

# **Space domain based modeling and inversion techniques of gravity anomalies using variable density-depth models**

**A Thesis submitted to the University of Hyderabad for the Degree of  
DOCTOR OF PHILOSOPHY  
in University Centre for Earth and Space Sciences, School of Physics**



by  
**M. Pramod Kumar**

**November, 2015**



## **CERTIFICATE**

This is to certify that the thesis entitled “*Space domain based modeling and inversion techniques of gravity anomalies using variable density-depth models*” submitted by Mr. M. Pramod Kumar, bearing regd. No 13ESPE02, in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in Geophysics is a bonafide work carried out by her under my supervision and guidance which is a plagiarism free thesis.

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

Date: 26.11.2015

(V. Chakravarthi)  
Supervisor

Head of the Centre

Dean of the School

## Declaration

---

*University Centre for Earth and Space Sciences  
School of Physics  
University of Hyderabad  
Hyderabad – 500 046*

Dated: 26.11.2015

This is to certify that I, M. Pramod Kumar have carried out the research embodied in the present thesis for the full period prescribed under Ph.D ordinances of the University.

I declare to the best of my knowledge that no part of this thesis was earlier submitted for the award of research degree of any University.

(Signature of the candidate)

Name: M. Pramod Kumar

Enrollment No: 13ESPE02

Signature of the Supervisor

**Head of the Department**

**Dean of the School**

*To my beloved parents....*



## Acknowledgements

---

*It is with a deep sense of indebtedness and gratitude that I thank my research supervisor Dr. V. Chakravarthi, Associate Professor, University Centre for Earth and Space Sciences (UCESS), University of Hyderabad for his expert guidance and constant encouragement throughout the progress of my research work. His ever friendly nature and obliging attitude made my every scientific discussion with him something special.*

*Special thanks go to the Doctoral Research Committee members, Professor B. Madhusudan Rao and Dr. S. Sri Lakshmi, for giving appropriate inputs, when and wherever required, during the course my research work.*

*I sincerely thank Professor M. Jayananda, Head, UCESS for providing all the necessary facilities to carry out my research work. Drs. P. S. Roy, K. Ashok, and Sri T. Suryanarayana have extended their timely support and encouragement, for which I am ever grateful.*

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

*Dr. B. Ramamma, Mr. K. Mallesh, Mr. N. Naveen Kumar have helped me a lot during the final compilation of my thesis. I would like to thank all of them including my fellow research scholars and*

*supporting staff in the Centre for their benevolent attitude and friendliness.*

*I owe my affection to my friends Rupini Priyanka, Neelesh Babu and Nagabhushanam with whom I have shared all my happiness and sorrows. Their association is indeed a fabulous gift given to me by the Almighty.*

*Words fall short to express my deep sense of respects and regards to my parents and other family members for their cordial support and encouragement all through my academic career, for which I am ever indebted.*

\*\*\*

# Contents

---

Certificate	
Declaration	
Acknowledgements	
Preface	i
List of Figures	ix
List of Tables	xv
List of research publications (SCI)	xvi

## **I Introduction**

1.1 General	1
1.2 Exponential Density Contrast Model (EDCM)	5
1.3 Gravity corrections	9
1.4 Interpretation	10
1.4.1 Qualitative interpretation	10
1.4.2 Regional and residual anomaly separation	11
1.4.3 Quantitative interpretation	12
1.5 Review of existing methods	14
1.6 Aim and scope of the thesis	19

## **II Analysis of gravity anomalies of 2D listric fault morphologies using a prescribed exponential density contrast model**

2.1 General	22
2.2 Status of existing interpretational techniques	24
2.3 Forward modeling – Theoretical considerations	27
2.4 Inversion of gravity anomalies	29
2.5 Description of software – GRIN2DFL	32
2.6 Applications	35
2.6.1 Synthetic example – Inversion of noisy gravity anomalies	35
2.6.2 Field example	40
2.7 Results and discussion	46

## **III Interactive gravity modeling of 2.5D strike listric fault sources with arbitrarily varying density-depth relationship**

3.1 General	48
-------------	----

3.2	Status of existing interpretational techniques	49
3.3	Forward modeling – Theoretical considerations	52
3.3.1	Effects of offset and strike length on the magnitude of anomaly	53
3.4	Description of the software - FRGMLSTRK	56
3.5	Applications	65
3.5.1	Synthetic example	65
3.5.2	Field example	69
3.6	Results and discussion	73
<b>IV</b>	<b>Simultaneous estimation of multiple density-depth parameters and fault plane geometry from gravity inversion: Application to 2.5D strike listric fault morphologies</b>	
4.1	General	76
4.2	Inversion of gravity anomalies	77
4.3	Description of the software – GRAVLIS	80
4.4	Applications	82
4.4.1	Synthetic example	82
4.4.1.1	Simultaneous estimation of densities and fault plane geometry	84
4.4.1.2	Simultaneous estimation of depths and fault plane geometry	87
4.4.2	Field example	91
4.5	Results and discussion	99
<b>V</b>	<b>Automatic gravity modeling of sedimentary basins by means of polygonal source and exponential density contrast model</b>	
5.1	General	102
5.2	Status of existing interpretational techniques	103
5.3	Forward modeling – Theoretical considerations	107
5.3.1	Forward modeling of a homogeneous vertical prism	109
5.3.2	Forward modeling of a vertical prism with EDCM	111
5.3.3	Forward modeling of fault and trapezoidal geometries with EDCM	111

5.3.4	Forward modeling of a sedimentary basin	113
5.4	Automatic modeling of gravity anomalies	115
5.5	Description of the software - MOD2DGREXP	117
5.6	Applications	119
5.6.1	Synthetic example	119
5.6.2	Field example	121
5.7	Results and discussion	124
<b>VI</b>	<b>Automatic gravity inversion of sedimentary basins by means of growing polygonal source and exponential density contrast model</b>	
6.1	General	126
6.2	Status of existing interpretational techniques	127
6.3	Inversion	129
6.4	Description of the software - IN2DGREXP	130
6.5	Applications	
6.5.1	Synthetic example	132
6.5.2	Field example	135
6.6	Results and discussion	137
<b>VII</b>	<b>Conclusions</b>	139
	Scope for future research	141
<b>Annexures</b>	2A - Code listing of INGREXP	143
	2B - Sample output	167
	3A - Code listing of FRGMLSTRK	169
	4A - Code listing of GRAVLIS	216
	4B - Sample output	255
	4C - Sample output	257
	5A - Code listing of MOD2DGREXP	259
	5B - Sample output	276
	6A - Code listing of IN2DGREXP	277
	6B - Sample output	298
	<b>References</b>	299

## Preface

---

Geophysical methods play an important and indispensable role in the exploration of natural mineral resources such as oil and natural gas, hydrogeological investigations, geodynamic and tectonic studies, engineering and environmental problems, issues related to glaciology and volcanology, archaeological investigations etc. A wide spectrum of geophysical methods is in vogue for the exploration of the subsurface depending on the nature and type of the problem under consideration. The contrasts in physical properties such as the density, magnetization, electrical resistivity/conductivity, elastic properties etc. between the target source and the surroundings pave the way for the measurement of the corresponding anomalous fields. Unwarranted geophysical signals are suppressed/removed from the measured ones before being subjected to analysis for the subsurface structures.

In this direction, it is emphasized in unequivocal terms that the gravity method, which is the subject of this thesis, depends exclusively on considerable density contrast between the source/sources of interest and the surrounding formations, which generates a favorable gravity anomaly. Although, the density of sedimentary rocks does not lend itself to be simulated by any mathematical formulation; sometimes it is possible to model the density variation of sediments with depth by an exponential law. However, such simulation by an

exponential law makes it difficult to derive closed form analytical expressions for gravity anomalies in the space domain. On the other hand, closed form gravity expressions could be derivable in the frequency domain using exponential density-depth relationship; however, such forward modeling schemes find restricted practical application because truncation errors arise while transforming the anomalies from the frequency to the spatial domain. Thus it is imperative to develop appropriate mathematical tools in the space domain not only to realize forward gravity modeling of geologic sources/structures but also to estimate the source parameters making use of the exponential density function.

In the thesis, both analytical and numerical approaches have been used judiciously in the space domain to derive the expressions for calculating the gravity anomalies of selected geologic structures using Exponential Density Contrast Model (EDCM). Based on these forward modeling schemes, new automatic techniques are developed in the spatial domain using EDCM to interpret gravity anomalies due to 2D listric fault morphologies and sedimentary basins besides relevant GUI based softwares. Furthermore, a semiautomatic/interactive gravity modeling scheme and an optimization coupled with relevant softwares are developed to interpret the gravity anomalies produced by 2.5D listric fault sources, wherein the hanging wall systems consist of several formations. Applicability of each proposed technique is demonstrated both with synthetic and real field data analysis. The results are

highly encouraging and thereby substantiating the validity of the proposed techniques.

The thesis under study is organized into seven chapters as detailed below.

In chapter-I, general introduction of the gravity method, mathematical foundations, concept and role of exponential density contrast model, and a brief account of the earlier work done on modeling and inversion techniques are discussed.

In Chapter II, the principles of inversion are used to develop an automatic technique in the space domain to interpret the gravity anomalies of 2D listric fault morphologies, wherein the density contrast within the detached hanging wall varies exponentially with depth. The fault ramp of the structure is described by a predefined polynomial function having arbitrary but specific degree, whose coefficients become the unknown parameters to be estimated from the observed gravity anomalies in addition to the thickness of the detached hanging wall. The proposed inversion identifies approximate parameters pertaining to the location of the origin of fault plane and depth to the decollement horizon from a set of characteristic anomalies. Based on the errors between the observed and modeled gravity anomalies, the inversion technique constructs and subsequently solves the system of normal equations to estimate the improvements in the depth and coefficients of the polynomial in an iterative approach until the specified convergence criteria is fulfilled. Based on the proposed inversion methodology, a GUI based software, GRIN2DFL,



coded in JAVA is developed. This code, works on Model-View-Controller (MVC) pattern, reads the input parameters as specified by the interpreter and analyzes the anomalies for the concealed structure in an automatic manner. The software has inbuilt graphical user interface, which enables the interpreter to visualize the animated versions of the model growth and corresponding improvement of model response, changes in misfit, and variation of density contrast with depth. The efficiency of the inversion and software are illustrated with the gravity anomaly of a synthetic model of a 2D listric fault source in the presence of pseudorandom noise. The real field gravity anomalies across the Ahiri-Cherla master fault of the Godavari sub-basin in India are analyzed and found that the results are in line with the available/reported geologic information.

A semiautomatic/interactive modeling technique coupled with relevant GUI based JAVA code, FRGMLSTRK, is developed in chapter-III to analyze the gravity anomalies of 2.5D strike limited listric fault sources in real time. The hanging wall systems may consist in any number of formations irrespective of their thicknesses and densities. The fault ramps of the structures are described with analytical functions. The effects of profile azimuth and strike length of the structure on the magnitude of gravity anomaly due to a 2.5D listric fault source are demonstrated in length. The proposed scheme allows one to construct the fault plane geometry, depths and the densities of various subsurface formations in an interactive mode using simple mouse operations. The business logic of the algorithm computes the gravity response

arises from the model in real time and the inbuilt graphical user interface compares the model response with the observed anomalies. The differences between the said two anomalies could be minimized by modifying the model space, density and depth parameters either independently or in combination. This could be realized by simple drag and drop mouse operations. The applicability of the method and software is exemplified with both synthetic and real field gravity anomalies. In case of field example, the analysis of gravity anomalies across the Aswaraopet master fault from the eastern margin of the Chintalpudi sub-basin in India has yielded a structure that is marginally deviated from the one realized by Deep Seismic Sounding (DSS) studies.

Chapter-IV deals with the development of an inversion technique and associated GUI software in JAVA, using the principles of inversion, to simultaneously estimate the geometry of fault ramps and density or thickness parameters of the formations within the hanging wall systems of strike limited listric fault sources. The proposed inversion requires initial/guess parameters pertaining to the densities and depths of the formations within the hanging wall systems, whereas the parameters required for initiating the fault plane are calculated automatically. The MVC based software, GRAVLIS, reads the input parameters specified by the interpreter and performs the inversion to recover the structure. In addition to generating output in ASCII and graphical forms, the software displays the animated versions of model space improvement and corresponding changes in model gravity response and density-depth improvement with the iteration number. The efficiency of inversion is

demonstrated with a set of noisy gravity anomalies of a 2.5D synthetic model. The inversion when performed on the gravity anomalies of the Aswaraopet master fault of the Chintalpudi sub-basin has yielded results that are excellently comparable with the borehole information.

Based on the principles of modeling, an automatic technique coupled with GUI based software is developed using EDCM to model the gravity anomalies arise from 2D sedimentary basins and presented in chapter-V. The cross-section of a sedimentary basin is simulated with a polygonal source defined with appropriate number of vertices. The depth co-ordinates of the vertices of the polygon become the unknown parameters to be found from the observed gravity anomalies. Expression for the gravity anomaly of such a source with EDCM is derived in the space domain by making use of the Stoke's theorem. The validity of the proposed numerical method of forward modeling is demonstrated on three regular geophysical models, namely; a prism, vertical fault and a trapezoid. In all the cases, the anomalies realized from the proposed numerical method compare excellently well with those obtained from respective analytical gravity expressions of the models. The present modeling method computes the initial depths of a sedimentary basin from the observed gravity anomalies and updates them automatically in an iterative approach within the specified convergence criteria. Based on the algorithm, a software, MOD2DGREXP, coded in JAVA, with GUI compatibility is developed. This code, works on MVC pattern, reads the residual gravity anomalies of a sedimentary basin and estimates basement

depths at plurality of locations on the profile. Besides generating the output in both ASCII and graphical forms, the software displays the changes in the depth structure, nature of fit between the observed and modeled gravity anomalies, changes in misfit between the observed and model gravity anomalies, and variation of density contrast with depth against iteration in animated forms. The efficiency of proposed modeling is illustrated with a synthetic model of a sedimentary basin, whose anomalies are corrupted with random noise. In case of real field example, the gravity anomalies of the San Jacinto graben, California are analyzed using a derived exponential density contrast model. The estimated depth structure is compared with the one previously reported along with the available subsurface geologic information derived from seismic studies.

In Chapter-VI, an automatic gravity optimization technique, using the principles of inversion, is developed together with a GUI based software to estimate the depths to the basement topography above which the density contrast follows exponential decay with depth. The density interface between the sediments and underlying basement is simulated with a polygonal geometry with suitable number of vertices. Unlike the case with automatic modeling, this technique make use of the errors between the observed and model anomalies at all the locations to build the system of normal equations. These equations are solved for the improvements in the depth ordinates of the polygon. Depth parameters of the vertices are updated and the process repeats till the convergence criterion is fulfilled. The software, IN2DGREXP, reads the input

parameters and performs the business logic of the inversion to estimate the depths of the density interfaces in an automatic mode. The GUI capability of the software enables the interpreter to visualize the convergence of the solution with the iteration. The inversion when performed on the two gravity anomalies considered in the previous Chapter-V has yielded structural solutions that are more or less coincide with those obtained from automatic modeling. It was found that the inversion performs lesser number of iterations for proper convergence of the solution in comparison to automatic modeling.

All the softwares' presented in this thesis are platform independent.

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

\*\*\*

## List of Figures

Sl. No	Figure Number	Description	Page No
1	2.1	Schematic representation of a 2D listric fault source. AB is a profile across the strike along which the interpretation is intended	27
2	2.2	Structural relationships between Model, View and Controller	32
3	2.3	View module of INGREXP	33
4	2.4	(a) Observed and modeled noisy gravity anomalies, (b) assumed, initial and modeled structures. Prescribed Exponential Density Contrast Model (EDCM) is shown in the inset of (b)	36
5	2.5	(a) Error analysis of the gravity anomaly, (b) variation of misfit and various shape parameters with iteration number	39
6	2.6	Geologic and Bouguer gravity anomaly map of the Godavari sub-basin, India (after Chakravarthi and Sundararajan, 2004)	41
7	2.7	(a) Observed, initial and modeled gravity anomalies, (b) initial and estimated structures, Ahiri-Cherla master fault, Godavari sub-basin, India. Interpreted structure by Chakravarthi and Sundararajan (2004) is also shown for comparison. Simulated EDCM is shown in the inset of (b)	43
8	2.8	(a) Error analysis of the gravity anomaly, (b) variation of misfit and various shape parameters with iteration number, Ahiri-Cherla master fault, Godavari sub-basin, India	45
9	3.1	Schematic representation of a strike limited listric fault source. The detached downthrown block (hanging wall) is consisting of N number formations with differing thicknesses and densities. The limited strike length prevents the structure to represent as a 2D source.	51
10	3.2	(a) Gravity anomalies along two selected profiles, PP' and QQ', across a 2.5D strike listric fault source, whose geometry is shown in (b), (c) density-depth relationship within the hanging	54

		wall, (d) plan view showing the locations of the profiles, (e) differences in magnitudes of gravity anomalies observed along the profiles, PP' and QQ'.	
11	3.3	View module of FRGMLSTRK	57
12	3.4	Observed gravity anomaly (anomaly panel) and control points to describe fault plane geometry (structural panel)	58
13	3.5	Observed gravity anomaly (anomaly panel) and the geometry of an analytically defined fault plane (structural panel)	59
14	3.6	Observed gravity anomaly (anomaly panel) and control points selected for specifying the depths to the density interfaces within the structural panel	60
15	3.7	Observed gravity anomaly (anomaly panel) and selected control points to specify the densities within the density-depth panel	61
16	3.8	Observed and modeled gravity anomalies (anomaly panel), listric fault morphology (structural panel) and density-depth distribution (density-depth panel). The model gravity anomalies are also displayed in ASCII form in the ASCII layout	62
17	3.9	Observed and model gravity anomalies (anomaly panel), listric fault morphology (structural panel) with modified density-depth distribution (density-depth panel). The ASCII layout shows the modeled anomalies in ASCII form	63
18	3.10	Observed and modeled gravity anomalies (anomaly panel), representation of different formations within the hanging wall by different colours (structural panel), and modified density-depth distribution (density-depth panel).	63
19	3.11	Interpreted results in html format	64
20	3.12	Sample output	64
21	3.13	Gravity anomalies (anomaly panel) over a four-layered synthetic listric fault morphology (structural panel) with the fault plane described by a 9 <sup>th</sup> degree polynomial. Assumed density-depth data within the hanging wall is shown as a step line in the density-depth panel	65

22	3.14	Observed and modeled gravity anomalies (anomaly panel) and corresponding modeled listric fault source (structural panel) with the fault plane described by a 3 <sup>rd</sup> degree polynomial. The structure is modeled by modifying the depths of the density interfaces alone	67
23	3.15	Observed and modeled gravity anomalies (anomaly panel) and corresponding modeled listric fault source (structural panel) with the fault plane described by a 3 <sup>rd</sup> degree polynomial. The structure is modeled by modifying the densities of the formations alone	68
24	3.16	(a) Geological map of the Chintalpudi sub-basin, India (modified after Kaila et al., 1990), (b) Measured density contrast-depth data, (c) Gravity anomaly map, Chintalpudi sub-basin, India (after Chakravarthi and Sundararajan, 2007)	70
25	3.17	Observed gravity anomalies (anomaly panel) across the Aswaraopet master fault, Chintalpudi sub-basin, India. The model geometry shown in the structural panel is based on the DSS investigations (after Kaila et al., 1990). The gravity response of the model is shown by a solid line in the anomaly panel	72
26	3.18	Observed and model gravity response (anomaly panel) and modified depth structure (structure panel) from present modeling, Chintalpudi sub-basin, India	73
27	4.1	View module of GRAVLIS	80
28	4.2	(a) Observed and modeled noisy gravity anomalies, (b) four layered hanging wall system of a synthetic listric fault source with assumed and modeled fault planes described by 6 <sup>th</sup> and 2 <sup>nd</sup> degree polynomials, (c) assumed, initial and modeled densities. Depths of density interfaces are fixed during the inversion	83
29	4.3	(a) Error analysis between the observed and modeled gravity anomalies, (b) changes in misfit, coefficients of a 2 <sup>nd</sup> degree polynomial, and densities of subsurface formations against the iteration number.	86
30	4.4	(a) Observed and modeled noisy gravity anomalies, (b) four layered hanging wall system	88



		of a synthetic listric fault source with assumed and modeled fault planes described by 6th and 2nd degree polynomials, (c) assumed and estimated depths to density interfaces. Densities remain unchanged during the inversion	
31	4.5	(a) Error analysis between the observed and modeled gravity anomalies, (b) changes in misfit, coefficients of a 2nd degree polynomial, and depths of various density interfaces against the iteration number	90
32	4.6	(a) Observed and modeled gravity anomalies, (b) inferred geometry of the Aswaraopet master fault morphology, Chintalpudi subbasin, India, (c) measured, initial and estimated densities. Depths of density interfaces are fixed during the inversion	92
33	4.7	(a) Error analysis between the observed and modeled gravity anomalies across the Aswaraopet master fault morphology, Chintalpudi subbasin, India, (b) changes in misfit, coefficients of a 2nd degree polynomial, and densities of subsurface formations against the iteration number	94
34	4.8	(a) Observed and modeled gravity anomalies, (b) inferred geometry of the Aswaraopet master fault, Chintalpudi subbasin, India, (c) density-depth relationship. Anomalies are analyzed to estimate the depths of density interfaces	96
35	4.9	(a) Error analysis between the observed and modeled gravity anomalies across the Aswaraopet master fault morphology, Chintalpudi subbasin, India, (b) changes in misfit, coefficients of a 2nd degree polynomial, and depths to various density interfaces against the iteration number	98
36	5.1	Schematic representations of a sedimentary basin (black solid line) and its approximation by a polygon (blue solid line). B and C are the vertices of the K <sup>th</sup> side of the polygon. The colour gradation from yellow to red within the structure represents the increase in density with depth	107
37	5.2	(a) Gravity anomalies with uniform and exponential density contrast models, (b) geometry of assumed structure, (c) prescribed	110

		EDCM. The color gradation from yellow to red within the prism indicates the increase in density in case of EDCM	
38	5.3	Forward gravity modeling based on analytical and numerical approaches (a) and (c) over a vertical fault (b) and trapezoidal (d) models, prescribed exponential density contrast model (EDCM) and fitted quadratic density model (QDM) over two selected depth ranges are shown in (e) and (f). The color gradation from yellow to red within the structures indicates increase in density with depth	112
39	5.4	(a) Forward gravity modeling with analytical and numerical approaches based on uniform and exponential density contrast models (UDM & EDCM), (b) geometry of a sedimentary basin, (c) prescribed EDCM. The color gradation from yellow to red within the structure indicates increase in density with depth in case of EDCM	114
40	5.5	View module of MOD2DGREXP	118
41	5.6	(a) Observed and modeled noisy gravity anomalies by automatic modeling, (b) assumed and estimated structures, (c) prescribed EDCM, (d) changes in misfit with iteration, (e) error between observed and modeled anomalies at the end of 14th iteration. The color gradation from yellow to red within the structure indicates increase in density	120
42	5.7	(a) Observed and theoretical gravity anomalies by automatic modeling, San Jacinto graben, California. Modeled anomalies by Cordell (1973) are also shown for comparison, (b) estimated structures by the present method and Cordell's (1973), (c) derived density contrast-depth data and fitted EDCMs, (d) changes in r.m.s. error with iteration, (e) comparison of errors in anomaly observed in case of the present and Cordell's (1973) methods	122
43	6.1	View module of IN2DGREXP	131
44	6.2	(a) Observed and theoretical noisy gravity anomalies by inversion, (b) assumed and estimated structures, (c) changes in misfit with iteration, (d) error between the observed and modeled anomalies at the end of 7th iteration. The color gradation from yellow to red within the structure represents increase in density	134

		with depth	
45	6.3	(a) Observed and theoretical gravity anomalies by inversion, San Jacinto graben, California. Modeled anomalies by Cordell (1973) are also shown for comparison, (b) estimated structure by present inversion. Depth structures inferred by Cordell (1973), and by automatic modeling (Chapter-V) are shown, (c) changes in r.m.s. error with iteration, (d) comparison of residuals in anomaly observed in the present method and Cordell's (1973) method	136

## List of Tables

---

Sl. No	Table	Description	Page No
1	2.1	Assumed and estimated coefficients of polynomials, synthetic example	37
2	2.2	Estimated coefficients of polynomials, Ahiri-Cherla master fault Godavari sub-basin, India	42
3	3.1	Assumed and estimated coefficients of polynomials, synthetic example	66
4	3.2	Assumed and estimated parameters by interactive modeling, synthetic example	68
5	3.3	Coefficients of 6 <sup>th</sup> degree polynomial, Aswaraopet fault, Chintalpudi sub-basin, India	71
6	4.1	Assumed and estimated densities in case of synthetic example	82
7	4.2	Assumed and estimated coefficients of the polynomial, $\zeta(z)$ , synthetic example	84
8	4.3	Assumed and estimated depths of density interfaces, synthetic example	89
9	4.4	Measured and estimated densities, Chintalpudi subbasin, India	93
10	4.5	Measured and estimated depths to density interfaces, Chintalpudi subbasin, India	95
11	4.6	Estimated coefficients of the 2 <sup>nd</sup> degree polynomial, Chintalpudi subbasin, India	97

## List of research publications (SCI)

---

Chakravarthi, V., Rajeswara Sastry, S., **Pramod Kumar, M.**, 2014, 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 (Springer)*, 83, 577-585.

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, *Geophysica Internacional (Springer)*, 54, 49-65.

Chakravarthi, V., **Pramod Kumar, M.**, Ramamma, B., Rajeswara Sastry, S., 2015, Automatic gravity modeling of sedimentary basins by means of polygonal source geometry and exponential density contrast variation: Two space domain based algorithms, *Journal of Applied Geophysics (Elsevier)*, DOI: 10.1016/j.jappgeo.2015.11.007.

Chakravarthi, V., **Pramod Kumar, M.**, Ramamma, B., Rajeswara Sastry, S., 2015, Gravity anomaly interpretation of 2D listric fault morphologies using exponential density contrast model: A space domain technique, Under review with *Arabian Journal of Geosciences (Springer)*.

## Introduction

---

### 1.1 General

Geophysical methods are the inevitable tools for the investigation of subsurface geology at micro, macro and global scales. In addition, these methods occupy a strategic role in the exploration programs for natural resources. Several geophysical methods viz., gravity, magnetic, electrical, electromagnetic, seismic, radiometric etc. are in vogue to map and explore the concealed geology. Each geophysical method exploits the corresponding physical property contrast to produce relevant geophysical signals/anomalies for exploring the subsurface. These geophysical signatures/anomalies shall be collected on the ground surface, boreholes and also from airborne platforms and subsequently processed and analyzed in terms of the parameters to quantify the concealed geology in detail.

However, in deciding which geophysical method or combination of methods to use under a given set of circumstances, close attention must be given to the inherent suitability of each method for investigating the problem under consideration. It may not be feasible to use the most suitable one and, as a

matter of expediency, a second or third choice procedure may also have to be adopted.

Undoubtedly, sophisticated and state-of-the-art instruments and computer aided technology made the geophysical field data acquisition, processing and interpretation effective and elegant, but at times, the process of interpretation continues to be cumbersome because of the subsurface heterogeneity and complexity. Although, interpretation techniques based on the thumb rules, curve matching etc. are in vogue; the development of highly powerful algorithms almost surpass the existing old techniques.

Interpretation of geophysical data involves two aspects; the first one is to estimate the parameters of the causative body, which is responsible for generating the anomalous field and the second one is to translate the interpreted geophysical model in terms of concealed geology. However, quantitative interpretation of field measurements taken at the surface is not unique because of the fact that the observed anomalies on the topography are the cumulative effect of useful signal and noises of different origin. Moreover, the observed anomaly on the topography can be equally explained by a number of sources. Such ambiguity and uncertainty in the interpretation can be reduced to a large extent by incorporating additional source of information obtained from surface outcrops, bore holes, and by other geophysical methods.

Gravity exploration, which is the subject matter of the thesis, comes under the category of passive geophysical methods. It does not rely on

controlled sources but seeks out naturally occurring variations in the earth's gravity field. A body rotating with the earth experiences the gravitational force of the masses of the earth, other celestial bodies, as well as the centrifugal force due to the earth's rotation. If the effects of centrifugal force and that of celestial bodies are removed the left out component becomes gravitation. The force of attraction between two point masses  $m_1$  and  $m_2$  is directly proportional to the product of their masses and inversely to the square of the distance between the centers of respective masses and can be expressed as

$$F = \frac{-Gm_1m_2}{r^2} \frac{k}{r}, \quad (1.1)$$

where,  $F$  is the force on  $m_2$ ,  $r$  is the distance between  $m_1$  and  $m_2$  and  $G$  is the universal gravitational constant. The minus sign indicates that the force is always attractive. The vector  $k$  may be expressed by the position vectors  $r$  and  $r'$  in the Cartesian coordinate system as

$$k = r - r', r^T = (X, Y, Z), r'^T = (X', Y', Z'), \quad (1.2)$$

with the magnitude of

$$k = \sqrt{(X - X')^2 + (Y - Y')^2 + (Z - Z')^2}. \quad (1.3)$$

By setting the mass at the attracted point to unity, the above equation transforms into the gravitational acceleration represented by

$$g = \frac{-Gm}{r^2} \frac{k}{r}. \quad (1.4)$$



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

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

The mass element  $dm$  can be expressed as

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

where,  $\sigma$  is the density of the volume element,  $du$ . Because the gravitational field is invariant to rotations

$$\text{curl } g = 0. \quad (1.7)$$

Also, the vector  $g$  may be expressed as the negative gradient of the potential  $U$  (Sigl, 1985),

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

For a point mass  $m$ , the gravity potential can be expressed as

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

For the earth,

$$U = G \iiint \frac{dm}{r} = G \iiint \frac{\sigma}{r} du, \lim_{r \rightarrow \infty} U. \quad (1.10)$$

From the above relations, it is clear that once the density function  $\sigma$  is known for the earth, the gravitation can be calculated as a function of the position.

The acceleration of gravity in the direction of  $z$ -axis can be derived from equation (1.9) with reference to the origin of the Cartesian coordinate system as

$$g_z = -\frac{\partial U}{\partial z} = G\sigma \iiint \frac{z}{r^3} dx dy dz, \quad (1.11)$$

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

If the body is very long in one direction, say, along the  $y$ -axis and has an uniform cross-section in the  $xz$ -plane, the gravity attraction can be obtained from a logarithmic 2D potential as

$$U = 2G\sigma \iint \log\left(\frac{1}{r}\right) dx dz. \quad (1.12)$$

Here,  $dx dz$  is the cross-sectional area of a 2D element within the body. The gravity effect of the 2D body can be expressed as

$$g_z = -\frac{\partial U}{\partial z} = 2G\sigma \iint \frac{z dx dz}{r^2}, \quad (1.13)$$

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

## 1.2 Exponential Density Contrast Model (EDM)

Equation (1.11) and (1.13) can be used to calculate the gravity effects of 3D and 2D geologic sources under the assumption that these sources possess uniform density throughout. Unequivocally, the success of the gravity method is largely dependent on the existence of significant lateral density contrast between the source/sources of interest with the adjacent formations. The dimensions of the source and its depth of occurrence also influence the magnitude of gravity

anomaly on the topography. Hence, the knowledge of the density of rocks both at the surface and subsurface is always essential not only for applying a few gravity corrections to the gravity measurements but also to obtain reliable geologic interpretations of the gravity anomalies.

In case of sedimentary rocks the density is rarely uniform (see for e.g., Athy, 1930; Hamilton and Menard, 1956; Becking and Moore, 1959; Prozorovich, 1960; Gealy, 1969; Hamilton, 1976; Bachman and Hamilton, 1976; Helmberger et al., 1979; Donato and Tully, 1981; Dimitropoulos and Donato, 1981; Foucher et al., 1982; Zervos 1987; Holliger and Klemperer, 1989; Thorne and Watts, 1989; Artemjev et al., 1994; Angell et al., 1997; Chaika and Williams, 2001; Azeglio et al., 2010; Wang et al., 2011; Gu et al., 2014; Martinez et al., 2014).

Athy (1930) was the first researcher to express the relations between the depth of burial and the density, porosity, and compaction of different types of sediment by exponential equations. Manger (1963) had documented large amount of measured data pertaining to the porosity and density measurements of sedimentary rocks and concluded that the porosity of sandstones generally decreases whereas the density increases with depth of burial and age. From an extensive study and analysis of the density logs measured in 435 deep wells in Western Canada, Maxant (1980) had shown that the density-depth relationship seldom follows a linear trend; and in the case of shale, the correlation between density and depth could be explained by exponential density-depth relationship. Dickinson (1953), Dallmus (1958), Storer (1959), McCulloh (1967), Eaton

(1969), Rieke et al. (1974), Castagna et al. (1993) have demonstrated that the measured densities for shale as a function of depth showed more or less similar behavior, although the samples were collected from a wide variety of locations with different geologic settings and histories.

On the other hand, Cowie and Karner (1990) have demonstrated from the measured density-depth data of different stratigraphic units in many sedimentary basins that the sediment densities exhibit a wide range but the mean density clearly increases with depth with the highest rate in the top few hundred meters. They have also showed that the density of sediments near the depocentres approaches to that of the basement rocks whereas near the basin margins, the sediments portray relatively lower density.

Recently, Tenzer and Gladkikh (2014), based on the analysis of density samples taken from 716 drill sites of the Deep Sea Drilling Project (DSDP), showed that the density increases nonlinearly with the increasing sediment depth due to compaction. They have also proposed theoretical models to compute the density contrasts of the interfaces between ocean-sediment and sediment-bedrock.

Cordell (1973), Cai and Zhdanov (2015) argue that the density-depth relationship of sedimentary rocks, in general, does not strictly follows any mathematical formulation because of the influence of several geologic factors such as compaction, stratigraphic layering, cementation, facies change, diagenesis etc. Cordell (1973) had established from the actual measurement of

sedimentary rock density from deep bore holes that the density contrast decreases drastically at shallow depths and less progressively at deeper depths. Such variation in density contrast of sedimentary rocks with depth could be effectively modeled by an exponential law (Cordell, 1973; Granser, 1987; Chai and Hinze, 1988; Chappel and Kusznir, 2008; Chakravarthi et al., 2015a, 2015c). Hence, the use of Exponential Density Contrast Model (EDCM) in the analysis of gravity anomalies ensures more reliable interpretations.

The exponential density contrast model is mathematically defined as (Cordell, 1973)

$$\Delta\rho(z) = \Delta\rho_0 e^{-\lambda z}, \quad (1.14)$$

where,  $\Delta\rho(z)$  is the density contrast of the sediments at a given depth,  $z$ ,  $\Delta\rho_0$  is the density contrast at the surface/topography and  $\lambda$  is a decay factor expressed in reciprocal length units. The values of  $\Delta\rho_0$  and  $\lambda$  can be obtained by fitting equation (1.14) to the known density contrast-depth data of subsurface geology.

However, the major intricacy associated with the exponential density contrast model is that analytical expressions for the gravity anomalies could not be derivable in the space domain even for simple geophysical models (Chakravarthi and Sundararajan, 2004). Although, closed form solutions are possible in the frequency domain with this density contrast model to realize forward modeling (Granser, 1987; Chai and Hinze, 1988; Chappel and Kusznir, 2008), significant truncation errors would popup in the analysis when the

anomalies transform back to the space domain (Chakravarthi and Sundararajan, 2007).

In this thesis, new approaches have been formulated in the spatial domain to calculate the gravity anomalies of geologic structures using an exponential density contrast model by judiciously combining both analytical and numerical approaches. New interpretation techniques with relevant softwares are then developed to recover the structures from the observed gravity anomalies. From the known density-depth information, the Exponential Density Contrast Model (EDCM) is constructed for the field data presented in chapters II, V and VI.

### **1.3 Gravity corrections**

The end product of a gravity survey yields a set of gravimeter readings and a set of station elevations measured with reference to the mean sea level or geoidal surface. Absolute gravity is established at each observation by connecting the field gravity meter readings to a base station. These absolute gravity values represent the cumulative effects of the sources being looked for, in addition to the earth's topography, its non-sphericity of shape and its centrifugal force. In principle, normal gravity is calculated on the spheroidal surface of the earth, projecting it to the level of observation and adding to it the gravity effect of the masses between the spheroid and the ground surface. This can be realized by applying a series of corrections viz., normal or latitude correction, free-air correction, Bouguer correction and topographic correction

to the measured gravity data. A detailed account of the application of gravity corrections is given by Rao and Murthy (1978). When the elevation differences between the field stations are small, the anomalies are deemed to have been made on a mean horizontal plane coinciding with the topography. Finally, the Bouguer gravity anomalies are presented in the form of contour maps and stacked profiles before being subjected to interpretation.

## **1.4 Interpretation**

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

### *1.4.1 Qualitative interpretation*

In qualitative interpretation, gravity highs and lows are identified on a contour map and their axes are correlated with the known surface or subsurface geology and the correlations are extrapolated to poorly mapped areas. Further, the nature and characteristics of the anomalies are studied to identify the source geometry, its orientation besides its approximate depth of occurrence.

Two-dimensional bodies of large strike lengths are represented by anomaly contours elongated roughly parallel to the strike of anomalous bodies; conversely the direction of elongation of contours or their axis is a measure of

the strike of the anomalous body. Three-dimensional bodies produce elliptical or closed contours.

When the anomalies are presented in the form of profiles, typical characteristics of individual profiles are being identified and correlated with those on neighboring profiles. Characteristic features persist on neighboring profiles indicate the strike/orientation of the anomalous body. Fault structures create dislocation or complete absence of characteristics on neighboring profiles. Widths of individual anomalies and their sharpness decide the order of depth or horizontal dimensions of the body.

#### *1.4.2 Regional and residual anomaly separation*

The measured Bouguer gravity anomalies on the topography are the cumulative gravity effects of the sources distributed both laterally and vertically. The contribution of gravity field due to the source/sources of interest is known as the residual gravity anomalies, which are high frequency in nature. The sources located at deeper levels/far away from the source/sources produce low frequency gravity signals, known as regional gravity anomalies. The gravity field due to sources above the source of interest is known as noise and can be identified and removed from the Bouguer anomalies owing to its conspicuous and random nature of occurrence. The process of separating the regional gravity anomalies from the residuals is known as the regional and residual anomaly separation.



Among many others, the methods based the graphical and smoothing techniques, empirical gridding, second derivative etc. (Telford et al., 1990), upward continuation (Jacobsen, 1987; Pacino and Introcaso, 1988), minimum curvature (Mickus et al., 1991), orthogonal polynomial fitting (Grant, 1957; Forsythe, 1957; Spitz, 1966; Merriam and Cocke, 1967; Beltrao et al., 1991; Agarwal and Sivagi, 1992), linear filtering (Strakhov, 1964; Strakhov and Lapina, 1967, Naidu, 1966; Naidu, 1967; Spector and Grant, 1970; Byerly, 1965; Sax, 1966; Lavin and Devane, 1970), nonlinear filtering (Naudy and Dreyer, 1968), Band pass filtering (Blakely, 1995), Wiener filtering (Pawlowski and Hansen, 1990), Green's equivalent layer (Pawlowski, 1994), fractals (Chapin, 1996), cellular neural networks (Albora et al., 2001); finite elements (Mallick and Sharma, 1999; Agarwal and Shalivahan, 2010), Hartley transform (Kadirov, 2000), multiscale edge and iterative lateral continuation and subtraction analysis (Boschetti et al., 2004), eigenimage extraction (Ganguly and Dimri, 2013), and higher order polynomial fitting (Chakravarthi et al., 2013a) have gained importance for separating regional gravity anomaly from the Bouguer anomalies. Nevertheless, known geology plays a decisive role in deciphering the best regional trend on a Bouguer anomaly map. The residual anomalies separated from the Bouguer gravity anomalies are subjected for the quantitative analysis.

#### *1.4.3 Quantitative interpretation*

Quantitative interpretation of gravity anomalies aims at estimating the parameters of geophysical models, which in turn explain the geology.

However, the interpretation of gravity anomalies is non-unique and ill-posed in the sense that the measured gravity anomalies on the plane of observation can be explained by a variety of density distributions as explained by Roy (1962), Backus and Gilbert (1967, 1968), and Blakely (1995).

To transform an ill-posed problem into a well-posed one, one may choose either to reduce the information demanded or introduce a *priori* information in the modeling space. Parker (1974,1975), Huestis and Parker (1977), Ander and Huestis (1987) have adopted the first approach to obtain the estimates for the lower and upper bounds of the physical property, the depth to the top and the thickness of the source. In contrast, the second approach of introducing a *priori* information could be followed provided the information were properly translated from the geological setting (Silva et al., 2001).

To introduce a *priori* information, one needs to consider an interpretation model either in physical property mode or geometric mode. In the former case, the subsurface containing the anomalous mass shall be viewed as an ensemble of elementary cells of fixed size and the density of each cell is estimated (Li and Oldenburg, 1998). In the later case, the density/density contrast of the anomalous body shall be specified/formulated and nonlinear operators are designed to determine the geometry of the source of the anomaly (Tarantola, 2005). Among the two approaches, the geometric mode is popular and being adopted by many researchers to develop interpretation algorithms to analyze the gravity anomalies (Murthy, 1998; Chakravarthi, 2003; Chakravarthi et al., 2013b, 2014, 2015a, 2015b, 2015c). Using the information

derived from drilling/other geophysical data in the model space would further reduce the degree of uncertainty in the interpretation.

The nature and characteristics of the gravity anomalies very often reveal whether the anomalous source is a 2D, 2.5D and/or 3D structure (Rao and Murthy, 1978; Murthy, 1998). Suitable interpretation strategies can therefore be applied to interpret the gravity data.

### **1.5 Review of existing methods**

Inverse problems are an important area of geophysical research because one has to make the quantitative estimates about the subsurface of the earth (i.e. to determine the unknown model space parameters) from a known/measured set of observations on the surface of the earth.

The process of quantitative interpretation involves three important aspects namely, i) parameterization of model space, ii) forward modeling, and iii) optimization or inversion. A set of appropriate size and shape factors describe the model space. Physical laws governing the model space supplemented with relevant mathematics provide a means to compute the model gravity response. The optimization/inversion process recovers the unknown model space from the observed gravity anomalies.

Several techniques are being developed to analyze the gravity anomalies for subsurface structures in both spatial and wave number domains. In classical trial and error methods, approximate model space is constructed from a set of

initial/guess parameters and the corresponding gravity response shall be calculated. Corrections are applied to the model parameters to minimize the data misfits between the observed and model/theoretical anomalies. This approach is repeated with several possible combinations of shape and size parameters till a satisfactory match between the observed and computed response is realized. Such an exercise is attempted with fast computing machines by means of suitable codes providing a comfortable graphical user interface and enabling a fast and easy way of changing the parameters of model space (Chakravarthi et al., 2014).

In wave number domain methods, Odegard and Berg (1965), Sharma and Geldart (1968), Collins et al. (1974), Mohan (1978), Bhimasankaram et al. (1977), Murthy and Rao (1980), Chacko and Battacharya (1980), Mareschal (1985), Rao et al. (1993), Annecchione et al. (2001) have proposed techniques based on the Fourier transforms, Sundararajan et al. (1983) have used the Hilbert transform, Shaw and Agarwal (1990) have applied Walsh transforms, Sundararajan and Brahman (1998) have adopted the Hartley transform to interpret the gravity anomalies. Sundararajan et al., (2000), Al-Garni et al. (2010) have proposed techniques based on the Sundararajan transform to analyze the potential field anomalies of simple geophysical models. Generally, in spectral methods the gravity response of a source is analyzed within a moving window of predetermined size. The size of the window shall be determined based on an expected depth to the source, which is generally unavailable in advance (Chávez et al., 1999). Furthermore, significant errors

would result in depth estimates in case the spectrum becomes complicated (Odegard, 2011).

On the other hand, the conventional Euler deconvolution method (Hood, 1965; Thompson, 1982; Wilsher, 1987, Corner and Wilsher, 1989; Reid et al., 1990; Klingele et al., 1991; Marson and Klingele, 1993; Fairhead et al., 1994; Huang et al., 1995; Reid, 1997; Zhang, 2000; Hu et al., 2011) is being widely used now-a-days to estimate source depths in a fast manner. However, this method invariably requires the information of the Source Structure Index (SSI). In order to choose SSI, the source needs to be approximated by specific geometries such as sphere, cylinder, etc. (Lafehr and Nabighian, 2012). In addition, wrong choice of SSI severely affects the depth estimates of the anomalous sources. Although, an extended Euler deconvolution (Mushayandebvu et al., 2001) can be used to simultaneously estimate the source depth and the structure index, this method is difficult adopt to deal with complex structures (Lafehr and Nabighian, 2012). Above all, the Euler deconvolution and spectral methods do not provide a direct comparison between the observed and modeled gravity anomalies; hence it becomes difficult to appraise the correctness of the solution.

The calculus based interpretational techniques such as the Newton and gradient methods solve the gravity inverse problems analytically if the functions describing the forward problem are differentiable (Tarantola, 2005). These methods are generally suitable if the objective function has one local minimum.

In case, the objective function is associated with several local minima then the global search optimization methods are preferred. These methods are categorized into two groups namely, guided and non-guided. The neural networks and evolutionary algorithms come under the guided methods, whereas the Monte Carlo methods fall under the category of non-guided methods. The neural networks operate on the basis of neurons to self organize and learn in respect of given external parameters (Brown and Poulton, 1996; Guang et al., 1998; Baan and Jutten, 2000; Osman et al., 2006, 2007; Leite and Filho, 2009). The evolutionary algorithms use the known principles of biological evolution to explore the whole parameter space for the best possible solution for a given inverse problem (Boschetti et al., 1997; Roy et al., 2002; Yao et al., 2003). The Monte Carlo methods search the model space randomly for the optimum solutions. These methods are very well suited if the number of unknown parameters to be solved are limited in number (Mosegaard and Tarantola, 1995; Tarantola, 2005).

Techniques based on wavelet transforms (Marlet et al., 2001; Hu et al., 2011; Oruç, 2014), 2D binary grids (Krahenbuhl and Li, 2006), tree-based geometry representation (Wildman and Gazonas, 2009), analytic signals (Beiki, 2010), Eigenvectors (Beiki and Pedersen, 2010) are also available to interpret the gravity anomalies. However, the approximation of uniform density for the anomalous source/sources in the enlisted methods is seldom valid particularly when reference is made to the structures associated with the sedimentary rocks.

In recent past, techniques based on tunneling algorithm (Levy and Montalvo, 1985, Mohan et al., 1986, Moharir, 1990); simulated annealing (Mundim et al., 1998; Nagihara and Hall, 2001; Roy et al., 2002; De Vicente et al., 2003; Jingxin et al., 2013), ant colony optimization (Dorigo and Blum, 2005; Gupta et al., 2011; Srivastava et al., 2013), particle swarm optimization (Kennedy and Eberhart, 1995; Eberhart and Kennedy, 1995; Shalivahan and Agarwal, 2010; Touthmalani, 2013; Pallero et al., 2015) are also found in literature. These strategies overcome the limitations associated with local optimization techniques.

Simulated Annealing (SA) simulates annealing process in which a substance is heated above its melting temperature and then gradually cools to produce the crystalline lattice, which minimizes its energy probability distribution. Ant Colony Optimization (ACO) is another evolutionary optimization algorithm which is inspired by the pheromone trail laying behavior of real ant colonies. On the other hand, Particle Swarm Optimization (PSO) is a population based stochastic optimization technique inspired by social behavior of bird flocking or fish schooling searching for food. PSO shares many similarities with genetic algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However; unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. These algorithms are increasingly finding application in geophysical

problems including potential field inverse problems, particularly in simultaneous inversion of gravity and magnetic data in order to reconstruct the shape of buried geological bodies; however, they are far more expensive both in terms of computational time and memory requirements.

## **1.6 Aim and scope of the thesis**

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

Chapter-I deals with general introduction, a brief account of various corrections to the measured gravity data, general principles of qualitative interpretation, concept of exponential density contrast model and a brief review of the earlier work on the development of gravity interpretation techniques etc.

In Chapter-II, an automatic inversion technique and related GUI based software to analyze the gravity anomalies of 2D listric fault morphologies with Exponential Density Contrast Model (EDCM) are dealt with. The reliability and applicability of the proposed technique and software are demonstrated on a synthetic model and also substantiated with real field data pertaining to the Ahiri-Cherla master fault of the Godavari sub-basin, India.



A semiautomatic/interactive modeling technique of gravity anomalies due to 2.5D strike limited listric fault sources with analytically defined fault planes and arbitrarily varying density-depth relationship is dealt with in Chapter-III along with relevant software. The effects of strike length of the structure and offset of the profile on the magnitude of gravity anomaly are described in detail. Both with a synthetic model of a 2.5D listric fault source and real field data measured from the eastern margin of the Chintalpudi sub-basin in India, the technique is demonstrated.

Chapter-IV deals with the development of an inversion technique and associated software in JAVA to simultaneously estimate the geometries of non-planar fault ramps and the parameters pertaining to either densities or thicknesses of formations within the detached hanging wall systems of 2.5D listric fault sources. Anomalies attributable to a synthetic model of a 2.5D listric fault source in the presence of pseudorandom noise and those observed across the Aswaraopet master fault of the Chintalpudi sub-basin, India are interpreted using the proposed inversion to demonstrate its applicability.

In Chapter-V, an automatic interpretation algorithm to analyze the gravity anomalies of 2D sedimentary basins with exponential density contrast model is dealt with along with relevant software programmed in JAVA. The algorithm and software operate on the principles of automatic modeling. The sediment basement interface is described with polygonal source geometry. The analysis is supported by a synthetic model and real field data across the San Jacinto graben, California.

In Chapter-VI, the principles of inversion are used to formulate an automatic optimization scheme and associated software in JAVA to estimate the depths of 2D density interfaces from the observed gravity anomalies. The density contrast within the sedimentary load follows exponential decay with depth. The successful interpretation of two gravity anomaly profiles, one synthetic and the other real, testifies the applicability of the proposed inversion.

In all the synthetic models presented in chapters II to VI the interpretations are compared with the assumed parameters, and in case of field data analysis the interpreted results are judged against the existing borehole /available geologic information.

A comprehensive conclusion of the entire work presented in the thesis is enumerated in Chapter-VII along with the scope for future research.

## Analysis of gravity anomalies of 2D listric fault morphologies using a prescribed exponential density contrast model\*

### 2.1 General:

The primary goal of studying detailed gravity data is to provide better understanding of the subsurface geology. The gravity anomaly across a fault increases progressively to a maximum value over the uplifted side and a low over the downthrown block because the displacement of material causes a horizontal density contrast across the fault plane.

More often than not, the crustal extension is often acclimatized by high-angle faults that become almost listric at depth. These listric faults are curved normal faults in which the fault surface is concave upwards because the main detachment fracture follows a curved path rather than a planar path. The study of listric fault geometries is important because the movement along this type of fault is instrumental in forming important structural traps for oil and gas such as rollover anticlines and upthrown-fault-block closures etc.

---

\* Under review with *Arabian Journal of Geosciences (Springer)*

Evidences for listric fault morphologies on the continental extensional regimes are plenty, to name a few, see for e.g., the Welshman's Rock, eastern Rhum (Emeleus, 1981), the North Sea Basin (Gibbs, 1983), the Corsair fault of offshore Texas (Christiansen, 1983), the Basin and Range (Wernicke and Burchfiel, 1982; Smith and Bruhn, 1984), the Murre fault of offshore Newfoundland (Tankard and Welsink, 1987), the Cordilleran fold and thrust belt (Constenius, 1996), the Cascadia continental margin (McNeill et al., 1997), Eskimo Lakes Fault Zone (Goussav et al., 2006).

Surface geologic studies do not easily reveal the listric nature of the faults because many a times the outcrop conditions prevent adequate geometric control of the fault planes. On the other hand, step like gravity anomalies can be observable if the detached rock masses on either side of such fault planes could create measurable lateral contrasts in rock densities. These gravity anomalies can be analyzed to estimate the parameters of such fault sources after properly accounting for regional gravity background.

Gravity anomalies due to two-dimensional bodies are elongated in one horizontal direction so that the anomaly length in this direction is at least twice the anomaly width. Such anomalies may be interpreted in terms of 2D structures, which theoretically extend to infinity in the elongate direction (strike) by using profiles at right angles to the strike direction (Kearey et al., 2002; Chakravarthi et al., 2015a).

## **2.2 Status of existing interpretational techniques**

Many techniques are in vogue to analyze the gravity anomalies of fault structures. For e.g., Geldart et al. (1966) had proposed a curve matching technique, whereas Paul et al. (1966) demonstrated the use of the upward continuation to estimate the parameters of a fault structure. Chacko and Bhattacharya (1980), Murthy and Rao (1980), Pal (1981) have developed techniques based on the Fourier Transform, Sundararajan et al. (1983) and Mohan et al. (1986) had proposed schemes using the Hilbert transform and the Mellin Transform respectively to analyze the gravity anomalies of faulted beds.

Based on a least squares minimization approach Gupta (1983) had determined the depth of a buried faulted structure from the observed gravity anomalies. Thanassoulas et al. (1987) proposed a method and computer program to estimate the parameters of faulted beds. Abdelrahman et al. (1989), Gupta and Pokhriyal (1990) have developed methods using the amplitudes of maximum positive and negative gravity anomalies to determine the dip of the fault planes. Murthy and Krishnamacharyulu (1990) used the Marquardt algorithm to analyze the gravity anomalies of fault structures. McGrath (1991) had proposed a method based on lateral offsets of the zero-crossover point of the second horizontal derivative of an upward-continued gravity profile to estimate the dip, vertical extent, and location of the model boundary. Rao et al. (2003) used generalized inversion and single value decomposition techniques to model the gravity anomalies. On the other hand, Abdelrahman et al. (2003) have developed two least-squares approaches to determine the depth and

amplitude coefficient of a buried faulted thin slab, successively from numerical first, second, third, and fourth derivative anomalies obtained from the observed gravity data using filters of successive graticule spacings. Stavrev and Reid (2010) used the concept of extended Euler homogeneity of potential fields to analyze the gravity anomalies of a faulted slab with large thickness relative to its depth. Aydogan (2011) had developed a technique based on the convolution between the templates obtained from the second horizontal derivative of the theoretical anomaly due to a truncated horizontal plate and the gravity anomaly to locate the positions of vertical or near vertical faults. Abdelrahman et al. (2013) developed a semi-automatic least squares method that uses first moving average residual gravity anomalies using filters of successive window lengths, whereas Essa (2013) proposed a technique using variance analysis to estimate the depth and dip angle of a buried fault structure. In recent past, Abdelrahman and Essa (2015) have developed three different least squares minimization approaches to analyze the gravity anomalies of dipping fault structures.

The practical utility of all the enlisted techniques becomes limited to analyze the gravity anomalies of the boundary faults associated with sedimentary basins because i) these faults are strongly curved in cross-section (Jackson and McKenzie, 1983; Gans et al., 1985; Jackson, 1987; Janecke et al., 1998; Brady et al., 2000; McKenzie and Jackson, 2012; Chakravarthi et al., 2015a), and ii) the density of sedimentary rocks is seldom uniform but increases with depth rapidly at shallow depths and less rapidly at progressively greater depths.

In this direction, although a few density functions were proposed and used by researchers in their interpretation strategies to analyze the gravity anomalies of fault structures viz., quadratic (Rao, 1985), linear (Sundararajan and Brahmam, 1998); these methods also presume planar surfaces for the fault planes. Moreover the linear density function, which is suitable to simulate the sediment density at large depths fails to explain the density variation at shallow depths (Chakravarthi and Sundararajan, 2007), whereas the quadratic density function falls short to replicate the true sediment density at depths as demonstrated by Chakravarthi (2009).

It is also to be realized that the use of existing commercial software, such as GM-SYS (Northwest Geophysical Associates Inc. 2004), to model the gravity anomalies of listric fault geometries is also problematic because a large number of constant density bodies are required to adequately explain the exponential density structure within the hanging wall (Zhou, 2013; Chakravarthi et al., 2015a).

For the above said reasons it is necessary to develop a new inversion technique coupled with software, which could overcome the drawbacks associated with the existing algorithms described in the text.

In this Chapter, two schemes are developed; one for realizing forward gravity modeling of a listric fault source from a known set of size and shape parameters and the other to perform automatic inversion on the observed gravity anomalies to recover the model space. In the later case, the unknown

parameters to be estimated to quantify a listric fault source are: i) depths to the top and bottom surfaces of the structure, ii) distance to the origin of the fault plane from an arbitrarily chosen reference on the profile, and iii) the geometry of the fault plane. Based on the proposed methodology, a GUI based software, GRIN2DFL, coded in JAVA to analyze the gravity anomalies is developed. Finally, the applicability of the proposed inversion is demonstrated on both synthetic and real field gravity anomalies.

### 2.3 Forward gravity modeling – Theoretical considerations

In a Cartesian coordinate system the geometry of listric fault model, shown in  $xz$  cross-section in Figure 2.1, is bounded by upper and lower horizontal planes  $z = z_T$ , and  $z_B$ , semi-infinite in the positive  $x$ -axis direction, bounded on the other side by a non-planar fault ramp defined by a function,  $\zeta(z)$ , and extends to infinity in the positive and negative  $y$ -axis directions. Presuming that the footwall remains undeformed and intact, the gravity anomaly of such a structure,  $g_{mod}(X_j, Z_j)$ , at any point,  $P(X_j, Z_j)$ , on a profile,

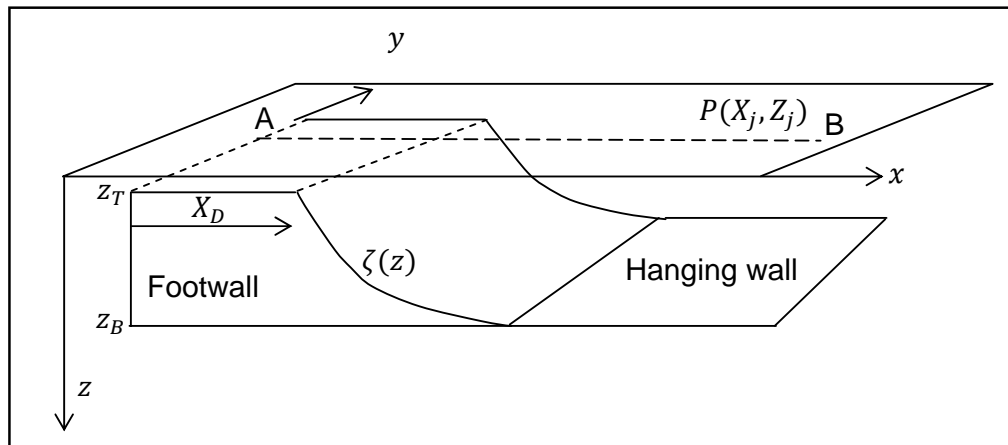


Figure 2.1 Schematic representation of a 2D listric fault source. AB is a profile across the strike along which the interpretation is intended



AB, outside the source region can be expressed as (Chakravarthi et al., 2015a),

$$g_{mod}(X_j, Z_j) = 2G \int_s \frac{\Delta\rho(z)\overline{z - Z_j}dxdz}{\overline{x - X_j}^2 + \overline{z - Z_j}^2}, \quad (2.1)$$

where,  $dxdz$  is the cross-sectional area of a representative element within the source; whose coordinates are given by  $(x, z)$ . Also,  $\Delta\rho(z)$  is the density contrast of sediments at any depth,  $z$ , within the hanging wall system represented by equation (1.14).

Substituting equation (1.14) for  $\Delta\rho(z)$  and upon integration with respect to  $x$ , equation (2.1) becomes

$$g_{mod}(X_j, Z_j) = 2G\Delta\rho_0 \int_{z_T}^{z_B} e^{-\lambda z} \left\{ \frac{\pi}{2} - \tan^{-1} \left\langle \frac{\zeta(z) - X_j}{z - Z_j} \right\rangle \right\} dz, \quad (2.2)$$

where,

$$\zeta(z) = \sum_{k=0}^{N1} f_k z^k. \quad (2.3)$$

Here,  $f_k$  represents a set of coefficients and  $N1$  stands for the degree of polynomial. It is to be noted that closed form analytical solution does not exist for equation (2.2) in the space domain; however, it can be solvable by means of a numerical integration. In this case, the Simpson's rule is used to solve equation (2.2). This method of solving the equation has an added advantage that it could overcome possible singularities. It is obvious that by letting  $\lambda$  to zero in equation (2.2), gravity anomalies of the structure with uniform density can be realized.

## 2.4 Inversion of gravity anomalies

Inversion of gravity anomalies of listric fault sources is tantamount to a mathematical exercise of trying to fit the modeled gravity anomalies,  $g_{mod}(X_k, Z_k)$ ,  $k = 1, 2, \dots, N_{obs}$ , to the observed ones,  $g_{obs}(X_k, Z_k)$ ,  $k = 1, 2, \dots, N_{obs}$ , in an iterative approach employing the principles of least squares. Here,  $N_{obs}$  stands for the number of observations. For the optimum parameters the modeled anomalies closely mimic the observed ones.

The advantage of the present algorithm is that it estimates the initial parameters of the source based on a few selective anomalies and subsequently improves them automatically by minimizing the errors between the observed and model gravity anomalies in an iterative approach.

To start with, the algorithm estimates one half of the maximum anomalous field and finds the corresponding distance,  $X_D$ , (Figure 2.1) on the profile from a chosen reference (Murthy, 1998; Chakravarthi et al., 2015a). The algorithm assigns this parameter value to the constant term,  $f_0$ , of the polynomial,  $\zeta(Z)$ , and sets the remaining coefficients to zero. Approximate depth to the bottom of the structure,  $z_B$ , is calculated based on the Bouguer slab formula (Cordell, 1973; Chakravarthi et al., 2015a) as

$$z_B = \frac{-1}{\lambda} \log \left[ 1 - \frac{\lambda g_{obsmx}}{2\pi G \Delta \rho_0} \right], \quad (2.4)$$

where,  $g_{obsmx}$  is the maximum observed gravity anomaly (absolute) on the profile. A small value of 1E-04 km is assigned to the depth parameter,  $z_T$ .

Using these initial parameters of the structure the algorithm calculates the modeled gravity anomalies,  $g_{mod}(X_j, Z_j)$ ,  $j = 1, 2, \dots, N_{obs}$  using equation (2.2) and quantifies the misfit ( $J$ ) between the observed and modeled anomalies based on the expression

$$J = \sqrt{\frac{\sum_{j=1}^{N_{obs}} [ERR(x_j, z_j)]^2}{N_{obs}}}, \quad (2.5)$$

where,  $Err(X_j, Z_j) = g_{obs}(X_j, Z_j) - g_{mod}(X_j, Z_j)$ ,  $j = 1, 2, \dots, N_{obs}$ .

The difference between the observed and modeled gravity anomalies at any observation,  $(X_j, Z_j)$ , on the profile can be expressed as a cumulative of the products of partial derivative of the anomaly with respect to each unknown parameter to be estimated and corresponding increments as (Chakravarthi et al., 2015a)

$$\begin{aligned} g_{obs}(X_j, Z_j) - g_{mod}(X_j, Z_j) \\ = \frac{\partial g_{mod}(X_j, Z_j)}{\partial z_B} dz_B + \sum_{k=0}^{N1} \frac{\partial g_{mod}(X_j, Z_j)}{\partial f_k} df_k, \end{aligned} \quad (2.6)$$

where,  $dz_B$  is an increment/decrement in  $z_B$  and  $df_k$ ,  $k = 0, 1, 2, \dots, N1$  are increments/decrements in  $f_k$  respectively. Similar linear equations are constructed for all observations,  $(X_j, Z_j)$ ,  $j = 1, 2, \dots, N_{obs}$  on the profile and  $(N1 + 2)$  normal equations are framed and solved by minimizing the misfit given in equation (2.5) using the ridge regression algorithm (Marquardt, 1970). The relevant system of normal equations is expressed in a matrix form as (Chakravarthi et al., 2015a)

$$(A + \delta I)T = S, \quad (2.7)$$

where,  $A$  is  $n \times n$  matrix whose elements  $A_{mj'}$  are given by

$$A_{mj'} = \sum_j^{N_{obs}} \sum_{m=1}^{N1+2} \frac{\partial g_{mod}(X_j, Z_j)}{\partial a_{j'}} \frac{\partial g_{mod}(X_j, Z_j)}{\partial a_m}, j' = 1, 2, \dots, (N1 + 2) \quad (2.8)$$

$$B = \sum_{j=1}^{N_{obs}} Err(X_j, Z_j) \frac{\partial g_{mod}(X_j, Z_j)}{\partial a_{j'}}, j' = 1, 2, \dots, (N1 + 2). \quad (2.9)$$

$$T = da_m, m = 1, 2, \dots, N1 + 2 \quad (2.10)$$

Further,  $a_1 = z_B$ ,  $a_m = f_{m-2}$  for  $m = 2$  to  $N1 + 2$ . Also,  $\delta$  is a damping factor and  $I$  is a diagonal matrix containing the diagonal elements of the matrix,  $A$ . The algorithm computes the partial derivatives required in equation (2.8) and (2.9) by a numerical method (Chakravarthi et al., 2001; Chakravarthi et al., 2015a).

The inversion algorithm initially assigns a value of 0.5 to the damping factor,  $\delta$ , and solves equation (2.7) for the increments/decrements,  $da_m$ ,  $m = 1$  to  $N1 + 2$ . These values are added to/subtracted from the existing parameters,  $a_m$ , to obtain the improved parameters,  $a'_m$ , for  $m = 1, 2, \dots, N1 + 2$ . If the resulting misfit,  $J_{mod}$ , (equation 2.5) obtained with the improved parameters is less than its previous value,  $J$ , then the algorithm assigns  $J_{mod}$  to  $J$  and  $a'_m$  to  $a_m$  and the present value of the damping factor,  $\delta$ , is further decreased by a factor of 1/2. If  $J_{mod}$  is greater than  $J$  at any stage during the process of inversion then the current value of  $\delta$  is doubled and equation (2.7) is

again solved for the parameters,  $\alpha$ . These values are added to/subtracted from the existing parameters,  $\alpha$ , till  $\alpha$  attains a value less than or equal to  $\alpha_{min}$ . The algorithm repeats the process till the specified number of iterations is completed or the misfit (equation 2.5) becomes less than the predefined allowable error or the damping factor,  $\lambda$ , assumes an unusually large value (Chakravarthi, 2003; Chakravarthi et al., 2015a). The parameters  $\alpha$ ,  $\beta$ , and  $\gamma$  remain stable during the process of inversion.

## 2.5 Description of the software – GRIN2DFL

A GUI based software, GRIN2DFL, coded in JAVA is developed based on the algorithm described in section 2.4 to analyze the gravity anomalies produced by 2D listric fault sources using EDCM (Annexure 2-A). The software is built on the Model-View-Controller (MVC) architecture according to the structural relationship shown in Figure 2.2.

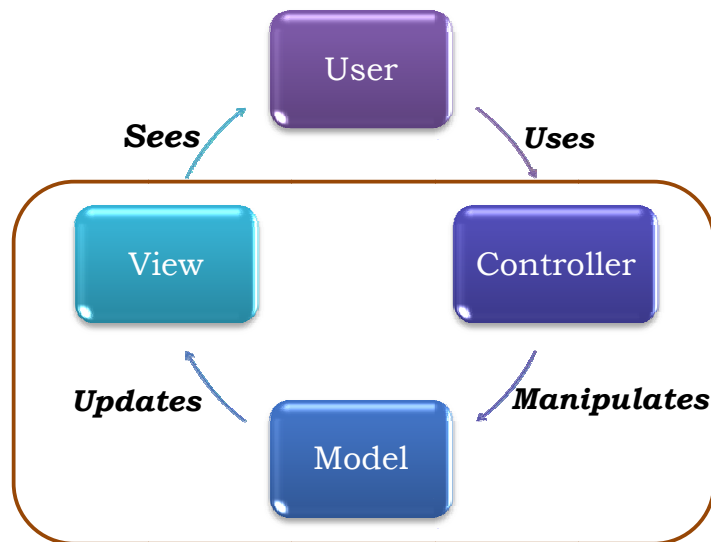


Figure 2.2 Structural relationships between Model, View and Controller

The module 'Model' (Figure 2.2) performs the task of finding the approximate location of the fault plane, computes the gravity anomalies of the structure and performs the business logic of the inversion algorithm. The 'View' module reads the input parameters as specified by the user and shows the interpreted results as output subsequent to inversion. The role of 'Controller' is to pass on the required actions to the view and model modules.

Upon invoking the batch file of the software, the view module appears on the monitor as shown in Figure 2.3.

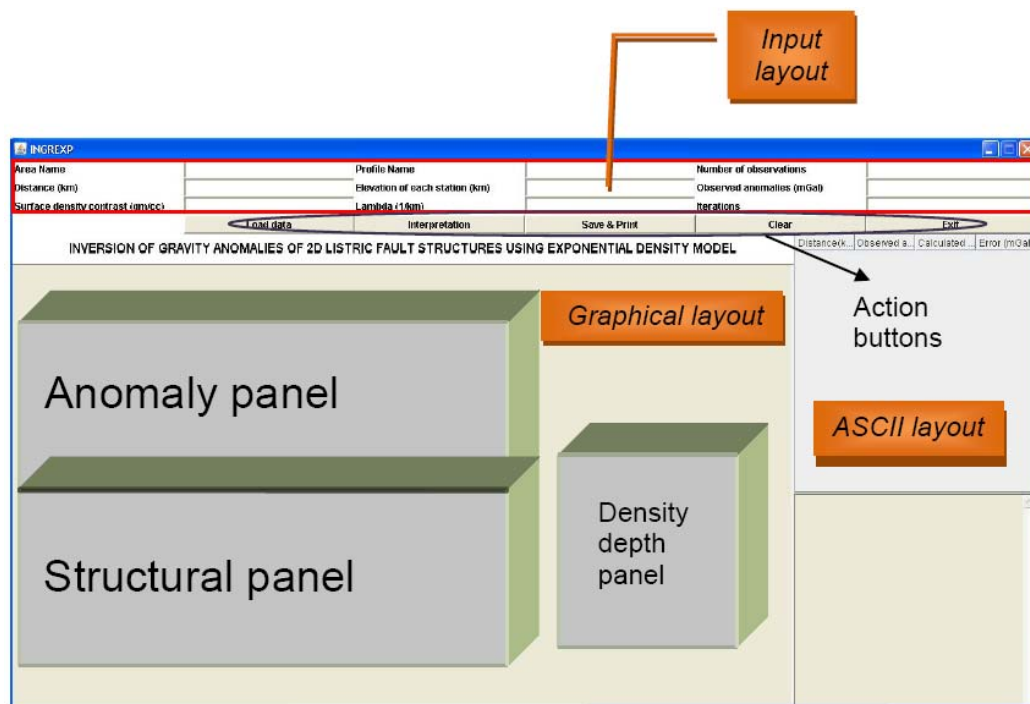


Figure 2.3 View module of INGREXP

The view module is structured into three layouts, namely i) input, ii) graphical, and iii) ASCII as shown in Figure 2.3. The input layout consists of nine fields and five action buttons. The graphical layout, which serves as an interface between the user and the software, is further divided into the anomaly

panel on top, structure panel in bottom, misfit and density-depth panels in the right. The ASCII layout displays the interpreted results in ASCII form. The user enters the information pertaining to the area name, profile name, number of observations, distance to each observation (km), station elevation (km), observed gravity anomalies (mGal), constants of EDCM namely  $\Delta\rho_0$  (gm/cm<sup>3</sup>) and  $\lambda$  (km<sup>-1</sup>), and number of iterations to be performed in respective fields in the input layout (Figure 2.3) and opts for the analysis by using the action button - Interpretation. Alternatively, the user can enter and save the input data in a Microsoft Office Excel sheet and invokes the same to the software by means of 'Load data' action button (Figure 2.3). To avail such an option of inputting the data in an Excel sheet and to load it to the software, the user needs to download the executable Jar File (<http://www.java2s.com/Code/Jar/j/Downloadjxl26jar.htm-jxl-2.6.jar>) in the root directory followed by setting up a class path in the batch file. 'Save and Print' action button enables the user to save the interpreted results and allows for printing.

The advantage and key features of the software are i) it is fully automatic, ii) it works on any operating system (platform independent) with at least jdk 1.6 version installed, and iii) it facilitates the user to visualize the improvements in the modeled space and corresponding model gravity anomalies, changes in the misfit, and variation of density contrast with depth in animated forms during the process of inversion.

## 2.6 Applications

Reliability and applicability of the inversion technique and the software are demonstrated with two examples, one synthetic and a real. The estimated parameters subsequent to inversion are compared with the assumed (actual) parameters in case of synthetic example and with available information in case of the field example. In both cases, the observer is on top of the topography at  $Z_j = 0$ .

### 2.6.1 Synthetic example-Inversion of noisy gravity anomalies

Noisy gravity anomalies produced by a synthetic model of a listric fault source (Figure 2.4b) at 21 equispaced observations in the interval,  $X_j \in [20 \text{ km}, 40 \text{ km}]$ , is shown in Figure 2.4a. In this case the fault ramp is exposed to the surface at 30<sup>th</sup> km on the profile (Figure 2.4b). The parameters assumed to generate the gravity anomalies of the source are:  $z_B = 2.0 \text{ km}$ ,  $\Delta\rho_0 = -0.35 \text{ gm/cm}^3$ ,  $\lambda = 0.4 \text{ km}^{-1}$ . The prescribed density contrast variation with depth is shown in the inset of Figure 2.4b. Further, a 7<sup>th</sup> degree polynomial with a set of 8 arbitrarily chosen coefficients (Table 2.1) simulates the geometry of the fault plane with depth (Figure 2.4b). In this case, pseudorandom noise present in the anomaly is Gaussian with zero mean and a standard deviation of 0.27 mGal. Treating the noisy data (shown as a solid line Figure 2.4a) as the observed anomalies, the inversion was performed by the present technique to recover the fault structure.



## 2D Inversion-Synthetic example

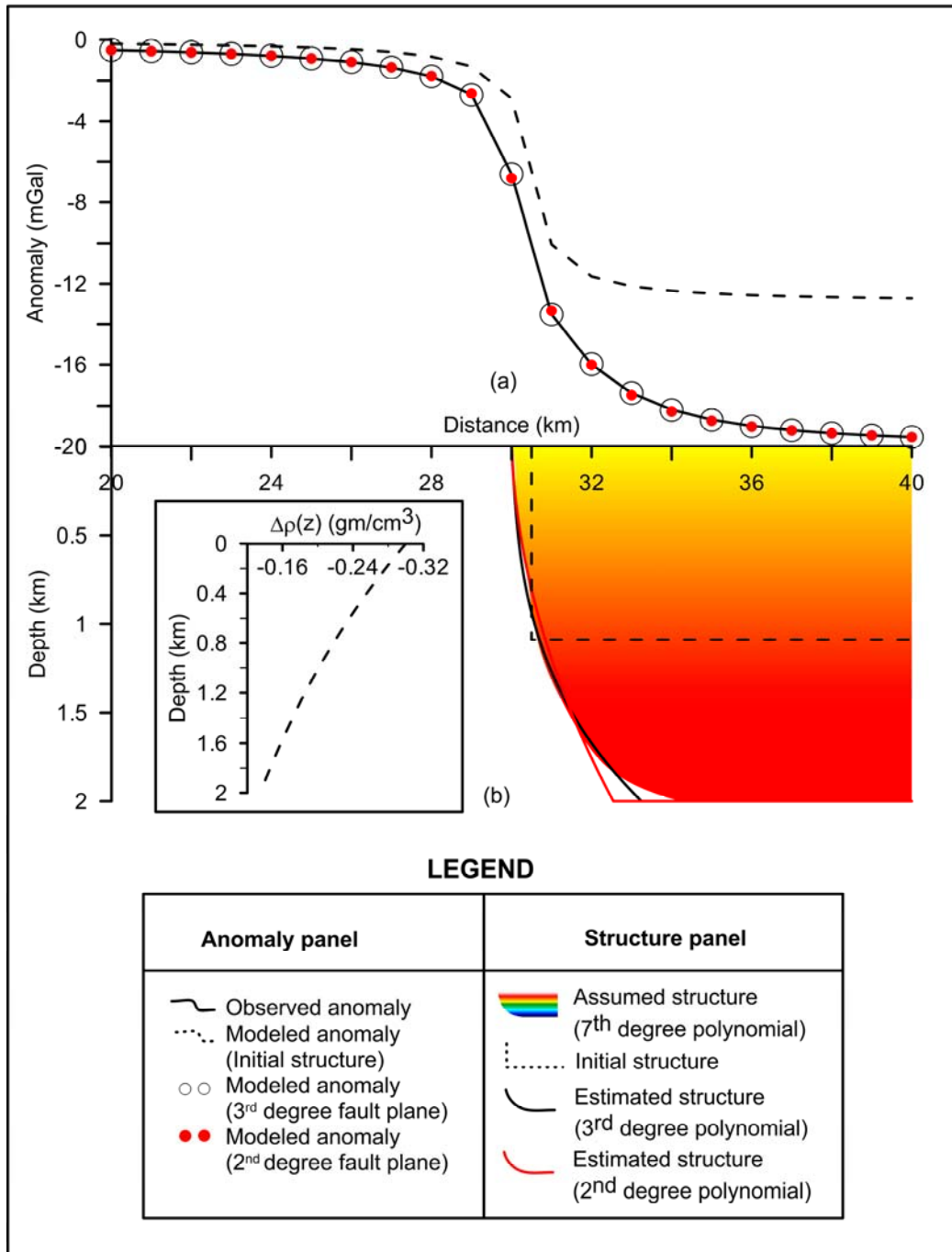


Figure 2.4 (a) Observed and modeled gravity anomalies, (b) assumed, initial and modeled structures. Prescribed Exponential Density Contrast Model (EDCM) is shown in the inset of (b). Colour gradation from yellow to red within the hanging wall represents increase in density

Having consider the fact that the degree of polynomial to be chosen to describe the fault plane geometry is always prone to be uncertain in the absence of additional source of information; two polynomials namely a 2<sup>nd</sup> degree and a 3<sup>rd</sup> degree are considered in the inversion to recover the fault plane geometry (it is to be realized that a 7<sup>th</sup> degree polynomial is used to describe the fault plane while generating the noisy anomalies). For such an approximation, the algorithm had computed the initial thickness of the structure as 1.09 km and the initial parameter value of the coefficient,  $f_0$ , of the polynomial,  $\zeta(z)$ , as 30.5 in each case (Figure 2.4 b).

*Table 2.1*  
*Assumed and estimated coefficients of polynomials, synthetic example.*

<b>Coefficient</b>	<b>Assumed (7<sup>th</sup> degree polynomial)</b>	<b>Estimated (3<sup>rd</sup> degree polynomial)</b>	<b>Estimated (2<sup>nd</sup> degree polynomial)</b>
$f_0$	29.994	30.0024	29.9999
$f_1$	1.963	0.2592	0.1842
$f_2$	-13.978	-0.0518	0.5440
$f_3$	45.868	0.3666	
$f_4$	-70.937		
$f_5$	56.408		
$f_6$	-22.189		
$f_7$	3.432		

For a 2<sup>nd</sup> degree polynomial approximation of the fault plane, the algorithm had performed 16 iterations and for a 3<sup>rd</sup> degree 41 iterations respectively. Further, in case of 2<sup>nd</sup> degree polynomial the damping factor,  $\delta$ , after the 16<sup>th</sup> iteration has attained a larger value than its preceding one thereby the algorithm got terminated. On the other hand, for a 3<sup>rd</sup> degree polynomial approximation, the misfit fell below a predefined allowable error of 0.003 at

the end of the 41<sup>st</sup> iteration. By and large, in either case the nature of fit between the observed and modeled gravity anomalies is satisfactory (Figure 2.4a). The error between observed and modeled gravity anomalies at the end of respective concluding iterations in both cases are shown in Figure 2.5a. The parameter values of the estimated coefficients of the two polynomials at the end of respective concluding iterations were given in Table 2.1. Figure 2.4b shows the geometries of fault structures recovered in either case after respective inversions. Error analysis of misfit, coefficients of selected polynomials and the thickness of the fault morphology versus the iteration number are shown in Figure 2.5b.

The predicted depth to the floor of the structure (decollement horizon) from present inversion in either case was 1.99 km, which agrees well with the assumed depth of 2 km (Figure 2.4b). A negligible error of 0.5% between the assumed and predicted depths is acceptable in both cases because the anomalies used in inversion are noisy.

Based on the analysis, it is concluded that the estimated thickness of a listric fault morphology is independent on the choice of the degree of polynomial used in the inversion to describe the fault plane geometry. However, the use of a lower order polynomial in the inversion would result in the underestimation of the amount of extension across the fault ramp (Figure 2.4b).

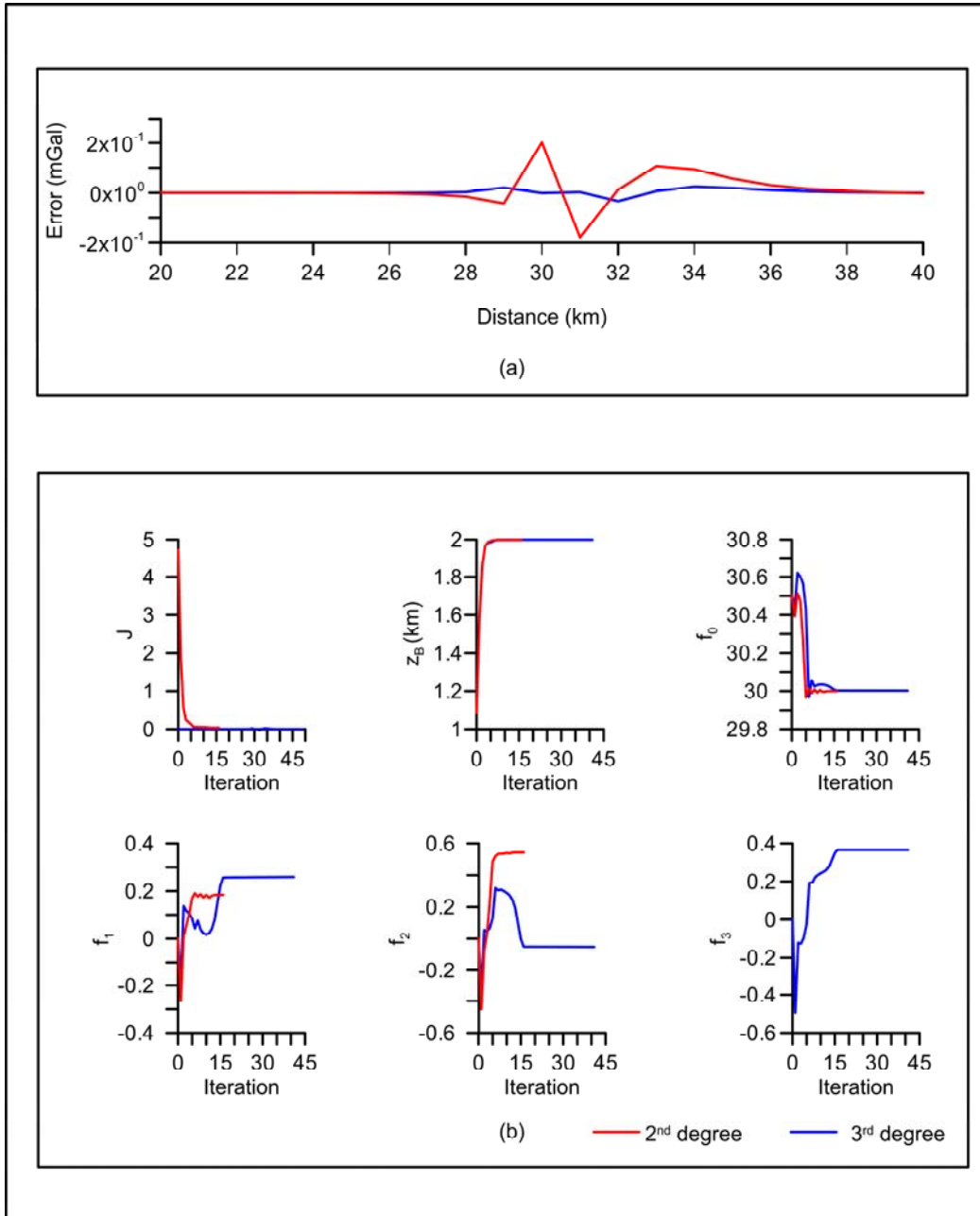


Figure 2.5 (a) Error analysis of the gravity anomaly, (b) variation of misfit and various shape parameters with iteration number

### 2.6.2 Field example

Narmada-Son-Damodar (NSD), Pranhita-Godavari (PG) and Mahanadi (M) river valleys are the main repositories in the peninsular India, where Gondwana succession has been preserved. Based on the geology, structure, and nature of lithic fill, Murthy and Parthasarathy (1988) have distinguished the Pranhita-Godavari Valley into four sub-basins namely; Godavari, Kothagudem, Chintalpudi, and Krishna-Godavari respectively. Out of the four, the Godavari sub-basin covers an area of about 12,350 sq km over a strike length of approximately 200 km (Figure 2.6). The sub-basin exhibits a half-graben structure defined by the Ahiri-Cherla master fault on the northeastern side (Qureshy et al., 1968; Ramakrishna and Chayanulu, 1988).

Mishra et al. (1987) had discussed in detail the gravity survey of the Pranhita-Godavari Valley including the distribution of gravity observations, application of various corrections to the measured data and the accuracy of the Bouguer gravity anomalies. The steep gradient observed on the Bouguer anomaly on the eastern side of the basin throughout its strike is associated with the Ahri-Cherla master fault (Figure 2.6). Chakravarthi and Sundararajan (2004) have analyzed the gravity anomalies of this master fault along a selected profile treating the fault plane of the structure as a planar surface. In the present case we analyze the gravity anomalies of the master fault along a profile, AA', (Location of the profile is shown in Figure 2.6) presuming the fault plane of the structure as non-planar. The smoothed residual gravity anomaly across the fault structure along the profile, AA', obtained after separating a regional trend is

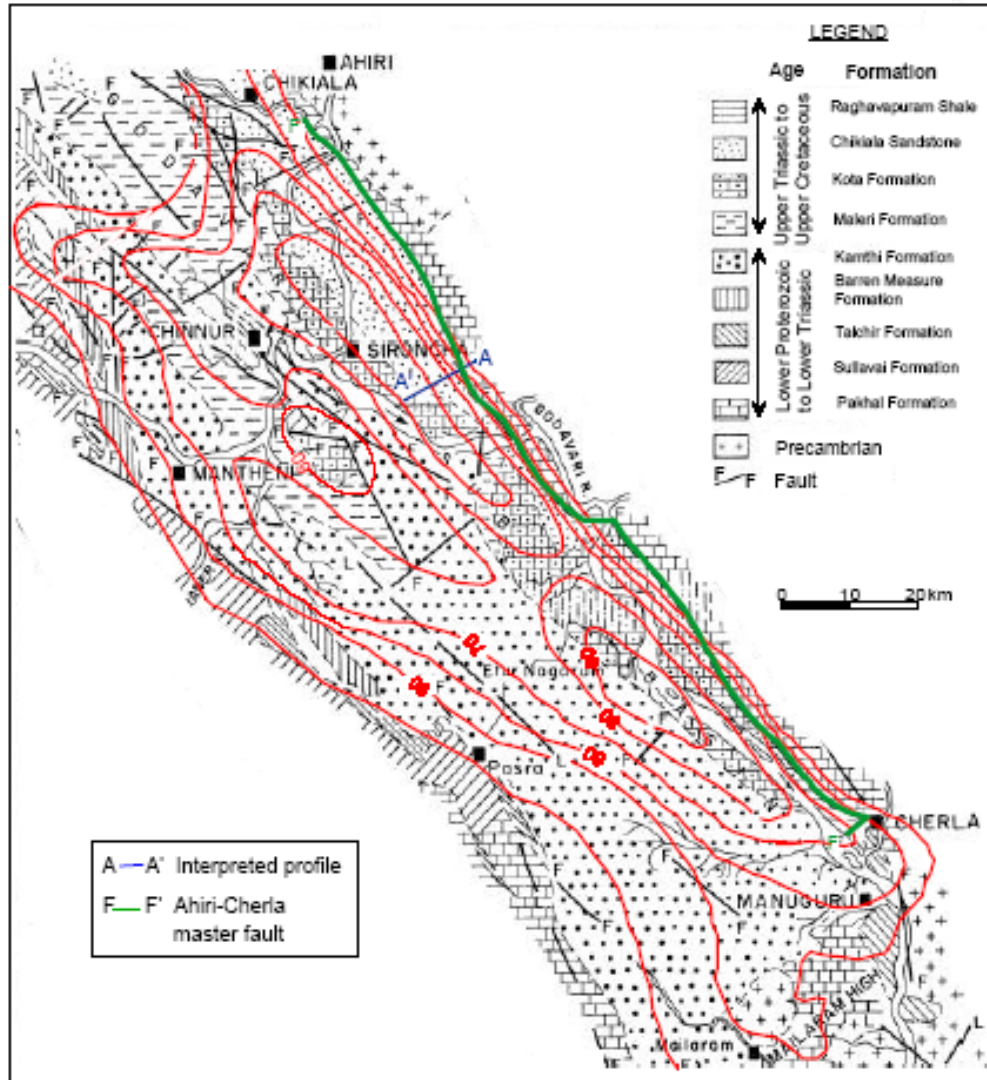


Figure 2.6 Geologic and Bouguer gravity anomaly map of the Godavari sub-basin, India (after Chakravarthi and Sundararajan, 2004). Note that anomaly values have to be read with a negative sign

shown as a solid line in black in Figure 2.7a. The exponential density contrast model (EDCM) obtained by fitting equation (1.14) to the derived parabolic density model of the basin (Chakravarthi and Sundararajan, 2004) was used to analyze the anomalies shown in Figure 2.7a. The fitted EDCM (inset of Figure 2.7b) is given by (Chakravarthi et al., 2015a)

$$\Delta\rho(z) = -0.4554e^{-0.3929Z}. \quad (2.11)$$

As in the case of synthetic example, the inversion was performed on the observed anomalies of the sub-basin considering both 2<sup>nd</sup> and 3<sup>rd</sup> degree polynomial approximations for the fault plane. The initial parameter value of  $z_B$  in each case is 1.44 km. The dashed line shown in Figure 2.7b represents the starting model of fault geometry based on the initial parameters estimated by the algorithm for  $f_0$  (Table 2.2) and  $z_B$ .

*Table 2.2*  
*Estimated coefficients of polynomials, Ahiri-Cherla master fault*  
*Godavari sub-basin, India.*

<b>Coefficient</b>	<b>Estimated (3<sup>rd</sup> degree polynomial)</b>	<b>Estimated (2<sup>nd</sup> degree polynomial)</b>
$f_0$	4.9698	4.9693
$f_1$	0.2226	0.2259
$f_2$	0.0872	0.0842
$f_3$	-0.0006	

The algorithm took 21 iterations to achieve proper convergence in case of a 2<sup>nd</sup> degree polynomial approximation for the fault plane and 47 iterations in case of 3<sup>rd</sup> degree approximation respectively. The modeled gravity anomaly

## 2D Inversion-Field example

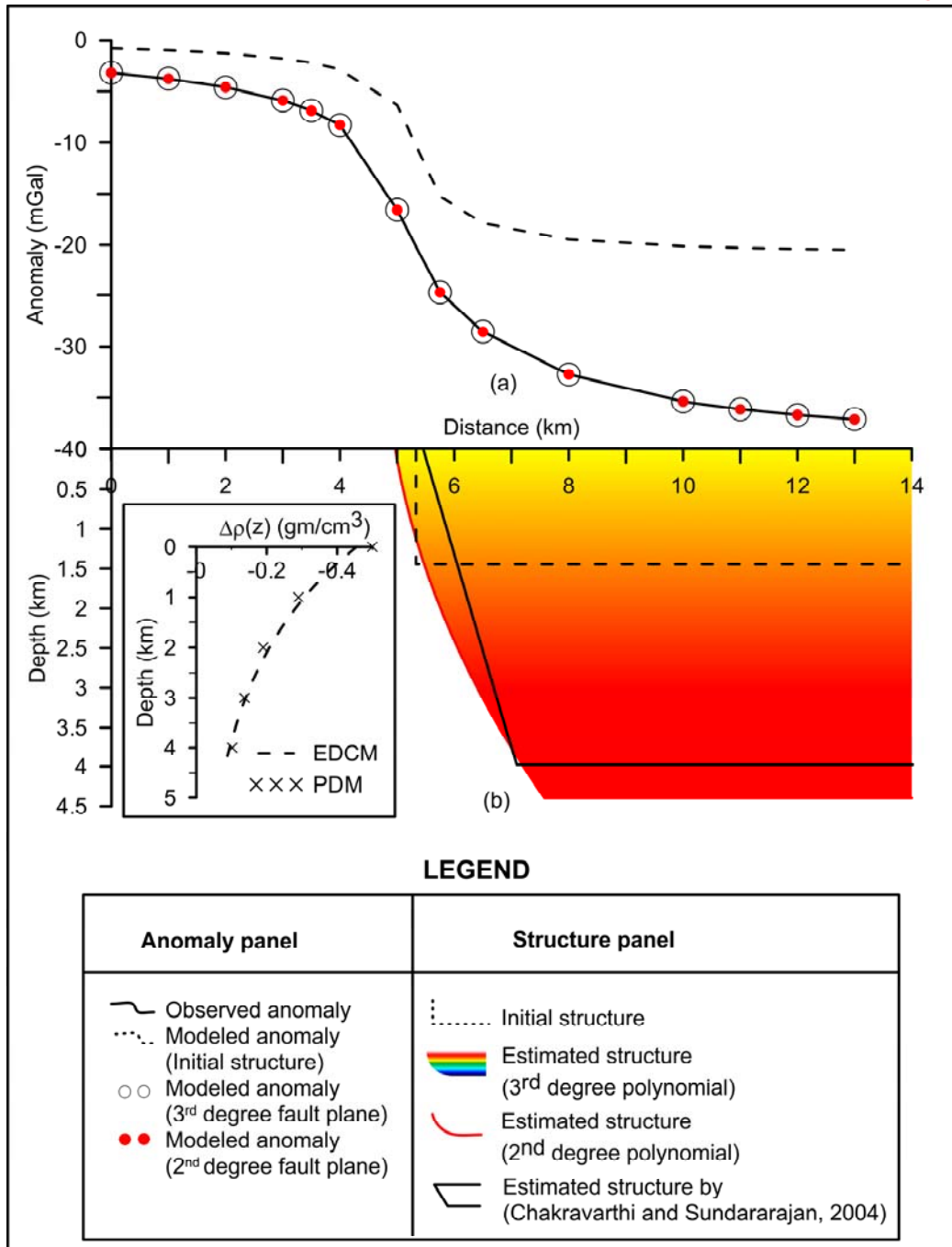


Figure 2.7 (a) Observed, initial and modeled gravity anomalies, (b) initial and estimated structures, Ahiri-Cherla master fault, Godavari sub-basin, India. Simulated exponential density contrast model is shown in the inset of (b)



subsequent to respective concluding iterations closely mimic the observed one (Figure 2.7a). The magnitude of residuals between the observed and modeled anomalies in either case is below  $\pm 0.095$  mGals (Figure 2.8a). The estimated parameter values of the coefficients for 2<sup>nd</sup> and 3<sup>rd</sup> degree polynomials after respective inversions are given in Table 2.2.

The estimated depth to the bottom of the fault structure in either case is 4.39 km, which again confirms the fact that the choice of the degree of polynomial in the inversion to describe the fault plane geometry does not appreciably affect the interpreted depth. Furthermore, the inferred structure of the Ahiri-Cherla fault morphology based on a 3<sup>rd</sup> degree polynomial approximation of the fault plane almost coincides the structure estimated using a 2<sup>nd</sup> degree polynomial (Figure 2.7b). The changes in misfit and other estimated parameters against the iteration are shown in Figure 2.8b in each case. The fact that the inferred thickness of the fault morphology from present inversion (4.39 km) closely matches with an estimated fault throw of 4.5 km by Murthy and Parthasarathy (1988) and about 4 km by Chakravarthi and Sundararajan (2004) demonstrates the practical applicability of the technique.

The interpreted geometry of the fault structure along the same profile with a planar fault surface approximation (Chakravarthi and Sundararajan, 2004) is also shown in Figure 2.7b (solid line) for comparison. It is noticed from Figure 2.7b that the interpreted dips of the fault plane from both the present method and the one reported by Chakravarthi and Sundararajan (2004) are correlated well with each other up to a depth of 1.5 km, beyond which the

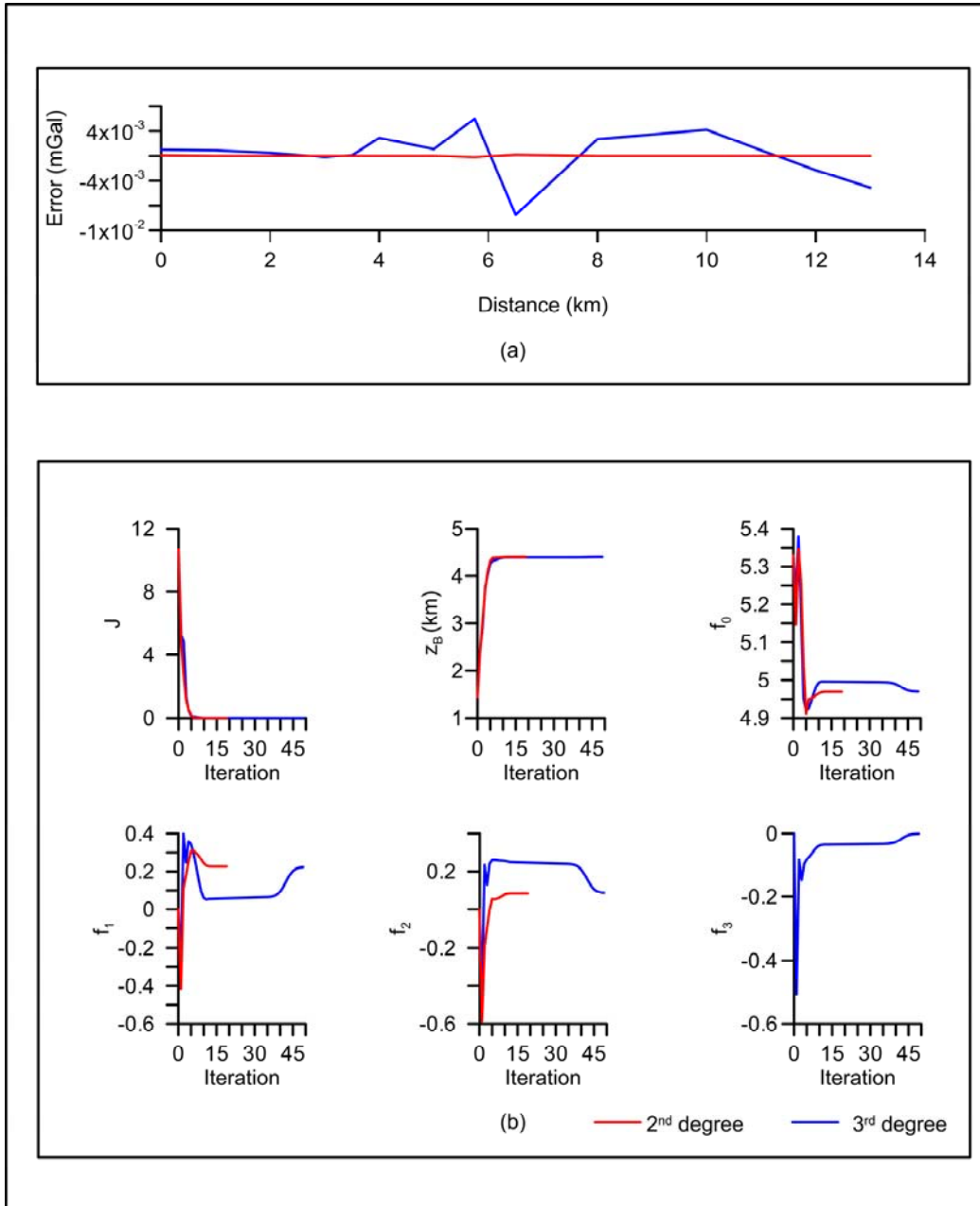


Figure 2.8 (a) Error analysis of the gravity anomaly, (b) variation of misfit and various shape parameters with iteration number, Ahiri-Cherla master fault, Godavari sub-basin, India.

estimated fault plane from the present method portrays progressively decreasing dips.

## **2.7 Results and discussion**

Although listric fault morphologies associated with thick sedimentary sequence of rocks warrants the use of exponential density contrast model in the analysis of gravity anomalies to obtain more reliable results, not many methods have been reported because of the fact that no closed form gravity expressions could be derivable in the space domain to realize forward modeling. A new method combining both analytical and numerical approaches has been developed in the space domain to realize forward gravity modeling of 2D listric fault morphologies using an exponential density contrast model. Prescribed polynomial functions of arbitrary but specific degree are used to describe the non-planar fault ramps. An automatic inversion technique coupled with relevant GUI based software is developed to analyze the gravity anomalies for the unknown parameters of fault morphology. The application of the technique is exemplified with both synthetic and real field gravity anomalies.

In case of a synthetic fault model, a 7<sup>th</sup> degree polynomial is used to describe the fault ramp to compute the gravity response. Pseudorandom noise with zero mean and standard deviation of 0.27 mGal is added to the residual gravity anomaly produced by the structure before being subjected for inversion by the proposed technique. Also, as uncertainty prevails on the selection of the degree of polynomial to be chosen to simulate the fault plane in the inversion

(in the absence of additional sub-surface information), a 2<sup>nd</sup> and a 3<sup>rd</sup> polynomials are assumed, independently, to study their affect on the interpretation, if any. It was found from the analysis that the estimated depth of the structure remains more or less the same irrespective of the degree of polynomial used in the inversion (whether a 2<sup>nd</sup> degree or a 3<sup>rd</sup> degree) to describe the fault ramp. However, it is shown that the choice of lower order polynomials in the inversion would lead to underestimate the amount of extension across the fault ramp.

The analysis of the observed gravity anomalies across the Ahri-Cherla master fault (boundary fault) of the Godavari sub-basin in India using the derived exponential density contrast model has yielded an interpretation that is consistent with the available/reported information.

In short, the successful application of the proposed technique on both synthetic and real world gravity anomalies testifies its applicability.

### Interactive gravity modeling of 2.5D strike listric fault sources with arbitrarily varying density-depth relationship\*

---

#### 3.1 General:

Modeling of concealed density interfaces from observed gravity anomalies is one of the classic geologic applications of the gravity method. Chakraborty et al. (2003), Chakravarthi (2011b) have shown that the marginal/boundary faults associated with some of the sedimentary basins, such as the pull-apart basins, are not only listric in nature but also possess finite strike lengths (see for e.g., Hutar, Auranga, Karanpura, Bokaro, Jharia, and Ranigunj basins of Damodar valley in India). Peirce and Lipkov (1988) have also showed that the faults on the southwestern margin of the Rukwa Rift have finite strike lengths. Therefore, approximating such listric fault morphologies to appropriate geophysical models having finite strike lengths is justified to analyze the gravity anomalies. However, 3D models are more expensive in terms of data requirement and computational time than 2D; therefore, the use

---

\* *Published in Journal of the Geological Society of India (2014, 83, 577-585), Springer.*

of 2.5 dimensionality (2.5D) in the interpretation of gravity anomalies is more appropriate.

### **3.2 Status of existing interpretational techniques**

Rao and Prakash (1990) have considered 2.5 dimensionality for the fault structures and developed an interpretation algorithm to analyze the gravity anomalies arise from such models using a quadratic density-depth relationship. This technique, being efficient on residual gravity anomalies, may yield unreliable interpretations in the presence of regional gravity background. In addition, this density function would pose problems in automatic modeling of gravity anomalies as elucidated by Chakravarthi and Sundararajan (2006). On the other hand, Chakravarthi (2008) had used a parabolic density model to devise an algorithm to estimate simultaneously the parameters of 2.5D strike fault structures and regional gravity background from a set of observed Bouguer gravity anomalies. Nevertheless, the practical applicability of the enlisted methods is limited because these techniques presume planar surfaces for fault planes, which in reality may not be so as large dip-slip faults are usually curved in cross-section (Peirce and Lipkov, 1988; Brun and Choukroune, 1983; Chakravarthi, 2011b; Chakravarthi et al., 2014).

Martín-Atienza and García-Abdeslem (1999) have also developed a technique based a quadratic density model to compute the gravity anomalies of geologic sources bounded either laterally or vertically by continuous functions. Even though, this method can be used to simulate the geometries of listric fault

sources to compute the gravity anomalies, again it is efficient only for 2D sources. In recent past, Chakravarthi (2010a) devised a method to compute the gravity responses of 2.5D listric fault sources; where the fault planes are described by non-planar surfaces, detached hanging wall system composed of thick-sectioned sediments and the density contrast of which varies continuously with depth following a parabolic equation. Later, Chakravarthi (2011b) developed an automatic inversion to estimate the geometries of 2.5D strike listric fault sources and regional gravity background from the observed Bouguer gravity anomalies.

The existing density functions including those enlisted above will fail to describe the sub-surface density distribution in case the detached hanging wall consists in both high and low-density formations. Chakravarthi (2010b) devised a strategy coupled with a code to realize forward gravity modeling of such listric fault sources. In order to calculate the gravity anomalies using this method, the coefficients of the polynomial to describe a fault plane need to be estimated separately and supplied to the code as a part of input along with the depths and densities of various sub-surface formations. The code then generates the output in ASCII format. In reality, one needs to improve the model parameters several times until the modeled gravity anomalies mimic the observed ones. It becomes a prodigious task to input several such combinations of model parameters to realize gravity modeling.

Therefore, a need exists to develop a new interpretation strategy with relevant software that could overcome the drawbacks associated with the

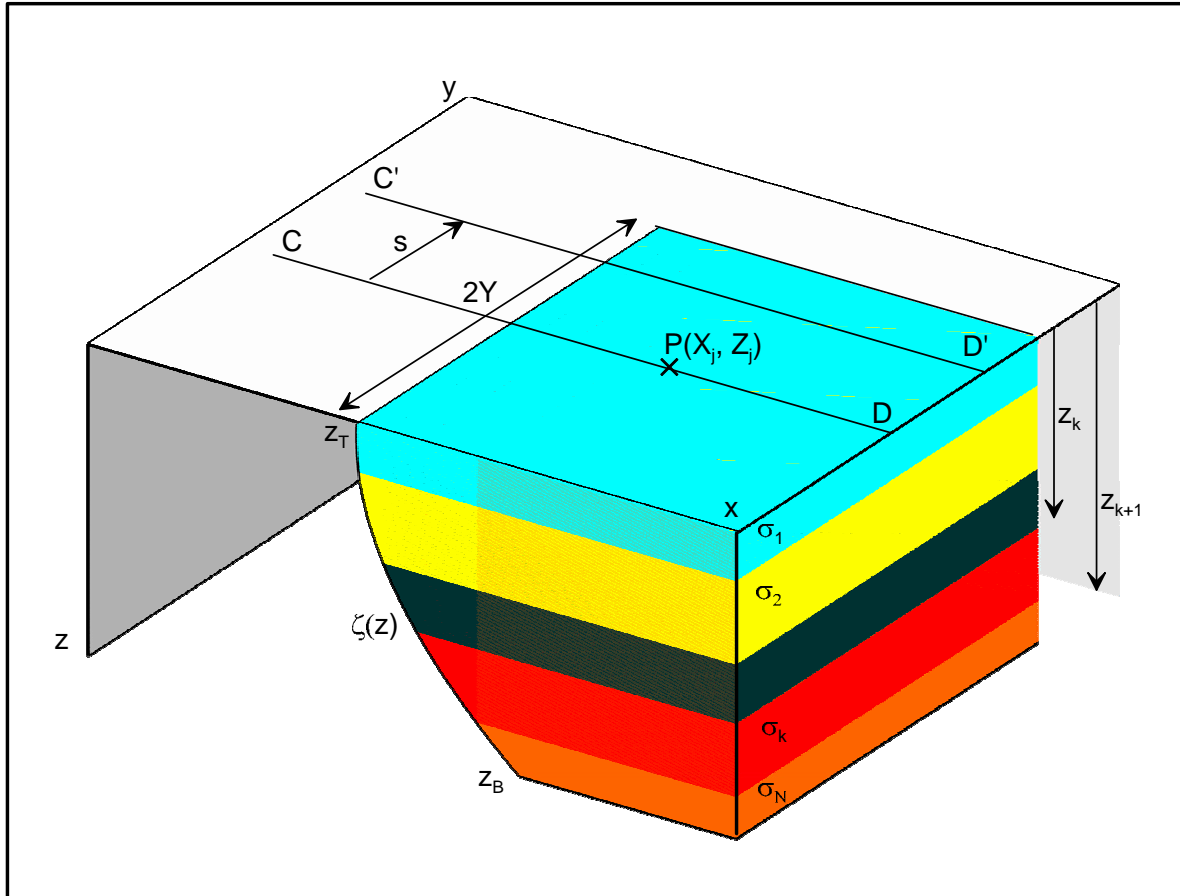


Figure 3.1 Schematic representation of a strike limited listric fault source. The detached downthrown block (hanging wall) is consisting of  $N$  number formations with differing thicknesses and densities. The limited strike length prevents the structure to represent as a 2D source.



existing algorithms to analyze the gravity anomalies of strike limited listric fault morphologies, where the detached hanging wall contains in both high and low density formations. In addition, the interpretation tool should be simple and user-friendly in terms of interactive model construction and subsequent modification to realize modeling in real time.

### 3.3 Forward gravity modeling – Theoretical considerations

In Cartesian co-ordinate system, let the  $z$ -axis be positive vertically downwards and the  $x$ -axis transverse to the strike of a listric fault source, whose geometry is shown in Figure 3.1. The structure is confined between the limits,  $z_T$  and  $z_B$ , along the  $z$ -axis, and along the  $x$ -axis it is bounded by a function,  $\zeta(z)$ , on the left and towards the right it is extending to infinity. Further, the structure is having a limited strike length of  $2Y$  along the  $y$ -axis transverse to the  $xz$  plane. The detached hanging wall of the structure is presumed to consist in several geologic formations,  $N$  in number; where each one has its own density,  $\sigma_i$ , and thickness,  $z_i$ ,  $i = 1, 2, \dots, N$ . For such a structure, the gravity anomaly,  $g_{mod}(X_j, Z_j)$ , at any point,  $P(X_j, Z_j)$ , on a profile, CD, that runs along the  $x$ -axis and bisects the strike length,  $2Y$ , of the structure outside the source region (Figure 3.1) is given by Chakravarthi et al., (2014) as

$$g_{mod}(X_j, Z_j) = 2G \sum_{k=1}^N \Delta\rho_k \int_{z_k}^{z_{k+1}} \left[ \tan^{-1} \left\langle \frac{Y}{z - Z_j} \right\rangle - \tan^{-1} \left\langle \frac{Y \overline{\zeta(z) - X_j}}{\overline{z - Z_j} \sqrt{\overline{\zeta(z) - X_j}^2 + \overline{z - Z_j}^2 + Y^2}} \right\rangle \right] dz, \quad (3.1)$$

where,  $z_k$  and  $z_{k+1}$  are depths to the top and bottom of the  $k^{\text{th}}$  formation within the hanging wall, and  $\zeta(z)$  is represented by the equation (2.3). It is convenient to solve equation (3.1) numerically rather than analytically because the polynomial,  $\zeta(z)$ , may take any degree (Chakravarthi et al., 2014). In case the profile does not bisect the strike length of the structure but runs at an offset,  $s$ , (such as the profile, C'D' in Figure 3.1) then the anomalous field at any point on the profile can be calculated as the average of equation (3.1) by substituting,  $Y-s$  and  $Y+s$  for  $Y$ . Also, if the profile runs at an angle,  $\alpha$ , with the  $x$ -axis then  $X_j$  in equation (3.1) needs to be replaced by  $X_j \cos \alpha$  (Chakravarthi and Ramamma, 2013b; Chakravarthi et al., 2014).

### **3.3.1 Effects of offset and strike length on the magnitude of anomaly**

Figure 3.2a illustrates the effects of profile offset and strike length of the structure on the magnitude of gravity anomaly produced by a listric fault source, whose geometry is shown in Figure 3.2b. In this case, the detached hanging wall is presumed to consist in two formations namely, 7 km thick-sectioned sediments at the bottom are covered on the top by 3 km thick basalt flow (Figure 3.2b). The assumed densities for the two formations under consideration are  $2.85 \text{ gm/cm}^3$  for basalt and  $2.3 \text{ gm/cm}^3$  for concealed sediments, respectively (Figure 3.2c). Further, each formation within the hanging wall has a strike length of 50 km (Figure 3.2d).

For such a geologic setting, the anomalies are calculated along two selected profiles PP' and QQ' (locations of the profiles are shown in a plan

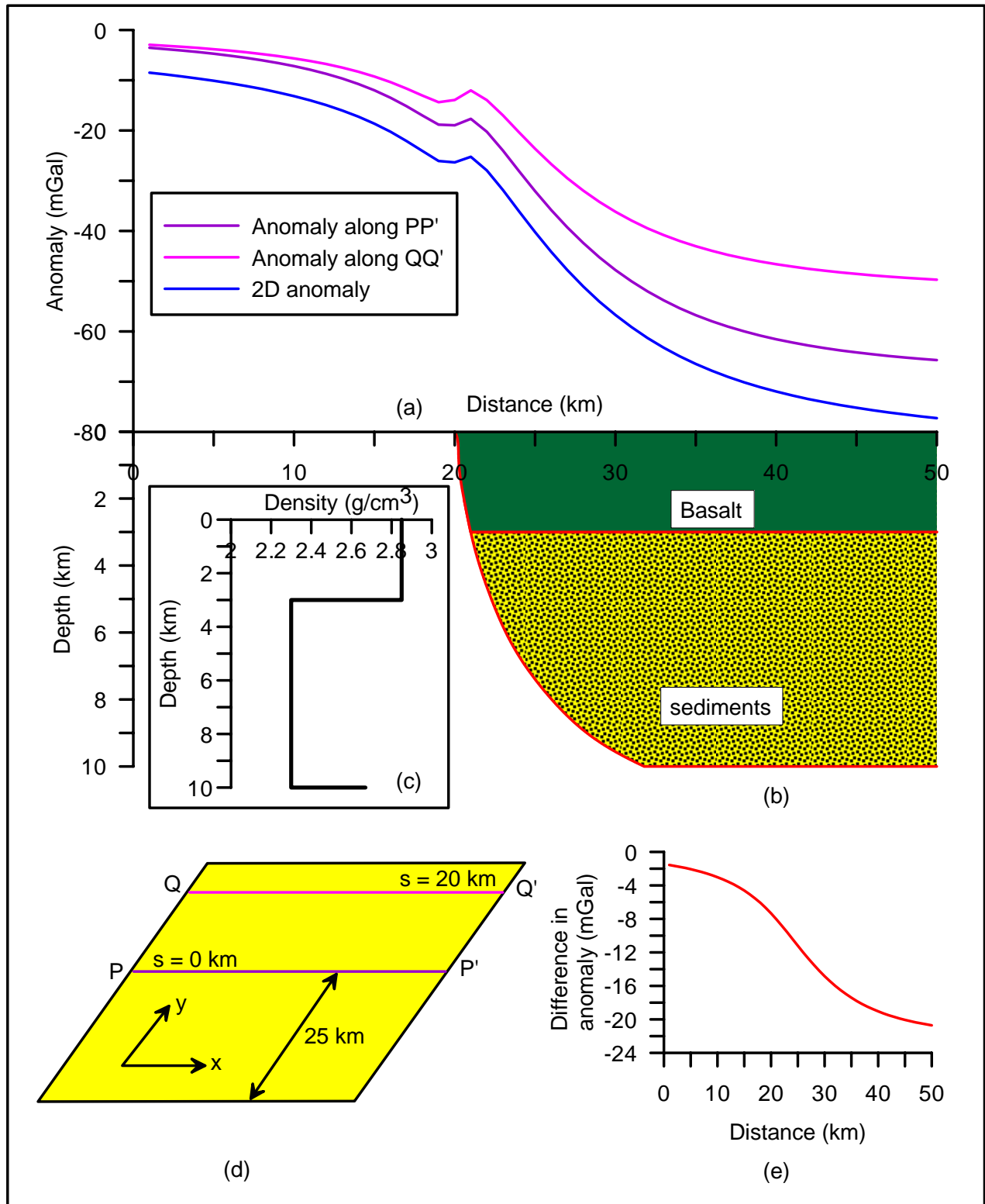


Figure 3.2 (a) Gravity anomalies along two selected profiles, PP' and QQ', across a 2.5D strike-slip fault source, whose geometry is shown in (b), (c) density-depth relationship within the hanging wall, (d) plan view showing the locations of the profiles, (e) differences in magnitudes of gravity anomalies observed along the profiles, PP' and QQ'.

view in Figure 3.2d) at 50 observations in the interval  $x_j \in [0 \text{ km}, 50 \text{ km}]$  and shown in Figure 3.2a. It is to be noted from Figure 3.2d that the two selected profiles (PP', QQ') run transverse to the strike of the structure but the profile, PP', bisects the strike length whereas the other one, QQ', does not (profile, QQ', is located at an offset of 20 km). One can notice from Figure 3.2a that the gravity anomaly along the profile PP' has portrayed larger a magnitude (absolute) than the corresponding one observed along the QQ' profile.

Furthermore, the anomalies along the two selected profiles closely mimic with each other at the observations resting on the footwall of the structure far away from the fault plane, whereas they depict large deviations over the hanging wall (Figure 3.2a) across the fault plane. The magnitude of the difference between the two anomalies varies from about -1.5 mGal over the footwall to more than -20 mGal above the hanging wall (Figure 3.2e). Therefore, the magnitude of gravity anomaly (absolute) of a 2.5D listric fault source decreases with the increase in offset of the profile, albeit the anomalous source remains the same.

To study the effect of strike length on the magnitude of gravity anomaly, a large value of 1500 km is assigned to the strike of the structure (to consider the structure as 2D) before being performed the forward modeling in the interval  $x_j \in [0 \text{ km}, 50 \text{ km}]$ . In this case, the other parameters of the structure remain unchanged. The anomalous field of such a 2D structure (shown in

Figure 3.2a as a solid line in blue) deviates significantly from the anomalies produced by the same structure with limited strike length.

Therefore, it is concluded that the gravity anomalies generated by a listric fault source are dependent on both the strike length of the structure and offset of the profile. Hence, these parameters play pivotal roles in the analysis of gravity anomalies of listric fault morphologies.

This chapter deals with the development of an interpretation strategy and related software, FRGMLSTRK, coded in JAVA to calculate as well as to model the gravity anomalies of 2.5D strike-limited listric fault morphologies in real time using a Graphical User Interface (GUI). The advantage of this software is that, besides being platform-independent, it also allows the user to initialize as well as to modify the structure interactively (using simple mouse click operations) until the modeled anomalies fit the observed ones. The applicability of the interpretation technique is demonstrated on a synthetic and a real world gravity anomaly.

### **3.4 Description of software – FRGMLSTRK**

The architecture of the present code, FRGMLSTRK, (Annexure-3A) is based on the MVC pattern shown in Figure 2.2 (Chakravarthi et al., 2014). The ‘Model’ performs the task of finding the origin of the fault plane from a set of observed gravity anomalies, estimates the coefficients of the prescribed polynomial to describe the fault plane geometry, and computes the modeled gravity anomalies of the structure. The ‘View’ module executes the task of

reading the input data and allows the user to modify the fault plane geometry, depths and densities of several sub-surface formations either independently or in combination and finally displays the results in both ASCII and graphical forms. The ‘Control’ unit executes the task of passing the required actions to the model and view modules. Upon invoking the batch file, the view module appears on the screen as shown in Figure 3.3.

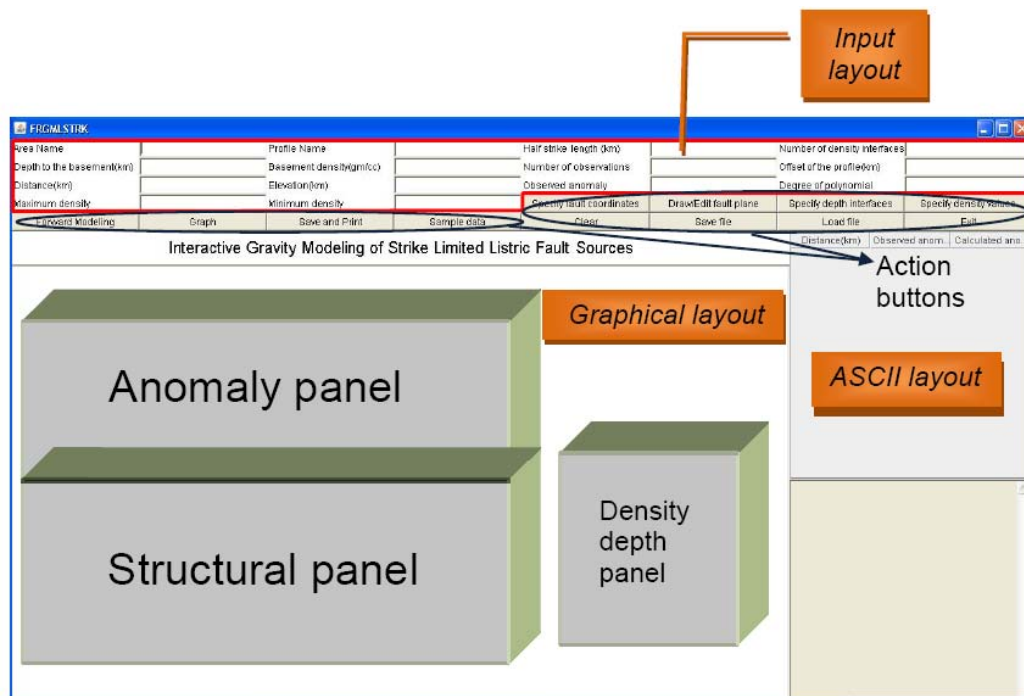


Figure 3.3 View module of FRGMLSTRK

The view module is organized into the input layout, graphical layout and ASCII layout as shown in Figure 3.3. The input layout consists of 14 input fields namely the area name, profile name, half-strike length of the structure in km, number of density interfaces within the hanging wall, depth to basement in km, basement density in  $\text{gm/cm}^3$ , number of observations, offset of the profile in km, distance and elevation of each observation expressed in km, observed

gravity anomaly in mGal, degree of the polynomial to describe the fault plane, minimum and maximum densities of the sub-surface formations in  $\text{gm/cm}^3$ .

The 12 action buttons (Figure 3.3) of the business logic are: Specify fault co-ordinates, Draw/Edit fault plane, Specify depth interfaces, Specify density values, Forward Modeling, Graph, Save and Print, Sample data, Clear, Save file, Load file, and Exit. Upon entering the input fields and invokes the action button ‘Specify fault coordinates’ (Figure 3.4), the code automatically identifies the location of the fault plane (shown as ‘X’ in Figure 3.4) following the methodology described in section 2.4 of Chapter-II. Based on available subsurface information the interpreter selects a few control points in the structural panel (by means of mouse clicks) to construct the geometry of a listric fault plane as shown in Figure 3.4.

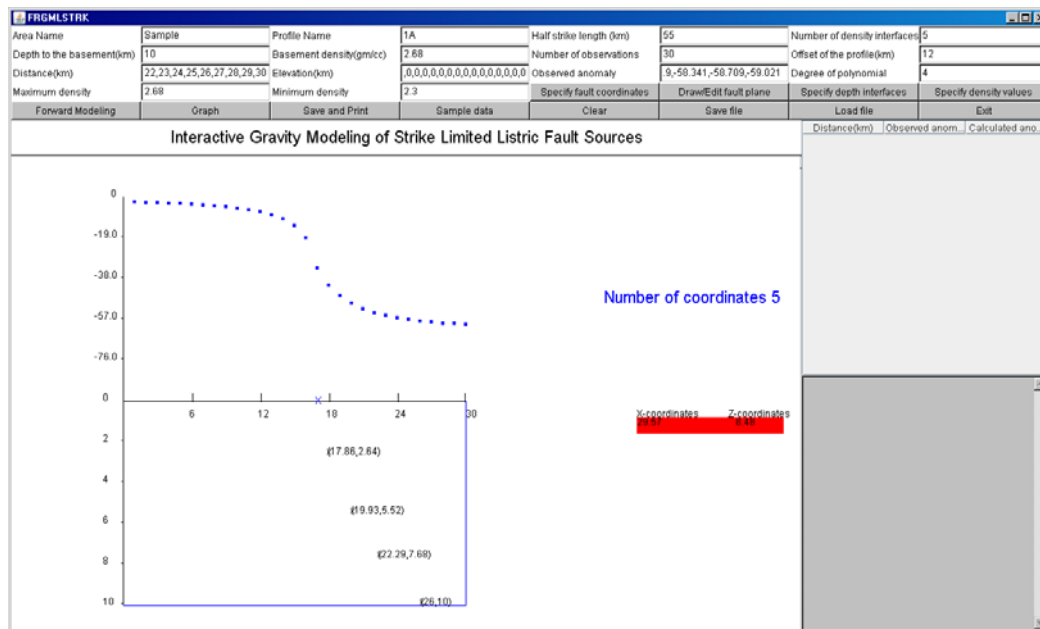


Figure 3.4 Observed gravity anomaly (anomaly panel) and control points to describe fault plane geometry (structural panel).

The number of control points and co-ordinates of each such selected point will be displayed on the right-hand side of the structural panel. If the number of selected control points to describe the fault plane is less than the optimum number (based on the degree of polynomial), then the digit describing the number of points shall be displayed in red. In such a case, the interpreter adds a few more control points in the structural panel till the digit turns to blue (Figure 3.4).

Once the action button ‘Draw/Edit fault plane’ is invoked (Figure 3.5), the code estimates the coefficients,  $f_k$ , by fitting the co-ordinates of the selected control points to a polynomial,  $\zeta(z)$ . The code uses these estimated coefficients to construct an analytically defined fault plane as shown in Figure 3.5 (structural panel). The same action button can also be used to edit/modify the coordinates of the selected control points (if required) by simple drag and

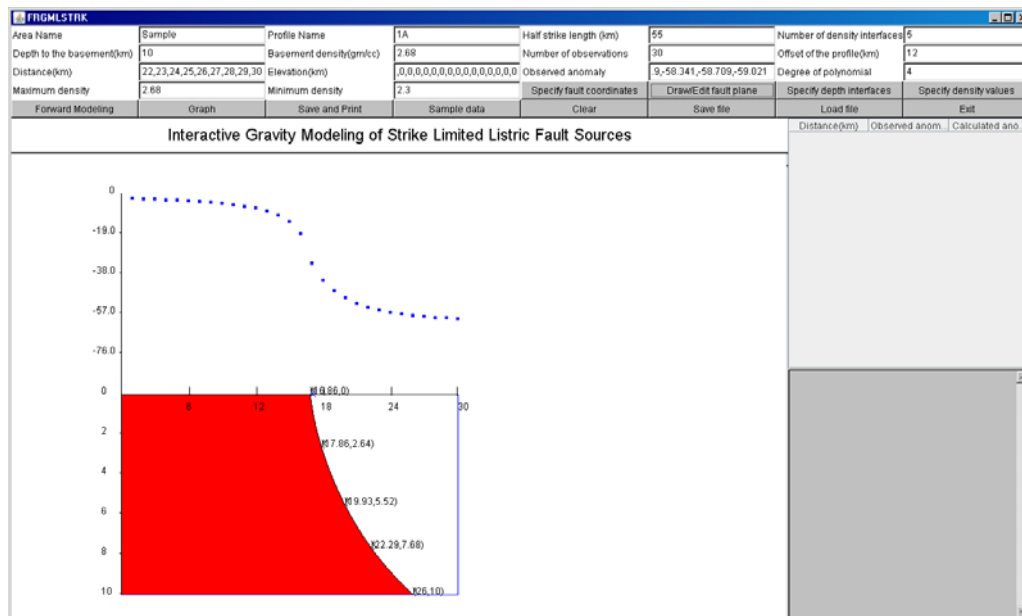


Figure 3.5 Observed gravity anomaly (anomaly panel) and the geometry of an analytically defined fault plane (structural panel)



drop mouse operations. The code then automatically updates the coefficients of the polynomial and reconstructs the modified fault plane.

By invoking the action button ‘Specify depth interfaces’ (Figure 3.6), the user specifies the depths to various density interfaces within the hanging wall of the fault morphology by means of simple mouse clicks in the structural panel (Figure 3.6). The number of density interfaces that are to be specified within the hanging wall shall be displayed on the right-hand side of the anomaly panel. Further, the depth at which a density interface has to be specified can be realized by the mouse operations within the structural panel.

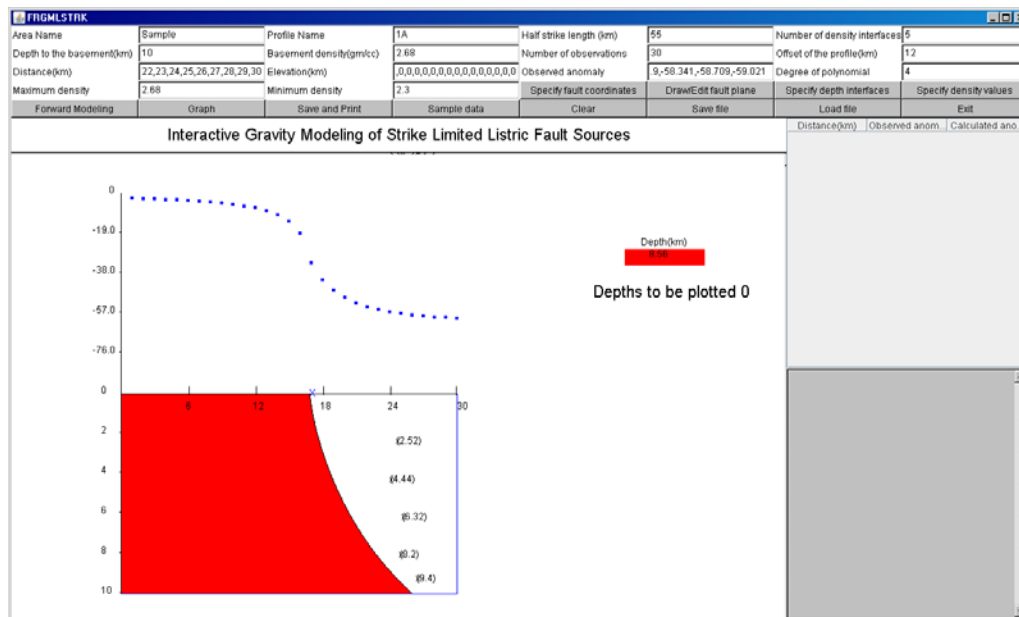


Figure 3.6 Observed gravity anomaly (anomaly panel) and control points selected for specifying the depths to the density interfaces within the structural panel

The interpreter specifies the densities of the sub-surface formations within the hanging wall by invoking the action button ‘Specify density values’ (Figure 3.7). The code then facilitates the user to specify these values by means of

mouse clicks in the density-depth panel (Figure 3.7). The number of density parameters to be specified shall be displayed above the density-depth panel.

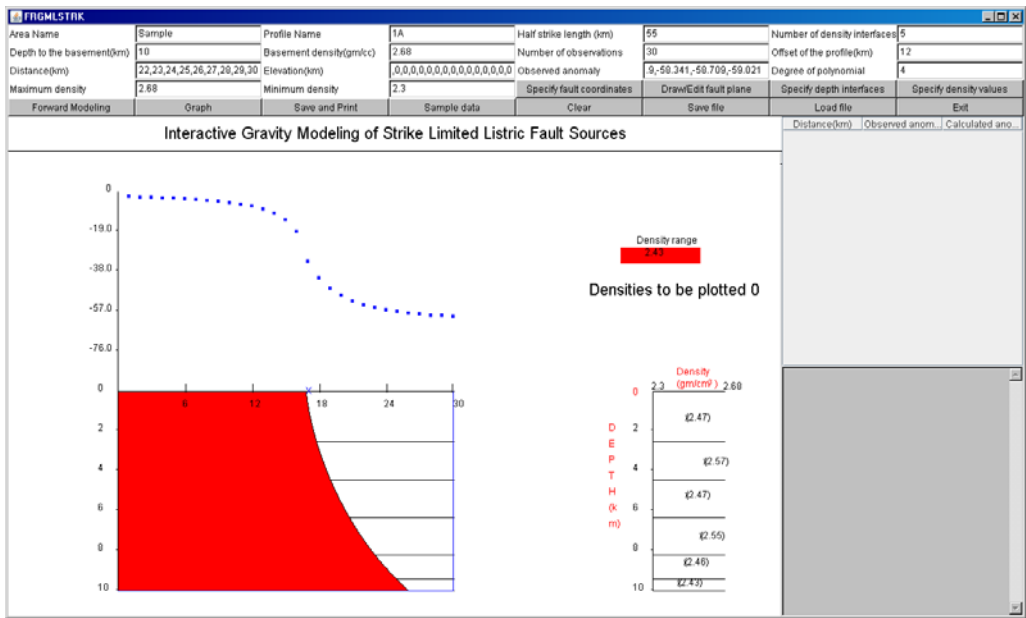


Figure 3.7 Observed gravity anomaly (anomaly panel) and selected control points to specify the densities within the density-depth panel

The code performs the business logic of the algorithm to compute the gravity anomalies of the model space by means of invoking the action button ‘Forward modeling’ (Figure 3.8). The model response shall be displayed in the anomaly panel (solid line in black in Figure 3.8) along with the observed anomalies (solid dots in blue in Figure 3.8). The estimated coefficients of the polynomial function, density-depth distribution of the sub-surface formations, observed and modeled gravity anomalies are also displayed in ASCII form in the ASCII layout as shown in Figure 3.8.

The differences between the observed and modeled gravity anomalies along the profile can be minimized by changing i) the geometry of the fault

plane or ii) densities of formations or iii) depths to density interfaces either independently or in combination.

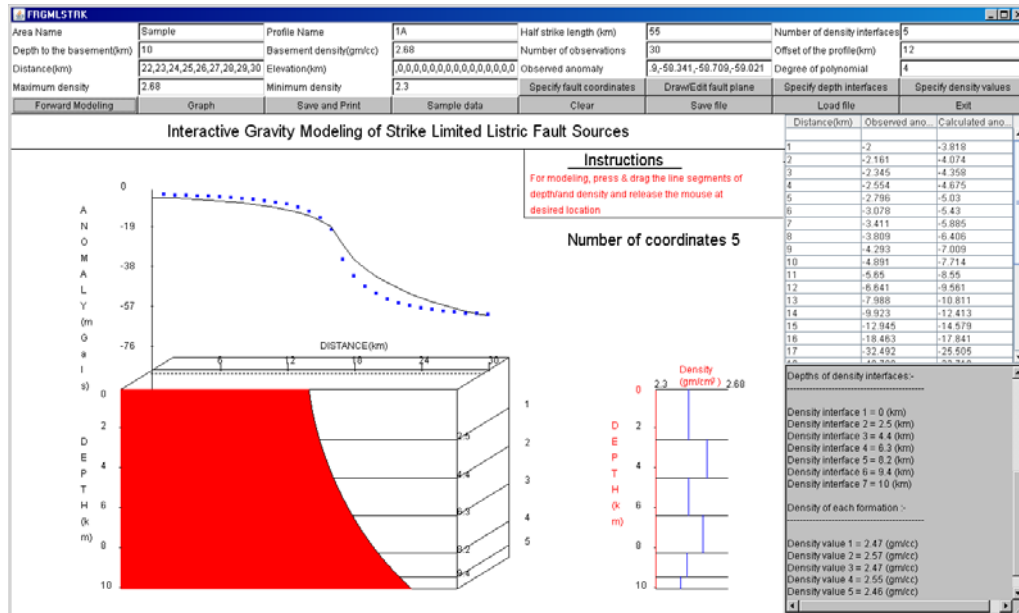


Figure 3.8 Observed and modeled gravity anomalies (anomaly panel), listric fault morphology (structural panel) and density-depth distribution (density-depth panel). The model gravity anomalies are also displayed in ASCII form in the ASCII layout

The geometry of the fault plane can be modified either by invoking 'Draw/Edit fault plane' option (Figure 3.5) or changing the degree of the polynomial. Changes to density and depth parameters can be realized by simple drag and drop mouse operations in the density-depth panel. The model gravity anomalies will be automatically updated and displayed in the anomaly panel along with the observed gravity anomalies (Figure 3.9).

Once the interpreter satisfies with the modeling, the action button 'Graph' shall be invoked to display the sub-surface structure of the hanging wall in colour mode as shown in Figure 3.10.

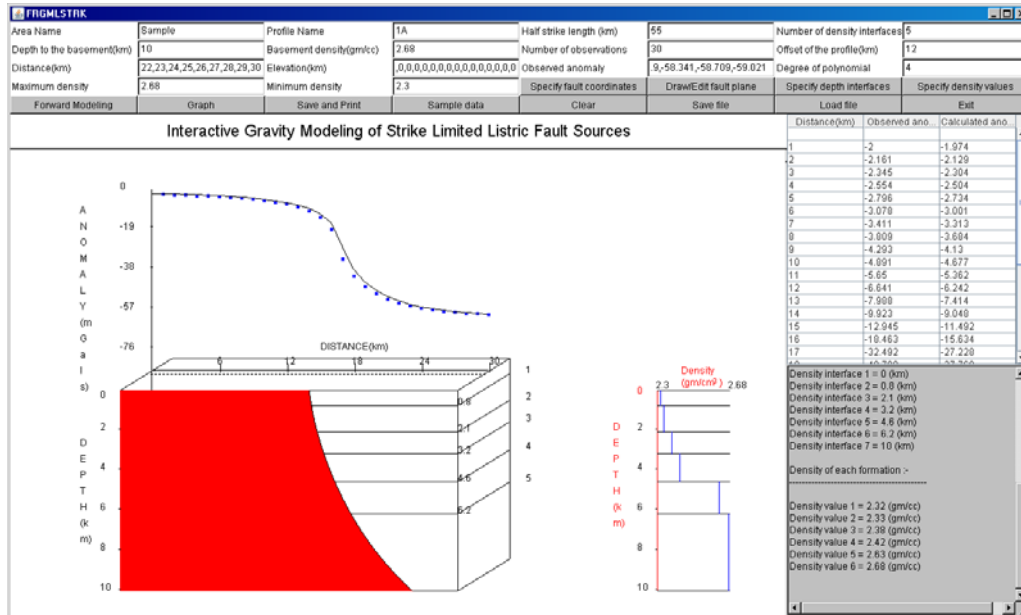


Figure 3.9 Observed and model gravity anomalies (anomaly panel), listric fault morphology (structural panel) with modified density-depth distribution (density-depth panel). The ASCII layout shows the modeled anomalies in ASCII form.

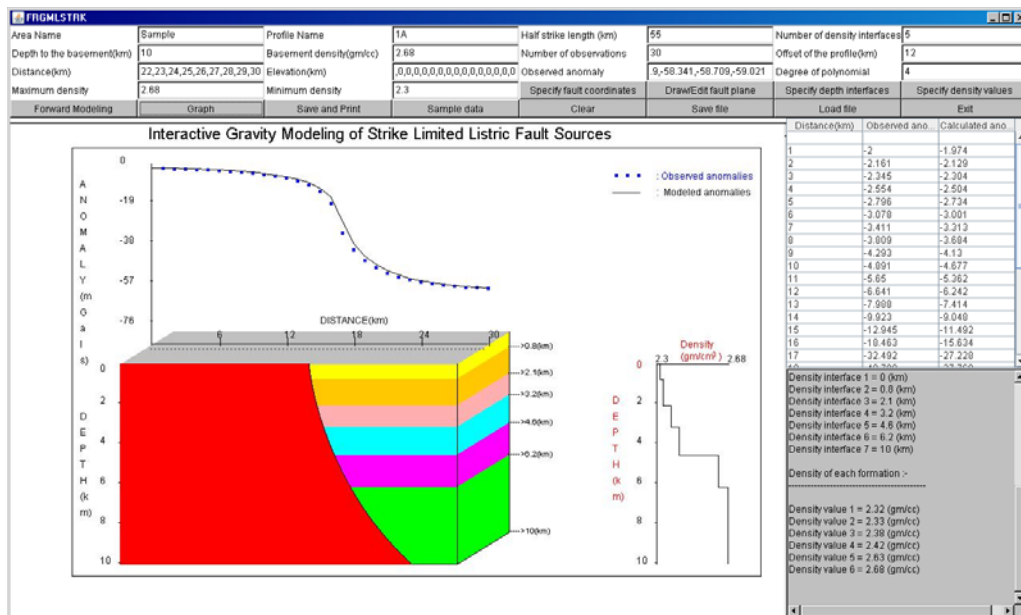


Figure 3.10 Observed and modeled gravity anomalies (anomaly panel), representation of different formations within the hanging wall by different colours (structural panel), and modified density-depth distribution (density-depth panel).

By activating the 'Save and Print' option (Figure 3.11), the interpreter saves the output (interpretation result) in both html and jpg formats at a desired location of the computing machine and opts for printing.

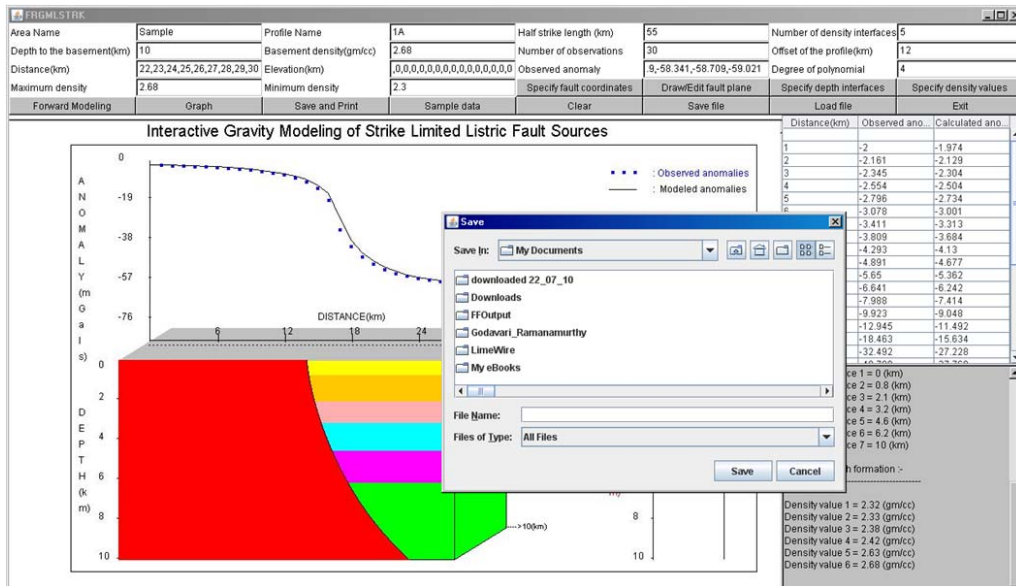


Figure 3.11 Interpreted results in html format

Upon double-click, the html file will be opened in a browser with a print dialog box attached to it. The user selects a printer and press OK to take a printout of the output (Figure 3.12).

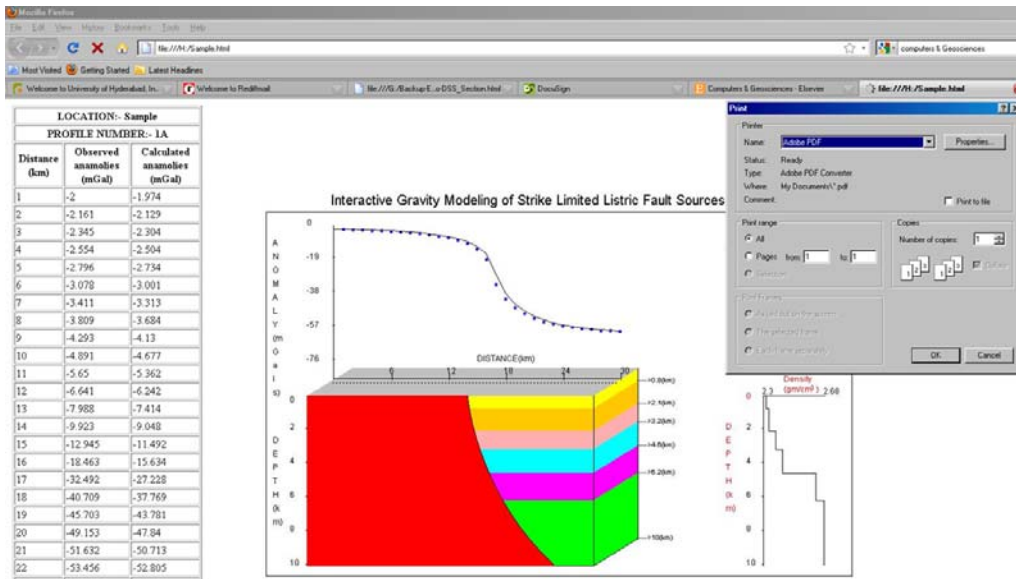


Figure 3.12 Sample output

### 3.5 Applications

The applicability of the proposed interpretation is demonstrated on a synthetic and a field gravity anomaly across the Aswaraopet master fault, India.

#### 3.5.1 Synthetic example

A set of gravity anomalies produced by a synthetic model of listric fault source (geometry is shown in the structural panel of Figure 3.13) having 40 km half-strike length at 30 km offset in the interval  $X_j \in [1, 50 \text{ km}]$  are shown in the anomaly panel of Figure 3.13 as solid dots. In the present case, a 9<sup>th</sup> degree polynomial defined with a set of ten arbitrarily chosen coefficients (Table 3.1) is used to describe the fault plane geometry (Figure 3.13). The other parameters assumed to generate the gravity anomaly of the structure are:  $z_T = 0.0 \text{ km}$ ,

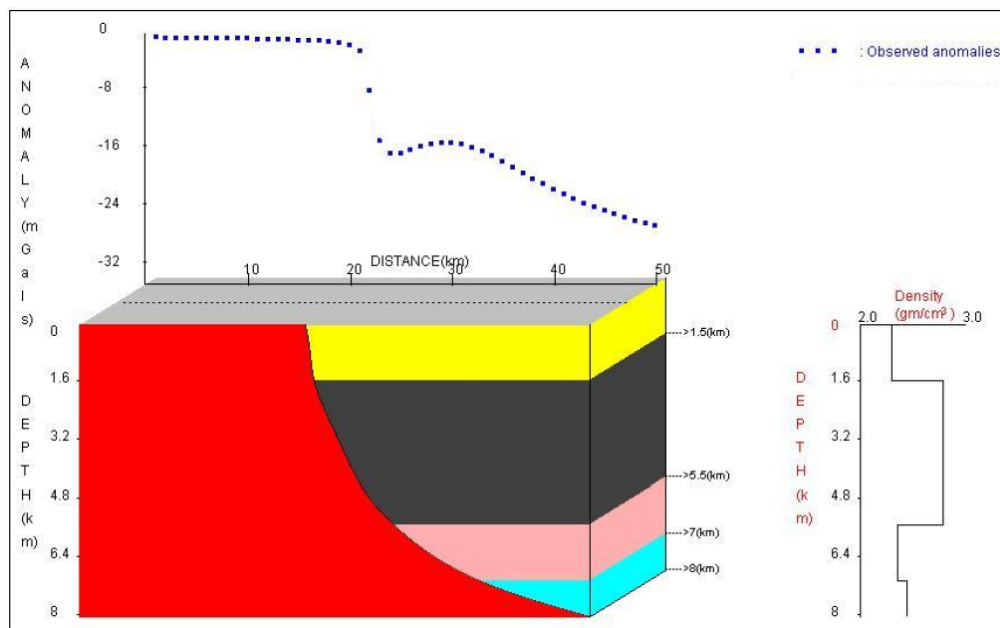


Figure 3.13 Gravity anomalies (anomaly panel) over a four-layered synthetic listric fault morphology (structural panel) with the fault plane described by a 9<sup>th</sup> degree polynomial. Assumed density-depth data within the hanging wall is shown as a step line in the density-depth panel

*Table 3.1*  
*Assumed and estimated coefficients of polynomials, synthetic example*

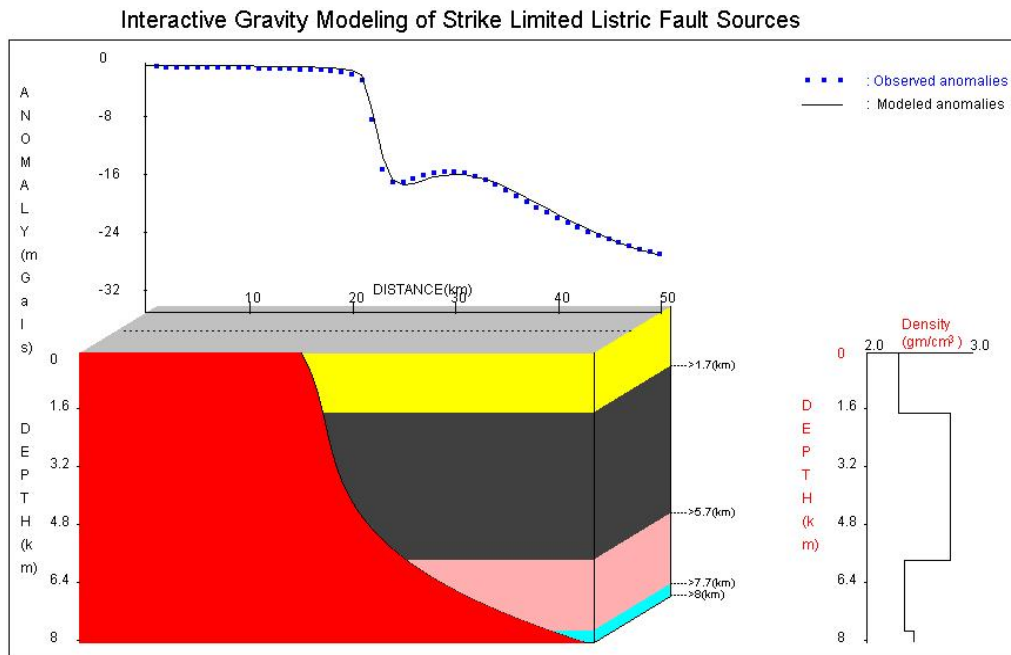
Coefficient	Assumed (9 <sup>th</sup> degree polynomial)	Estimated (3 <sup>rd</sup> degree polynomial)
$f_0$	22.1804	21.584
$f_1$	0.6284	1.9776
$f_2$	0.847	-0.5809
$f_3$	-2.3276	0.0953
$f_4$	2.1483	
$f_5$	-0.9379	
$f_6$	0.2233	
$f_7$	-0.0298	
$f_8$	0.0021	
$f_9$	-0.0001	

$z_B = 8.0$  km. The footwall of the structure shown in solid red in the structural panel (Figure 3.13) is assumed to be intact, undeformed and possesses uniform density, whereas the detached hanging wall is presumed to consist in four formations; each one has its own density (density-depth panel of Figure 3.13) and thickness (structural panel of Figure 3.13). The anomaly of such a structure varies from -0.39 mGal over the footwall to -26.21 mGal over the hanging wall with a large gradient in between (anomaly panel of Figure 3.13).

In reality, the absence of additional source(s) of information of the subsurface always makes it difficult to choose appropriate parameters pertaining to the density and thickness of the formations within the hanging wall in addition to the degree of polynomial to describe the fault plane geometry. The present code has the flexibility to incorporate such additional

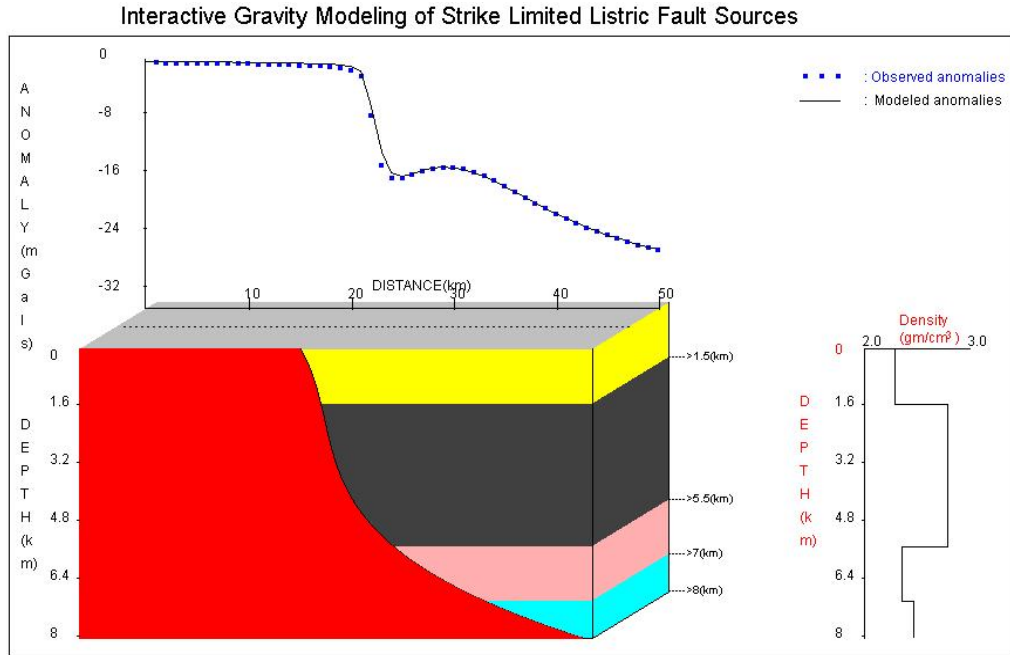
information in the model space. Accordingly, the user may incorporate or change the model parameters either independently or in combination.

To demonstrate the validity of the above statement, the gravity response of the structure (anomaly panel of Figure 3.13) is modeled presuming a 3<sup>rd</sup> degree polynomial for the fault plane and by (i) changing the depths of the formations alone while keeping the densities intact, and (ii) changing the densities of the formations alone while keeping the thicknesses unchanged. The geometry of the fault plane described by a 3<sup>rd</sup> degree polynomial with the estimated coefficients given in Table 3.1 is shown in the structural panels of Figures 3.14 and 3.15. Figure 3.14 shows the modeling result of gravity anomalies when the thickness parameters of the formations alone are changed



*Figure 3.14 Observed and modeled gravity anomalies (anomaly panel) and corresponding modeled listric fault source (structural panel) with the fault plane described by a 3<sup>rd</sup> degree polynomial. The structure is modeled by modifying the depths of the density interfaces alone.*





*Figure 3.15 Observed and modeled gravity anomalies (anomaly panel) and corresponding modeled listric fault source (structural panel) with the fault plane described by a 3<sup>rd</sup> degree polynomial. The structure is modeled by modifying the densities of the formations alone.*

and Figure 3.15 when densities alone are modified. The modeled gravity anomalies in either case closely fit the observed anomalies. Table 3.2 summarizes the differences between the assumed and estimated parameters of the structure subsequent to modeling in each case.

*Table 3.2  
Assumed and estimated parameters by interactive modeling  
Synthetic example*

Formation Number	Assumed depth (km)	Estimated depth (km)	%Error in depth	Assumed density (gm/cm <sup>3</sup> )	Estimated density (gm/cm <sup>3</sup> )	%Error in density
1	1.5*	1.7**	13.3	2.30*	2.29**	0.4
2	5.5*	5.7**	3.6	2.79*	2.79**	0
3	7.0*	7.7**	10	2.36*	2.36**	0
4	8.0*	8.0**	0	2.44*	2.47**	1.2

\* with 7<sup>th</sup> degree polynomial approximation of the fault plane

\*\* with 3<sup>rd</sup> degree polynomial approximation of the fault plane

### 3.5.2 Field Example

The practical applicability of the proposed interpretation is demonstrated with the analysis of gravity anomalies attributable to the Aswaraopet master fault of the Chintalpudi sub-basin in India.

The Chintalpudi sub-basin represents the south-easterly continuation of the Pranhita–Godavari valley. The Archaean gneisses form the basement for the Gondwana sequence within the sub-basin and on the eastern side the basin is bounded by the well-known Aswaraopet master fault, which is exposed to the surface and strikes NNW–SSE over a length of 20 km (Figure 3.16a).

Kaila et al. (1990) have carried out Deep Seismic Sounding (DSS) investigations along a profile across the basin connecting Kallur and Polavaram (Figure 3.16a). The Oil and Natural Gas Corporation Limited (ONGC), India drilled a deep borehole in the basin and encountered the Archaean basement at a depth of 2.935 km (Agarwal, 1995). The density contrast-depth data measured from this borehole of the basin is shown in Figure 3.16b (Chakravarthi, 2003). The gravity data of the basin (Figure 3.16c) has been analyzed by Chakravarthi and Sundararajan (2007) for its basement structure using a 3D inversion.

For the present study, the gravity anomalies of the basin are modeled along a profile, EE', (Figure 3.16a) with an aim to decipher the subsurface structure of the Aswaraopet fault morphology. It is to be noted that the profile, EE', is a part of the DSS profile. To start with, the inferred depth structure of

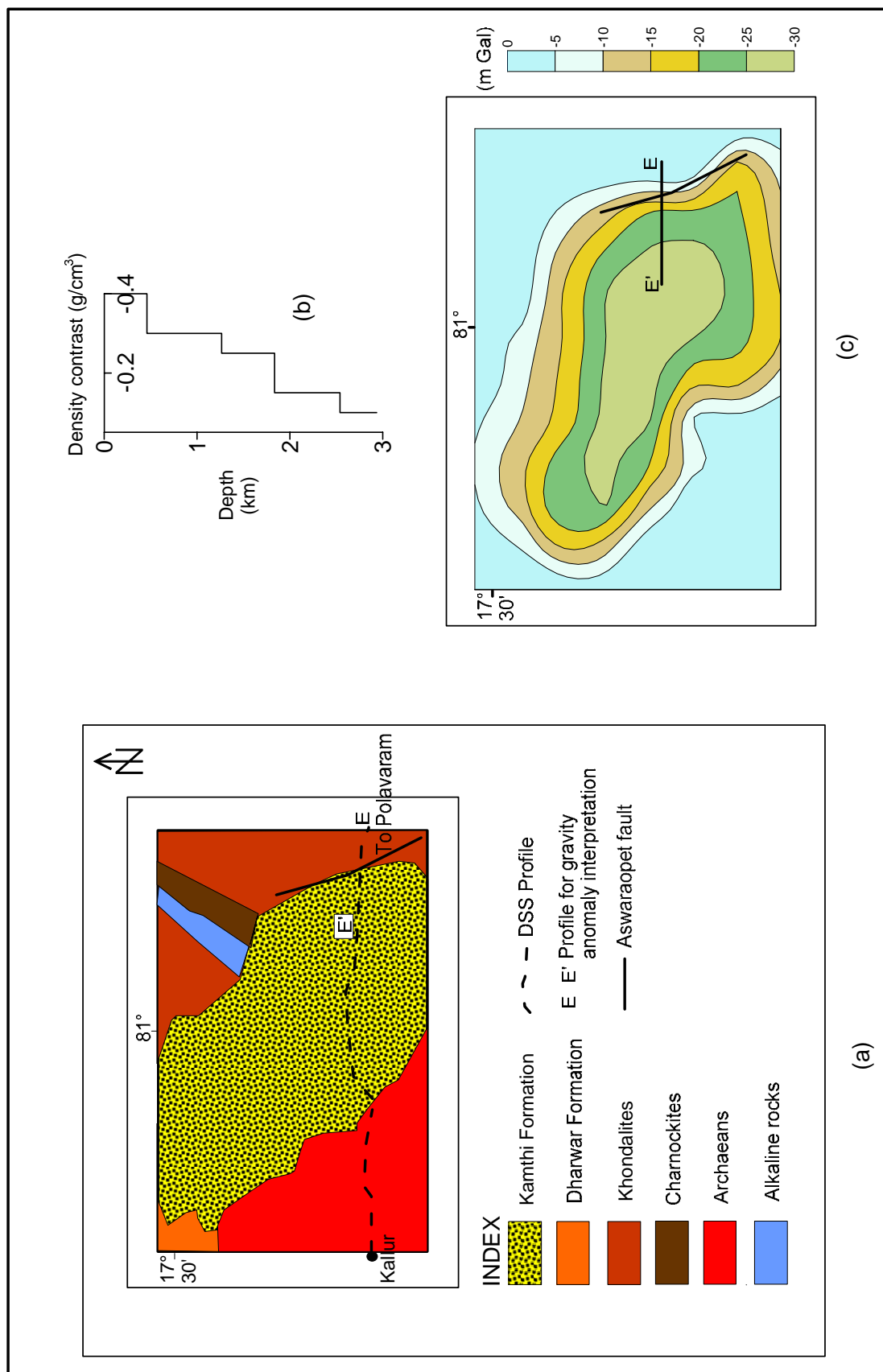


Figure 3.16 (a) Geological map of the Chintalpudi sub-basin, India (modified after Kaila et al., 1990), (b) Measured density contrast depth data, (c) Gravity anomaly map, Chintalpudi sub-basin, India (after Chakravarthi and Sundararajan, 2007)

the Aswaraopet fault derived from DSS investigations (after Kaila et al., 1990) was sampled at 25 closely spaced points in the  $xz$  plane. A 6<sup>th</sup> degree polynomial with a set of 7 coefficients (Table 3.3) was fitted to the coordinates of these selected points to analytically describe the fault plane geometry (structural panel of Figure 3.17). Although, any higher order polynomial can be used to describe the geometry of the inferred fault plane; in the present case, a 6<sup>th</sup> degree polynomial was found to be adequate.

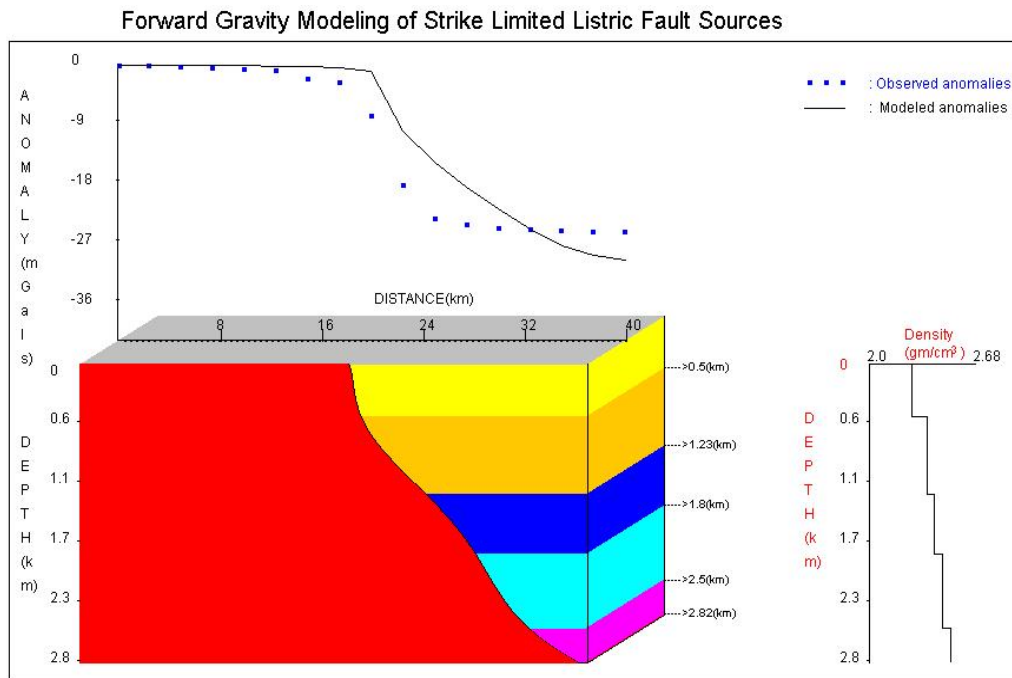
*Table 3.3*  
*Coefficients of 6<sup>th</sup> degree polynomial,*  
*Aswaraopet fault, Chintalpudi sub-basin, India.*

<b>Coefficient</b>	<b>DSS structure</b>	<b>Modeled structure</b>
$f_0$	21.2107	21.2024
$f_1$	3.3803	1.3303
$f_2$	-12.0856	-0.3347
$f_3$	26.4138	-0.0591
$f_4$	-18.4317	1.3942
$f_5$	5.3649	-1.1842
$f_6$	-0.5545	0.2826

The calculated gravity response of the fault morphology (derived from DSS studies) using the measured density contrast-depth data (Figure 3.16b) is shown in the anomaly panel of Figure 3.17. One can clearly see from Figure 3.17 that the calculated gravity response of the seismically derived structure could not explain the observed gravity anomaly.

Although the differences between the observed and calculated gravity anomalies can be minimized by several means as described in section 3.4; in this case, attempt has been made to modify the geometry of the fault plane

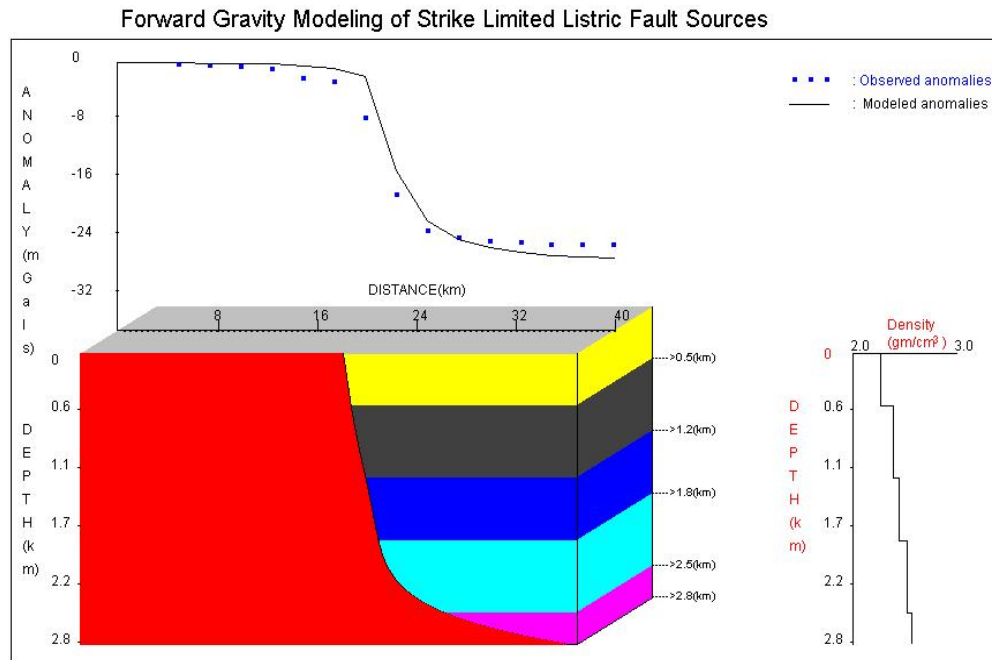
alone as the density and depth parameters used in the interpretation are the measured quantities. Accordingly, the fault plane geometry is modified to minimize the errors between the observed and calculated gravity responses (anomaly panel of Figure 3.18). The coefficients of the modified polynomial are given in Table 3.3 and corresponding fault plane geometry in Figure 3.18 respectively (structural panel). The gravity response of the modified structure is shown in the anomaly panel of Figure 3.18.



*Figure 3.17 Observed gravity anomalies (anomaly panel) across the Aswaraopet master fault, Chintalpudi sub-basin, India. The model geometry shown in the structural panel is based on the DSS investigations (after Kaila et al., 1990). The gravity response of the model is shown by a solid line in the anomaly panel*

One can observe from Figure 3.17 that the structure inferred from DSS studies shows high angle dip for the fault plane from the surface to a depth of about 0.6 km, then moderately varying dips up to 1.7 km beyond which it turns again into a high-angle normal fault. On the other hand, the present modeling

reveals that the fault plane (structural panel of Figure 3.18), which dips at high angle near the surface, shows similar attitude up to a depth of 1.7 km beyond which it shows moderate dips (Chakravarthi et al., 2014).



*Figure 3.18 Observed and model gravity response (anomaly panel) and modified depth structure (structure panel) from present modeling*

One can notice from the anomaly panels of Figure 3.17 and 3.18 that the observed anomalous field varies rather smoothly across the fault plane from the footwall to the hanging wall with a large gradient (4 mGal/km) in between. These characteristics of the observed anomaly are better explained by the interpreted model from the present modeling in comparison to the one proposed by Kaila et al. (1990).

### 3.6 Results and discussion

A method coupled with GUI based software, FRGMLSTRK, coded in JAVA has been developed to interactively model the gravity anomalies to

recover the geometries of 2.5D strike listric fault sources where the detached hanging wall of a structure consists of several geologic formations that have different densities and thicknesses. The software is simple and user friendly in the sense that it allows interactive model construction and modification, the display of fault geometry, depth and the densities of various subsurface formations, and real-time computation of the gravity anomalies arising from the model. Subsequent changes in these parameters, either independently or in combination, can be realized by simple drag and drop mouse operations.

The applicability of the code is demonstrated on a synthetic model and also is exemplified with real-world gravity anomalies arise from the eastern margin of the Chintalpudi sub-basin in India.

In case of synthetic example, the anomalies come up from a typical listric fault morphology that consists of four formations within the hanging wall and the fault plane described by a 9<sup>th</sup> degree polynomial are analyzed. The analysis showed that different combinations of density and thickness parameters and polynomial functions could equally explain the observed gravity anomalies. Appropriate model can therefore be chosen in light of the information obtained from other geophysical data.

In case of real field example, the observed gravity anomalies across the Aswaraopet master fault of the Chintalpudi sub-basin in India are analyzed using the measured density-depth data from a deep borehole. The depth

structure of the fault morphology inferred from the present method explains the observed gravity anomalies better than the model derived from DSS studies.

The advantage of the method and software is that it can be applied to analyze the gravity anomalies of faulted basin margins, even when the profile of interpretation fails to bisect the fault plane.



## Simultaneous estimation of multiple density-depth parameters and fault plane geometry from gravity inversion: Application to 2.5D strike listric fault morphologies\*

---

### 4.1 General:

Interpretation of gravity anomalies of listric fault morphologies amounts to estimating the parameters pertaining to the densities and thicknesses of the formations within the hanging wall, in addition, to decipher the geometry of the fault plane with depth. The interactive modeling technique, described in Chapter-III, is preferable to analyze the gravity anomalies as long as the unknown parameters to be estimated are limited in number. In case, the hanging wall system consists of large number geologic formations having differing densities and thicknesses, it is difficult to decide which parameter/combination of parameters to be modified appropriately in each scan. Such a constraint often impedes the speed of convergence of the solution. Hence, automatic inversion techniques are preferable to analyze the gravity anomalies of such listric fault sources (Chakravarthi and Pramod kumar, 2015b).

---

\* *Published in Geofisica Internacional, (2015, 54, 49-65), Springer.*

This chapter deals with an inversion technique and related GUI based software, GRAVLIS, to analyze the gravity anomalies of listric fault morphologies; where the structure is assumed as i) a strike limited (2.5D) source, and ii) the hanging wall system consists of several geologic formations, each one is having its own density and thickness (Figure 3.1). Based on the differences between the observed and model gravity anomalies, the present inversion estimates the unknown parameters pertaining to either thicknesses or densities of the subsurface formations (because the density and the volume of the source cannot be determined without prior information about one of them) and the coefficients of the predefined polynomial to describe the fault plane geometry (Chakravarthi and Pramod kumar, 2015b). Applicability of the technique is exemplified with the gravity anomalies of a synthetic model and also with the interpretation of real field anomalies of the Aswaraopet mater fault of the Chintalpudi sub-basin, India. The estimated parameters are compared with the assumed parameters in case of synthetic example and with measured density-depth parameters in case of field example.

## **4.2 Inversion of gravity anomalies**

Inversion of gravity anomalies is tantamount to a mathematical exercise of trying to fit the observed gravity anomalies to the anomaly expression to solve the unknown source parameters within specified convergence criteria such that the inferred model is geologically acceptable. Two variants of inversion are developed to analyze the gravity anomalies: i) densities and coefficients of the polynomial,  $\zeta(z)$ , are estimated while keeping the depths of

the density interfaces intact, and ii) depths and coefficients of the polynomial are estimated while keeping the densities of the formations intact. In either case, the interpretation starts by assigning a set of approximate parameters of the structure (densities or depths of the formations) either from known geology or the information supplemented by drilling/other geophysical methods. The proposed algorithm finds approximate location of the fault plane on the profile based on the procedure described in section 2.4 of Chapter-II followed by forward modeling realized through equation (3.1).

Because of the fact that the initial parameters are only approximate, the modeled gravity anomalies,  $g_{mod}(X_j, Z_j)$ , obtained from equation (3.1) would deviate from the observed anomalies,  $g_{obs}(X_j, Z_j)$ . The difference between these two anomalies at any point,  $P(X_j, Z_j)$ , on the profile can be expressed as a cumulative of the products of the gradient of the gravity anomaly with respect each unknown parameter to be estimated and corresponding increment as

$$g_{obs}(X_j, Z_j) - g_{mod}(X_j, Z_j) = \sum_{k=1}^N \frac{\partial g_{mod}(X_j, Z_j)}{\partial a_k} da_k + \sum_{m=0}^{N1} \frac{\partial g_{mod}(X_j, Z_j)}{\partial f_m} df_m, \quad (4.1)$$

where,  $da_k$  are increments/decrements in the parameters pertaining to either densities or depths, and  $df_m$  represents the increments/decrements to the coefficients of the polynomial used to describe the fault plane. Linear expressions similar to equation (4.1) are constructed for each observation on

the profile and  $(N+N1+1)$  normal equations are formed and solved by minimizing the misfit

$$\sum_{j=1}^{N_{obs}} [g_{obs}(X_j, Z_j) - g_{mod}(X_j, Z_j)]^2 \quad (4.2)$$

using Marquardt's algorithm (1970). The relevant system of normal equations is expressed by Chakravarthi and Pramod Kumar (2015b) as

$$\begin{aligned} \sum_j^{N_{obs}} \sum_{m=1}^{N+N1+1} \frac{\partial g_{mod}(X_j, Z_j)}{\partial a_{j'}} \frac{\partial g_{mod}(X_j, Z_j)}{\partial a_m} (1 + \tau_{mj}\delta) da_m \\ = \sum_{j=1}^{N_{obs}} Err(X_j, Z_j) \frac{\partial g_{mod}(X_j, Z_j)}{\partial a_{j'}}, j' = 1, 2, \dots, (N + N1 + 1). \end{aligned} \quad (4.3)$$

Here,  $\tau_{mj} = 1$ , for  $m = j$ ,  
 $= 0$ , for  $m \neq j$ .

Here,  $a_m = \sigma_m/z_m$  for  $m = 1, 2, \dots, N$  and  $a_m = f_{m-(N+1)}$  for  $m = N + 1, \dots, N + N1 + 1$ .

The partial derivatives required in the above system of equation (4.3) are obtained numerically by means of calculating the rate of change of gravity anomaly with respect to each unknown parameter to be estimated (Chakravarthi and Pramod Kumar, 2015b). The increments/ decrements,  $da_m$ , solved from equation (4.3) are added to/subtracted from the existing parameters,  $a_m$ , and the process goes on until one of the termination criteria of the algorithm is fulfilled as detailed in Chapter-II.

### 4.3 Description of the software – GRAVLIS

Based on the proposed methodology of inversion described in the text, GUI based software named, GRAVLIS, coded in JAVA has been developed to analyze the gravity anomalies produced by 2.5D strike listric fault sources (Annexure 4-A). This software, which works on the MVC pattern (Figure 2.2), can be operated on any platform with preloaded JAVA kit. The module ‘Model’ performs the task of finding the origin of the fault plane from the observed gravity anomalies, computes the modeled gravity anomalies of the structure, constructs and solves the system of normal equations besides performing the business logic of the algorithm. The ‘View’ module reads the input data and displays the animated versions of the inversion process, whereas the ASCII layout provides the interpreted results in ASCII form. The role of ‘Control’ unit is to pass on the required actions to the model and view modules as and when necessary.

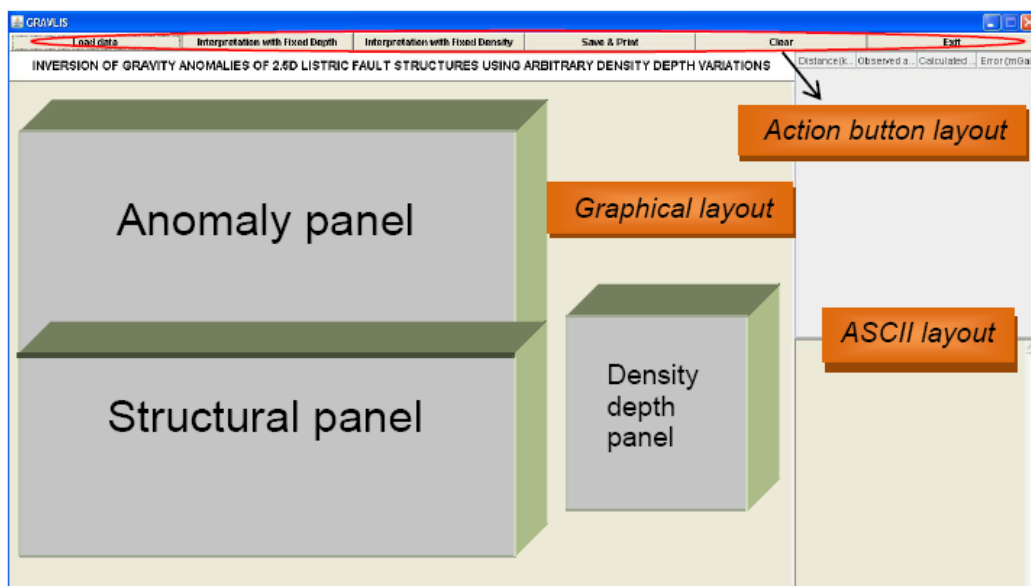


Figure 4.1 View module of GRAVLIS

The view module (Figure 4.1), which appears on the screen upon invoking the batch file, is further organized into the action button layout on the top, graphical layout in the bottom and ASCII layout in the right. The input to the code consists of the area name, profile ID, number of observations, distance and elevation of each observation expressed in km, observed anomaly in mGal, half-strike of the structure in km, offset of the profile in km, number of formations within the hanging wall, basement and formations densities in  $\text{gm/cm}^3$ , initial/approximate depths of interfaces in km (in case of depths and fault plane inversion) or initial/approximate densities of formations in  $\text{gm/cm}^3$  (in case of density and fault plane inversion), number of iterations to be performed and the allowable error.

The user enters the enlisted input parameters in a Microsoft Excel sheet and reads the file to the software by means of 'Load data' action button (Figure 4.1). The proposed variants of inversion (described in section 4.2) can be performed on the input data by selecting either 'Interpretation with Fixed Depth' or 'Interpretation with Fixed Density' option from the action button layout (Figure 4.1).

During the process of inversion, this software displays i) the changes in fault plane geometry, ii) changes in thicknesses/densities of the formations (depending upon the selected option), and iii) nature of fit between the observed and modeled gravity anomalies in animated forms in respective panels. The 'Save and Print' action button enables the user to save the interpreted results in both html and jpg formats and allows for printing.

## 4.4 Applications

The proposed inversion is applied over two anomaly profiles to demonstrate its validity and applicability; one synthetic and the other real.

### 4.4.1 Synthetic Example

Figure 4.2a shows a set of noisy gravity anomalies attributable to a synthetic model of 2.5D strike listric fault source, whose geometry is shown in Figure 4.2b. The structure has a half strike length of 50 km. The anomalies (shown as solid dots in Figure 4.2a) are produced at zero offset on a horizontal plane in the interval  $x_j \in [0, 80 \text{ km}]$ . In this case, the pseudorandom noise was Gaussian, with zero mean and a standard deviation of 0.14 mGal. The footwall remains intact and undisturbed, whereas the detached hanging wall consists in four formations namely: a massive basalt of 3.5 km thick at the top is followed successively by 1.5 km thick sediments, 3 km thick vesicular basalt and 2.0 km thick compacted sediments above the basement. The assumed densities of four formations under consideration are given in Table 4.1 and shown in Figure 4.2c.

*Table 4.1*  
*Assumed and estimated densities in case of synthetic example*

Formation	Assumed density (gm/cm <sup>3</sup> )	Initial density (gm/cm <sup>3</sup> )	Estimated density (gm/cm <sup>3</sup> )	Error (%)
Compact basalt	2.9	2.0	2.9	0.0
Sediments	2.4	2.0	2.44	1.67
Vesicular basalt	2.8	2.0	2.78	0.71
Compacted sediments	2.5	2.0	2.51	0.4

## 2.5D Inversion - Synthetic Example

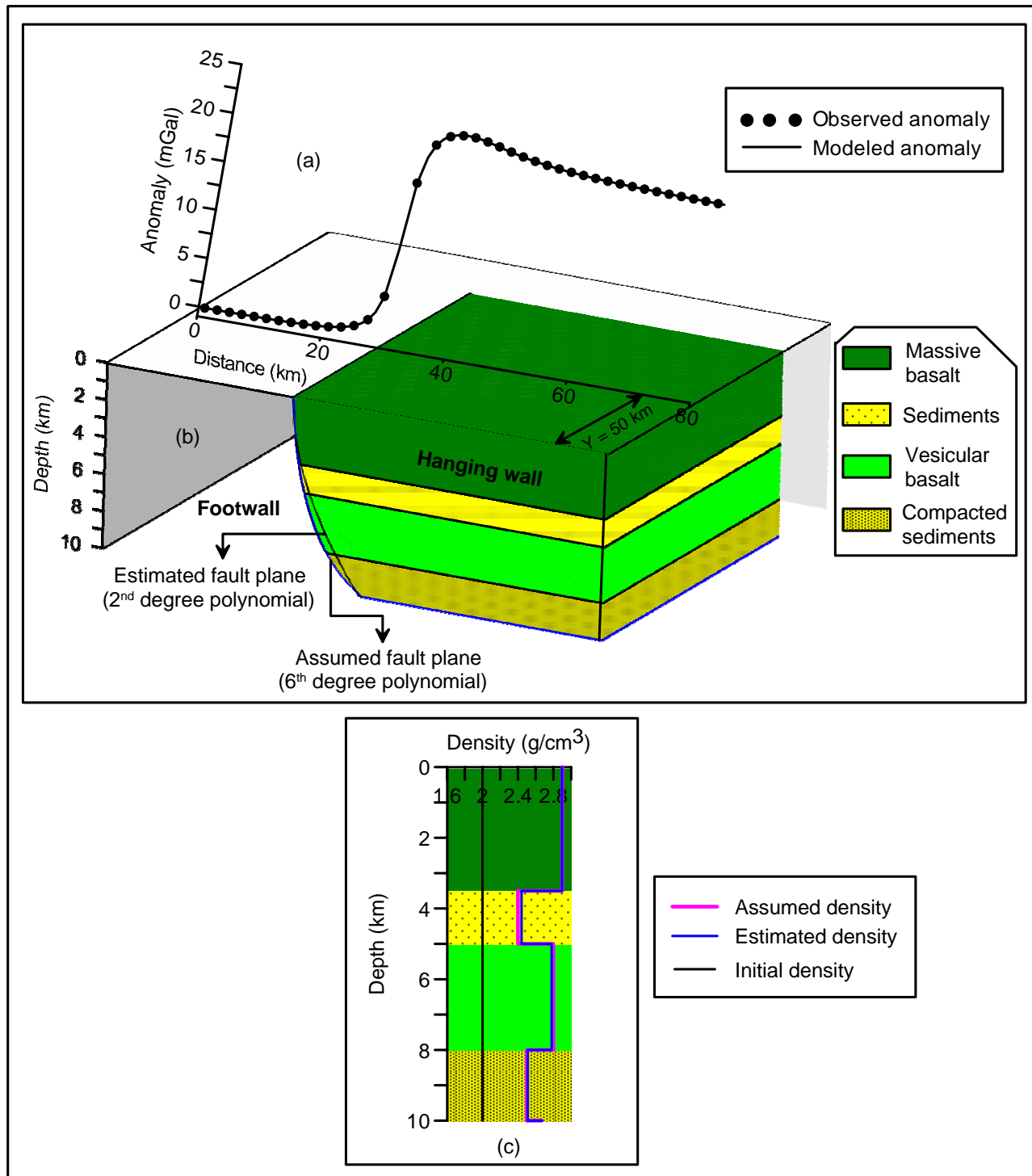


Figure 4.2 (a) Observed and modeled noisy gravity anomalies, (b) four layered hanging wall system of a synthetic listric fault source with assumed and modeled fault planes described by 6<sup>th</sup> and 2<sup>nd</sup> degree polynomials, (c) assumed, initial and modeled densities. Depths of density interfaces are fixed during the inversion



In this case, the density of the undisturbed footwall is assumed as 2.67 gm/cm<sup>3</sup>. Further, a 6th degree polynomial with a set of seven arbitrarily chosen coefficients (Table 4.2) is used to describe the geometry of the fault plane (shown as solid line in blue in Figure 4.2b).

*Table 4.2*  
*Assumed and estimated coefficients of the polynomial,  $\zeta(z)$ , synthetic example*

<b>Coefficient</b>	<b>Assumed coefficients of 6<sup>th</sup> degree polynomial</b>	<b>Estimated coefficients of the 2<sup>nd</sup> degree polynomial in case of density and fault plane inversion</b>	<b>Estimated coefficients of the 2<sup>nd</sup> degree polynomial in case of depth and fault plane inversion</b>
$f_0$	30.0144	30.0346	30.0299
$f_1$	0.0965	0.0729	0.0765
$f_2$	0.1845	0.1059	0.1102
$f_3$	-0.0732		
$f_4$	0.0171		
$f_5$	-0.0017		
$f_6$	7E-005		

Treating the noisy anomalies shown in Figure 4.2a (solid dots) as the observed ones, inversion was carried out in two prong strategies as discussed in section 4.2. In either case, a 2<sup>nd</sup> degree polynomial was used in the inversion to simulate the fault plane geometry, in place of a 6<sup>th</sup> degree, in order to study its effect on the interpretation, if any.

#### ***4.4.1.1 Simultaneous estimation of densities and fault plane geometry***

Initially a value of 2.0 gm/cm<sup>3</sup> has been assigned to the density of each subsurface formation (Table 4.1 and Figure 4.2c) prior to perform the inversion on noisy anomalies. The algorithm calculates the density contrast of each subsurface formation with respect to the footwall and uses them to compute the

gravity effect of the structure. The approximate location of the fault plane identified by the algorithm is at 30.07 km on the profile. The code had performed 69 iterations before it got terminated as the resulting misfit (equation 4.2) fell below a predefined allowable error of 0.017 mGal. No significant improvements in either densities or coefficients of the polynomial are observed beyond the concluding iteration (Figure 4.3b).

The modeled gravity anomalies (shown as a solid line in Figure 4.2a) at the end of the 69<sup>th</sup> iteration closely fit the observed ones. A maximum error of 0.044 mGal between the observed and modeled gravity anomalies is observed exactly at the 36<sup>th</sup> km on the profile (Figure 4.3a). The value of misfit had reduced drastically from its initial value of 3550019 to 1.43 at the end of the 34<sup>th</sup> iteration and then gradually reaches to 0.016 at the end of the concluding iteration (Figure 4.3b). The estimated parameters pertaining to the densities and coefficients of the 2<sup>nd</sup> degree polynomial from the inversion are given Table 4.1 and Table 4.2 and shown graphically in Figure 4.2c and 4.2b respectively. The percentage of error between the assumed and estimated densities is given in Table 4.1. Furthermore, the changes took place in each estimated parameter (densities and coefficients of the polynomial) against the iteration number are illustrated in Figure 4.3b. It is noticed from Figure 4.2b that the modeled fault plane by a 2<sup>nd</sup> degree polynomial deviates only marginally from the assumed fault plane described with a 6<sup>th</sup> degree polynomial.

The predicted density (Table 4.1 and Figure 4.2c) of the bottom most sedimentary pulse at depth is marginally overestimated, whereas the densities

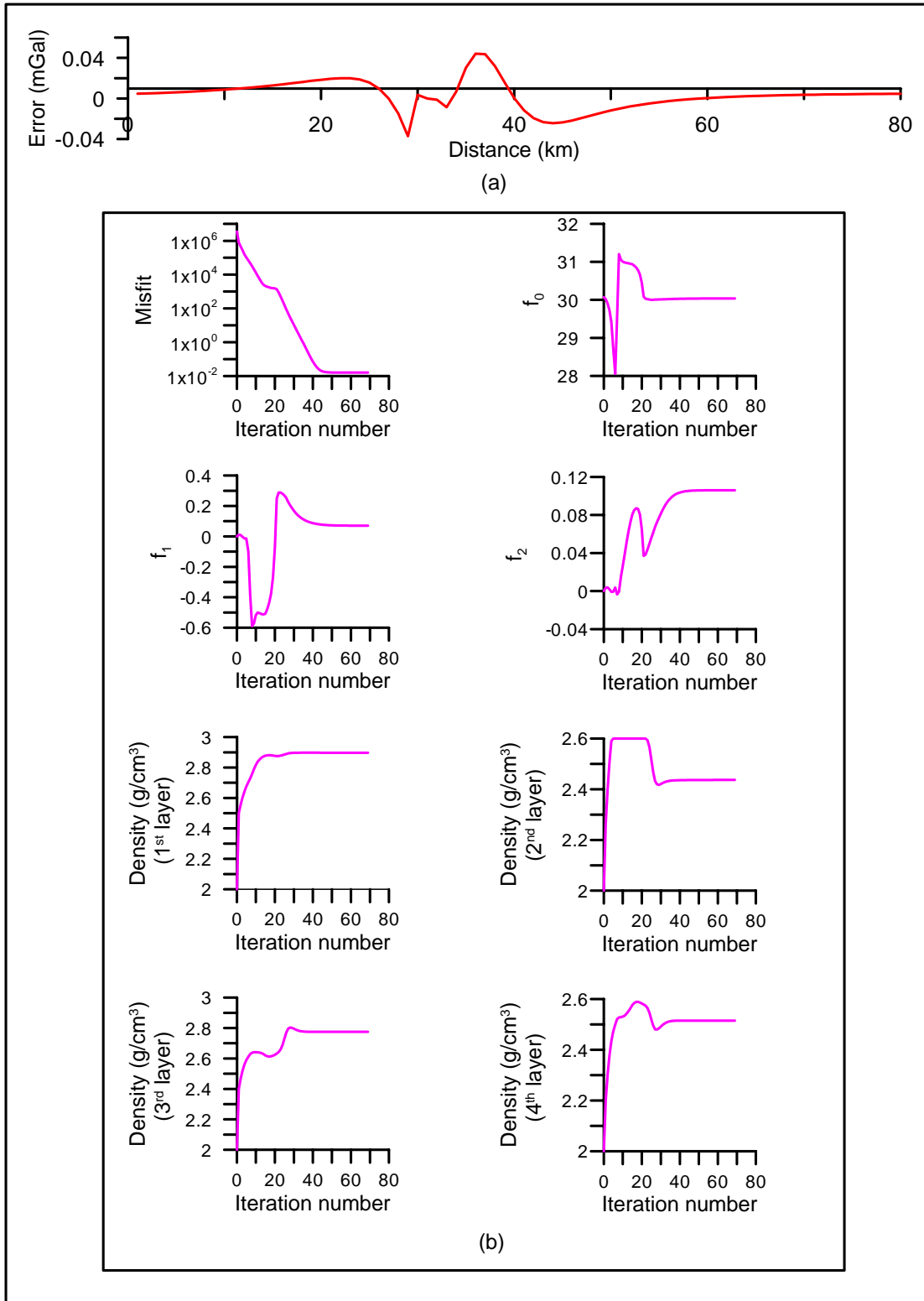


Figure 4.3 (a) Error analysis between the observed and modeled gravity anomalies, (b) changes in misfit, coefficients of a 2<sup>nd</sup> degree polynomial, and densities of subsurface formations against the iteration number.

of compact and vesicular basalts are slightly underestimated. Such errors between assumed and estimated densities are acceptable considering the presence of significant level of pseudorandom noise in the anomalies produced by the structure. Therefore, the fault plane whether it is described by a 2<sup>nd</sup> degree or a 6<sup>th</sup> degree polynomial does not appreciably affect the fault plane geometry and the estimated densities of the structure.

The sample output generated by GRAVLIS in html format is shown in Annexure 4-B.

#### ***4.4.1.2 Simultaneous estimation of depths and fault plane geometry***

The inversion process is repeated on the noisy gravity anomalies to estimate the depths of four concealed density interfaces and three coefficients of the polynomial by keeping the density parameters unchanged. In this case, the initial depths assigned to four density interfaces are given in Table 4.3 and shown in Figure 4.4c (dotted lines).

One can notice from Table 4.3 that the initial depths of density interfaces are significantly differ from the assumed/true model parameters. As in the previous case, the approximate location of the fault plane identified by the algorithm at 30.07 km was assigned to the first coefficient of the polynomial, whereas other coefficients are set to zero.

For such an inversion, the code took 45 iterations before it got terminated. The misfit (equation 4.2) had reduced from its initial value of

## 2.5D Inversion - Synthetic example

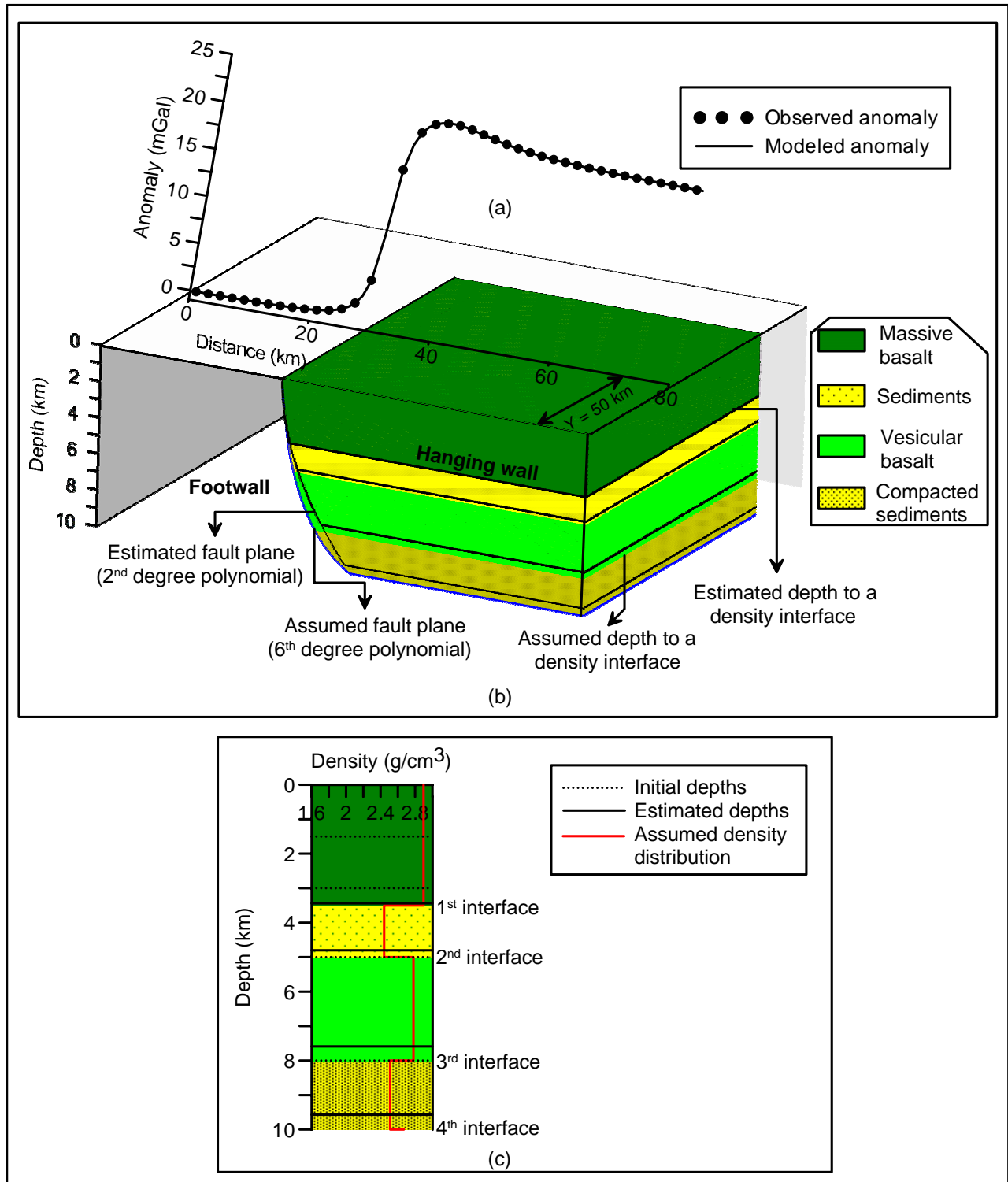


Figure 4.4 (a) Observed and modeled noisy gravity anomalies, (b) four layered hanging wall system of a synthetic listric fault source with assumed and modeled fault planes described by 6<sup>th</sup> and 2<sup>nd</sup> degree polynomials, (c) assumed and estimated depths to density interfaces. Densities remain unchanged during the inversion

45565.3 for the starting model to 0.7 at the end of the 19<sup>th</sup> iteration and then slowly attained to 0.004 at the end of the 45<sup>th</sup> iteration (Figure 4.5b). No appreciable changes in estimated depths and coefficients of the polynomial are found beyond the concluding iteration (Figure 4.5b).

*Table 4.3*  
*Assumed and estimated depths of density interfaces, synthetic example*

<b>Interface</b>	<b>Assumed depth (km)</b>	<b>Initial depth (km)</b>	<b>Estimated depth (km)</b>	<b>Error (%)</b>
<b>Compact basalt/sediments</b>	3.5	1.5	3.44	1.71
<b>Sediments/Vesicular basalt</b>	5.0	3	4.8	4.0
<b>Vesicular basalt/compact sediments</b>	8.0	5	7.6	5.0
<b>Compact sediments/basement</b>	10.0	8	9.57	4.3

The fit between the observed and modeled gravity anomalies at the end of the 45<sup>th</sup> iteration is satisfactory (Figure 4.4a). The estimated depths to four density interfaces are given in Table 4.3 and shown graphically in Figures 4.4b and 4.4c (solid lines). The estimated coefficients of the 2<sup>nd</sup> degree polynomial to describe the fault plane are given in Table 4.2 and shown in Figure 4.4b. By and large, the modeled fault plane (simulated by a 2<sup>nd</sup> degree polynomial) closely mimics the assumed one described by a 6<sup>th</sup> degree polynomial (Figure 4.4b). In this case, a maximum error of -0.021 mGal between the observed and modeled gravity anomalies is observed at the 40<sup>th</sup> km on the profile (Figure 4.5a). The changes in the estimated parameters (depths to density interfaces

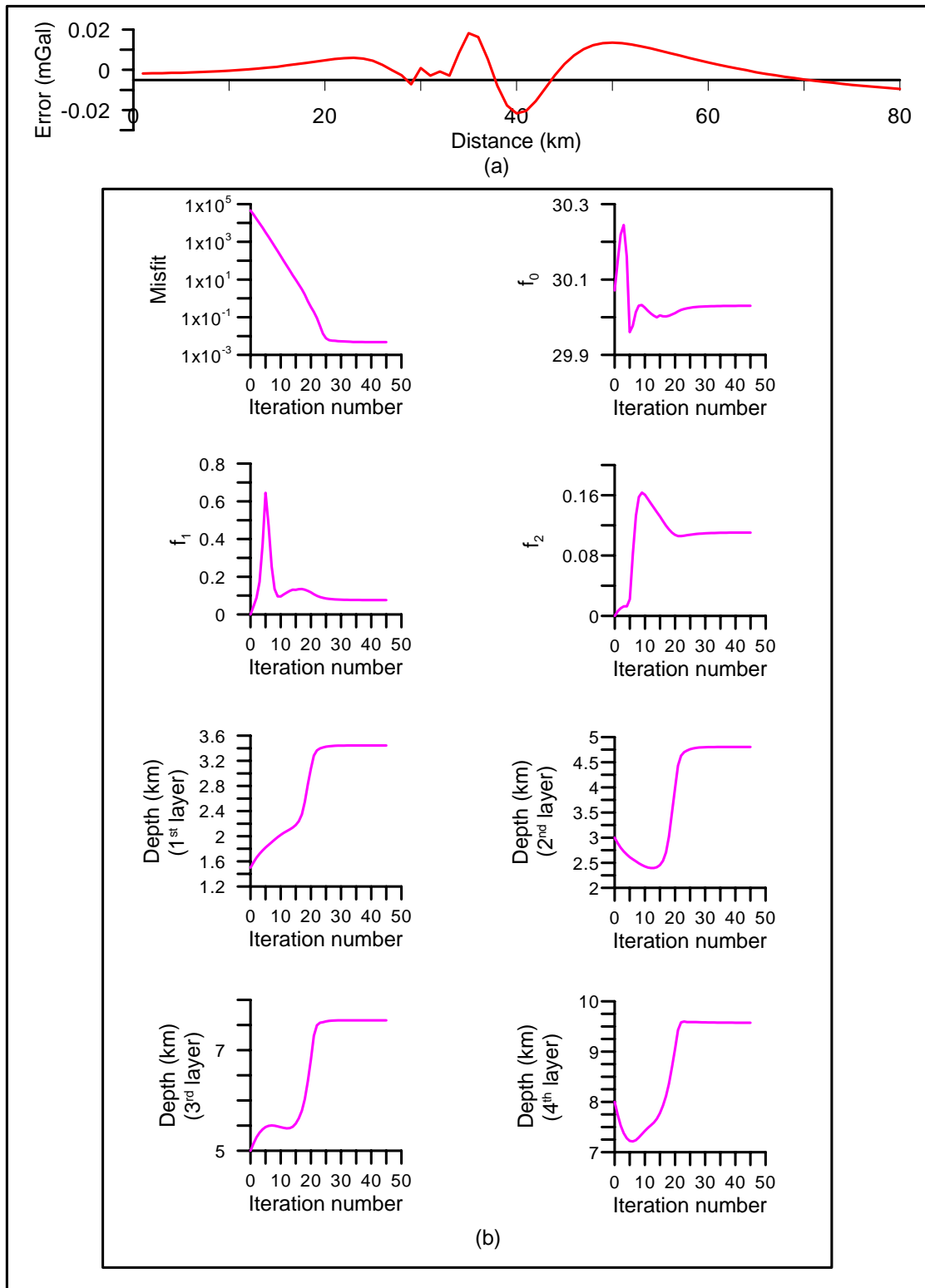


Figure 4.5 (a) Error analysis between the observed and modeled gravity anomalies, (b) changes in misfit, coefficients of a 2<sup>nd</sup> degree polynomial, and depths of various density interfaces against the iteration number.

and coefficients of the 2<sup>nd</sup> degree polynomial) against the iteration number are illustrated in Figure 4.5b.

It is to be noted from Table 4.3 and Figure 4.4c that the estimated depths to the four density interfaces are marginally underestimated, with a maximum error of 5% found at the interface between vesicular and compact sediments at depth. However, these errors between the assumed and estimated parameters are insignificant considering the fact that the observed anomalies of the structure are noisy.

In short, the fault plane whether it is described by a 2<sup>nd</sup> degree or a 6<sup>th</sup> degree does not appreciably affect the estimated densities or depths of the formations within the hanging wall of the structure. However, the choice of a 2<sup>nd</sup> degree polynomial in the inversion has resulted in slightly underestimating the amount of extension when the anomalies are analyzed to estimate the depths of the density interfaces (Chakravarthi and Pramod Kumar, 2015b).

#### **4.4.2 Field Example**

The proposed inversion technique is applied to analyze the observed gravity anomalies (shown in the anomaly panel of Figure 3.17) across the Aswaraopet master fault of the Chintalpudi sub-basin in India. It is to note that the same profile was interpreted in section 3.5.2 of Chapter-III by interactive modeling. In this case, the anomalies are scaled with reference to an observation in the vicinity of the exposed fault trace. A total of 29 gravity anomalies spread over a profile length of 24 km across the fault trace are



## 2.5D Inversion - Field Example

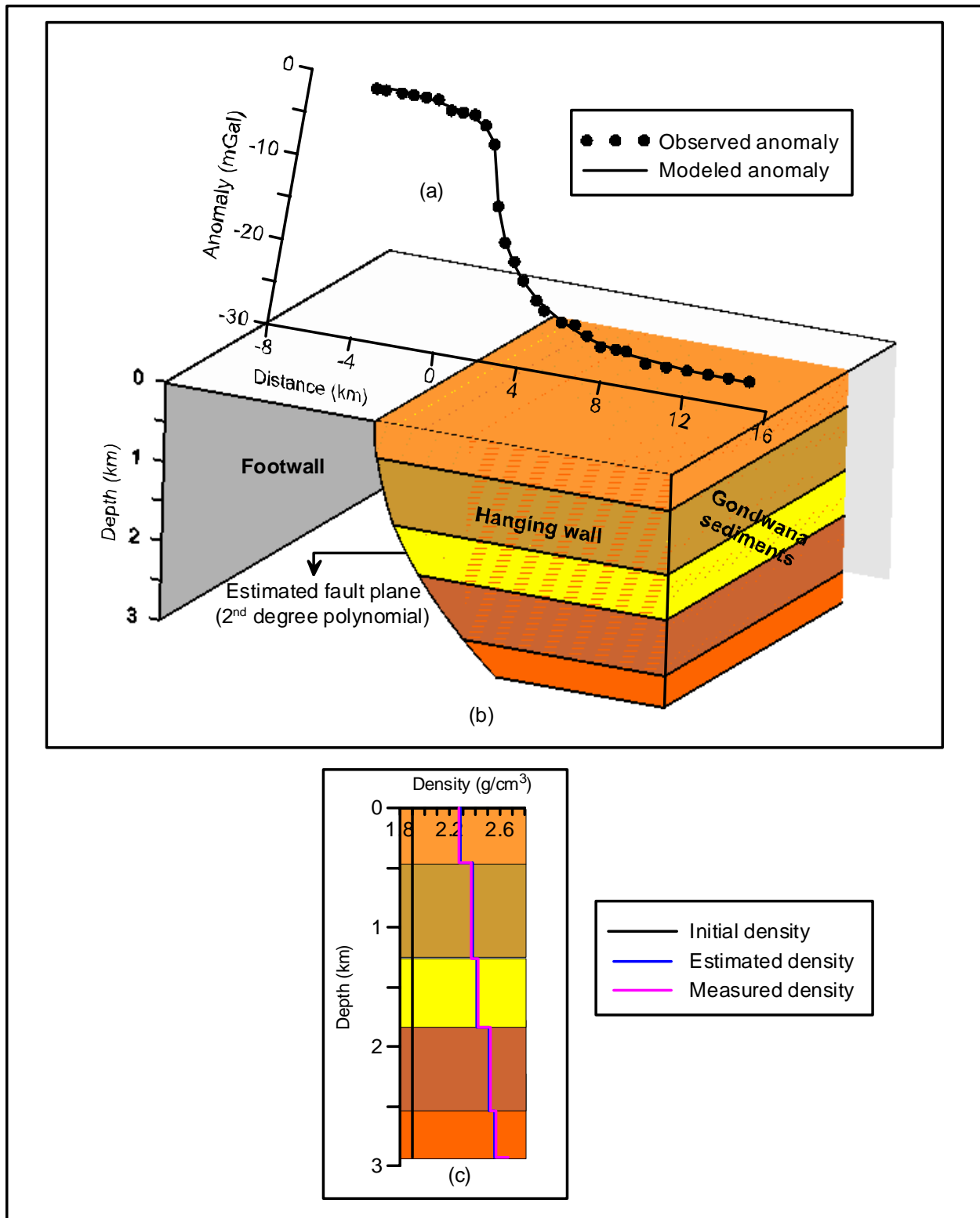


Figure 4.6 (a) Observed and modeled gravity anomalies, (b) inferred geometry of the Aswaraopet master fault morphology, Chintalpudi subbasin, India, (c) measured, initial and estimated densities. Depths of density interfaces are fixed during the inversion

considered for the inversion. The gravity anomalies attain minimum and maximum magnitudes (absolute) over the footwall and hanging wall systems of the fault structure and hence the choice of considering a profile length of 24 km across the fault ramp is justified.

As in the case of synthetic example, discussed in Chapter 4.4.1, the observed anomalies are analyzed in two ways. In either case, a 2<sup>nd</sup> degree polynomial is opted in the inversion to simulate the geometry of the fault plane. The initial/approximate parameters pertaining to densities (in case the inversion is performed for estimating the densities and polynomial coefficients) and depths (in case the inversion is performed for estimating the depths and polynomial coefficients) are given in Table 4.4 and Table 4.5 and graphically shown in Figure 4.6c and 4.8c (solid lines in black) respectively.

*Table 4.4*  
*Measured and estimated densities, Chintalpudi subbasin, India.*

<b>Formation</b>	<b>Measured density (gm/cm<sup>3</sup>)</b>	<b>Initial density (gm/cm<sup>3</sup>)</b>	<b>Estimated density (gm/cm<sup>3</sup>)</b>	<b>Error (%)</b>
<b>1</b>	2.27	1.9	2.279	0.396
<b>2</b>	2.37	1.9	2.380	0.422
<b>3</b>	2.42	1.9	2.410	0.410
<b>4</b>	2.52	1.9	2.517	0.119
<b>5</b>	2.57	1.9	2.562	0.311

Although, the measured density-depth data of the basin is known apriori, the inversion was performed on the observed anomalies to examine whether the estimated parameters from the inversion mimic the measured ones or not. It can be seen from Tables 4.4 and 4.5 and Figures 4.6c and 4.8c that the initial/

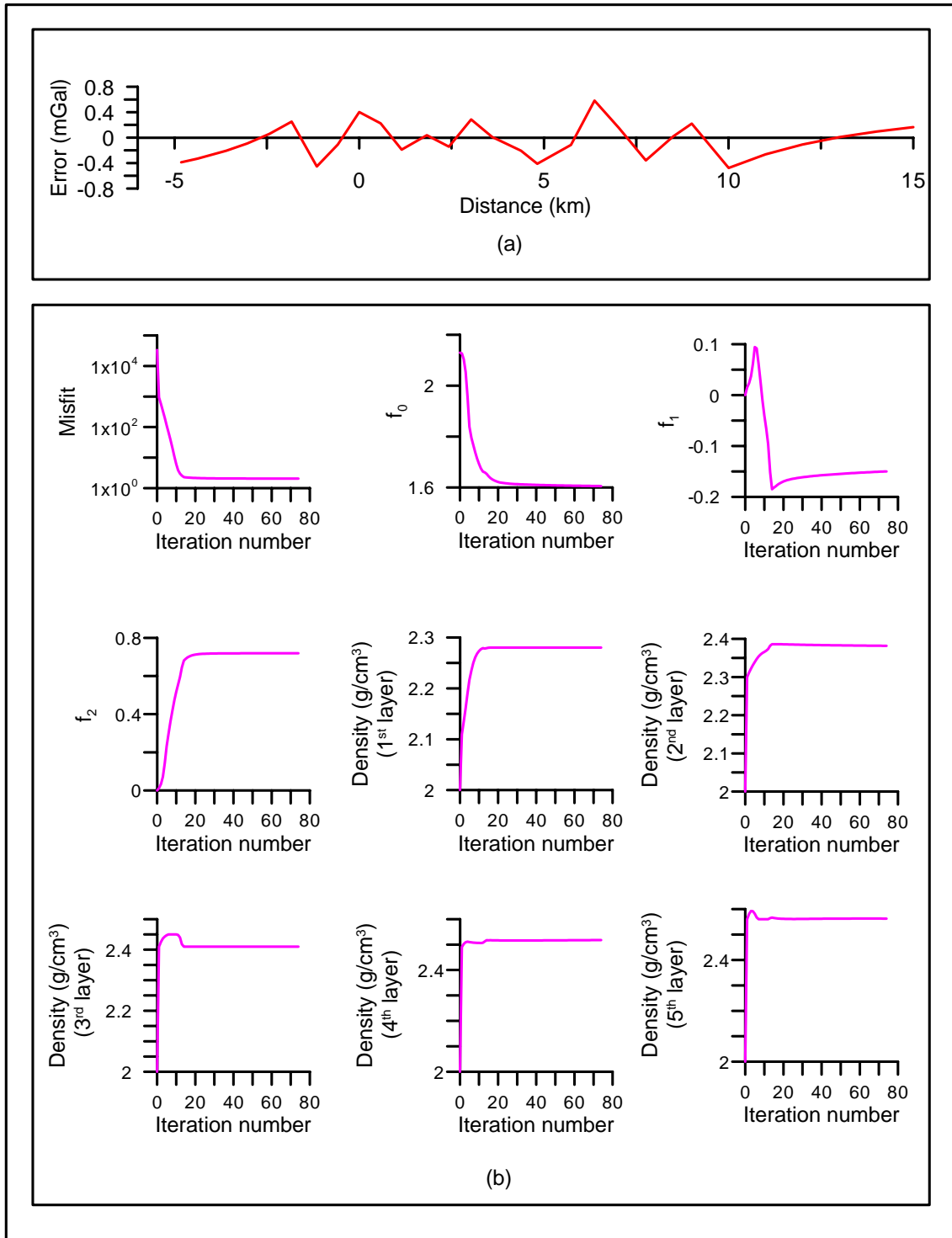


Figure 4.7 (a) Error analysis between the observed and modeled gravity anomalies across the Aswaraopet master fault morphology, Chintalpudi subbasin, India, (b) changes in misfit, coefficients of a 2<sup>nd</sup> degree polynomial, and densities of subsurface formations against the iteration number.

approximate parameters significantly differ in magnitude from the measured quantities.

*Table 4.5*  
*Measured and estimated depths to density interfaces,*  
*Chintalpudi subbasin, India*

<b>Formation</b>	<b>Measured depth (km)</b>	<b>Initial depth (km)</b>	<b>Estimated depth (km)</b>	<b>Error (%)</b>
<b>1</b>	0.46	0.2	0.43	6.5
<b>2</b>	1.265	0.9	1.10	13.0
<b>3</b>	1.835	1.2	1.87	1.9
<b>4</b>	2.54	2	2.33	8.3
<b>5</b>	2.935	2.5	3.01	2.5

The algorithm had identified the approximate location of the fault plane at 2.13 km in each case. Initially, this value was assigned to the first coefficient of the polynomial in either case, whereas the other coefficients were set to zero. The algorithm had performed 74 and 16 iterations respectively before terminating. The estimated parameters remained more or less unchanged beyond the concluding iterations (Figure 4.7b and Figure 4.9b). The modeled gravity anomalies are shown in Figure 4.6a and Figure 4.8a as solid lines. The fit between the observed and modeled gravity anomalies in either case is satisfactory (Figure 4.6a and Figure 4.8a). A maximum error of 0.58 mGal between the observed and modeled gravity anomalies is observed at 6.3 km on the profile (Figure 4.7a) when the inversion was performed to estimate the densities and fault plane geometry, whereas a maximum error of 0.64 mGal (absolute) is observed at the 10<sup>th</sup> km (Figure 4.9a) when the anomalies are inverted for depths and fault plane geometry. The estimated density and depth parameters subsequent to respective inversions are given in Table 4.4 and

## 2.5D Inversion - Field Example

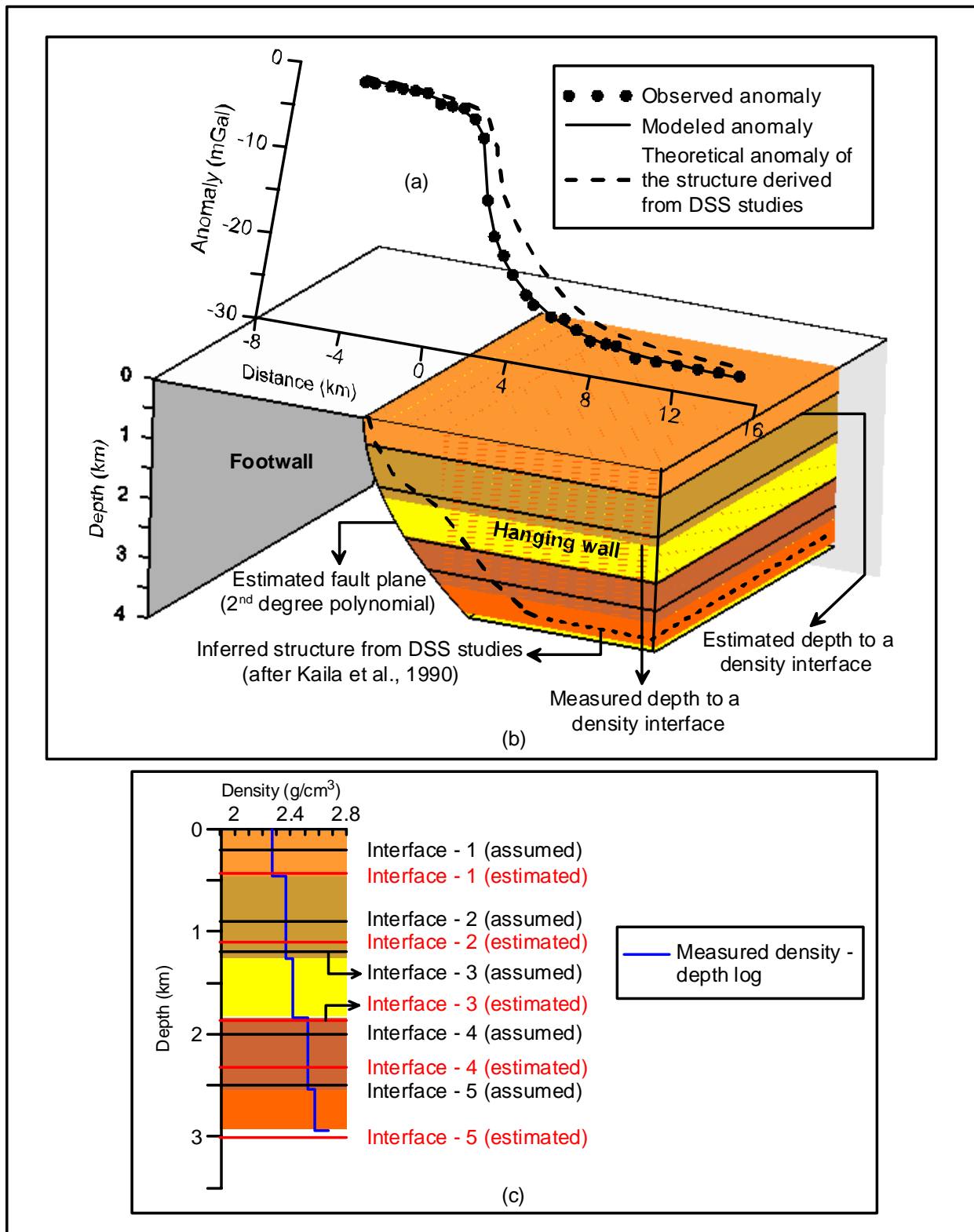


Figure 4.8 (a) Observed and modeled gravity anomalies, (b) inferred geometry of the Aswaraopet master fault, Chintalpudi subbasin, India, (c) density-depth relationship. Anomalies are analyzed to estimate the depths of density interfaces

Table 4.5 and shown in Figure 4.6c and Figure 4.8c respectively. The errors (%) between the estimated and measured parameters in each case are also given in Tables 4.4 and 4.5. The inferred coefficients of the polynomial to construct the fault plane geometry in either case are tabulated in Table 4.6 and illustrated in Figures 4.6b and 4.8b respectively.

*Table 4.6*  
*Estimated coefficients of the 2<sup>nd</sup> degree polynomial,*  
*Chintalpudi subbasin, India.*

<b>Coefficient</b>	<b>Estimated coefficients of the 2<sup>nd</sup> degree polynomial in case of densities and fault plane inversion</b>	<b>Estimated coefficients of the 2<sup>nd</sup> degree polynomial in case of depths and fault plane inversion</b>
$f_0$	1.606	1.564
$f_1$	-0.149	0.143
$f_2$	0.719	0.506

When the anomalies are subjected for inversion to estimate densities and the fault plane geometry, the modeled densities of the first and second formations are slightly overestimated (~0.4%) while others marginally underestimated (Table 4.4 and Figure 4.6c).

On the other hand, when the inversion was performed for estimating both depths and fault plane geometry simultaneously, the modeled depths of the first, second and fourth density interfaces are modestly underestimated whereas the third and fifth density interfaces are slightly overestimated (Table 4.5 and Figure 4.8c). The changes in the estimated parameters with the iteration number in each case are shown in Figures 4.7b and Figure 4.9b respectively.

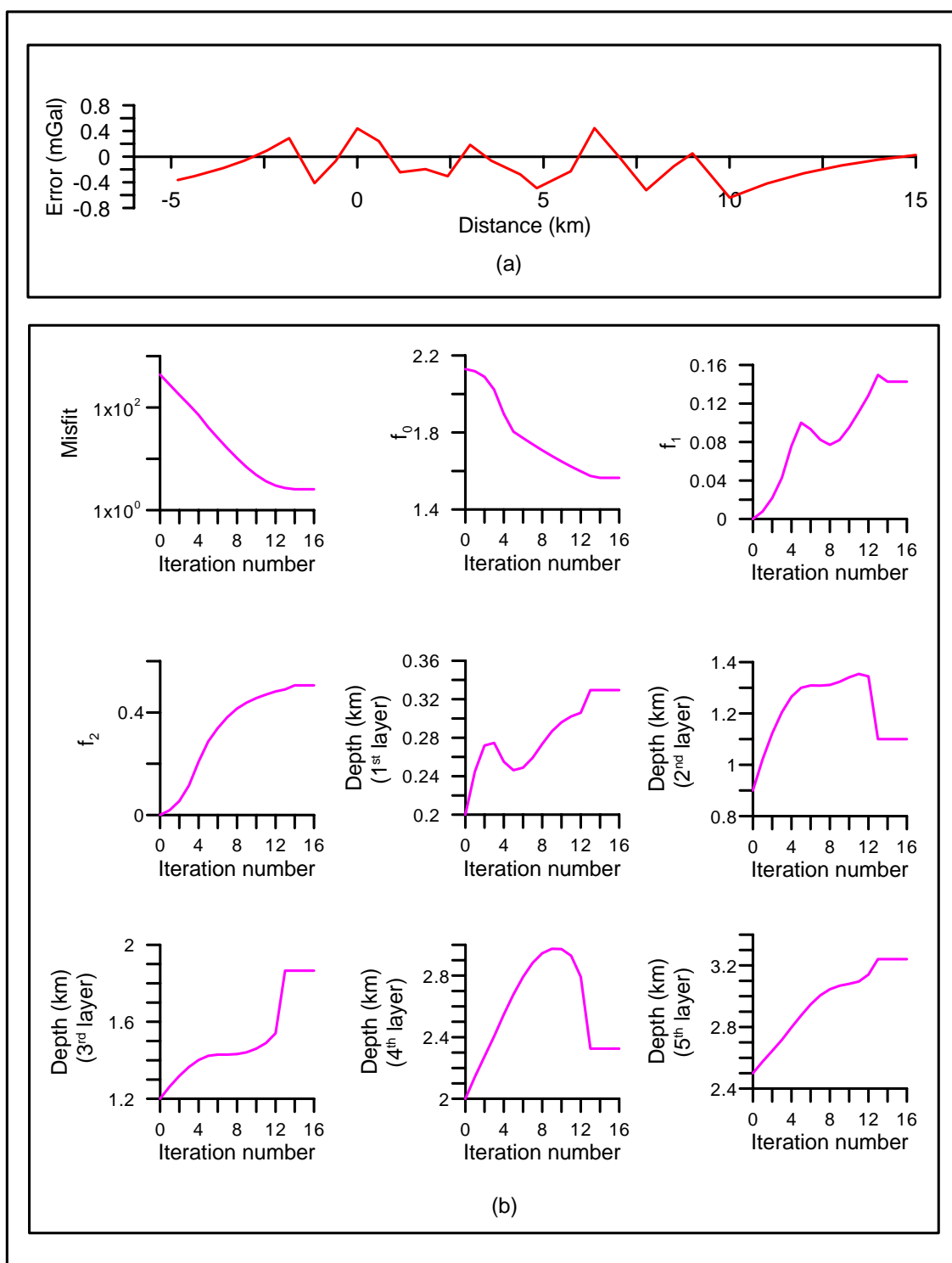


Figure 4.9 (a) Error analysis between the observed and modeled gravity anomalies across the Aswaraopet master fault morphology, Chintalpudi subbasin, India, (b) changes in misfit, coefficients of a 2<sup>nd</sup> degree polynomial, and depths to various density interfaces against the iteration number.

Furthermore, the inferred structure of the basin across the Aswaraopet fault from DSS studies (after Kaila et al. 1990) is also shown in Figure 4.8b for comparison. The theoretical gravity response of this structure is shown as a dashed line in Figure 4.8a along with the observed anomaly. It can be seen from Figure 4.8a that the modeled gravity anomalies of the structure from present inversion closely mimic the observed ones, whereas the gravity response of the seismically derived structure (Kaila et al., 1990) does not. In addition, the large gradient (4.5 mGal/km) observed in the anomaly across the fault plane does not agree well with the interpreted model of Kaila et al. (1990), whereas it agrees reasonably well with the present inversion result. Further, the error (4.6%) between the measured and estimated thickness of the basin from DSS studies near the existing deep borehole is relatively more than the one estimated (2.55%) from the present inversion.

#### **4.5 Results and Discussion**

A gravity inversion technique and related software, GRAVLIS, are developed to simultaneously estimate the geometries of fault planes and the parameters pertaining to either densities or depths of various subsurface formations of strike limited listric fault sources from the observed gravity anomalies. The advantage of the proposed algorithm is that it can be used to analyze the gravity anomalies of the structures even when the profile along which the interpretation is intended fails to bisect the fault plane.



The algorithm is applied to both synthetic and real field gravity anomalies. In case of synthetic example; significant level of pseudorandom noise was added to the gravity anomalies produced by a structure, whose fault plane was described with a 6<sup>th</sup> degree polynomial. To study the effect of the choice of the degree polynomial in the interpretation, the noisy anomalies were inverted presuming a 2<sup>nd</sup> degree polynomial for the fault plane. The noisy anomalies were then analyzed to estimate i) the densities and fault plane geometry, keeping the depths of density interfaces unchanged, and ii) depths and fault plane geometry, keeping the densities intact. In either case, the estimated parameters pertaining either to densities or depths closely mimic the assumed parameters, although random noise is present in the anomaly. However, the choice of a lower order polynomial lead to underestimate the amount extension across the normal fault, when inversion is performed to recover the fault plane geometry and depths of density interfaces.

The observed gravity anomalies across the Aswaraopet master fault from the eastern margin of the Chintalpudi subbasin in India are analyzed by the proposed inversion and found that the estimated parameters (densities and thicknesses of subsurface formations within the hanging wall) from independent gravity inversion reasonably coincide with the measured ones. On the other hand, the calculated gravity response of the structure derived from DSS investigations (Kaila et al. 1990) using the measured density-depth data was significantly deviated from the observed anomaly. Further, the large gradient in the observed gravity anomaly over the fault plane is better

explained by the present interpreted model rather than the one reported from seismic data interpretation (Kaila et al., 1990).

By and large, the modeled structure of the Aswaraopet fault morphology from gravity inversion closely resembles the one obtained from interactive gravity modeling (Chapter-III). However, given a choice, the estimated model from gravity inversion is more preferable because the model gravity anomalies in case of inversion closely replicate the observed anomalies; whereas it is not so prominent in case of interactive modeling.

## Automatic gravity modeling of sedimentary basins by means of polygonal source and exponential density contrast model \*

---

### 5.1 General

Estimation of sediment thickness and underlying bedrock topography from the observed gravity anomalies are important in hydrocarbon exploration to find out the location of possible stratigraphic traps (Silva et al., 2010), in glaciology to infer the base flow rate of discharge (Krimmel, 1970; Stern, 1978; Venteris and Miller, 1993), in landfill analysis as a tool for density determination (Mantlik et al., 2009), in hydrogeological investigations to understand the geological structures of aquifers (Adema et al., 2007; Bohidar et al., 2001), in ground motion amplification studies to study source characterization (Torizin et al. 2009; Jacoby and Smilde, 2009; Aydemir et al., 2014) etc.

In general, sedimentary rocks have densities lower than the basement rocks, in which case, negative gravity anomalies are usually observed over

---

\* *Journal of Applied Geophysics (Elsevier)*, DOI: 10.1016/j.jappgeo.2015.11.007

thick sedimentary basins. These anomalies can be analyzed not only to map the basin boundaries but also to determine the depth distribution of sedimentary basins.

## **5.2 Status of existing interpretational techniques**

Interpretation of gravity anomalies of sedimentary basin can be realized by direct and indirect methods. More often than not the terms modeling and inversion are used more or less synonymously to refer to direct methods of gravity interpretation to trace the boundaries or outlines of anomalous bodies using iterative approach, although they differ from each other in their operation principles (Murthy, 1998; Chakravarthi et al., 2013a).

In direct methods, many algorithms employ either the stacked prism model of Bott (1960) or the polygonal model of Talwani et al. (1959) to analyze the gravity anomalies of sedimentary basins. The techniques developed by Murthy et al. (1988), Pilkington and Crossley (1986), Murthy et al. (1990), Leão et al. (1996), Barbosa et al. (1997, 1999), Annecchione et al. (2001), Mendonca (2004), Gabalda et al. (2005), Silva Dias et al. (2007), Osman et al. (2007) are based on the assumption that the sedimentary load over the basement possess uniform density; hence these techniques fall short of the modeling needs for gravity anomalies in sedimentary basins, where geologic settings warrant the use of variable density. It is also true that gradual increase in density with depth necessitates the introduction of discrete bodies which

differ only in having slightly different densities that may not necessarily correspond to geologically distinct units.

In this context, Rao (1986) used a quadratic density function to analyze the gravity anomalies of sedimentary basins, whereas Pan (1989) employed constant horizontal density gradient to compute the gravity anomalies of irregularly shaped 2D bodies, Reamer and Ferguson (1989), Hansen (1999), Hamayun et al. (2009), D'Urso (2014a) adopted linear density-depth relationship, Kwok (1991) considered linear density variation with horizontal position and depth to compute the gravity anomalies using complex contour integrals, Oliva and Ravazzoli (1997) proposed a complex variables formulation for the computation of the gravity effect of 2D bodies having densities varying both laterally and with depth, Zhang et al. (2001) make use of a polynomial function of arbitrary degree expressed in terms of depth and lateral position to calculate the gravity anomalies of 2D polygonal bodies, Zhou (2008) used line integrals to calculate the gravity anomaly of a 2D mass of complicated geometry and spatially variable density contrast, and Zhou (2009) extended the line integral method to compute the gravity anomalies of irregular 2D masses with horizontally and vertically dependent density contrast. However, each enlisted method has its own merits and demerits in its application (Chakravarthi, 2009).

García-Abdeslem (2003) had developed a 2D inversion incorporating a cubic density polynomial to analyze the gravity anomalies of geologic sources, where the source-basement geometry was described by the Fourier series. In

this case the problem of gravity interpretation reduces to estimate the coefficients of the Fourier series. In recent past, Zhou (2013) has developed an automated iterative forward modeling scheme based on the line integral and maximum difference reduction methods to estimate 2D bedrock topography from a given gravity anomaly profile.

However, due to the fact that gravity anomalies of sedimentary basins cannot be realized in the space domain using EDCM; many existing algorithms perform forward modeling in the frequency domain and then transform the anomalies back to the spatial domain for further analysis. For e.g., Cordell (1973) had developed a recursive method that uses both the gravity field and its vertical derivative (determined by convolution in discrete Fourier series) to solve the structure of a sedimentary basin from the observed gravity anomalies. The method of Chai and Hinze (1988) involved the calculation of gravity anomalies in the wave number domain followed by their transformation to the space domain by a shift-sampling technique. Although, Rao et al. (1993) could derive expressions for the Fourier transforms of the gravity anomalies of a few simple geometric models with EDCM and adopt them to analyze the gravity anomalies of sedimentary basins; these strategies prone to yield unreliable interpretations when sediment-basement interface has major undulations.

On the other hand, a few researchers have accommodated exponential density variation in gravity interpretation by alternative means. For e.g., Murthy and Rao (1979), Rao and Murthy (1989) proposed subdivision of each side of a polygon into a number of segments, among each of which the density

contrast was assumed to vary linearly with depth. Guspí (1990) described the exponential density function by a series approximation to compute the gravity response. However, the former technique consumes significant amount of time even for forward modeling (Visweswara Rao et al., 1994); whereas the later one requires the knowledge of the degree of the polynomial, which is not known *a priori*. It also becomes a strenuous task to accommodate exponential density variation in the available commercial software.

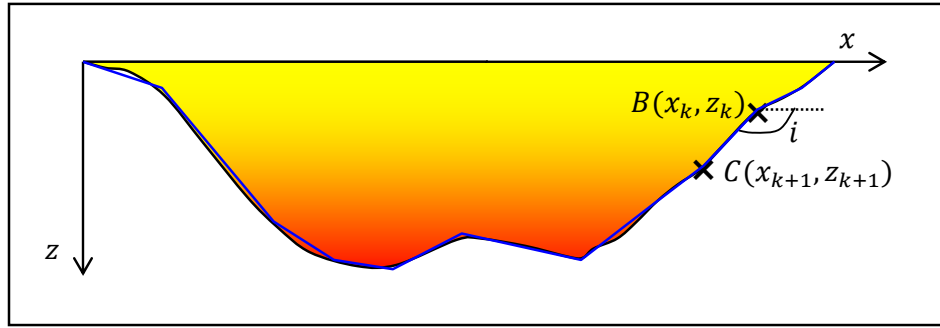
Chakravarthi et al. (2013a, 2013c) have extended the Bott's method (stacked prism model) to develop algorithms in the space domain to analyze gravity anomalies of 2.5D and 2D sedimentary basins using EDCM, where forward modeling was performed by a combination of analytical and numerical approaches. However, the Bott's method is strictly applicable only when the anomalies are available at equal station interval; elsewhere, such criteria may or may not be fulfilled. Therefore, a need exists to develop an interpretation strategy in the space domain using EDCM to analyze the gravity anomalies of sedimentary basins, which is free from the enlisted limitations.

In this Chapter, an equation is derived in the space domain to compute the gravity anomalies of 2D sedimentary basins treating the source as a polygonal geometry within which the density contrast varies in a prescribed exponential form. Based on the principles of automatic modeling (Chakravarthi et al., 2015c) an interpretation strategy coupled with relevant software in JAVA is developed to analyze the gravity anomalies of sedimentary basins. The applicability of proposed modeling is demonstrated on both synthetic and real

world gravity anomalies. The estimated depth parameters from automatic modeling are compared with the assumed parameters in case of synthetic example and with the available/previously reported information in case of real field example.

### 5.3 Forward gravity modeling – Theoretical considerations

In Cartesian co-ordinate system let the  $z$ -axis is positive vertically downwards and the  $x$ -axis transverse to the strike of a 2D sedimentary basin (Figure 5.1). The strike of the basin is along the  $y$ -axis perpendicular to the  $xz$  plane.



*Figure 5.1 Schematic representations of a sedimentary basin (black solid line) and its approximation by a polygon (blue solid line). B and C are the vertices of the  $K^{th}$  side of the polygon. The colour gradation from yellow to red within the structure represents the increase in density with depth*

The gravity anomaly of such a sedimentary basin at any point,  $P(X_j, Z_j)$ , on a profile outside the source region can be obtained by integrating the gravity effect of a 2D element throughout its cross-sectional area as expressed by Chakravarthi et al. (2015c)

$$\Delta g_{2D}(X_j, Z_j) = 2G \int_s \frac{\Delta \rho(z)(z - Z_j)}{(x - X_j)^2 + (z - Z_j)^2} dx dz, \quad (5.1)$$



where,  $G$  is universal gravitational constant,  $(x, z)$  are source coordinates of a 2D element within the source and  $dx dz$  represents its cross-sectional area. Here  $\Delta\rho(z)$  is the density contrast of the sediments at any depth  $z$  represented by equation (1.14).

Upon substitution equation (1.14) for  $\Delta\rho(z)$  and applying Stoke's theorem, equation (5.1) becomes

$$\Delta g_{2D}(x_j, z_j) = 2G\Delta\rho_0 \oint_z e^{-\lambda z} \tan^{-1} \frac{\overline{x - X_j}}{\overline{z - Z_j}} dz. \quad (5.2)$$

Approximating the outline of the basin by a polygon BC....(Figure 5.1), the  $x$  term in the integrand of equation (5.2) can be expressed for the  $k^{\text{th}}$  side, such as BC as

$$x = a + z \cot i, \quad (5.3)$$

where,  $a = x_k - z_k \cot i$ . Here,  $i$  is the angle made by  $k^{\text{th}}$  side with the  $x$ -axis. Furthermore,  $(x_k, z_k)$  are the coordinates of the vertex B. The gravity effect of the  $k^{\text{th}}$  side of the polygon is finally expressed as

$$\Delta g_{BC}(X_j, Z_j) = 2G\Delta\rho_0 \int_{z_k}^{z_{k+1}} e^{-\lambda z} \tan^{-1} \frac{\overline{(a + z \cot i) - X_j}}{\overline{z - Z_j}} dz. \quad (5.4)$$

Here,  $z_{k+1}$  is the depth ordinate of the vertex C, whose coordinate is represented by  $x_{k+1}$ . The total gravity effect of the polygon can be obtained as (Chakravarthi et al., 2015c)

$$\Delta g_{2D}(X_j, Z_j) = \sum_{k=1}^N \Delta g_k(X_j, Z_j), \quad (5.5)$$

where,  $N$  is the number of sides of the polygon. It is to be noted that equation (5.4) needs to be solved numerically because no closed form solution exists for it in the space domain. Further, equation (5.4) and (5.5) are strictly valid for the profile runs transverse to the strike of the basin. In case it runs at an angle,  $\alpha$ , with the  $x$ -axis, then the anomalous field can be computed by replacing  $X_j$  in equation (5.4) by  $X_j \cos \alpha$  (Chakravarthi and Ramamma, 2013b). Further, the gravity anomaly of a structure with uniform density can be realized by letting  $\lambda=0$  in equation (5.4) followed by using equation (5.5). The accuracy of the proposed numerical method of anomaly calculation is demonstrated with a few selected examples in the following sections.

### ***5.3.1 Forward modeling of a homogeneous vertical prism***

Figure 5.2a shows the gravity response of a homogeneous vertical prism (Figure 5.2b) at 21 equispaced observations in the interval  $X_j \in [0 \text{ km}, 20 \text{ km}]$  obtained from both the numerical solution of equation (5.4) and (5.5), and the analytic solution of the prism by Murthy (1998). In this case, the prism is considered to have 5 km width, 4 km depth extent and possess uniform density contrast of  $-0.45 \text{ gm/cm}^3$ . In order to compute the gravity anomalies of the prism by the proposed numerical method, the structure is simulated with a polygon whose vertices are given by P(7.5,0), Q(12.5,0), R(12.5,4) and S(7.5,4). The solid line in red shown in Figure 5.2a represents the anomalies

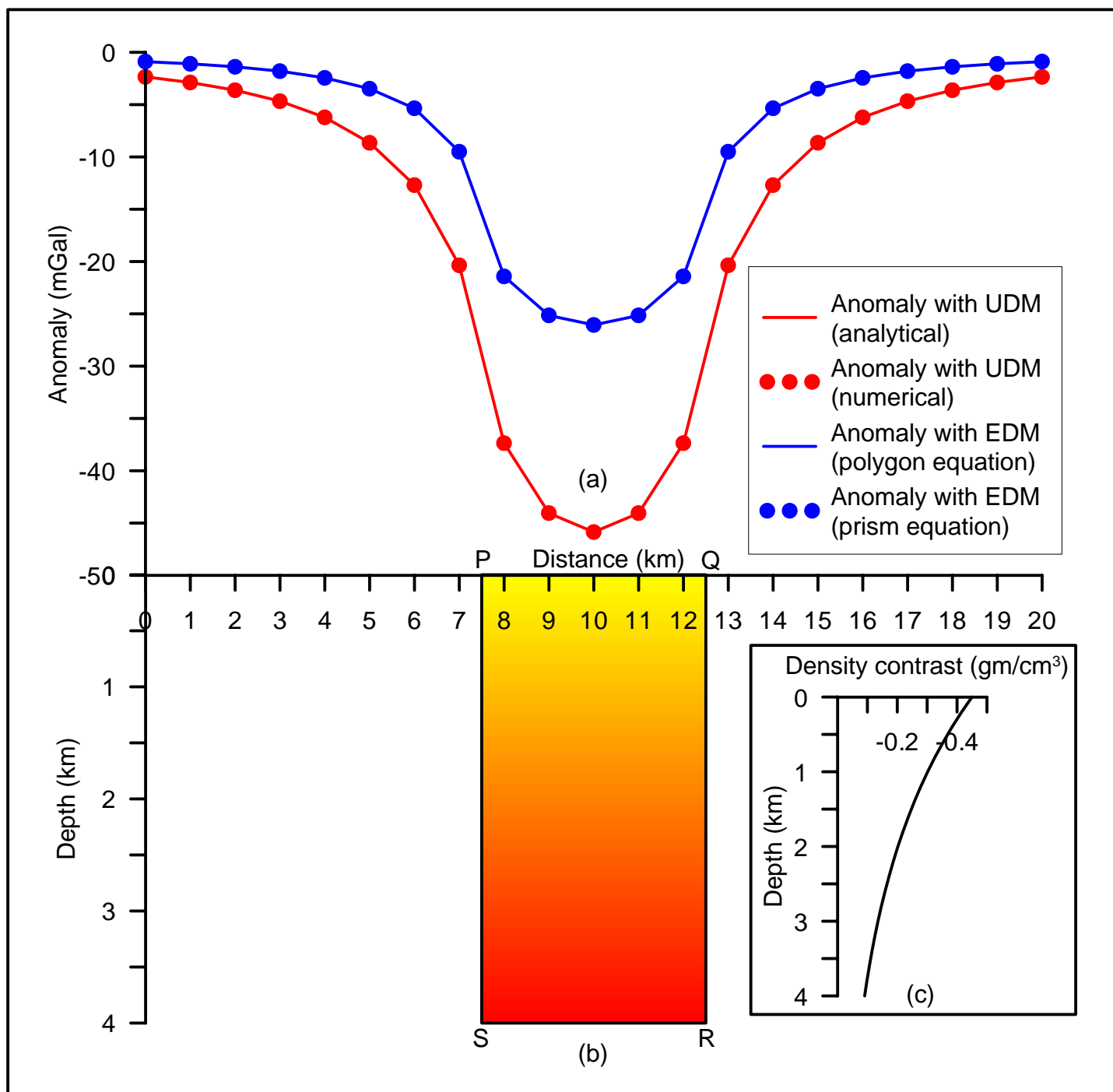


Figure 5.2 (a) Gravity anomalies with uniform and exponential density contrast models, (b) geometry of assumed structure, (c) prescribed EDM. The color gradation from yellow to red within the prism indicates the increase in density in case of EDM

obtained from the analytical method and the filled circles in red correspond to the anomalies realized from the numerical method. In this case, the maximum difference between the two anomalies hardly exceeds  $5\text{E-}05$  mGal, which shows the accuracy of the proposed method.

### ***5.3.2 Forward modeling of a vertical prism with EDCM***

The gravity anomalies of the same structure (Figure 5.2b) at 21 observations in the interval  $X_j \in [0 \text{ km}, 20 \text{ km}]$  with exponential decay in density contrast obtained from both the present method and the one from numerical solution of the prism (Chakravarthi et al., 2013c) are also shown in Figure 5.2a. In both cases the values of  $\Delta\rho_0$  and  $\lambda$  are assumed as  $-0.45 \text{ gm/cm}^3$  and  $0.4 \text{ km}^{-1}$  respectively (Figure 5.2c). In this case, the maximum difference between the magnitudes of the two anomalies is  $4\text{E-}05$ , which again is insignificant.

### ***5.3.3 Forward modeling of fault and trapezoidal geometries with EDCM***

Solid lines in blue in Figures 5.3a and 5.3c represent the gravity anomalies calculated by Rao et al. (1993) over a vertical fault and trapezoidal geometries (Figures 5.3b and 5.3d) using a quadratic density model (QDM). The authors have used a set of quadratic density functions, one defined with  $-0.4957 + 0.2257z - 0.0344z^2$  and the other by  $-0.3654 + 0.1140z - 0.0098z^2$  (solid lines in blue in Figures 5.3e and 5.3f) to simulate a prescribed exponential density contrast model (EDCM)  $-0.5e^{-0.5z}$  (solid lines in red in Figures 5.3e and 5.3f) over two selected depth ranges 0 km to 2 km, and 2 km to 4 km

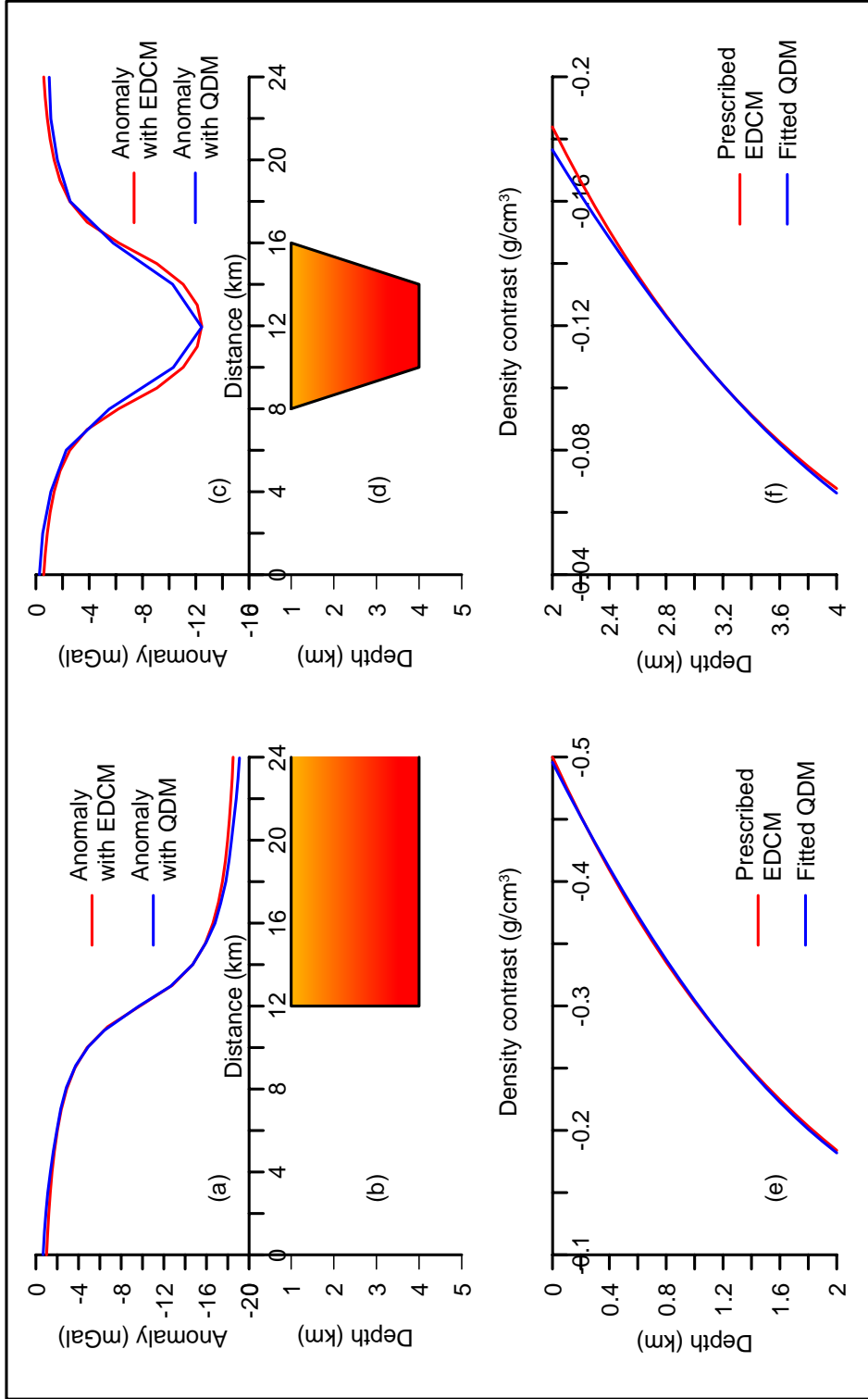


Figure 5.3 Forward gravity modeling based on analytical and numerical approaches (a) and (c) over a vertical fault (b) and trapezoidal (d) models, prescribed exponential density contrast model (EDCM) and fitted quadratic density model (QDM) over two selected depth ranges are shown in (e) and (f). The color gradation from yellow to red within the structures indicates increase in density with depth

respectively. These fitted quadratic density models are then used in respective analytical gravity expressions of the geometries (Rao, 1985; Rao, 1990) to generate the gravity anomalies in the interval  $X_j \in [0 \text{ km}, 24 \text{ km}]$  in each case as shown in Figures 5.3a and 5.3c. The gravity responses of the two models obtained from the present numerical method using the prescribed EDCM are also shown in Figures 5.3a and 5.3c by solid lines in red.

One can notice from Figure 5.3a and Figure 5.3c that the gravity anomalies obtained from the present method with prescribed EDCM closely coincide with the anomalies obtained with the fitted QDMs by Rao et al. (1993). The marginal deviations noticed between these two anomalies in each case are obvious because the fitted QDM shows moderate deviations from the prescribed EDCM, more particularly so beyond a depth of 2 km (Figures 5.3e and 5.3f). Furthermore, the advantage of the present method is that a single equation is suffice to calculate the gravity anomalies of the two structures (geometries can be simulated with appropriate vertices), whereas the strategies proposed by Rao (1985, 1990) involved the use of different anomaly equations for different geometries.

#### ***5.3.4 Forward modeling of a homogeneous sedimentary basin***

The filled circles and solid line in red shown in Figure 5.4a correspond to the calculated gravity anomalies over a hypothetical homogeneous sedimentary basin (Figure 5.4b) obtained from the proposed numerical method and the analytical method of Murthy and Rao (1989). In this case, the density

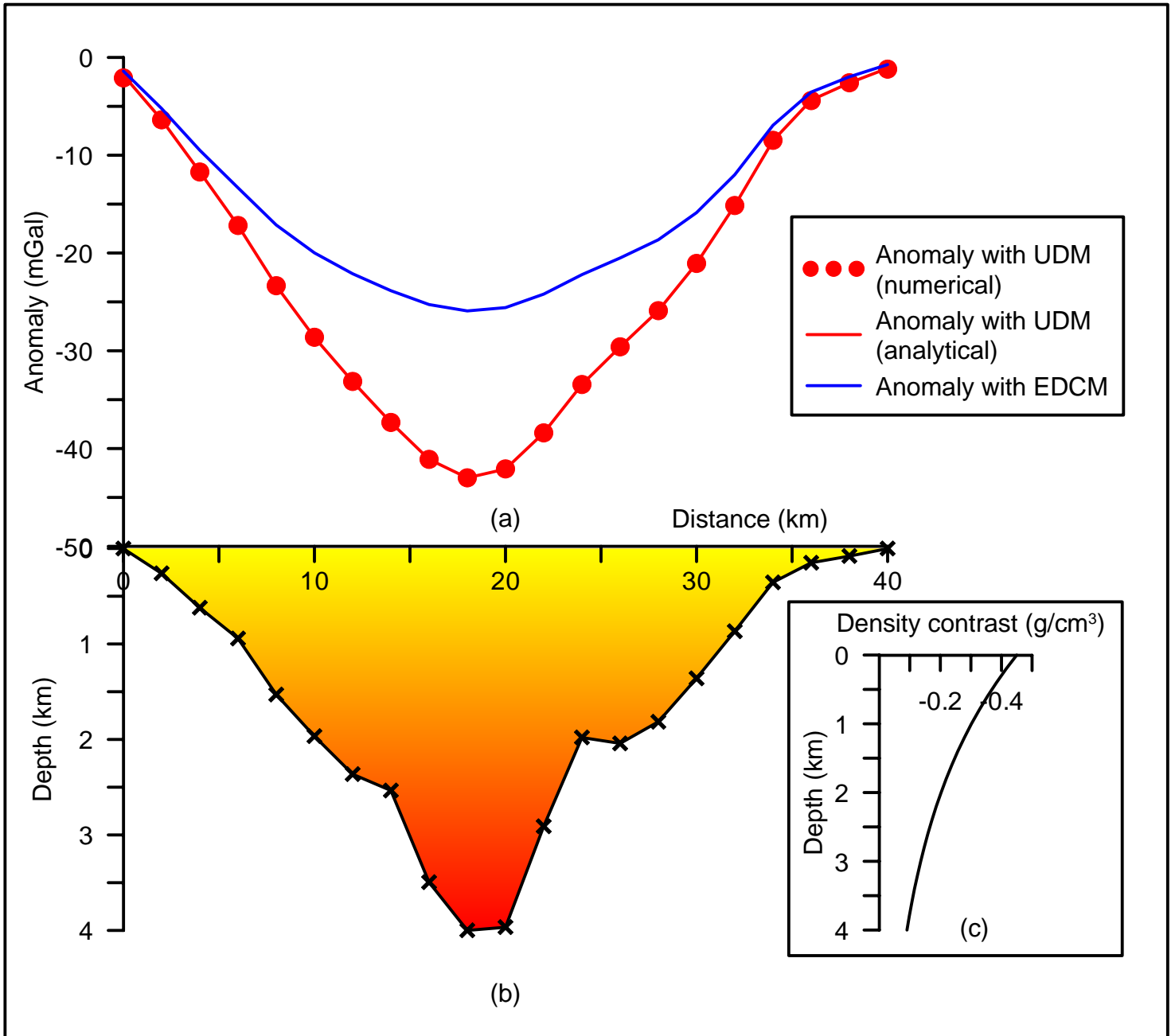


Figure 5.4 (a) Forward gravity modeling with analytical and numerical approaches based on uniform and exponential density contrast models (UDM & EDCM), (b) geometry of a sedimentary basin, (c) prescribed EDCM. The color gradation from yellow to red within the structure indicates increase in density with depth in case of EDCM.

interface (floor of the sedimentary basin) is described with a polygon having 21 vertices as illustrated in Figure 5.4b. In each case, the anomalies are calculated on a plain topography,  $Z_j = 0$ , at 21 observations in the interval  $X_j \in [0 \text{ km}, 40 \text{ km}]$  at 2 km spacing using an uniform density contrast of  $-0.35 \text{ gm/cm}^3$  (Figure 5.4a). One can notice from Figure 5.4a that the anomalies obtained from the present method excellently coincide with those realized from the analytical method of Murthy and Rao (1989). The solid line in blue in Figure 5.4a represents the gravity anomalies produced by the same structure but with exponential decay in density contrast defined with the constants  $\Delta\rho_0 = -0.35 \text{ gm/cm}^3$  and  $\lambda = 0.35 \text{ km}^{-1}$ . The maximum anomaly (absolute magnitude) produced by the basin with uniform density model (UDM) is 43 mGal, whereas it hardly exceeds 26 mGal in case of exponential density contrast model (EDCM) (Figure 5.4a). Thus, sedimentary basins in which the density contrast varies exponentially with depth would generate only moderate gravity anomalies; and hence the interpretation algorithms that are dependent on the magnitude of the anomalous field must consider the exponential density contrast model as a crucial parameter in the analysis for reliable results.

#### **5.4 Automatic modeling of gravity anomalies**

The aim of gravity modeling of a sedimentary basin is to estimate the depths to the floor of the basin from the observed gravity anomalies at plurality of observations. The proposed algorithm initializes a sedimentary basin from the observed gravity anomalies and subsequently improves the structure based



on the differences between the observed and modeled gravity anomalies within the specified convergence criteria. In the present case, the number of vertices of the polygon equals the number of observations on the profile and distance to each observation on the profile becomes the  $x$  coordinate of the corresponding vertex of the polygon. Hence, the problem of gravity modeling simplifies to estimate the optimum depth co-ordinates of the vertices of the polygon.

The initial/approximate depths to the floor of a sedimentary basin are calculated at all observations on the profile by substituting the observed residual gravity anomaly,  $g_{obs}(X_j, Z_j)$ , for,  $g_{obsmx}$  in equation (2.4). Having calculated the initial depths, the modeled gravity anomalies,  $\Delta g_{2D}(X_j, Z_j)$ , of a sedimentary basin can be realized through equation (5.4) and (5.5). The difference between the observed and modeled gravity anomalies at any observation can be quantified using equation (2.5).

The depths of a sedimentary basin are then automatically improved by using the principles of automatic modeling (Chakravarthi et al., 2013a; 2015c). Accordingly, the basin depth at any observation,  $(X_j, Z_j)$ , can be improved based on the expression

$$Z_B(X_j, Z_j)^{k+1} = Z_B(X_j, Z_j)^k + \frac{ERR(X_j, Z_j)}{2\pi G \Delta \rho (Z_B(X_j, Z_j)^k)}, \quad (5.6)$$

where,  $k$  stands for the number of iterations. Here,

$$ERR(X_j, Z_j) = g_{obs}(X_j, Z_j) - \Delta g_{2D}(X_j, Z_j), \quad (5.7)$$

The interpretation process continues until i) the number of iterations completed, or ii) the value of r.m.s. error falls below a predefined allowable error, or iii) the current/existing r.m.s. error exceeds the preceding value.

### **5.5 Description of software – MOD2DGREXP**

The modeling procedure described in section 5.4 is used to develop a GUI based software, MOD2DGREXP, coded in JAVA to interpret the gravity anomalies of 2D sedimentary basins using EDCM (Annexure 5-A). The architecture of the software is based on the MVC (Model-View-Controller) pattern shown in Figure 2.2. The module ‘Model’ calculates the initial/approximate depths, computes the gravity response of the structure, and executes the business logic of the algorithm to improve the model space. The role of ‘View’ module is to read the input parameters and display the output in both graphical and ASCII formats. The ‘Controller’ passes the required actions to the view and model modules.

The view module of the code appears on the monitor as shown in Figure 5.5 after invoking the batch file. The input layout of the view module consists in ten input fields namely the area name, profile name, number of observations, distance to each observation expressed in km, observed gravity anomalies in mGal, surface density contrast in  $\text{gm/cm}^3$ , Lambda in  $\text{km}^{-1}$ , minimum and maximum permissible depths expressed in km and the number of iterations to be performed. The graphical layout is arranged into anomaly, structural, misfit

and density-depth panels respectively as depicted in Figure 5.5. The ASCII form of the output shall be displayed in the ASCII layout.

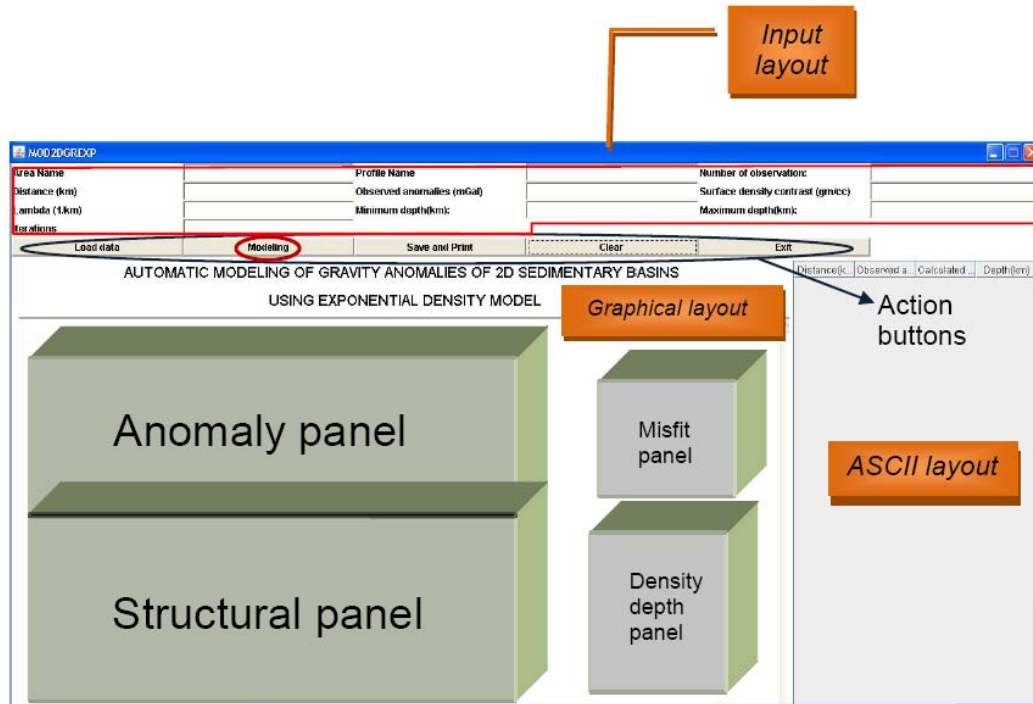


Figure 5.5 View module of MOD2DGREXP

The user enters the input parameters either by using the respective fields of the input layout or specifies the parameters in a formatted Excel sheet and reads the file to the code by means of 'Load data' action button. The interpreter invokes the action button 'Modeling' to perform the interpretation.

The growth in the model space shall be displayed in an animated form along with the improvements in modeled anomalies against the iteration number. The changes in misfit and density contrast shall also be displayed in animated forms in respective panels. The user saves the output in both html and jpg formats and opts for printing by invoking 'Save and print' action button.

## 5.6 Applications

The applicability of proposed method of automatic modeling is demonstrated on both synthetic and real world gravity anomalies. The real field gravity data pertains to the San Jacinto graben, California (after Cordell, 1973).

### 5.6.1 Synthetic example

Figure 5.6a shows, in the interval  $X_j \in [0 \text{ km}, 60 \text{ km}]$ , 25 randomly distributed noisy gravity anomalies (solid circles) produced by a synthetic model of sedimentary basin, whose geometry is shown in Figure 5.6b. The assumed structure represents a typical rifted sedimentary basin, whose basement consists in numerous discontinuities caused by several synthetic and antithetic faults. Furthermore, these faults observed in the basement in the interval  $X_j \in [15 \text{ km}, 42 \text{ km}]$  have minor throws and show relatively complex pattern compared to the marginal faults of the basin. In this case, pseudorandom noise was Gaussian with zero mean and a standard deviation of 0.11 mGal. The density contrast within structure obeys exponential decrease with depth following equation (1.14) defined by the constants  $\Delta\rho_0 = -0.45 \text{ gm/cm}^3$  and  $\lambda = 0.4 \text{ km}^{-1}$ , respectively (Figure 5.6c).

The noisy anomalies shown in Figure 5.6a are analyzed by the present method to recover the basement structure. The constants of the prescribed EDCM remain unchanged during the analysis, whereas the depths of the basin are allowed to vary within the specified bounds as specified in section 5.5. The modeling algorithm has performed 14 iterations before it got terminated. The

## Automatic modeling - Synthetic example

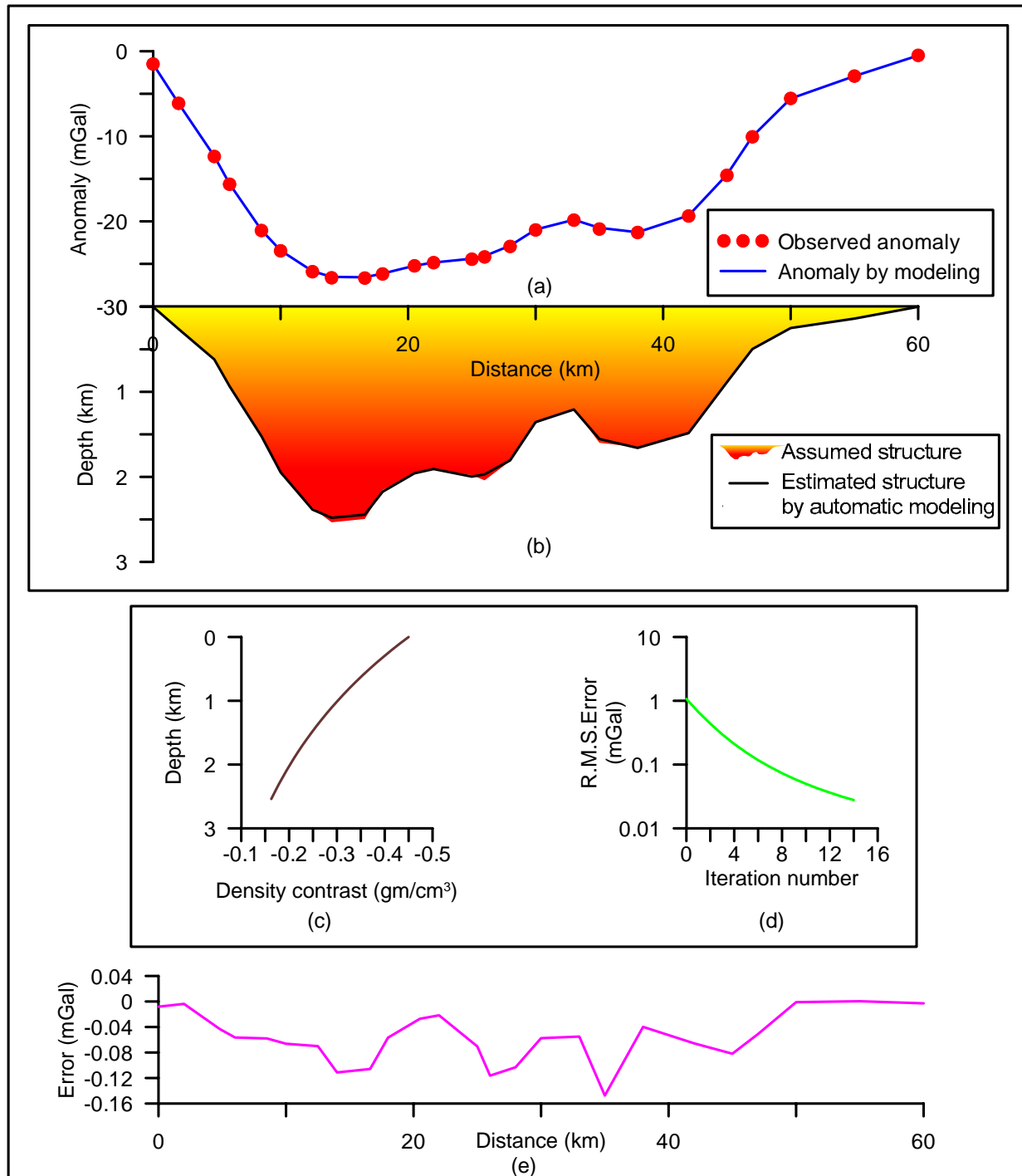


Figure 5.6 (a) Observed and modeled noisy gravity anomalies by automatic modeling, (b) assumed and estimated structures, (c) prescribed EDCM, (d) changes in misfit with iteration, (e) error between observed and modeled anomalies at the end of 14<sup>th</sup> iteration. The color gradation from yellow to red within the structure indicates increase in density

misfit (equation 2.5) had reduced from 1.08 mGal to 0.02 mGal at the end of the 14<sup>th</sup> iteration (Figure 5.6d), beyond which the new misfit got exceeded its previous value thereby the modeling process ended. The model gravity anomalies and the observed anomalies show good correlation with each other at the end of the concluding iteration (Figure 5.6a), with a maximum error (absolute) between them ( $\sim 0.15$  mGal) observed at the 35<sup>th</sup> km. By and large, the algorithm had recovered the structure successfully over the length of the profile, however, with a few exceptions. For e.g. the inferred structure had shown low to moderate deviations from the assumed one in and around the depocentre of the basin in the interval  $X_j \in [12 \text{ km}, 16 \text{ km}]$ . Such deviations are tolerable considering the presence of significant level of random noise in the observed anomaly.

A sample output generated by the code in html format is shown in Annexure 5-B.

### 5.6.2 Field Example

The solid dots shown in the interval  $X_j \in [0 \text{ km}, 9.982 \text{ km}]$  in Figure 5.7a represent the observed gravity anomalies of the San Jacinto graben, California (after Cordell, 1973). The density contrast-depth data of the graben derived from seismic refraction surveys by Fett (Unpublished thesis, University of California at Riverside, 1968) as reported by Cordell (1973) was shown as a step line in green in Figure 5.7c. Cordell (1973) had used two exponential density contrast models (EDCMs); one defined with  $\Delta\rho_0 = -0.55 \text{ gm/cm}^3$  and

## Automatic modeling - Field example

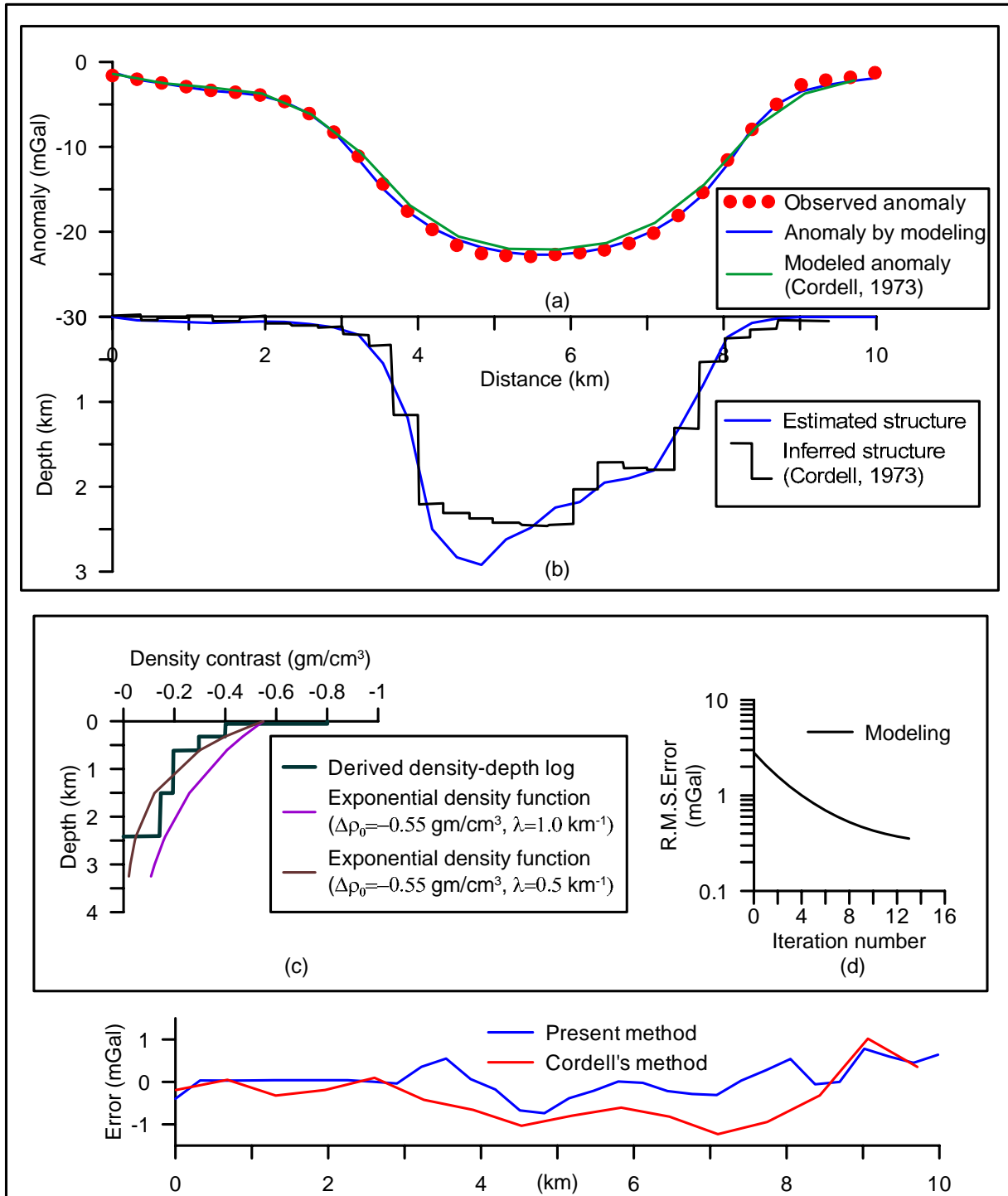


Figure 5.7(a) Observed and theoretical gravity anomalies by automatic modeling, San Jacinto graben, California. Modeled anomalies by Cordell (1973) are also shown for comparison, (b) estimated structures by the present method and Cordell's (1973), (c) derived density contrast-depth data and fitted EDCMs, (d) changes in r.m.s. error with iteration, (e) comparison of errors in anomaly observed in case of the present and Cordell's (1973) methods

$\lambda = 0.5 \text{ km}^{-1}$  (shown as a solid line in purple in Figure 5.7c) and the other with  $\Delta\rho_0 = -0.55 \text{ gm/cm}^3$  and  $\lambda = 1.0 \text{ km}^{-1}$  (shown as a solid line in brown in Figure 5.7c) to describe the derived density contrast-depth data of the graben. He had shown that the use of former density model in the gravity interpretation had yielded a structural solution (shown as a step line in Figure 5.7b) that was consistent with seismically derived information.

In the present case, we have analyzed the observed gravity data using the proposed automatic modeling to decipher the basement configuration of the graben with an aim to examine whether the inferred model is consistent with the one reported by Cordell (1973). The prescribed EDCM defined with  $\Delta\rho_0 = -0.55 \text{ gm/cm}^3$  and  $\lambda = 0.5 \text{ km}^{-1}$  was used in the present analysis to describe the density contrast variation with depth. For such an interpretation, the algorithm had performed 13 iterations. Figure 5.7d shows the changes in r.m.s. error with the iteration number. The modeled gravity anomalies at the end of the concluding iteration are shown in Figure 5.7a and the estimated basement configuration of the graben in Figure 5.7b respectively. The theoretical gravity anomalies and corresponding inferred structure realized by Cordell (1973) were also shown in Figures 5.7a and 5.7b for comparison. The maximum depth to the basement estimated from the present method is 2.9 km against a figure of 2.4 km reported by Cordell (1973).



## 5.7 Results and Discussion

A space domain based algorithm using the principles of automatic modeling is developed along with a GUI based software to analyze the gravity anomalies of sedimentary basins among which the density contrast obeys exponential decrease with depth. The cross-section of a sedimentary basin is viewed as a polygon, whose gravity anomaly equation is derived in the space domain using a combination of analytical and numerical approaches with exponential density contrast model. The proposed algorithm is fully automatic in the sense that it generates the initial structure of a sedimentary basin from the observed gravity anomalies at plurality of observations and improves the structure in an iterative approach based on the differences between the observed and modeled gravity anomalies until the modeled anomalies closely mimic the observed ones. A software named, MOD2DGREXP, coded in JAVA with inbuilt GUI is also developed, which is platform independent. Besides being user friendly the software is easy to operate.

To demonstrate the applicability of the algorithm the gravity anomalies attributed to a synthetic model are analyzed in the presence of pseudorandom noise. It was found from the analysis that, by and large, the algorithm had recovered the structure successfully including the minor faults exist in the basement.

In case of field example, the derived structure of the San Jacinto graben, California from the proposed method was compared with the previously

reported model by Cordell (1973). The deciphered structural models from the present analysis and the one reported by Cordell (1973) showed more or less similar morphological features. For e.g., these two models had revealed that the graben is bounded by steeply dipping fault system towards the west and by moderately dipping fault system towards the east. However, unlike the case with the Cordell's (1973) interpretation the present model shows progressive deepening of the basement towards the west (Figure 5.7b). Furthermore, the basement high which was reflected prominently in the Cordell's (1973) model in the interval  $X_j \in [6 \text{ km}, 7 \text{ km}]$  was not repeated in the present inferred model. It is pertinent to mention here that the modeled gravity anomalies from the present analysis almost coincide with the observed anomalies; where as it is not so in case of Cordell's (1973) interpretation (Figure 5.7a). One can also notice from Figure 5.7e that the magnitude of error between the observed and modeled gravity anomalies is more significant in case of Cordell's (1973) interpretation when compared to the present one. The above factors suggest that the basement configuration derived from the present analysis appears be more reliable than the structural solution proposed by Cordell (1973).

Thus the validity and applicability of the proposed technique is justified.

## Automatic gravity inversion of sedimentary basins by means of growing polygonal source and exponential density contrast model \*

---

### 6.1 General

Deciphering concealed basement structures under sedimentary load from the measured gravity anomalies is one of the classic applications of the gravity method. Automatic modeling schemes, such as the one described in Chapter-V, are preferred to analyze the gravity anomalies of open-ended bodies only; whereas, the inversion strategies are used to analyze the anomalies of both open ended and closed bodies (Murthy, 1998; Chakravarthi et al., 2013a). Further, in case of automatic modeling the error between the observed and modeled gravity anomalies at any observation is used to improve only one model parameter (such as depth to the density interface) at respective station, while in inversion the errors at all observations are being used to calculate the improvements in all model parameters. Automatic modeling schemes become handicapped in case the

---

\* *Journal of Applied Geophysics (Elsevier)*, DOI: 10.1016/j.jappgeo.2015.11.007

residual gravity anomaly of an anomalous source is associated with regional gravity component. On the other hand, inversion algorithms can be designed to simultaneously estimate both the parameters defining the anomalous source and regional gravity component (Chakravarthi et al., 2015c).

## **6.2 Status of existing interpretational techniques**

Although, forward modeling schemes to compute the gravity anomalies of a 2D object with uniform density are available (see for eg., Okabe, 1979; Won and Bavis, 1987; Singh, 2002; D’Urso, 2014b), the practical utility of these schemes to analyze the gravity anomalies of sedimentary basins is rather limited because 1) the density of sedimentary rocks varies with depth as detailed in Chapter-I, and 2) the depth to the floor of a sedimentary basin is not known *a priori*. Bhaskara Rao and Venkateswarulu (1974) have proposed a method of gravity interpretation over sedimentary basins; where the gravity anomalies are interpreted treating the basin as a pair of outcropping faults, one each on either margin of the basin dipping inwards. Such a methodology should be adopted with caution because the gravity anomaly of each fault shall influence of the gravity response of the other.

Murthy and Rao (1989) have designed an inversion algorithm to trace the topography of a 2D density interface from the observed gravity anomalies, where the structure was represented by a stack of vertical prisms over a specified mean depth. Based on the inverse theory of Backus and Gilbert (1967, 1968, 1970),

Mickus and Peeples (1992) have developed a technique to analyze the gravity data for the lower surface of a sedimentary basin. Inoue et al. (2012) have proposed an inversion strategy for gravity anomalies of a 2D polygonal source with application to sedimentary basins using simulated annealing. In recent past, Rao (2013) has developed a technique based on the Marquardt's (1963) algorithm to process the anomalies of 2D bodies that are bounded by planar surfaces, where the vertical and horizontal components of gravitational attractions were worked out from the gravitational components in the directions perpendicular and parallel to each surface. Again all the above-mentioned techniques presume uniform density for the sedimentary rocks, which is seldom valid in reality. The technique of Grandis and Dahrin (2014) incorporates the 2D algorithm of Talwani et al. (1959) for forward modeling and the axes of the anomalous mass concentration as constraints in inversion.

In this Chapter, a new interpretation technique is developed based on the principles of inversion (described in section 2.4 of Chapter-II) to analyze the gravity anomalies of sedimentary basins using a prescribed exponential density contrast model (EDCM) followed by the development of relevant software. The proposed approach was then applied to recover the structure from the gravity data produced by a simulated graben defined by steeply dipping boundary faults. The method's ability was also demonstrated on a gravity profile across the San Jacinto graben in California.

### 6.3 Inversion

The process of inversion of gravity anomalies of sedimentary basins is analogous to a mathematical procedure of finding the depths of a sedimentary basin from the observed gravity anomalies using a prescribed exponential density contrast model (Chakravarthi et al., 2015c).

The inversion process begins with the initialization of a sedimentary basin (based on the Bouguer slab approximation) followed by the calculation of its gravity response as described in section 5.4 of Chapter-V. Because the initial depths are only ballpark, the model gravity anomalies realized through equation (5.4) and (5.5) clearly depart from the observed anomalies. The difference between the two anomalies at any observation,  $(X_j, Z_j)$ , can be expressed as a sum of the products of vertical gradient of anomaly and improvements in depth coordinates of the vertices of the polygon (Chakravarthi et al., 2015c) as

$$g_{obs}(X_j, Z_j) - \Delta g_{2D}(X_j, Z_j) = \sum_{k=1}^N \frac{\partial \Delta g_{2D}(X_j, Z_j)}{\partial z_k} dz_k. \quad (6.1)$$

Linear equation similar to (6.1) is constructed for each observation and  $N$  normal equations are framed and solved for the improvements in  $N$  depth ordinates of the polygon by minimizing the r.m.s. error defined by equation (2.5) using the principles of inversion as described in Chapter-II. The system of normal equations is finally expressed as (Chakravarthi et al., 2015c)

$$\begin{aligned}
& \sum_j^N \sum_{m=1}^N \frac{\partial \Delta g_{2D}(X_j, Z_j)}{\partial a_{j'}} \frac{\partial \Delta g_{2D}(X_j, Z_j)}{\partial a_m} (1 + \tau_{mj} \delta) da_m \\
& = \sum_{j=1}^N Err(X_j, Z_j) \frac{\partial \Delta g_{2D}(X_j, Z_j)}{\partial a_{j'}}, j' = 1, 2, \dots, N.
\end{aligned} \tag{6.2}$$

The partial derivatives required in equation (6.2) are evaluated numerically, which involves the calculation of the rate of change of the gravity anomaly with respect to each ordinate of the vertex of the polygon as described by Chakravarthi et al. (2015c). The improvements in depth ordinates of the polygon,  $da_m (= dz_m)$ ,  $m = 1, 2, \dots, N$  obtained from the solution of equation (6.2) are used to update the model space in an iterative approach until one of the termination criteria of the inversion is fulfilled as described in Chapter-II.

#### 6.4 Description of software – IN2DGREXP

Based on the proposed methodology of inversion enumerated in section 6.3, a GUI based software, IN2DGREXP, coded in JAVA is developed to analyze the gravity anomalies produced by a sedimentary basin using a prescribed EDCM (Annexure 6-A). The software is fully automatic as it generates the initial structure and improves it iteratively within the specified convergence criteria without any manual intervention. The software works on the MVC pattern as shown in Figure 2.2. The module ‘Model’ performs the task of finding the initial depths of a sedimentary basin and computes the gravity response, frames and solves the

normal equations for the improvements in the depth ordinates, and updates the depth parameters and model anomalies. The roles of ‘View’ and ‘Controller’ modules are described in section 5.5 of Chapter-V.

Once the batch file of the software is call upon, the view module appears on the monitor of the computing machine as shown below

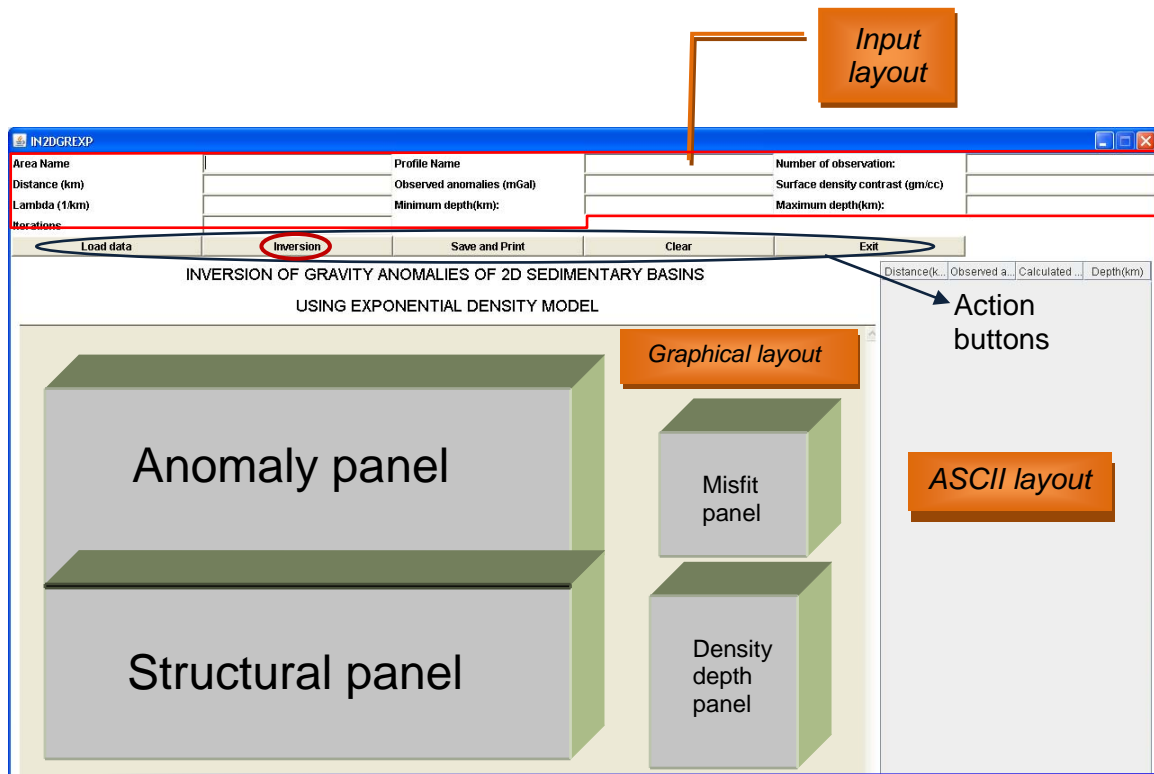


Figure 6.1 View module of IN2DGREXP

As in the case of MOD2DGREXP, the user may enter the input parameters to the code in two ways, either by using the fields of the input layout or by means of a formatted Microsoft Excel sheet. In case the input is stored in an Excel sheet, the same shall be read into the code by invoking ‘Load data’ action button (Figure 6.1). The details of the input parameters are described in section 5.5 of Chapter-V.



Upon activation, the action button ‘Inversion’ (Figure 6.1) performs the task of analyzing the gravity anomalies for the basement configuration of a sedimentary basin using the principles of inversion. The anomaly and structural panels of the graphical layout display the animated versions of the model improvement and corresponding changes in model anomaly. The changes in EDCM and the misfit shall also be displayed in animated forms in respective panels during the process of inversion.

## **6.5 Applications**

Interpretation of two gravity anomalies, one over a synthetic geometry of a sedimentary basin and the other over a real-world case, demonstrates the practical applicability of the proposed inversion.

### **6.5.1 Synthetic example**

The efficacy of the inversion technique and the software are exemplified with the noisy gravity anomalies, which were analyzed previously in section 5.6.1 by automatic modeling. The objective of choosing the same anomaly is to ascertain whether or not the estimated structure from the inversion is comparable to the one obtained from automatic modeling and also with the assumed structure. In this case, the inversion took only 7 iterations for proper a convergence of theoretical anomalies with the observed ones against 14 iterations in case of automatic modeling. The r.m.s. error, which attained a value of 1.08 mGal for the

initial structure was reduced drastically to a value less than the predefined allowable error at the end of the 7<sup>th</sup> iteration (Figure 6.2c), hence, the algorithm got terminated.

The gravity response of the estimated structure after the inversion is shown in Figure 6.2a along with the observed noisy gravity anomaly. The fit between these two anomalies is agreeable. The inferred structure of the basin for which a minimum misfit obtained was shown in Figure 6.2b together with the assumed structure. It was noticed that the estimated depths of the basin and the fit between the observed and modeled gravity anomalies do not exhibit appreciable changes subsequent to 7<sup>th</sup> iteration, being the concluding one. The residuals, defined as the observed minus the modeled gravity anomalies, after the inversion are shown graphically in Figure 6.2d. A maximum error (absolute magnitude) of 0.04 mGal between the two anomalies is observed at the 16<sup>th</sup> km on the profile. Minor deviations in the estimated depths from the true ones particularly in the interval  $X_j \in [12 \text{ km}, 16 \text{ km}]$  and at the 25<sup>th</sup> km are tolerable considering the fact that the anomalies used in the inversion are noisy.

The estimated structure from inversion (Figure 6.2b) remarkably coincides with the structure predicted from automatic modeling (Figure 5.6b) and also compares sensibly well with the assumed structure. In short, the gravity anomaly of the synthetic model analyzed both by automatic modeling and inversion yields exactly the same results.

## Inversion - Synthetic example

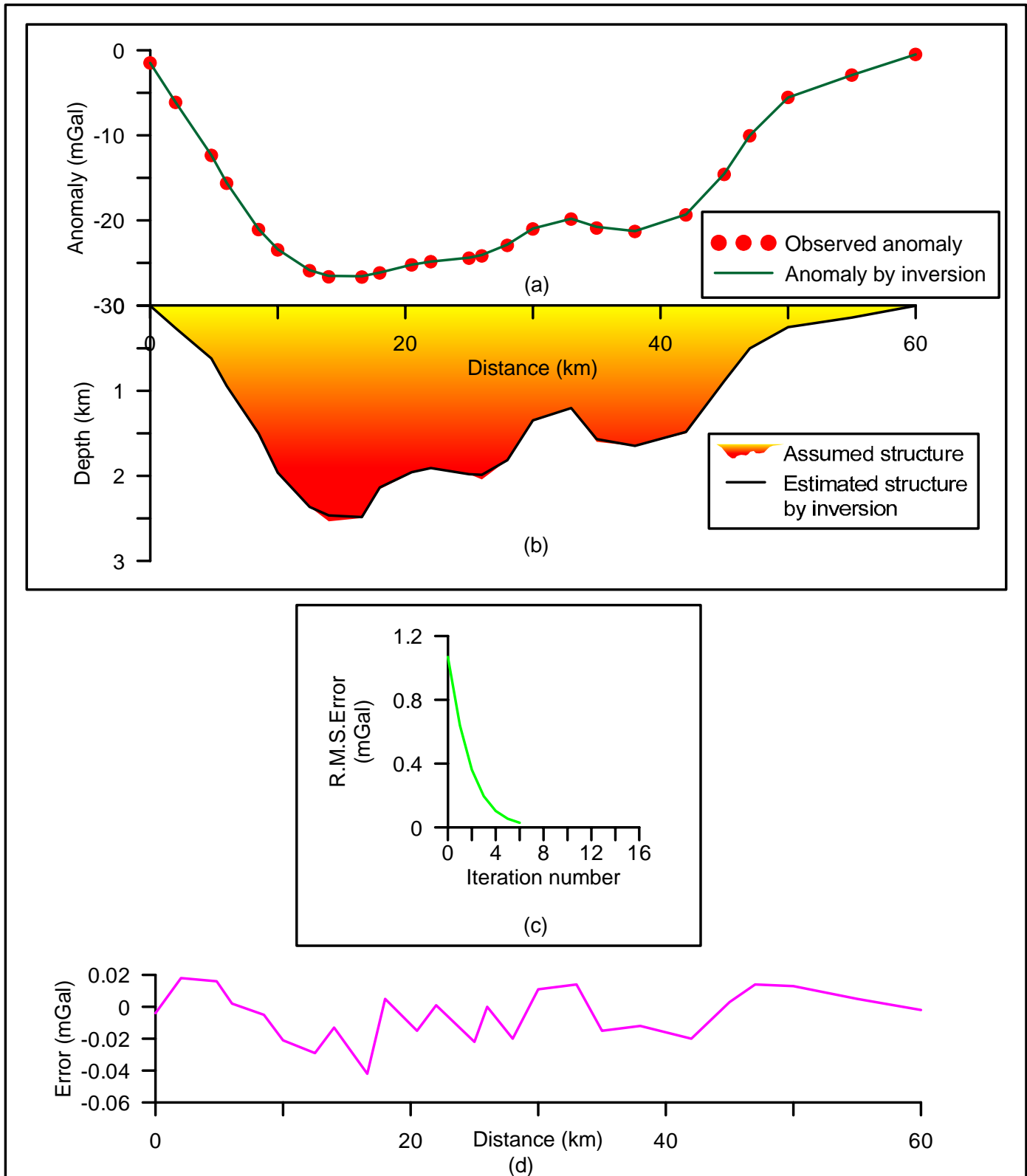


Figure 6.2 (a) Observed and theoretical noisy gravity anomalies by inversion, (b) assumed and estimated structures, (c) changes in misfit with iteration, (d) error between the observed and modeled anomalies at the end of 7<sup>th</sup> iteration. The description for color gradation from yellow to red within the structure is given in Figure 5.6.

### 6.5.2 Field Example

The inversion technique is applied to analyze the real field gravity data measured over the San Jacinto graben in California (Figure 5.7a) using a predefined EDCM and the interpretation is shown in Figure 6.2. It is to note that the same anomaly was interpreted in section 5.6.2 of Chapter-V by automatic modeling.

The gravity anomaly of the graben (Cordell, 1973) shown in Figure 6.3a when subjected to inversion with a prescribed EDCM (Figure 5.7c) the algorithm had performed 7 iterations. It was observed that the estimated structure did not show noticeable changes beyond 7<sup>th</sup> iteration. The modeled gravity anomaly of the structure subsequent to the inversion is shown in Figure 6.3a along with the observed anomaly and the inverted depth structure of the graben in Figure 6.3b, respectively. The inferred structure of the graben and corresponding gravity anomaly by Cordell (1973) were also shown in Figure 6.3a and 6.3b. One can notice from Figure 6.3a that the gravity response of the estimated structure from the present inversion closely resembles the observed gravity anomaly in comparison to Cordell's (1973) interpretation. The r.m.s. error between the observed and calculated gravity anomalies for the starting model was 2.81 mGal (Figure 6.3c) as observed in the case of automatic modeling. However, the decay of r.m.s. error with iteration was found to be rapid in case of inversion compared to automatic modeling (Figures 5.7d and 6.3c). Figure 6.3d compares the

## Inversion - Field example

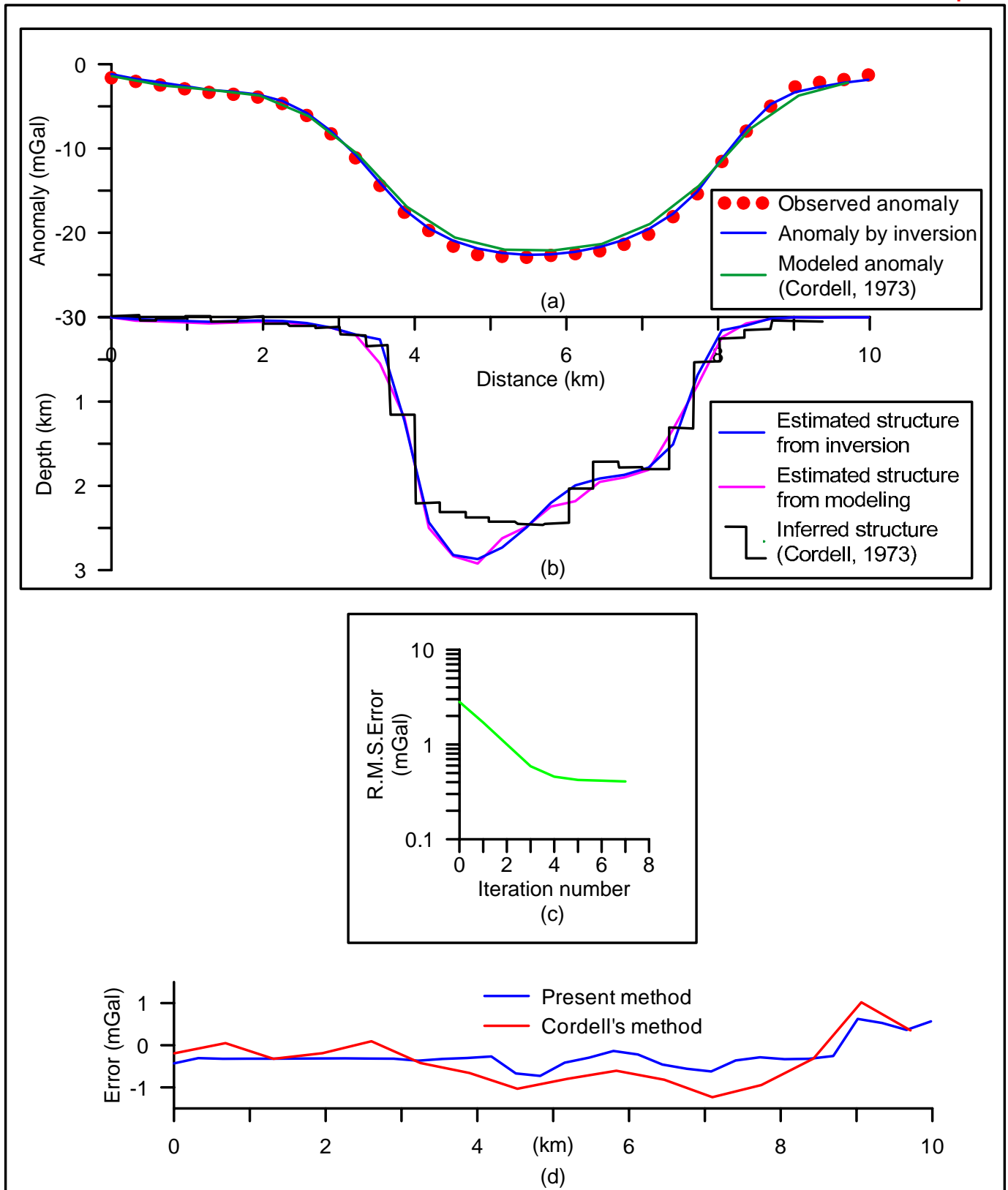


Figure 6.3 (a) Observed and theoretical gravity anomalies by inversion, San Jacinto graben, California. Modeled anomalies by Cordell (1973) are also shown for comparison, (b) estimated structure by present inversion. Depth structures inferred by Cordell (1973), and by automatic modeling (Chapter-V) are shown, (c) changes in r.m.s. error with iteration, (d) comparison of residuals in anomaly observed in the present method and Cordell's (1973) method.

magnitude of error (between the observed and model gravity anomalies) noticed in the present method with the one observed in Cordell's (1973) interpretation.

The inferred basement topography of the graben from the present inversion shows progressive deepening towards the west as observed in case of automatic modeling (Figure 5.7b). The maximum thickness of the graben estimated from the present inversion was 2.86 km, whereas Cordell reported a figure of 2.4 km (Figure 6.3b). The deciphered structure of the graben from automatic modeling is also shown in Figure 6.3b for comparison.

In short, the gravity anomalies of the San Jacinto graben analyzed by both automatic modeling and inversion yield more or less similar structural configuration with the estimated depths comparable to each other (Figure 6.3b).

## **6.5 Results and Discussion**

Based on the principles of inversion, a space domain based algorithm and related software are developed to automatically interpret the gravity anomalies attributed to sedimentary basins among which the density contrast decreases with depth following a prescribed exponential form.

The noisy gravity anomalies of a synthetic model of a sedimentary basin were analyzed using the present software in the light of proposed inversion and found that the estimated structure almost coincides with the assumed structure. All

the discontinuities present in the basement because of several synthetic and antithetic faults were also recovered successfully.

Further, the analysis of gravity anomalies of the San Jacinto graben, California using the proposed inversion has yielded a structural solution that is marginally deviated from the Cordell's (1973) interpreted model. The close fit between the observed and modeled gravity anomalies of the graben observed in the proposed method when compared to Cordell's (1973) interpretation has clearly demonstrates that the structural solution obtained from the present inversion is relatively more reliable than the one proposed by Cordell (1973).

It is of paramount interest to note that the estimated models from automatic modeling and inversion closely mimic with each other in both synthetic and real field examples. Further, from the analysis of gravity anomalies of synthetic and real field gravity anomalies it is found that the inversion algorithm performs lesser number of iterations in comparison to automatic modeling.

## Conclusions

---

The thesis consists of seven chapters. The novel features and important conclusions originate from the research work presented in Chapters II, III, IV, V, and VI are mainly on the following lines

1. Novelty in the interpretation methodologies including those of new forward modeling schemes,
2. Efficiency of an automatic inversion algorithm and related GUI software to analyze the gravity anomalies produced by 2D listric fault morphologies using a predefined exponential density contrast model,
3. Efficacy of an interactive modeling scheme to interpret the gravity anomalies of 2.5D listric fault morphologies using arbitrarily varying density contrast-depth models and related software,
4. Efficiency of an inversion algorithm to analyze the gravity anomalies for simultaneously estimating the density/depth parameters and fault plane geometries of 2.5D listric fault morphologies and related software,



5. Efficacies of automatic modeling and inversion strategies to analyze the gravity anomalies of 2D density interfaces using prescribed exponential density contrast models and related softwares’.

New forward modeling strategies combining both analytical and numerical approaches have been formulated in the spatial domain to compute the gravity anomalies of geologic structures among which the density contrast obeys exponential decrease with depth. The reliability of the forward modeling schemes are demonstrated on a few selected geophysical geometries by comparing the anomalies realized from the proposed methods with those obtained from the analytical methods.

Based on the interpretation methodologies presented in the thesis, a total of five new softwares’ have been developed using the JAVA programming language to analyze the gravity anomalies for respective subsurface geologic structures. The inclusion of GUI makes the softwares more elegant and easy to operate. The novelty of the softwares is that besides generating the output in ASCII and graphical formats, each one displays the animated versions of the subsurface model growth and corresponding improvement in gravity response of the structure with iteration. The variations in the data misfit and density contrast shall also be displayed in animated forms in respective panels. Furthermore, the 2.5D algorithms presented in Chapters III and IV are fairly applicable to analyze the gravity anomalies even when the profiles run at an offset across the source(s).

The effects of strike length and offset of the profile on the magnitude of gravity anomaly is discussed in length in Chapter-III. It is concluded that these parameters should be considered invariably in the analysis of gravity anomalies to achieve reliable interpretations.

The applicability of each technique is exemplified with synthetic as well as real field gravity anomalies. In case of synthetic examples, pseudorandom noise was added to the gravity response of respective structures before being subjected to analysis. In all cases, the interpreted results are compared excellently well with the assumed parameters in case of synthetic examples and with the drilling/available information in case of field examples.

### **Scope for future research**

An important problem to be addressed in gravity modeling studies is to develop new interpretation strategies, which shall take into account the presence of interfering sources within the sedimentary pack. The methodology elucidated in Chapter-VI can be extended to design appropriate inversion strategies to address the said problem.

An equally unsolved problem in gravity interpretation is to estimate the source parameters and regional gravity background simultaneously from a set of observed Bouguer gravity anomalies. Although, a few techniques are in vogue in this direction, each one has its own merits and demerits in its application. Therefore, it is expected that future research will contemplate more sophisticated regional-residual separation techniques, possibly integrated with the inversion

processes described in the thesis, to cater more lucid interpretations of gravity anomalies.

In case of strike limited listric fault morphologies, the techniques presented in Chapters-III and IV presume that the detached hanging wall systems consist of several geologic formations; with each one bounded on the top and bottom by flat surfaces, which in reality may or not be so. The proposed techniques are more effective when the assumptions are relatively valid. It is anticipated that future research shall consider imperfect polynomial limiting surface for the fault plane and uneven bounding surfaces for the formations within the hanging wall systems to develop appropriate inversion schemes.

\*\*\*\*\*

```

1  package com.ingrexp.view;
2
3  import java.awt.Frame;
4  import java.awt.event.MouseAdapter;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.WindowAdapter;
7  import java.awt.event.WindowEvent;
8  import java.io.File;
9
10 import javax.swing.JFrame;
11 import javax.swing.JOptionPane;
12
13 import com.ingrexp.control.INGREXP_Controller;
14 import com.ingrexp.model.INGREXP_CalculateValues;
15
16
17 public class INGREXP_MainView extends Frame {
18
19     /**
20      *
21      */
22     private static final long serialVersionUID = 1L;
23
24     public static void main(String s[])
25     {
26         INGREXP_MainView cm = new INGREXP_MainView();
27         cm.setSize(1280, 768);
28
29         cm.setTitle("INGREXP");
30         cm.setResizable(false);
31         cm.add(new INGREXP_MainPanel());
32         cm.addWindowListener(new WindowAdapter(){
33             public void windowClosing(WindowEvent e){
34                 JFrame frame = null;
35                 int r = JOptionPane.showConfirmDialog(
36                     frame,
37                     "Exit INGREXP ?",
38                     "Confirm Exit ",
39                     JOptionPane.YES_NO_OPTION);
40                 if(r == JOptionPane.YES_OPTION ){
41                     if(INGREXP_Controller.success==false){
42                         String fileName = INGREXP_CalculateValues.input_area_name+".jpg";
43                         File f = new File(fileName);
44                         f.delete();
45                     }
46                     System.exit(0);
47                 }
48             }
49         });
50         INGREXP_MainPanel.img.addMouseListener(new MouseAction());
51         cm.setVisible(true);
52     }
53 }
54
55 class MouseAction extends MouseAdapter{
56     public void mousePressed(MouseEvent e) {
57
58         INGREXP_CalculateValues.drawGraph();
59     }
60 }
61 }
62 -----
63
64 package com.ingrexp.view;
65
66 import java.awt.BorderLayout;
67 import java.awt.Button;
68 import java.awt.Color;
69 import java.awt.Font;
70 import java.awt.Graphics;
71 import java.awt.GridLayout;
72 import java.awt.Label;
73 import java.awt.Panel;
74 import java.awt.TextArea;

```

```

75     import java.awt.TextField;
76
77     import java.io.File;
78     import java.io.IOException;
79     import java.util.HashMap;
80
81     import javax.swing.JFileChooser;
82
83     import jxl.Cell;
84     import jxl.CellType;
85     import jxl.Sheet;
86     import jxl.Workbook;
87     import jxl.read.biff.BiffException;
88
89     import com.ingrexp.view.INGREXP_MainPanel;
90
91     public class INGREXP_MainPanel extends Panel {
92
93         /**
94          *
95          */
96         private static final long serialVersionUID = 1L;
97         public static TextArea img = new TextArea(36,140);
98         Panel p_North, p_West;
99         public static Panel p_East;
100
101         static Panel p_South;
102
103         public static Panel p_Center;
104         static TextField inputValues [] = new TextField[12];
105
106         Button actionButton[] = new Button[6];
107         String rowdata[][]={};
108
109         /**Field Area Name*/
110         final static int AREA_FE = 0;
111         /**Number of the Profile*/
112         final static int NUM_PROFILE = 1;
113         /**Number of observations*/
114         final static int N_OBS = 2 ;
115         /**Distance(km)*/
116         final static int X_KM = 3;
117         /**Elevation(km)*/
118         final static int ELE_KM = 4;
119         /**observed anomalies*/
120         final static int NOB_GOB = 5;
121         /** Surface density contrast (gm/cc) */
122         final static int SD_POLY = 6;
123         /**Lambda*/
124         final static int LAMBDA_ST = 7 ;
125         /**Number of iteration values*/
126         final static int NOB_ITER = 8;
127
128         public INGREXP_MainPanel(){
129
130             this.setLayout(new BorderLayout());
131             p_North = new Panel();
132             p_West = new Panel();
133             p_East = new Panel ();
134             p_South = new Panel();
135             p_Center = new Panel();
136
137             Label graphLabel = new Label("INVERSION OF GRAVITY ANOMALIES OF 2D LISTRIC FAULT
STRUCTURES USING EXPONENTIAL DENSITY MODEL", Label.CENTER);
138             graphLabel.setFont(new Font("Bold", 1, 15));
139             p_Center.add(graphLabel);
140
141             for(int i = 0; i < 9; i++){
142                 inputValues[i] = new TextField();
143             }
144             p_North.setFont(new Font("Bold",1,12));
145             actionButton[0] = new Button("Load data");
146             actionButton[1] = new Button("Interpretation");
147             actionButton[2] = new Button("Save & Print");
148             actionButton[3] = new Button("Clear");

```

```

149         actionButton[4] = new Button("Exit");
150
151         this.populateNorthPanel();
152         INGREXP_TableView.populateEastPanel(rowdata);
153         this.add(p_North, BorderLayout.NORTH);
154         p_Center.setSize(1000, 760);
155         this.add(p_Center, BorderLayout.CENTER);
156         img.setEditable(false);
157         p_Center.add(img);
158         this.add(p_East, BorderLayout.EAST);
159         this.setVisible(true);
160     }
161
162     public void populateNorthPanel(){
163         p_North.setLayout(new GridLayout(4,6));
164         p_North.add(new Label("Area Name"));
165         p_North.add(inputValues[0]);
166         p_North.add(new Label("Profile Name"));
167         p_North.add(inputValues[1]);
168         p_North.add(new Label("Number of observations"));
169         p_North.add(inputValues[2]);
170         p_North.add(new Label("Distance (km)"));
171         p_North.add(inputValues[3]);
172         p_North.add(new Label("Elevation of each station (km)"));
173         p_North.add(inputValues[4]);
174         p_North.add(new Label("Observed anomalies (mGal)"));
175         p_North.add(inputValues[5]);
176         p_North.add(new Label("Surface density contrast (gm/cc)"));
177         p_North.add(inputValues[6]);
178         p_North.add(new Label("Lambda (1/km)"));
179         p_North.add(inputValues[7]);
180         p_North.add(new Label("Iterations"));
181         p_North.add(inputValues[8]);
182
183         p_North.add(new Label(""));
184         p_North.add(actionButton[0]);
185         p_North.add(actionButton[1]);
186         p_North.add(actionButton[2]);
187         p_North.add(actionButton[3]);
188         p_North.add(actionButton[4]);
189
190         actionButton[0].addActionListener(new com.ingrexp.control.INGREXP_Controller());
191         actionButton[1].addActionListener(new com.ingrexp.control.INGREXP_Controller());
192         actionButton[2].addActionListener(new com.ingrexp.control.INGREXP_Controller());
193         actionButton[3].addActionListener(new com.ingrexp.control.INGREXP_Controller());
194         actionButton[4].addActionListener(new com.ingrexp.control.INGREXP_Controller());
195
196     }
197
198     public static HashMap captureValues(){
199         HashMap h_Map = new HashMap();
200         try {
201             h_Map.put("AREA_FE", inputValues[AREA_FE].getText());
202             h_Map.put("NUM_PROFILE", inputValues[NUM_PROFILE].getText());
203             h_Map.put("N_OBS", inputValues[N_OBS].getText());
204             h_Map.put("X_KM", inputValues[X_KM].getText());
205             h_Map.put("ELE_KM", inputValues[ELE_KM].getText());
206             h_Map.put("NOB_GOB", inputValues[NOB_GOB].getText());
207             h_Map.put("SD_POLY", inputValues[SD_POLY].getText());
208             h_Map.put("ALPHA_ST", inputValues[LAMBDA_ST].getText());
209             h_Map.put("NOB_ITER", inputValues[NOB_ITER].getText());
210
211         }
212         catch (Exception e) {
213             e.printStackTrace();
214         }
215         return h_Map;
216
217     }
218
219     public static void clearPanel(Panel p) {
220         Graphics g = p.getGraphics();
221         g.setColor(Color.WHITE);
222         g.fillRect(0, 30, 1280, 650);
223     }

```

```

224
225
226     public static void loadData() throws IOException {
227         try{
228
229             String current = System.getProperty("user.dir");
230             JFileChooser chooser=new JFileChooser(current);
231             int returnVal = chooser.showOpenDialog(null);
232             String dis[], ele[],gobs[];
233             String disval = " ",elevel="",gobsval="";
234             Workbook w;
235
236             if(returnVal == JFileChooser.APPROVE_OPTION) {
237                 File f = chooser.getSelectedFile();
238                 w = Workbook.getWorkbook(f);
239                 Sheet sheet = w.getSheet(0);
240                 dis = new String[sheet.getRows()+1];
241                 ele = new String[sheet.getRows()+1];
242                 gobs = new String[sheet.getRows()+1];
243                 for (int j = 0; j < sheet.getColumns(); j++) {
244                     for (int i = 1; i < sheet.getRows(); i++) {
245                         Cell cell = sheet.getCell(j, i);
246                         CellType type = cell.getType();
247                         if (type == CellType.LABEL) {
248                             INGREXP_MainPanel.inputValues[INGREXP_MainPanel.AREA_FE].setText
249 (cell.getContents());
250                             }
251                             if (type == CellType.NUMBER) {
252                                 if (j == 1){
253                                     INGREXP_MainPanel.inputValues[INGREXP_MainPanel.NUM_PROFI]
254 .setText(cell.getContents());
255                                 }
256                                 if (j == 2){
257                                     INGREXP_MainPanel.inputValues[INGREXP_MainPanel.N_OBS].setT
258 xt(cell.getContents());
259                                 }
260                                 if (j == 3){
261                                     dis[i] = cell.getContents()+" ";
262                                     disval = disval + dis[i];
263                                 }
264                                 if (j == 4){
265                                     ele[i] = cell.getContents()+" ";
266                                     elevel = elevel + ele[i];
267                                 }
268                                 if (j == 5){
269                                     gobs[i] = cell.getContents()+" ";
270                                     gobsval = gobsval + gobs[i];
271                                 }
272                                 if (j == 6){
273                                     INGREXP_MainPanel.inputValues[INGREXP_MainPanel.SD_POLY].se
274 tText(cell.getContents());
275                                 }
276                                 if (j == 7){
277                                     INGREXP_MainPanel.inputValues[INGREXP_MainPanel.LAMBDA_ST].
278 etText(cell.getContents());
279                                 }
280                                 if (j == 8){
281                                     INGREXP_MainPanel.inputValues[INGREXP_MainPanel.NOB_ITER].s
282 tText(cell.getContents());
283                                 }
284                             }
285                         }
286                     }
287                 }
288                 INGREXP_MainPanel.inputValues[INGREXP_MainPanel.X_KM].setText(" "+disval);
289                 INGREXP_MainPanel.inputValues[INGREXP_MainPanel.ELE_KM].setText(" "+elevel);
290                 INGREXP_MainPanel.inputValues[INGREXP_MainPanel.NOB_GOB].setText(" "+gobsval);
291             }
292         }
293     }
294     catch (BiffException e) {
295         e.printStackTrace();
296     }
297 }

```

```

291     public static void clearDefaultValues(){
292         inputValues[AREA_FE].setText( " ");
293         inputValues[NUM_PROFILE].setText( " ");
294         inputValues[N_OBS].setText( " ");
295         inputValues[X_KM].setText( " ");
296         inputValues[ELE_KM].setText( " ");
297         inputValues[NOB_GOB].setText( " ");
298         inputValues[SD_POLY].setText( " ");
299         inputValues[LAMBDA_ST].setText( " ");
300         inputValues[NOB_ITER].setText( " ");
301     }
302 }
303 -----
304 package com.ingrexp.view;
305
306 import java.awt.Color;
307 import java.awt.Dimension;
308 import java.awt.Font;
309 import java.awt.Graphics;
310 import java.awt.GridLayout;
311 import java.awt.Panel;
312 import java.awt.TextArea;
313
314 import javax.swing.JScrollPane;
315 import javax.swing.JTable;
316
317 import com.ingrexp.view.INGREXP_MainPanel;
318
319 public class INGREXP_TableView extends Panel{
320
321     /**
322     *
323     */
324     private static final long serialVersionUID = 1L;
325     public static TextArea val = new TextArea(5,5);
326     public static void populateEastPanel(Object rowData[][]) {
327         com.ingrexp.view.INGREXP_MainPanel.p_East.removeAll();
328
329         com.ingrexp.view.INGREXP_MainPanel.p_East.setLayout(new GridLayout(2,1));
330
331         Object columnNames[] = {"Distance(km)", "Observed anomalies (mGal)", "Calculated
332 anomalies (mGal)", "Error (mGal)"};
333         JTable table = new JTable(rowData, columnNames);
334         table.setPreferredScrollableViewportSize(new Dimension(300,500));
335         JScrollPane scrollPane = new JScrollPane(table);
336         scrollPane.setAutoscrolls(true);
337
338         com.ingrexp.view.INGREXP_MainPanel.p_East.add(scrollPane);
339         val.setEditable(false);
340         com.ingrexp.view.INGREXP_MainPanel.p_East.add(val);
341         try{
342             com.ingrexp.view.INGREXP_MainPanel.p_East.validate();
343         }
344         catch(Exception e){
345             Graphics g =INGREXP_MainPanel.img.getGraphics();
346             g.setColor(Color.white);
347             g.fillRect(0, 0, 1000, 600);
348             g.setColor(Color.black);
349             g.setFont(new Font("Arial", 20, 40));
350             g.drawString("ERROR...", 300, 400);
351         }
352         com.ingrexp.view.INGREXP_MainPanel.p_East.setVisible(true);
353     }
354 }
355 -----
356 package com.ingrexp.view;
357
358 import java.applet.Applet;
359
360
361
362

```



```

363 import java.awt.Color;
364 import java.awt.Font;
365 import java.awt.GradientPaint;
366 import java.awt.Graphics2D;
367 import java.awt.geom.Line2D;
368 import java.awt.geom.Rectangle2D;
369 import java.text.DecimalFormat;
370
371
372
373 import com.ingrexp.model.INGREXP_CalculateValues;
374 import com.ingrexp.util.INGREXP_Utility;
375
376 public class INGREXP_DrawGraph extends Applet{
377     private static final long serialVersionUID = 1L;
378     public static int i_no_obs;
379     float maxY,maxZ,maxX ;
380     double obs[];
381     double inidep;
382     public void drawGraph(Graphics2D g2) {
383
384         g2.setFont(new Font("Arial", 20, 12));
385         g2.setColor(Color.BLACK);
386         g2.draw(new Line2D.Float(150, 50, 150, 300));
387         g2.drawLine(90,35,1040,35);
388         g2.drawLine(780, 35, 780, 560);
389         g2.drawLine(90, 560, 1040, 560);
390         g2.drawLine(1040, 35, 1040, 560);
391         g2.drawLine(90, 35, 90, 560);
392
393         String []a = {"A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
394         String []b = {"D", "E", "P", "T", "H", "(k", "m)"};
395         for (int i = 0; i < a.length; i++) {
396             g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
397         }
398         for (int i = 0; i < b.length; i++) {
399             g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
400         }
401     }
402
403
404     public void plot(Graphics2D g2) {
405
406         g2.setFont( new Font("Arial", 12, 12) );
407         DecimalFormat f = new DecimalFormat("0.##");
408         g2.setColor(Color.BLACK);
409
410         i_no_obs = INGREXP_CalculateValues.input_n_obs;
411         inidep = INGREXP_CalculateValues.o_par[1];
412         obs = new double[i_no_obs+1];
413         for (int i = 1; i <= i_no_obs; i++) {
414             obs[i] = INGREXP_CalculateValues.input_x_km[i];
415         }
416
417         maxX = (float)obs[i_no_obs];
418         float maxy = (float)
INGREXP_Utility.findMaximumNumber(INGREXP_CalculateValues.input_nob_gob);
419         float maxy1 = (float)
INGREXP_Utility.findMaximumNumber(INGREXP_CalculateValues.o_GC);
420         if(maxy > maxy1)
421             maxY = maxy;
422         else
423             maxY = maxy1;
424         maxZ = (float)INGREXP_CalculateValues.o_par[1];
425
426         g2.drawString("|", (float) 600 , 308);
427         g2.drawString(""+f.format(INGREXP_CalculateValues.input_x_km[i_no_obs]), (float) 600
, 323);
428         g2.drawString("0", 125, 310);
429         g2.drawString("DISTANCE(km)", 315, 295);
430         float xplot = 0;
431         float xInterval = (float) (INGREXP_CalculateValues.input_x_km[i_no_obs] / 5);
432
433         int zInterval = 50;
434         for (float x = xInterval, j = 1; x < 600; x += xInterval){

```

```

435         xplot = xplot + xInterval;
436         if(j > 4)
437             break;
438         g2.drawString("|", (float) (150 + (450 * x / maxX) ), 308);
439         g2.drawString(" " + f.format(xplot), (float) (150 + (450 * x / maxX) ) - 3, 323);
440         j++;
441     }
442
443     DecimalFormat d = new DecimalFormat("0.#");
444     float points1 = maxZ / 5 ;
445     for (int x = zInterval + 250, j = 1; x < 550; x += zInterval){
446         g2.drawString("-", 148, 52 + x);
447         g2.drawString(" " + d.format(points1 * j), 125, 50 + x);
448         j++;
449     }
450 }
451
452
453 public void plotXYCoordinates(Graphics2D g2){
454
455     double minAno =
INGREXP_Utility.findMinimumNumber1(INGREXP_CalculateValues.input_nob_gob);
456     double maxAno =
INGREXP_Utility.findMaximumNumber(INGREXP_CalculateValues.input_nob_gob, minAno);
457     double minObAno = INGREXP_Utility.findMinimumNumber1(INGREXP_CalculateValues.o_GC);
458     double maxObAno = INGREXP_Utility.findMaximumNumber(INGREXP_CalculateValues.o_GC,
minObAno);
459
460     if (minAno < 0 && maxObAno < 0 && maxAno < 0 && minObAno < 0 ){
461         plotXYCoordinates1(g2);
462     }
463     if (minAno >= 0 && maxObAno > 0 ){
464         plotXYCoordinates1(g2);
465     }
466     if (minAno < 0 && maxObAno > 0 || maxAno>0 && minObAno<0){
467         plotXYCoordinates2(g2);
468     }
469 }
470
471 public void plotXYCoordinates1 (Graphics2D g2) {
472
473     g2.setFont(new Font("Arial", 20, 12));
474     g2.setColor(Color.black);
475     float maxval = (float)
INGREXP_Utility.findMaximumNumber(INGREXP_CalculateValues.o_GC);
476     float maxval1 = (float)
INGREXP_Utility.findMaximumNumber(INGREXP_CalculateValues.input_nob_gob);
477     if(Math.abs(maxval)>Math.abs(maxval1))
478         maxY = maxval;
479     else
480         maxY = maxval1;
481     int points = (int)maxY / 5;
482     int yInterval = 50;
483     g2.drawString("0", 125, 50);
484     for (int x = yInterval, j = 1; x < 250; x += yInterval){
485
486         g2.drawString("-", 148, 50 + x);
487         g2.drawString(" " + (points*j), 125, 50 + x);
488         j++;
489     }
490     float prevx = (float) (150 + ( 450 * obs[1] / maxX));
491     float prevy = (float)( 50 + ( 250 * INGREXP_CalculateValues.o_GC[1] / maxY ) );
492     float xpoint = 0;
493     float ypoint = 0;
494     float gypoint = 0;
495
496     for (int k = 1; k <= i_no_obs; k++) {
497         xpoint = (float)( 450 * obs[k] / maxX);
498         ypoint = (float)( ( 250 * INGREXP_CalculateValues.o_GC[k] / maxY ) );
499         gypoint = (float)( ( 250 * INGREXP_CalculateValues.input_nob_gob[k] / maxY ) );
500         g2.setColor(Color.BLACK);
501         g2.draw(new Line2D.Float(prevx, prevy, 150+ xpoint, 50 + ypoint ));
502         g2.setColor(Color.BLUE);
503         g2.setFont(new Font("Arial", 20, 40));
504         g2.drawString(".", 150+xpoint - 6 , 50 + gypoint + 3);

```

```

505         g2.setFont(new Font("Arial", 20, 12));
506         g2.setColor(Color.black);
507         prevx = 150 + xpoint;
508         prevy = 50 + ypoint ;
509     }
510 }
511
512
513 public void plotXYCoordinates2 (Graphics2D g2) {
514     g2.setFont(new Font("Arial", 20, 12));
515     g2.setColor(Color.black);
516     int countPosObs=0, countNegObs=0, countPosCal=0, countNegCal=0;
517     double store[] = new double[INGREXP_CalculateValues.input_n_obs + 1];
518     double store1[] = new double[INGREXP_CalculateValues.input_n_obs + 1];
519     double negStore[] = new double[INGREXP_CalculateValues.input_n_obs + 1];
520     double negStore1[] = new double[INGREXP_CalculateValues.input_n_obs + 1];
521     for(int i = 1; i <= INGREXP_CalculateValues.input_n_obs; i++){
522         if(INGREXP_CalculateValues.o_GC[i] > 0){
523             store[i] = INGREXP_CalculateValues.o_GC[i];
524             countPosCal = countPosCal + 1;
525         }
526         else{
527             negStore[i] = INGREXP_CalculateValues.o_GC[i];
528             countNegCal = countNegCal + 1;
529         }
530         if(INGREXP_CalculateValues.input_nob_gob[i]>0){
531             store1[i] = INGREXP_CalculateValues.input_nob_gob[i];
532             countPosObs = countPosObs + 1;
533         }
534         else{
535             negStore1[i] = INGREXP_CalculateValues.input_nob_gob[i];
536             countNegObs = countNegObs + 1;
537         }
538     }
539
540     float maxPos = (float) INGREXP_Utility.findMaximumNumber1(store);
541     float maxPos1 = (float) INGREXP_Utility.findMaximumNumber1(store1);
542     float maxNeg = (float) INGREXP_Utility.findMaximumNumber(negStore);
543     float maxNeg1 = (float) INGREXP_Utility.findMaximumNumber(negStore1);
544     float posNum =0;
545
546     if(maxPos > maxPos1)
547         posNum = maxPos;
548     else
549         posNum = maxPos1;
550     if(Math.abs(maxNeg) > Math.abs(maxNeg1))
551         maxY = maxNeg;
552     else
553         maxY = maxNeg1;
554
555     float prevx = (float) (150 +( 450 * obs[1] / maxX));;
556     float prevy = 0;
557     if(INGREXP_CalculateValues.o_GC[1]>0){
558         if(countNegCal > countPosObs)
559             prevy = 100 - (float)(( 50 * INGREXP_CalculateValues.o_GC[1] / posNum ) );
560         else
561             prevy = 200 - (float)(( 150 * INGREXP_CalculateValues.o_GC[1] / posNum ) );
562     }
563     else{
564         if(countNegCal > countPosObs)
565             prevy = 100 + (float)(( 200 * INGREXP_CalculateValues.o_GC[1] / maxY ) );
566         else{
567             prevy = 200+(float)(( 100 * INGREXP_CalculateValues.o_GC[1] / posNum ) );
568         }
569     }
570     float xpoint = 0;
571     float ypoint = 0;
572     float gypoint = 0;
573     float points = 0;
574
575     DecimalFormat f = new DecimalFormat("0.#");
576     if(countNegCal > countPosObs){
577         g2.drawString("-", 148, 100);
578         g2.drawString("0", 125, 100);
579     }

```

```

580         else{
581             g2.drawString("-", 148, 200);
582             g2.drawString("0",125,200);
583         }
584         g2.drawString("-", 148, 55);
585         g2.drawString(""+f.format(posNum), 125, 55);
586
587         if(countNegCal > countPosObs){
588             points = maxY / 4;
589             int yInterval=50;
590             for (int x = yInterval, j = 1; x < 250; x+=yInterval){
591
592                 g2.drawString("-", 148, 100 + x );
593                 g2.drawString("" + f.format(points * j), 125, 100 + x );
594                 j++;
595             }
596         }
597         else{
598             points = posNum / 3;
599             int yInterval=50;
600             for (int x = yInterval, j = 1; x < 200; x+=yInterval){
601
602                 g2.drawString("-", 148, 205 - x );
603                 g2.drawString("" + f.format(points * j), 125, 205 - x );
604                 j++;
605             }
606
607             g2.drawString("-", 148, 250 );
608             g2.drawString("-", + f.format(posNum/2), 125 , 250 );
609             g2.drawString("-", 148, 300 );
610             g2.drawString("-", + f.format(posNum), 125, 300 );
611
612         }
613         for (int k = 1; k <= i_no_obs; k++) {
614
615             xpoint = (float)( 450 * obs[k] / maxX);
616             if(INGREXP_CalculateValues.o_GC[k]>0){
617                 if(countNegCal > countPosObs)
618                     ypoint = 100-(float)( ( 50 * INGREXP_CalculateValues.o_GC[k] / posNum
619 );
620                 else
621                     ypoint = 200-(float)( ( 150 * INGREXP_CalculateValues.o_GC[k] / posNum
622 );
623             }
624             else{
625                 if(countNegCal > countPosObs)
626                     ypoint = 100+(float)( ( 200 * INGREXP_CalculateValues.o_GC[k] / maxY )
627 );
628                 else
629                     ypoint = 200+(float)( ( 100 * INGREXP_CalculateValues.o_GC[k] / posNum
630 );
631             }
632             if(INGREXP_CalculateValues.input_nob_gob[k]>0){
633                 if(countNegCal > countPosObs)
634                     gypoint = 100-(float)( ( 50 * INGREXP_CalculateValues.input_nob_gob[k]
635 posNum ) );
636                 else
637                     gypoint = 200-(float)( ( 150 * INGREXP_CalculateValues.input_nob_gob[k]
638 / posNum ) );
639             }
640             else{
641                 if(countNegCal > countPosObs)
642                     gypoint = 100+(float)( ( 200 * INGREXP_CalculateValues.input_nob_gob[k]
643 / maxY ) );
644                 else
645                     gypoint = 200+(float)( ( 100 * INGREXP_CalculateValues.input_nob_gob[k]
646 / posNum ) );
647             }
648             g2.setColor(Color.BLACK);
649             g2.draw(new Line2D.Float(prevx, prevy, 150 + xpoint, ypoint ));
650
651             g2.setColor(Color.BLUE);
652             g2.setFont(new Font("Arial", 20, 40));
653             g2.drawString(".", 150+xpoint - 6 , gypoint+3 );
654

```

```

647         g2.setFont(new Font("Arial", 20, 12));
648         g2.setColor(Color.black);
649         prevx = 150 + xpoint;
650         prevy = ypoint ;
651     }
652 }
653 }
654 public void drawOBJ(Graphics2D g2) {
655
656     g2.setColor(Color.BLACK);
657     g2.drawLine(820, 70, 820, 160);
658     g2.drawLine(820, 160, 910, 160);
659     g2.drawString("J", 800, 90);
660
661     double maxOb = INGREXP_Utility.findMaximumNumber1(INGREXP_CalculateValues.o_funcnt).
662     int ini = INGREXP_Utility.findMaximumNumber(INGREXP_CalculateValues.o_iter);
663     if(ini == 5)
664         ini= ini + 1;
665     int maxiter = ( ini / 3 * 5 ) * 2;
666     int point;
667     int xInterval = 22;
668     point = ( ( ini ) / 3 * 5 ) / 5;
669
670     for (int x = xInterval, j = 1; x < 90 ; x += xInterval) {
671
672         g2.drawString("", 821 + x, 170);
673         g2.drawString(" " + (point*j), 820 + x-3, 175);
674         j++;
675     }
676
677     float prevx = 820;
678     float prevy = 70;
679     float xpoint = 0;
680     float ypoint = 0;
681
682     for (int i = 1; i <= INGREXP_CalculateValues.o_iter; i++) {
683
684         xpoint = (float)( 250 * i /maxiter );
685         ypoint = 70 - (float) ( ( 90 * (INGREXP_CalculateValues.o_funcnt[i]) / maxOb )
686 );
687
688         if(i==INGREXP_CalculateValues.o_iter){
689             g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint-4, 90 + ypoint));
690         }
691         else {
692             g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 90 + ypoint));
693         }
694         prevx = 820 + xpoint;
695         prevy = 90 + ypoint;
696     }
697     DecimalFormat d1= new DecimalFormat("0.###");
698     DecimalFormat d= new DecimalFormat("0.#");
699     g2.drawString(" "+d.format(INGREXP_CalculateValues.o_funcnt[1]), 780, 70);
700     g2.drawString("
+d1.format(INGREXP_CalculateValues.o_funcnt[INGREXP_CalculateValues.o_iter]), 820 +
xpoint, 90 + ypoint);
701     g2.setFont(new Font("Arial", 40,11));
702     g2.drawString ("Iterations",850,186);
703 }
704
705 public void drawSd(Graphics2D g2) {
706     g2.setColor(Color.black);
707     g2.drawLine(780, 200, 1040, 200);
708     g2.setColor(Color.red);
709     g2.setFont(new Font("Arial", 20, 12));
710     DecimalFormat d= new DecimalFormat("0.###");
711     DecimalFormat d2= new DecimalFormat("0.#");
712     DecimalFormat d1= new DecimalFormat("0.###");
713
714     g2.drawString(""+d.format(inidep), 790, 550);
715     g2.drawString("-", 820, 552);
716     g2.drawString("0", 807 , 300);
717     g2.drawLine(820, 300, 910, 300);
718     g2.draw(new Line2D.Float(820, 300, 820, 550));

```

```

719
720 double maxOb1 = INGREXP_Utility.findMaximumNumber1(INGREXP_CalculateValues.vsd);
721 float points = maxZ / 5 ;
722 int zInterval = 50;
723 for(int x = zInterval+250,j = 1; x < 550; x+=zInterval){
724     if(j>4)
725         break;
726     g2.drawString("-", 820, 50 + x + 2);
727     g2.drawString(" " +d2.format(points * j), 790, 50 + x);
728     j++;
729 }
730 float prevx = 820+ (float) ( ( 90 * ( Math.abs(INGREXP_CalculateValues.vsd[1] )) /
maxOb1 ) );
731 float prevy = 300;
732 float xpoint = 0;
733 float ypoint = 0;
734
735 for (int i = 1; i <= INGREXP_CalculateValues.count; i++) {
736     xpoint = (float)( 90 * Math.abs(INGREXP_CalculateValues.vsd[i]) / maxOb1 );
737     ypoint = (float)( 250 * INGREXP_CalculateValues.dep[i] / maxZ );
738     g2.setColor(Color.blue);
739     g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 300 + ypoint));
740
741     prevx = 820 + xpoint;
742     prevy = 300 + ypoint;
743 }
744
745 g2.drawString(""+d.format(INGREXP_CalculateValues.vsd[1] ),805+ (float) ( ( 90 * (
Math.abs(INGREXP_CalculateValues.vsd[1] )) / maxOb1 ) ) ,300 );
746 g2.drawString(""+d1.format(INGREXP_CalculateValues.vsd[INGREXP_CalculateValues.cour
] ),820+ (float) ( ( 90 * (
Math.abs(INGREXP_CalculateValues.vsd[INGREXP_CalculateValues.count] )) / maxOb1 ) ) ,
300+(float)( 250 * inidep / maxZ ) );
747 g2.setColor(Color.BLACK);
748 g2.drawString("Variation of density contrast " , 800,220);
749 g2.drawString("with depth" , 850,240);
750 g2.setFont(new Font("Arial", 40,11));
751 g2.drawString ( "Density contrast",830,285);
752 g2.drawString ( "(gm/cc)",843,295);
753 g2.drawString("Z(km)", 790,(float)( 300+((250*inidep/maxZ))/2));
754
755 }
756 public void plotZCoordinates (Graphics2D g2) {
757
758     GradientPaint gradient = new GradientPaint(10, 10, Color.yellow, 30, 200,
Color.MAGENTA, true);
759     g2.setPaint(gradient);
760     g2.fill(new Rectangle2D.Float(150, 300, 450 , 250));
761     g2.setColor(Color.BLACK);
762     i_no_obs = INGREXP_CalculateValues.input_n_obs;
763     obs = new double[i_no_obs+1];
764     for (int i = 1; i <= i_no_obs; i++) {
765         obs[i] = INGREXP_CalculateValues.input_x_km[i];
766     }
767     maxX = (float) obs[i_no_obs];
768     maxZ = (float)INGREXP_CalculateValues.o_par[1];
769     float s = 0;
770     float fc = 0;
771     float spoint = 300;
772     float fcpoint = (float)( 150 + ( 450 * INGREXP_CalculateValues.o_par[2] / maxX ) );
773     float xpoint = 0;
774     float zpoint = 0;
775     while (s <= INGREXP_CalculateValues.o_par[1]) {
776         float z1 = (float) 0.001;
777         s = s + z1;
778         fc = 0;
779         for (int i = 1; i<4; i++){
780             fc = (float) (fc + INGREXP_CalculateValues.o_par[i + 1] * Math.pow(s, i -
1));
781         }
782         xpoint = (float)( 450 * fc / maxX);
783         zpoint = (float)( 250 * s / maxZ);
784         g2.setColor(Color.BLACK);
785         g2.draw(new Line2D.Float(fcpoint, spoint,(float)150 + xpoint, 300 + zpoint));
786         g2.setColor(Color.RED);

```

```

787         g2.draw(new Line2D.Float(150, 300 + zpoint, (float) 150 + xpoint, 300 +
zpoint));
788         fcpoint = (float) 150 + xpoint;
789         spoint = 300 + zpoint;
790     }
791     g2.setColor(Color.BLACK);
792     g2.drawLine(150, 300, 600, 300);
793     g2.drawLine(150, 300, 550, 300);
794 }
795 public void idex(Graphics2D g){
796     g.setColor(Color.BLUE);
797     g.setFont(new Font("Arial", 20, 50));
798     g.drawString(" ... ",595,70);
799     g.setFont(new Font("Arial", 20, 12));
800     g.drawString("Observed anomalies",650,70);
801     g.setColor(Color.BLACK);
802     g.drawString("____:",615,87);
803     g.drawString("Calculated anomalies",650,90);
804     GradientPaint gradient = new GradientPaint(10, 10, Color.yellow, 30, 50,
Color.MAGENTA, true);
805     g.setPaint(gradient);
806     g.fillRect(615, 100, 35, 10);
807     g.setColor(Color.BLACK);
808     g.drawString(" : Estimated Depth", 650, 110);
809     g.drawString("Structure", 660, 130);
810 }
811 }
812 -----

```

```

813 package com.ingrexp.model;
814
815 import java.awt.Color;
816 import java.awt.Graphics;
817 import java.awt.Graphics2D;
818 import java.awt.event.MouseAdapter;
819 import java.awt.event.MouseEvent;
820 import java.awt.event.MouseListener;
821 import java.awt.image.BufferedImage;
822 import java.io.File;
823 import java.io.FileOutputStream;
824 import java.text.DecimalFormat;
825 import java.util.HashMap;
826
827 import javax.imageio.ImageIO;
828 import com.ingrexp.model.INGREXP_NOREQ;
829
830 import com.ingrexp.util.INGREXP_Utility;
831 import com.ingrexp.view.INGREXP_MainPanel;
832 import com.ingrexp.view.INGREXP_TableView;
833
834 public class INGREXP_CalculateValues {
835     public static int input_n_obs = 0;
836     public static double input_x_km[];
837     public static double input_nob_gob[];
838     public static Object obj[][] = null;
839     public static double []o_GC ;
840     public static double []o_err ;
841     public static double []o_par;
842     public static double []o_funct ;
843     public static double []vsd = null;
844     public static double []dep = null;
845     public static double o_func;
846     public static int o_iter,count;
847     public static String input_area_name,input_profile_num="";
848     public static int np;
849     public static BufferedImage image;
850
851     public void getAnamolyValues(HashMap h_Map) {
852         double ALER = 0.0000001;
853         double input_sd_poly = 0;
854         double input_lambda_val = 0;
855         int input_ndeg = 2;
856         double []input_ele_km = null;
857         int input_nob_iter = 0;
858     }

```



```

859
860
861     try {
862
863         input_n_obs = INGREXP_Utility.convertInteger((String)h_Map.get("N_OBS"));
864         input_x_km = INGREXP_Utility.convertDoubleArray((String)h_Map.get("X_KM"));
865         input_ele_km = INGREXP_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
866         input_nob_gob =
INGREXP_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
867         input_sd_poly = INGREXP_Utility.convertDouble((String)h_Map.get("SD_POLY"));
868         input_lambda_val =
INGREXP_Utility.convertDouble((String)h_Map.get("ALPHA_ST"));
869         input_nob_iter = INGREXP_Utility.convertInteger((String)h_Map.get("NOB_ITER"));
870         input_area_name = INGREXP_Utility.convertString((String)h_Map.get("AREA_FE"));
871         input_profile_num =
INGREXP_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
872     }
873     catch(Exception e) {
874         e.printStackTrace();
875     }
876
877     o_GC = new double[input_n_obs + 1];
878     o_err = new double[input_n_obs + 1];
879     o_par = new double[input_n_obs + 1];
880     o_funct = new double[input_nob_iter + 1];
881
882     double []g1 = new double[input_n_obs + 1];
883     double []g2 = new double[input_n_obs + 1];
884     double gc[] = new double[input_n_obs + 1];
885     double err[] = new double[input_n_obs + 1];
886
887     double [][]p = new double[8][9];
888     double [][]s = new double[8][input_n_obs + 1];
889     double []b = new double[8];
890
891     double []par = new double[8];
892     double []par1 = new double[8];
893     double []par2 = new double[8];
894     double []dupar = new double[8];
895
896     double []KS = new double[2];
897     double funct2 = 0;
898     double lambda = 0.5;
899     np = input_ndeg + 2;
900     double gmax = Math.abs(input_nob_gob[1]);
901
902     for (int k = 1; k <= input_n_obs; k++) {
903         if (Math.abs(input_nob_gob[k]) - gmax > 0)
904             gmax = Math.abs(input_nob_gob[k]);
905     }
906
907     double datum = input_nob_gob[1];
908     double r = input_nob_gob[input_n_obs] - input_nob_gob[1];
909     int kk = 1;
910     double gh = 0.5 * r;
911     kk = kk + 1;
912     double XH = 0;
913
914     while (((input_nob_gob[kk] - datum) / gh) - 1.0 < 0) {
915         kk = kk + 1;
916     }
917     if (((input_nob_gob[kk] - datum) / gh) - 1.0 > 0) {
918         XH = input_x_km[kk-1] + ( ( gh + datum - input_nob_gob[kk-1]) * ( input_x_km[k]
- input_x_km[kk-1] ) ) / ( input_nob_gob[kk] - input_nob_gob[kk-1] );
919     }
920     if (((input_nob_gob[kk] - datum) / gh) - 1.0 == 0) {
921         XH = input_x_km[kk];
922     }
923
924     double amax = - gmax;
925     if(input_lambda_val == 0)
926         par[1] = (amax) / ( 13.3333 * 3.14159265 * input_sd_poly);
927     else
928         par[1] = (1 / input_lambda_val) * Math.log(1 + ((input_lambda_val * amax) /
(13.3333 * 3.14159265 * input_sd_poly)));

```



```

929     par[2] = XH;
930     double dpar = 0.01;
931
932     INGREXP_NOREQ.getGF (input_n_obs, np, input_ndeg, input_ele_km, input_sd_poly,
input_lambda_val, input_x_km, par, gc);
933     double funct1 = 0;
934     for (int k = 1; k <= input_n_obs; k++) {
935         err[k] = input_nob_gob[k] - gc[k];
936         funct1 = funct1 + Math.pow( err[k] , 2 );
937     }
938     funct1 = Math.sqrt(funct1 / input_n_obs);
939     lambda = 0.5;
940     int NP1 = np + 1;
941     int IER = 1;
942
943     while (IER <= input_nob_iter) {
944         int ITER1 = IER ;
945         o_funct[ITER1] = funct1;
946         for (int K = 1; K <= np; K++) {
947             par1[K] = par[K];
948         }
949
950         for (int I = 1; I <= np; I++) {
951             par1[I] = par[I] + dpar / 2.0;
952             INGREXP_NOREQ.getGF (input_n_obs, np, input_ndeg, input_ele_km,
input_sd_poly, input_lambda_val, input_x_km, par1, g1);
953             par1[I] = par[I] - dpar / 2.0;
954             INGREXP_NOREQ.getGF (input_n_obs, np, input_ndeg, input_ele_km,
input_sd_poly, input_lambda_val, input_x_km, par1, g2);
955             for (int K = 1; K <= input_n_obs; K++) {
956                 s[I][K] = ( g1[K] - g2[K] ) / dpar;
957             }
958         }
959         for (int J = 1; J <= NP1; J++) {
960             for(int I = 1; I <= np; I++) {
961                 p[I][J] = 0.0;
962             }
963         }
964         for (int J = 1; J <= np; J++) {
965             for (int I = 1; I <= np; I++) {
966                 for (int K = 1; K <= input_n_obs; K++){
967                     p[I][J] = p[I][J] + s[I][K] * s[J][K];
968                 }
969             }
970         }
971
972         for (int J = 1; J <= np; J++) {
973             for (int K = 1; K <= input_n_obs; K++) {
974                 p[J][NP1] = p[J][NP1] + err[K] * s[J][K];
975             }
976         }
977
978         do {
979             double con = lambda + 1.0;
980             for (int I = 1; I <= np; I++) {
981                 dupar[I] = par[I];
982             }
983             for (int L = 1; L <= np; L++) {
984                 for (int J = 1; J <= np; J++) {
985                     if (L - J == 0)
986                         p[L][J] = p[L][J] * con;
987                 }
988             }
989             INGREXP_NOREQ.getNOREQ(p, b, np, KS);
990             for (int I = 1; I <= np; I++) {
991                 par2[I] = dupar[I] + b[I];
992             }
993
994             INGREXP_NOREQ.getGF (input_n_obs, np, input_ndeg, input_ele_km,
input_sd_poly, input_lambda_val, input_x_km, par2, gc);
995             funct2 = 0.0;
996             for (int K = 1; K <= input_n_obs; K++) {
997                 err[K] = input_nob_gob[K] - gc[K];
998                 funct2 = funct2 + Math.pow(err[K] , 2);
999             }

```

```

1000         funct2 = Math.sqrt(funct2 / input_n_obs);
1001
1002         if (funct1 - funct2 < 0) {
1003             lambda = lambda * 2.0;
1004             for (int i = 1; i <= np; i++) {
1005                 for (int j = 1; j <= np; j++) {
1006                     if (i - j == 0) {
1007                         p[i][j] = p[i][j] / con;
1008                     }
1009                 }
1010             }
1011         }
1012     } while (funct1 - funct2 < 0);
1013
1014     funct1 = funct2;
1015     IER++;
1016     for (int I = 1; I <= np; I++) {
1017         par[I] = par2[I];
1018     }
1019     o_iter = ITER1;
1020     o_func = funct2;
1021     for (int K = 1; K <= input_n_obs; K++) {
1022         o_GC[K] = gc[K];
1023         o_err[K] = err[K];
1024     }
1025     for (int l = 1; l <= np; l++) {
1026         o_par[l] = par[l];
1027     }
1028
1029     if ( funct2 < ALER || ITER1 == input_nob_iter || lambda -12 > 0 ) {
1030         o_iter = ITER1;
1031         o_func = funct2;
1032         for (int K = 1; K <= input_n_obs; K++) {
1033             o_GC[K] = gc[K];
1034             o_err[K] = err[K];
1035         }
1036         for (int l = 1; l <= np; l++) {
1037             o_par[l] = par[l];
1038         }
1039
1040         denCal(input_sd_poly, input_lambda_val);
1041         setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
1042 o_err, o_par, o_func, input_area_name);
1043         drawGraph();
1044     }
1045     else{
1046         denCal(input_sd_poly, input_lambda_val);
1047         setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
1048 o_err, o_par, o_func, input_area_name);
1049         drawGraph();
1050     }
1051
1052     //Thread.sleep(10);
1053
1054     if ( funct2 < ALER || lambda -12 > 0)
1055         break;
1056
1057     lambda = lambda / 2.0;
1058 }
1059
1060 public static void denCal(double sd, double la){
1061
1062     int i = 1;
1063     double z1 = 0.0001;
1064     double z2 = INGREXP_CalculateValues.o_par[1];
1065     vsd = new double[(int) Math.pow(input_n_obs, 2)];
1066     dep = new double[(int) Math.pow(input_n_obs, 2)];
1067     while(z1 <= z2){
1068         double dc = sd * Math.exp(-la * z1);
1069         vsd[i] = dc;
1070         dep[i] = z1;
1071         z1 = z1 + 0.1;
1072         i++;

```

```

1073     }
1074     count = i;
1075     vsd[count] = sd * Math.exp(-la * z2);
1076     dep[count] = z2;
1077
1078 }
1079
1080 public static void setGraphValues(int i_no_obs, int np, int ite, double []dis, double
[]gobs, double []gcal, double []error, double []PARA, double FUNCT, String Area_fe) {
1081
1082     obj = new Object[i_no_obs + 1][4];
1083
1084     DecimalFormat df = new DecimalFormat("0.###");
1085     DecimalFormat d = new DecimalFormat("0.##");
1086     for(int K = 1; K <= i_no_obs; K++){
1087         obj[K][0] = "" + dis[K];
1088         obj[K][1] = "" + df.format(gobs[K]);
1089         obj[K][2] = "" + df.format(gcal[K]);
1090         obj[K][3] = "" + df.format(error[K]);
1091     }
1092
1093     obj[0][0] = "ITERATION";
1094     obj[0][1] = "=" + "" + ite;
1095
1096     INGREXP_TableView.val.setText("");
1097     INGREXP_TableView.val.append("ITERATION NUMBER =" + ite + "\n");
1098     INGREXP_TableView.val.appendText("\n");
1099     INGREXP_TableView.val.append("INTERPRETED PARAMETERS:-\n");
1100     INGREXP_TableView.val.appendText("-----\n");
1101     INGREXP_TableView.val.appendText("\n");
1102     INGREXP_TableView.val.appendText("DEPTH TO THE BOTTOM OF THE FAULT =" +
d.format(PARA[1]) + "(km)\n");
1103     INGREXP_TableView.val.appendText("\n");
1104     INGREXP_TableView.val.appendText("OBJECTIVE FUNCION  =" + df.format(FUNCT) + "\n");
1105     INGREXP_TableView.val.appendText("\n");
1106
1107     INGREXP_TableView.val.appendText("\n");
1108     INGREXP_TableView.val.appendText("COEFFICIENTS OF THE POLYNOMIAL:-");
1109     INGREXP_TableView.val.appendText("\n");
1110     INGREXP_TableView.val.appendText("-----
----\n");
1111     INGREXP_TableView.val.appendText(df.format(PARA[2]) + "\n");
1112     INGREXP_TableView.val.appendText(df.format(PARA[3]) + "\n");
1113     INGREXP_TableView.val.appendText(df.format(PARA[4]) + "\n");
1114
1115 }
1116 public static void drawGraph(){
1117     final com.ingrexp.view.INGREXP_DrawGraph dg = new
com.ingrexp.view.INGREXP_DrawGraph();
1118     try
1119     {
1120         int width = 1280;
1121         int height = 650;
1122         BufferedImage buffer = new
BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
1123         Graphics gl= buffer.createGraphics();
1124         gl.setColor(Color.WHITE);
1125         gl.fillRect(0,0,width,height);
1126         Graphics2D g2 = (Graphics2D)gl ;
1127         dg.plot(g2);
1128         dg.plotXYCoordinates(g2);
1129         dg.drawGraph(g2);
1130         dg.drawOBJ(g2);
1131         dg.plotZCoordinates(g2);
1132         //dg.plotXYCoordinates(g2);
1133         dg.drawSd(g2);
1134         dg.plot(g2);
1135         dg.idex(g2);
1136
1137         FileOutputStream os = new FileOutputStream(
INGREXP_CalculateValues.input_area_name + ".jpg");
1138         ImageIO.write(buffer, "jpg", os);
1139         os.close();
1140
1141         String path = INGREXP_CalculateValues.input_area_name + ".jpg";

```

```

1142         image = ImageIO.read(new File(path));
1143
1144         Graphics g_image = INGREXP_MainPanel.img.getGraphics();
1145         g_image.drawImage(image, -60, -30, image.getWidth(), image.getHeight(), dg);
1146         MouseListener ml3 = new MouseAdapter() {
1147             public void mouseClicked(MouseEvent e) {
1148                 Graphics g_image = INGREXP_MainPanel.img.getGraphics();
1149                 g_image.drawImage(image, -60, -30, image.getWidth(),
image.getHeight(), dg);
1150             }
1151         };
1152         INGREXP_MainPanel.img.addMouseListener(ml3);
1153     }
1154     catch (Exception e2) {
1155
1156         // e2.printStackTrace();
1157     }
1158 }
1159 }
1160 -----

```

```

1161 package com.ingrexp.model;
1162
1163 public class INGREXP_NOREQ {
1164     public static double []vsd = null;
1165     public static double []dep = null;
1166     public static int N2 = 0;
1167
1168     public static void main(String[] args) {
1169
1170         //Methods that support the main class
1171     }
1172
1173
1174
1175     public static double []getGF(int n,int NNP,int deg,double ele[],double sd,double
lambda,double []x,double par[],double []GC) {
1176
1177         double Z1 = 0.00;
1178         double []Z ;
1179         double []GS ;
1180         double GGC = 0;
1181         double tsum;
1182
1183         for (int K = 1; K <= n; K++ ) {
1184             GC[K] = 0;
1185         }
1186         double DX = ( x[2] - x[1] ) / 10;
1187         double ZB = par[1] - Z1;
1188
1189         int ND = (int)( ZB / DX ) + 1;
1190         int N1 = ND / 2;
1191         if (ND - ( 2 * N1 ) < 0 || ND - ( 2 * N1 ) > 0) {
1192             ND = ND + 1;
1193         }
1194         double DZ = ZB / ND;
1195         N2 = ND + 1;
1196         Z = new double[N2 + 1];
1197         GS = new double[N2 + 1];
1198         vsd = new double[N2 + 1];
1199         dep = new double[N2 + 1];
1200         for (int JZ = 1; JZ <= N2; JZ++) {
1201             Z[JZ] = Z1 + DZ * ( JZ - 1 );
1202         }
1203         for (int K = 1; K <= n; K++) {
1204             double XX = x[K] ;
1205             for ( int JZ = 1; JZ <= N2; JZ++) {
1206                 double dc = sd * Math.exp(-lambda*Z[JZ]);
1207                 vsd[JZ] = dc;
1208                 dep[JZ] = Z[JZ];
1209                 tsum = 0;
1210                 for(int kk = 2; kk <= deg + 2; kk++){
1211                     tsum = tsum + par[kk] * Math.pow(Z[JZ],(kk - 2));
1212                 }
1213

```

```

1214         double tr1 = 3.14159265 / 2.0;
1215         double tr2 = Math.atan( ( ( -XX + tsum) ) / ( ( Z[JZ] - ele[K] ) ) );
1216         GS[JZ] = 13.3333 * dc * (tr1 - tr2);
1217
1218     }
1219     GC[K] = getSIMP(GS, Z, N2, GGC);
1220 }
1221 return GC;
1222 }
1223 public static double getSIMP(double []gs,double []z,int n,double ggc) {
1224
1225     double dz = z[2] - z[1];
1226     double sum1 = 0.0;
1227     double sum2 = 0.0;
1228     int n1 = n / 2;
1229     int n4 = n1 - 1;
1230     for(int I = 1; I <= n1; I++) {
1231         int n2 = 2 * I;
1232         sum1 = sum1 + gs[n2];
1233     }
1234     for(int I = 1; I <= n4; I++) {
1235         int n3 = 2 * I + 1;
1236         sum2 = sum2 + gs[n3];
1237     }
1238     ggc = gs[1] + 4 * sum1 + 2 * sum2 + gs[n];
1239     ggc = ggc * dz / 3.0;
1240     return ggc;
1241 }
1242 public static double []getNOREQ(double p[][], double b[], int n, double KS[]) {
1243
1244     int I = n + 1;
1245     double []a = new double[n * n + 1];
1246     for (int I1 = 1; I1 <= n; I1++) {
1247         for (int I2 = 1; I2 <= n; I2++) {
1248
1249             int I3 = (I1 - 1) * n + I2;
1250             a[I3] = p[I2][I1];
1251         }
1252     }
1253
1254     for (int I4 = 1; I4 <= n; I4++) {
1255         b[I4] = p[I4][I];
1256     }
1257     double TOL = 0;
1258     KS[0] = 0;
1259     int JJ = - n;
1260     int IT;
1261     int NY = 0;
1262     for (int J = 1; J <= n; J++) {
1263         int JY = J + 1;
1264         JJ = JJ + n + 1;
1265         double biga = 0;
1266         IT = JJ - J;
1267         int imax = 0;
1268         for (int i = J; i <= n; i++) {
1269             int IJ = IT + i;
1270             if (Math.abs(biga) - Math.abs(a[IJ]) < 0) {
1271                 biga = a[IJ];
1272                 imax = i;
1273             }
1274         }
1275         int I1 = 0;
1276         if (Math.abs(biga) - TOL <= 0) {
1277             KS[1] = 1;
1278             return KS;
1279         }
1280         else {
1281             I1 = J + n * (J - 2);
1282             IT = imax - J;
1283         }
1284         double save;
1285         for (int K = J; K <= n; K++) {
1286             I1 = I1 + n;
1287             int I2 = I1 + IT;
1288             save = a[I1];

```

```

1289         a[I1] = a[I2];
1290         a[I2] = save;
1291         a[I1] = a[I1] / biga;
1292     }
1293     save = b[imax];
1294     b[imax] = b[J];
1295     b[J] = save / biga;
1296     int IQS = 0;
1297     if (J - n < 0 || J - n > 0) {
1298         IQS = n * (J - 1);
1299         for (int IX = JY; IX <= n; IX++) {
1300             int IXJ = IQS + IX;
1301             IT = J - IX;
1302             for (int JX = JY; JX <= n; JX++) {
1303                 int IXJX = n * (JX - 1) + IX;
1304                 int JJX = IXJX + IT;
1305                 a[IXJX] = a[IXJX] - (a[IXJ] * a[JJX]);
1306             }
1307             b[IX] = b[IX] - (b[J] * a[IXJ]);
1308         }
1309     }
1310 }
1311 NY = n - 1;
1312 IT = n * n;
1313 for (int J = 1; J <= NY; J++) {
1314     int ia = IT - J;
1315     int ib = n - J;
1316     int ic = n;
1317     for (int K = 1; K <= J; K++) {
1318         b[ib] = b[ib] - a[ia] * b[ic];
1319         ia = ia - n;
1320         ic = ic - 1;
1321     }
1322 }
1323 return b;
1324 }
1325 }
1326
-----

```

```

1327 package com.ingrexp.control;
1328
1329 import java.awt.Color;
1330 import java.awt.Graphics;
1331 import java.awt.event.ActionEvent;
1332 import java.awt.event.ActionListener;
1333 import java.io.File;
1334 import java.io.FileWriter;
1335 import java.io.IOException;
1336 import java.text.DecimalFormat;
1337
1338 import javax.swing.JFileChooser;
1339 import javax.swing.JFrame;
1340 import javax.swing.JOptionPane;
1341
1342 import com.ingrexp.model.INGREXP_CalculateValues;
1343 import com.ingrexp.view.INGREXP_MainPanel;
1344 import com.ingrexp.view.INGREXP_TableView;
1345
1346 public class INGREXP_Controller implements ActionListener {
1347     String rowdata[][] = {};
1348     com.ingrexp.model.INGREXP_CalculateValues cv = new
1349 com.ingrexp.model.INGREXP_CalculateValues();
1350     FileWriter myWriter = null;
1351     public static boolean success = false;
1352
1353     public void actionPerformed(ActionEvent ae) {
1354         if(ae.getActionCommand().equals("Interpretation")) {
1355
1356             com.ingrexp.view.INGREXP_TableView.populateEastPanel(rowdata);
1357             INGREXP_TableView.val.setText("");
1358             cv.getAnamolyValues(com.ingrexp.view.INGREXP_MainPanel.captureValues());
1359             com.ingrexp.view.INGREXP_TableView.populateEastPanel(INGREXP_CalculateValues.ob
1360 );

```

```

1361         INGREXP_CalculateValues.obj = null;
1362         com.ingrexp.view.INGREXP_MainPanel.p_East.repaint();
1363         com.ingrexp.view.INGREXP_MainView mv = new com.ingrexp.view.INGREXP_MainView();
1364         mv.setResizable(true);
1365
1366     } else if(ae.getActionCommand().equals("Save & Print")){
1367
1368         try{
1369             String current = System.getProperty("user.dir");
1370             File img_file = new File( INGREXP_CalculateValues.input_area_name+".jpg");
1371             JFileChooser saveFile = new JFileChooser(current);
1372             File OutFile = saveFile.getSelectedFile();
1373             if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
1374             {
1375                 OutFile = saveFile.getSelectedFile();
1376
1377                 if (OutFile.canWrite() || !OutFile.exists())
1378                 {
1379                     File dir = new File(OutFile.getParent());
1380                     success = img_file.renameTo(new File(dir,img_file.getName()));
1381                     System.out.println("save successful" + success);
1382                     myWriter = new FileWriter(OutFile+".html");
1383                     myWriter.write(" </table> </td> <td> <img src = '"+
INGREXP_CalculateValues.input_area_name
1384                     +".jpg'></td></tr></table>");
1385                     myWriter.write("<html><Body onLoad = \"window.print()\"> <table>
<tr> <td>" +
1386                                     "<table border = 1> <tr> <th colspan = 4>LOCATION:-
"+INGREXP_CalculateValues.input_area_name+"</th> </tr>");
1387
1388                                     DecimalFormat df =new DecimalFormat("0.###");
1389                                     myWriter.write(" <tr><th colspan = 4> PROFILE NUMBER:-"+"
"+INGREXP_CalculateValues.input_profile_num+" </th></tr>");
1390                                     myWriter.write(" <tr><th colspan = 4> ITERATION"+"
"+INGREXP_CalculateValues.o_iter+" </th></tr>");
1391
1392                                     myWriter.write("<tr > <th>Distance (km) </th> <th> Observed
anomalies (mGal) </th> <th> Calculated anomalies (mGal) </th> <th>
Error (mGal) </th></tr>");
1393
1394                                     for ( int K = 1; K <= INGREXP_CalculateValues.input_n_obs; K++){
1395
1396                                         myWriter.write("<tr> <td>" +
INGREXP_CalculateValues.input_x_km[K]+ "</td>
<td>"+df.format(INGREXP_CalculateValues.input_nob_gob[K])+"</td>
<td>"+df.format(INGREXP_CalculateValues.o_GC[K])+"</td>
<td>"+df.format(INGREXP_CalculateValues.o_err[K])+"</td></tr>");
1397                                     }
1398                                     // myWriter.write(" </table> </td> <td> <img src = '"+
INGREXP_CalculateValues.input_area_name+INGREXP_CalculateValues.inpu
t_profile_num +".jpg'></td></tr></table><BR> Interpreted Parameters
:<BR>");
1399
1400                                     myWriter.write("</table>");
1401                                     myWriter.write("INTERPRETED PARAMETERS: <BR>");
1402                                     myWriter.write("----- <BR>");
1403                                     DecimalFormat d =new DecimalFormat("0.##");
1404                                     DecimalFormat d1 =new DecimalFormat("0.###");
1405
1406                                     myWriter.write("DEPTH TO THE BOTTOM OF THE FAULT ="+"
"+d.format(INGREXP_CalculateValues.o_par[1])+"(km) <BR>");
1407                                     myWriter.write("OBJECTIVE FUNCTION ="+"
"+d1.format(INGREXP_CalculateValues.o_func)+"<BR>");
1408                                     myWriter.write("<BR>");
1409
1410                                     for ( int i = 2; i <= 4; i++) {
1411                                         if ( i == 2 ) {
1412                                             myWriter.write("COEFFICIENTS OF THE POLYNOMIAL:-"+"<BR>");
1413                                             myWriter.write(d1.format(INGREXP_CalculateValues.o_par[i])+"<BR>");
1414                                         }
1415                                     }
1416                                     myWriter.close();
1417             }
1418         }

```

```

1417         }
1418         else
1419         {
1420             //pops up error message
1421         }
1422     }
1423     catch(Exception e1) {
1424
1425         e1.printStackTrace();
1426     }
1427
1428 }else if(ae.getActionCommand().equals("Load data")){
1429     try {
1430         INGREXP_MainPanel.loadData1();
1431     } catch (IOException e) {
1432         // TODO Auto-generated catch block
1433         e.printStackTrace();
1434     }
1435 }else if(ae.getActionCommand().equals("Clear")){
1436
1437     INGREXP_MainPanel.clearDefaultValues();
1438     com.ingrexp.view.INGREXP_MainPanel.clearPanel(INGREXP_MainPanel.p_Center);
1439     com.ingrexp.view.INGREXP_TableView.populateEastPanel(rowdata);
1440     INGREXP_TableView.val.setText("");
1441     Graphics g = INGREXP_MainPanel.img.getGraphics();
1442     g.setColor(Color.white);
1443     g.fillRect(0, 0, 1000, 600);
1444
1445 }else if(ae.getActionCommand().equals("Exit")){
1446     JFrame frame = null;
1447     int r = JOptionPane.showConfirmDialog(
1448         frame,
1449         "Exit INGREXP ?",
1450         "Confirm Exit ",
1451         JOptionPane.YES_NO_OPTION);
1452     if(r == JOptionPane.YES_OPTION ){
1453         if(success==false){
1454             String fileName = INGREXP_CalculateValues.input_area_name+".jpg";
1455             File f = new File(fileName);
1456             f.delete();
1457         }
1458         System.exit(0);
1459     }
1460 }
1461 }
1462 }
1463

```

---

```

1464 package com.ingrexp.util;
1465
1466 import javax.swing.JFrame;
1467 import javax.swing.JOptionPane;
1468
1469 import com.ingrexp.model.INGREXP_CalculateValues;
1470
1471 public class INGREXP_Utility {
1472     public static double convertDouble(String str) throws Exception {
1473
1474
1475         Double temp = null;
1476
1477         try {
1478             temp = new Double(str.trim());
1479         }
1480         catch(Exception e){
1481             JFrame frame = null;
1482             JOptionPane.showMessageDialog(frame,
1483                 "Enter a numerical value.",
1484                 "Number format error",
1485                 JOptionPane.ERROR_MESSAGE);
1486         }
1487         return temp.doubleValue();
1488     }
1489 }
1490

```



```

1491 public static String convertString(String str) throws Exception {
1492
1493     String temp = new String(str.trim());
1494     return temp;
1495 }
1496
1497 public static int convertInteger(String str) throws Exception {
1498     Integer temp = null;
1499     try {
1500         temp = new Integer(str.trim());
1501     }
1502     catch(Exception e){
1503
1504         JFrame frame = null;
1505         JOptionPane.showMessageDialog(frame,
1506             "Enter a numerical value.",
1507             "Number format error",
1508             JOptionPane.ERROR_MESSAGE);
1509
1510     }
1511     return temp.intValue();
1512 }
1513
1514 public static double findMaximumNumber( double observe[]) {
1515
1516     double max = 0.0d;
1517     for (int i = 0; i < observe.length; i++) {
1518
1519         if (Math.abs(observe[i]) > Math.abs(max)) {
1520
1521             max = observe[i];
1522         }
1523     }
1524
1525     double maxVal = max/3*5;
1526     return maxVal;
1527 }
1528
1529 public static double findMinimumNumber( double observe[], double denVal) {
1530
1531     double max = denVal;
1532     for (int i = 1; i < observe.length; i++) {
1533
1534         if (Math.abs(observe[i]) < Math.abs(max)) {
1535
1536             max = Math.abs(observe[i]);
1537         }
1538     }
1539
1540     double maxVal = max;
1541     return maxVal;
1542 }
1543
1544 public static double findMinimumNumber1( double observe[]) {
1545
1546     double max = 0.0d;
1547     for (int i = 1; i < observe.length; i++) {
1548
1549         if ((observe[i]) < (max)) {
1550
1551             max = (observe[i]);
1552         }
1553     }
1554
1555     double maxVal = max;
1556     return maxVal;
1557 }
1558
1559 public static double findMaximumNumber1( double observe[]) {
1560
1561     double max = 0.0d;
1562     for (int i = 1; i < observe.length; i++) {
1563
1564         if (Math.abs(observe[i]) > Math.abs(max)) {
1565

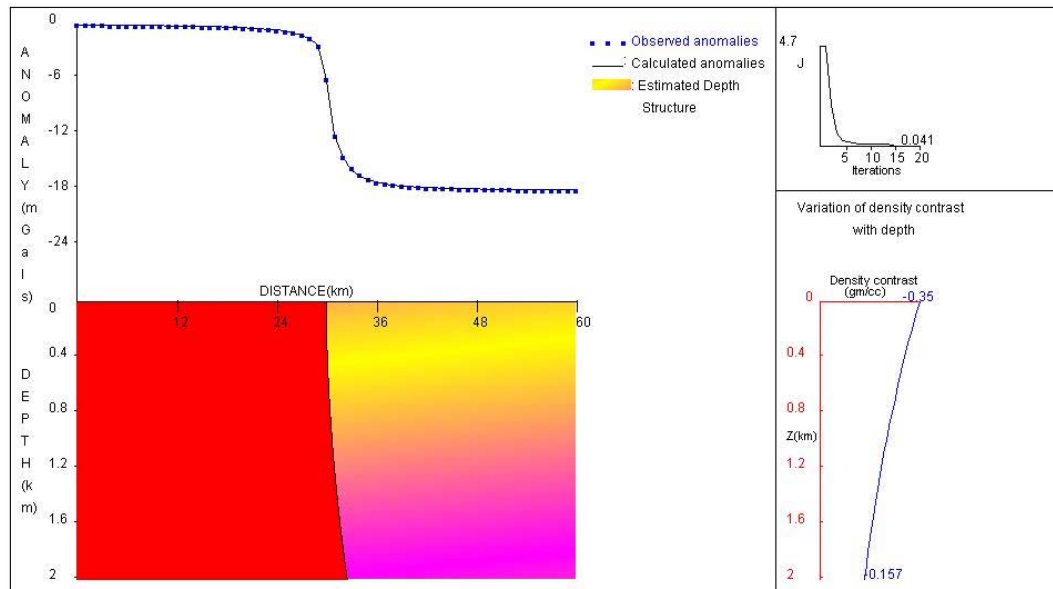
```

```

1566         max =Math.abs(observe[i]);
1567     }
1568 }
1569
1570 double maxVal = max;
1571 return maxVal;
1572 }
1573 public static double findMaximumNumber( double observe[], double anoVal) {
1574
1575     double max = anoVal;
1576     for (int i = 1; i < observe.length; i++) {
1577
1578         if ((observe[i]) > (max)) {
1579
1580             max = (observe[i]);
1581         }
1582     }
1583
1584     double maxVal = max;
1585     return maxVal;
1586 }
1587 public static int findMaximumNumber( double observe) {
1588
1589     double max = 0.0d;
1590     int maxVal=0;
1591     max = observe;
1592
1593     if (max < 5) {
1594
1595         maxVal = 5;
1596     }
1597     else if (max >= 5 && max <= 10) {
1598
1599         maxVal = 10;
1600     }
1601     else if ( max > 10 && max <= 15) {
1602
1603         maxVal = 15;
1604     }
1605     else if (max > 15 && max <= 20) {
1606
1607         maxVal = 20;
1608     }
1609     else
1610     {
1611         maxVal = INGREXP_CalculateValues.o_iter;
1612     }
1613     return maxVal;
1614 }
1615
1616
1617 public static double[] convertDoubleArray(String str) throws Exception {
1618
1619     java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
1620     String temp = "";
1621     java.util.ArrayList arr = new java.util.ArrayList();
1622
1623     while(st.hasMoreTokens()) {
1624
1625         temp = st.nextToken();
1626         arr.add(temp);
1627     }
1628     double d_array[] = new double[arr.size() + 1];
1629
1630     for (int i = 0; i <= arr.size(); i++) {
1631
1632         if (i == 0)
1633             d_array[i] = 0.0;
1634         else {
1635
1636             try {
1637                 d_array[i] = convertDouble( arr.get(i - 1).toString() );
1638             }
1639             catch(Exception e){
1640                 JFrame frame = null;

```

```
1641             JOptionPane.showMessageDialog(frame,
1642             "Enter numerical values.",
1643             "Number format error",
1644             JOptionPane.ERROR_MESSAGE);
1645         }
1646     }
1647 }
1648 return d_array;
1649 }
1650 }
```



LOCATION:- sample			
PROFILE NUMBER:- 1			
ITERATION 14			
Distance (km)	Observed anomalies (mGal)	Calculated anomalies (mGal)	Error (mGal)
0.0	-0.178	-0.179	0.001
1.0	-0.184	-0.184	0
2.0	-0.191	-0.191	-0
3.0	-0.198	-0.198	-0
4.0	-0.205	-0.205	-0
5.0	-0.212	-0.213	0.001
6.0	-0.221	-0.221	0
7.0	-0.23	-0.23	0
8.0	-0.24	-0.24	0
9.0	-0.251	-0.251	-0
10.0	-0.263	-0.263	-0
11.0	-0.276	-0.276	-0
12.0	-0.29	-0.29	0
13.0	-0.306	-0.306	0
14.0	-0.324	-0.324	-0
15.0	-0.344	-0.344	-0
16.0	-0.366	-0.367	0.001
17.0	-0.392	-0.392	0
18.0	-0.422	-0.422	0
19.0	-0.456	-0.457	0.001
20.0	-0.497	-0.497	0
21.0	-0.546	-0.546	0
22.0	-0.605	-0.606	0.001
23.0	-0.679	-0.68	0.001
24.0	-0.774	-0.774	0
25.0	-0.9	-0.9	-0
26.0	-1.076	-1.074	-0.002
27.0	-1.339	-1.333	-0.006
28.0	-1.777	-1.762	-0.015
29.0	-2.672	-2.63	-0.042
30.0	-6.614	-6.817	0.203
31.0	-13.512	-13.335	-0.177
32.0	-15.998	-16.01	0.012
33.0	-17.367	-17.474	0.107
34.0	-18.184	-18.278	0.094
35.0	-18.682	-18.739	0.057
36.0	-18.995	-19.026	0.031
37.0	-19.204	-19.219	0.015
38.0	-19.351	-19.357	0.006
39.0	-19.459	-19.461	0.002

40.0	-19.543	-19.541	-0.002
41.0	-19.609	-19.605	-0.004
42.0	-19.663	-19.657	-0.006
43.0	-19.707	-19.701	-0.006
44.0	-19.745	-19.737	-0.008
45.0	-19.777	-19.769	-0.008
46.0	-19.804	-19.796	-0.008
47.0	-19.828	-19.82	-0.008
48.0	-19.85	-19.841	-0.009
49.0	-19.869	-19.859	-0.01
50.0	-19.885	-19.876	-0.009
51.0	-19.9	-19.891	-0.009
52.0	-19.914	-19.904	-0.01
53.0	-19.926	-19.917	-0.009
54.0	-19.938	-19.928	-0.01
55.0	-19.948	-19.938	-0.01
56.0	-19.958	-19.948	-0.01
57.0	-19.966	-19.956	-0.01
58.0	-19.974	-19.964	-0.01
59.0	-19.982	-19.972	-0.01
60.0	-19.989	-19.979	-0.01

INTERPRETED PARAMETERS:

-----  
DEPTH TO THE BOTTOM OF THE FAULT = 2(km)  
OBJECTIVE FUNCTION = 0.041

COEFFICIENTS OF THE POLYNOMIAL:-  
30  
0.184  
0.544

```

1 package com.frgmlstrk.view;
2
3 import java.awt.*;
4 import java.awt.event.WindowAdapter;
5 import java.awt.event.WindowEvent;
6 import java.io.File;
7
8 import javax.swing.JFrame;
9 import javax.swing.JOptionPane;
10
11 import com.frgmlstrk.control.FRGMSTRK_Controller;
12 import com.frgmlstrk.model.FRGMSTRK_CalculateValues;
13
14 public class FRGMSTRK_MainView extends JFrame{
15
16     /**
17      *
18      */
19     private static final long serialVersionUID = 1L;
20
21     public static void main(String s[])
22     {
23         FRGMSTRK_MainView cm = new FRGMSTRK_MainView();
24         cm.setSize(1280, 768);
25         cm.addWindowListener(new WindowAdapter(){
26             public void windowClosing(WindowEvent e){
27                 JFrame frame = null;
28                 int r = JOptionPane.showConfirmDialog(
29                     frame,
30                     "Exit FRGMSTRK ?",
31                     "Confirm Exit ",
32                     JOptionPane.YES_NO_OPTION);
33                 if(r == JOptionPane.YES_OPTION ){
34                     if(FRGMSTRK_Controller.success == false){
35                         String fileName = FRGMSTRK_CalculateValues.input_area_name+".jpg";
36                         File f = new File(fileName);
37                         f.delete();
38                     }
39                     System.exit(0);
40                 }
41             }
42         });
43         cm.setTitle("FRGMSTRK");
44         cm.setResizable(true);
45         cm.add(new FRGMSTRK_MainPanel(cm));
46         cm.setVisible(true);
47     }
48 }
49
50 }
51
-----

```

```

52 package com.frgmlstrk.view;
53
54 import java.awt.*;
55 import java.io.BufferedReader;
56 import java.io.File;
57 import java.io.FileReader;
58 import java.util.HashMap;
59 import javax.swing.JFileChooser;
60 import com.frgmlstrk.control.FRGMSTRK_Controller;
61 import com.frgmlstrk.model.FRGMSTRK_CalculateValues;
62 import com.frgmlstrk.util.FRGMSTRK_Utility;
63 import com.frgmlstrk.view.event.FRGMSTRK_PlotDensity;
64 import com.frgmlstrk.view.event.FRGMSTRK_PlotDepth;
65 import com.frgmlstrk.view.event.FRGMSTRK_PlotFault;
66
67 public class FRGMSTRK_MainPanel extends Panel {
68
69     /**
70      *
71      */
72     private static final long serialVersionUID = 1L;
73     public static JTextArea img = new JTextArea(46,140);
74

```

```

75     public static TextArea img1 = new TextArea(46,140);
76     public static TextArea img2 = new TextArea(46,140);
77     public static TextArea img3 = new TextArea(46,140);
78     public static TextArea img4 = new TextArea(46,140);
79     public static TextArea fl = new TextArea(46,140);
80     public static TextArea den = new TextArea(46,140);
81     public static TextArea im = new TextArea(46,140);
82
83     Panel p_North, p_West,p_South;
84     public static Panel p_East;
85     public static Label graphLabel;
86     public static Panel p_Center;
87     public static TextField inputValues [] = new TextField[18];
88
89     Button actionButton[] = new Button[12];
90     Object rowdata[][]={};
91
92     /**Field Area Name*/
93     public final static int AREA_FE = 0;
94     /**Number of the Profile*/
95     public final static int NUM_PROFILE = 1;
96     /**Strike of the structure(km)*/
97     public static final int STR_ST=2 ;
98     /**Number of density interfaces*/
99     public static final int NOB_DI=3;
100    /**Maximum depth(km)*/
101    public static final int MAX_DEP=4;
102    /**Basement density(gm/cc)*/
103    public static final int BASE_DEN=5;
104    /**Number of observation*/
105    public static final int N_OBS = 6 ;
106    /**Offset of the profile(km)*/
107    public static final int OFF_PRO=7;
108    /**Distances(km)*/
109    public static final int DIS_KM=8;
110    /**Elevation(km)*/
111    public static final int ELE_KM=9;
112    /**Observed anomaly*/
113    public static final int OBS_ANO=10;
114    /** Degree of polynomial*/
115    public static final int D_POLY=11;
116    /** Maximum Density*/
117    public static final int MAX_DEN=12;
118    /** Minimum Density*/
119    public static final int MIN_DEN=13;
120
121    public FRGMLSTRK_MainPanel(FRGMLSTRK_MainView cm){
122
123        this.setLayout(new BorderLayout());
124        p_North = new Panel();
125        p_West = new Panel();
126        p_East = new Panel ();
127        p_South = new Panel();
128        p_Center = new Panel();
129
130        graphLabel = new Label("Interactive Gravity Modeling of Strike Limited Listric Fau'
Sources", Label.CENTER);
131        graphLabel.setFont(new Font("Arial", 40, 20));
132        p_Center.add(graphLabel);
133
134        for(int i = 0; i < 15; i++){
135            inputValues[i] = new TextField();
136        }
137
138        actionButton[0] = new Button("Specify fault coordinates");
139        actionButton[1] = new Button("Draw/Edit fault plane");
140        actionButton[2] = new Button("Specify depth interfaces");
141        actionButton[3] = new Button("Specify density values");
142        actionButton[4] = new Button("Forward Modeling");
143        actionButton[5] = new Button("Graph");
144        actionButton[6] = new Button("Save and Print");
145        actionButton[7] = new Button("Sample data");
146        actionButton[8] = new Button("Clear");
147        actionButton[9] = new Button("Save file");
148        actionButton[10] = new Button("Load file");

```

```

149     actionBar[11] = new Button("Exit");
150
151     this.populateNorthPanel();
152     FRGMLSTRK_TableView.populateEastPanel(rowdata);
153
154     this.add(p_North, BorderLayout.NORTH);
155     p_Center.setSize(1000, 760);
156     this.add(p_Center, BorderLayout.CENTER);
157     p_Center.add(img);
158     this.add(p_East, BorderLayout.EAST);
159     MenuBar mb = new MenuBar();
160     cm.setMenuBar(mb);
161     Menu file = new Menu("File");
162     Menu file2 = new Menu("Fault");
163     Menu file3 = new Menu("Depth");
164     Menu file4 = new Menu("Interpretation");
165     MenuItem i1,i2,i3,i5,i6,i7,i8,i9,i10,i11,i12,i13;
166     file.add(i1 = new MenuItem("New"));
167     file.add(i2 = new MenuItem("Load file"));
168     file.add(i3 = new MenuItem("Save file"));
169     file.add(i5 = new MenuItem("Clear"));
170     file.add(i6 = new MenuItem("Exit"));
171
172     file2.add(i7 = new MenuItem("Specify fault coordinates"));
173     file2.add(i8 = new MenuItem("Draw/Edit fault plane"));
174
175     file3.add(i9 = new MenuItem("Specify depth interfaces"));
176     file3.add(i10 = new MenuItem("Specify density values"));
177
178     file4.add(i11 = new MenuItem("Forward Modeling"));
179     file4.add(i12 = new MenuItem("Graph"));
180     file4.add(i13 = new MenuItem("Save and Print"));
181     mb.add(file);
182     mb.add(file2);
183     mb.add(file3);
184     mb.add(file4);
185     i1.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
186     i2.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
187     i3.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
188     i5.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
189     i6.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
190     i7.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
191     i8.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
192     i9.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
193     i10.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
194     i11.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
195     i12.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
196     i13.addActionListener(new com.frgmlstrk.control.FRGMLSTRK_Controller());
197
198
199
200
201     this.setVisible(true);
202 }
203
204 public void populateNorthPanel(){
205     p_North.setLayout(new GridLayout(5,6));
206
207     p_North.add(new Label("Area Name"));
208     p_North.add(inputValues[0]);
209
210     p_North.add(new Label("Profile Name"));
211     p_North.add(inputValues[1]);
212
213     p_North.add(new Label("Half strike length (km)"));
214     p_North.add(inputValues[2]);
215
216     p_North.add(new Label("Number of density interfaces"));
217     p_North.add(inputValues[3]);
218
219     p_North.add(new Label("Depth to the basement(km)"));
220     p_North.add(inputValues[4]);
221
222     p_North.add(new Label("Basement density(gm/cc)"));
223     p_North.add(inputValues[5]);

```



```

224
225 p_North.add(new Label("Number of observations"));
226 p_North.add(inputValues[6]);
227
228 p_North.add(new Label("Offset of the profile(km)"));
229 p_North.add(inputValues[7]);
230
231 p_North.add(new Label("Distance(km)"));
232 p_North.add(inputValues[8]);
233
234 p_North.add(new Label("Elevation(km)"));
235 p_North.add(inputValues[9]);
236
237 p_North.add(new Label("Observed anomaly"));
238 p_North.add(inputValues[10]);
239
240 p_North.add(new Label("Degree of polynomial"));
241 p_North.add(inputValues[11]);
242
243 p_North.add(new Label("Maximum density"));
244 p_North.add(inputValues[12]);
245
246 p_North.add(new Label("Minimum density"));
247 p_North.add(inputValues[13]);
248
249 p_North.add(actionButton[0]);
250 p_North.add(actionButton[1]);
251 p_North.add(actionButton[2]);
252 p_North.add(actionButton[3]);
253 p_North.add(actionButton[4]);
254 p_North.add(actionButton[5]);
255 p_North.add(actionButton[6]);
256 p_North.add(actionButton[7]);
257 p_North.add(actionButton[8]);
258 p_North.add(actionButton[9]);
259 p_North.add(actionButton[10]);
260 p_North.add(actionButton[11]);
261
262 actionPerformed[0].addActionListener(new FRGMLSTRK_Controller());
263 actionPerformed[1].addActionListener(new FRGMLSTRK_Controller());
264 actionPerformed[2].addActionListener(new FRGMLSTRK_Controller());
265 actionPerformed[3].addActionListener(new FRGMLSTRK_Controller());
266 actionPerformed[4].addActionListener(new FRGMLSTRK_Controller());
267 actionPerformed[5].addActionListener(new FRGMLSTRK_Controller());
268 actionPerformed[6].addActionListener(new FRGMLSTRK_Controller());
269 actionPerformed[7].addActionListener(new FRGMLSTRK_Controller());
270 actionPerformed[8].addActionListener(new FRGMLSTRK_Controller());
271 actionPerformed[9].addActionListener(new FRGMLSTRK_Controller());
272 actionPerformed[10].addActionListener(new FRGMLSTRK_Controller());
273 actionPerformed[11].addActionListener(new FRGMLSTRK_Controller());
274 }
275
276
277 public static HashMap captureValues(){
278
279     HashMap h_Map = new HashMap();
280     try {
281
282         h_Map.put("AREA_FE", inputValues[AREA_FE].getText());
283         h_Map.put("NUM_PROFILE", inputValues[NUM_PROFILE].getText());
284         h_Map.put("N_OBS", inputValues[N_OBS].getText());
285         h_Map.put("D_POLY", inputValues[D_POLY].getText());
286         h_Map.put("STR_ST", inputValues[STR_ST].getText());
287         h_Map.put("OFF_PRO", inputValues[OFF_PRO].getText());
288         h_Map.put("DIS_KM", inputValues[DIS_KM].getText());
289         h_Map.put("ELE_KM", inputValues[ELE_KM].getText());
290         h_Map.put("NOB_DI", inputValues[NOB_DI].getText());
291         h_Map.put("MAX_DEP", inputValues[MAX_DEP].getText());
292         h_Map.put("BASE_DEN", inputValues[BASE_DEN].getText());
293         h_Map.put("OBS_ANO", inputValues[OBS_ANO].getText());
294         h_Map.put("MAX_DEN", inputValues[MAX_DEN].getText());
295         h_Map.put("MIN_DEN", inputValues[MIN_DEN].getText());
296
297     }
298     catch (Exception e) {

```

[illegible]

```

364         }
365         if (count==4){
366             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.NOB_DI].set
367         ext( ""+st);
368             FRGMLSTRK_CalculateValues.inter =
FRGMLSTRK_Utility.convertInteger(st)+2;
369         }
370         if (count==5){
371             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.MAX_DEP].set
372         Text( ""+st);
373         }
374         if (count==6){
375             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.BASE_DEN].set
376         tText( ""+st);
377         }
378         if (count==7){
379             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.N_OBS].set
380         xt( ""+st);
381         }
382         if (count==8){
383             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.OFF_PRO].set
384         Text( ""+st);
385         }
386         if (count==9){
387             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.DIS_KM].set
388         ext( ""+st);
389         }
390         if (count==10){
391             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.ELE_KM].set
392         ext( ""+st);
393         }
394         if (count==11){
395             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.OBS_ANO].set
396         Text( ""+st);
397         }
398         if (count==12){
399             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.D_POLY].set
400         ext( ""+st);
401         }
402         if (count==13){
403             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.MAX_DEN].set
404         Text( ""+st);
405         }
406         if (count==14){
407             FRGMLSTRK_MainPanel.inputValues[FRGMLSTRK_MainPanel.MIN_DEN].set
408         Text( ""+st);
409         }
410         if (count==15){
411             FRGMLSTRK_PlotFault.val =
FRGMLSTRK_Utility.convertDoubleArray(st);
412         }
413         if (count==16){

```

```

426             FRGMLSTRK_PlotFault.val1 =
FRGMLSTRK_Utility.convertDoubleArray(st);
427
428         }
429         if (count==17){
430             FRGMLSTRK_CalculateValues.len =
FRGMLSTRK_Utility.convertInteger(st);
431
432         }
433         if (count==18){
434             FRGMLSTRK_PlotDepth.val2 =
FRGMLSTRK_Utility.convertDoubleArray(st);
435
436         }
437         if (count==19){
438             FRGMLSTRK_PlotDensity.val3 =
FRGMLSTRK_Utility.convertDoubleArray(st);
439
440         }
441         if (count==20){
442             FRGMLSTRK_CalculateValues.inter =
FRGMLSTRK_Utility.convertInteger(st);
443
444         }
445     }
446     catch(Exception e) {
447
448         e.printStackTrace();
449     }
450     st = br.readLine();
451     count++;
452 }
453 }
454 }
455 }
456 catch(Exception e){
457 }
458 }
459 }
460
461 public static void clearDefaultValues(){
462
463     inputValues[AREA_FE].setText("");
464     inputValues[NUM_PROFILE].setText("");
465     inputValues[N_OBS].setText("");
466     inputValues[D_POLY].setText("");
467     inputValues[STR_ST].setText("");
468     inputValues[OFF_PRO].setText("");
469     inputValues[DIS_KM].setText("");
470     inputValues[ELE_KM].setText("");
471     inputValues[NOB_DI].setText("");
472     inputValues[MAX_DEP].setText("");
473     inputValues[BASE_DEN].setText("");
474     inputValues[OBS_ANO].setText("");
475     inputValues[MAX_DEN].setText("");
476     inputValues[MIN_DEN].setText("");
477 }
478 }
479 }
480 -----
481
482 package com.frgmlstrk.view;
483
484 import java.awt.*;
485 import javax.swing.JScrollPane;
486 import javax.swing.JTable;
487
488 public class FRGMLSTRK_TableView extends Panel{
489
490     /**
491     *
492     */
493     private static final long serialVersionUID = 1L;
494

```

```

495     public static TextArea val = new TextArea(5,5);
496     public static void populateEastPanel(Object rowData[][]) {
497
498         com.frgmlstrk.view.FRGMSTRK_MainPanel.p_East.removeAll();
499         com.frgmlstrk.view.FRGMSTRK_MainPanel.p_East.setLayout(new GridLayout(2,1));
500
501         Object columnNames[] = {"Distance(km)", "Observed anomalies (mGal)", "Calculated
anomalies (mGal)"};
502         JTable table = new JTable(rowData, columnNames);
503         table.setPreferredScrollableViewportSize(new Dimension(300,500));
504         JScrollPane scrollPane = new JScrollPane(table);
505         scrollPane.setAutoscrolls(true);
506         com.frgmlstrk.view.FRGMSTRK_MainPanel.p_East.add(scrollPane);
507         val.setEditable(false);
508         com.frgmlstrk.view.FRGMSTRK_MainPanel.p_East.add(val);
509         com.frgmlstrk.view.FRGMSTRK_MainPanel.p_East.validate();
510         com.frgmlstrk.view.FRGMSTRK_MainPanel.p_East.setVisible(true);
511     }
512 }
513 -----
514 package com.frgmlstrk.view;
515
516 import java.awt.*;
517 import java.applet.*;
518 import java.awt.geom.Line2D;
519 import java.awt.geom.Rectangle2D;
520 import java.text.DecimalFormat;
521 import com.frgmlstrk.model.FRGMSTRK_CalculateValues;
522 import com.frgmlstrk.util.FRGMSTRK_Utility;
523 import com.frgmlstrk.view.graph.FRGMSTRK_DensityGraph;
524
525
526
527 public class FRGMSTRK_DrawGraph extends Applet {
528
529     /**
530     *
531     */
532     private static final long serialVersionUID = 1L;
533     public static int i_no_obs;
534     float maxY,maxZ,maxX ;
535     public static float strike,strike1,strike2;
536     double obs[];
537
538     public void drawGraph(Graphics2D g2) {
539
540         g2.setFont(new Font("Arial", 20, 12));
541         g2.setColor(Color.BLACK);
542         g2.draw(new Line2D.Float(215-strike1, 50, 215-strike1, 300));
543         String []a = {"A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
544         String []b = {"D", "E", "P", "T", "H", "(k", "m)"};
545
546         for (int i = 0; i < a.length; i++) {
547
548             g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
549         }
550         for (int i = 0; i < b.length; i++) {
551
552             g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
553         }
554     }
555
556     public void plot(Graphics2D g2) {
557
558         g2.setFont( new Font("Arial", 12, 12) );
559         DecimalFormat f = new DecimalFormat("0.##");
560         g2.setColor(Color.BLACK);
561         strike = (float)
562         (FRGMSTRK_CalculateValues.d_str_st-FRGMSTRK_CalculateValues.d_off_pro);
563         strike1 = (float) (65 * (strike / (2 * FRGMSTRK_CalculateValues.d_str_st)));
564         strike2= (float) (40 * (strike /(2 * FRGMSTRK_CalculateValues.d_str_st)));
565
566

```

```

567         i_no_obs = FRGMLSTRK_CalculateValues.i_no_obs;
568
569         obs = new double[i_no_obs+1];
570         for (int i = 1; i <= i_no_obs; i++) {
571
572             obs[i] = FRGMLSTRK_CalculateValues.d_dis_km[i];
573         }
574
575         maxX = (float)obs[i_no_obs];
576         maxY = FRGMLSTRK_Utility.findMaximumNumber(FRGMLSTRK_CalculateValues.input_nob_gob);
577         maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
578
579         g2.drawString("|", (float) 600 + 65 - strikel, 260 + strike2);
580         g2.drawString(" "+f.format(FRGMLSTRK_CalculateValues.d_dis_km[i_no_obs]), (float) 600
+ 65 - strikel, 260 + strike2 - 8);
581         g2.drawString("0", 125, 310);
582         g2.drawString("DISTANCE(km)", 400, 250);
583         float xplot = 0;
584         float xInterval = (float) (FRGMLSTRK_CalculateValues.d_dis_km[i_no_obs] / 5);
585
586         int zInterval = 50;
587         for (float x = xInterval, j = 1; x < 600; x += xInterval){
588
589             xplot = xplot + xInterval;
590             if(j > 4)
591                 break;
592             g2.drawString("|", (float) (215 + (450 * x / maxX) - strikel), 260 + strike2);
593             g2.drawString(" "+ f.format(xplot), (float) (215 + (450 * x / maxX) - strikel)
3, 260 + strike2 - 8);
594             j++;
595         }
596
597         DecimalFormat d = new DecimalFormat("0.#");
598         float points1 = maxZ / 5 ;
599         for (int x = zInterval + 250, j = 1; x < 550; x += zInterval){
600
601             g2.drawString("-", 148, 52 + x);
602             g2.drawString(" "+d.format(points1 * j), 125, 50 + x);
603             j++;
604         }
605     }
606 }
607
608
609 public void plotXYCoordinates(Graphics2D g2){
610
611     double minAno =
FRGMLSTRK_Utility.findMinimumNumber1(FRGMLSTRK_CalculateValues.input_nob_gob);
612     double maxAno =
FRGMLSTRK_Utility.findMaximumNumber(FRGMLSTRK_CalculateValues.input_nob_gob,
minAno);
613     double minObAno =
FRGMLSTRK_Utility.findMinimumNumber1(FRGMLSTRK_CalculateValues.ano);
614     double maxObAno = FRGMLSTRK_Utility.findMaximumNumber(FRGMLSTRK_CalculateValues.ano,
minObAno);
615
616     if (minAno < 0 && maxObAno < 0 && maxAno < 0 && minObAno <0 ){
617         plotXYCoordinates1(g2);
618     }
619     if (minAno >= 0 && maxObAno > 0 ){
620         plotXYCoordinates1(g2);
621     }
622
623     if (minAno < 0 && maxObAno > 0 || maxAno>0 && minObAno<0){
624         plotXYCoordinates2(g2);
625     }
626 }
627
628 public void plotXYCoordinates1 (Graphics2D g2) {
629
630     g2.setFont(new Font("Arial", 20, 12));
631     g2.setColor(Color.black);
632     float maxval = (float)
FRGMLSTRK_Utility.findMaximumNumber(FRGMLSTRK_CalculateValues.ano);
633     float maxval1 = (float)

```

```

634 FRGMLSTRK_Utility.findMaximumNumber(FRGMLSTRK_CalculateValues.input_nob_gob);
635     if(Math.abs(maxval)>Math.abs(maxval1))
636     else
637         maxY = maxval1;
638     int points = (int)maxY / 5;
639     int yInterval = 50;
640     g2.drawString("0", 215 - strikel - 40, 50);
641     for (int x = yInterval, j = 1; x < 250; x += yInterval){
642
643         g2.drawString("-", 215 - strikel - 2, 50 + x);
644         g2.drawString(" " + (points*j), 215 - strikel - 40, 50 + x);
645         j++;
646     }
647     float prevx = 215 - strikel;
648     float prevy = (float)( 50 + ( 250 * FRGMLSTRK_CalculateValues.ano[1] / maxY ) );
649     float xpoint = 0;
650     float ypoint = 0;
651     float gypoint = 0;
652
653     for (int k = 1; k <= i_no_obs; k++) {
654
655         xpoint = (float)( 450 * obs[k] / maxX);
656         ypoint = (float)( ( 250 * FRGMLSTRK_CalculateValues.ano[k] / maxY ) );
657         gypoint = (float)( ( 250 * FRGMLSTRK_CalculateValues.input_nob_gob[k] / maxY )
658     );
659
660     g2.setColor(Color.BLACK);
661     g2.draw(new Line2D.Float(prevx, prevy, 215-strikel+ xpoint, 50 + ypoint ));
662
663     g2.setColor(Color.BLUE);
664     g2.setFont(new Font("Arial", 20, 40));
665     g2.drawString(".", 215-strikel+xpoint - 6 , 50 + gypoint + 3);
666
667     g2.setFont(new Font("Arial", 20, 12));
668     g2.setColor(Color.black);
669     prevx = 215-strikel + xpoint;
670     prevy = 50 + ypoint ;
671 }
672 }
673
674
675
676 public void plotXYCoordinates2 (Graphics2D g2) {
677
678
679     g2.setFont(new Font("Arial", 20, 12));
680     g2.setColor(Color.black);
681
682     double store[] = new double[FRGMLSTRK_CalculateValues.i_no_obs+1];
683     double store1[] = new double[FRGMLSTRK_CalculateValues.i_no_obs+1];
684     double negstore[] = new double[FRGMLSTRK_CalculateValues.i_no_obs+1];
685     double negstore1[] = new double[FRGMLSTRK_CalculateValues.i_no_obs+1];
686     for(int i = 1; i <= FRGMLSTRK_CalculateValues.i_no_obs; i++){
687         if(FRGMLSTRK_CalculateValues.ano[i]>0)
688             store[i] = FRGMLSTRK_CalculateValues.ano[i];
689         else
690             negstore[i] = FRGMLSTRK_CalculateValues.ano[i];
691         if(FRGMLSTRK_CalculateValues.input_nob_gob[i]>0)
692             store1[i] = FRGMLSTRK_CalculateValues.input_nob_gob[i];
693         else
694             negstore1[i] = FRGMLSTRK_CalculateValues.input_nob_gob[i];
695     }
696     float maxpos = (float) FRGMLSTRK_Utility.findMaximumNumber1(store);
697     float maxpos1 = (float) FRGMLSTRK_Utility.findMaximumNumber1(store1);
698     float maxneg = (float) FRGMLSTRK_Utility.findMaximumNumber(negstore);
699     float maxneg1 = (float) FRGMLSTRK_Utility.findMaximumNumber(negstore1);
700     float posnum =0;
701
702     if(maxpos>maxpos1)
703         posnum = maxpos;
704     else
705         posnum = maxpos1;
706     if(Math.abs(maxneg)>Math.abs(maxneg1))

```

```

707         maxY = maxneg;
708     else
709         maxY = maxneg1;
710
711     float prevx = 215-strike1;
712     float prevy = 0;
713     if(FRGMLSTRK_CalculateValues.ano[1]>0)
714         prevy = 100-(float)(( 50 * FRGMLSTRK_CalculateValues.ano[1] / posnum ));
715     else
716         prevy = 100+(float)(( 200 * FRGMLSTRK_CalculateValues.ano[1] / maxY ));
717
718     float xpoint = 0;
719     float ypoint = 0;
720     float gypoint = 0;
721
722     DecimalFormat df = new DecimalFormat("0.##");
723     g2.drawString("0",215 - strike1 - 40,100);
724     g2.drawString("-", 215-strike1, 55);
725     g2.drawString(""+df.format(posnum), 215-strike1-40, 50);
726     DecimalFormat f = new DecimalFormat("0.#");
727     float points = maxY / 4;
728     int yInterval=50;
729     for (int x = yInterval, j = 1; x < 250; x+=yInterval){
730
731         g2.drawString("-", 215 - strike1 - 2, 100 + x );
732         g2.drawString(" " + f.format(points * j), 215 - strike1 - 40, 100 + x );
733         j++;
734     }
735     for (int k = 1; k <= i_no_obs; k++) {
736
737         xpoint = (float)(( 450 * obs[k] / maxX);
738         if(FRGMLSTRK_CalculateValues.ano[k]>0)
739             ypoint = 100-(float)(( 50 * FRGMLSTRK_CalculateValues.ano[k] / posnum ));
740         else
741             ypoint = 100+(float)(( 200 * FRGMLSTRK_CalculateValues.ano[k] / maxY ));
742         if(FRGMLSTRK_CalculateValues.input_nob_gob[k]>0)
743             gypoint = 100-(float)(( 50 * FRGMLSTRK_CalculateValues.input_nob_gob[k] /
posnum ));
744         else
745             gypoint = 100+(float)(( 200 * FRGMLSTRK_CalculateValues.input_nob_gob[k] ,
maxY ));
746
747         g2.setColor(Color.BLACK);
748         g2.draw(new Line2D.Float(prevx, prevy, 215-strike1 + xpoint, ypoint ));
749
750         g2.setColor(Color.BLUE);
751         g2.setFont(new Font("Arial", 20, 40));
752         g2.drawString(".", 215-strike1+xpoint - 6 , gypoint+3 );
753
754         g2.setFont(new Font("Arial", 20, 12));
755         g2.setColor(Color.black);
756         prevx = 215-strike1 + xpoint;
757         prevy = ypoint ;
758     }
759 }
760
761 }
762
763
764 public void drawDepth(Graphics2D g2) {
765
766     DecimalFormat df = new DecimalFormat("0.#");
767     float zpoint1 = 0;
768     float zpoint = 0;
769     g2.setColor(Color.red);
770     g2.fill(new Rectangle2D.Float(151, 300, 450, 250 ));
771     g2.fillRect(150, 300,450, 250);
772     for (float j = 300; j <= 550; j++){
773         g2.draw(new Line2D.Float(600, j, 600+65, j-40 ));
774     }
775     Color col[] =
{Color.BLACK,Color.BLACK,Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELL
OW,Color.ORANGE,Color.PINK,Color.WHITE,
776     Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
olor.PINK,Color.WHITE,

```



```

777         Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
olor.PINK,Color.WHITE,
778         Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
olor.PINK,Color.WHITE,
779         Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
olor.PINK,Color.WHITE};
780
781         Color coll[] =
{Color.BLACK,Color.BLACK,Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAG
ENTA,Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
782         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREEN,
Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
783         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREEN,
Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
784         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREEN,
Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
785         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREEN,
Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA};
786
787         for (int i = 2; i < FRGMLSTRK_CalculateValues.d_dep_di_arr.length; i++) {
788
789             if (FRGMLSTRK_CalculateValues.d_den_for_arr[i-1] <= 2.38)
790                 g2.setColor(col[i]);
791             else
792                 g2.setColor(coll[i]);
793
794             zpoint = (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i] / maxZ);
795             zpoint1 = (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i-1] / maxZ);
796
797             if (i == 2) {
798                 g2.fill(new Rectangle2D.Float(151, 301, 450, zpoint));
799                 for (float j = 301; j <= 300 + zpoint; j++){
800                     g2.draw(new Line2D.Float(600, j, 600+65, j-40));
801                 }
802                 g2.setColor(Color.BLACK);
803                 g2.setFont(new Font("Arial", 40,10));
804                 g2.drawString("----->" + df.format(FRGMLSTRK_CalculateValues.d_dep_di_arr[i]
" (km)", 600 + 65, 300 + zpoint - 38));
805             }
806             else if (i > 2) {
807
808                 g2.fill(new Rectangle2D.Float(151, 300 + zpoint1, 450, zpoint - zpoint1 ));
809                 for (float j = 300 + zpoint1; j <= 300 + zpoint; j++){
810                     g2.draw(new Line2D.Float(600, j, 600 + 65, j - 40));
811                 }
812
813                 g2.setColor(Color.BLACK);
814                 g2.setFont(new Font("Arial", 40, 10));
815                 g2.drawString("----->" +
df.format(FRGMLSTRK_CalculateValues.d_dep_di_arr[i])+" (km)", 600 +65, 300 +
zpoint - 38);
816
817             }
818
819         }
820
821         g2.setColor(Color.LIGHT_GRAY);
822         for (float j = 151; j <= 600; j++){
823
824             g2.draw(new Line2D.Float(j, 300, j + 65, 300 - 40));
825         }
826         g2.setColor(Color.BLACK);
827         g2.draw(new Line2D.Float(215 - strikel, 260 + strike2, (float) 600 + 65 - strikel,
260 + strike2));
828         g2.draw(new Line2D.Float(600, 300, 600, 300+zpoint));
829         g2.draw(new Line2D.Float(600 + 65, 300 - 40, 600 + 65, 300 + zpoint - 40));
830         g2.draw(new Line2D.Float(600, 300 + zpoint, 600 + 65, 300 + zpoint - 40));
831         g2.draw(new Line2D.Float(150, 550, 600, 550));
832         g2.setColor(Color.BLACK);
833
834         for (int i = 188; i <= 600 + 32; i++){
835
836             g2.drawString("-", i, 280 + 4);
837             i = i + 4;
838         }

```

```

839         g2.drawLine(90, 30, 950, 30);
840         g2.drawLine(90, 565, 950, 565);
841         g2.drawLine(950, 30, 950, 565);
842         g2.drawLine(90, 30, 90, 565);
843         g2.setFont(new Font("Arial", 40,20));
844         g2.drawString("Interactive Gravity Modeling of Strike Limited Listric Fault
Sources", 185, 20);
845         g2.setFont(new Font("Arial", 40,12));
846     }
847
848     public void plotZCoordinates (Graphics2D g2) {
849
850         i_no_obs = FRGMLSTRK_CalculateValues.i_no_obs;
851         obs = new double[i_no_obs+1];
852         for (int i = 1; i <= i_no_obs; i++) {
853
854             obs[i] = FRGMLSTRK_CalculateValues.d_dis_km[i];
855
856         }
857         maxX = (float) obs[i_no_obs];
858         maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
859         float s = 0;
860         float fc = 0;
861         float spoint = 300;
862         float fcpoint = (float)( 150 + ( 450 * FRGMLSTRK_CalculateValues.d_cftnt_arr[1] /
maxX ) );
863         float xpoint = 0;
864         float zpoint = 0;
865
866         while (s <= FRGMLSTRK_CalculateValues.d_max_dep) {
867
868             float z1 = (float) 0.001;
869             s = s + z1;
870             fc = 0;
871             for (int i = 1; i<FRGMLSTRK_CalculateValues.d_cftnt_arr.length; i++){
872
873                 fc = (float) (fc + FRGMLSTRK_CalculateValues.d_cftnt_arr[i] * Math.pow(s, :
- 1));
874
875             }
876             xpoint = (float)( 450 * fc / maxX);
877             zpoint = (float)( 250 * s / maxZ);
878
879             g2.setColor(Color.BLACK);
880             g2.draw(new Line2D.Float(fcpoint, spoint,(float)150 + xpoint, 300 + zpoint));
881
882             g2.setColor(Color.RED);
883             g2.draw(new Line2D.Float(150, 300 + zpoint, (float) 150 + xpoint, 300 +
zpoint));
884
885             fcpoint = (float) 150 + xpoint;
886             spoint = 300 + zpoint;
887             if (fcpoint > (float) 150 + ((700 * obs[i_no_obs] / maxX))){
888                 for (float j = spoint; j <= 550; j++){
889                     g2.setColor(Color.RED);
890                     g2.draw(new Line2D.Float(150, j, (float)( 150 + ((450 * obs[i_no_obs] ,
maxX))), j));
891                     g2.draw(new Line2D.Float((float)( 151 + ((450 * obs[i_no_obs] / maxX))
j, (float)( 151 + ((450 * obs[i_no_obs] / maxX)) + 65, j - 40));
892                 }
893                 break;
894             }
895         }
896     }
897 }
898
899
900     public void drawDen(Graphics2D g2){
901
902         FRGMLSTRK_DensityGraph de = new FRGMLSTRK_DensityGraph();
903         de.denIndex(g2);
904         g2.setFont(new Font("Arial", 20, 12));
905         double denMax1 = FRGMLSTRK_CalculateValues.input_max_den -
FRGMLSTRK_CalculateValues.input_min_den;
906         float denXpoint = 0;

```

```

907         float denXpoint1 = 0;
908         double denpoint1 = 0;
909         float z1 = 0;
910         float z2 = 0;
911         for (int i = 1; i < FRGMLSTRK_CalculateValues.d_den_for_arr.length; i++){
912
913             double denpoint = FRGMLSTRK_CalculateValues.d_den_for_arr[i] -
FRGMLSTRK_CalculateValues.input_min_den;
914             if (i <= FRGMLSTRK_CalculateValues.d_den_for_arr.length - 2)
915                 denpoint1 = FRGMLSTRK_CalculateValues.d_den_for_arr[i + 1] -
FRGMLSTRK_CalculateValues.input_min_den;
916             denXpoint = (float) (90 * denpoint / denMax1);
917             denXpoint1 = (float) (90 * denpoint1 / denMax1);
918
919             z1 = (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i + 1] / maxZ);
920             z2 = (float) (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i] / maxZ);
921             g2.setColor(Color.black);
922             g2.draw(new Line2D.Float(820 + denXpoint, 300 + z2, 820 + denXpoint, 300 + z1))
923             g2.draw(new Line2D.Float(820 + denXpoint, 300 + z1, 820 + denXpoint1, 300 +
z1));
924         }
925         g2.drawLine(820, 300, 910, 300);
926         g2.drawLine(820, 300, 820, 550);
927         g2.setColor(Color.white);
928         g2.drawLine(821, 550, 910, 550);
929     }
930
931
932     public void index(Graphics2D g){
933
934         g.setColor(Color.BLUE);
935         g.setFont(new Font("Arial", 20, 50));
936         g.drawString(" ... ", 750, 67);
937         g.setFont(new Font("Arial", 20, 12));
938         g.drawString(": Observed anomalies", 820, 70);
939         g.setColor(Color.BLACK);
940         g.drawString("____", 765, 85);
941         g.drawString(": Modeled anomalies", 820, 90);
942     }
943
944 }
945 -----
946
947 package com.frgmlstrk.view.event;
948
949 import java.applet.Applet;
950 import java.awt.*;
951 import java.awt.event.*;
952 import java.awt.geom.Line2D;
953 import java.text.DecimalFormat;
954 import com.frgmlstrk.model.FRGMLSTRK_CalculateValues;
955 import com.frgmlstrk.view.FRGMLSTRK_MainPanel;
956 import com.frgmlstrk.view.graph.FRGMLSTRK_PlainGraph;
957
958 public class FRGMLSTRK_PlotFault extends Applet{
959
960
961     /**
962      *
963      */
964     private static final long serialVersionUID = 1L;
965     public static double val[] ;
966     public static double val1[];
967     public static int pos;
968     public static MouseListener ml;
969     com.frgmlstrk.model.FRGMLSTRK_CalculateValues cv = new
com.frgmlstrk.model.FRGMLSTRK_CalculateValues();
970     FRGMLSTRK_PlainGraph pg = new FRGMLSTRK_PlainGraph();
971
972     public void paint(Graphics g){
973
974         FRGMLSTRK_MainPanel.p_Center.removeAll();
975         FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.graphLabel);
976

```

```

977 FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.im);
978 FRGMLSTRK_CalculateValues.len=0;
979 val = new double[50] ;
980 val1 = new double[50] ;
981 Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.im.getGraphics();
982 g2.setFont(new Font("Arial", 40,12));
983 g2.setColor(Color.red);
984 FRGMLSTRK_MainPanel.im.setEditable(false);
985 FRGMLSTRK_MainPanel.im.setBackground(Color.WHITE);
986 FRGMLSTRK_MainPanel.p_Center.validate();
987
988 for (int j = 0; j < val.length; j++){
989     val[j] = 0;
990     val1[j] = 0;
991 }
992 pos = 2;
993 cv.getAnamolyValues(com.frgmlstrk.view.FRGMLSTRK_MainPanel.captureValues());
994 Graphics2D g1 = (Graphics2D)FRGMLSTRK_MainPanel.im.getGraphics();
995 pg.drawPlainGraph(g1);
996
997 ml = new MouseAdapter(){
998     public void mouseClicked(MouseEvent e) {
999         DecimalFormat f = new DecimalFormat("0.##");
1000         float polyx = e.getX();
1001         float polyy = e.getY();
1002         float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1003         float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1004         float x[] = new float[50];
1005         float z[] = new float[50];
1006         int get = pos;
1007         FRGMLSTRK_CalculateValues.len = get;
1008         Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.im.getGraphics();
1009         if (polyy >= 300 && polyy <= (300 + zer) && polyx >= 150 && polyx <= 600){
1010
1011             FRGMLSTRK_PlotFault.val[get] = (float)
Math.abs((FRGMLSTRK_PlainGraph.maxX1 * (150 - polyx)) / 450);
1012             FRGMLSTRK_PlotFault.val1[get] = (float)
Math.abs(FRGMLSTRK_CalculateValues.d_max_dep * (300 - polyy) / zer);
1013             x[get] = polyx;
1014             z[get] = polyy;
1015             g2.setColor(Color.white);
1016             g2.draw(new Line2D.Float(150, polyy, 600, polyy) );
1017             g2.draw(new Line2D.Float(600, polyy, 600 + 65, polyy - 40) );
1018             g2.fillRect(730, 160, 250, 40);
1019
1020             g2.setColor(Color.BLACK);
1021             g2.drawString("x", polyx, polyy);
1022             g2.drawString("(" + f.format(FRGMLSTRK_PlotFault.val[get]) + ", " + f.format(FRGMLSTRK_PlotFault.val1[get]) + ")", polyx, polyy);
1023             g2.setFont(new Font("Arial", 40, 20));
1024             if (get <= FRGMLSTRK_CalculateValues.i_d_poly)
1025                 g2.setColor(Color.red);
1026             else
1027                 g2.setColor(Color.blue);
1028
1029             g2.drawString("Number of coordinates " + get, 740, 180);
1030             g2.setFont(new Font("Arial", 40, 12));
1031             FRGMLSTRK_PlotFault.pos++;
1032         }
1033
1034         pg.drawPlainGraph(g2);
1035         for (int i = 2; i <= FRGMLSTRK_CalculateValues.len; i++){
1036             g2.setColor(Color.BLACK);
1037             g2.drawString("(" + f.format(FRGMLSTRK_PlotFault.val[FRGMLSTRK_CalculateValues.len]) + ", " + f.format(FRGMLSTRK_PlotFault.val1[FRGMLSTRK_CalculateValues.len]) + ")", x[get], z[get]);
1038         }
1039         repaint();
1040     }
1041
1042     public void mouseMoved(MouseEvent e){
1043
1044         DecimalFormat f = new DecimalFormat("0.##");
1045         float xer = e.getX();
1046         float yer = e.getY();

```

```

1047         float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1048         float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1049         float xpoint =(float) Math.abs((FRGMLSTRK_PlainGraph.maxX1 * (150 - xer)) /
450);
1050         float zpoint = (float) Math.abs(FRGMLSTRK_CalculateValues.d_max_dep * (300
yer) / zer);
1051         if (yer >= 300 && yer <= (300 + zer) && xer >= 150 && xer <= 600){
1052             Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.im.getGraphics();
1053             g2.setColor(Color.black);
1054             g2.drawString("X-coordinates           Z-coordinates", 780, 320);
1055             g2.setColor(Color.red);
1056             g2.fillRect(780, 320, 180, 20);
1057             g2.setColor(Color.black);
1058             g2.drawString(" "+f.format(xpoint)+"
+f.format(zpoint)+"", 780, 330);
1059         }
1060     }
1061 }
1062 };
1063
1064
1065 FRGMLSTRK_MainPanel.im.addMouseListener(ml);
1066 FRGMLSTRK_MainPanel.im.addMouseMotionListener((MouseMotionListener)ml);
1067 g2.setFont(new Font("Arial", 40, 20));
1068 g2.setColor(Color.black);
1069 g2.drawLine(615, 0, 615, 125);
1070 g2.drawLine(615, 125, 990, 125);
1071 g2.drawString("Instructions", 730, 20);
1072 g2.drawString("_____", 710, 20);
1073 g2.setFont(new Font("Arial", 40, 12));
1074 g2.setColor(Color.red);
1075 g2.drawString("1) Select points on the fault plane by clicking the mouse ", 640,
40);
1076 g2.drawString("in the structure panel ", 644, 60);
1077 g2.setColor(Color.blue);
1078 g2.drawString("X", 640, 120);
1079 g2.setColor(Color.red);
1080 g2.drawString(":is origin of the fault plane ", 650, 120);
1081 g2.setColor(Color.red);
1082 g2.drawString("2) Select the points within the specified boundary ", 640, 80);
1083 g2.drawString("of the structure panel", 644, 100);
1084 }
1085
1086 }
1087 -----
1088 package com.frgmlstrk.view.event;
1089
1090 import java.awt.*;
1091 import java.awt.event.*;
1092 import java.text.DecimalFormat;
1093 import com.frgmlstrk.model.FRGMLSTRK_CalculateValues;
1094 import com.frgmlstrk.util.FRGMLSTRK_Utility;
1095 import com.frgmlstrk.view.FRGMLSTRK_MainPanel;
1096 import com.frgmlstrk.view.graph.FRGMLSTRK_PlainGraph;
1097
1098 public class FRGMLSTRK_PlotDepth {
1099
1100     public static double val2[] = new double [50];
1101
1102     public static int pos1;
1103     public static MouseListener ml2;
1104     com.frgmlstrk.model.FRGMLSTRK_CalculateValues cv = new
1105 com.frgmlstrk.model.FRGMLSTRK_CalculateValues();
1106     com.frgmlstrk.view.FRGMLSTRK_DrawGraph dg = new
1107 FRGMLSTRK_PlainGraph pg = new FRGMLSTRK_PlainGraph();
1108
1109     public void paint(Graphics g){
1110
1111         FRGMLSTRK_MainPanel.p_Center.removeAll();
1112         FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.graphLabel);
1113         FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.img2);
1114

```

```

1115 FRGMLSTRK_MainPanel.img2.setEditable(false);
1116 FRGMLSTRK_MainPanel.img2.setBackground(Color.WHITE);
1117 FRGMLSTRK_MainPanel.p_Center.validate();
1118 cv.getAnamolyValues(com.frgmlstrk.view.FRGMLSTRK_MainPanel.captureValues());
1119 FRGMLSTRK_CalculateValues.inter = 2 + FRGMLSTRK_CalculateValues.i_nob_di;
1120 val2 = new double[50] ;
1121
1122 final FRGMLSTRK_PlainGraph pg = new FRGMLSTRK_PlainGraph();
1123 try{
1124     cv.getAnamolyValues(com.frgmlstrk.view.FRGMLSTRK_MainPanel.captureValues());
1125     cv.getCoefficients();
1126     Graphics2D g1 = (Graphics2D)FRGMLSTRK_MainPanel.img2.getGraphics();
1127     pg.drawPlainGraph(g1);
1128     dg.plotZCoordinates(g1);
1129     pg.drawPlainGraph(g1);
1130     Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.img2.getGraphics();
1131     for (int j = 0; j < val2.length; j++){
1132         val2[j] = 0;
1133     }
1134 }
1135
1136 val2[1] = 0;
1137 val2[FRGMLSTRK_CalculateValues.inter] = FRGMLSTRK_CalculateValues.d_max_dep;
1138 pos1 = 2;
1139
1140 ml2 = new MouseAdapter(){
1141     public void mouseClicked(MouseEvent e) {
1142
1143         DecimalFormat f = new DecimalFormat("0.##");
1144         float polyx = e.getX();
1145         float polyy = e.getY();
1146         float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1147         float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1148         float x[] = new float[50];
1149         float z[] = new float[50];
1150         int get = pos1;
1151
1152         Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.img2.getGraphics();
1153         if (polyy >= 300 && polyy <= (300 + zer) && polyx >= 150 && polyx <=
600){
1154             if (get < FRGMLSTRK_CalculateValues.inter){
1155
1156                 val2[get] = (float) Math.abs(FRGMLSTRK_CalculateValues.d_max_de
* (300 - polyy) / zer);
1157                 String alter = f.format(FRGMLSTRK_PlotDepth.val2[get]);
1158                 try {
1159                     val2[get]= FRGMLSTRK_Utility.convertDouble(alter);
1160                 } catch (Exception e1) {
1161                     e1.printStackTrace();
1162                 }
1163                 x[get] = polyx;
1164                 g2.setColor(Color.white);
1165                 g2.fillRect(730, 160, 250, 40);
1166                 g2.setColor(Color.BLACK);
1167                 g2.drawString("x", polyx, polyy);
1168                 g2.drawString("(" + f.format(FRGMLSTRK_PlotDepth.val2[get]) + ")",
polyx, polyy);
1169                 g2.setFont(new Font("Arial", 40, 20));
1170                 g2.drawString("Depths to be plotted
"+Math.abs(get-(FRGMLSTRK_CalculateValues.inter-1)), 740, 180);
1171                 g2.setFont(new Font("Arial", 40, 12));
1172             }
1173             FRGMLSTRK_PlotDepth.pos1++;
1174         }
1175         pg.drawPlainGraph(g2);
1176         dg.plotZCoordinates(g2);
1177         pg.drawPlainGraph(g2);
1178         for (int i = 2; i <= get; i++){
1179             g2.setColor(Color.BLACK);
1180             g2.drawString("(" + f.format(FRGMLSTRK_PlotDepth.val2[get]) + ")",
x[get], z[get]);
1181         }
1182         FRGMLSTRK_MainPanel.img2.repaint();
1183     }
1184 }

```

```

1185
1186         public void mouseMoved(MouseEvent e){
1187
1188             DecimalFormat f = new DecimalFormat("0.#");
1189             float xer = e.getX();
1190             float yer = e.getY();
1191             float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1192             float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1193             float zpoint = (float) Math.abs(FRGMLSTRK_CalculateValues.d_max_dep *
(300 - yer) / zer);
1194             if (yer >= 300 && yer <= (300 + zer) && xer >= 150 && xer <= 600){
1195                 Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.img2.getGraphics();
1196                 g2.setColor(Color.red);
1197                 g2.fillRect(780, 120, 100, 20);
1198                 g2.setColor(Color.black);
1199                 g2.drawString("Depth(km)", 800, 115);
1200                 g2.drawString(""+f.format(zpoint)+"", 810, 130);
1201
1202             }
1203
1204         }
1205
1206     };
1207
1208     FRGMLSTRK_MainPanel.img2.addMouseListener(ml2);
1209     FRGMLSTRK_MainPanel.img2.addMouseMotionListener((MouseMotionListener)ml2);
1210     g2.setFont(new Font("Arial", 40, 20));
1211     g2.setColor(Color.black);
1212     g2.drawString("Depths to be plotted "+(FRGMLSTRK_CalculateValues.inter-2), 740,
180);
1213     g2.drawLine(615, 0, 615, 85);
1214     g2.drawLine(615, 85, 990, 85);
1215     g2.drawString("Instructions", 730, 20);
1216     g2.drawString("_____", 710, 20);
1217     g2.setFont(new Font("Arial", 40, 12));
1218     g2.setColor(Color.red);
1219     g2.drawString("1) Select the depth values by clicking the mouse ", 660, 40);
1220     g2.drawString("in the structure panel.", 664, 60);
1221     g2.setColor(Color.red);
1222     g2.drawString("2) Click the mouse in ascending order of depth", 660, 80);
1223 }
1224
1225     catch(Exception e){
1226         //exception
1227     }
1228 }
1229 }
1230
-----
1231 package com.frgmlstrk.view.event;
1232
1233 import java.awt.*;
1234 import java.awt.event.*;
1235 import java.text.DecimalFormat;
1236 import com.frgmlstrk.model.FRGMLSTRK_CalculateValues;
1237 import com.frgmlstrk.util.FRGMLSTRK_Utility;
1238 import com.frgmlstrk.view.FRGMLSTRK_MainPanel;
1239 import com.frgmlstrk.view.graph.FRGMLSTRK_DensityGraph;
1240 import com.frgmlstrk.view.graph.FRGMLSTRK_DepthLines;
1241 import com.frgmlstrk.view.graph.FRGMLSTRK_PlainGraph;
1242
1243
1244 public class FRGMLSTRK_PlotDensity {
1245
1246     public static double val3[];
1247     public static int pos2;
1248     public static MouseListener ml4;
1249     com.frgmlstrk.model.FRGMLSTRK_CalculateValues cv = new
com.frgmlstrk.model.FRGMLSTRK_CalculateValues();
1250     com.frgmlstrk.view.FRGMLSTRK_DrawGraph dg = new
com.frgmlstrk.view.FRGMLSTRK_DrawGraph();
1251     FRGMLSTRK_PlainGraph pg = new FRGMLSTRK_PlainGraph();
1252     FRGMLSTRK_DensityGraph de = new FRGMLSTRK_DensityGraph();
1253     FRGMLSTRK_DepthLines dl = new FRGMLSTRK_DepthLines();
1254

```



```

1255
1256 public void paint(Graphics g){
1257
1258     val3 = new double[50] ;
1259     FRGMLSTRK_MainPanel.p_Center.removeAll();
1260     FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.graphLabel);
1261     FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.den);
1262     FRGMLSTRK_MainPanel.den.setEditable(false);
1263     FRGMLSTRK_MainPanel.den.setBackground(Color.WHITE);
1264     FRGMLSTRK_MainPanel.p_Center.validate();
1265     try{
1266         com.frgmlstrk.view.FRGMLSTRK_MainPanel.clearPanel(FRGMLSTRK_MainPanel.den);
1267         cv.getAnamolyValues(com.frgmlstrk.view.FRGMLSTRK_MainPanel.captureValues());
1268         FRGMLSTRK_CalculateValues.inter = 2 + FRGMLSTRK_CalculateValues.i_nob_di;
1269         cv.getCoefficients();
1270         cv.depthValues();
1271         Graphics2D g1 = (Graphics2D)FRGMLSTRK_MainPanel.den.getGraphics();
1272         pg.drawPlainGraph(g1);
1273         dl.drawPlaneLines(g1);
1274         dg.plotZCoordinates(g1);
1275         pg.drawPlainGraph(g1);
1276         de.drawPlain(g1);
1277         Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.den.getGraphics();
1278
1279         for (int j = 0; j < val3.length; j++){
1280             val3[j] = 0;
1281         }
1282         pos2 = 1;
1283         ml4 = new MouseAdapter(){
1284             public void mouseClicked(MouseEvent e) {
1285
1286                 float diff = (float) (FRGMLSTRK_CalculateValues.input_max_den -
FRGMLSTRK_CalculateValues.input_min_den);
1287                 float denx = e.getX();
1288                 float deny = e.getY();
1289                 float maxZ = diff;
1290                 float zer = (float) (90 * diff / maxZ);
1291                 float []y = new float[50];
1292                 float []x = new float[50];
1293                 int get = pos2;
1294
1295                 DecimalFormat f = new DecimalFormat("0.##");
1296                 Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.den.getGraphics();
1297                 if (deny >= 300 && deny <= 550 && denx >= 820 && denx <= 910){
1298                     if (get < FRGMLSTRK_CalculateValues.inter){
1299
1300                         val3[get] = (float)(FRGMLSTRK_CalculateValues.input_min_den +
Math.abs(diff * (820 - denx) / zer));
1301                         String alter = f.format(FRGMLSTRK_PlotDensity.val3[get]);
1302                         try {
1303                             val3[get] = FRGMLSTRK_Utility.convertDouble(alter);
1304                         } catch (Exception e1) {
1305                             e1.printStackTrace();
1306                         }
1307                         y[get] = deny;
1308                         x[get] = denx;
1309                         g2.setColor(Color.white);
1310                         g2.fillRect(730, 160, 250, 40);
1311                         g2.setColor(Color.BLACK);
1312                         g2.drawString("x", denx, deny);
1313                         g2.drawString("(" + f.format(FRGMLSTRK_PlotDensity.val3[get]) + ")");
1314                         denx, deny);
1315                         g2.setFont(new Font("Arial", 40, 20));
1316                         g2.drawString("Densities to be plotted",
"+Math.abs(get-(FRGMLSTRK_CalculateValues.inter-1)), 740, 180);
1317                         g2.setFont(new Font("Arial", 40, 12));
1318                     }
1319                     FRGMLSTRK_PlotDensity.pos2++;
1320                 }
1321
1322                 dl.drawPlaneLines(g2);
1323                 dg.plotZCoordinates(g2);
1324                 pg.drawPlainGraph(g2);
1325

```



```

1326         de.drawPlain(g2);
1327         for (int i = 1; i <= get; i++){
1328             g2.setColor(Color.BLACK);
1329             g2.drawString("(" + f.format(FRGMLSTRK_PlotDensity.val3[get]) + ")",
1330 x[get], y[get]);
1331         }
1332         FRGMLSTRK_MainPanel.den.repaint();
1333     }
1334
1335     public void mouseMoved(MouseEvent e){
1336
1337         float diff = (float) (FRGMLSTRK_CalculateValues.input_max_den -
FRGMLSTRK_CalculateValues.input_min_den);
1338         DecimalFormat f = new DecimalFormat("0.##");
1339         float xer = e.getX();
1340         float yer = e.getY();
1341         float maxZ = diff;
1342         float zer = (float) (90 * diff / maxZ);
1343         float zpoint = (float) Math.abs(diff * (820 - xer) / zer);
1344         if (yer >= 300 && yer <= 550 && xer >= 820 && xer <= 910){
1345             Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.den.getGraphics();
1346             g2.setColor(Color.red);
1347             g2.fillRect(780, 120, 100, 20);
1348             g2.setColor(Color.black);
1349             g2.drawString("Density range", 800, 115);
1350             g2.drawString("(" + f.format(FRGMLSTRK_CalculateValues.input_min_den+z
oint) + " ", 810, 130);
1351         }
1352     }
1353
1354     };
1355     FRGMLSTRK_MainPanel.den.addMouseListener(ml4);
1356     FRGMLSTRK_MainPanel.den.addMouseMotionListener((MouseMotionListener)ml4);
1357
1358     g2.setFont(new Font("Arial", 40, 20));
1359     g2.setColor(Color.black);
1360     g2.drawString("Densities to be plotted " + (FRGMLSTRK_CalculateValues.inter-1),
740, 180);
1361     g2.drawLine(615, 0, 615, 85);
1362     g2.drawLine(615, 85, 990, 85);
1363     g2.drawString("Instructions", 730, 20);
1364     g2.drawString("_____", 710, 20);
1365     g2.setFont(new Font("Arial", 40, 12));
1366     g2.setColor(Color.red);
1367     g2.drawString("1) Select the density values by clicking the mouse ", 642, 40);
1368     g2.drawString("in the density-depth panel ", 646, 60);
1369     g2.setColor(Color.red);
1370     g2.drawString("2) Click the mouse between two consecutive depth interfaces",
642, 80);
1371     }
1372     catch(Exception e){
1373         e.printStackTrace();
1374     }
1375 }
1376 -----
1377 package com.frgmlstrk.view.event;
1378
1379 import java.awt.*;
1380 import java.awt.event.*;
1381 import java.text.DecimalFormat;
1382 import com.frgmlstrk.model.FRGMLSTRK_CalculateValues;
1383 import com.frgmlstrk.util.FRGMLSTRK_HandleException;
1384 import com.frgmlstrk.view.FRGMLSTRK_MainPanel;
1385 import com.frgmlstrk.view.graph.FRGMLSTRK_PlainGraph;
1386
1387 public class FRGMLSTRK_EditFault {
1388     public static float zpoint = 0;
1389     public static int count = 0;
1390     public static MouseListener ml1;
1391     com.frgmlstrk.view.FRGMLSTRK_DrawGraph dg = new
com.frgmlstrk.view.FRGMLSTRK_DrawGraph();

```

```

1393     FRGMLSTRK_PlainGraph pg = new FRGMLSTRK_PlainGraph();
1394     com.frgmlstrk.model.FRGMLSTRK_CalculateValues cv = new
com.frgmlstrk.model.FRGMLSTRK_CalculateValues();
1395     public void paint(Graphics g){
1396         final DecimalFormat f = new DecimalFormat("0.##");
1397         FRGMLSTRK_MainPanel.p_Center.removeAll();
1398         FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.graphLabel);
1399         FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.fl);
1400         FRGMLSTRK_MainPanel.fl.setEditable(false);
1401         FRGMLSTRK_MainPanel.fl.setBackground(Color.WHITE);
1402         FRGMLSTRK_MainPanel.p_Center.validate();
1403         cv.getAnomalyValues(com.frgmlstrk.view.FRGMLSTRK_MainPanel.captureValues());
1404         try {
1405             cv.getCoefficients();
1406         } catch (FRGMLSTRK_HandleException e2) {
1407
1408             e2.printStackTrace();
1409         }
1410         Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.fl.getGraphics();
1411
1412         pg.drawPlainGraph(g2);
1413         dg.plotZCoordinates(g2);
1414         pg.drawPlainGraph(g2);
1415         g2.setColor(Color.black);
1416         for(int i = 1;i<=FRGMLSTRK_CalculateValues.len;i++){
1417             float xpoint = (float)
(450*FRGMLSTRK_CalculateValues.d_x_km_arr[i]/FRGMLSTRK_PlainGraph.maxX1);
1418             float zpoint = (float)(250*
FRGMLSTRK_CalculateValues.d_z_km_arr[i]/FRGMLSTRK_CalculateValues.d_max_dep);
1419             g2.drawString("(" + f.format(FRGMLSTRK_CalculateValues.d_x_km_arr[i]) + ", " + f.forma
(FRGMLSTRK_CalculateValues.d_z_km_arr[i]) + ")", 160+xpoint, 304+zpoint);
1420             g2.drawString("X", 150+xpoint-3, 304+zpoint);
1421         }
1422
1423         m11 = new MouseAdapter(){
1424
1425             public void mouseDragged(MouseEvent e1) {
1426                 DecimalFormat f = new DecimalFormat("0.##");
1427                 float x2 = e1.getX();
1428                 float y2 = e1.getY();
1429                 float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1430                 float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1431                 Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.fl.getGraphics();
1432                 if (y2 > 300 && y2 <= (300 + zer) && x2 <= 600){
1433
1434                     g2.setColor(Color.red);
1435                     g2.fillRect(750, 180, 180, 30);
1436                     g2.setColor(Color.BLACK);
1437                     zpoint = (float) Math.abs((FRGMLSTRK_PlainGraph.maxX1 * (150 - x2)) /
450);
1438                     g2.drawString("Sl.no          x-coordinate(km)", 750,
180);
1439                     g2.drawString("(" + f.format(zpoint), 870, 200);
1440                     g2.setColor(Color.black);
1441                     g2.drawString("(" + (count) , 770, 200);
1442                 }
1443
1444                 pg.drawPlainGraph(g2);
1445                 dg.plotZCoordinates(g2);
1446                 pg.drawPlainGraph(g2);
1447             }
1448
1449         }
1450
1451         public void mousePressed(MouseEvent e1) {
1452
1453             float x1 = e1.getX();
1454             float y1 = e1.getY();
1455             float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1456             float diff = (float)
(FRGMLSTRK_CalculateValues.d_dis_km[FRGMLSTRK_CalculateValues.i_no_obs]/100)
1457 ;
1458             float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1459             float xer = (float) Math.abs((FRGMLSTRK_PlainGraph.maxX1 * (150 - (x1-3)))

```

```

450);
1460
1461     Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.fl.getGraphics();
1462     if (y1 > 300 && y1 < (300 + zer) && x1 <= 600){
1463
1464         for (int kk = 2; kk <= FRGMLSTRK_CalculateValues.len; kk++){
1465             if (Math.abs((float)FRGMLSTRK_CalculateValues.d_x_km_arr[kk] - xer
1466 )<= diff || Math.abs((float)FRGMLSTRK_CalculateValues.d_z_km_arr[kk]
1467 - zer )<= 0 ){
1468                 count = kk;
1469             }
1470             g2.setColor(Color.white);
1471             g2.drawString(""+count, 750, 380);
1472         }
1473     }
1474 }
1475 }
1476 }
1477 }
1478 }
1479
1480 public void mouseReleased(MouseEvent e1) {
1481
1482     float x = e1.getX();
1483     float y = e1.getY();
1484     float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1485     float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1486     Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.fl.getGraphics();
1487     FRGMLSTRK_PlainGraph pg = new FRGMLSTRK_PlainGraph();
1488     if (y > 300 && y <= (300 + zer) && x <= 600){
1489
1490         float xer = (float) Math.abs((FRGMLSTRK_PlainGraph.maxX1 * (150 - x)) /
450);
1491
1492         FRGMLSTRK_PlotFault.val[count]=xer;
1493         FRGMLSTRK_CalculateValues.d_x_km_arr[count] = xer;
1494     }
1495
1496     try {
1497
1498         cv.getCoefficients();
1499     } catch (FRGMLSTRK_HandleException e) {
1500         e.printStackTrace();
1501     }
1502     FRGMLSTRK_MainPanel.clearPanel(FRGMLSTRK_MainPanel.fl);
1503
1504     Graphics2D g1 = (Graphics2D)FRGMLSTRK_MainPanel.fl.getGraphics();
1505     pg.drawPlainGraph(g1);
1506     dg.plotZCoordinates(g1);
1507     g1.setColor(Color.black);
1508     for(int i = 1;i<=FRGMLSTRK_CalculateValues.len;i++){
1509         float xpoint = (float)
(450*FRGMLSTRK_CalculateValues.d_x_km_arr[i]/FRGMLSTRK_PlainGraph.maxX1)
;
1510         float zpoint = (float)(250*
FRGMLSTRK_CalculateValues.d_z_km_arr[i]/FRGMLSTRK_CalculateValues.d_max_
dep);
1511         g2.drawString("(" +f.format(FRGMLSTRK_CalculateValues.d_x_km_arr[i])+", "
f.format(FRGMLSTRK_CalculateValues.d_z_km_arr[i])+")", 160+xpoint,
304+zpoint);
1512         g2.drawString("X", 150+xpoint-3, 304+zpoint);
1513     }
1514 }
1515 }
1516 };
1517
1518 FRGMLSTRK_MainPanel.fl.addMouseListener(ml1);
1519 FRGMLSTRK_MainPanel.fl.addMouseMotionListener((MouseMotionListener)ml1);
1520 }
1521 }
1522 -----
1523

```

```

1524 package com.frgmlstrk.view.event;
1525
1526 import java.awt.*;
1527 import java.awt.event.*;
1528 import java.awt.geom.Line2D;
1529 import java.text.DecimalFormat;
1530 import com.frgmlstrk.model.FRGMSTRK_CalculateValues;
1531 import com.frgmlstrk.util.FRGMSTRK_Utility;
1532 import com.frgmlstrk.view.FRGMSTRK_MainPanel;
1533 import com.frgmlstrk.view.graph.FRGMSTRK_DensityGraph;
1534 import com.frgmlstrk.view.graph.FRGMSTRK_DepthLines;
1535
1536
1537 public class FRGMSTRK_EditDepth {
1538
1539     public static float zpoint = 0;
1540     public static int count = 0;
1541     public static MouseListener mll;
1542     com.frgmlstrk.view.FRGMSTRK_DrawGraph dg = new
com.frgmlstrk.view.FRGMSTRK_DrawGraph();
1543     FRGMSTRK_DepthLines dl = new FRGMSTRK_DepthLines();
1544
1545     public void paint(Graphics g){
1546         mll = new MouseAdapter(){
1547             DecimalFormat f = new DecimalFormat("0.#");
1548             public void mouseDragged(MouseEvent e1) {
1549
1550                 float x2 = e1.getX();
1551                 float y2 = e1.getY();
1552                 float maxZ = (float)FRGMSTRK_CalculateValues.d_max_dep;
1553                 float zer = (float) (250 * FRGMSTRK_CalculateValues.d_max_dep / maxZ);
1554                 Graphics2D g2 = (Graphics2D)FRGMSTRK_MainPanel.img3.getGraphics();
1555                 if (y2 > 300 && y2 <= (300 + zer) && x2 <= 600){
1556
1557                     g2.setColor(Color.red);
1558                     g2.fillRect(750, 180, 180, 30);
1559                     g2.setColor(Color.BLACK);
1560                     zpoint = (float) Math.abs(FRGMSTRK_CalculateValues.d_max_dep * (300 -
y2) / zer);
1561
1562                     g2.drawString("Interface Depth", 750, 180);
1563                     g2.drawString(""+f.format(zpoint), 870, 200);
1564                     g2.setColor(Color.black);
1565                     g2.drawString(""+(count - 1), 770, 200);
1566
1567                 }
1568                 dg.plot(g2);
1569                 dg.plotXYCoordinates(g2);
1570                 dl.drawDepthLine(g2);
1571             }
1572
1573
1574     public void mousePressed(MouseEvent e1) {
1575
1576         float x1 = e1.getX();
1577         float y1 = e1.getY();
1578         float maxZ = (float)FRGMSTRK_CalculateValues.d_max_dep;
1579         float dep = (float)FRGMSTRK_CalculateValues.d_max_dep / 100;
1580         float zer = (float) (250 * FRGMSTRK_CalculateValues.d_max_dep / maxZ);
1581         Graphics2D g2 = (Graphics2D)FRGMSTRK_MainPanel.img3.getGraphics();
1582         if (y1 > 300 && y1 < (300 + zer) && x1 <= 600){
1583
1584             float zpoint1 = (float) Math.abs(FRGMSTRK_CalculateValues.d_max_dep *
(300 - y1) / zer);
1585             for (int kk = 2; kk < FRGMSTRK_CalculateValues.inter; kk++){
1586
1587                 if (Math.abs((float)FRGMSTRK_CalculateValues.d_dep_di_arr[kk] -
zpoint1) <= dep){
1588                     count = kk;
1589                 }
1590                 g2.setColor(Color.white);
1591                 g2.drawString(""+count, 750, 380);
1592             }
1593         }
1594     }

```

```

1595         }
1596         dg.plot(g2);
1597         dg.plotXYCoordinates(g2);
1598         dl.drawDepthLine(g2);
1599     }
1600
1601
1602
1603     public void mouseReleased(MouseEvent e1) {
1604
1605         float x = e1.getX();
1606         float y = e1.getY();
1607         float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1608         float zer = (float) (250 * FRGMLSTRK_CalculateValues.d_max_dep / maxZ);
1609         Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.img3.getGraphics();
1610
1611         if (y > 300 && y <= (300 + zer) && x <= 600 && FRGMLSTRK_CalculateValues.inte
> 2){
1612
1613             g2.setColor(Color.BLACK);
1614             g2.draw(new Line2D.Float(150, y, 600, y));
1615             g2.draw(new Line2D.Float(600, y, 600 + 65, y - 40));
1616             zpoint = (float) Math.abs(FRGMLSTRK_CalculateValues.d_max_dep * (300 -
y) / zer);
1617             String alter = f.format(zpoint);
1618             try{
1619                 FRGMLSTRK_PlotDepth.val2[count] =
FRGMLSTRK_Utility.convertDouble(alter);
1620                 FRGMLSTRK_CalculateValues.d_dep_di_arr[count] =
FRGMLSTRK_Utility.convertDouble(alter);
1621             }
1622             catch(Exception e){
1623
1624             }
1625             g2.drawString(""+f.format(zpoint), 600, y);
1626         }
1627         dg.plot(g2);
1628         dg.plotXYCoordinates(g2);
1629         dl.drawDepthLine(g2);
1630
1631         com.frgmlstrk.model.FRGMLSTRK_CalculateValues cv = new
com.frgmlstrk.model.FRGMLSTRK_CalculateValues();
1632         cv.getAnamolyValues(com.frgmlstrk.view.FRGMLSTRK_MainPanel.captureValues())
1633         try {
1634             cv.cal();
1635             com.frgmlstrk.view.FRGMLSTRK_TableView.populateEastPanel(FRGMLSTRK_Calc
lateValues.obj);
1636             drawGraph(FRGMLSTRK_MainPanel.img3);
1637         } catch (Exception e) {
1638
1639         }
1640     }
1641 }
1642
1643 };
1644
1645 FRGMLSTRK_MainPanel.img3.addMouseListener(ml1);
1646 FRGMLSTRK_MainPanel.img3.addMouseMotionListener((MouseMotionListener)ml1);
1647 }
1648 public void drawGraph(TextArea t){
1649
1650     FRGMLSTRK_DensityGraph de = new FRGMLSTRK_DensityGraph();
1651     Graphics2D g2 = (Graphics2D)t.getGraphics();//(Graphics2D)g1 ;
1652     com.frgmlstrk.view.FRGMLSTRK_MainPanel.clearPanel(t);
1653     dg.plot(g2);
1654     dg.plotXYCoordinates(g2);
1655     dg.drawGraph(g2);
1656     dl.drawDepthLine(g2);
1657     dg.plotZCoordinates(g2);
1658     dg.plot(g2);
1659     de.Plainden(g2);
1660 }
1661 }
1662 -----

```

```

1663
1664 package com.frgmlstrk.view.event;
1665
1666 import java.awt.*;
1667 import java.awt.event.*;
1668 import java.awt.geom.Line2D;
1669 import java.text.DecimalFormat;
1670 import com.frgmlstrk.model.FRGMSTRK_CalculateValues;
1671 import com.frgmlstrk.util.FRGMSTRK_Utility;
1672 import com.frgmlstrk.view.FRGMSTRK_MainPanel;
1673 import com.frgmlstrk.view.graph.FRGMSTRK_DensityGraph;
1674
1675 public class FRGMSTRK_EditDensity {
1676
1677     MouseListener ml5;
1678     public static float dpoint = 0;
1679     public static int count1 = 0;
1680     FRGMSTRK_DensityGraph de = new FRGMSTRK_DensityGraph();
1681
1682     public void paint(Graphics g){
1683
1684         ml5 = new MouseAdapter(){
1685
1686             public void mouseDragged(MouseEvent e1) {
1687
1688                 float diff = (float) (FRGMSTRK_CalculateValues.input_max_den -
FRGMSTRK_CalculateValues.input_min_den);
1689                 float x2 = e1.getX();
1690                 float y2 = e1.getY();
1691                 float maxZ = diff;
1692                 float zer = (float) (90 * diff / maxZ);
1693                 Graphics2D g2 = (Graphics2D)FRGMSTRK_MainPanel.img3.getGraphics();
1694                 if (y2 > 300 && y2 <= 570 && x2 >= 820 && x2 <= 910){
1695                     g2.setColor(Color.red);
1696                     g2.fillRect(750, 240, 180, 30);
1697                     g2.setColor(Color.black);
1698                     dpoint = (float)(FRGMSTRK_CalculateValues.input_min_den + Math.abs(diff
* (820 - x2) / zer));
1699                     DecimalFormat d = new DecimalFormat("0.##");
1700                     g2.drawString("Formation Density", 750, 235);
1701                     g2.drawString(""+d.format(dpoint), 870, 250);
1702                     g2.drawString(""+count1, 770, 250);
1703                 }
1704                 de.Plainden(g2);
1705             }
1706         }
1707     }
1708
1709     public void mousePressed(MouseEvent e1) {
1710
1711         float x1 = e1.getX();
1712         float y1 = e1.getY();
1713
1714         float maxZ1 = (float)FRGMSTRK_CalculateValues.d_max_dep;
1715         float zer1 = (float) (250 * FRGMSTRK_CalculateValues.d_max_dep / maxZ1);
1716         Graphics2D g2 = (Graphics2D)FRGMSTRK_MainPanel.img3.getGraphics();
1717         if (y1 > 300 && y1 < 570 && x1 >= 820 && x1 <= 910){
1718             float zp = (float) Math.abs(FRGMSTRK_CalculateValues.d_max_dep * (300
y1) / zer1);
1719             for (int kk = 1; kk < FRGMSTRK_CalculateValues.inter; kk++){
1720                 if (zp > FRGMSTRK_CalculateValues.d_dep_di_arr[kk] && zp <
FRGMSTRK_CalculateValues.d_dep_di_arr[kk + 1]){
1721                     count1 = kk;
1722                 }
1723                 g2.setColor(Color.white);
1724                 g2.drawString(""+count1, 750, 380);
1725             }
1726             de.Plainden(g2);
1727         }
1728     }
1729 }
1730
1731 }
1732
1733

```

```

1734         public void mouseReleased(MouseEvent e1) {
1735
1736             float diff = (float) (FRGMLSTRK_CalculateValues.input_max_den -
FRGMLSTRK_CalculateValues.input_min_den);
1737             DecimalFormat d = new DecimalFormat("0.##");
1738             float x = e1.getX();
1739             float y = e1.getY();
1740             float maxZ = diff;
1741             float zer = (float) (90 * diff / maxZ);
1742             Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.img3.getGraphics();
1743             if (y > 300 && y <= 570 && x >= 820 && x <= 910){
1744
1745                 float zp = (float)(FRGMLSTRK_CalculateValues.input_min_den + (90 *
(FRGMLSTRK_CalculateValues.d_den_for_arr[count1]-FRGMLSTRK_CalculateValu
es.input_min_den) / maxZ));
1746                 g2.setColor(Color.WHITE);
1747                 g2.draw(new Line2D.Float(820 + zp, y, 820 + zp, y));
1748                 g2.drawString(""+d.format(FRGMLSTRK_CalculateValues.d_den_for_arr[count
]), 910, 300 + zp);
1749                 g2.setColor(Color.black);
1750                 dpoint = (float)(FRGMLSTRK_CalculateValues.input_min_den+ Math.abs(diff
* (820 - x) / zer));
1751                 String alter = d.format(dpoint);
1752                 try {
1753                     FRGMLSTRK_PlotDensity.val3[count1] =
FRGMLSTRK_Utility.convertDouble(alter);
1754                     } catch (Exception e) {
1755                         e.printStackTrace();
1756                     }
1757                     g2.drawString(""+d.format(dpoint), 910, y);
1758                 }
1759                 de.Plainden(g2);
1760             }
1761         };
1762
1763         FRGMLSTRK_MainPanel.img3.addMouseListener(ml5);
1764         FRGMLSTRK_MainPanel.img3.addMouseMotionListener((MouseMotionListener)ml5);
1765     }
1766 }
1767 }
1768 -----
1769 package com.frgmlstrk.view.graph;
1770
1771 import java.awt.*;
1772 import java.awt.geom.Line2D;
1773 import java.text.DecimalFormat;
1774 import com.frgmlstrk.model.FRGMLSTRK_CalculateValues;
1775 import com.frgmlstrk.util.FRGMLSTRK_Utility;
1776
1777 public class FRGMLSTRK_PlainGraph {
1778
1779     int i_no_obs;
1780     public static float maxX1;
1781     public void drawPlainGraph(Graphics2D g2){
1782
1783         g2.setColor(Color.BLACK);
1784         g2.setFont(new Font("Arial", 20, 12));
1785         i_no_obs = FRGMLSTRK_CalculateValues.i_no_obs;
1786         double obser[] = new double[i_no_obs + 1];
1787         for (int i = 1; i <= i_no_obs; i++) {
1788
1789             obser[i] = FRGMLSTRK_CalculateValues.d_dis_km[i];
1790         }
1791         maxX1 = (float) FRGMLSTRK_CalculateValues.d_dis_km[i_no_obs];
1792         float maxZ1 = (float)FRGMLSTRK_CalculateValues.d_max_dep;
1793         float points = maxX1 / 5;
1794         int zInterval = 50;
1795         DecimalFormat f = new DecimalFormat("0.##");
1796         g2.drawLine(150,50,150,300);
1797         g2.drawString("|", (float) 598, 300);
1798         g2.drawString(""+f.format(FRGMLSTRK_CalculateValues.d_dis_km[i_no_obs]),600, 320);
1799         int xInterval=90;
1800         for (int x = xInterval, j =1; x < 600; x+=xInterval)
1801

```



```

1802     {
1803         if(j > 4)
1804             break;
1805         g2.drawString("|",150+x,300);
1806         g2.drawString(" " + (points*j), 147+x, 320);
1807         j++;
1808     }
1809     points = maxZl / 5 ;
1810     g2.drawString("0", 125, 305);
1811     DecimalFormat d = new DecimalFormat("0.##");
1812     for (int x = zInterval+250,j = 1; x < 550; x += zInterval){
1813
1814         g2.drawString("-", 148, 52 + x);
1815         g2.drawString(" " +d.format(points * j), 125, 50 + x);
1816         j++;
1817     }
1818     g2.draw(new Line2D.Float(150, 300, 600, 300));
1819     g2.draw(new Line2D.Float(150, 300, 150, 550));
1820     float maxY = (float)
FRGMLSTRK_Utility.findMaximumNumber(FRGMLSTRK_CalculateValues.input_nob_gob);
1821     double minAno =
FRGMLSTRK_Utility.findMinimumNumber1(FRGMLSTRK_CalculateValues.input_nob_gob);
1822     double maxAno =
FRGMLSTRK_Utility.findMaximumNumber(FRGMLSTRK_CalculateValues.input_nob_gob,
minAno);
1823
1824     if (minAno < 0 && maxAno > 0){
1825
1826         float xpoint = 0;
1827         float gypoint = 0;
1828         g2.setFont(new Font("Arial", 20, 12));
1829         g2.setColor(Color.black);
1830         double store1[] = new double[FRGMLSTRK_CalculateValues.i_no_obs+1];
1831         double negstore1[] = new double[FRGMLSTRK_CalculateValues.i_no_obs+1];
1832         for(int i = 1; i <= FRGMLSTRK_CalculateValues.i_no_obs; i++){
1833
1834             if(FRGMLSTRK_CalculateValues.input_nob_gob[i]>0)
1835                 store1[i] = FRGMLSTRK_CalculateValues.input_nob_gob[i];
1836             else
1837                 negstore1[i] = FRGMLSTRK_CalculateValues.input_nob_gob[i];
1838         }
1839         float posnum = (float) FRGMLSTRK_Utility.findMaximumNumber1(store1);
1840         maxY = (float) FRGMLSTRK_Utility.findMaximumNumber(negstore1);
1841         DecimalFormat df = new DecimalFormat("0.##");
1842         g2.drawString("0",115,100);
1843         g2.drawString("-",148, 55);
1844         g2.drawString(" "+df.format(posnum), 115, 50);
1845
1846         points = maxY / 4;
1847         int yInterval=50;
1848         for (int x = yInterval, j = 1; x < 250; x+=yInterval){
1849
1850             g2.drawString("-", 148, 100 + x );
1851             g2.drawString(" " + f.format(points * j), 115, 100 + x );
1852             j++;
1853         }
1854         for (int k = 1; k <= i_no_obs; k++) {
1855
1856             xpoint = (float)( 450 * obser[k] / maxX1);
1857             if(FRGMLSTRK_CalculateValues.input_nob_gob[k]>0)
1858                 gypoint = 100-(float)( ( 50 * FRGMLSTRK_CalculateValues.input_nob_gob[
/ posnum ) );
1859             else
1860                 gypoint = 100+(float)( ( 200 *
FRGMLSTRK_CalculateValues.input_nob_gob[k] / maxY ) );
1861             g2.setColor(Color.BLUE);
1862             g2.setFont(new Font("Arial", 20, 40));
1863             g2.drawString(".", 150 + xpoint - 6 , gypoint + 3 );
1864             g2.setFont(new Font("Arial", 20, 12));
1865             g2.setColor(Color.black);
1866
1867         }
1868     }
1869 }
1870 else{

```



```

1871         g2.drawString("0", 135, 50);
1872         points = (int)maxY / 5;
1873
1874         int yInterval = 50;
1875         for (int x = yInterval, j = 1; x < 250; x += yInterval){
1876
1877             g2.drawString("-", 148, 52 + x);
1878             g2.drawString(" " + (points * j), 115, 50 + x);
1879             j++;
1880         }
1881
1882         float xpoint = 0;
1883         float gypoint = 0;
1884
1885         for (int k = 1; k <= i_no_obs; k++) {
1886
1887             xpoint = (float)( 450 * obser[k] / maxX1);
1888             gypoint = (float)( ( 250 * FRGMLSTRK_CalculateValues.input_nob_gob[k] / max
    ) );
1889
1890             g2.setColor(Color.BLUE);
1891             g2.setFont(new Font("Arial", 20, 40));
1892             g2.drawString(".", 150 + xpoint - 6 , 50 + gypoint + 3);
1893
1894         }
1895     }
1896     double inipoint = calInip();
1897     float plotini = (float)( 450 * inipoint / maxX1);
1898     g2.setFont(new Font("Arial", 20, 12));
1899     g2.drawString("X", 150 + plotini-3 , 304);
1900     g2.draw(new Line2D.Float(600, 300, 600, 300 + (float)(250 *
FRGMLSTRK_CalculateValues.d_max_dep / maxZ1)));
1901     g2.draw(new Line2D.Float(150, 300 + (float)(250 *
FRGMLSTRK_CalculateValues.d_max_dep / maxZ1), 600, 300 + (float)(250 *
FRGMLSTRK_CalculateValues.d_max_dep / maxZ1)));
1902 }
1903
1904 public double calInip(){
1905     double gmax = Math.abs(FRGMLSTRK_CalculateValues.input_nob_gob[1]);
1906
1907     for (int k = 1; k <= i_no_obs; k++) {
1908
1909         if (Math.abs(FRGMLSTRK_CalculateValues.input_nob_gob[k]) - gmax > 0)
1910             gmax = Math.abs(FRGMLSTRK_CalculateValues.input_nob_gob[k]);
1911     }
1912     double datum = FRGMLSTRK_CalculateValues.input_nob_gob[1];
1913     double r =
FRGMLSTRK_CalculateValues.input_nob_gob[i_no_obs]-FRGMLSTRK_CalculateValues.input_no
b_gob[1];
1915     int kk1 = 1;
1916     double gh = 0.5 * r;
1917     kk1 = kk1 + 1;
1918     double XH = 0;
1919
1920     while ((( FRGMLSTRK_CalculateValues.input_nob_gob[kk1] - datum) / gh ) - 1.0 < 0) .
1921
1922         kk1 = kk1 + 1;
1923     }
1924
1925     if ((( FRGMLSTRK_CalculateValues.input_nob_gob[kk1] - datum) / gh) - 1.0 > 0) {
1926
1927         XH = FRGMLSTRK_CalculateValues.d_dis_km[kk1-1] + ( ( gh + datum -
FRGMLSTRK_CalculateValues.input_nob_gob[kk1-1]) * (FRGMLSTRK_CalculateValues.
d_dis_km[kk1] - FRGMLSTRK_CalculateValues.d_dis_km[kk1-1] ) ) / (
FRGMLSTRK_CalculateValues.input_nob_gob[kk1] -
FRGMLSTRK_CalculateValues.input_nob_gob[kk1-1] );
1928     }
1929     if ((( FRGMLSTRK_CalculateValues.input_nob_gob[kk1] - datum ) / gh) - 1.0 == 0) {
1930
1931         XH = FRGMLSTRK_CalculateValues.d_dis_km[kk1];
1932     }
1933
1934     double ini = XH;
1935     return ini;

```

```

1936     }
1937 }
1938 -----
1939 package com.frgmlstrk.view.graph;
1940
1941 import java.awt.*;
1942 import java.awt.geom.Line2D;
1943 import java.text.DecimalFormat;
1944 import com.frgmlstrk.model.FRGMSTRK_CalculateValues;
1945 import com.frgmlstrk.view.FRGMSTRK_DrawGraph;
1946
1947 public class FRGMSTRK_DepthLines {
1948
1949     int i_no_obs = FRGMSTRK_CalculateValues.i_no_obs;
1950     float maxX;
1951     float maxZ = (float)FRGMSTRK_CalculateValues.d_max_dep;
1952     public void drawDepthLine(Graphics2D g2) {
1953
1954         maxX = (float)FRGMSTRK_CalculateValues.d_dis_km[i_no_obs];
1955         float zpoint = 0;
1956         DecimalFormat d = new DecimalFormat("0.#");
1957         for (int i = 1; i < FRGMSTRK_CalculateValues.d_dep_di_arr.length; i++) {
1958
1959             zpoint = (float) (250 * FRGMSTRK_CalculateValues.d_dep_di_arr[i] / maxZ);
1960             g2.draw(new Line2D.Float(151, 300 + zpoint, 600, 300 + zpoint));
1961             g2.draw(new Line2D.Float(600, 300 + zpoint, 600 + 65, 300 + zpoint - 40));
1962             if (i > 1 && i < FRGMSTRK_CalculateValues.inter){
1963                 g2.drawString(""+d.format(FRGMSTRK_CalculateValues.d_dep_di_arr[i]), 600,
1964 300 + zpoint);
1965                 g2.drawString(""+(i - 1), 600 + 85, 300 + zpoint - 40);
1966             }
1967
1968             g2.setColor(Color.BLACK);
1969             g2.draw(new Line2D.Float(215 - FRGMSTRK_DrawGraph.strike1, 260 +
FRGMSTRK_DrawGraph.strike2, (float) 600 + 65 - FRGMSTRK_DrawGraph.strike1, 260 +
FRGMSTRK_DrawGraph.strike2));
1970             g2.draw(new Line2D.Float(600, 300, 600, 300 + zpoint));
1971             g2.draw(new Line2D.Float(600 + 65, 300 - 40, 600 + 65, 300 + zpoint - 40));
1972             g2.draw(new Line2D.Float(600, 300 + zpoint, 600 + 65, 300 + zpoint - 40));
1973             g2.draw(new Line2D.Float(150, 300, 150 + 65, 300 - 40));
1974             g2.draw(new Line2D.Float(150 + 65, 300 - 40, 600 + 65, 300 - 40));
1975             g2.draw(new Line2D.Float(600, 300, 600 + 65, 300 - 40));
1976             g2.setColor(Color.BLACK);
1977
1978             for (int i = 188; i <= 600 + 32; i++){
1979
1980                 g2.drawString("-", i, 280 + 4);
1981                 i = i + 4;
1982             }
1983         }
1984     }
1985
1986     public void drawPlaneLines(Graphics2D g){
1987         maxX = (float)FRGMSTRK_CalculateValues.d_dis_km[i_no_obs];
1988         float zpoint = 0;
1989         for (int i = 1; i < FRGMSTRK_CalculateValues.d_dep_di_arr.length; i++) {
1990             zpoint = (float) (250 * FRGMSTRK_CalculateValues.d_dep_di_arr[i] / maxZ);
1991             g.setColor(Color.black);
1992             g.draw(new Line2D.Float(150, 300 + zpoint, 600, 300 + zpoint));
1993         }
1994     }
1995 }
1996 }
1997 -----
1998
1999 package com.frgmlstrk.view.graph;
2000
2001 import java.awt.*;
2002 import java.awt.geom.Line2D;
2003 import java.text.DecimalFormat;
2004 import com.frgmlstrk.model.FRGMSTRK_CalculateValues;
2005

```

```

2006
2007 public class FRGMLSTRK_DensityGraph {
2008
2009     float maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
2010     public void drawPlain(Graphics2D g2){
2011
2012         g2.setColor(Color.black);
2013         g2.drawLine(820,300,910,300);
2014         g2.drawLine(820, 300, 820,550);
2015         float zpoint = 0;
2016         denIndex(g2);
2017         for (int i = 1; i < FRGMLSTRK_CalculateValues.d_dep_di_arr.length; i++) {
2018             zpoint = (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i] / maxZ);
2019             g2.setColor(Color.black);
2020             g2.draw(new Line2D.Float(820, 300 + zpoint, 910, 300 + zpoint ));
2021
2022         }
2023
2024     }
2025
2026     public void Plainden(Graphics2D g2){
2027
2028         denIndex(g2);
2029
2030         g2.drawLine(820, 300, 910, 300);
2031         g2.drawLine(820, 300, 820, 550);
2032         float zpoint = 0;
2033         for (int i = 1; i < FRGMLSTRK_CalculateValues.d_dep_di_arr.length; i++) {
2034             zpoint = (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i] / maxZ);
2035             g2.setColor(Color.black);
2036             g2.draw(new Line2D.Float(820, 300 + zpoint, 910, 300 + zpoint));
2037
2038         }
2039
2040         float diff = (float) (FRGMLSTRK_CalculateValues.input_max_den -
FRGMLSTRK_CalculateValues.input_min_den);
2041         float maxDen = diff;
2042         for (int i = 1; i < FRGMLSTRK_CalculateValues.d_den_for_arr.length; i++){
2043             zpoint = (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i] / maxZ);
2044             float zpoint1 = (float) (250 * FRGMLSTRK_CalculateValues.d_dep_di_arr[i + 1] /
maxZ);
2045             float denpoint = (float) (90 * (FRGMLSTRK_CalculateValues.d_den_for_arr[i] -
FRGMLSTRK_CalculateValues.input_min_den) / maxDen);
2046             g2.setColor(Color.blue);
2047             g2.draw(new Line2D.Float(820 + denpoint, 300 + zpoint, 820 + denpoint, 300 +
zpoint1));
2048         }
2049     }
2050
2051     public void denIndex(Graphics2D g){
2052
2053         DecimalFormat d = new DecimalFormat("0.#");
2054         maxZ = (float)FRGMLSTRK_CalculateValues.d_max_dep;
2055         float points = maxZ / 5 ;
2056         int zInterval = 50;
2057
2058         for (int x = zInterval + 250, j = 1; x < 550; x += zInterval){
2059
2060             g.drawString("-", 818, 52 + x);
2061             g.drawString(" " +d.format(points * j), 795, 50 + x);
2062             j++;
2063         }
2064
2065         g.setColor(Color.red);
2066         String []b = {"D", "E", "P", "T", "H", "(k", "m)"};
2067
2068         for (int i = 0; i < b.length; i++) {
2069
2070             g.drawString(" "+b[i], 765, 350 + ( i * 20 ) );
2071         }
2072         g.setColor(Color.black);
2073         g.drawString(" "+FRGMLSTRK_CalculateValues.input_min_den, 818, 298);
2074         g.drawString(" "+FRGMLSTRK_CalculateValues.input_max_den, 908, 298);
2075         g.setColor(Color.red);
2076         g.drawString("Density", 850, 280);

```

```

2077         g.drawString("(gm/cm )",850 , 295);
2078         g.setFont(new Font("Arial", 20, 9));
2079         g.drawString("3",890 , 292);
2080         g.setFont(new Font("Arial", 20, 12));
2081         g.setColor(Color.black);
2082         g.drawString("0", 795, 305);
2083     }
2084 }
2085
2086 -----
2087 package com.frgmlstrk.model;
2088
2089 import java.text.DecimalFormat;
2090 import java.util.Arrays;
2091 import java.util.HashMap;
2092
2093 import javax.swing.JFrame;
2094 import javax.swing.JOptionPane;
2095
2096
2097
2098 import com.frgmlstrk.util.FRGMSTRK_HandleException;
2099 import com.frgmlstrk.util.FRGMSTRK_Utility;
2100 import com.frgmlstrk.view.FRGMSTRK_TableView;
2101 import com.frgmlstrk.view.event.FRGMSTRK_PlotDensity;
2102 import com.frgmlstrk.view.event.FRGMSTRK_PlotDepth;
2103 import com.frgmlstrk.view.event.FRGMSTRK_PlotFault;
2104
2105
2106 public class FRGMSTRK_CalculateValues {
2107
2108     public static Object obj[][] = null;
2109     public static int inter = 2;
2110     public static int i_no_obs, i_d_poly, i_nob_di,len = 0;
2111     public static double d_str_st, d_off_pro, d_max_dep, d_base_den, input_max_den,
2112 input_min_den = 0;
2113     public static double d_dep_di_arr[], d_ele_km[], d_den_for_arr[], d_dis_km[], ano[],
d_cftnt_arr[], x[], cftnt[] = null;
2114     public static double input_nob_gob[] = null;
2115     public static String input_area_name, input_profile_num = "";
2116     public static double d_x_km_arr[], d_z_km_arr[];
2117
2118     public void getAnamolyValues(HashMap h_Map) {
2119         try {
2120             i_no_obs = FRGMSTRK_Utility.convertInteger((String)h_Map.get("N_OBS"));
2121             i_d_poly = FRGMSTRK_Utility.convertInteger((String)h_Map.get("D_POLY"));
2122             d_str_st = FRGMSTRK_Utility.convertDouble((String)h_Map.get("STR_ST"));
2123             d_off_pro = FRGMSTRK_Utility.convertDouble((String)h_Map.get("OFF_PRO"));
2124             d_dis_km = FRGMSTRK_Utility.convertDoubleArray((String)h_Map.get("DIS_KM"));
2125             d_ele_km = FRGMSTRK_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
2126             i_nob_di = FRGMSTRK_Utility.convertInteger((String)h_Map.get("NOB_DI"));
2127             d_max_dep = FRGMSTRK_Utility.convertDouble((String)h_Map.get("MAX_DEP"));
2128             d_base_den = FRGMSTRK_Utility.convertDouble((String)h_Map.get("BASE_DEN"));
2129             input_area_name = FRGMSTRK_Utility.convertString((String)h_Map.get("AREA_FE"));
2130             input_profile_num =
FRGMSTRK_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
2131             input_nob_gob =
FRGMSTRK_Utility.convertDoubleArray((String)h_Map.get("OBS_ANO"));
2132             input_max_den = FRGMSTRK_Utility.convertDouble((String)h_Map.get("MAX_DEN"));
2133             input_min_den = FRGMSTRK_Utility.convertDouble((String)h_Map.get("MIN_DEN"));
2134         }
2135         catch(Exception e) {
2136
2137             e.printStackTrace();
2138         }
2139     }
2140
2141     public void depthValues(){
2142         inter = 2;
2143         inter = inter + i_nob_di;
2144         d_dep_di_arr = new double[inter + 1];
2145         for (int i = 1 ;i <= inter; i++){
2146

```

```

2147         d_dep_di_arr[i] = FRGMLSTRK_PlotDepth.val2[i];
2148     }
2149 }
2150
2151 }
2152
2153 public void cal() throws FRGMLSTRK_HandleException{
2154     getCoefficients();
2155     depthValues();
2156
2157     d_den_for_arr = new double[inter];
2158     for (int i = 1; i < inter; i++){
2159         d_den_for_arr[i] = FRGMLSTRK_PlotDensity.val3[i];
2160     }
2161
2162     double z[] = new double[1000];
2163     ano = new double[i_no_obs + 1];
2164     x = new double[i_no_obs + 1];
2165
2166     for (int i = 1; i <= i_no_obs; i++){
2167         x[i] = d_dis_km[i];
2168     }
2169
2170     cftnt = d_cftnt_arr;
2171     int effd = 0;
2172     int jjk = 0;
2173     double []lyy = new double[3];
2174
2175     double []lgg = new double[3];
2176     double []lgs = new double[2500];
2177     double []lgsmod = new double[2500];
2178     double GC;
2179     double []G = new double[i_no_obs+1];
2180
2181     double zb;
2182     double wgc[][] = new double[2500][2500];
2183     for (int kk = 1; kk <= inter; kk++){
2184         if(kk == inter)
2185             break;
2186         double zt = d_dep_di_arr[kk];
2187         zb = d_dep_di_arr[kk + 1];
2188         effd = effd + 1;
2189         yy[1] = d_str_st + d_off_pro;
2190         yy[2] = d_str_st - d_off_pro;
2191         double DX = ((x[2] - x[1]) / 10);
2192         double Zdif = zb - zt;
2193         int ND = (int)(Zdif / DX) + 1;
2194         int nal = ND / 2;
2195         if ((ND - 2 * nal) < 0 || (ND - 2 * nal) > 0)
2196             ND = ND + 1;
2197
2198         double dz = Zdif / ND;
2199         int N2 = ND + 1;
2200         for (int JZ = 1; JZ <= N2; JZ++){
2201             z[JZ] = zt + dz * (JZ - 1);
2202         }
2203
2204         for (int k = 1; k <= i_no_obs; k++){
2205             double xx = x[k];
2206             for (int jz = 1; jz <= N2; jz++){
2207                 double DC;
2208                 DC = d_den_for_arr[effd] - d_base_den;
2209                 double sum = 0.0;
2210                 for (int jj = 1; jj <= i_d_poly + 1; jj++){
2211                     sum = sum + cftnt[jj] * Math.pow(z[jz], jj - 1);
2212                 }
2213             }
2214         }
2215     }
2216 }

```

```

2222
2223         for (int kl = 1; kl <= 2; kl++){
2224             double effy = yy[kl];
2225             double tr1 = Math.atan(effy/(z[jz]-d_ele_km[k]));
2226             double ttp = Math.pow(-xx + sum, 2) + Math.pow(effy, 2) +
2227 Math.pow(z[jz] - d_ele_km[k], 2);
2228             double tr2 = Math.atan((effy * (-xx + sum)) / ((z[jz] - d_ele_km[k]
* Math.sqrt(ttp))));
2229             gg[kl] = 13.3333 * DC * (tr1 - tr2);
2230         }
2231         gs[jz] = (gg[2] + gg[1]) / 2;
2232     }
2233     double DZ = z[2] - z[1];
2234     double Sum1 = 0.0;
2235     double Sum2 = 0.0;
2236     double N1 = N2 / 2;
2237     double N4 = N1 - 1;
2238     for (int i3 = 1; i3 <= N1; i3++){
2239
2240         N2 = 2 * i3;
2241         Sum1 = Sum1 + gs[N2];
2242
2243     }
2244     for (int j2 = 1; j2 <= N4; j2++){
2245
2246         int N3 = 2 * j2 + 1;
2247         Sum2 = Sum2 + gs[N3];
2248     }
2249     GC = gs[1] + 4 * Sum1 + 2 * Sum2 + gs[N2];
2250     GC = (GC * DZ) / 3.0;
2251     G[k] = GC;
2252     jjk = jjk + 1;
2253     gdmod[jjk] = G[k];
2254
2255 }
2256 if (zb == d_max_dep){
2257     for (int lk = 1; lk <= i_no_obs; lk++){
2258
2259         int klkl = lk;
2260         for (int jh = 1; jh <= inter - 1; jh++){
2261
2262             wgc[lk][jh] = gdmod[klkl];
2263             klkl = klkl + i_no_obs;
2264         }
2265     }
2266 }
2267 }
2268
2269
2270 for (int il = 1; il <= i_no_obs; il++){
2271
2272     double sum = 0.0;
2273     for (int ih = 1; ih <= inter - 1; ih++){
2274
2275         sum = sum + wgc[il][ih];
2276     }
2277     ano[il] = sum;
2278
2279 }
2280 double functl=0;
2281 double err[] = new double [i_no_obs+1];
2282 for (int k = 1; k <= i_no_obs; k++) {
2283
2284     err[k] = input_nob_gob[k] - ano[k];
2285     functl = functl + Math.pow( err[k] , 2 );
2286     // System.out.println(err[k]);
2287 }
2288 setGraphValues(x, input_nob_gob, ano, cftnt);
2289 }
2290
2291
2292 public void getCoefficients() throws FRGMLSTRK_HandleException{
2293
2294     double gmax = Math.abs(input_nob_gob[1]);

```

```

2295
2296     for (int k = 1; k <= i_no_obs; k++) {
2297
2298         if (Math.abs(input_nob_gob[k]) - gmax > 0)
2299             gmax = Math.abs(input_nob_gob[k]);
2300
2301     }
2302
2303     double datum = input_nob_gob[1];
2304     double r = input_nob_gob[i_no_obs] - input_nob_gob[1];
2305     int kk1 = 1;
2306     double gh = 0.5 * r;
2307     kk1 = kk1 + 1;
2308     double XH = 0;
2309
2310     while ((( input_nob_gob[kk1] - datum) / gh ) - 1.0 < 0) {
2311
2312         kk1 = kk1 + 1;
2313     }
2314
2315     if ((( input_nob_gob[kk1] - datum) / gh) - 1.0 > 0) {
2316
2317         XH = d_dis_km[kk1-1] + ( ( gh + datum - input_nob_gob[kk1-1]) * ( d_dis_km[kk1-1]
- d_dis_km[kk1-1] ) ) / ( input_nob_gob[kk1] - input_nob_gob[kk1-1] );
2318     }
2319     if ((( input_nob_gob[kk1] - datum ) / gh) - 1.0 == 0) {
2320
2321         XH = d_dis_km[kk1];
2322     }
2323
2324     double ini = XH;
2325
2326     d_x_km_arr = new double[len + 1];
2327     d_z_km_arr = new double[len + 1];
2328
2329     if (len <= i_d_poly){
2330         JFrame frame = null;
2331         JOptionPane.showMessageDialog(frame,
2332             "The polynomial fit requires\n"
2333             + "Order+1 data points.\n"
2334             + "Use additional data points",
2335             "Error!",
2336             JOptionPane.ERROR_MESSAGE);
2337         throw new FRGMLSTRK_HandleException();
2338     }
2339
2340     for (int i = 0; i <= len; i++){
2341         if ( i == 0){
2342             d_x_km_arr[i] = 0;
2343
2344             d_z_km_arr[i] = 0;
2345         }
2346         if ( i == 1){
2347             d_x_km_arr[i] = ini;
2348             d_z_km_arr[i] = 0;
2349         }
2350
2351         else{
2352             d_x_km_arr[i] = FRGMLSTRK_PlotFault.val[i];
2353             d_z_km_arr[i] = FRGMLSTRK_PlotFault.val1[i];
2354         }
2355     }
2356
2357
2358
2359     Arrays.sort(d_x_km_arr);
2360     Arrays.sort(d_z_km_arr);
2361     double d_coeff_values[] = new double[i_d_poly + 1];
2362     double d_coeff_xzval_arr[] = new double[i_d_poly + 1];
2363     double d_coeff_zval_arr[][] = new double[i_d_poly + 1][i_d_poly + 1] ;
2364
2365     double d_sum_lmatrix = 0;
2366     double d_sum_rmatrix = 0;
2367
2368     for (int l = 0; l < i_d_poly + 1; l++){

```

```

2369
2370     d_sum_lmatrix = 0;
2371     for(int i = 0; i < this.d_x_km_arr.length; i++){
2372
2373         if (l == 0){
2374
2375             d_sum_lmatrix = d_sum_lmatrix + this.d_x_km_arr[i];
2376         }
2377         else{
2378             double d_sum_xz = this.d_x_km_arr[i] * Math.pow(this.d_z_km_arr[i], l);
2379             d_sum_lmatrix = d_sum_lmatrix+d_sum_xz;
2380         }
2381     }
2382
2383     d_coeff_xzval_arr[l] = d_sum_lmatrix;
2384 }
2385
2386 double d_rmatrix_arr[] = new double[2 * i_d_poly + 1];
2387 for (int l = 0; l < d_rmatrix_arr.length; l++){
2388
2389     d_sum_rmatrix = 0;
2390     for (int i = 0; i < d_x_km_arr.length; i++){
2391
2392         if (l == 0){
2393
2394             d_sum_rmatrix = d_sum_rmatrix + d_z_km_arr[i];
2395             d_rmatrix_arr[l] = d_sum_rmatrix;
2396         }
2397         else{
2398
2399             double d_sum_zpow = Math.pow(d_z_km_arr[i], l);
2400             d_sum_rmatrix = d_sum_rmatrix + d_sum_zpow;
2401             d_rmatrix_arr[l] = d_sum_rmatrix;
2402         }
2403     }
2404 }
2405
2406 }
2407 for (int i = 0; i < i_d_poly + 1; i++){
2408
2409     for (int j = 0; j < i_d_poly + 1; j++){
2410
2411         if ( i == 0 && j == 0){
2412
2413             d_coeff_zval_arr[i][j] = d_x_km_arr.length - 1;
2414         }
2415         else{
2416
2417             d_coeff_zval_arr[i][j] = d_rmatrix_arr[i + j];
2418         }
2419     }
2420 }
2421
2422 }
2423
2424 int index[] = new int[i_d_poly + 1];
2425 d_coeff_values = solve(d_coeff_zval_arr, d_coeff_xzval_arr, index);
2426
2427
2428 d_cftnt_arr = new double[i_d_poly + 2];
2429 d_cftnt_arr[0] = 0.0;
2430
2431
2432 for (int i = 1; i <= i_d_poly + 1; i++){
2433
2434     d_cftnt_arr[i] = d_coeff_values[i - 1];
2435 }
2436
2437 }
2438
2439
2440 public double[] solve(double a[][],
2441     double b[], int index[]) {
2442     int n = b.length;
2443     double x[] = new double[n];

```



```

2444         gaussian(a, index);
2445
2446     for (int i = 0; i < n - 1; ++i) {
2447         for (int j = i + 1; j < n; ++j) {
2448             b[index[j]] -= a[index[j]][i] * b[index[i]];
2449         }
2450     }
2451 }
2452
2453
2454     x[n - 1] = b[index[n - 1]] / a[index[n - 1]][n - 1];
2455     for (int i = n - 2; i >= 0; --i) {
2456         x[i] = b[index[i]];
2457         for (int j = i + 1; j < n; ++j) {
2458             x[i] -= a[index[i]][j] * x[j];
2459         }
2460         x[i] /= a[index[i]][i];
2461     }
2462     return x;
2463 }
2464
2465
2466 public void gaussian(double a[][],
2467     int index[]) {
2468     int n = index.length;
2469     double c[] = new double[n];
2470
2471     for (int i = 0; i < n; ++i) index[i] = i;
2472
2473     for (int i = 0; i < n; ++i) {
2474         double c1 = 0;
2475         for (int j = 0; j < n; ++j) {
2476             double c0 = Math.abs(a[i][j]);
2477             if (c0 > c1) c1 = c0;
2478         }
2479         c[i] = c1;
2480     }
2481
2482     int k = 0;
2483     for (int j = 0; j < n - 1; ++j) {
2484         double pi1 = 0;
2485         for (int i = j; i < n; ++i) {
2486             double pi0 = Math.abs(a[index[i]][j]);
2487             pi0 /= c[index[i]];
2488             if (pi0 > pi1) {
2489                 pi1 = pi0;
2490                 k = i;
2491             }
2492         }
2493
2494         int itmp = index[j];
2495         index[j] = index[k];
2496         index[k] = itmp;
2497         for (int i = j + 1; i < n; ++i) {
2498             double pj = a[index[i]][j] / a[index[j]][j];
2499             a[index[i]][j] = pj;
2500             for (int l = j + 1; l < n; ++l)
2501                 a[index[i]][l] -= pj * a[index[j]][l];
2502         }
2503     }
2504 }
2505
2506
2507
2508     public static void setGraphValues(double []x,double []gobs, double []anomaly,double
2509 []coeff) {
2510
2511         obj = new Object[i_no_obs + 1][3];
2512         DecimalFormat d1 = new DecimalFormat("0.#");
2513         DecimalFormat d = new DecimalFormat("0.##");
2514         DecimalFormat df = new DecimalFormat("0.###");
2515         DecimalFormat df1 = new DecimalFormat("0.####");
2516         for (int K = 1; K <= i_no_obs; K++){
2517             obj[K][0]= "" + d.format(x[K]);

```

```

2518         obj[K][1]= " " + df.format(gobs[K]);
2519         obj[K][2]= " " + df.format(anomaly[K]);
2520     }
2521 }
2522
2523
2524
2525     FRGMLSTRK_TableView.val.setText(" ");
2526     FRGMLSTRK_TableView.val.append("Coordinates of selected points on the fault
plane(x,z):-\n");
2527     FRGMLSTRK_TableView.val.appendText("-----\n")
2528     for (int i = 1; i <= len; i++){
2529
2530         FRGMLSTRK_TableView.val.appendText("(" + d.format(d_x_km_arr[i]) + ", " + d.format(d_z
km_arr[i]) + ")\n") ;
2531     }
2532     FRGMLSTRK_TableView.val.appendText("\n");
2533     FRGMLSTRK_TableView.val.append("Polynomial coefficients of the fault plane:-\n");
2534     FRGMLSTRK_TableView.val.appendText("-----\n")
2535     FRGMLSTRK_TableView.val.appendText("\n");
2536
2537     for (int i = 1; i < coeff.length; i++){
2538         FRGMLSTRK_TableView.val.appendText("f" + (i - 1) + " = " + df1.format(coeff[i]) +
"\n");
2539     }
2540
2541     FRGMLSTRK_TableView.val.appendText("\n");
2542     FRGMLSTRK_TableView.val.append("Depths of density interfaces:-\n");
2543     FRGMLSTRK_TableView.val.appendText("-----\n")
2544     FRGMLSTRK_TableView.val.appendText("\n");
2545
2546     for (int i = 1; i <= FRGMLSTRK_CalculateValues.inter; i++){
2547         FRGMLSTRK_TableView.val.appendText("Density interface " + i + " = "
+ d1.format(FRGMLSTRK_CalculateValues.d_dep_di_arr[i]) + " (km)" + "\n");
2548     }
2549
2550     FRGMLSTRK_TableView.val.appendText("\n");
2551     FRGMLSTRK_TableView.val.append("Density of each formation :-\n");
2552     FRGMLSTRK_TableView.val.appendText("-----\n")
2553     FRGMLSTRK_TableView.val.appendText("\n");
2554
2555     for (int i = 1; i < FRGMLSTRK_CalculateValues.inter; i++){
2556         FRGMLSTRK_TableView.val.appendText("Density value " + i + " = "
+ d.format(FRGMLSTRK_CalculateValues.d_den_for_arr[i]) + " (gm/cc)" + "\n");
2557     }
2558 }
2559 }
2560 }
2561 -----

```

```

2562 package com.frgmlstrk.control;
2563
2564 import java.awt.event.*;
2565 import java.awt.image.BufferedImage;
2566 import java.awt.*;
2567 import java.io.*;
2568 import java.text.DecimalFormat;
2569 import javax.imageio.ImageIO;
2570 import javax.swing.*;
2571 import com.frgmlstrk.model.FRGMLSTRK_CalculateValues;
2572 import com.frgmlstrk.util.FRGMLSTRK_HandleException;
2573 import com.frgmlstrk.view.FRGMLSTRK_MainPanel;
2574 import com.frgmlstrk.view.graph.FRGMLSTRK_DensityGraph;
2575 import com.frgmlstrk.view.graph.FRGMLSTRK_DepthLines;
2576 import com.frgmlstrk.view.graph.FRGMLSTRK_PlainGraph;
2577 import com.frgmlstrk.view.event.*;
2578
2579
2580
2581 public class FRGMLSTRK_Controller implements ActionListener {
2582
2583     Object rowdata [][] = {};
2584     BufferedImage image;
2585     com.frgmlstrk.view.FRGMLSTRK_DrawGraph dg = new
2586

```

```

com.frgmlstrk.view.FRGMSTRK_DrawGraph();
2587     com.frgmlstrk.model.FRGMSTRK_CalculateValues cv = new
com.frgmlstrk.model.FRGMSTRK_CalculateValues();
    public static boolean success = false;

    public void actionPerformed(ActionEvent ae) {

        if (ae.getActionCommand().equals("Specify depth interfaces")) {

            try{
                if (FRGMSTRK_CalculateValues.len <= FRGMSTRK_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 FRGMSTRK_HandleException();
                }

                FRGMSTRK_MainPanel.img2.removeMouseListener(FRGMSTRK_PlotDepth.ml2);
                FRGMSTRK_MainPanel.img2.removeMouseMotionListener((MouseMotionListener)FRG
LSTRK_PlotDepth.ml2);
                FRGMSTRK_PlotDepth pf = new FRGMSTRK_PlotDepth();
                cv.getAnamolyValues(FRGMSTRK_MainPanel.captureValues());

                cv.depthValues();
                Graphics g = FRGMSTRK_MainPanel.p_Center.getGraphics();
                pf.paint(g);

            }
            catch(Exception e){

            }

        }else if(ae.getActionCommand().equals("Specify density values")){
            try{

                FRGMSTRK_MainPanel.den.removeMouseListener(FRGMSTRK_PlotDensity.ml4);
                FRGMSTRK_MainPanel.den.removeMouseMotionListener((MouseMotionListener)FRGM
STRK_PlotDensity.ml4);
                FRGMSTRK_CalculateValues.inter = 2 + FRGMSTRK_CalculateValues.i_nob_di;
                FRGMSTRK_PlotDensity pf = new FRGMSTRK_PlotDensity();
                Graphics g = FRGMSTRK_MainPanel.p_Center.getGraphics();
                pf.paint(g);

            }
            catch(Exception e) {

            }

        }else if(ae.getActionCommand().equals("Draw/Edit fault plane")){
            FRGMSTRK_MainPanel.fl.removeMouseListener(FRGMSTRK_EditFault.ml1);
            FRGMSTRK_MainPanel.fl.removeMouseMotionListener((MouseMotionListener)FRGMSTRK
EditFault.ml1);
            FRGMSTRK_EditFault ef = new FRGMSTRK_EditFault();
            try{
                if (FRGMSTRK_CalculateValues.len <= FRGMSTRK_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 FRGMSTRK_HandleException();
                }

                Graphics g = FRGMSTRK_MainPanel.p_Center.getGraphics();
                ef.paint(g);

            }
            catch(Exception e) {

            }
            //e.printStackTrace();
        }
    }
}

```

```

2657     }
2658 }else if(ae.getActionCommand().equals("Load file")){
2659
2660     FRGMLSTRK_MainPanel.loadData();
2661     FRGMLSTRK_MainPanel.p_Center.removeAll();
2662     FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.graphLabel);
2663     FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.im);
2664     FRGMLSTRK_PlainGraph pg = new FRGMLSTRK_PlainGraph();
2665     try{
2666         if (FRGMLSTRK_CalculateValues.len <= FRGMLSTRK_CalculateValues.i_d_poly){
2667             JFrame frame = null;
2668             JOptionPane.showMessageDialog(frame,
2669                 "The polynomial fit requires\n"
2670                 + "degree+1 data points.\n"
2671                 + "Use additional data points",
2672                 "Error!",
2673                 JOptionPane.ERROR_MESSAGE);
2674             throw new FRGMLSTRK_HandleException();
2675         }
2676
2677         cv.getAnamolyValues(com.frgmlstrk.view.FRGMLSTRK_MainPanel.captureValues());
2678         cv.getCoefficients();
2679         cv.depthValues();
2680         Graphics2D gl = (Graphics2D)FRGMLSTRK_MainPanel.im.getGraphics();
2681         pg.drawPlainGraph(gl);
2682         dg.plotZCoordinates(gl);
2683         pg.drawPlainGraph(gl);
2684         FRGMLSTRK_MainPanel.im.removeMouseListener(FRGMLSTRK_PlotFault.ml);
2685         FRGMLSTRK_MainPanel.im.removeMouseMotionListener((MouseMotionListener)FRGMI
2686         TRK_PlotFault.ml);
2687     }
2688     catch(Exception e) {
2689         e.printStackTrace();
2690     }
2691 }
2692
2693
2694 }
2695
2696 else if(ae.getActionCommand().equals("Graph")){
2697
2698     FRGMLSTRK_MainPanel.p_Center.removeAll();
2699     FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.img4);
2700     FRGMLSTRK_MainPanel.img4.setEditable(false);
2701     FRGMLSTRK_MainPanel.p_Center.validate();
2702     FRGMLSTRK_MainPanel.clearPanel(FRGMLSTRK_MainPanel.img4);
2703     try
2704     {
2705         int width = 1280;
2706         int height = 650;
2707         BufferedImage buffer = new
2708         BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
2709         Graphics g1= buffer.createGraphics();
2710         g1.setColor(Color.WHITE);
2711         g1.fillRect(0, 0, width, height);
2712         Graphics2D g2 = (Graphics2D)g1 ;
2713         dg.plot(g2);
2714         dg.plotXYCoordinates(g2);
2715         dg.drawGraph(g2);
2716         dg.drawDepth(g2);
2717         dg.plotZCoordinates(g2);
2718         //dg.plotXYCoordinates(g2);
2719         dg.drawDen(g2);
2720         dg.plot(g2);
2721         dg.idx(g2);
2722
2723         FileOutputStream os = new FileOutputStream(
2724         FRGMLSTRK_CalculateValues.input_area_name + ".jpg");
2725         ImageIO.write(buffer, "jpg", os);
2726         os.close();
2727
2728         String path = FRGMLSTRK_CalculateValues.input_area_name + ".jpg";
2729         image = ImageIO.read(new File(path));
2730     }

```

```

2729         Graphics g_image = FRGMLSTRK_MainPanel.img4.getGraphics();
2730         g_image.drawImage(image, 0, 0, image.getWidth(), image.getHeight(), dg);
2731
2732         MouseListener ml3 = new MouseAdapter(){
2733             public void mouseClicked(MouseEvent e){
2734                 Graphics g_image = FRGMLSTRK_MainPanel.img4.getGraphics();
2735                 g_image.drawImage(image, 0, 0, image.getWidth(),
image.getHeight(), dg);
2736             }
2737         };
2738         FRGMLSTRK_MainPanel.img4.addMouseListener(ml3);
2739     }
2740     catch (Exception e2) {
2741
2742         e2.printStackTrace();
2743     }
2744
2745 }else if(ae.getActionCommand().equals("Save and Print")){
2746
2747     try{
2748
2749         File img_file = new File(FRGMLSTRK_CalculateValues.input_area_name+".jpg");
2750         JFileChooser saveFile = new JFileChooser();
2751         File OutFile = saveFile.getSelectedFile();//new File(new
File(INGRLSTRK_CalculateValues1.input_area_name+".html").getCanonicalPath())
2752         ; //saveFile.getSelectedFile();
2753         FileWriter myWriter = null;
2754
2755         if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION){
2756
2757             OutFile = saveFile.getSelectedFile();
2758
2759             if (OutFile.canWrite() || !OutFile.exists()){
2760
2761                 File dir = new File(OutFile.getParent());
2762
2763                 success = img_file.renameTo(new File(dir, img_file.getName()));
2764                 myWriter = new FileWriter(OutFile+".html");
2765                 myWriter.write("<html> <Body onLoad = \"window.print()\"><table>
<tr> <td>" +
2766
2767                     "<table border = 1> <tr> <th colspan = 4>LOCATION:-
"+FRGMLSTRK_CalculateValues.input_area_name+"</th> </tr>");
2768
2769                 DecimalFormat df =new DecimalFormat("0.###");
2770                 DecimalFormat d = new DecimalFormat("0.##");
2771                 DecimalFormat fl = new DecimalFormat("0.#");
2772                 myWriter.write(" <tr><th colspan = 4> PROFILE NUMBER:-"+
"+FRGMLSTRK_CalculateValues.input_profile_num+" </th></tr>");
2773                 myWriter.write(" <tr> <th>Distance (km) </th><th> Observed anamolies
(mGal) </th> <th> Calculated anamolies (mGal) </th> </tr>");
2774
2775                 for ( int K = 1; K <= FRGMLSTRK_CalculateValues.i_no_obs; K++){
2776
2777                     myWriter.write(" <tr> <td>" +
d.format(FRGMLSTRK_CalculateValues.x[K])+"</td><td>" +df.format(F
RGMLSTRK_CalculateValues.input_nob_gob[K])+"</td>
<td>" +df.format(FRGMLSTRK_CalculateValues.ano[K])+"</td></tr>");
2778                 }
2779                 myWriter.write(" </table> </td> <td> <img src = '"+
FRGMLSTRK_CalculateValues.input_area_name
+".jpg"></td></tr></table><BR>Coordinates of selected points on the
fault plane(x,z): <BR>");
2780                 myWriter.write("----- <BR>");
2781
2782                 DecimalFormat dl =new DecimalFormat("0.####");
2783
2784                 for (int i = 1; i <= FRGMLSTRK_CalculateValues.len; i++){
2785
2786                     myWriter.write(" (" +d.format(FRGMLSTRK_CalculateValues.d_x_km_a
[i])+", "+d.format(FRGMLSTRK_CalculateValues.d_z_km_arr[i])+" )"+
"<BR>") ;
2787
2788                 }

```

```

2788         myWriter.write("<BR>");
2789         myWriter.write("Coefficient Values for the fault plane::"+<BR>");
2790         myWriter.write("-----<BR>");
2791         for ( int i = 1; i < FRGMLSTRK_CalculateValues.cftnt .length; i++)
2792             myWriter.write("f+ ( i - 1 ) + " =
2793 "+d1.format(FRGMLSTRK_CalculateValues.cftnt[i])+<BR>");
2794     }
2795     myWriter.write("<BR>");
2796     myWriter.write("Depths of density interfaces:"+<BR>");
2797     myWriter.write("-----<BR>");
2798     for (int i = 1; i <= FRGMLSTRK_CalculateValues.inter; i++){
2799         myWriter.write("Density interface "+i+" = "
+fl.format(FRGMLSTRK_CalculateValues.d_dep_di_arr[i])+" (km)" +
"<BR>");
2800     }
2801
2802     myWriter.write("<BR>");
2803     myWriter.write("Density of each formation:"+<BR>");
2804     myWriter.write("-----<BR>");
2805     for (int i = 1; i < FRGMLSTRK_CalculateValues.inter; i++){
2806         myWriter.write("Density value "+i+" = "
+d.format(FRGMLSTRK_CalculateValues.d_den_for_arr[i])+"
(gm/cc)" + "<BR>");
2807     }
2808     myWriter.close();
2809 }
2810 }
2811 else
2812 {
2813     //pops up error message
2814 }
2815 }
2816 }
2817 }
2818 catch(Exception e1) {
2819     e1.printStackTrace();
2820 }
2821 }
2822 }
2823 else if (ae.getActionCommand().equals("Sample data")){
2824     FRGMLSTRK_MainPanel.setDefaultValues();
2825 }
2826 }
2827 else if (ae.getActionCommand().equals("Save file")){
2828     try{
2829
2830         JFileChooser saveFile = new JFileChooser();
2831         File OutFile = saveFile.getSelectedFile();//new File(new
File(INGRLSTRK_CalculateValues1.input_area_name+".html").getCanonicalPath())
; //saveFile.getSelectedFile();
2832         DecimalFormat d = new DecimalFormat("0.##");
2833         FileWriter myWriter = null;
2834         if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION){
2835
2836             OutFile = saveFile.getSelectedFile();
2837
2838             if (OutFile.canWrite() || !OutFile.exists()){
2839
2840                 myWriter = new FileWriter(OutFile+".txt");
2841                 myWriter.write(" "+FRGMLSTRK_CalculateValues.input_area_name);
2842                 myWriter.append(System.getProperty("line.separator"));
2843                 myWriter.write(" "+FRGMLSTRK_CalculateValues.input_profile_num);
2844                 myWriter.append(System.getProperty("line.separator"));
2845                 myWriter.write(" "+FRGMLSTRK_CalculateValues.d_str_st);
2846                 myWriter.append(System.getProperty("line.separator"));
2847                 myWriter.write(" "+FRGMLSTRK_CalculateValues.i_nob_di);
2848                 myWriter.append(System.getProperty("line.separator"));
2849                 myWriter.write(" "+FRGMLSTRK_CalculateValues.d_max_dep);
2850                 myWriter.append(System.getProperty("line.separator"));
2851                 myWriter.write(" "+FRGMLSTRK_CalculateValues.d_base_den);
2852                 myWriter.append(System.getProperty("line.separator"));
2853                 myWriter.write(" "+FRGMLSTRK_CalculateValues.i_no_obs);
2854

```

```

2856 myWriter.append(System.getProperty("line.separator"));
2857 myWriter.write(" "+FRGMLSTRK_CalculateValues.d_off_pro);
2858 myWriter.append(System.getProperty("line.separator"));
2859
2860 for (int i = 1; i <= FRGMLSTRK_CalculateValues.i_no_obs; i++){
2861
2862     myWriter.write(" "+(FRGMLSTRK_CalculateValues.d_dis_km[i])+",")
2863 }
2864
2865 myWriter.append(System.getProperty("line.separator"));
2866 for (int i = 1; i <= FRGMLSTRK_CalculateValues.i_no_obs; i++){
2867
2868     myWriter.write(" "+FRGMLSTRK_CalculateValues.d_ele_km[i]+",") ;
2869
2870 }
2871 myWriter.append(System.getProperty("line.separator"));
2872 for (int i = 1; i <= FRGMLSTRK_CalculateValues.i_no_obs; i++){
2873
2874     myWriter.write(" "+(FRGMLSTRK_CalculateValues.input_nob_gob[i])·
2875 ,") ;
2876
2877 }
2878 myWriter.append(System.getProperty("line.separator"));
2879 myWriter.write(" "+FRGMLSTRK_CalculateValues.i_d_poly);
2880 myWriter.append(System.getProperty("line.separator"));
2881 myWriter.write(" "+FRGMLSTRK_CalculateValues.input_max_den);
2882 myWriter.append(System.getProperty("line.separator"));
2883 myWriter.write(" "+FRGMLSTRK_CalculateValues.input_min_den);
2884 myWriter.append(System.getProperty("line.separator"));
2885 for (int i = 1; i <= FRGMLSTRK_CalculateValues.len; i++){
2886
2887     myWriter.write(" "+d.format(FRGMLSTRK_CalculateValues.d_x_km_ar
2888 i)+"",") ;
2889
2890 }
2891 myWriter.append(System.getProperty("line.separator"));
2892
2893 for (int i = 1; i <= FRGMLSTRK_CalculateValues.len; i++){
2894
2895     myWriter.write(" "+d.format(FRGMLSTRK_CalculateValues.d_z_km_ar
2896 i)+"",") ;
2897
2898 }
2899 myWriter.append(System.getProperty("line.separator"));
2900 myWriter.write(" "+FRGMLSTRK_CalculateValues.len);
2901 myWriter.append(System.getProperty("line.separator"));
2902 for (int i = 1; i <= FRGMLSTRK_CalculateValues.inter; i++){
2903
2904     myWriter.write(" "+d.format(FRGMLSTRK_CalculateValues.d_dep_di_
2905 r[i])+",") ;
2906
2907 }
2908 myWriter.append(System.getProperty("line.separator"));
2909 for (int i = 1; i < FRGMLSTRK_CalculateValues.inter; i++){
2910
2911     myWriter.write(" "+d.format(FRGMLSTRK_CalculateValues.d_den_for_
2912 rr[i])+",") ;
2913
2914 }
2915 myWriter.append(System.getProperty("line.separator"));
2916 myWriter.write(" "+FRGMLSTRK_CalculateValues.inter);
2917 myWriter.close();
2918
2919 }
2920
2921 }
2922 else
2923 {
2924     //pops up error message
2925
2926 }
2927
2928 }
2929 catch(Exception e1) {
2930
2931     e1.printStackTrace();
2932
2933 }

```



```

2926     }
2927
2928     else if (ae.getActionCommand().equals("Clear")){
2929
2930         FRGMLSTRK_MainPanel.clearDefaultValues();
2931         FRGMLSTRK_MainPanel.p_Center.removeAll();
2932         com.frgmlstrk.view.FRGMLSTRK_TableView.populateEastPanel(rowdata);
2933         com.frgmlstrk.view.FRGMLSTRK_TableView.val.setText("");
2934
2935     }else if(ae.getActionCommand().equals("Exit")){
2936
2937         JFrame frame = null;
2938         int r = JOptionPane.showConfirmDialog(
2939             frame,
2940             "Exit FGMLSTRK ?",
2941             "Confirm Exit ",
2942             JOptionPane.YES_NO_OPTION);
2943         if(r == JOptionPane.YES_OPTION ){
2944             if(success==false){
2945                 String fileName = FRGMLSTRK_CalculateValues.input_area_name+".jpg";
2946                 File f = new File(fileName);
2947                 f.delete();
2948             }
2949             System.exit(0);
2950         }
2951     }else if(ae.getActionCommand().equals("Forward Modeling")){
2952
2953         FRGMLSTRK_MainPanel.img3.removeMouseListener(FRGMLSTRK_EditDepth.m11);
2954         FRGMLSTRK_MainPanel.img3.removeMotionListener((MouseListener)FRGMLSTRK_
K_EditDepth.m11);
2955         FRGMLSTRK_MainPanel.p_Center.removeAll();
2956         FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.graphLabel);
2957         FRGMLSTRK_MainPanel.p_Center.add(FRGMLSTRK_MainPanel.img3);
2958         FRGMLSTRK_MainPanel.img3.setEditable(false);
2959         FRGMLSTRK_MainPanel.p_Center.validate();
2960         cv.getAnamolyValues(FRGMLSTRK_MainPanel.captureValues());
2961         FRGMLSTRK_CalculateValues.inter = 2 + FRGMLSTRK_CalculateValues.i_nob_di;
2962         FRGMLSTRK_PlotDepth.val2[1] = 0;
2963         FRGMLSTRK_PlotDepth.val2[FRGMLSTRK_CalculateValues.inter] =
FRGMLSTRK_CalculateValues.d_max_dep;
2964
2965         try {
2966             cv.cal();
2967             com.frgmlstrk.view.FRGMLSTRK_TableView.populateEastPanel(FRGMLSTRK_Calculat
Values.obj);
2968             drawGraph(FRGMLSTRK_MainPanel.img3);
2969         } catch (Exception e) {
2970
2971         }
2972
2973
2974         Graphics g = FRGMLSTRK_MainPanel.p_Center.getGraphics();
2975         FRGMLSTRK_EditDensity eden = new FRGMLSTRK_EditDensity();
2976         eden.paint(g);
2977         FRGMLSTRK_EditDepth ed = new FRGMLSTRK_EditDepth();
2978         ed.paint(g);
2979
2980         Graphics2D g2 = (Graphics2D)FRGMLSTRK_MainPanel.img3.getGraphics();
2981         g2.setFont(new Font("Arial", 40,20));
2982         g2.setColor(Color.black);
2983         g2.drawString("Number of coordinates "+FRGMLSTRK_CalculateValues.len, 710, 120);
2984         g2.drawLine(655, 0, 655, 85);
2985         g2.drawLine(655, 85, 990, 85);
2986         g2.drawString("Instructions",730, 20);
2987         g2.drawString("_____ ", 710, 20);
2988         g2.setFont(new Font("Arial", 40, 12));
2989         g2.setColor(Color.red);
2990         g2.drawString(" For modeling, press & drag the line segments of", 660, 40);
2991         g2.drawString(" depth/and density and release the mouse at",660,60);
2992         g2.drawString(" desired location", 660, 80);
2993         g2.setColor(Color.BLACK);
2994
2995
2996     }else if(ae.getActionCommand().equals("Specify fault coordinates")){
2997

```



```

2998         cv.getAnamolyValues(FRGMLSTRK_MainPanel.captureValues());
2999         if(FRGMLSTRK_CalculateValues.d_dis_km.length -
FRGMLSTRK_CalculateValues.i_no_obs != 1){
3000
3001             JFrame frame = null;
3002             JOptionPane.showMessageDialog(frame,
3003                 "Distance values must be equal to\n"
3004                 +"number of observations.",
3005                 "Out of bounds error",
3006                 JOptionPane.ERROR_MESSAGE);
3007             try {
3008
3009                 throw new FRGMLSTRK_HandleException();
3010
3011             } catch (FRGMLSTRK_HandleException e) {
3012
3013             }
3014         }
3015         if (FRGMLSTRK_CalculateValues.d_ele_km.length -
FRGMLSTRK_CalculateValues.i_no_obs != 1){
3016
3017             JFrame frame = null;
3018             JOptionPane.showMessageDialog(frame,
3019                 "Elevation values must be equal to\n"
3020                 +"number of observations.",
3021                 "Out of bounds error",
3022                 JOptionPane.ERROR_MESSAGE);
3023             try {
3024
3025                 throw new FRGMLSTRK_HandleException();
3026
3027             } catch (FRGMLSTRK_HandleException e) {
3028
3029             }
3030         }
3031
3032         if (FRGMLSTRK_CalculateValues.input_nob_gob.length -
FRGMLSTRK_CalculateValues.i_no_obs != 1){
3033
3034             JFrame frame = null;
3035             JOptionPane.showMessageDialog(frame,
3036                 "Observed anomalies must be equal to\n"
3037                 +"number of observations.",
3038                 "Out of bounds error",
3039                 JOptionPane.ERROR_MESSAGE);
3040             try {
3041
3042                 throw new FRGMLSTRK_HandleException();
3043
3044             } catch (FRGMLSTRK_HandleException e) {
3045
3046             }
3047         }
3048         FRGMLSTRK_MainPanel.im.removeMouseListener(FRGMLSTRK_PlotFault.ml);
3049         FRGMLSTRK_MainPanel.im.removeMouseMotionListener((MouseMotionListener)FRGMLSTRK
PlotFault.ml);
3050         FRGMLSTRK_PlotFault pf = new FRGMLSTRK_PlotFault();
3051         Graphics g = FRGMLSTRK_MainPanel.p_Center.getGraphics();
3052         pf.paint(g);
3053
3054     }
3055 }
3056
3057 public void drawGraph(TextArea t){
3058
3059     FRGMLSTRK_DensityGraph de = new FRGMLSTRK_DensityGraph();
3060     FRGMLSTRK_DepthLines dl = new FRGMLSTRK_DepthLines();
3061     Graphics2D g2 = (Graphics2D)t.getGraphics();
3062     FRGMLSTRK_MainPanel.clearPanel(t);
3063     dg.plot(g2);
3064     dg.plotXYCoordinates(g2);
3065     dg.drawGraph(g2);
3066     dl.drawDepthLine(g2);
3067     dg.plotZCoordinates(g2);
3068     dg.plot(g2);

```

```

3069         de.Plainden(g2);
3070     }
3071 }
3072 }
3073 -----
3074 package com.frgmlstrk.util;
3075
3076 import javax.swing.JFrame;
3077 import javax.swing.JOptionPane;
3078
3079
3080
3081 public class FRGMLSTRK_Utility {
3082
3083     public static double convertDouble(String str) throws Exception {
3084
3085
3086         Double temp = null;
3087
3088         try {
3089             temp = new Double(str.trim());
3090
3091         }
3092         catch(Exception e){
3093
3094             JFrame frame = null;
3095             JOptionPane.showMessageDialog(frame,
3096                 "Enter a numerical value.",
3097                 "Number format error",
3098                 JOptionPane.ERROR_MESSAGE);
3099
3100         }
3101         return temp.doubleValue();
3102     }
3103
3104     public static String convertString(String str) throws Exception {
3105
3106         String temp = new String(str.trim());
3107         return temp;
3108     }
3109
3110     public static int convertInteger(String str) throws Exception {
3111
3112
3113         Integer temp = null;
3114         try {
3115             temp = new Integer(str.trim());
3116
3117         }
3118         catch(Exception e){
3119
3120             JFrame frame = null;
3121             JOptionPane.showMessageDialog(frame,
3122                 "Enter a numerical value.",
3123                 "Number format error",
3124                 JOptionPane.ERROR_MESSAGE);
3125
3126         }
3127         return temp.intValue();
3128     }
3129
3130     public static int findMaximumNumber( double observe[]) {
3131
3132         double max = 0.0d;
3133         for (int i = 0; i < observe.length; i++) {
3134
3135             if (Math.abs(observe[i]) > Math.abs(max)) {
3136
3137                 max = observe[i];
3138             }
3139         }
3140
3141         int maxVal = (int) max/3*5;
3142

```

```

3143         return maxVal;
3144     }
3145
3146     public static double findMinimumNumber( double observe[], double denVal) {
3147
3148         double max = denVal;
3149         for (int i = 1; i < observe.length; i++) {
3150
3151             if (Math.abs(observe[i]) < Math.abs(max)) {
3152
3153                 max = Math.abs(observe[i]);
3154             }
3155         }
3156
3157         double maxVal = max;
3158         return maxVal;
3159     }
3160
3161     public static double findMinimumNumber1( double observe[]) {
3162
3163         double max = 0.0d;
3164         for (int i = 1; i < observe.length; i++) {
3165
3166             if ((observe[i]) < (max)) {
3167
3168                 max = (observe[i]);
3169             }
3170         }
3171
3172         double maxVal = max;
3173         return maxVal;
3174     }
3175
3176     public static double findMaximumNumber1( double observe[]) {
3177
3178         double max = 0.0d;
3179         for (int i = 1; i < observe.length; i++) {
3180
3181             if (Math.abs(observe[i]) > Math.abs(max)) {
3182
3183                 max = Math.abs(observe[i]);
3184             }
3185         }
3186
3187         double maxVal = max;
3188         return maxVal;
3189     }
3190
3191     public static double findMaximumNumber( double observe[], double anoVal) {
3192
3193         double max = anoVal;
3194         for (int i = 1; i < observe.length; i++) {
3195
3196             if ((observe[i]) > (max)) {
3197
3198                 max = (observe[i]);
3199             }
3200         }
3201
3202         double maxVal = max;
3203         return maxVal;
3204     }
3205
3206     public static double[] convertDoubleArray(String str) throws Exception {
3207
3208         java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
3209         String temp = "";
3210         java.util.ArrayList arr = new java.util.ArrayList();
3211
3212         while(st.hasMoreTokens()) {
3213
3214             temp = st.nextToken();
3215             arr.add(temp);
3216         }
3217         double d_array[] = new double[arr.size() + 1];

```

```

3218
3219         for (int i = 0; i <= arr.size(); i++) {
3220
3221             if (i == 0)
3222                 d_array[i] = 0.0;
3223             else {
3224
3225                 try {
3226                     d_array[i] = convertDouble( arr.get(i - 1).toString() );
3227                 }
3228                 catch(Exception e){
3229                     JFrame frame = null;
3230                     JOptionPane.showMessageDialog(frame,
3231                         "Enter numerical values.",
3232                         "Number format error",
3233                         JOptionPane.ERROR_MESSAGE);
3234                     throw new FRGMLSTRK_HandleException();
3235                 }
3236             }
3237         }
3238     }
3239     return d_array;
3240 }
3241 }
3242 -----
3243
3244 package com.frgmlstrk.util;
3245
3246 import java.awt.*;
3247 import com.frgmlstrk.view.FRGMLSTRK_MainPanel;
3248
3249 public class FRGMLSTRK_HandleException extends Exception{
3250
3251     /**
3252     *
3253     */
3254     private static final long serialVersionUID = 1L;
3255
3256     public FRGMLSTRK_HandleException(){
3257         Graphics g = FRGMLSTRK_MainPanel.p_Center.getGraphics();
3258         g.setColor(Color.white);
3259         g.fillRect(0, 0, 1000, 600);
3260         g.setColor(Color.black);
3261         g.setFont(new Font("Arial", 20, 40));
3262         g.drawString("ERROR...", 400, 325);
3263     }
3264 }
3265 -----

```

```

1  package com.gravlis.view;
2
3  import java.awt.Frame;
4  import java.awt.event.MouseAdapter;
5  import java.awt.event.MouseEvent;
6  import java.awt.event.WindowAdapter;
7  import java.awt.event.WindowEvent;
8  import java.io.File;
9
10 import javax.swing.JFrame;
11 import javax.swing.JOptionPane;
12
13 import com.gravlis.control.GRAVLIS_Controller;
14 import com.gravlis.model.GRAVLIS_CalculateValues;
15
16
17
18
19 public class GRAVLIS_MainView extends Frame{
20     /**
21      *
22      */
23     private static final long serialVersionUID = 1L;
24
25     public static void main(String s[]){
26
27         GRAVLIS_MainView cm = new GRAVLIS_MainView();
28         cm.setSize(1280, 768);
29         cm.addWindowListener(new WindowAdapter(){
30             public void windowClosing(WindowEvent e){
31                 JFrame frame = null;
32                 int r = JOptionPane.showConfirmDialog(
33                     frame,
34                     "Exit GRAVLIS ?",
35                     "Confirm Exit ",
36                     JOptionPane.YES_NO_OPTION);
37                 if(r == JOptionPane.YES_OPTION ){
38                     if(GRAVLIS_Controller.success==false){
39                         String fileName = GRAVLIS_CalculateValues.input_area_name+".jpg";
40                         File f = new File(fileName);
41                         f.delete();
42                     }
43                     System.exit(0);
44                 }
45             }
46         });
47         cm.setTitle("GRAVLIS");
48         cm.setResizable(false);
49         cm.add(new GRAVLIS_MainPanel());
50         GRAVLIS_MainPanel.img.addMouseListener(new MouseAction());
51         cm.setVisible(true);
52     }
53 }
54
55
56
57 class MouseAction extends MouseAdapter{
58     public void mousePressed(MouseEvent e) {
59
60         GRAVLIS_CalculateValues.drawGraph();
61     }
62 }
63
64
65 -----
66
67 package com.gravlis.view;
68
69 import java.awt.*;
70 import java.io.File;
71 import java.io.IOException;
72 import java.util.HashMap;
73 import javax.swing.JFileChooser;
74 import javax.swing.JFileChoo

```

```

75 import jxl.CellType;
76 import jxl.Sheet;
77 import jxl.Workbook;
78 import jxl.read.biff.BiffException;
79
80
81 public class GRAVLIS_MainPanel extends Panel {
82
83     /**
84      *
85      */
86     private static final long serialVersionUID = 1L;
87
88     public static TextArea img = new TextArea(46,140);
89     Panel p_North, p_West;
90     public static Panel p_East;
91
92     static Panel p_South;
93
94     public static Panel p_Center;
95     static TextField inputValues [] = new TextField[16];
96
97     Button actionButton[] = new Button[7];
98     String rowdata[][]={};
99
100     /**Field Area Name*/
101     final static int AREA_FE = 0;
102     /**Number of the Profile*/
103     final static int NUM_PROFILE = 1;
104     /**Number of observations*/
105     final static int N_OBS = 2 ;
106     /**Distance(km)*/
107     final static int X_KM = 3;
108     /**Elevation(km)*/
109     final static int ELE_KM = 4;
110     /**observed anomalies*/
111     final static int NOB_GOB = 5;
112     /**Y-values*/
113     final static int Y_KM = 6;
114     /** Surface density contrast (gm/cc) */
115     final static int SD_POLY = 7;
116     /**STRIKE-values*/
117     final static int STRIKE_KM = 8;
118     /**Maximum Depth*/
119     final static int Z_VAL = 9;
120     /**Number of formations*/
121     final static int NUM_FOR = 10;
122     /**Minimum Values*/
123     final static int MIN_VAL = 11;
124     /**Maximum values*/
125     final static int MAX_VAL = 12;
126     /**Number of iteration values*/
127     final static int NUM_ITE =13;
128     /**Density values*/
129     final static int DEN_VAL = 14;
130     /**Allowable Error*/
131     final static int AL_ERR = 15;
132
133     public GRAVLIS_MainPanel(){
134
135         this.setLayout(new BorderLayout());
136         p_North = new Panel();
137         p_West = new Panel();
138         p_East = new Panel ();
139         p_South = new Panel();
140         p_Center = new Panel();
141
142         Label graphLabel = new Label("INVERSION OF GRAVITY ANOMALIES OF 2.5D LISTRIC FAULT
STRUCTURES USING ARBITRARY DENSITY DEPTH VARIATIONS", Label.CENTER);
143         graphLabel.setFont(new Font("Bold", 1, 15));
144         p_Center.add(graphLabel);
145
146         for(int i = 0; i < 16; i++){
147             inputValues[i] = new TextField();
148         }

```

```

149
150 p_North.setFont(new Font("Bold",1,12));
151 actionPerformed[0] = new Button("Load data");
152 actionPerformed[1] = new Button("Interpretation with Fixed Depth");
153 actionPerformed[2] = new Button("Interpretation with Fixed Density");
154 actionPerformed[3] = new Button("Save & Print");
155 actionPerformed[4] = new Button("Clear");
156 actionPerformed[5] = new Button("Exit");
157
158 this.populateNorthPanel();
159 GRAVLIS_TableView.populateEastPanel(rowdata);
160
161 this.add(p_North, BorderLayout.NORTH);
162 p_Center.setSize(1200, 760);
163 this.add(p_Center, BorderLayout.CENTER);
164 img.setEditable(false);
165 p_Center.add(img);
166 this.add(p_East, BorderLayout.EAST);
167 this.setVisible(true);
168 }
169
170 public void populateNorthPanel(){
171     p_North.setLayout(new GridLayout(1,6));
172
173     p_North.add(actionButton[0]);
174     p_North.add(actionButton[1]);
175     p_North.add(actionButton[2]);
176     p_North.add(actionButton[3]);
177     p_North.add(actionButton[4]);
178     p_North.add(actionButton[5]);
179
180     actionPerformed[0].addActionListener(new com.gravlis.control.GRAVLIS_Controller());
181     actionPerformed[1].addActionListener(new com.gravlis.control.GRAVLIS_Controller());
182     actionPerformed[2].addActionListener(new com.gravlis.control.GRAVLIS_Controller());
183     actionPerformed[3].addActionListener(new com.gravlis.control.GRAVLIS_Controller());
184     actionPerformed[4].addActionListener(new com.gravlis.control.GRAVLIS_Controller());
185     actionPerformed[5].addActionListener(new com.gravlis.control.GRAVLIS_Controller());
186 }
187
188 public static HashMap captureValues(){
189
190     HashMap h_Map = new HashMap();
191
192     try {
193         h_Map.put("AREA_FE",inputValues[AREA_FE].getText());
194         h_Map.put("NUM_PROFILE", inputValues[NUM_PROFILE].getText());
195         h_Map.put("N_OBS", inputValues[N_OBS].getText());
196         h_Map.put("X_KM", inputValues[X_KM].getText());
197         h_Map.put("ELE_KM", inputValues[ELE_KM].getText());
198         h_Map.put("NOB_GOB", inputValues[NOB_GOB].getText());
199         h_Map.put("Y_KM", inputValues[Y_KM].getText());
200         h_Map.put("SD_POLY", inputValues[SD_POLY].getText());
201         h_Map.put("STRIKE_KM", inputValues[STRIKE_KM].getText());
202         h_Map.put("Z_VAL", inputValues[Z_VAL].getText());
203         h_Map.put("NUM_FOR", inputValues[NUM_FOR].getText());
204         h_Map.put("MIN_VAL", inputValues[MIN_VAL].getText());
205         h_Map.put("MAX_VAL", inputValues[MAX_VAL].getText());
206         h_Map.put("DEN_VAL", inputValues[DEN_VAL].getText());
207         h_Map.put("NUM_ITE", inputValues[NUM_ITE].getText());
208         h_Map.put("AL_ERR", inputValues[AL_ERR].getText());
209     }
210     catch (Exception e) {
211         e.printStackTrace();
212     }
213
214     return h_Map;
215 }
216
217
218 public static void clearPanel(Panel p) {
219
220     Graphics g = p.getGraphics();
221     g.setColor(Color.WHITE);
222     g.fillRect(0, 30, 1280, 650);
223 }

```

```

224
225
226     public static void loadData()throws IOException {
227         try{
228
229             String current = System.getProperty("user.dir");
230             JFileChooser chooser=new JFileChooser(current);
231             int returnVal = chooser.showOpenDialog(null);
232             String dis[], ele[], gobs[], min[], max[], den[], dep[];
233             String disval = ""
,eleval="",gobsval="",minval="",maxval="",denval="",depval="";
234             Workbook w;
235
236             if(returnVal == JFileChooser.APPROVE_OPTION) {
237                 File f = chooser.getSelectedFile();
238                 w = Workbook.getWorkbook(f);
239                 Sheet sheet = w.getSheet(0);
240                 dis = new String[sheet.getRows()+1];
241                 ele = new String[sheet.getRows()+1];
242                 gobs = new String[sheet.getRows()+1];
243                 min = new String[sheet.getRows()+1];
244                 max = new String[sheet.getRows()+1];
245                 den = new String[sheet.getRows()+1];
246                 dep = new String[sheet.getRows()+1];
247                 for (int j = 0; j < sheet.getColumns(); j++) {
248                     for (int i = 1; i < sheet.getRows(); i++) {
249                         Cell cell = sheet.getCell(j, i);
250                         CellType type = cell.getType();
251                         if (type == CellType.LABEL) {
252                             GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.AREA_FE].setTe
(cell.getContents());
253                         }
254                         if (type == CellType.NUMBER) {
255                             if (j==1){
256                                 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.NUM_PROFI
.set
.setText(cell.getContents());
257                             }
258                             if (j==2){
259                                 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.N_OBS].set
xt(cell.getContents());
260                             }
261                             if (j==3){
262                                 dis[i] = cell.getContents()+" ";
263                                 disval = disval+dis[i];
264                             }
265                             if (j==4){
266                                 ele[i] = cell.getContents()+" ";
267                                 eleval = eleval+ele[i];
268                             }
269                             if (j==5){
270                                 gobs[i] = cell.getContents()+" ";
271                                 gobsval = gobsval+gobs[i];
272                             }
273                             if (j==6){
274                                 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.STRIKE_KM]
etText(cell.getContents());
275                             }
276                             if (j==7){
277                                 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.Y_KM].setT
t(cell.getContents());
278                             }
279                             if (j==8){
280                                 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.NUM_FOR].se
Text(cell.getContents());
281                             }
282                             if (j==9){
283                                 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.SD_POLY].se
Text(cell.getContents());
284                             }
285                             if (j==10){
286                                 den[i] = cell.getContents()+" ";
287                                 denval = denval+den[i];
288                             }
289                             if (j==11){
290                                 dep[i] = cell.getContents()+" ";

```



```

291         depval = depval+dep[i];
292     }
293     if (j==12){
294         GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.NUM_ITE].se
Text(cell.getContents());
295     }
296     if (j==13){
297         GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.AL_ERR].set
ext(cell.getContents());
298     }
299     try{
300         if (j==14){
301             min[i] = cell.getContents()+" ";
302             minval = minval+min[i];
303         }
304         if (j==15){
305             max[i] = cell.getContents()+" ";
306             maxval = maxval+max[i];
307         }
308     }
309     catch(Exception e){
310         //NullPointerException
311     }
312 }
313 }
314 }
315 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.X_KM].setText(" "+disval);
316 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.ELE_KM].setText(" "+elevall);
317 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.NOBOB].setText(" "+gobbsval);
;
318 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.MIN_VAL].setText(" "+minval);
319 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.MAX_VAL].setText(" "+maxval);
320 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.DEN_VAL].setText(" "+denval);
321 GRAVLIS_MainPanel.inputValues[GRAVLIS_MainPanel.Z_VAL].setText(" "+depval);
322 }
323 }
324 catch (BiffException e) {
325     e.printStackTrace();
326 }
327
328
329
330 }
331
332 public static void clearDefaultValues(){
333
334     inputValues[AREA_FE].setText(" ");
335     inputValues[NUM_PROFILE].setText(" ");
336     inputValues[N_OBS].setText(" ");
337     inputValues[X_KM].setText(" ");
338     inputValues[ELE_KM].setText(" ");
339     inputValues[NOB_GOB].setText(" ");
340     inputValues[Y_KM].setText(" ");
341     inputValues[SD_POLY].setText(" ");
342     inputValues[STRIKE_KM].setText(" ");
343     inputValues[Z_VAL].setText(" ");
344     inputValues[NUM_FOR].setText(" ");
345     inputValues[NUM_ITE].setText(" ");
346     inputValues[AL_ERR].setText(" ");
347 }
348 }
349 -----
350 package com.gravlis.view;
351
352 import java.awt.Color;
353 import java.awt.Dimension;
354 import java.awt.Font;
355 import java.awt.Graphics;
356 import java.awt.GridLayout;
357 import java.awt.Panel;
358

```

```

359 import java.awt.TextArea;
360
361 import javax.swing.JScrollPane;
362 import javax.swing.JTable;
363
364
365 public class GRAVLIS_TableView extends Panel{
366
367     /**
368     *
369     */
370     private static final long serialVersionUID = 1L;
371     public static TextArea val = new TextArea(5,5);
372     public static void populateEastPanel(Object rowData[][]) {
373         com.gravlis.view.GRAVLIS_MainPanel.p_East.removeAll();
374
375         com.gravlis.view.GRAVLIS_MainPanel.p_East.setLayout(new GridLayout(2,1));
376
377
378         Object columnNames[] = {"Distance(km)", "Observed anomalies (mGal)", "Calculated
379 anomalies (mGal)", "Error (mGal)"};
380         JTable table = new JTable(rowData, columnNames);
381         table.setPreferredScrollableViewportSize(new Dimension(300,500));
382         JScrollPane scrollPane = new JScrollPane(table);
383         scrollPane.setAutoscrolls(true);
384
385         com.gravlis.view.GRAVLIS_MainPanel.p_East.add(scrollPane);
386         val.setEditable(false);
387         com.gravlis.view.GRAVLIS_MainPanel.p_East.add(val);
388         try{
389             com.gravlis.view.GRAVLIS_MainPanel.p_East.validate();
390         }
391         catch(Exception e){
392
393             Graphics g = GRAVLIS_MainPanel.img.getGraphics();
394             g.setColor(Color.white);
395             g.fillRect(0, 0, 1000, 600);
396             g.setColor(Color.black);
397             g.setFont(new Font("Arial", 20, 40));
398             g.drawString("ERROR...", 300, 400);
399         }
400         com.gravlis.view.GRAVLIS_MainPanel.p_East.setVisible(true);
401
402     }
403
404 }
405
406 }
407
408 -----
409
410 package com.gravlis.view;
411
412 import java.awt.*;
413 import java.applet.*;
414 import java.awt.geom.Line2D;
415 import java.awt.geom.Rectangle2D;
416 import java.text.DecimalFormat;
417 import java.util.Arrays;
418
419 import com.gravlis.model.GRAVLIS_CalculateValues;
420 import com.gravlis.util.GRAVLIS_Utility;
421
422
423 public class GRAVLIS_DrawGraph extends Applet {
424
425     /**
426     *
427     */
428     private static final long serialVersionUID = 1L;
429     public static int i_no_obs;
430
431

```

```

432     float maxY,maxZ,maxX ;
433     public static float strike,strike1,strike2;
434     double obs[];
435
436     public void drawGraph(Graphics2D g2) {
437
438         g2.setFont(new Font("Arial", 20, 12));
439         g2.setColor(Color.BLACK);
440         g2.draw(new Line2D.Float(215-strike1, 50, 215-strike1, 300));
441         String []a = {"A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
442         String []b = {"D", "E", "P", "T", "H", "(k", "m)"};
443         for (int i = 0; i < a.length; i++) {
444             g2.drawString(""+a[i], 100, 20 + 60 + ( i * 20 ) );
445         }
446         for (int i = 0; i < b.length; i++) {
447             g2.drawString(""+b[i], 100, 20 + 350 + ( i * 20 ) );
448         }
449     }
450
451     public void plot(Graphics2D g2) {
452
453         g2.setFont( new Font("Arial", 12, 12) );
454         DecimalFormat f = new DecimalFormat("0.#");
455         g2.setColor(Color.BLACK);
456         strike = (float)
457         (GRAVLIS_CalculateValues.input_strike_km-GRAVLIS_CalculateValues.input_y_km);
458         strike1 = (float) (65 * (strike / (2 * GRAVLIS_CalculateValues.input_strike_km)));
459         strike2= (float) (40 * (strike /(2 * GRAVLIS_CalculateValues.input_strike_km)));
460         i_no_obs = GRAVLIS_CalculateValues.input_n_obs;
461
462         obs = new double[i_no_obs+1];
463         for (int i = 1; i <= i_no_obs; i++) {
464             obs[i] = GRAVLIS_CalculateValues.input_x_km[i];
465         }
466
467         maxX = (float)obs[i_no_obs];
468         float maxy = (float)
GRAVLIS_Utility.findMaximumNumber(GRAVLIS_CalculateValues.input_nob_gob);
469         float maxy1 = (float)
GRAVLIS_Utility.findMaximumNumber(GRAVLIS_CalculateValues.o_GC);
470         if(maxy > maxy1)
471             maxY = maxy;
472         else
473             maxY = maxy1;
474         maxZ =
(float)GRAVLIS_CalculateValues.input_zval[GRAVLIS_CalculateValues.input_num_for];
475
476         g2.drawString("|", (float) 600 + 65 - strike1, 260 + strike2);
477         g2.drawString(""+f.format(GRAVLIS_CalculateValues.input_x_km[i_no_obs]), (float) 600
+ 65 - strike1, 260 + strike2 - 8);
478         g2.drawString("0", 125, 310);
479         g2.drawString("DISTANCE(km)", 400, 250);
480         float xplot = 0;
481         float xInterval = (float) (GRAVLIS_CalculateValues.input_x_km[i_no_obs] / 5);
482
483         int zInterval = 50;
484         for (float x = xInterval, j = 1; x < 600; x += xInterval){
485             xplot = xplot + xInterval;
486             if(j > 4)
487                 break;
488             g2.drawString("|", (float) (215 + (450 * x / maxX) - strike1), 260 + strike2);
489             g2.drawString("" + f.format(xplot), (float) (215 + (450 * x / maxX) - strike1)
3, 260 + strike2 - 8);
490             j++;
491         }
492
493         DecimalFormat d = new DecimalFormat("0.#");
494         float points1 = maxZ / 5 ;
495         for (int x = zInterval + 250, j = 1; x < 550; x += zInterval){
496             g2.drawString("-", 148, 52 + x);
497             g2.drawString("" + d.format(points1 * j), 125, 50 + x);
498             j++;
499         }
500

```

```

501     }
502
503
504     public void plotXYCoordinates(Graphics2D g2){
505
506         double minAno =
GRAVLIS_Utility.findMinimumNumber1(GRAVLIS_CalculateValues.input_nob_gob);
507         double maxAno =
GRAVLIS_Utility.findMaximumNumber(GRAVLIS_CalculateValues.input_nob_gob, minAno);
508         double minObAno = GRAVLIS_Utility.findMinimumNumber1(GRAVLIS_CalculateValues.o_GC);
509         double maxObAno = GRAVLIS_Utility.findMaximumNumber(GRAVLIS_CalculateValues.o_GC,
minObAno);
510
511         if (minAno < 0 && maxObAno < 0 && maxAno < 0 && minObAno < 0 ){
512             plotXYCoordinates1(g2);
513         }
514         if (minAno >= 0 && maxObAno > 0 ){
515             plotXYCoordinates1(g2);
516         }
517         if (minAno < 0 && maxObAno > 0 || maxAno>0 && minObAno<0){
518             plotXYCoordinates2(g2);
519         }
520     }
521
522     public void plotXYCoordinates1 (Graphics2D g2) {
523
524         g2.setFont(new Font("Arial", 20, 12));
525         g2.setColor(Color.black);
526         float maxval = (float)
GRAVLIS_Utility.findMaximumNumber(GRAVLIS_CalculateValues.o_GC);
527         float maxvall = (float)
GRAVLIS_Utility.findMaximumNumber(GRAVLIS_CalculateValues.input_nob_gob);
528         if(Math.abs(maxval)>Math.abs(maxvall))
529             maxY = maxval;
530         else
531             maxY = maxvall;
532         int points = (int)maxY / 5;
533         int yInterval = 50;
534         g2.drawString("0", 215 - strikel - 40, 50);
535         for (int x = yInterval, j = 1; x < 250; x += yInterval){
536
537             g2.drawString("-", 215 - strikel - 2, 50 + x);
538             g2.drawString(" " + (points*j), 215 - strikel - 40, 50 + x);
539             j++;
540         }
541         float prevx = (float) (215 - strikel+( 450 * obs[1] / maxX));
542         float prevy = (float)( 50 + ( 250 * GRAVLIS_CalculateValues.o_GC[1] / maxY ) );
543         float xpoint = 0;
544         float ypoint = 0;
545         float gypoint = 0;
546
547         for (int k = 1; k <= i_no_obs; k++) {
548
549             xpoint = (float)( 450 * obs[k] / maxX);
550             ypoint = (float)( ( 250 * GRAVLIS_CalculateValues.o_GC[k] / maxY ) );
551             gypoint = (float)( ( 250 * GRAVLIS_CalculateValues.input_nob_gob[k] / maxY ) );
552
553             g2.setColor(Color.BLACK);
554             g2.draw(new Line2D.Float(prevx, prevy, 215-strikel+ xpoint, 50 + ypoint ));
555
556             g2.setColor(Color.BLUE);
557             g2.setFont(new Font("Arial", 20, 40));
558             g2.drawString(".", 215-strikel+xpoint - 6 , 50 + gypoint + 3);
559
560             g2.setFont(new Font("Arial", 20, 12));
561             g2.setColor(Color.black);
562             prevx = 215-strikel + xpoint;
563             prevy = 50 + ypoint ;
564
565         }
566     }
567
568
569
570     public void plotXYCoordinates2 (Graphics2D g2) {

```

```

571 g2.setFont(new Font("Arial", 20, 12));
572 g2.setColor(Color.black);
573 int countPosObs=0, countNegObs=0, countPosCal=0, countNegCal=0;
574 double store[] = new double[GRAVLIS_CalculateValues.input_n_obs+1];
575 double store1[] = new double[GRAVLIS_CalculateValues.input_n_obs+1];
576 double negStore[] = new double[GRAVLIS_CalculateValues.input_n_obs+1];
577 double negStore1[] = new double[GRAVLIS_CalculateValues.input_n_obs+1];
578 for(int i = 1; i <= GRAVLIS_CalculateValues.input_n_obs; i++){
579     if(GRAVLIS_CalculateValues.o_GC[i]>0){
580         store[i] = GRAVLIS_CalculateValues.o_GC[i];
581         countPosCal = countPosCal+1;
582     }
583     else{
584         negStore[i] = GRAVLIS_CalculateValues.o_GC[i];
585         countNegCal = countNegCal+1;
586     }
587     if(GRAVLIS_CalculateValues.input_nob_gob[i]>0){
588         store1[i] = GRAVLIS_CalculateValues.input_nob_gob[i];
589         countPosObs = countPosObs+1;
590     }
591     else{
592         negStore1[i] = GRAVLIS_CalculateValues.input_nob_gob[i];
593         countNegObs = countNegObs+1;
594     }
595 }
596
597 float maxPos = (float) GRAVLIS_Utility.findMaximumNumber1(store);
598 float maxPos1 = (float) GRAVLIS_Utility.findMaximumNumber1(store1);
599 float maxNeg = (float) GRAVLIS_Utility.findMaximumNumber(negStore);
600 float maxNeg1 = (float) GRAVLIS_Utility.findMaximumNumber(negStore1);
601 float posNum =0;
602
603 if(maxPos>maxPos1)
604     posNum = maxPos;
605 else
606     posNum = maxPos1;
607 if(Math.abs(maxNeg)>Math.abs(maxNeg1))
608     maxY = maxNeg;
609 else
610     maxY = maxNeg1;
611
612 float prevx = (float) (215 - strikel+( 450 * obs[1] / maxX));;
613 float prevy = 0;
614 if(GRAVLIS_CalculateValues.o_GC[1]>0){
615     if(countNegCal > countPosObs)
616         prevy = 100 - (float)(( 50 * GRAVLIS_CalculateValues.o_GC[1] / posNum ) );
617     else
618         prevy = 200 - (float)(( 150 * GRAVLIS_CalculateValues.o_GC[1] / posNum ) );
619 }
620 else{
621     if(countNegCal > countPosObs)
622         prevy = 100 + (float)(( 200 * GRAVLIS_CalculateValues.o_GC[1] / maxY ) );
623     else{
624         prevy = 200+(float)(( 100 * GRAVLIS_CalculateValues.o_GC[1] / posNum ) );
625     }
626 }
627 float xpoint = 0;
628 float ypoint = 0;
629 float gypoint = 0;
630 float points = 0;
631
632 DecimalFormat f = new DecimalFormat("0.#");
633 if(countNegCal > countPosObs){
634     g2.drawString("-", 215-strikel-4, 100);
635     g2.drawString("0",215 - strikel - 40,100);
636 }
637 else{
638     g2.drawString("-", 215-strikel-4, 200);
639     g2.drawString("0",215 - strikel - 40,200);
640 }
641 g2.drawString("-", 215-strikel-4, 55);
642 g2.drawString(""+f.format(posNum), 215-strikel-40, 55);
643
644 if(countNegCal > countPosObs){
645     points = maxY / 4;

```

```

646         int yInterval=50;
647         for (int x = yInterval, j = 1; x < 250; x+=yInterval){
648
649             g2.drawString("-", 215 - strikel - 4, 100 + x );
650             g2.drawString(" " + f.format(points * j), 215 - strikel - 40, 100 + x );
651             j++;
652         }
653     }
654     else{
655         points = posNum / 3;
656         int yInterval=50;
657         for (int x = yInterval, j = 1; x < 200; x+=yInterval){
658
659             g2.drawString("-", 215 - strikel - 4, 205 - x );
660             g2.drawString(" " + f.format(points * j), 215 - strikel - 40, 205 - x );
661             j++;
662         }
663
664         g2.drawString("-", 215 - strikel - 4, 250 );
665         g2.drawString("-" + f.format(posNum/2), 215 - strikel - 40, 250 );
666         g2.drawString("-", 215 - strikel - 4, 300 );
667         g2.drawString("-" + f.format(posNum), 215 - strikel - 40, 300 );
668
669     }
670     for (int k = 1; k <= i_no_obs; k++) {
671
672         xpoint = (float)( 450 * obs[k] / maxX);
673         if(GRAVLIS_CalculateValues.o_GC[k]>0){
674             if(countNegCal > countPosObs)
675                 ypoint = 100-(float)( ( 50 * GRAVLIS_CalculateValues.o_GC[k] / posNum
676 );
677                 else
678                 ypoint = 200-(float)( ( 150 * GRAVLIS_CalculateValues.o_GC[k] / posNum
679 );
680             }
681             else{
682                 if(countNegCal > countPosObs)
683                 ypoint = 100+(float)( ( 200 * GRAVLIS_CalculateValues.o_GC[k] / maxY )
684 );
685                 else
686                 ypoint = 200+(float)( ( 100 * GRAVLIS_CalculateValues.o_GC[k] / posNum
687 );
688             }
689             if(GRAVLIS_CalculateValues.input_nob_gob[k]>0){
690                 if(countNegCal > countPosObs)
691                 gypoint = 100-(float)( ( 50 * GRAVLIS_CalculateValues.input_nob_gob[k]
692 posNum ) );
693                 else
694                 gypoint = 200-(float)( ( 150 * GRAVLIS_CalculateValues.input_nob_gob[k]
695 / posNum ) );
696             }
697             else{
698                 if(countNegCal > countPosObs)
699                 gypoint = 100+(float)( ( 200 * GRAVLIS_CalculateValues.input_nob_gob[k]
700 / maxY ) );
701                 else
702                 gypoint = 200+(float)( ( 100 * GRAVLIS_CalculateValues.input_nob_gob[k]
703 / posNum ) );
704             }
705             g2.setColor(Color.BLACK);
706             g2.draw(new Line2D.Float(prevx, prevy, 215-strikel + xpoint, ypoint ));
707
708             g2.setColor(Color.BLUE);
709             g2.setFont(new Font("Arial", 20, 40));
710             g2.drawString(".", 215-strikel+xpoint - 6 , gypoint+3 );
711
712             g2.setFont(new Font("Arial", 20, 12));
713             g2.setColor(Color.black);
714             prevx = 215-strikel + xpoint;
715             prevy = ypoint ;
716         }
717     }

```

```

713     public void drawDepth(Graphics2D g2) {
714
715         DecimalFormat df = new DecimalFormat("0.#");
716         float zpoint1 = 0;
717         float zpoint = 0;
718         if(GRAVLIS_CalculateValues.denal[1] <= 2.38)
719             g2.setColor(Color.YELLOW);
720         else
721             g2.setColor(Color.GREEN);
722         g2.fill(new Rectangle2D.Float(151, 300, 450, 250));
723         g2.fillRect(150, 300, 450, 250);
724         for (float j = 300; j <= 550; j++){
725             g2.draw(new Line2D.Float(600, j, 600+65, j-40));
726         }
727         Color col[] =
728         {Color.BLACK,Color.ORANGE,Color.WHITE,Color.PINK,Color.YELLOW,Color.ORANGE,Color.PIN
729         K,Color.WHITE,
730         Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
731         Color.PINK,Color.WHITE,
732         Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
733         Color.PINK,Color.WHITE,
734         Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
735         Color.PINK,Color.WHITE,
736         Color.YELLOW,Color.ORANGE,Color.PINK,Color.WHITE,Color.YELLOW,Color.ORANGE,
737         Color.PINK,Color.WHITE};
738
739         Color coll[] =
740         {Color.BLACK,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREEN,Color.D
741         ARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
742         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREE
743         Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
744         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREE
745         Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
746         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREE
747         Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,
748         Color.GREEN,Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA,Color.GREE
749         Color.DARK_GRAY,Color.BLUE,Color.CYAN,Color.MAGENTA};
750
751         for (int i =1; i < GRAVLIS_CalculateValues.input_num_for; i++) {
752
753             if (GRAVLIS_CalculateValues.denal[i+1] <= 2.38)
754                 g2.setColor(col[i]);
755             else
756                 g2.setColor(coll[i]);
757             zpoint = (float) (250 * GRAVLIS_CalculateValues.input_zval[i+1] / maxZ);
758             zpoint1 = (float) (250 * GRAVLIS_CalculateValues.input_zval[i] / maxZ);
759             if(i==1||i==GRAVLIS_CalculateValues.input_num_for-1){
760                 if(GRAVLIS_CalculateValues.denal[i+1] <= 2.38)
761                     g2.setColor(col[i]);
762                 else{
763                     g2.setColor(coll[i]);
764                 }
765             }
766             g2.fill(new Rectangle2D.Float(151, 300 + zpoint1, 450, zpoint - zpoint1));
767             for (float j = 300 + zpoint1; j <= 300 + zpoint; j++){
768                 g2.draw(new Line2D.Float(600, j, 600 + 65, j - 40));
769             }
770             g2.setColor(Color.BLACK);
771             g2.setFont(new Font("Arial", 40, 10));
772             g2.drawString("----->" +
773             df.format(GRAVLIS_CalculateValues.input_zval[i])+"(km)", 600 +65, 300 + zpoint1
774             - 38);
775         }
776         zpoint1 = (float) (250 *
777         GRAVLIS_CalculateValues.input_zval[GRAVLIS_CalculateValues.input_num_for] / maxZ);
778         g2.drawString("----->" +
779         df.format(GRAVLIS_CalculateValues.input_zval[GRAVLIS_CalculateValues.input_num_for])
780         +"(km)", 600 +65, 300 + zpoint1 - 38);
781         g2.setColor(Color.LIGHT_GRAY);
782         for (float j = 151; j <= 600; j++){
783             g2.draw(new Line2D.Float(j, 300, j + 65, 300 - 40));
784         }
785         g2.setColor(Color.BLACK);
786         g2.draw(new Line2D.Float(215 - strike1, 260 + strike2, (float) 600 + 65 - strike1,
787         260 + strike2));

```



```

770 g2.draw(new Line2D.Float(600, 300, 600, 300+zpoint));
771 g2.draw(new Line2D.Float(600 + 65, 300 - 40, 600 + 65, 300 + zpoint - 40));
772 g2.draw(new Line2D.Float(600, 300 + zpoint, 600 + 65, 300 + zpoint - 40));
773 g2.draw(new Line2D.Float(150, 550, 600, 550));
774 g2.setColor(Color.BLACK);
775
776 for (int i = 188; i <= 600 + 32; i++){
777     g2.drawString("-", i, 280 + 4);
778     i = i + 4;
779 }
780 g2.drawLine(90, 30, 950, 30);
781 g2.drawLine(90, 565, 950, 565);
782 g2.drawLine(950, 30, 950, 565);
783 g2.drawLine(90, 30, 90, 565);
784 g2.setFont(new Font("Arial", 40,12));
785 }
786
787 public void plotZCoordinates (Graphics2D g2) {
788     i_no_obs = GRAVLIS_CalculateValues.input_n_obs;
789     obs = new double[i_no_obs+1];
790     for (int i = 1; i <= i_no_obs; i++) {
791         obs[i] = GRAVLIS_CalculateValues.input_x_km[i];
792     }
793     maxX = (float) obs[i_no_obs];
794     maxZ =
795     (float)GRAVLIS_CalculateValues.input_zval[GRAVLIS_CalculateValues.input_num_for];
796     float s = 0;
797     float fc = 0;
798     float spoint = 300;
799     float fcpoint = (float)( 150 + ( 450 * GRAVLIS_CalculateValues.cft1[1] / maxX ) );
800     float xpoint = 0;
801     float zpoint = 0;
802
803     while (s <=
804     GRAVLIS_CalculateValues.input_zval[GRAVLIS_CalculateValues.input_num_for]) {
805
806         float z1 = (float) 0.001;
807         s = s + z1;
808         fc = 0;
809         for (int i = 1; i<4; i++){
810             fc = (float) (fc + GRAVLIS_CalculateValues.cft1[i] * Math.pow(s, i - 1));
811         }
812         xpoint = (float)( 450 * fc / maxX);
813         zpoint = (float)( 250 * s / maxZ);
814         g2.setColor(Color.BLACK);
815         g2.draw(new Line2D.Float(fcpoint, spoint,(float)150 + xpoint, 300 + zpoint));
816         g2.setColor(Color.RED);
817         g2.draw(new Line2D.Float(150, 300 + zpoint, (float) 150 + xpoint, 300 +
818         zpoint));
819         fcpoint = (float) 150 + xpoint;
820         spoint = 300 + zpoint;
821         if (fcpoint > (float) 150 + ((700 * obs[i_no_obs] / maxX))){
822             for (float j = spoint; j <= 550; j++){
823                 g2.setColor(Color.RED);
824                 g2.draw(new Line2D.Float(150, j, (float)( 150 + ((450 * obs[i_no_obs] ,
825                 maxX))), j));
826                 g2.draw(new Line2D.Float((float)( 151 + ((450 * obs[i_no_obs] / maxX))),
827                 j, (float)( 151 + ((450 * obs[i_no_obs] / maxX)) + 65, j - 40));
828             }
829             break;
830         }
831     }
832 }
833
834 public void drawDen(Graphics2D g2){
835     g2.setFont(new Font("Arial", 20, 12));
836     double minarr[] = new double[GRAVLIS_CalculateValues.denval.length];
837     for (int i = 1; i <= GRAVLIS_CalculateValues.input_num_for; i++){
838         minarr[i] = GRAVLIS_CalculateValues.denval[i];
839     }
840     Arrays.sort(minarr);
841     int min_den = (int)minarr[1];
842     denIndex(g2,min_den);
843     double max_den = minarr[GRAVLIS_CalculateValues.input_num_for];

```



```

840     double denMax1 = max_den - min_den;
841     float denXpoint = 0;
842     float denXpoint1 = 0;
843     double denpoint1 = 0;
844     float z1 = 0;
845     float z2 = 0;
846     float ini = 0;
847     double denpoint = GRAVLIS_CalculateValues.denval[1] - min_den ;
848     denpoint1 = GRAVLIS_CalculateValues.denval[2] - min_den;
849     denXpoint = (float) (90 * denpoint / denMax1);
850     ini = (float) (250 * GRAVLIS_CalculateValues.input_zval[1] / maxZ);
851     denXpoint1 = (float) (90 * denpoint1 / denMax1);
852     g2.draw(new Line2D.Float(820 + denXpoint, 300 , 820 + denXpoint, 300+ini ));
853     g2.draw(new Line2D.Float(820 + denXpoint, 300 + ini, 820 + denXpoint1, 300 + ini)).
854     for (int i = 1; i < GRAVLIS_CalculateValues.input_num_for; i++){
855         denpoint = GRAVLIS_CalculateValues.denval[i+1] - min_den ;
856         if (i < GRAVLIS_CalculateValues.input_num_for -1){
857             denpoint1 = GRAVLIS_CalculateValues.denval[i + 2] - min_den;
858             z1 = (float) (250 * GRAVLIS_CalculateValues.input_zval[i + 1] / maxZ);
859         }
860         denXpoint = (float) (90 * denpoint / denMax1);
861         denXpoint1 = (float) (90 * denpoint1 / denMax1);
862         z2 = (float) (250 * GRAVLIS_CalculateValues.input_zval[i] / maxZ);
863         g2.setColor(Color.black);
864         g2.draw(new Line2D.Float(820 + denXpoint, 300 + z2, 820 + denXpoint, 300 + z1));
865         g2.draw(new Line2D.Float(820 + denXpoint, 300 + z1, 820 + denXpoint1, 300 +
z1));
866     }
867     g2.draw(new Line2D.Float(820 + denXpoint1, 300 + z1, 820 + denXpoint1, 550));
868     g2.drawLine(820, 300, 910, 300);
869     g2.drawLine(820, 300, 820, 550);
870     g2.setColor(Color.white);
871     g2.drawLine(821, 550, 910, 550);
872 }
873
874 public void denIndex(Graphics2D g,int minden){
875
876     DecimalFormat d = new DecimalFormat("0.#");
877     DecimalFormat d1 = new DecimalFormat("0.##");
878     maxZ =
(float)GRAVLIS_CalculateValues.input_zval[GRAVLIS_CalculateValues.input_num_for];
879     float points = maxZ / 5 ;
880     int zInterval = 50;
881     g.setColor(Color.black);
882     for (int x = zInterval + 250, j = 1; x < 550; x += zInterval){
883         g.drawString("-", 818, 52 + x);
884         g.drawString(" " + d.format(points * j), 795, 50 + x);
885         j++;
886     }
887     g.setColor(Color.red);
888     String []b = {"D", "E", "P", "T", "H", "(k", "m)"};
889     for (int i = 0; i < b.length; i++) {
890         g.drawString(" "+b[i], 765, 350 + ( i * 20 ) );
891     }
892     g.setColor(Color.black);
893     g.drawString(" "+minden, 818, 298);
894     double max = GRAVLIS_Utility.findMaximumNumber1(GRAVLIS_CalculateValues.denval);
895     g.drawString(" "+d1.format(max), 908, 298);
896     g.setColor(Color.red);
897     g.drawString("Density", 850, 280);
898     g.drawString("(gm/cm )",850 , 295);
899     g.setFont(new Font("Arial", 20, 9));
900     g.drawString("3",890 , 292);
901     g.setFont(new Font("Arial", 20, 12));
902     g.setColor(Color.black);
903     g.drawString("0", 795, 305);
904
905 }
906 public void idx(Graphics2D g){
907     g.setColor(Color.BLUE);
908     g.setFont(new Font("Arial", 20, 50));
909     g.drawString(" ... ", 750, 67);
910     g.setFont(new Font("Arial", 20, 12));
911     g.drawString(": Observed anomalies", 820, 70);

```

```

912         g.setColor(Color.BLACK);
913         g.drawString("_____", 765, 85);
914         g.drawString(": Modeled anomalies", 820, 90);
915     }
916 }
917 }
918 -----
919 package com.gravlis.model;
920
921 import java.awt.Color;
922 import java.awt.Graphics;
923 import java.awt.Graphics2D;
924 import java.awt.image.BufferedImage;
925 import java.io.File;
926 import java.io.FileOutputStream;
927 import java.text.DecimalFormat;
928 import java.util.HashMap;
929
930 import javax.imageio.ImageIO;
931 import com.gravlis.model.GRAVLIS_CalNOREQ;
932 import com.gravlis.model.GRAVLIS_CalculateValues;
933 import com.gravlis.util.GRAVLIS_Utility;
934 import com.gravlis.view.GRAVLIS_MainPanel;
935 import com.gravlis.view.GRAVLIS_TableView;
936
937 public class GRAVLIS_CalculateValues {
938
939     public static Object obj[][] = null;
940     public static double input_x_km[]; public static double input_nob_gob[];
941     public static double []o_GC ; public static double []o_err ; public static double
942     []o_par; public static double []o_funcnt ;
943     public static double []cft; public static double []cft1; public static double
944     []denval; public static double []input_den_val;
945     public static double []input_ele_km; public static double []input_min_val; public static
946     double []input_max_val;
947     public static double []par; public static double []var; public static double
948     []input_zval;
949     public static double input_y_km, input_sd_poly, input_al_err, o_func, input_strike_km;
950     public static int o_iter, input_num_for, np, input_nob_iter, input_n_obs;
951     public static int input_nregcft = 2;
952     public static String input_area_name, input_profile_num = "";
953     public static String parValue = "";
954     public static BufferedImage image;
955
956     public void getAnamolyValues(HashMap h_Map) {
957
958         try {
959
960             input_n_obs = GRAVLIS_Utility.convertInteger((String)h_Map.get("N_OBS"));
961             input_x_km = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("X_KM"));
962             input_ele_km = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
963             input_nob_gob =
964             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
965             input_sd_poly = GRAVLIS_Utility.convertDouble((String)h_Map.get("SD_POLY"));
966             input_strike_km = GRAVLIS_Utility.convertDouble((String)h_Map.get("STRIKE_KM"));
967             input_y_km = GRAVLIS_Utility.convertDouble((String)h_Map.get("Y_KM"));
968             input_zval = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("Z_VAL"));
969             input_num_for = GRAVLIS_Utility.convertInteger((String)h_Map.get("NUM_FOR"));
970             input_area_name = GRAVLIS_Utility.convertString((String)h_Map.get("AREA_FE"));
971             input_profile_num =
972             GRAVLIS_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
973             input_min_val =
974             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("MIN_VAL"));
975             input_max_val =
976             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("MAX_VAL"));
977             input_den_val =
978             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("DEN_VAL"));
979             input_nob_iter = GRAVLIS_Utility.convertInteger((String)h_Map.get("NUM_ITE"));
980             input_al_err = GRAVLIS_Utility.convertDouble((String)h_Map.get("AL_ERR"));
981         }
982         catch(Exception e) {
983             e.printStackTrace();
984         }
985     }
986 }

```

```

977     }
978     par = new double[input_nregcft + input_num_for + 2];
979     var = new double[input_num_for + 1];
980 }
981
982 public void getFalDen(){
983
984     o_GC = new double[input_n_obs + 1];
985     o_err = new double[input_n_obs + 1];
986     o_par = new double[input_n_obs + 1];
987     o_funct = new double[input_nob_iter + 1];
988
989     double []g1 = new double[input_n_obs + 1];
990     double []g2 = new double[input_n_obs + 1];
991     double gc[] = new double[input_n_obs + 1];
992     double err[] = new double[input_n_obs + 1];
993
994     double [][]p = new double[input_nregcft + input_num_for + 2][input_nregcft +
input_num_for + 3];
995     double [][]s = new double[input_nregcft + input_num_for + 2][input_n_obs + 1];
996     double []b = new double[input_nregcft + input_num_for + 2];
997     double []par1 = new double[input_nregcft + input_num_for + 2];
998     double []par2 = new double[input_nregcft + input_num_for + 2];
999     double []dpar = new double[input_nregcft + input_num_for + 2];
1000
1001     cft = new double[input_n_obs + 1];
1002     cft1 = new double[input_n_obs + 1];
1003     double []KS = new double[2];
1004     double funct2 = 0;
1005     double lambda = 0.5;
1006     np = input_nregcft + input_num_for + 1;
1007     denval = new double[input_num_for + 1];
1008     double funct1 = 0;
1009
1010     double datum = input_nob_gob[1];
1011     double r = input_nob_gob[input_n_obs] - input_nob_gob[1];
1012     int kk = 1;
1013     double gh = 0.5 * r;
1014     double XH = 0;
1015     while ((( input_nob_gob[kk] - datum) / gh ) - 1.0 < 0) {
1016         kk = kk + 1;
1017     }
1018     if ((( input_nob_gob[kk] - datum) / gh) - 1.0 > 0) {
1019         XH = input_x_km[kk-1] + ( ( gh + datum - input_nob_gob[kk-1]) * ( input_x_km[k]
- input_x_km[kk-1] ) ) / ( input_nob_gob[kk] - input_nob_gob[kk-1] );
1020     }
1021     if ((( input_nob_gob[kk] - datum ) / gh) - 1.0 == 0) {
1022         XH = input_x_km[kk];
1023     }
1024     cft[1] = XH;
1025     for(int k=1; k <= input_nregcft + 1; k++){
1026         par[k] = cft[k];
1027     }
1028     double dpar = 0.1;
1029     if(var == input_zval)
1030         GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km, par, np,
input_zval, gc);
1031     else
1032         GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km, par, np,
input_den_val, gc);
1033     for (int k = 1; k <= input_n_obs; k++) {
1034         err[k] = input_nob_gob[k] - gc[k];
1035         funct1 = funct1 + Math.pow( err[k] , 2 );
1036     }
1037
1038     lambda = 0.5;
1039     int NP1 = np + 1;
1040     int IER = 1;
1041
1042     while (IER <= input_nob_iter) {
1043
1044         int ITER1 = IER ;
1045

```

```

1046         o_funct[ITER1] = funct1;
1047
1048         for (int K = 1; K <= np; K++) {
1049             par1[K] = par[K];
1050         }
1051
1052         for (int I = 1; I <= np; I++) {
1053             par1[I] = par[I] + dpar / 2.0;
1054             if(var == input_zval)
1055                 GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_zval, g1);
1056             else
1057                 GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_den_val, g1);
1058             par1[I] = par[I] - dpar / 2.0;
1059             if(var == input_zval)
1060                 GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_zval, g2);
1061             else
1062                 GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_den_val, g2);
1063
1064             for (int K = 1; K <= input_n_obs; K++) {
1065                 s[I][K] = ( g1[K] - g2[K] ) / dpar;
1066             }
1067         }
1068         for (int J = 1; J <= NP1; J++) {
1069             for(int I = 1; I <= np; I++) {
1070                 p[I][J] = 0.0;
1071             }
1072         }
1073         for (int J = 1; J <= np; J++) {
1074             for (int I = 1; I <= np; I++) {
1075                 for (int K = 1; K <= input_n_obs; K++){
1076                     p[I][J] = p[I][J] + s[I][K] * s[J][K];
1077                 }
1078             }
1079         }
1080
1081         for (int J = 1; J <= np; J++) {
1082             for (int K = 1; K <= input_n_obs; K++) {
1083                 p[J][NP1] = p[J][NP1] + err[K] * s[J][K];
1084             }
1085         }
1086
1087         do {
1088             double con = lambda + 1.0;
1089             for (int I = 1; I <= np; I++) {
1090                 dupar[I] = par[I];
1091             }
1092             for (int L = 1; L <= np; L++) {
1093                 for (int J = 1; J <= np; J++) {
1094                     if (L - J == 0)
1095                         p[L][J] = p[L][J] * con;
1096                 }
1097             }
1098             GRAVLIS_CalNOREQ.getNOREQ(p, b, np, KS);
1099             for (int I = 1; I <= np; I++) {
1100                 par2[I] = dupar[I] + 0.25 * b[I];
1101             }
1102             if(var==input_zval){
1103                 for(int j=1; j <= input_num_for; j++){
1104                     if (par2[input_nregcft + 1 + j] < input_min_val[j]) {
1105                         par2[input_nregcft + 1 + j] = input_min_val[j];
1106                     }
1107                     if (par2[input_nregcft + 1 + j]>input_max_val[j]) {
1108                         par2[input_nregcft + 1 + j] = input_max_val[j];
1109                     }
1110                 }
1111             }
1112             if(var == input_zval)

```

```

1113         GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par2, np, input_zval, gc);
1114     else
1115         GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par2, np, input_den_val, gc);
1116         funct2 = 0.0;
1117         for (int K = 1; K <= input_n_obs; K++) {
1118             err[K] = input_nob_gob[K] - gc[K];
1119             funct2 = funct2 + Math.pow(err[K] , 2);
1120         }
1121         if (funct1 - funct2 < 0) {
1122             lambda = lambda * 2.0;
1123             if (lambda - 12 < 0) {
1124                 for (int I = 1; I <= np; I++) {
1125                     for (int J = 1; J <= np; J++) {
1126                         if(I - J == 0)
1127                             p[I][J] = p[I][J] / con;
1128                     }
1129                 }
1130             }
1131             else
1132                 break;
1133         }
1134     }
1135
1136     } while (funct1 - funct2 < 0);
1137
1138     funct1 = funct2;
1139     DecimalFormat d = new DecimalFormat("0.##");
1140     IER++;
1141     for (int I = 1; I <= np; I++) {
1142         par[I] = par2[I];
1143     }
1144     o_iter = ITER1;
1145     o_func = funct2;
1146     parValue = d.format(par[1]);
1147     for (int K = 1; K <= input_n_obs; K++) {
1148         o_GC[K] = gc[K];
1149         o_err[K] = err[K];
1150     }
1151     for (int l = 1; l <= np; l++) {
1152         o_par[l] = par[l];
1153     }
1154     for(int j = 1; j <= input_nregcft + 1; j++){
1155         cft1[j] = par[j];
1156     }
1157     if(var == input_zval){
1158         for(int j = 1; j <= input_num_for; j++){
1159             denval[j] = par[input_nregcft + 1 + j];
1160         }
1161     }
1162     else{
1163         for(int j = 1; j <= input_num_for; j++){
1164             input_zval[j] = par[input_nregcft + 1 + j];
1165             denval[j] = input_den_val[j];
1166         }
1167     }
1168
1169     }
1170
1171     if ( funct2 < input_al_err || ITER1 == input_nob_iter || lambda -12 > 0 ) {
1172         o_iter = ITER1;
1173         o_func = funct2;
1174         parValue = d.format(par[1]);
1175         for (int K = 1; K <= input_n_obs; K++) {
1176             o_GC[K] = gc[K];
1177             o_err[K] = err[K];
1178         }
1179         for (int l = 1; l <= np; l++) {
1180             o_par[l] = par[l];
1181

```

```

1184         }
1185
1186         setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
o_err, o_par, o_func, input_area_name);
1187         drawGraph();
1188     }
1189     else{
1190         setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
o_err, o_par, o_func, input_area_name);
1191         drawGraph();
1192     }
1193
1194     try {
1195         Thread.sleep(10);
1196     } catch (InterruptedException e) {
1197         // TODO Auto-generated catch block
1198         e.printStackTrace();
1199     }
1200
1201     if ( funct2 < input_al_err || lambda -12 > 0)
1202         break;
1203
1204     lambda = lambda / 2.0;
1205
1206 }
1207
1208 }
1209
1210 public static void setGraphValues(int i_no_obs, int np, int ite, double []dis, double
[]GOBS, double []GCAL, double []ERROR, double []PARA, double FUNCT, String Area_fe) {
1211
1212     obj = new Object[i_no_obs+1][4];
1213
1214     DecimalFormat df = new DecimalFormat("0.###");
1215     DecimalFormat d = new DecimalFormat("0.##");
1216     DecimalFormat dl = new DecimalFormat("0.#####");
1217     for(int K=1; K <= i_no_obs; K++){
1218         obj[K][0]= " " + dis[K];
1219         obj[K][1]= " " + df.format(GOBS[K]);
1220         obj[K][2]= " " + df.format(GCAL[K]);
1221         obj[K][3]= " " + df.format(ERROR[K]);
1222     }
1223
1224     obj[0][0] ="ITERATION";
1225     obj[0][1] = "=" + " "+ite;
1226
1227     GRAVLIS_TableView.val.setText("");
1228     GRAVLIS_TableView.val.appendText("ITERATION NUMBER:-"+ite);
1229     GRAVLIS_TableView.val.appendText("\n");
1230     GRAVLIS_TableView.val.appendText("OBJECTIVE FUNCTION  "+dl.format(FUNCT)+"\n");
1231     GRAVLIS_TableView.val.appendText("\n");
1232     GRAVLIS_TableView.val.append("INTERPRETED PARAMETERS:-\n");
1233     GRAVLIS_TableView.val.appendText("-----\n");
1234     GRAVLIS_TableView.val.appendText("\n");
1235     GRAVLIS_TableView.val.appendText("COEFFICIENTS OF THE POLYNOMIAL ");
1236     GRAVLIS_TableView.val.appendText("\n");
1237     GRAVLIS_TableView.val.appendText("-----\n");
1238
1239     GRAVLIS_TableView.val.appendText(dl.format(PARA[1])+"\n");
1240     GRAVLIS_TableView.val.appendText(dl.format(PARA[2])+"\n");
1241     GRAVLIS_TableView.val.appendText(dl.format(PARA[3])+"\n");
1242     GRAVLIS_TableView.val.appendText("\n");
1243
1244     if(var == input_zval)
1245         GRAVLIS_TableView.val.appendText("ESTIMATED DENSITIES OF THE FORMATIONS
:-"+ "\n");
1246     else
1247         GRAVLIS_TableView.val.appendText("ESTIMATED DEPTHS TO THE INTERFACES :-"+ "\n");
1248     GRAVLIS_TableView.val.appendText("-----\n");
1249     //GRAVLIS_TableView.val.appendText("\n");
1250     for(int k = 4; k <= input_nregcft + input_num_for + 1; k++){
1251         GRAVLIS_TableView.val.appendText(d.format(PARA[k])+"\n");
1252     }

```

```

1253     }
1254
1255     public static void drawGraph(){
1256         com.gravlis.view.GRAVLIS_DrawGraph dg = new com.gravlis.view.GRAVLIS_DrawGraph();
1257         try
1258         {
1259             int width = 1280;
1260             int height = 650;
1261             BufferedImage buffer = new
1262             BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
1263             Graphics g1= buffer.createGraphics();
1264             g1.setColor(Color.WHITE);
1265             g1.fillRect(0,0,width,height);
1266             Graphics2D g2 = (Graphics2D)g1 ;
1267             dg.plot(g2);
1268             dg.plotXYCoordinates(g2);
1269             dg.drawGraph(g2);
1270             dg.drawDepth(g2);
1271             dg.plotZCoordinates(g2);
1272             //dg.plotXYCoordinates(g2);
1273             dg.drawDen(g2);
1274             dg.plot(g2);
1275             dg.idx(g2);
1276
1277             FileOutputStream os = new FileOutputStream(
1278             GRAVLIS_CalculateValues.input_area_name + ".jpg");
1279             ImageIO.write(buffer, "jpg", os);
1280             os.close();
1281
1282             String path = GRAVLIS_CalculateValues.input_area_name + ".jpg";
1283             BufferedImage image = ImageIO.read(new File(path));
1284
1285             Graphics g_image = GRAVLIS_MainPanel.img.getGraphics();
1286             g_image.drawImage(image, -40, 0, image.getWidth(), image.getHeight(), dg);
1287
1288         }
1289         catch (Exception e2) {
1290             // e2.printStackTrace();
1291         }
1292     }
1293 }
1294 package com.gravlis.model;
1295
1296 import java.awt.Color;
1297 import java.awt.Graphics;
1298 import java.awt.Graphics2D;
1299 import java.awt.image.BufferedImage;
1300 import java.io.File;
1301 import java.io.FileOutputStream;
1302 import java.text.DecimalFormat;
1303 import java.util.HashMap;
1304
1305 import javax.imageio.ImageIO;
1306 import com.gravlis.model.GRAVLIS_CalNOREQ;
1307 import com.gravlis.model.GRAVLIS_CalculateValues;
1308 import com.gravlis.util.GRAVLIS_Utility;
1309 import com.gravlis.view.GRAVLIS_MainPanel;
1310 import com.gravlis.view.GRAVLIS_TableView;
1311
1312 public class GRAVLIS_CalculateValues {
1313
1314     public static Object obj[][] = null;
1315     public static double input_x_km[];public static double input_nob_gob[];
1316     public static double []o_GC ;public static double []o_err ;public static double
1317     []o_par;public static double []o_func ;
1318     public static double []cft;public static double []cft1;public static double
1319     []denval;public static double []input_den_val;
1320     public static double []input_ele_km;public static double []input_min_val;public static
1321     double []input_max_val;
1322     public static double []par; public static double []var; public static double
1323     []input_z_val;
1324     public static double input_y_km,input_sd_poly,input_al_err,o_func,input_strike_km;
1325     public static int o_iter,input_num_for,np,input_nob_iter,input_n_obs;

```



```

1322     public static int input_nregcft = 2;
1323     public static String input_area_name, input_profile_num = "";
1324     public static String parValue = "";
1325     public static BufferedImage image;
1326
1327     public void getAnamolyValues(HashMap h_Map) {
1328
1329
1330         try {
1331
1332             input_n_obs = GRAVLIS_Utility.convertInteger((String)h_Map.get("N_OBS"));
1333             input_x_km = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("X_KM"));
1334             input_ele_km = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
1335             input_nob_gob =
GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
1336             input_sd_poly = GRAVLIS_Utility.convertDouble((String)h_Map.get("SD_POLY"));
1337             input_strike_km = GRAVLIS_Utility.convertDouble((String)h_Map.get("STRIKE_KM"));
1338             input_y_km = GRAVLIS_Utility.convertDouble((String)h_Map.get("Y_KM"));
1339             input_zval = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("Z_VAL"));
1340             input_num_for = GRAVLIS_Utility.convertInteger((String)h_Map.get("NUM_FOR"));
1341             input_area_name = GRAVLIS_Utility.convertString((String)h_Map.get("AREA_FE"));
1342             input_profile_num =
GRAVLIS_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
1343             input_min_val =
GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("MIN_VAL"));
1344             input_max_val =
GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("MAX_VAL"));
1345             input_den_val =
GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("DEN_VAL"));
1346             input_nob_iter = GRAVLIS_Utility.convertInteger((String)h_Map.get("NUM_ITE"));
1347             input_al_err = GRAVLIS_Utility.convertDouble((String)h_Map.get("AL_ERR"));
1348         }
1349         catch(Exception e) {
1350             e.printStackTrace();
1351         }
1352         par = new double[input_nregcft + input_num_for + 2];
1353         var = new double[input_num_for + 1];
1354     }
1355
1356     public void getFalDen(){
1357
1358
1359         o_GC = new double[input_n_obs + 1];
1360         o_err = new double[input_n_obs + 1];
1361         o_par = new double[input_n_obs + 1];
1362         o_funct = new double[input_nob_iter + 1];
1363
1364         double []g1 = new double[input_n_obs + 1];
1365         double []g2 = new double[input_n_obs + 1];
1366         double gc[] = new double[input_n_obs + 1];
1367         double err[] = new double[input_n_obs + 1];
1368
1369         double [][]p = new double[input_nregcft + input_num_for + 2][input_nregcft +
input_num_for + 3];
1370         double [][]s = new double[input_nregcft + input_num_for + 2][input_n_obs + 1];
1371         double []b = new double[input_nregcft + input_num_for + 2];
1372         double []par1 = new double[input_nregcft + input_num_for + 2];
1373         double []par2 = new double[input_nregcft + input_num_for + 2];
1374         double []dubar = new double[input_nregcft + input_num_for + 2];
1375
1376         cft = new double[input_n_obs + 1];
1377         cft1 = new double[input_n_obs + 1];
1378         double []KS = new double[2];
1379         double funct2 = 0;
1380         double lambda = 0.5;
1381         np = input_nregcft + input_num_for + 1;
1382         denval = new double[input_num_for + 1];
1383         double funct1 = 0;
1384
1385         double datum = input_nob_gob[1];
1386         double r = input_nob_gob[input_n_obs] - input_nob_gob[1];
1387         int kk = 1;
1388         double gh = 0.5 * r;
1389         double XH = 0;
1390         while (((input_nob_gob[kk] - datum) / gh) - 1.0 < 0) {

```



```

1391         kk = kk + 1;
1392     }
1393     if ((( input_nob_gob[kk] - datum) / gh) - 1.0 > 0) {
1394         XH = input_x_km[kk-1] + ( ( gh + datum - input_nob_gob[kk-1]) * ( input_x_km[kk]
- input_x_km[kk-1] ) ) / ( input_nob_gob[kk] - input_nob_gob[kk-1] );
1395     }
1396     if ((( input_nob_gob[kk] - datum ) / gh) - 1.0 == 0) {
1397         XH = input_x_km[kk];
1398     }
1399     cft[1] = XH;
1400     for(int k=1; k <= input_nregcft + 1; k++){
1401         par[k] = cft[k];
1402     }
1403     double dpar = 0.1;
1404     if(var == input_zval)
1405         GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km, par, np,
input_zval, gc);
1406     else
1407         GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km, par, np,
input_den_val, gc);
1408     for (int k = 1; k <= input_n_obs; k++) {
1409         err[k] = input_nob_gob[k] - gc[k];
1410         funct1 = funct1 + Math.pow( err[k] , 2 );
1411     }
1412
1413     lambda = 0.5;
1414     int NP1 = np + 1;
1415     int IER = 1;
1416
1417     while (IER <= input_nob_iter) {
1418
1419         int ITER1 = IER ;
1420         o_funct[ITER1] = funct1;
1421
1422         for (int K = 1; K <= np; K++) {
1423             par1[K] = par[K];
1424         }
1425
1426         for (int I = 1; I <= np; I++) {
1427             par1[I] = par[I] + dpar / 2.0;
1428             if(var == input_zval)
1429                 GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_zval, g1);
1430             else
1431                 GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_den_val, g1);
1432             par1[I] = par[I] - dpar / 2.0;
1433             if(var == input_zval)
1434                 GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_zval, g2);
1435             else
1436                 GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_den_val, g2);
1437
1438             for (int K = 1; K <= input_n_obs; K++) {
1439                 s[I][K] = ( g1[K] - g2[K] ) / dpar;
1440             }
1441         }
1442         for (int J = 1; J <= NP1; J++) {
1443             for(int I = 1; I <= np; I++) {
1444                 p[I][J] = 0.0;
1445             }
1446         }
1447         for (int J = 1; J <= np; J++) {
1448             for (int I = 1; I <= np; I++) {
1449                 for (int K = 1; K <= input_n_obs; K++){
1450                     p[I][J] = p[I][J] + s[I][K] * s[J][K];
1451                 }
1452             }

```

```

1453     }
1454
1455     for (int J = 1; J <= np; J++) {
1456         for (int K = 1; K <= input_n_obs; K++) {
1457             p[J][NP1] = p[J][NP1] + err[K] * s[J][K];
1458         }
1459     }
1460
1461     do {
1462         double con = lambda + 1.0;
1463         for (int I = 1; I <= np; I++) {
1464             dupar[I] = par[I];
1465         }
1466         for (int L = 1; L <= np; L++) {
1467             for (int J = 1; J <= np; J++) {
1468                 if (L - J == 0)
1469                     p[L][J] = p[L][J] * con;
1470             }
1471         }
1472         GRAVLIS_CalNOREQ.getNOREQ(p, b, np, KS);
1473         for (int I = 1; I <= np; I++) {
1474             par2[I] = dupar[I] + 0.25 * b[I];
1475         }
1476         if(var==input_zval){
1477             for(int j =1; j <= input_num_for; j++){
1478                 if (par2[input_nregcft + 1 + j] < input_min_val[j]) {
1479                     par2[input_nregcft + 1 + j] = input_min_val[j];
1480                 }
1481                 if (par2[input_nregcft + 1 + j]>input_max_val[j]) {
1482                     par2[input_nregcft+ 1 + j] = input_max_val[j];
1483                 }
1484             }
1485         }
1486         if(var == input_zval)
1487             GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par2, np, input_zval, gc);
1488         else
1489             GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par2, np, input_den_val, gc);
1490         funct2 = 0.0;
1491         for (int K = 1; K <= input_n_obs; K++) {
1492             err[K] = input_nob_gob[K] - gc[K];
1493             funct2 = funct2 + Math.pow(err[K] , 2);
1494         }
1495         if (funct1 - funct2 < 0) {
1496
1497             lambda = lambda * 2.0;
1498             if (lambda - 12 < 0) {
1499
1500                 for (int I = 1; I <= np; I++) {
1501                     for (int J = 1; J <= np; J++) {
1502                         if(I - J == 0)
1503                             p[I][J] = p[I][J] / con;
1504                     }
1505                 }
1506             }
1507             else
1508                 break;
1509
1510         }
1511
1512     } while (funct1 - funct2 < 0);
1513
1514     funct1 = funct2;
1515     DecimalFormat d = new DecimalFormat("0.##");
1516     IER++;
1517     for (int I = 1; I <= np; I++) {
1518         par[I] = par2[I];
1519     }
1520     o_iter = ITER1;
1521     o_func = funct2;
1522     parValue = d.format(par[1]);
1523

```

```

1524         for (int K = 1; K <= input_n_obs; K++) {
1525             o_GC[K] = gc[K];
1526             o_err[K] = err[K];
1527         }
1528         for (int l = 1; l <= np; l++) {
1529             o_par[l] = par[l];
1530         }
1531         for(int j =1; j <= input_nregcft + 1; j++){
1532             cft1[j] = par[j];
1533         }
1534         if(var == input_zval){
1535             for(int j =1; j <= input_num_for; j++){
1536                 denval[j] = par[input_nregcft + 1 + j];
1537             }
1538         }
1539         else{
1540             for(int j = 1; j <= input_num_for; j++){
1541                 input_zval[j] = par[input_nregcft + 1 + j];
1542                 denval[j] = input_den_val[j];
1543             }
1544         }
1545     }
1546
1547     if ( funct2 < input_al_err || ITER1 == input_nob_iter || lambda -12 > 0 ) {
1548
1549         o_iter = ITER1;
1550         o_func = funct2;
1551         parValue = d.format(par[1]);
1552         for (int K = 1; K <= input_n_obs; K++) {
1553             o_GC[K] = gc[K];
1554             o_err[K] = err[K];
1555         }
1556         for (int l = 1; l <= np; l++) {
1557             o_par[l] = par[l];
1558         }
1559
1560         setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
o_err, o_par, o_func, input_area_name);
1561         drawGraph();
1562     }
1563     else{
1564         setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
o_err, o_par, o_func, input_area_name);
1565         drawGraph();
1566     }
1567
1568     try {
1569         Thread.sleep(10);
1570     } catch (InterruptedException e) {
1571         // TODO Auto-generated catch block
1572         e.printStackTrace();
1573     }
1574
1575     if ( funct2 < input_al_err || lambda -12 > 0)
1576         break;
1577
1578     lambda = lambda / 2.0;
1579
1580 }
1581 }
1582
1583
1584     public static void setGraphValues(int i_no_obs, int np, int ite, double []dis, double
[]GOBS, double []GCAL, double []ERROR, double []PARA, double FUNCT, String Area_fe) {
1585
1586         obj = new Object[i_no_obs+1][4];
1587
1588         DecimalFormat df = new DecimalFormat("0.###");
1589         DecimalFormat d = new DecimalFormat("0.##");
1590         DecimalFormat dl = new DecimalFormat("0.#####");
1591         for(int K=1; K <= i_no_obs; K++){
1592             obj[K][0]= "" + dis[K];
1593             obj[K][1]= "" + df.format(GOBS[K]);
1594             obj[K][2]= "" + df.format(GCAL[K]);
1595             obj[K][3]= "" + df.format(ERROR[K]);

```

```

1596     }
1597
1598     obj[0][0] = "ITERATION";
1599     obj[0][1] = "=" + " " + ite;
1600
1601     GRAVLIS_TableView.val.setText("");
1602     GRAVLIS_TableView.val.appendText("ITERATION NUMBER:-"+ite);
1603     GRAVLIS_TableView.val.appendText("\n");
1604     GRAVLIS_TableView.val.appendText("OBJECTIVE FUNCTION  =" + dl.format(FUNCT) + "\n");
1605     GRAVLIS_TableView.val.appendText("\n");
1606     GRAVLIS_TableView.val.appendText("INTERPRETED PARAMETERS:-\n");
1607     GRAVLIS_TableView.val.appendText("-----\n");
1608     GRAVLIS_TableView.val.appendText("\n");
1609     GRAVLIS_TableView.val.appendText("COEFFICIENTS OF THE POLYNOMIAL ");
1610     GRAVLIS_TableView.val.appendText("\n");
1611     GRAVLIS_TableView.val.appendText("-----\n");
1612
1613     GRAVLIS_TableView.val.appendText(dl.format(PARA[1]) + "\n");
1614     GRAVLIS_TableView.val.appendText(dl.format(PARA[2]) + "\n");
1615     GRAVLIS_TableView.val.appendText(dl.format(PARA[3]) + "\n");
1616     GRAVLIS_TableView.val.appendText("\n");
1617
1618     if(var == input_zval)
1619         GRAVLIS_TableView.val.appendText("ESTIMATED DENSITIES OF THE FORMATIONS
:-"+ "\n");
1620     else
1621         GRAVLIS_TableView.val.appendText("ESTIMATED DEPTHS TO THE INTERFACES :-"+ "\n");
1622         GRAVLIS_TableView.val.appendText("-----\n");
1623         //GRAVLIS_TableView.val.appendText("\n");
1624         for(int k = 4; k <= input_nregcft + input_num_for + 1; k++){
1625             GRAVLIS_TableView.val.appendText(d.format(PARA[k]) + "\n");
1626         }
1627     }
1628
1629     public static void drawGraph(){
1630         com.gravlis.view.GRAVLIS_DrawGraph dg = new com.gravlis.view.GRAVLIS_DrawGraph();
1631         try
1632         {
1633             int width = 1280;
1634             int height = 650;
1635             BufferedImage buffer = new
BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
1636             Graphics g1= buffer.createGraphics();
1637             g1.setColor(Color.WHITE);
1638             g1.fillRect(0,0,width,height);
1639             Graphics2D g2 = (Graphics2D)g1 ;
1640             dg.plot(g2);
1641             dg.plotXYCoordinates(g2);
1642             dg.drawGraph(g2);
1643             dg.drawDepth(g2);
1644             dg.plotZCoordinates(g2);
1645             //dg.plotXYCoordinates(g2);
1646             dg.drawDen(g2);
1647             dg.plot(g2);
1648             dg.idex(g2);
1649
1650             FileOutputStream os = new FileOutputStream(
GRAVLIS_CalculateValues.input_area_name + ".jpg");
1651             ImageIO.write(buffer, "jpg", os);
1652             os.close();
1653
1654             String path = GRAVLIS_CalculateValues.input_area_name + ".jpg";
1655             BufferedImage image = ImageIO.read(new File(path));
1656
1657             Graphics g_image = GRAVLIS_MainPanel.img.getGraphics();
1658             g_image.drawImage(image, -40, 0, image.getWidth(), image.getHeight(), dg);
1659
1660         }
1661         catch (Exception e2) {
1662
1663             // e2.printStackTrace();
1664         }
1665     }

```

```

1666     }
1667 }
1668 package com.gravlis.model;
1669
1670 import java.awt.Color;
1671 import java.awt.Graphics;
1672 import java.awt.Graphics2D;
1673 import java.awt.image.BufferedImage;
1674 import java.io.File;
1675 import java.io.FileOutputStream;
1676 import java.text.DecimalFormat;
1677 import java.util.HashMap;
1678
1679 import javax.imageio.ImageIO;
1680 import com.gravlis.model.GRAVLIS_CalNOREQ;
1681 import com.gravlis.model.GRAVLIS_CalculateValues;
1682 import com.gravlis.util.GRAVLIS_Utility;
1683 import com.gravlis.view.GRAVLIS_MainPanel;
1684 import com.gravlis.view.GRAVLIS_TableView;
1685
1686 public class GRAVLIS_CalculateValues {
1687
1688     public static Object obj[][] = null;
1689     public static double input_x_km[]; public static double input_nob_gob[];
1690     public static double []o_GC ; public static double []o_err ; public static double
1691     []o_par; public static double []o_funcnt ;
1692     public static double []cft; public static double []cft1; public static double
1693     []denval; public static double []input_den_val;
1694     public static double []input_ele_km; public static double []input_min_val; public static
1695     double []input_max_val;
1696     public static double []par; public static double []var; public static double
1697     []input_zval;
1698     public static double input_y_km, input_sd_poly, input_al_err, o_func, input_strike_km;
1699     public static int o_iter, input_num_for, np, input_nob_iter, input_n_obs;
1700     public static int input_nregcft = 2;
1701     public static String input_area_name, input_profile_num = "";
1702     public static String parValue = "";
1703     public static BufferedImage image;
1704
1705     public void getAnamolyValues(HashMap h_Map) {
1706
1707         try {
1708
1709             input_n_obs = GRAVLIS_Utility.convertInteger((String)h_Map.get("N_OBS"));
1710             input_x_km = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("X_KM"));
1711             input_ele_km = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("ELE_KM"));
1712             input_nob_gob =
1713             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
1714             input_sd_poly = GRAVLIS_Utility.convertDouble((String)h_Map.get("SD_POLY"));
1715             input_strike_km = GRAVLIS_Utility.convertDouble((String)h_Map.get("STRIKE_KM"));
1716             input_y_km = GRAVLIS_Utility.convertDouble((String)h_Map.get("Y_KM"));
1717             input_zval = GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("Z_VAL"));
1718             input_num_for = GRAVLIS_Utility.convertInteger((String)h_Map.get("NUM_FOR"));
1719             input_area_name = GRAVLIS_Utility.convertString((String)h_Map.get("AREA_FE"));
1720             input_profile_num =
1721             GRAVLIS_Utility.convertString((String)h_Map.get("NUM_PROFILE"));
1722             input_min_val =
1723             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("MIN_VAL"));
1724             input_max_val =
1725             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("MAX_VAL"));
1726             input_den_val =
1727             GRAVLIS_Utility.convertDoubleArray((String)h_Map.get("DEN_VAL"));
1728             input_nob_iter = GRAVLIS_Utility.convertInteger((String)h_Map.get("NUM_ITE"));
1729             input_al_err = GRAVLIS_Utility.convertDouble((String)h_Map.get("AL_ERR"));
1730         }
1731         catch(Exception e) {
1732             e.printStackTrace();
1733         }
1734         par = new double[input_nregcft + input_num_for + 2];
1735         var = new double[input_num_for + 1];
1736     }
1737
1738     public void getFalDen(){

```

```

1732
1733     o_GC = new double[input_n_obs + 1];
1734     o_err = new double[input_n_obs + 1];
1735     o_par = new double[input_n_obs + 1];
1736     o_funct = new double[input_nob_iter + 1];
1737
1738     double []g1 = new double[input_n_obs + 1];
1739     double []g2 = new double[input_n_obs + 1];
1740     double gc[] = new double[input_n_obs + 1];
1741     double err[] = new double[input_n_obs + 1];
1742
1743     double [][]p = new double[input_nregcft + input_num_for + 2][input_nregcft +
input_num_for + 3];
1744     double [][]s = new double[input_nregcft + input_num_for + 2][input_n_obs + 1];
1745     double []b = new double[input_nregcft + input_num_for + 2];
1746     double []par1 = new double[input_nregcft + input_num_for + 2];
1747     double []par2 = new double[input_nregcft + input_num_for + 2];
1748     double []dpar = new double[input_nregcft + input_num_for + 2];
1749
1750     cft = new double[input_n_obs + 1];
1751     cft1 = new double[input_n_obs + 1];
1752     double []KS = new double[2];
1753     double funct2 = 0;
1754     double lambda = 0.5;
1755     np = input_nregcft + input_num_for + 1;
1756     denval = new double[input_num_for + 1];
1757     double funct1 = 0;
1758
1759     double datum = input_nob_gob[1];
1760     double r = input_nob_gob[input_n_obs] - input_nob_gob[1];
1761     int kk = 1;
1762     double gh = 0.5 * r;
1763     double XH = 0;
1764     while ((( input_nob_gob[kk] - datum) / gh ) - 1.0 < 0) {
1765         kk = kk + 1;
1766     }
1767     if ((( input_nob_gob[kk] - datum) / gh) - 1.0 > 0) {
1768         XH = input_x_km[kk-1] + ( ( gh + datum - input_nob_gob[kk-1] ) * ( input_x_km[kk]
- input_x_km[kk-1] ) ) / ( input_nob_gob[kk] - input_nob_gob[kk-1] );
1769     }
1770     if ((( input_nob_gob[kk] - datum ) / gh) - 1.0 == 0) {
1771         XH = input_x_km[kk];
1772     }
1773     cft[1] = XH;
1774     for(int k=1; k <= input_nregcft + 1; k++){
1775         par[k] = cft[k];
1776     }
1777     double dpar = 0.1;
1778     if(var == input_zval)
1779         GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km, par, np,
input_zval, gc);
1780     else
1781         GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km, par, np,
input_den_val, gc);
1782     for (int k = 1; k <= input_n_obs; k++) {
1783         err[k] = input_nob_gob[k] - gc[k];
1784         funct1 = funct1 + Math.pow( err[k] , 2 );
1785     }
1786
1787     lambda = 0.5;
1788     int NP1 = np + 1;
1789     int IER = 1;
1790
1791     while (IER <= input_nob_iter) {
1792
1793         int ITER1 = IER ;
1794         o_funct[ITER1] = funct1;
1795
1796         for (int K = 1; K <= np; K++) {
1797             par1[K] = par[K];
1798         }
1799
1800         for (int I = 1; I <= np; I++) {

```

```

1801         par1[I] = par[I] + dpar / 2.0;
1802         if(var == input_zval)
1803             GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_zval, g1);
1804         else
1805             GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_den_val, g1);
1806         par1[I] = par[I] - dpar / 2.0;
1807         if(var == input_zval)
1808             GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_zval, g2);
1809         else
1810             GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par1, np, input_den_val, g2);
1811
1812         for (int K = 1; K <= input_n_obs; K++) {
1813             s[I][K] = ( g1[K] - g2[K] ) / dpar;
1814         }
1815     }
1816     for (int J = 1; J <= NP1; J++) {
1817         for(int I = 1; I <= np; I++) {
1818             p[I][J] = 0.0;
1819         }
1820     }
1821     for (int J = 1; J <= np; J++) {
1822         for (int I = 1; I <= np; I++) {
1823             for (int K = 1; K <= input_n_obs; K++){
1824                 p[I][J] = p[I][J] + s[I][K] * s[J][K];
1825             }
1826         }
1827     }
1828
1829     for (int J = 1; J <= np; J++) {
1830         for (int K = 1; K <= input_n_obs; K++) {
1831             p[J][NP1] = p[J][NP1] + err[K] * s[J][K];
1832         }
1833     }
1834
1835     do {
1836         double con = lambda + 1.0;
1837         for (int I = 1; I <= np; I++) {
1838             dupar[I] = par[I];
1839         }
1840         for (int L = 1; L <= np; L++) {
1841             for (int J = 1; J <= np; J++) {
1842                 if (L - J == 0)
1843                     p[L][J] = p[L][J] * con;
1844             }
1845         }
1846         GRAVLIS_CalNOREQ.getNOREQ(p, b, np, KS);
1847         for (int I = 1; I <= np; I++) {
1848             par2[I] = dupar[I] + 0.25 * b[I];
1849         }
1850         if(var==input_zval){
1851             for(int j =1; j <= input_num_for; j++){
1852                 if (par2[input_nregcft + 1 + j] < input_min_val[j]) {
1853                     par2[input_nregcft +1 + j] = input_min_val[j];
1854                 }
1855                 if (par2[input_nregcft + 1 + j]>input_max_val[j]) {
1856                     par2[input_nregcft+ 1 + j] = input_max_val[j];
1857                 }
1858             }
1859         }
1860         if(var == input_zval)
1861             GRAVLIS_CalNOREQ.getGF (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par2, np, input_zval, gc);
1862         else
1863             GRAVLIS_CalNOREQ.getGF1 (input_n_obs, input_sd_poly, input_num_for,
input_nregcft, input_ele_km, input_strike_km, input_y_km, input_x_km,
par2, np, input_den_val, gc);

```

```

1864         funct2 = 0.0;
1865         for (int K = 1; K <= input_n_obs; K++) {
1866             err[K] = input_nob_gob[K] - gc[K];
1867             funct2 = funct2 + Math.pow(err[K] , 2);
1868         }
1869         if (funct1 - funct2 < 0) {
1870
1871             lambda = lambda * 2.0;
1872             if (lambda - 12 < 0) {
1873
1874                 for (int I = 1; I <= np; I++) {
1875                     for (int J = 1; J <= np; J++) {
1876                         if(I - J == 0)
1877                             p[I][J] = p[I][J] / con;
1878                     }
1879                 }
1880             }
1881             else
1882                 break;
1883
1884         }
1885
1886     } while (funct1 - funct2 < 0);
1887
1888     funct1 = funct2;
1889     DecimalFormat d = new DecimalFormat("0.##");
1890     IER++;
1891     for (int I = 1; I <= np; I++) {
1892         par[I] = par2[I];
1893     }
1894     o_iter = ITER1;
1895     o_func = funct2;
1896     parValue = d.format(par[1]);
1897     for (int K = 1; K <= input_n_obs; K++) {
1898         o_GC[K] = gc[K];
1899         o_err[K] = err[K];
1900     }
1901     for (int l = 1; l <= np; l++) {
1902         o_par[l] = par[l];
1903     }
1904     for(int j =1; j <= input_nregcft + 1; j++){
1905         cft1[j] = par[j];
1906     }
1907     if(var == input_zval){
1908         for(int j =1; j <= input_num_for; j++){
1909             denval[j] = par[input_nregcft + 1 + j];
1910         }
1911     }
1912     else{
1913         for(int j = 1; j <= input_num_for; j++){
1914             input_zval[j] = par[input_nregcft + 1 + j];
1915             denval[j] = input_den_val[j];
1916         }
1917     }
1918 }
1919
1920
1921 if ( funct2 < input_al_err || ITER1 == input_nob_iter||lambda -12 > 0 ) {
1922
1923     o_iter = ITER1;
1924     o_func = funct2;
1925     parValue = d.format(par[1]);
1926     for (int K = 1; K <= input_n_obs; K++) {
1927         o_GC[K] = gc[K];
1928         o_err[K] = err[K];
1929     }
1930     for (int l = 1; l <= np; l++) {
1931         o_par[l] = par[l];
1932     }
1933
1934     setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
o_err, o_par, o_func, input_area_name);
1935     drawGraph();
1936 }
1937 else{

```



```

1938         setGraphValues(input_n_obs, np, o_iter, input_x_km, input_nob_gob, o_GC,
o_err, o_par, o_func, input_area_name);
1939         drawGraph();
1940     }
1941
1942     try {
1943         Thread.sleep(10);
1944     } catch (InterruptedException e) {
1945         // TODO Auto-generated catch block
1946         e.printStackTrace();
1947     }
1948
1949     if ( funct2 < input_al_err || lambda -12 > 0)
1950         break;
1951
1952     lambda = lambda / 2.0;
1953 }
1954 }
1955 }
1956
1957
1958 public static void setGraphValues(int i_no_obs, int np, int ite, double []dis, double
[]GOBS, double []GCAL, double []ERROR, double []PARA, double FUNCT, String Area_fe) {
1959
1960     obj = new Object[i_no_obs+1][4];
1961
1962     DecimalFormat df = new DecimalFormat("0.###");
1963     DecimalFormat d = new DecimalFormat("0.##");
1964     DecimalFormat dl = new DecimalFormat("0.#####");
1965     for(int K=1; K <= i_no_obs; K++){
1966         obj[K][0]= " " + dis[K];
1967         obj[K][1]= " " + df.format(GOBS[K]);
1968         obj[K][2]= " " + df.format(GCAL[K]);
1969         obj[K][3]= " " + df.format(ERROR[K]);
1970     }
1971
1972     obj[0][0] ="ITERATION";
1973     obj[0][1] = "=" + " "+ite;
1974
1975     GRAVLIS_TableView.val.setText("");
1976     GRAVLIS_TableView.val.appendText("ITERATION NUMBER:-"+ite);
1977     GRAVLIS_TableView.val.appendText("\n");
1978     GRAVLIS_TableView.val.appendText("OBJECTIVE FUNCTION "+dl.format(FUNCT)+"\n");
1979     GRAVLIS_TableView.val.appendText("\n");
1980     GRAVLIS_TableView.val.appendText("INTERPRETED PARAMETERS:-\n");
1981     GRAVLIS_TableView.val.appendText("-----\n");
1982     GRAVLIS_TableView.val.appendText("\n");
1983     GRAVLIS_TableView.val.appendText("COEFFICIENTS OF THE POLYNOMIAL ");
1984     GRAVLIS_TableView.val.appendText("\n");
1985     GRAVLIS_TableView.val.appendText("-----
----\n");
1986
1987     GRAVLIS_TableView.val.appendText(dl.format(PARA[1])+"\n");
1988     GRAVLIS_TableView.val.appendText(dl.format(PARA[2])+"\n");
1989     GRAVLIS_TableView.val.appendText(dl.format(PARA[3])+"\n");
1990     GRAVLIS_TableView.val.appendText("\n");
1991
1992     if(var == input_zval)
1993         GRAVLIS_TableView.val.appendText("ESTIMATED DENSITIES OF THE FORMATIONS
:-"+ "\n");
1994     else
1995         GRAVLIS_TableView.val.appendText("ESTIMATED DEPTHS TO THE INTERFACES :-"+ "\n");
1996     GRAVLIS_TableView.val.appendText("-----
-----\n");
1997     //GRAVLIS_TableView.val.appendText("\n");
1998     for(int k = 4; k <= input_nregcft + input_num_for + 1; k++){
1999         GRAVLIS_TableView.val.appendText(d.format(PARA[k])+"\n");
2000     }
2001 }
2002
2003 public static void drawGraph(){
2004     com.gravlis.view.GRAVLIS_DrawGraph dg = new com.gravlis.view.GRAVLIS_DrawGraph();
2005     try
2006     {
2007         int width = 1280;

```

```

2008         int height = 650;
2009         BufferedImage buffer = new
BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
2010         Graphics g1= buffer.createGraphics();
2011         g1.setColor(Color.WHITE);
2012         g1.fillRect(0,0,width,height);
2013         Graphics2D g2 = (Graphics2D)g1 ;
2014         dg.plot(g2);
2015         dg.plotXYCoordinates(g2);
2016         dg.drawGraph(g2);
2017         dg.drawDepth(g2);
2018         dg.plotZCoordinates(g2);
2019         //dg.plotXYCoordinates(g2);
2020         dg.drawDen(g2);
2021         dg.plot(g2);
2022         dg.idex(g2);
2023
2024         FileOutputStream os = new FileOutputStream(
GRAVLIS_CalculateValues.input_area_name + ".jpg");
2025         ImageIO.write(buffer, "jpg", os);
2026         os.close();
2027
2028         String path = GRAVLIS_CalculateValues.input_area_name + ".jpg";
2029         BufferedImage image = ImageIO.read(new File(path));
2030
2031         Graphics g_image = GRAVLIS_MainPanel.img.getGraphics();
2032         g_image.drawImage(image, -40, 0, image.getWidth(), image.getHeight(), dg);
2033
2034     }
2035     catch (Exception e2) {
2036
2037         // e2.printStackTrace();
2038     }
2039 }
2040 }
2041 }
2042 -----
2043 package com.gravlis.model;
2044
2045 public class GRAVLIS_CalNOREQ {
2046
2047     public static double []vsd = null;
2048     public static double []dep = null;
2049     public static int N2=0;
2050
2051     public static void main(String[] args) {
2052
2053         //Methods that support the main class
2054     }
2055
2056     public static double []getGF(int n,double bden,int nfm,int ndgre,double ele[],double
strk,double offset,double []x,double par[],double np,double []zr,double []ano) {
2057
2058         double []cft = new double[ndgre + 2];
2059         double []den = new double[nfm + 2];
2060         for(int k = 1; k <= ndgre + 1; k++){
2061             cft[k] = par[k];
2062         }
2063         for(int k = 1; k <= nfm; k++){
2064             den[k] = par[ndgre + 1 + k];
2065         }
2066         int effd = 0;
2067         int jjk = 0;
2068         int kk=1;
2069         double zt =0;
2070         double []Z ;
2071         double []YY = new double[3];
2072         double []GS ;
2073         double []GG = new double[3];
2074         double wgc[][] = new double[2500][2500];
2075         double []gmod = new double[2500];
2076         double []gdmod = new double[2500];
2077         double GC = 0;
2078

```

```

2079
2080     do{
2081         effd = effd + 1;
2082         YY[1] = strk + offset;
2083         YY[2] = strk - offset;
2084         double DX = ( x[2] - x[1] ) / 10;
2085         double ZB = zr[kk];
2086         double zdif = ZB - zt;
2087         int NDIV = (int)( zdif / DX ) + 1;
2088         int N1 = NDIV / 2;
2089         if (NDIV - ( 2 * N1 ) < 0 || NDIV - ( 2 * N1 ) > 0) {
2090             NDIV = NDIV + 1;
2091         }
2092         double DZ = zdif / NDIV;
2093         N2 = NDIV + 1;
2094         Z = new double[N2+1];
2095         GS = new double[N2+1];
2096         vsd = new double[N2+1];
2097         dep = new double[N2+1];
2098         for (int JZ = 1; JZ <= N2; JZ++) {
2099             Z[JZ] = zt + DZ * ( JZ - 1 );
2100         }
2101
2102         double dc = den[effd] - bden;
2103         for (int K = 1; K <= n; K++) {
2104             double XX = x[K] ;
2105
2106             for ( int JZ = 1; JZ <= N2; JZ++) {
2107                 double sum = 0;
2108                 for(int jj = 1; jj <= ndgre + 1; jj++){
2109                     sum = sum + cft[jj] * Math.pow(Z[JZ], jj - 1);
2110                 }
2111                 vsd[JZ] = dc;
2112                 dep[JZ] = Z[JZ];
2113
2114                 for (int KL = 1; KL <= 2; KL++) {
2115                     double EFFY = YY[KL];
2116                     double tr1 = Math.atan( EFFY / ( Z[JZ] - ele[K] ) );
2117                     double ttp = Math.pow( ( -XX + sum ) , 2 ) + Math.pow( EFFY , 2 ) +
2118 Math.pow( ( Z[JZ] - ele[K] ) , 2 );
2119                     double tr2 = Math.atan( ( EFFY * ( -XX + sum ) ) / ( ( Z[JZ] - ele[K]
2120 ) * Math.sqrt( ttp ) ) );
2121                     GG[KL] = 13.3333 * dc * ( tr1 - tr2 );
2122                 }
2123                 GS[JZ] = ( ( GG[2] + GG[1] ) / 2 );
2124             }
2125             gmod[K]= getSIMP(GS,Z,N2,GC);
2126             jjk = jjk + 1;
2127             gdmod[jjk] = gmod[K];
2128         }
2129         kk = kk + 1;
2130         if(kk - nfm <= 0)
2131             zt = zr[kk - 1];
2132     }while(kk - nfm <= 0);
2133     for (int lk = 1; lk <= n; lk++){
2134         int klkl = lk;
2135         for (int jh = 1; jh <= nfm; jh++){
2136             wgc[lk][jh] = gdmod[klkl];
2137             klkl = klkl + n;
2138         }
2139     }
2140     for (int il = 1; il <= n; il++){
2141         double sum = 0.0;
2142         for (int ih = 1; ih <= nfm; ih++){
2143             sum = sum + wgc[il][ih];
2144         }
2145         ano[il] = sum;
2146     }
2147     return ano;
2148 }
2149 public static double []getGF1(int n,double bden,int nfm,int ndgre,double ele[],double
strk,double offset,double []x,double par[],double np,double []den,double []ano) {
2150

```

```

2151     double []cft = new double[ndgre + 2];
2152     double []dep = new double[nfm + 5];
2153     for(int k = 1; k <= ndgre + 1; k++){
2154         cft[k] = par[k];
2155     }
2156     for(int k=1; k <= nfm; k++){
2157         dep[k] = par[ndgre + 1 + k];
2158     }
2159     int effd = 0;
2160     int jjk = 0;
2161     int kk = 1;
2162     double zt =0;
2163     double []Z ;
2164     double []YY = new double[3];
2165     double []GS ;
2166     double []GG = new double[3];
2167     double wgc[][] = new double[2500][2500];
2168     double []gmod = new double[2500];
2169     double []gdmod = new double[2500];
2170     double GC = 0;
2171
2172     do{
2173         effd = effd+1;
2174         YY[1] = strk + offset;
2175         YY[2] = strk - offset;
2176         double DX = ( x[2] - x[1] ) / 10;
2177         double ZB = dep[kk];
2178         double zdif = ZB - zt;
2179         int NDIV = (int)( zdif / DX ) + 1;
2180         int N1 = NDIV / 2;
2181         if (NDIV - ( 2 * N1 ) < 0 || NDIV - ( 2 * N1 ) > 0 ) {
2182             NDIV = NDIV + 1;
2183         }
2184         double DZ = zdif / NDIV;
2185         N2 = NDIV + 1;
2186         Z = new double[N2 + 1];
2187         GS = new double[N2 + 1];
2188         vsd = new double[N2 + 1];
2189         for (int JZ = 1; JZ <= N2; JZ++) {
2190             Z[JZ] = zt + DZ * ( JZ - 1 );
2191         }
2192         double dc = den[effd] - bden;
2193         for (int K = 1; K <= n; K++) {
2194             double XX = x[K] ;
2195             for ( int JZ = 1; JZ <= N2; JZ++) {
2196                 double sum = 0;
2197                 for(int jj = 1; jj <= ndgre + 1; jj++){
2198                     sum = sum + cft[jj] * Math.pow(Z[JZ], jj - 1);
2199                 }
2200                 vsd[JZ] = dc;
2201
2202                 for (int KL = 1; KL <= 2; KL++) {
2203                     double EFFY = YY[KL];
2204                     double tr1 = Math.atan( EFFY / ( Z[JZ] - ele[K] ) );
2205                     double ttp = Math.pow( ( -XX + sum) , 2) + Math.pow( EFFY , 2) +
Math.pow( ( Z[JZ] - ele[K] ) , 2 );
2206                     double tr2 = Math.atan( ( EFFY * ( -XX + sum) ) / ( ( Z[JZ] - ele[K] ) * Math.sqrt( ttp ) ) );
2207                     GG[KL] = 13.3333 * dc * ( tr1 - tr2 );
2208                 }
2209                 GS[JZ] = ( ( GG[2] + GG[1] ) / 2);
2210             }
2211             gmod[K] = getSIMP(GS,Z,N2,GC);
2212             jjk = jjk + 1;
2213             gdmod[jjk] = gmod[K];
2214         }
2215         kk = kk + 1;
2216         if(kk - nfm <= 0)
2217             zt = dep[kk - 1];
2218     }while(kk - nfm <= 0);
2219     for (int lk = 1; lk <= n; lk++){
2220         int klkl = lk;
2221         for (int jh = 1; jh <= nfm; jh++){
2222             wgc[lk][jh] = gdmod[klkl];
2223             klkl = klkl + n;

```

```

2224     }
2225 }
2226
2227
2228     for (int il = 1; il <= n; il++){
2229         double sum = 0.0;
2230         for (int ih = 1; ih <= nfm; ih++){
2231             sum = sum + wgc[il][ih];
2232         }
2233         ano[il] = sum;
2234     }
2235     return ano;
2236 }
2237 public static double getSIMP(double []gs,double []z,int n,double ggc) {
2238
2239     double dz = z[2] - z[1];
2240     double sum1 = 0.0;
2241     double sum2 = 0.0;
2242     int n1 = n / 2;
2243     int n4 = n1 - 1;
2244     for(int I = 1; I <= n1; I++) {
2245         int n2 = 2 * I;
2246         sum1 = sum1 + gs[n2];
2247     }
2248     for(int I = 1; I <= n4; I++) {
2249         int n3 = 2 * I + 1;
2250         sum2 = sum2 + gs[n3];
2251     }
2252     ggc = gs[1] + 4 * sum1 + 2 * sum2 + gs[n];
2253     ggc = ggc * dz / 3.0;
2254     return ggc;
2255 }
2256 public static double []getNOREQ(double p[][], double b[], int n, double KS[]) {
2257
2258     int I = n + 1;
2259     double []a = new double[n * n + 1];
2260     for (int I1 = 1; I1 <= n; I1++) {
2261         for (int I2 = 1; I2 <= n; I2++) {
2262             int I3 = (I1 - 1) * n + I2;
2263             a[I3] = p[I2][I1];
2264         }
2265     }
2266
2267     for (int I4 = 1; I4 <= n; I4++) {
2268         b[I4] = p[I4][I];
2269     }
2270     double TOL = 0;
2271     KS[0] = 0;
2272     int JJ = - n;
2273     int IT;
2274     int NY = 0;
2275     for (int J = 1; J <= n; J++) {
2276         int JY = J + 1;
2277         JJ = JJ + n + 1;
2278         double biga = 0;
2279         IT = JJ - J;
2280         int imax = 0;
2281         for (int i = J; i <= n; i++) {
2282             int IJ = IT + i;
2283             if (Math.abs(biga) - Math.abs(a[IJ]) < 0) {
2284                 biga = a[IJ];
2285                 imax = i;
2286             }
2287         }
2288         int I1 = 0;
2289         if (Math.abs(biga) - TOL <= 0) {
2290             KS[1] = 1;
2291             return KS;
2292         }
2293         else {
2294             I1 = J + n * (J - 2);
2295             IT = imax - J;
2296         }
2297         double save;
2298         for (int K = J; K <= n; K++) {

```

```

2299         I1 = I1 + n;
2300         int I2 = I1 + IT;
2301         save = a[I1];
2302         a[I1] = a[I2];
2303         a[I2] = save;
2304         a[I1] = a[I1] / biga;
2305     }
2306
2307     save = b[imax];
2308     b[imax] = b[J];
2309     b[J] = save / biga;
2310     int IQS = 0;
2311
2312     if (J - n < 0 || J - n > 0) {
2313         IQS = n * (J - 1);
2314         for (int IX = JY; IX <= n; IX++) {
2315             int IXJ = IQS + IX;
2316             IT = J - IX;
2317             for (int JX = JY; JX <= n; JX++) {
2318                 int IXJX = n * (JX - 1) + IX;
2319                 int JJX = IXJX + IT;
2320                 a[IXJX] = a[IXJX] - (a[IXJ] * a[JJX]);
2321             }
2322             b[IX] = b[IX] - (b[J] * a[IXJ]);
2323         }
2324     }
2325 }
2326 NY = n - 1;
2327 IT = n * n;
2328 for (int J = 1; J <= NY; J++) {
2329     int ia = IT - J;
2330     int ib = n - J;
2331     int ic = n;
2332     for (int K = 1; K <= J; K++) {
2333         b[ib] = b[ib] - a[ia] * b[ic];
2334         ia = ia - n;
2335         ic = ic - 1;
2336     }
2337 }
2338 return b;
2339 }
2340 }
2341
2342

```

```

2343 package com.gravlis.control;
2344
2345 import java.awt.event.ActionEvent;
2346 import java.awt.event.ActionListener;
2347 import java.awt.*;
2348 import java.io.*;
2349 import java.text.DecimalFormat;
2350 import javax.swing.*;
2351
2352
2353
2354 import com.gravlis.model.GRAVLIS_CalculateValues;
2355 import com.gravlis.view.GRAVLIS_MainPanel;
2356 import com.gravlis.view.GRAVLIS_TableView;
2357
2358
2359 public class GRAVLIS_Controller implements ActionListener{
2360
2361     String rowdata[][]={};
2362     com.gravlis.model.GRAVLIS_CalculateValues cv = new
2363 com.gravlis.model.GRAVLIS_CalculateValues();
2364     FileWriter myWriter = null;
2365     public static boolean success = false;
2366
2367     public void actionPerformed(ActionEvent ae) {
2368         if(ae.getActionCommand().equals("Interpretation with Fixed Depth")) {
2369             com.gravlis.view.GRAVLIS_TableView.populateEastPanel(rowdata);
2370
2371

```

```

2372 GRAVLIS_TableView.val.setText("");
2373
2374 cv.getAnamolyValues(com.gravlis.view.GRAVLIS_MainPanel.captureValues());
2375 for(int k=1;k<=GRAVLIS_CalculateValues.input_num_for;k++){
2376     GRAVLIS_CalculateValues.par[k+GRAVLIS_CalculateValues.input_nregcft+1] =
GRAVLIS_CalculateValues.input_den_val[k];
2377 }
2378 GRAVLIS_CalculateValues.var = GRAVLIS_CalculateValues.input_zval;
2379
2380 cv.getFalDen();
2381 com.gravlis.view.GRAVLIS_TableView.populateEastPanel(GRAVLIS_CalculateValues.o
);
2382 GRAVLIS_CalculateValues.obj = null;
2383
2384 com.gravlis.view.GRAVLIS_MainPanel.p_East.repaint();
2385 com.gravlis.view.GRAVLIS_MainView mv = new com.gravlis.view.GRAVLIS_MainView().
2386 mv.setResizable(true);
2387
2388 }else if(ae.getActionCommand().equals("Interpretation with Fixed Density")){
2389     com.gravlis.view.GRAVLIS_TableView.populateEastPanel(rowdata);
2390     GRAVLIS_TableView.val.setText("");
2391
2392     cv.getAnamolyValues(com.gravlis.view.GRAVLIS_MainPanel.captureValues());
2393     for(int k=1;k<=GRAVLIS_CalculateValues.input_num_for;k++){
2394         GRAVLIS_CalculateValues.par[k+GRAVLIS_CalculateValues.input_nregcft+1] =
GRAVLIS_CalculateValues.input_zval[k];
2395     }
2396     GRAVLIS_CalculateValues.var = GRAVLIS_CalculateValues.input_den_val;
2397
2398     cv.getFalDen();
2399
2400     com.gravlis.view.GRAVLIS_TableView.populateEastPanel(GRAVLIS_CalculateValues.o
);
2401 GRAVLIS_CalculateValues.obj = null;
2402
2403     com.gravlis.view.GRAVLIS_MainPanel.p_East.repaint();
2404     com.gravlis.view.GRAVLIS_MainView mv = new com.gravlis.view.GRAVLIS_MainView().
2405     mv.setResizable(true);
2406 }
2407 else if(ae.getActionCommand().equals("Save & Print")){
2408     try{
2409         String current = System.getProperty("user.dir");
2410         File img_file = new File( GRAVLIS_CalculateValues.input_area_name+".jpg");
2411         JFileChooser saveFile = new JFileChooser(current);
2412         File OutFile = saveFile.getSelectedFile();
2413         if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
2414         {
2415             OutFile = saveFile.getSelectedFile();
2416
2417             if (OutFile.canWrite() || !OutFile.exists())
2418             {
2419                 File dir = new File(OutFile.getParent());
2420                 GRAVLIS_Controller.success = img_file.renameTo(new
File(dir,img_file.getName()));
2421                 //System.out.println("save successful" + success);
2422                 myWriter = new FileWriter(OutFile+".html");
2423                 myWriter.write(" </table> </td> <td> <img src = '"+
GRAVLIS_CalculateValues.input_area_name
+ ".jpg"></td></tr></table>");
2424                 myWriter.write("<html><Body onLoad = \"window.print()\"> <table>
<tr> <td> " +
2425                 "<table border = 1> <tr> <th colspan = 4>LOCATION:-
"+GRAVLIS_CalculateValues.input_area_name+"</th> </tr>");
2426
2427                 DecimalFormat df =new DecimalFormat("0.###");
2428
2429                 myWriter.write(" <tr><th colspan = 4> PROFILE NUMBER:-"+
"+GRAVLIS_CalculateValues.input_profile_num+ " </th></tr>");
2430                 myWriter.write(" <tr><th colspan = 4> ITERATION"+
"+GRAVLIS_CalculateValues.o_iter+ " </th></tr>");
2431
2432                 myWriter.write("<tr > <th>Distance (km) </th> <th> Observed
anomalies (mGal) </th> <th> Calculated anomalies (mGal) </th> <th>

```

```

2434 Error (mGal) </th></tr>");
2435
2436         for ( int K = 1; K <= GRAVLIS_CalculateValues.input_n_obs; K++){
2437             myWriter.write("<tr> <td>" +
GRAVLIS_CalculateValues.input_x_km[K]+"</td>
<td>"+df.format(GRAVLIS_CalculateValues.input_nob_gob[K])+"</td>
<td>"+df.format(GRAVLIS_CalculateValues.o_GC[K])+"</td>
<td>"+df.format(GRAVLIS_CalculateValues.o_err[K])+"</td></tr>");
2438         }
2439         myWriter.write("</table>");
2440
2441         DecimalFormat d =new DecimalFormat("0.##");
2442         DecimalFormat dl =new DecimalFormat("0.#####");
2443
2444         myWriter.write("<BR>");
2445         myWriter.write("OBJECTIVE FUNCTION ="+
"+dl.format(GRAVLIS_CalculateValues.o_func)+"<BR>");
2446         myWriter.write("<BR>");
2447         myWriter.write("INTERPRETED PARAMETERS:"+<BR>");
2448         myWriter.write("----- <BR>");
2449         myWriter.write("<BR>");
2450         myWriter.write(" COEFFICIENTS OF THE POLYNOMIAL :-"+<BR>");
2451         myWriter.write("-----
<BR>");
2452         myWriter.write(dl.format(GRAVLIS_CalculateValues.o_par[1])+"<BR>");
2453         myWriter.write(dl.format(GRAVLIS_CalculateValues.o_par[2])+"<BR>");
2454         myWriter.write(dl.format(GRAVLIS_CalculateValues.o_par[3])+"<BR>");
2455         myWriter.write("<BR>");
2456         if(GRAVLIS_CalculateValues.var==GRAVLIS_CalculateValues.input_zval
2457             myWriter.write("ESTIMATED DENSITIES OF THE FORMATIONS
:-"+<BR>");
2458         else
2459             myWriter.write("ESTIMATED DEPTHS TO THE INTERFACES :-"+<BR>");
2460         myWriter.write("-----
---- <BR>");
2461         for ( int i = 4; i <=
GRAVLIS_CalculateValues.input_nregcft+GRAVLIS_CalculateValues.input_
num_for+1; i++) {
2462             myWriter.write(d.format(GRAVLIS_CalculateValues.o_par[i])+"<BR>
);
2463         }
2464         myWriter.close();
2465     }
2466 }
2467 else
2468 {
2469     //pops up error message
2470 }
2471 }
2472 }
2473 }
2474 catch(Exception e1) {
2475     e1.printStackTrace();
2476 }
2477 }
2478 }else if(ae.getActionCommand().equals("Load data")){
2479     try {
2480         GRAVLIS_MainPanel.loadData1();
2481     } catch (IOException e) {
2482         // TODO Auto-generated catch block
2483         e.printStackTrace();
2484     }
2485 }else if(ae.getActionCommand().equals("Clear")){
2486     GRAVLIS_MainPanel.clearDefaultValues();
2487     clearStaticValues();
2488     com.gravlis.view.GRAVLIS_MainPanel.clearPanel(GRAVLIS_MainPanel.p_Center);
2489     com.gravlis.view.GRAVLIS_TableView.populateEastPanel(rowdata);
2490     GRAVLIS_TableView.val.setText("");
2491     Graphics g = GRAVLIS_MainPanel.img.getGraphics();
2492     g.setColor(Color.white);
2493     g.fillRect(0, 0, 1000, 600);
2494 }else if(ae.getActionCommand().equals("Exit")){
2495     JFrame frame = null;
2496

```



```

2497         int r = JOptionPane.showConfirmDialog(
2498             frame,
2499             "Exit GRAVLIS ?",
2500             "Confirm Exit ",
2501             JOptionPane.YES_NO_OPTION);
2502         if(r == JOptionPane.YES_OPTION ){
2503             if(success==false){
2504                 String fileName = GRAVLIS_CalculateValues.input_area_name+".jpg";
2505                 File f = new File(fileName);
2506                 f.delete();
2507             }
2508             System.exit(0);
2509         }
2510     }
2511 }
2512 }
2513 public void clearStaticValues(){
2514     GRAVLIS_CalculateValues.cft = null;
2515     GRAVLIS_CalculateValues.cft1 = null;
2516     GRAVLIS_CalculateValues.denval = null;
2517     GRAVLIS_CalculateValues.input_al_err = 0;
2518     GRAVLIS_CalculateValues.input_area_name = null;
2519     GRAVLIS_CalculateValues.input_den_val = null;
2520     GRAVLIS_CalculateValues.input_ele_km = null;
2521     GRAVLIS_CalculateValues.input_max_val = null;
2522     GRAVLIS_CalculateValues.input_min_val = null;
2523     GRAVLIS_CalculateValues.input_n_obs = 0;
2524     GRAVLIS_CalculateValues.input_nob_gob = null;
2525     GRAVLIS_CalculateValues.input_nob_iter = 0;
2526     GRAVLIS_CalculateValues.input_num_for = 0;
2527     GRAVLIS_CalculateValues.input_sd_poly = 0;
2528     GRAVLIS_CalculateValues.input_strike_km = 0;
2529     GRAVLIS_CalculateValues.input_x_km = null;
2530     GRAVLIS_CalculateValues.input_y_km = 0;
2531     GRAVLIS_CalculateValues.input_zval = null;
2532     GRAVLIS_CalculateValues.o_err = null;
2533     GRAVLIS_CalculateValues.o_funcnt = null;
2534     GRAVLIS_CalculateValues.o_GC = null;
2535     GRAVLIS_CalculateValues.o_iter = 0;
2536     GRAVLIS_CalculateValues.o_par = null;
2537     System.gc();
2538 }
2539 }
2540 }
2541 -----

```

```

2542 package com.gravlis.util;
2543
2544
2545 import javax.swing.JFrame;
2546 import javax.swing.JOptionPane;
2547
2548
2549
2550 public class GRAVLIS_Utility {
2551
2552     public static double convertDouble(String str) throws Exception {
2553
2554
2555         Double temp = null;
2556
2557         try {
2558             temp = new Double(str.trim());
2559         }
2560     }
2561     catch(Exception e){
2562
2563         JFrame frame = null;
2564         JOptionPane.showMessageDialog(frame,
2565             "Enter a numerical value.",
2566             "Number format error",
2567             JOptionPane.ERROR_MESSAGE);
2568
2569
2570

```

```

2571     }
2572     return temp.doubleValue();
2573 }
2574
2575 public static String convertString(String str) throws Exception {
2576
2577     String temp = new String(str.trim());
2578     return temp;
2579 }
2580
2581 public static int convertInteger(String str) throws Exception {
2582
2583     Integer temp = null;
2584     try {
2585         temp = new Integer(str.trim());
2586     }
2587     catch(Exception e){
2588
2589         JFrame frame = null;
2590         JOptionPane.showMessageDialog(frame,
2591             "Enter a numerical value.",
2592             "Number format error",
2593             JOptionPane.ERROR_MESSAGE);
2594     }
2595     return temp.intValue();
2596 }
2597
2598 }
2599
2600 public static double findMaximumNumber( double observe[]) {
2601
2602     double max = 0.0d;
2603     for (int i = 0; i < observe.length; i++) {
2604
2605         if (Math.abs(observe[i]) > Math.abs(max)) {
2606
2607             max = observe[i];
2608         }
2609     }
2610
2611     double maxVal = max/3*5;
2612     return maxVal;
2613 }
2614
2615 public static double findMinimumNumber( double observe[], double denVal) {
2616
2617     double max = denVal;
2618     for (int i = 1; i < observe.length; i++) {
2619
2620         if (Math.abs(observe[i]) < Math.abs(max)) {
2621
2622             max = Math.abs(observe[i]);
2623         }
2624     }
2625
2626     double maxVal = max;
2627     return maxVal;
2628 }
2629
2630 public static double findMinimumNumber1( double observe[]) {
2631
2632     double max = 0.0d;
2633     for (int i = 1; i < observe.length; i++) {
2634
2635         if ((observe[i]) < (max)) {
2636
2637             max = (observe[i]);
2638         }
2639     }
2640
2641     double maxVal = max;
2642     return maxVal;
2643 }
2644 public static double findMaximumNumber1( double observe[]) {
2645

```

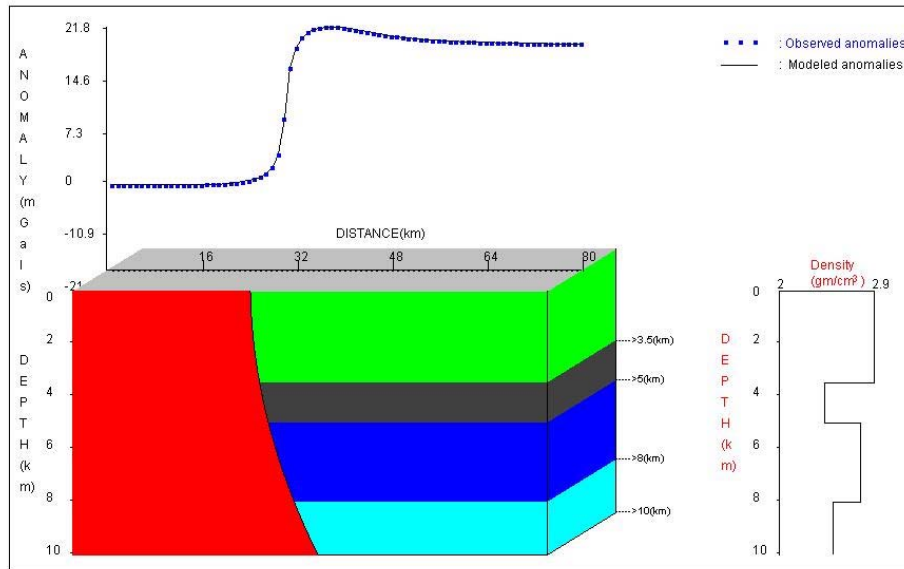
```

2646     double max = 0.0d;
2647     for (int i = 1; i < observe.length; i++) {
2648
2649         if (Math.abs(observe[i]) > Math.abs(max)) {
2650
2651             max = Math.abs(observe[i]);
2652         }
2653     }
2654
2655     double maxVal = max;
2656     return maxVal;
2657 }
2658 public static double findMaximumNumber( double observe[], double anoVal) {
2659
2660     double max = anoVal;
2661     for (int i = 1; i < observe.length; i++) {
2662
2663         if ((observe[i]) > (max)) {
2664
2665             max = (observe[i]);
2666         }
2667     }
2668
2669     double maxVal = max;
2670     return maxVal;
2671 }
2672
2673
2674
2675 public static double[] convertDoubleArray(String str) throws Exception {
2676
2677     java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
2678     String temp = "";
2679     java.util.ArrayList arr = new java.util.ArrayList();
2680
2681     while(st.hasMoreTokens()) {
2682
2683         temp = st.nextToken();
2684         arr.add(temp);
2685     }
2686     double d_array[] = new double[arr.size() + 1];
2687
2688     for (int i = 0; i <= arr.size(); i++) {
2689
2690         if (i == 0)
2691             d_array[i] = 0.0;
2692         else {
2693
2694             try {
2695                 d_array[i] = convertDouble( arr.get(i - 1).toString() );
2696             }
2697             catch(Exception e){
2698                 JFrame frame = null;
2699                 JOptionPane.showMessageDialog(frame,
2700                     "Enter numerical values.",
2701                     "Number format error",
2702                     JOptionPane.ERROR_MESSAGE);
2703                 // throw new FGM_HandleException();
2704             }
2705         }
2706     }
2707     return d_array;
2708 }
2709 }
2710

```

# Annexure 4-B

## Sample output



LOCATION:- Sample-Fixed depths			
PROFILE NUMBER:- 1			
ITERATION 69			
Distance (km)	Observed anomalies (mGal)	Calculated anomalies (mGal)	Error (mGal)
1.0	-0.043	-0.047	0.004
2.0	-0.042	-0.047	0.005
3.0	-0.04	-0.046	0.006
4.0	-0.039	-0.044	0.005
5.0	-0.036	-0.043	0.007
6.0	-0.034	-0.04	0.006
7.0	-0.03	-0.037	0.007
8.0	-0.026	-0.034	0.008
9.0	-0.021	-0.029	0.008
10.0	-0.015	-0.024	0.009
11.0	-0.008	-0.017	0.009
12.0	0.001	-0.009	0.01
13.0	0.012	0.001	0.011
14.0	0.025	0.013	0.012
15.0	0.042	0.029	0.013
16.0	0.062	0.048	0.014
17.0	0.087	0.072	0.015
18.0	0.118	0.102	0.016
19.0	0.157	0.139	0.018
20.0	0.206	0.188	0.018
21.0	0.27	0.251	0.019
22.0	0.353	0.334	0.019
23.0	0.465	0.445	0.02
24.0	0.618	0.599	0.019
25.0	0.834	0.819	0.015
26.0	1.156	1.146	0.01
27.0	1.663	1.663	0
28.0	2.534	2.549	-0.015
29.0	4.247	4.284	-0.037
30.0	9.209	9.205	0.004
31.0	16.205	16.205	0
32.0	18.958	18.96	-0.002
33.0	20.341	20.349	-0.008
34.0	21.067	21.061	0.006
35.0	21.472	21.441	0.031
36.0	21.698	21.654	0.044
37.0	21.808	21.764	0.044
38.0	21.83	21.798	0.032
39.0	21.785	21.769	0.016
40.0	21.69	21.689	0.001
41.0	21.561	21.573	-0.012
42.0	21.413	21.433	-0.02
43.0	21.257	21.28	-0.023
44.0	21.1	21.125	-0.025

45.0	20.949	20.973	-0.024
46.0	20.807	20.829	-0.022
47.0	20.676	20.696	-0.02
48.0	20.557	20.574	-0.017
49.0	20.449	20.463	-0.014
50.0	20.351	20.363	-0.012
51.0	20.264	20.274	-0.01
52.0	20.185	20.193	-0.008
53.0	20.115	20.121	-0.006
54.0	20.052	20.057	-0.005
55.0	19.995	19.998	-0.003
56.0	19.944	19.946	-0.002
57.0	19.897	19.899	-0.002
58.0	19.855	19.856	-0.001
59.0	19.817	19.818	-0.001
60.0	19.783	19.783	0
61.0	19.752	19.751	0.001
62.0	19.723	19.721	0.002
63.0	19.697	19.695	0.002
64.0	19.672	19.67	0.002
65.0	19.65	19.648	0.002
66.0	19.63	19.627	0.003
67.0	19.611	19.608	0.003
68.0	19.594	19.591	0.003
69.0	19.578	19.574	0.004
70.0	19.563	19.559	0.004
71.0	19.549	19.545	0.004
72.0	19.536	19.532	0.004
73.0	19.524	19.52	0.004
74.0	19.513	19.509	0.004
75.0	19.502	19.498	0.004
76.0	19.492	19.488	0.004
77.0	19.483	19.479	0.004
78.0	19.474	19.47	0.004
79.0	19.466	19.462	0.004
80.0	19.459	19.454	0.005

OBJECTIVE FUNCTION = 0.01634

INTERPRETED PARAMETERS:

-----

COEFFICIENTS OF THE POLYNOMIAL :-

-----

30.03632  
0.07052  
0.10606

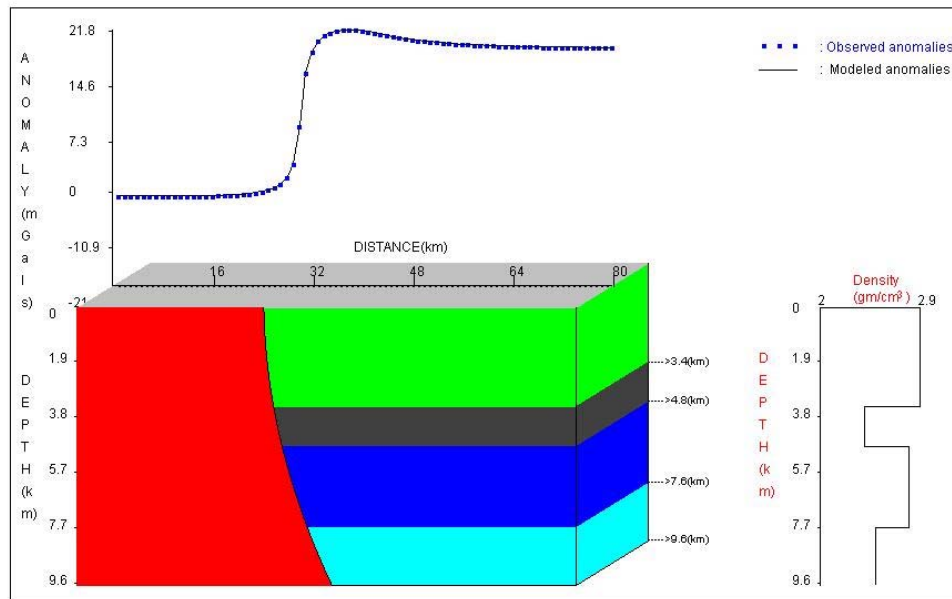
ESTIMATED DENSITIES OF THE FORMATIONS :-

-----

2.9  
2.44  
2.78  
2.51

# Annexure 4-C

## Sample output



LOCATION:- Sample-Fixed densities			
PROFILE NUMBER:- 1			
ITERATION 45			
Distance (km)	Observed anomalies (mGal)	Calculated anomalies (mGal)	Error (mGal)
1.0	-0.043	-0.041	-0.002
2.0	-0.042	-0.04	-0.002
3.0	-0.04	-0.039	-0.001
4.0	-0.039	-0.037	-0.002
5.0	-0.036	-0.035	-0.001
6.0	-0.034	-0.032	-0.002
7.0	-0.03	-0.029	-0.001
8.0	-0.026	-0.025	-0.001
9.0	-0.021	-0.021	-0
10.0	-0.015	-0.015	-0
11.0	-0.008	-0.008	-0
12.0	0.001	0.001	0
13.0	0.012	0.011	0.001
14.0	0.025	0.024	0.001
15.0	0.042	0.04	0.002
16.0	0.062	0.06	0.002
17.0	0.087	0.084	0.003
18.0	0.118	0.114	0.004
19.0	0.157	0.153	0.004
20.0	0.206	0.202	0.004
21.0	0.27	0.265	0.005
22.0	0.353	0.348	0.005
23.0	0.465	0.459	0.006
24.0	0.618	0.612	0.006
25.0	0.834	0.83	0.004
26.0	1.156	1.153	0.003
27.0	1.663	1.663	0
28.0	2.534	2.537	-0.003
29.0	4.247	4.254	-0.007
30.0	9.209	9.208	0.001
31.0	16.205	16.208	-0.003
32.0	18.958	18.959	-0.001
33.0	20.341	20.344	-0.003
34.0	21.067	21.059	0.008
35.0	21.472	21.453	0.019
36.0	21.698	21.682	0.016
37.0	21.808	21.803	0.005
38.0	21.83	21.838	-0.008
39.0	21.785	21.802	-0.017
40.0	21.69	21.711	-0.021

41.0	21.561	21.582	-0.021
42.0	21.413	21.429	-0.016
43.0	21.257	21.266	-0.009
44.0	21.1	21.103	-0.003
45.0	20.949	20.946	0.003
46.0	20.807	20.8	0.007
47.0	20.676	20.666	0.01
48.0	20.557	20.545	0.012
49.0	20.449	20.435	0.014
50.0	20.351	20.338	0.013
51.0	20.264	20.25	0.014
52.0	20.185	20.173	0.012
53.0	20.115	20.103	0.012
54.0	20.052	20.041	0.011
55.0	19.995	19.985	0.01
56.0	19.944	19.935	0.009
57.0	19.897	19.89	0.007
58.0	19.855	19.849	0.006
59.0	19.817	19.813	0.004
60.0	19.783	19.779	0.004
61.0	19.752	19.749	0.003
62.0	19.723	19.721	0.002
63.0	19.697	19.696	0.001
64.0	19.672	19.673	-0.001
65.0	19.65	19.651	-0.001
66.0	19.63	19.632	-0.002
67.0	19.611	19.614	-0.003
68.0	19.594	19.597	-0.003
69.0	19.578	19.582	-0.004
70.0	19.563	19.567	-0.004
71.0	19.549	19.554	-0.005
72.0	19.536	19.542	-0.006
73.0	19.524	19.53	-0.006
74.0	19.513	19.52	-0.007
75.0	19.502	19.51	-0.008
76.0	19.492	19.5	-0.008
77.0	19.483	19.491	-0.008
78.0	19.474	19.483	-0.009
79.0	19.466	19.476	-0.01
80.0	19.459	19.468	-0.009

OBJECTIVE FUNCTION = 0.00485

INTERPRETED PARAMETERS:

COEFFICIENTS OF THE POLYNOMIAL :-

30.03018  
0.07627  
0.11048

ESTIMATED DEPTHS TO THE INTERFACES :-

3.44  
4.8  
7.59  
9.57

```

1 package com.mod2dgrexp.view;
2
3 import java.awt.*;
4 import java.awt.event.WindowAdapter;
5 import java.awt.event.WindowEvent;
6 import java.io.File;
7
8 import javax.swing.JFrame;
9 import javax.swing.JOptionPane;
10
11
12 import com.mod2dgrexp.control.MOD2DGREXP_Controller;
13 import com.mod2dgrexp.model.MOD2DGREXP_CalculateValues;
14
15 public class MOD2DGREXP_MainView extends JFrame {
16
17     /**
18      *
19      */
20     private static final long serialVersionUID = 1L;
21
22     public static void main(String s[]) {
23
24         MOD2DGREXP_MainView cm = new MOD2DGREXP_MainView();
25         cm.setSize(1280, 768);
26         cm.addWindowListener(new WindowAdapter(){
27             public void windowClosing(WindowEvent e){
28                 JFrame frame = null;
29
30                 int r = JOptionPane.showConfirmDialog(
31                     frame,
32                     "Exit MOD2DGREXP ?",
33                     "Confirm Exit ",
34                     JOptionPane.YES_NO_OPTION);
35                 if(r == JOptionPane.YES_OPTION ){
36                     if(MOD2DGREXP_Controller.success==false){
37                         String fileName = MOD2DGREXP_CalculateValues.input_area_name+".jpg"
38                         File f = new File(fileName);
39                         f.delete();
40                     }
41                     System.exit(0);
42                 }
43             }
44         });
45         cm.setTitle("MOD2DGREXP");
46         cm.setResizable(false);
47         cm.add(new MOD2DGREXP_MainPanel());
48         cm.setVisible(true);
49     }
50 }
51
52 }
53
-----

```

```

54 package com.mod2dgrexp.view;
55
56 import java.awt.*;
57 import java.io.File;
58 import java.io.IOException;
59 import java.util.HashMap;
60 import javax.swing.JFileChooser;
61
62 import jxl.Cell;
63 import jxl.CellType;
64 import jxl.Sheet;
65 import jxl.Workbook;
66 import jxl.read.biff.BiffException;
67
68
69
70 public class MOD2DGREXP_MainPanel extends Panel {
71
72     /**
73      *
74

```



```

75     */
76     private static final long serialVersionUID = 1L;
77
78     Panel p_North, p_West, p_South;
79     public static TextArea img = new TextArea(36,135);
80     public static Panel p_East, p_Center;
81     static TextField inputValues [] = new TextField[12];
82     static TextArea graphValues = new TextArea(38,30);
83     Button actionButton[] = new Button[5];
84     Object rowdata[][]={};
85
86     /**Field Area Name*/
87     final static int AREA_FE = 0;
88     /**Profile Name*/
89     final static int PROFILE_NAME = 1;
90     /**Number of observation*/
91     public static final int N_OBS = 2 ;
92     /**Distance(km)*/
93     public static final int X_KM = 3;
94     /**Number of Observed values*/
95     public static final int NOB_GOB = 4;
96     /** SD */
97     public static final int SD_POLY = 5;
98     /**LAMBDA*/
99     public static final int LAMBDA_VAL = 6 ;
100    /**ZMIN(km)*/
101    public static final int DEP_ZMIN = 7;
102    /**ZMAX(km)*/
103    public static final int DEP_ZMAX = 8;
104    /**Number of iteration values*/
105    public static final int NOB_ITER = 9;
106
107
108    public MOD2DGREXP_MainPanel() {
109
110        this.setLayout(new BorderLayout());
111        p_North = new Panel();
112        p_West = new Panel();
113        p_East = new Panel ();
114        p_South = new Panel();
115        p_Center = new Panel();
116
117        Label graphLabel = new Label("AUTOMATIC MODELING OF GRAVITY ANOMALIES OF 2D
SEDIMENTARY BASINS ", Label.CENTER);
118        Label graphLabel1 = new Label(" USING EXPONENTIAL DENSITY MODEL", Label.CENTER);
119        graphLabel.setFont(new Font("Arial", 10, 18));
120        graphLabel1.setFont(new Font("Arial", 10, 18));
121        p_Center.add(graphLabel);
122        p_Center.add(graphLabel1);
123
124        for(int i = 0; i < 11; i++){
125            inputValues[i] = new TextField();
126        }
127        p_North.setFont(new Font("Bold",1,12));
128        actionButton[0] = new Button("Load data");
129        actionButton[1] = new Button("Modeling");
130        actionButton[2] = new Button("Save and Print");
131        actionButton[3] = new Button("Clear");
132        actionButton[4] = new Button("Exit");
133
134        this.populateNorthPanel();
135        MOD2DGREXP_TableView.populateEastPanel(rowdata);
136        this.add(p_North, BorderLayout.NORTH);
137        this.add(p_West, BorderLayout.WEST);
138        this.add(p_East, BorderLayout.EAST);
139        this.add(p_South, BorderLayout.SOUTH);
140        p_Center.setSize(1000, 760);
141        this.add(p_Center, BorderLayout.CENTER);
142        img.setEditable(false);
143        p_Center.add(img);
144        this.setVisible(true);
145    }
146
147    public void populateNorthPanel(){
148        p_North.setLayout(new GridLayout(5,6));

```

```

149
150     p_North.add(new Label("Area Name"));
151     p_North.add(inputValues[0]);
152     p_North.add(new Label("Profile Name"));
153     p_North.add(inputValues[1]);
154     p_North.add(new Label("Number of observation:"));
155     p_North.add(inputValues[2]);
156     p_North.add(new Label("Distance (km)"));
157     p_North.add(inputValues[3]);
158     //p_North.add(new Label("Elevation of each station (km)"));
159     //p_North.add(inputValues[4]);
160     p_North.add(new Label("Observed anomalies (mGal)"));
161     p_North.add(inputValues[4]);
162     p_North.add(new Label("Surface density contrast (gm/cc)"));
163     p_North.add(inputValues[5]);
164     p_North.add(new Label("Lambda (1/km)"));
165     p_North.add(inputValues[6]);
166     p_North.add(new Label("Minimum depth(km):"));
167     p_North.add(inputValues[7]);
168     p_North.add(new Label("Maximum depth(km):"));
169     p_North.add(inputValues[8]);
170     p_North.add(new Label("Iterations"));
171     p_North.add(inputValues[9]);
172     p_North.add(new Label(""));
173     p_North.add(new Label(""));
174     p_North.add(new Label(""));
175     p_North.add(new Label(""));
176
177
178     p_North.add(actionButton[0]);
179     p_North.add(actionButton[1]);
180     p_North.add(actionButton[2]);
181     p_North.add(actionButton[3]);
182     p_North.add(actionButton[4]);
183
184     actionPerformed[0].addActionListener(new
com.mod2dgrexp.control.MOD2DGREXP_Controller());
185     actionPerformed[1].addActionListener(new
com.mod2dgrexp.control.MOD2DGREXP_Controller());
186     actionPerformed[2].addActionListener(new
com.mod2dgrexp.control.MOD2DGREXP_Controller());
187     actionPerformed[3].addActionListener(new
com.mod2dgrexp.control.MOD2DGREXP_Controller());
188     actionPerformed[4].addActionListener(new
com.mod2dgrexp.control.MOD2DGREXP_Controller());
189
190 }
191
192 public static HashMap captureValues(){
193
194     HashMap h_Map = new HashMap();
195
196     try {
197
198         h_Map.put("N_OBS", inputValues[N_OBS].getText());
199         h_Map.put("SD_POLY", inputValues[SD_POLY].getText());
200         h_Map.put("LAMBDA_VAL", inputValues[LAMBDA_VAL].getText());
201         h_Map.put("DEP_ZMIN", inputValues[DEP_ZMIN].getText());
202         h_Map.put("DEP_ZMAX", inputValues[DEP_ZMAX].getText());
203         h_Map.put("X_KM", inputValues[X_KM].getText());
204         h_Map.put("NOB_GOB", inputValues[NOB_GOB].getText());
205         h_Map.put("NOB_ITER", inputValues[NOB_ITER].getText());
206         h_Map.put("AREA_FE", inputValues[AREA_FE].getText());
207         h_Map.put("PROFILE_NAME", inputValues[PROFILE_NAME].getText());
208     }
209     catch (Exception e) {
210         e.printStackTrace();
211     }
212
213     return h_Map;
214
215 }
216
217 public static void clearPanel(TextArea p) {
218

```

```

219         Graphics g = p.getGraphics();
220         g.setColor(Color.WHITE);
221         g.fillRect(0, 0, 1280, 650);
222     }
223
224
225
226
227     public static void loadData1()throws IOException {
228         try{
229
230             String current = System.getProperty("user.dir");
231             JFileChooser chooser=new JFileChooser(current);
232             int returnVal = chooser.showOpenDialog(null);
233             String dis[],gobs[];
234             String disval = "" ,gobsval="";
235             Workbook w;
236
237             if(returnVal == JFileChooser.APPROVE_OPTION) {
238                 File f = chooser.getSelectedFile();
239                 w = Workbook.getWorkbook(f);
240                 Sheet sheet = w.getSheet(0);
241                 dis = new String[sheet.getRows()+1];
242                 //ele = new String[sheet.getRows()+1];
243                 gobs = new String[sheet.getRows()+1];
244                 for (int j = 0; j < sheet.getColumns(); j++) {
245                     for (int i = 1; i < sheet.getRows(); i++) {
246                         Cell cell = sheet.getCell(j, i);
247                         CellType type = cell.getType();
248                         if (type == CellType.LABEL) {
249                             // System.out.println("I got a label "
250                                 // + cell.getContents());
251                             MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.AREA_FE].
etText(cell.getContents());
252
253                         }
254
255                         if (type == CellType.NUMBER) {
256                             if (j == 1){
257
258                                 MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.PROF:
E_NAME].setText(cell.getContents());
259
260                             }
261                             if (j == 2){
262
263                                 MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.N_OB:
.setText(cell.getContents());
264
265                             }
266
267                             if (j == 3){
268
269                                 dis[i] = cell.getContents()+",";
270                                 disval = disval + dis[i];
271                             }
272                             //if (j == 4){
273
274                                 // ele[i] = cell.getContents()+",";
275                                 // eleval = eleval + ele[i];
276                                 //}
277                             if (j == 4){
278
279                                 gobs[i] = cell.getContents()+",";
280                                 gobsval = gobsval+gobs[i];
281                             }
282
283                             if (j == 5){
284
285                                 MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.SD_PC
Y].setText(cell.getContents());
286
287                             }
288                             if (j == 6){
289

```

```

290         MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.LAMBD
_VAL].setText(cell.getContents());
291     }
292
293     if (j == 7){
294
295         MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.DEP_Z
296     IN].setText(cell.getContents());
297     }
298
299     if (j == 8){
300
301         MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.DEP_Z
302     AX].setText(cell.getContents());
303     }
304
305     if (j == 9){
306         MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.NOB_I
307     ER].setText(cell.getContents());
308     }
309
310     //System.out.println("I got a number "
311     // + cell.getContents());
312 }
313 }
314 }
315 MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.X_KM].setText(" "+disv
1);
316 MOD2DGREXP_MainPanel.inputValues[MOD2DGREXP_MainPanel.NOB_GOB].setText(" "+g
bsval);
317 }
318 }
319 catch (BiffException e) {
320     e.printStackTrace();
321 }
322 }
323 }
324 }
325 }
326 }
327
328 public static void clearDefaultValues() {
329
330     inputValues[N_OBS].setText("");
331     inputValues[SD_POLY].setText("");
332     inputValues[LAMBDA_VAL].setText("");
333     inputValues[DEP_ZMIN].setText("");
334     inputValues[DEP_ZMAX].setText("");
335     inputValues[X_KM].setText("");
336     inputValues[NOB_GOB].setText("");
337     inputValues[NOB_ITER].setText("");
338     inputValues[AREA_FE].setText("");
339     inputValues[PROFILE_NAME].setText("");
340 }
341 }
342 }
343 -----
344 package com.mod2dgrexp.view;
345
346 import java.applet.Applet;
347 import java.awt.*;
348 import java.awt.geom.Line2D;
349 import java.awt.geom.Rectangle2D;
350 import java.text.DecimalFormat;
351 import com.mod2dgrexp.model.MOD2DGREXP_CalculateValues;
352 import com.mod2dgrexp.util.MOD2DGREXP_Utility;
353
354
355 public class MOD2DGREXP_DrawGraph extends Applet {
356
357

```

```

358     /**
359     *
360     */
361     private static final long serialVersionUID = 1L;
362     float maxZ, maxX, maxY;
363     double inidep;
364     public void drawGraph(Graphics2D g){
365         g.setColor(Color.black);
366         g.drawLine(150,50,150,550);
367         g.drawLine(90,45,1040,45);
368         g.drawString("DISTANCE(Km)",335,295);
369         String a[] = { " A ", " N ", " O ", " M ", " A ", " L ", " Y ", "(m ", " G ", " a
", " l ", " s)" };
370         String b[] = { " D ", " E ", " P ", " T ", " H ", "(k ", " m)" };
371         for (int i = 0; i < a.length; i++) {
372
373             g.drawString(" "+ a[i], 100, 60 + (i * 20));
374         }
375         for (int i = 0; i < b.length; i++) {
376
377             g.drawString(" "+b[i], 100, 350 + (i * 20));
378         }
379     }
380
381
382     public void plot(Graphics2D g){
383         DecimalFormat df = new DecimalFormat("0.#");
384         DecimalFormat df1 = new DecimalFormat("0.##");
385         maxX = (float)
MOD2DGREXP_Utility.findMaximumNumber1(MOD2DGREXP_CalculateValues.input_x_km);
386         maxY =
MOD2DGREXP_Utility.findMaximumNumber(MOD2DGREXP_CalculateValues.input_nob_gob);
387         inidep = MOD2DGREXP_Utility.findMaximumNumber1(MOD2DGREXP_CalculateValues.z);
388         maxZ = (float) inidep;
389         g.setColor(Color.BLACK);
390         g.drawString("0",140,60);
391         g.drawString(""+(int)maxY,125,300);
392
393         g.drawString("0",125,310);
394         g.drawString("|",600,308);
395         g.drawString(""+df.format(MOD2DGREXP_CalculateValues.input_x_km[MOD2DGREXP_Calculat
Values.input_n_obs]),600,320);
396         g.draw(new Line2D.Float(150,300,600,300));
397         float points = maxX / 5;
398
399         int yInterval = 50;
400         int zInterval = 50;
401
402         float xInterval=(float)
(MOD2DGREXP_CalculateValues.input_x_km[MOD2DGREXP_CalculateValues.input_n_obs]/5);
403         float xplot=0;
404
405         for (float x = xInterval, j =1; x < 600; x+=xInterval){
406
407             xplot = xplot + xInterval;
408             if(j > 4)
409                 break;
410             g.drawString("|",(float) (150 + (450 * x / maxX)), 308);
411             g.drawString(" "+ df.format(xplot), (float) (150 + (450 * x / maxX)) - 3, 323).
412                 j++;
413         }
414
415         points = maxY / 5;
416         for (int x = yInterval, j = 1; x < 250; x += yInterval){
417             g.drawString("-",148,50 + x);
418             g.drawString(" "+ (int)(points * j),125,50 + x);
419             j++;
420         }
421         float point = maxZ / 5 ;
422
423         for(int x = zInterval+250,j =1; x < 550; x+=zInterval){
424
425             if(j > 4)
426                 break;
427             g.drawString("-",148,50 + x);

```

```

428         g.drawString(" " + df.format(point * j), 125, 50 + x);
429         j++;
430     }
431     g.drawString("-", 148, 552);
432     g.drawString(" " + df1.format(maxZ), 125, 550);
433 }
434
435
436 public void plotXYCoordinates (Graphics2D g){
437
438     float prevx = 150;
439     float prevy = 50;
440     float xpoint = 0;
441     float ypoint = 0;
442     float gypoint = 0;
443     for (int k = 1; k <= MOD2DGREXP_CalculateValues.input_n_obs; k++){
444         xpoint = (float)(450 * MOD2DGREXP_CalculateValues.input_x_km[k] / maxX) ;
445         ypoint = (float)(250 * MOD2DGREXP_CalculateValues.gc[k] / maxY);
446         gypoint = (float)(250 * MOD2DGREXP_CalculateValues.input_nob_gob[k] / maxY);
447         g.setColor(Color.BLACK);
448         g.draw(new Line2D.Float(prevx, prevy, 150 + xpoint, 50 + ypoint));
449         g.setColor(Color.BLUE);
450         g.setFont(new Font("Arial", 20, 55));
451         g.drawString(".", 150 + xpoint - 6, 50 + gypoint);
452         prevx = 150 + xpoint;
453         prevy = 50 + ypoint;
454     }
455 }
456
457
458 public void plotZCoordinates (Graphics2D g){
459     float prevx = 150;
460     float prevz = 300 + (float)(250 * MOD2DGREXP_CalculateValues.input_zmin_km / maxZ
);
461
462     float xpoint = 0;
463     float zpoint = 0;
464     GradientPaint gradient = new GradientPaint(10, 10, Color.yellow, 30, 200,
Color.MAGENTA, true);
465     g.setPaint(gradient);
466     g.fill(new Rectangle2D.Float(151, 300, 450, 250));
467     for (int k = 1; k <= MOD2DGREXP_CalculateValues.input_n_obs; k++) {
468         xpoint = (float)(450 * MOD2DGREXP_CalculateValues.input_x_km[k] / maxX);
469         zpoint = (float)(250 * MOD2DGREXP_CalculateValues.z[k] / maxZ);
470         float vary = prevx;
471         g.setColor(Color.red);
472         if (prevz <= 300 + zpoint){
473             while (vary <= 150 + xpoint){
474                 g.draw(new Line2D.Float(prevx, prevz, vary, 300 + zpoint));
475                 vary = (float) (vary + 0.001);
476             }
477             g.fill(new Rectangle2D.Float(prevx, 300 + zpoint, (150 + xpoint) - prevx,
250 - zpoint));
478         }
479         else{
480             vary = prevz;
481             while (300 + zpoint <= vary){
482                 g.draw(new Line2D.Float(prevx, prevz, 150 + xpoint, vary));
483                 vary = (float) (vary - 0.001);
484             }
485             g.fill(new Rectangle2D.Float(prevx, prevz, (150 + xpoint) - prevx, 550 - prev
));
486         }
487         g.setColor(Color.black);
488         g.draw(new Line2D.Float(prevx, prevz, 150 + xpoint, 300 + zpoint));
489         prevx = 150 + xpoint;
490         prevz = 300 + zpoint;
491     }
492 }
493 g.setColor(Color.white);
494 g.fill(new Rectangle2D.Float(151, 550, 450, 50));
495
496
497 }
498

```

```

499     public void drawOBJ(Graphics2D g2) {
500
501         g2.setColor(Color.BLACK);
502         g2.drawLine(780, 45, 780, 560);
503         g2.drawLine(90, 560, 1040, 560);
504         g2.drawLine(1040, 45, 1040, 560);
505         g2.drawLine(90, 45, 90, 560);
506
507         g2.drawLine(820, 70, 820, 160);
508         g2.drawLine(820, 160, 910, 160);
509         g2.drawString("J", 800, 90);
510
511         double maxOb =
MOD2DGREXP_Utility.findMaximumNumber1(MOD2DGREXP_CalculateValues.o_funcnt) ;
512         int ini = MOD2DGREXP_Utility.findMaximumNumber(MOD2DGREXP_CalculateValues.o_iter);
513         if(ini == 5)
514             ini= ini+1;
515         int maxiter = ( ini / 3 * 5 ) * 2;
516         int point;
517         int xInterval = 22;
518         point = ( ( ini ) / 3 * 5 ) / 5;
519
520         for (int x = xInterval, j = 1; x < 90 ; x += xInterval) {
521
522             g2.drawString("", 821 + x, 170);
523             g2.drawString("" + (point * j), 820 + x - 3, 175);
524             j++;
525         }
526         float prevx = 820;
527         float prevy = 70;
528         float xpoint = 0;
529         float ypoint = 0;
530
531         for (int i = 1; i <= MOD2DGREXP_CalculateValues.o_iter; i++) {
532             xpoint = (float)( 250 * i /maxiter );
533             ypoint = 70 - (float) ( ( 90 * (MOD2DGREXP_CalculateValues.o_funcnt[i]) / maxOb
);
534
535             if(i==MOD2DGREXP_CalculateValues.o_iter){
536                 g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint - 4, 90 + ypoint));
537             }
538             else {
539                 g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 90 + ypoint));
540             }
541             prevx = 820 + xpoint;
542             prevy = 90 + ypoint;
543         }
544         DecimalFormat d1= new DecimalFormat("0.###");
545         DecimalFormat d= new DecimalFormat("0.#");
546         g2.drawString(""+d.format(MOD2DGREXP_CalculateValues.o_funcnt[1]), 780, 70);
547         g2.drawString("
"+d1.format(MOD2DGREXP_CalculateValues.o_funcnt[MOD2DGREXP_CalculateValues.o_iter]),
820 + xpoint, 90 + ypoint);
548         g2.setFont(new Font("Arial", 40,11));
549         g2.drawString ("Iterations",850,186);
550     }
551
552     public void drawSd(Graphics2D g2) {
553
554
555         g2.setColor(Color.black);
556         g2.drawLine(780, 200, 1040, 200);
557         g2.setColor(Color.red);
558         g2.setFont(new Font("Arial", 20, 12));
559         DecimalFormat d= new DecimalFormat("0.##");
560         DecimalFormat d1= new DecimalFormat("0.###");
561         g2.draw(new Line2D.Float(820, 300, 820, 550));
562         g2.drawString(""+d.format(inidep),790,550);
563         g2.drawString("-",820,550+2);
564         g2.drawString("0",807 ,300);
565         g2.drawLine(820, 300, 910, 300);
566         double maxOb1 =
MOD2DGREXP_Utility.findMaximumNumber1(MOD2DGREXP_CalculateValues.vsd);
567         DecimalFormat df = new DecimalFormat("0.#");
568         float points = (float)inidep / 5 ;

```

```

569         int zInterval = 50;
570         for(int x = zInterval+250,j = 1; x < 550; x+=zInterval){
571
572             if(j > 4)
573                 break;
574
575             g2.drawString("-",820,50 + x + 2);
576             g2.drawString(" "+df.format(points * j), 790, 50 + x);
577             j++;
578
579         }
580
581         float prevx = 820 + (float) ( ( 90 * ( Math.abs(MOD2DGREXP_CalculateValues.vsd[1] )
/ maxOb1 ) );
582         float prevy = 300 + (float)(250 * MOD2DGREXP_CalculateValues.input_zmin_km / maxZ );
583         float xpoint = 0;
584         float ypoint = 0;
585
586         for (int i = 1; i <= MOD2DGREXP_CalculateValues.count; i++) {
587
588             xpoint = (float)( 90 * Math.abs(MOD2DGREXP_CalculateValues.vsd[i]) / maxOb1 );
589             ypoint = (float)( 250 * MOD2DGREXP_CalculateValues.dep[i] / inidep );
590             g2.setColor(Color.blue);
591             g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 300 + ypoint));
592             prevx = 820 + xpoint;
593             prevy = 300 + ypoint;
594         }
595         g2.drawString(""+d.format(MOD2DGREXP_CalculateValues.vsd[1] ),805+ (float) ( ( 90 *
( Math.abs(MOD2DGREXP_CalculateValues.vsd[1] ) ) / maxOb1 ) ) ,300 );
596         g2.drawString(""+d1.format(MOD2DGREXP_CalculateValues.vsd[MOD2DGREXP_CalculateValue
.count] ),820+ (float) ( ( 90 * (
Math.abs(MOD2DGREXP_CalculateValues.vsd[MOD2DGREXP_CalculateValues.count] ) ) /
maxOb1 ) ) , 550);
597         g2.setColor(Color.BLACK);
598         g2.drawString("Variation of density contrast " , 800,220);
599         g2.drawString("with depth" , 850,240);
600         g2.setFont(new Font("Arial", 40,11));
601         g2.drawString ( "Density contrast",830,285);
602         g2.drawString ( "(gm/cc)",843,295);
603         g2.drawString("Z(km)", 790,(float)( 300+250/2));
604     }
605
606     public void idex(Graphics2D g){
607         g.setColor(Color.BLUE);
608         g.setFont(new Font("Arial", 20, 50));
609         g.drawString(" ... " ,595,70);
610         g.setFont(new Font("Arial", 20, 12));
611         g.drawString("Observed anomalies",650,70);
612         g.setColor(Color.BLACK);
613         g.drawString("____:" ,615,87);
614         g.drawString("Calculated anomalies",650,90);
615         g.setColor(Color.black);
616         g.drawString("<-----:" ,450,340);
617         g.setColor(Color.RED);
618         g.drawString("Estimated Depth " ,660,340);
619         g.drawString(" Structure",660,355);
620     }
621
622 }
623
624 -----
625
626 package com.mod2dgrexp.view;
627
628 import java.awt.Dimension;
629
630 import javax.swing.JScrollPane;
631 import javax.swing.JTable;
632
633 public class MOD2DGREXP_TableView {
634
635     public static void populateEastPanel(Object rowData[][] ) {
636         com.mod2dgrexp.view.MOD2DGREXP_MainPanel.p_East.removeAll();
637         Object columnNames[] = {"Distance(km)", "Observed anamolies (mGal)", "Calculated

```



```

638         anamolies (mGal)","Depth(km)");
639         JTable table = new JTable(rowData, columnNames);
640         table.setPreferredScrollableViewportSize(new Dimension(300,550));
641         JScrollPane scrollPane = new JScrollPane(table);
642         scrollPane.setAutoscrolls(true);
643         com.mod2dgrexp.view.MOD2DGREXP_MainPanel.p_East.add(scrollPane);
644         com.mod2dgrexp.view.MOD2DGREXP_MainPanel.p_East.validate();
645         com.mod2dgrexp.view.MOD2DGREXP_MainPanel.p_East.setVisible(true);
646     }
647 }
648 -----
649 package com.mod2dgrexp.model;
650
651 import java.awt.*;
652 import java.awt.event.*;
653 import java.awt.image.BufferedImage;
654 import java.io.File;
655 import java.io.FileOutputStream;
656 import java.text.DecimalFormat;
657 import java.util.HashMap;
658 import javax.imageio.ImageIO;
659 import com.mod2dgrexp.util.MOD2DGREXP_Utility;
660 import com.mod2dgrexp.view.MOD2DGREXP_DrawGraph;
661 import com.mod2dgrexp.view.MOD2DGREXP_MainPanel;
662
663
664 public class MOD2DGREXP_CalculateValues {
665
666     public static Object obj[][] = null;
667     public static double []o_funcnt;
668     public static int o_iter,count ;
669     public static double input_zmax_km = 0;
670     public static double input_zmin_km = 0;
671     public static int input_n_obs=0 ;
672     public static double []input_x_km;
673     public static double []input_nob_gob;
674     public static double []gc;
675     public static double []z;
676     public static double []vsd = null;
677     public static double []dep = null;
678     public static String input_area_name = "";
679     public static String input_profile="";
680     static BufferedImage image;
681
682     public void getAnamolyValues(HashMap h_Map) {
683
684         double PI = 22.0/7.0;
685         double GK = 13.3333;
686         double input_sd_poly = 0;
687         double input_lambda_val = 0;
688         int input_nob_iter = 0;
689         try {
690
691             input_n_obs = MOD2DGREXP_Utility.convertInteger((String)h_Map.get("N_OBS"));
692             input_sd_poly = MOD2DGREXP_Utility.convertDouble((String)h_Map.get("SD_POLY"));
693             input_lambda_val =
694             MOD2DGREXP_Utility.convertDouble((String)h_Map.get("LAMBDA_VAL"));
695             input_zmin_km = MOD2DGREXP_Utility.convertDouble((String)h_Map.get("DEP_ZMIN"));
696             input_zmax_km = MOD2DGREXP_Utility.convertDouble((String)h_Map.get("DEP_ZMAX"));
697             input_x_km = MOD2DGREXP_Utility.convertDoubleArray((String)h_Map.get("X_KM"));
698             input_nob_gob =
699             MOD2DGREXP_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
700             input_nob_iter =
701             MOD2DGREXP_Utility.convertInteger((String)h_Map.get("NOB_ITER"));
702             input_area_name =
703             MOD2DGREXP_Utility.convertString((String)h_Map.get("AREA_FE"));
704             input_profile =
705             MOD2DGREXP_Utility.convertString((String)h_Map.get("PROFILE_NAME"));
706         }
707         catch (Exception e) {
708             e.printStackTrace();
709         }
710     }

```

```

706
707     double ALER = 0.001*input_n_obs;
708     o_funct = new double[input_nob_iter+1];
709     gc = new double[input_n_obs+1];
710     double err[] = new double[input_n_obs+1];
711     z = new double[input_n_obs+2];
712     double dcz[] = new double [input_n_obs+1];
713
714     z[1]=0.0001;
715     z[input_n_obs]=0.0001;
716     for (int kk = 2; kk <= input_n_obs - 1; kk++) {
717         if(input_lambda_val==0)
718             z[kk]=input_nob_gob[kk]/(GK*PI*input_sd_poly);
719         else
720             z[kk]=-(1 / input_lambda_val) * Math.log(1 - ((input_lambda_val *
input_nob_gob[kk]) / (GK * PI * input_sd_poly)));
721     }
722
723     GF(input_n_obs,input_x_km,z,input_sd_poly,input_lambda_val,GK,gc);
724
725     double funct1 = 0;
726
727     for (int k = 1;k <= input_n_obs; k++) {
728
729         err[k] = input_nob_gob[k] - gc[k];
730         funct1 = funct1 + Math.pow(input_nob_gob[k] - gc[k], 2);
731     }
732     funct1=Math.sqrt(funct1 /input_n_obs);
733
734     for (int ITER = 1;ITER <= input_nob_iter; ITER++) {
735         for(int kk = 2; kk <= input_n_obs - 1; kk++) {
736             err[kk] = input_nob_gob[kk] - gc[kk];
737             if(input_lambda_val == 0)
738                 z[kk]= z[kk]+err[kk]/(GK*PI*input_sd_poly);
739             else{
740                 double st = GK*PI*input_sd_poly*Math.exp(-input_lambda_val*z[kk]);
741                 dcz[kk]=-(1/input_lambda_val)*Math.log(1-((input_lambda_val*err[kk])/st
);
742
743                 z[kk]=z[kk]+0.5*dcz[kk];
744             }
745             if (z[kk] <= input_zmin_km)
746                 z[kk] = input_zmin_km;
747
748             if (z[kk] > input_zmax_km)
749                 z[kk] = input_zmax_km;
750
751         }
752         GF(input_n_obs,input_x_km,z,input_sd_poly,input_lambda_val,GK,gc);
753
754         double funct2 = 0;
755         for (int LI = 1; LI <= input_n_obs; LI++) {
756
757             funct2 = funct2 + Math.pow((input_nob_gob[LI] - gc[LI]), 2);
758
759         }
760         funct2 = Math.sqrt(funct2/input_n_obs);
761         o_iter = ITER;
762         o_funct[ITER] = funct1;
763         setGraphValues(input_n_obs,o_iter, input_x_km,z, input_nob_gob, gc, funct1,
input_area_name);
764         denCal(input_sd_poly,input_lambda_val);
765         drawGraph();
766
767         if (funct2 - funct1 < 0 || funct2 - funct1 == 0) {
768
769             if (funct2 - ALER <= 0) {
770
771                 break;
772             }
773
774             else if (funct2 - ALER > 0) {
775
776                 funct1 = funct2;
777

```

```

778     }
779     else if (funct2 - funct1 > 0) {
780         break;
781     }
782 }
783 }
784 }
785 }
786 }
787
788 public static void denCal(double sd,double lambda){
789
790     int i = 1;
791     double z1=MOD2DGREXP_CalculateValues.input_zmin_km;
792     double z2 = MOD2DGREXP_Utility .findMaximumNumber1(MOD2DGREXP_CalculateValues.z);
793     vsd = new double[(int) Math.pow(input_n_obs,2)];
794     dep = new double[(int) Math.pow(input_n_obs,2)];
795     while(z1 <= z2){
796
797         double dc = sd*Math.exp(-lambda*z1);
798         vsd[i] = dc;
799         dep[i] = z1;
800
801         z1 = z1+0.1;
802         i++;
803     }
804
805     count = i;
806     vsd[count] = sd*Math.exp(-lambda*z2);
807     dep[count] = z2;
808
809 }
810
811
812 public double [] GF(int n,double []x,double []zv,double sd,double la,double gk,double
[]gc) {
813
814     double ggc=0;
815     double []xx = new double [n+2];
816     double []zt = new double[10000];
817     double []x1 = new double[10000];
818     double []gs = new double[10000];
819
820     for(int JJ = 1 ; JJ <= n ;JJ++){
821         gc[JJ]=0.0;
822     }
823     for(int k1=1;k1<=n;k1++){
824         for(int k2=1;k2<=n;k2++){
825             xx[k2]=x[k2]-x[k1];
826         }
827         xx[n+1]=xx[1];
828         zv[n+1]=zv[1];
829         double grav = 0;
830         for(int i=1;i<=n;i++){
831             double dxx = xx[i+1] - xx[i];
832             double dzv = zv[i+1] - zv[i];
833             double r = Math.sqrt(Math.pow(dxx,2)+Math.pow(dzv,2));
834             double c = dxx / r;
835             double s = dzv / r;
836             double ct = c / s;
837             double dx = (x[2]-x[1])/4;
838             double ZB = Math.abs(zv[i+1]-zv[i]);
839             int nd = (int)(ZB/dx)+1;
840             double n1 = nd / 2;
841             if (nd - ( 2 * n1 ) < 0 || nd - ( 2 * n1 ) > 0 ) {
842                 nd = nd + 1;
843             }
844             double DZ = ZB / nd;
845             int N2 = nd + 1;
846
847             if(zv[i+1]-zv[i]==0)
848                 break;
849             for(int JZ=1;JZ<=N2;JZ++){
850                 if(zv[i+1]-zv[i]<0) {
851                     zt[JZ] = zv[i] - DZ * (JZ-1);

```

```

852     }
853     if(zv[i+1]-zv[i]>0) {
854         zt[JZ] = zv[i] + DZ * (JZ-1);
855     }
856
857     if(zt[JZ]<0)
858         zt[JZ] = 0;
859     x1[JZ] = xx[i]+(zt[JZ]-zv[i]) * ct;
860     if(Math.abs(x1[JZ])<0.01)
861         x1[JZ] = 0;
862
863     }
864
865     for(int JZ=1;JZ<=N2;JZ++){
866         double DC = (sd * Math.exp(-la * zt[JZ]));
867         double a = xx[i] - zv[i] * ct;
868         double anum = a + zt[JZ] * ct;
869         gs[JZ]= -13.3333*DC*Math.atan(anum / zt[JZ]);
870
871     }
872     ggc=SIMP(gs,zt,N2,ggc);
873     grav = grav + ggc;
874
875     }
876
877     gc[k1] = grav;
878
879
880     }
881     return gc;
882 }
883
884 public double SIMP(double []gs,double []z,int n,double ggc) {
885
886     double dz = z[2]-z[1];
887     double sum1 = 0.0;
888     double sum2 = 0.0;
889     int n1 = n / 2;
890     int n4 = n1 - 1;
891     for(int I = 1; I <= n1; I++) {
892         int n2 = 2 * I;
893         sum1 = sum1 + gs[n2];
894     }
895     for(int I = 1; I <= n4; I++) {
896         int n3 = 2 * I +1;
897         sum2 = sum2 + gs[n3];
898     }
899     ggc = gs[1]+4*sum1+2*sum2+gs[n];
900     ggc = ggc * dz / 3.0;
901     return ggc;
902 }
903
904 public static void setGraphValues(int i_no_obs,int ite,double []dis,double []dep,doub
[]GOBS,double []GCAL,double FUNCT,String Area_fe) {
905
906     obj = new Object[i_no_obs+21][4];
907     DecimalFormat df =new DecimalFormat("0.###");
908     DecimalFormat d1 =new DecimalFormat("0.#####");
909     for(int K=1;K<=i_no_obs;K++){
910         obj[K][0]= "" + dis[K];
911         obj[K][1]= "" + df.format(GOBS[K]);
912         obj[K][2]= "" + df.format(GCAL[K]);
913         obj[K][3]= "" + df.format(dep[K]);
914     }
915
916     obj[0][0] ="ITERATION";
917     obj[0][1] = "=" + "" +ite;
918     obj[i_no_obs+2][0] = "OBJECTIVE " ;
919     obj[i_no_obs+2][1] = "FUNCTION =";
920     obj[i_no_obs+2][2] = d1.format(FUNCT);
921 }
922
923 public static void drawGraph(){
924     final MOD2DGREXP_DrawGraph dg = new MOD2DGREXP_DrawGraph();
925     try

```

```

926     {
927         int width = 1280;
928         int height = 650;
929         BufferedImage buffer = new
BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
930         Graphics g1= buffer.createGraphics();
931         g1.setColor(Color.WHITE);
932         g1.fillRect(0,0,width,height);
933         Graphics2D g2 = (Graphics2D)g1 ;
934         dg.plot(g2);
935         dg.plotXYCoordinates(g2);
936         dg.plotZCoordinates(g2);
937         dg.idex(g2);
938         dg.drawGraph(g2);
939         dg.plot(g2);
940         dg.drawOBJ(g2);
941         dg.drawSd(g2);
942         FileOutputStream os = new FileOutputStream(
MOD2DGREXP_CalculateValues.input_area_name+".jpg");
943         ImageIO.write(buffer, "jpg", os);
944         os.close();
945
946         String path = MOD2DGREXP_CalculateValues.input_area_name+".jpg";
947         image = ImageIO.read(new File(path));
948
949         Graphics g_image = MOD2DGREXP_MainPanel.img.getGraphics();
950         g_image.drawImage(image, -80, -40,image.getWidth(), image.getHeight(),dg);
951
952         MouseListener ml3 = new MouseAdapter(){
953             public void mouseClicked(MouseEvent e){
954                 Graphics g_image = MOD2DGREXP_MainPanel.img.getGraphics();
955                 g_image.drawImage(image, -80,-40,image.getWidth(),
image.getHeight(),dg);
956             }
957         };
958         MOD2DGREXP_MainPanel.img.addMouseListener(ml3);
959     }
960     catch (Exception e2) {
961
962         e2.printStackTrace();
963     }
964 }
965 }
966 -----
967 package com.mod2dgrexp.control;
968
969 import java.awt.event.ActionEvent;
970 import java.awt.event.ActionListener;
971 import java.io.File;
972 import java.io.IOException;
973
974 import javax.swing.JFrame;
975 import javax.swing.JOptionPane;
976 import com.mod2dgrexp.model.MOD2DGREXP_CalculateValues;
977 import com.mod2dgrexp.view.MOD2DGREXP_DrawGraph;
978 import com.mod2dgrexp.view.MOD2DGREXP_MainPanel;
979 import com.mod2dgrexp.view.MOD2DGREXP_TableView;
980
981 public class MOD2DGREXP_Controller implements ActionListener {
982
983     MOD2DGREXP_DrawGraph dg = new MOD2DGREXP_DrawGraph();
984     Object rowdata[][]={};
985     public static boolean success=false;
986     public void actionPerformed(ActionEvent ae) {
987
988         if(ae.getActionCommand().equals("Modeling")){
989
990             com.mod2dgrexp.model.MOD2DGREXP_CalculateValues cv = new
com.mod2dgrexp.model.MOD2DGREXP_CalculateValues();
991             cv.getAnamolyValues(com.mod2dgrexp.view.MOD2DGREXP_MainPanel.captureValues());
992             com.mod2dgrexp.view.MOD2DGREXP_TableView.populateEastPanel(MOD2DGREXP_Calculate
alues.obj);
993
994

```

```

995         com.mod2dgrexp.view.MOD2DGREXP_MainPanel.p_East.repaint();
996     }else if(ae.getActionCommand().equals("Save and Print")){
997         try
998         {
999             MOD2DGREXP_PrintValues.printGraphValues();
1000         }
1001         catch(Exception e1) {
1002             e1.printStackTrace();
1003         }
1004     }else if(ae.getActionCommand().equals("Load data")){
1005         try {
1006             MOD2DGREXP_MainPanel.loadData1();
1007         } catch (IOException e) {
1008             // TODO Auto-generated catch block
1009             e.printStackTrace();
1010         }
1011     }
1012     else if(ae.getActionCommand().equals("Clear")){
1013         MOD2DGREXP_MainPanel.clearDefaultValues();
1014         MOD2DGREXP_MainPanel.clearPanel(MOD2DGREXP_MainPanel.img);
1015         MOD2DGREXP_TableView.populateEastPanel(rowdata);
1016     }
1017     else if(ae.getActionCommand().equals("Exit")){
1018         JFrame frame = null;
1019
1020         int r = JOptionPane.showConfirmDialog(
1021             frame,
1022             "Exit MOD2DGREXP ?",
1023             "Confirm Exit ",
1024             JOptionPane.YES_NO_OPTION);
1025         if(r == JOptionPane.YES_OPTION ){
1026             if(success==false){
1027                 String fileName = MOD2DGREXP_CalculateValues.input_area_name+".jpg";
1028                 File f = new File(fileName);
1029                 f.delete();
1030             }
1031             System.exit(0);
1032         }
1033     }
1034 }
1035
1036 }
1037
1038 }
1039
-----
1040 package com.mod2dgrexp.control;
1041
1042 import java.io.File;
1043 import java.io.FileWriter;
1044 import java.text.DecimalFormat;
1045 import javax.swing.JFileChooser;
1046 import com.mod2dgrexp.model.MOD2DGREXP_CalculateValues;
1047
1048 public class MOD2DGREXP_PrintValues {
1049
1050     public static void printGraphValues() throws Exception {
1051
1052         try{
1053             String current = System.getProperty("user.dir");
1054             File img_file = new File(MOD2DGREXP_CalculateValues.input_area_name+".jpg");
1055             JFileChooser saveFile = new JFileChooser(current);
1056             File OutFile = saveFile.getSelectedFile();
1057             FileWriter myWriter = null;
1058             if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
1059             {
1060                 OutFile = saveFile.getSelectedFile();
1061                 if (OutFile.canWrite() || !OutFile.exists())
1062                 {
1063                     File dir = new File(OutFile.getParent());
1064                     MOD2DGREXP_Controller.success = img_file.renameTo(new
1065 File(dir,img_file.getName()));
1066                     myWriter = new FileWriter(OutFile+".html");
1067                     //myWriter.write("AUTOMATIC MODELING OF GRAVITY ANOMALIES OF 2D

```

```

SEDIMENTARY BASINS USING EXPONENTIAL DENSITY FUNCTION");
1068         myWriter.write(" </table> </td> <td> <img src = '"+
MOD2DGREXP_CalculateValues.input_area_name + ".jpg'></td></tr></table>");
1069         myWriter.write("<html> <Body onLoad = \"window.print()\"><table> <tr>
<td>\" +
1070             \"<table border = 1> <tr> <th colspan = 4>LOCATION:-
\"+MOD2DGREXP_CalculateValues.input_area_name+\"</th> </tr>");
1071             DecimalFormat df =new DecimalFormat("0.###");
1072             myWriter.write(" <tr><th colspan = 4> PROFILE NUMBER:-\"+
\"+MOD2DGREXP_CalculateValues.input_profile+\" </th></tr>");
1073             myWriter.write(" <tr><th colspan = 4> ITERATION:-\"+
\"+MOD2DGREXP_CalculateValues.o_iter+\" </th></tr>");
1074             myWriter.write("<tr > <th>Distance (km) </th> <th> Observed anomalies
(mGal) </th> <th> Calculated anomalies (mGal) </th> <th> Depth (km)
</th></tr>");
1075             for ( int K = 1; K <= MOD2DGREXP_CalculateValues.input_n_obs; K++){
1076                 myWriter.write("<tr> <td>\" +
MOD2DGREXP_CalculateValues.input_x_km[K]+\"</td>
<td>\"+df.format(MOD2DGREXP_CalculateValues.input_nob_gob[K])+\"</td>
<td>\"+df.format(MOD2DGREXP_CalculateValues.gc[K])+\"</td>
<td>\"+df.format(MOD2DGREXP_CalculateValues.z[K])+\"</td></tr>");
1077                 }
1078             myWriter.close();
1079         }
1080     }
1081     else
1082     {
1083         //pops up error message
1084     }
1085 }
1086 catch(Exception e1) {
1087     e1.printStackTrace();
1088 }
1089 }
1090 }
1091 }
1092 }
1093 -----
package com.mod2dgrexp.util;
1094
1095 import com.mod2dgrexp.model.MOD2DGREXP_CalculateValues;
1096
1097 public class MOD2DGREXP_Utility {
1098
1099     public static double convertDouble(String str) throws Exception {
1100
1101         Double temp = new Double(str.trim());
1102         return temp.doubleValue();
1103     }
1104
1105     public static String convertString(String str) throws Exception {
1106
1107         String temp = new String(str.trim());
1108         return temp;
1109     }
1110
1111     public static int convertInteger(String str) throws Exception {
1112
1113         Integer temp = new Integer(str.trim());
1114         return temp.intValue();
1115     }
1116
1117     public static int findMaximumNumber( double observe[]) {
1118
1119         double max = 0.0d;
1120         for (int i = 0; i < observe.length; i++) {
1121
1122             if (Math.abs(observe[i]) > Math.abs(max)) {
1123
1124                 max = observe[i];
1125             }
1126         }
1127
1128         int maxVal = (int) max/3*5;
1129

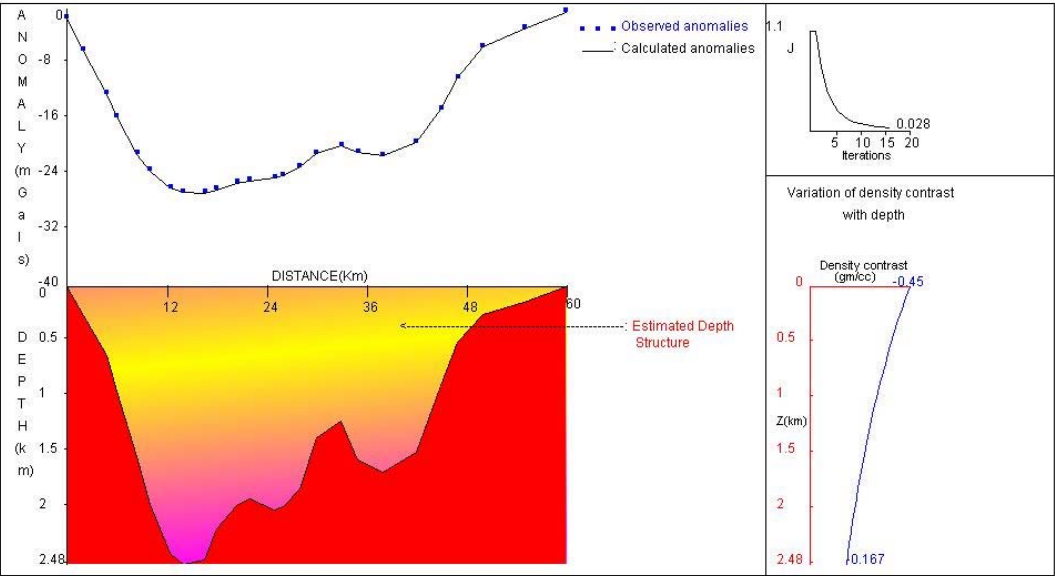
```

```

1130         return maxVal;
1131     }
1132
1133     public static int findMaximumNumber( double observe) {
1134
1135         double max = 0.0d;
1136         int maxVal=0;
1137         max = observe;
1138
1139         if (max < 5) {
1140
1141             maxVal = 5;
1142         }
1143         else if (max >= 5 && max <= 10) {
1144
1145             maxVal = 10;
1146         }
1147         else if ( max > 10 && max <= 15) {
1148
1149             maxVal = 15;
1150         }
1151         else if (max > 15 && max <= 20) {
1152
1153             maxVal = 20;
1154         }
1155         else
1156         {
1157             maxVal = MOD2DGREXP_CalculateValues.o_iter;
1158         }
1159
1160         return maxVal;
1161     }
1162
1163     public static double findMaximumNumber1( double observe[]) {
1164
1165         double max = 0.0d;
1166         for (int i = 1; i < observe.length; i++) {
1167
1168             if (Math.abs(observe[i]) > Math.abs(max)) {
1169
1170                 max =Math.abs(observe[i]);
1171             }
1172         }
1173
1174         double maxVal = max;
1175         return maxVal;
1176     }
1177
1178     public static double[] convertDoubleArray(String str) throws Exception {
1179
1180         java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
1181         String temp = "";
1182         java.util.ArrayList arr = new java.util.ArrayList();
1183
1184         while(st.hasMoreTokens()) {
1185
1186             temp = st.nextToken();
1187             arr.add(temp);
1188         }
1189         double d_array[] = new double[arr.size() + 1];
1190
1191         for(int i = 0; i <= arr.size(); i++) {
1192
1193             if (i == 0)
1194                 d_array[i] = 0.0;
1195             else
1196                 d_array[i] = convertDouble( arr.get(i-1).toString() );
1197         }
1198         return d_array;
1199     }
1200 }
1201
1202
1203

```





LOCATION:- Modeling - Output			
PROFILE NUMBER:- 1			
ITERATION:- 15			
Distance (km)	Observed anamolies (mGal)	Calculated anamolies (mGal)	Depth (km)
0.0	-1.504	-1.497	0
2.0	-6.136	-6.133	0.261
4.8	-12.331	-12.336	0.619
6.0	-15.6	-15.591	0.935
8.5	-20.99	-21.028	1.52
10.0	-23.431	-23.416	1.947
12.5	-25.885	-25.87	2.386
14.0	-26.549	-26.543	2.483
16.6	-26.634	-26.581	2.448
18.0	-26.099	-26.141	2.173
20.5	-25.193	-25.21	1.957
22.0	-24.82	-24.844	1.906
25.0	-24.412	-24.395	1.999
26.0	-24.088	-24.077	1.972
28.0	-22.908	-22.86	1.81
30.0	-20.922	-20.959	1.354
33.0	-19.76	-19.798	1.206
35.0	-20.819	-20.776	1.557
38.0	-21.265	-21.277	1.659
42.0	-19.33	-19.317	1.486
45.0	-14.505	-14.52	0.886
47.0	-10.018	-10.016	0.498
50.0	-5.558	-5.557	0.25
55.0	-2.927	-2.927	0.139
60.0	-0.499	-0.496	0

```

1 package com.in2dgrexp.view;
2
3 import java.awt.*;
4 import java.awt.event.WindowAdapter;
5 import java.awt.event.WindowEvent;
6 import java.io.File;
7 import javax.swing.JFrame;
8 import javax.swing.JOptionPane;
9 import com.in2dgrexp.control.IN2DGREXP_Controller;
10 import com.in2dgrexp.model.IN2DGREXP_CalculateValues;
11
12
13 public class IN2DGREXP_MainView extends Frame {
14
15     /**
16      *
17      */
18     private static final long serialVersionUID = 1L;
19
20     public static void main(String s[])
21     {
22         IN2DGREXP_MainView cm = new IN2DGREXP_MainView();
23         cm.setSize(1280, 768);
24         cm.addWindowListener(new WindowAdapter(){
25             public void windowClosing(WindowEvent e){
26                 JFrame frame = null;
27                 int r = JOptionPane.showConfirmDialog(
28                     frame,
29                     "Exit IN2DGREXP ?",
30                     "Confirm Exit ",
31                     JOptionPane.YES_NO_OPTION);
32                 if(r == JOptionPane.YES_OPTION ){
33                     if(IN2DGREXP_Controller.success==false){
34                         String fileName = IN2DGREXP_CalculateValues.input_area_name+".jpg";
35                         File f = new File(fileName);
36                         f.delete();
37                     }
38                     System.exit(0);
39                 }
40             }
41         });
42         cm.setTitle("IN2DGREXP");
43         cm.setResizable(false);
44         cm.add(new IN2DGREXP_MainPanel());
45         cm.setVisible(true);
46     }
47 }
48
49 -----
50

```

```

51 package com.in2dgrexp.view;
52
53 import java.awt.*;
54 import java.io.File;
55 import java.io.IOException;
56 import java.util.HashMap;
57 import javax.swing.JFileChooser;
58
59 import jxl.Cell;
60 import jxl.CellType;
61 import jxl.Sheet;
62 import jxl.Workbook;
63 import jxl.read.biff.BiffException;
64
65
66
67 public class IN2DGREXP_MainPanel extends Panel {
68
69     /**
70      *
71      */
72     private static final long serialVersionUID = 1L;
73
74

```

```

75     Panel p_North, p_West, p_South;
76     public static TextArea img = new TextArea(36,135);
77     public static Panel p_East, p_Center;
78     static TextField inputValues [] = new TextField[12];
79     static TextArea graphValues = new TextArea(38,30);
80     Button actionButton[] = new Button[5];
81     Object rowdata[][]={};
82
83     /**Field Area Name*/
84     final static int AREA_FE = 0;
85     /**Profile Name*/
86     final static int PROFILE_NAME = 1;
87     /**Number of observation*/
88     public static final int N_OBS = 2 ;
89     /**Distance(km)*/
90     public static final int X_KM = 3;
91     /**Number of Observed values*/
92     public static final int NOB_GOB = 4;
93     /** SD */
94     public static final int SD_POLY = 5;
95     /**LAMBDA*/
96     public static final int LAMBDA_ST = 6 ;
97     /**ZMIN(km)*/
98     public static final int DEP_ZMIN = 7;
99     /**ZMAX(km)*/
100    public static final int DEP_ZMAX = 8;
101    /**Number of iteration values*/
102    public static final int NOB_ITER = 9;
103
104
105    public IN2DGREXP_MainPanel() {
106
107        this.setLayout(new BorderLayout());
108        p_North = new Panel();
109        p_West = new Panel();
110        p_East = new Panel ();
111        p_South = new Panel();
112        p_Center = new Panel();
113
114        Label graphLabel = new Label("INVERSION OF GRAVITY ANOMALIES OF 2D SEDIMENTARY
BASINS ", Label.CENTER);
115        Label graphLabel1 = new Label(" USING EXPONENTIAL DENSITY MODEL ", Label.CENTER);
116        graphLabel.setFont(new Font("Arial", 10, 18));
117        graphLabel1.setFont(new Font("Arial", 10, 18));
118        p_Center.add(graphLabel);
119        p_Center.add(graphLabel1);
120
121        for(int i = 0; i < 11; i++){
122            inputValues[i] = new TextField();
123        }
124        p_North.setFont(new Font("Bold",1,12));
125        actionButton[0] = new Button("Load data");
126        actionButton[1] = new Button("Inversion");
127        actionButton[2] = new Button("Save and Print");
128        actionButton[3] = new Button("Clear");
129        actionButton[4] = new Button("Exit");
130
131        this.populateNorthPanel();
132        IN2DGREXP_TableView.populateEastPanel(rowdata);
133        this.add(p_North, BorderLayout.NORTH);
134        this.add(p_West, BorderLayout.WEST);
135        this.add(p_East, BorderLayout.EAST);
136        this.add(p_South, BorderLayout.SOUTH);
137        p_Center.setSize(1000, 760);
138        this.add(p_Center, BorderLayout.CENTER);
139        img.setEditable(false);
140        p_Center.add(img);
141        this.setVisible(true);
142    }
143
144    public void populateNorthPanel(){
145        p_North.setLayout(new GridLayout(5,6));
146
147        p_North.add(new Label("Area Name"));
148        p_North.add(inputValues[0]);

```

```

149         p_North.add(new Label("Profile Name"));
150         p_North.add(inputValues[1]);
151         p_North.add(new Label("Number of observation:"));
152         p_North.add(inputValues[2]);
153         p_North.add(new Label("Distance (km)"));
154         p_North.add(inputValues[3]);
155         p_North.add(new Label("Observed anomalies (mGal)"));
156         p_North.add(inputValues[4]);
157         p_North.add(new Label("Surface density contrast (gm/cc)"));
158         p_North.add(inputValues[5]);
159         p_North.add(new Label("Lambda (1/km)"));
160         p_North.add(inputValues[6]);
161         p_North.add(new Label("Minimum depth(km):"));
162         p_North.add(inputValues[7]);
163         p_North.add(new Label("Maximum depth(km):"));
164         p_North.add(inputValues[8]);
165         p_North.add(new Label("Iterations"));
166         p_North.add(inputValues[9]);
167         p_North.add(new Label(""));
168         p_North.add(new Label(""));
169         p_North.add(new Label(""));
170         p_North.add(new Label(""));
171
172         p_North.add(actionButton[0]);
173         p_North.add(actionButton[1]);
174         p_North.add(actionButton[2]);
175         p_North.add(actionButton[3]);
176         p_North.add(actionButton[4]);
177
178         actionButton[0].addActionListener(new com.in2dgrexp.control.IN2DGREXP_Controller());
179         actionButton[1].addActionListener(new com.in2dgrexp.control.IN2DGREXP_Controller());
180         actionButton[2].addActionListener(new com.in2dgrexp.control.IN2DGREXP_Controller());
181         actionButton[3].addActionListener(new com.in2dgrexp.control.IN2DGREXP_Controller());
182         actionButton[4].addActionListener(new com.in2dgrexp.control.IN2DGREXP_Controller());
183
184     }
185
186     public static HashMap captureValues(){
187
188         HashMap h_Map = new HashMap();
189
190         try {
191
192             h_Map.put("N_OBS", inputValues[N_OBS].getText());
193             h_Map.put("SD_POLY", inputValues[SD_POLY].getText());
194             h_Map.put("LAMBDA_ST", inputValues[LAMBDA_ST].getText());
195             h_Map.put("DEP_ZMIN", inputValues[DEP_ZMIN].getText());
196             h_Map.put("DEP_ZMAX", inputValues[DEP_ZMAX].getText());
197             h_Map.put("X_KM", inputValues[X_KM].getText());
198             h_Map.put("NOB_GOB", inputValues[NOB_GOB].getText());
199             h_Map.put("NOB_ITER", inputValues[NOB_ITER].getText());
200             h_Map.put("AREA_FE", inputValues[AREA_FE].getText());
201             h_Map.put("PROFILE_NAME", inputValues[PROFILE_NAME].getText());
202         }
203         catch (Exception e) {
204             e.printStackTrace();
205         }
206
207         return h_Map;
208     }
209
210
211     public static void clearPanel(TextArea p) {
212
213         Graphics g = p.getGraphics();
214         g.setColor(Color.WHITE);
215         g.fillRect(0, 0, 1280, 650);
216     }
217
218     public static void loadData1()throws IOException {
219         try{
220
221             String current = System.getProperty("user.dir");
222             JFileChooser chooser=new JFileChooser(current);
223             int returnVal = chooser.showOpenDialog(null);

```

```

224         String dis[], gobs[];
225         String disval = "", gobsval = "";
226         Workbook w;
227
228         if (returnVal == JFileChooser.APPROVE_OPTION) {
229             File f = chooser.getSelectedFile();
230             w = Workbook.getWorkbook(f);
231             Sheet sheet = w.getSheet(0);
232             dis = new String[sheet.getRows()+1];
233             gobs = new String[sheet.getRows()+1];
234             for (int j = 0; j < sheet.getColumns(); j++) {
235                 for (int i = 1; i < sheet.getRows(); i++) {
236                     Cell cell = sheet.getCell(j, i);
237                     CellType type = cell.getType();
238                     if (type == CellType.LABEL) {
239                         // System.out.println("I got a label "
240                         // + cell.getContents());
241                         IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.AREA_FE].set
Text(cell.getContents());
242
243                     }
244
245                     if (type == CellType.NUMBER) {
246                         if (j == 1){
247
248                             IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.PROFILEI
NAME].setText(cell.getContents());
249
250                         }
251                         if (j == 2){
252
253                             IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.N_OBS].
setText(cell.getContents());
254
255                         }
256
257                         if (j == 3){
258
259                             dis[i] = cell.getContents()+" ";
260                             disval = disval + dis[i];
261                         }
262
263                         if (j == 4){
264
265                             gobs[i] = cell.getContents()+" ";
266                             gobsval = gobsval+gobs[i];
267                         }
268
269                         if (j == 5){
270
271                             IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.SD_POLI
].setText(cell.getContents());
272
273                         }
274                         if (j == 6){
275
276                             IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.LAMBDA_
T].setText(cell.getContents());
277
278                         }
279
280                         if (j == 7){
281
282                             IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.DEP_ZM:
].setText(cell.getContents());
283
284                         }
285                         if (j == 8){
286
287
288                             IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.DEP_ZM:
].setText(cell.getContents());
289
290                         }
291                         if (j == 9){
292                             IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.NOBI_TI

```

```

292     ].setText(cell.getContents());
293     }
294
295     //System.out.println("I got a number "
296     // + cell.getContents());
297 }
298
299     }
300 }
301     IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.X_KM].setText(" "+disval
;
302     IN2DGREXP_MainPanel.inputValues[IN2DGREXP_MainPanel.NOB_GOB].setText(" "+gob
val);
303     }
304 }
305 catch (BiffException e) {
306     e.printStackTrace();
307 }
308
309
310
311 }
312
313
314 public static void clearDefaultValues() {
315
316     inputValues[N_OBS].setText("");
317     inputValues[SD_POLY].setText("");
318     inputValues[LAMBDA_ST].setText("");
319     inputValues[DEP_ZMIN].setText("");
320     inputValues[DEP_ZMAX].setText("");
321     inputValues[X_KM].setText("");
322     inputValues[NOB_GOB].setText("");
323     inputValues[NOB_ITER].setText("");
324     inputValues[AREA_FE].setText("");
325     inputValues[PROFILE_NAME].setText("");
326 }
327
328 }
329 -----
330 package com.in2dgrexp.view;
331
332 import java.applet.Applet;
333 import java.awt.*;
334 import java.awt.Font;
335 import java.awt.geom.Line2D;
336 import java.awt.geom.Rectangle2D;
337 import java.text.DecimalFormat;
338 import com.in2dgrexp.model.IN2DGREXP_CalculateValues;
339 import com.in2dgrexp.util.IN2DGREXP_Utility;
340
341
342 public class IN2DGREXP_DrawGraph extends Applet {
343     private static final long serialVersionUID = 1L;
344     float maxX,maxY;
345     float maxZ;
346     double inidep;
347     public void drawGraph(Graphics2D g){
348
349         g.setColor(Color.black);
350         g.drawLine(150,50,150,550);
351         g.drawLine(90,45 ,1040,45);
352
353         g.drawLine(780, 45, 780, 560);
354         g.drawLine(90, 560, 1040, 560);
355         g.drawLine(1040, 45, 1040, 560);
356         g.drawLine(90, 45, 90, 560);
357
358         g.drawString("DISTANCE(Km)",315 ,295);
359         String a[]={ "A", "N", "O", "M", "A", "L", "Y", "(m", "G", "a", "l", "s)"};
360         String b[]={ "D", "E", "P", "T", "H", "(k", "m)"};
361         for(int i=0;i<a.length;i++){
362

```

```

363         g.drawString(""+a[i],100 ,60+(i*20));
364     }
365     for(int i=0;i<b.length;i++){
366
367         g.drawString(""+b[i],100 ,350+(i*20));
368     }
369 }
370
371
372 public void plot(Graphics2D g){
373
374     maxX = (float)
IN2DGREXP_Utility.findMaximumNumber1(IN2DGREXP_CalculateValues.input_x_km);
//i_no_obs;
375     maxY = IN2DGREXP_Utility.findMaximumNumber(IN2DGREXP_CalculateValues.input_nob_gob);
376     inidep = IN2DGREXP_Utility.findMaximumNumber1(IN2DGREXP_CalculateValues.z);
377     maxZ = (float) inidep;
378     DecimalFormat df = new DecimalFormat("0.#");
379     DecimalFormat df1= new DecimalFormat("0.##");
380     g.drawString("0",140,60);
381     g.drawString(""+(int)maxY ,125,300);
382     g.drawString("0", 125,310);
383     g.drawString("|", 600, 308);
384     g.drawString(""+df.format(IN2DGREXP_CalculateValues.input_x_km[IN2DGREXP_CalculateV
lues.input_n_obs]), 600,320);
385     g.draw(new Line2D.Float(150, 300,600,300));
386
387     float points = maxX/5;
388     int yInterval=50;
389     int zInterval=50;
390     float xInterval=(float)
(IN2DGREXP_CalculateValues.input_x_km[IN2DGREXP_CalculateValues.input_n_obs]/5);
391     float xplot=0;
392     for (float x = xInterval, j =1; x < 600; x+=xInterval){
393
394         xplot=xplot+xInterval;
395         if(j>4)
396             break;
397         g.drawString("|",(float) (150+(450*x/maxX)), 308);
398         g.drawString(""+ df.format(xplot), (float) (150+(450*x/maxX))-3, 323);
399         j++;
400     }
401     points = maxY/5;
402     for (int x = yInterval, j =1; x < 250; x+=yInterval){
403
404         g.drawString("-",148,50+x);
405         g.drawString(""+(int)(points*j), 125,50+x);
406         j++;
407     }
408     float point =maxZ/5 ;
409     for(int x = zInterval+250,j =1; x < 550; x+=zInterval){
410
411         if(j>4)
412             break;
413         g.drawString("-",148,50+x);
414         g.drawString(""+df.format(point*j),125,50+x);
415         j++;
416     }
417     g.drawString("-",148,552);
418     g.drawString(""+df1.format(maxZ), 125, 550);
419 }
420
421 public void plotXYCoordinates (Graphics2D g){
422
423     float prevx = 150;
424     float prevy =50;
425     float xpoint=0;
426     float ypoint=0;
427     float gypoint=0;
428
429     for (int k =1; k <= IN2DGREXP_CalculateValues.input_n_obs; k++){
430
431         xpoint = (float)(450 * IN2DGREXP_CalculateValues.input_x_km[k]/ maxX) ;
432         ypoint = (float)(250 * IN2DGREXP_CalculateValues.gc[k]/ maxY);
433         gypoint = (float)(250 * IN2DGREXP_CalculateValues.input_nob_gob[k]/ maxY);

```

```

434         g.setColor(Color.BLACK);
435         g.draw(new Line2D.Float(prevx, prevy, 150+xpoin, 50+ypoint));
436         g.setColor(Color.BLUE);
437         g.setFont(new Font("Arial", 20, 55));
438         g.drawString(".", 150+xpoin-6, 50+ypoint);
439
440         prevx = 150+xpoin;
441         prevy = 50+ypoint;
442     }
443 }
444
445 public void plotZCoordinates (Graphics2D g){
446
447     float prevx = 150;
448     float prevz = 300;
449     float xpoin=0;
450     float zpoin=0;
451
452     GradientPaint gradient = new GradientPaint(10, 10, Color.yellow, 30, 200,
453     Color.MAGENTA, true);
454     g.setPaint(gradient);
455     g.fill(new Rectangle2D.Float(151, 300, 450, 250));
456     for (int k = 1; k <= IN2DGREXP_CalculateValues.input_n_obs; k++){
457
458         xpoin = (float)(450 * IN2DGREXP_CalculateValues.input_x_km[k] / maxX);
459         zpoin = (float)(250 * IN2DGREXP_CalculateValues.z[k]/maxZ);
460         float vary=prevx;
461
462         g.setColor(Color.red);
463         if(prevz<=300+zpoin){
464             while(vary<= 150+xpoin){
465                 g.draw(new Line2D.Float(prevx, prevz, vary, 300+zpoin));
466                 vary = (float) (vary+0.001);
467             }
468             g.fill(new Rectangle2D.Float(prevx, 300+zpoin, (150+xpoin)-prevx,
469 250-zpoin));
470         }
471         else{
472             vary = prevz;
473             while( 300 + zpoin <= vary){
474                 g.draw(new Line2D.Float(prevx, prevz, 150+xpoin, vary));
475                 vary = (float) (vary-0.001);
476             }
477             g.fill(new Rectangle2D.Float(prevx, prevz, (150+xpoin)-prevx, 550-prevz));
478         }
479         g.setColor(Color.black);
480         g.draw(new Line2D.Float(prevx, prevz, 150+xpoin, 300+zpoin));
481
482         prevx = 150+xpoin;
483         prevz = 300+zpoin;
484     }
485     g.setColor(Color.white);
486     g.fill(new Rectangle2D.Float(151, 550, 450, 50));
487 }
488
489 public void drawOBJ(Graphics2D g2) {
490
491     g2.setColor(Color.BLACK);
492
493     g2.drawLine(820, 70, 820, 160);
494     g2.drawLine(820, 160, 910, 160);
495     g2.drawString("J", 800, 90);
496
497     double maxOb =
498     IN2DGREXP_Utility.findMaximumNumber1(IN2DGREXP_CalculateValues.o_func);
499     int ini = IN2DGREXP_Utility.findMaximumNumber(IN2DGREXP_CalculateValues.o_iter);
500     if(ini==5)
501         ini= ini+1;
502     int maxiter = ( ini / 3 * 5 ) * 2;
503     int point;
504     int xInterval = 22;

```



```

506         point = ( ( ini ) / 3 * 5 ) / 5;
507
508         for (int x = xInterval, j = 1; x < 90 ; x += xInterval) {
509
510             g2.drawString( " ", 821+x, 170);
511             g2.drawString( " " + (point*j), 820 + x-3, 175);
512             j++;
513         }
514
515         float prevx = 820;
516         float prevy = 70;
517         float xpoint = 0;
518         float ypoint = 0;
519
520         for (int i = 1; i <= IN2DGREXP_CalculateValues.o_iter; i++) {
521
522             xpoint = (float)( 250 * i /maxiter );
523             ypoint = 70 - (float) ( ( 90 * (IN2DGREXP_CalculateValues.o_funcnt[i]) / maxOb
);
524
525             if(i==IN2DGREXP_CalculateValues.o_iter){
526                 g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint-4, 90 + ypoint));
527             }
528             else {
529                 g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 90 + ypoint));
530             }
531             prevx = 820 + xpoint;
532             prevy = 90 + ypoint;
533
534         }
535         DecimalFormat d1= new DecimalFormat("0.###");
536         DecimalFormat d= new DecimalFormat("0.#");
537         g2.drawString( " "+d.format(IN2DGREXP_CalculateValues.o_funcnt[1]), 780, 70);
538         g2.drawString( "
+d1.format(IN2DGREXP_CalculateValues.o_funcnt[IN2DGREXP_CalculateValues.o_iter]),
820 + xpoint, 90 + ypoint);
539         g2.setFont(new Font("Arial", 40,11));
540         g2.drawString ( "Iterations",850,186);
541     }
542
543     public void drawSd(Graphics2D g2) {
544         g2.setColor(Color.black);
545         g2.drawLine(780, 200, 1040, 200);
546         g2.setColor(Color.red);
547         g2.setFont(new Font("Arial", 20, 12));
548         DecimalFormat d= new DecimalFormat("0.##");
549         DecimalFormat d2= new DecimalFormat("0.#");
550         DecimalFormat d1= new DecimalFormat("0.###");
551
552         g2.drawString( " "+d.format(inidep),790,550);
553         g2.drawString( "-",820,552);
554         g2.drawString( "0",807 ,300);
555         g2.drawLine(820, 300, 910, 300);
556         g2.draw(new Line2D.Float(820, 300, 820, 550));
557
558         double maxOb1 = IN2DGREXP_Utility.findMaximumNumber1(IN2DGREXP_CalculateValues.vsd
559         float points =maxZ/5 ;
560         int zInterval=50;
561         for(int x = zInterval+250,j =1; x < 550; x+=zInterval){
562
563             if(j>4)
564                 break;
565             g2.drawString( "-",820,50+x+2);
566             g2.drawString( " "+d2.format(points*j),790,50+x);
567             j++;
568         }
569
570
571         float prevx = 820+ (float) ( ( 90 * ( Math.abs(IN2DGREXP_CalculateValues.vsd[1] ) )
maxOb1 ) );
572         float prevy = 300;
573         float xpoint = 0;
574         float ypoint = 0;
575
576         for (int i = 1; i <= IN2DGREXP_CalculateValues.count; i++) {

```

```

577
578         xpoint = (float)( 90 * Math.abs(IN2DGREXP_CalculateValues.vsd[i]) / maxOb1 );
579         ypoint = (float)( 250 * IN2DGREXP_CalculateValues.dep[i] / maxZ );
580
581         g2.setColor(Color.blue);
582         g2.draw(new Line2D.Float(prevx, prevy, 820 + xpoint, 300 + ypoint));
583
584         prevx = 820 + xpoint;
585         prevy = 300 + ypoint;
586     }
587
588     g2.drawString(""+d.format(IN2DGREXP_CalculateValues.vsd[1] ),805+(float) ( ( 90 *
Math.abs(IN2DGREXP_CalculateValues.vsd[1] ) ) / maxOb1 ) ,300 );
589     g2.drawString(""+d1.format(IN2DGREXP_CalculateValues.vsd[IN2DGREXP_CalculateValues.
ount] ),820+(float) ( ( 90 * (
Math.abs(IN2DGREXP_CalculateValues.vsd[IN2DGREXP_CalculateValues.count] ) ) / maxOb1
) ), 300+(float)( 250 * inidep / maxZ ) );
590     g2.setColor(Color.BLACK);
591     g2.drawString("Variation of density contrast " , 800,220);
592     g2.drawString("with depth" , 850,240);
593     g2.setFont(new Font("Arial", 40,11));
594     g2.drawString ( "Density contrast",830,285);
595     g2.drawString ( "(gm/cc)",843,295);
596     g2.drawString("Z(km)", 790,(float)( 300+((250*inidep/maxZ))/2));
597
598 }
599
600 /** Index of Graph*/
601 public void index(Graphics2D g){
602
603     g.setColor(Color.BLUE);
604     g.setFont(new Font("Arial", 20, 50));
605     g.drawString(" ... ",595,70);
606     g.setFont(new Font("Arial", 20, 12));
607     g.drawString("Observed anomalies",650,70);
608     g.setColor(Color.BLACK);
609     g.drawString("____:",615,87);
610     g.drawString("Calculated anomalies",650,90);
611     g.setColor(Color.black);
612     g.drawString("<-----:",450,340);
613     g.setColor(Color.RED);
614     g.drawString("Estimated Depth ",660,340);
615     g.drawString(" Structure",660,355);
616 }
617
618 }
619
620 -----
621
622 package com.in2dgrexp.view;
623
624 import java.awt.Dimension;
625
626 import javax.swing.JScrollPane;
627 import javax.swing.JTable;
628
629 public class IN2DGREXP_TableView {
630
631     public static void populateEastPanel(Object rowData[][]){
632         com.in2dgrexp.view.IN2DGREXP_MainPanel.p_East.removeAll();
633
634         Object columnNames[] = {"Distance(km)", "Observed anamolies (mGal)", "Calculated
635 anomalies (mGal)", "Depth(km)", "ERROR(mGal)"};
636
637         JTable table = new JTable(rowData, columnNames);
638
639         table.setPreferredScrollableViewportSize(new Dimension(300,550));
640
641         JScrollPane scrollPane = new JScrollPane(table);
642         scrollPane.setAutoscrolls(true);
643
644
645

```

```

646         com.in2dgrexp.view.IN2DGREXP_MainPanel.p_East.add(scrollPane);
647         com.in2dgrexp.view.IN2DGREXP_MainPanel.p_East.validate();
648
649         com.in2dgrexp.view.IN2DGREXP_MainPanel.p_East.setVisible(true);
650
651         //System.out.println(p_East.isVisible());
652     }
653
654 }
655 -----
656 package com.in2dgrexp.model;
657
658 import java.awt.Color;
659 import java.awt.Graphics;
660 import java.awt.Graphics2D;
661 import java.awt.event.MouseAdapter;
662 import java.awt.event.MouseEvent;
663 import java.awt.event.MouseListener;
664 import java.awt.image.BufferedImage;
665 import java.io.File;
666 import java.io.FileOutputStream;
667 import java.text.DecimalFormat;
668 import java.util.HashMap;
669
670 import javax.imageio.ImageIO;
671
672 import com.in2dgrexp.util.IN2DGREXP_Utility;
673 import com.in2dgrexp.view.IN2DGREXP_DrawGraph;
674 import com.in2dgrexp.view.IN2DGREXP_MainPanel;
675
676
677 public class IN2DGREXP_CalculateValues {
678
679     public static double []o_funcnt;
680     public static int o_iter,count;
681     public static int input_n_obs ;
682     public static double funct1,lambda,funct2;
683     public static double input_x_km[];
684     public static double input_nob_gob[] ;
685     public static double []gc;
686     public static double []z;
687     public static double []err;
688     public static double []vsd = null;
689     public static double []ldep = null;
690     public static String input_area_name = "";
691     public static String input_profile = "";
692     static BufferedImage image;
693     public static Object obj[][] = null;
694
695     public void getAnamolyValues(HashMap h_Map) {
696
697
698         double GK = 13.3333;
699         double PI = 22.0 / 7.0;
700         double input_sd_poly = 0;
701         double input_lambda_val = 0;
702         double input_zmin_km = 0;
703         double input_zmax_km = 0;
704         int input_nob_iter=0;
705
706         try {
707             input_n_obs = IN2DGREXP_Utility.convertInteger((String)h_Map.get("N_OBS"));
708             input_sd_poly = IN2DGREXP_Utility.convertDouble((String)h_Map.get("SD_POLY"));
709             input_lambda_val =
710 IN2DGREXP_Utility.convertDouble((String)h_Map.get("LAMBDA_ST"));
711             input_zmin_km = IN2DGREXP_Utility.convertDouble((String)h_Map.get("DEP_ZMIN"));
712             input_zmax_km = IN2DGREXP_Utility.convertDouble((String)h_Map.get("DEP_ZMAX"));
713             input_x_km = IN2DGREXP_Utility.convertDoubleArray((String)h_Map.get("X_KM"));
714             input_nob_gob =
715 IN2DGREXP_Utility.convertDoubleArray((String)h_Map.get("NOB_GOB"));
716             input_nob_iter =
717 IN2DGREXP_Utility.convertInteger((String)h_Map.get("NOB_ITER"));
718             input_area_name = IN2DGREXP_Utility.convertString((String)h_Map.get("AREA_FE"))

```

```

717         input_profile =
IN2DGREXP_Utility.convertString((String)h_Map.get("PROFILE_NAME"));
718     }
719     catch (Exception e) {
720         e.printStackTrace();
721     }
722
723     int np = input_n_obs - 2;
724     int np1 = np + 1;
725
726     double ALERR = 0.001*input_n_obs;
727     o_funct = new double[input_nob_iter+1];
728     gc = new double[input_n_obs +1];
729
730     double []b = new double[input_n_obs +1];
731     err = new double[input_n_obs +1];
732     double []par = new double[input_n_obs +2];
733     double []par1 = new double[input_n_obs +2];
734     double []par2 = new double[input_n_obs +2];
735
736     z = new double[input_n_obs +2];
737
738     double []g1 = new double[input_n_obs+1];
739     double []g2 = new double[input_n_obs+1];
740     //double err[]=new double[input_n_obs+1];
741     double [][]s = new double[input_n_obs+1][input_n_obs+1];
742     double []dupar = new double[input_n_obs+1];
743     double [][]p1 = new double[input_n_obs+1][input_n_obs+1];
744
745     double DPAR = 0.1;
746     int stn=0;
747
748     lambda = 0.5;
749
750     for (int K = 2; K <= input_n_obs - 1; K++) {
751
752         stn=stn+1;
753         if(input_lambda_val==0)
754             par[stn]= input_nob_gob[K]/(13.3333*PI*input_sd_poly);
755         else
756             par[stn] = -(1 / input_lambda_val) * Math.log(1 - ((input_lambda_val *
input_nob_gob[K]) / (GK * PI * input_sd_poly)));
757     }
758
759     GBSN(input_n_obs,input_x_km,par,input_sd_poly,input_lambda_val,GK,gc);
760     funct1 = 0.0;
761     for (int K = 1; K <= input_n_obs; K++) {
762
763         err[K] = input_nob_gob[K] - gc[K];
764         funct1 = funct1 + Math.pow(err[K], 2);
765     }
766     funct1=Math.sqrt(funct1/input_n_obs);
767
768     int iter ;
769
770     for (iter = 1; iter <= input_nob_iter; iter++) {
771
772         o_funct[iter]=funct1;
773         o_iter = iter;
774
775         z[1]=0.0001;
776         z[input_n_obs]=0.0001;
777         int ist =0;
778         for (int KK = 2; KK <= input_n_obs-1; KK++) {
779
780             ist =ist+1;
781             z[KK] = par[ist];
782
783         }
784
785         for (int k = 1; k <= np; k++) {
786             par1[k] =par[k];
787

```

```

789     }
790
791     for (int i = 1; i <= np; i++) {
792         par1[i] = par[i]+DPAR/2;
793         GBSN(input_n_obs,input_x_km,par1,input_sd_poly,input_lambda_val,GK,g1);
794         par1[i] = par[i]-DPAR/2;
795         GBSN(input_n_obs,input_x_km,par1,input_sd_poly,input_lambda_val,GK,g2);
796
797         for (int k = 1; k <= input_n_obs; k++) {
798
799             s[i][k] = (g1[k]-g2[k])/DPAR;
800
801         }
802     }
803     for (int j = 1; j <= np1; j++) {
804
805         for (int I = 1; I <= np; I++) {
806
807             p1[I][j] = 0;
808         }
809     }
810     for ( int J = 1; J <= np; J++) {
811
812         for ( int I = 1; I <= np; I++) {
813
814             for ( int K = 1; K <= input_n_obs; K++) {
815
816                 p1[I][J] = p1[I][J] + s[I][K] * s[J][K];
817
818             }
819         }
820     }
821
822     for ( int J = 1; J <= np; J++) {
823
824         for ( int K = 1; K <= input_n_obs; K++) {
825
826             p1[J][np1] = p1[J][np1] + err[K] * s[J][K];
827
828         }
829     }
830
831     do {
832
833         double CON  = (lambda + 1.0);
834
835         for(int i=1;i<=np;i++){
836             dupar[i] = par[i];
837         }
838         for (int L = 1; L <= np; L++) {
839
840             for (int J = 1; J <= np; J++) {
841
842                 if (L - J == 0) {
843
844                     p1[L][J] = p1[L][J] * CON;
845
846                 }
847             }
848         }
849
850         double KS [] = new double[2];
851
852         SIMEQ(p1,b,np,KS);
853         for (int I = 1; I <= np; I++) {
854
855             par2[I] = dupar[I] +0.5* b[I];
856
857             if (par2[I] <= input_zmin_km)
858                 par2[I] = input_zmin_km;
859
860             if (par2[I] > input_zmax_km)
861                 par2[I] = input_zmax_km;
862
863

```

```

864     }
865     GBSN(input_n_obs,input_x_km,par2,input_sd_poly,input_lambda_val,GK,gc);
866
867     funct2 = 0;
868
869     for (int K = 1; K <= input_n_obs; K++) {
870
871         err[K] = input_nob_gob[K] - gc[K];
872         funct2 = funct2 + Math.pow(err[K], 2);
873     }
874     funct2 = Math.sqrt(funct2/input_n_obs);
875
876     if (funct1 - funct2 < 0) {
877
878         if (lambda - 13.0 > 0) {
879
880             break;
881         }
882     }
883
884     if (funct1 - funct2 < 0) {
885
886         if (lambda - 13.0 <= 0) {
887
888             lambda = lambda * 2.0;
889
890             for (int I = 1; I <= np; I++) {
891
892                 for (int J = 1; J <= np; J++) {
893
894                     if (I - J == 0) {
895
896                         pl[I][J] = pl[I][J] / CON;
897
898                     }
899                 }
900             }
901         }
902     }
903
904     } while (funct1 <= funct2);
905
906     setGraphValues(input_n_obs,o_iter, input_x_km, z, input_nob_gob, gc, err,
907     funct1,lambda, input_area_name);
908     denCal(input_sd_poly,input_lambda_val);
909     drawGraph();
910     for(int I = 2; I <= input_n_obs-1; I++) {
911         z[I]=par[I-1];
912     }
913     funct1 = funct2;
914     lambda = (lambda / 2.0);
915     for(int I = 1; I <= input_n_obs; I++) {
916
917         par[I] = par2[I];
918
919     }
920
921     if (funct2 - ALERR <= 0)
922         break;
923
924 }
925
926 }
927
928
929 public static void denCal(double sd,double la){
930
931     int i = 1;
932     double z1= 0.0001;
933     double z2 = IN2DGREXP_Utility .findMaximumNumber1(IN2DGREXP_CalculateValues.z);
934     vsd = new double[(int) Math.pow(input_n_obs,2)];
935     dep = new double[(int) Math.pow(input_n_obs,2)];
936     while(z1 <= z2){
937

```

```

938         double dc = sd*Math.exp(-la*z1);
939         vsd[i] = dc;
940         dep[i] = z1;
941
942         z1 = z1+0.1;
943         i++;
944     }
945
946     count = i;
947     vsd[count] = sd*Math.exp(-la*z2);
948     dep[count] = z2;
949
950 }
951
952
953
954 public double[] GBSN(int n,double []x,double []zvv,double sd,double la,double gk,double
[]gc){
955
956     double ggc=0;
957     double []xx = new double [n+2];
958     double []zv = new double [n+2];
959     double []zt = new double[10000];
960     double []x1 = new double[10000];
961     double []gs = new double[10000];
962
963     zv[1] = 0.0001;
964     zv[n] = 0.0001;
965     int itn =0;
966     for(int jl =2;jl<=n-1;jl++){
967         itn = itn+1;
968         zv[jl] = zvv[itn];
969     }
970     for(int JJ = 1 ; JJ <= n ;JJ++){
971         gc[JJ]=0.0;
972     }
973     for(int k1=1;k1<=n;k1++){
974         for(int k2=1;k2<=n;k2++){
975             xx[k2]=x[k2]-x[k1];
976         }
977         xx[n+1]=xx[1];
978         zv[n+1]=zv[1];
979         double grav = 0;
980         for(int i=1;i<=n;i++){
981             double dxx = xx[i+1] - xx[i];
982             double dzv = zv[i+1] - zv[i];
983             double r = Math.sqrt(Math.pow(dxx,2)+Math.pow(dzv,2));
984             double c = dxx / r;
985             double s = dzv / r;
986             double ct = c / s;
987             double dx = (x[2]-x[1])/4;
988             double ZB = Math.abs(zv[i+1]-zv[i]);
989             int nd = (int)(ZB/dx)+1;
990             double n1 = nd / 2;
991             if (nd - ( 2 * n1 ) < 0 || nd - ( 2 * n1 ) > 0 ) {
992                 nd = nd + 1;
993             }
994             double DZ = ZB / nd;
995             int N2 = nd + 1;
996
997             if(zv[i+1]-zv[i]==0)
998                 break;
999             for(int JZ=1;JZ<=N2;JZ++){
1000                 if(zv[i+1]-zv[i]<0) {
1001                     zt[JZ] = zv[i] - DZ * (JZ-1);
1002                 }
1003                 if(zv[i+1]-zv[i]>0) {
1004                     zt[JZ] = zv[i] + DZ * (JZ-1);
1005                 }
1006
1007                 if(zt[JZ]<0)
1008                     zt[JZ] = 0;
1009                 x1[JZ] = xx[i]+(zt[JZ]-zv[i]) * ct;
1010                 if(Math.abs(x1[JZ])<0.01)
1011                     x1[JZ] = 0;

```

```

1012     }
1013
1014     for(int JZ=1;JZ<=N2;JZ++){
1015         double DC = (sd * Math.exp(-la * zt[JZ]));
1016         double a = xx[i] - zv[i] * ct;
1017         double anum = a + zt[JZ] * ct;
1018         gs[JZ] = -13.3333*DC*Math.atan(anum / zt[JZ]);
1019
1020     }
1021     ggc=SIMP(gs,zt,N2,ggc);
1022     grav = grav + ggc;
1023
1024 }
1025
1026 gc[k1] = grav;
1027
1028
1029 }
1030
1031 return gc;
1032 }
1033
1034 public double SIMP(double []gs,double []z,int n,double ggc) {
1035
1036     double dz = z[2]-z[1];
1037     double sum1 = 0.0;
1038     double sum2 = 0.0;
1039     int n1 = n / 2;
1040     int n4 = n1 - 1;
1041     for(int I = 1; I <= n1; I++) {
1042         int n2 = 2 * I;
1043         sum1 = sum1 + gs[n2];
1044     }
1045     for(int I = 1; I <= n4; I++) {
1046         int n3 = 2 * I +1;
1047         sum2 = sum2 + gs[n3];
1048     }
1049     ggc = gs[1]+4*sum1+2*sum2+gs[n];
1050     ggc = ggc * dz / 3.0;
1051     return ggc;
1052 }
1053 public static double []SIMEQ(double p[][], double b[], int n, double KS[]) {
1054
1055     int I = n + 1;
1056     double []a = new double[n*n+1];
1057
1058     for (int I1 = 1; I1 <= n; I1++) {
1059
1060         for (int I2 = 1; I2 <= n; I2++) {
1061
1062             int I3 = (I1 - 1) * n + I2;
1063             a[I3] = p[I2][I1];
1064
1065         }
1066     }
1067
1068     for (int I4 = 1;I4 <= n; I4++) {
1069
1070         b[I4] = p[I4][I];
1071
1072     }
1073     double TOL = 0;
1074
1075     KS[0] = 0;
1076     int JJ = - n;
1077     int IT;
1078     int NY = 0;
1079     for (int J = 1;J <= n; J++) {
1080
1081         int JY = J + 1;
1082         JJ = JJ + n + 1;
1083         double biga = 0;
1084
1085         IT = JJ - J;
1086         int imax = 0;

```



```

1087     for (int i = J; i <= n; i++) {
1088
1089         int IJ = IT + i;
1090         if (Math.abs(biga) - Math.abs(a[IJ]) < 0) {
1091
1092             biga = a[IJ];
1093             imax = i;
1094         }
1095     }
1096
1097     int I1 = 0;
1098
1099     if (Math.abs(biga) - TOL <= 0) {
1100
1101         KS[1] = 1;
1102         return KS;
1103     }
1104
1105     else {
1106
1107         I1 = J + n * (J - 2);
1108         IT = imax - J;
1109     }
1110
1111
1112
1113     double save;
1114     for (int K = J; K <= n; K++) {
1115
1116         I1 = I1 + n;
1117         int I2 = I1 + IT;
1118         save = a[I1];
1119         a[I1] = a[I2];
1120         a[I2] = save;
1121         a[I1] = a[I1] / biga;
1122     }
1123
1124     save = b[imax];
1125     b[imax] = b[J];
1126     b[J] = save / biga;
1127     int IQS = 0;
1128
1129     if (J - n < 0 || J - n > 0) {
1130
1131         IQS = n * (J - 1);
1132         for (int IX = JY; IX <= n; IX++) {
1133
1134             int IXJ = IQS + IX;
1135             IT = J - IX;
1136             for (int JX = JY; JX <= n; JX++) {
1137
1138                 int IXJX = n * (JX - 1) + IX;
1139                 int JJX = IXJX + IT;
1140
1141                 a[IXJX] = a[IXJX] - (a[IXJ] * a[JX]);
1142             }
1143
1144             b[IX] = b[IX] - (b[J] * a[IXJ]);
1145         }
1146     }
1147 }
1148
1149 NY = n - 1;
1150 IT = n * n;
1151
1152
1153 for (int J = 1; J <= NY; J++) {
1154
1155     int ia = IT - J;
1156     int ib = n - J;
1157     int ic = n;
1158     for (int K = 1; K <= J; K++) {
1159
1160         b[ib] = b[ib] - a[ia] * b[ic];
1161         ia = ia - n;

```

```

1162         ic = ic - 1;
1163     }
1164 }
1165 }
1166 return b;
1167 }
1168
1169 public static void setGraphValues(int i_no_obs, int ite, double []dis, double []dep,
double []GOBS, double []GCAL, double ERR[], double FUNCT, double lamb,String Area_fe) {
1170
1171     obj = new Object[i_no_obs+21][5];
1172
1173     DecimalFormat df =new DecimalFormat("0.###");
1174     DecimalFormat dl =new DecimalFormat("0.#####");
1175     for(int K=1;K<=i_no_obs;K++){
1176
1177         obj[K][0]= " " + dis[K];
1178         obj[K][1]= " " + df.format(GOBS[K]);
1179         obj[K][2]= " " + df.format(GCAL[K]);
1180         obj[K][3]= " " + df.format(dep[K]);
1181         obj[K][4]= " " + df.format(ERR[K]);
1182     }
1183
1184     obj[0][0] ="ITERATION";
1185     obj[0][1] = "=" + " "+ite;
1186
1187     obj[i_no_obs+2][0] = "OBJECTIVE " ;
1188     obj[i_no_obs+2][1] = "FUNCTION =";
1189     obj[i_no_obs+2][2] = dl.format(FUNCT);
1190
1191 }
1192
1193 public static void drawGraph(){
1194     final IN2DGREXP_DrawGraph dg = new IN2DGREXP_DrawGraph();
1195     try{
1196         int width = 1280;
1197         int height = 650;
1198         BufferedImage buffer = new
BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);
1199         Graphics gl= buffer.createGraphics();
1200         gl.setColor(Color.WHITE);
1201         gl.fillRect(0,0,width,height);
1202         Graphics2D g2 = (Graphics2D)gl ;
1203         dg.plot(g2);
1204         dg.plotXYCoordinates(g2);
1205         dg.plotZCoordinates(g2);
1206         dg.idex(g2);
1207         dg.drawGraph(g2);
1208         dg.plot(g2);
1209         dg.drawOBJ(g2);
1210         dg.drawSd(g2);
1211         FileOutputStream os = new FileOutputStream(
IN2DGREXP_CalculateValues.input_area_name+".jpg");
1212         ImageIO.write(buffer, "jpg", os);
1213         os.close();
1214
1215         String path = IN2DGREXP_CalculateValues.input_area_name+".jpg";
1216         image = ImageIO.read(new File(path));
1217
1218         Graphics g_image = IN2DGREXP_MainPanel.img.getGraphics();
1219         g_image.drawImage(image, -80, -40, image.getWidth(), image.getHeight(), dg);
1220         MouseListener ml3 = new MouseAdapter(){
1221             public void mouseClicked(MouseEvent e){
1222                 Graphics g_image = IN2DGREXP_MainPanel.img.getGraphics();
1223                 g_image.drawImage(image, -80,-40,image.getWidth(),
image.getHeight(),dg);
1224             }
1225         };
1226         IN2DGREXP_MainPanel.img.addMouseListener(ml3);
1227     }
1228     catch (Exception e2) {
1229
1230         e2.printStackTrace();
1231     }
1232 }

```

```

1233 }
1234 }
1235 -----
1236 package com.in2dgrexp.control;
1237
1238 import java.awt.event.*;
1239 import java.io.File;
1240 import java.io.FileWriter;
1241 import java.io.IOException;
1242 import java.text.DecimalFormat;
1243 import javax.swing.*;
1244 import com.in2dgrexp.model.IN2DGREXP_CalculateValues;
1245 import com.in2dgrexp.view.IN2DGREXP_DrawGraph;
1246 import com.in2dgrexp.view.IN2DGREXP_MainPanel;
1247
1248
1249 public class IN2DGREXP_Controller implements ActionListener {
1250     IN2DGREXP_DrawGraph dg = new IN2DGREXP_DrawGraph();
1251     public static boolean success=false;
1252     Object rowdata[][]={};
1253     public void actionPerformed(ActionEvent ae) {
1254
1255         if(ae.getActionCommand().equals("Inversion")){
1256
1257             com.in2dgrexp.model.IN2DGREXP_CalculateValues cv = new
1258 com.in2dgrexp.model.IN2DGREXP_CalculateValues();
1259             cv.getAnamolyValues(com.in2dgrexp.view.IN2DGREXP_MainPanel.captureValues());
1260             com.in2dgrexp.view.IN2DGREXP_TableView.populateEastPanel(IN2DGREXP_CalculateVal
es.obj);
1261             com.in2dgrexp.view.IN2DGREXP_MainPanel.p_East.repaint();
1262
1263         }else if(ae.getActionCommand().equals("Save and Print")){
1264             try {
1265                 String current = System.getProperty("user.dir");
1266                 File img_file = new File(IN2DGREXP_CalculateValues.input_area_name
+".jpg");
1267
1268                 JFileChooser saveFile = new JFileChooser(current);
1269                 File OutFile = saveFile.getSelectedFile();
1270                 FileWriter myWriter = null;
1271                 if(saveFile.showSaveDialog(null) == JFileChooser.APPROVE_OPTION)
1272                 {
1273                     OutFile = saveFile.getSelectedFile();
1274
1275                     if (OutFile.canWrite() || !OutFile.exists())
1276                     {
1277                         File dir = new File(OutFile.getParent());
1278                         success = img_file.renameTo(new File(dir,img_file.getName()));
1279                         myWriter = new FileWriter(OutFile+".html");
1280                         myWriter.write("</table> </td> <td> <img src = '"+
IN2DGREXP_CalculateValues.input_area_name
+".jpg'></td></tr></table>");
1281                         myWriter.write("<html> <Body onLoad = \"window.print()\"><table>
<tr> <td>\" +
1282
1283                             "<table border = 1> <tr> <th colspan = 4>LOCATION:-
"+IN2DGREXP_CalculateValues.input_area_name+"</th> </tr>");
1284
1285                             DecimalFormat df =new DecimalFormat("0.###");
1286                             myWriter.write("<tr><th colspan = 4> PROFILE NUMBER:-"+
"+IN2DGREXP_CalculateValues.input_profile+" </th></tr>");
1287                             myWriter.write("<tr><th colspan = 4> ITERATION:-"+
"+IN2DGREXP_CalculateValues.o_iter+" </th></tr>");
1288                             myWriter.write("<tr> <th>Distance (km) </th> <th> Observed
anomalies (mGal) </th> <th> Calculated anomalies (mGal) </th> <th>
Depth (km) </th><th> ERROR (mGal) </th></tr>");
1289                             for ( int K = 1; K <= IN2DGREXP_CalculateValues.input_n_obs; K++){
1290
1291                                 myWriter.write("<tr> <td>\" +
IN2DGREXP_CalculateValues.input_x_km[K]+ "</td>
<td>"+df.format(IN2DGREXP_CalculateValues.input_nob_gob[K])+"</t
d> <td>"+df.format(IN2DGREXP_CalculateValues.gc[K])+"</td>
<td>"+df.format(IN2DGREXP_CalculateValues.z[K])+"</td><td>"+df.f
ormat(IN2DGREXP_CalculateValues.err[K])+"</td></tr>");
1292                             }
1293                         }
1294                     }
1295                 }
1296             }
1297         }
1298     }
1299 }

```

```

1290
1291         }
1292         myWriter.close();
1293     }
1294 }
1295 catch (Exception e) {
1296
1297     e.printStackTrace();
1298 }
1299
1300 }
1301 else if(ae.getActionCommand().equals("Load data")){
1302     try {
1303         IN2DGREXP_MainPanel.loadData1();
1304     } catch (IOException e) {
1305         // TODO Auto-generated catch block
1306         e.printStackTrace();
1307     }
1308 }
1309 else if(ae.getActionCommand().equals("Clear")){
1310     IN2DGREXP_MainPanel.clearDefaultValues();
1311     IN2DGREXP_MainPanel.clearPanel(IN2DGREXP_MainPanel.img);
1312     com.in2dgrexp.view.IN2DGREXP_TableView.populateEastPanel(rowdata);
1313 }
1314
1315 else if(ae.getActionCommand().equals("Exit")){
1316     JFrame frame = null;
1317
1318     int r = JOptionPane.showConfirmDialog(
1319         frame,
1320         "Exit IN2DGREXP ?",
1321         "Confirm Exit ",
1322         JOptionPane.YES_NO_OPTION);
1323     if(r == JOptionPane.YES_OPTION ){
1324         if(success==false){
1325             String fileName = IN2DGREXP_CalculateValues.input_area_name+".jpg";
1326             File f = new File(fileName);
1327             f.delete();
1328         }
1329         System.exit(0);
1330     }
1331 }
1332
1333 }
1334
1335 }
1336
-----

```

```

1337 package com.in2dgrexp.util;
1338
1339 import com.in2dgrexp.model.IN2DGREXP_CalculateValues;
1340
1341 public class IN2DGREXP_Utility {
1342
1343     public static double convertDouble(String str) throws Exception {
1344
1345         Double temp = new Double(str.trim());
1346         return temp.doubleValue();
1347     }
1348
1349     public static String convertString(String str) throws Exception {
1350
1351         String temp = new String(str.trim());
1352         return temp;
1353     }
1354
1355     public static int convertInteger(String str) throws Exception {
1356
1357         Integer temp = new Integer(str.trim());
1358         return temp.intValue();
1359     }
1360
1361     public static int findMaximumNumber( double observe[]) {
1362
1363

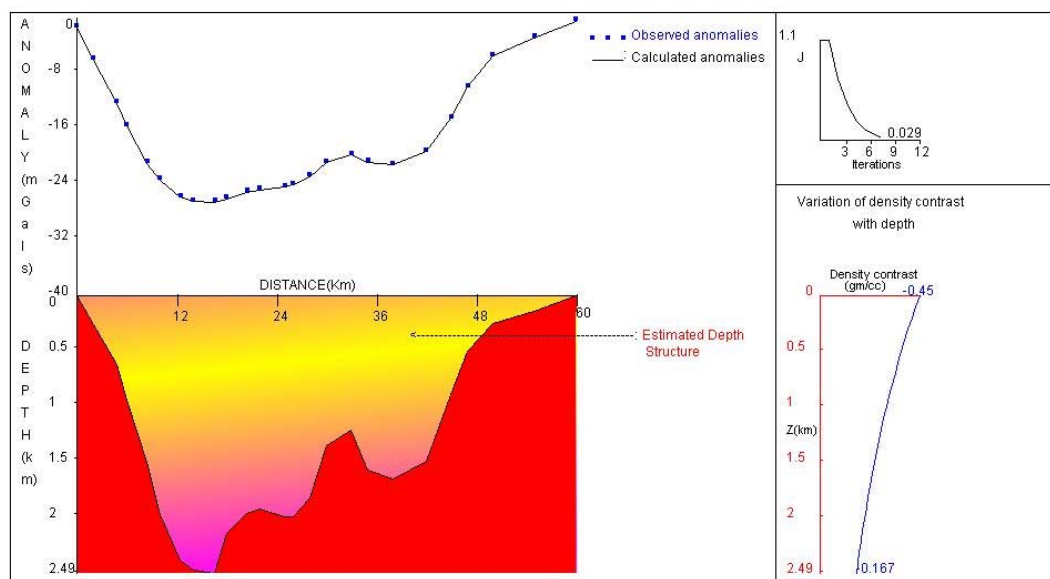
```

```

1364         double max = 0.0d;
1365         for (int i = 0; i < observe.length; i++) {
1366             if (Math.abs(observe[i]) > Math.abs(max)) {
1367                 max = observe[i];
1368             }
1369         }
1370     }
1371 }
1372
1373     int maxVal = (int) max/3*5;
1374     return maxVal;
1375 }
1376
1377 public static int findMaximumNumber( double observe) {
1378
1379     double max = 0.0d;
1380     int maxVal=0;
1381     max = observe;
1382
1383     if (max < 5) {
1384
1385         maxVal = 5;
1386     }
1387     else if (max >= 5 && max <= 10) {
1388
1389         maxVal = 10;
1390     }
1391     else if ( max > 10 && max <= 15) {
1392
1393         maxVal = 15;
1394     }
1395     else if (max > 15 && max <= 20) {
1396
1397         maxVal = 20;
1398     }
1399     else
1400     {
1401         maxVal = IN2DGREXP_CalculateValues.o_iter;
1402     }
1403     return maxVal;
1404 }
1405
1406 public static double findMaximumNumber1( double observe[]) {
1407
1408     double max = 0.0d;
1409     for (int i = 1; i < observe.length; i++) {
1410
1411         if (Math.abs(observe[i]) > Math.abs(max)) {
1412             max =Math.abs(observe[i]);
1413         }
1414     }
1415 }
1416
1417     double maxVal = max;
1418     return maxVal;
1419 }
1420
1421 public static double[] convertDoubleArray(String str) throws Exception {
1422
1423     java.util.StringTokenizer st = new java.util.StringTokenizer(str, ",");
1424     String temp = "";
1425     java.util.ArrayList arr = new java.util.ArrayList();
1426
1427     while(st.hasMoreTokens()) {
1428
1429         temp = st.nextToken();
1430         arr.add(temp);
1431     }
1432     double d_array[] = new double[arr.size() + 1];
1433
1434     for(int i = 0; i <= arr.size(); i++) {
1435
1436         if (i == 0)
1437             d_array[i] = 0.0;
1438         else

```

```
1439         d_array[i] = convertDouble( arr.get(i-1).toString() );
1440     }
1441     return d_array;
1442 }
1443 }
```



LOCATION:- Inversion-Output				
PROFILE NUMBER:- 1				
ITERATION:- 7				
Distance (km)	Observed anomalies (mGal)	Calculated anomalies (mGal)	Depth (km)	ERROR (mGal)
0.0	-1.504	-1.5	0	-0.004
2.0	-6.136	-6.154	0.264	0.018
4.8	-12.331	-12.347	0.621	0.016
6.0	-15.6	-15.602	0.943	0.002
8.5	-20.99	-20.985	1.5	-0.005
10.0	-23.431	-23.41	1.961	-0.021
12.5	-25.885	-25.856	2.366	-0.029
14.0	-26.549	-26.536	2.465	-0.013
16.6	-26.634	-26.592	2.485	-0.042
18.0	-26.099	-26.104	2.136	0.005
20.5	-25.193	-25.178	1.957	-0.015
22.0	-24.82	-24.821	1.909	0.001
25.0	-24.412	-24.39	1.981	-0.022
26.0	-24.088	-24.088	1.988	0
28.0	-22.908	-22.888	1.816	-0.02
30.0	-20.922	-20.933	1.349	0.011
33.0	-19.76	-19.774	1.203	0.014
35.0	-20.819	-20.804	1.569	-0.015
38.0	-21.265	-21.253	1.648	-0.012
42.0	-19.33	-19.31	1.484	-0.02
45.0	-14.505	-14.508	0.885	0.003
47.0	-10.018	-10.032	0.501	0.014
50.0	-5.558	-5.571	0.251	0.013
55.0	-2.927	-2.932	0.14	0.005
60.0	-0.499	-0.497	0	-0.002

## References

---

- Abdelrahman, E.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.M., Essa, K.S., Erabo-Ezz, 2013, A least-squares window curves method to interpret gravity data due to dipping faults: Journal of Geophysics and Engineering, 10, 025003.
- Abdelrahman, E.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, 535-543.
- Abdelrahman, E.M., Bayoumi, A.I., El-Araby, H.M., 1989, Dip angle determination of fault planes from gravity data: Pure and Applied Geophysics, 130, 735-742.
- Adema, G.W., Breckenridge, R.M., Sprenke, K.F., 2007, Gravity, morphology, and bedrock depth of the Rathdrum Prairie, Idaho, Technical Report, Idaho Geological Survey, University of Idaho.
- Agarwal, B. N. P., Shalivahan, S., 2010, A FORTRAN code to implement the finite element method to compute regional and residual anomalies from the gravity data: Computers & Geosciences, 36, 848-852.



- Agarwal, B. N. P., Sivagi, C., 1992, Separation of regional and residual anomalies by least squares orthogonal polynomial and relaxation techniques: a performance evaluation: *Geophysical Prospecting*, 40, 143-156.
- Agarwal, B. P., 1995, Hydrocarbon prospects of the Pranhita - Godavari graben, India: *Proceedings of Petrotech*, 95, 115-121.
- Albora, A.M., Ucan, O.N., Ozmen, A., 2001, Residual separation of magnetic fields using a cellular neural net-work approach, *Pure and Applied Geophysics*, 158, 1797-1818.
- Al-Garni, M.A., Srinivas, Y., and Sundararajan, N., 2010, Sundararajan transform - an application to geophysical data analysis: *Arabian Journal of Geosciences*, 3, 27-32.
- Ander, M.E., Huestis, S.P., 1987, Gravity ideal bodies: *Geophysics*, 52, 1265-1278.
- Angell, H.T., Negrinia, R.M., 1997, Gravity Model of The Southern San Joaquin Basin Constrained by Density Data from a 24,000 Foot Well, Elk Hills, California, AAPG Pacific Section Meeting, Bakersfield, California.
- Annechione, M. A., Chouteau, M., Keating, P., 2001, Gravity interpretation of bedrock topography: the case of the Oak Ridges Moraine, southern Ontario, Canada: *Journal of Applied Geophysics*, 47, 63-81.
- Artemjev, M.E., Kaban, M.K., Kucherinenko, V.A., Demyanov, G.V., Taranov, V.A., 1994, Subcrustal density inhomogeneities of Northern Eurasia as

- derived from the gravity data and isostatic models of the lithosphere: *Tectonophysics*, 240, 249–280.
- Athy, L.F., 1930, Density, porosity and compaction of sedimentary rocks: *Bulletin of the American Association of Petroleum Geology*, 14, 1-24.
- Aydemir, A., Ates, A., Bilim, F., Buyuksarac, A., Bektas, O., 2014, Evaluation of gravity and aeromagnetic anomalies for the deep structure and possibility of hydrocarbon potential of the region surrounding Lake Van, Eastern Anatolia, Turkey: *Surveys in Geophysics*, 35, 431–448
- Aydogan, D., 2011, Extraction of lineaments from gravity anomaly maps using the gradient calculation: Application to Central Anatolia: *Earth Planets Space*, 63, 903-913.
- Azeglio, E.A., Gimenez, M.E., Introcaso, A., 2010, Interpretation of Las Salinas sedimentary basin - Argentina, based on integration of geological and geophysical data: *Geofísica Internacional*, 49, 225-244.
- Baan, M.V.D., Jutten, C., 2000, Neural Networks in geophysics: *Geophysics*, 65, 1032-1047.
- Bachman, R.T., Hamilton, E.L., 1976, Density, Porosity, and Grain Density of Samples From Deep Sea Drilling Project Site 222 (LEG 23) in the Arabian Sea: *Journal of Sedimentary Petrology*, 46, 654-658.
- Backus, G.E., Gilbert, F.J., 1967, Numerical applications of a formalism for geophysical inverse problems: *Geophysical Journal of the Royal Astronomical Society*, 13, 247-276.

- Backus, G.E., Gilbert, F.J., 1968, The resolving power of gross earth data: Geophysical Journal of the Royal Astronomical Society, 16, 169-205.
- Backus, G.E., Gilbert, F.J., 1970, Uniqueness in the inversion of inaccurate gross earth data: Philosophical Transactions of the Royal of London, A226, 123-192.
- Barbosa, V.C., Silva, J.B.C., Medeiros, W., 1997, Gravity inversion of basement relief using approximate equality constraints on depths: Geophysics, 62, 1745-1757.
- Barbosa, V.C., Silva, J.B.C., Medeiros, W., 1999, Gravity inversion of a discontinuous relief stabilized by weighted smoothness constraints on depth: Geophysics, 64, 1429-1437.
- Becking, L.G.M.B., Moore, D., 1959, Density Distribution in Sediments: Journal of Sedimentary Petrology, 29, 47-55.
- Beiki, M., 2010, Analytic signals of gravity gradient tensor and their application to estimate source location: Geophysics, 75, I59–I74.
- Beiki, M., Pedersen, L., 2010, Eigenvector analysis of gravity gradient tensor to locate geologic bodies: Geophysics, 75, I37–I49.
- Beltrao, J.F., Silva, J.B.C., Costa, J.C., 1991, Robust polynomial fitting method for regional gravity estimation: Geophysics, 56, 80-89.
- Bhaskara Rao, V., Venkateswarulu, P.D., 1974, A simple method of interpreting gravity anomalies over sedimentary basins: Geophysical Research Bulletin, 12, 177-182

- Bhimasankaram, V.L.S., Nagendra, R., Seshagiri Rao, S.V., 1977, Interpretation of gravity anomalies due to finite inclined dikes using Fourier Transformation: *Geophysics*, 42, 51-59.
- Blakely, R.J., 1995, *Potential Theory in Gravity and Magnetic Applications*: Cambridge University Press, 464 pp.
- Bohidar, R.N., Sullivan, J.P., Hermance, J.F., 2001, Delineating depth to bedrock beneath shallow unconfined aquifers: a gravity transect across the Palmer river basin, *Groundwater*, 39, 729–736.
- Boschetti, F., Dentith, M., List, R., 1997, Inversion of potential field data by genetic algorithms: *Geophysical Prospecting*, 45, 461-478.
- Boschetti, F., Therond, V., Hornby, P., 2004, Feature removal and isolation in potential field data: *Geophysical Journal International*, 159, 833-841.
- Bott, M. H. P., 1960, The use of rapid digital computing methods for direct gravity interpretation of sedimentary basins: *Geophysical Journal of Royal Astronomical Society*, 3, 63-67.
- Brady, R., Wernicke, B., Fryxell, J., 2000, Kinematic evolution of a large-offset continental normal fault system, South Virgin Mountains, Nevada: *Geological Society of America Bulletin*, 112, 1375-1397.
- Brown, M.P., POULTON, M.M., 1996, Location of buried objects for environmental site investigations using Neural Networks: *Journal of Environmental & Engineering Geophysics*, 1, 179-188.

- Brun, J.P., Choukroune, P., 1983, Normal faulting, blocktilting, and décollement in a stretched crust: *Tectonics*, 2, 345-356.
- Byerly, P.E., 1965, Convolution filtering of gravity and magnetic maps: *Geophysics*, 30, 281-283.
- Cai, H., Zhdanov, M., 2015, Application of Cauchy-type integrals in developing effective methods for depth-to-basement inversion of gravity and gravity gradiometry data: *Geophysics*, 80, G81–G94.
- Castagna, J.P., Batzle, M.L., Kan, T.K., 1993, Rock physics—the link between rock properties and AVO response. In *Offset-Dependent Reflectivity—Theory and Practice of AVO Analysis*, ed. P. Castagna and M.M. Backus, No. 8, 124–157. Tulsa, Oklahoma: Investigations in Geophysics series, Society of Exploration Geophysicists.
- Chacko, S., Battacharya, B., 1980, A method for analyzing gravity anomalies due to a geologic contact by Fourier transform: *Geoexploration*, 18, 43-50.
- Chai, Y., Hinze, W.J., 1988, Gravity inversion of an interface above which the density contrast varies exponentially with depth: *Geophysics*, 53, 837-845.
- Chaika, C., Williams, L.A., 2001, Density and mineralogy variations as a function of porosity in Miocene Monterey Formation oil and gas reservoirs in California, *Bulletin of the American Association of Petroleum Geologists*, 85, 149-167.
- Chakraborty, C., Mandal, N., Ghosh, S.K., 2003, Kinematics of the Gondwana basins of peninsular India: *Tectonophysics*, 377, 299–324.

- Chakravarthi, V., Pramod Kumar, M., Ramamma, B., Rajeswara Sastry, S., 2015a, Gravity anomaly interpretation of 2D listric fault morphologies using exponential density contrast model: A space domain technique: Arabian Journal of Geosciences (under review).
- Chakravarthi, V., Pramod Kumar, M., 2015b, Estimation of multiple density-depth parameters from gravity inversion: Application to detached hanging wall systems of strike limited listric fault morphologies: Geophysica Internacional (Springer), 54, 49-65.
- Chakravarthi, V., Pramod Kumar, M., Ramamma, B., Rajeswara Sastry, S., 2015c, Automatic gravity modeling of sedimentary basins by means of polygonal source geometry and exponential density contrast variation: Two space domain based algorithms: Journal of Applied Geophysics (in press).
- Chakravarthi, V., Rajeswara Sastry, S., Pramod Kumar, M., 2014, 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, 83, 577-585.
- Chakravarthi, V., Rajeswara Sastry, S., Ramamma, B., 2013a, 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., Ramamma, B., 2013b, Gravity anomaly modelling of multiple geological sources having differing strike lengths and arbitrary density contrast variations: *Near Surface Geophysics*, 11, 363-370.
- Chakravarthi, V., Ramamma, B., Venkat Reddy, T., 2013c, Gravity Anomaly Modeling of Sedimentary Basins by Means of Multiple Structures and Exponential Density Contrast-depth Variations: A Space Domain Approach: *Journal of the Geological Society of India*, 82, 561-569.
- Chakravarthi, V., 2011a, Geodesy, Physical: *Encyclopedia of Solid Earth Geophysics* (Springer), 331-335.
- Chakravarthi, V., 2011b, Automatic gravity optimization of 2.5D strike listric fault sources with analytically defined fault planes and depth-dependent density: *Geophysics*, 76, I21-I31.
- Chakravarthi, V., 2010a, LSTRKFALTG — A forward modeling program to compute theoretical gravity anomalies of strike limited listric fault structures with prescribed vertical variation in density: *Computers & Geosciences*, 36, 675–679.
- 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, 279-286.
- Chakravarthi, V., 2009, Automatic gravity inversion for simultaneous estimation of model parameters and regional gravity background: an application to 2D pull-apart basins: *Current Science*, 96, 1349-1360.

- Chakravarthi. V., 2008, Gravity inversion of 2.5D faulted beds using depth dependent density, *Current Science*, 95, 1618-1622.
- Chakravarthi, V., Sundararajan, N., 2007, 3D gravity inversion of basement relief – A depth dependent density approach: *Geophysics*, 72, I23- I32.
- Chakravarthi, V., Sundararajan, N., 2006, Discussion on "The gravitational attraction of a right rectangular prism with density varying with depth following a cubic polynomial" by García-Abdeslem, J (November-December 2005, *Geophysics*, j39-j42), *Geophysics*, 71, X17–X19.
- Chakravarthi, V., Sundararajan, N., 2004, Ridge regression algorithm for gravity inversion of fault structures with variable density: *Geophysics*, 69, 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.B., AshokBabu, G., 2001, INVER2DBASE - A program to compute basement depths of density interfaces above which the density contrast varies with depth: *Computers & Geosciences*, 27, 1127–1133.
- Chapin, D. A., 1996, A deterministic approach toward isostatic gravity residuals; a case study from South America: *Geophysics*, 61, 1022 -1033.
- Chappell, A., Kusznir, N., 2008, An algorithm to calculate the gravity anomaly of sedimentary basins with exponential density-depth relationships: *Geophysical Prospecting*, 56, 249–258.



- Chávez, R.E., Lazaro-Mancilla, O., Campos-Enríquez, J.O., Flores-Márquez, E.L., 1999, Basement topography of the Mexicali Valley from spectral and ideal body analysis of gravity data: *Journal of South American Earth Sciences*, 12, 579–587.
- Christiansen, A.F., 1983, An example of a major syndepositional listric fault. In: Bally AW (ed) *Seismic Expression of Structural Styles: American Association of Petroleum Geologists Studies in Geology*, 15, 36-40.
- Collins, S.J., Dodds, A.R., Johnson, B.D., 1974, Gravity profile interpretation using Fourier transform: *Geophysics*, 39, 862-866.
- Constenius, K.N., 1996, Late Paleogene extensional collapse of the Cordilleran foreland fold and thrust belt: *Geological Society of America Bulletin*, 108, 20- 30.
- Cordell, L., 1973, Gravity anomalies using an exponential density-depth function- San Jacinto Graben, California: *Geophysics*, 38, 684-690.
- Corner, B., Wilsher, W.A., 1989, Structure of the Witwatersrand basin derived from interpretation of the aeromagnetic and gravity data, in *Proceedings of exploration '87, Third decennial international conference on geophysical and geochemical exploration for minerals and groundwater*, Ontario: Geological Survey, 3, 532–546.
- Cowie, P.A., Karner, G.D., 1990, Gravity effect of sediment compaction: examples from the North Sea and Rhine Graben: *Earth and Planetary science Letters*, 99, 141-153.

- Dallmus, K.F., 1958, Mechanics of basin evolution and its relation to the habitat of oil in the basin: In *Habitat of Oil*, L.G. Weeks, No. 36, 2071–2174. Tulsa, Oklahoma: AAPG Memoir, American Association of Petroleum Geologists.
- De Vicente, J., Lanchares, J., Hermida, R., 2003, Placement by thermodynamic simulated annealing: *Physics Letters*, A317, 415-423.
- Dickinson, G., 1953, Geological Aspects of Abnormal Reservoir Pressures in Gulf Coast Louisiana: *Bulletin-American Association of Petroleum Geologists*, 37, 410-432.
- Dimitropoulos, K., Donato, J.A., 1981, The inner moray firth central ridge, a geophysical interpretation: *Scottish Journal of Geology*, 17, 27–38.
- Donato, J.A., Tully, M.C., 1981, A regional interpretation of North Sea gravity data, in *Petroleum Geology of the Continental Shelf of North-West Europe*: Eds. L.V. Illing and G.D. Hobson, 65–75.
- Dorigo, M., Blum, C., 2005, Ant colony optimization theory: a survey: *Theoretical Computational Science*, 344, 243-278.
- D’Urso, M.G., 2014a, Gravity effects of polyhedral bodies with linearly varying density: *Celestial Mechanics and Dynamical Astronomy*, <http://library.seg.org/doi/abs/10.1190/1.1440973>
- D’Urso, M.G., 2014b, Analytical computation of gravity effects for polyhedral bodies: *Journal of Geodesy*, 88, 13–29.

- Eaton, B.A., 1969, Fracture gradient Prediction and Its Application in Oilfield Operations: Journal of Petroleum Technology and Alternative Fuels, 21,1353–1360.
- Eberhart, R.C., Kennedy, J., 1995, A new optimizer using particle swarm theory: Proceedings of sixth International symposium on micro machine and human science, IEEE service centre, Piscataway, Nagoya, Japan, 39-43.
- Emeleus, C.H., 1981, A thrust fault at Welshman's Rock, Isle of Rhum, Scottish: Journal of Geology, 17, 1-6.
- Essa, K.S., 2013, Gravity interpretation of dipping faults using the variance analysis method: Journal of Geophysics and Engineering, 10: 015003.
- Fairhead, J.D., Bennett, K.J., Gordon, R.H., Huang, D., 1994, Euler: Beyond the 'BlackBox', 64<sup>th</sup> Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 422–424.
- Fett, J.D., 1968, Geophysical investigation of the San Jacinto Valley, Riverside County, California: Unpublished thesis, University of California at Riverside, 87 pp.
- Forsythe, G.E., 1957, Generation and use of orthogonal polynomials for data filtering with a digital computer: Journal Society Indian Applied Mathematics, 5, 75-88.
- Foucher, J.P., LePichon, X., Sibuet, J.C., 1982, The ocean-continent transition in the uniform stretching model: role of partial melting in the mantle: Philosophical Transactions of the Royal Society A, 305, 1489, 27–41.

- Gabalda, G., Nalpas, T., Bonvalot, S., 2005, The Base of the Atacama Gravels Formation (26°S, Northern Chile): first results from gravity data, 6<sup>th</sup> International Symposium on Andean Geodynamics (ISAG2005, Barcelona). IRD, Paris, 286–289 (Extended Abstracts).
- Ganguly, S.S., Dimri, V.P., 2013, Interpretation of gravity data using eigenimage with Indian case study: A SVD approach: *Journal of Applied Geophysics*, 95, 23-35.
- Gans, P.B., Miller, E.L., McCarthy, J., Ouldcott, M.L., 1985, Tertiary extensional faulting and evolving ductile-brittle transition zones in the northern Snake Range and vicinity: New insights from seismic data, *Geology*, 13, 189-193.
- García-Abdeslem, J., 2003, 2D modeling and inversion of gravity data using density contrast varying with depth and source-basement geometry described by the Fourier series: *Geophysics*, 68, 1909-1916.
- Gealy, E., 1969, Saturated bulk density, grain density, and porosity of sediment cores from the western equatorial Pacific, *Glomar Challenger, Initial Report of the Deep Sea Drilling Project*, VII, part 2.
- Geldart, L.P., Gill, D.E., Sharma, B., 1966, Gravity anomalies of two dimensional faults: *Geophysics*, 31, 372-397.
- Gibbs, A.D., 1983, Balanced cross-section construction from seismic sections in areas of extensional tectonics: *Journal Structural Geology*, 5, 153- 160.

- 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, Calgary, Canada, Expanded Abstracts, 435-439.
- Grandis, H., Dahrin, D., 2014, Constrained two-dimensional inversion of gravity data: Journal of mathematical and fundamental sciences, 46, 1-13.
- Granser, H., 1987, Three-dimensional interpretation of gravity data from sedimentary basins using an exponential density-depth function: Geophysical Prospecting, 35, 1030-1041.
- Grant, F.S., 1957, A problem in the analysis of geophysical data: Geophysics, 22, 309-344.
- Gu, X., Tenzer, R., Gladkikh, V., 2014, Empirical models of the ocean-sediment and marine sediment-bedrock density contrasts: Geosciences Journal (in press), DOI 10.1007/s12303-014-0015-9.
- Guang, Z., Hou, J., Huang, L., Yao, C., 1998, Inversion of gravity and magnetic anomalies using pseudo-BP neural network method and its application: Chinese Journal of Geophysics, 41, 139-149.
- Gupta, D.K., Gupta, J.P., Arora, Y., Singh, U.K., 2011, Recursive Ant Colony Global Optimization: a new technique for the inversion of geophysical data: American Geophysical Union, Fall Meeting, abstract #H13B-1196.
- Gupta, O.P., Pokhriyal, S.K., 1990, New formula for determining the dip angle of a fault from gravity data: SEG Technical Program Expanded Abstracts 9646-9.

- Gupta, O.P., 1983, A least-squares approach to depth determination from gravity data: *Geophysics*, 48 (3), 357-360
- Guspi, F., 1990, General 2D inversion with density contrast varying with depth: *Geoexploration*, 26, 253-265.
- Hamayun, Prutkin, I., Tenzer, 2009, The optimum expression for the gravitational potential of polyhedral bodies having a linearly varying density distribution: *Journal of Geodesy*, **83**, 1163-1170
- Hamilton, E.L., Menard, H.W., 1956, Density and porosity of seafloor sediments off San Diego, California: *Bulletin of the American Association of Petroleum Geologists*, 40, 754-761.
- Hamilton, E.L., 1976, Variations of density and porosity with depth in deep-sea sediments: *Journal of Sedimentary Petrology*, 46, 1-2, 280–300.
- Hansen, R.O., 1999, An analytical expression for the gravity field of a polyhedral body with linearly varying density: *Geophysics*, 64, 75–77.
- Helmberger, D.V., Engen, G., Scott, P., 1979, A note on velocity, density, and attenuation models for marine sediments determined from multi-bounce phases: *Journal of Geophysical Research*, 84, 667–671.
- Hinze, W.J., Aiken, C., Brozena, J., Coakley, B., Dater, D., Flanagan, G., Forsberg, R., Hildenbrand, T., Keller, G. R., Kellogg, J., Kucks, R., Li, X., Mainville, A., Morin, R., Pilkington, M., Plouff, D., Ravat, D., Roman, D., Urrutia-Fucugauchi, J., Veronneau, M., Webring, M., Winster, D., 2005,

- New standards for reducing gravity data: The North American gravity database: *Geophysics*, 70, J25-J32.
- Hood, P., 1965, Gradient measurements in aeromagnetic surveying, *Geophysics*, 30, 891–802.
- Holliger, K., Klemperer, S.L., 1989, A comparison of the Moho interpreted from gravity data and from deep seismic reflection data in the northern North Sea, *Geophysical Journal International*, 97, 247–258.
- Hu, C., Fan, M., Wu, Y., Pei, Y., 2011, Euler deconvolution method of gravity anomaly based on wavelet analysis: International Conference on Multimedia Technology (ICMT 2011), Hangzhou, China, 305-308.
- Huang, D., Gubbins, D., Clark, R.A., Whaler, K.A., 1995, Combined study of Euler's homogeneity equation for gravity and magnetic field, 57<sup>th</sup> Conference & Technical Exhibition: European Association of Exploration Geophysicists, Extended Abstracts, 144.
- Huestis, S.P., Parker, R.L., 1977, Bounding the thickness of the oceanic magnetized layer: *Journal of Geophysical Research*, 82, 5293–5303.
- Inoue, N., Kitada, N., Takemura, K., 2012, Blind fault Configuration in Osaka, Japan based on 2D gravity inversion, 15 WCEE conference ([http://www.iitk.ac.in/nicee/wcee/article/WCEE2012\\_2615.pdf](http://www.iitk.ac.in/nicee/wcee/article/WCEE2012_2615.pdf)).
- Jackson, J.A., 1987, Active normal faulting and crustal extension. *Geol. Soc. London Spec. Publ.*, 28, 3–17.

- Jackson, J.A. McKenzie, D., 1983, The geometrical evolution of normal fault systems: *Jour. Struct. Geol.*, 5, 471-482.
- Jacobsen, B.H., 1987, A case for upward continuation as a standard separation filter for potential field maps: *Geophysics*, 52, 1138-1148.
- Jacoby, W., Smilde, P.L., 2009, Gravity interpretation—fundamentals and application of gravity inversion and geological interpretation: Springer, Berlin.
- Janecke, S.U., Vandenburg, C.J. Blankenau, J.J., 1998, Geometry, mechanisms and significance of extensional folds from examples in the Rocky Mountain Basin and Range province, U.S.A., *Jour. Struct. Geol.*, 20, 841-856.
- Jingxin, Q., Ziqi, G., Jianying, L., Yanchao, Q., 2013, Research in Modified Simulated Annealing Interface Inversion Method: Near Surface Geophysics Asia Pacific Conference, Beijing, China, 657-659.
- Kadirov, F.A., 2000, Application of the Hartley transform for interpretation of gravity anomalies in the Shamakhy–Gobustan and Absheron oil-and gas-bearing regions, Azerbaijan: *Journal of Applied Geophysics*, 45, 49-61.
- Kaila, K.L., Murthy, P.R.K., Rao, V.K., Venkateswarlu, N., 1990, Deep seismic sounding in the Godavari Graben and Godavari (Coastal) Basin, India: *Tectonophysics*, 173, 307–317.
- Kearey, P., Brooks, M., Healy I., 2002, An introduction to Geophysical exploration, third ed. Blackwell Science.



- Kennedy, J., Eberhart, R., 1995, Particle swarm optimization, IEEE International Conference on Neural Networks, 4, 1942–1948.
- Klinge, E.E., Marson, I., Kahle, H.G., 1991, Automatic interpretation of gravity gradiometric data in two dimensions: Vertical gradients, Geophysical Prospecting, 39, 407–434.
- Krahenbuhl, R.A., Li, Y.G., 2006, Inversion of gravity data using a binary formulation: Geophysical Journal International, 167, 543–556.
- Krimmel, R.M., 1970, Gravimetric ice thickness determination, South Cascade Glacier, Washington: Northwest Science, 44, 147–153.
- Kwok, Y.K., 1991, Contour integrals for gravity computation of horizontal  $2^{1/2}$ -D bodies with variable density. Applied mathematical modeling 15, 98-103.
- LaFehr, T.R., Nabighian, M.N., 2012, Fundamentals of gravity exploration, Society of Exploration Geophysicists, USA.
- Lavin, P.M., Devane, J.F., 1970, Direct design of two-dimensional digital wave number filters: Geophysics, 35, 1073-1078.
- Leão, J.W.D., Menezes, P.T.L., Beltrão, J.F., Silva, J.B.C., 1996, Gravity inversion of basement relief constrained by the knowledge of depth at isolated points: Geophysics, 61, 1702–1714.
- Leite, E.P., Filho, C.R., de. S., 2009, Artificial neural networks applied to mineral potential mapping for copper-gold mineralizations in the Carajás Mineral Province, Brazil: Geophysical Prospecting 57, 1049-1065.

- Levy, A.V., Montalvo, A., 1985, The Tunneling algorithm for the global minimization of functions: SIAM Journal on Scientific and Statistical Computing, 6, 15–29.
- Li, Y., Oldenburg, D.W., 1998, 3-D gravity inversion of gravity data: Geophysics, 63, 109-119.
- Mallick, K., Sharma, K.K., 1999, A finite element method for computation of the regional gravity anomaly: Geophysics, 64, 461-469.
- Manger, G.E., 1963, Porosity and bulk density of sedimentary rocks, U.S. Geological Survey Bulletin, 1144-E, E1-E56.
- Mantlík, F., Matias, M., Lourenço, J., Grangeia, C., Tareco, H., 2009, The use of gravity methods in the internal characterization of landfills – a case study: Journal of Geophysics and Engineering, 6, 357–364.
- Mareschal, J.C., 1985, Inversion of potential field data in Fourier transform domain: Geophysics, 50, 685-691.
- Marlet, G., Sailhac, P., Moreau, F., Diament, M., 2001, Characterization of geological boundaries using 1-D wavelet transform on gravity data; theory and application to the Himalayas: Geophysics, 66, 1116-1129.
- Marson, I., Klingele, E.E., 1993, Advantages of using the vertical gradient of gravity for 3-D interpretation: Geophysics, 58, 1588–1595.
- Marquardt, D.W., 1963, An algorithm for least squares estimation of nonlinear parameters: Journal Society Indian Applied Mathematics, 11, 431-441.

- Marquardt, D.W., 1970, Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation: *Technometrics*, 12, 591-612.
- Martín Atienza, B., García-Abdeslem, J., 1999, 2-D gravity modeling with analytically defined geometry and quadratic polynomial density functions: *Geophysics*, 64, 1730–1734.
- Martinez, P., Gimenez, M., Folguera, A., Klinger, F.L., 2014, Integrated seismic and gravimetric model of Jocolí Basin, Argentina: *Interpretation*, 2, T57–T68.
- Maxant, J., 1980, Variation of density with rock type, depth, and formation in the Western Canada basin from density logs: *Geophysics*, 45, 1061-1076.
- McCulloh, T.H. 1967, Mass properties of sedimentary rocks and gravimetric effects of petroleum and natural-gas reservoirs. USGS Professional Paper 528-A, Department of the Interior, United States Geological Survey, Washington, DC.
- McGrath, P.H., 1991, Dip and depth extent of density boundaries using horizontal derivatives of upward-continued gravity data: *Geophysics*, 56, 1533-1542.
- McKenzie, D., Jackson, J., 2012, Tsunami earthquake generation by the release of gravitational potential energy: *Earth and Planetary Science Letters*, 345-348, 1-8.
- McNeill, L.C., Piper, K.A., Goldfinger, C., Kulm, L.D., Yeats, R.S., 1997, Listric normal faulting on the Cascadia continental margin: *Journal of Geophysical Research* 102: 123-12,138.

- Mendonca, C.A., 2004, Inversion of gravity-field inclination to map the basement relief of sedimentary basins: *Geophysics*, 69, 1240-1251.
- Merriam, D.F., Cocke, N.C., 1967, Colloquium on trend analysis: Kansas State Geological survey Comput. Contr., 12, P.62.
- Mickus, K.L., Aiken, C.L.V., Kennedy, W.D., 1991, Regional-residual gravity anomaly separation using minimum curvature technique: *Geophysics*, 56, 279-283.
- Mickus, K.L., Peeples, W.J., 1992, Inversion of gravity and magnetic data for the lower surface of a two and one-half dimensional sedimentary basin: *Geophysical Prospecting*, 40, 171-193.
- Mishra, D.C., Gupta, S.B., Rao, M.B.S.V., Venkatarayudu, M., Laxman, G., 1987, Godavaribasin - a geophysical study: *Journal of the Geological Society of India*, 30, 469–476.
- Mohan, N.L., 1978, Interpretation techniques in geophysical exploration using Fourier Transforms: Ph. D thesis, Osmania University, Hyderabad, India (unpublished).
- Mohan, C.R., Sastry, R.G.S., Moharir, P.S., 1986, Inversion of gravity anomalies using Tunneling algorithm: In twelfth annual convention and seminar on Exploration Geophysics, A7-A8.
- Moharir, P.S., 1990, Inversion of potential field data, *Journal of Earth System Science*, 99, 473-514.
- Moritz, H., 1980, *Advanced Physical Geodesy*: Karlsruhe:Wichmann.

- Mosegaard, K., Tarantola, A., 1995, Monte Carlo sampling of solutions to inverse problems: *Journal of Geophysical Research*, 100, 12431-12447.
- Mundim, K.C., Lemaire, T.J., Bassrei, A., 1998, Optimization of non-linear gravity models through generalized simulated annealing, *Physica A*, 252, 405–416.
- Murty, B.V.R., Parthasarathy, E.V.R., 1988, On the evolution of the Godavari Gondwana Graben based on LANDSAT Imagery interpretation, *Journal of the Geological Society of India*, 32, 417–425.
- Murthy, I.V.R., 1998, Gravity and Magnetic Interpretation in Exploration Geophysics: Memoir 40, Geological Society of India.
- Murthy, I.V.R., Krishnamacharyulu, S.K.G., 1990, Automatic inversion of gravity anomalies of faults: *Computers & Geosciences*, 16, 539-548.
- Murthy, I.V.R., Rao, P.R., Rao, S.J., 1990, The density difference and generalized programs for two and three-dimensional gravity modeling: *Computers & Geosciences*, 16, 277-287.
- Murthy, I.V.R., Rao, S.J., 1989, A Fortran 77 program for inverting gravity anomalies of two-dimensional basement structures: *Computers & Geosciences*, 15, 1149-1156.
- Murthy, I.V.R., Krishna, P.R., Rao, S.J., 1988, A generalized computer program for two-dimensional gravity modeling of bodies with a flat top or a flat

- bottom or undulating over a mean depth: *Journal of Association of Exploration Geophysicists*, 9, 93-103.
- Murthy, I.V.R., Rao, D.B., 1980, Interpretation of gravity anomalies over faults and dykes by Fourier transforms, employing end correction: *Geophysical Research Bulletin*, 18, 95-110.
- Murthy, I.V.R., Rao, D.B., 1979, Gravity anomalies of two-dimensional bodies of irregular cross-section with density contrast varying with depth: *Geophysics*, 44, 1525-1530.
- Mushayandebvu, M.F., Driel, P.V., Reid, A.B., Fairhead, J.D., 2001, Magnetic source parameters of two-dimensional structures using extended Euler deconvolution: *Geophysics*, 66, 814–823.
- Nagihara, S., Hall, S.A., 2001, Three-dimensional gravity inversion using simulated annealing, constraints on the diapiric roots of allochthonous salt structures: *Geophysics*, 66, 1438-1449.
- Naidu, P.S., 1966, Extraction of a potential field signal from a background of random noise by Strakhov's method: *Journal of Geophysical Research*, 71, 5987-5995.
- Naidu, P.S., 1967, Two-dimensional Strakhov's filter for extraction of a potential field signal: *Geophysical Prospecting*, 15, 135-150.
- Naudy, H., Dreyer, H., 1968, Essai de filtrage non-lineaire appliqué aux profils aeromagnetiques: *Geophysical Prospecting*, 16, 171-178.

- Odegard, M.E., 2011, A sediment thickness map of South America using automated inversion of magnetic and gravity data for depth to basement: 12<sup>th</sup> International Congress of the Brazilian Geophysical Society and EXPOGEF, SEG and Brazilian Geophysical Society, 581–586.
- Odegard, M.E., Berg, J.W., 1965, Gravity interpretation using the Fourier integral: *Geophysics*, 30(3), 424-438.
- Okabe, M., 1979, Analytical expressions for gravity anomalies due to homogeneous polyhedral bodies and translations into magnetic anomalies: *Geophysics*, 44, 730-741.
- Oliva, S.E. Ravazzoli, C.L., 1997, Complex polynomials for the computation of 2D gravity anomalies: *Geophysical Prospecting* 45, 809–818.
- Oruç, B., 2014, Structural interpretation of south part of western Anatolian using analytic signal of the second order gravity gradients and discrete wavelet transform analysis: *Journal of Applied Geophysics* (In Press, Accepted Manuscript), Available online 24 January 2014.
- Osman, O., Albora, A.M., Ucan, O. N., 2006, A new approach for residual gravity anomaly profile interpretations: Forced Neural Network (FNN): *Annals of Geophysics*, 49, 1201-1208.
- Osman, O., Albora, A.M., Ucan, O.N., 2007, Forward modeling with forced neural networks for gravity anomaly profile: *Mathematical Geology*, 39, 593–605.

- Pacino, M.C., Introcaso, A., 1988, Regional anomaly determination using the upwards – continuation method: *Bollettino di Geofisica Teorica e Applicata*, XXIX (114), 113- 122.
- Pal, P.C., 1981, Gravity profile interpretation using Fourier transform: *Geoexploration*, 19, 167-177.
- Pallero, J.L.G., Fernández-Martínez, J.L., Bonvalot, S., Fudym, O., 2015, Gravity inversion and uncertainty assessment of basement relief via Particle Swarm Optimization: *Pure and Applied Geophysics*, 116, 180–191.
- Pan J.J., 1989. Gravity anomalies of irregularly shaped two-dimensional bodies with constant horizontal density gradient: *Geophysics* 54, 528-530.
- Parker, R.L., 1974, Best bounds on density and depth from gravity data: *Geophysics*, 39, 644-649.
- Parker, R.L., 1975, The theory of ideal bodies for gravity interpretation: *The Geophysical journal of the Royal Astronomical Society*, 42, 315–334.
- Paul, M.K., Datta S., Banerjee, B., 1966, Direct interpretation of two dimensional structural fault from gravity data: *Geophysics*, 31, 940-948.
- Pawlowski, R.S., 1994, Green's equivalent-layer concept in gravity band-pass filter design: *Geophysics*, 59, 69-76.
- Pawlowski, R.S., Hansen, R.O., 1990, Gravity anomaly separation by Wiener filtering: *Geophysics*, 55, 539–548.
- Peirce, J. W., Lipkov, L., 1988, Structural interpretation of the Rukwa Rift, Tanzania: *Geophysics*, 53, 824-836.



- Pilkington M., Crossley D.J., 1986, Determination of crustal interface topography from potential fields: *Geophysics* 51, 1277–1284.
- Prozorovich, E.A., 1960, Density and Porosity of Clayey Rocks Under Differing Conditions of Moisture, *Petroleum Geology: A Digest of Russian Literature*, 4, 2B, 98-104.
- Qureshy, M.N., KrishnaBrahmam, N., Garde, S.C., BMathur, .K., 1968, Gravity anomalies and the Godavaririft, India: *Geological Society of America*, 79, 1221–1230.
- Ramakrishna, T.S., Chayanulu, A.Y.S.R., 1988, A geophysical appraisal of the Purana basins of India: *Journal of the Geological Society of India*, 32, 48–60.
- Rao, B.S.R., Murthy, I.V.R., 1978, *Gravity and Magnetic methods of prospecting: Arnold-Heinemann publishers (India) Pvt. Ltd.*, 390 pp.
- Rao, D.B., 2013, Gravity anomalies of two-dimensional bodies: *Journal of Indian Geophysical Union*, 17, 129-137.
- Rao, D.B., 1990, Analysis of gravity anomalies of sedimentary basins by an asymmetrical trapezoidal model with quadratic density function: *Geophysics*, 55, 226-231.
- Rao, D.B., 1986, Modeling of sedimentary basins from gravity anomalies with variable density contrast: *Geophysical Journal of the Royal Astronomical Society*, 84, 207-212.

- Rao, D.B., 1985 Analysis of gravity anomalies over an inclined fault with quadratic density function: *Pure and Applied Geophysics*, 123, 250-260.
- Rao, D.B., Prakash, M.J., 1990, Interpretation of gravity anomalies over an inclined fault of finite strike length with quadratic density function: *Australian Journal of Exploration Geophysics*, 21, 169-173.
- Rao, D.B., Prakash, M.J., Babu, N.R., 1993, Gravity interpretation using Fourier transforms and simple geometrical models with exponential density contrast: *Geophysics*, 58, 1074-1083.
- Rao, P.R., Murthy, I.V.R., 1989. Two fortran77 function subprograms to calculate gravity anomalies of bodies of finite and infinite strike length with the density contrast differing with depth: *Computers and Geosciences* 15, 1265–1277.
- Rao, M.M.M., Murthy, T.V.R., Murthy, K.S.R. Vasudeva, R.Y., 2003, Application of natural generalized inversetechnique in reconstruction of gravity anomalies due to a fault: *Indian Jour. Pure Appld. Mathematics*, 34, 31-47.
- Reid, A.B., 1997, Euler Deconvolution, Past, Present and Future: A Review, *Proceedings of Exploration 97: Fourth Decennial International Conference on Mineral Exploration*, 861–864.
- Reid, A.B., Allsop, J.M., Granser, H., Millett, A.J., Somerton, I.W., 1990, Magnetic interpretation in three dimensions using Euler deconvolution: *Geophysics*, 55, 80–91.

- Reamer, S.K., Ferguson J.F., 1989, Regularized two-dimensional Fourier gravity inversion method with application to the Silent Canyon caldera, Nevada: *Geophysics*, 54, 486-496.
- Rieke, H.H., Chilingarian, G.V., 1974, *Compaction of Argillaceous Sediments*. Amsterdam, The Netherlands: Elsevier Scientific Publishing Company.
- Roy, A., 1962, Ambiguity in geophysical interpretation: *Geophysics*, 27, 90-99.
- Roy, L., Shaw, R., Agarwal, B.N.P., 2002, Inversion of gravity anomalies over sedimentary basins; applications of genetic algorithm and simulated annealing: SEG Annual Meeting Expanded Technical Program Abstracts with Biographies, 72, 747-750.
- Sax, R.L., 1966, Application of filter theory and information theory to the interpretation of gravity measurements: *Geophysics*, 31, 570-575.
- Shalivahan, S., Agarwal, B.N.P., 2010, Inversion of the amplitude of the analytic signal of the magnetic anomaly by particle swarm optimization technique: *Geophysical Journal International*, 182, 652-662.
- Sharma, B., Geldart, L.P., 1968, Analysis of gravity anomalies using Fourier transforms: *Geophysical Prospecting*, 16, 77-93.
- Shaw, R.K., Agarwal, B.N.P., 1990, The application of Walsh transforms to interpret gravity anomalies due to some simple geometrically shaped causative sources: A feasibility study: *Geophysics*, 55, 843-850.
- Sigl, R., 1985, *Introduction to Potential Theory*: Cambridge, Abacus Press.

- Silva, J.B.C., Oliveira, A.S., Barbosa, V.C.F., 2010, Gravity inversion of 2D basemen trelief using entropic regularization: *Geophysics*, 75, I29–I35.
- Silva, J.B.C., Medeiros, W.E., Barbosa, V.C.F., 2001, Potential-field inversion: Choosing the appropriate technique to solve a geologic problem: *Geophysics*, 66, 511–520.
- Silva Dias, F.J.S., Barbosa, V.C.F., Silva, J.B.C., 2007, 2D gravity inversion of a complex interface in the presence of interfering sources: *Geophysics*, 72, I13–I22.
- Singh, B., 2002, Simultaneous computation of gravity and magnetic anomalies resulting from a 2-D object: *Geophysics*, 67, 801–806.
- 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*, 89, 5733–5762.
- Spector, A., Grant, F.S., 1970, Statistical models for interpreting aeromagnetic data: *Geophysics*, 35, 293–302.
- Spitz, O.T., 1966, Generation of orthogonal polynomials for trend surfacing with a digital computer: *Proceedings of symposium on operations research*, In Mineral Ind., Pennsylvania State University, 3, 2–7.
- Srivastava, S., Datta, D., Agarwal, B.N.P., Mehta, S., 2013, Applications of Ant Colony Optimization in determination of source parameters from total

- gradient of potential fields: Near Surface Geophysics (Early Online), DOI: 10.3997/1873-0604.2013054.
- Stavrev, P., Reid, A., 2010, Euler deconvolution of gravity anomalies from thick contact/fault structures with extended negative structural index: Geophysics, 75, I51-I58.
- Stern, T.A., 1978, Gravity survey of the Taylor Glacier, Victoria Land, Antarctica: Technical Report 8, Geology Department, Victoria University of Wellington.
- Storer, D., 1959, Compaction of the argillaceous sediments in the Padano Basin: In The Gasiferous Deposits of Western Europe, 2, 519–536. Roma, Italy.
- Strakhov, V.N., 1964, The smoothing of observed strength of potential fields: Akad. Nauk. SSSR. Izv. Fizika Zemli, Part I, 897-904.
- Strakhov, V.N., Lapina, M.I., 1967, A method of smoothing potential fields: Akad. Nauk. SSSR. Izv. Fizika Zemli, 40-57.
- Sundararajan, N., Brahmam, G.R., 1998, Spectral analysis of gravity anomalies caused by slab-like structures: A Harley transform technique: Journal of Applied Geophysics, 39, 53-61.
- Sundararajan, N., Srinivas, Y., Rao, T.L., 2000, Sundararajan Transform - a tool to interpret potential field anomalies: Exploration Geophysics, 31, 622 – 628.
- Sundararajan, N., Mohan, N.L., Rao, S.V.S., 1983, Gravity interpretation of 2D fault structures using Hilbert transforms: Journal of Geophysics, 53, 34-41.

- Talwani, M., Worzel, J., Ladisman, M., 1959, Rapid gravity computations for two dimensional bodies with application to the Mendocino submarine fracture zone: *Journal of Geophysical Research*, 64, 49 - 59.
- Tankard, A.J., Welsink, H.J., 1987, Extensional tectonics and stratigraphy of Hibernia oil field, Grand Banks, Newfoundland: *American Association of Petroleum Geologists Bulletin*, 71, 1210-1232.
- Tarantola, A., 2005, *Inverse Problem Theory and Model Parameter Estimation*: Society for Industrial and Applied Mathematics.
- Telford, W.M., Geldert, L.P., Sheriff, R.E., 1990, *Applied Geophysics*, Cambridge University Press.
- Tenzer, R., Gladkikh, V., 2014, Assessment of Density Variations of Marine Sediments with Ocean and Sediment Depths: *The Scientific World Journal*, Article ID 823296, <http://dx.doi.org/10.1155/2014/823296>.
- Thanassoulas, C., Tselentis, G.A., Dimitriadis, K., 1987, Gravity inversion of a fault by Marquardt's method: *Computers & Geosciences*, 13, 399–404.
- Thompson, D.T., 1982, EULDPH – A new technique for making computer assisted depth estimates from magnetic data: *Geophysics*, 47, 31–37.
- Thorne, J.A., Watts, A.B., 1989, Quantitative analysis of North Sea subsidence: *The American Association of Petroleum Geologists Bulletin*, 73, 88–116.
- Torizin, J., Jentzsch, G., Malischewsky, P., Kley, N., Ab, J., akanov, 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, 269–278.
- Toushmalani, R., 2013, Gravity inversion of a fault by Particle swarm optimization (PSO): *Springer Plus*, 2, 315.
- Venteris, E., Miller, M., 1993, Gravitational profiles on the Taku Glacier System, Glaciological and Arctic Sciences Institute, University of Idaho, Open File Report.
- Visweswara Rao, C., Chakravarthi, V., Raju, M.L., 1994, Forward Modelling: Gravity anomalies of two-dimensional bodies of arbitrary shape with hyperbolic and parabolic density functions: *Computers and Geosciences*, 20, 873-880.
- Wang, T., Tucholke, J.L.B., Chen, Y.J., 2011, Crustal thickness anomalies in the North Atlantic Ocean basin from gravity analysis: *Geochemistry, Geophysics, Geosystems*, 12, 3, Q0AE02, doi:10.1029/2010GC003402.
- Wernicke, B., Burchfiel, B.C., 1982, Modes of extensional tectonics: *Journal Structural Geology* 4: 105-115.
- Wildman, R.A., Gazonas, G.A., 2009, Gravitational and magnetic anomaly inversion using a tree-based geometry representation: *Geophysics*, 74, I23–I35.
- Wilsher, W.A., 1987, A structural interpretation of the Witwatersrand basin through the application of automated depth algorithms to both gravity and aeromagnetic data, M.Sc. thesis, University of Witwatersrand.

- Won, I.J., Bevis, M., 1987, Computing the gravitational and magnetic anomalies due to a polygon: Algorithms and Fortran subroutines: *Geophysics*, 52, 232-238.
- Yao, C., Hao, T., Guan, Z., Zhang, Y., 2003, High-speed computation and efficient storage in 3-D gravity and magnetic inversion based on genetic algorithms: *Acta Geophysica Sinica*, 46, 252-258.
- Zervos, F., 1987, A compilation and regional interpretation of the northern North Sea gravity map: in *Continental Extensional Tectonics*, Eds. Coward, M.P., Dewey, J.F., and Hancock, P.L., 28, 477–493.
- Zhang, J., Zhong, B., Zhou, X., Dai, Y., 2001, Gravity anomalies of 2-D bodies with variable density contrast: *Geophysics*, 66, 809-813.
- Zhang, C., Mushayandebvu, M.F., Reid, A.B., Fairhead, J.D., Odegard, M.E., 2000, Euler deconvolution of gravity tensor gradient data: *Geophysics*, 65, 512–520.
- Zhou, X., 2008, Two-dimensional (2D) vector gravity potential and general line integrals for gravity anomaly due to 2D masses of depth-dependent density contrast: *Geophysics*, 73, I43–I50.
- Zhou, X., 2009, General line integrals for gravity anomalies of irregular two-dimensional (2D) masses with horizontally- and vertically-dependent density contrast: *Geophysics*, 74, I1–I7.



Zhou, X., 2013, Gravity inversion of 2D bedrock topography for heterogeneous sedimentary basins based on line integral and maximum difference reduction methods: *Geophysical Prospecting*, 61, 220–234.