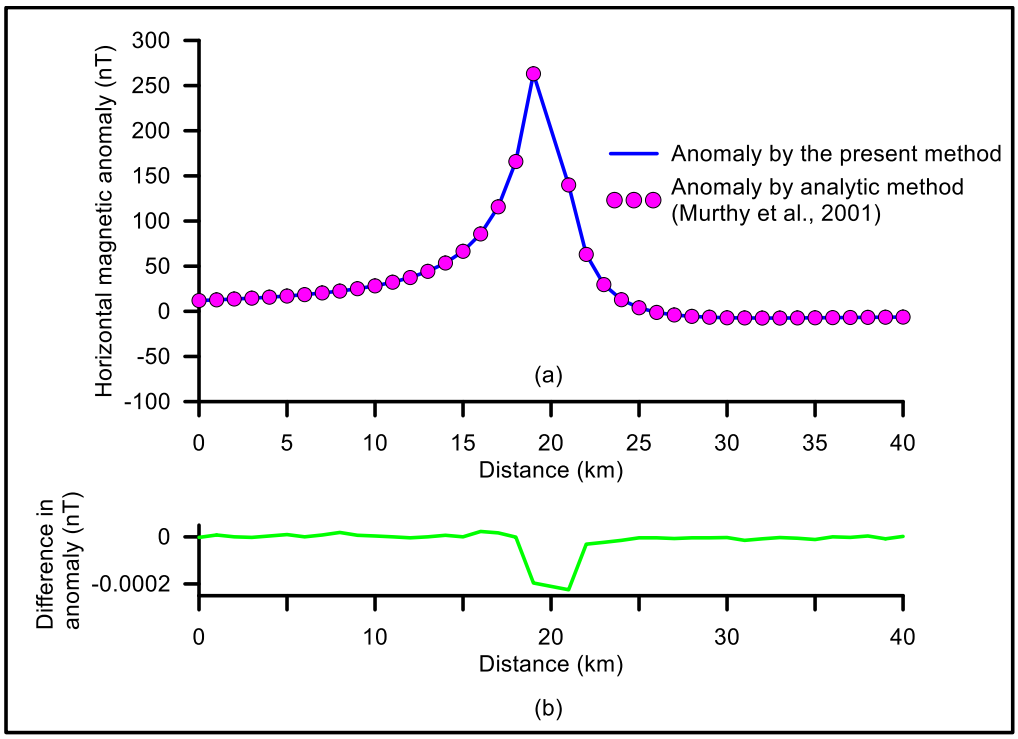


Fig. 2.3: (a) Comparison of horizontal magnetic anomalies obtained from the present method (Ani Nibisha et al., 2021) and the analytic equation (Murthy et al.,2001), (b) differences between the horizontal anomalies obtained from the two methods.

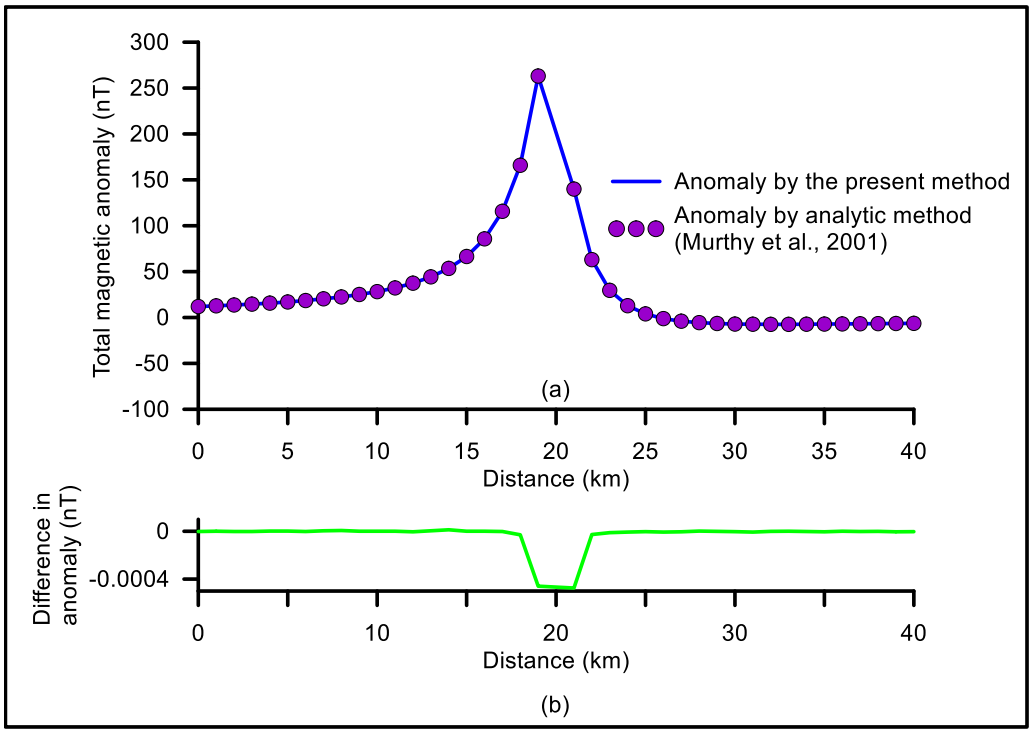


Fig. 2.4: Total magnetic anomalies realized from the present method (Ani Nibisha et al., 2021) against analytic equation (Murthy et al., 2001), (b) differences between the total anomalies from the two methods.

In the case of vertical magnetic anomaly, a maximum difference of -6E-04nT is observed at the 19th km on the profile (Fig. 2.2c), whereas in the horizontal and total components the maximum differences are -2E-04nT and -4E-04nT respectively (Figs. 2.3b and 2.4b). The insignificant differences noticed between the anomalies in each component obtained from the present method against the anomalies found from the analytic equation proves the effectiveness of the proposed method.

2.3 FORMAG - Forward Modeling Software

Based on the procedure described in the text, a GUI based software, FORMAG (Appendix 2A), is developed using object-oriented JAVA programming language to compute the magnetic anomalies of a 2D listric fault source in any component.

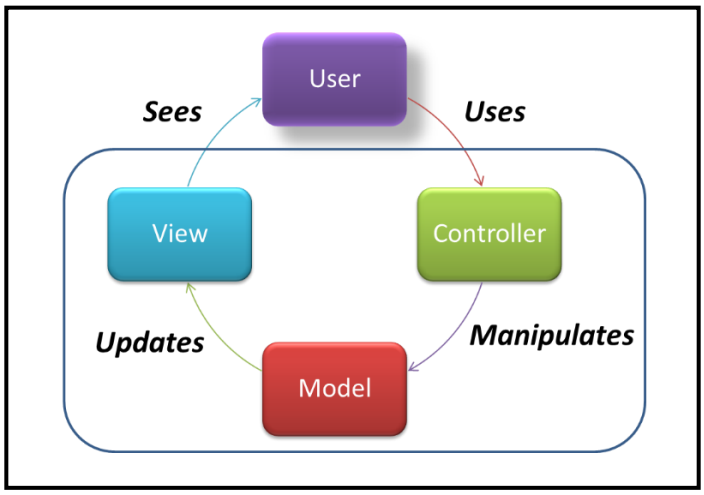


Fig. 2.5: Structure relationship between Model, View and Controller

The present software is built on Model-View-Controller (MVC) architecture as shown in Fig. 2.5. The element 'Model' constructs the model space and computes the magnetic anomalies in the required component (specified by the user by means of a code number - 1 for vertical component, 2 for horizontal component, and 3 for total field component). The 'View' module allows the user to input the data either interactively or by reading the data from an external file by making use of 'load file' option. This module displays the output in both graphical and

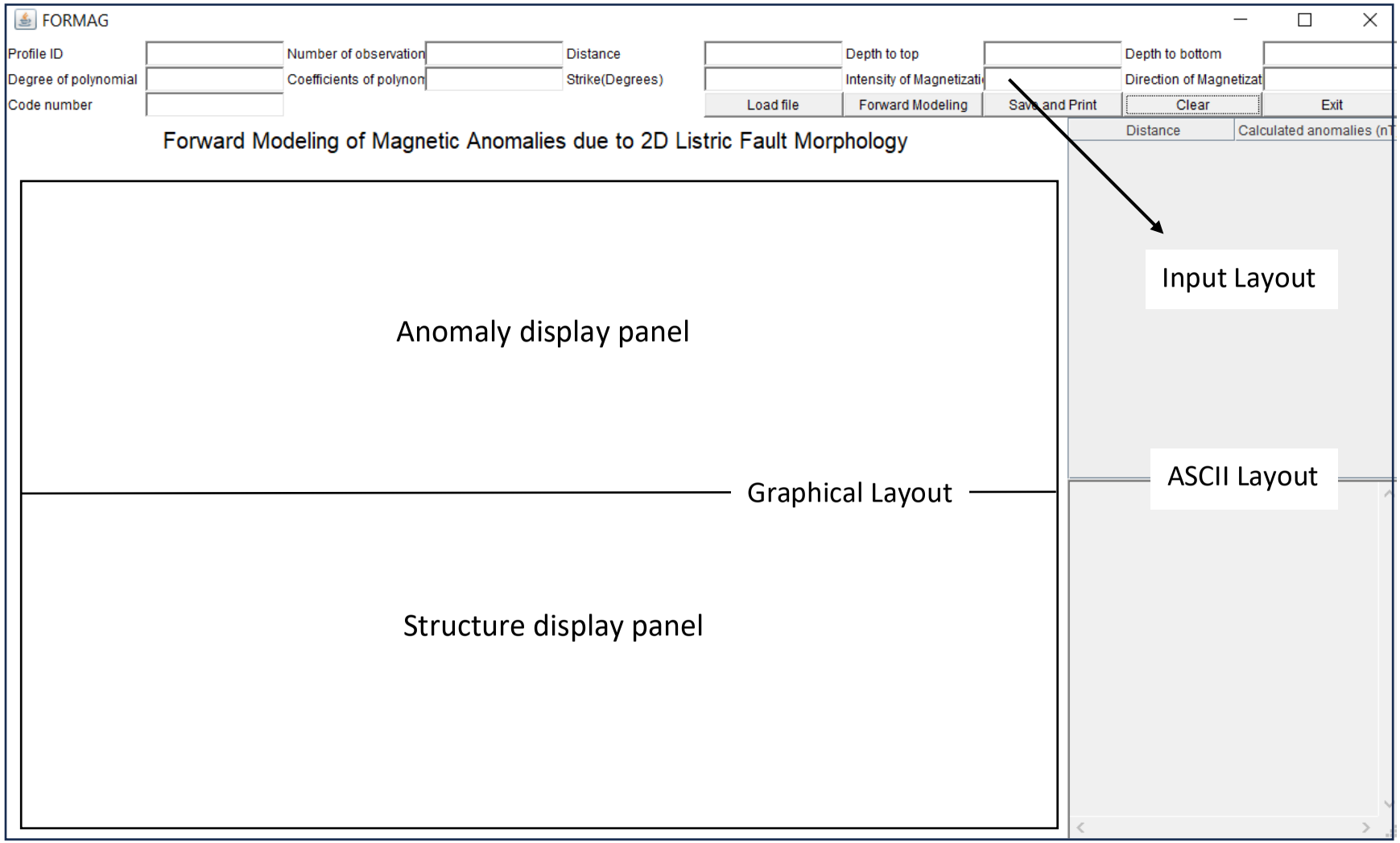


Fig. 2.6: View module of FORMAG

ASCII formats. The ‘Controller’ implements the task of passing required actions to Model and View modules whenever they are called for.

Once the batch file of the software, FORMAG, is invoked the view module appears on the screen as shown in Fig. 2.6. The view module is composed of three layouts - input, graphical, and ASCII. In turn the input layout consists of 11 input fields, and 5 action buttons (Fig. 2.6). On the other hand, the graphical layout is configured into two display panels - one for the anomaly and the other for the structure. The ASCII layout on the right side displays the output in ASCII format. The input data required for FORMAG to compute the anomaly response in any component are: profile ID, number of observations, distance to each observation (any units), depth to top of the fault, depth to bottom of the fault, degree of polynomial, coefficients of polynomial of prescribed degree, strike of the structure with reference to the magnetic north (degrees), intensity of magnetization (nT), direction of magnetization (degrees), and a code number (1 for vertical, 2 for horizontal, and 3 for total magnetic anomaly). The five action buttons are – Load file, forward modelling, save and print, clear, and exit. The forward modelling operator computes the anomalous magnetic field of the structure in the user defined component, and provides the output both in a tabular form in the ASCII layout, and graphical form in the anomaly and structure panels respectively. The ‘save and print’ option facilitates the user to save the output in html format and allows for printing.

2.4 Application

The applicability of the method and code are demonstrated on a synthetic model of a thick listric fault structure positioned at mid-crustal levels. The geometry of the assumed structure is shown in Fig. 2.7b. In this case, the fault ramp is treated as non-planar and described it by a 4^{th} degree polynomial (Fig. 2.7b), whose coefficients are given in Table 2.1.

Table 2.1: Coefficients of 4th degree polynomial, synthetic example

Coefficient	*Magnitude*
a_0	17.9734
a_1	0.60451
a_2	-0.06495
a_3	0.00374
a_4	-3.852 E-005

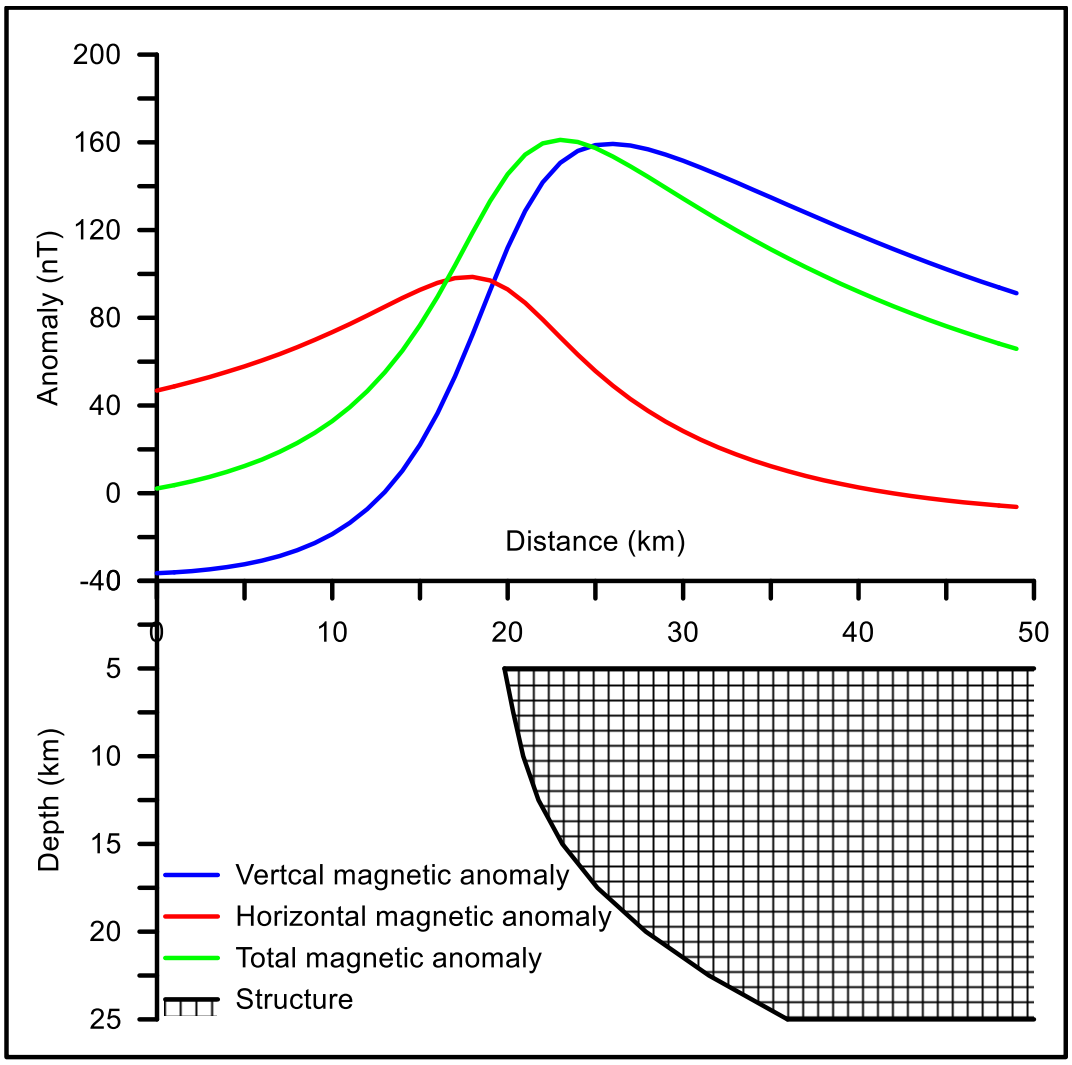


Fig. 2.7: (a) Vertical, horizontal, and total magnetic anomalies calculated from the present method over a thick mid-crustal listric fault source (b).

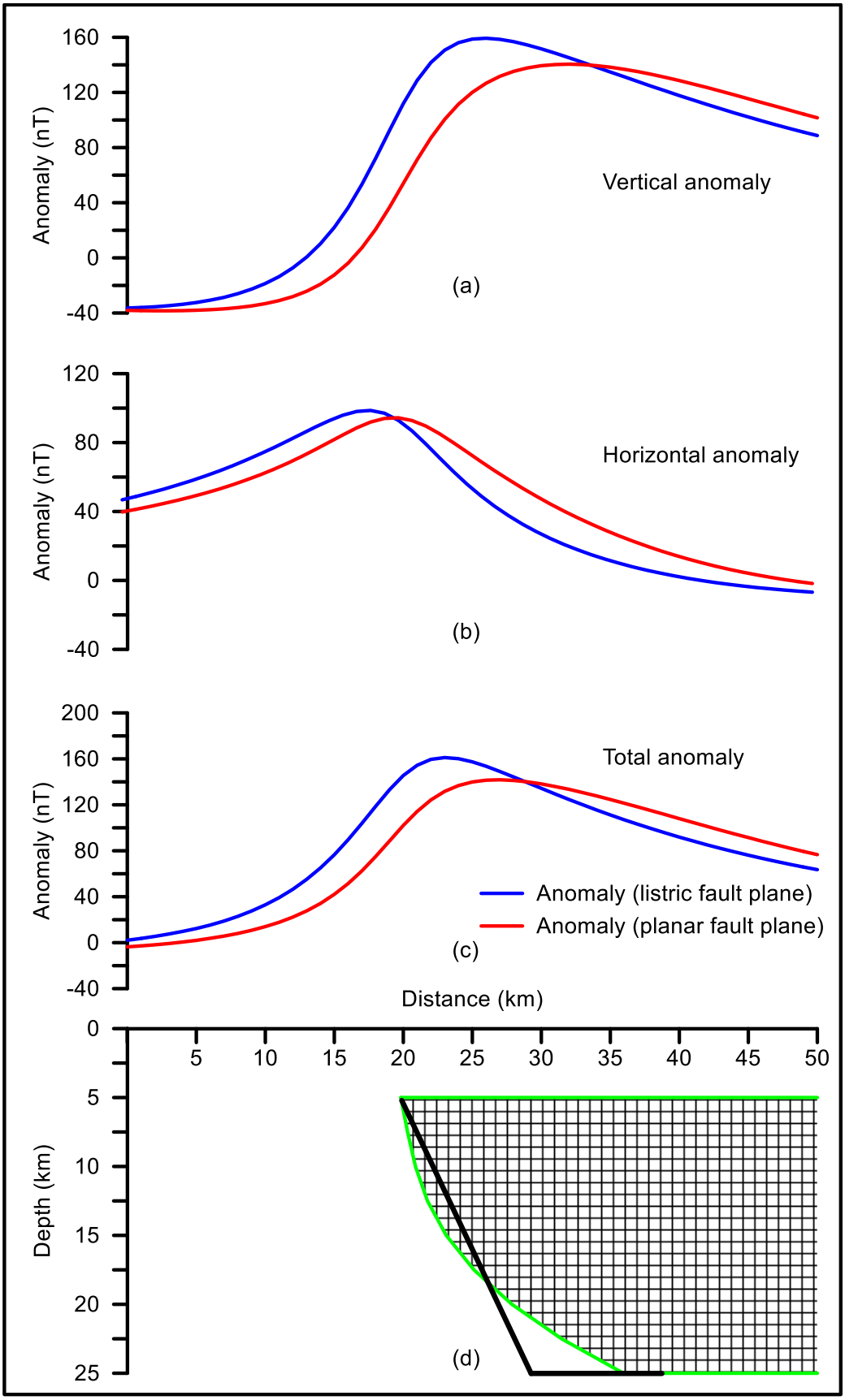


Fig. 2.8: Magnetic anomalies produced by a 2D listric fault structure and a 2D fault structure with planar fault ramp.

For such a model set up, the magnetic anomalies in the vertical, horizontal, and total field are calculated on a profile across the structure at 51 equispaced observations in the interval $x_j \in$ (0, 50 km) and shown in Figs. 2.7a. In order to examine the nature of anomalies produced by a listric fault structure and that of a fault structure with a planar fault plane, the anomalies in respective components are also calculated over the same structure (Fig. 2.7b) presuming a planar fault ramp (Murthy *et al.,* 2001). Figs. 2.8a to 2.8c compare the anomaly responses obtained from the two models. It is clearly evident from Fig. 2.8a to 2.8c that the anomalous magnetic fields produced by a listric fault structure notably differ from the corresponding anomalous fields produced by the same structure with a planar fault plane. Therefore, the assumption of planar fault plane should be accepted with caution while interpretating the magnetic anomalies caused by listric fault structures.

Sample input and output for the case of horizontal magnetic anomaly computation are given in Annexures 2B and 2C.

2.5 Conclusions

1. Both analytic and numeric approaches are used to derive a new generalized equation to compute the magnetic anomalies of an arbitrarily magnetized 2D listric fault source. The specific equation is able to compute the anomalous magnetic field of a 2D listric fault morphology in any component.
2. Based on the derived equation of magnetic anomaly, a software FORMAG, is developed using JAVA programming language to compute the magnetic anomalies in any user-defined component, and display the response and the structure geometry graphically.
3. It is revealed with a synthetic example that the magnitude of the anomalous field produced by a listric fault source, in any component, differs considerably from the

corresponding anomalous field produced by the same structure having a planar fault ramp.

4. The use of modelling and inversion algorithms which consider the fault planes as planar surfaces to analyse the magnetic anomalies of large normal faults is discouraged (Ani Nibisha et al., 2021).

CHAPTER

THREE

Interactive modelling of magnetic anomalies of 2D Listric fault sources

3.1 General

Due to structure complexity, many times listric faults are not easily detectable from measured geophysical anomalies. Sometimes, under favourable conditions the gravity method is able to detect these structures if significant lateral density changes exist between the displaced rocks. The changes in the sub-surface formation densities produce detectable gravity anomalies across the fault ramps (Chakravarthi, 2010a; Chakravarthi et al., 2017). However, large topographic undulations and heterogeneity in near-surface geology often limit the effectiveness of the gravity method. On the other hand, seismic methods can image the subsurface and detect the changes in rock properties associated with the fault zones. Still, the complex nature of listric faults would seriously distort the travel paths of the seismic waves and in turn effect the interpretation. Since fault zones are frequently associated with the magnetic minerals with contrasting magnetisations, with proper data collection, processing and interpretation the magnetic method can yield valuable information on listric faults despite their complexity.

A few techniques have been put forth to find the parameters of fault sources from the observed magnetic anomalies. The use of characteristic curves in the interpretation of magnetic anomalies had been proposed by Moo (1965), Grant and West (1965), Telford et al. (1976), Rao and Murthy (1978), and Rao et al. (1980). The use of characteristic curves may lead to

errors in parameter estimation because the characteristic distances and the amplitudes of anomalies related to target parameters are highly subjective (Al-Garni, 2016). Based on the application of Fourier integral, Sengupta (1974) had proposed a method to analyse the magnetic anomalies caused by vertical fault structures. Stanley (1977) demonstrated that the horizontal gradient of total magnetic anomaly over a vertical contact is the same as the total magnetic anomaly over a thin dyke and that specific points on the gradient are related to the fault parameters. Qureshy and Nalaye (1978) proposed a technique based on decomposing magnetic anomaly into symmetric and anti-symmetric parts. Rao and Babu (1983) presented standard curves for magnetic anomaly interpretation treating the angle of fault plane as 90° and that the magnetization is caused purely by induction. Murthy (1985) had developed an efficient method of interpreting magnetic anomalies of arbitrarily magnetized fault structures, wherein the anomalies across the structure are scaled at two different elevations followed by identifying the maximum and minimum anomalies and their mid points, which in turn were used to estimate the source parameters. Mushayandebvu et al. (2001) had developed a method using extended Euler deconvolution to analyse the anomalies produced by fault structures, while Murthy et al. (2001) used Marquardt's (1963) algorithm to analyse the magnetic anomalies. The inversion scheme proposed by Murthy et al. (2001) is noteworthy because their algorithm presumes arbitrary magnetization for the fault structures and analyses the anomalies in any component for the source parameters.

Using analytic signal and Euler deconvolution, Doo et al. (2007) had devised a technique to estimate the source parameters of a 2D magnetic contact. Subrahmanyam and Rao (2009) have suggested a simple method that uses a few characteristic positions on the magnetic anomaly to find the parameters of a fault structure. Interpretation techniques based on constrained optimization theory (Asfahani and Tlas 2004) and stochastic algorithms (Asfahani and Tlas 2007) are also available currently to analyse the magnetic anomalies of fault structures.

However, the practical utility of all the above techniques is limited to analyse the magnetic anomalies caused by large normal faults having non-planar fault planes (Ani Nibisha et al., 2021). Chakravarthi (2010a) developed a forward modelling technique to compute the gravity anomalies of listric fault sources, among which the density contrast differs continuously with depth. An automatic inversion scheme to simultaneously estimate the fault parameters (listric) and regional gravity background from a set of observed gravity anomalies was also proposed (Chakravarthi, 2011).

However, no algorithm is reported/available explicitly to analyse the magnetic anomalies caused by 2D listric fault sources (Ani Nibisha et al., 2021). Therefore, a need exists to develop a suitable technique to analyse the magnetic anomalies produced by 2D fault structures presuming (i) arbitrary magnetization for the source, and (ii) non-planar surfaces for fault planes. In this Chapter, an interactive modelling scheme is developed along with a modelling software, 2DLISMAG.

3.2 Interactive modelling of magnetic anomalies

The approach of interactive modelling starts by initialising the model space in the light of known subsurface geologic information available if any for example from boreholes (Singh, 2013), seismic imaging (Goussav et al., 2006; McKenzie and Jackson, 2012) etc. Accordingly, the model space is constructed and the magnetic anomaly response is calculated in user-defined component using eq (2.12). A polynomial function of prescribed degree describes the non-planar fault ramp analytically, which is guided by a few control points. The known co-ordinates (x, z), of the control points are fitted to the polynomial function to estimate the unknown polynomial coefficients using the equation

$$\sum_{i=1}^{NC}\sum_{m=1}^{n+1}\frac{f(z)}{\partial p_k}\frac{f(z)}{\partial p_m}P_k = \sum_{i=1}^{NC}(x_i)\frac{f(z)}{\partial p_k}, k = 1, 2, \ldots., n+1. \qquad (3.1)$$

Here, NC is number of control points, n is degree of polynomial, P_k, $k = 1, 2, \ldots., n + 1$ are the coefficients of the polynomial under consideration. These coefficients are used to construct the fault plane. The deviations between the observed and model responses are then minimized in real time by modifying the parameters interactively using simple drag and drop mouse operations (Ani Nibisha et al., 2021).

3.3 LISMAG2D

LISMAG2D is an interactive modelling software developed for analysing the magnetic anomalies caused by 2D listric faults in any component (Appendix 3A). The software is developed using the object-oriented JAVA programming language. The advantage of this software is that it is platform-independent and works on any GUI-based operating system with at least jdk1.6 version installed. The software was based on the Model-View-Controller architecture as shown in Fig. 2.5. The module – 'Model' estimates the coefficients of the prescribed polynomial to construct the geometry of a fault plane and computes the magnetic anomaly of the structure in the specific component. The 'View' module reads the input data and displays the output in both graphical and ASCII forms. The 'Controller' executes the task of passing the required actions to Model and View modules as and when they are called for.

Once the batch file is called in, the view module of LISMAG2D appears on the monitor as shown in Fig. 3.1. The input layout consists in eleven input fields and ten action buttons. The data required for the input layout are: profile ID, number of observations, distance (any units), observed anomalies (nT), depth to top (any units), depth to bottom (any units), degree of polynomial, strike of the structure with reference to the magnetic north (degrees), intensity of magnetization (nT), direction of magnetization (degrees), and code number (1 for vertical, 2 for horizontal, and 3 for total field). Alternatively, the user enters the data in a notepad and the same is called in to the program by 'load file' operator (Fig. 3.1). The action buttons of the

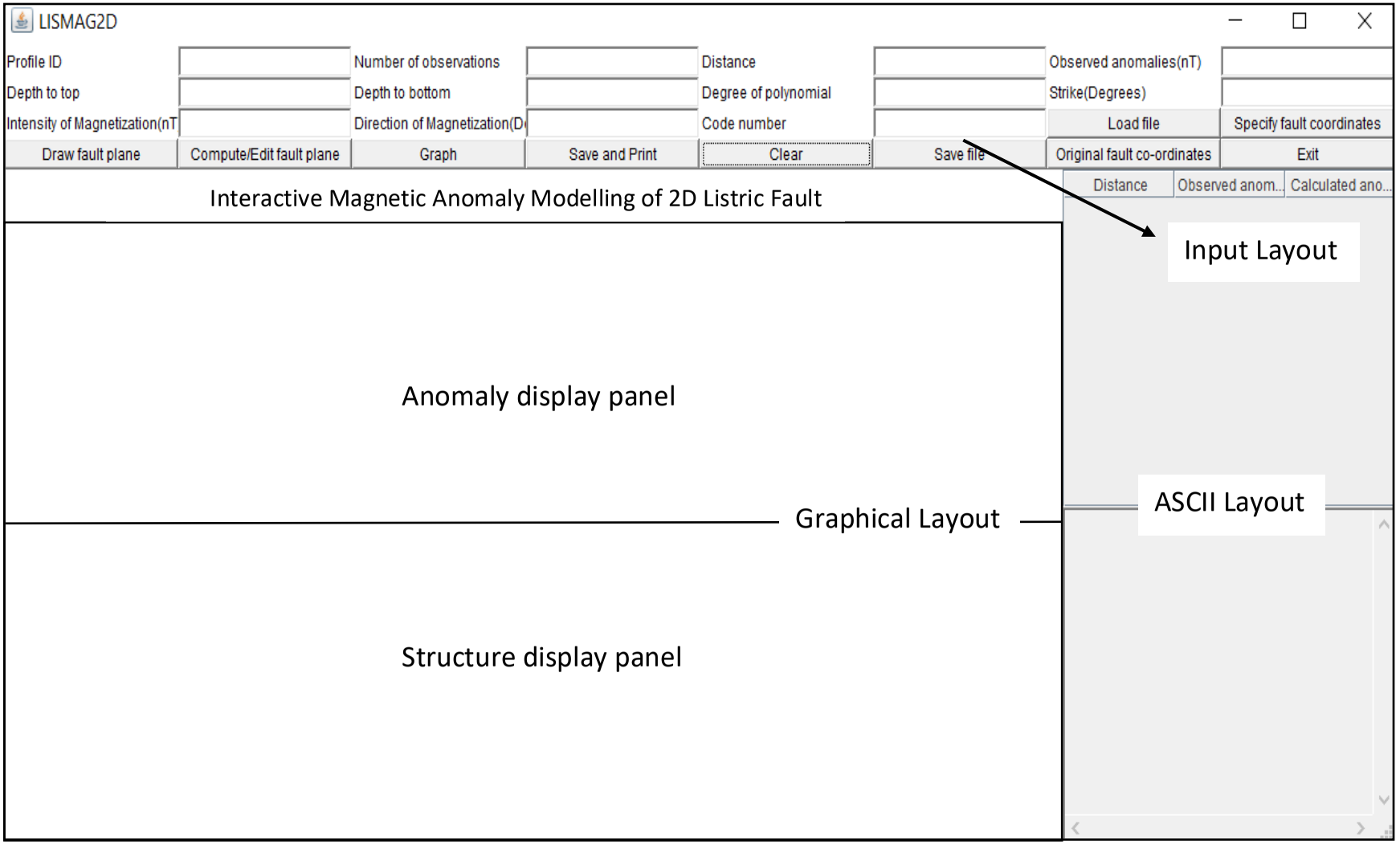


Fig. 3.1: View module of LISMAG2D

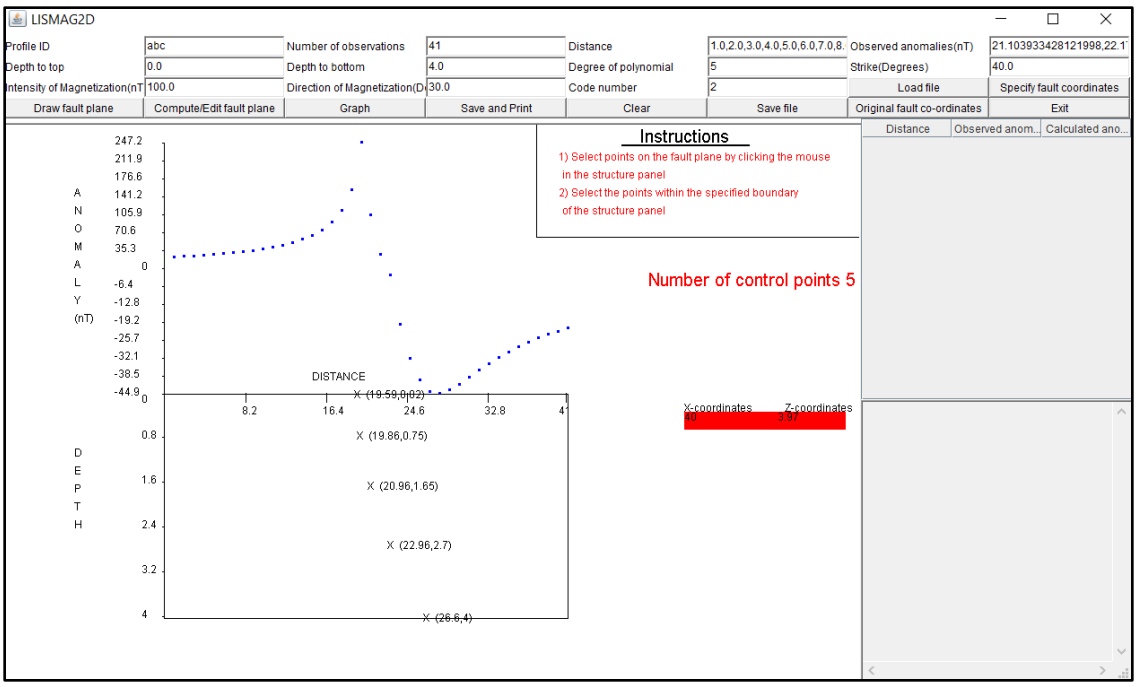


Fig. 3.2: Observed anomalies (anomaly panel) and control points (structure panel). Note that the number displaying the control points in red warrants additional control points.

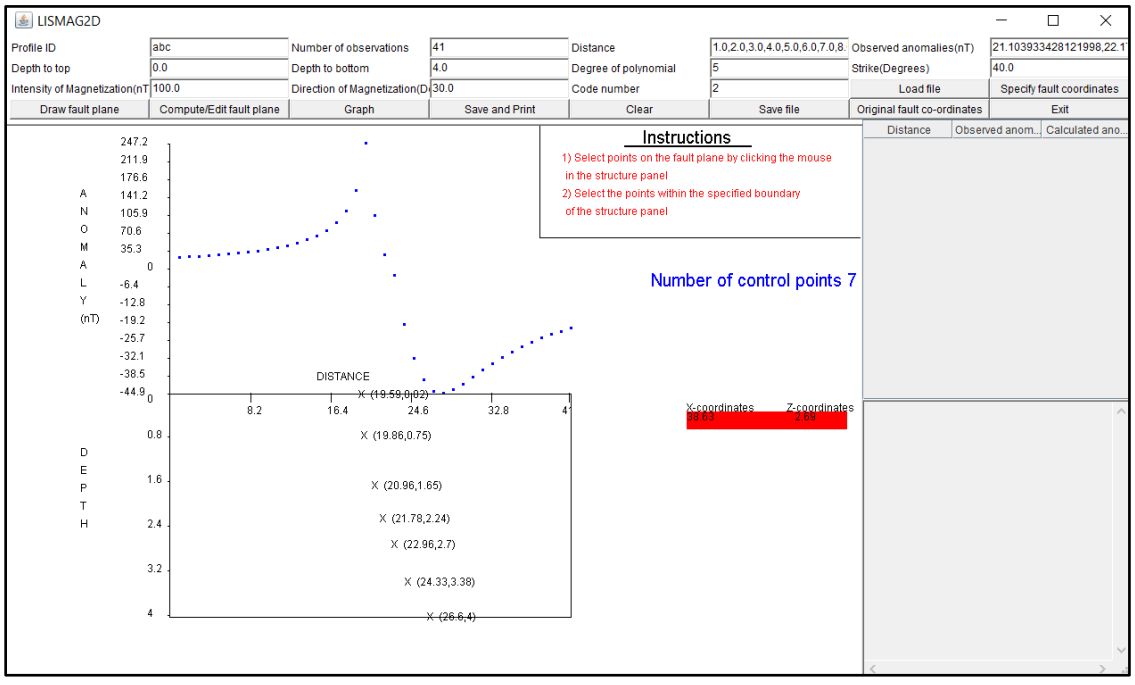


Fig. 3.3: Observed anomalies (anomaly panel) and the control points (structure panel). Display of number of control points in blue indicates selection of optimum control points.

input layout are: specify fault coordinates, draw fault plane, compute/edit fault plane, save and print (for the output), clear, save file (for input), original fault coordinates and exit.

Once the input data is entered and launch the action button – 'specify coordinates', a display panel appears on the monitor showing the observed anomalies on the top with a structure panel attached to it at the bottom (Fig. 3.2). The user selects a few control points in the structure pane by means of mouse clicks to aid the construction of listric fault plane. The code assigns coordinates to all such points, and display the coordinates and the number of control points selected (Fig. 3.2). In addition, the code notifies the user whether or not the optimum number of control points are chosen to construct the fault plane. For example, if the number of control points chosen to describe the fault plane is insufficient (depending upon the degree of opted polynomial), then the number displaying the selected points appears in red (as shown in Fig. 3.2). In such a case, the user needs to select a few more points in the structure panel till the colour of font displaying the number of points turns to blue (Fig. 3.3). Upon launching the 'draw/edit fault plane' action button, the code solves the polynomial coefficients, f_k, by fitting the prescribed polynomial to the coordinates of the control points and constructs the fault plane as shown in Fig. 3.4. Once the model space is ready, the anomalous field in a specific component (by specifying code number 1 for vertical, 2 for horizontal, and 3 for total) can be realized by activating the 'compute/Edit fault plane' operator (Fig. 3.1). The business logic computes the structure anomaly and display the response along with the observed anomaly in the anomaly panel (Fig. 3.5). The observed and computed anomalies, coordinates of control points, and the solved coefficients of the polynomial are also displayed in a tabular form in the ASCII layout (Fig. 3.5). The departures of the model anomalies from the observed ones are minimized by modifying the geometry of the fault plane by simple drag and drop mouse operations. Once the coordinates of the control points are altered the coefficients of the

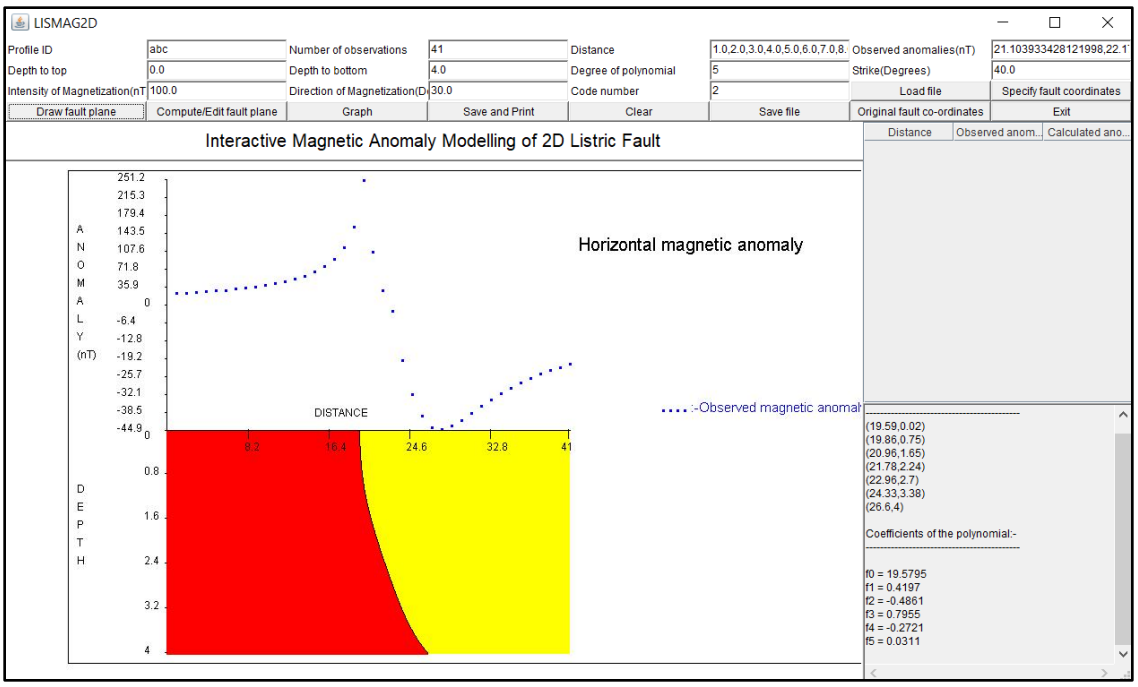


Fig. 3.4: Observed anomalies (anomaly panel) and analytically defined fault plane (structure panel). The coordinates of the control points and the estimated polynomial coefficients are displayed in the ASCII panel.

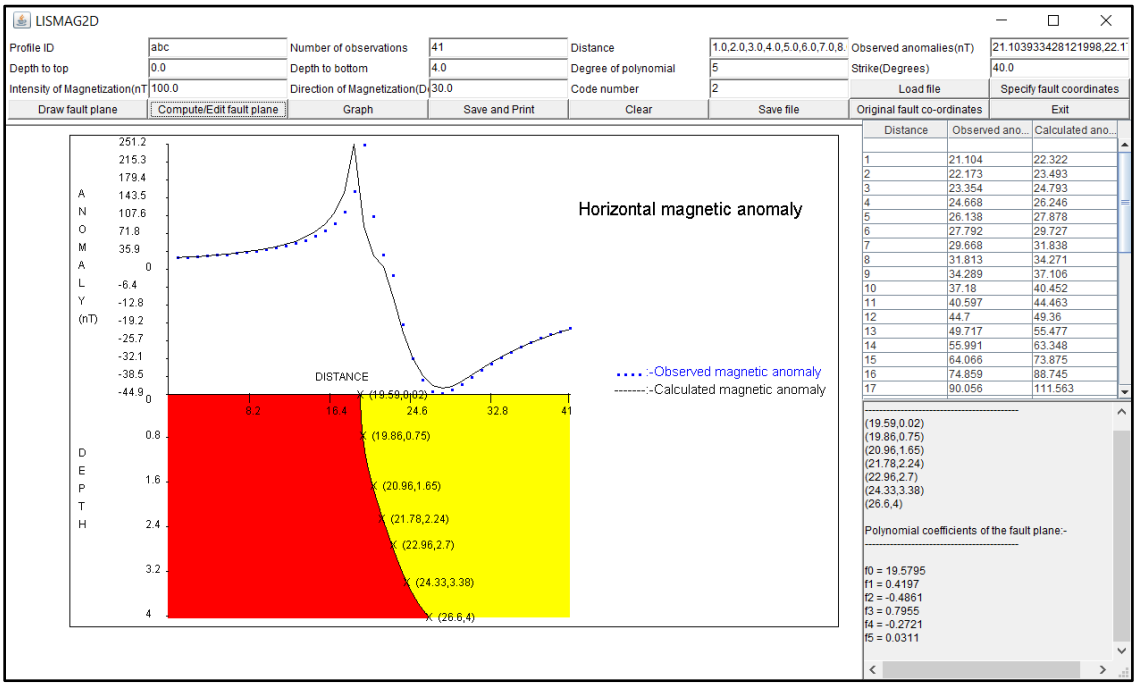


Fig. 3.5: Observed and model magnetic anomaly responses in horizontal component (anomaly panel) and model space (structure panel).

polynomial are recalculated automatically, and the structure and corresponding anomalous response are also updated and displayed in real-time as shown in Fig. 3.6.

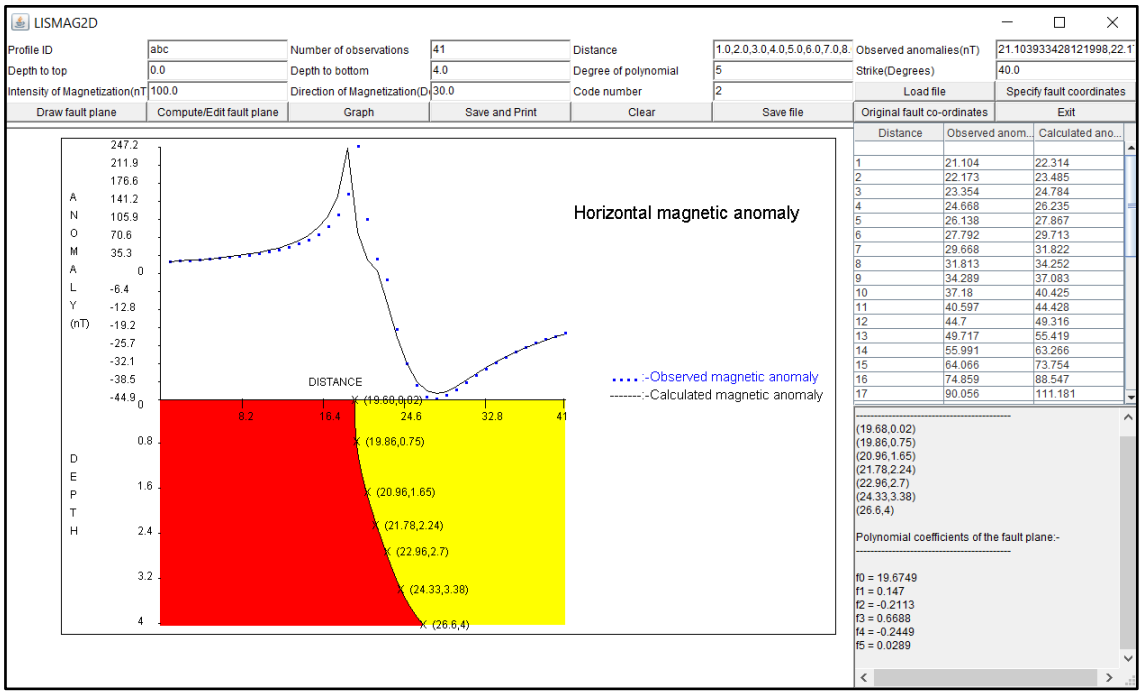


Fig. 3.6: Observed and refined model magnetic anomaly responses in horizontal component (anomaly panel) and updated model space (structure panel).

The user has an option to change other parameters as well including the degree of polynomial at any stage to realise an acceptable fit between the observed and modelled anomalies. The 'save and print' option saves the output as html file and print the output.

3.4 Applications

The method of interactive modelling and the usefulness of the software, 2DLISMAG, are demonstrated on a synthetic listric fault structure. In this case, a 5th degree polynomial with a set of 6 coefficients (Table 3.1) is used to characterize the non-planar nature of the fault plane. The fault ramp is surfaced out to the topography at the 20th km on the profile and extended downwards to a maximum depth of 4 km along the z-axis. For such a structure, the magnetic anomalies in the horizontal component are calculated on the top of topography, z_j = 0 km, at

42 equispaced observations in the interval $x_j \epsilon$ (0, 41 km) using eq (2.12) and shown in Fig. 3.7a.

Table 3.1: Coefficients of assumed 5rd degree polynomial, and coefficients of estimated 3rd degree polynomial for fault plane construction

Coefficient	*5th degree polynomial (Assumed)*	*3rd degree polynomial (Estimated)*
a_0	19.6749	19.6611
a_1	0.1470	0.0757
a_2	-0.2113	0.5332
a_3	0.6688	-0.03
a_4	-0.2449	
a_5	0.0289	

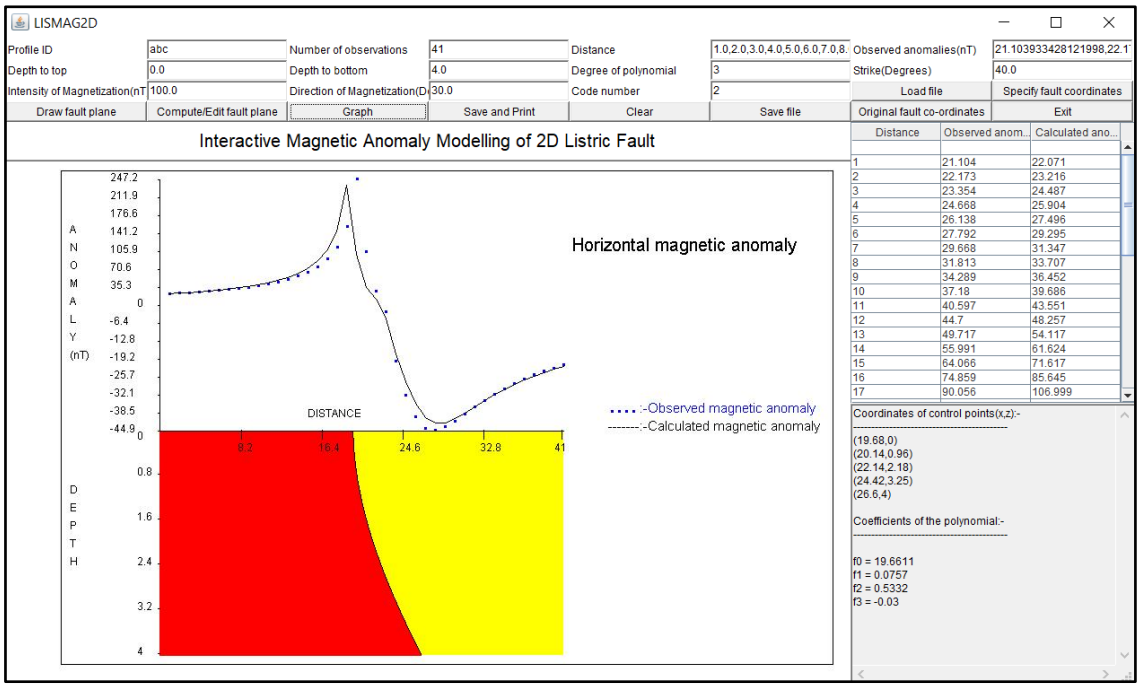


Fig. 3.7: (a) Observed and modelled anomalies (anomaly panel), and (b) derived structure (structure panel).

To perform modelling with 2DLISMAG, these theoretical anomalies are contemplated as the observed anomalies to recover the structure. In reality, it is difficult to choose the optimum polynomial to model the fault ramp if additional information on the subsurface is not available. For this reason, in this modelling a 3rd degree polynomial is considered in place of the 5th degree to simulate the fault plane. The estimated coefficient of the 3rd degree polynomial are given Table 3.1. The sequence of modelling steps described in section 3.3 is followed to model the structure and the result is shown in Fig. 3.7.

Sample input and output files are given in Annexures 3B and 3C, respectively.

3.5 Conclusion

1) An interactive modelling methodology and related GUI software, LISMAG2D, coded in JAVA are presented to quantify the magnetic anomalies generated by 2D listric fault sources.
2) The advantage of LISMAG2D is that it can be used to model the magnetic anomalies of listric faults irrespective of the component the anomalies are measured.
3) The validity of the proposed modelling and software are epitomised with a synthetic model of a 2D listric fault structure, where the nonplanar fault ramp is described with a 5th degree polynomial to generate the theoretical anomalies. Theses anomalies are treated as the observed anomalies in the interactive modelling to recover the structure. In doing so, the fault plane is replicated with a 3rd degree polynomial, keeping in view that in real world it is always difficult to choose the optimum polynomial in the absence of additional independent information.

CHAPTER

FOUR

Automatic inversion of magnetic anomalies caused by 2D Listric fault sources

4.1 General

More than not faults that are originated as a result of rifting, drifting, and collapsed orogeny are listric in nature. Though fault planes of large normal faults are often nonplanar, many interpretation techniques have been reported considering the fault planes as planar surfaces to analyse the potential field anomalies resulted from them (see for e.g., Sundararajan et al., 1983; Rao, 1985; Murthy et al., 1980; Rao and Babu, 1983; Raju et al., 1998; Murthy, 1998; Murthy et al., 2001; Abdelrahman et al. 2003; Chakravarthi and Sundararajan, 2004; Essa, 2013; Abdelrahman and Essa, 2015; Anderson et al., 2020; Elhussein, 2021; Essa et al., 2021; Essa, 2021; Rao and Biswas, 2021).

In recent past, techniques are also published to calculate the gravity and magnetic responses of listric fault sources. For example, Chakravarthi (2010a), Chakravarthi (2010b) presented a set of formulae to compute the gravity responses of 2D and 2.5D fault structures, where analytic functions were used to simulate the geometries of fault planes. A few inversion schemes to quantify listric fault sources from measured gravity anomalies are also available. Chakravarthi (2011) proposed an inversion technique to infer the shape parameters of 2.5D listric fault structures and regional gravity field from measured gravity anomalies. This technique is efficient to model the listric fault morphologies, where the disconnected hanging wall

comprises in thick-sectioned sediments with progressive increase in density with increasing depths. Another optimisation proposed by Chakravarthi and Pramod Kumar (2015) simultaneously estimates from the gravity anomalies the fault plane geometries and densities of several lithologic units or their thicknesses within the hanging wall. Roy et al. (2022) have used the particle swarm optimisation (PSO) to interpret the gravity anomalies of listric faults by simulating the fault planes with quadratic Bezier curve.

It is pertinent to note that not many techniques are developed in magnetics for fault model interpretation because the unknown model parameters to be solved from the magnetic anomalies are generally large in number, particularly when the direction of magnetisation is presumed as arbitrary (Murthy et al. 2001; Ani Nibisha et al., 2022). Making use of the symmetry of the analytic function representing the magnetic anomalies of dyke and vertical fault models, Powell (1967) devised a method to interpret the magnetic anomalies of fault structures, whereas Sengupta (1974) used the Fourier integral to analyse the anomalies. Following Koulomzine et al. (1970), Qureshi and Nalaye (1978) proposed the decomposition of the magnetic anomaly profile into even and odd components for fault model interpretation. Green (1979) developed a harmonic method that uses the locations of two turning points and two inflection points on the magnetic gradient profile to interpret the anomalies caused by a contact.

Murthy (1985) used the properties of midpoints between the maximum and minimum anomalies on the profiles at two selected heights, the distances between the said anomalies, and the ratio of horizontal to vertical gradients of the anomalies to decipher arbitrarily magnetised fault structures. Murthy et al. (2001) applied the Ridge Regression algorithm in an inversion to quantify the magnetic anomalies caused by fault sources in any component. This technique initialises the fault structure using the maximum and minimum anomalies and their positions on the anomaly profile. These initial model parameters are then refined iteratively till

an acceptable predefined convergence is achieved between the observed and modelled anomalies. Stavrev (2006) had proposed the concept of magnitude transform that makes use of both magnetic field components and the first-order derivatives to invert magnetic anomalies of simple geometric shapes. Subrahmanyam and Rao (2009) proposed a method to infer fault parameters using characteristic points on the magnetic profile. An inversion method based on modular neural network was proposed by Al-Garni (2016) to decipher fault parameters. Essa and Elhussein (2018) employed the Particle Swarm Optimization (PSO) in the analysis of magnetic anomalies of simple geometric shapes. Ekinci et al. (2019) applied both differential evolution algorithm (DE) and particle swarm optimisation (PSO) on the gravity and magnetic anomalies caused by deep-seated faults and reported that DE is more efficient than PSO in terms of convergence rate, model space recovery, and robustness. Recently, Biswas and Rao (2021) have used simulated annealing (SA) to interpret the anomalies due to fault and sheet-type models. However, the intrinsic assumption that the fault planes are either vertical or planar in the aforementioned methods limit their utility to analyse the anomalies caused by fault sources that display listric behaviour with depth.

Therefore, it is necessary to design a new inversion strategy to determine the parameters of listric fault structures from the measured magnetic anomalies. Here, a generalised inversion technique and a software 2DINMGLS are presented to quantify the listric fault sources from the magnetic anomalies observed in any component, viz., vertical, horizontal or total field. The present inversion utilises the scheme of Ani Nibisha et al. (2021) for anomaly calculation. The proposed technique is applied on the magnetic anomalies of a synthetic listric fault structure in the presence of pseudorandom noise, followed by its application to analyse the real field anomalies across the western margin of the Perth Basin, Australia.

4.2 Inversion of magnetic anomalies

In inversion, theoretical anomalies that arise from a starting model are fitted to the observed anomalies and based on the misfit between the observed and theoretical anomalies, the parameters of model space are adjusted iteratively within the permissible limits, so that the optimum model becomes geologically acceptable. The present inversion utilises eq (2.12) for forward modelling of magnetic anomalies of listric fault sources in any component (Ani Nibisha et al., 2021). In the present case, the size factor, $2J'$, in eq (2.12) is independent of the source body parameters, hence, the magnitude of J_{ef} can be determined from the size factor, in principle. On the other hand, the shape of the anomaly profile in any component over the structure with a known strike is influenced by the body parameters, viz., z_T (depth to the top of the fault plane), z_B (depth to the bottom of the fault plane), coefficients of the function, $f(z)$, $f_i, i = 0, 1, 2, .., n$, and θ_{ef}. In other words, the body parameters of the structure and θ_{ef} can be estimated from the shape of the anomaly profile.

In general, the process of any geophysical inversion starts by initialising a model space with a set of approximate shape parameters coupled with appropriate physical property contrast/s. Usually, these initial parameters are specified by the interpreter as in the case of interactive modelling (section 3.2). At times, the starting model is described with a simple geometry, thereby enabling the interpreter to instruct the computer to identify some characteristic anomalies and their locations on the profile, which are then used to initialise the source. In the proposed inversion, approximate/ initial body parameters are obtained by treating the fault source as a vertical step (Murthy et al., 2001, Ani Nibisha et al., 2022). Such an approximation was also proposed by Chakravarthi (2011) to analyse the gravity anomalies resulted from strike-limited listric fault sources. For a vertical step, the approximate distance to the top of the fault edge, D_{App}, on the profile can be found at an observation, where the magnitude of the

anomalous field equals the sum of the minimum ((ΔT_{min}) and maximum (ΔT_{max}) anomalies (Murthy et al., 2001; Ani Nibisha et al., 2022). In case the anomaly profile is symmetric about the anomaly axis with a single turning point, then the observer location corresponding to the maximum anomaly is assigned to D_{App}. The value of θ' is obtained as (Murthy et al., 2001; Ani Nibisha et al., 2022)

$$\left.\begin{aligned} \theta' = \arctan\frac{2\sqrt{\frac{\Delta T_{min}}{\Delta T_{max}}}}{\left(1-\frac{\Delta T_{min}}{\Delta T_{max}}\right)}, \text{for } 0.05 \leq \frac{\Delta T_{min}}{\Delta T_{max}} \leq 0.55 \\ \theta' = \frac{\pi}{2}, \text{for } \frac{\Delta T_{min}}{\Delta T_{max}} > 0.55 \\ \theta' = 0°, \text{for } \frac{\Delta T_{min}}{\Delta T_{max}} < 0.05. \end{aligned}\right\} \quad (4.1)$$

The value of θ' calculated from eq. (4.1) is placed in the appropriate quadrant based on the plus-minus method (Murthy et al., 2001). Eq. (4.1) conveys that θ' is dependent only on the magnitudes of minimum and maximum anomalies. Also, the approximate depth to the top of the fault edge is obtained as

$$\left.\begin{aligned} z_{TApp} = \frac{|X_{max} - X_{min}| \sin\theta'}{2\sqrt{9 - 4\,sin^2\theta'}}, \text{for } 0.05 \leq \frac{\Delta T_{min}}{\Delta T_{max}} \leq 0.55 \text{ or } \frac{\Delta T_{min}}{\Delta T_{max}} > 0.55 \\ z_{TApp} = 0.224\,X\,W, \text{for } \frac{\Delta T_{min}}{\Delta T_{max}} < 0.05, \end{aligned}\right\} \quad (4.2)$$

where, W is the distance between the two half-maximum anomalies on the profile. Depth to the bottom of the fault plane is initialized as,

$$z_{BApp} = 8Xz_{TApp}. \quad (4.3)$$

A ratio of 8 for $\frac{z_{BApp}}{z_{TApp}}$ is found satisfactory to invert the magnetic anomalies of listric fault sources with $2 \leq \frac{z_B}{z_T} \leq 15$ (Ani Nibisha et al., 2022).

It is to note that for a vertical step, the coefficients, $f_i, i = 1, 2, .., n$ become zero. Hence, the magnetic anomaly, $\Delta T(x_j)$, in eq (2.12) takes the form

$$\Delta T(x_j, z_j) = C_1 \int_{z=z_T}^{z_B} \frac{(z - z_j)}{(f_0 - x_j)^2 + (z - z_j)^2} dz - C_2 \int_{z=z_T}^{z_B} \frac{(f_0 - x_j)}{(f_0 - x_j)^2 + (z - z_j)^2} dz. \quad (4.4)$$

where, $C_1 = 2J' \cos\theta'$, and $C_2 = 2J' \sin\theta'$.

As many linear equations (similar to eq 4.4) are framed as the number of observations, and two normal equations are constructed and solved for C1 and C2. From the estimated values of C_1 and C_2, the value of J' can be obtained as

$$J' = \frac{\sqrt{C_1^2 + C_2^2}}{2} \quad (4.5)$$

Eq (2.12) computes the theoretical anomalies of the model space at all observations, x_j, j = 1, 2,.., N_{obs}. In this process, D_{App} value is initially assigned to f_0, and all other coefficients are set to zero (i.e., $f_i = 0$, i = 1, 2, …, N). The theoretical anomalies, $\Delta T_{cal}(x_{j,}\, z_j)$, of the initial structure obtained from eq (2.12) obviously deviates from the observed anomalies, $\Delta T_{obs}(x_{j,}\, z_j)$. The misfit between the two anomalies can be quantified as (Ani Nibisha et al., 2022)

$$\hat{J} = \sqrt{\sum_{j=1}^{N_{obs}} \frac{\left[\Delta T_{obs}(x_{j,}\, z_j) - \Delta T_{cal}(x_{j,}\, z_j)\right]^2}{N_{obs}}}. \quad (4.6)$$

The deviation of theoretical anomalies from the observed ones can be written using Tylor series approximation as

$$\Delta T_{obs}(x_{j,}\, z_j) - \Delta T_{cal}(x_{j,}\, z_j)$$

$$= \frac{\Delta T(x_{j,}\, z_j)}{\partial z_T} dz_T + \frac{\Delta T(x_{j,}\, z_j)}{\partial z_B} dz_B + \frac{\Delta T(x_{j,}\, z_j)}{\partial f_0} df_0 + \frac{\Delta T(x_{j,}\, z_j)}{\partial f_1} df_1$$

$$+ \frac{\Delta T(x_{j,}\, z_j)}{\partial f_2} df_2 + \cdots \frac{\Delta T(x_{j,}\, z_j)}{\partial f_n} df_n \tag{4.7}$$

A total of N_{obs} linear equations in number are framed from which $(n+3)$ normal equations are built and solved for $(n+3)$ unknown parameters using the Marquardt algorithm (Marquardt, 1970; Chakravarthi, 2003; Ani Nibisha et al., 2022). The normal equations can be put in the form (Ani Nibisha et al., 2022)

$$\sum_{i=1}^{N_{obs}} \sum_{m=1}^{n+3} \frac{\Delta T(x_{i,}\, z_i)}{\partial p_k} \frac{\Delta T(x_{i,}\, z_i)}{\partial p_m} (1 + \xi\lambda) dp_k =$$

$$\sum_{i=1}^{N_{obs}} Err(x_{i,}\, z_i) \frac{\Delta T(x_{i,}\, z_i)}{\partial p_k}, k = 1, 2, \ldots., n+3. \tag{4.8}$$

Here, λ is the damping factor, and $Err(x_{i,}\, z_i) = [\Delta T_{obs}(x_{i,}\, z_i) - \Delta T_{cal}(x_{i,}\, z_i)]$, $i = 1, 2, \ldots., N_{obs}$. Also, $\xi = 1$ for $m = k$, otherwise 0. Equation (4.8) can be put in a matrix form as

$$(A + \lambda I)\, X = B, \tag{4.9}$$

where, the elements, A_{ik}, of matrix A are expressed as

$$\left.\begin{aligned} A_{ik} &= \sum_{i=1}^{N_{obs}} \sum_{m=1}^{n+3} \frac{\Delta T(x_{i,}\, z_i)}{\partial p_k} \frac{\Delta T(x_{i,}\, z_i)}{\partial p_m}, k = 1, 2, \ldots., n+3, \\ B &= \sum_{i=1}^{N_{obs}} Err(x_{i,}\, z_i) \frac{\Delta T(x_{i,}\, z_i)}{\partial p_k}, k = 1, 2, \ldots., n+3, \\ X &= dp_k, k = 1, 2, \ldots., n+3. \end{aligned}\right\} \tag{4.10}$$

Here, I is the diagonal matrix containing the elements of A. Also, dp_k, $k = 1, 2, \ldots, n+3$ are the estimated improvements in the model parameters, p_k. Here, $dp_1 = dz_T$, $dp_2 = dz_B$, $dp_3 = df_0$, $dp_4 = df_1$, ……, $dp_{(n+3)} = df_{(n+3)}$. Eq (4.9) conveys that the diagonal elements of matrix A are multiplied by (1+ λ). Further, it is to note that the polynomial degree, n, should be chosen in such a way that the number of unknowns doesn't exceed the number of observations. The partial derivatives of the anomaly in eq (4.10) are obtained by numerical differentiation (Chakravarthi et al., 2001; Ani Nibisha et al., 2022).

To start with, the initial data misfit, $\widehat{f_A}$, is obtained from eq (4.6). The damping factor, λ, is set to 0.5 (Chakravarthi, 2003) and eq (4.9) is worked out for $dp_k, k = 1, 2, \ldots, n+3$. The existing source parameters, P_k, $k = 1, 2, \ldots, n+3$ are updated to P_M, $M = 1, 2, \ldots, n+3$ by adding/subtracting $dp_k, k = 1, 2, \ldots, n+3$ to P_k. The theoretical response of the improved model is again calculated, and the corresponding misfit, $\widehat{f_N}$, is estimated. If the magnitude of the new misfit, $\widehat{f_N}$, falls below its preceding value, $\widehat{f_A}$, then λ is reduced by a factor of 2. In such a case, $\widehat{f_N}$ is assigned to $\widehat{f_A}$ and P_M to P_k and the process repeats iteratively. By any chance if the misfit at the end of any specific iteration exceeds its preceding value, then the existing value of λ is multiplied by 2, and eq (4.9) is worked out again for the values of dp_k. This process continues within the specific iteration till the resulting misfit falls below its preceding value. The inversion process automatically terminates upon the completion of specific number of iterations, or the prevailing misfit at the end of any iteration equals to or less than the prescribed threshold, or the current value of, λ, is abnormally large (Chakravarthi, 2003, Ramamma et al. 2021, Ani Nibisha et al., 2022).

4.3 INVMGLSTRK

Based on the above inversion methodology a software named, INVMGLSTRK, is developed using the JAVA programming language to interpret the magnetic anomalies caused by 2D

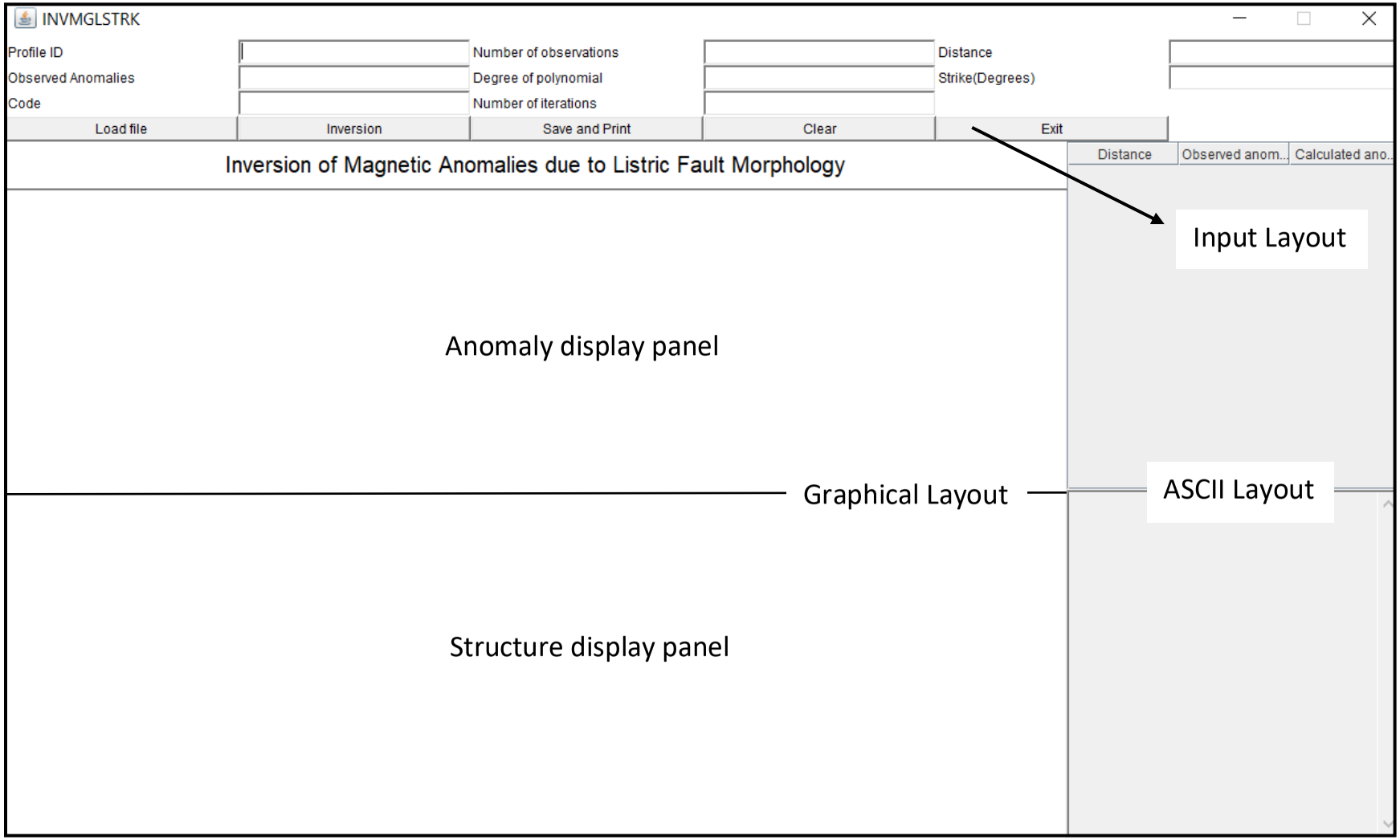


Fig. 4.1: View module of INVMGLSTRK

listric fault sources in any component (Appendix 4A). The work flow of the software is based on the Model-View-Controller setup (Fig. 2.5). The software initializes the structure using a few characteristic anomalies and their positions on the profile, and refines the structure iteratively until a specified convergence criteria is satisfied.

The view module of INVMGLSTRK is shown in Fig. 4.1. The input layout consists of 8 input fields and 5 action buttons. The input data required for the code are: profile ID, number of observations, distance to each observation, observed anomaly (nT), degree of polynomial, strike of the source with reference to magnetic north (degrees), code number (1 for the vertical component, 2 for horizontal, and 3 for the total field), and number of iterations to be performed. As in the case of FORMAG, and LISMAG2D the user can read the input data from an external file using the 'load file' action button. Upon activating the action button - 'Inversion', the software performs the optimisation and recovers the structure. The user saves the output in html format and opts for printing by clicking the 'save and print' action button. The notable feature of the software is that during the process of analysis it displays the changes in data misfit, model space and nature of fit between the observed and theoretical anomalies against iteration in animated form.

4.4 Applications

To justify the relevance of the proposed method of inversion, two magnetic anomaly profiles were interpreted. In the first case, the vertical magnetic anomalies due to a synthetic structure are analyzed. In the latter case, the total magnetic anomalies attributable to a deep-seated fault from the western margin of the Perth Basin, Australia are interpreted.

4.4.1 Synthetic example

Fig. 4.2b describes the model geometry of an assumed listric fault structure in the xz-plane. The fault originates near the topography at the 20^{th} km and extends downwards with decreasing

Table 4.1

Assumed and estimated coefficients of different polynomials (synthetic example) and estimated coefficients (real field example) (Ani Nibisha et al., 2022)

Synthetic example												
Degree of polynomial	**Assumed coefficients**						**Estimated coefficients after inversion**					
	f_0	f_1	f_2	f_3	f_4	f_5	f_0	f_1	f_2	f_3	f_4	f_5
5	20.014	-0.1479	0.4836	0.0711	-0.0023	0.0004	20.014	-0.1479	0.4836	0.0711	-0.0023	0.0004
3							19.9248	0.2435	-0.0125	0.1666		
2							20.3011	-1.025	0.8824			
Field example (western margin of the Perth Basin, Australia)												
2							10.5274	1.5683	-0.0265			

dips to a maximum depth of 4 km (Ani Nibisha et al., 2022). A 5^{th} degree polynomial, whose coefficients are given in Table 4.1, describes the geometry of the fault ramp. Assuming the intensity and direction of magnetisation as 100 gammas and 30nT respectively, the structure anomalies in the vertical component are calculated in the observation range, 0 -40 km, on the profile to which pseudorandom noise (Gaussian) is added and shown in Fig. 4.2a (Ani Nibisha et al., 2022). The noise has zero mean and a standard deviation of 8.82nT. We treat these noisy anomalies as the observed anomalies in the inversion to recover the structure by setting 4.1nT as the acceptable threshold for the data misfit. The algorithm executed 8 iterations for inversion and then terminated. It was found from the inversion result that the misfit had decayed rapidly within first six iterations, from 40.6 to 8.32 gammas, thence gradually before it attained a value of 4.1nT at the end of the 8^{th} iteration. The estimated values of intensity of magnetisation and direction of magnetisation from the inversion were 99.8 gammas and 31nT, respectively. The recalculated theoretical anomalies of the estimated model after the 8^{th} iteration closely fit the observed anomalies and the estimated structure exactly mimics the assumed one (this case is not shown for brevity). The determined polynomial coefficients after inversion precisely coincide with the assumed ones (Table 4.1).

In reality, it is always tricky to select the best polynomial to describe the fault plane geometry, if additional information on the subsurface is either scanty or unavailable. Therefore, it is necessary to examine whether or not the inversion technique could recover the structure, if a polynomial degree other than the optimum is used in the inversion. For this, the inversion was repeated independently using a 2^{nd} degree and a 3^{rd} degree polynomial (in place of a 5^{th} degree) to simulate the fault surfaces (Ani Nibisha et al., 2022). Here also the tolerable misfit between the observed and modelled anomalies was kept 4.1 gammas. For the 2^{nd} degree polynomial approximation of the fault plane, the code had performed 10 iterations and for 3^{rd} degree 6 iterations, respectively. In both cases, the algorithm initially identified the origin of the fault

plane at 19.99 km, and placed the top edge of the fault plane at 0.29 km depth and the bottom at 2.39 km, respectively. The estimated intensity of magnetisation in either case was 99.4 gammas, and the direction of magnetisation was 35°. The initial model parameters were updated in iterative mode along with the other unknowns (coefficients of the function, $f(z)$). After the inversion, the misfit calculated through eq (4.6) in either case was less than the predefined threshold. The modelled anomalies after the inversion in both cases were shown in Fig. 4.2a along with the observed ones. Fig. 4.2b shows the corresponding estimated structures. The maximum difference between the observed and theoretical anomalies corresponding to the optimum structures were found within ± 8nT. The estimated initial and final values of the polynomial coefficients were given in Table 4.1. Though the initial misfit for the starting model was 40.61 gammas in either case, it's decay with iteration was more rapid with the 3rd degree polynomial approximation of the fault plane when compared to the 2nd degree (Fig. 4.3).

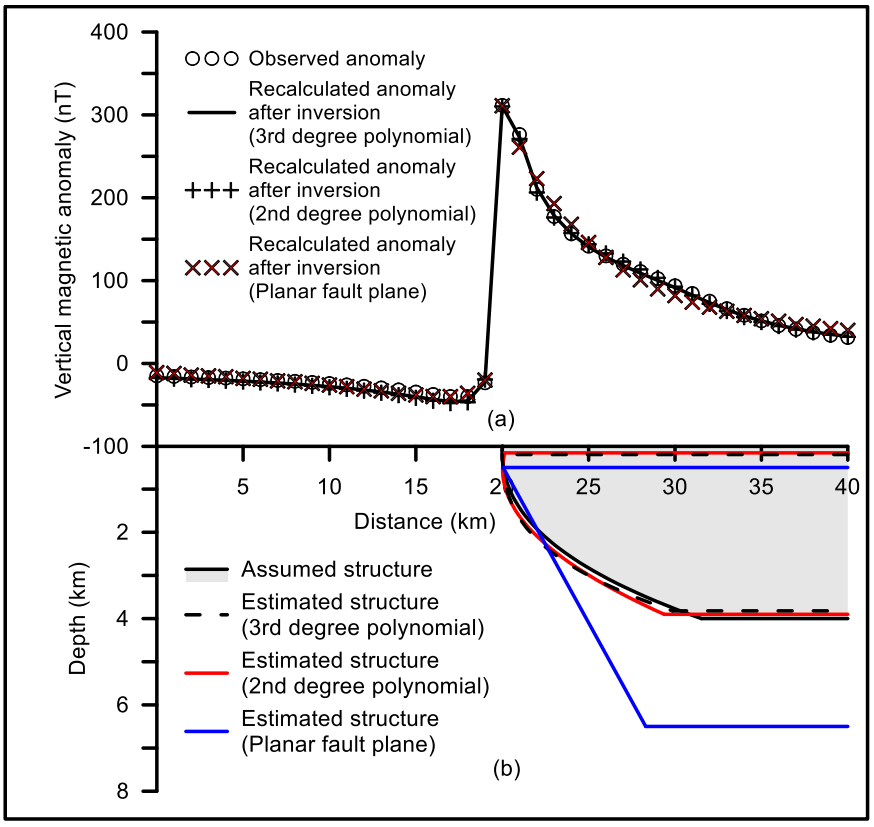


Fig. 4.2 (a) Observed and recalculated anomalies after inversion, (b) assumed and estimated structures after inversion (Ani Nibisha et al., 2022).

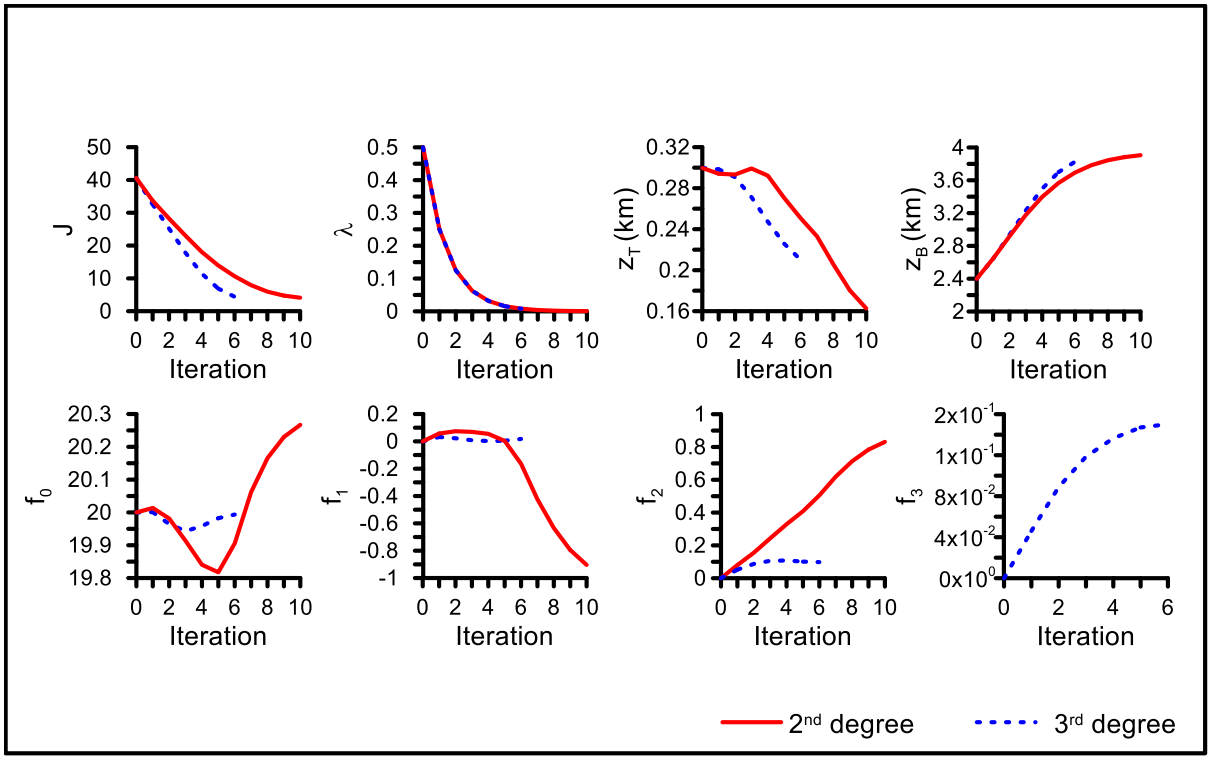


Fig. 4.3 Changes in misfit, damping factor and other parameters of model space with iteration number (Ani Nibisha et al., 2022).

It was found that when the 2nd degree polynomial was used to model the fault plane, the inversion had placed the top of the fault at 0.16 km, and the bottom at 3.91 km (Fig. 4.2b). In case of the 3rd degree, the top edge was placed at 0.19 km, and the bottom at 3.93 km, respectively. Therefore, the bottom depth of the recovered structure remains almost the same regardless the degree of polynomial used in the inversion. On the other hand, the depth to the top of the fault edge was slightly overestimated in either case when compared to the assumed one. Also, the choice of 2nd degree in the inversion had led to the underestimation of the extension by about 6.5%. The changes in the data misfit, damping factor, and other unknown parameters of the model space against the iteration are shown in Fig. 4.3.

To assess the effect of planar fault plane assumption on the interpretation of anomalies caused by listric faults, the anomalies in Fig. 4.2a were also analysed using the method of Murthy et al. (2001). The recalculated anomalies of the optimum model, and the resultant structure resulted from the inversion method of Murthy et al. (2001) were shown in Figs. 4.2a and 4.2b.

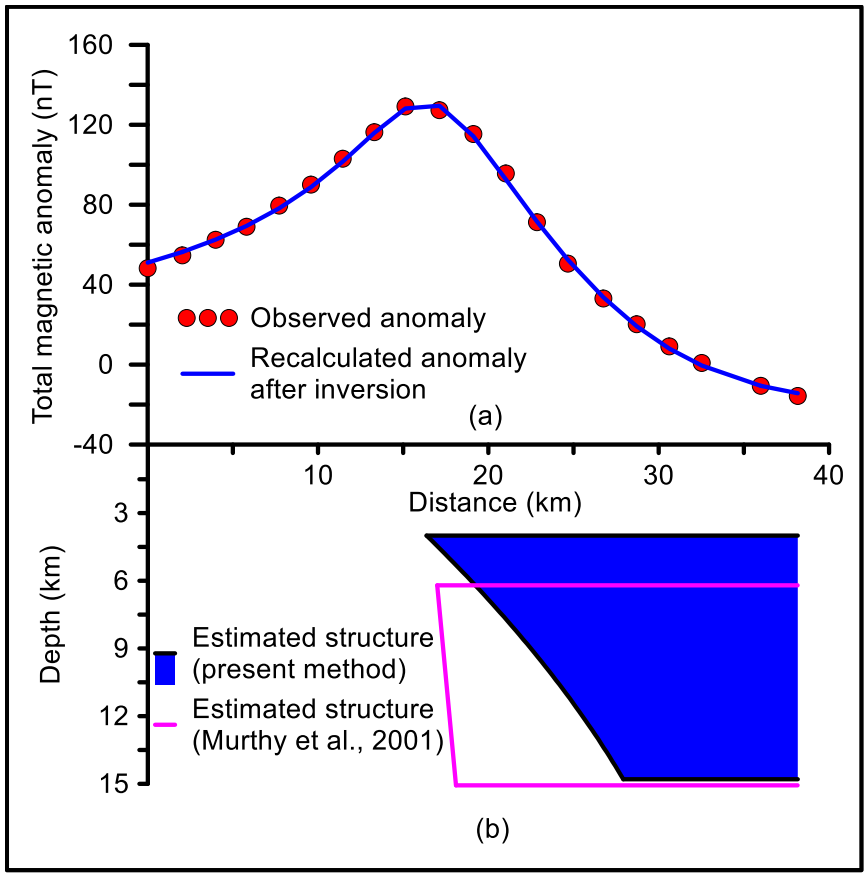


Fig. 4.4 (a) Observed and recalculated total magnetic anomalies and (b) estimated structure after inversion, western margin of the Perth Basin, Australia. The inverted model by Murthy et al. (2001) is also shown (Ani Nibisha et al., 2022).

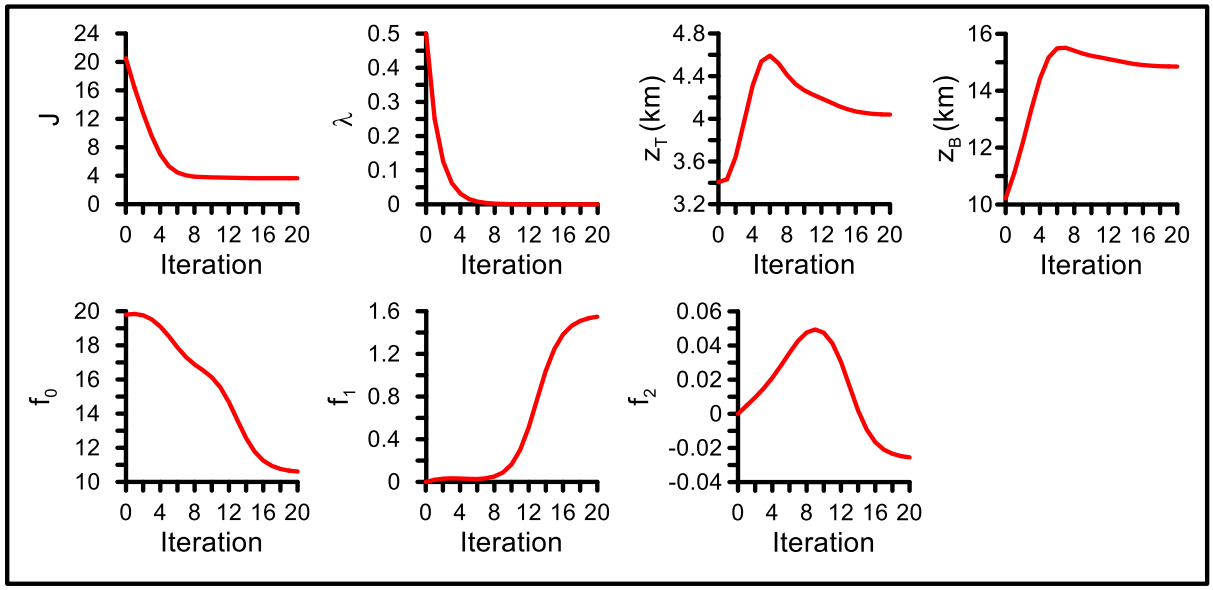


Fig. 4.5 Changes in misfit, damping factor and other parameters of model space with iteration number, western margin of the Perth Basin, Australia (Ani Nibisha et al., 2022).

It is clearly seen from Fig. 4.2b that the deduced structure with its top at 0.5 km, bottom at 6.5 km, with a fault angle of 120° does not comply well with the true structure, hence, the model is refuted (Ani Nibisha et al., 2022).

Sample input and output files of the inversion are given Annexures 4B and 4C.

4.42 Field example – Western margin of the Perth Basin, Australia

The Perth Basin of Australia is well-known for its hydrocarbon prospects. This rift basin is bounded by the Carnarvon Basin to the north, and the Yilgarn Craton to the east. To the southeast it is boarded by the Bight Basin, and on the west and south it shares the oceanic crust of the Indian and Southern oceans (Kovac et al., 2016). Fig. 4.4a shows an aeromagnetic anomaly profile across the western margin of the basin at 30°S (Qureshi and Nalaye 1978). For the present study, the anomaly profile was digitized into 20 observations and subjected to inversion, taking a 2^{nd} degree polynomial to simulate the fault plane. The theoretical anomalies and the deciphered structure after the 20^{th} iteration, being the concluding one, were shown in Fig. 4a and 4b, respectively. Overall, the residuals among the observed and theoretical anomalies vary between -2.7 to 2.9 gammas along the length of the profile. The present technique found the origin of the fault plane on the profile at the 16^{th} km at a depth of 4 km, and placed its bottom at 14.8 km. The intensity and direction of magnetisation are estimated as 68.3 gammas and 315°.

The estimated coefficients of the 2^{nd} degree polynomial are given in Table 4.1. The changes in the data misfit, damping factor and other parameters of the model space as a function of iteration number are shown in Fig. 4.5. The interpreted model of the same anomaly treating the fault plane as a planar surface (Murthy et al., 2001) was also shown in Fig. 4.4b for comparison. The deduced structure from the present inversion is more appropriate because the identified

faults across the basin margins from seismic imaging also revealed listric nature with depth (Middleton et al., 1993).

4.5 Conclusions

1) An automatic inversion technique was presented to quantify 2D listric fault sources from the measured magnetic anomalies in any component. This method uses the Marquardt's algorithm in its implementation. From the measured anomalies the proposed method estimates the parameters of the source, intensity and direction of magnetization. To calculate the model response, this technique makes use of the forward modelling scheme developed by Ani Nibisha et al. (2022).
2) The validity of the proposed technique was exemplified with a theoretical model and a real field case. In the theoretical example, vertical magnetic responses of a 2D listric fault source with predefined parameters were analysed. Though the fault plane of the assumed structure was defined with a 5^{th} degree polynomial in generating the synthetic data, the inversion was carried out independently with 2^{nd} degree and 3^{rd} degree polynomials.
3) It was demonstrated that, by and large, the technique was successful in restoring the assumed structure, though the degrees of polynomials were different from the optimum. On the other hand, interpretation of anomalies with a planar fault surface had yielded a structure that was not in line with the assumed structure. Therefore, it is discouraged to analyse the magnetic anomalies caused by listric fault structures by the techniques, which treat the fault planes as planar surfaces.
4) In the case of real field application, the total magnetic field anomalies attributed to a deep-seated fault from the western margin of the Perth Basin, Australia are interpreted presuming a non-planar surface for the fault plane. The interpreted results were compared with the reported ones.

CHAPTER

FIVE

Conclusions

The thesis has been structured into five chapters to present the research work on the development of new interpretation techniques for magnetic anomalies of 2D listric fault sources. Chapters 2, 3, and 4 specifically cover the detailed findings of the research work.

The key highlights of these chapters presenting new techniques are as follows:

1. A new generalised forward modelling operator and related software are developed to compute the magnetic anomalies of 2D listric fault sources in any component,
2. Interactive modelling of magnetic anomalies and related software for recovering 2D listric fault structures from magnetic anomalies measured in any component are developed, and
3. An automatic inversion technique coupled with relevant software are developed to analyse magnetic anomalies of 2D non-planar fault sources in any component.

The software presented in this thesis are coded in object-oriented, class-based JAVA programming language. The main advantages of the presented software are that they are platform-independent and work on any GUI-based operating system with at least JDK 1.6 version installed. Another notable feature of the software presented in Chapter 3 for interactive modelling is that the model space construction and subsequent modifications to the model space to achieve goodness of fit between the observed and theoretical anomalies shall be realized by simple drag and drop mouse operations. Accordingly, the model space and corresponding theoretical anomaly response shall be updated and displayed in real-time. Yet

another noteworthy feature of the software presented in Chapter 4 for automatic inversion is that the theoretical anomaly fit with the observed one, model space growth, and the decay of data misfit with iteration are displayed in animated forms to witness the solution convergence.

5.1 Scope for future research

Interpretation of magnetic anomalies caused by finite-strike listric fault sources is not yet reported. It is expected that future research will focus on developing suitable and appropriate algorithms to address this aspect.

```
package com.formag.view;



import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;

import javax.swing.JFrame;
import javax.swing.JOptionPane;


import com.frmglstrk.control.FRMGLSTRK_Controller;
import com.frmglstrk.model.FRMGLSTRK_CalculateValues;

public class FORMAG_MainView extends Frame{

	/**
	Forward Modeling of Magnetic Anomalies due to 2D Listric Fault Morphology
	 *
	 */


	public static void main(String s[])
	{
		FORMAG_MainView cm = new FORMAG_MainView();
		cm.setSize(1280, 768);
		cm.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				JFrame frame = null;
				int r = JOptionPane.showConfirmDialog(
						frame,
						"Exit FORMAG ?",
						"Confirm Exit ",
						JOptionPane.YES_NO_OPTION);
				if(r == JOptionPane.YES_OPTION ){
					if(FRMGLSTRK_Controller.success == false){
						String fileName =
FRMGLSTRK_CalculateValues.input_profile_num+".jpg";
						File f = new File(fileName);
						f.delete();
					}
					System.exit(0);
				}
			}
		});
		cm.setTitle("FORMAG");
		cm.setResizable(true);
		cm.add(new FORMAG_MainPanel(cm));
		cm.setVisible(true);

	}

}
```

```
package com.formag.view;
```

```
import java.awt.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.HashMap;
import javax.swing.JFileChooser;
import com.formag.control.FORMAG_Controller;

public class FORMAG_MainPanel extends Panel {

	public static TextArea img = new TextArea(46,140);
	Panel p_North, p_West,p_South;
	public static Panel p_East;
	public static Label graphLabel;
	public static Panel p_Center;
	public static TextField inputValues [] = new TextField[18];

	Button actionButton[] = new Button[12];
	Object rowdata[][]={};

	/**Number of the Profile*/
	public final static int NUM_PROFILE = 0;
	/**Number of observation*/
	public static final int N_OBS = 1 ;
	/**Distances(km)*/
	public static final int DIS_KM = 2;
	/**Minimum depth(km)*/
	public static final int MIN_DEP = 3;
	/**Maximum depth(km)*/
	public static final int MAX_DEP = 4;
	/** Degree of polynomial*/
	public static final int D_POLY = 5;
	/** Fault coefficients*/
	public static final int CO_EFF =6;
	/**Value Of Strike*/
	public static final int VAL_STR = 7;
	/**Value Of IOM*/
	public static final int VAL_IOM = 8;
	/**Value Of DOM*/
	public static final int VAL_DOM = 9;
	/**Value Of Code*/
	public static final int VAL_CODE = 10;



	public FORMAG_MainPanel(FORMAG_MainView cm){

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

		graphLabel = new Label("Forward Modeling of Magnetic Anomalies due to 2D
Listric Fault Morphology", Label.CENTER);
		graphLabel.setFont(new Font("Arial", 40, 20));
		p_Center.add(graphLabel);
```

```
        for(int i = 0; i < 11; i++){
                inputValues[i] = new TextField();
        }

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

        this.populateNorthPanel();
        FORMAG_TableView.populateEastPanel(rowdata);

        this.add(p_North, BorderLayout.NORTH);
        p_Center.setSize(1100, 1000);
        this.add(p_Center, BorderLayout.CENTER);
        p_Center.add(img);
        this.add(p_East, BorderLayout.EAST);



        this.setVisible(true);
}

public void populateNorthPanel(){
        p_North.setLayout(new GridLayout(3,4));

        p_North.add(new Label("Profile ID"));
        p_North.add(inputValues[0]);

        p_North.add(new Label("Number of observations"));
        p_North.add(inputValues[1]);

        p_North.add(new Label("Distance"));
        p_North.add(inputValues[2]);

        p_North.add(new Label("Depth to top"));
        p_North.add(inputValues[3]);

        p_North.add(new Label("Depth to bottom"));
        p_North.add(inputValues[4]);

        p_North.add(new Label("Degree of polynomial"));
        p_North.add(inputValues[5]);

        p_North.add(new Label("Coefficients of polynomial"));
        p_North.add(inputValues[6]);

        p_North.add(new Label("Strike(Degrees)"));
        p_North.add(inputValues[7]);

        p_North.add(new Label("Intensity of Magnetization(nT)"));
        p_North.add(inputValues[8]);

        p_North.add(new Label("Direction of Magnetization(Degrees)"));
        p_North.add(inputValues[9]);

        p_North.add(new Label("Code number"));
        p_North.add(inputValues[10]);
```

```
		p_North.add(new Label(""));
		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]);


		actionButton[0].addActionListener(new FORMAG_Controller());
		actionButton[1].addActionListener(new FORMAG_Controller());
		actionButton[2].addActionListener(new FORMAG_Controller());
		actionButton[3].addActionListener(new FORMAG_Controller());
		actionButton[4].addActionListener(new FORMAG_Controller());

}


public static HashMap captureValues(){

		HashMap h_Map = new HashMap();
		try {

			h_Map.put("NUM_PROFILE", inputValues[NUM_PROFILE].getText());
			h_Map.put("N_OBS", inputValues[N_OBS].getText());
			h_Map.put("D_POLY", inputValues[D_POLY].getText());
			h_Map.put("DIS_KM", inputValues[DIS_KM].getText());
			h_Map.put("CO_EFF", inputValues[CO_EFF].getText());
			h_Map.put("MIN_DEP", inputValues[MIN_DEP].getText());
			h_Map.put("MAX_DEP", inputValues[MAX_DEP].getText());
			h_Map.put("VAL_STR", inputValues[VAL_STR].getText());
			h_Map.put("VAL_IOM", inputValues[VAL_IOM].getText());
			h_Map.put("VAL_DOM", inputValues[VAL_DOM].getText());
			h_Map.put("VAL_CODE", inputValues[VAL_CODE].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, 1000, 600);
}




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){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.NUM_PROFILE].setText(""+st);

			}
			if (count==2){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.N_OBS].setText(""+st);

			}

			if (count==3){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.DIS_KM].setText(""+st);

			}
			if (count==4){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.MIN_DEP].setText(""+st);

			}
			if (count==5){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.MAX_DEP].setText(""+st);

			}
			if (count==6){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.D_POLY].setText(""+st);

			}

			if (count==7){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.CO_EFF].setText(""+st);

			}
```

```
                                        if (count==8){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.VAL_STR].setText(""+st);

                                        }
                                        if (count==9){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.VAL_IOM].setText(""+st);

                                        }


                                        if (count==10){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.VAL_DOM].setText(""+st);

                                        }

                                        if (count==11){

FORMAG_MainPanel.inputValues[FORMAG_MainPanel.VAL_CODE].setText(""+st);

                                        }

                                    }
                                    catch(Exception e) {

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

                                }
                                br.close();
                            }

                        }
                        catch(Exception e){

                        }
                }

                public static void clearDefaultValues(){


                        inputValues[NUM_PROFILE].setText("");
                        inputValues[N_OBS].setText("");
                        inputValues[D_POLY].setText("");
                        inputValues[DIS_KM].setText("");
                        inputValues[MAX_DEP].setText("");
                        inputValues[MIN_DEP].setText("");
                        inputValues[VAL_STR].setText("");
                        inputValues[VAL_IOM].setText("");
                        inputValues[VAL_DOM].setText("");
                        inputValues[VAL_CODE].setText("");
```

```
                inputValues[CO_EFF].setText("");

        }
}
```

```
package com.formag.view;



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


public class FORMAG_TableView extends Panel{

        public static TextArea val = new TextArea(5,5);
        public static  void populateEastPanel(Object rowData[][]) {

                com.formag.view.FORMAG_MainPanel.p_East.removeAll();
                com.formag.view.FORMAG_MainPanel.p_East.setLayout(new GridLayout(2,1));

                Object columnNames[] = {"Distance","Calculated anomalies (nT)"};
                JTable table = new JTable(rowData, columnNames);
                table.setPreferredScrollableViewportSize(new Dimension(300,500));
                JScrollPane scrollPane = new JScrollPane(table);
                scrollPane.setAutoscrolls(true);
                com.formag.view.FORMAG_MainPanel.p_East.add(scrollPane);
                val.setEditable(false);
                com.formag.view.FORMAG_MainPanel.p_East.add(val);
                com.formag.view.FORMAG_MainPanel.p_East.validate();
                com.formag.view.FORMAG_MainPanel.p_East.setVisible(true);
        }
}
```

```
package com.formag.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.formag.model.FORMAG_CalculateValues;
import com.formag.util.FORMAG_Utility;

public class FORMAG_DrawGraph extends Applet{

        public static int i_no_obs;
        float maxY,maxZ,maxX ;
        double obs[];
```

```
	public void drawGraph(Graphics2D g2) {

		g2.setFont(new Font("Arial", 20, 12));
		g2.setColor(Color.BLACK);
		g2.draw(new Line2D.Float(200, 20, 200, 300));
		g2.drawLine(90,10 ,950,10);
		g2.drawLine(90, 560, 950, 560);
		g2.drawLine(950, 10, 950, 560);
		g2.drawLine(90, 10, 90, 560);

		String []a = {"A", "N", "O", "M", "A", "L", "Y", "(nT)"};
		String []b = {"D", "E", "P", "T", "H"};
		for (int i = 0; i < a.length; i++) {
			g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
		}
		for (int i = 0; i < b.length; i++) {
			g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
		}

	}

	public void plot(Graphics2D g2) {

		g2.setFont( new Font("Arial", 12, 12) );
		DecimalFormat f = new DecimalFormat("0.#");
		g2.setColor(Color.BLACK);

		i_no_obs = FORMAG_CalculateValues.i_no_obs;
		obs = new double[i_no_obs+1];
		for (int i = 1; i <= i_no_obs; i++) {
			obs[i] = FORMAG_CalculateValues.d_dis_km[i];
		}

		maxX = (float)obs[i_no_obs];
		float maxy = (float)
FORMAG_Utility.findMaximumNumber(FORMAG_CalculateValues.ano);
		float maxy1 = (float)
FORMAG_Utility.findMaximumNumber(FORMAG_CalculateValues.ano);
		if(maxy > maxy1)
			maxY = maxy;
		else
			maxY = maxy1;
		maxZ = (float)FORMAG_CalculateValues.d_max_dep;

		g2.drawString("|",(float) 600-3+50  , 308);
		g2.drawString(""+f.format(FORMAG_CalculateValues.d_dis_km[i_no_obs]),(float)
600-10+50 , 323);
		g2.drawString("0", 125+50, 310);
		g2.drawString("DISTANCE", 315+50, 285);
		float xplot = 0;
		float xInterval = (float) (FORMAG_CalculateValues.d_dis_km[i_no_obs] / 5);

		int zInterval = 50;
		for (float x = xInterval, j = 1; x < 600; x += xInterval){
			xplot = xplot + xInterval;
			if(j > 4)
				break;
			g2.drawString("|",(float) (200 + (450 * x / maxX) ), 308);
			g2.drawString("" + f.format(xplot), (float) (150 + (450 * x / maxX) ) - 3+50,
323);
```

```
                j++;
            }

            DecimalFormat d =  new DecimalFormat("0.#");
            float points1 = maxZ / 5 ;
            for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                g2.drawString("-", 148+50, 52 + x);
                g2.drawString("" +d.format(points1 * j), 125+50, 50 + x);
                j++;
            }
        }

        public void plotXYCoordinates(Graphics2D g2){

            double minAno =
FORMAG_Utility.findMinimumNumber1(FORMAG_CalculateValues.ano);
            double maxAno =
FORMAG_Utility.findMaximumNumber(FORMAG_CalculateValues.ano, minAno);

            if (minAno < 0 &&  maxAno <= 0  ){
                plotXYCoordinates1(g2);
            }
            if (minAno >= 0 && maxAno > 0){
                plotXYCoordinates1(g2);
            }
            if (minAno < 0  && maxAno>0 ){
                plotXYCoordinates2(g2);
            }

        }

        public void plotXYCoordinates1 (Graphics2D g2) {

            g2.setColor(Color.white);
            g2.drawLine(200, 20, 200, 50);
            g2.setFont(new Font("Arial", 20, 12));
            g2.setColor(Color.black);
            float maxval = (float)
FORMAG_Utility.findMaximumNumber(FORMAG_CalculateValues.ano);
            maxY = maxval;

            int points = (int)maxY / 5;
            int yInterval = 50;
            g2.drawString("0", 125+50, 50);
            for (int x = yInterval, j = 1; x < 250; x += yInterval){

                g2.drawString("-", 148+50, 50 + x);
                g2.drawString("" + (points*j), 145, 50 + x);
                j++;
            }
            float prevx = (float) (200 +( 450 * obs[1] / maxX));;
            float prevy = (float)( 50 + ( 250 * FORMAG_CalculateValues.ano[1] / maxY ) );
            float xpoint = 0;
            float ypoint = 0;

            for (int k = 1; k <= i_no_obs; k++) {

                xpoint = (float)( 450 * obs[k] / maxX);
                ypoint = (float)( ( 250 * FORMAG_CalculateValues.ano[k] / maxY ) );
```

```
                g2.setColor(Color.BLACK);
                g2.draw(new Line2D.Float(prevx, prevy, 200+ xpoint, 50 + ypoint ));
                g2.setFont(new Font("Arial", 20, 12));
                g2.setColor(Color.black);
                prevx = 200 + xpoint;
                prevy = 50 + ypoint ;

            }

        }


        public void plotXYCoordinates2 (Graphics2D g2) {

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

            double store[] = new double[FORMAG_CalculateValues.i_no_obs+1];
            double store1[] = new double[FORMAG_CalculateValues.i_no_obs+1];
            double negstore[] = new double[FORMAG_CalculateValues.i_no_obs+1];
            double negstore1[] = new double[FORMAG_CalculateValues.i_no_obs+1];
            for(int i = 1; i <= FORMAG_CalculateValues.i_no_obs; i++){
                if(FORMAG_CalculateValues.ano[i]>0)
                    store[i] = FORMAG_CalculateValues.ano[i];
                else
                    negstore[i] = FORMAG_CalculateValues.ano[i];
                if(FORMAG_CalculateValues.ano[i]>0)
                    store1[i] = FORMAG_CalculateValues.ano[i];
                else
                    negstore1[i] = FORMAG_CalculateValues.ano[i];
            }
            float maxpos = (float) FORMAG_Utility.findMaximumNumber1(store);
            float maxpos1 = (float) FORMAG_Utility.findMaximumNumber1(store1);
            float maxneg = (float) FORMAG_Utility.findMaximumNumber1(negstore);
            float maxneg1 = (float) FORMAG_Utility.findMaximumNumber1(negstore1);
            float posnum =0;

            if(maxpos>maxpos1)
                posnum = Math.abs(maxpos);
            else
                posnum = Math.abs(maxpos1);
            if(Math.abs(maxneg)>Math.abs(maxneg1))
                maxY = maxneg;
            else
                maxY = maxneg1;

            if(Math.abs(maxY)>10){

                float prevx = (float) (200 +( 450 * obs[1] / maxX));
                float prevy = 0;
                if(FORMAG_CalculateValues.ano[1]>0)
                    prevy = 160-(float)( ( 140 * FORMAG_CalculateValues.ano[1] /
posnum ) );
                else
                    prevy = 160+(float)( ( 140 * FORMAG_CalculateValues.ano[1] /
maxY ) );

                float xpoint = 0;
                float ypoint = 0;
```

```
                        g2.drawString("0",125+50,162);
                        g2.drawString("-", 148+50, 164 );

                        DecimalFormat f = new DecimalFormat("0.#");
                        float points = maxY / 7;
                        float points1 = posnum / 7;
                        int yInterval=20;
                        for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                                g2.drawString("-", 148+50, 164 - x );
                                g2.drawString("" + f.format(points1 * j), 145, 162 - x );
                                j++;
                        }
                        for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                                g2.drawString("-", 148+50, 164 + x );
                                g2.drawString("" + f.format(points * j), 145, 162 + x );
                                j++;
                        }
                        for (int k = 1; k <= i_no_obs; k++) {

                                xpoint = (float)( 450 * obs[k] / maxX);
                                if(FORMAG_CalculateValues.ano[k]>0)
                                        ypoint = 160-(float)( ( 140 *
FORMAG_CalculateValues.ano[k] / posnum ) );
                                else
                                        ypoint = 160+(float)( ( 140 * FORMAG_CalculateValues.ano[k]
/ maxY ) );


                                g2.setColor(Color.BLACK);
                                g2.draw(new Line2D.Float(prevx, prevy, 200 + xpoint,  ypoint  ));

                                g2.setFont(new Font("Arial", 20, 12));
                                g2.setColor(Color.black);
                                prevx = 200 + xpoint;
                                prevy = ypoint ;

                        }
                }

                else{
                        maxY = -100;
                        float prevx = (float) (200 +( 450 * obs[1] / maxX));
                        float prevy = 0;
                        if(FORMAG_CalculateValues.ano[1]>0)
                                prevy = 260-(float)( ( 240 * FORMAG_CalculateValues.ano[1] /
posnum ) );
                        else
                                prevy = 260+(float)( ( 40 * FORMAG_CalculateValues.ano[1] /
maxY ) );

                        float xpoint = 0;
                        float ypoint = 0;
                        g2.drawString("0",125+50,262);
                        g2.drawString("-", 148+50, 264 );
                        g2.drawString("-100",145,300);
                        DecimalFormat f = new DecimalFormat("0.#");
```

```
                        float points1 = posnum / 8;
                        int yInterval=30;
                        for (int x = yInterval, j = 1; x <= 240; x+=yInterval){

                                g2.drawString("-", 148+50, 264 - x );
                                g2.drawString("" + f.format(points1 * j), 145, 262 - x );
                                j++;
                        }

                        for (int k = 1; k <= i_no_obs; k++) {

                                xpoint = (float)( 450 * obs[k] / maxX);
                                if(FORMAG_CalculateValues.ano[k]>0)
                                        ypoint = 260-(float)( ( 240 *
FORMAG_CalculateValues.ano[k] / posnum ) );
                                else
                                        ypoint = 260+(float)( ( 40 * FORMAG_CalculateValues.ano[k] /
maxY ) );


                                g2.setColor(Color.BLACK);
                                g2.draw(new Line2D.Float(prevx, prevy, 200 + xpoint,  ypoint  ));
                                g2.setFont(new Font("Arial", 20, 12));
                                g2.setColor(Color.black);
                                prevx = 200 + xpoint;
                                prevy = ypoint ;

                        }
                }
        }

        public void plotZCoordinates (Graphics2D g2) {

                DecimalFormat d =  new DecimalFormat("0.#");
                g2.drawString("|",(float) 600-3+50  , 308);
                g2.drawString(""+d.format(FORMAG_CalculateValues.d_dis_km[i_no_obs]),(float)
600-10+50 , 323);
                g2.drawString("0", 125+50, 310);
                g2.drawString("DISTANCE", 315+50, 285);
                int zInterval = 50;
                float xplot = 0;
                float xInterval = (float) (FORMAG_CalculateValues.d_dis_km[i_no_obs] / 5);

                float xend =0;
                maxZ = (float)FORMAG_CalculateValues.d_max_dep;
                float points1 = maxZ / 5 ;
                for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                        g2.drawString("-", 148+50, 52 + x);
                        g2.drawString("" +d.format(points1 * j), 125+50, 50 + x);
                        j++;
                }
                g2.setColor(Color.YELLOW);
                g2.fill(new Rectangle2D.Float(200, 300+(float)( 250 *
FORMAG_CalculateValues.d_min_dep / maxZ), 450 , 250-(float)( 250 *
FORMAG_CalculateValues.d_min_dep / maxZ)));

                g2.setColor(Color.BLACK);
                i_no_obs = FORMAG_CalculateValues.i_no_obs;
                obs = new double[i_no_obs+1];
                for (int i = 1; i <= i_no_obs; i++) {
```

```
                    obs[i] = FORMAG_CalculateValues.d_dis_km[i];
                }
                maxX = (float) obs[i_no_obs];
                maxZ = (float)FORMAG_CalculateValues.d_max_dep;
                float s = 0;
                float fc = 0;
                float spoint = 300;
                float fcpoint = (float)( 200 + ( 450 * FORMAG_CalculateValues.d_cftnt_arr[1] /
maxX ) );
                float xpoint = 0;
                float zpoint = 0;
                while (s <= FORMAG_CalculateValues.d_max_dep) {
                    float z1 = (float) 0.001;
                    s = s + z1;
                    fc = 0;
                    for (int i = 1; i< FORMAG_CalculateValues.d_cftnt_arr.length; i++){
                        fc = (float) (fc + FORMAG_CalculateValues.d_cftnt_arr[i] *
Math.pow(s, i - 1));
                    }
                    xpoint = (float)( 450 * fc / maxX);
                    zpoint = (float)( 250 * s / maxZ);
                    g2.setColor(Color.BLACK);
                    if(200+xpoint >600+50)
                        xend = 600+50;
                    else
                        xend = 200+xpoint;
                    g2.draw(new Line2D.Float(fcpoint, spoint,xend, 300 + zpoint));
                    g2.setColor(Color.RED);
                    g2.draw(new Line2D.Float(200, 300 + zpoint, xend, 300 + zpoint));
                    fcpoint = (float)200 + xpoint;
                    spoint = 300 + zpoint;
                }
                g2.setColor(Color.red);
                g2.fill(new Rectangle2D.Float(200, 300, 450 , (float)( 250 *
FORMAG_CalculateValues.d_min_dep / maxZ)));

                g2.setColor(Color.WHITE);
                g2.fill(new Rectangle2D.Float(601+50, 300, 250 , 255));

                g2.setColor(Color.BLACK);
                g2.drawLine(200, 300, 650, 300);
                g2.drawLine(200, 300, 600, 300);
                g2.drawString("|",(float) 600-3 +50 , 308);
                g2.drawString(""+d.format(FORMAG_CalculateValues.d_dis_km[i_no_obs]),(float)
600-10+50 , 323);

                g2.draw(new Line2D.Float(200, 50, 200, 300));
                g2.drawLine(90,10 ,950,10);
                g2.drawLine(90, 560, 950, 560);
                g2.drawLine(950, 10, 950, 560);
                g2.drawLine(90, 10, 90, 560);

                for (float x = xInterval, j = 1; x < 600; x += xInterval){
                    xplot = xplot + xInterval;
                    if(j > 4)
                        break;
                    g2.drawString("|",(float) (200 + (450 * x / maxX) ), 308);
                    g2.drawString("" + d.format(xplot), (float) (200 + (450 * x / maxX) ) - 3, 323);
                    j++;
                }
```

```
                g2.setFont(new Font("Arial", 40,20));
                if(FORMAG_CalculateValues.val_code==1)
                        g2.drawString("Vertical magnetic anomaly",660, 100);
                if(FORMAG_CalculateValues.val_code==2)
                        g2.drawString("Horizontal magnetic anomaly",660, 100);
                if(FORMAG_CalculateValues.val_code==3)
                        g2.drawString("Total magnetic anomaly",660, 100);
        }
}
```

```
package com.formag.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.formag.util.FORMAG_HandleException;
import com.formag.util.FORMAG_Utility;
import com.formag.view.FORMAG_TableView;
import com.formag.view.FORMAG_MainPanel;

public class FORMAG_CalculateValues {

        public static Object obj[][] = null;

        public static int i_no_obs, i_d_poly,len, val_code = 0;
        public static double  d_max_dep,d_min_dep, val_iom, val_dom,  val_str = 0;
        public static double  d_dis_km[], ano[], d_cftnt_arr[], x[], cftnt[] = null;

        public static String  input_profile_num = "";
        public static double d_x_km_arr[], d_z_km_arr[];
        public static BufferedImage image;

        public void getAnamolyValues(HashMap h_Map) {

                try {
                        i_no_obs = FORMAG_Utility.convertInteger((String)h_Map.get("N_OBS"));
                        i_d_poly = FORMAG_Utility.convertInteger((String)h_Map.get("D_POLY"));
                        d_dis_km =
FORMAG_Utility.convertDoubleArray((String)h_Map.get("DIS_KM"));
                        d_min_dep =
FORMAG_Utility.convertDouble((String)h_Map.get("MIN_DEP"));
                        d_max_dep =
FORMAG_Utility.convertDouble((String)h_Map.get("MAX_DEP"));
                        val_str = FORMAG_Utility.convertDouble((String)h_Map.get("VAL_STR"));
                        input_profile_num =
FORMAG_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
                        val_iom = FORMAG_Utility.convertDouble((String)h_Map.get("VAL_IOM"));
                        val_dom = FORMAG_Utility.convertDouble((String)h_Map.get("VAL_DOM"));
```

```
                        val_code =
FORMAG_Utility.convertInteger((String)h_Map.get("VAL_CODE"));
                        d_cftnt_arr =
FORMAG_Utility.convertDoubleArray((String)h_Map.get("CO_EFF"));
                }
                catch(Exception e) {

                        e.printStackTrace();
                }
        }

        public void cal() throws FORMAG_HandleException{

                double z[] = new double[1000];
                double gs[] = new double[1000];
                ano = new double[i_no_obs + 1];
                x = new double[i_no_obs + 1];
                double dm = 0;
                double gc = 0;
                if(val_code==1)
                        dm = 90;
                if(val_code==2)
                        dm = 0;
                if(val_code==3)
                        dm = val_dom;
                for (int i = 1; i <= i_no_obs; i++){
                        x[i] = d_dis_km[i];
                }

                cftnt = d_cftnt_arr;
                double zt = d_min_dep;
                double rad = 3.14159265 / 180;
                double dmr = rad * dm;
                double aphikt = val_dom * rad;
                double strr = val_str * rad;
                double cont = 2 * val_iom * Math.sqrt(1 - (Math.pow(Math.cos(strr),2) *
Math.pow(Math.cos(dmr),2)));
                double ter = aphikt - Math.atan(Math.sin(strr) / Math.tan(dmr));
                double dx =(x[2] - x[1]) / 10;
                double zdif = d_max_dep - zt;
                int nd = (int)(zdif / dx) + 1;
                double na1 = nd/2;

                if(nd - 2 * na1 != 0)
                        nd = nd + 1;
                double dz = zdif / nd;
                int n2 = nd + 1;

                for(int jz =1; jz <= n2; jz++){
                        z[jz] = zt + dz * (jz - 1);
                }


                for(int k = 1; k <= i_no_obs; k++)
                {
                        double xx = x[k];
                        for(int jz = 1; jz <= n2; jz++)
                        {

                                double sum = 0.0;
```

```
                        for (int jj = 1; jj <= i_d_poly + 1; jj++){

                                sum = sum + cftnt[jj] * Math.pow(z[jz], jj - 1);

                        }

                        double c = Math.cos(ter);
                        double s = Math.sin(ter);
                        double tnum1 = (z[jz]) * c;
                        double tnum2 = (sum-xx) * s;
                        double tden1 = Math.pow((z[jz]), 2);
                        double tden2 = Math.pow((sum-xx), 2);
                        gs[jz] = cont * ((tnum1 - tnum2) / (tden1 + tden2));
                }

                gc = SIMP(gs,z,n2);
                ano[k] = gc;

        }
        setGraphValues(x, ano, cftnt);
}


public double SIMP(double []gs,double []z,int n) {
        double ggc=0;
        double dz = z[2] - z[1];
        double sum1 = 0;
        double sum2 = 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 void  setGraphValues(double []x, double []anomaly,double []coeff) {

        obj = new Object[i_no_obs + 1][3];

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

                obj[K][0]= "" + d.format(x[K]);

                obj[K][1]= "" + df.format(anomaly[K]);
```

```
		}



		FORMAG_TableView.val.setText("");

		FORMAG_TableView.val.appendText("\n");
		FORMAG_TableView.val.append("Coefficients of the polynomial:-\n");
		FORMAG_TableView.val.appendText("----------------------------------------\n");
		FORMAG_TableView.val.appendText("\n");

		for (int i = 1; i < coeff.length; i++){
			FORMAG_TableView.val.appendText("f"+(i - 1)+" = " +df1.format(coeff[i])+
"\n");
		}


	}


	public static void drawGraph(){
		final com.formag.view.FORMAG_DrawGraph dg = new
com.formag.view.FORMAG_DrawGraph();
		try
		{
			int width = 850;
			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);
			dg.plotXYCoordinates(g2);
			dg.plotZCoordinates(g2);
			dg.plot(g2);


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

			String path =  FORMAG_CalculateValues.input_profile_num  +".jpg";
			image = ImageIO.read(new File(path));

			Graphics g_image = FORMAG_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 = FORMAG_MainPanel.img.getGraphics();
					g_image.drawImage(image, -60,-30,image.getWidth(),
image.getHeight(),dg);
				}
			};
			FORMAG_MainPanel.img.addMouseListener(ml3);
		}
```

```
            catch (Exception e2) {
                  //       e2.printStackTrace();
            }
      }
}
```

```
package com.formag.control;



import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.awt.*;
import java.io.*;
import java.text.DecimalFormat;
import javax.imageio.ImageIO;
import javax.swing.*;

import com.formag.model.FORMAG_CalculateValues;
import com.formag.view.FORMAG_MainPanel;


public class FORMAG_Controller implements ActionListener {

      Object rowdata [][] = {};
      BufferedImage image;
      com.formag.view.FORMAG_DrawGraph dg = new
com.formag.view.FORMAG_DrawGraph();
      com.formag.model.FORMAG_CalculateValues cv = new
com.formag.model.FORMAG_CalculateValues();
      public static boolean success = false;

      public void actionPerformed(ActionEvent ae)  {

            if(ae.getActionCommand().equals("Load file")){

                  FORMAG_MainPanel.loadData();

FORMAG_MainPanel.p_Center.add(FORMAG_MainPanel.graphLabel);
                  FORMAG_MainPanel.p_Center.add(FORMAG_MainPanel.img);

                  try{

cv.getAnamolyValues(com.formag.view.FORMAG_MainPanel.captureValues());

                  }
                  catch(Exception e) {
```

```
                    e.printStackTrace();
               }

          }

          else if(ae.getActionCommand().equals("Save and Print")){

               try{
                    String current = System.getProperty("user.dir");
                    File img_file = new
File(FORMAG_CalculateValues.input_profile_num+".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()));
                              myWriter = new
FileWriter(OutFile+".html");
                              myWriter.write("<html> <Body onLoad =
\"window.print()\"><table> <tr> <td>" +
                                             "<table border = 1> <tr> <th
colspan = 4>Profile ID:- "+FORMAG_CalculateValues.input_profile_num+"</
th> </tr>");

                              DecimalFormat df =new
DecimalFormat("0.###");
                              DecimalFormat d = new
DecimalFormat("0.##");

                              myWriter.write("<tr > <th>Distance </
th><th> Calculated anamolies (nT) </th> </tr>");

                              for ( int K = 1; K <=
FORMAG_CalculateValues.i_no_obs; K++){
```

```
                                    myWriter.write("<tr> <td>" +
d.format(FORMAG_CalculateValues.x[K])+"</td>
<td>"+df.format(FORMAG_CalculateValues.ano[K])+"</td></tr>");
                                }
                                myWriter.write(" </table> </td> <td>
<img src = '"+ FORMAG_CalculateValues.input_profile_num +".jpg'></td></
tr></table>");


                                DecimalFormat d1 =new
DecimalFormat("0.####");


                                myWriter.write("<BR>");
                                myWriter.write("Coefficients of the
polynomial::"+"<BR>");

myWriter.write("-----------------------------------------<BR>");
                                for ( int i = 1; i <
FORMAG_CalculateValues.cftnt .length; i++) {

                                    myWriter.write("f"+ ( i - 1 ) +" =
"+d1.format(FORMAG_CalculateValues.cftnt[i])+"<BR>");
                                }

                                myWriter.close();
                           }
                      }
                      else
                      {
                           //pops up error message

                      }

                }
                catch(Exception e1) {

                      e1.printStackTrace();
                }
           }



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

                FORMAG_MainPanel.clearDefaultValues();
                FORMAG_MainPanel.p_Center.removeAll();
```

```
com.formag.view.FORMAG_TableView.populateEastPanel(rowdata);
                com.formag.view.FORMAG_TableView.val.setText("");

FORMAG_MainPanel.p_Center.add(FORMAG_MainPanel.graphLabel);
           }else if(ae.getActionCommand().equals("Exit")){

                JFrame frame = null;
                int r =    JOptionPane.showConfirmDialog(
                           frame,
                           "Exit FORMAG ?",
                           "Confirm Exit ",
                           JOptionPane.YES_NO_OPTION);
                if(r == JOptionPane.YES_OPTION ){
                     if(success==false){
                           String fileName =
FORMAG_CalculateValues.input_profile_num+".jpg";
                           File f = new File(fileName);
                           f.delete();
                     }
                     System.exit(0);
                }
           }else if(ae.getActionCommand().equals("Forward Modeling")){


FORMAG_MainPanel.p_Center.add(FORMAG_MainPanel.graphLabel);
                FORMAG_MainPanel.img.setEditable(true);

                try
                {

cv.getAnamolyValues(com.formag.view.FORMAG_MainPanel.captureValues());

                     if(FORMAG_CalculateValues.val_code<=0 ||
FORMAG_CalculateValues.val_code>3){

                           JFrame frame = null;
                           JOptionPane.showMessageDialog(frame,
                                      "Code value must be 1\n"
                                                +"or 2 or 3",
                                                "Out of bounds error",

JOptionPane.ERROR_MESSAGE);

                     }
                     if(FORMAG_CalculateValues.val_code==1 ||
FORMAG_CalculateValues.val_code==2 || FORMAG_CalculateValues.val_code ==
3){
```

```
                              cv.cal();
                              int width = 1280;
                              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);
                              dg.plotZCoordinates(g2);
                              dg.plot(g2);

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

                              String path =
FORMAG_CalculateValues.input_profile_num +".jpg";
                              image = ImageIO.read(new File(path));

                              Graphics g_image =
FORMAG_MainPanel.img.getGraphics();
                              g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(), dg);

                              MouseListener ml3 = new MouseAdapter(){
                                    public void mouseClicked(MouseEvent e){
                                          Graphics g_image =
FORMAG_MainPanel.img.getGraphics();
                                          g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(),dg);
                                    }
                              };
                              FORMAG_MainPanel.img.addMouseListener(ml3);

com.formag.view.FORMAG_TableView.populateEastPanel(FORMAG_CalculateValues
.obj);
                        }
                  }
                  catch (Exception e2) {

                        e2.printStackTrace();
                  }
```

```
            }
      }

}
```

```
package com.formag.util;



import javax.swing.JFrame;
import javax.swing.JOptionPane;



public class FORMAG_Utility {

      public static double convertDouble(String str) throws Exception {


            Double temp = null;

            try {
                  temp = new Double(str.trim());

            }
            catch(Exception e){

                  JFrame frame = null;
                  JOptionPane.showMessageDialog(frame,
                              "Enter a numerical value.",
                              "Number format error",
                              JOptionPane.ERROR_MESSAGE);


            }
            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 = null;
            try {
                  temp = new Integer(str.trim());
            }
            catch(Exception e){

                  JFrame frame = null;
                  JOptionPane.showMessageDialog(frame,
                              "Enter a numerical value.",
                              "Number format error",
                              JOptionPane.ERROR_MESSAGE);

            }
            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 double findMinimumNumber( double observe[], double
denVal) {

            double max = denVal;
            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 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 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 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 {
```

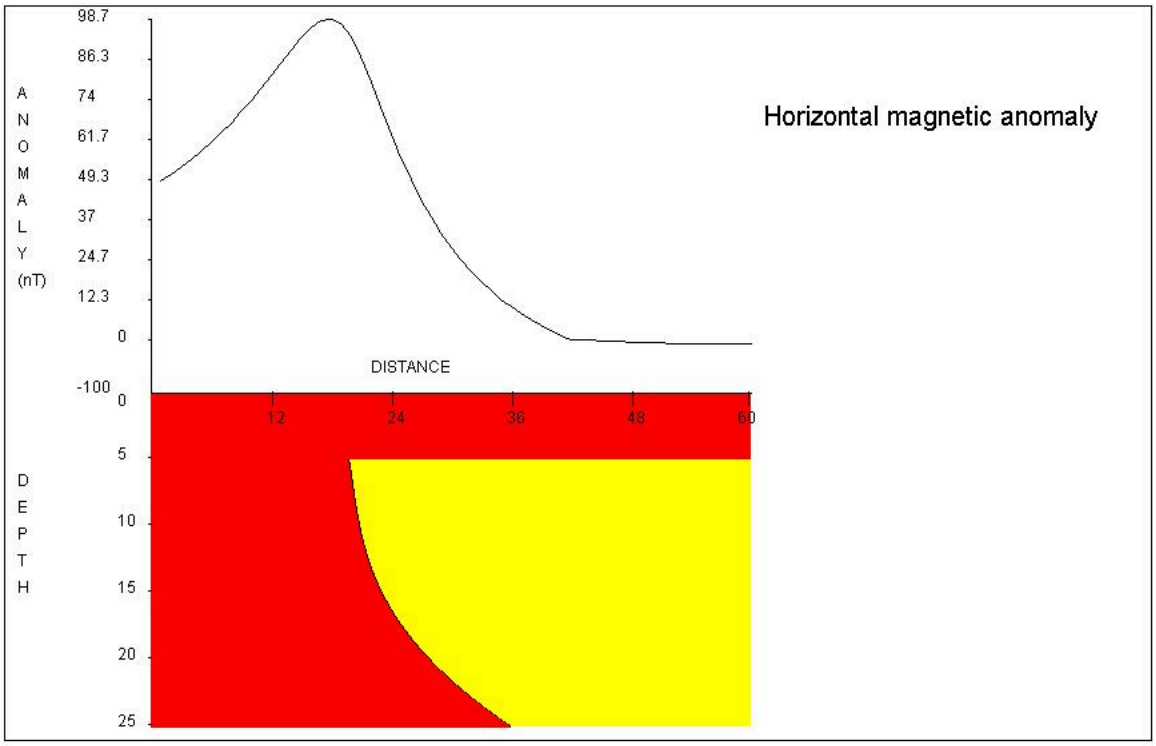
```
            java.util.StringTokenizer st = new
java.util.StringTokenizer(str, ",");
            String temp = "";
            java.util.ArrayList arr = new java.util.ArrayList();

            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 {

                        try {
                              d_array[i] = convertDouble( arr.get(i -
1).toString() );
                        }
                        catch(Exception e){
                              JFrame frame = null;
                              JOptionPane.showMessageDialog(frame,
                                          "Enter numerical values.",
                                          "Number format error",
                                          JOptionPane.ERROR_MESSAGE);
                              throw new FORMAG_HandleException();

                        }
                  }
            }
            return d_array;
      }
}
```

```
package com.formag.util;


import java.awt.*;

import com.formag.view.FORMAG_MainPanel;

public class FORMAG_HandleException extends Exception{
```

```
	public FORMAG_HandleException(){
		Graphics g = FORMAG_MainPanel.p_Center.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);
	}
}
```

```
p-1
60
1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0
,19.0,20.0,21.0,22.0,23.0,24.0,25.0,26.0,27.0,28.0,29.0,30.0,31.0,32.0,33.0,34.0
,35.0,36.0,37.0,38.0,39.0,40.0,41.0,42.0,43.0,44.0,45.0,46.0,47.0,48.0,49.0,50.0
,51.0,52.0,53.0,54.0,55.0,56.0,57.0,58.0,59.0,60.0
5.0
25.0
4
17.97335422,0.6045061731,-0.06495029775,0.003744390665,-0.00003852543696
30.0
70.0
50.0
2
```

Profile ID:- p-1	
Distance	**Calculated anamolies (nT)**
1	48.736
2	50.786
3	52.981
4	55.335
5	57.859
6	60.566
7	63.468
8	66.577
9	69.897
10	73.429
11	77.159
12	81.054
13	85.048
14	89.017
15	92.758
16	95.945
17	98.108
18	98.658
19	97.031
20	92.978
21	86.794
22	79.235
23	71.159
24	63.221
25	55.791
26	49.021
27	42.935
28	37.493
29	32.633
30	28.288
31	24.394
32	20.895
33	17.743
34	14.897
35	12.323
36	9.991
37	7.876
38	5.957
39	4.216
40	2.637
41	1.205
42	-0.092
43	-1.266
44	-2.326
45	-3.283
46	-4.144
47	-4.918
48	-5.613
49	-6.234
50	-6.789
51	-7.283



52	-7.722
53	-8.11
54	-8.452
55	-8.752
56	-9.015
57	-9.243
58	-9.441
59	-9.61
60	-9.755

Coefficients of the polynomial::
---
f0 = 17.9734
f1 = 0.6045
f2 = -0.065
f3 = 0.0037
f4 = -0

```
package com.lismag2d.view;


import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import com.lismag2d.control.LISMAG2D_Controller;
import com.lismag2d.model.LISMAG2D_CalculateValues;

public class LISMAG2D_MainView extends Frame{

	/**
	Interactive Modeling of Magnetic Anomalies due to Listric Fault Morphology
	 *
	 */

	public static void main(String s[])
	{
		LISMAG2D_MainView cm = new LISMAG2D_MainView();
		cm.setSize(1280, 768);
		cm.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				JFrame frame = null;
				int r = JOptionPane.showConfirmDialog(
						frame,
						"Exit LISMAG2D ?",
						"Confirm Exit ",
						JOptionPane.YES_NO_OPTION);
				if(r == JOptionPane.YES_OPTION ){
					if(LISMAG2D_Controller.success == false){
						String fileName =
LISMAG2D_CalculateValues.input_profile_num+".jpg";
						File f = new File(fileName);
						f.delete();
					}
					System.exit(0);
				}
			}
		});
		cm.setTitle("LISMAG2D");
		cm.setResizable(true);
		cm.add(new LISMAG2D_MainPanel(cm));
		cm.setVisible(true);

	}

}
```

```
package com.lismag2d.view;


import java.awt.*;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.HashMap;
import javax.swing.JFileChooser;
import com.lismag2d.control.LISMAG2D_Controller;

import com.lismag2d.util.LISMAG2D_Utility;
import com.lismag2d.view.event.LISMAG2D_PlotFault;
import com.lismag2d.model.LISMAG2D_CalculateValues;

public class LISMAG2D_MainPanel extends Panel {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	public static TextArea img = new TextArea(46,140);
	public static TextArea img1 = new TextArea(46,140);
	public static TextArea img2 = new TextArea(46,140);
	public static TextArea img3 = new TextArea(46,140);
	public static TextArea img4 = new TextArea(46,140);
	public static TextArea fl = new TextArea(46,140);
	public static TextArea den = new TextArea(46,140);
	public static TextArea im = new TextArea(46,140);

	Panel p_North, p_West,p_South;
	public static Panel p_East;
	public static Label graphLabel;
	public static Panel p_Center;
	public static TextField inputValues [] = new TextField[18];

	Button actionButton[] = new Button[12];
	Object rowdata[][]={};

	/**Number of the Profile*/
	public final static int NUM_PROFILE = 0;
	/**Number of observation*/
	public static final int N_OBS = 1 ;
	/**Distances(km)*/
	public static final int DIS_KM = 2;
	/**Observed Anomalies*/
	public static final int OBS_ANO = 3;
	/**Min depth(km)*/
	public static final int MIN_DEP = 4;
	/**Maximum depth(km)*/
	public static final int MAX_DEP = 5;
	/** Degree of polynomial*/
	public static final int D_POLY = 6;
	/**Value Of Strike*/
	public static final int VAL_STR = 7;
	/**Value Of IOM*/
	public static final int VAL_IOM = 8;
	/**Value Of DOM*/
	public static final int VAL_DOM = 9;
	/**Value Of Code*/
	public static final int VAL_CODE = 10;
```

```
	public LISMAG2D_MainPanel(LISMAG2D_MainView cm){

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

		graphLabel = new Label("Interactive Magnetic Anomaly Modelling of 2D Listric
Fault", Label.CENTER);
		graphLabel.setFont(new Font("Arial", 40, 20));
		p_Center.add(graphLabel);

		for(int i = 0; i < 11; i++){
			inputValues[i] = new TextField();
		}

		actionButton[0] = new Button("Load file");
		actionButton[1] = new Button("Specify fault coordinates");
		actionButton[2] = new Button("Draw fault plane");
		actionButton[3] = new Button("Compute/Edit fault plane");
		actionButton[4] = new Button("Graph");
		actionButton[5] = new Button("Save and Print");
		actionButton[6] = new Button("Clear");
		actionButton[7] = new Button("Save file");
		actionButton[8] = new Button("Original fault co-ordinates");
		actionButton[9] = new Button("Exit");

		this.populateNorthPanel();
		LISMAG2D_TableView.populateEastPanel(rowdata);

		this.add(p_North, BorderLayout.NORTH);
		p_Center.setSize(1100, 1000);
		this.add(p_Center, BorderLayout.CENTER);
		p_Center.add(img);
		this.add(p_East, BorderLayout.EAST);
		this.setVisible(true);
	}

	public void populateNorthPanel(){
		p_North.setLayout(new GridLayout(4,5));

		p_North.add(new Label("Profile ID"));
		p_North.add(inputValues[0]);

		p_North.add(new Label("Number of observations"));
		p_North.add(inputValues[1]);

		p_North.add(new Label("Distance"));
		p_North.add(inputValues[2]);

		p_North.add(new Label("Observed anomalies(nT)"));
		p_North.add(inputValues[3]);

		p_North.add(new Label("Depth to top"));
		p_North.add(inputValues[4]);
		p_North.add(new Label("Depth to bottom"));
		p_North.add(inputValues[5]);
```

```
        p_North.add(new Label("Degree of polynomial"));
        p_North.add(inputValues[6]);

        p_North.add(new Label("Strike(Degrees)"));
        p_North.add(inputValues[7]);

        p_North.add(new Label("Intensity of Magnetization(nT)"));
        p_North.add(inputValues[8]);

        p_North.add(new Label("Direction of Magnetization(Degrees)"));
        p_North.add(inputValues[9]);

        p_North.add(new Label("Code number"));
        p_North.add(inputValues[10]);

        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]);
        p_North.add(actionButton[6]);
        p_North.add(actionButton[7]);
        p_North.add(actionButton[8]);
        p_North.add(actionButton[9]);

        actionButton[0].addActionListener(new LISMAG2D_Controller());
        actionButton[1].addActionListener(new LISMAG2D_Controller());
        actionButton[2].addActionListener(new LISMAG2D_Controller());
        actionButton[3].addActionListener(new LISMAG2D_Controller());
        actionButton[4].addActionListener(new LISMAG2D_Controller());
        actionButton[5].addActionListener(new LISMAG2D_Controller());
        actionButton[6].addActionListener(new LISMAG2D_Controller());
        actionButton[7].addActionListener(new LISMAG2D_Controller());
        actionButton[8].addActionListener(new LISMAG2D_Controller());
        actionButton[9].addActionListener(new LISMAG2D_Controller());
}


public static HashMap captureValues(){

        HashMap h_Map = new HashMap();
        try {

                h_Map.put("NUM_PROFILE", inputValues[NUM_PROFILE].getText());
                h_Map.put("N_OBS", inputValues[N_OBS].getText());
                h_Map.put("D_POLY", inputValues[D_POLY].getText());
                h_Map.put("DIS_KM", inputValues[DIS_KM].getText());
                h_Map.put("OBS_ANO", inputValues[OBS_ANO].getText());
                h_Map.put("MIN_DEP", inputValues[MIN_DEP].getText());
                h_Map.put("MAX_DEP", inputValues[MAX_DEP].getText());
                h_Map.put("VAL_STR", inputValues[VAL_STR].getText());
                h_Map.put("VAL_IOM", inputValues[VAL_IOM].getText());
                h_Map.put("VAL_DOM", inputValues[VAL_DOM].getText());
                h_Map.put("VAL_CODE", inputValues[VAL_CODE].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, 1000, 750);
	}

	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){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.NUM_PROFILE].setText(""+st);

						}
						if (count==2){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.N_OBS].setText(""+st);

						}

						if (count==3){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.DIS_KM].setText(""+st);

						}
						if (count==4){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.OBS_ANO].setText(""+st);

						}
						if (count==5){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.MIN_DEP].setText(""+st);
```

```
                                        }
                                        if (count==6){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.MAX_DEP].setText(""+st);

                                        }
                                        if (count==7){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.D_POLY].setText(""+st);

                                        }

                                        if (count==8){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.VAL_STR].setText(""+st);

                                        }
                                        if (count==9){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.VAL_IOM].setText(""+st);

                                        }


                                        if (count==10){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.VAL_DOM].setText(""+st);

                                        }

                                        if (count==11){

LISMAG2D_MainPanel.inputValues[LISMAG2D_MainPanel.VAL_CODE].setText(""+st);

                                        }

                                        if (count==12){
                                                LISMAG2D_PlotFault.val =
LISMAG2D_Utility.convertDoubleArray(st);

                                        }
                                        if (count==13){
                                                LISMAG2D_PlotFault.val1 =
LISMAG2D_Utility.convertDoubleArray(st);

                                        }
                                        if (count==14){
                                                LISMAG2D_CalculateValues.len =
LISMAG2D_Utility.convertInteger(st);

                                        }


                                }
```

```
                                        catch(Exception e) {

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

                                }
                                br.close();
                        }
                }
                catch(Exception e){

                }
        }

        public static void clearDefaultValues(){

                inputValues[NUM_PROFILE].setText("");
                inputValues[N_OBS].setText("");
                inputValues[OBS_ANO].setText("");
                inputValues[D_POLY].setText("");
                inputValues[DIS_KM].setText("");
                inputValues[MIN_DEP].setText("");
                inputValues[MAX_DEP].setText("");
                inputValues[VAL_STR].setText("");
                inputValues[VAL_IOM].setText("");
                inputValues[VAL_DOM].setText("");
                inputValues[VAL_CODE].setText("");

        }
}
```

```
package com.lismag2d.view;

import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Panel;
import java.awt.TextArea;

import javax.swing.JScrollPane;
import javax.swing.JTable;

public class LISMAG2D_TableVal extends Panel{
      public static TextArea val = new TextArea(5,5);
      public static  void populateEastPanel(Object rowData[][]) {

            com.lismag2d.view.LISMAG2D_MainPanel.p_East.removeAll();
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.setLayout(new
GridLayout(2,1));

            Object columnNames[] = {"Distance","Observed anomalies (nT)"};
            JTable table = new JTable(rowData, columnNames);
```

```
            table.setPreferredScrollableViewportSize(new
Dimension(300,500));
            JScrollPane scrollPane = new JScrollPane(table);
            scrollPane.setAutoscrolls(true);
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.add(scrollPane);
            val.setEditable(false);
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.add(val);
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.validate();
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.setVisible(true);

      }
}
```

```
package com.lismag2d.view;



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


public class LISMAG2D_TableView extends Panel{

      public static TextArea val = new TextArea(5,5);
      public static  void populateEastPanel(Object rowData[][]) {

            com.lismag2d.view.LISMAG2D_MainPanel.p_East.removeAll();
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.setLayout(new
GridLayout(2,1));

            Object columnNames[] = {"Distance","Observed anomalies
(nT)","Calculated anomalies (nT)"};
            JTable table = new JTable(rowData, columnNames);
            table.setPreferredScrollableViewportSize(new
Dimension(300,500));
            JScrollPane scrollPane = new JScrollPane(table);
            scrollPane.setAutoscrolls(true);
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.add(scrollPane);
            val.setEditable(false);
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.add(val);
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.validate();
            com.lismag2d.view.LISMAG2D_MainPanel.p_East.setVisible(true);

      }
}
```

```
package com.lismag2d.view;

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.lismag2d.model.LISMAG2D_CalculateValues;
import com.lismag2d.util.LISMAG2D_Utility;

public class LISMAG2D_Plot {
      public static int i_no_obs;
      float maxY,maxZ,maxX ;
      double obs[];

      public void drawGraph(Graphics2D g2) {

            g2.setFont(new Font("Arial", 20, 12));
            g2.setColor(Color.BLACK);
            g2.draw(new Line2D.Float(200, 20, 200, 300));
            g2.drawLine(90,10 ,1040,10);
            g2.drawLine(90, 560, 1040, 560);
            g2.drawLine(1040, 10, 1040, 560);
            g2.drawLine(90, 10, 90, 560);

            String []a = {"A", "N", "O", "M", "A", "L", "Y", "(nT)"};
            String []b = {"D", "E", "P", "T", "H"};
            for (int i = 0; i < a.length; i++) {
                  g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
            }
            for (int i = 0; i < b.length; i++) {
                  g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
            }

      }

      public void plot(Graphics2D g2) {

            g2.setFont( new Font("Arial", 12, 12) );
            DecimalFormat f = new DecimalFormat("0.#");
            g2.setColor(Color.BLACK);

            i_no_obs = LISMAG2D_CalculateValues.i_no_obs;
            obs = new double[i_no_obs+1];
            for (int i = 1; i <= i_no_obs; i++) {
                  obs[i] = LISMAG2D_CalculateValues.d_dis_km[i];
```

```
            }

            maxX = (float)obs[i_no_obs];
            float maxy = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano);
            float maxy1 = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.ano);
            if(maxy > maxy1)
                  maxY = maxy;
            else
                  maxY = maxy1;
            maxZ = (float)LISMAG2D_CalculateValues.d_max_dep;

            g2.drawString("|",(float) 600-3+50  , 308);

g2.drawString(""+f.format(LISMAG2D_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);
            g2.drawString("0", 125+50, 310);
            g2.drawString("DISTANCE", 315+50, 285);
            float xplot = 0;
            float xInterval = (float)
(LISMAG2D_CalculateValues.d_dis_km[i_no_obs] / 5);

            int zInterval = 50;
            for (float x = xInterval, j = 1; x < 600; x += xInterval){
                  xplot = xplot + xInterval;
                  if(j > 4)
                        break;
                  g2.drawString("|",(float) (200 + (450 * x / maxX) ),
308);
                  g2.drawString("" + f.format(xplot), (float) (150 + (450
* x / maxX) ) - 3+50, 323);
                  j++;
            }

            DecimalFormat d =  new DecimalFormat("0.#");
            float points1 = maxZ / 5 ;
            for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                  g2.drawString("-", 148+50, 52 + x);
                  g2.drawString("" +d.format(points1 * j), 125+50, 50 +
x);
                  j++;
            }
      }


      public void plotXYCoordinates(Graphics2D g2){
```

```
          double minAno =
LISMAG2D_Utility.findMinimumNumber1(LISMAG2D_CalculateValues.obs_ano);
          double maxAno =
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano);


          if (  maxAno <= 0 &&   minAno <= 0 ){
               plotXYCoordinates1(g2);
          }
          if (minAno >= 0 && maxAno > 0 ){
               plotXYCoordinates1(g2);
          }
          if (minAno < 0  && maxAno>0  ){
               plotXYCoordinates2(g2);
          }
     }

     public void plotXYCoordinates1 (Graphics2D g2) {

          g2.setColor(Color.white);
          g2.drawLine(200, 20, 200, 50);
          g2.setFont(new Font("Arial", 20, 12));
          g2.setColor(Color.black);
          float maxval1 = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano);

          maxY = maxval1;
          int points = (int)maxY / 5;
          int yInterval = 50;
          g2.drawString("0", 125+50, 50);
          for (int x = yInterval, j = 1; x < 250; x += yInterval){

               g2.drawString("-", 148+50, 50 + x);
               g2.drawString("" + (points*j), 145, 50 + x);
               j++;
          }
          ;
          float xpoint = 0;
          float ypoint1 = 0;


          for (int k = 1; k <= i_no_obs; k++) {

               xpoint = (float)( 450 * obs[k] / maxX);
               ypoint1 = (float)( ( 250 *
LISMAG2D_CalculateValues.ano[k] / maxY ) );
```

```
                g2.setColor(Color.BLUE);
                g2.setFont(new Font("Arial", 20, 30));
                g2.drawString(".", 200 + xpoint-4, 50+ypoint1+1);

                g2.setFont(new Font("Arial", 20, 12));
                //    g2.setColor(Color.black);
           }

     }


     public void plotXYCoordinates2 (Graphics2D g2) {


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

           double store1[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];
           double negstore1[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];

           for(int i = 1; i <= LISMAG2D_CalculateValues.i_no_obs; i++){

                if(LISMAG2D_CalculateValues.obs_ano[i]>0)
                     store1[i] = LISMAG2D_CalculateValues.obs_ano[i];
                else
                     negstore1[i] = LISMAG2D_CalculateValues.obs_ano[i];
           }

           float maxpos1 = (float)
LISMAG2D_Utility.findMaximumNumber1(store1);
           float maxneg1 = (float)
LISMAG2D_Utility.findMaximumNumber1(negstore1);
           float posnum =0;
           posnum = Math.abs(maxpos1);

           maxY = maxneg1;

           if(Math.abs(maxY)>10){

                float xpoint = 0;
                float ypoint1 = 0;

                g2.drawString("0",125+50,162);
                g2.drawString("-", 148+50, 164 );

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

```
                float points = maxY / 7;
                float points1 = posnum / 7;
                int yInterval=20;
                for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                     g2.drawString("-", 148+50, 164 - x );
                     g2.drawString("" + f.format(points1 * j), 145, 162
- x );
                     j++;
                }
                for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                     g2.drawString("-", 148+50, 164 + x );
                     g2.drawString("" + f.format(points * j), 145, 162 +
x );
                     j++;
                }
                for (int k = 1; k <= i_no_obs; k++) {

                     xpoint = (float)( 450 * obs[k] / maxX);

                     if(LISMAG2D_CalculateValues.obs_ano[k]>0)
                          ypoint1 = 160-(float)( ( 140 *
LISMAG2D_CalculateValues.obs_ano[k] / posnum ) );
                     else
                          ypoint1 = 160+(float)( ( 140 *
LISMAG2D_CalculateValues.obs_ano[k] / maxY ) );

                     g2.setColor(Color.BLUE);
                     g2.setFont(new Font("Arial", 20, 30));
                     g2.drawString(".", 200 + xpoint-4, ypoint1+1);

                     g2.setFont(new Font("Arial", 20, 12));
                     //   g2.setColor(Color.black);
                }
           }

           else{
                maxY = -100;

                float xpoint = 0;
                float ypoint1 = 0;
                g2.drawString("0",125+50,262);
                g2.drawString("-", 148+50, 264 );
                g2.drawString("-100",145,300);

                DecimalFormat f = new DecimalFormat("0.#");
                float points1 = posnum / 8;
```

```
                int yInterval=30;
                for (int x = yInterval, j = 1; x <= 240; x+=yInterval){

                      g2.drawString("-", 148+50, 264 - x );
                      g2.drawString("" + f.format(points1 * j), 145, 262
- x );
                      j++;
                }

                for (int k = 1; k <= i_no_obs; k++) {

                      xpoint = (float)( 450 * obs[k] / maxX);

                      if(LISMAG2D_CalculateValues.obs_ano[k]>0)
                            ypoint1 = 260-(float)( ( 240 *
LISMAG2D_CalculateValues.obs_ano[k] / posnum ) );
                      else
                            ypoint1 = 260+(float)( ( 40 *
LISMAG2D_CalculateValues.obs_ano[k] / maxY ) );


                      g2.setColor(Color.BLUE);
                      g2.setFont(new Font("Arial", 20, 30));
                      g2.drawString(".", 200 + xpoint-4, ypoint1+1);

                      g2.setFont(new Font("Arial", 20, 12));
                      //g2.setColor(Color.black);
                }
          }
     }


     public void plotZCoordinates (Graphics2D g2) {
          g2.setFont(new Font("Arial", 20, 12));
          DecimalFormat d =  new DecimalFormat("0.#");
          g2.drawString("|",(float) 600-3+50  , 308);

g2.drawString(""+d.format(LISMAG2D_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);
          g2.drawString("0", 125+50, 310);
          g2.drawString("DISTANCE", 315+50, 285);
          int zInterval = 50;
          float xplot = 0;
          float xInterval = (float)
(LISMAG2D_CalculateValues.d_dis_km[i_no_obs] / 5);

          float xend =0;
          maxZ = (float)LISMAG2D_CalculateValues.d_max_dep;
```

```
            float points1 = maxZ / 5 ;
            for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                  g2.drawString("-", 148+50, 52 + x);
                  g2.drawString("" +d.format(points1 * j), 125+50, 50 +
x);
                  j++;
            }
            g2.setColor(Color.YELLOW);
            g2.fill(new Rectangle2D.Float(200, 300+(float)( 250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ), 450 , 250-(float)( 250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ)));
            g2.setColor(Color.BLACK);
            i_no_obs = LISMAG2D_CalculateValues.i_no_obs;
            obs = new double[i_no_obs+1];
            for (int i = 1; i <= i_no_obs; i++) {
                  obs[i] = LISMAG2D_CalculateValues.d_dis_km[i];
            }
            maxX = (float) obs[i_no_obs];
            maxZ = (float)LISMAG2D_CalculateValues.d_max_dep;
            float s = 0;
            float fc = 0;
            float spoint = 300;
            float fcpoint = (float)( 200 + ( 450 *
LISMAG2D_CalculateValues.d_cftnt_arr[1] / maxX ) );
            float xpoint = 0;
            float zpoint = 0;
            while (s <= LISMAG2D_CalculateValues.d_max_dep) {
                  float z1 = (float) 0.001;
                  s = s + z1;
                  fc = 0;
                  for (int i = 1; i<
LISMAG2D_CalculateValues.d_cftnt_arr.length; i++){
                        fc = (float) (fc +
LISMAG2D_CalculateValues.d_cftnt_arr[i] * Math.pow(s, i - 1));
                  }
                  xpoint = (float)( 450 * fc / maxX);
                  zpoint = (float)( 250 * s / maxZ);
                  g2.setColor(Color.BLACK);
                  if(200+xpoint >600+50)
                        xend = 600+50;
                  else
                        xend = 200+xpoint;
                  g2.draw(new Line2D.Float(fcpoint, spoint,xend, 300 +
zpoint));
                  g2.setColor(Color.RED);
                  g2.draw(new Line2D.Float(200, 300 + zpoint, xend, 300 +
zpoint));
                  fcpoint = (float)200 + xpoint;
```

```
                spoint = 300 + zpoint;
            }
            g2.setColor(Color.red);
            g2.fill(new Rectangle2D.Float(200, 300, 450 , (float)( 250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ)));
            g2.setColor(Color.WHITE);
            g2.fill(new Rectangle2D.Float(601+50, 300, 450 , 255));

            g2.setColor(Color.BLACK);
            g2.drawLine(200, 300, 650, 300);
            g2.drawLine(200, 300, 600, 300);
            g2.drawString("|",(float) 600-3 +50 , 308);

g2.drawString(""+d.format(LISMAG2D_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);

            g2.draw(new Line2D.Float(200, 50, 200, 300));
            g2.drawLine(90,10 ,1040,10);
            g2.drawLine(90, 560, 1040, 560);
            g2.drawLine(1040, 10, 1040, 560);
            g2.drawLine(90, 10, 90, 560);

            for (float x = xInterval, j = 1; x < 600; x += xInterval){
                xplot = xplot + xInterval;
                if(j > 4)
                    break;
                g2.drawString("|",(float) (200 + (450 * x / maxX) ),
308);
                g2.drawString("" + d.format(xplot), (float) (200 + (450
* x / maxX) ) - 3, 323);
                j++;
            }
            g2.setFont(new Font("Arial", 40,20));
            if(LISMAG2D_CalculateValues.val_code==1)
                g2.drawString("Vertical magnetic anomaly",660, 100);
            if(LISMAG2D_CalculateValues.val_code==2)
                g2.drawString("Horizontal magnetic anomaly",660, 100);
            if(LISMAG2D_CalculateValues.val_code==3)
                g2.drawString("Total magnetic anomaly",660, 100);


            g2.setColor(Color.BLUE);
            g2.setFont(new Font("Arial", 20,30));
            g2.drawString("....",750, 280);
            g2.setFont(new Font("Arial", 20,15));
            g2.drawString(":-Observed magnetic anomaly",785, 280);
            //    g2.setColor(Color.BLACK);
```

```
	}

}
```

```
package com.lismag2d.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.lismag2d.model.LISMAG2D_CalculateValues;
import com.lismag2d.util.LISMAG2D_Utility;


public class LISMAG2D_DrawGraph extends Applet{

	public static int i_no_obs;
	float maxY,maxZ,maxX ;
	double obs[];

	public void drawGraph(Graphics2D g2) {

		g2.setFont(new Font("Arial", 20, 12));
		g2.setColor(Color.BLACK);
		g2.draw(new Line2D.Float(200, 20, 200, 300));
		g2.drawLine(90,10 ,950,10);
		g2.drawLine(90, 560, 950, 560);
		g2.drawLine(950, 10, 950, 560);
		g2.drawLine(90, 10, 90, 560);

		String []a = {"A", "N", "O", "M", "A", "L", "Y", "(nT)"};
		String []b = {"D", "E", "P", "T", "H"};
		for (int i = 0; i < a.length; i++) {
			g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
		}
		for (int i = 0; i < b.length; i++) {
			g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
		}

	}

	public void plot(Graphics2D g2) {
```

```
            g2.setFont( new Font("Arial", 12, 12) );
            DecimalFormat f = new DecimalFormat("0.#");
            g2.setColor(Color.BLACK);

            i_no_obs = LISMAG2D_CalculateValues.i_no_obs;

            obs = new double[i_no_obs+1];
            for (int i = 1; i <= i_no_obs; i++) {
                  obs[i] = LISMAG2D_CalculateValues.d_dis_km[i];
            }

            maxX = (float)obs[i_no_obs];
            float maxy = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano);
            float maxy1 = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.ano);
            if(maxy > maxy1)
                  maxY = maxy;
            else
                  maxY = maxy1;
            maxZ = (float)LISMAG2D_CalculateValues.d_max_dep;

            g2.drawString("|",(float) 600-3+50  , 308);

g2.drawString(""+f.format(LISMAG2D_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);
            g2.drawString("0", 125+50, 310);
            g2.drawString("DISTANCE", 315+50, 285);
            float xplot = 0;
            float xInterval = (float)
(LISMAG2D_CalculateValues.d_dis_km[i_no_obs] / 5);

            int zInterval = 50;
            for (float x = xInterval, j = 1; x < 600; x += xInterval){
                  xplot = xplot + xInterval;
                  if(j > 4)
                        break;
                  g2.drawString("|",(float) (200 + (450 * x / maxX) ),
308);
                  g2.drawString("" + f.format(xplot), (float) (150 + (450
* x / maxX) ) - 3+50, 323);
                  j++;
            }

            DecimalFormat d =  new DecimalFormat("0.#");
            float points1 = maxZ / 5 ;
            for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                  g2.drawString("-", 148+50, 52 + x);
```

```
                g2.drawString("" +d.format(points1 * j), 125+50, 50 +
x);
                j++;
           }
     }



     public void plotXYCoordinates(Graphics2D g2){

           double minAno =
LISMAG2D_Utility.findMinimumNumber1(LISMAG2D_CalculateValues.ano);
           double maxAno =
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.ano, minAno);

           double minAno1 =
LISMAG2D_Utility.findMinimumNumber1(LISMAG2D_CalculateValues.obs_ano);
           double maxAno1 =
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano,
minAno1);

           if (minAno < 0 &&  maxAno <= 0 && minAno1 < 0 &&  maxAno1 <= 0
){
                plotXYCoordinates1(g2);
           }
           if (minAno >= 0 && maxAno > 0 && minAno1 >= 0 && maxAno1 > 0){
                plotXYCoordinates1(g2);
           }
           if (minAno < 0  && maxAno>0 && minAno1 < 0  && maxAno1>0 ){
                plotXYCoordinates2(g2);
           }



     }

     public void plotXYCoordinates1 (Graphics2D g2) {

           g2.setColor(Color.white);
           g2.drawLine(200, 20, 200, 50);
           g2.setFont(new Font("Arial", 20, 12));
           g2.setColor(Color.black);
           float maxval = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.ano);
           float maxval1 = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano);
           if (maxval> maxval1)
```

```
                maxY = maxval;
            else
                maxY = maxval1;
            int points = (int)maxY / 5;
            int yInterval = 50;
            g2.drawString("0", 125+50, 50);
            for (int x = yInterval, j = 1; x < 250; x += yInterval){

                g2.drawString("-", 148+50, 50 + x);
                g2.drawString("" + (points*j), 145, 50 + x);
                j++;
            }
            float prevx = (float) (200 +( 450 * obs[1] / maxX));;
            float prevy = (float)( 50 + ( 250 *
LISMAG2D_CalculateValues.ano[1] / maxY ) );
            float xpoint = 0;
            float ypoint = 0;
            float ypoint1 = 0;


            for (int k = 1; k <= i_no_obs; k++) {

                xpoint = (float)( 450 * obs[k] / maxX);
                ypoint = (float)( ( 250 *
LISMAG2D_CalculateValues.ano[k] / maxY ) );
                ypoint1 = (float)( ( 250 *
LISMAG2D_CalculateValues.ano[k] / maxY ) );

                g2.setColor(Color.BLACK);
                g2.draw(new Line2D.Float(prevx, prevy, 200+ xpoint, 50 +
ypoint ));
                g2.setColor(Color.BLUE);

                g2.setFont(new Font("Arial", 20, 30));
                g2.drawString(".", 200 + xpoint-4, 50+ypoint1+1);

                g2.setFont(new Font("Arial", 20, 12));
                g2.setColor(Color.black);
                prevx = 200 + xpoint;
                prevy = 50 + ypoint ;
            }

      }


      public void plotXYCoordinates2 (Graphics2D g2) {
```

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

            double store[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];
            double store1[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];
            double negstore[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];
            double negstore1[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];


            for(int i = 1; i <= LISMAG2D_CalculateValues.i_no_obs; i++){
                  if(LISMAG2D_CalculateValues.ano[i]>0)
                        store[i] = LISMAG2D_CalculateValues.ano[i];
                  else
                        negstore[i] = LISMAG2D_CalculateValues.ano[i];
                  if(LISMAG2D_CalculateValues.obs_ano[i]>0)
                        store1[i] = LISMAG2D_CalculateValues.obs_ano[i];
                  else
                        negstore1[i] = LISMAG2D_CalculateValues.obs_ano[i];
            }
            float maxpos = (float)
LISMAG2D_Utility.findMaximumNumber1(store);
            float maxpos1 = (float)
LISMAG2D_Utility.findMaximumNumber1(store1);
            float maxneg = (float)
LISMAG2D_Utility.findMaximumNumber1(negstore);
            float maxneg1 = (float)
LISMAG2D_Utility.findMaximumNumber1(negstore1);
            float posnum =0;

            if(maxpos>maxpos1)
                  posnum = Math.abs(maxpos);
            else
                  posnum = Math.abs(maxpos1);
            if(Math.abs(maxneg)>Math.abs(maxneg1))
                  maxY = maxneg;
            else
                  maxY = maxneg1;

            if(Math.abs(maxY)>10){

                  float prevx = (float) (200 +( 450 * obs[1] / maxX));
                  float prevy = 0;
                  if(LISMAG2D_CalculateValues.ano[1]>0){
```

```
                    prevy = 160-(float)( ( 140 *
LISMAG2D_CalculateValues.ano[1] / posnum ) );

               }
               else{
                    prevy = 160+(float)( ( 140 *
LISMAG2D_CalculateValues.ano[1] / maxY ) );

               }


               float xpoint = 0;
               float ypoint = 0;
               float ypoint1 = 0;

               g2.drawString("0",125+50,162);
               g2.drawString("-", 148+50, 164 );

               DecimalFormat f = new DecimalFormat("0.#");
               float points = maxY / 7;
               float points1 = posnum / 7;
               int yInterval=20;
               for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                    g2.drawString("-", 148+50, 164 - x );
                    g2.drawString("" + f.format(points1 * j), 145, 162
- x );
                    j++;
               }
               for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                    g2.drawString("-", 148+50, 164 + x );
                    g2.drawString("" + f.format(points * j), 145, 162 +
x );
                    j++;
               }
               for (int k = 1; k <= i_no_obs; k++) {

                    xpoint = (float)( 450 * obs[k] / maxX);
                    if(LISMAG2D_CalculateValues.ano[k]>0)
                         ypoint = 160-(float)( ( 140 *
LISMAG2D_CalculateValues.ano[k] / posnum ) );
                    else
                         ypoint = 160+(float)( ( 140 *
LISMAG2D_CalculateValues.ano[k] / maxY ) );

                    if(LISMAG2D_CalculateValues.obs_ano[k]>0)
```

```
                        ypoint1 = 160-(float)( ( 140 *
LISMAG2D_CalculateValues.obs_ano[k] / posnum ) );
                    else
                        ypoint1 = 160+(float)( ( 140 *
LISMAG2D_CalculateValues.obs_ano[k] / maxY ) );

                    g2.setColor(Color.BLACK);
                    g2.draw(new Line2D.Float(prevx, prevy, 200 +
xpoint,  ypoint  ));
                    g2.setColor(Color.BLUE);
                    g2.setFont(new Font("Arial", 20, 30));
                    g2.drawString(".", 200 + xpoint-4, ypoint1+1);

                    g2.setFont(new Font("Arial", 20, 12));
                    g2.setColor(Color.black);
                    prevx = 200 + xpoint;
                    prevy = ypoint ;
               }
          }

          else{
               maxY = -100;
               float prevx = (float) (200 +( 450 * obs[1] / maxX));
               float prevy = 0;
               if(LISMAG2D_CalculateValues.ano[1]>0)
                    prevy = 260-(float)( ( 240 *
LISMAG2D_CalculateValues.ano[1] / posnum ) );
               else
                    prevy = 260+(float)( ( 40 *
LISMAG2D_CalculateValues.ano[1] / maxY ) );

               float xpoint = 0;
               float ypoint = 0;
               float ypoint1 = 0;

               g2.drawString("0",125+50,262);
               g2.drawString("-", 148+50, 264 );
               g2.drawString("-100",145,300);

               DecimalFormat f = new DecimalFormat("0.#");
               float points1 = posnum / 8;
               int yInterval=30;
               for (int x = yInterval, j = 1; x <= 240; x+=yInterval){

                    g2.drawString("-", 148+50, 264 - x );
                    g2.drawString("" + f.format(points1 * j), 145, 262
- x );
                    j++;
```

```
                }

                for (int k = 1; k <= i_no_obs; k++) {

                      xpoint = (float)( 450 * obs[k] / maxX);
                      if(LISMAG2D_CalculateValues.ano[k]>0)
                            ypoint = 260-(float)( ( 240 *
LISMAG2D_CalculateValues.ano[k] / posnum ) );
                      else
                            ypoint = 260+(float)( ( 40 *
LISMAG2D_CalculateValues.ano[k] / maxY ) );


                      if(LISMAG2D_CalculateValues.obs_ano[k]>0)
                            ypoint1 = 260-(float)( ( 240 *
LISMAG2D_CalculateValues.obs_ano[k] / posnum ) );
                      else
                            ypoint1 = 260+(float)( ( 40 *
LISMAG2D_CalculateValues.obs_ano[k] / maxY ) );

                      g2.setColor(Color.BLACK);
                      g2.draw(new Line2D.Float(prevx, prevy, 200 +
xpoint,  ypoint  ));
                      g2.setColor(Color.BLUE);
                      g2.setFont(new Font("Arial", 20, 30));
                      g2.drawString(".", 200 + xpoint-4, ypoint1+1);

                      g2.setFont(new Font("Arial", 20, 12));
                      g2.setColor(Color.black);
                      prevx = 200 + xpoint;
                      prevy = ypoint ;
                }
          }
     }


     public void plotZCoordinates (Graphics2D g2) {
          g2.setFont(new Font("Arial", 20, 12));
          DecimalFormat d =  new DecimalFormat("0.#");
          g2.drawString("|",(float) 600-3+50  , 308);

g2.drawString(""+d.format(LISMAG2D_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);
          g2.drawString("0", 125+50, 310);
          g2.drawString("DISTANCE", 315+50, 285);
          int zInterval = 50;
          float xplot = 0;
```

```
            float xInterval = (float)
(LISMAG2D_CalculateValues.d_dis_km[i_no_obs] / 5);

            float xend =0;
            maxZ = (float)LISMAG2D_CalculateValues.d_max_dep;
            float points1 = maxZ / 5 ;
            for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                  g2.drawString("-", 148+50, 52 + x);
                  g2.drawString("" +d.format(points1 * j), 125+50, 50 +
x);
                  j++;
            }
            g2.setColor(Color.YELLOW);
            g2.fill(new Rectangle2D.Float(200, 300+(float)( 250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ), 450 , 250-(float)( 250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ)));
            g2.setColor(Color.BLACK);
            i_no_obs = LISMAG2D_CalculateValues.i_no_obs;
            obs = new double[i_no_obs+1];
            for (int i = 1; i <= i_no_obs; i++) {
                  obs[i] = LISMAG2D_CalculateValues.d_dis_km[i];
            }
            maxX = (float) obs[i_no_obs];
            maxZ = (float)LISMAG2D_CalculateValues.d_max_dep;
            float s = 0;
            float fc = 0;
            float spoint = 300;
            float fcpoint = (float)( 200 + ( 450 *
LISMAG2D_CalculateValues.d_cftnt_arr[1] / maxX ) );
            float xpoint = 0;
            float zpoint = 0;
            while (s <= LISMAG2D_CalculateValues.d_max_dep) {
                  float z1 = (float) 0.001;
                  s = s + z1;
                  fc = 0;
                  for (int i = 1; i<
LISMAG2D_CalculateValues.d_cftnt_arr.length; i++){
                        fc = (float) (fc +
LISMAG2D_CalculateValues.d_cftnt_arr[i] * Math.pow(s, i - 1));
                  }
                  xpoint = (float)( 450 * fc / maxX);
                  zpoint = (float)( 250 * s / maxZ);
                  g2.setColor(Color.BLACK);
                  if(200+xpoint >600+50)
                        xend = 600+50;
                  else
                        xend = 200+xpoint;
```

```
                g2.draw(new Line2D.Float(fcpoint, spoint,xend, 300 +
zpoint));
                g2.setColor(Color.RED);
                g2.draw(new Line2D.Float(200, 300 + zpoint, xend, 300 +
zpoint));
                fcpoint = (float)200 + xpoint;
                spoint = 300 + zpoint;
           }
           g2.setColor(Color.red);
           g2.fill(new Rectangle2D.Float(200, 300, 450 , (float)( 250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ)));
           g2.setColor(Color.WHITE);
           g2.fill(new Rectangle2D.Float(601+50, 300, 250 , 255));

           g2.setColor(Color.BLACK);
           g2.drawLine(200, 300, 650, 300);
           g2.drawLine(200, 300, 600, 300);
           g2.drawString("|",(float) 600-3 +50 , 308);

g2.drawString(""+d.format(LISMAG2D_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);

           g2.draw(new Line2D.Float(200, 50, 200, 300));
           g2.drawLine(90,10 ,950,10);
           g2.drawLine(90, 560, 950, 560);
           g2.drawLine(950, 10, 950, 560);
           g2.drawLine(90, 10, 90, 560);

           for (float x = xInterval, j = 1; x < 600; x += xInterval){
                xplot = xplot + xInterval;
                if(j > 4)
                     break;
                g2.drawString("|",(float) (200 + (450 * x / maxX) ),
308);
                g2.drawString("" + d.format(xplot), (float) (200 + (450
* x / maxX) ) - 3, 323);
                j++;
           }
           g2.setFont(new Font("Arial", 40,20));
           if(LISMAG2D_CalculateValues.val_code==1)
                g2.drawString("Vertical magnetic anomaly",660, 100);
           if(LISMAG2D_CalculateValues.val_code==2)
                g2.drawString("Horizontal magnetic anomaly",660, 100);
           if(LISMAG2D_CalculateValues.val_code==3)
                g2.drawString("Total magnetic anomaly",660, 100);


           g2.setColor(Color.BLUE);
```

```
            g2.setFont(new Font("Arial", 20,30));
            g2.drawString("....",700, 280);
            g2.setFont(new Font("Arial", 20,15));
            g2.drawString(":-Observed magnetic anomaly",735, 280);
            g2.setColor(Color.BLACK);
            g2.drawString("-------:-Calculated magnetic anomaly",700,
300);
      }

}
```

```
package com.lismag2d.view.graph;


import java.awt.*;
import java.awt.geom.Line2D;
import java.text.DecimalFormat;
import com.lismag2d.model.LISMAG2D_CalculateValues;
import com.lismag2d.util.LISMAG2D_Utility;
import com.lismag2d.view.LISMAG2D_MainPanel;


public class LISMAG2D_PlainGraph {


      int i_no_obs;
      public static float maxX1,maxY;
      public void drawPlainGraph(Graphics2D g2){
            g2 = (Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
            g2.setColor(Color.BLACK);
            g2.setFont(new Font("Arial", 20, 12));
            i_no_obs = LISMAG2D_CalculateValues.i_no_obs;
            double obser[] = new double[i_no_obs + 1];
            for (int i = 1; i <= i_no_obs; i++) {

                  obser[i] = LISMAG2D_CalculateValues.d_dis_km[i];
            }
            maxX1 = (float) LISMAG2D_CalculateValues.d_dis_km[i_no_obs];
            float maxZ1 = (float)LISMAG2D_CalculateValues.d_max_dep;

            int zInterval = 50;
            DecimalFormat f = new DecimalFormat("0.#");
            g2.drawLine(200,20,200,300);

            float xplot = 0;
```

```
          float xInterval = (float)
(LISMAG2D_CalculateValues.d_dis_km[i_no_obs] / 5);
          for (float x = xInterval, j = 1; x < 600; x += xInterval){
               xplot = xplot + xInterval;
               if(j > 4)
                    break;
               g2.drawString("|",(float) (200 + (450 * x / maxX1) ),
308);
               g2.drawString("" + f.format(xplot), (float) (150 + (450
* x / maxX1) ) - 3+50, 323);
               j++;
          }
          DecimalFormat d =  new DecimalFormat("0.#");
          float maxZ = (float)LISMAG2D_CalculateValues.d_max_dep;
          float points1 = maxZ / 5 ;
          for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
               g2.drawString("-", 148+50, 52 + x);
               g2.drawString("" +d.format(points1 * j), 125+50, 50 +
x);
               j++;
          }

          g2.drawString("|",(float) 600-3+50  , 308);

g2.drawString(""+f.format(LISMAG2D_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);
          g2.drawString("0", 125+50, 310);
          g2.drawString("DISTANCE", 315+50, 285);

          double minAno =
LISMAG2D_Utility.findMinimumNumber1(LISMAG2D_CalculateValues.obs_ano);
          double maxAno =
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano);


          if (  maxAno <= 0 &&   minAno <= 0 ){
               plotXYCoordinates1(g2);
          }
          if (minAno >= 0 && maxAno > 0 ){
               plotXYCoordinates1(g2);
          }
          if (minAno < 0  && maxAno>0  ){
               plotXYCoordinates2(g2);
          }

          g2.draw(new Line2D.Float(200, 300, 650, 300));
          g2.draw(new Line2D.Float(200, 300, 200, 550));
```

```
            g2.draw(new Line2D.Float(650, 300, 650, 300 + (float)(250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ1)));
            g2.draw(new Line2D.Float(200, 300 + (float)(250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ1), 650, 300 + (float)(250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ1)));

            g2.setColor(Color.BLUE);
            g2.setFont(new Font("Arial", 20,30));
            g2.drawString("....",750, 380);
            g2.setFont(new Font("Arial", 20,15));
            g2.drawString(":-Observed magnetic anomaly",785, 380);

            g2.setFont(new Font("Arial", 40, 20));
            g2.setColor(Color.black);
            g2.drawLine(615, 0, 615, 125);
            g2.drawLine(615, 125, 990, 125);
            g2.drawLine(990, 125, 990, 0);
            g2.drawString("Instructions", 730, 20);
            g2.drawString("_____________", 710, 20);
            g2.setFont(new Font("Arial", 40, 12));
            g2.setColor(Color.red);
            g2.drawString("1) Select points on the fault plane by clicking
the mouse ", 640, 40);
            g2.drawString("in the structure panel ", 644, 60);

            g2.setColor(Color.red);
            g2.drawString("2) Select the points within the specified
boundary ", 640, 80);
            g2.drawString("of the structure panel", 644, 100);


            g2.setColor(Color.black);
            String []a = {"A", "N", "O", "M", "A", "L", "Y", "(nT)"};
            String []b = {"D", "E", "P", "T", "H"};
            for (int i = 0; i < a.length; i++) {
                  g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
            }
            for (int i = 0; i < b.length; i++) {
                  g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
            }


      }

      public void plotXYCoordinates1 (Graphics2D g2) {
            g2 = (Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
            g2.setColor(Color.white);
            g2.drawLine(200, 20, 200, 50);
```

```
            g2.setFont(new Font("Arial", 20, 12));
            g2.setColor(Color.black);
            float maxval1 = (float)
LISMAG2D_Utility.findMaximumNumber(LISMAG2D_CalculateValues.obs_ano);
            maxY = maxval1;
            int points = (int)maxY / 5;
            int yInterval = 50;
            g2.drawString("0", 125+50, 50);
            for (int x = yInterval, j = 1; x < 250; x += yInterval){

                  g2.drawString("-", 148+50, 50 + x);
                  g2.drawString("" + (points*j), 145, 50 + x);
                  j++;
            }

            float xpoint1 = 0;
            float ypoint1 = 0;

            for (int k = 1; k <= i_no_obs; k++) {

                  ypoint1 = (float)( ( 250 *
LISMAG2D_CalculateValues.obs_ano[k] / maxY ) );
                  g2.setColor(Color.BLUE);
                  g2.setFont(new Font("Arial", 20, 30));
                  g2.drawString(".", 200 + xpoint1-4, 50+ypoint1+1);

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

            }

      }


      public void plotXYCoordinates2 (Graphics2D g2) {
            g2 = (Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
            g2.setFont(new Font("Arial", 20, 12));
            g2.setColor(Color.black);
            double store1[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];
            double negstore1[] = new
double[LISMAG2D_CalculateValues.i_no_obs+1];

            for(int i = 1; i <= LISMAG2D_CalculateValues.i_no_obs; i++){

                  if(LISMAG2D_CalculateValues.obs_ano[i]>0)
                        store1[i] = LISMAG2D_CalculateValues.obs_ano[i];
                  else
```

```
                    negstore1[i] = LISMAG2D_CalculateValues.obs_ano[i];
          }

          float maxpos1 = (float)
LISMAG2D_Utility.findMaximumNumber1(store1);
          float maxneg1 = (float)
LISMAG2D_Utility.findMaximumNumber1(negstore1);
          float posnum =0;
          posnum = Math.abs(maxpos1);

          maxY = maxneg1;

          if(Math.abs(maxY)>10){

               float xpoint = 0;
               float ypoint1 = 0;

               g2.drawString("0",125+50,162);
               g2.drawString("-", 148+50, 164 );

               DecimalFormat f = new DecimalFormat("0.#");
               float points = maxY / 7;
               float points1 = posnum / 7;
               int yInterval=20;
               for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                    g2.drawString("-", 148+50, 164 - x );
                    g2.drawString("" + f.format(points1 * j), 145, 162
- x );
                    j++;
               }
               for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                    g2.drawString("-", 148+50, 164 + x );
                    g2.drawString("" + f.format(points * j), 145, 162 +
x );
                    j++;
               }
               for (int k = 1; k <= i_no_obs; k++) {

                    xpoint = (float)( 450 *
LISMAG2D_CalculateValues.d_dis_km[k] / maxX1);

                    if(LISMAG2D_CalculateValues.obs_ano[k]>0)
                         ypoint1 = 160-(float)( ( 140 *
LISMAG2D_CalculateValues.obs_ano[k] / posnum ) );
                    else
```

```
                        ypoint1 = 160+(float)( ( 140 *
LISMAG2D_CalculateValues.obs_ano[k] / maxY ) );

                    g2.setColor(Color.BLUE);
                    g2.setFont(new Font("Arial", 20, 30));
                    g2.drawString(".", 200 + xpoint-4, ypoint1+1);

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

          else{
               maxY = -100;

               float xpoint = 0;
               float ypoint1 = 0;

               g2.drawString("0",125+50,262);
               g2.drawString("-", 148+50, 264 );
               g2.drawString("-100",145,300);

               DecimalFormat f = new DecimalFormat("0.#");
               float points1 = posnum / 8;
               int yInterval=30;
               for (int x = yInterval, j = 1; x <= 240; x+=yInterval){

                    g2.drawString("-", 148+50, 264 - x );
                    g2.drawString("" + f.format(points1 * j), 145, 262
- x );
                    j++;
               }

               for (int k = 1; k <= i_no_obs; k++) {

                    xpoint = (float)( 450 *
LISMAG2D_CalculateValues.d_dis_km[k] / maxX1);
                    if(LISMAG2D_CalculateValues.obs_ano[k]>0)
                         ypoint1 = 260-(float)( ( 240 *
LISMAG2D_CalculateValues.obs_ano[k] / posnum ) );
                    else
                         ypoint1 = 260+(float)( ( 40 *
LISMAG2D_CalculateValues.obs_ano[k] / maxY ) );

                    g2.setColor(Color.BLUE);
                    g2.setFont(new Font("Arial", 20, 30));
                    g2.drawString(".", 200 + xpoint-4, ypoint1+1);
```

```
                    g2.setFont(new Font("Arial", 20, 12));
                    g2.setColor(Color.black);
                }
            }
      }

}
```

```
package com.lismag2d.view.event;

import com.lismag2d.model.LISMAG2D_CalculateValues;


public class LISMAG2D_StoreValues {
      public static double values[][] = new double [50][50];

      public void store(){

            for(int i = 1; i<=LISMAG2D_CalculateValues.len;i++ ){

                  values[i][1]= LISMAG2D_PlotFault.val[i];
                  values[i][2]= LISMAG2D_PlotFault.val1[i];

            }
      }

}
```

```
package com.lismag2d.view.event;


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

import javax.swing.JScrollPane;

import com.lismag2d.model.LISMAG2D_CalculateValues;
import com.lismag2d.view.LISMAG2D_MainPanel;
import com.lismag2d.view.graph.LISMAG2D_PlainGraph;

public class LISMAG2D_PlotFault extends Applet{
```

```
      public static double val[] ;
      public static double val1[];
      public static int pos;
      public static MouseListener ml;
      com.lismag2d.model.LISMAG2D_CalculateValues cv = new
com.lismag2d.model.LISMAG2D_CalculateValues();
      LISMAG2D_PlainGraph pg = new LISMAG2D_PlainGraph();
      LISMAG2D_StoreValues sv = new LISMAG2D_StoreValues();
      public void paint(Graphics g){

            LISMAG2D_MainPanel.p_Center.removeAll();
            LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.im);
            LISMAG2D_CalculateValues.len=0;
            val = new double[50] ;
            val1 = new double[50] ;
            Graphics2D g2 =
(Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
            g2.setFont(new Font("Arial", 40,12));
            g2.setColor(Color.red);
            LISMAG2D_MainPanel.im.setEditable(false);
            LISMAG2D_MainPanel.im.setBackground(Color.WHITE);
            LISMAG2D_MainPanel.p_Center.validate();

            for (int j = 0; j < val.length; j++){
                  val[j] = 0;
                  val1[j] = 0;
            }
            pos = 1;

cv.getAnamolyValues(com.lismag2d.view.LISMAG2D_MainPanel.captureValues())
;
            Graphics2D g1 =
(Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
            pg.drawPlainGraph(g1);

            ml = new MouseAdapter(){
                  public void mouseClicked(MouseEvent e) {

                        Graphics2D g2 =
(Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
                        pg.drawPlainGraph(g2);
                        DecimalFormat f = new DecimalFormat("0.##");
                        float polyx = e.getX();
                        float polyy = e.getY();
                        float maxZ =
(float)LISMAG2D_CalculateValues.d_max_dep;
```

```
                        float zer = (float) (250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ);
                        float zer1 = (float) (250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ);
                        float x[] = new float[50];
                        float z[] = new float[50];
                        int get = pos;

                        LISMAG2D_CalculateValues.len = get;

                        if (polyy >= 300+zer1 && polyy <= (300 + zer) &&
polyx >= 200 && polyx <= 650){

                              LISMAG2D_PlotFault.val[get] = (float)
Math.abs((LISMAG2D_PlainGraph.maxX1 * (200 - polyx)) / 450);
                              LISMAG2D_PlotFault.val1[get] = (float)
Math.abs(LISMAG2D_CalculateValues.d_max_dep * (300 - polyy) / zer);
                              x[get] = polyx;
                              z[get] = polyy;
                              g2.setColor(Color.white);
                              g2.draw(new Line2D.Float(200, polyy, 650,
polyy ) );
                              g2.draw(new Line2D.Float(650, polyy, 650 +
65, polyy - 40 ) );
                              g2.fillRect(730, 160, 250, 40);
                              g2.setFont(new Font("Arial", 40, 12));
                              g2.setColor(Color.BLACK);
                              g2.drawString("X", polyx-3, polyy+4);

g2.drawString("("+f.format(LISMAG2D_PlotFault.val[get])
+","+f.format(LISMAG2D_PlotFault.val1[get])+")", 10+polyx, 4+polyy);
                              g2.setFont(new Font("Arial", 40, 20));
                              if(get<=LISMAG2D_CalculateValues.i_d_poly)
                                    g2.setColor(Color.red);
                              else
                                    g2.setColor(Color.blue);

                              g2.drawString("Number of control points
"+get, 720, 180);
                              g2.setFont(new Font("Arial", 40, 12));
                              LISMAG2D_PlotFault.pos++;
                        }

                        pg.drawPlainGraph(g2);
                        for (int i = 1; i <= LISMAG2D_CalculateValues.len;
i++){
                              g2.setColor(Color.BLACK);
```

```
g2.drawString("("+f.format(LISMAG2D_PlotFault.val[LISMAG2D_CalculateValue
s.len])
+","+f.format(LISMAG2D_PlotFault.val1[LISMAG2D_CalculateValues.len])+")",
10+x[get], z[get]+4);
                    }
                    for (int i = 1; i <= LISMAG2D_CalculateValues.len;
i++){
                         float xpoint1  = (float)
(450*LISMAG2D_PlotFault.val[i]/LISMAG2D_PlainGraph.maxX1);
                         float zpoint1  = (float)(250*
LISMAG2D_PlotFault.val1[i]/LISMAG2D_CalculateValues.d_max_dep);

                         g2.setColor(Color.BLACK);
                         if(LISMAG2D_CalculateValues.len>1){

g2.drawString("("+f.format(LISMAG2D_PlotFault.val[i])
+","+f.format(LISMAG2D_PlotFault.val1[i])+")", 210+xpoint1, 304+zpoint1);
                              g2.drawString("X", 200+xpoint1-3,
304+zpoint1);
                         }
                    }

                    sv.store();
               }

               public void mouseMoved(MouseEvent e){

                    pg.drawPlainGraph(g2);
                    DecimalFormat f = new DecimalFormat("0.##");
                    float xer = e.getX();
                    float yer = e.getY();
                    float maxZ =
(float)LISMAG2D_CalculateValues.d_max_dep;
                    float zer = (float) (250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ);
                    float xpoint =(float)
Math.abs((LISMAG2D_PlainGraph.maxX1 * (200 - xer)) / 450);
                    float zpoint = (float)
Math.abs(LISMAG2D_CalculateValues.d_max_dep * (300 - yer) / zer);
                    if (yer >= 300 && yer <= (300 + zer) && xer >= 200
&& xer <= 650){
                         Graphics2D g2 =
(Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
                         g2.setColor(Color.black);
                         g2.drawString("X-coordinates            Z-
coordinates", 780, 320);
                         g2.setColor(Color.red);
```

```
                              g2.fillRect(780, 320, 190, 20);
                              g2.setColor(Color.black);
                              g2.drawString(""+f.format(xpoint)+"
"+f.format(zpoint)+"", 780, 330);


                         }
                         pg.drawPlainGraph(g2);

                    }

               };

               LISMAG2D_MainPanel.im.addMouseListener(ml);

LISMAG2D_MainPanel.im.addMouseMotionListener((MouseMotionListener)ml);
               g2.setFont(new Font("Arial", 40, 20));
               g2.setColor(Color.black);
               g2.drawLine(615, 0, 615, 125);
               g2.drawLine(615, 125, 990, 125);
               g2.drawString("Instructions", 730, 20);
               g2.drawString("_____________", 710, 20);
               g2.setFont(new Font("Arial", 40, 12));
               g2.setColor(Color.red);
               g2.drawString("1) Select points on the fault plane by clicking
the mouse ", 640, 40);
               g2.drawString("in the structure panel ", 644, 60);

               g2.setColor(Color.red);
               g2.drawString("2) Select the points within the specified
boundary ", 640, 80);
               g2.drawString("of the structure panel", 644, 100);


          }

}
```

```
package com.lismag2d.view.event;



import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;

import com.lismag2d.view.graph.LISMAG2D_PlainGraph;
```

```
import com.lismag2d.model.LISMAG2D_CalculateValues;
import com.lismag2d.util.LISMAG2D_HandleException;
import com.lismag2d.view.LISMAG2D_MainPanel;
import com.lismag2d.view.LISMAG2D_TableView;


public class LISMAG2D_EditFault {
      public static float zpoint = 0;
      public static int count = 0;
      public static MouseListener ml1;

      com.lismag2d.view.LISMAG2D_DrawGraph dg = new
com.lismag2d.view.LISMAG2D_DrawGraph();
      LISMAG2D_PlainGraph pg = new LISMAG2D_PlainGraph();
      com.lismag2d.model.LISMAG2D_CalculateValues cv = new
com.lismag2d.model.LISMAG2D_CalculateValues();
      public void paint(Graphics g){
            final DecimalFormat f = new DecimalFormat("0.##");
            LISMAG2D_MainPanel.p_Center.removeAll();
            LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.fl);
            LISMAG2D_MainPanel.fl.setEditable(false);
            LISMAG2D_MainPanel.fl.setBackground(Color.WHITE);
            LISMAG2D_MainPanel.p_Center.validate();


cv.getAnamolyValues(com.lismag2d.view.LISMAG2D_MainPanel.captureValues())
;
            try {
                  cv.getCoefficients();
                  cv.cal();

com.lismag2d.view.LISMAG2D_TableView.populateEastPanel(LISMAG2D_Calculate
Values.obj);
                  setGraphValues();

            } catch (LISMAG2D_HandleException e2) {

                  e2.printStackTrace();
            }
            Graphics2D g2 =
(Graphics2D)LISMAG2D_MainPanel.fl.getGraphics();

            dg.plot(g2);
            dg.drawGraph(g2);
            dg.plotXYCoordinates(g2);
            dg.plotZCoordinates(g2);
            g2.setFont(new Font("Arial", 20, 12));
            dg.plot(g2);
```

```
            g2.setColor(Color.black);
            for(int i = 1;i<=LISMAG2D_CalculateValues.len;i++){
                  float xpoint  = (float)
(450*LISMAG2D_CalculateValues.d_x_km_arr[i]/LISMAG2D_PlainGraph.maxX1);
                  float zpoint  = (float)(250*
LISMAG2D_CalculateValues.d_z_km_arr[i]/
LISMAG2D_CalculateValues.d_max_dep);

if(LISMAG2D_CalculateValues.d_z_km_arr[i]<=LISMAG2D_CalculateValues.d_max
_dep){

g2.drawString("("+f.format(LISMAG2D_CalculateValues.d_x_km_arr[i])
+","+f.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+")", 160+xpoint+50,
304+zpoint);
                        g2.drawString("X", 200+xpoint-3, 304+zpoint);
                  }
            }

            ml1 = new MouseAdapter(){

                  public void mouseDragged(MouseEvent e1) {
                        DecimalFormat f = new DecimalFormat("0.##");
                        float x2 = e1.getX();
                        float y2 = e1.getY();
                        float maxZ =
(float)LISMAG2D_CalculateValues.d_max_dep;
                        float zer = (float) (250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ);
                        float zer1 = (float) (250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ);
                        Graphics2D g2 =
(Graphics2D)LISMAG2D_MainPanel.fl.getGraphics();
                        if (y2 > 300+zer1 && y2 <= (300 + zer) && x2>200 &&
x2 <= 600+50){

                              g2.setColor(Color.red);
                              g2.fillRect(700, 180, 210, 30);
                              g2.setColor(Color.BLACK);
                              zpoint = (float)
Math.abs((LISMAG2D_PlainGraph.maxX1 * (200 - x2)) / 450);
                              g2.drawString("Sl.no
x-coordinate(km)", 700, 180);
                              g2.drawString(""+f.format(zpoint), 870, 200);
                              g2.setColor(Color.black);
                              g2.drawString(""+(count ), 700, 200);

                        }
```

```
                    try {
                         cv.getCoefficients();
                         cv.cal();

com.lismag2d.view.LISMAG2D_TableView.populateEastPanel(LISMAG2D_Calculate
Values.obj);
                         setGraphValues();


                    } catch (LISMAG2D_HandleException e2) {

                         e2.printStackTrace();
                    }

                    dg.plot(g2);
                    dg.drawGraph(g2);
                    dg.plotXYCoordinates(g2);
                    dg.plotZCoordinates(g2);
                    g2.setFont(new Font("Arial", 20, 12));
                    dg.plot(g2);

                    for(int i = 1;i<=LISMAG2D_CalculateValues.len;i++){
                         float xpoint  = (float)
(450*LISMAG2D_CalculateValues.d_x_km_arr[i]/LISMAG2D_PlainGraph.maxX1);
                         float zpoint  = (float)(250*
LISMAG2D_CalculateValues.d_z_km_arr[i]/
LISMAG2D_CalculateValues.d_max_dep);

if(LISMAG2D_CalculateValues.d_z_km_arr[i]<=LISMAG2D_CalculateValues.d_max
_dep){

g2.drawString("("+f.format(LISMAG2D_CalculateValues.d_x_km_arr[i])
+","+f.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+")", 160+xpoint+50,
304+zpoint);
                                   g2.drawString("X", 200+xpoint-3,
304+zpoint);
                         }
                    }

               }


               public void mousePressed(MouseEvent e1) {

                    float x1 = e1.getX();
                    float y1 = e1.getY();
```

```
                        float maxZ =
(float)LISMAG2D_CalculateValues.d_max_dep;
                        float diff = (float)
(LISMAG2D_CalculateValues.d_dis_km[LISMAG2D_CalculateValues.i_no_obs]/
100);
                        float zer = (float) (250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ);
                        float zer1 = (float) (250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ);
                        float xer = (float)
Math.abs((LISMAG2D_PlainGraph.maxX1 * (200 - (x1-3))) / 450);

                        Graphics2D g2 =
(Graphics2D)LISMAG2D_MainPanel.fl.getGraphics();
                        if (y1 >= 300+zer1 && y1 < (300 + zer) && x1>200 &&
x1 <= 600+50){


                              for (int kk = 1; kk <=
LISMAG2D_CalculateValues.len; kk++){

                                    if
(Math.abs((float)LISMAG2D_CalculateValues.d_x_km_arr[kk] - xer )<= diff
|| Math.abs((float)LISMAG2D_CalculateValues.d_z_km_arr[kk] - zer )<= 0 ){
                                          if (kk>0);
                                          count = kk;

                                    }

                              }
                              g2.setColor(Color.red);
                              g2.fillRect(700, 180, 210, 30);
                              g2.setColor(Color.black);
                              zpoint = (float)
Math.abs((LISMAG2D_PlainGraph.maxX1 * (200 - x1)) / 450);
                              g2.drawString("Sl.no
x-coordinate(km)", 700, 180);
                              g2.drawString(""+f.format(zpoint), 870, 200);

                              g2.drawString(""+(count ), 770, 200);

                              dg.drawGraph(g2);
                              dg.plotXYCoordinates(g2);
                              dg.plotZCoordinates(g2);

                              for(int i =
1;i<=LISMAG2D_CalculateValues.len;i++){
```

```
                                    float xpoint  = (float)
(450*LISMAG2D_CalculateValues.d_x_km_arr[i]/LISMAG2D_PlainGraph.maxX1);
                                    float zpoint  = (float)(250*
LISMAG2D_CalculateValues.d_z_km_arr[i]/
LISMAG2D_CalculateValues.d_max_dep);

if(LISMAG2D_CalculateValues.d_z_km_arr[i]<=LISMAG2D_CalculateValues.d_max
_dep){

g2.drawString("("+f.format(LISMAG2D_CalculateValues.d_x_km_arr[i])
+","+f.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+")", 160+xpoint+50,
304+zpoint);
                                          g2.drawString("X", 200+xpoint-3,
304+zpoint);
                                    }
                               }

                          }

                     }

                     public void mouseReleased(MouseEvent e1) {

                          float x = e1.getX();
                          float y = e1.getY();
                          float maxZ =
(float)LISMAG2D_CalculateValues.d_max_dep;
                          float zer = (float) (250 *
LISMAG2D_CalculateValues.d_max_dep / maxZ);
                          float zer1 = (float) (250 *
LISMAG2D_CalculateValues.d_min_dep / maxZ);
                          Graphics2D g2 =
(Graphics2D)LISMAG2D_MainPanel.fl.getGraphics();
                          if (y > 300+zer1 && y <= (300 + zer) && x>200 && x
<= 600+50){

                               float xer = (float)
Math.abs((LISMAG2D_PlainGraph.maxX1 * (200 - x)) / 450);
                               if(count>0){
                                    LISMAG2D_PlotFault.val[count]=xer;

LISMAG2D_CalculateValues.d_x_km_arr[count] = xer;
                               }
                          }

                          try {

                               cv.getCoefficients();
```

```
                              cv.cal();

com.lismag2d.view.LISMAG2D_TableView.populateEastPanel(LISMAG2D_Calculate
Values.obj);
                              setGraphValues();

                         } catch (LISMAG2D_HandleException e) {
                              e.printStackTrace();
                         }

LISMAG2D_MainPanel.clearPanel(LISMAG2D_MainPanel.fl);

                         Graphics2D g1 =
(Graphics2D)LISMAG2D_MainPanel.fl.getGraphics();
                         dg.plot(g2);
                         dg.drawGraph(g2);
                         dg.plotXYCoordinates(g2);
                         dg.plotZCoordinates(g1);
                         g2.setFont(new Font("Arial", 20, 12));
                         dg.plot(g2);
                         g1.setColor(Color.black);
                         for(int i = 1;i<=LISMAG2D_CalculateValues.len;i++){
                              float xpoint  = (float)
(450*LISMAG2D_CalculateValues.d_x_km_arr[i]/LISMAG2D_PlainGraph.maxX1);
                              float zpoint  = (float)(250*
LISMAG2D_CalculateValues.d_z_km_arr[i]/
LISMAG2D_CalculateValues.d_max_dep);

if(LISMAG2D_CalculateValues.d_z_km_arr[i]<=LISMAG2D_CalculateValues.d_max
_dep){

g2.drawString("("+f.format(LISMAG2D_CalculateValues.d_x_km_arr[i])
+","+f.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+")", 160+xpoint+50,
304+zpoint);
                                   g2.drawString("X", 200+xpoint-3,
304+zpoint);
                              }
                         }
                    }

               };

               LISMAG2D_MainPanel.fl.addMouseListener(ml1);

LISMAG2D_MainPanel.fl.addMouseMotionListener((MouseMotionListener)ml1);
          }

          public static void  setGraphValues() {
```

```
            DecimalFormat d = new DecimalFormat("0.##");
            DecimalFormat df1 = new DecimalFormat("0.####");

            LISMAG2D_TableView.val.setText("");
            LISMAG2D_TableView.val.append("Coordinates of selected points
on the fault plane(x,z):-\n");

LISMAG2D_TableView.val.appendText("-------------------------------------
-----\n");
            for (int i = 1; i <= LISMAG2D_CalculateValues.len; i++){


LISMAG2D_TableView.val.appendText("("+d.format(LISMAG2D_CalculateValues.d
_x_km_arr[i])+","+d.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+")\n")
;

            }
            LISMAG2D_TableView.val.appendText("\n");
            LISMAG2D_TableView.val.append("Polynomial coefficients of the
fault plane:-\n");

LISMAG2D_TableView.val.appendText("-------------------------------------
-----\n");
            LISMAG2D_TableView.val.appendText("\n");

            for (int i = 1; i <= LISMAG2D_CalculateValues.i_d_poly+1; i++)
{
                  LISMAG2D_TableView.val.appendText("f"+(i - 1)+" = "
+df1.format(LISMAG2D_CalculateValues.d_cftnt_arr[i])+ "\n");
            }
      }
}
```

```
package com.lismag2d.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.Arrays;
```

```
import java.util.HashMap;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

import com.lismag2d.util.LISMAG2D_HandleException;
import com.lismag2d.util.LISMAG2D_Utility;
import com.lismag2d.view.LISMAG2D_TableView;

import com.lismag2d.view.event.LISMAG2D_PlotFault;
import com.lismag2d.view.LISMAG2D_MainPanel;


public class LISMAG2D_CalculateValues {

      public static Object obj[][] = null;

      public static int i_no_obs, i_d_poly,len, val_code = 0;
      public static double  d_max_dep, d_min_dep, val_iom, val_dom,
val_str = 0;
      public static double  d_dis_km[], obs_ano[],ano[], d_cftnt_arr[],
x[], cftnt[] = null;

      public static String  input_profile_num = "";
      public static double d_x_km_arr[], d_z_km_arr[];
      public static BufferedImage image;

      public void getAnamolyValues(HashMap h_Map) {

            try {
                  i_no_obs =
LISMAG2D_Utility.convertInteger((String)h_Map.get("N_OBS"));
                  i_d_poly =
LISMAG2D_Utility.convertInteger((String)h_Map.get("D_POLY"));
                  d_dis_km =
LISMAG2D_Utility.convertDoubleArray((String)h_Map.get("DIS_KM"));
                  obs_ano =
LISMAG2D_Utility.convertDoubleArray((String)h_Map.get("OBS_ANO"));
                  d_min_dep =
LISMAG2D_Utility.convertDouble((String)h_Map.get("MIN_DEP"));
                  d_max_dep =
LISMAG2D_Utility.convertDouble((String)h_Map.get("MAX_DEP"));
                  val_str =
LISMAG2D_Utility.convertDouble((String)h_Map.get("VAL_STR"));
                  input_profile_num =
LISMAG2D_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
```

```
                val_iom =
LISMAG2D_Utility.convertDouble((String)h_Map.get("VAL_IOM"));
                val_dom =
LISMAG2D_Utility.convertDouble((String)h_Map.get("VAL_DOM"));
                val_code =
LISMAG2D_Utility.convertInteger((String)h_Map.get("VAL_CODE"));
           }
           catch(Exception e) {

                e.printStackTrace();
           }
     }



     public void cal() throws LISMAG2D_HandleException{

           getCoefficients();

           double z[] = new double[1000];
           double gs[] = new double[1000];
           ano = new double[i_no_obs + 1];
           x = new double[i_no_obs + 1];
           double dm = 0;
           double gc = 0;
           if(val_code == 1)
                dm = 90;
           if(val_code == 2)
                dm = 0;
           if(val_code == 3)
                dm = val_dom;
           for (int i = 1; i <= i_no_obs; i++){

                x[i] = d_dis_km[i];

           }

           cftnt = d_cftnt_arr;
           double zt = d_min_dep;

           double rad = 3.14159265 / 180;
           double dmr = rad * dm;
           double aphikt = val_dom * rad;
           double strr = val_str * rad;
           double cont = 2 * val_iom * Math.sqrt(1 -
(Math.pow(Math.cos(strr),2) * Math.pow(Math.cos(dmr),2)));
           double ter = aphikt - Math.atan(Math.sin(strr) /
Math.tan(dmr));
```

```
            double dx = (x[2] - x[1]) / 10;
            double zdif = d_max_dep - zt;
            int nd = (int)(zdif / dx) + 1;
            double na1 = nd / 2;

            if(nd - 2 * na1 != 0)
                  nd = nd + 1;
            double dz = zdif / nd;
            int n2 = nd + 1;

            for(int jz = 1; jz <= n2; jz++){
                  z[jz] = zt + dz * (jz - 1);
            }


            for(int k = 1; k<=i_no_obs; k++)
            {
                  double XX = x[k];
                  for(int jz = 1; jz <= n2; jz++)
                  {
                        double sum = 0.0;

                        for (int jj = 1; jj <= i_d_poly + 1; jj++){

                              sum = sum + cftnt[jj] * Math.pow(z[jz], jj -
1);

                        }

                        double c = Math.cos(ter);
                        double s = Math.sin(ter);
                        double     TNUM1 = (z[jz]) * c;
                        double     TNUM2 = (sum - XX) * s;
                        double     TDEN1 = Math.pow((z[jz]), 2);
                        double     TDEN2 = Math.pow((sum - XX), 2);
                        gs[jz] = cont * ((TNUM1 - TNUM2) / (TDEN1 +
TDEN2));
                  }

                  gc = SIMP(gs,z,n2);
                  ano[k]=gc;

            }
            setGraphValues(x, ano, cftnt);

      }
```

```
public double SIMP(double []gs,double []z,int n) {
      double ggc=0;
      double dz = z[2] - z[1];
      double sum1 = 0;
      double sum2 = 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 void getCoefficients() throws LISMAG2D_HandleException{

      d_x_km_arr = new double[len + 1];
      d_z_km_arr = new double[len + 1];

      double []str = new double[(i_d_poly+1)*(i_d_poly+1)];
      if (len <= i_d_poly){
            JFrame frame = null;
            JOptionPane.showMessageDialog(frame,
                        "The polynomial fit requires\n"
                                    +"Order+1 data points.\n"
                                    +"Use additional data points",
                                    "Error!",
                                    JOptionPane.ERROR_MESSAGE);
            throw new LISMAG2D_HandleException();
      }

      for (int i = 0; i <= len; i++){
            if ( i == 0){
                  d_x_km_arr[i] = 0;

                  d_z_km_arr[i] = 0;
            }

            d_x_km_arr[i] = LISMAG2D_PlotFault.val[i];
            d_z_km_arr[i] = LISMAG2D_PlotFault.val1[i];
```

```
            }

            Arrays.sort(d_x_km_arr);
            Arrays.sort(d_z_km_arr);
            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 + 2] ;
            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_x_km_arr[i];
                        }
                        else{
                              double d_sum_xz = d_x_km_arr[i] *
Math.pow(d_z_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_z_km_arr[i];
                              d_rmatrix_arr[l] = d_sum_rmatrix;
                        }
                        else{

                              double d_sum_zpow = Math.pow(d_z_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];
                    }

               }
          }

          for (int i = 0; i < i_d_poly + 1; i++){

               for (int j = 0; j < i_d_poly + 1; j++){

                    str[(i*(i_d_poly+1))+j]= d_coeff_zval_arr[i][j];

               }
          }

          d_coeff_values   = guaSolve(d_coeff_zval_arr,
d_coeff_xzval_arr);

          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++){

               d_cftnt_arr[i] = d_coeff_values[i - 1];

          }
```

```
	}


	public double[] guaSolve(double a_calmat[][], double b_calmat[]){

		int n = b_calmat.length;
		double x_cal_mat[] = new double[n + 2];

		for (int i = 0; i < i_d_poly + 1; i++){

			for (int j = 0; j <= i_d_poly + 1; j++){
				if(j == i_d_poly + 1 ){
					a_calmat[i][j] = b_calmat[i];
				}

			}
		}

		for(int l = 0; l < n-1 ; l++ ){
			for(int i = l+1; i < n ; i++ ){
				for(int j = l+1; j < n + 1; j++){
					a_calmat[i][j] = a_calmat[i][j] - a_calmat[i]
[l] / a_calmat[l][l] * a_calmat[l][j];
				}
			}
		}

		x_cal_mat[n-1] = a_calmat[n - 1][n] / a_calmat[n - 1][n - 1];
		for (int k = n-1; k >= 0; k--){
			double cal_tot = 0;

			for(int j = k+1; j< n; j++){


				cal_tot = cal_tot + a_calmat[k][j] * x_cal_mat[j];
			}
			x_cal_mat[k] = 1 / a_calmat[k][k] * (a_calmat[k][n] -
cal_tot);
		}

		return x_cal_mat;
	}

	public static void  setGraphValues(double []x, double
[]anomaly,double []coeff) {

		obj = new Object[i_no_obs + 1][3];
```

```
            DecimalFormat d = new DecimalFormat("0.##");
            DecimalFormat df = new DecimalFormat("0.###");
            DecimalFormat df1 = new DecimalFormat("0.####");
            for (int K = 1;K <= i_no_obs; K++){

                  obj[K][0]= "" + d.format(x[K]);
                  obj[K][1]= "" + df.format(obs_ano[K]);
                  obj[K][2]= "" + df.format(anomaly[K]);

            }

            LISMAG2D_TableView.val.setText("");
            LISMAG2D_TableView.val.append("Coordinates of control
points(x,z):-\n");

LISMAG2D_TableView.val.appendText("-------------------------------------
-----\n");
            for (int i = 1; i <= len; i++){


LISMAG2D_TableView.val.appendText("("+d.format(d_x_km_arr[i])
+","+d.format(d_z_km_arr[i])+")\n") ;

            }
            LISMAG2D_TableView.val.appendText("\n");
            LISMAG2D_TableView.val.append("Coefficients of the
polynomial:-\n");

LISMAG2D_TableView.val.appendText("-------------------------------------
-----\n");
            LISMAG2D_TableView.val.appendText("\n");

            for (int i = 1; i < coeff.length; i++){
                  LISMAG2D_TableView.val.appendText("f"+(i - 1)+" = "
+df1.format(coeff[i])+ "\n");
            }


      }


      public static void drawGraph(){
            final com.lismag2d.view.LISMAG2D_DrawGraph dg = new
com.lismag2d.view.LISMAG2D_DrawGraph();
            try
            {
                  int width = 1280;
                  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);
                dg.plotXYCoordinates(g2);
                dg.plotZCoordinates(g2);
                dg.plot(g2);


                FileOutputStream os = new
FileOutputStream( LISMAG2D_CalculateValues.input_profile_num +".jpg");
                ImageIO.write(buffer, "jpg", os);
                os.close();

                String path =
LISMAG2D_CalculateValues.input_profile_num  +".jpg";
                image = ImageIO.read(new File(path));

                Graphics g_image = LISMAG2D_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 =
LISMAG2D_MainPanel.img.getGraphics();
                            g_image.drawImage(image,
-60,-30,image.getWidth(), image.getHeight(),dg);
                      }
                };
                LISMAG2D_MainPanel.img.addMouseListener(ml3);
           }
           catch (Exception e2) {

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

```
package com.lismag2d.control;

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

```
import java.io.*;
import java.text.DecimalFormat;
import javax.imageio.ImageIO;
import javax.swing.*;

import com.lismag2d.model.LISMAG2D_CalculateValues;
import com.lismag2d.util.LISMAG2D_HandleException;
import com.lismag2d.view.LISMAG2D_MainPanel;
import com.lismag2d.view.LISMAG2D_TableVal;
import com.lismag2d.view.event.LISMAG2D_EditFault;
import com.lismag2d.view.event.LISMAG2D_PlotFault;
import com.lismag2d.view.event.LISMAG2D_StoreValues;
import com.lismag2d.view.graph.LISMAG2D_PlainGraph;

public class LISMAG2D_Controller implements ActionListener {

      Object rowdata [][] = {};
      Object rowdata1 [][] = null;
      BufferedImage image;
      com.lismag2d.view.LISMAG2D_DrawGraph dg = new
com.lismag2d.view.LISMAG2D_DrawGraph();
      com.lismag2d.view.LISMAG2D_Plot pd = new
com.lismag2d.view.LISMAG2D_Plot();
      com.lismag2d.model.LISMAG2D_CalculateValues cv = new
com.lismag2d.model.LISMAG2D_CalculateValues();
      LISMAG2D_StoreValues sv = new LISMAG2D_StoreValues();
      public static boolean success = false;

      public void actionPerformed(ActionEvent ae)  {

            if(ae.getActionCommand().equals("Compute/Edit fault plane")){

LISMAG2D_MainPanel.fl.removeMouseListener(LISMAG2D_EditFault.ml1);

LISMAG2D_MainPanel.fl.removeMouseMotionListener((MouseMotionListener)LISM
AG2D_EditFault.ml1);
                  sv.store();
                  LISMAG2D_EditFault ef = new LISMAG2D_EditFault();
                  try{
                        if (LISMAG2D_CalculateValues.len <=
LISMAG2D_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 LISMAG2D_HandleException();
                        }
                        Graphics g =
LISMAG2D_MainPanel.p_Center.getGraphics();
                        ef.paint(g);



                  }
                  catch(Exception e) {
                        e.printStackTrace();
                  }
            }
            if(ae.getActionCommand().equals("Draw fault plane")){
                  LISMAG2D_MainPanel.p_Center.removeAll();

LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.graphLabel);

LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.img4);
                  LISMAG2D_MainPanel.img4.setEditable(false);
                  LISMAG2D_MainPanel.p_Center.validate();
                  LISMAG2D_MainPanel.clearPanel(LISMAG2D_MainPanel.img4);
                  DecimalFormat d = new DecimalFormat("0.##");
                  DecimalFormat df = new DecimalFormat("0.###");
                  DecimalFormat df1 = new DecimalFormat("0.####");
                  try
                  {

cv.getAnamolyValues(com.lismag2d.view.LISMAG2D_MainPanel.captureValues())
;

                        if(LISMAG2D_CalculateValues.val_code<=0 ||
LISMAG2D_CalculateValues.val_code>3){

                              JFrame frame = null;
                              JOptionPane.showMessageDialog(frame,
                                          "Code value must be 1\n"
                                                      +"or 2 or 3",
                                                      "Out of bounds error",

JOptionPane.ERROR_MESSAGE);

                        }
                        cv.getCoefficients();
```

```
                        rowdata1 = new
Object[LISMAG2D_CalculateValues.i_no_obs + 1][2];
                        for (int K = 1;K <=
LISMAG2D_CalculateValues.i_no_obs; K++){

                              rowdata1[K][0]= "" +
d.format(LISMAG2D_CalculateValues.d_dis_km[K]);
                              rowdata1[K][1]= "" +
df.format(LISMAG2D_CalculateValues.obs_ano[K]);


                        }


com.lismag2d.view.LISMAG2D_TableVal.populateEastPanel(rowdata1);
                        LISMAG2D_TableVal.val.setText("");
                        LISMAG2D_TableVal.val.append("Coordinates of
control points(x,z):-\n");

LISMAG2D_TableVal.val.appendText("----------------------------------------
----\n");
                        for (int i = 1; i <= LISMAG2D_CalculateValues.len;
i++){


LISMAG2D_TableVal.val.appendText("("+d.format(LISMAG2D_CalculateValues.d_
x_km_arr[i])+","+d.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+")
\n") ;

                        }
                        LISMAG2D_TableVal.val.appendText("\n");
                        LISMAG2D_TableVal.val.append("Coefficients of the
polynomial:-\n");

LISMAG2D_TableVal.val.appendText("----------------------------------------
----\n");
                        LISMAG2D_TableVal.val.appendText("\n");

                        for (int i = 1; i <
LISMAG2D_CalculateValues.d_cftnt_arr.length; i++){
                              LISMAG2D_TableVal.val.appendText("f"+(i -
1)+" = " +df1.format(LISMAG2D_CalculateValues.d_cftnt_arr[i])+ "\n");
                        }
```

```
                    if(LISMAG2D_CalculateValues.val_code==1 ||
LISMAG2D_CalculateValues.val_code==2 || LISMAG2D_CalculateValues.val_code
== 3){
                              cv.cal();
                              int width = 1280;
                              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 ;
                              pd.plot(g2);
                              pd.plotXYCoordinates(g2);
                              pd.drawGraph(g2);
                              pd.plotZCoordinates(g2);
                              pd.plot(g2);

                              FileOutputStream os = new
FileOutputStream( LISMAG2D_CalculateValues.input_profile_num +".jpg");
                              ImageIO.write(buffer, "jpg", os);
                              os.close();

                              String path =
LISMAG2D_CalculateValues.input_profile_num +".jpg";
                              image = ImageIO.read(new File(path));

                              Graphics g_image =
LISMAG2D_MainPanel.img4.getGraphics();
                              g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(), dg);

                              MouseListener ml3 = new MouseAdapter(){
                                    public void mouseClicked(MouseEvent e){
                                          Graphics g_image =
LISMAG2D_MainPanel.img4.getGraphics();
                                          g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(),dg);
                                    }
                              };

LISMAG2D_MainPanel.img4.addMouseListener(ml3);

                    }
               }
               catch (Exception e2) {

                    e2.printStackTrace();
```

```
                }

           }

           else if(ae.getActionCommand().equals("Original fault co-
ordinates")){


LISMAG2D_MainPanel.fl.removeMouseListener(LISMAG2D_EditFault.ml1);

LISMAG2D_MainPanel.fl.removeMouseMotionListener((MouseMotionListener)LISM
AG2D_EditFault.ml1);
                LISMAG2D_EditFault ef = new LISMAG2D_EditFault();

                for (int x=1;x<=LISMAG2D_CalculateValues.len;x++){

                     if (LISMAG2D_StoreValues.values[x][1]!
=LISMAG2D_PlotFault.val[x])
                          LISMAG2D_PlotFault.val[x] =
LISMAG2D_StoreValues.values[x][1];
                     System.out.println(LISMAG2D_StoreValues.values[x]
[1]+"\t"+LISMAG2D_CalculateValues.d_x_km_arr[x]);
                     if (LISMAG2D_StoreValues.values[x][2]!
=LISMAG2D_PlotFault.val1[x])
                          LISMAG2D_PlotFault.val1[x] =
LISMAG2D_StoreValues.values[x][2];

                }
                try{
                     if (LISMAG2D_CalculateValues.len <=
LISMAG2D_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 LISMAG2D_HandleException();
                     }
                     Graphics g =
LISMAG2D_MainPanel.p_Center.getGraphics();
                     ef.paint(g);

                }
```

```
                catch(Exception e) {
                      e.printStackTrace();
                }
           }


           else if(ae.getActionCommand().equals("Load file")){

                LISMAG2D_MainPanel.loadData();
                LISMAG2D_MainPanel.p_Center.removeAll();

LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.graphLabel);
                LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.im);
                LISMAG2D_PlainGraph pg = new LISMAG2D_PlainGraph();
                try{
                      if (LISMAG2D_CalculateValues.len <=
LISMAG2D_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 LISMAG2D_HandleException();
                      }


cv.getAnamolyValues(com.lismag2d.view.LISMAG2D_MainPanel.captureValues())
;
                      cv.getCoefficients();

                      Graphics2D g1 =
(Graphics2D)LISMAG2D_MainPanel.im.getGraphics();
                      pg.drawPlainGraph(g1);
                      dg.plotZCoordinates(g1);
                      pg.drawPlainGraph(g1);

LISMAG2D_MainPanel.im.removeMouseListener(LISMAG2D_PlotFault.ml);

LISMAG2D_MainPanel.im.removeMouseMotionListener((MouseMotionListener)LISM
AG2D_PlotFault.ml);

                }
                catch(Exception e) {
```

```
                    e.printStackTrace();
               }

          }

          else if(ae.getActionCommand().equals("Save and Print")){

               try{
                    String current = System.getProperty("user.dir");
                    File img_file = new
File(LISMAG2D_CalculateValues.input_profile_num+".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()));
                              myWriter = new
FileWriter(OutFile+".html");
                              myWriter.write("<html> <Body onLoad =
\"window.print()\"><table> <tr> <td>" +"<td> <img src = '"+
LISMAG2D_CalculateValues.input_profile_num +".jpg'></td>"+
                                             "<table border = 1> <tr> <th
colspan = 4>LOCATION:- "+LISMAG2D_CalculateValues.input_profile_num+"</
th> </tr>");

                              DecimalFormat df =new
DecimalFormat("0.###");
                              DecimalFormat d = new
DecimalFormat("0.##");

                              myWriter.write("<tr > <th>Distance </
th><th> Observed anamolies (nT) </th><th> Calculated anamolies (nT) </th>
</tr>");

                              for ( int K = 1; K <=
LISMAG2D_CalculateValues.i_no_obs; K++){
```

```
                                        myWriter.write("<tr> <td>" +
d.format(LISMAG2D_CalculateValues.x[K])+"</td>
<td>"+df.format(LISMAG2D_CalculateValues.obs_ano[K])+"</
td><td>"+df.format(LISMAG2D_CalculateValues.ano[K])+"</td></tr>");
                                    }
                                    myWriter.write(" </table> </td> </tr></
table><BR>Coordinates of control points(x,z): <BR>");

myWriter.write("---------------------------------- <BR>");

                                    DecimalFormat d1 =new
DecimalFormat("0.####");

                                    for (int i = 1; i <=
LISMAG2D_CalculateValues.len; i++){


myWriter.write("("+d.format(LISMAG2D_CalculateValues.d_x_km_arr[i])
+","+d.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+")"+"<BR>") ;

                                    }
                                    myWriter.write("<BR>");
                                    myWriter.write("Coefficients of the
polynomial::"+"<BR>");

myWriter.write("-----------------------------------------<BR>");
                                    for ( int i = 1; i <
LISMAG2D_CalculateValues.cftnt .length; i++) {

                                         myWriter.write("f"+ ( i - 1 ) +" =
"+d1.format(LISMAG2D_CalculateValues.cftnt[i])+"<BR>");
                                    }

                                    myWriter.close();
                               }
                          }
                          else
                          {
                               //pops up error message

                          }

                     }
                     catch(Exception e1) {

                          e1.printStackTrace();
                     }
               }
```

```
            else if(ae.getActionCommand().equals("Save file")){

                  try{

                        String current = System.getProperty("user.dir");
                        JFileChooser saveFile = new JFileChooser(current);
                        File OutFile = saveFile.getSelectedFile();
                        DecimalFormat d = new DecimalFormat("0.##");
                        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(""+LISMAG2D_CalculateValues.input_profile_num);

myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.i_no_obs);

myWriter.append(System.getProperty("line.separator"));
                                    for (int i = 1; i <=
LISMAG2D_CalculateValues.i_no_obs; i++){

                                          myWriter.write(""+
(LISMAG2D_CalculateValues.d_dis_km[i])+",") ;

                                    }

myWriter.append(System.getProperty("line.separator"));
                                    for (int i = 1; i <=
LISMAG2D_CalculateValues.i_no_obs; i++){

                                          myWriter.write(""+
(LISMAG2D_CalculateValues.obs_ano[i])+",") ;

                                    }

myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.d_min_dep);
```

```
myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.d_max_dep);

myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.i_d_poly);

myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.val_str);

myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.val_iom);

myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.val_dom);

myWriter.append(System.getProperty("line.separator"));

myWriter.write(""+LISMAG2D_CalculateValues.val_code);


myWriter.append(System.getProperty("line.separator"));
                                for (int i = 1; i <=
LISMAG2D_CalculateValues.len; i++){


myWriter.write(""+d.format(LISMAG2D_CalculateValues.d_x_km_arr[i])+",") ;

                                }

myWriter.append(System.getProperty("line.separator"));

                                for (int i = 1; i <=
LISMAG2D_CalculateValues.len; i++){


myWriter.write(""+d.format(LISMAG2D_CalculateValues.d_z_km_arr[i])+",") ;

                                }

myWriter.append(System.getProperty("line.separator"));
```

```
myWriter.write(""+LISMAG2D_CalculateValues.len);

                                    myWriter.close();
                               }
                          }
                          else
                          {
                               //pops up error message

                          }

                     }
                     catch(Exception e1) {

                          e1.printStackTrace();
                     }
               }

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

                     LISMAG2D_MainPanel.clearDefaultValues();
                     LISMAG2D_MainPanel.p_Center.removeAll();

com.lismag2d.view.LISMAG2D_TableView.populateEastPanel(rowdata);
                     com.lismag2d.view.LISMAG2D_TableView.val.setText("");

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

                     JFrame frame = null;
                     int r =    JOptionPane.showConfirmDialog(
                                frame,
                                "Exit LISMAG2D ?",
                                "Confirm Exit ",
                                JOptionPane.YES_NO_OPTION);
                     if(r == JOptionPane.YES_OPTION ){
                          if(success==false){
                                String fileName =
LISMAG2D_CalculateValues.input_profile_num+".jpg";
                                File f = new File(fileName);
                                f.delete();
                          }
                          System.exit(0);
                     }
               }else if(ae.getActionCommand().equals("Graph")){


                     LISMAG2D_MainPanel.p_Center.removeAll();
```

```
LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.graphLabel);

LISMAG2D_MainPanel.p_Center.add(LISMAG2D_MainPanel.img4);
                LISMAG2D_MainPanel.img4.setEditable(false);
                LISMAG2D_MainPanel.p_Center.validate();
                LISMAG2D_MainPanel.clearPanel(LISMAG2D_MainPanel.img4);
                try
                {

cv.getAnamolyValues(com.lismag2d.view.LISMAG2D_MainPanel.captureValues())
;

                    if(LISMAG2D_CalculateValues.val_code<=0 ||
LISMAG2D_CalculateValues.val_code>3){

                        JFrame frame = null;
                        JOptionPane.showMessageDialog(frame,
                                  "Code value must be 1\n"
                                            +"or 2 or 3",
                                            "Out of bounds error",

JOptionPane.ERROR_MESSAGE);

                    }
                    if(LISMAG2D_CalculateValues.val_code==1 ||
LISMAG2D_CalculateValues.val_code==2 || LISMAG2D_CalculateValues.val_code
== 3){
                        cv.cal();
                        int width = 1280;
                        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);
                        dg.plotZCoordinates(g2);
                        dg.plot(g2);

                        FileOutputStream os = new
FileOutputStream( LISMAG2D_CalculateValues.input_profile_num +".jpg");
                        ImageIO.write(buffer, "jpg", os);
                        os.close();
```

```
                              String path =
LISMAG2D_CalculateValues.input_profile_num +".jpg";
                              image = ImageIO.read(new File(path));

                              Graphics g_image =
LISMAG2D_MainPanel.img4.getGraphics();
                              g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(), dg);

                              MouseListener ml3 = new MouseAdapter(){
                                    public void mouseClicked(MouseEvent e){
                                          Graphics g_image =
LISMAG2D_MainPanel.img4.getGraphics();
                                          g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(),dg);
                                    }
                              };

LISMAG2D_MainPanel.img4.addMouseListener(ml3);

com.lismag2d.view.LISMAG2D_TableView.populateEastPanel(LISMAG2D_Calculate
Values.obj);
                        }
                  }
                  catch (Exception e2) {

                        e2.printStackTrace();
                  }



            }else if(ae.getActionCommand().equals("Specify fault
coordinates")){
                  LISMAG2D_MainPanel.p_Center.removeAll();

                  DecimalFormat d = new DecimalFormat("0.##");
                  DecimalFormat df = new DecimalFormat("0.###");

                  cv.getAnamolyValues(LISMAG2D_MainPanel.captureValues());
                  rowdata1 = new Object[LISMAG2D_CalculateValues.i_no_obs
+ 1][2];
                  for (int K = 1;K <= LISMAG2D_CalculateValues.i_no_obs;
K++){

                        rowdata1[K][0]= "" +
d.format(LISMAG2D_CalculateValues.d_dis_km[K]);
                        rowdata1[K][1]= "" +
df.format(LISMAG2D_CalculateValues.obs_ano[K]);
```

```
                }


com.lismag2d.view.LISMAG2D_TableVal.populateEastPanel(rowdata1);
                com.lismag2d.view.LISMAG2D_TableVal.val.setText("");

                if(LISMAG2D_CalculateValues.d_dis_km.length -
LISMAG2D_CalculateValues.i_no_obs != 1){

                     JFrame frame = null;
                     JOptionPane.showMessageDialog(frame,
                               "Distance values must be equal to\n"
                                         +"number of observations.",
                                         "Out of bounds error",
                                         JOptionPane.ERROR_MESSAGE);
                     try {

                          throw new LISMAG2D_HandleException();

                     } catch (LISMAG2D_HandleException e) {

                     }
                }


LISMAG2D_MainPanel.im.removeMouseListener(LISMAG2D_PlotFault.ml);

LISMAG2D_MainPanel.im.removeMouseMotionListener((MouseMotionListener)LISM
AG2D_PlotFault.ml);
                LISMAG2D_PlotFault pf = new LISMAG2D_PlotFault();
                Graphics g = LISMAG2D_MainPanel.p_Center.getGraphics();
                pf.paint(g);

           }
      }

}
```

```
package com.lismag2d.util;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
```

```
public class LISMAG2D_Utility {

	public static double convertDouble(String str) throws Exception {


		Double temp = null;

		try {
			temp = new Double(str.trim());

		}
		catch(Exception e){

			JFrame frame = null;
			JOptionPane.showMessageDialog(frame,
					"Enter a numerical value.",
					"Number format error",
					JOptionPane.ERROR_MESSAGE);


		}
		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 = null;
		try {
			temp = new Integer(str.trim());
		}
		catch(Exception e){

			JFrame frame = null;
			JOptionPane.showMessageDialog(frame,
					"Enter a numerical value.",
					"Number format error",
					JOptionPane.ERROR_MESSAGE);

		}
		return temp.intValue();
```

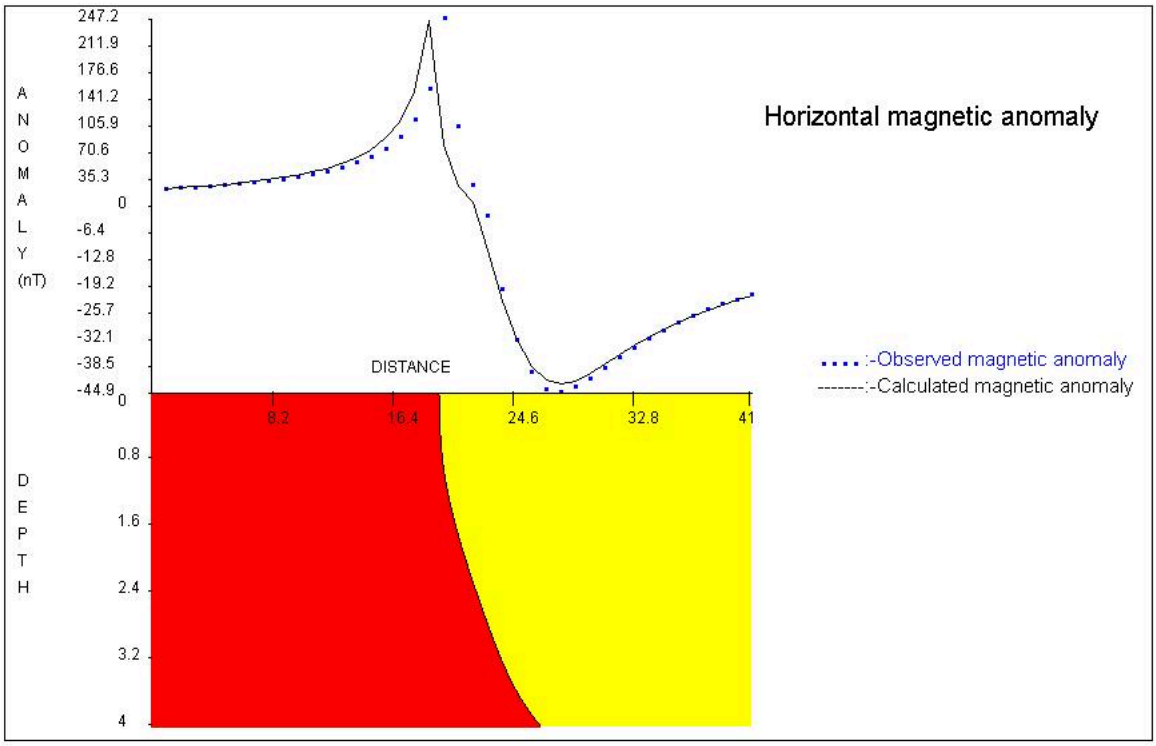
```
	}

	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 double findMinimumNumber( double observe[], double
denVal) {

		double max = denVal;
		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 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 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 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 {

                      try {
                            d_array[i] = convertDouble( arr.get(i -
1).toString() );
                      }
                      catch(Exception e){
                            JFrame frame = null;
                            JOptionPane.showMessageDialog(frame,
                                        "Enter numerical values.",
                                        "Number format error",
                                        JOptionPane.ERROR_MESSAGE);
                            throw new LISMAG2D_HandleException();

                      }
                }
          }
          return d_array;
     }
}
```

```
package com.lismag2d.util;

import java.awt.*;

import com.lismag2d.view.LISMAG2D_MainPanel;

public class LISMAG2D_HandleException extends Exception{

     public LISMAG2D_HandleException(){
          Graphics g = LISMAG2D_MainPanel.p_Center.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);
     }
}
```

```
abc
41
1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0
,19.0,20.0,21.0,22.0,23.0,24.0,25.0,26.0,27.0,28.0,29.0,30.0,31.0,32.0,33.0,34.0
,35.0,36.0,37.0,38.0,39.0,40.0,41.0,
21.103933428121998,22.17256839508961,23.354434320855674,24.66844379422022,26.137
956519740868,27.79216650246589,29.66804337547583,31.81310520419792,34.2894716327
5696,37.17994785255616,40.59744212384507,44.700079614978826,49.71652696726892,55
.99071357886667,64.06615600274834,74.85897567524766,90.05569002237151,113.190049
01867628,153.42129984532798,247.17643482252205,104.5041696731591,27.178336190066
904,-2.5700352067367715,-20.344137944684082,-32.420255416252736,-40.230330689981
71,-44.13245778632276,-44.90432416094882,-43.7101911700241,-41.552392181018924,-
39.058035507988784,-36.552481554011415,-34.18255042132575,-32.002902146455234,-3
0.024164443486697,-28.23741205046394,-26.62608667228895,-25.171614277029892,-23.
85593080300154,-22.662511237387005,-21.57668684726927,
4.0
3
40.0
100.0
30.0
2
19.68,20.14,22.14,24.42,26.6,
0,0.96,2.18,3.25,4,
5
```



LOCATION:- abc		
Distance	**Observed anamolies (nT)**	**Calculated anamolies (nT)**
1	21.104	22.071
2	22.173	23.216
3	23.354	24.487
4	24.668	25.904
5	26.138	27.496
6	27.792	29.295
7	29.668	31.347
8	31.813	33.707
9	34.289	36.452
10	37.18	39.686
11	40.597	43.551
12	44.7	48.257
13	49.717	54.117
14	55.991	61.624
15	64.066	71.617
16	74.859	85.645
17	90.056	106.999
18	113.19	144.404
19	153.421	234.896
20	247.176	98.904
21	104.504	34.724
22	27.178	10.857
23	-2.57	-4.96
24	-20.344	-17.375
25	-32.42	-27.502
26	-40.23	-35.243
27	-44.132	-40.155
28	-44.904	-42.22

29	-43.71	-42.083
30	-41.552	-40.637
31	-39.058	-38.583
32	-36.552	-36.346
33	-34.183	-34.141
34	-32.003	-32.065
35	-30.024	-30.153
36	-28.237	-28.409
37	-26.626	-26.827
38	-25.172	-25.391
39	-23.856	-24.088
40	-22.663	-22.902
41	-21.577	-21.821

Coordinates of control points(x,z):

(19.68,0)
(20.14,0.96)
(22.14,2.18)
(24.42,3.25)
(26.6,4)

Coefficients of the polynomial::
---
f0 = 19.6611
f1 = 0.0757
f2 = 0.5332
f3 = -0.03

```
package com.invmglstrk.view;

import java.awt.Frame;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import com.invmglstrk.control.INVMGLSTRK_Controller;
import com.invmglstrk.model.INVMGLSTRK_CalculateValues;
import com.invmglstrk.view.INVMGLSTRK_MainPanel;
import com.invmglstrk.view.INVMGLSTRK_MainView;

public class INVMGLSTRK_MainView extends Frame{

	/**
	 Inversion of Magnetic Anomalies due to Listric Fault Morphology
	 *
	 */


	public static void main(String s[])
	{
		INVMGLSTRK_MainView cm = new INVMGLSTRK_MainView();
		cm.setSize(1280, 768);
		cm.addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent e){
				JFrame frame = null;
				int r = JOptionPane.showConfirmDialog(
							frame,
							"Exit INVMGLSTRK ?",
							"Confirm Exit ",
							JOptionPane.YES_NO_OPTION);
				if(r == JOptionPane.YES_OPTION ){
					if(INVMGLSTRK_Controller.success == false){
							String fileName =
INVMGLSTRK_CalculateValues.input_profile_num+".jpg";
							File f = new File(fileName);
							f.delete();
					}
					System.exit(0);
				}
			}
		});
		cm.setTitle("INVMGLSTRK");
		cm.setResizable(false);
```

```
            cm.add(new INVMGLSTRK_MainPanel(cm));
            cm.setVisible(true);

      }

}
```

```
package com.invmglstrk.view;

import java.awt.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.HashMap;
import javax.swing.JFileChooser;
import com.invmglstrk.control.INVMGLSTRK_Controller;

public class INVMGLSTRK_MainPanel extends Panel {


      public static TextArea img = new TextArea(46,140);
      Panel p_North, p_West,p_South;
      public static Panel p_East;
      public static Label graphLabel;
      public static Panel p_Center;
      public static TextField inputValues [] = new TextField[18];

      Button actionButton[] = new Button[12];
      Object rowdata[][]={};

      /**Number of the Profile*/
      public final static int NUM_PROFILE = 0;
      /**Number of observation*/
      public static final int N_OBS = 1 ;
      /**Distance(km)*/
      public static final int DIS_KM = 2;
      /**Observed anomaly*/
      public static final int OBS_ANO = 3;
      /** Degree of polynomial*/
      public static final int D_POLY = 4;
      /**Value Of Strike*/
      public static final int VAL_STR = 5;
      /**Value Of Code*/
      public static final int VAL_CODE = 6;
      /**Number of iteration */
      public static final int NUM_ITER = 7;
```

```
	public INVMGLSTRK_MainPanel(INVMGLSTRK_MainView cm){

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

		graphLabel = new Label("Inversion of Magnetic Anomalies due to
Listric Fault Morphology", Label.CENTER);
		graphLabel.setFont(new Font("Arial", 40, 20));
		p_Center.add(graphLabel);

		for(int i = 0; i < 8; i++){
			inputValues[i] = new TextField();
		}

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

		this.populateNorthPanel();
		INVMGLSTRK_TableView.populateEastPanel(rowdata);

		this.add(p_North, BorderLayout.NORTH);
		p_Center.setSize(1100, 1000);
		this.add(p_Center, BorderLayout.CENTER);
		p_Center.add(img);
		this.add(p_East, BorderLayout.EAST);
		this.setVisible(true);
	}

	public void populateNorthPanel(){
		p_North.setLayout(new GridLayout(4,4));

		p_North.add(new Label("Profile ID"));
		p_North.add(inputValues[0]);

		p_North.add(new Label("Number of observations"));
		p_North.add(inputValues[1]);

		p_North.add(new Label("Distance"));
		p_North.add(inputValues[2]);
```

```
            p_North.add(new Label("Observed anomalies(nT)"));
            p_North.add(inputValues[3]);

            p_North.add(new Label("Degree of polynomial"));
            p_North.add(inputValues[4]);

            p_North.add(new Label("Strike(Degrees)"));
            p_North.add(inputValues[5]);

            p_North.add(new Label("Code number"));
            p_North.add(inputValues[6]);

            p_North.add(new Label("Number of iterations"));
            p_North.add(inputValues[7]);
            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]);

            actionButton[0].addActionListener(new
INVMGLSTRK_Controller());
            actionButton[1].addActionListener(new
INVMGLSTRK_Controller());
            actionButton[2].addActionListener(new
INVMGLSTRK_Controller());
            actionButton[3].addActionListener(new
INVMGLSTRK_Controller());
            actionButton[4].addActionListener(new
INVMGLSTRK_Controller());
      }


      public static HashMap captureValues(){

            HashMap h_Map = new HashMap();
            try {
                  h_Map.put("NUM_PROFILE",
inputValues[NUM_PROFILE].getText());
                  h_Map.put("N_OBS", inputValues[N_OBS].getText());
                  h_Map.put("D_POLY", inputValues[D_POLY].getText());
                  h_Map.put("DIS_KM", inputValues[DIS_KM].getText());
                  h_Map.put("NUM_ITER", inputValues[NUM_ITER].getText());
                  h_Map.put("OBS_ANO", inputValues[OBS_ANO].getText());
                  h_Map.put("VAL_STR", inputValues[VAL_STR].getText());
```

```
                h_Map.put("VAL_CODE", inputValues[VAL_CODE].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, 1000, 600);
     }

     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){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.NUM_PROFILE].setTex
t(""+st);

                                  }
                                  if (count==2){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.N_OBS].setText(""+s
t);
```

```
                                    }

                                    if (count==3){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.DIS_KM].setText(""+
st);

                                    }
                                    if (count==4){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.OBS_ANO].setText(""
+st);

                                    }
                                    if (count==5){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.D_POLY].setText(""+
st);

                                    }

                                    if (count==6){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.VAL_STR].setText(""
+st);

                                    }

                                    if (count==7){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.VAL_CODE].setText("
"+st);

                                    }

                                    if (count==8){


INVMGLSTRK_MainPanel.inputValues[INVMGLSTRK_MainPanel.NUM_ITER].setText("
"+st);

                                    }
```

```
                        }
                        catch(Exception e) {

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

                  }
                  br.close();
            }
      }
      catch(Exception e){

      }
}

public static void clearDefaultValues(){

      inputValues[NUM_PROFILE].setText("");
      inputValues[N_OBS].setText("");
      inputValues[D_POLY].setText("");
      inputValues[DIS_KM].setText("");
      inputValues[OBS_ANO].setText("");
      inputValues[VAL_STR].setText("");
      inputValues[NUM_ITER].setText("");
      inputValues[VAL_CODE].setText("");

}
}
```

```
package com.invmglstrk.view;

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


public class INVMGLSTRK_TableView extends Panel{

      public static TextArea val = new TextArea(5,5);
      public static  void populateEastPanel(Object rowData[][]) {

            com.invmglstrk.view.INVMGLSTRK_MainPanel.p_East.removeAll();
```

```
            com.invmglstrk.view.INVMGLSTRK_MainPanel.p_East.setLayout(new
GridLayout(2,1));

            Object columnNames[] = {"Distance","Observed anomalies
(nT)","Calculated anomalies (nT)"};
            JTable table = new JTable(rowData, columnNames);
            table.setPreferredScrollableViewportSize(new
Dimension(300,500));
            JScrollPane scrollPane = new JScrollPane(table);
            scrollPane.setAutoscrolls(true);

com.invmglstrk.view.INVMGLSTRK_MainPanel.p_East.add(scrollPane);
            val.setEditable(false);
            com.invmglstrk.view.INVMGLSTRK_MainPanel.p_East.add(val);
            com.invmglstrk.view.INVMGLSTRK_MainPanel.p_East.validate();

com.invmglstrk.view.INVMGLSTRK_MainPanel.p_East.setVisible(true);
      }
}
```

```
package com.invmglstrk.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.invmglstrk.model.INVMGLSTRK_CalculateValues;
import com.invmglstrk.util.INVMGLSTRK_Utility;


public class INVMGLSTRK_DrawGraph extends Applet{

      public static int i_no_obs;
      float maxY,maxZ,maxX ;
      double obs[];

      public void drawGraph(Graphics2D g2) {

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

```
            g2.setColor(Color.BLACK);
            g2.draw(new Line2D.Float(200, 20, 200, 300));
            g2.drawLine(90,10 ,1040,10);
            g2.drawLine(90, 560, 1040, 560);
            g2.drawLine(1040, 10, 1040, 560);
            g2.drawLine(90, 10, 90, 560);

            String []a = {"A", "N", "O", "M", "A", "L", "Y", "(nT)"};
            String []b = {"D", "E", "P", "T", "H"};
            for (int i = 0; i < a.length; i++) {
                  g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
            }
            for (int i = 0; i < b.length; i++) {
                  g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
            }

      }

      public void plot(Graphics2D g2) {

            g2.setFont( new Font("Arial", 12, 12) );
            DecimalFormat f = new DecimalFormat("0.#");
            g2.setColor(Color.BLACK);

            i_no_obs = INVMGLSTRK_CalculateValues.i_no_obs;
            obs = new double[i_no_obs+1];
            for (int i = 1; i <= i_no_obs; i++) {
                  obs[i] = INVMGLSTRK_CalculateValues.d_dis_km[i];
            }

            maxX = (float)obs[i_no_obs];
            float maxy = (float)
INVMGLSTRK_Utility.findMaximumNumber(INVMGLSTRK_CalculateValues.obs_ano);
            float maxy1 = (float)
INVMGLSTRK_Utility.findMaximumNumber(INVMGLSTRK_CalculateValues.ano);
            if(maxy > maxy1)
                  maxY = maxy;
            else
                  maxY = maxy1;
            maxZ = (float)INVMGLSTRK_CalculateValues.cftnt[2];

            g2.drawString("|",(float) 600-3+50  , 308);

g2.drawString(""+f.format(INVMGLSTRK_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);
            g2.drawString("0", 125+50, 310);
            g2.drawString("DISTANCE", 315+50, 285);
            float xplot = 0;
```

```
            float xInterval = (float)
(INVMGLSTRK_CalculateValues.d_dis_km[i_no_obs] / 5);

            int zInterval = 50;
            for (float x = xInterval, j = 1; x < 600; x += xInterval){
                  xplot = xplot + xInterval;
                  if(j > 4)
                        break;
                  g2.drawString("|",(float) (200 + (450 * x / maxX) ),
308);
                  g2.drawString("" + f.format(xplot), (float) (150 + (450
* x / maxX) ) - 3+50, 323);
                  j++;
            }

            DecimalFormat d =  new DecimalFormat("0.#");
            float points1 = maxZ / 5 ;
            for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                  g2.drawString("-", 148+50, 52 + x);
                  g2.drawString("" +d.format(points1 * j), 125+50, 50 +
x);
                  j++;
            }
      }




      public void plotXYCoordinates(Graphics2D g2){

            double minAno =
INVMGLSTRK_Utility.findMinimumNumber1(INVMGLSTRK_CalculateValues.ano);
            double maxAno =
INVMGLSTRK_Utility.findMaximumNumber(INVMGLSTRK_CalculateValues.ano,
minAno);

            double minAno1 =
INVMGLSTRK_Utility.findMinimumNumber1(INVMGLSTRK_CalculateValues.obs_ano)
;
            double maxAno2 =
INVMGLSTRK_Utility.findMaximumNumber(INVMGLSTRK_CalculateValues.obs_ano,
minAno);

            if (minAno < 0 &&  maxAno <= 0 || minAno1 < 0 &&  maxAno2 <=
0){
                  plotXYCoordinates1(g2);
            }
            if (minAno >= 0 && maxAno > 0 || minAno1 >= 0 && maxAno2 > 0){
```

```
                plotXYCoordinates1(g2);
            }
            if (minAno < 0  && maxAno>0 || minAno1 < 0  && maxAno2>0){
                plotXYCoordinates2(g2);
            }



      }

      public void plotXYCoordinates1 (Graphics2D g2) {

            g2.setColor(Color.white);
            g2.drawLine(200, 20, 200, 50);
            g2.setFont(new Font("Arial", 20, 12));
            g2.setColor(Color.black);
            float maxval1 = (float)
INVMGLSTRK_Utility.findMaximumNumber(INVMGLSTRK_CalculateValues.obs_ano);
            float maxval = (float)
INVMGLSTRK_Utility.findMaximumNumber(INVMGLSTRK_CalculateValues.ano);
            if(maxval>maxval1)
                maxY = maxval;
            else
                maxY = maxval1;
            int points = (int)maxY / 5;
            int yInterval = 50;
            g2.drawString("0", 125+50, 50);
            for (int x = yInterval, j = 1; x < 250; x += yInterval){

                g2.drawString("-", 148+50, 50 + x);
                g2.drawString("" + (points*j), 145, 50 + x);
                j++;
            }
            float prevx = (float) (200 +( 450 * obs[1] / maxX));;
            float prevy = (float)( 50 + ( 250 *
INVMGLSTRK_CalculateValues.ano[1] / maxY ) );
            float xpoint = 0;
            float ypoint = 0;
            float ypoint1 = 0;

            for (int k = 1; k <= i_no_obs; k++) {

                xpoint = (float)( 450 * obs[k] / maxX);
                ypoint = (float)( ( 250 *
INVMGLSTRK_CalculateValues.ano[k] / maxY ) );
                ypoint1 = (float)( ( 250 *
INVMGLSTRK_CalculateValues.obs_ano[k] / maxY ) );
```

```
                g2.setColor(Color.BLUE);
                g2.setFont(new Font("Arial", 20, 30));
                g2.drawString(".", 200 + xpoint-4, 50+ypoint1+1);

                g2.setColor(Color.BLACK);
                g2.draw(new Line2D.Float(prevx, prevy, 200+ xpoint, 50 +
ypoint ));

                g2.setFont(new Font("Arial", 20, 12));
                g2.setColor(Color.black);
                prevx = 200 + xpoint;
                prevy = 50 + ypoint ;

           }

     }


     public void plotXYCoordinates2 (Graphics2D g2) {


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

           double store[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];
           double store1[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];
           double negstore[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];
           double negstore1[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];

           double store2[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];
           double store3[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];
           double negstore2[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];
           double negstore3[] = new
double[INVMGLSTRK_CalculateValues.i_no_obs+1];
           for(int i = 1; i <= INVMGLSTRK_CalculateValues.i_no_obs; i++){
                if(INVMGLSTRK_CalculateValues.ano[i]>0)
                     store[i] = INVMGLSTRK_CalculateValues.ano[i];
                else
                     negstore[i] = INVMGLSTRK_CalculateValues.ano[i];
                if(INVMGLSTRK_CalculateValues.ano[i]>0)
                     store1[i] = INVMGLSTRK_CalculateValues.ano[i];
```

```
                else
                      negstore1[i] = INVMGLSTRK_CalculateValues.ano[i];


                if(INVMGLSTRK_CalculateValues.obs_ano[i]>0)
                      store2[i] = INVMGLSTRK_CalculateValues.obs_ano[i];
                else
                      negstore2[i] =
INVMGLSTRK_CalculateValues.obs_ano[i];
                if(INVMGLSTRK_CalculateValues.obs_ano[i]>0)
                      store3[i] = INVMGLSTRK_CalculateValues.obs_ano[i];
                else
                      negstore3[i] =
INVMGLSTRK_CalculateValues.obs_ano[i];

           }
           float maxpos = (float)
INVMGLSTRK_Utility.findMaximumNumber1(store);
           float maxpos1 = (float)
INVMGLSTRK_Utility.findMaximumNumber1(store1);
           float maxneg = (float)
INVMGLSTRK_Utility.findMaximumNumber1(negstore);
           float maxneg1 = (float)
INVMGLSTRK_Utility.findMaximumNumber1(negstore1);

           float maxpos2 = (float)
INVMGLSTRK_Utility.findMaximumNumber1(store2);
           float maxpos3 = (float)
INVMGLSTRK_Utility.findMaximumNumber1(store3);
           float maxneg2 = (float)
INVMGLSTRK_Utility.findMaximumNumber1(negstore2);
           float maxneg3 = (float)
INVMGLSTRK_Utility.findMaximumNumber1(negstore3);

           float posnum =0;
           float posnum1 =0;

           float yval =0;
           float yval1 =0;
           float yval2 =0;
           float yval3 =0;

           float posval=0;


           if(maxpos>maxpos1)
                 posnum = Math.abs(maxpos);
           else
```

```
                posnum = Math.abs(maxpos1);

            if(maxpos2>maxpos3)
                posnum1 = Math.abs(maxpos2);
            else
                posnum1 = Math.abs(maxpos3);

            if(posnum>posnum1)
                posval = posnum;
            else
                posval = posnum1;


            if(Math.abs(maxneg)>Math.abs(maxneg1))

                yval = maxneg;
            if(Math.abs(maxneg2)>Math.abs(maxneg3))

                yval1 = maxneg2;
            if(Math.abs(yval)>Math.abs(yval1))

                maxY = yval;
            else
                maxY = yval1;


            if(Math.abs(maxneg)<Math.abs(maxneg1))

                yval2 = maxneg;
            if(Math.abs(maxneg2)<Math.abs(maxneg3))

                yval3 = maxneg3;
            if(Math.abs(yval2)>Math.abs(yval3))

                maxY = yval2;
            else
                maxY = yval3;


            if(Math.abs(maxY)>10){

                float prevx = (float) (200 +( 450 * obs[1] / maxX));
                float prevy = 0;
                if(INVMGLSTRK_CalculateValues.ano[1]>0)
                     prevy = 160-(float)( ( 140 *
INVMGLSTRK_CalculateValues.ano[1] / posval ) );
                else
```

```
                    prevy = 160+(float)( ( 140 *
INVMGLSTRK_CalculateValues.ano[1] / maxY ) );


                float xpoint = 0;
                float ypoint = 0;
                float ypoint1 = 0;

                g2.drawString("0",125+50,162);
                g2.drawString("-", 148+50, 164 );


                DecimalFormat f = new DecimalFormat("0.#");
                float points = maxY / 7;
                float points1 = posnum / 7;
                int yInterval=20;
                for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                     g2.drawString("-", 148+50, 164 - x );
                     g2.drawString("" + f.format(points1 * j), 145, 162
- x );
                     j++;
                }
                for (int x = yInterval, j = 1; x <= 140; x+=yInterval){

                     g2.drawString("-", 148+50, 164 + x );
                     g2.drawString("" + f.format(points * j), 145, 162 +
x );
                     j++;
                }
                for (int k = 1; k <= i_no_obs; k++) {

                     xpoint = (float)( 450 * obs[k] / maxX);
                     if(INVMGLSTRK_CalculateValues.ano[k]>0)
                          ypoint = 160-(float)( ( 140 *
INVMGLSTRK_CalculateValues.ano[k] / posval ) );
                     else
                          ypoint = 160+(float)( ( 140 *
INVMGLSTRK_CalculateValues.ano[k] / maxY ) );

                     if(INVMGLSTRK_CalculateValues.obs_ano[k]>0)
                          ypoint1 = 160-(float)( ( 140 *
INVMGLSTRK_CalculateValues.obs_ano[k] / posval ) );
                     else
                          ypoint1 = 160+(float)( ( 140 *
INVMGLSTRK_CalculateValues.obs_ano[k] / maxY ) );
```

```
                    g2.setColor(Color.BLUE);
                    g2.setFont(new Font("Arial", 20, 30));
                    g2.drawString(".", 200 + xpoint-4, ypoint1+1);
                    g2.setColor(Color.BLACK);
                    g2.draw(new Line2D.Float(prevx, prevy, 200 +
xpoint,  ypoint  ));
                    g2.setFont(new Font("Arial", 20, 12));
                    g2.setColor(Color.black);
                    prevx = 200 + xpoint;
                    prevy = ypoint ;

               }
          }

          else{
               maxY = -100;
               float prevx = (float) (200 +( 450 * obs[1] / maxX));
               float prevy = 0;

               if(INVMGLSTRK_CalculateValues.ano[1]>0)
                    prevy = 260-(float)( ( 240 *
INVMGLSTRK_CalculateValues.ano[1] / posval ) );
               else
                    prevy = 260+(float)( ( 40 *
INVMGLSTRK_CalculateValues.ano[1] / maxY ) );

               float xpoint = 0;
               float ypoint = 0;
               float ypoint1 = 0;


               g2.drawString("0",125+50,262);
               g2.drawString("-", 148+50, 264 );
               g2.drawString("-100",145,300);

               DecimalFormat f = new DecimalFormat("0.#");
               float points1 = posnum / 8;
               int yInterval=30;
               for (int x = yInterval, j = 1; x <= 240; x+=yInterval){

                    g2.drawString("-", 148+50, 264 - x );
                    g2.drawString("" + f.format(points1 * j), 145, 262
- x );
                    j++;
               }

               for (int k = 1; k <= i_no_obs; k++) {
```

```
                    xpoint = (float)( 450 * obs[k] / maxX);
                    if(INVMGLSTRK_CalculateValues.ano[k]>0)
                         ypoint = 260-(float)( ( 240 *
INVMGLSTRK_CalculateValues.ano[k] / posval ) );
                    else
                         ypoint = 260+(float)( ( 40 *
INVMGLSTRK_CalculateValues.ano[k] / maxY ) );


                    if(INVMGLSTRK_CalculateValues.obs_ano[k]>0)
                         ypoint1 = 260-(float)( ( 240 *
INVMGLSTRK_CalculateValues.obs_ano[k] / posval ) );
                    else
                         ypoint1 = 260+(float)( ( 40 *
INVMGLSTRK_CalculateValues.obs_ano[k] / maxY ) );

                    g2.setColor(Color.BLUE);
                    g2.setFont(new Font("Arial", 20, 30));
                    g2.drawString(".", 200 + xpoint-4, ypoint1+1);
                    g2.setColor(Color.BLACK);
                    g2.draw(new Line2D.Float(prevx, prevy, 200 +
xpoint,  ypoint));
                    g2.setFont(new Font("Arial", 20, 12));
                    g2.setColor(Color.black);
                    prevx = 200 + xpoint;
                    prevy = ypoint ;

               }
          }
     }


     public void plotZCoordinates (Graphics2D g2) {

          DecimalFormat d =  new DecimalFormat("0.#");
          g2.drawString("|",(float) 600-3+50  , 308);

g2.drawString(""+d.format(INVMGLSTRK_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);
          g2.drawString("0", 125+50, 310);
          g2.drawString("DISTANCE", 315+50, 285);
          int zInterval = 50;
          float xplot = 0;
          float xInterval = (float)
(INVMGLSTRK_CalculateValues.d_dis_km[i_no_obs] / 5);

          float xend =0;
          maxZ = (float)INVMGLSTRK_CalculateValues.cftnt[2];
```

```
            float points1 = maxZ / 5 ;
            for (int x = zInterval + 250,j = 1; x < 550; x += zInterval){
                  g2.drawString("-", 148+50, 52 + x);
                  g2.drawString("" +d.format(points1 * j), 125+50, 50 +
x);
                  j++;
            }
            g2.setColor(Color.YELLOW);
            g2.fill(new Rectangle2D.Float(200, 300+(float)( 250 *
INVMGLSTRK_CalculateValues.cftnt[1] / maxZ), 450 , 250-(float)( 250 *
INVMGLSTRK_CalculateValues.cftnt[1] / maxZ)));

            g2.setColor(Color.BLACK);
            i_no_obs = INVMGLSTRK_CalculateValues.i_no_obs;
            obs = new double[i_no_obs+1];
            for (int i = 1; i <= i_no_obs; i++) {
                  obs[i] = INVMGLSTRK_CalculateValues.d_dis_km[i];
            }
            maxX = (float) obs[i_no_obs];
            maxZ = (float)INVMGLSTRK_CalculateValues.cftnt[2];
            float s = 0;
            float fc = 0;
            float spoint = 300;
            float fcpoint = (float)( 200 + ( 450 *
INVMGLSTRK_CalculateValues.cftnt[3] / maxX ) );
            float xpoint = 0;
            float zpoint = 0;
            while (s <= INVMGLSTRK_CalculateValues.cftnt[2]) {
                  float z1 = (float) 0.001;
                  s = s + z1;
                  fc = 0;
                  for (int i = 1;
i<=INVMGLSTRK_CalculateValues.i_d_poly+1; i++){
                        fc = (float) (fc +
INVMGLSTRK_CalculateValues.cftnt[i+2] * Math.pow(s, i - 1));
                  }
                  xpoint = (float)( 450 * fc / maxX);
                  zpoint = (float)( 250 * s / maxZ);
                  g2.setColor(Color.BLACK);
                  if(200+xpoint >600+50)
                        xend = 600+50;
                  else
                        xend = 200+xpoint;
                  g2.draw(new Line2D.Float(fcpoint, spoint,xend, 300 +
zpoint));
                  g2.setColor(Color.RED);
                  g2.draw(new Line2D.Float(200, 300 + zpoint, xend, 300 +
zpoint));
```

```
                fcpoint = (float)200 + xpoint;
                spoint = 300 + zpoint;
            }
            g2.setColor(Color.red);
            g2.fill(new Rectangle2D.Float(200, 300, 450 , (float)( 250 *
INVMGLSTRK_CalculateValues.cftnt[1] / maxZ)));
            g2.setColor(Color.WHITE);
            g2.fill(new Rectangle2D.Float(601+50, 300, 450 , 255));

            g2.setColor(Color.BLACK);
            g2.drawLine(200, 300, 650, 300);
            g2.drawLine(200, 300, 600, 300);
            g2.drawString("|",(float) 600-3 +50 , 308);

g2.drawString(""+d.format(INVMGLSTRK_CalculateValues.d_dis_km[i_no_obs]),
(float) 600-10+50 , 323);

            g2.draw(new Line2D.Float(200, 50, 200, 300));
            g2.drawLine(90,10 ,1040,10);
            g2.drawLine(90, 560, 1040, 560);
            g2.drawLine(1040, 10, 1040, 560);
            g2.drawLine(90, 10, 90, 560);

            for (float x = xInterval, j = 1; x < 600; x += xInterval){
                xplot = xplot + xInterval;
                if(j > 4)
                    break;
                g2.drawString("|",(float) (200 + (450 * x / maxX) ),
308);
                g2.drawString("" + d.format(xplot), (float) (200 + (450
* x / maxX) ) - 3, 323);
                j++;
            }
            g2.setFont(new Font("Arial", 40,20));
            if(INVMGLSTRK_CalculateValues.val_code==1)
                g2.drawString("Vertical magnetic anomaly",660, 100);
            if(INVMGLSTRK_CalculateValues.val_code==2)
                g2.drawString("Horizontal magnetic anomaly",660, 100);
            if(INVMGLSTRK_CalculateValues.val_code==3)
                g2.drawString("Total magnetic anomaly",660, 100);
    }

    public void drawOBJ(Graphics2D g2) {

            g2.setFont(new Font("Arial", 40,11));
            g2.setColor(Color.BLACK);
            g2.drawLine(800, 200, 800, 290);
            g2.drawLine(800, 290, 890, 290);
```

```
            g2.drawString("J", 780, 220);

            double maxOb =
INVMGLSTRK_Utility.findMaximumNumber1(INVMGLSTRK_CalculateValues.o_funct)
;
            int ini =
INVMGLSTRK_Utility.findMaximumNumber(INVMGLSTRK_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) {
                  g2.drawString("'", 801+x, 300);
                  g2.drawString("" + (point*j), 800 + x-3, 305);
                  j++;
            }

            float prevx = 800;
            float prevy = 200;
            float xpoint = 0;
            float ypoint = 0;

            for (int i = 1; i <= INVMGLSTRK_CalculateValues.o_iter; i++) {


                  xpoint = (float)( 250 * i /maxiter );
                  ypoint = 200 - (float)  ( ( 90 *
(INVMGLSTRK_CalculateValues.o_funct[i]) / maxOb ) );

                  if(i == INVMGLSTRK_CalculateValues.o_iter){
                        g2.draw(new Line2D.Float(prevx, prevy, 800 +
xpoint-4, 90 + ypoint));
                  }
                  else {
                        g2.draw(new Line2D.Float(prevx, prevy, 800 +
xpoint, 90 + ypoint));
                  }


                  prevx = 800 + xpoint;
                  prevy = 90 + ypoint;

            }

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

```
            g2.drawString("
"+d.format(INVMGLSTRK_CalculateValues.o_funct[1]), 760, 200);
            g2.setFont(new Font("Arial", 40,11));
            g2.drawString ("Iterations",830,316);

            g2.drawLine(750, 180, 910, 180);
            g2.drawLine(750, 180, 750, 320);
            g2.drawLine(750, 320, 910, 320);
            g2.drawLine(910, 180, 910, 320);

            g2.setColor(Color.BLUE);
            g2.setFont(new Font("Arial", 20,30));
            g2.drawString(".... ",750,400);
            g2.setFont(new Font("Arial", 40,15));
            g2.drawString(":Observed Anomalies ", 785, 400);
            g2.setColor(Color.BLACK);
            g2.drawString("-------:Calculated Anomalies ", 750, 420);
      }
}
```

```
package com.invmglstrk.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.invmglstrk.util.INVMGLSTRK_HandleException;
import com.invmglstrk.util.INVMGLSTRK_Utility;
import com.invmglstrk.view.INVMGLSTRK_TableView;
import com.invmglstrk.view.INVMGLSTRK_MainPanel;


public class INVMGLSTRK_CalculateValues {

      public static Object obj[][] = null;
```

```
     public static int i_no_obs, i_d_poly,len,
val_code,iter,max,min,o_iter = 0;
     public static double val_str ,xmax,xmin,rint,tmax=0,tmin= 0;
     public static double d_dis_km[], ano[], obs_ano[], o_funct[],  x[],
cftnt[] = null;
     public static String  input_profile_num = "";

     public static BufferedImage image;

     public void getAnamolyValues(HashMap h_Map) {

          try {
               i_no_obs =
INVMGLSTRK_Utility.convertInteger((String)h_Map.get("N_OBS"));
               i_d_poly =
INVMGLSTRK_Utility.convertInteger((String)h_Map.get("D_POLY"));
               d_dis_km =
INVMGLSTRK_Utility.convertDoubleArray((String)h_Map.get("DIS_KM"));
               obs_ano =
INVMGLSTRK_Utility.convertDoubleArray((String)h_Map.get("OBS_ANO"));
               val_str =
INVMGLSTRK_Utility.convertDouble((String)h_Map.get("VAL_STR"));
               input_profile_num =
INVMGLSTRK_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
               iter =
INVMGLSTRK_Utility.convertInteger((String)h_Map.get("NUM_ITER"));
               val_code =
INVMGLSTRK_Utility.convertInteger((String)h_Map.get("VAL_CODE"));
          }
          catch(Exception e) {

               e.printStackTrace();
          }
     }

     public void cal() throws INVMGLSTRK_HandleException{

          double funct1 = 0.0;
          double []par = new double[i_no_obs + 1];
          double []par1 = new double[i_no_obs + 1];
          double []par2 = new double[i_no_obs + 1];
          double []dupar = new double[i_no_obs + 1];
          double []g1 = new double[i_no_obs + 1];
          double []g2 = new double[i_no_obs + 1];
          double [][]s = new double[i_no_obs + 1][i_no_obs + 1];
          double [][]ppa = new double[i_no_obs + 1][i_no_obs + 1];
          double []b = new double[i_no_obs + 1];
```

```
double []ks = new double[2];
double []aa = new double[5];
double []pb = new double[5];
double []gc  = new double[i_no_obs + 1];
double []err  = new double[i_no_obs + 1];

double d, sn, z1, phir,w,phi1, phi2, xmxmn=0;
int np1 = i_no_obs + 1;
double [][]p = new double[i_no_obs+1][np1+1];
double lambda = 0.5;
ano = new double[i_no_obs + 1];
cftnt = new double [i_d_poly + 4];
o_funct = new double[iter + 5];

x = new double[i_no_obs + 5];
double dm = 0,dm1, phidr,thpph,aj = 0;

if(val_code==1)
      dm = 90;
if(val_code==2)
      dm = 0;
if(val_code==3)
      dm = 22/7;
for (int i = 1; i <= i_no_obs; i++){

      x[i] = d_dis_km[i];

}

double rad = 0.0174532924;
double dmr = rad * dm;
double ori =0;
int np = i_d_poly + 3;
double str = rad * val_str;
double theta = 90.0;
double thir = rad * theta;
int nnk2 = 2;
double piby2 = 1.570796362;
double pi = piby2 * 2.0;
double aer = i_no_obs * 0.1;

double sd = Math.cos(str) * Math.cos(dmr);
sd = sd * sd;
if(dmr == 0.0)
      dm1 = piby2;
else
      dm1 = Math.atan(Math.sin(str ) / Math.tan(dmr));
```

```
            getMAXMIN();

            double r = Math.abs(tmin / tmax);

            if(r > 0.05) {
                  xmxmn = xmax - xmin;
                  d = getORIGIN(ori);
                  if(r > 0.55)
                        phidr = piby2;
                  else
                        phidr = Math.atan((2 * Math.sqrt(r)) / (1 - r));

                  sn = Math.sin(phidr);
                  z1 = 0.5 * Math.abs(xmxmn * sn) / Math.sqrt(9.0 - 4.0 *
sn * sn);
            }
            else{
                  d = xmax;
                  phidr = 0.0;
                  z1 = 0.224 * getWIDTH(d_dis_km, obs_ano, i_no_obs,
tmax);
            }

            if(xmax - xmin < 0){

                  if(tmax < 0)

                        phidr = pi - phidr;

                  else{
                        phidr = 2.0 * pi - phidr;
                        if(phidr >= 2.0 * pi)
                              phidr = phidr - 2.0 * pi;
                  }
            }
            else{

                  if(tmax<0)

                        phidr = pi + phidr;

            }
            phir = phidr + dm1;
            if(phir >= 2.0 * pi) phir = phir - 2.0 * pi;
            if(phir < 0.0)phir = phir + 2.0 * pi;
            par[1]= z1;
            par[2]= 8.0 * z1;
```

```
            for( int ii = 1; ii <= 500; ii++){
                  if(theta == 90.0)
                        w = 0.0;
                  else
                        w = (par[2] - par[1]) / Math.tan(thir);

                  for(int k = 1; k <= i_no_obs; k++){
                        double XX = x[k] - d;
                        if(z1 != 0.0)
                              phi1 = Math.atan(XX / par[1]);
                        else if(XX == 0.0)
                              phi1 = 0.0;
                        else
                              phi1 = piby2 * XX / Math.abs(XX);

                        phi2 = Math.atan((XX + w) / par[2]);
                        double  R1 = (XX * XX + par[1] * par[1]);
                        if(R1 == 0.0)R1 = 1.0E-06;
                        double R2 = (Math.pow(XX + w, 2) + par[2] *
par[2]);
                        double tangle = phi2 - phi1;
                        double tlog = 0.5 * Math.log(R2 / R1);
                        aa[1] = tangle;
                        aa[2] = tlog;
                        aa[3] = obs_ano[k];
                        for(int j = 1;j <= 2; j++){
                              for(int l = 1; l <= 3; l++){
                                    ppa[j][l] = ppa[j][l] + aa[j] * aa[l];
                              }
                        }
                  }
                  SIMEQ(ppa, pb, nnk2, ks);
                  double c1 = pb[1];
                  double  c2 = pb[2];
                  double effj = Math.sqrt(c1 * c1 + c2 * c2);
                  effj = effj / Math.sqrt(1.-sd);

                  aj = effj * 0.5;
                  if(Math.abs(c1) == 0.0)
                        thpph = piby2;
                  else
                        thpph = Math.atan(c2 / c1);

                  if(c1 < 0.0)
                        thpph = pi + thpph;
                  thir = thpph - phidr;

                  if(thir < 0.0)
```

```
                    thir = 2.0 * pi + thir;
               theta = thir / rad;

          }

          par[3] = d;

          for(int ii = 4; ii <= i_d_poly + 3; ii++){
               par[ii] = 0.0;
          }
          double dpar = 0.01;
          GF(x,i_no_obs, np, i_d_poly, aj, phir, str, val_code, par,
gc);
          funct1 = 0.0;

          for(int k = 1; k<=i_no_obs; k++){
               err[k] = obs_ano[k] - gc[k];
               funct1 = funct1+Math.pow(err[k], 2);
          }
          funct1 = Math.sqrt(funct1 / i_no_obs);
          np1 = np + 1;
          double funct2 = 0.0;

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

               o_funct[it] = funct1;
               for(int k = 1; k <= np; k++){
                    par1[k] = par[k];
               }

               for(int i = 1; i <= np; i++){
                    par1[i] = par[i] + dpar / 2.0;

GF(x,i_no_obs,np,i_d_poly,aj,phir,str,val_code,par1,g1);
                    par1[i] = par[i] - dpar / 2.0;

GF(x,i_no_obs,np,i_d_poly,aj,phir,str,val_code,par1,g2);

                    for(int k = 1; k <= i_no_obs; k++){
                         s[i][k] = (g1[k] - g2[k]) / dpar;
                    }

               }
               for (int  J = 1; J <= np1; J++) {
                    for (int I = 1; I <= np; I++) {
                         p[I][J] = 0.0;
                    }
               }
```

```
                for(int J = 1;J <= np;J++)  {
                      for(int I = 1;I <= np;I++) {
                            for(int K = 1;K <= i_no_obs; K++) {
                                  p[I][J] = p[I][J] + s[I][K] * s[J][K];
                            }
                      }
                }

                for (int J = 1; J <= np; J++) {
                      for (int K = 1; K <= i_no_obs; K++) {
                            p[J][np1] = p[J][np1] + err[K] * s[J][K];
                      }
                }


                do {
                      double con = lambda + 1.0;
                      for (int I = 1; I <= np; I++) {

                            dupar[I] = par[I];
                      }
                      for (int L = 1; L <= np; L++) {

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

                                  if (L - J == 0)
                                        p[L][J] = p[L][J] * con;
                            }
                      }
                      SIMEQ(p,b,np,ks);
                      for (int I = 1; I <= np; I++) {

                            par2[I] = dupar[I] + 0.5* b[I];
                      }


GF(x,i_no_obs,np,i_d_poly,aj,phir,str,val_code,par2,gc);
                      funct2 = 0.0;
                      for (int K = 1; K <= i_no_obs; K++) {

                            err[K] = obs_ano[K] - gc[K];
                            ano[K] = gc[K];
                            funct2 = funct2 + Math.pow(err[K] , 2);

                      }

                      funct2 = Math.sqrt(funct2/i_no_obs);
```

```
                    if (funct1 - funct2 < 0) {

                         lambda = lambda * 2.0;
                         for (int I = 1; I <= np; I++) {

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

                                   if (I - J == 0) {

                                        p[I][J] = p[I][J] / con;
                                   }
                              }
                         }

                    }


               } while (funct1 - funct2 < 0);

               o_iter = it;

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

                    par[I] = par2[I];
                    cftnt[I] = par[I];
               }

               setGraphValues(x, obs_ano,ano, cftnt);
               drawGraph();
               if ( funct1 <= aer)
                    break;
               funct1 = funct2;
               lambda = lambda / 2.0;
          }


     }

     public static double []GF(double []x,int nobs,int np,int
ndgre,double aj,double phi,double str,int icode,double []par,double []gc)
{

          double []z = new double[100];
          double []cftnt = new double[10];
          double []gs= new double[1000] ;
          double gmod = 0;
          double dm = 0;
```

```
            if (icode==1) dm=90;
            if (icode==2) dm=0;
            if (icode==3) dm=phi;
            double z1=par[1];
            double z2=par[2];

            for(int it=3; it<=np;it++){
                  cftnt[it-2] = par[it];
            }
            double rad = 3.14159265 / 180;
            double dmr = rad * dm;
            double cont = 2 * aj * Math.sqrt(1 - (Math.pow(Math.cos(str),
2) * Math.pow(Math.cos(dmr), 2)));
            double ter = phi - Math.atan(Math.sin(str) / Math.tan(dmr));
            double dx = (x[2] - x[1]) / 10;
            double zdif = z2 - z1;

            int nd = (int)( zdif / dx ) + 1;
            int na1 = nd / 2;
            if (nd - (2 * na1 ) < 0 || nd - ( 2 * na1 ) > 0) {

                  nd = nd + 1;
            }
            double dz = zdif / nd;
            int nn2 = nd + 1;

            for (int JZ = 1; JZ <= nn2; JZ++) {

                  z[JZ] = z1 + dz * ( JZ - 1 );
            }

            for(int K = 1; K <= nobs; K++){
                  double  xx = x[K];

                  for(int JZ = 1; JZ <= nn2; JZ++){
                        double sum=0.0;

                        for(int jj = 1; jj <= ndgre + 1; jj++){
                              sum = sum + cftnt[jj] * Math.pow(z[JZ],(jj -
1));
                        }
                        double c = Math.cos(ter);
                        double S = Math.sin(ter);
                        double tnum1 = (z[JZ]) * c;
                        double tnum2 = (sum - xx) * S;
                        double tden1 = Math.pow((z[JZ]), 2);
                        double tden2= Math.pow((sum - xx), 2);
```

```
                    gs[JZ] = cont * ((tnum1 - tnum2) / (tden1 +
tden2));

               }
               gmod = SIMP(gs, z, nn2);
               gc[K] = gmod;

          }
          return gc;

     }

     public static double SIMP(double []gs,double []z,int n) {
          double ggc=0;
          double dz = z[2] - z[1];
          double sum1 = 0;
          double sum2 = 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 getMAXMIN(){

      for ( int K = 1; K <= i_no_obs; K++ ){
            if(Math.abs(tmax) - Math.abs(obs_ano[K]) <= 0)
            {
                  tmax = obs_ano[K];
                  max = K;

            }
      }

      double GR1 = obs_ano[max] - obs_ano[max - 1];
      double GR2 = obs_ano[max + 1] - obs_ano[max];

      if((GR1 + GR2) != 0.0){
            xmax = 0.5 * (d_dis_km[max] + d_dis_km[max - 1] - GR1 *
(d_dis_km[max + 1] - d_dis_km[max - 1]) / (GR2 - GR1));
            tmax = RINT(d_dis_km, i_no_obs, obs_ano, xmax);

      }
      else{
            tmax = obs_ano[max];
            xmax = d_dis_km[max];
```

```
        }
        tmin = 0.0;
        xmin = d_dis_km[i_no_obs];

        if(tmax <= 0){

              for(int K = 1; K <= i_no_obs; K++){
                    if(tmin - obs_ano[K] <= 0)
                    {
                          tmin = obs_ano[K];
                          xmin = d_dis_km[K];
                          min = K;

                    }

              }
        }

        else{

              for(int K = 1; K <= i_no_obs; K++){

                    if(tmin - obs_ano[K] >= 0)
                    {
                          tmin = obs_ano[K];
                          xmin = d_dis_km[K];
                          min = K;

                    }
              }
        }

        if(tmin != 0)
        {
              GR1 = obs_ano[min] - obs_ano[min - 1];
              GR2 = obs_ano[min + 1] - obs_ano[min];
              if((GR1+GR2) != 0.0){
                    xmin = 0.5 * (d_dis_km[min] + d_dis_km[min - 1] -
GR1 * (d_dis_km[min + 1] - d_dis_km[min - 1]) / (GR2 - GR1));
                    tmin = RINT(d_dis_km, i_no_obs, obs_ano, xmin);

              }
              else{
                    tmin = obs_ano[min];
                    xmin = d_dis_km[min];

              }
        }
```

```
	}

	public static double getORIGIN(double ORIG){

		double  T0 = tmax + tmin;

		if(xmax-xmin >= 0)
		{
			if(tmax < 0)
			{
				for(int K= (min + 1); K <= i_no_obs;){
					if(T0 - obs_ano[K] == 0){
						ORIG = d_dis_km[K];
						return ORIG;
					}
					if(T0 - obs_ano[K] > 0){
						ORIG = d_dis_km[K-1] + (d_dis_km[K] -
d_dis_km[K-1]) * (T0 - obs_ano[K-1]) / (obs_ano[K] - obs_ano[K-1]);
						return ORIG;
					}
					if(T0 - obs_ano[K] < 0)
						K = K + 1;
				}

			}

			else{
				for(int K= (min + 1); K <= i_no_obs;){

					if(T0-obs_ano[K] == 0){
						ORIG = d_dis_km[K];
						return ORIG;
					}
					if(T0 - obs_ano[K] < 0){

						ORIG = d_dis_km[K - 1] + (d_dis_km[K] -
d_dis_km[K-1]) * (T0-obs_ano[K - 1]) / (obs_ano[K] - obs_ano[K-1]);
						return ORIG;
					}
					if(T0 - obs_ano[K] > 0){
						K = K + 1;
					}
				}
			}

		}
```

```
            else{

                  if(tmax < 0){
                        for(int K = (max + 1); K <= i_no_obs;){
                              if(T0 - obs_ano[K] == 0){
                                    ORIG = d_dis_km[K];
                                    return ORIG;
                              };
                              if(T0 - obs_ano[K] < 0){
                                    ORIG = d_dis_km[K - 1] + (d_dis_km[K] -
d_dis_km[K - 1]) * (T0 - obs_ano[K - 1]) / (obs_ano[K] - obs_ano[K - 1]);
                                    break;
                              }
                              if(T0 - obs_ano[K] > 0)
                                    K = K + 1;
                        }
                  }
                  else{
                        for(int K = (max + 1); K <= i_no_obs; K++){
                              if(T0 - obs_ano[K] == 0){
                                    ORIG = d_dis_km[K];

                                    return ORIG;
                              }
                              if(T0 - obs_ano[K] > 0){
                                    ORIG = d_dis_km[K - 1] + (d_dis_km[K] -
d_dis_km[K - 1]) * (T0 - obs_ano[K - 1]) / (obs_ano[K] - obs_ano[K - 1]);

                                    return ORIG;
                              }
                              if(T0 - obs_ano[K] < 0)
                                    K = K + 1;
                        }
                  }

            }
            return ORIG;

      }


      public static double  RINT(double []x,int n,double[]fx,double xx){

            int m = 0;

            for(int K = 1; K <= n; K++){
```

```
                if(xx-x[K]==0) {
                      rint = fx[K];

                      return rint;
                }
                if(xx - x[K] < 0)
                {
                      m = K - 1;
                      break;
                }
           }

           if((xx - x[m]) > (x[m + 1] - xx))m = m + 1;
           if(m <= 2 || m > (n - 2)){
                rint = fx[m];
                return rint;
           }

           double X1 = x[m - 2];
           double X2 = x[m - 1];
           double X0 = x[m];
           double X3 = x[m + 1];
           double X4 = x[m + 2];
           double T1 = fx[m - 2];
           double T2 = fx[m - 1];
           double T0 = fx[m];
           double T3 = fx[m + 1];
           double T4 = fx[m + 2];
           double A = (xx - X1) * (xx - X2) * (xx - X0) * (xx - X4) * (xx
- X3);
           double B = T1 / ((xx - X1) * (X1 - X2) * (X1 - X3) * (X1 - X4)
* (X1 - X0));
           double C = T2 / ((xx - X2) * (X2 - X1) * (X2 - X3) *(X2 - X4)
* (X2 - X0));
           double D = T0 / ((xx - X0) * (X0 - X1) * (X0 - X2) *(X0 - X3)
* (X0 - X4));
           double E = T3 / ((xx - X3) * (X3 - X1) * (X3 - X2) * (X3 - X4)
* (X3 - X0));
           double F = T4 / ((xx - X4) * (X4 - X1) * (X4 - X2) * (X4 - X3)
* (X4 - X0));
           rint = A * (B + C + D + E + F);
           return rint;

     }

     public static double getWIDTH(double []X,double []T,int N, double
TMAX){
```

```
            double XH1 = 0, XH2=0;
            double TH = 0.5 * TMAX;

            if(TMAX >= 0)

            {

                  for(int K = 1; K <= N;){
                        if(TH - T[K] == 0)
                              XH1 = X[K];
                        if(TH - T[K] < 0)
                              XH1 = X[K - 1] + (X[K] - X[K - 1]) * (TH -
T[K - 1]) / (T[K] - T[K - 1]);
                        if(TH - T[K] > 0)
                              K = K + 1;
                        if(TH - T[K] == 0)
                              XH2 = X[K];
                        if(TH - T[K] < 0)
                              XH2 = X[K - 1] + (X[K] - X[K - 1]) * (TH -
T[K - 1]) / (T[K] - T[K - 1]);

                        if(TH-T[K] > 0)
                              K = K + 1;

                  }
            }
            else{
                  for(int K = 1; K <= N;){
                        if(TH - T[K] == 0)
                              XH1 = X[K];

                        if(TH - T[K] < 0)
                              XH1 = X[K - 1] + (X[K] - X[K - 1]) * (TH -
T[K - 1]) / (T[K] - T[K - 1]);

                        if(TH - T[K] > 0)
                              K = K + 1;
                        if(TH - T[K] == 0)
                              XH2 = X[K];

                        if(TH - T[K] < 0)
                              XH2 = X[K - 1] + (X[K] - X[K - 1]) * (TH-T[K
- 1]) / (T[K] - T[K - 1]);

                        if(TH - T[K] > 0)
                              K = K + 1;
                  }
            }
```

```
            double WIDTH = Math.abs(XH2 - XH1);
            return WIDTH;
      }

      public static void  setGraphValues(double []x, double
[]obsano,double []calano, double []coeff) {

            obj = new Object[i_no_obs + 1][4];

            DecimalFormat d = new DecimalFormat("0.##");
            DecimalFormat df = new DecimalFormat("0.###");
            DecimalFormat df1 = new DecimalFormat("0.####");

            for (int K = 1;K <= i_no_obs; K++){

                  obj[K][0]= "" + d.format(x[K]);

                  obj[K][1]= "" + df.format(obsano[K]);

                  obj[K][2]= "" + df.format(calano[K]);
            }

            INVMGLSTRK_TableView.val.setText("");

            INVMGLSTRK_TableView.val.appendText("\n");
            INVMGLSTRK_TableView.val.appendText("Iteration number:"+" = "
+o_iter+ "\n");
            INVMGLSTRK_TableView.val.appendText("\n");
            INVMGLSTRK_TableView.val.appendText("Depth to the top:"+" = "
+df1.format(coeff[1])+ "\n");
            INVMGLSTRK_TableView.val.appendText("\n");
            INVMGLSTRK_TableView.val.appendText("Depth to the Bottom:"+" =
" +df1.format(coeff[2])+ "\n");
            INVMGLSTRK_TableView.val.appendText("\n");
            INVMGLSTRK_TableView.val.append("Coefficients of the
polynomial:-\n");

INVMGLSTRK_TableView.val.appendText("----------------------------------
-------\n");
            INVMGLSTRK_TableView.val.appendText("\n");

            for (int i = 3; i < coeff.length; i++){

                  INVMGLSTRK_TableView.val.appendText("a"+(i - 3)+" = "
+df1.format(coeff[i])+ "\n");
            }
```

```
	}


	public static void drawGraph(){
		final com.invmglstrk.view.INVMGLSTRK_DrawGraph dg = new
com.invmglstrk.view.INVMGLSTRK_DrawGraph();
		try
		{
			int width = 1280;
			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);
			dg.plotXYCoordinates(g2);
			dg.plotZCoordinates(g2);
			dg.drawOBJ(g2);
			dg.plot(g2);


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

			String path =
INVMGLSTRK_CalculateValues.input_profile_num  +".jpg";
			image = ImageIO.read(new File(path));

			Graphics g_image =
INVMGLSTRK_MainPanel.img.getGraphics();
			g_image.drawImage(image, 0, 0, image.getWidth(),
image.getHeight(), dg);
			MouseListener ml3 = new MouseAdapter(){
				public void mouseClicked(MouseEvent e){
					Graphics g_image =
INVMGLSTRK_MainPanel.img.getGraphics();
					g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(),dg);
				}
			};
			INVMGLSTRK_MainPanel.img.addMouseListener(ml3);
		}
		catch (Exception e2) {
```

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

```
package com.invmglstrk.control;

import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.awt.*;
import java.io.*;
import java.text.DecimalFormat;
import javax.imageio.ImageIO;
import javax.swing.*;

import com.invmglstrk.model.INVMGLSTRK_CalculateValues;
import com.invmglstrk.view.INVMGLSTRK_MainPanel;


public class INVMGLSTRK_Controller implements ActionListener {

      Object rowdata [][] = {};
      BufferedImage image;
      com.invmglstrk.view.INVMGLSTRK_DrawGraph dg = new
com.invmglstrk.view.INVMGLSTRK_DrawGraph();
      com.invmglstrk.model.INVMGLSTRK_CalculateValues cv = new
com.invmglstrk.model.INVMGLSTRK_CalculateValues();
      public static boolean success = false;

      public void actionPerformed(ActionEvent ae)  {
            if(ae.getActionCommand().equals("Load file")){

                  INVMGLSTRK_MainPanel.loadData();


INVMGLSTRK_MainPanel.p_Center.add(INVMGLSTRK_MainPanel.graphLabel);

INVMGLSTRK_MainPanel.p_Center.add(INVMGLSTRK_MainPanel.img);
                  try{



cv.getAnamolyValues(com.invmglstrk.view.INVMGLSTRK_MainPanel.captureValue
s());
```

```
                }
                catch(Exception e) {
                      e.printStackTrace();
                }

           }

           else if(ae.getActionCommand().equals("Save and Print")){

                try{
                      String current = System.getProperty("user.dir");
                      File img_file = new
File(INVMGLSTRK_CalculateValues.input_profile_num+".jpg");
                      JFileChooser saveFile = new JFileChooser(current);
                      File OutFile = saveFile.getSelectedFile();//new
File(new
File(INGRLSTRK_CalculateValues1.input_area_name+".html").getCanonicalPath
());  //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()));
                                  myWriter = new
FileWriter(OutFile+".html");
                                  myWriter.write("<html> <Body onLoad =
\"window.print()\"><table> <tr> <td>" +"<td> <img src = '"+
INVMGLSTRK_CalculateValues.input_profile_num +".jpg'></td>"+
                                                "<table border = 1> <tr> <th
colspan = 4>Profile ID:-
"+INVMGLSTRK_CalculateValues.input_profile_num+"</th> </tr>");

                                  DecimalFormat df =new
DecimalFormat("0.###");
                                  DecimalFormat d = new
DecimalFormat("0.##");
```

```
                                    myWriter.write("<tr > <th>Distance </
th><th> Observed anamolies (nT) </th><th> Calculated anamolies (nT) </th>
</tr>");

                                    for ( int K = 1; K <=
INVMGLSTRK_CalculateValues.i_no_obs; K++){

                                         myWriter.write("<tr> <td>" +
d.format(INVMGLSTRK_CalculateValues.x[K])+"</td>
<td>"+df.format(INVMGLSTRK_CalculateValues.obs_ano[K])+"</
td><td>"+df.format(INVMGLSTRK_CalculateValues.ano[K])+"</td></tr>");
                                    }
                                    myWriter.write(" </table> </td> </tr></
table><BR>");

myWriter.write("---------------------------------- <BR>");

                                    DecimalFormat d1 =new
DecimalFormat("0.####");
                                    myWriter.write("<BR>");
                                    myWriter.write("Iteration number =
"+INVMGLSTRK_CalculateValues.o_iter+"<BR>");
                                    myWriter.write("<BR>");
                                    myWriter.write("Depth to the top =
"+d1.format(INVMGLSTRK_CalculateValues.cftnt[1])+"<BR>");
                                    myWriter.write("<BR>");
                                    myWriter.write("Depth to the bottom =
"+d1.format(INVMGLSTRK_CalculateValues.cftnt[2])+"<BR>");

                                    myWriter.write("<BR>");
                                    myWriter.write("Coefficient of the
polynomial::"+"<BR>");

myWriter.write("-----------------------------------------<BR>");
                                    for ( int i = 3; i <
INVMGLSTRK_CalculateValues.cftnt .length; i++) {

                                         myWriter.write("a"+ ( i - 3 ) +" =
"+d1.format(INVMGLSTRK_CalculateValues.cftnt[i])+"<BR>");
                                    }

                                    myWriter.close();
                               }
                          }
                          else
                          {
                               //pops up error message
```

```
                    }

               }
               catch(Exception e1) {

                    e1.printStackTrace();
               }
          }



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

               INVMGLSTRK_MainPanel.clearDefaultValues();
               INVMGLSTRK_MainPanel.p_Center.removeAll();

com.invmglstrk.view.INVMGLSTRK_TableView.populateEastPanel(rowdata);

com.invmglstrk.view.INVMGLSTRK_TableView.val.setText("");

INVMGLSTRK_MainPanel.p_Center.add(INVMGLSTRK_MainPanel.graphLabel);
          }else if(ae.getActionCommand().equals("Exit")){

               JFrame frame = null;
               int r =    JOptionPane.showConfirmDialog(
                          frame,
                          "Exit INVMGLSTRK ?",
                          "Confirm Exit ",
                          JOptionPane.YES_NO_OPTION);
               if(r == JOptionPane.YES_OPTION ){
                    if(success==false){
                          String fileName =
INVMGLSTRK_CalculateValues.input_profile_num+".jpg";
                          File f = new File(fileName);
                          f.delete();
                    }
                    System.exit(0);
               }
          }else if(ae.getActionCommand().equals("Inversion")){



INVMGLSTRK_MainPanel.p_Center.add(INVMGLSTRK_MainPanel.graphLabel);
               INVMGLSTRK_MainPanel.img.setEditable(true);

               try
               {
```

```
cv.getAnamolyValues(com.invmglstrk.view.INVMGLSTRK_MainPanel.captureValue
s());

                    if(INVMGLSTRK_CalculateValues.val_code<=0 ||
INVMGLSTRK_CalculateValues.val_code>3){

                         JFrame frame = null;
                         JOptionPane.showMessageDialog(frame,
                                   "Code value must be 1\n"
                                             +"or 2 or 3",
                                             "Out of bounds error",

JOptionPane.ERROR_MESSAGE);

                    }
                    if(INVMGLSTRK_CalculateValues.val_code==1 ||
INVMGLSTRK_CalculateValues.val_code==2 ||
INVMGLSTRK_CalculateValues.val_code == 3){
                         cv.cal();
                         int width = 1280;
                         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);
                         dg.plotZCoordinates(g2);
                         dg.drawOBJ(g2);
                         dg.plot(g2);


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

                         String path =
INVMGLSTRK_CalculateValues.input_profile_num +".jpg";
                         image = ImageIO.read(new File(path));

                         Graphics g_image =
INVMGLSTRK_MainPanel.img.getGraphics();
```

```
                              g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(), dg);

                              MouseListener ml3 = new MouseAdapter(){
                                    public void mouseClicked(MouseEvent e){
                                          Graphics g_image =
INVMGLSTRK_MainPanel.img.getGraphics();
                                          g_image.drawImage(image, 0,
0,image.getWidth(), image.getHeight(),dg);
                                    }
                              };

INVMGLSTRK_MainPanel.img.addMouseListener(ml3);

com.invmglstrk.view.INVMGLSTRK_TableView.populateEastPanel(INVMGLSTRK_Cal
culateValues.obj);
                        }
                  }
                  catch (Exception e2) {

                        e2.printStackTrace();
                  }


            }
      }


}
```

```
package com.invmglstrk.util;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import com.invmglstrk.model.INVMGLSTRK_CalculateValues;



public class INVMGLSTRK_Utility {

      public static double convertDouble(String str) throws Exception {


            Double temp = null;
```

```
	try {
		temp = new Double(str.trim());

	}
	catch(Exception e){

		JFrame frame = null;
		JOptionPane.showMessageDialog(frame,
				"Enter a numerical value.",
				"Number format error",
				JOptionPane.ERROR_MESSAGE);


	}
	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 = null;
	try {
		temp = new Integer(str.trim());
	}
	catch(Exception e){

		JFrame frame = null;
		JOptionPane.showMessageDialog(frame,
				"Enter a numerical value.",
				"Number format error",
				JOptionPane.ERROR_MESSAGE);

	}
	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 double findMinimumNumber( double observe[], double
denVal) {

          double max = denVal;
          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 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 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]);
               }
```

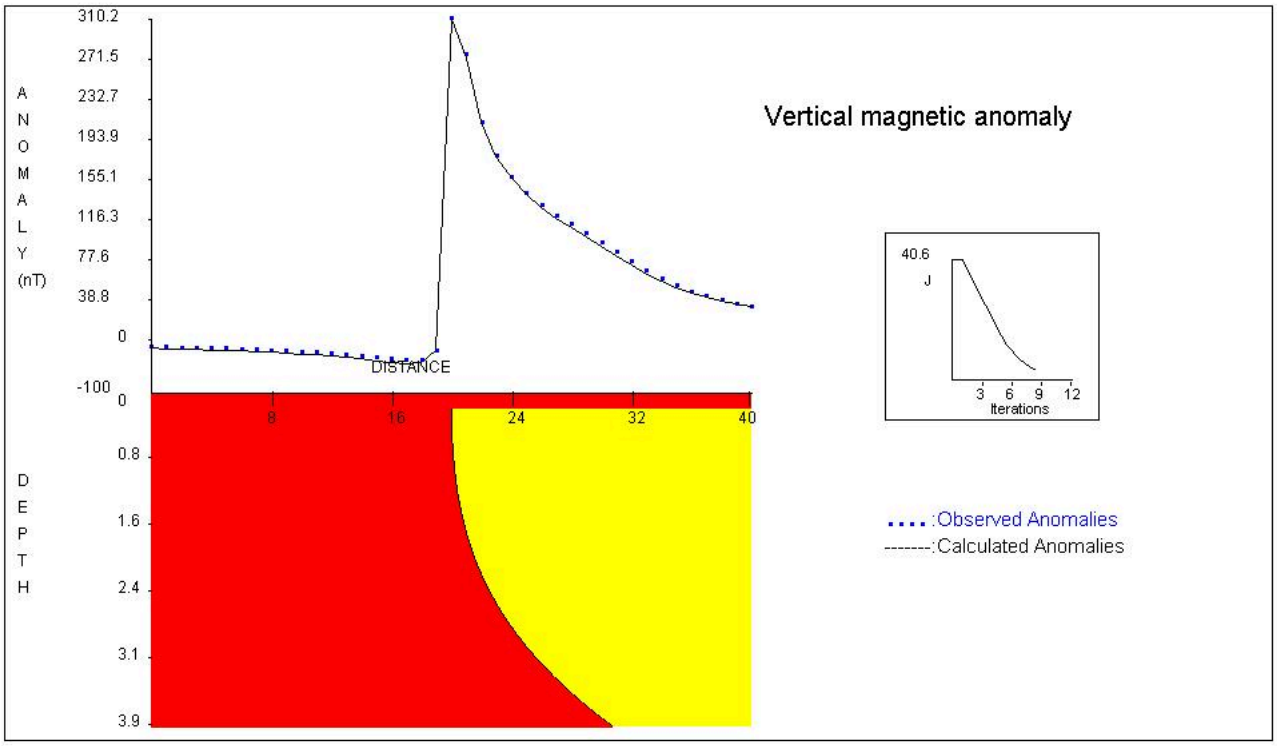
```
            }

            double maxVal =  max;
            return maxVal;
      }
      public static double 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 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 = INVMGLSTRK_CalculateValues.o_iter;
            }

            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 {

				try {
					d_array[i] = convertDouble( arr.get(i -
1).toString() );
				}
				catch(Exception e){
					JFrame frame = null;
					JOptionPane.showMessageDialog(frame,
							"Enter numerical values.",
							"Number format error",
							JOptionPane.ERROR_MESSAGE);
					throw new INVMGLSTRK_HandleException();

				}
			}
		}
		return d_array;
	}
}
```

```
package com.invmglstrk.util;

import java.awt.*;
```

```
import com.invmglstrk.view.INVMGLSTRK_MainPanel;

public class INVMGLSTRK_HandleException extends Exception{

	public INVMGLSTRK_HandleException(){
		Graphics g = INVMGLSTRK_MainPanel.p_Center.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);
	}
}
```

```
abc
41
0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0,10.0,11.0,12.0,13.0,14.0,15.0,16.0,17.0,18.0,
19.0,20.0,21.0,22.0,23.0,24.0,25.0,26.0,27.0,28.0,29.0,30.0,31.0,32.0,33.0,34.0,35.
0,36.0,37.0,38.0,39.0,40.0
-15.03084,-15.62240,-16.26270,-16.95801,-17.71572,-18.54461,-19.45510,-20.45974,-21
.57364,-22.81512,-24.20640,-25.77434,-27.55085,-29.57192,-31.87226,-34.46532,-37.27
201,-39.84222,-39.99644,-23.36207,311.4382,276.3909,210.1111,177.3356,156.4969,141.
4931,129.7508,119.8962,111.0390,102.5033,93.76633,84.57867,75.16166,66.14302,58.117
70,51.32764,45.71421,41.09349,37.26857,34.07107,31.36824
3
40
1
100
```



Profile ID:- abc		
Distance	**Observed anamolies (nT)**	**Calculated anamolies (nT)**
0	-15.031	-17.082
1	-15.622	-17.773
2	-16.263	-18.521
3	-16.958	-19.335
4	-17.716	-20.224
5	-18.545	-21.198
6	-19.455	-22.27
7	-20.46	-23.456
8	-21.574	-24.772
9	-22.815	-26.243
10	-24.206	-27.893
11	-25.774	-29.754
12	-27.551	-31.865
13	-29.572	-34.262
14	-31.872	-36.978
15	-34.465	-40.005
16	-37.272	-43.184
17	-39.842	-45.788
18	-39.996	-44.591
19	-23.362	-20.08
20	311.438	310.232
21	276.391	272.742
22	210.111	210.928
23	177.336	177.604
24	156.497	155.99
25	141.493	140.375
26	129.751	128.164
27	119.896	117.908

28	111.039	108.631
29	102.503	99.6
30	93.766	90.347
31	84.579	80.854
32	75.162	71.606
33	66.143	63.219
34	58.118	56.019
35	51.328	50.01
36	45.714	45.033
37	41.093	40.895
38	37.269	37.425
39	34.071	34.484
40	31.368	31.966

Iteration number = 8

Depth to the top = 0.1964

Depth to the bottom = 3.9302

Coefficient of the polynomial::
---
a0 = 19.9904
a1 = 0.0505
a2 = 0.0981
a3 = 0.1488

J. Earth Syst. Sci. (2021)130 130
https://doi.org/10.1007/s12040-021-01634-x

Forward modelling: Magnetic anomalies of arbitrarily magnetized 2D fault sources with analytically defined fault planes

V Ani Nibisha[1,*], B Ramamma[1], S Rajeswara Sastry[2] and V Chakravarthi[1]

[1]*Centre for Earth, Ocean and Atmospheric Sciences, University of Hyderabad, Hyderabad 500 046, India.*
[2]*203, SkyPX, Nallagandala, Hyderabad 500 019, India.*
**Corresponding author. e-mail: aninibisha@gmail.com*

MS received 29 January 2021; revised 11 March 2021; accepted 11 March 2021

Based on Poisson's relation, a generalized equation to realize forward modelling of magnetic anomalies due to arbitrarily magnetized 2D listric fault sources in any component is derived in the space domain. The non-planar fault plane of a listric fault structure is described with a generalized polynomial equation. The estimated coefficients of a prescribed polynomial are used to construct the fault plane analytically. The validity of the presented formula is established against the theoretical anomalies that are realized by an analytic equation over a vertical fault structure. It is demonstrated with a synthetic example that the magnetic anomalies in any component produced by a typical listric fault source always have lesser magnitude when compared to the corresponding anomalous field produced by the same structure with a planar fault plane assumption.

Keywords. Listric fault morphology; arbitrary magnetization; magnetic anomaly; forward modelling.

1. Introduction

Faults are planar or gently curved fractures in lithosphere rocks that are caused by tectonic and other related disturbances/movements. Large scale faults formed during rifting, drifting, and evolution of passive continental margins are normally associated with the basin development process. More often than not, fault planes of marginal faults associated with thick sedimentary basins are non-planar because the primary detachment fracture more often follows a curved path rather than planar (McKenzie 1978; Smith and Bruhn 1984; Jackson 1987; Goussav *et al.* 2006; Chakravarthi 2011).

The geometry and kinematics of listric faults have gained paramount importance in understanding the large-scale extensional processes and exploring commercially viable mineralized targets. Torizin *et al.* (2009) argue that the study of the nature of fault dips with increasing depth could improve the precision of seismic source characterization. Due to the non-planar nature of fault planes, it is indeed difficult to accurately estimate the major extension and throw of faults from surface geologic observations alone (McKenzie 1978; Chakravarthi 2011). In such cases, the existence of magnetization contrast(s) between the displaced/detached rock masses on either side of fault planes could generate measurable magnetic anomalies, which can be mapped and parameterized to quantify the listric fault morphologies.

A few techniques are available to estimate the parameters of fault structures from observed magnetic anomalies. The use of characteristic curves in the interpretation of magnetic anomalies

Published online: 24 June 2021

had been proposed by Moo (1965), Grant and West (1965), Rao and Murthy (1978), and Rao *et al.* (1980). Based on the application of Fourier integral, Sengupta (1974) had proposed a method to analyse the magnetic anomalies caused by vertical fault structures. Stanley (1977) demonstrated that the horizontal gradient of total magnetic anomaly over a vertical contact is the same as the total magnetic anomaly over a thin dyke and that specific points on the gradient are related to the fault parameters. Qureshy and Nalaye (1978) proposed a technique based on decomposing magnetic anomaly into symmetric and anti-symmetric parts. Rao and Babu (1983) presented standard curves for magnetic anomaly interpretation treating the angle of fault plane as 90° and that the magnetization is caused purely by induction. Murthy (1985) had developed an efficient method of interpreting magnetic anomalies of arbitrarily magnetized fault structures, wherein the anomalies across the structure are scaled at two different elevations followed by identifying the maximum and minimum anomalies and their mid points, which in turn were used to estimate the source parameters.

Mushayandebvu *et al.* (2001) had developed a method using extended Euler deconvolution to analyse the anomalies produced by fault structures, while Murthy *et al.* (2001) used Marquardt's (1963) algorithm to analyse the magnetic anomalies. The inversion scheme proposed by Murthy *et al.* (2001) is noteworthy because their algorithm presumes arbitrary magnetization for the fault structures and analyses the anomalies in any component for the source parameters. Using analytic signal and Euler deconvolution, Doo *et al.* (2007) had devised a technique to estimate the source parameters of a 2D magnetic contact. Subrahmanyam and Rao (2009) have suggested a simple method that uses a few characteristic positions on the magnetic anomaly to find the parameters of a fault structure. Interpretation techniques based on constrained optimization theory (Asfahani and Tlas 2004) and stochastic algorithms (Asfahani and Tlas 2007) are also available currently to analyse the magnetic anomalies of fault structures. However, the practical utility of all the above techniques is limited to analyse the magnetic anomalies caused by large normal faults having non-planar fault planes.

Chakravarthi (2010) had developed a forward modelling technique to compute the gravity anomalies of listric fault sources, among which the density contrast differs continuously with depth. An automatic inversion scheme to simultaneously estimate the fault parameters (listric) and regional gravity background from a set of observed gravity anomalies was also proposed (Chakravarthi 2011). To the best of authors' knowledge, no algorithm is reported/available explicitly to analyse the magnetic anomalies generated by listric fault sources. Therefore, a need exists to develop suitable techniques to analyse magnetic anomalies produced by fault structures presuming (i) arbitrary magnetization for the source and (ii) non-planar surfaces for fault planes.

In this paper, we derive a generalized equation for computing the magnetic anomaly due to a listric fault morphology in any component (i.e., horizontal, vertical, and total field). A computer code in JAVA is developed to realize forward modelling in an interactive mode. The advantage of this code is that it is platform-independent and works on any GUI-based operating system with at least the jdk 1.6 version installed.

2. Forward modelling: Magnetic anomaly of an arbitrary magnetized 2D listric fault source

In the Cartesian co-ordinate system, let the z-axis is positive vertically downwards and x-axis traverse to the strike of a listric fault source whose geometry is shown in figure 1. The 2D fault structure (infinite strike length) is confined between the depth limits z_T and z_B ($z_B > z_T$) along the z-axis. Here, we treat the sediments within the hanging wall as magnetically transparent, while the magnetic interface (fault plane) is only responsible for generating the anomalies. Further, the structure is bounded on the left by a non-planar surface defined by $f(z) = \sum_{i=0}^{n} f_i z^i$, and towards the right, it extends to infinity. Here, f_i represents a set of coefficients, and n stands for the degree of the polynomial. Because both induced and remanent magnetizations are responsible for generating magnetic anomalies over a geologic structure, we presume that the structure is magnetized in an unknown direction along the resultant of both induced and remanent magnetic vectors. For a 2D source, the resultant magnetic field vector, J, can be resolved spatially into three mutually orthogonal components namely, $J\sin\theta$, $J\cos\theta\cos\delta$, and $J\cos\theta\sin\delta$ along the vertical, parallel to the strike of the source, and perpendicular to the strike of the source in the horizontal plane, respectively. Here,

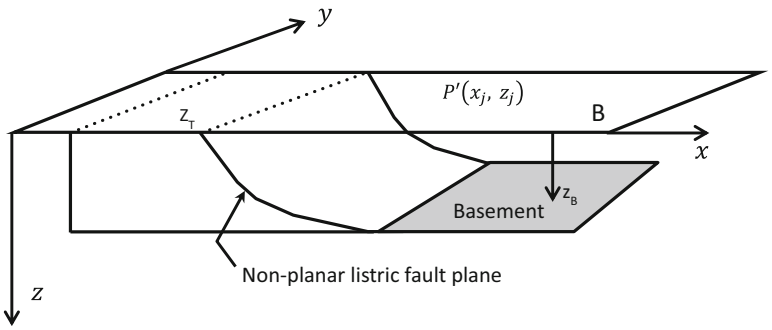


Figure 1. Schematic diagram showing a conceptual geometry of a typical listric fault source. The x-axis is transverse to the strike of the structure, z-axis is vertically positive downwards, and the fault plane is described by a polynomial of specific degree. The source is striking infinitely along the y-axis.

θ is the resultant magnetic dip, and δ denotes the resultant magnetic field vector's declination from the source's strike. Because the component resolved along the strike of the body fails to generate magnetic anomalies, the effective magnetization which is responsible for producing the anomalies can be expressed (Murthy 1998) as:

$$J_{ef} = J\sqrt{(1 - \cos^2\theta\cos^2\delta)}. \tag{1}$$

The dip of the effective magnetization vector is given by

$$\theta_{ef} = \tan^{-1}(\tan\theta\,\mathrm{cosec}\,\delta). \tag{2}$$

The effective magnetization always lies in a vertical plane perpendicular to the strike of the source.

Now considering d_c as the density contrast of the structure, the magnetic potential, W, at any point $P(0,0)$ outside the source region can be expressed using the Poisson's relation as (Murthy 1998)

$$W = -\frac{J_{ef}}{Gd_c}\left\{\frac{\partial U}{\partial x}\cos\theta_{ef} + \frac{\partial U}{\partial z}\sin\theta_{ef}\right\}, \tag{3}$$

where G is universal gravitational constant, and U represents the gravity potential.

The vertical magnetic anomaly ΔV outside the source region can be expressed as:

$$\Delta V = \frac{J_{ef}}{Gd_c}\left[\frac{\partial^2 U}{\partial x\,\partial z}\cos\theta_{ef} + \frac{\partial^2 U}{\partial z^2}\sin\theta_{ef}\right]. \tag{4}$$

The gravity potential U due to the 2D source at the point $P(0,0)$ is given by

$$U = -Gd_c\int_s \ln(x^2 + z^2)\,ds, \tag{5}$$

where s is the cross-sectional area of the structure and (x, z) stands for the source coordinates of an element within the structure. Substituting the expressions for partial derivatives of U from equation (5) in equation (4), we obtain

$$\Delta V = 2J_{ef}\int_s \frac{(z^2 - x^2)\sin\theta_{ef} + 2xz\,cos\,\theta_{ef}}{(x^2 + z^2)^2}\,ds. \tag{6}$$

Applying Stokes' theorem, equation (6) can be rewritten as

$$\Delta V = 2J_{ef}\int_{z=z_T}^{z_B}\left[\int_{x=f(z)}^{\infty}\frac{(z^2 - x^2)\sin\theta_{ef} + 2xz\cos\theta_{ef}}{(x^2 + z^2)^2}\,dx\right]dz. \tag{7}$$

Upon simplification equation (7) takes the form

$$\Delta V = 2J_{ef}\int_{z=z_T}^{z_B}\frac{z\cos\theta_{ef} - f(z)\sin\theta_{ef}}{f^2(z) + z^2}\,dz. \tag{8}$$

The vertical magnetic anomaly due to the structure at any point $P'(x_j, z_j)$ on the topography along the principal profile can be obtained as:

$$\Delta V(x_j, z_j) = 2J_{ef}\int_{z=z_T}^{z_B}\frac{(z - z_j)\cos\theta_{ef} - (f(z) - x_j)\sin\theta_{ef}}{(f(z) - x_j)^2 + (z - z_j)^2}\,dz. \tag{9}$$

Further, the anomaly in horizontal component ΔH at the point $P(0,0)$ can be obtained from equation (3) as:

$$\Delta H = \frac{J_{ef}\sin\vartheta}{Gd_c}\left[\frac{\partial^2 U}{\partial x\,\partial z}\cos\left(\theta_{ef} - \frac{\pi}{2}\right) + \frac{\partial^2 U}{\partial z^2}\sin\left(\theta_{ef} - \frac{\pi}{2}\right)\right], \tag{10}$$

where ϑ is strike of the body. The horizontal magnetic anomaly due to the structure at any

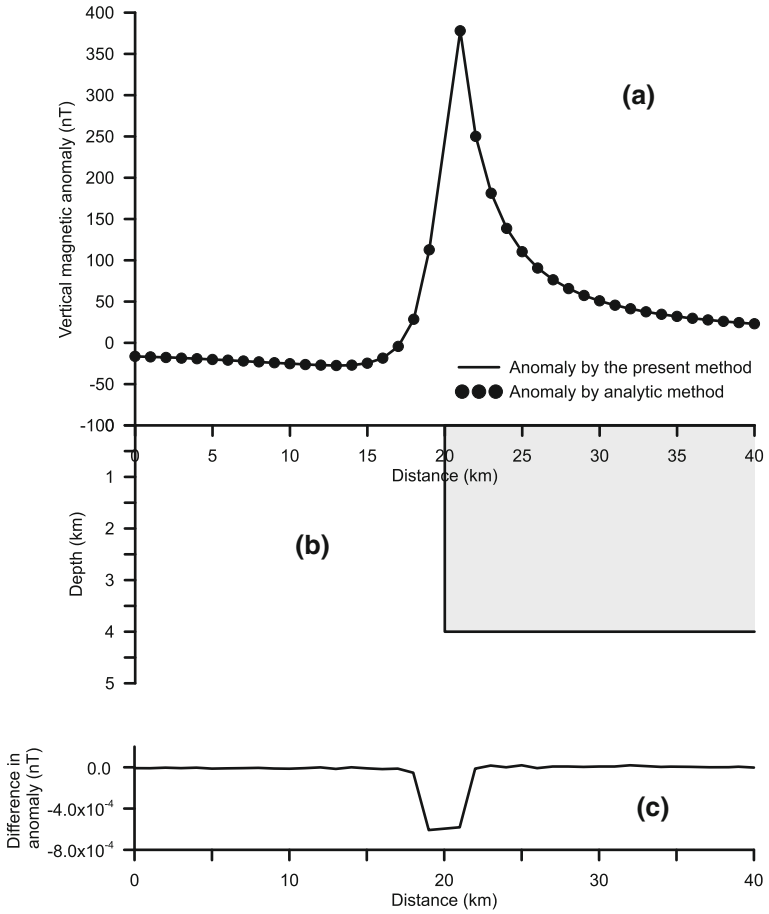


Figure 2. (**a**) Comparison of vertical magnetic anomalies obtained from the present method and the analytic equation (Murthy *et al.* 2001), (**b**) geometry of vertical fault, and (**c**) differences between the anomalies from the two methods.

point $P'(x_j, z_j)$ on the principal profile outside the source region can be expressed using equation (5) as:

$$\Delta H(x_j, z_j) = 2J_{ef} \sin\vartheta \times \int_{z=z_T}^{z_B} \frac{(z - z_j)\cos(\theta_{ef} - \frac{\pi}{2}) - (f(z) - x_j)\sin(\theta_{ef} - \frac{\pi}{2})}{(f(z) - x_j)^2 + (z - z_j)^2} dz. \quad (11)$$

From equations (9 and 11), one can realize that the vertical magnetic anomaly produced by a 2D listric fault source is similar to the horizontal magnetic anomaly produced by the same structure but with a different amplitude and phase.

The generalized equation for the magnetic anomaly in any component due to a listric fault source at an observer location at $P'(x_j, z_j)$ can be finally expressed as:

$$\Delta T(x_j, z_j) = 2J' \int_{z=z_T}^{z_B} \frac{(z - z_j)\cos\theta' - (f(z) - x_j)\sin\theta'}{(f(z) - x_j)^2 + (z - z_j)^2} dz \quad (12)$$

where $J' = J_{ef}\sqrt{(1 - \cos^2\vartheta\cos^2\alpha)}$ and $\theta' = \theta_{ef} - \tan^{-1}(\sin\vartheta/\tan\alpha)$.

Equation (12) is a standard form to calculate the magnetic anomaly of a listric fault source in any specific component. For example, by setting α to 90°, 0°, and i in equation (12) the vertical, horizontal, and total magnetic anomalies can be realized, respectively. Further, it is more appropriate to solve equation (12) by a numerical approach rather than analytical because of the simple fact that the polynomial, $f(z)$, in the integrand may assume any degree (Chakravarthi 2010, 2011).

We demonstrate the validity of equation (12) by comparing the anomalies (in each component) obtained from the present method against the ones realized from an analytical equation (Murthy *et al.* 2001) over a vertical fault structure (figure 2b). In this case, the assumed parameters of the source are $z_T = 0$ km, $z_B = 4.0$ km, $\theta_{ef} = 30°$, $J_{ef} = 100$ nT and $\vartheta = 40°$. The anomalies in each component are calculated along a profile in the interval $x_j \epsilon (0, 40\,\text{km})$ on the observational plane $z_j = 0$ and

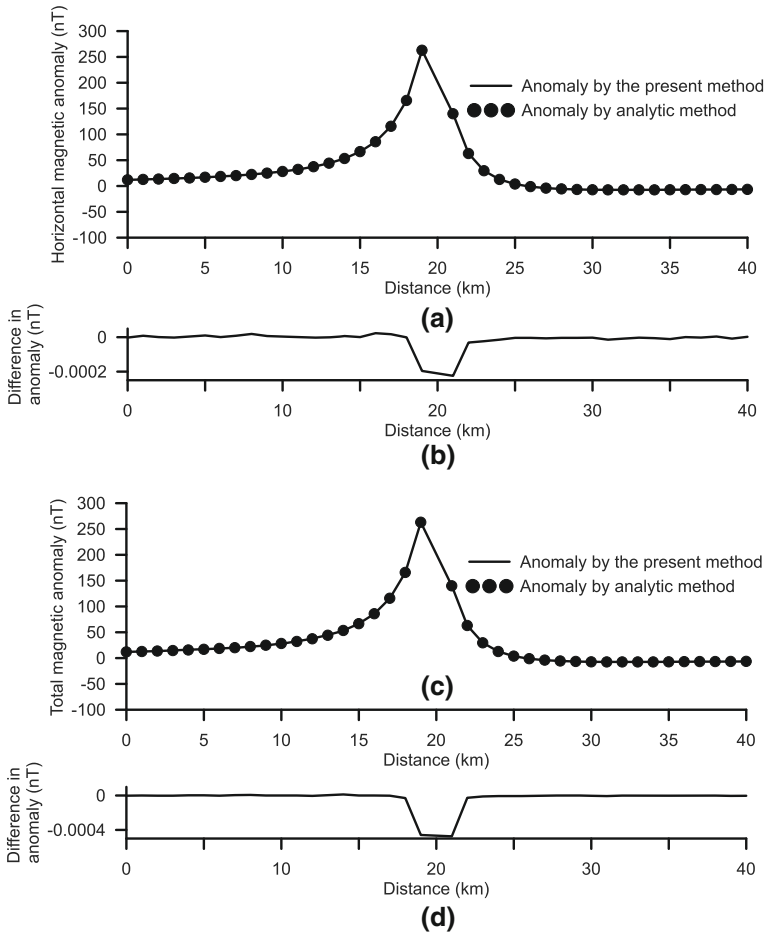


Figure 3. (**a**) Comparison of horizontal magnetic anomalies obtained from the present method and the analytic equation (Murthy *et al.* 2001), (**b**) differences between the horizontal anomalies obtained from the two methods, (**c**) total magnetic anomalies from the present method and the analytic equation (Murthy *et al.* 2001), and (**d**) differences between the total anomalies from the two methods.

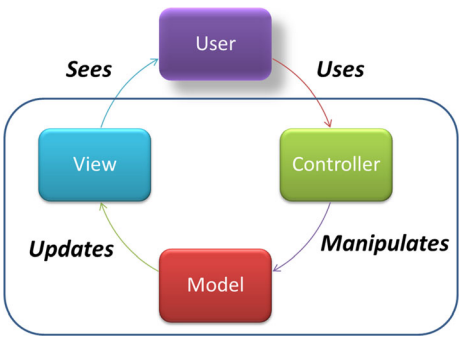


Figure 4. Structural relationship between Model, View and Controller.

shown in figures 2(a) and 3(a, c), respectively. The differences between the anomalies obtained from the present method and the analytic equation in each component are shown in figures 2(c) and 3(b, d). In the case of vertical anomaly, a maximum difference of −6E−04 nT is observed at the 19th km on the profile (figure 2c), whereas in horizontal and total components, the observed maximum differences are −2E−04 nT and −4E−04 nT, respectively (figure 3b and d). These insignificant differences between the anomalies in all the components demonstrate the accuracy of the proposed method of forward modelling.

3. Computer code

Based on the algorithm described in the text, a GUI-based software, FRMGLSTRK, coded in JAVA, is developed to compute the magnetic anomalies of a 2D listric fault source in any component.

The code is built on the Model-View-Controller (MVC) architecture according to the structural relationship shown in figure 4. The module 'Model' estimates the coefficients of a prescribed polynomial to construct the geometry of a fault plane and computes the magnetic anomalies of the structure

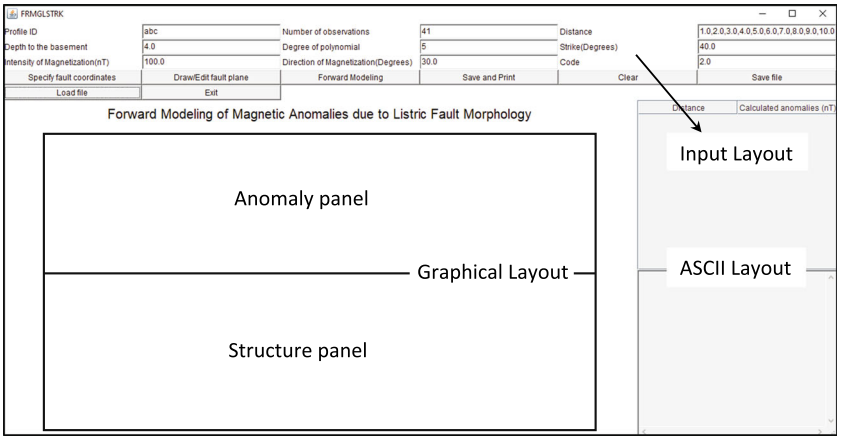


Figure 5. View module of FRMGLSTRK.

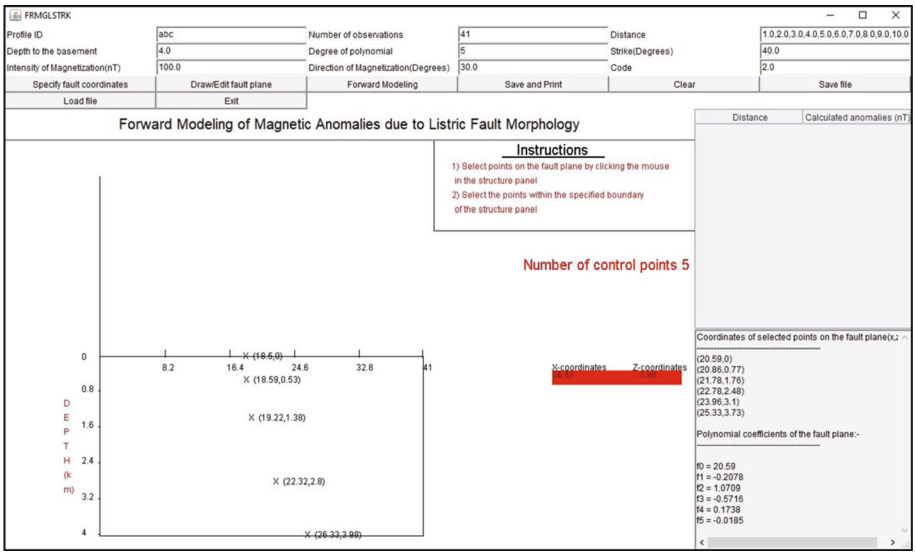


Figure 6. Selection of control points by mouse clicks in the structure panel. The appearance of number of control points in red warrants the selection of addition points.

in the required component. The 'View' module reads the input data and displays the output in both graphical and ASCII forms. The 'Controller' executes the task of passing the required actions to Model and View modules whenever they called for.

Once the batch file of the software is invoked, the view module appears on the monitor, as shown in figure 5. The view module is arranged into the input, graphical, and ASCII layouts (figure 5). The input layout consists of nine input fields and eight action buttons. The graphical layout is divided into the anomaly panel on top and the structural panel at the bottom. The ASCII layout towards the right displays the output in ASCII format.

The input parameters to the code are: profile and ID, number of observations, distance to each observation as measured with reference to the first station (any units), depth to the basement (any units), degree of the polynomial, the strike of the source with reference to magnetic north (degrees), intensity of magnetization (nT), direction of magnetization (degrees), and code number (1 for vertical, 2 for horizontal, and 3 for total magnetic anomaly). The code allows the user to specify the input in two ways: (1) the data can be entered in a notepad and loaded to the code by the 'load file' action button, or (2) data can be directly entered in respective fields of the input layout (figure 5). The action buttons of the input layout are: specify fault coordinates, draw/edit fault plane, forward modelling, save and print, clear, save file, load file, and exit.

After specifying the input parameters and invoking the action button 'specify fault

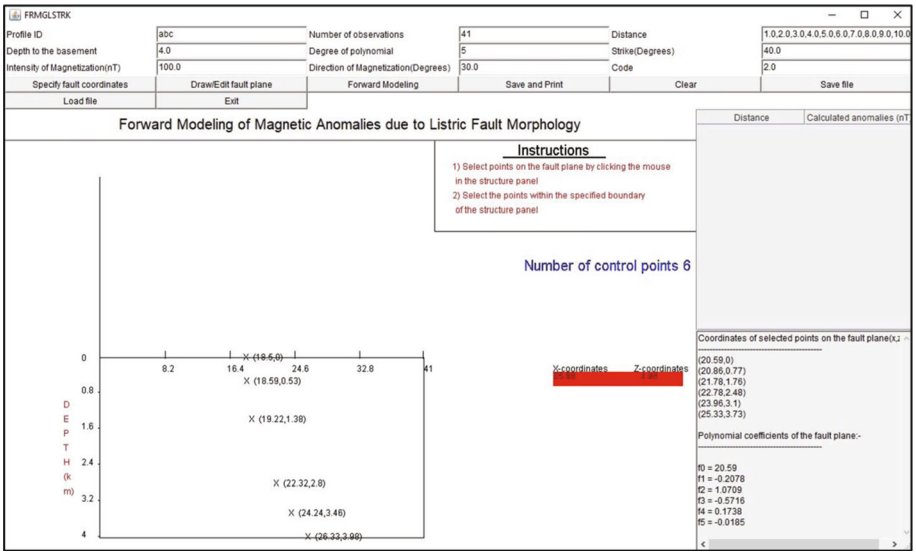


Figure 7. The number of control points (six) appear in blue indicates the selection of sufficient control points in the structure panel.

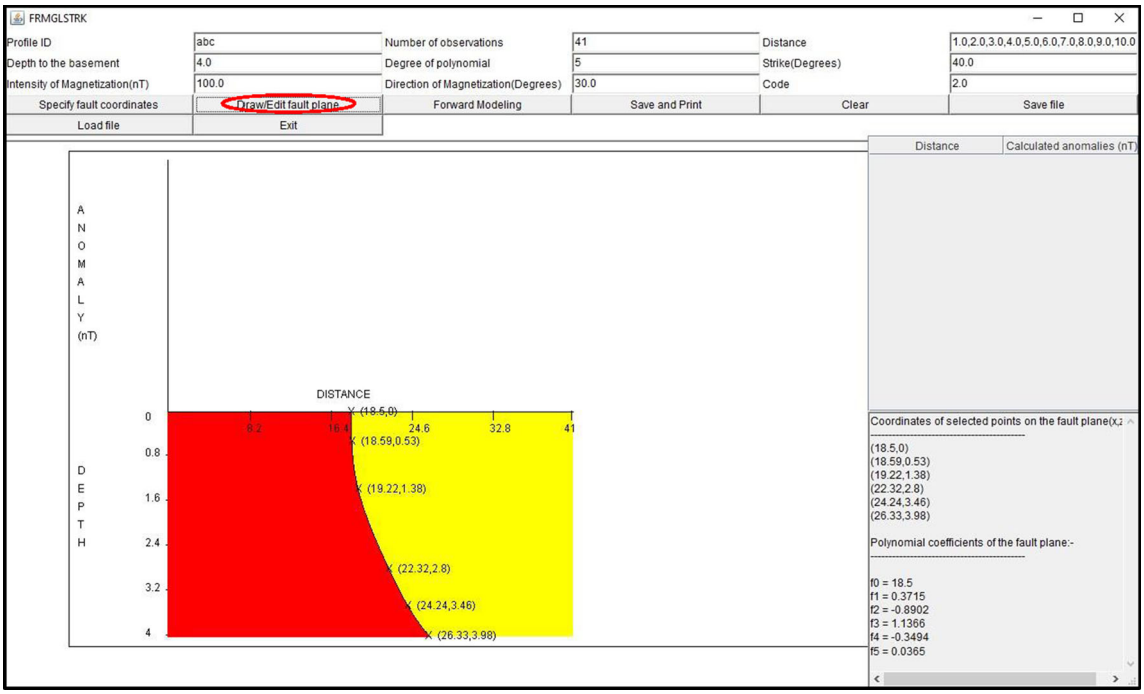


Figure 8. Analytically defined fault plane by a 5th degree polynomial. The footwall is represented with solid red and the hanging wall in yellow. The estimated coefficients of the polynomial and the coordinates of six control points are displayed in the lower panel of ASCII layout.

coordinates', the user selects a few control points in the structure panel by means of mouse clicks to construct a fault plane. The code automatically assigns the coordinates (x, z) to each such selected point in the structure panel, and the same is displayed on the right-hand side of the structure panel. The number of selected control points are also displayed in the graphical layout, as shown in figure 6. In addition, the code guides the user to select the optimum number of control points in the structure panel to construct the fault plane. For example, if the number of control points to describe the fault plane is insufficient (depending upon the degree of chosen polynomial), then the number of selected control points is displayed in red (as shown in figure 6). In such a case, the user needs to select a few more points in the structure panel till the font colour turns to blue (figure 7).

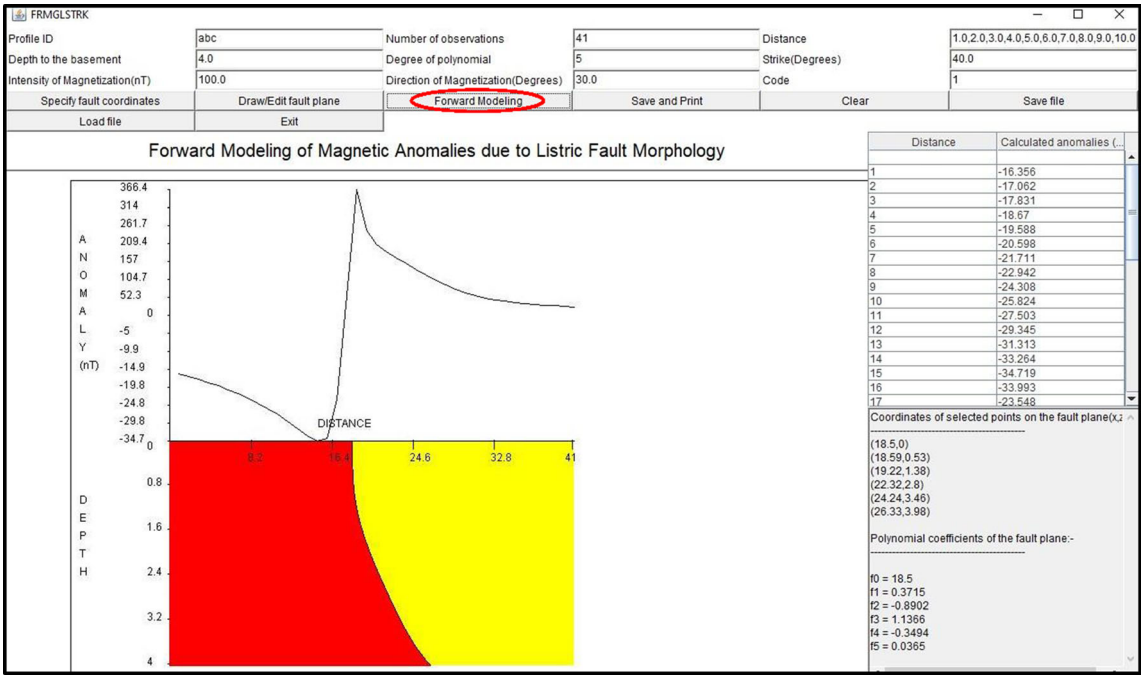
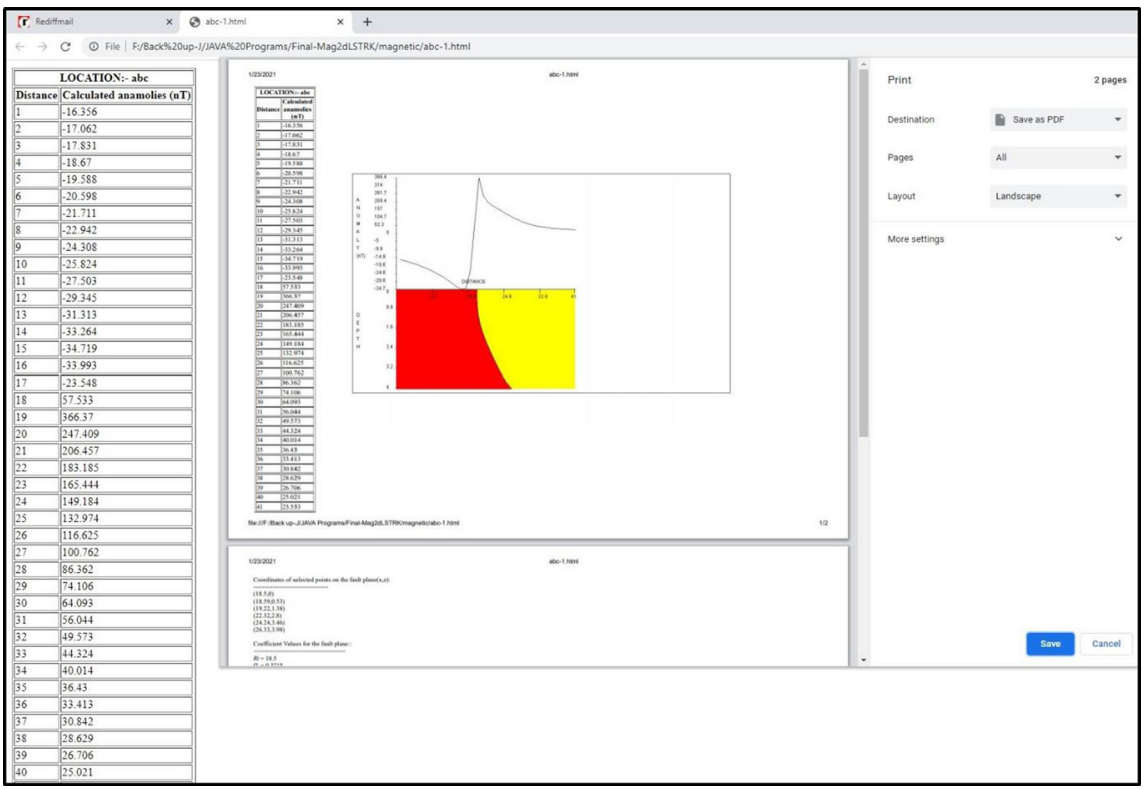


Figure 9. Vertical magnetic anomaly over a listric fault morphology. The non-planar fault plane is described with a 5th degree polynomial. The magnitude of anomaly at each observation is displayed in the top panel of ASCII layout.



LOCATION:- abc	
Distance	Calculated anamolies (nT)
1	-16.356
2	-17.062
3	-17.831
4	-18.67
5	-19.588
6	-20.598
7	-21.711
8	-22.942
9	-24.308
10	-25.824
11	-27.503
12	-29.345
13	-31.313
14	-33.264
15	-34.719
16	-33.993
17	-23.548
18	57.533
19	366.37
20	247.409
21	206.457
22	183.185
23	165.444
24	149.184
25	132.974
26	116.625
27	100.762
28	86.362
29	74.106
30	64.093
31	56.044
32	49.573
33	44.324
34	40.014
35	36.43
36	33.413
37	30.842
38	28.629
39	26.706
40	25.021

Figure 10. Output of forward modelling in html format with a print dialogue box attached to it.

Upon invoking the 'draw/edit fault plane' action button in the input layout, the code solves the polynomial coefficients, f_k, by fitting the prescribed polynomial to the coordinates of control points. These estimated coefficients are then used to construct an analytically defined fault plane, as shown

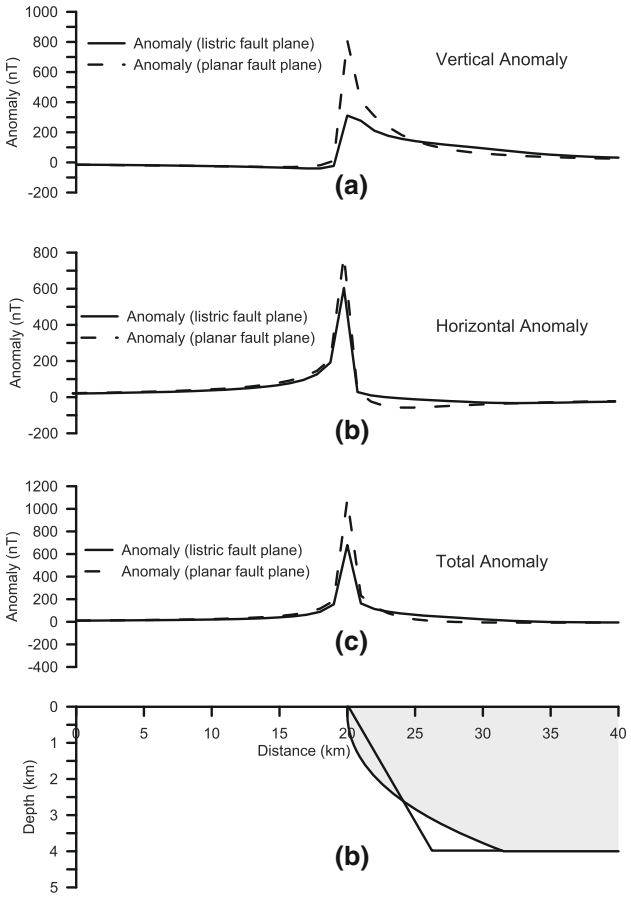


Figure 11. **(a)** Vertical, horizontal, and total magnetic anomalies obtained from the present method over a listric fault source (fault plane is described with a 5th degree polynomial) whose geometry is shown in **(d)**. The magnetic anomalies in the three components calculated over the same structure by an analytic equation (Murthy *et al.* 2001) presuming a planar fault plane are also shown for comparison.

Table 1. *Coefficients of 5th degree polynomial.*

Coefficient	Magnitude
a_0	20.014
a_1	–0.1479
a_2	0.4836
a_3	0.0711
a_4	–0.0023
a_5	0.00039

in figure 8. If the user opts for modifying the fault plane, then the control points in the structure panel are edited by simple drag and drop mouse operations. Accordingly, the coefficients of the polynomial get updated, and the fault plane is reconstructed and displayed in real-time. The user also has the option to change the degree of the polynomial, if required.

Once the model space is constructed, the anomalous field in any specific component (by specifying code number 1 for vertical, 2 for horizontal, and 3 for total) can be realized by invoking the 'forward modeling' operator (figure 9). The business logic computes the magnetic anomalies in the required component and displays the response in the anomaly panel, as shown in figure 9. The computed anomalies are also displayed in a tabular form in the ASCII layout (figure 9). The user saves the output by invoking the 'save and print' action button (figure 10).

4. Example

The applicability of method and code is demonstrated on a synthetic listric fault model, whose geometry is shown in figure 11(d). The assumed model space remains the same as in figure 2(b), but in this case, a 5th degree polynomial is used to describe the listric (non-planar) fault plane, as shown in figure 11(d). The coefficients of the chosen polynomial (5th degree) are given in table 1. For such a structure, the magnetic anomalies in vertical, horizontal, and total components are calculated from the present method and compared to the anomalies obtained from the analytic equation (Murthy *et al.* 2001), which presumes the planar surface for the fault plane. In both cases, the observer is on the top of the topography at $z_j = 0$ km. The theoretical anomalies obtained from both methods at 41 equi-spaced observations on a profile in the interval $x_j \epsilon (0, 40\,\text{km})$ are shown in figure 11(a–c). It is clearly seen from figure 11(a–c) that the magnetic anomalies produced by the listric fault structure differ in magnitude from the corresponding anomalies realized from the analytic equation. Therefore, the assumption of a planar surface for a fault plane should be accepted with caution, particularly when interpreting the magnetic anomalies caused by large normal faults.

5. Conclusions

A generalized equation that combines both analytic and numeric approaches to compute the magnetic anomalies due to an arbitrarily magnetized 2D listric fault source in any component is presented. It is demonstrated with a synthetic example that the magnitude of anomalies produced by a listric fault source, in any component, portray lesser magnitude than the anomalies produced by

the same structure with a planar fault surface. Therefore, the application of routine modelling and inversion algorithms that consider the fault planes as planar surfaces to analyse the magnetic anomalies of large normal faults is discouraged.

Acknowledgements

The authors sincerely thank the reviewers and Associate Editor for their suggestions and feedback. Ramamma Batta thank DST, Government of India for granting the Women Scientist scheme.

Author statement

Ani Nibisha has written the manuscript. Ramamma and Rajeswara Sastry have programmed the algorithm. Chakravarthi has developed the algorithm.

References

Asfahani J and Tlas M 2007 A robust nonlinear inversion for the interpretation of magnetic anomalies caused by faults, thin dikes and spheres like structure using stochastic algorithms; *Pure Appl. Geophys.* **164** 2023–2042.

Asfahani J and Tlas M 2004 Nonlinearly constrained optimization theory to interpret magnetic anomalies due to vertical faults and thin dikes; *Pure Appl. Geophys.* **161(1)** 203–219.

Chakravarthi V 2011 Automatic gravity optimization of strike listric fault sources with analytically defined fault planes and depth dependent density; *Geophysics* **76** I21–I31.

Chakravarthi V 2010 Gravity anomalies of 2D fault structures with fault planes described by polynomial functions of arbitrary degree; *Curr. Sci.* **99** 654–656.

Doo W B, Hsu S K and Yeh Y C 2007 A derivative-based interpretation approach to estimating source parameters of simple 2D magnetic sources from Euler deconvolution, the analytic-signal method and analytical expressions of the anomalies; *Geophys. Prospect.* **55** 255–264.

Goussev S, Charters R and Peirce J 2006 Mackenzie Delta: A case of one residual gravity anomaly and 16 dry exploration wells. 2006 CSPG/CSEG/CWLS Joint Conference 15–18 May Calgary Canada, http://www.searchanddiscovery.com/documents/2015/10726goussev/ndx_goussev.pdf.

Grant F S and West G F 1965 *Interpretation Theory in Applied Geophysics*, McGraw-Hili Book Co.

Jackson J A 1987 Active normal faulting and crustal extension; *Geol. Soc. London Spec. Publ.* **28** 3–17.

McKenzie D 1978 Some remarks on the development of sedimentary basins; *Earth Planet. Sci. Lett.* **40(1)** 25–32.

Marquardt D W 1963 An algorithm for least-square optimization of non-linear parameters; *J. Soc. Ind. Appl. Math.* **11** 431–441.

Moo J K C 1965 Analytical aeromagnetic interpretation – The inclined prism; *Geophys. Prospect.* **13** 203–224.

Murthy I V R 1985 The mid-point method: Magnetic interpretation of dykes and faults; *Geophysics* **50(5)** 834–839.

Murthy I V R 1998 Gravity and magnetic interpretation in exploration geophysics; *Geol. Soc. India Memoir* **40**.

Murthy I V R, Swamy K V and Rao S J 2001 Automatic inversion of magnetic anomalies of faults; *Comput. Geosci.* **27** 315–325.

Mushayandebvu M F, van Driel P, Reid A B and Fairhead J D 2001 Magnetic source parameters of two-dimensional structures using extended Euler deconvolution; *Geophysics* **66(3)** 814–823.

Qureshy I R and Nalaye A M 1978 A method for direct interpretation of magnetic anomalies caused by two-dimensional vertical faults; *Geophysics* **43** 179–199.

Rao B S R and Murthy I V R 1978 *Gravity and Magnetic Methods of Prospecting,* Arnold Heinemann Publishers (Pvt.) Ltd.

Rao D A and Babu H V R 1983 Standard curves for the interpretation of magnetic anomalies over vertical faults; *Geophys. Res. Bull.* **21(1)** 71.

Rao D A, Babu H V R and Narayan P V S 1980 Relationship of magnetic anomalies due to subsurface features and the interpretation of sloping contacts; *Geophysics* **45(1)** 32–36.

Sengupta S 1974 Fourier transforms of magnetic anomalies of two-dimensional Bodies; *Pure Appl. Geophys.* **112(6)** 987–995.

Smith R B and Bruhn R L 1984 Intraplate extensional tectonics of the Eastern Basin-Range: Inferences on structural style from seismic reflection data regional tectonics and thermal-mechanical models of brittle–ductile deformation; *J. Geophys. Res.* **89** 5733–5762.

Stanley J M 1977 Simplified magnetic interpretation of the geologic contact and thin dike; *Geophysics* **42(6)** 1236–1240.

Subrahmanyam M and Rao T K S P 2009 Interpretation of magnetic anomalies using some simple characteristic positions over tabular bodies; *Expl. Geophys.* **40** 265–276.

Torizin J G, Jentzsch P, Malischewsky J, Abakanov Kley N and Kurskeev A 2009 Rating of seismicity and reconstruction of the fault geometries in northern Tien Shan within the project 'Seismic Hazard Assessment for Almaty'; *J. Geodyn.* **48(3–5)** 269–278.

Corresponding editor: Arkoprovo Biswas

J. Earth Syst. Sci. (2022)131 265
https://doi.org/10.1007/s12040-022-02012-x

Automatic inversion of magnetic anomalies caused by 2D listric fault sources with arbitrary magnetisation

V Ani Nibisha*, B Ramamma and V Chakravarthi

Centre for Earth, Ocean and Atmospheric Sciences, University of Hyderabad, Hyderabad 500 046, India.
**Corresponding author. e-mail: aninibisha@gmail.com*

MS received 25 February 2022; revised 10 August 2022; accepted 11 August 2022

An automatic inversion technique is presented to analyse the magnetic anomalies produced by 2D fault sources having nonplanar fault planes. This technique provides a means to analyse the magnetic anomalies of the structure measured in any component (i.e., vertical, horizontal or total). The present technique uses the polynomial functions to describe the fault plane, the coefficients of which become the unknown parameters to be estimated from the magnetic anomalies along with the other shape parameters, namely, the depths to the top and bottom of the fault structure, the location of the fault edge, besides intensity and direction of magnetisation. This technique initialises the model space based on some characteristic anomalies and their positions on the anomaly profile and subsequently updates them iteratively following the predefined convergence conditions. The closeness of fit between the recovered and assumed theoretical models of a fault structure from the analysis of noise-added vertical magnetic anomalies justifies the strength and applicability of the proposed technique. The observed total magnetic anomalies along an EW profile across the western margin of the Perth Basin, Australia are interpreted and the results are compared with the available/reported information.

Keywords. 2D listric fault; arbitrary magnetisation; magnetic anomaly; automatic inversion.

1. Introduction

Faults that are originated as a result of rifting, drifting, and collapsed orogeny are often listric in nature rather than planar. Also, thick sedimentary basins are commonly delimited on their margins by faults that are curved in cross-section (Jackson 1987; Middleton *et al.* 1993). These fault planes are characterised by steep dips near the surface/topography and with increasing depths the fault dips progressively decrease (Shelton 1984; Spahić *et al.* 2011; Zhao *et al.* 2020).

Though fault planes of large normal faults are often nonplanar, many interpretation techniques have been developed considering the fault planes as planar surfaces to analyse the potential field anomalies resulted from them (see for e.g., Sundararajan *et al.* 1983; Rao 1985; Murthy *et al.* 1980, 2001; Rao and Babu 1983; Raju *et al.* 1998; Murthy 1998; Abdelrahman *et al.* 2003; Chakravarthi and Sundararajan 2004; Essa 2013; Abdelrahman and Essa 2015; Anderson *et al.* 2020; Elhussein 2021; Essa *et al.* 2021; Essa 2021; Rao and Biswas 2021). In recent past, techniques are also reported to calculate the gravity and magnetic responses of listric fault sources. For example, Chakravarthi (2010a, b) presented a set of formulae to compute the gravity responses of 2D and 2.5D fault structures, where analytic functions were used

Published online: 16 December 2022

to simulate the geometries of fault planes. A few inversion schemes to quantify listric fault sources from measured gravity anomalies are also in vogue. Chakravarthi (2011) proposed a technique using the Marquardt's (1970) algorithm to contemporarily infer the shape parameters of 2.5D listric fault structures and regional gravity field from the measured gravity anomalies. This technique is efficient to model the listric fault morphologies, where the disconnected hanging wall comprises in thick-sectioned sediments with progressive increase in density with increasing depths. Another optimisation proposed by Chakravarthi and Pramod Kumar (2015) simultaneously estimates from the gravity anomalies the fault plane geometries and densities of several lithologic units or their thicknesses within the hanging wall. Interpretation of gravity anomalies caused by 2D listric fault sources among which the density contrast follows the exponential decay was also reported (Chakravarthi *et al.* 2017). Roy *et al.* (2022) have used the particle swarm optimisation (PSO) to interpret the gravity anomalies of listric faults by simulating the fault planes with quadratic Bezier curve. In recent past, Nibisha *et al.* (2021) have developed a unified technique to compute the magnetic anomalies due to the listric fault sources in any component.

It is pertinent to note that not many techniques are developed in magnetics for fault model interpretation because the unknown model parameters to be solved from the magnetic anomalies are generally large in number, particularly when the direction of magnetisation is presumed as arbitrary (Murthy *et al.* 2001). Making use of the symmetry of the analytic function representing the magnetic anomalies of dyke and vertical fault models, Powell (1967) devised a method to interpret the magnetic anomalies of fault structures, whereas Sengupta (1974) used the Fourier integral to analyse the anomalies. Following the formulation of Koulomzine *et al.* (1970), Qureshi and Nalaye (1978) proposed the decomposition of the magnetic anomaly profile into even and odd components for fault model interpretation. Green (1979) developed a harmonic method that uses the locations of two turning points and two inflection points on the magnetic gradient profile to interpret the anomalies caused by a contact. Murthy (1985) used the properties of midpoints between the maximum and minimum anomalies on the profiles at two selected heights, the distances between the said anomalies, and the ratio of horizontal to vertical gradients of the anomalies to decipher arbitrarily magnetised fault structures. Murthy *et al.* (2001) applied the Ridge Regression algorithm in an inversion to quantify the magnetic anomalies caused by fault sources in any component. This technique initialises the fault structure using the maximum and minimum anomalies and their positions on the anomaly profile. These initial model parameters are then refined iteratively till an acceptable predefined convergence is achieved between the observed and modelled anomalies. Stavrev (2006) had proposed the concept of magnitude transform that makes use of both magnetic field components and the first-order derivatives to invert magnetic anomalies of simple geometric shapes. Subrahmanyam and Rao (2009) proposed a method to infer fault parameters using characteristic points on the magnetic profile. An inversion method based on modular neural network was proposed by Al-Garni (2016) to decipher fault parameters. Essa and Elhussein (2018) employed the Particle Swarm Optimization (PSO) in the analysis of magnetic anomalies of simple geometric shapes. Ekinci *et al.* (2019) applied both differential evolution algorithm (DE) and particle swarm optimisation (PSO) on the gravity and magnetic anomalies caused by deep-seated faults and reported that the DE is more efficient than PSO in terms of convergence rate, model space recovery, and robustness. Recently, Biswas and Rao (2021) have used simulated annealing (SA) to interpret the anomalies due to fault and sheet-type models. However, the intrinsic assumption that the fault planes are either vertical or planar in the aforementioned methods limit their utility to analyse the anomalies caused by fault sources that display listric behaviour.

Therefore, it is necessary to design a new inversion strategy to determine the parameters of listric fault structures from the measured magnetic anomalies. Here, we present a generalised inversion technique, employing the Marquardt's (1970) algorithm, to quantify the listric fault sources from the magnetic anomalies observed in any component, viz., vertical, horizontal or total field. We presume that the causative source responsible for generating the anomalies is magnetised along the resultant of induced and remanent magnetic vectors. The present inversion utilises the forward modelling scheme of Nibisha *et al.* (2021). The proposed technique is applied on the magnetic anomalies of a synthetic listric fault structure in the presence of pseudorandom noise, followed by its application to analyse the real field anomalies across the western margin of the Perth Basin, Australia.

2. Forward modelling – theoretical considerations

Figure 1 depicts a schematic representation of a 2D listric fault geometry, in which the footwall remains undistorted, whereas the hanging wall is displaced downwards. The source parameters of the structure are defined in the Cartesian coordinate system in which the x-axis is placed transverse to the strike of the fault source and the z-axis vertically downwards. The strike of the source is presumed along the y-axis perpendicular to the xz plane. The structure is bounded by a function, $f(z) = \sum_{i=0}^{n} f_i z^i$, on one side and on the other it extends to infinity (Nibisha *et al.* 2021). Here, n is the degree of a prescribed polynomial, and $f_i, i = 0, 1, 2, \ldots, n$ are the unknown coefficients of the polynomial to be estimated from the inversion. The vertical $(\Delta V(x_j, z_j))$, and horizontal $(\Delta H(x_j, z_j))$ magnetic anomalies due to such a source at any observation, (x_j, z_j), can be expressed as (Nibisha *et al.* 2021)

$$\Delta V(x_j, z_j) = 2J_{ef} \times \int_{z=z_T}^{z_B} \frac{(z - z_j)\cos\theta_{ef} - (f(z) - x_j)\sin\theta_{ef}}{(f(z) - x_j)^2 + (z - z_j)^2}\, dz, \quad (1)$$

$$\Delta H(x_j, z_j) = 2J_{ef}\sin\vartheta \int_{z=z_T}^{z_B} \frac{(z - z_j)\cos(\theta_{ef} - \frac{\pi}{2}) - (f(z) - x_j)\sin(\theta_{ef} - \frac{\pi}{2})}{(f(z) - x_j)^2 + (z - z_j)^2}\, dz, \quad (2)$$

where

$$J_{ef} = J\sqrt{(1 - \cos^2\theta\cos^2\delta)},$$

and

$$\theta_{ef} = \tan^{-1}(\tan\theta\,\mathrm{cosec}\,\delta).$$

Here, J represents the magnitude of the resultant magnetic vector of both induced and remanent magnetisations, and θ represents its dip. Also, J_{ef} and θ_{ef} refer to the magnitude and dip of the effective magnetisation vector that lies in an upright plane perpendicular to the strike of the source. Equations (1 and 2) reveal that the vertical and horizontal magnetic anomalies of the structure differ from each other in terms of amplitude and phase. The expression for magnetic anomaly in any component can be expressed as (Nibisha *et al.* 2021):

$$\Delta T(x_j, z_j) = 2J' \times \int_{z=z_T}^{z_B} \frac{(z - z_j)\cos\theta' - (f(z) - x_j)\sin\theta'}{(f(z) - x_j)^2 + (z - z_j)^2}\, dz. \quad (3)$$

Here, $J' = J_{ef}\sqrt{(1 - \cos^2\vartheta\cos^2\alpha)}$, $\theta' = \theta_{ef} - \tan^{-1}(\sin\vartheta/\tan\alpha)$. ϑ is the strike of the source measured with reference to the magnetic north either due east or west. It is to realise that upon substituting $\alpha = 90°$ in equation (3), the vertical magnetic anomaly of the structure can be calculated. Similarly, by setting α to 0° and i (dip of earth's magnetic field) in equation (3), the anomalies in horizontal and total field can be realised, respectively.

3. Inversion of magnetic anomalies

In inversion, theoretical anomalies that arise from a starting model are fitted to the observed anomalies and based on the misfit between the observed and theoretical anomalies, the model parameters of the space are adjusted iteratively within the permissible limits, so that the optimum

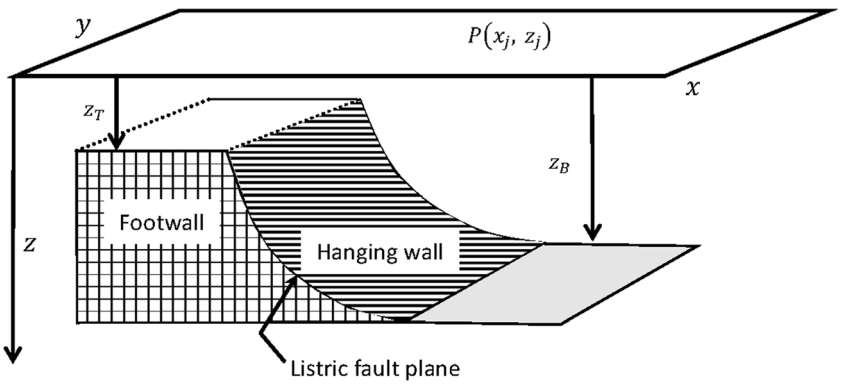


Figure 1. Schematic diagram showing the geometry of a 2D listric fault source.

model becomes geologically acceptable. In the present case, the size factor, $2J'$, in equation (3) is independent of the source body parameters, hence, the magnitude of J_{ef} can be determined from the size factor, in principle. On the other hand, the shape of the anomaly profile in any component over the structure with a known strike is influenced by the body parameters, viz., z_T (depth to the top of the fault plane), z_B (depth to the bottom of the fault plane), coefficients of the function, $f(z)$, $f_i, i = 0, 1, 2, \ldots, n$, and θ_{ef}. In other words, the body parameters of the structure and θ_{ef} can be estimated from the shape of the anomaly profile.

In general, the process of any geophysical inversion starts by initialising a model space with a set of approximate shape parameters coupled with appropriate physical property contrast/s. Usually, these initial parameters are specified by the interpreter. At times, the starting model is described with a simple geometry, thereby enabling the interpreter to instruct the computer to identify some characteristic anomalies and their locations on the profile, which are then used to initialise the source. In the proposed inversion, approximate/initial body parameters are obtained by treating the fault source as a vertical step (Murthy *et al.* 2001). Such an approximation was also proposed by Chakravarthi (2011) to analyse the gravity anomalies resulted from strike-limited listric fault sources. Following Murthy *et al.* (2001), for a vertical step, the approximate distance to the top of the fault edge, D_{App}, on the profile can be found at an observation, where the magnitude of the anomalous field equals the sum of the minimum ($\Delta T_{\min}$) and maximum ($\Delta T_{\max}$) anomalies. In case the anomaly profile is symmetric about the anomaly axis with a single turning point, then the observer location corresponding to the maximum anomaly is assigned to D_{App}. The value of θ' is obtained from Murthy *et al.* (2001)

$$\left.\begin{aligned} \theta' &= \arctan \frac{2\sqrt{\dfrac{\Delta T_{\min}}{\Delta T_{\max}}}}{\left(1 - \dfrac{\Delta T_{\min}}{\Delta T_{\max}}\right)}, \\ &\text{for } 0.05 \le \frac{\Delta T_{\min}}{\Delta T_{\max}} \le 0.55 \\ \theta' &= \frac{\pi}{2}, \quad \text{for } \frac{\Delta T_{\min}}{\Delta T_{\max}} > 0.55 \\ \theta' &= 0^{\circ}, \quad \text{for } \frac{\Delta T_{\min}}{\Delta T_{\max}} < 0.05. \end{aligned}\right\} \tag{4}$$

The value of θ' calculated from equation (4) is placed in the appropriate quadrant based on the plus–minus method (Murthy *et al.* 2001). Equation (4) conveys that θ' is dependent only on the magnitudes of minimum and maximum anomalies. Also, the approximate depth to the fault edge is obtained as:

$$\left.\begin{aligned} z_{T\text{App}} &= \frac{|X_{\max} - X_{\min}| \sin\theta'}{2\sqrt{9 - 4\sin^2\theta'}}, \\ &\text{for } 0.05 \le \frac{\Delta T_{\min}}{\Delta T_{\max}} \le 0.55 \text{ or } \frac{\Delta T_{\min}}{\Delta T_{\max}} > 0.55 \\ z_{T\text{App}} &= 0.224XW, \quad \text{for } \frac{\Delta T_{\min}}{\Delta T_{\max}} < 0.05, \end{aligned}\right\} \tag{5}$$

where W is the distance between the two half-maximum anomalies on the profile. Depth to the bottom of the fault plane is initialised as:

$$z_{B\text{App}} = 8Xz_{T\text{App}}. \tag{6}$$

A ratio of 8 for $\frac{Z_{B\text{App}}}{Z_{T\text{App}}}$ is found satisfactory to invert the magnetic anomalies of listric fault sources with $2 \le \frac{Z_B}{Z_T} \le 15$.

It is to note that for a vertical step, the coefficients, $f_i, i = 1, 2, \ldots, n$ become zero. Hence, the magnetic anomaly, $\Delta T(x_j)$, in equation (3) takes the form

$$\begin{aligned} \Delta T(x_j, z_j) = C_1 \int_{z=z_T}^{z_B} \frac{(z - z_j)}{(f_0 - x_j)^2 + (z - z_j)^2}\, dz \\ - C_2 \int_{z=z_T}^{z_B} \frac{(f_0 - x_j)}{(f_0 - x_j)^2 + (z - z_j)^2}\, dz. \end{aligned} \tag{7}$$

where $C_1 = 2J'\cos\theta'$ and $C_2 = 2J'\sin\theta'$.

As many linear equations (similar to equation 7) are framed as the number of observations, and two normal equations are constructed and solved for C_1 and C_2. From the estimated values of C_1 and C_2, the value of J' can be obtained as:

$$J' = \frac{\sqrt{C_1^2 + C_2^2}}{2}. \tag{8}$$

Equation (3) computes the theoretical anomalies of the model space at all observations, x_j, j = 1, 2,.., N_{obs}. In this process, D_{App} value is initially assigned to f_0, and all other coefficients are set to zero (i.e., $f_i = 0$, i = 1, 2, …, N). The theoretical

anomalies, $\Delta T_{\text{cal}}(x_j, z_j)$, of the initial structure obtained from equation (3) deviates from the observed anomalies, $\Delta T_{\text{obs}}(x_j, z_j)$. The misfit between the two anomalies can be quantified as:

$$\widehat{J} = \sqrt{\sum_{j=1}^{N_{\text{obs}}} \frac{\left[\Delta T_{\text{obs}}(x_j, z_j) - \Delta T_{\text{cal}}(x_j, z_j)\right]^2}{N_{\text{obs}}}}. \quad (9)$$

The deviation of theoretical anomalies from the observed ones can be written using Tylor series approximation as:

$$\begin{aligned}
&\Delta T_{\text{obs}}(x_j, z_j) - \Delta T_{\text{cal}}(x_j, z_j) \\
&= \frac{\Delta T(x_j, z_j)}{\partial z_T} dz_T + \frac{\Delta T(x_j, z_j)}{\partial z_B} dz_B \\
&+ \frac{\Delta T(x_j, z_j)}{\partial f_0} df_0 + \frac{\Delta T(x_j, z_j)}{\partial f_1} df_1 \\
&+ \frac{\Delta T(x_j, z_j)}{\partial f_2} df_2 + \cdots + \frac{\Delta T(x_j, z_j)}{\partial f_n} df_n.
\end{aligned} \quad (10)$$

A total of N_{obs} linear equations in number are framed from which $(n+3)$ normal equations are built and solved for $(n+3)$ unknown parameters using the Marquardt algorithm (Marquardt 1970; Chakravarthi 2003). The normal equations can be stated as:

$$\begin{aligned}
&\sum_{i=1}^{N_{\text{obs}}} \sum_{m=1}^{n+3} \frac{\Delta T(x_i, z_i)}{\partial p_k} \frac{\Delta T(x_i, z_i)}{\partial p_m} (1 + \xi\lambda) dp_k \\
&= \sum_{i=1}^{N_{\text{obs}}} \text{Err}(x_i, z_i) \frac{\Delta T(x_i, z_i)}{\partial p_k}, k = 1, 2, \ldots, n+3.
\end{aligned} \quad (11)$$

Here λ is the damping factor and $\text{Err}(x_i, z_i) = [\Delta T_{\text{obs}}(x_i, z_i) - \Delta T_{\text{cal}}(x_i, z_i)], i = 1, 2, \ldots, N_{\text{obs}}$. Also, $\xi = 1$ for $m = k$, otherwise 0. Equation (11) can be put in a matrix form as:

$$(A + \lambda I)X = B, \quad (12)$$

where the elements, A_{ik}, of matrix A are expressed as:

$$\left.\begin{aligned}
A_{ik} &= \sum_{i=1}^{N_{\text{obs}}} \sum_{m=1}^{n+3} \frac{\Delta T(x_i, z_i)}{\partial p_k} \frac{\Delta T(x_i, z_i)}{\partial p_m}, \\
&k = 1, 2, \ldots, n+3, \\
B &= \sum_{i=1}^{N_{\text{obs}}} \text{Err}(x_i, z_i) \frac{\Delta T(x_i, z_i)}{\partial p_k}, \\
&k = 1, 2, \ldots, n+3, \\
X &= dp_k,\ k = 1, 2, \ldots, n+3.
\end{aligned}\right\} \quad (13)$$

Here I is the diagonal matrix containing the elements of A. Also, dp_k, $k = 1, 2, \ldots, n+3$ are the estimated improvements in the model parameters, p_k. Here, $dp_1 = dz_T$, $dp_2 = dz_B$, $dp_3 = df_0$, $dp_4 = df_1, \ldots, dp_{(n+3)} = df_{(n+3)}$. Equation (12) conveys that the diagonal elements of matrix A are multiplied by $(1+\lambda)$. Further, it is to note that the polynomial degree, n, should be chosen in such a way that the total number of unknowns does not exceed the number of observations. The partial derivatives of the anomaly in equation (13) are obtained by numerical differentiation (Chakravarthi *et al.* 2001).

To start with, the initial data misfit, $\widehat{J_A}$, is obtained from equation (9). The damping factor, λ, is set to 0.5 (Chakravarthi 2003) and equation (12) is worked out for $dp_k, k = 1, 2, \ldots, n+3$. The existing source parameters, P_k, $k = 1, 2, \ldots, n+3$ are updated to P_M, $M = 1, 2, \ldots, n+3$ by adding/subtracting $dp_k, k = 1, 2, \ldots, n+3$ to P_k. The theoretical response of the improved model is again calculated, and the corresponding misfit, $\widehat{J_N}$, is estimated. If the magnitude of the new misfit, $\widehat{J_N}$, falls below its preceding value, $\widehat{J_A}$, then λ is reduced by a factor of 2. In such a case, $\widehat{J_N}$ is assigned to $\widehat{J_A}$ and P_M to P_k and the process repeats iteratively. By any chance if the misfit at the end of any specific iteration exceeds its preceding value, then the existing value of λ is multiplied by 2, and equation (12) is worked out again for the values of dp_k. This process continues within the specific iteration till the resulting misfit falls below its preceding value. The inversion process automatically terminates upon the completion of a specific number of iterations, or the prevailing misfit at the end of any iteration equals to, or less than the prescribed threshold, or the current value of λ is abnormally large (Chakravarthi 2003; Ramamma *et al.* 2021).

4. Examples

To justify the relevance of the proposed method of inversion, two magnetic anomaly profiles were interpreted. In the first case, the vertical magnetic anomalies due to a synthetic structure are analysed. In the latter case, the total magnetic anomalies attributable to a deep-seated fault from the western margin of the Perth Basin, Australia, are interpreted.

4.1 *Synthetic example*

Figure 2(b) describes the model geometry of an assumed listric fault structure in the *xz*-plane. The fault originates near the topography at the 20th km and extends downwards with decreasing dips to a maximum depth of 4 km (figure 2b). A fifth-degree polynomial, whose coefficients are given in table 1, describes the geometry of the fault plane. Assuming the intensity and direction of magnetisation as 100 gammas and 30° respectively, the structure anomalies are calculated in the observation range, 0–40 km, on the profile to which pseudorandom noise (Gaussian) is added and shown in figure 2(a). The noise has zero mean and a standard deviation of 8.82 nT. We treat these noisy anomalies as the observed anomalies in the inversion to recover the structure by setting 4.1 nT as the acceptable threshold for the misfit. The algorithm executed five iterations for inversion and then terminated. It was found from the inversion result that the misfit had decayed rapidly within first four iterations, from 40.6 to 8.32 gammas, thence gradually before it attained a value <4.1 gammas at the end of the 5th iteration. The estimated values of intensity of magnetisation and direction of magnetisation from the inversion were 99.8 gammas and 31°, respectively. The recalculated theoretical anomalies of the estimated model after the 5th iteration closely fit the observed anomalies and the estimated structure exactly mimics the assumed one (this case is not shown for brevity). The determined polynomial coefficients after inversion precisely coincide with the assumed ones (table 1).

In reality, it is always tricky to select the best polynomial to describe the fault plane geometry if additional information on the subsurface is either scanty or unavailable. Therefore, it is necessary to examine whether or not the inversion technique could recover the structure, if a polynomial degree other than the optimum is used in the inversion.

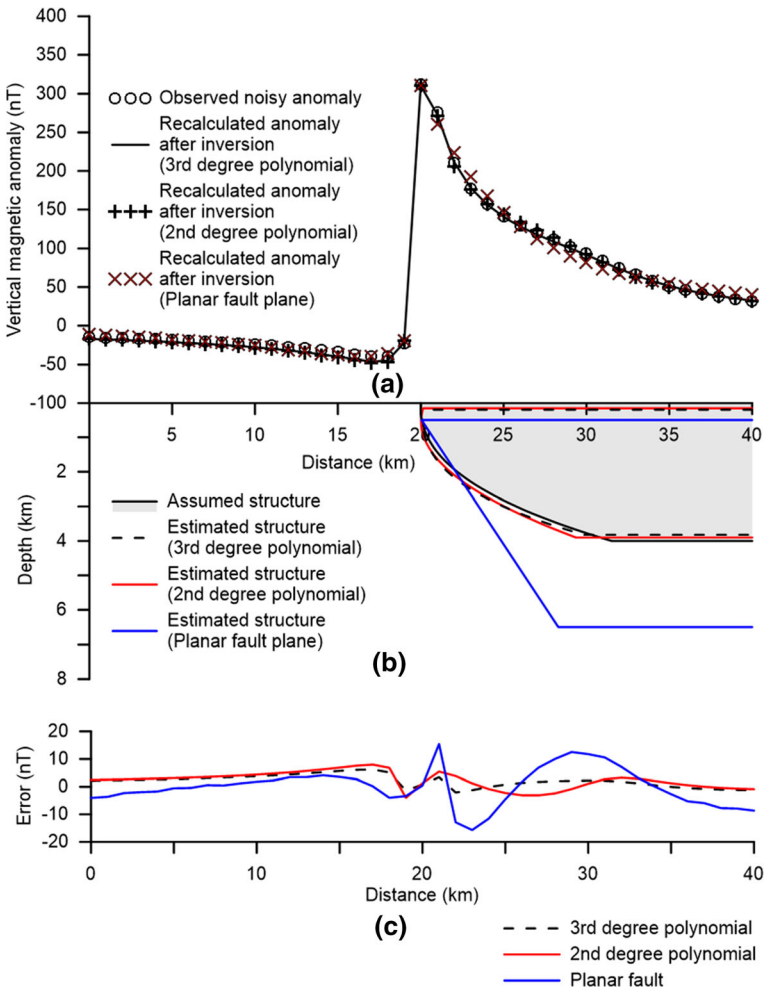


Figure 2. **(a)** Observed and recalculated anomalies after inversion, **(b)** assumed and estimated structures after inversion, and **(c)** error plot showing the residuals between the observed and recalculated anomalies after inversion.

Table 1. *Assumed and estimated coefficients of polynomials (synthetic example) and estimated coefficients (real field example).*

Degree of polynomial	Assumed coefficients						Estimated coefficients after inversion					
	f_0	f_1	f_2	f_3	f_4	f_5	f_0	f_1	f_2	f_3	f_4	f_5
Synthetic example												
5	20.014	–0.1479	0.4836	0.0711	–0.0023	0.0004	20.014	–0.1479	0.4836	0.0711	–0.0023	0.0004
3							19.9248	0.2435	–0.0125	0.1666		
2							20.3011	–1.025	0.8824			
Field example (*western margin of the Perth Basin, Australia*)												
2							10.5274	1.5683	–0.0265			

For this, we repeat the inversion using a 2nd degree and a 3rd degree polynomial (in place of a 5th degree) to simulate the fault surface. Here also we kept the tolerable misfit between the observed and modelled anomalies at 4.1 gammas.

For 2nd-degree polynomial approximation of the fault plane, the technique had performed 10 iterations and for 3rd degree, 6 iterations, respectively. In both cases, the algorithm had identified the origin of the fault plane at 19.99 km and placed the top edge of the fault plane at 0.29 km depth and the bottom at 2.39 km, respectively. The estimated intensity of magnetisation, in either case, was 99.4 gammas and the direction of magnetisation was 35°. The initial model parameters were updated in iterative mode along with the other unknowns (coefficients of the function, $f(z)$). After the inversion, the misfit calculated through equation (9) in either case was less than the predefined threshold. Though the initial misfit for the starting model was 40.61 gammas, in either case, its decay with iteration was more rapid with the 3rd degree polynomial approximation of the fault plane when compared to the 2nd degree (figure 3). The modelled anomalies after the inversion in both cases were shown in figure 2(a) along with the observed ones. Figure 2(b) shows the corresponding estimated structures. The maximum difference between the observed and theoretical anomalies corresponding to the optimum structures was found within ±8 gammas (figure 2c). The estimated initial and final values of the polynomial coefficients are given in table 1.

It was found that when a 2nd degree polynomial was used to model the fault plane, the inversion had placed the top of the fault at 0.16 km, and the bottom at 3.91 km (figure 2b). In case of a 3rd degree, the top edge was placed at 0.21 km and the bottom at 3.84 km, respectively. Therefore, the bottom depth of the recovered structure remains almost the same regardless of the degree of polynomial used in the inversion. On the other hand, the depth to the top of the fault edge was slightly overestimated in either case when compared to the assumed one. Also, the choice of 2nd degree in the inversion has led to the underestimation of the extension by about 6.5%. However, such small deviations in the estimated structure are acceptable because of the fact that the anomalies used in the analysis contain random noise. The changes in the data misfit, damping factor, and other unknown parameters of the model space against the iteration are shown in figure 3.

To assess the effect of planar fault plane on the interpretation of anomalies caused by listric faults, the anomalies in figure 2(a) are also analysed using the method of Murthy *et al.* (2001). The recalculated anomalies of the optimum model and the resultant structure are shown in figure 2(a and b). It is clearly seen from figure 2(b) that the deduced structure with its top at 0.5 km, bottom at 6.5 km, and fault angle of 120° does not comply well with the true structure.

4.2 *Field example – Western margin of the Perth Basin, Australia*

The Perth Basin of Australia is well-known for its hydrocarbon prospects. This rift basin is bounded by the Carnarvon Basin to the north and the Yilgarn Craton to the east. To the southeast, it is boarded by the Bight Basin, and on the west and south, it shares the oceanic crust of the Indian and Southern oceans (Kovac *et al.* 2016). Figure 4(a) shows an aeromagnetic anomaly profile across the western margin of the basin at 30°S (Qureshi and Nalaye 1978). For the present study, the anomaly profile was digitised into 20 observations and subjected to inversion, taking a 2nd-degree polynomial to simulate the fault plane. The

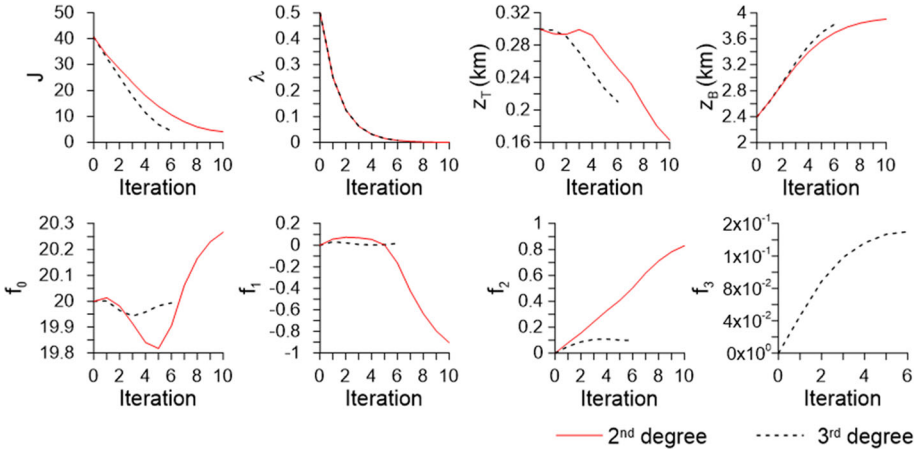


Figure 3. Changes in misfit, damping factor and other parameters of model space with iteration number.

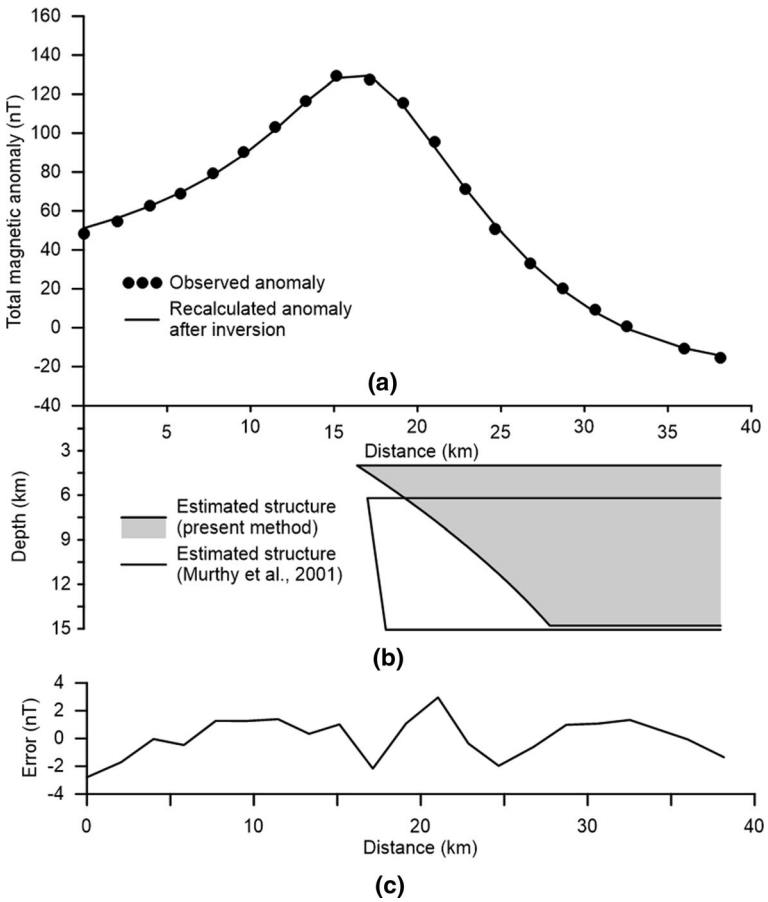


Figure 4. **(a)** Observed and recalculated total magnetic anomalies and **(b)** estimated structure after inversion, inverted depth model by Murthy *et al.* (2001) is also shown. **(c)** The left-out residuals between observed and recalculated anomalies by the present method, western margin of the Perth Basin, Australia.

theoretical anomalies and the deciphered structure after the 20th iteration, being the concluding one, were shown in figure 4(a and b), respectively. Overall, the residuals among the observed and theoretical anomalies vary between −2.7 and 2.9 gammas along the length of the profile (figure 4c). The present technique found the origin of the fault plane on the profile at the 16th km at a depth of 4 km and placed its bottom at 14.8 km. The intensity and direction of magnetisation are estimated as 68.3 gammas and 315°. The inferred coefficients of the 2nd degree polynomial are given in table 1. The changes in the data misfit, damping factor and other parameters of the model space as a

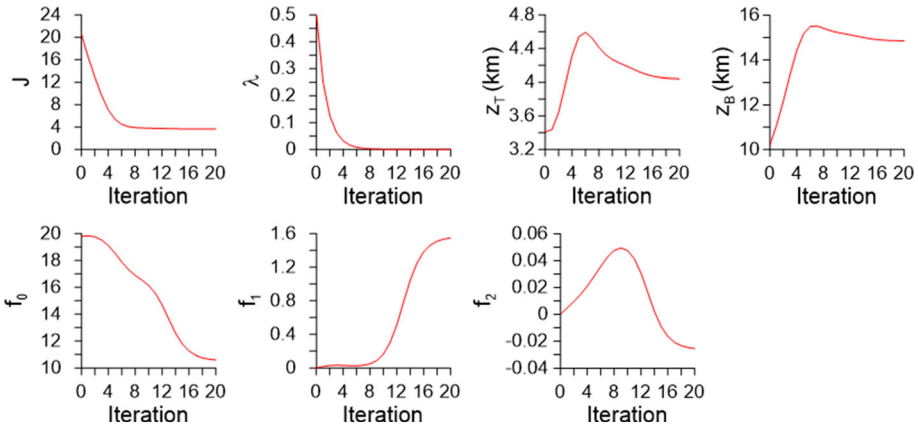


Figure 5. Changes in misfit, damping factor and other parameters of model space with iteration number, western margin of the Perth Basin, Australia.

function of iteration number are shown in figure 5. The interpreted model of the same anomaly treating the fault plane as a planar surface (Murthy *et al.* 2001) was also shown in figure 4(b) for comparison. The deduced structure from the present inversion appears to be more appropriate because the identified faults across the basin margins from seismic imaging also reveal listric nature with depth (Middleton *et al.* 1993).

5. Discussion

An automatic inversion technique was presented to quantify 2D listric fault sources from the measured magnetic anomalies in any component. This method uses the Marquardt's algorithm in its implementation. From the measured anomalies, the proposed method estimates the parameters of the source, intensity and direction of magnetisation. To calculate the model response, this technique makes use of the forward modelling scheme developed by the same authors earlier.

The validity of the proposed technique was exemplified with a theoretical model and a real field case. In the theoretical example, the model space was constructed using a set of known parameters, and its magnetic response was calculated in the vertical component. In doing so, the fault plane was described with a 5th degree polynomial. Pseudorandom noise was added to the structure anomalies before being inverted to recover the structure. Though the structure anomalies were noisy, the proposed technique had fairly recovered the structure, where the fault plane was modelled with a 5th degree polynomial. Later, the noisy anomalies were also analysed using different polynomials (2nd and 3rd degrees) for fault plane simulations. It was demonstrated that, by and large, the technique was successful in restoring the assumed structure, though the degrees of polynomials were different from the optimum. On the other hand, the structure obtained from the interpretation of anomalies with planar fault plane assumption was not in line with the assumed structure. Therefore, it is discouraged to analyse the magnetic anomalies caused by listric fault sources by the techniques, which treat the fault planes as planar surfaces.

In the case of real field application, the total magnetic field anomalies attributed to a deep-seated fault from the western margin of the Perth Basin, Australia, are interpreted presuming a nonplanar surface for the fault plane. The interpreted results are compared with the reported ones.

Acknowledgements

The authors thank the reviewers and Associate Editor for their suggestions and feedback. Ramamma Batta thank DST, Government of India, for granting the Women Scientist scheme.

Author statement

Ani Nibisha and Chakravarthi developed the inversion methodology. Ramamma developed the code, tested the algorithm on data sets, and prepared illustrations and tables. Chakravarthi wrote the manuscript.

References

Abdelrahman E M and Essa K S 2015 Three least-squares minimisation approaches to interpret gravity data due to dipping faults; *Pure Appl. Geophys.* **172** 427–438.

Abdelrahman E M, El-Araby H M, El-Araby T M and Abo-Ezz E R 2003 A least-squares derivatives analysis of gravity anomalies due to faulted thin slabs; *Geophysics* **68** 535–543.

Al-Garni M A 2016 Artificial neural network inversion of magnetic anomalies caused by 2D fault structures; *Arab. J. Geosci.* **9**, 156, https://doi.org/10.1007/s12517-015-2256-y.

Anderson N L, Essa K S and Elhussein M 2020 A comparison study using particle swarm optimisation inversion algorithm for gravity anomaly interpretation due to a 2D vertical fault structure; *J. Appl. Geophys.* **179**, 104120, https://doi.org/10.1016/j.jappgeo.2020.104120.

Biswas A and Rao K 2021 Interpretation of magnetic anomalies over 2D fault and sheet-type mineralised structures using very fast simulated annealing global optimisation: An understanding of uncertainty and geological implications; *Lithosphere* 2964057, https://doi.org/10.2113/2021/2964057.

Chakravarthi V 2003 Digitally implemented method for automatic optimisation of gravity fields obtained from three-dimensional density interfaces using depth dependent density – US Patent # 6,615,139.

Chakravarthi V 2010a Gravity anomalies of 2D fault structures with fault planes described by polynomial functions of arbitrary degree; *Curr. Sci.* **99** 654–656.

Chakravarthi V 2010b Gravity anomalies of strike limited listric fault sources with analytically defined fault planes and arbitrary density contrast variations with depth; *Near Surf. Geophys.* **8** 279–286.

Chakravarthi V 2011 Automatic gravity optimisation of 2.5D strike listric fault sources with analytically defined fault planes and depth dependent density; *Geophysics* **76** I21–I31.

Chakravarthi V and Pramod Kumar M 2015 Estimation of multiple density-depth parameters from gravity inversion: Application to detached hanging wall systems of strike limited listric fault morphologies; *Geofis. Int.* **54** 49–65.

Chakravarthi V and Sundararajan N 2004 Ridge regression algorithm for gravity inversion of fault structures with variable density; *Geophysics* **69** 1394–1404.

Chakravarthi V, Singh S B and Ashok Babu G 2001 INVER2DBASE – A program to compute basement depths of density interfaces above which the density contrast varies with depth; *Comput. Geosci.* **27** 1127–1133.

Chakravarthi V, Pramod Kumar M, Ramamma B and Rajeswara Sastry S 2017 Gravity anomaly interpretation of 2D fault morphologies by means of nonplanar fault planes and exponential density contrast model: A space domain technique; *Arab. J. Geosci.* **10** 64, https://doi.org/10.1007/s12517-017-2845-z.

Ekinci Y L, Balkaya C and Göktürkler G 2019 Parameter estimations from gravity and magnetic anomalies due to deep-seated faults: Differential evolution versus particle swarm optimisation; *Turkish J. Earth Sci.* **28** 860–881.

Elhussein M 2021 New inversion approach for interpreting gravity data caused by dipping faults; *Earth Space Sci.* **8** e2020EA001075, https://doi.org/10.1029/2020EA001075.

Essa K S 2013 Gravity interpretation of dipping faults using the variance analysis method; *J. Geophys. Eng.* **10** 015003, https://doi.org/10.1088/1742-2132/10/1/015003.

Essa K S 2021 Evaluation of the parameters of the fault-like geologic structure from the gravity anomalies applying the particle swarm; *Environ. Earth Sci.* **80**, https://doi.org/10.1007/s12665-021-09786-1.

Essa K S and Elhussein M 2018 PSO (Particle Swarm Optimization) for interpretation of magnetic anomalies caused by simple geometrical structures; *Pure Appl. Geophys.* **175** 3539–3553.

Essa K S, Géraud Y and Diraison M 2021 Fault parameters assessment from the gravity data profiles applying the global particle swarm optimisation; *J. Petrol. Sci. Eng.* **207**, 109129, https://doi.org/10.1016/j.petrol.2021.109129.

Green R 1979 The harmonic method of inverting a magnetic profile over a contact; *Geoexploration* **17** 261–268.

Jackson J A 1987 Active normal faulting and crustal extension; *Geol. Soc. London, Spec. Publ.* **28** 3–17.

Koulomzine T, Lamontamte Y and Nadeau A 1970 New methods for the direct interpretation of magnetic anomalies caused by inclined dikes of infinite length; *Geophysics* **35** 812–830.

Kovac P, Cevallos C and Feijth J 2016 Targeting oil and gas in the Perth Basin using an airborne gravity gradiometer; *First Break* **34** 51–58.

Marquardt D W 1970 Generalised inverses, ridge regression, biased linear estimation, and nonlinear estimation; *Technometrics* **12** 591–612.

Middleton M F, Wilde S, Evans B A, Long A and Dentith M 1993 Preliminary interpretation of deep seismic reflection and other geophysical data from the Darling Fault Zone, Western Australia; *Explor. Geophys.* **24** 711–718.

Murthy I V R 1985 The mid-point method: Magnetic interpretation of dykes and faults; *Geophysics* **50** 834–839.

Murthy I V R 1998 Gravity and magnetic interpretation in exploration geophysics; *Geol. Soc. India Memoir* **40**.

Murthy I V R, Rao C V and Krishna G G 1980 A gradient method for interpreting magnetic anomalies due to horizontal circular cylinders, infinite dykes and vertical steps; *Proc. Indian Acad. Sci. (Earth Planet. Sci.)* **89** 31–42.

Murthy I V R, Swamy K V and Rao S J 2001 Automatic inversion of magnetic anomalies of faults; *Comput. Geosci.* **27** 315–325.

Nibisha V A, Ramamma B, Sastry R S and Chakravarthi V 2021 Forward modelling: Magnetic anomalies of arbitrarily magnetised 2D fault sources with analytically defined fault planes; *J. Earth Syst. Sci.* **130** 130, https://doi.org/10.1007/s12040-021-01634-x.

Powell D W 1967 Fitting observed profiles to a magnetised dyke or fault-step model; *Geophys. Prospect.* **15** 208–220.

Qureshi I R and Nalaye A M 1978 A method for the direct interpretation of magnetic anomalies caused by two-dimensional vertical faults; *Geophysics* **43** 179–188.

Raju D, Ch V, Ravikumar S and Mishra D C 1998 Inversion of gravity anomaly due to a contact (fault) and its application for graben tectonics across Godavari basin; *Curr. Sci.* **75** 1184–1188.

Ramamma B, Mallesh K and Chakravarthi V 2021 3D Spatial domain gravity inversion with growing multiple polygonal cross-sections and exponential mass density contrast; *J.*

Earth Syst. Sci. **130** 73, https://doi.org/10.1007/s12040-021-01576-4.

Rao D B 1985 Analysis of gravity anomalies over an inclined fault with quadratic density function; *Pure Appl. Geophys.* **123** 250–260.

Rao A D and Babu H V R 1983 Standard curves for the interpretation of magnetic anomalies over vertical faults; *Geophys. Res. Bull.* **21** 71–89.

Rao K and Biswas A 2021 Modeling and uncertainty estimation of gravity anomaly over 2D fault using very fast simulated annealing global optimisation; *Acta Geophys.* **69** 1735–1751.

Roy A, Kumar T S and Sharma R K 2022 Structure estimation of 2D listric faults using quadratic Bezier curve for depth varying density distributions; *Earth Space Sci.* **9** e2021EA002061, https://doi.org/10.1029/2021EA002061.

Sengupta S 1974 Fourier transforms of magnetic anomalies of two-dimensional Bodies; *Pure Appl. Geophys.* **112** 987–995.

Shelton J W 1984 Listric normal faults: An illustrated summary; *Bull. Am. Assoc. Petrol. Geol.* **68** 801–815.

Spahić D, Exner U, Behm M, Grasemann B, Haring A and Pretsch H 2011 Listric versus planar normal fault geometry: An example from the Eisenstadt-Sopron Basin (E Austria); *Int. J. Earth Sci. (Geol Rundsch)* **100** 1685–1695.

Stavrev P 2006 Inversion of elongated magnetic anomalies using magnitude transforms; *Geophys. Prospect.* **54** 153–166.

Subrahmanyam M and Rao T K S P 2009 Interpretation of magnetic anomalies using some simple characteristic positions over tabular bodies; *Explor. Geophys.* **40** 265–276.

Sundararajan N, Mohan N L and Rao S V S 1983 Gravity interpretation of 2D fault structures using Hilbert transforms; *J. Geophys. (Zeitschrift fur Geophysik)* **53** 34–41.

Zhao H, Zhang J, Qu J, Zhang B, Yun L, Niu P, Hui J and Zhang Y 2020 Formation of listric normal faults by extensional duplexing: A case study from the active Langshan piedmont fault, NW China; *J. Struct. Geol.* **140**, 104158, https://doi.org/10.1016/j.jsg.2020.104158.

Corresponding editor: Arkoprovo Biswas

References

Abdelrahman, E. S. M., & Essa, K. S. (2015). Three least-squares minimization approaches to interpret gravity data due to dipping faults. *Pure and Applied Geophysics*, *172*, 427–438.

Abdelrahman, E. S. M., El-Araby, H. M., El-Araby, T. M., & Abo-Ezz, E. R. (2003). A least-squares derivatives analysis of gravity anomalies due to faulted thin slabs. *Geophysics*, *68*(2), 535–543.

Abedi, M., Siahkoohi, H. R., Gholami, A., & Norouzi, G. H. (2015). 3D Inversion of Magnetic Data through Wavelet based Regularization Method. *International Journal of Mining and Geo-Engineering, 49*, 1-18.

Agocs, W. B. (1951). Least squares residual anomaly determination. *Geophysics*, *16*(4), 686–696.

Al-Garni, M. A. (2016). Artificial neural network inversion of magnetic anomalies caused by 2D fault structures. *Arabian Journal of Geosciences*, *9*, 1–8.

Anderson, N. L., Essa, K. S., & Elhussein, M. (2020). A comparison study using particle swarm optimization inversion algorithm for gravity anomaly interpretation due to a 2D vertical fault structure. *Journal of Applied Geophysics*, *179*, 104120.

Ani Nibisha, V., Ramamma, B., & Chakravarthi, V. (2022). Automatic inversion of magnetic anomalies caused by 2D listric fault sources with arbitrary magnetisation. *Journal of Earth System Science*, *131*(4), 265.

Ani Nibisha, V., Ramamma, B., Sastry, S. R., & Chakravarthi, V. (2021). Forward modelling: Magnetic anomalies of arbitrarily magnetized 2D fault sources with analytically defined fault planes. *Journal of Earth System Science*, *130*(3), 130.

Asfahani, J., & Tlas, M. (2007). A robust nonlinear inversion for the interpretation of magnetic anomalies caused by faults, thin dikes and spheres like structure using stochastic algorithms. *Pure and Applied Geophysics*, *164*, 2023–2042.

Asfahani, J., & Tlas, M. (2004). Nonlinearly constrained optimization theory to interpret magnetic anomalies due to vertical faults and thin dikes. *Pure and Applied Geophysics*, *161*, 203–219.

Baranov, V., & Naudy, H. (1964). Numerical calculation of the formula of reduction to the magnetic pole. *Geophysics*, *29*(1), 67–79.

Baranov, V. (1957). A new method for interpretation of aeromagnetic maps: Pseudo-gravimetric anomalies. *Geophysics*, *22*(2), 359–382.

Bhattacharya, J. P., & Davies, R. K. (2001). Growth faults at the prodelta to delta-front transition, Cretaceous Ferron sandstone, Utah. *Marine and Petroleum Geology*, *18*(5), 525–534.

Bhattacharyya, B. K. (1965). Two-dimensional harmonic analysis as a tool for magnetic interpretation. *Geophysics*, *30*(5), 829–857.

Biswas, A., & Rao, K. (2021). Interpretation of magnetic anomalies over 2D fault and sheet-type mineralized structures using very fast simulated annealing global optimization: An understanding of uncertainty and geological implications. *Lithosphere*, *2021*(Special 6), 2964057.

Biswas, A. (2018). Inversion of source parameters from magnetic anomalies for mineral/ore deposits exploration using global optimization technique and analysis of uncertainty. *Natural Resources Research*, *27*(1), 77–107.

Biswas, A. (2016). Interpretation of gravity and magnetic anomaly over thin sheet-type structure using very fast simulated annealing global optimization technique. *Modeling Earth Systems and Environment*, *2*(1), 30.

Biswas, A., & Acharya, T. (2016). A very fast simulated annealing method for inversion of magnetic anomaly over semi-infinite vertical rod-type structure. *Modeling Earth Systems and Environment*, *2*, 1–10.

Blakely, R. J. (1996). Potential theory in gravity and magnetic applications. *Cambridge university press.*

Bosch, M., & McGaughey, J. (2001). Joint inversion of gravity and magnetic data under lithologic constraints. *The Leading Edge*, *20*(8), 877–881.

Broome, J. (1986). Display and enhancement of aeromagnetic data with examples from Guysborough County, Nova Scotia. *Interpretation of Gravity and Magnetic Anomalies for Non-Specialists*, 212–252.

Bruckshaw, J. M., & Kunaratnam, K. (1963). The interpretation of magnetic anomalies due to dykes. *Geophysical Prospecting*, *11*(4), 509–522.

Byerly, P. E. (1965). Convolution filtering of gravity and magnetic maps. *Geophysics*, *30*(2), 281–283.

Cai, H., Xiong, B., & Zhu, Y. (2018). 3D modeling and inversion of gravity data in exploration scale. *Gravity-Geoscience Applications, Industrial Technology and Quantum Aspect*, *183*, 401–406.

Campbell, G., & Mason, R. (1979). The application of air-borne and ground geophysical techniques to the search for magnetite-quartzite associated base-metal deposits in Southern Africa. *Geol. Surv. Can., Econ. Geol. Rep., 31*, 757-777.

Chakravarthi, V., Pramod Kumar, M., Ramamma, B., & Rajeswara Sastry, S. (2017). Gravity anomaly interpretation of 2D fault morphologies by means of nonplanar fault planes and exponential density contrast model: A space domain technique. *Arabian Journal of Geosciences*, *10*, 1–10.

Chakravarthi, V., & Pramod Kumar, M. (2015). Estimation of multiple density-depth parameters from gravity inversion: Application to detached hanging wall systems of strike limited listric fault morphologies. *Geofísica Internacional*, *54*(1), 49–65.

Chakravarthi, V., Sastry, S. R., & Ramamma, B. (2013). 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. (2011). Automatic gravity optimization of 2.5 D strike listric fault sources with analytically defined fault planes and depth-dependent density. *Geophysics*, *76*(2), I21–I31.

Chakravarthi, V. (2010a). Gravity anomalies of 2D fault structures with fault planes described by polynomial functions of arbitrary degree. *Current Science*, *99*(5), 00113891.

Chakravarthi, V. (2010b). Gravity anomalies of strike limited listric fault sources with analytically defined fault planes and arbitrary density contrast variations with depth. *Near Surface Geophysics*, *8*(4), 279–286.

Chakravarthi, V., & Sundararajan, N. (2004). Ridge-regression algorithm for gravity inversion of fault structures with variable density. *Geophysics*, *69*(6), 1394–1404.

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., Singh, S., & Babu, G. A. (2001). INVER2DBASE—A program to compute basement depths of density interfaces above which the density contrast varies with depth. *Computers & Geosciences*, *27*(10), 1127–1133.

Chandler, V. W. (1985). Interpretation of Precambrian geology in Minnesota using low-altitude, high-resolution aeromagnetic data. *The utility of regional gravity and magnetic anomaly maps (ed.) by Hinze W. J. (Soc. Exploration Geophysicists)*, 375–391.

Commer, M. (2011). Three-dimensional gravity modelling and focusing inversion using rectangular meshes. *Geophysical Prospecting*, *59*, 966–979.

Cong, F., Zhang, H., Hao, F., & Xu, S. (2020). Direct control of normal fault in hydrocarbon migration and accumulation in northwestern Bozhong subbasin, Bohai Bay Basin, China. *Marine and Petroleum Geology*, *120*, 104555.

Currenti, G., Del Negro, C., Fortuna, L., & Ganci, G. (2007). Integrated inversion of ground deformation and magnetic data at Etna volcano using a genetic algorithm technique. *Ann. Geophys., 50*(1), 21-30.

Currenti, G., Del Negro, C., & Nunnari, G. (2005). Inverse modelling of volcanomagnetic fields using a genetic algorithm technique. *Geophysical Journal International*, *163*(1), 403–418.

Dampney, C. (1969). The equivalent source technique. *Geophysics*, *34*(1), 39–53.

Dean, W. C. (1958). Frequency analysis for gravity and magnetic interpretation. *Geophysics*, *23*(1), 97–127.

Desmarais, J. K., & Spiteri, R. J. (2017). Fast automated airborne electromagnetic data interpretation using parallelized particle swarm optimization. *Computers & Geosciences*, *109*, 268–280.

Dods, S., Teskey, D., & Hood, P. (1985). The new series of 1: 1 000 000-scale magnetic anomaly maps of the Geological Survey of Canada. *The Utility of Regional Gravity and Magnetic Anomaly Maps. Tulsa*.

Doo, W. B., Hsu, S. K., & Yeh, Y. C. (2007). A derivative-based interpretation approach to estimating source parameters of simple 2D magnetic sources from Euler deconvolution, the analytic-signal method and analytical expressions of the anomalies. *Geophysical Prospecting*, *55*(2), 255–264.

Duwiquet, H., Arbaret, L., Bellanger, M., Guillou-Frottier, L., & Heap, M. (2019). Are Crustal Fault Zones viable geothermal resources? Insights from the Pontgibaud Fault in French Massif Central. *European Geothermal Congress*, La Haye, Netherlands.

Ekinci, Y. L., Balkaya, Ç., & Göktürkler, G. (2019). Parameter estimations from gravity and magnetic anomalies due to deep-seated faults: Differential evolution versus particle swarm optimization. *Turkish Journal of Earth Sciences*, *28*(6), 860–881.

Ekinci, Y. L. (2016). MATLAB-based algorithm to estimate depths of isolated thin dike-like sources using higher-order horizontal derivatives of magnetic anomalies. *Springer Plus*, *5*, 1–15.

Elhussein, M. (2021). New inversion approach for interpreting gravity data caused by dipping faults. *Earth and Space Science*, *8*(2), e2020EA001075.

Erickson, S. G., Strayer, L. M., & Suppe, J. (2001). Mechanics of extension and inversion in the hanging walls of listric normal faults. *Journal of Geophysical Research: Solid Earth*, *106*(B11), 26655–26670.

Essa, K. S. (2021). Evaluation of the parameters of the fault-like geologic structure from the gravity anomalies applying the particle swarm. *Environmental Earth Sciences*, *80*(15), 489.

Essa, K. S., Géraud, Y., & Diraison, M. (2021). Fault parameters assessment from the gravity data profiles applying the global particle swarm optimization. *Journal of Petroleum Science and Engineering*, *207*, 109129.

Essa, K. S., & Elhussein, M. (2018). PSO (particle swarm optimization) for interpretation of magnetic anomalies caused by simple geometrical structures. *Pure and Applied Geophysics*, *175*, 3539-3553.

Essa, K. S. (2013). Gravity interpretation of dipping faults using the variance analysis method. *Journal of Geophysics and Engineering*, *10*(1), 015003.

Fedi, M., & Rapolla, A. (1999). 3-D inversion of gravity and magnetic data with depth resolution. *Geophysics*, *64*(2), 452–460.

Fraser, D., Fuller, B., & Ward, S. (1966). Some numerical techniques for application in mining exploration. *Geophysics*, *31*(6), 1066–1077.

Fregoso, E., & Gallardo, L. A. (2009). Cross-gradients joint 3D inversion with applications to gravity and magnetic data. *Geophysics*, *74*(4), L31–L42.

Fuller, B. D. (1967). Two-dimensional frequency analysis and design of grid operators. *Mining Geophysics, Soc. Of Explor. Geophys., 2,* 658-708.

Furness, P. (2007). Modelling magnetic fields due to steel drum accumulations. *Geophysical Prospecting*, *55*(5), 737–748.

Gay, S. P. (1965). Standard curves for magnetic anomalies over long horizontal cylinders. *Geophysics*, *30*(5), 818–828.

Gay, S. P. (1963). Standard curves for interpretation of magnetic anomalies over long tabular bodies. *Geophysics*, *28*(2), 161–200.

Gerovska, D., & Araúzo-Bravo, M. J. (2003). Automatic interpretation of magnetic data based on Euler deconvolution with unprescribed structural index. *Computers & Geosciences*, *29*(8), 949–960.

Godio, A., & Santilano, A. (2018). On the optimization of electromagnetic geophysical data: Application of the PSO algorithm. *Journal of Applied Geophysics*, *148*, 163–174.

Goussev, S., Charters, R., & Peirce, J. (2006). Mackenzie Delta: A case of one residual gravity anomaly and 16 dry exploration wells. *CSPG/CSEG/CWLS Joint Conference, 15–18 May, Calgary, Canada, Expanded Abstracts*, 435–439.

Grant, F. S. (1972). Review of data processing and interpretation methods in gravity and magnetics, 1964–71. *Geophysics*, *37*(4), 647–661.

Grant, F. S., & West, G. F. (1965). *Interpretation theory in applied geophysics*. New York: McGraw-Hill.

Green, R. (1979). The harmonic method of inverting a magnetic profile over a contact. *Geoexploration*, *17*(4), 261–268.

Griffin, W. R. (1949). Residual gravity in theory and practice. *Geophysics*, *14*(1), 39–56.

Gupta, V., & Ramani, N. (1980). Some aspects of regional-residual separation of gravity anomalies in a Precambrian terrain. *Geophysics*, *45*(9), 1412–1426.

Haigh, J. E., & Smith, M. J. (1975). Standard Curves for Interpretation of Magnetic Anomalies Due to Thin Dykes of Finite Depth Extent (Vol. 152). *Australian Government Publishing*

Service.

Hammer, S. (1963). Deep gravity interpretation by stripping. *Geophysics*, *28*(3), 369–378.

Hardman, R. F. P., & Booth, J. E. (1991). The significance of normal faults in the exploration and production of North Sea hydrocarbons. *Geological Society, London, Special Publications*, *56*(1), 1–13.

Hartman, R. R., Teskey, D. J., & Friedberg, J. L. (1971). A system for rapid digital aeromagnetic interpretation. *Geophysics*, *36*(5), 891–918.

Henderson, R. G. (1960). A comprehensive system of automatic computation in magnetic and gravity interpretation. *Geophysics*, *25*(3), 569–585.

Henderson, R. G., & Zietz, I. (1949). The upward continuation of anomalies in total magnetic intensity fields. *Geophysics*, *14*(4), 517–534.

Hinze, W. J. (1990). The role of gravity and magnetic methods in engineering and environmental studies. *Geotechnical and environmental geophysics: Volume I: Review and tutorial, Society of Exploration Geophysicists*, 75–126.

Hutchison, R. D. (1958). Magnetic analysis by logarithmic curves. *Geophysics*, *23*(4), 749–769.

Jackson, J. (1987). Active normal faulting and crustal extension. *Geological Society, London, Special Publications*, *28*(1), 3–17.

Keith, S. B., Gest, D. E., DeWitt, E., Woode Toll, N., & Everson, B. (1983). Metallic mineral districts and production in Arizona. *Arizona Bureau of Mines Bulletin, 194*, 58.

Keller, G. R., Smith, R. A., Hinze, W. J., & Aiken, C. L. V. (1985). Regional gravity and magnetic study of west Texas. *The utility of regional gravity and magnetic anomaly maps (ed.) by Hinze W. J. (Soc. Exploration Geophysicists)*, 198-212.

Khalil, M. H. (2016). Subsurface faults detection based on magnetic anomalies investigation: A field example at Taba protectorate, South Sinai. *Journal of Applied Geophysics*, *131*, 123–132.

Khalil, M. H. (2014). Detection of magnetically susceptible dyke swarms in a fresh coastal aquifer. *Pure and Applied Geophysics*, *171*, 1829–1845.

Khurana, K., Rao, S. S., & Pal, P. (1981). Frequency domain least-squares inversion of thick dike magnetic anomalies using Marquardt algorithm. *Geophysics*, *46*(12), 1745–1748.

Koulomzine, T., Lamontagne, Y., & Nadeau, A. (1970). New methods for the direct interpretation of magnetic anomalies caused by inclined dikes of infinite length. *Geophysics*, *35*(5), 812–830.

Kovac, P., Cevallos, C., & Feijth, J. (2016). Targeting oil and gas in the Perth Basin using an airborne gravity gradiometer. *First Break*, *34*(4).

Ku, C. C., & Sharp, J. A. (1983). Werner deconvolution for automated magnetic interpretation and its refinement using Marquardt's inverse modeling. *Geophysics*, *48*(6), 754–774.

Lane, L. S. (2002). Tectonic evolution of the Canadian Beaufort Sea–Mackenzie Delta region: A brief review. *Canadian Society of Exploration Geophysicists Recorder*, *27*(2), 49–56.

Lelievre, P. G., & Oldenburg, D. W. (2006). Magnetic forward modelling and inversion for high susceptibility. *Geophysical Journal International*, *166*(1), 76–90.

Li, Y., & Oldenburg, D. W. (1998). Separation of regional and residual magnetic field data. *Geophysics*, *63*(2), 431–439.

Li, Y., & Oldenburg, D. W. (1996). 3-D inversion of magnetic data. *Geophysics*, *61*(2), 394–408.

Lidiak, E., Hinze, W., Keller, G., Reed, J., Braile, L., & Johnson, R. (1985). Geologic significance of regional gravity and magnetic anomalies in the east-central midcontinent. *The utility of regional gravity and magnetic anomaly maps (ed.) by Hinze W. J. (Soc. Exploration Geophysicists),* 287–307.

Liu, S., Liang, M., & Hu, X. (2018). Particle swarm optimization inversion of magnetic data: Field examples from iron ore deposits in China. *Geophysics*, *83*(4), J43–J59.

Marquardt, D. W. (1970). Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation. *Technometrics*, *12*(3), 591–612.

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, *11*(2), 431–441.

McKenzie, D., & Jackson, J. (2012). Tsunami earthquake generation by the release of gravitational potential energy. *Earth and Planetary Science Letters*, *345*, 1–8.

McKenzie, D. (1978). Some remarks on the development of sedimentary basins. *Earth and Planetary Science Letters*, *40*(1), 25–32.

Mesko, C. (1966). Two-dimensional filtering and the second derivative method. *Geophysics*, *31*(3), 606–617.

Middleton, M., Wilde, S., Evans, B., Long, A., & Dentith, M. (1993). A preliminary interpretation of deep seismic reflection and other geophysical data from the Darling Fault zone, Western Australia. *Exploration Geophysics*, *24*(4), 711–718.

Mirzaei, M., & Bredewout, J. (1996). An iterative method for finding small magnetic objects in the subsurface by linear and non-linear inversion1. *Geophysical Prospecting*, *44*(2), 289–312.

Mohan, N., Sundararajan, N., & Seshagiri Rao, S. (1982). Interpretation of some two-dimensional magnetic bodies using Hilbert transforms. *Geophysics*, *47*(3), 376–387.

Montesinos, F. G., Blanco-Montenegro, I., & Arnoso, J. (2016). Three-dimensional inverse modelling of magnetic anomaly sources based on a genetic algorithm. *Physics of the Earth and Planetary Interiors*, *253*, 74–87.

Montesinos, F. G., Arnoso, J., & Vieira, R. (2005). Using a genetic algorithm for 3-D inversion of gravity data in Fuerteventura (Canary Islands). *International Journal of Earth Sciences*, *94*, 301–316.

Moo, J. (1965). Analytical Aeromagnetic Interpretation the Inclined PRISM. *Geophysical Prospecting*, *13*(2), 203–224.

Murthy, I. V. R., Swamy, K., & Rao, S. J. (2001). Automatic inversion of magnetic anomalies of faults. *Computers & Geosciences*, *27*(3), 315–325.

Murthy, I. V. R. (1998). *Gravity and magnetic interpretation in exploration geophysics*. Geological Society of India.

Murthy, I. V. R. (1990). Magnetic anomalies of two-dimensional bodies and algorithms for magnetic inversion of dykes and basement topographies. *Proceedings of the Indian Academy of Sciences-Earth and Planetary Sciences*, *99*, 549–579.

Murthy, I. V. R. (1985). The midpoint method: Magnetic interpretation of dikes and faults. *Geophysics*, *50*(5), 834–839.

Murthy, I. V. R., Rao, C. V., & Krishna, G. G. (1980). A gradient method for interpreting magnetic anomalies due to horizontal circular cylinders, infinite dykes and vertical steps. *Proceedings of the Indian Academy of Sciences-Earth and Planetary Sciences*, *89*, 31-42.

Mushayandebvu, M. F., van Driel, P., Reid, A. B., & Fairhead, J. D. (2001). Magnetic source parameters of two-dimensional structures using extended Euler deconvolution. *Geophysics*, *66*(3), 814–823.

Negi, J. G. (1967). Convergence and divergence in downward continuation. *Geophysics*, *32*(5), 867–871.

Nettleton, L. L. (1954). Regionals, residuals, and structures. *Geophysics*, *19*(1), 1–22.

Oldenburg, D. W., & Li, Y. (2005). Inversion for applied geophysics: A tutorial. *Near-surface geophysics*, 89-150.

Oldham, C., & Sutherland, D. (1955). Orthogonal polynomials: Their use in estimating the regional effect. *Geophysics*, *20*(2), 295–306.

Patalakha, Y. I. (1986). The Problem of Listric Faults. *International Geology Review*, *28*(12), 1416–1422.

Paterson, N. R., & Reeves, C. V. (1985). Applications of gravity and magnetic surveys: The state-of-the-art in 1985. *Geophysics*, *50*(12), 2558–2594.

Paul, M. K. (1972). Interpretation of the gravity anomaly over a causative body with circular symmetry. *Geophysical Prospecting*, *20*(1), 118–129.

Pauta, J., Mandon, C., Piispa, E., & Urquizo, M. (2021). Magnetometry Survey Applied to Geothermal Exploration in Chachimbiro, Northern Ecuador. *NSG2021 27th European Meeting of Environmental and Engineering Geophysics*, *2021*(1), 1–5.

Pawlowski, R. S., & Hansen, R. O. (1990). Gravity anomaly separation by Wiener filtering. *Geophysics*, *55*(5), 539–548.

Peters, L. J. (1949). The direct approach to magnetic interpretation and its practical application. *Geophysics*, *14*(3), 290–320.

Pilkington, M. (2009). 3D magnetic data-space inversion with sparseness constraints. *Geophysics*, *74*(1), L7–L15.

Pilkington, M. (1997). 3-D magnetic imaging using conjugate gradients. *Geophysics, 62*(4), 1132-1142.

Powell, D. (1967). Fitting observed profiles to a magnetized dyke or fault-step model. *Geophysical Prospecting*, *15*(2), 208–220.

Qureshi, I., & Nalaye, A. (1978). A method for the direct interpretation of magnetic anomalies caused by two-dimensional vertical faults. *Geophysics*, *43*(1), 179–188.

Radhakrishna Murthy, I. V., & Pradeep Kumar, Ch. (1983). On the application of uniqueness theorem in downward continuation of gravity and magnetic anomalies. *J. Assn. Expl. Geophysics, 4*, 13-23.

Raju, D. C. V., Ravikumar, S., & Mishra, D. (1998). Inversion of gravity anomaly due to a contact (fault) and its application for graben tectonics across Godavari basin. *Current Science*, 1184–1188.

Ramamma, B., Mallesh, K., & Chakravarthi, V. (2021). 3D spatial domain gravity inversion with growing multiple polygonal cross-sections and exponential mass density contrast. *Journal of Earth System Science*, *130*, 1–16.

Rao, B. S. R., & Murthy, I. V. R. (1978). Gravity and magnetic methods of prospecting. *Arnold-Heinemann Publishers (India) Pvt. Ltd., New Delhi, 390*.

Rao D. A., & Babu, H. V. R. (1983). Standard curves for the interpretation of magnetic anomalies over vertical faults. *Geophy Res Bull, 21,* 71-89.

Rao, D. A., Babu, H. V. R., & Narayan, P. V. S. (1980). Relationship of magnetic anomalies due to subsurface features and the interpretation of sloping contacts. *Geophysics*, *45*(1), 32–36.

Rao, D. B. (1985). Analysis of gravity anomalies over an inclined fault with quadratic density function. *Pure and Applied Geophysics*, *123*, 250–260.

Rao, K., & Biswas, A. (2021). Modeling and uncertainty estimation of gravity anomaly over 2D fault using very fast simulated annealing global optimization. *Acta Geophysica*, *69*(5), 1735–1751.

Reid, A. B., Allsop, J. M., Granser, H., Millett, A. T, & Somerton, I. W. (1990). Magnetic interpretation in three dimensions using Euler deconvolution. *Geophysics*, *55*(1), 80–91.

Roux, W.F. (1979). The development of growth fault structures. *AAPG, Structural Geology School Course Notes, 33.*

Roy, A., Kumar, T. S., & Sharma, R. K. (2022). Structure estimation of 2D listric faults using quadratic Bezier curve for depth varying density distributions. *Earth and Space Science*, *9*(2), e2021EA002061.

Salem, A. (2005). Interpretation of magnetic data using analytic signal derivatives. *Geophysical Prospecting*, *53*(1), 75–82.

Salem, A., & Ravat, D. (2003). A combined analytic signal and Euler method (AN-EUL) for automatic interpretation of magnetic data. *Geophysics*, *68*(6), 1952–1961.

Sen, M. K., & Stoffa, P. L. (2013). Genetic algorithms. In *Global optimization methods in geophysical inversion*. Cambridge University Press, 119-147.

Sengupta, S. (1974). Fourier transforms of magnetic anomalies of two-dimensional bodies. *Pure and Applied Geophysics*, *112*, 987–995.

Sharma, P. V. (1987). Magnetic method applied to mineral exploration, *Ore Geology Reviews, 2* (4), 323-357. https://doi.org/10.1016/0169-1368(87)90010-2.

Shelton, J. W. (1984). Listric normal faults: An illustrated summary. *AAPG Bulletin*, *68*(7), 801–815.

Singh, K. T. (2013). Gravity and magnetic surveys to delineate favorable zones of uranium mineralization along Khangaon Budrukh Kabalapur tract, Belgaum district, Karnataka, India. *University of Hyderabad, Dissertation*.

Skeels, D. (1967). What is residual gravity? *Geophysics*, *32*(5), 872–876.

Smith, R. B., & Bruhn, R. L. (1984). Intraplate extensional tectonics of the eastern Basin-Range: Inferences on structural style from seismic reflection data, regional tectonics, and thermal-mechanical models of brittle-ductile deformation. *Journal of Geophysical Research: Solid Earth*, *89*(B7), 5733–5762.

Soengkono, S., Hochstein, M. P., Smith, I. E. M., & Itaya, T. (1992). Geophysical evidence for widespread reversely magnetised pyroclastics in the western Taupo Volcanic Zone (New Zealand). *New Zealand Journal of Geology and Geophysics*, *35*(1), 47–55.

Spahić, D., Exner, U., Behm, M., Grasemann, B., Haring, A., & Pretsch, H. (2011). Listric versus planar normal fault geometry: An example from the Eisenstadt-Sopron Basin (E Austria). *International Journal of Earth Sciences*, *100*, 1685–1695.

Spencer, J. E., & Reynolds, S. J. (1989). Geology and Mineral Resources of the Buckskin and Rawhide Mountains, West Central Arizona. *Arizona Geological Survey Bulletin 198 (Shackelford Volume),* 279.

Spencer, J. E., & Welty, J. W. (1986). Possible controls of base-and precious-metal mineralization associated with Tertiary detachment faults in the lower Colorado River trough, Arizona and California. *Geology*, *14*(3), 195–198.

Stanley, J. M. (1977). Simplified gravity and magnetic interpretation of contact and dyke-like structures. *Exploration Geophysics*, *8*(3), 60–64.

Stavrev, P. (2006). Inversion of elongated magnetic anomalies using magnitude transforms. *Geophysical Prospecting*, *54*(2), 153–166.

Subrahmanyam, M., & Rao, T. P. (2009). Interpretation of magnetic anomalies using some simple characteristic positions over tabular bodies. *Exploration Geophysics*, *40*(3), 265–276.

Sundararajan, N., Mohan, N., & Rao, S. S. (1983). Gravity interpretation of two dimensional fault structures using Hilbert transforms. *Journal of Geophysics-Zeitschrift Fur Geophysik*, *53*(1), 34–41.

Tarits, A. J. M., Hautot, P., Bellanger, S., Coutant, M., & Maïa, O. M. (2019). Joint inversion of gravity and surface wave data constrained by magnetotelluric: application to deep geothermal exploration of crustal fault zone in felsic basement. *Geothermics, 80*, 56-68.

Tavakoli, M., Kalateh, A. N., & Ghomi, S. (2016). The interpretation of magnetic anomalies by 3D inversion: A case study from Central Iran. *Journal of African Earth Sciences*, *115*, 85–91.

Teimouri, R., & Baseri, H. (2014). Optimization of magnetic field assisted EDM using the continuous ACO algorithm. *Applied Soft Computing*, *14*, 381–389.

Telford, W. M., Geldart, L. P., & Sheriff, R. E. (1990). Applied geophysics. *Cambridge university press, New York.*

Telford, W. M., Geldart, L. P., Sheriff, R. E., & Keys, D. A. (1976). Applied Geophysics. *Cambridge University Press, New York*.

Tlas, M., & Asfahani, J. (2011). Fair function minimization for interpretation of magnetic anomalies due to thin dikes, spheres and faults. *Journal of Applied Geophysics*, *75*(2), 237–243.

Torizin, J., Jentzsch, G., Malischewsky, P., Kley, J., Abakanov, N., & Kurskeev, A. (2009). Rating of seismicity and reconstruction of the fault geometries in northern Tien Shan within the project "Seismic Hazard Assessment for Almaty." *Journal of Geodynamics*, *48*(3–5), 269–278.

Ulugergerli, E. U., & Meju, M. A. (1997). Inversion of 2d magnetotelluric data using the complex singular value decomposition method. *5th International Congress of the Brazilian Geophysical Society*, 299.

Wang, G., Zhang, S., Yan, C., Xu, G., Ma, M., Li, K., & Feng, Y. (2012). Application of the multifractal singular value decomposition for delineating geophysical anomalies associated with molybdenum occurrences in the Luanchuan ore field (China). *Journal of Applied Geophysics*, *86*, 109–119.

Wang, L., & Lilley, F. (1999). Inversion of magnetometer array data by thin-sheet modelling. *Geophysical Journal International*, *137*(1), 128–138.

Won, I. (1981). Application of Gauss's method to magnetic anomalies of dipping dikes. *Geophysics*, *46*(2), 211–215.

Xu, G. H., Yu, Q. F., & Yuan, X.C. (2007). A tentative discussion on the application of the deep geothermal exploration method in Beijing area. *Geophysical & Geochemical Exploration, 31* (1), 9–13.

Yarger, H. L. (1985). Kansas basement study using spectrally filtered aeromagnetic data. *The utility of regional gravity and magnetic anomaly maps (ed.) by Hinze W. J. (Soc. Exploration Geophysicists),* 213-232.

Yu, J., You, X., & Liu, S. (2021). Ant colony algorithm based on magnetic neighborhood and filtering recommendation. *Soft Computing*, *25*(13), 8035–8050.

Zhao, H., Zhang, J., Qu, J., Zhang, B., Yun, L., Niu, P., Hui, J., & Zhang, Y. (2020). Formation of listric normal faults by extensional duplexing: A case study from the active Langshan piedmont fault, NW China. *Journal of Structural Geology*, *140*, 104158.

Zurflueh, E. G. (1967). Applications of two-dimensional linear wavelength filtering. *Geophysics*, *32*(6), 1015–1035.

Space domain-based algorithms and related software to analyze magnetic anomalies of 2D Listric fault structures

by V. Ani Nibisha

Submission date: 26-Apr-2024 04:21PM (UTC+0530)
Submission ID: 2362502236
File name: V._Ani_Nibisha.pdf (2.01M)
Word count: 15161
Character count: 79516

Space domain-based algorithms and related software to analyze magnetic anomalies of 2D Listric fault structures

ORIGINALITY REPORT

28%	24%	26%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	link.springer.com Internet Source Certified that this is student's publication in which she is the first author 	20%
2	www.ias.ac.in Internet Source Certified that this is student's publication in which she is the first author 	2%
3	Submitted to University of Hyderabad, Hyderabad Student Paper	2%
4	V. Chakravarthi, S. Rajeswara Sastry, M. Pramod Kumar. "A method and a GUI based JAVA code for interactive gravity modeling of strike limited listric fault sources with arbitrary density-depth variations", Journal of the Geological Society of India, 2014 Publication	1%
5	Nathani Basavaiah. "Geomagnetism", Springer Nature, 2011 Publication	1%
6	V Ani Nibisha, B Ramamma, S Rajeswara Sastry, V Chakravarthi. "Forward modelling: Magnetic anomalies of arbitrarily magnetized	<1%

2D fault sources with analytically defined fault planes", Journal of Earth System Science, 2021
Publication

---

7 V. Chakravarthi. "LSTRKFALTG – A forward modeling program to compute theoretical gravity anomalies of strike limited listric fault structures with prescribed vertical variation in density", Computers & Geosciences, 2010 <1%
Publication

---

8 Christoph Clauser. "Introduction to Geophysics", Springer Science and Business Media LLC, 2024 <1%
Publication

---

9 D. A. Clark, S. J. Saul, D. W. Emerson. "Magnetic and gravity anomalies of a triaxial ellipsoid", Exploration Geophysics, 2018 <1%
Publication

---

10 V. Chakravarthi, S. Rajeswara Sastry, B. Ramamma. "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, 2013 <1%
Publication

---

11 V Ani Nibisha, B Ramamma, V Chakravarthi. "Automatic inversion of magnetic anomalies <1%

caused by 2D listric fault sources with arbitrary magnetisation", Journal of Earth System Science, 2022
Publication

12	library.seg.org Internet Source	<1%
13	V. Chakravarthi. "Automatic gravity optimization of 2.5D strike listric fault sources with analytically defined fault planes and depth-dependent density", GEOPHYSICS, 2011 Publication	<1%
14	madsci.org Internet Source	<1%

Exclude quotes	On	Exclude matches	< 14 words
Exclude bibliography	On		