

NUMERICAL METHODS FOR IMAGE RECONSTRUCTION FOR THE
CALIBRATION OF THE NASA-GLENN ICING RESEARCH WIND
TUNNEL: A COMPUTER-BASED APPROACH

by ALEXANDER ROMANOVICH STATNIKOV

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

Thesis Advisor: Dr. Steven H. Izen

Department of Mathematics
CASE WESTERN RESERVE UNIVERSITY

August 2002

Dedication

To my wonderful family - my wife, Kristina Statnikova, my mother, Elena Feofilova, my father, Roman Statnikov, my sister, Irina Statnikova, and my mother-in-law, Ludmila Morozjuk - for their support and understanding.

Nothing I do would be possible without their help.

Table of Contents

1	Introduction	1
2	Mathematical Model	6
3	Mathematical Approaches	13
3.1	The Truncated Singular Value Decomposition	14
3.2	The Tikhonov Regularization	18
3.3	LSQR	22
3.3.1	Lanczos Bidiagonalization	22
3.3.2	Best Approximate Solutions	23
3.3.3	LSQR Algorithm	24
3.3.4	Stopping Criteria	26
3.4	The Iterative Singular Value Decomposition	28
3.4.1	Equivalent Eigenvalue Problems	28
3.4.2	Subspace Iteration Method	29
3.4.3	Trace Minimization Method	30
3.4.4	Single-Vector Lanczos Method	31
3.4.5	Lanczos Block Method	33
4	Computer Implementations	34

4.1	LGEN Overview	35
4.2	LINE2MAT Overview	36
4.3	INVERT Overview	39
4.3.1	TSVD in INVERT	40
4.3.2	Tikhonov Regularization in INVERT	42
4.3.3	LSQR in INVERT	44
4.4	Truncated SVD Implementation	45
4.5	Tikhonov Regularization Implementation	47
4.6	LSQR Implementation	49
4.7	Inefficient Implementation of Sparse Iterative SVD	50
5	Programming and Design Considerations	54
5.1	Choice of Compiler	54
5.2	Choice of Libraries	55
5.2.1	Scientific Computations Library	55
5.2.2	X-Windows Graphical User Interface Library	56
5.2.3	Computer Graphics Libraries	57
5.2.4	Data Manipulation Libraries	57
5.3	Separation of Interface from Implementation	58
5.4	LGEN Design	58
5.5	LINE2MAT Design	62
5.6	INVERT Design	64
6	Numerical Experiments	69
6.1	Reconstruction via TSVD	70

6.1.1	Using the Dataset d_1	72
6.1.2	Using the Dataset d_2	74
6.1.3	Using Laboratory Data	76
6.2	Reconstruction via Tikhonov Regularization	79
6.2.1	Using the Dataset d_1	80
6.2.2	Using the Dataset d_2	82
6.2.3	Using Laboratory Data	83
6.3	Reconstruction via LSQR	84
6.3.1	Using the Dataset d_1	84
6.3.2	Using the Dataset d_2	86
6.3.3	Using Laboratory Data	87
7	Comparison and Conclusions	89
8	Appendix A: Software Package User Manual	93
8.1	Introduction	93
8.2	System Requirements	94
8.3	Hardware Requirements	94
8.4	Installation Package Contents	95
8.5	Using LGEN	97
8.5.1	Starting LGEN Session	97
8.5.2	Starting a New LGEN Project	97
8.5.3	Loading Existing LGEN Project	99
8.5.4	Drawing All Lines	100
8.5.5	Showing Line Origins	100

8.5.6	Changing “Skip Line” Factor	100
8.5.7	Specifying Numbering Properties	101
8.5.8	Showing Source and Detector Numbers	102
8.5.9	Showing Grid	102
8.5.10	Changing Grid Spacing	102
8.5.11	Changing Grid Origin	102
8.5.12	Toggling Background	103
8.5.13	Selecting Object	103
8.5.14	Moving Object Along the Frame	103
8.5.15	Changing Object Properties	104
8.5.16	Adding Object	104
8.5.17	Removing Object	105
8.5.18	Undoing Object Movement/Removal	105
8.5.19	Connecting and Disconnecting Two Objects	106
8.5.20	Changing Object Line Destinations	107
8.5.21	Showing Lines for Specific Object	107
8.5.22	Deactivating or Activating the Object	108
8.5.23	Equispacing Objects	108
8.5.24	Changing Project Properties	109
8.5.25	Viewing Geometry Information	111
8.5.26	Resizing Objects	111
8.5.27	Resizing LGEN Window	112
8.5.28	Saving the Lines File	112
8.5.29	Saving TIFF File	113

8.5.30	Saving Encapsulated Postscript File	114
8.5.31	Saving LGEN Project	114
8.5.32	Ending LGEN Session	116
8.6	Using LINE2MAT	116
8.6.1	General Principles	116
8.6.2	Starting a New “Dense Task”: Required Options	117
8.6.3	Loading an Existing “Dense Task”: Required Options	117
8.6.4	Additional Options for a “Dense Task”	118
8.6.5	Starting a New “Sparse Task”: Required Options	123
8.6.6	Loading an Existing “Sparse Task”: Required Options	126
8.6.7	Additional Options for a “Sparse Task”	127
8.6.8	LINE2MAT_INTERFACE	128
8.7	Using INVERT	136
8.7.1	Starting INVERT, Task Manager Window	136
8.7.2	INVERT GUI Elements Common to All Methods	141
8.7.3	Using TSVD method	142
8.7.4	Using Tikhonov Regularization Method	144
8.7.5	Using LSQR Method	146
9	Appendix B: Software Package Developer Manual	151
9.1	Preface	151
9.2	Developing LGEN	152
9.2.1	Compiling the Program	152
9.2.2	Design Overview	154
9.2.3	GUI Design	155

9.2.4	Constants and Limits	156
9.2.5	Artificial Data-Structures	157
9.2.6	Key Methods Description	159
9.3	Developing LINE2MAT	164
9.3.1	Compiling the Program	164
9.3.2	Design Overview	166
9.3.3	GUI Design	168
9.3.4	Artificial Data-Structures	168
9.3.5	Constants and Limits	170
9.3.6	Key Methods Description	171
9.3.7	Adding New Basis Functions	171
9.4	Developing LINE2MAT Interface: LINE2MAT_INTERFACE	174
9.4.1	Compiling the Program	174
9.4.2	GUI Design	175
9.4.3	Updating this Program	176
9.4.4	Key Methods Description	176
9.5	Developing INVERT	176
9.5.1	Compiling the Program	176
9.5.2	Design Overview	179
9.5.3	GUI Design	181
9.5.4	Key Methods Description	182
9.5.5	Adding a New Computational Engine	182

10 Appendix C: Sparse Matrix Formats 189

List of Tables

8.1	Starting a new “dense task”. Required options.	118
8.2	Loading an existing “dense task”. Required option.	118
8.3	Additional options for a “dense task” (Part 1).	124
8.4	Additional options for a “dense task” (Part 2).	125
8.5	Starting a new “sparse task”. Required options.	126
8.6	Loading an existing “sparse task”. Required option.	126
8.7	Additional options for a “sparse task”.	129
8.8	LINE2MAT_INTERFACE task file format [left]. Equivalent LINE2MAT options [right].	135
8.9	Task-specific task file options.	138
8.10	Common task file options.	139
8.11	LSQR termination flag values.	150
9.1	Methods of the class <i>lg</i> (callback and event “drivers” are not listed). Part 1.	160
9.2	Methods of the class <i>lg</i> (callback and event “drivers” are not listed). Part 2.	161
9.3	Methods of the class <i>lg</i> (callback and event “drivers” are not listed). Part 3.	162

9.4	Methods of the class <i>lg</i> (callback and event “drivers” are not listed). Part 4.	163
9.5	Method of the class <i>object</i>	163
9.6	Methods of the class <i>linked_list</i>	164
9.7	Methods of the class <i>lines_database</i>	164
9.8	Main application methods.	171
9.9	Methods of the class <i>cl2m</i> . Part 1.	172
9.10	Methods of the class <i>cl2m</i> . Part 2.	173
9.11	Methods of the class <i>line_item</i>	173
9.12	Methods of the class <i>line_list</i>	173
9.13	Method of the class <i>endpoints</i>	173
9.14	Method of the class <i>l2m</i> (the only class of <code>LINE2MAT_INTERFACE</code>). Callback “drivers” are not listed.	177
9.15	Main application methods.	183
9.16	Methods of the class <i>work_window</i> (callback and event “drivers” are not listed). Part 1.	184
9.17	Methods of the class <i>work_window</i> (callback and event “drivers” are not listed). Part 2.	185
9.18	Methods of the class <i>work_window</i> (callback and event “drivers” are not listed). Part 3.	186
9.19	Methods of the class <i>svd</i>	187
9.20	Methods of the class <i>tikh</i>	187
9.21	Methods of the class <i>iter</i>	188

List of Figures

1.1	Plan view of Icing Research Tunnel facility.	2
1.2	Photograph of the lab scale model. The small disks appearing around the edge of the frame are detectors, which are connected by fiber-optic cables to a CCD camera. The water stream which is imaged by the apparatus is emitted from the jet which appears in the middle, just left of center.	3
1.3	The source is in the range of the detector (located on the other wall). Likewise, the detector is in the range of the source.	4
2.1	Line l_i from a source on one wall to the detector on another wall is defined via distance p_i and the angle ϕ_i	7
2.2	Graphics display of the discretized Radon transform matrix. Black corresponds to 0 (i.e. line l_i which defines the row of the matrix does not intersect pixel j corresponding to the matrix column), while gray and white corresponds to nonzero line integral value (i.e. line l_i intersects pixel j , and the line segment in the pixel is of nonzero length). The depicted matrix is sparse. Lines are sorted by sources starting from the left upper corner of the sensor system.	8

2.3	Lines starting from the sources located on the top side of the rectangular frame.	11
4.1	LGEN in action: Geometry project view.	36
4.2	INVERT in action: Multiple views for computational engines. Reconstruction via SVD [left]. Reconstruction via LSQR [right].	41
5.1	Sketch of the LGEN UML class diagram. Only a few methods and data structures are shown. The diagram was generated by Rational Rose.	59
5.2	Sketch of the LINE2MAT UML class diagram. Only a few methods and data structures are shown.	63
5.3	Sketch of the INVERT UML class diagram. Only a few methods and data structures are shown.	65
5.4	Proposed design of INVERT. Sketch of UML class diagram.	68
6.1	Simulated datasets d_1 [left] and d_2 [right] consisting of 864 pixels displayed as 36 by 24 pixel image in grayscale. The dataset d_1 represents a typical liquid water distribution in the wind tunnel. The dataset d_2 consists of one white pixel.	70
6.2	A plot of the singular value ratio σ_k/σ_1 when weighting was not applied to the matrix A . There are 864 singular values. The slow drop indicates that the inversion of A is not ill-conditioned.	71
6.3	A plot of the singular value ratio σ_k/σ_1 when weighting was applied to the matrix A . There are 864 singular values. The slow drop indicates that the inversion of A is not ill-conditioned.	71

6.4	The right singular vectors v_1, v_{17}, v_{806} , and v_{857} when weighting was not applied to the matrix A . The corresponding singular value ratios are $\sigma_1/\sigma_1 = 1$, $\sigma_{17}/\sigma_1 = 0.3994$, $\sigma_{806}/\sigma_1 = 0.0046$, and $\sigma_{857}/\sigma_1 = 0.0001$	73
6.5	The right singular vectors v_1, v_{17}, v_{806} , and v_{857} when weighting was applied to the matrix A . The corresponding singular value ratios are $\sigma_1/\sigma_1 = 1$, $\sigma_{17}/\sigma_1 = 0.4044$, $\sigma_{806}/\sigma_1 = 0.0049$, and $\sigma_{857}/\sigma_1 = 0.0001$	73
6.6	Reconstructions without weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triples [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triples [bottom,right] with noise level 5%.	74
6.7	Reconstructions with weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triples [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triples [bottom,right] with noise level 5%.	75

6.8	Reconstructions without weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triples [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triples [bottom,right] with noise level 5%.	76
6.9	Reconstructions with weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triples [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triples [bottom,right] with noise level 5%.	77
6.10	Reconstructions without weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left], 785 triplets [top,center], 521 triplets [top,right], 328 triples [bottom,left], and 256 triplets [bottom,right].	78
6.11	Reconstructions with weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left], 785 triplets [top,center], 521 triplets [top,right], 328 triples [bottom,left], and 256 triplets [bottom,right].	78
6.12	A graphical grayscale representation of the matrix $A^T A$. Black pixels correspond to minimum elements, and white pixels correspond to maximum elements.	80

6.13	Reconstructions without weighting. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] without noise. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] with noise level 5%.	81
6.14	Reconstructions with weighting. Using $\tau^2 = 0.3$ without noise [left] and with 5% noise [right].	81
6.15	Reconstructions without weighting. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] without noise. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] with noise level 5%.	82
6.16	Reconstructions with weighting. Using $\tau^2 = 0.3$ without noise [left] and with 5% noise [right].	83
6.17	Reconstructions without weighting. Using $\tau^2 = 0.001$ [left], 1.5 [center], and 20 [right].	83
6.18	Reconstruction with weighting. Using $\tau^2 = 0.3$	84
6.19	Reconstruction without damping, noise, and weighting. Using 178 iterations [left] and 20 iterations [right].	85
6.20	Reconstruction without weighting with 5% noise. Using 1841 iterations and 0 damping [top, left], 136 iterations and 0.5 damping [top, right], 40 iterations and 2 damping [bottom, left], and 12 iterations and 10 damping [bottom, right].	86
6.21	Reconstruction with weighting and 5% noise. Using 1841 iterations without damping [left] and 68 iterations with damping 0.4 [right].	86

6.22	Reconstruction without weighting with noise level 5% [top]. Using 1841 iterations and 0 damping [top, left], 616 iterations and 0.1 damping [top, center], 20 iterations with 0.1 damping [top, right]. Reconstructions with weighting with noise level 5% [bottom]. Using 1841 iterations and 0 damping [bottom, left], 243 iterations and 0.1 damping [bottom, center], 20 iterations with 0.1 damping [bottom, right].	87
6.23	Reconstruction without weighting 5% [top]. Using 1841 iterations and 0 damping [top, left], 138 iterations and 0.5 damping [top, center], 73 iterations with 1 damping [top, right]. Reconstructions with weighting [bottom]. Using 1841 iterations and 0 damping [bottom, left], 128 iterations and 0.2 damping [bottom, center], 56 iterations with 0.5 damping [bottom, right].	88
8.1	Starting a new LGEN project.	97
8.2	The illustration of major geometry initialization.	98
8.3	A new project is initialized.	99
8.4	Specifying numbering properties.	101
8.5	Changing object properties.	105
8.6	Connecting and disconnecting two objects.	106
8.7	Changing object line destinations.	107
8.8	Equispacing objects.	109
8.9	Changing project properties.	110
8.10	Viewing geometry information.	111
8.11	Resizing objects.	112

8.12	Saving lines file.	113
8.13	Viewing projection matrix A on the screen. Using internal viewer.	120
8.14	LINE2MAT_INTERFACE: Dense task options.	131
8.15	LINE2MAT_INTERFACE: Sparse task options.	132
8.16	INVERT Task manager window.	137
8.17	Using TSVD method.	143
8.18	Using Tikhonov regularization method.	145
8.19	Using LSQR method.	147
9.1	Sketch of the LGEN UML class diagram. Only a few methods and data structures are shown. The diagram was generated by Rational Rose.	154
9.2	Sketch of the LINE2MAT UML class diagram. Only a few methods and data structures are shown.	167
9.3	Sketch of the INVERT UML class diagram. Only a few methods and data structures are shown.	180

Acknowledgements

I am thankful for all the positive support my advisor, Professor Steven Izen, has given me and for introducing to the exciting world of imaging and software engineering.

I am also thankful to my committee members, Professors Marshall Leitman and Andy Podgurski.

I express my heartfelt thanks to Professors Daniela Calvetti, Nancy Nichols, Marc Buchner, and Shiva Sastry, who provided me with the educational background necessary to accomplish this project.

I greatly acknowledge the time and efforts of Professor Scott Kersey, Graduate Student Phil Hahn, and Undergraduate Student Alexander Alekseyenko, who helped me with some parts of this project.

Numerical Methods for Image Reconstruction for the Calibration of the NASA-Glenn Icing Research Wind Tunnel: A Computer-Based Approach

Abstract

by

Alexander Romanovich Statnikov

Ice formation on the aircraft wings has been implicated in a number of aviation accidents, so understanding this process is of significant interest to the aeronautics industry. To understand ice formation process, one has to know the liquid water content as a function of position of the cloud.

This thesis focuses on the application and implementation of numerical methods for the reconstruction of the liquid water content from both laboratory and synthetic data. The theory, computer implementations, and analysis of the results are presented for each method used. A software package with a user-friendly interface to model the geometry, perform reconstructions, and analyze results was developed. This software incorporated a number of existing off-the-shelf computer libraries to maximize the performance, reliability, and user interface features. The implemented software architecture allows developers to easily in add new computational engines or extend functionality.

Chapter 1

Introduction

Ice formation on the aircraft wings has been implicated in a number of aviation accidents, so understanding the process of the ice formation, as well as testing the performance of objects under icing conditions is of significant interest to the aeronautics industry [13]. Such conditions can be achieved in NASA-Glenn Icing Research Tunnel (IRT), a facility which duplicates the natural icing environment that aircraft typically encounters (see Figure 1.1). The conditions are simulated via the refrigeration plant and a spray-bar system that generates a cloud of microscopic droplets of supercooled water [21]. After the air is cooled to the freezing temperatures by the refrigerator, the system of nozzles injects supercooled water in the airflow. The resulting cloud of tiny water droplets surrounds the testing object located in the $9' \times 6'$ test section. In order to understand the process of ice growth, it is necessary to know the liquid water content as a function of position of the cloud as it arrives in the test section [13]. Ideally, the particle distribution in the cloud would be uniform. However, in reality it is not uniform, and the tunnel has to be calibrated in order to produce a more uniform water distribution. For that purpose the measurements of the water distribution followed by an adjustment

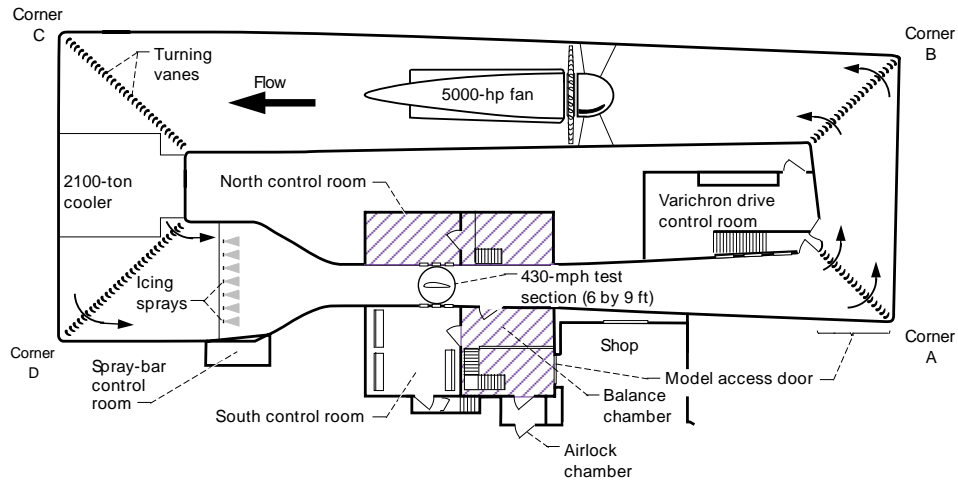


Figure 1.1: Plan view of Icing Research Tunnel facility.

of the nozzles should be performed.

One of the most promising ways to take the measurements of the cloud distributions is to use the optical sensor system proposed by Tim Bencic of the Glenn Research Center's Optical Instrumentation Technology Branch of the Instrumentation and Controls Division. The system consists of the sources and detectors placed along the walls of a section of wind tunnel proximate to the test section (see Figure 1.2). This sensor system will operate in almost real-time without any impact on the flow and will allow to obtain distribution profiles at high resolutions.

The sources are illuminated rapidly in a sequence, and the attenuation from each source to each detector not on the same wall (provided that the source is in the range of a detector and vice-versa) is recorded (see Figure 1.3). The liquid water content is measured by ρa^2 , where ρ is the particle density and a is its radius. Given that the multiple scattering interactions are not significant, the probability of a photon being scattered at a particular location

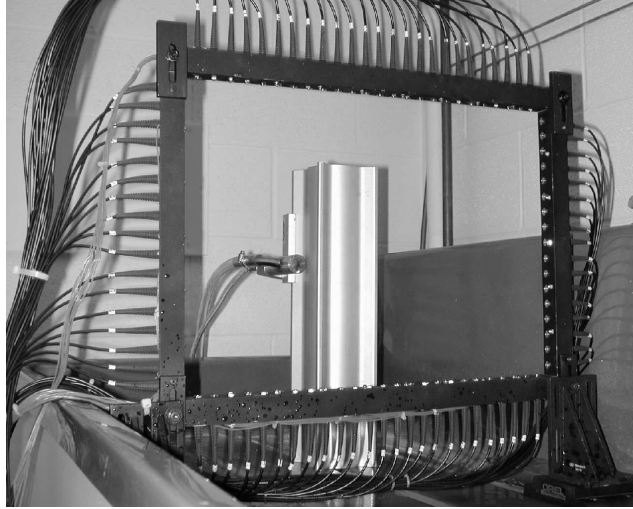


Figure 1.2: Photograph of the lab scale model. The small disks appearing around the edge of the frame are detectors, which are connected by fiber-optic cables to a CCD camera. The water stream which is imaged by the apparatus is emitted from the jet which appears in the middle, just left of center.

along a line from the source to the detector is proportional to the liquid water content at the point [13]. The Radon transform of the liquid water content along the line l_i of length s connecting a source with a detector is proportional to the logarithm of the ratio between the intensity I_0 of a source and the final intensity I measured at the detector:

$$-\ln \frac{I}{I_0} = \int_{l_i} \mu(x) ds, \quad (1.1)$$

where $\mu = c\rho a^2$ is the attenuation coefficient with c proportionality constant.

A recent paper by Steven H. Izen and Timothy J. Bencic [13] reports on the feasibility of the usage of the proposed sensor system investigating to what extent liquid water content could be reliably reconstructed from the data measured. The paper also takes into account different acquisition geometries. Namely, placement of the sensors on three walls of the wind tunnel; three

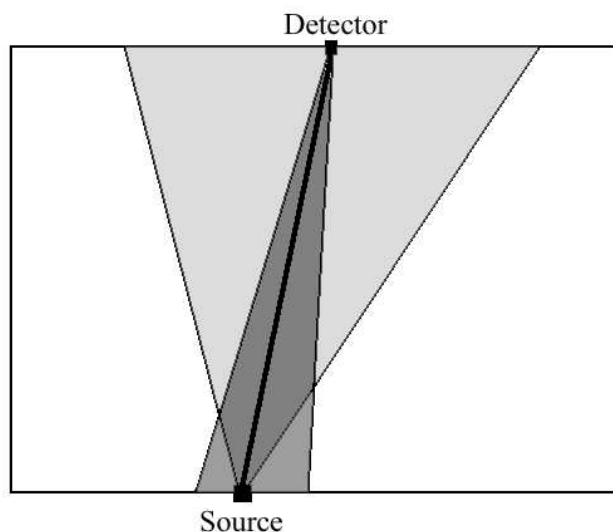


Figure 1.3: The source is in the range of the detector (located on the other wall). Likewise, the detector is in the range of the source.

walls with the partially covered floor omitting the central region; and the full four wall coverage. Analyzing reconstructions of the liquid water content, the authors conclude that the proposed sensor system is robust enough to provide meaningful information about the distribution of the super-cooled water in the Icing Research Wind Tunnel when sources and detectors are located on the four walls.

This thesis focuses on the application and implementation of both iterative and non-iterative numerical mathematical methods for the reconstruction of the liquid water content in the wind tunnel from the laboratory as well as synthetic data using four walls acquisition geometry. The theory, computer implementations, and the analysis of the numerical results are presented for each method used. A software package with user-friendly interface to model the geometry of the sensor system, perform reconstructions

from the given data, and analyze obtained results was developed and tested. This software incorporated a number of existing off-the-shelf computer code libraries in order to maximize the performance, reliability, as well as user interface features of the program. The implemented software architecture allows developers to reuse the program code easily in order to add new computational engines or extend functionality of the existing ones.

Chapter 2

Mathematical Model

The sensor geometry determines the set of lines $L = \{l_i\}$ from the sources to detectors not on the same wall (provided that the each source is in the range of the detector and vice-versa). Each line l_i is defined via its distance from the origin (located in the geometric center of the test section) p_i and the angle ϕ_i formed by the normal and the horizontal line going through the origin:

$$x \cdot \psi_i = p_i,$$

where $\psi_i = (\cos \phi_i, \sin \phi_i)^T$ (see Figure 2.1).

The the samples of Radon transform of unknown attenuation $\mu(x)$ on the set of lines L are given by:

$$R\mu(l_i) = \int_{l_i \in L} \mu(x) ds, \quad (2.1)$$

where the Radon transform is given by

$$R : L^2[D] \rightarrow L^2[R^+ \times S^1, W], \quad (2.2)$$

where $D = [a, b] \times [c, d]$ is a rectangular region in \mathbb{R}^2 . The weighting factor $W = \{w_{ii}\}$, where w_{ii} is reciprocal of the length of intersection of line l_i with the rectangular region.

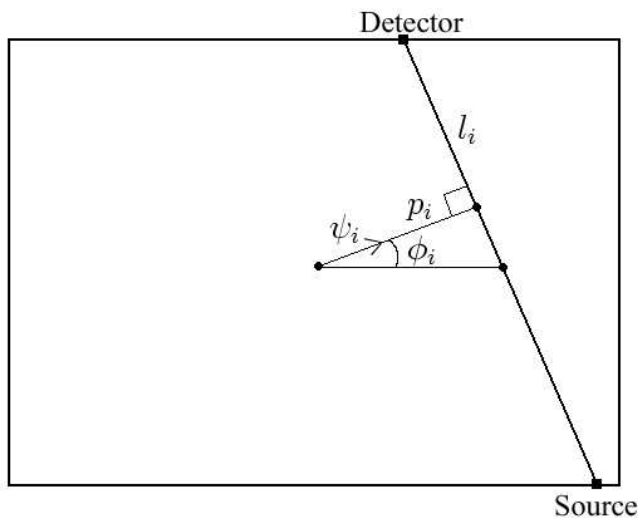


Figure 2.1: Line l_i from a source on one wall to the detector on another wall is defined via distance p_i and the angle ϕ_i .

The rectangular geometry leads to the unavoidable irregular sampling of p and ψ . Hence, direct reconstruction methods such as convolution-backprojection would require interpolation which does not guarantee to provide an accurate reconstruction. Moreover, design constraints may cause large group of samples missing. Direct methods do not handle this case well. In addition to that, direct limited angle algorithms would be difficult to apply because of the irregular sampling and sparsity of the discretized Radon transform matrix (see Figure 2.2) [13].

As suggested in [13], the best way to remedy the irregular sampling and consider the sparsity of the discretized Radon transform matrix is to treat the problem as a discrete inversion. Namely, instead of discretizing the inverse transform R^{-1} , the Radon transform R is discretized before inversion. The characteristic functions of pixels were selected as a basis for this discretization,

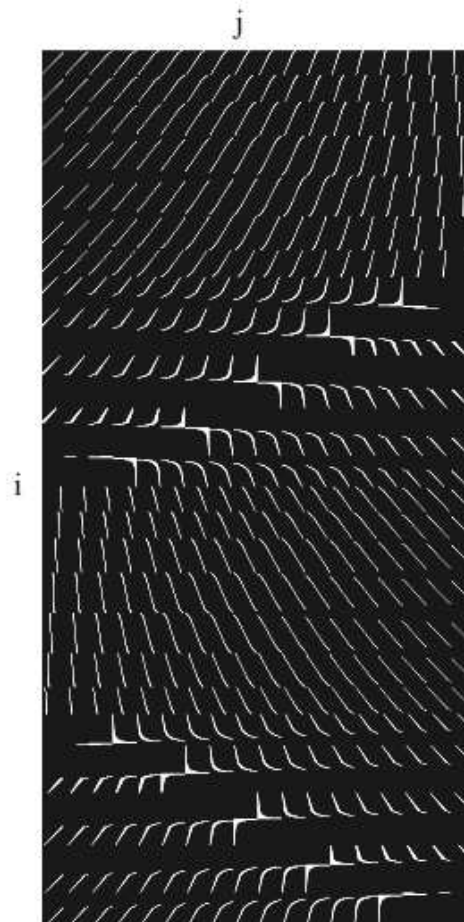


Figure 2.2: Graphics display of the discretized Radon transform matrix. Black corresponds to 0 (i.e. line l_i which defines the row of the matrix does not intersect pixel j corresponding to the matrix column), while gray and white corresponds to nonzero line integral value (i.e. line l_i intersects pixel j , and the line segment in the pixel is of nonzero length). The depicted matrix is sparse. Lines are sorted by sources starting from the left upper corner of the sensor system.

since it simplifies visualization of the reconstructed images. The reconstruction region was divided into n pixels.

The attenuation which will be reconstructed from the image vector x is given as follows:

$$\mu = \sum_{j=1}^n x_j \Phi_j. \quad (2.3)$$

By linearity, the discretized Radon transform operator becomes a matrix A (see Figure 2.2):

$$A = \{a_{ij}\}, \quad (2.4)$$

$$a_{ij} = \int_{l_i} \Phi_j ds, \quad (2.5)$$

where a_{ij} entry is the Radon transform of Φ_j , the characteristic function for the j^{th} pixel evaluated on the i^{th} line.

The index i of the matrix A ranges from 1 to the total number of lines m , which in turn depends on the particular sensor geometry. The lines can be sorted based on the sources, detectors, distance p , and angle ϕ . Each sorting choice produces a different matrix A . The j index of the matrix ranges from 1 to the total number of pixels n , which defines the expected resolution of the reconstruction.

Given the intensity data $b \in \mathbb{R}^m$ and the discretized Radon transform matrix $A : \mathbb{R}^n \rightarrow \mathbb{R}^m$, the image vector $x \in \mathbb{R}^n$ is obtained by solving the system of linear equations:

$$Ax = b. \quad (2.6)$$

The described model setup does not account for the weighting in the reconstructed image. The matrix A^*A is a discrete approximation of $R^T R$. However, we need to approximate R^*R instead, where R^* is a backprojection

operator which is mathematically adjoint to R defined in (2.2). Hence, the discrete approximation should incorporate the weighting, and we will compute A^*WA , where W is a diagonal of positive weight elements. Instead of computing A^*WA , we compute $(W^{1/2}A)^*(W^{1/2}A)$ which leads to the new system:

$$W^{1/2}Ax = W^{1/2}b. \quad (2.7)$$

Hence the scaled matrix $W^{1/2}A$ will have elements of the form:

$$W^{1/2}A = \{w_{ii}^{1/2}a_{ij}\}, \quad w_{ii}^{1/2} = \frac{1}{\sqrt{\sum_{j=1}^m (\int_{l_i} \Phi_j ds)^2}}, \quad (2.8)$$

where w_{ii} is reciprocal of the length of intersection of line l_i with the rectangular region.

The reconstruction region designed to model a 9' by 6' feet rectangle of the wind tunnel test section. The geometry of the laboratory mock model was represented by 24 sources with 5 evenly distributed sources on both left and right sides of the test section, 8 evenly distributed sources on the bottom side, and 7 sources on on the top side. The sources on the top side of the test section were placed as there were 8 evenly distributed sources and the last one was missing. The 84 detectors were placed evenly along the sides: 25 on top and bottom sides and 17 on the left and right sides. No source or detector was placed in the corner due to the engineering constraints of the proposed sensor system. Each source and detector was associated with a particular angle of acceptance. A line connecting a source and a detector was generated provided that both objects were in the range of each other. See Figure 2.3 for the lines starting from the sources located on the top side of the rectangular frame.

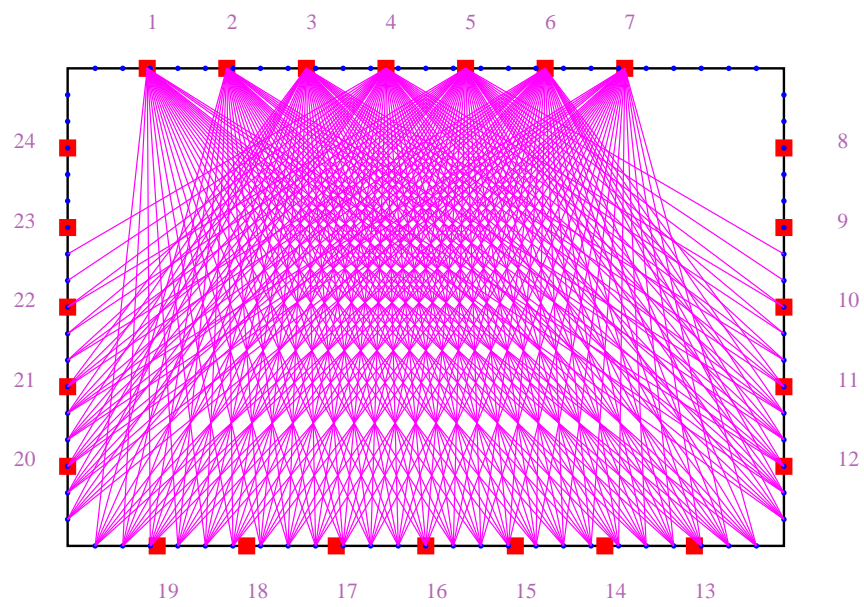


Figure 2.3: Lines starting from the sources located on the top side of the rectangular frame.

The 9 by 6 foot reconstruction region was divided into pixels in order to compute the matrix A as described above. Namely, the square pixels used with sides $\frac{1}{4}$ feet ($36 \times 24 = 864$ pixels) were as the reconstruction basis. Since the reconstructed image has to be in 3:2 ratio, and the geometry consists of only 977 lines, and the numerical techniques described in this thesis operate on overdetermined matrices (when the number of columns is greater than the number of rows), no better resolution for this particular geometry can be achieved.

Chapter 3

Mathematical Approaches

The linear system (2.6) will, in general, be well-conditioned for a four-sided geometry. However, the constraints on geometry can easily imply an ill-conditioned system. For instance, if we restrict the placement of the sources to three sides of the test section, the system will become poorly conditioned, since the data needed to reconstruct significant features is missing. In addition to the possible ill-conditioning, measured data b always contains noise, which in turn gets reflected on the reconstructed image. In order to avoid ill-conditioning and reduce the noise, regularization of the solution x is necessary. Furthermore, the issue of the large size of the system certainly has to be considered to obtain meaningful results in a reasonable amount of time.

The sizes of the wind tunnel system considered in this study allow the application of non-iterative techniques to perform relatively fast on workstation class computers. However, to solve large systems, iterative methods are required. Iterative methods often require significantly less resources, perform much faster, and provide meaningful results after certain iteration.

From among the regularization methods available and discussed in [6]

and [11], the Truncated Singular Value Decomposition (TSVD) provides a good qualitative, as well as quantitative sense of which features in the image can be reconstructed, and which features will be sensitive to the presence of noise in the data [13]. On the other hand, TSVD is not the most efficient non-iterative algorithm, and it does not necessarily provide the “best” reconstructions.

The computation of the Singular Value Decomposition (SVD) of matrix A , the first step in the TSVD, is expensive in terms of memory and execution time. Due to the memory and execution time limitations, the use of a large number of sources and detectors leading to large matrices A is restricted by SVD implementation. Nevertheless, optimized computer codes for the use of TSVD method are readily available [3]. Moreover, a number of numerical methods library vendors succeeded in the implementation for multiprocessor computers of the TSVD method which takes advantage of the system architecture [7].

3.1 The Truncated Singular Value Decomposition

The SVD provides a diagonal form the matrix A under an orthogonal equivalence transformation [6]. For a matrix $A \in \mathbb{R}^{m \times n}$ of some rank K , SVD can be defined as the matrix factorization:

$$A = U\Sigma V^T, \quad \Sigma = \begin{pmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{pmatrix}, \quad (3.1.1)$$

where $\Sigma \in \mathbb{R}^{m \times n}$, $\Sigma_1 = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_K)$, and $\{\sigma_i\}$ is a decreasing sequence of positive numbers. The numbers σ_i are called the singular values of matrix A . Matrices $U = (u_1, u_2, \dots, u_m) \in \mathbb{R}^{m \times m}$ and $V = (v_1, v_2, \dots, v_n) \in \mathbb{R}^{n \times n}$ are

unitary matrices. The vectors u_i and v_i are respectively left and right singular vectors associated with singular values $\sigma_i, i = 1, \dots, K$.

Moreover, $\{u_1, u_2, \dots, u_K\}$ and $\{v_1, v_2, \dots, v_K\}$ form orthonormal bases for $\text{Range}(A)$ and $\text{Null}(A)^\perp$ respectively. The bases are extended to \mathbb{R}^m and \mathbb{R}^n respectively, by choosing an orthonormal basis $\{u_{K+1}, u_{K+2}, \dots, u_m\}$ for $\text{Range}(A)^\perp$ and $\{v_{K+1}, v_{K+2}, \dots, v_n\}$ for $\text{Null}(A)$.

A rectangular matrix A can be also viewed as the linear map $A : \mathbb{R}^n \mapsto \mathbb{R}^m$ represented in the light of SVD by:

$$Ax = \sum_{i=1}^K u_i \sigma_i (v_i^T x), \quad (3.1.2)$$

where $x \in \mathbb{R}^n$. By this representation the effect of a rank K matrix A is the same as of the sum of K matrices of rank 1.

The Moore-Penrose pseudo-inverse $A^\dagger : \mathbb{R}^m \mapsto \mathbb{R}^n$ uniquely solves the generalized least squares problem, i.e. it selects the unique solution $x \in \mathbb{R}^n$ such that $\|Ax - b\|_2$ is minimized. The choice of 2-norm can be defended by various geometric and statistic arguments as well as that is leads to simple algorithms - ultimately because the derivative of a quadratic function, which must be set for minimization, is linear [22]. Geometrically the obtained solution $x \in \mathbb{R}^n$ makes the vector Ax the closest point in $\text{Range}(A)$ to b .

In terms of SVD the solution can be represented as

$$x = A^\dagger b = V \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T b, \quad (3.1.3)$$

or similarly in vector notation:

$$x = A^\dagger b = \sum_{k=1}^K \sigma_k^{-1} (u_k^T b) v_k. \quad (3.1.4)$$

Due to orthonormality of left singular vectors, the right-hand side b can be expanded as follows:

$$b = \sum_{i=1}^m (u_i^T b) u_i. \quad (3.1.5)$$

Then only the components of b which are in the range of A have influence on $A^\dagger b$, since the others are equal to 0. Each such component $(u_k^T b) u_k$ is translated to the component $\sigma_k^{-1} (u_k^T b) v_k$ in formula (3.1.4). By that translation the size of each component in data b is amplified by σ_k^{-1} in the solution $x = A^\dagger b$.

The white noise in the data b gets evenly distributed in all the directions u_i . The application of the Moore-Penrose operator on the data b reconstructs the noisy image. Since the noise the direction v_1 is amplified by σ_1^{-1} , and in the direction k is amplified by σ_k^{-1} , the noise in the direction k gets amplified by σ_1/σ_k compared to the noise in the direction v_1 . When σ_k/σ_1 is comparable or larger than the noise level of the data, the noise level in the reconstruction is comparable or smaller than the signal level. In other words, we may get a meaningful reconstructed image. On the other hand, when this ratio is smaller than the noise level, the reconstruction components in (3.1.4) get amplified by a value large enough to swamp the image with noise.

Given the ill-conditioned system, σ_k^{-1} becomes extremely large after some value of k . Hence the solution becomes amplified by large numbers, which might cause information loss and overflow.

In order to remedy the problems discussed above, a regularization of the Moore-Penrose inverse must be used. We simply ignore the SVD components associated with the small singular values $\sigma_\lambda, \sigma_{\lambda+1}, \dots, \sigma_K$, and

compute the regularized solution as follows:

$$x_\lambda = A_\lambda^\dagger b = \sum_{k=1}^K f_\lambda(\sigma_k) (u_k^T b) v_k, \quad (3.1.6)$$

where λ is the index of the first singular value we want to ignore and $f_\lambda(\sigma_k)$ is the filtering function which approximates σ_k^{-1} for $k \leq \lambda$, but decays or equals to 0 for $k > \lambda$. The most common filtering function choice is the following:

$$f_\lambda(\sigma_k) = \begin{cases} \sigma_k^{-1}, & k \leq \lambda \\ 0, & \text{otherwise.} \end{cases} \quad (3.1.7)$$

Using equation (3.1.7), we simply compute the regularized solution by:

$$x_\lambda = A_\lambda^\dagger b = \sum_{k=1}^{\lambda} \sigma_k^{-1} (u_k^T b) v_k. \quad (3.1.8)$$

If the truncation parameter λ in the sum (3.1.8) is too high, then the reconstructed image is swamped with noise, or equivalently may encounter singular values small enough to affect the image dramatically. On the other hand, if λ is too low, the noise is reduced, but by the same token we may lose the relevant information. For some ill-conditioned problems, it is often easy to determine λ , since the singular values σ_k exhibit sharp decay after some value of k . Then we set $\lambda = k$. The value of λ can be also determined using the so-called ‘‘L-curve criterion’’, an algorithm designed to minimize the function $\Theta(\lambda) = \log(\|x_\lambda\|) + \log(\|Ax_\lambda - b\|)$. There are a number of other implementations of ‘‘L-curve criterion’’ which minimize various functions to locate λ . However, when the singular values decay gradually the choice of λ is not evident. A number of criteria for selection of λ have been suggested in [11].

When the parameter λ is properly selected the noise in the image is reduced, and the reconstruction $x_\lambda = A_\lambda b$ contains all information from the

data b . It includes all the components v_k which are reconstructible from the data, and it excludes all components which cannot be reconstructed [13].

The filtering of the components corresponding to small singular values can also be accomplished by Tikhonov regularization method, where damping is added to each SVD component of the solution. Tikhonov regularization avoids the expensive computation of the singular value decomposition. The standard floating operation count (“addition” and “multiplication” are considered separate operations) for the SVD method for solution of least-square problems is of the order of $2mn^2 + 11n^3$ flops. On the other hand, the non-iterative implementation of Tikhonov regularization via Cholesky factorization requires only $n^3/3$ operations which is significantly cheaper given that $m \gg n$ for a typical matrix A discussed above. The disadvantage of Tikhonov regularization is that it does not allow one to analyze the contributions of each singular vector separately like with TSVD. Nevertheless, most importantly it provides accurate reconstructions similar to those obtained by TSVD.

Similarly to the TSVD method, the solution of system (2.6) by Tikhonov regularization can be efficiently implemented using available numerical mathematics libraries such as [3].

3.2 The Tikhonov Regularization

The idea behind Tikhonov regularization is to measure the “size” of the solution x by $\|Lx\|_2^2$ for some matrix L and minimize the linear combination of

the squared norm of the residual and $\|Lx\|_2^2$ for some value of $\tau > 0$:

$$\min(\|Ax - b\|_2^2 + \tau^2\|Lx\|_2^2) \quad (3.2.1)$$

Using this formulation of the problem we assume that errors in b are uncorrelated and have covariance matrix $\sigma_0 I$. If the covariance matrix of noise is of the form $C^T C$ (where C is of full rank), then one should solve instead [11]:

$$\min(\|C^{-1}(Ax - b)\|_2^2 + \tau^2\|Lx\|_2^2). \quad (3.2.2)$$

The problem (3.2.1) also called a damped least squares, is equivalent to the least squares problem:

$$\min \left\| \begin{pmatrix} \tau L \\ A \end{pmatrix} x - \begin{pmatrix} 0 \\ b \end{pmatrix} \right\|_2, \quad (3.2.3)$$

where the matrix A has been modified by appending the matrix τL . When $\tau > 0$, which implies that $\text{Null}(A) \cap \text{Null}(L) = \{0\}$, this problem is always of full column rank and has a unique solution.

The advantage of equation (3.2.3) is that its solution can be computed from the QR factorization

$$\begin{pmatrix} \tau L \\ A \end{pmatrix} = Q \begin{pmatrix} R \\ 0 \end{pmatrix} \quad (3.2.4)$$

using slightly modified algorithms of the Householder or Givens QR factorizations. In comparison with the Householder QR factorization of an m by n matrix, the operation count will increase with $2n^3/3$ flops to $2mn^2$ [6]. A similar increase occurs in Givens QR factorization. Given that $m \gg n$ for the matrix A , the performance of this method is still comparable to the usage of singular value decomposition which costs roughly $2mn^2 + 11n^3$ flops. Hence

we search for another algorithm which computational complexity does not depend on m .

If one considers an alternative version of (3.2.3):

$$(A^T A + \tau^2 L^T L)x = A^T b, \quad (3.2.5)$$

then the unique Tikhonov solution is given by:

$$x_{\tau,L} = (A^T A + \tau^2 L^T L)^{-1} A^T b. \quad (3.2.6)$$

Theorem. Given $L = I$, the solution of (3.2.5) can be expressed in terms of the SVD as:

$$x_\tau = \sum_{i=1}^n f_i \frac{u_i^T b}{\sigma_i} v_i, \quad (3.2.7)$$

where $f_i = \frac{\sigma_i^2}{\sigma_i^2 + \tau^2}$ are filter factors.

Proof. Assume, that $A = U\Sigma V^T$. Then

$$\begin{aligned} x_\tau &= (V\Sigma^T \Sigma V^T + \tau^2 I)^{-1} V\Sigma^T U^T b = \\ &= (V(\Sigma^T \Sigma + \tau^2 I)V^T)^{-1} V\Sigma^T U^T b = \\ &= V(\Sigma^T \Sigma + \tau^2 I)^{-1} V^T V\Sigma^T U^T b = \\ &= \sum_{i=1}^n \frac{\sigma_i}{\sigma_i^2 + \tau^2} u_i^T b v_i = \\ &= \sum_{i=1}^n \frac{\sigma_i^2}{\sigma_i^2 + \tau^2} \frac{u_i^T b}{\sigma_i} v_i = \\ &= \sum_{i=1}^n f_i \frac{u_i^T b}{\sigma_i} v_i. \end{aligned}$$

Notice that as long as $\tau \ll \sigma_i$ we have $f_i \approx 1$, and if $\tau \gg \sigma_i$ then $f_i \approx 0$. This establishes relation to the SVD solution, and solution (3.2.7) will be approximately equal to (3.1.8) for some $\lambda = \lambda(\tau)$.

If $L = I$ and $\tau > 0$, then the matrix $(A^T A + \tau^2 L^T L)$ in the left-hand side of (3.2.5) becomes symmetric positive definite. Hence, the solution of (3.2.5) can be obtained by Cholesky factorization, a variant of Gaussian elimination that operates on both the left and the right of the matrix at once, preserving and exploiting symmetry [22]. Unlike QR factorization which requires $2mn^2$ flops to solve (3.2.1), Cholesky requires only $n^3/3$. Given that $m \gg n$, we consider Cholesky factorization the method of choice. Even if the performance of Cholesky factorization is relatively fast, one has to keep in mind that Cholesky factorization should be performed for every choice of τ .

Similarly to the SVD, the “L-curve criterion” discussed above can be involved in determination of the value of regularization parameter τ . The idea is to choose τ near the “corner” of “L-curve”, since it represents a compromise between a small residual and a small solution [6].

The non-iterative methods discussed above had significant limitations on the sizes of the input matrix A . In other words, we had to limit the number of lines in the geometry, as well as focus only on the low resolutions. This often happened because non-iterative methods such as TSVD or QR require the storage and use of the full “history” of the previous steps. Moreover, the sparsity of the matrix A was not considered for the solution of $Ax = b$. This dramatically affects the performance of the method.

In addition to that, iterative methods used in this project take advantage of the special structure of matrix A . Moreover, it was often necessary to compute only a few iterations, since the contribution of the following iterations was insignificant. Next, the iterative methods did not have

to remember the history of the preceding iterations; only a few vectors and scalars were sufficient.

An Algorithm for Sparse Linear Equations and Sparse Least Squares (LSQR) [19] is an extremely efficient iterative method for solution of $Ax = b$, as well as the corresponding damped least squares (3.2.3). The method reduces the size of the computation which effectively regularizes the solution.

3.3 LSQR

3.3.1 Lanczos Bidiagonalization

First we use the modified Lanczos bidiagonalization (LBD) process suggested by Golub and Khan [19].

The matrix A is iteratively reduced to lower bidiagonal form:

$$A = U \begin{pmatrix} B \\ 0 \end{pmatrix} V^T, \quad U^T U = I_m, \quad V^T V = I_n, \quad (3.3.1)$$

where $U = (u_1, \dots, u_m)$ and $V = (v_1, \dots, v_n)$ are chosen as products of orthogonal transformations and

$$B = B_n = \begin{pmatrix} \beta_1 & \alpha_1 & & & \\ & \beta_2 & \alpha_2 & & \\ & & \beta_3 & \ddots & \\ & & & \ddots & \alpha_n \\ & & & & \beta_{n+1} \end{pmatrix} \in \mathbb{R}^{(n+1) \times n}. \quad (3.3.2)$$

If we set $U_1 = (u_1, \dots, u_n)$ then we have:

$$AV = U_1 B, \quad A^T U_1 = V B^T. \quad (3.3.3)$$

Equating the j^{th} columns in the two equations and defining $\beta_1 v_0 \equiv 0$ and $\alpha_{n+1} v_{n+1} \equiv 0$ yields:

$$A^T u_j = \beta_j v_{j-1} + \alpha_j v_j, \quad A v_j = \alpha_j u_j + \beta_{j+1} u_{j+1}, \quad j = 1, \dots, n. \quad (3.3.4)$$

Using the orthonormality of vectors u_j and v_j , $j = 1, \dots, n$, and starting with vector $u_1 \in \mathbb{R}^m$, we generate recursively vectors $v_1, u_2, v_2, \dots, u_{m+1}$ and the elements of matrix B_n :

$$r_j = A^T u_j - \beta_j v_{j-1}, \quad \alpha_j = \|r_j\|_2, \quad v_j = r_j / \alpha_j, \quad (3.3.5)$$

$$p_j = A v_j - \alpha_j u_j, \quad \beta_{j+1} = \|p_j\|_2, \quad u_{j+1} = p_j / \beta_{j+1}. \quad (3.3.6)$$

For this bidiagonalization scheme we have:

$$u_j \in \mathbb{K}_j(AA^T, u_1), \quad v_j \in \mathbb{K}_j(A^T A, A^T u_1), \quad (3.3.7)$$

where $\mathbb{K}_k(F, c) = \text{span} \{c, Fc, \dots, F^{k+1}c\}$ is the Krylov subspace for a matrix F and a vector c . Hence v_1, \dots, v_j and u_1, \dots, u_j form an orthogonal basis for these two Krylov spaces [6].

3.3.2 Best Approximate Solutions

Now we consider computing a sequence of approximate solutions to the linear least squares problem:

$$\min_x \|Ax - b\|_2, \quad A \in \mathbb{R}^{m \times n}, \quad m \geq n, \quad (3.3.8)$$

We start the recursion (3.3.5) and (3.3.6) with the vector b :

$$\beta_1 u_1 = b, \quad \alpha_1 v_1 = A^T u_1, \quad (3.3.9)$$

which produces the following iterations:

$$\beta_{j+1} u_{j+1} = A v_j - \alpha_j u_j, \quad (3.3.10)$$

$$\alpha_{j+1} v_{j+1} = A^T u_{j+1} - \beta_{j+1} v_j, \quad (3.3.11)$$

where again elements α_j and β_j are nonnegative and are determined so that $\|u_{j+1}\| = \|v_{j+1}\| = 1$. After k steps we have computed corresponding matrices V_k, U_k , and B_k by the LBD algorithm. We rewrite our recurrence relationships (3.3.10) and (3.3.11) in the matrix form as follows:

$$\beta_1 U_{k+1} e_1 = b, \quad (3.3.12)$$

$$AV_k = U_{k+1} B_k, \quad A^T U_{k+1} = V_k B_k^T + \alpha_{k+1} v_{k+1} e_{k+1}^T, \quad (3.3.13)$$

where e_k is the zero-vector with 1 on the k^{th} position. Since $\mathbb{K}_k = (A^T A, A^T b) = \text{span} \{V_k\}$, the best approximation in the Krylov subspace is given by [6]:

$$x_k = V_k y_k, \quad (3.3.14)$$

where y_k is some vector. Now we consider the multiplication of equation $AV_k = U_{k+1} B_k$ by y_k which yields $Ax_k = U_{k+1} B_k y_k$, and then from (3.3.12) we obtain:

$$r_k = b - Ax_k = U_{k+1} t_{k+1}, \quad t_{k+1} = \beta_1 e_1 - B_k y_k. \quad (3.3.15)$$

Since we want $\|r_k\|$ to be small, and since U_{k+1} and V_{k+1} are bounded and orthogonal in theory, we have to choose y_k to minimize $\|t_k\|$, i.e. consider the least-squares problem:

$$\min_{y_k} \|\beta_1 e_1 - B_k y_k\|_2. \quad (3.3.16)$$

3.3.3 LSQR Algorithm

The problem (3.3.16) forms the basis of LSQR method. It is advantageous to solve (3.3.16) using standard factorization of B_k , i.e. we consider

$$Q_k B_k = \begin{pmatrix} R_k \\ 0 \end{pmatrix}, \quad (3.3.17)$$

where Q_k is orthogonal and R_k is upper bidiagonal matrix consisting of the main and super diagonals [19]. If we consider the matrix $[B_k, \beta_1 e_1]$ then by (3.3.17) we have:

$$Q_k[B_k, \beta_1 e_1] = \begin{pmatrix} R_k & f_k \\ 0 & \psi_{k+1} \end{pmatrix} = \begin{pmatrix} \rho_1 & \theta_2 & & & & | & \psi_1 \\ & \rho_2 & \theta_2 & & & | & \psi_2 \\ & & \ddots & \ddots & & | & \vdots \\ & & & \rho_{k-1} & \theta_k & | & \psi_{k-1} \\ & & & & \rho_k & | & \psi_k \\ - & - & - & - & - & | & \psi_{k+1} \end{pmatrix}, \quad (3.3.18)$$

where Q_k is an orthogonal matrix, a product of plane Givens rotations $Q_k = Q_{k,k+1} \dots Q_{2,3} Q_{1,2}$ designed to eliminate the subdiagonal elements $\beta_2, \dots, \beta_{k+1}$ of B_k . Then we can easily find the vectors y_k and t_{k+1} as follows:

$$R_k y_k = f_k, \quad t_k = Q_k^T \begin{pmatrix} 0 \\ \psi_{k+1} \end{pmatrix}. \quad (3.3.19)$$

However, y_k in (3.3.19) will usually have no elements common with y_{k-1} . In order to make use of the power of iterative technique, we note that $[R_k, f_k]$ is the same as $[R_{k-1}, f_{k-1}]$ with a new row and column added. We combine (3.3.14) and the first equation of (3.3.19), so that:

$$x_k = V_k y_k = V_k R_k^{-1} f_k \equiv D_k f_k, \quad (3.3.20)$$

where d_k , the columns of D_k , can be found successively from the system $R_k^T D_k^T = V_k^T$ by forward substitution. Starting with $x_0 = d_0 = 0$, this yields:

$$d_k = \frac{1}{\rho_k} (v_k - \theta_k d_{k-1}), \quad x_k = x_{k-1} + \psi_k d_k. \quad (3.3.21)$$

It is important to realize that here we have only to save the recent iterate, which reduces our work and storage significantly. Namely, LSQR

requires $3m + 5n$ multiplications and storage of two m -vectors and three n -vectors in order to compute next iterate of x_k [6].

3.3.4 Stopping Criteria

Any iterative algorithm must have rules for deciding whether the current iterate x_k is an acceptable approximation to the true solution x or not. In Paige and Saunders [19] reliable stopping criteria were developed for LSQR method. The stopping rules are formulated in terms of three dimensionless quantities ATOL, BTOL, and CONDLIM, which are specified by the user. We can stop the iterations of the LSQR when the following conditions hold:

$$\|r_k\| \leq \text{BTOL}\|b\| + \text{ATOL}\|A\|\|x_k\|, \quad (3.3.22)$$

$$\frac{\|A^T r_k\|}{\|A\|\|r_k\|} \leq \text{ATOL}, \quad (3.3.23)$$

$$\text{cond}(A) \geq \text{CONDLIM}. \quad (3.3.24)$$

The first two criteria stated in terms of the residual vector $r_k = b - Ax_k$ are based on allowable perturbations in the input data. The quantities ATOL and BTOL are usually selected according to the accuracy in the data. For instance, when A and b are true data, and \tilde{A} and \tilde{b} represent the unknown true values, then

$$\text{ATOL} = \frac{\|A - \tilde{A}\|}{\|A\|},$$

$$\text{BTOL} = \frac{\|b - \tilde{b}\|}{\|b\|}$$

should be used given the available estimates. Since the matrix A in this project was artificially generated, we do not consider the criterion (3.3.23).

The last criterion represents an attempt to regularize an ill-conditioned system. Suppose A has singular values $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n > 0$. As the number of iterations k increases, the established in [19] estimate $\text{cond}(A) \approx \|B_k\|_F \|D_k\|_F$ temporarily levers off near some of the values of the ordered sequence $\frac{\sigma_1}{\sigma_1}, \frac{\sigma_1}{\sigma_2}, \dots, \frac{\sigma_1}{\sigma_n}$ with different number of iterations near each level. This usually happens when there are very small singular values grouped close together; and therefore this suggests the criterion (3.3.24) which controls the condition number as a means of regularizing such problems.

Even if the iterative LSQR algorithm is provides very good approximations to the final reconstruction using only a few iterates and taking into consideration the input matrix sparsity, this method does not give us much information for the analysis of the system (2.6). On the other hand, the TSVD method discussed above shows us which features of the image can be reconstructed, and which are contaminated by noise. This motivates us to consider available iterative singular value decomposition (ISVD) methods for computing certain number of the largest singular triplets (i.e. singular values and corresponding left and right singular vectors).

Similarly to the other methods, employed in this project, there are a number of available optimized computer codes for the computation of ISVD. SVDPACKC [5], a computer implementation of an ISVD using Lanczos, Lanczos block, subspace iteration, and trace minimization methods was considered in this project. One of the main advantages of SVDPACKC is the ability to use the input matrix A in the sparse format, which releases restrictions on the usage of a large number of sensors in the geometry.

3.4 The Iterative Singular Value Decomposition

3.4.1 Equivalent Eigenvalue Problems

Associated with an $m \times n$ matrix A is the symmetric $(m+n) \times (m+n)$ matrix

$$B = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}. \quad (3.4.1)$$

Assuming that matrix A is of full column rank, the eigenvalues of B are the n pairs, $\pm\sigma_i$ with $(m-n)$ additional zero eigenvalues, since $m > n$.

Given the eigenvalues and eigenvectors of B , we can generate the SVD of A . First, for all positive eigenvalue σ_i of B let $(u_i, v_i)^T$ denote the corresponding eigenvector of length $\sqrt{2}$. Then σ_i is a singular value of A and u_i, v_i are respectively left and right singular vectors of A corresponding to that singular value. If B has corresponding orthogonal eigenvectors $(u_i, v_i)^T$ to the zero eigenvalues σ_i with $u_i \neq 0, v_i \neq 0$, then 0 is the singular value of the matrix A , and the corresponding left and right singular vectors can be obtained by orthogonalizing these u_i and v_i respectively. Otherwise, matrix has a full column rank.

Alternatively, one can compute SVD of A indirectly by the eigenpairs of either $n \times n$ matrix $A^T A$ or the $m \times m$ matrix AA^T . If $V = (v_1, \dots, v_n)$ is a matrix of eigenvectors of $A^T A$, so that

$V^T(A^T A)V = \text{diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_r^2, 0, \dots, 0)$ (where r is the rank of A), the σ_i is the i^{th} nonzero singular value of A corresponding to the right singular vector v_i and the left singular vector $u_i = \frac{1}{\sigma_i} A v_i$. One has to carefully consider the

resulting singular values σ_i at this point to make sure that they are not too small to cause information loss in u_i . Similarly we can consider the way to get

singular triplets from the eigendecomposition of AA^T . However, one has to keep in mind that for our problem $m \gg n$, so that the second way is much slower, since it requires the eigendecomposition of the $m \times m$ matrix.

It is important to emphasize that computing the singular values of A via the eigensystems of either AA^T or $A^T A$ may be sufficient to the computation of the large singular triplets of A , but often causes loss of accuracy for the small ones. This happens since the conditioning of the new matrix either AA^T or $A^T A$ becomes in general much worse than the original matrix A .

3.4.2 Subspace Iteration Method

Subspace iterations is the conceptually simplest algorithm used to solve large sparse eigenvalue problems. Adapted to the matrix B (3.4.1) this method would involve forming the sequence:

$$Z_k = B^k Z_0, \quad (3.4.2)$$

where $Z_0 = (z_1, z_2, \dots, z_s)$ is an $(m+n) \times s$ matrix (for some value of s). If the column vectors z_i are normalized separately, then these vectors will converge to the largest eigenvector of B . In order to approximate p largest eigenpairs of B , the columns z_i should be orthogonalized at each step to maintain linear independence. The resulting convergence will be linear, which can be improved by Rayleigh-Ritz procedure, as well as by Tchebychev polynomials as discussed in [5].

The primary cost of the described optimized procedure lies in the total number of sparse matrix-vector multiplications required. If s vectors z_i are

used to approximate p largest eigenvectors of the $(m+n) \times (m+n)$ matrix B with $s \geq p$, the cost in floating-point operations per one iteration would be

$$s \times [2(1 + \mu_r)m + 2(1 + \mu_c)n], \quad (3.4.3)$$

where μ_r and μ_c are the row and column sparsity indices respectively, representing the average number of nonzeros per row and column.

3.4.3 Trace Minimization Method

Another method for the iterative SVD of sparse matrices is based on the trace minimization algorithm for the generalized eigenvalue problem

$$Hx = \lambda Gx, \quad (3.4.4)$$

where H and G are symmetric and G is also positive definite. For the computation of the SVD of matrix A we either replace H with $\tilde{B} = B + \gamma I$, where γ is chosen to so that \tilde{B} is positive definite, or replace H with $A^T A$ if we want to consider the solution of the second formulation of the eigenproblem. Next, we define $G = I_{m+n}$ or $G = I_n$ respectively. Without loss of generality, assume that we are solving the first formulation of the eigenproblem, and define $H = \tilde{B}$ and $G = I_{m+n}$. The trace minimization SVD scheme is based on the following result [5]:

Theorem. If \mathbb{Y} is the set of all $(m+n) \times p$ matrices Y for which $Y^T Y = I_p$, then:

$$\min_{Y \in \mathbb{Y}} \text{trace}(Y^T \tilde{B} Y) = p\gamma - \sum_{i=1}^p \sigma_i, \quad (3.4.5)$$

where σ_i is a singular value of A , $\lambda_i = \gamma \pm \sigma_i$ is an eigenvalue of \tilde{B} , and $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p$.

orthonormal $(m + n)$ vector v_1 . Now recursively define matrices T_j (3.4.7).

Namely, we start with $\beta_1 \equiv 0$, and $v_0 \equiv 0$. Then for $i = 1, 2, \dots, l$ define

Lanczos vectors w_i and diagonal elements of T_j as follows:

$$\beta_{i+1}w_{i+1} = Bw_i - \alpha_iw_i - \beta_iw_{i-1}, \quad (3.4.8)$$

$$\alpha_i = w_i^T(Bw_i - \beta_iw_{i-1}), \quad (3.4.9)$$

$$|\beta_{i+1}| = \|Bw_i - \alpha_iw_i - \beta_iw_{i-1}\|_2. \quad (3.4.10)$$

By definition, the vectors α_iw_i and β_iw_{i-1} are respectively, the orthogonal projections of Bw_i onto the corresponding w_i and w_{i-1} . For every i the next Lanczos vector w_{i+1} is obtained by orthogonalizing the projection Bw_i with respect to w_i and w_{i-1} . The resulting coefficients define the Lanczos matrix (3.4.7). If we define as an $n \times j$ matrix consisting of Lanczos vectors

$W = [w_1, \dots, w_j]$, then we can state the described process in the matrix form:

$$BW_j = W_jT_j + \beta_{j+1}w_{j+1}e_j^T. \quad (3.4.11)$$

Given the matrices T_j we compute its relevant eigenvalues, which is equivalent to computing the best approximations to the eigenvalues and eigenvectors of B restricted to the corresponding Krylov subspaces. Next, we select some or all of these eigenvalues as approximations to the eigenvalues of the matrix B , and hence the singular values of A . Finally, for each eigenvalue λ we compute a corresponding unit eigenvector z such that $T_k z = \lambda z$, and map such vector into corresponding Ritz vectors $y \equiv W_q z$ [5], which are then used as approximations of the desired singular vectors of the matrix A .

In finite precision arithmetic, Lanczos procedure often fails due to the loss of orthogonality of the Lanczos vectors w_i . One of the best remedies to

this problem is to consider selective reorthogonalization strategy which was used in this project. Then, no redundant copies of eigenvectors are computed; a posteriori error bounds and estimates cost almost nothing and are used in order to stop the iterations as soon as possible; and finally multiple eigenvalues and their eigenvectors are found naturally [20].

3.4.5 Lanczos Block Method

As an alternative to the Lanczos method described above, we can compute ISVD using the block Lanczos method based on LBD algorithm applied to the matrix B (3.4.1) followed by the QR factorization. The description of this method goes beyond this thesis, however its outline can be found in [5] and [6].

Chapter 4

Computer Implementations

In order to perform the reconstructions using discussed methods, a comprehensive software package was designed, developed, and tested on both modelled and laboratory data. The packages emphasizes a number of quality attributes, both visible at runtime, as well as not discernable during program execution. Namely, such attributes as performance, efficiency, functionality, usability, memorability, as well as error avoidance were considered. Moreover, other issues such as modifiability, portability, integrability, and reusability were also taken into account while developing the software.

The package consists of three main applications - LGEN, LINE2MAT^{4.1}, and INVERT^{4.2}. LGEN application represents the first stage of the engineering experiment - the design of the sensor system - yielding the geometry profile. LINE2MAT generates the matrices A and $W^{1/2}$ for the specific geometry file created by LGEN; computes the SVD and forms the matrix $A^T A$ intended to be used in INVERT; generates simulated data; and allows the visualization of some output results. Finally, INVERT is responsible for the interactive image

^{4.1}The initial IRIX version of this software designed for TSVD method was developed by Steven H. Izen.

^{4.2}The initial IRIX/ViewKit version of this software implementing TSVD reconstruction was developed by Steven H. Izen.

reconstruction, implementing the discussed numerical methods.

The complete user and developer manuals of the software package are included in the appendix.

4.1 LGEN Overview

This application is responsible for the generation of lines from the sources to detectors for the certain geometries as discussed in chapter on the mathematical model.

Given the locations of sources and detectors implemented as the objects organized in the double circular linked lists, LGEN considers other parameters of the objects, such as angle of acceptance, angle of orientation, and activity in order to generate a line connecting a source with a detector. The user can add lines manually, change properties of the particular object, as well as global properties of the geometry as discussed in the user manual. The output of LGEN is the ASCII file representing lines from sources to detectors by coordinates on the frame. Each line is defined by four double precision numbers - two coordinates for the source coordinate, and two coordinates for the detector coordinates. The lines can be either sorted by numbers of sources, numbers of detectors, distances to the geometrical center of the frame, or angles formed by the normal to the given line and the horizontal line. This leads to different matrices in LINE2MAT.

In addition to project-specific features, LGEN possesses all features of the standard graphics user application of this type (See Figure 4.1). Moreover, LGEN provides an opportunity to save the geometry view in multiple output

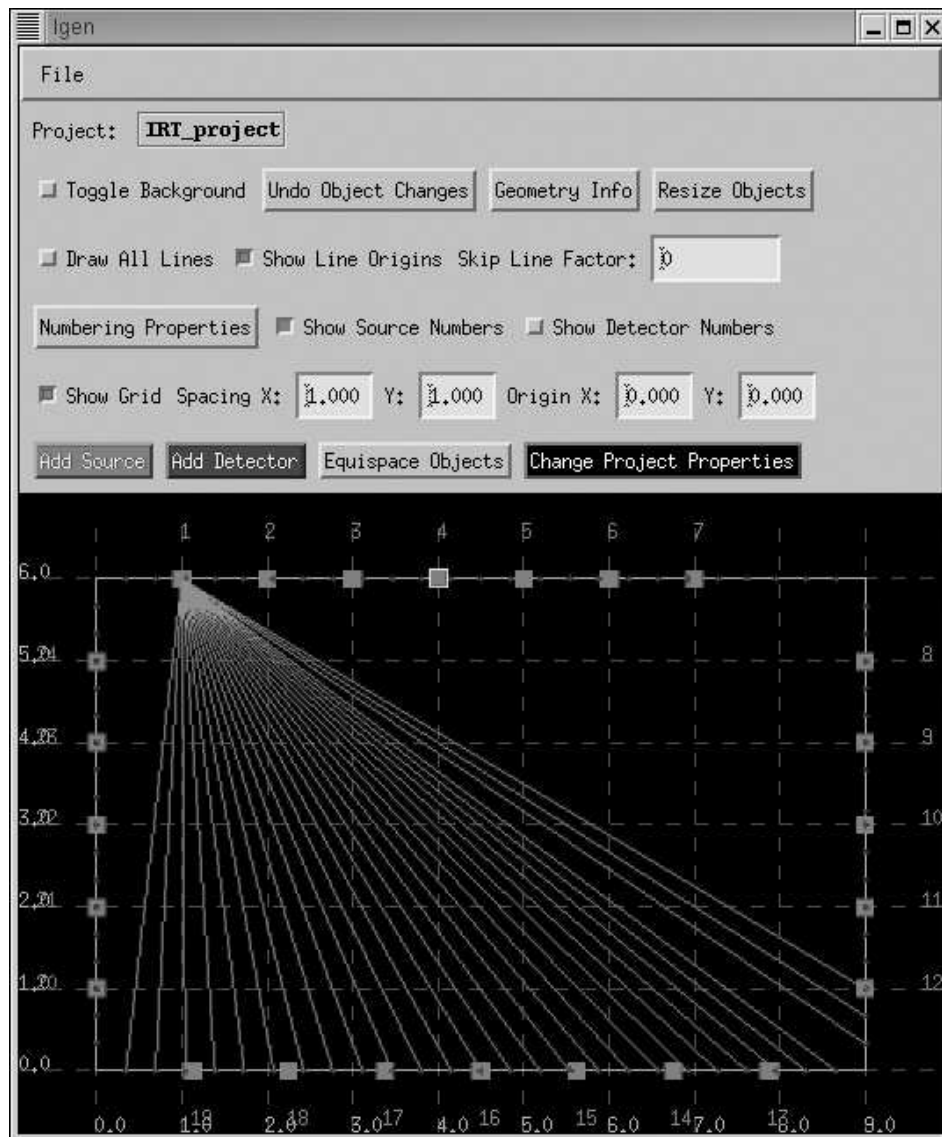


Figure 4.1: LGEN in action: Geometry project view.

formats.

4.2 LINE2MAT Overview

This program is mainly responsible for the model setup and other time-consuming computations. For integrability concerns LINE2MAT is not

supplied with graphical user interface. However, a small utility - `LINE2MAT_INTERFACE` - included in this package provides simple interface for working with `LINE2MAT`.

First of all, given the lines file - the output of `LGEN`, `LINE2MAT` generates the matrix A as described in the chapter on the mathematical model. The algorithm for generation of matrix A requires special consideration of the cases when the line is either vertical, horizontal, passes through the common point of the neighboring pixels, or has a length of intersection with a pixel of the order of machine epsilon.

The matrix A can be stored in binary compressed format. `LINE2MAT` also provides user an option to save matrix A in the sparse matrix format. This scheme is more beneficial for this project, since A is extremely sparse (on average it has 95 – 99.5% zero elements), and the usage of its dense version leads to inefficient memory and disk space management, and affects the performance dramatically. See appendix C for the description of the matrix sparse storage formats.

`LINE2MAT` also computes and saves the matrix $A^T A$ which will be later used by `INVERT` for the Tikhonov regularization. Given that A is an $m \times n$ matrix, we need to compute n^2 elements, each of which will cost $2m - 1$ floating point operation. In other words, the total complexity will be approximately $2mn^2$ flops. The straightforward loop-based formation of $A^T A$ is not efficient even after the compiler optimization. In order to form the matrix $A^T A$ very quickly (up to 10 times faster than using loops in the program), one should consider the LAPACK [15] package, which offers a

routine DDOT for vector-vector multiplication. The optimized versions of this routine perform computations at assembler level. In addition, if we take into consideration that the matrix $A^T A$ is symmetric. Hence, instead of computing of n^2 elements we have to compute only $n^2/2$ elements.

Moreover, in order to generate model data LINE2MAT can perform projections from a user specified TIFF image or ASCII data file treated as a vector d of n elements. Then, the projection is computed as $b = Ad$, or $b = W^{1/2}Ad$ if weighting is used. The computation of projection takes advantage of the sparse matrix format and uses templates offered by SparseLib++ [2] to perform matrix-vector multiplication efficiently. This projection serves as a right-hand side b in the system (2.6) when laboratory data is not available.

LINE2MAT also allows the user to save the diagonal matrix $W^{1/2} = \{w_{ii}^{1/2}\}$ defined in (2.8) as discussed in chapter on the mathematical model. Because of the loss of accuracy caused by division, we save the elements $\frac{1}{w_{ii}^{1/2}} = \sqrt{\sum_{j=1}^m (\int_{l_i} \Phi_j ds)^2}$ and then compute their reciprocals.

Finally, LINE2MAT computes reduced singular value decomposition of the matrix A as discussed below in details. The corresponding matrices U, V , and Σ (such that $A = U\Sigma V^T$) are computed and saved in binary compressed formats (U^T and V) and in ASCII format (Σ) if no weighting in the matrix A is used. However, if weighting is incorporated in the elements of A , instead of saving U^T we save $U^T W^{1/2}$. Then when we perform the reconstruction in INVERT, we do not have to access the matrix W , since if $W^{1/2}A = U\Sigma V^T$ then the SVD solution of (2.7) will be given as

$$x = (W^{1/2}A)^\dagger W^{1/2}b = V \begin{pmatrix} \Sigma_1^{-1} & 0 \\ 0 & 0 \end{pmatrix} U^T W^{1/2}b.$$

We do not take full advantage from the compression scheme here, since both matrices U and V are dense and we save only diagonal elements of Σ .

LINE2MAT is also supplied with an internal matrix viewer which can display matrices A , $A^T A$, and selected right singular vectors on the screen (using a specified magnification) in grayscale with an option to change the width and center parameters used for the formation of the raster image. In addition, LINE2MAT allows the user to save $A^T A$, as well as right singular vectors, in compressed TIFF image format.

4.3 INVERT Overview

This application is responsible for the image reconstruction. It implements three mathematical methods discussed in the previous chapter. However, the iterative singular decomposition for sparse matrices is not present in this application due to the inefficient implementation of this algorithm in the SVDPACKC [5] package, which is considered to be the most popular computer library for ISVD, as discussed below.

INVERT possesses three computational engines - TSVD, Tikhonov Regularization, and LSQR - responsible for the reconstruction of the original image. Each computational engine is implemented as an object and is supplied with algorithm specific inputs and returns reconstruction outputs. The major output of all engines is the reconstructed image, i.e. the unknown vector x in the system (2.6).

One of the inputs of INVERT common to all computational engines, the

right-hand side data vector b represents the samples of (1.1) - either laboratory or simulated data. However, the user can also supply the intensity I data, and `INVERT` will convert it to the form (1.1) where I_0 is either specified by user or selected as the maximum intensity I in the intensity data. Moreover, `INVERT` incorporates the byte-swapping option for reading binary data files from different platforms. This makes the application more portable.

The computational engines communicate with the graphical user interface in order to represent the reconstruction results and to allow the user to interactively manipulate input parameters of the methods. The engine specific inputs are incorporated in the user interface of `INVERT`. In order to make the reconstruction process clear to the user, each engine possesses a single “view”; whereas all of them has a common fields, such as the graphics area for the reconstructed image representation (See Figure 4.2).

4.3.1 TSVD in `INVERT`

In order to use TSVD for the image reconstruction in `INVERT`, the user has to specify all the components of the reduced singular value decomposition computed in `LINE2MAT`. Namely, as the inputs for TSVD engine, the user supplies matrices U^T (or U^TW in case of weighted system matrix), V , and singular values stored in ASCII format. Moreover, like for other engines, the user supplies right-hand side data vector b .

The user can specify how many singular triplets should be used for the reconstruction as in (3.1.8). The L-curve criterion is used to get an estimate of the optimal number of singular triplets. The user can also add noise and consider its effect on the solution interactively changing the number of triples

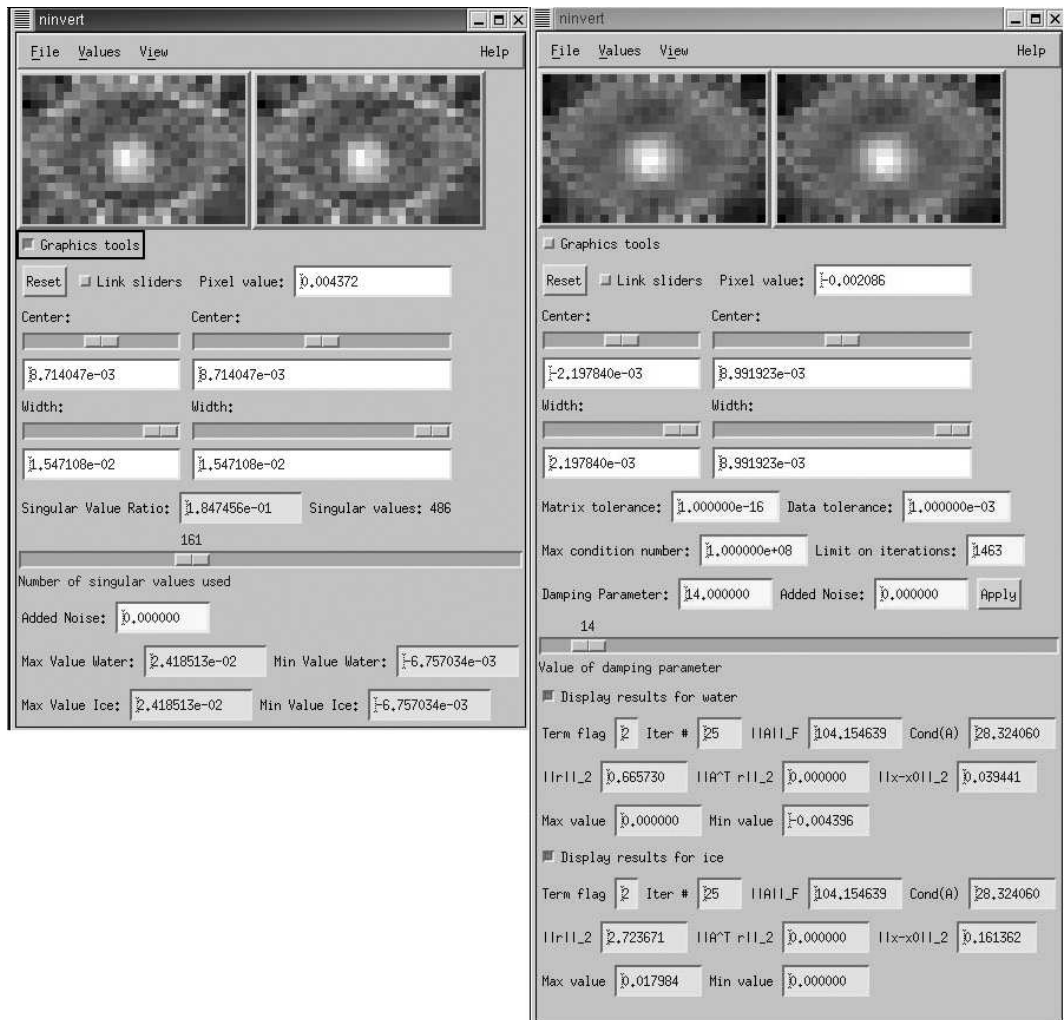


Figure 4.2: INVERT in action: Multiple views for computational engines. Reconstruction via SVD [left]. Reconstruction via LSQR [right].

used for the reconstruction. At the same time the resulting image is displayed in the graphics area in grayscale.

The solution's minimal and maximum values are displayed on the screen. If there exists a solution prototype, the user can compare the obtained solution with it. The residual, defined as a 2-norm of the difference between vectors, will be displayed on the screen then.

Finally, the user can consider individual pixel values and change solution visualization parameters interactively, adjusting center and width of the grayscale image, which can be saved in TIFF image format.

4.3.2 Tikhonov Regularization in INVERT

The implementation of Tikhonov regularization requires the matrix A to be in dense format (stored as compressed binaries). The weighting matrix W generated by `LINE2MAT` can be also specified. The user can either input the matrix $A^T A$ or compute it during the initialization of the Tikhonov regularization computation engine. As for TSVD, the user is required to specify the right-hand side data vector b . In addition to that, the initial regularization parameter $\tau \geq 0$ as in (3.2.1) should be selected.

The choice of τ is crucial, since we do require that the matrix $(A^T A + \tau^2 L^T L)$ in the alternative formulation of Tikhonov regularization (3.2.5) is positive definite in order to be factored by Cholesky. However, if we select $L = I$ and some very small τ and consider the ill-conditioned matrix A , then the matrix $A^T A + \tau^2 I$ may not necessary be positive definite, since $A^T A$ may be rank-deficient. Hence we need to select τ to make the resulting matrix of the full rank. Given that $A^T A + \tau^2 I$ is symmetric and does not have

negative elements, Cholesky factorization will solve the Tikhonov regularization problem successfully.

The user interface allows to change τ^2 interactively from 0 to the maximum eigenvalue λ_{max} of the matrix A , which is determined iteratively using the following algorithm (starting with some vector x and scalar λ_{old}):

while $|\lambda_{max} - \lambda_{old}| > \text{precision}$

$$\begin{aligned}\lambda_{old} &= \lambda_{max} \\ \lambda_{max} &= \frac{\|Ax\|}{\|x\|} \\ x &= \frac{Ax}{\|Ax\|}.\end{aligned}$$

There is no need to use larger τ since the reconstructed image would not visually change for the larger values of τ and will represent the effect of the first several singular triplets of the singular value decomposition.

Like for TSVD, the user can add noise and consider its effect on the solution interactively changing the value of τ used for the reconstruction. At the same time the resulting image is displayed in the graphics area in grayscale.

The implementation of Tikhonov regularization (see below) provides the estimates of the backward and forward errors in the reconstruction, as well as the estimate of the condition number of the matrix $A^T A + \tau^2 I$. This information is displayed on the screen, since it helps to make decisions about the accuracy of the obtained solutions.

The solution minimum, maximum values, as well as residual are also displayed on the screen as in TSVD realization. Likewise, the user can take

advantage of changing image visualization parameters and export the image to TIFF image format.

4.3.3 LSQR in INVERT

The user is required to specify the matrix A in compressed column storage format (see (10.5), (10.6), and (10.7)) and the right-hand side data vector b . The weighting matrix W is an optional parameter.

If the regularization parameter τ is not specified, then INVERT performs reconstruction and display of results solving the system (2.6) in the least squares sense. Otherwise, INVERT solves the damped least squares problem (3.2.3) which makes the solution by components less sensitive to the changes in data.

The user can change the regularization parameter interactively in the range from 0 (no regularization) to the largest singular value of A which is equal to the 2-norm of the matrix. The 2-norm of the matrix is bounded by the Frobenius norm which is approximated as $\sqrt{n_z(\max_{i=1,\dots,m} a_{i1})^2}$, where n_z is the number of nonzero elements in the matrix. Like in Tikhonov regularization, there is no need to specify a larger τ , since after some large value of the regularization parameter, the reconstructed image will reproduce the first singular triplets corresponding to the largest singular values.

Similarly as to TSVD and Tikhonov, the user can add noise and consider its effect on the solution, while changing the value of τ interactively. The resulting image is displayed in the graphics area.

The implementation of LSQR (see below) requires the user to specify stopping criteria on the iterations. The three stopping criteria - maximum

matrix tolerance, maximum vector tolerance, and maximum matrix condition number (see discussion of stopping criteria for LSQR) are used to stop LSQR iterates. Moreover, the restriction on the maximum number of iterations in LSQR is also considered a stopping criterion. When LSQR encounters any of the four criteria discussed above, it terminates.

The solution minimum, maximum values, as well as residual are displayed on the screen as in Tikhonov regularization and TSVD. Moreover, an LSQR termination flag (pointing to the stopping criteria used), number of the last iterate, matrix Frobenius norm $\|A\|_F$, condition number of the matrix, 2-norm of the residual $\|r\|_2 = \|Ax - b\|_2$, norm of the residual projection $\|A^T r\|_2$, as well as difference between the initial guess and last iterate $\|x - x_{initial}\|_2$ are displayed after the computation is complete (i.e. LSQR is terminated).

Similarly to other methods, the user can change the visualization parameters of the reconstructed image and import it to TIFF image format.

4.4 Truncated SVD Implementation

Given the LGEN lines file, LINE2MAT generates the matrix A and computes its singular value decomposition using the following steps. First of all, if the user requires the computation of the SVD of the weighted matrix, the matrix A is multiplied by diagonal weighing matrix W , and the result is stored in the matrix A workspace. The resulting matrix is represented by a double precision array is transposed to meet the input matrix requirements of LAPACK [3] manifested by its Fortran background. Next, LAPACK performs the singular

value decomposition using the routine `DGESVD`. The detailed description of the singular value decomposition implementation in LAPACK is presented in [9]. The resulting singular values representing diagonal elements of Σ are dumped to an ASCII file. The double precision array representing right singular vectors is transposed (due to Fortran array indexing we have now matrix V) in order to save some computational time in `INVERT`, and is saved in compressed binary format. The left singular vectors are also saved in the file without transposition (due to Fortran array indexing we have now matrix U^T). In order to measure the accuracy of the computed SVD, the dot-product of the first left singular vector with the second one is displayed on the screen. This helps user to estimate the loss of orthogonality, an effect of computing the SVD in finite precision.

`INVERT` loads the matrices U^T , V , and Σ and performs the reconstruction using equation (3.1.8) based on the required number of triplets involved in the computation specified by the user. In order to speed up computation of (3.1.8), BLAS's functions for matrix-vector and vector-vector multiplication were employed.

It is important to underline that the restrictions on sizes of sensors geometry are mainly caused by memory limitations of the `DGESVD` routine. For a general matrix, `DGESVD` requires approximately $m^2 + n^2$ memory locations to store double precision elements. If the size of each elements is 8 bytes which is common on most platform, then `DGESVD` requires $8m^2 + 8n^2$. Since $m \gg n$, we are mostly concerned about the term $8m^2$. If there are approximately 12,000 lines in the geometry, which is typical for large sensor systems,

$8m^2 = 1,152,000,000$ bytes. In other words we need at least 1 Gb memory. Similarly, when we write results of the SVD on the disk, U^T will occupy roughly 1 Gb. Even for modern computers these numbers are large and it takes hours to do the singular value decomposition; hence due to the necessity to repeat the experiments over the short time, we try to avoid such resource usage.

However, there is one big advantage of using LAPACK's SVD implementation. The scalable versions of LAPACK, which are widely used for the IRIX platform simply require a single line of compiler directives in the code to perform part of the SVD in parallel. This saves tremendous amount of programming work required for the processors communication.

4.5 Tikhonov Regularization Implementation

The lines file of LGEN is used to generate the matrix A in LINE2MAT. As discussed above, the user can also generate a weighting matrix W and a dense matrix $A^T A$ efficiently in LINE2MAT. The matrix W is stored in ASCII format. The other two matrices are stored in compressed binary formats. While the savings of using compressed formats for A are significant, the matrix $A^T A$ does not yield any savings because of its high density.

The matrices are loaded in the Tikhonov regularization computational engine in INVERT. If the matrix $A^T A$ was not precomputed, INVERT can compute it. Next, the factor $\tau^2 I$ is added to the matrix to ensure positive definiteness. The data right-hand b side is multiplied by the transpose of matrix A to meet the setup requirements of equation (3.2.5). Notice, that like

in the TSVD implementation all matrix-vector and vector-vector products are computed using BLAS routines to increase software performance. Next, the LAPACK's routine `DPOSVX` solves (3.2.5) using Cholesky factorization to obtain the reconstructed image for each regularization parameter. A good discussion of Cholesky factorization implementation can be found in [22] and [8].

Since Cholesky factorization requires only of the order of n^2 memory locations, we pretty much do not have any computer resources restrictions working with this computational engine. The only situation when we may encounter memory limitations is extremely high resolution images. For instance, if the total number of pixels in the reconstructed region is more than 10^8 (i.e. the image is larger than $10^4 \times 10^4$) we may run out of memory on machines with only 1 Gb of RAM. The time of computation is also not in issue in Tikhonov regularization since only approximately $n^3/3$ floating point operations are needed to do the reconstruction. This translates into seconds of computer time which is significantly faster than the hours required to do the SVD.

Some scalable implementations of LAPACK allows to perform Tikhonov regularization in parallel. This saves much programming work. Moreover, the routine `DPOSVX` has an option to solve (3.2.5) for multiple right-hand sides at the same time. This turns out to be more efficient in terms of computer resources compared to the repeated usage of `DPOSVX`.

4.6 LSQR Implementation

The lines file output of LGEN is used to generate the sparse matrix A using the compressed row storage scheme outlined above. Again the user can generate the weighting matrix W for later use in INVERT.

INVERT loads the matrices, as well as the right-hand side b in the LSQR computational engine. After necessary memory allocation steps and setup of LSQR stopping criteria values, the LSQR C implementation by Paige and Saunders [18] is used to solve either (2.6) or (3.2.3) for the specified regularization parameter. The description of LSQR method and the algorithm overview are given in the previous chapter, while the exact algorithm is presented in [6] and [18].

The C implementation of LSQR does not require direct access to the matrix A . INVERT supplies only routines for the computation of vector updates $y = y + Ax$ along with $x = x + A^T y$ for some vectors x and y . Given that A is in the sparse format, we take full advantage of the package SparseLib++ [16]. Namely, instead of operating on elements of A directly, we count on C++ STL (Standard Template Library) to simplify programming work and improve overall efficiency.

The memory advantages of using LSQR method are discussed in the previous chapter. The main idea is the largest storage of memory is used for nonzero elements. The computational time is limited by the number of iterations. In general we do not have to wait more than a minute to obtain reasonable reconstructions even for a high resolution.

Unfortunately, unlike routines for computing TSVD and Tikhonov

regularization in LAPACK, existing LSQR implementations are not designed to perform scalable computations easily. One of the major obstacles in making LSQR completely scalable is the dot product whose execution time is independent of the size of the block [4]. Finally, we need to go through algorithm again in order to solve our system for another right-hand side.

4.7 Inefficient Implementation of Sparse Iterative SVD

The SVDPACKC [5] library consists of six ANSI C routines - SI1, SI2, LA1, LA2, BL1, BL2, TM1, and TM2. Each routine represents a certain method for the iterative computation of singular value decomposition of a sparse matrix. SI1 performs SVD via subspace iterations using 2-cyclic matrices; SI2 performs SVD via subspace iteration using $A^T A$ eigenvalues; LA1 performs SVD via single-vector Lanczos algorithm with selective re-orthogonalization operating on matrix $B = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$; LA2 performs SVD via single-vector Lanczos algorithm with selective re-orthogonalization operating on matrix $A^T A$; BL1 performs SVD via hybrid block Lanczos procedure for equivalent 2-cyclic eigensystems; BL2 performs SVD via hybrid block Lanczos procedure for eigensystems of the form $A^T A$; TM1 performs SVD via trace minimization using 2-cyclic matrices; and finally TM2 performs SVD via trace minimization using $A^T A$ eigensystems. The outline of methods is given in the previous chapter, while the detailed description can be found at [5].

Moreover, the authors of SVDPACKC [5] present the comparison analysis of the methods in terms of CPU time based on several test problems of various sizes. According to that result, LA2 works much faster (at least

several times) than other methods for all of the test problems. The similar comparison of robustness and memory usage applied to about 20 wind tunnel systems of various systems of the type (2.6) showed that LA2 performs much better than other methods in SVDPACKC. Moreover, other methods performed even slower and required more resources than the non-iterative SVD implemented in LAPACK [3]. This served a motivation to consider only LA2 in this project.

Since the source code for LA2 is available, we considered it necessary to examine this implementation from the programming perspective before deciding whether to use this as the computational engine in INVERT. At first glance the code is well structured in terms of method subdivision. However, the basic algorithms (such as Lanczos Bidiagonalization) and elementary operations (such as matrix-vector multiplication) are put together in the same file. This makes difficult to proofread the code and follow the algorithm. A better implementation should separate drivers from algorithms and elementary operations methods.

In addition to imperfect structure of the code, the vendors of SVDPACKC re-implement BLAS routines in a straightforward manner using ANSI C loops in the source code. Even though it allows the SVDPACKC users to avoid the use of the BLAS library, the efficiency of computation decreases significantly. A better implementation should take advantage of BLAS routines. This would decrease the computational time.

Finally, one can easily trace several minor memory leaks throughout the source code which cause instability and can crash the computation. Since

the SVDPACKC package was designed to work with large systems, it is understood that the users can run experiments for several hours for their systems. The users can easily lose the results of the prior computation due to a memory leak, which is unacceptable from a business perspective.

The issues of code structure, usage of inefficient elementary operations routines, as well as memory leaks were fixed in the LA2 source code yielding a new program, myLA2. All further experiments were done using the new program.

Given the geometry yielding a matrix A with 12170 lines and 1944 pixels we compute both iterative sparse (LA2) and non-iterative dense SVD in order to compare the performance.

The dense version of matrix A occupies 185 Mb (given that each element is 8 bytes long). The computation of a dense SVD requires 324 Mb memory for the full decomposition (including 185 Mb of dense matrix storage). It takes roughly 40 minutes on a Pentium-IV machine.

However, the matrix A has only about 580,000 nonzero elements. Hence, the sparse version of it would require approximately 5 Mb for element storage. LA2 allocates and uses 287 Mb for the SVD computation. This means that LA2 probably does not really take advantage of the matrix sparsity or does not efficiently handle input/output of Lanczos iterates. Moreover, it takes about 30 minutes to do the decomposition. These savings compared to the non-iterative implementation SVD are insignificant and do not reflect the actual benefits of computing sparse iterative SVD.

Finally, LA2 fails to compute SVD accurately enough. One way to

show this is to compare the large singular values computed by non-iterative dense SVD implementation (with the positive validation history) with those computed by LA2. For the small systems similar to those described in [5] the results are exactly the same, while for the large systems the singular values differ by up to 2 percent. Most importantly, the computed right singular vectors in LA2 suffer from a significant loss of orthogonality. This demonstrates that the results of LA2 are unreliable, and the method as-is should not be incorporated in the project software package.

Chapter 5

Programming and Design Considerations

Since the software development played a crucial role in this project, it is important to consider the reconstruction package software design and implementation, as well as the motivation for the current choice of solutions.

Initially the software was developed for the SGI IRIX platform and X-Windows system (in order to provide a graphical user interface). Later the software was ported to the Linux platform. For this reason, one of the most important considerations during the design and development was portability.

5.1 Choice of Compiler

The MIPS C/C++ compiler available on IRIX systems offers powerful code optimization. Namely, this compiler, as well as other EPI compilers, employs advanced optimization technology that aggressively transforms compiled programs to minimize the number of computations when these programs are running on a target RISC processor. Most EPI compilers fully utilize those properties of RISC processors that are crucial for obtaining the highest possible performance, such as multi-stage pipelines, multiple issue of instructions, large number of hardware registers, and multiple functional units.

Unfortunately, EPI compilers are not available for the Intel

microprocessor architectures used on the Linux platform. Therefore, we considered the GNU Compiler Collection, a part of FSP (Free Software Project) that is available for multiple system architectures and platforms. However, in exchange for portability we lose the powerful code optimization.

5.2 Choice of Libraries

All libraries used for this project are available for both IRIX and Linux platforms in order to make the application package portable. The list of libraries used is presented in the developer manual.

5.2.1 Scientific Computations Library

Another drawback of portability was the choice of version of scientific library that includes LAPACK and BLAS. The SGI Scientific Computing Software Library (SCSL) includes LAPACK, BLAS, and signal processing libraries as well as sparse direct solvers. The available version of SCSL for IRIX operating system allows one to do scientific computations in parallel on the multiple-processor machines. This saves significant execution time.

On the other hand, a parallelized SCSL is not available on Linux platforms. This definitely effects the computational speed, but the effect is not significant since many algorithms are not completely parallelizable. For instance, only the bidiagonalization part of the SVD implemented in SCSL LAPACK is parallelized. Moreover, it is not necessary to parallelize the iterative methods considered in this project due to their fast performance. For instance, as described in the chapter on numerical experiments, only a few iterations are needed to obtain a fair reconstruction, even for the large systems.

It takes seconds to do it. Large efforts for parallelization are unnecessary.

5.2.2 X-Windows Graphical User Interface Library

Motif, the standard graphical user interface API libraries incorporating a widget set for X-Windows platforms, was used in the software package. A good discussion of software development using Motif can be found in [12] and [24].

Initially, `INVERT` was developed using `ViewKit`, a proprietary application framework for Motif which simplifies development in the Motif environment. However, `ViewKit` is not a standard library on most systems, and its use would make the installation of the reconstruction software package inconvenient. Moreover, `ViewKit` is not generally available for Linux platforms. Finally, `ViewKit` is in some sense equivalent to MFC (Microsoft Foundation Classes) and restricts control of many essential aspects of the application. For these reasons, `INVERT` was later converted to “pure” Motif, and later graphical user interface development was done in the Motif environment only.

The choice of Motif implementation was also crucial while developing the software. For a long time, the development was done on the IRIX platform using commercial SGI Motif. When the application was ported to Linux, the freely available and widely-used `LessTif` libraries were used instead of Motif. However, we found that `LessTif` cannot be completely integrated with the OpenGL graphics library (see below). Therefore, a recently released version of Motif, `OpenMotif`, was used in all graphical user interface applications.

Finally, it is important to emphasize that, in general, conversion from one version of Motif to another one was not too painful. All versions of Motif

are attempts to be clones of the original Motif.

5.2.3 Computer Graphics Libraries

Compared to the experience the with graphical user interface API library, the choice of a computer graphics library was not problematic at all. OpenGL, a freely available and portable library was used on both IRIX and Linux platforms to do computer graphics within the drawing area widget. Moreover, the hardware optimized to OpenGL is readily available and cheap. OpenGL offers numerous opportunities for using mathematics in the field of computer graphics. An excellent comprehensive discussion of programming using OpenGL can be found in [23].

Another graphics library extensively used in this project is the TIFF image library. This is a portable library which allows the programmer to code file input and output operations using the TIFF image format. The library offers multiple compression schemes used in the project to save disk space occupied by the TIFF images.

5.2.4 Data Manipulation Libraries

In order to store and retrieve compressed data, the portable ZIP compression library [10] was used. ZIP compression library implements the most efficient and widely-used “zip” compression scheme which can be applied to any data type.

Another portable library, SparseLib++ [16] (C++, Standard Template Library based) offers an interface for reading, saving, and manipulating sparse data in various compressed formats. Moreover, the standard package

SparseLib++ includes the sparse implementation of BLAS, which saves significantly computation time and computer resources.

5.3 Separation of Interface from Implementation

One of the most important concerns while developing this software package was the separation of interface from implementation. Programming in this manner helps to produce reusable code which will be very easy to maintain and modify. Say, given three computational engines implemented in `INVERT` (TSVD, Tikhonov Regularization, and LSQR), we want to add a fourth. Using the current design, it will not take much effort to accomplish it, since the computational part is completely separated from the visualization and interface parts. More details on software maintenance and modification can be found in the developer manual.

5.4 LGEN Design

A sketch of the UML class diagram of the `LGEN` application summarizing its class static relationships is shown in the Figure 5.1.

This application consists of the main class `lg` which is responsible for the Motif based graphical user interface, operations including events and callbacks. The attributes `slist` and `dlist` of this object are double circular linked lists (instances of the class `linked_list`). The linked lists include objects of the type `object` which in turn represents a source or a detector on the frame in the test section. There is no difference between sources and detectors from the programming perspective.

The class `lg` also has two databases `db_stod` (lines given by sources

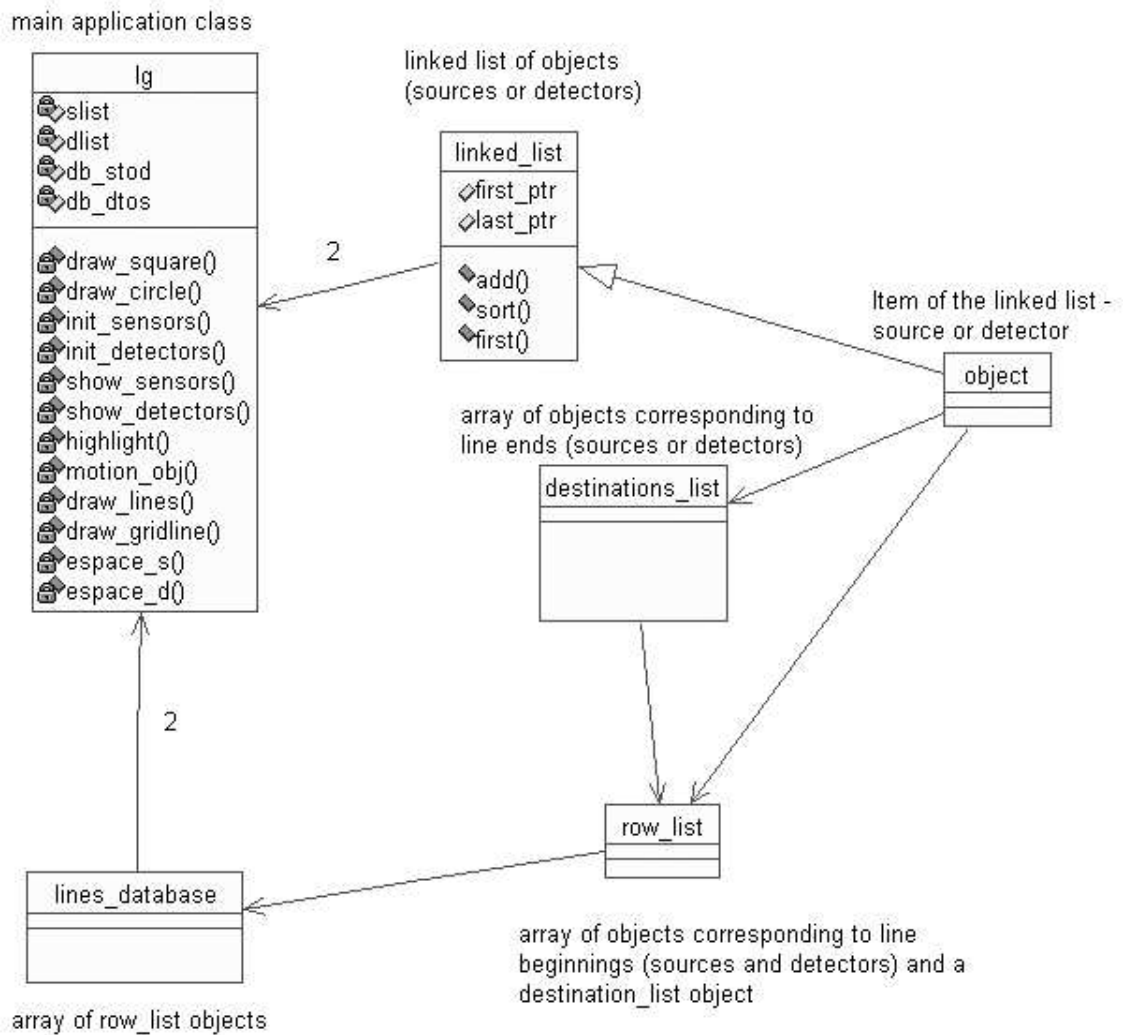


Figure 5.1: Sketch of the LGEN UML class diagram. Only a few methods and data structures are shown. The diagram was generated by Rational Rose.

connected to detectors) and *db_dtos* (lines given by detectors connected to sources). The database objects are instances of the class *lines_database* which is related to the sources and detectors as shown in the diagram. Since the two databases include identical information on the line connections, one database is usually generated from the other. It is not problematic to program LGEN such that we use only one database. However, when we have geometries with a large number of sources and detectors, the “reverse tracing” (say, show lines starting at the given detectors versus show the lines starting at the given source) of the database can be very slow.

As a potential improvement of LGEN, one can consider the implementation of mySQL database. This will allow one to work with only one lines database and operate on it very quickly. In addition, in the future development of LGEN this will save programming work doing database queries and exporting and importing the database. Finally, a mySQL is available for both IRIX and Linux platforms, which meets the portability requirements of the software package.

One of the main components of LGEN is the drawing area widget. Even though a Motif drawing area widget provides all required functionality for the application, its use is inconvenient since it requires a significant amount of low-level X-Windows programming to establish the OpenGL-Motif interface. A simpler approach is to consider an OpenGL-specific Widget. Unfortunately, this widget is not available on all platforms, and its installation is painful. That is why, LGEN was implemented using a Motif drawing area widget. A detailed description of integrating Motif with OpenGL can be found in [14].

The Motif drawing area widget provides an interface to interpret and display OpenGL statements inside the widget. Moreover, Motif provides a collection of callbacks and events which are registered to that widget.

Among these events are right and left click of the mouse and the motion of the mouse pointer over the widget. Motif allows a program to read the states of the keyboard buttons within the event. This increases the functionality and interactivity of the user interface.

Among these drawing area widget callbacks employed are *expose* and *resize*. The *resize* callback is called when the user is resizing the screen. This callback is programmed to preserve the image ratio. The *expose* callback is called upon exposure of the drawing area widget and calls the registered function to redraw all contents of the widget executing required OpenGL statements. The use of double screen buffers allows a program to make expose actions invisible for the user. In other words, the user sees only the final result. The double buffering technique is especially useful for the implementation of the function to move an object (source or detector) over the the frame in the drawing area widget. This widget has to be exposed after every movement of the mouse to have a feeling of the real-time movement. If a single buffer is used, the widget may blink.

In order to save and print the high resolution images of the drawing area widget, each OpenGL statement was echoed by an analogous statement to draw Encapsulated Postscript (EPS). EPS is a standard format for importing and exporting PostScript language files in all environments. It is a single page PostScript language program that describes an illustration using

vector graphics. Using EPS the resolution of the print-out or screen image is defined by the user and is bounded by the maximum output device (such as printer) resolution. A detailed description of programming vector graphics in PostScript can be found in [1].

5.5 LINE2MAT Design

Unlike other applications, LINE2MAT does not take much advantage of the object-oriented design. The sketch of the LINE2MAT UML class diagram is shown in the Figure 5.2. The methods of the main class are responsible for all computation and functionality of LINE2MAT. The only purpose of the remaining three classes *line_descriptor*, *line_list*, and *endpoints* is to simplify the generation of matrix A from the lines file representing line endpoints.

There was no need to implement a more sophisticated design, since LINE2MAT mostly consists of single computational functions which do not have interactions with each other.

As a potential improvement of LINE2MAT one can consider the conversion of major functions such as *dense_svd* to subprograms which will be executed by either a graphical user interface extension application or a simple script file. This will be more convenient for the user.

LINE2MAT is also supplied with a simple Motif graphical user interface implemented as a separate program LINE2MAT_INTERFACE which communicates with LINE2MAT through the command line. This program also performs elementary validation of the LINE2MAT inputs such as consistency of matrix dimensions and so on.

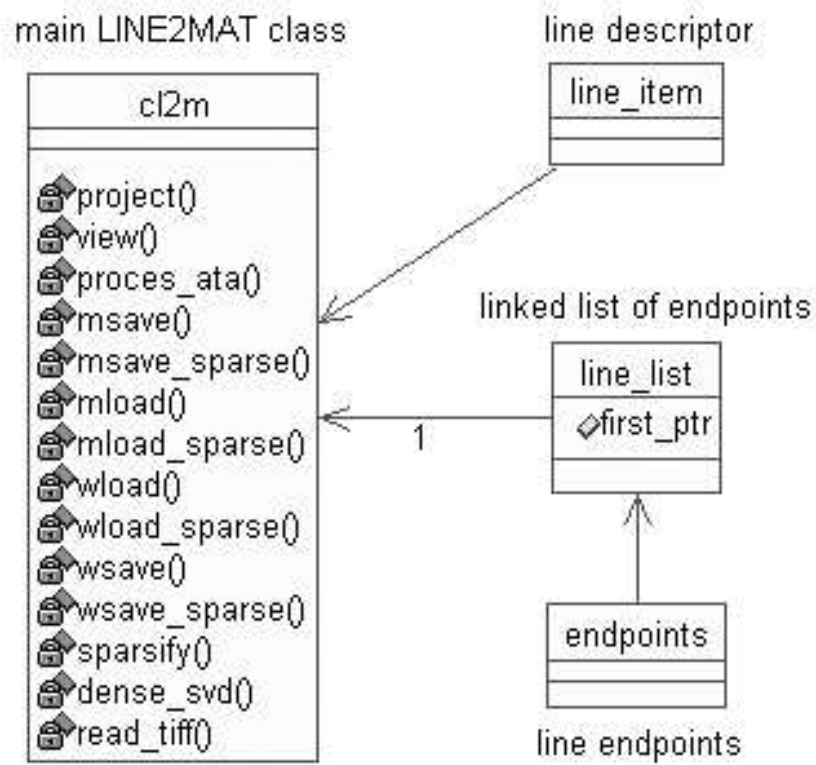


Figure 5.2: Sketch of the LINE2MAT UML class diagram. Only a few methods and data structures are shown.

5.6 INVERT Design

The schematic of the current design of INVERT is shown in the UML diagram in the Figure 5.3.

INVERT consists of 5 classes. One of the main responsibilities of the class *work_window* is to provide a Motif graphical user interface to the computation engines. The interface consists of two major windows - the task window and the reconstruction window.

The task window is implemented in the *work_window* class and communicates with the object of the type *task* which stores all information concerning the current task including the numerical method used, input file locations, etc. Changes in the reconstruction window are reflected when the user clicks either “OK” or “Apply” button. This action “calls” a registered callback which communicates with the computational engines.

The reconstruction window is the main window programmed in the constructor of the *work_window* class. It consists of three parts. Its first part is the menu pane. The second fragment (Motif form) is the viewer for the reconstructed image. These two parts are common for all computational engines (except for several items in the menu pane). The third part (Motif form) represents the inputs and outputs specific for each method implemented. When the *work_window* class is instantiated, only the first two parts of the window are created. The third part (form) is created upon the user request from the task window. If the user selects some numerical reconstruction method while working with another one, the third part of the reconstruction window is automatically “unmanaged” providing control to another

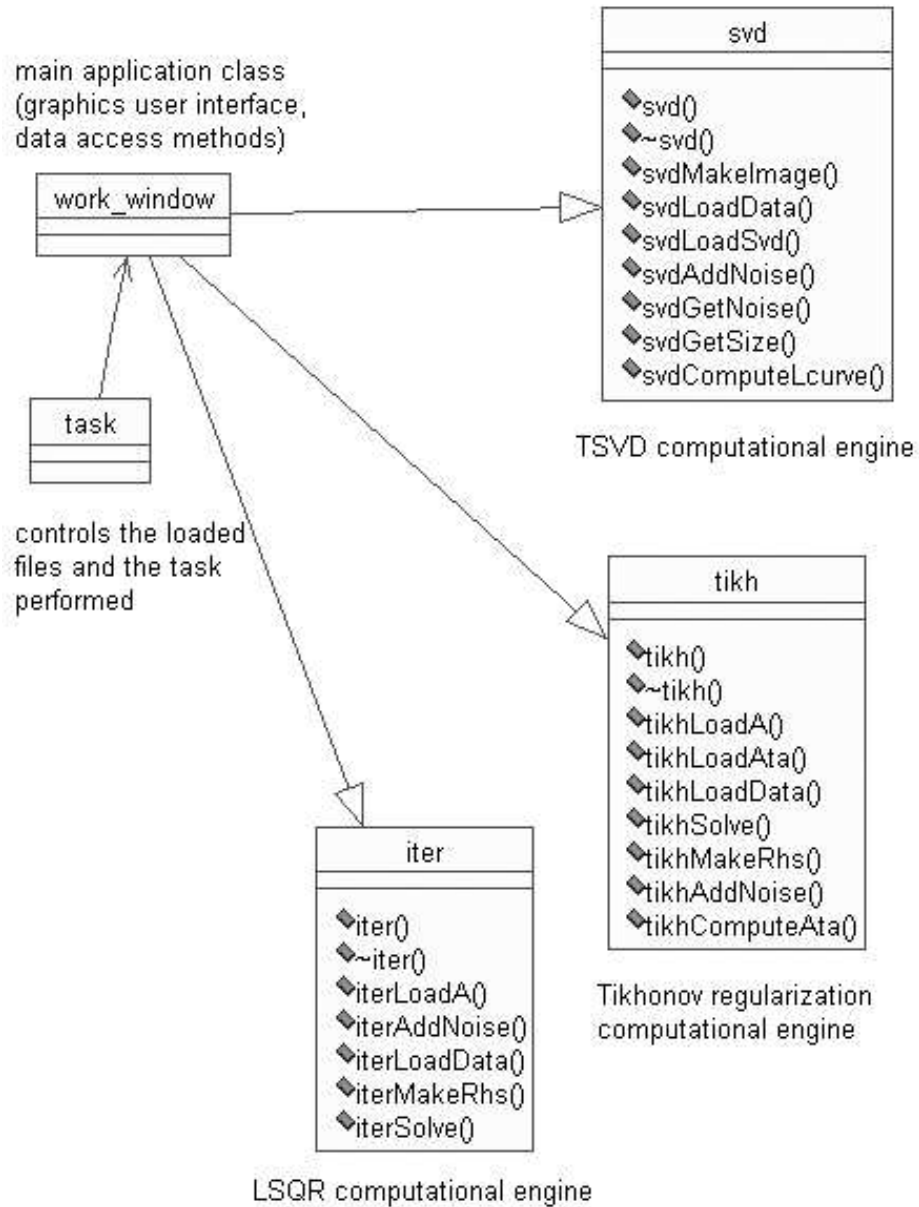


Figure 5.3: Sketch of the INVERT UML class diagram. Only a few methods and data structures are shown.

implementation of this part.

INVERT provides an interface to load ASCII, binary, zip-compressed, and sparse data using the functions *read_ascii*, *read_rhs*, *read_gz*, and *read_hb* respectively. The application also allows the user to add noise using the method *add_noise*. The functions *read_ascii*, *read_rhs*, *read_gz*, *read_hb*, and *add_noise* do not belong to any class due to access problems caused by the existing software design.

The classes *iter*, *tikh*, and *tsvd* are computational engines implementing LSQR, Tikhonov regularization, and truncated singular value decomposition respectively. Each of the computational engines loads the data using a function from the set *read_ascii*, *read_rhs*, *read_gz*, and *read_hb* available from INVERT. The noise in the right-hand side b can be added via *add_noise* method. Given the required inputs, each computational engine takes care of the reconstruction and returns the solution, as well as other information which helps to analyze the results to the *work_window* object. The solution is displayed in the image viewer area and the other results are shown in the method specific part of the reconstruction window.

The drawing area is implemented as the Motif drawing area widget. However, the OpenGL graphics solution presented in the LGEN design description was not used here due to the difficulties in Motif-OpenGL interface initialization. Instead, we considered using X-Windows low level graphics, since the task was simply to show the bitmap image. The image (output of the computational engine) was represented as an n -vector of double precision numbers. The elements of this vector filled the pixels of the drawing area

widget preserving the pixel sizes of the frame inside the test section. The zooming function implemented in *work_window* allows users to magnify the image. A similar image viewer was also implemented in the function of *cl2m* class in LINE2MAT in order to display the matrix A , $A^T A$, and the selected right singular vector with certain magnitudes.

While developing INVERT, we experimented with a number of packages allowing us to solve system (2.6) in parallel using OpenMPI (Message Passing Interface). We found out that it is extremely difficult to keep track of the data-flow and communicate between processors using the current software architecture style implemented in INVERT. One of the solutions to simplify the usage of MPI is to consider other architecture styles such as the data flow style and if necessary implement the corresponding design pattern.

As potential changes in INVERT, we propose the modification of the entire software design. The computational engines should be preserved, however they should use data-access functions of their new parent class *data_access*. The *work_window* class should not have inheritance relationships with the computational engines and should be responsible only for the graphical user interface. Even though the current design allows the easy addition of new computational engines, the suggested architecture would make it even easier. The UML diagram of the proposed design is shown in the Figure 5.4.

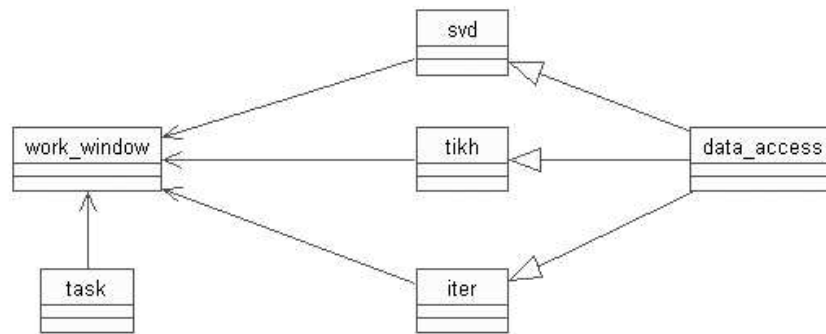


Figure 5.4: Proposed design of INVERT. Sketch of UML class diagram.

Chapter 6

Numerical Experiments

The numerical experiments summarized in this section were performed for the laboratory mock model described in this thesis. The model consists of 24 sources and 84 detectors placed on the four sides of the 9' by 6' frame of the test section surrounding the reconstruction region of interest. Since the given geometry consists of 977 lines and only overdetermined systems were considered, we require the number of pixels in the reconstruction region to be equal or less than 977. Moreover, we want to preserve the 3:2 image ratio which will make the reconstructions correspond to the ratios of the test section.

Consider a matrix A consisting of 864 (36×24) pixels achieving the maximum resolution of the reconstructed image. The matrix A consists of 977 rows, corresponding to the number of lines in the sensor system.

Both laboratory and simulated data were used for the right-hand side b . When performing reconstructions for the laboratory dataset, we know *a priori* that we should reconstruct an image of a circular object placed in the center of the test section. However, we do not know the exact answer, and this hinders our conclusions about the quality of the reconstruction. On the other

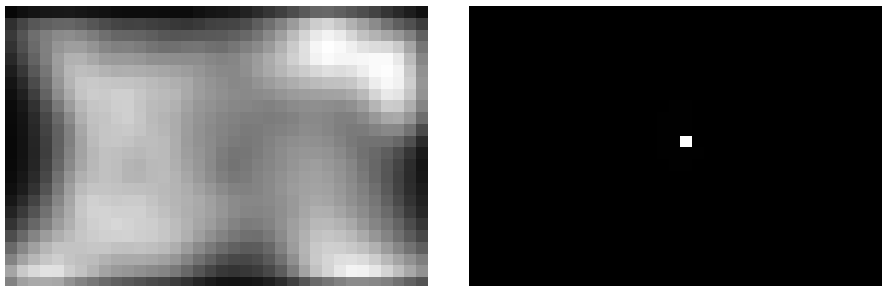


Figure 6.1: Simulated datasets d_1 [left] and d_2 [right] consisting of 864 pixels displayed as 36 by 24 pixel image in grayscale. The dataset d_1 represents a typical liquid water distribution in the wind tunnel. The dataset d_2 consists of one white pixel.

hand, using simulated data we know exactly what the reconstruction should look like. We used two simulated datasets d_1 and d_2 yielding the right-hand sides $b_{1,2} = Ad_{1,2}$ or $b_{1,2} = W^{1/2}Ad_{1,2}$ if weighting was used (see Figure 6.1 for the simulated datasets consisting of 864 pixels displayed as 36 by 24 pixel images). Then, ideally, after the reconstruction we should get d back. We can evaluate the quality of resulting image by computing the residual as described in the chapter on computer implementations.

6.1 Reconstruction via TSVD

When sensors are placed on the four walls, the reconstruction becomes well-conditioned or mild ill-conditioned. However, since no line crosses the pixels in the corners of the reconstruction region, there are several zero singular values which we do not consider here. There is a relatively gentle singular value decay, as observed in Figure 6.2 and Figure 6.3 for non-weighted and weighted matrices respectively.

The ratio of the largest singular value to the smallest one is of the order of 10^4 , so a robust reconstruction may require imposing constraints on

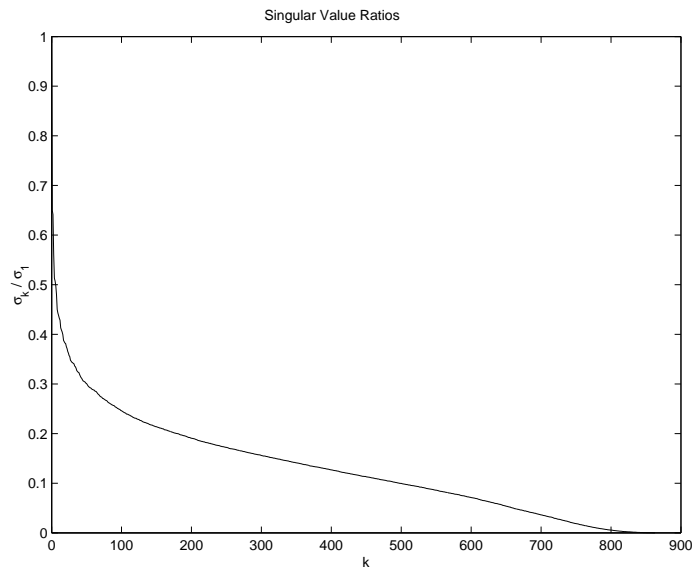


Figure 6.2: A plot of the singular value ratio σ_k/σ_1 when weighting was not applied to the matrix A . There are 864 singular values. The slow drop indicates that the inversion of A is not ill-conditioned.

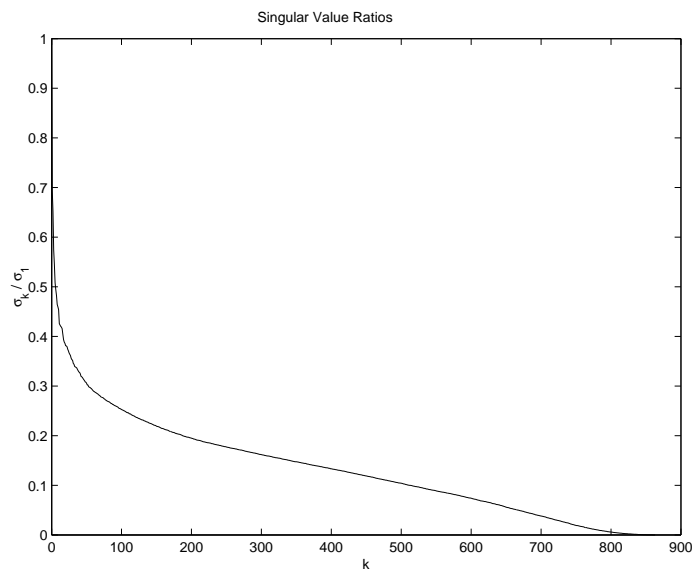


Figure 6.3: A plot of the singular value ratio σ_k/σ_1 when weighting was applied to the matrix A . There are 864 singular values. The slow drop indicates that the inversion of A is not ill-conditioned.

the experimental hardware noise levels.

The application of the weighting matrix $W^{1/2}$ reduces the magnitudes of the singular values of the matrix $W^{1/2}A$ compared to those of A . However, as observed in the Figure 6.2 and Figure 6.3, the weighting does not change the ratios of the singular values significantly. Nevertheless, the weighting affects the singular vectors (see Figure 6.4 and Figure 6.5), and the final reconstructed images become slightly smoother.

As shown in the Figure 6.4 and Figure 6.5, the last right singular vectors of both non-weighted and weighted matrices provide high frequency content, as well as information used to reconstruct near the corners, neither of which are significant for this application. This motivates us to use truncated singular value decomposition, allowing for noise suppression and truncating the number of singular triplets used in the reconstruction.

6.1.1 Using the Dataset d_1

The projection $b = Ad_1$ was used as the right-hand side for the “non-weighted” reconstruction. As observed in the Figure 6.6, in the absence of noise, the reconstruction using all singular triplets corresponding to the nonzero singular values is visually identical to the original (compared to the original displayed in the Figure 6.1). If we use only 600 out of 864 singular triplets, the reconstruction is still visually acceptable and resembles the shapes of the original image. Adding 5% of noise produces extremely degraded image. The degradation is due to the noise causing the amplification of the coefficients of the singular vectors corresponding to the smallest singular values. Due to the mild ill-conditioning of the matrix, the reconstructed image contains no visibly

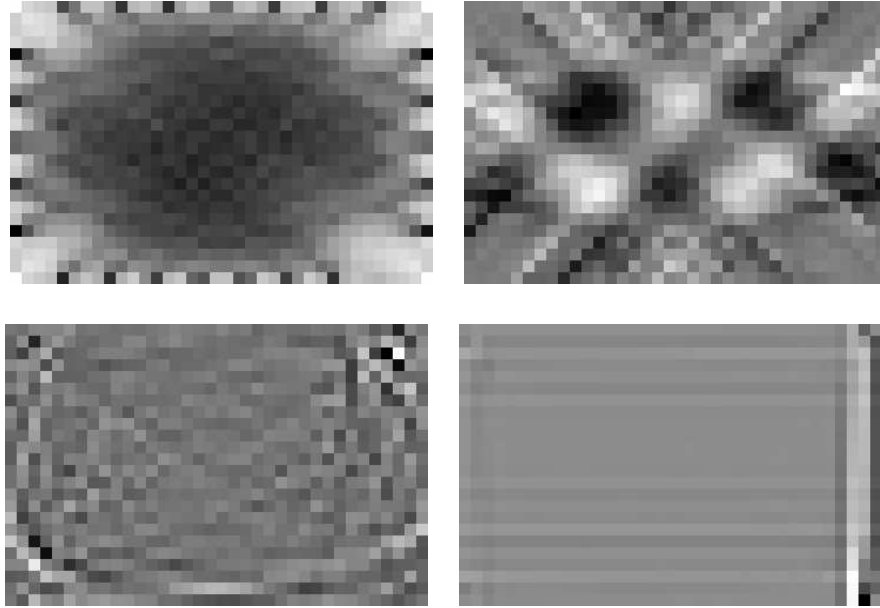


Figure 6.4: The right singular vectors v_1, v_{17}, v_{806} , and v_{857} when weighting was not applied to the matrix A . The corresponding singular value ratios are $\sigma_1/\sigma_1 = 1$, $\sigma_{17}/\sigma_1 = 0.3994$, $\sigma_{806}/\sigma_1 = 0.0046$, and $\sigma_{857}/\sigma_1 = 0.0001$.

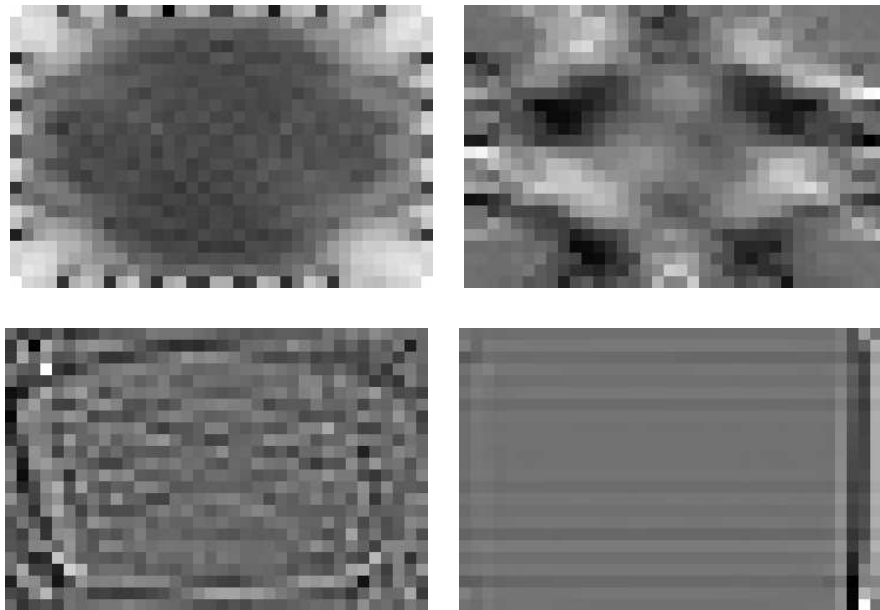


Figure 6.5: The right singular vectors v_1, v_{17}, v_{806} , and v_{857} when weighting was applied to the matrix A . The corresponding singular value ratios are $\sigma_1/\sigma_1 = 1$, $\sigma_{17}/\sigma_1 = 0.4044$, $\sigma_{806}/\sigma_1 = 0.0049$, and $\sigma_{857}/\sigma_1 = 0.0001$.

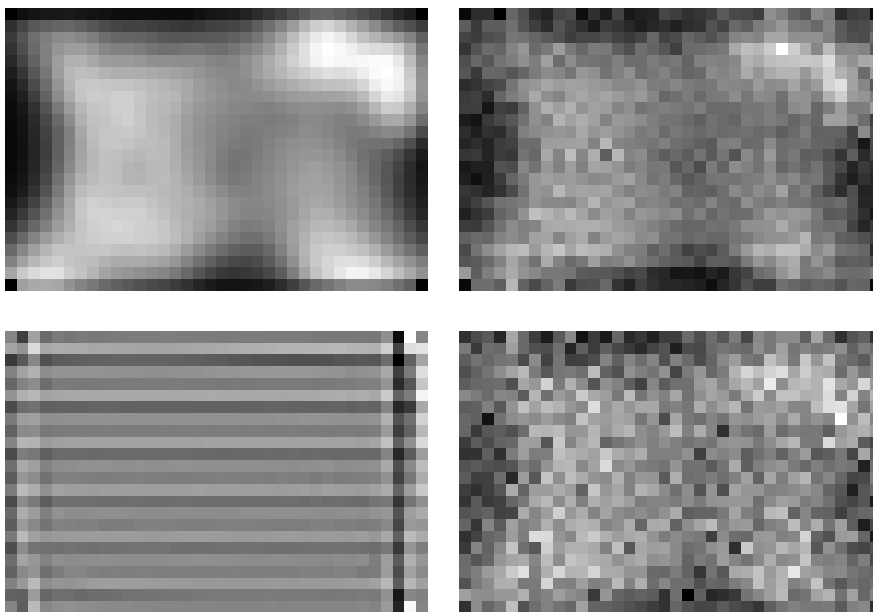


Figure 6.6: Reconstructions without weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triples [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triples [bottom,right] with noise level 5%.

useful information. If 600 out of 864 singular triplets are used, the effect of noise is reduced, and we are able to see the shape of the image again.

The “weighted” reconstruction (working with the projection $b = W^{1/2}Ad_1$), gives essentially the same results (see Figure 6.7). However, even if it can be hardly visible, the images are slightly smoother compared to the previous reconstructions due to weighting. This is more visible for the reconstructions of the higher resolutions.

6.1.2 Using the Dataset d_2

Again, the projection $b = Ad_2$ was used as the right-hand side for the “non-weighted” reconstruction. According to the Figure 6.8, in the absence of

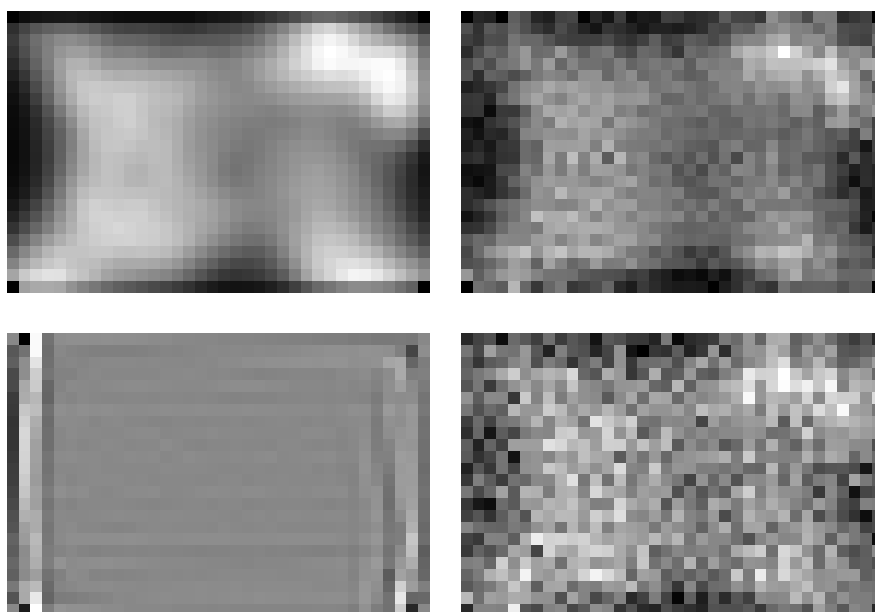


Figure 6.7: Reconstructions with weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triples [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triples [bottom,right] with noise level 5%.

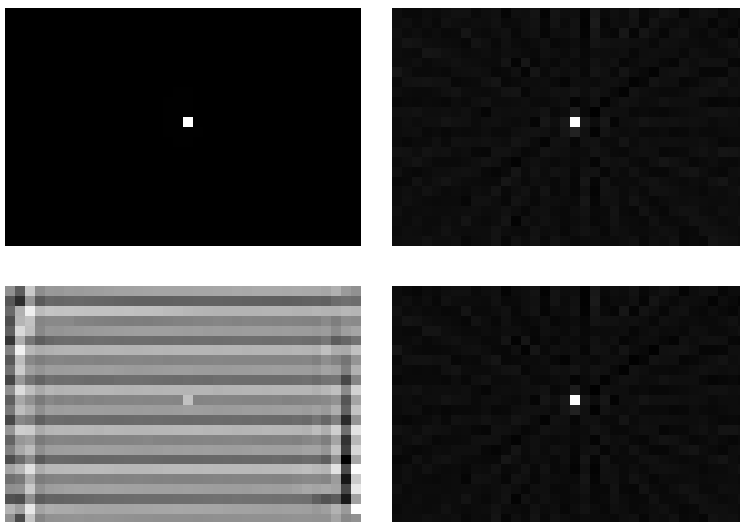


Figure 6.8: Reconstructions without weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triples [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triples [bottom,right] with noise level 5%.

noise, the reconstruction using all singular triplets corresponding to the nonzero singular values is visually identical to the original. Using only 600 out of 864 singular triplets, the reconstruction is still acceptable. Adding 5% of noise produces extremely degraded image due to the mild ill-conditioning of the matrix. However, as in previous case, if we use only 600 out of 864 singular triplets, the effect of noise decreases, and we are able to see the image again.

For the “weighted” reconstruction (working with the projection $b = W^{1/2}Ad_2$), we get similar but slightly “blurred” results (see Figure 6.9).

6.1.3 Using Laboratory Data

Here b is given, and we reconstruct the image. The only thing that we know about the image is that it represents a roughly circular object placed near the geometrical center of the test section. Again, we consider both

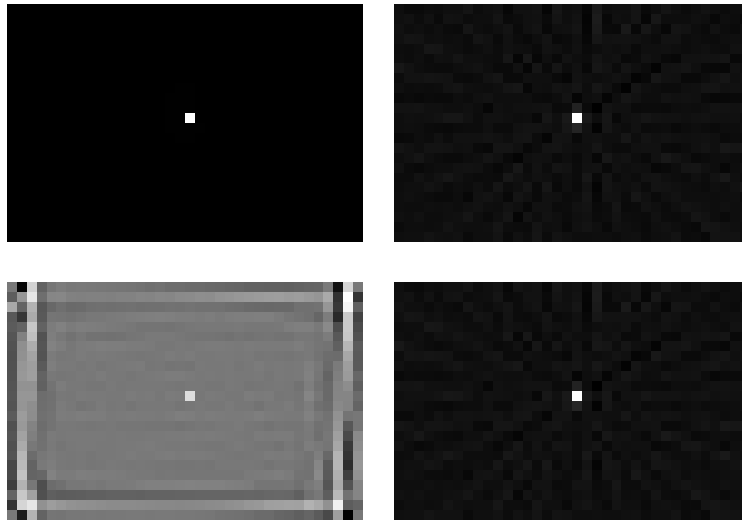


Figure 6.9: Reconstructions with weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left] and 600 triplets [top,right] without noise. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [bottom,left] and 600 triplets [bottom,right] with noise level 5%.

“non-weighted”, as well as “weighted” reconstructions.

When no weighting is used, we work with the given right-hand side b . When all singular triplets corresponding to nonzero singular values are used, the image is degraded with noise since the data is obtained from the laboratory operating in the real-life conditions (see Figure 6.10). Truncating the reconstruction to 785 triplets, we begin to see something in the center, but the object is hardly visible. Using 521 singular triplets, we see some object in the center, however we still cannot determine its shape exactly. Now if we use 328, or better 256 singular triplets we clearly see the circular object.

If we employ weighting, we work with the right-hand side $W^{1/2}b$ instead of b . The results are similar to those when no weighting was used but are slightly smoother. Again, it is really hard to see the difference here, but

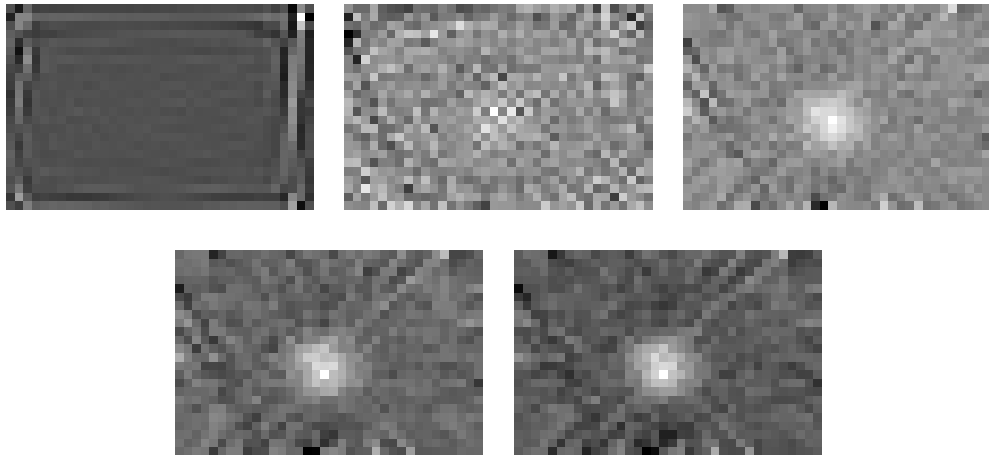


Figure 6.10: Reconstructions without weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left], 785 triplets [top,center], 521 triplets [top,right], 328 triples [bottom,left], and 256 triplets [bottom,right].

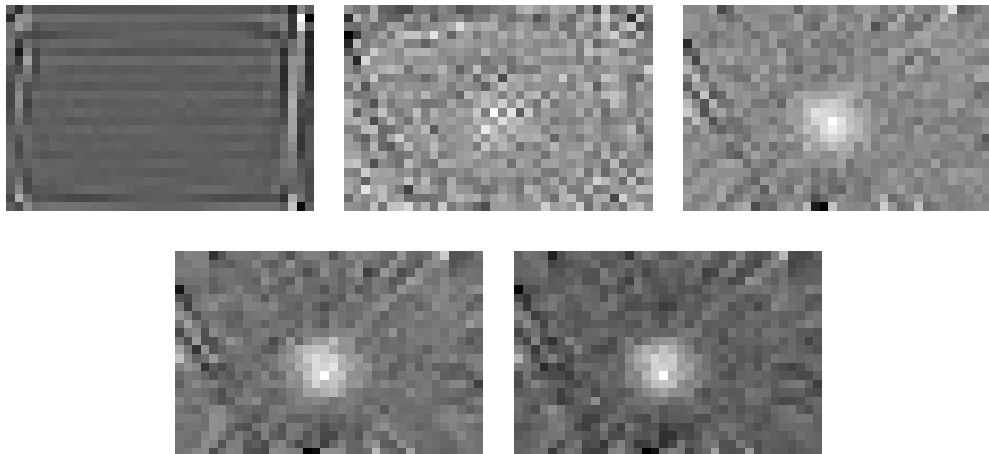


Figure 6.11: Reconstructions with weighting. Reconstructed images using all singular triplets (except for those corresponding to zero singular values) [top,left], 785 triplets [top,center], 521 triplets [top,right], 328 triples [bottom,left], and 256 triplets [bottom,right].

while working with the higher resolution images this effect is more visible.

6.2 Reconstruction via Tikhonov Regularization

Unlike the reconstruction via truncated singular value decomposition (where we have to precompute the components of SVD which is expensive in both computer resources and time), the Tikhonov regularization requires a solution of a small system at each step which is relatively cheap. Unfortunately, as a trade-off for resources and time benefits, we lose the power of analysis provided by the singular value decomposition.

As discussed before, Tikhonov regularization is solved by Cholesky factorization applied to the system $(A^T A + \tau^2 I)x = A^T b$. Given that typically $m \gg n$, the matrix $A^T A + \tau^2 I$ holds much fewer elements compared to the original matrix A . This compression affects the condition number of the new matrix which is equal up to the square of the condition number of the matrix A . Hence, for ill-conditioned matrices, this technique may not be always applicable.

The matrix A we are working with is mild ill-conditioned, hence it makes sense to use Tikhonov regularization for the reconstruction.

As observed in the Figure 6.12, the matrix $A^T A$ has large elements along the diagonal which correspond to the white pixels in the Figure 6.12. However, in order to ensure positive definiteness of the matrix $A^T A$ we add the matrix $\tau^2 I$, where the selection of regularization parameter τ is discussed in the previous chapters.

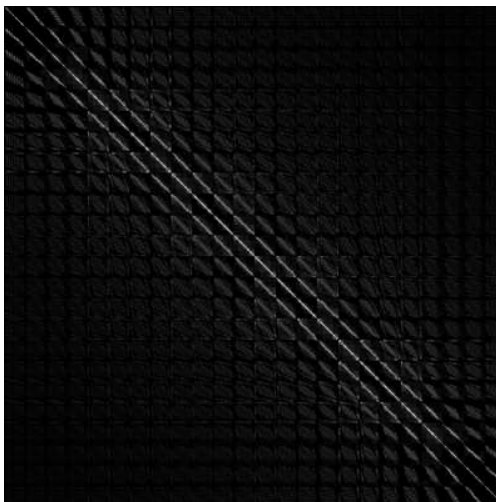


Figure 6.12: A graphical grayscale representation of the matrix $A^T A$. Black pixels correspond to minimum elements, and white pixels correspond to maximum elements.

6.2.1 Using the Dataset d_1

First of all, we try to perform the reconstruction with $\tau^2 = 0$. The solution of the system fails, since the matrix $A^T A$ contains zeros along the main diagonal, and therefore is not positive definite. Hence we have to use nonzero regularization parameter τ . We try $\tau^2 = 0.001$. As observed in the Figure 6.13 the reconstructed image is visually identical to the original. Using the regularization parameter $\tau^2 = 1.5$ produces similar but slightly blurred image. If τ^2 is too large, (say, equal to 20), the image is similar to the right singular vector v_1 of the original matrix A .

Now, we add 5% of noise. Due to the mild ill-conditioning of the matrix and the relatively large level of noise, we are unable to get reasonable reconstructions for $\tau^2 = 0.001$. When we use more regularization, i.e. $\tau^2 = 1.5$, we obtain the expected image. Again, using too large regularization parameter results in the image corresponding to v_1 .

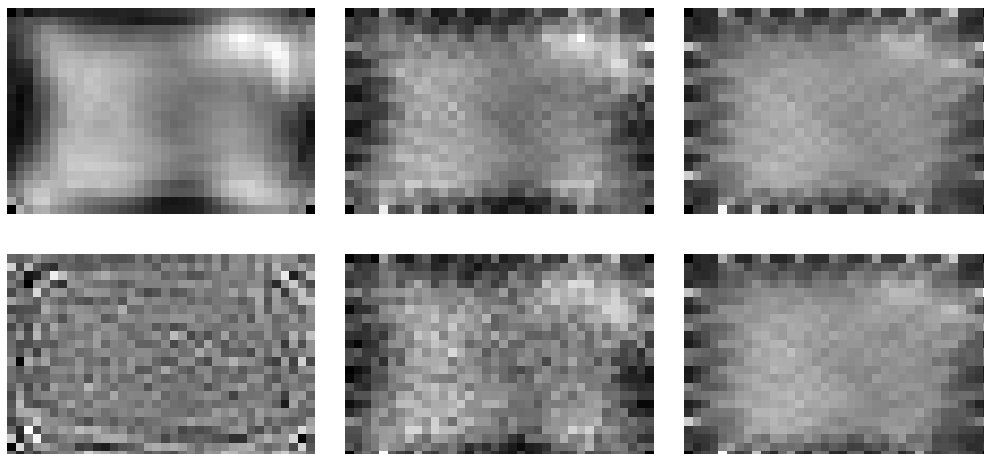


Figure 6.13: Reconstructions without weighting. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] without noise. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] with noise level 5%.

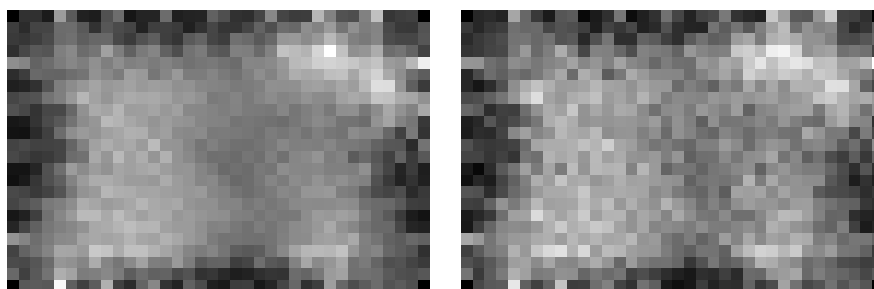


Figure 6.14: Reconstructions with weighting. Using $\tau^2 = 0.3$ without noise [left] and with 5% noise [right].

The weighting matrix $W^{1/2}$ applied to the original matrix A changes the problem, but produces similar reconstructions (see Figure 6.14). Unlike the singular value decomposition, it is hard to compare “weighted” and “non-weighted” reconstructions since the eigenvalues of the matrix change, and the same regularization parameters have different effects in “weighted” and “non-weighted” reconstructions. However, since we know the singular values from our experiments, as well the filtering factors, we can make a comparison.

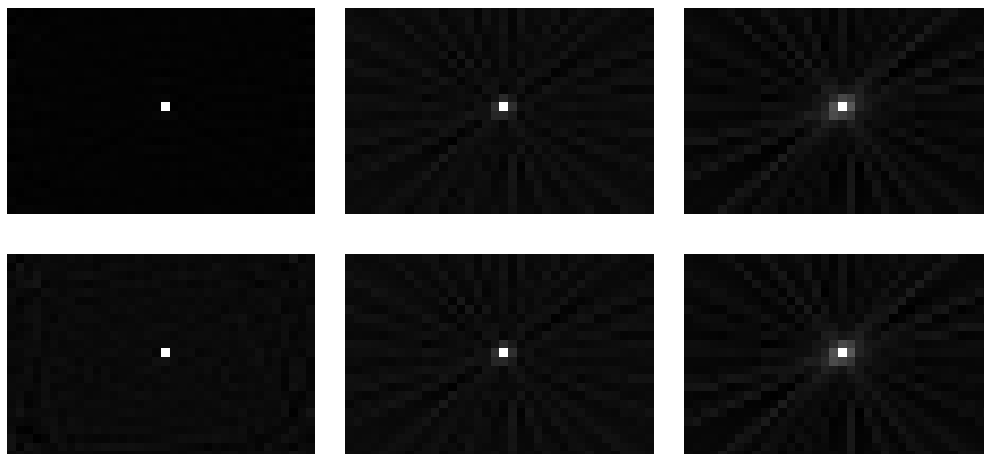


Figure 6.15: Reconstructions without weighting. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] without noise. Using $\tau^2 = 0.001$ [top, left], 1.5 [top, center], and 20 [top, right] with noise level 5%.

6.2.2 Using the Dataset d_2

Using $\tau^2 = 0.001$ results in the visually identical to the original reconstruction (see Figure 6.15). Due to the nature of this dataset, the regularization parameter $\tau^2 = 1.5$ also produces visually identical to the original reconstruction. The large values of τ^2 yield the reconstructions similar to those obtained by the first several singular triplets of the matrix A using singular value decomposition.

Adding 5% of noise does not visibly degrade the image due to the nature of right-hand side. Hence for this dataset we do not observe the effect of the regularization on the noise reduction in the reconstructed image.

The Figure 6.16 presents the results using weighting matrix $W^{1/2}$ applied to the matrix A . It is hard to say anything certain in this situation, since both “weighted” and “non-weighted” reconstructions look very close to being visually identical to the original.

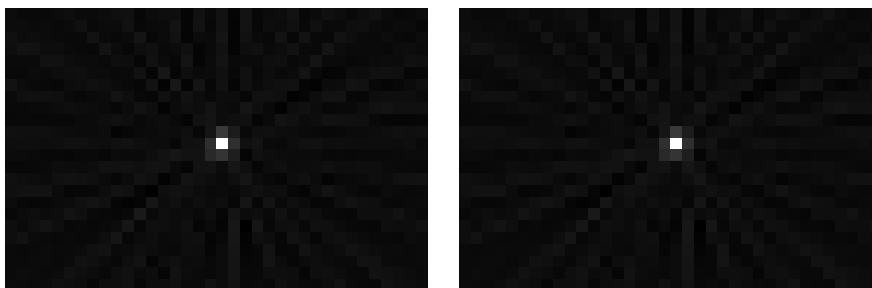


Figure 6.16: Reconstructions with weighting. Using $\tau^2 = 0.3$ without noise [left] and with 5% noise [right].

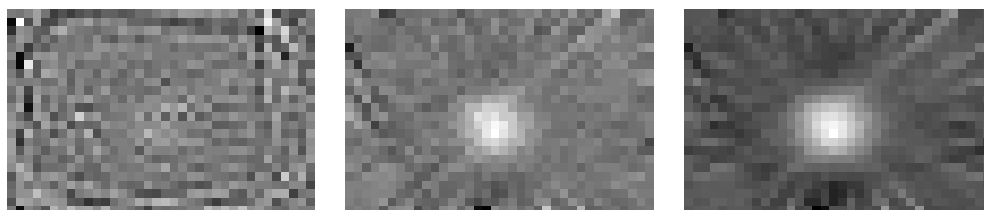


Figure 6.17: Reconstructions without weighting. Using $\tau^2 = 0.001$ [left], 1.5 [center], and 20 [right].

6.2.3 Using Laboratory Data

In this experiment our goal was to reconstruct the image which should represent a circular object in the geometrical center of the test section.

Using very small regularization parameter $\tau^2 = 0.001$ does not result in reasonable reconstruction due to the strong contamination of data with noise and the mild ill-conditioning of the original matrix (see Figure 6.17). Larger regularization parameters, such as 1.5 produces expected image. Selecting very large regularization parameters (such as 20) results in the images similar to those obtained by using a few large singular triplets in the SVD. As observed in the Figure 6.18, the application of the weighting factor produces similar images.

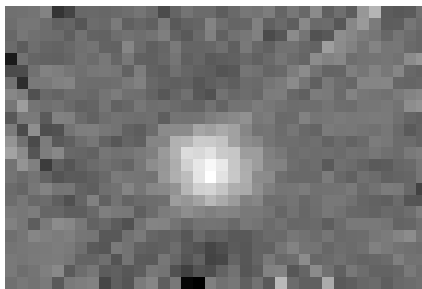


Figure 6.18: Reconstruction with weighting. Using $\tau^2 = 0.3$.

6.3 Reconstruction via LSQR

Even if the results of TSVD and Tikhonov regularization were acceptable, LSQR allows to obtain similar reconstructions using much less computer resources and time solving the system iteratively. This technique is the only method of choice when the input matrix A is very large.

Given that the software package allows to use four stopping criteria to terminate LSQR iterations - maximum matrix tolerance, maximum vector tolerance, maximum matrix condition number (see discussion of stopping criteria for LSQR), and maximum number of iterations - in the experiments we mainly consider the maximum number of iterations which is directly related to the computational time. Since the reconstructed image often does not change significantly after several iterations, it does not make sense to continue LSQR algorithm after that point.

6.3.1 Using the Dataset d_1

First of all, we notice that the stopping criteria provided by LSQR are not efficient for the image reconstruction. Reconstructing image without damping, noise, and weighing, LSQR algorithm terminates after 178 iterations yielding

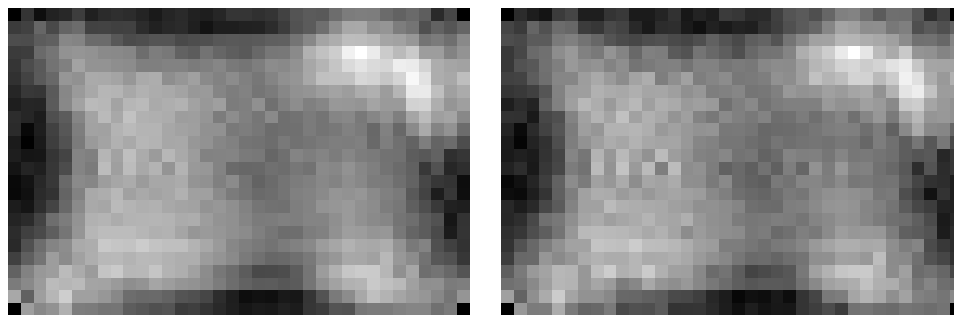


Figure 6.19: Reconstruction without damping, noise, and weighting. Using 178 iterations [left] and 20 iterations [right].

the image similar to the original depicted in the Figure 6.1 (see Figure 6.19).

On the other hand, using only 20 iterations we get more or less the same reconstruction. Hence, there is no need to continue LSQR after the 20th iteration for this case in order obtain a reasonable reconstruction.

If we reconstruct with 5% noise without weighting, we are still able to get original images back using nonzero damping as observed in the Figure 6.20. When damping τ is 0, the algorithm runs 1841 iterations = $n + m$ which is the default limit on the number of iterations. Due to the noise and mild ill-conditioning of A , we do not get a reasonable reconstruction. However, if we use damping $\tau = 0.5$ (136 iterations) we see the noisy image. We are able to get rid of some noise using $\tau = 2$ (40 iterations). However, as in Tikhonov regularization, using very large parameters τ (i.e. $\tau = 10$ - 10 iterations) results in the images similar to those obtained by using a few singular triplets of the singular value decomposition of the matrix A .

If weighting matrix is employed, we solve a slightly different problem, however we are still able to get fair reconstructions. The reconstructions with the 5% noise level using damping 0 (1841 iterations) and 0.4 (64) iterations

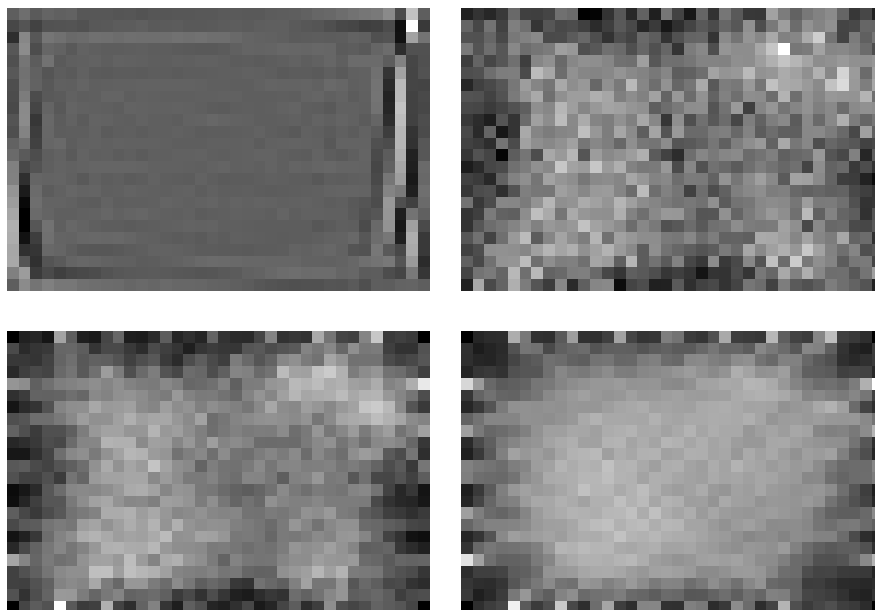


Figure 6.20: Reconstruction without weighting with 5% noise. Using 1841 iterations and 0 damping [top, left], 136 iterations and 0.5 damping [top, right], 40 iterations and 2 damping [bottom, left], and 12 iterations and 10 damping [bottom, right].

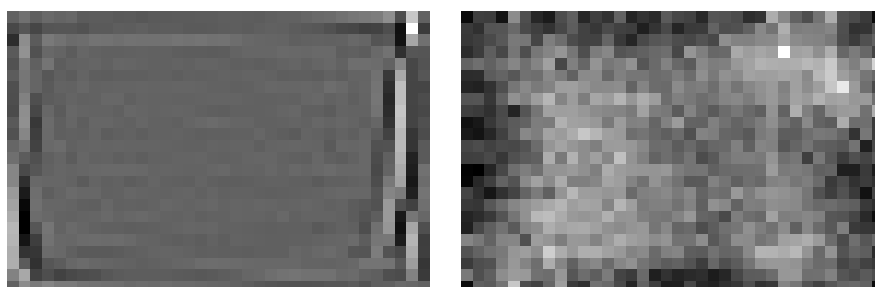


Figure 6.21: Reconstruction with weighting and 5% noise. Using 1841 iterations without damping [left] and 68 iterations with damping 0.4 [right].

are shown in the Figure 6.21.

6.3.2 Using the Dataset d_2

The reconstructions of the dataset d_2 are shown in the Figure 6.22. Since 5% noise is used, we are not able to get good reconstructions for $\tau = 0$. However, if we use some nonzero τ , such as 0.1, we will get the original picture back.

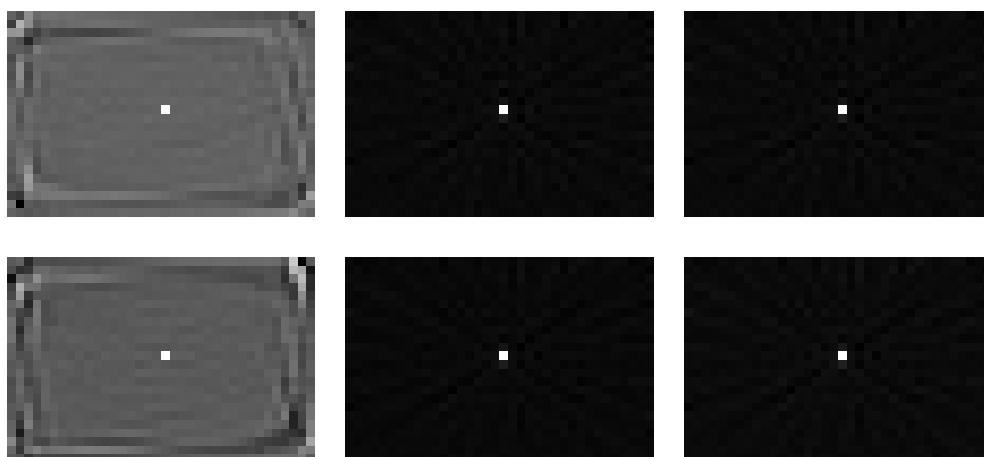


Figure 6.22: Reconstruction without weighting with noise level 5% [top]. Using 1841 iterations and 0 damping [top, left], 616 iterations and 0.1 damping [top, center], 20 iterations with 0.1 damping [top, right]. Reconstructions with weighting with noise level 5% [bottom]. Using 1841 iterations and 0 damping [bottom, left], 243 iterations and 0.1 damping [bottom, center], 20 iterations with 0.1 damping [bottom, right].

LSQR does not need to run hundreds of iterates for the reconstruction as suggested by other stopping criteria, since using only 20 reconstruction results in very similar reconstruction.

6.3.3 Using Laboratory Data

The reconstructions for the laboratory data look very similar to those obtained by Tikhonov regularization and truncated singular value decomposition. The resulting images are shown in the Figure 6.23.

The optimal damping parameter τ for the “non-weighted” problem is 1, and it takes LSQR 73 iterates to get the reconstructed image. For the ‘weighted’ problem, we need a lower damping, such as $\tau = 0.5$, and it takes LSQR 56 iterates to do the reconstruction. Again, reconstruction with and without matrix $W^{1/2}$ are similar.

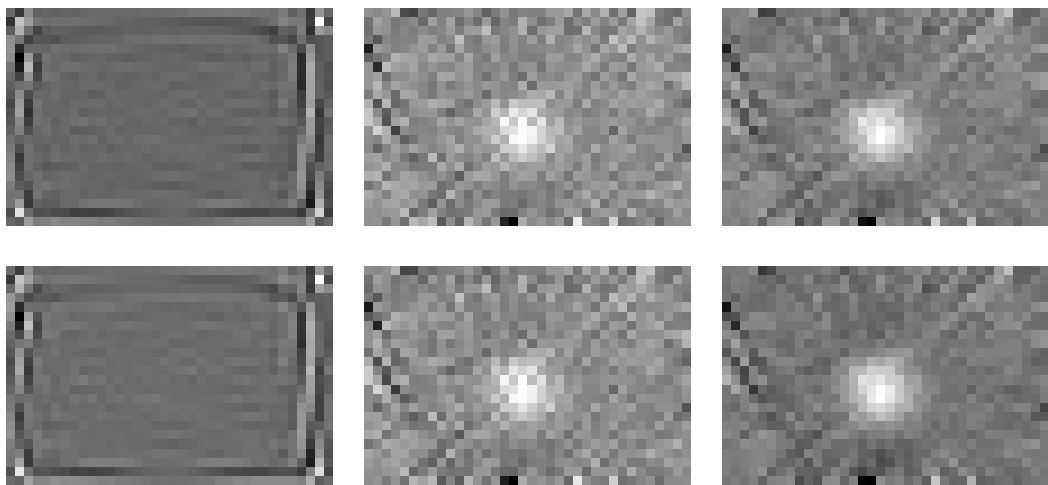


Figure 6.23: Reconstruction without weighting 5% [top]. Using 1841 iterations and 0 damping [top, left], 138 iterations and 0.5 damping [top, center], 73 iterations with 1 damping [top, right]. Reconstructions with weighting [bottom]. Using 1841 iterations and 0 damping [bottom, left], 128 iterations and 0.2 damping [bottom, center], 56 iterations with 0.5 damping [bottom, right].

Chapter 7

Comparison and Conclusions

A software package with user-friendly interface to model the geometry of the sensor system, perform reconstructions from the given data, and analyze obtained results was written and applied to both synthetic and laboratory data.

Although all the numerical methods implemented in this software package produce very similar reconstructions, each method has its own advantages and disadvantages in terms of performance, analysis, and results.

The truncated singular value decomposition provides reasonable reconstructions and can be used to reduce effects of the noise. Moreover, the implementation of TSVD permits the analysis of the singular triplets which make up the reconstructed image. This information helps one to make a decision regarding the number of SVD components needed to perform an optimal reconstruction. The main disadvantage is that the user has to precompute non-iteratively the reduced singular value decomposition, which is costly in both memory and resources. This takes hours of computer time for geometry profiles with a large number of lines. However, once this decomposition has been computed, the reconstruction is only a summation of

the right number of SVD components in `INVERT` which can be done quickly. In the conclusion, the truncated singular value decomposition approach is preferable for the relatively small problems when the analysis plays a more significant role than the reconstruction, however it is not feasible for large problems.

An alternative to singular value decomposition - Tikhonov regularization is, in some sense, similar to the TSVD method as described in the chapter on mathematical approaches. Reconstructions are obtained by Cholesky factorization of the matrix $A^T A + \tau^2 I$. This is significantly cheaper and faster than TSVD. The system matrix A is “compressed” to the matrix $A^T A$ which increases the condition number, deducing numerical stability. That is why the application of Tikhonov regularization, as any other method requires a preliminary conditioning analysis of the system. Since the matrix A used in the project is mildly ill-conditioned, we observe that the “compression” of the matrix does not visually affect the solution. Similar to the singular value decomposition, Tikhonov regularization regularizes the problem, blurring out the noise. The main advantage is that “precomputations” except for possibly $A^T A$ are not needed. The Tikhonov regularization, although it is non-iterative, works fast enough to compute reconstructions as the sliding bar controlling the regularization parameter is changed interactively. On the other hand, from the computational perspective, for each value of the parameter τ we have to solve the system $(A^T A + \tau^2 I)x = A^T b$. Hence, theoretically, the total price of using Tikhonov regularization may be very expensive. Since the typical geometry has many more lines than pixels, this is not a serious issue.

In the final analysis, although Tikhonov regularization lacks powerful analysis tools such as offered by TSVD, it is beneficial to use this method for the large well-conditioned or mildly ill-conditioned problems.

The LSQR iterative method is quickest in terms of performance, although, like Tikhonov regularization, it lacks the powerful analysis techniques provided by SVD. The results produced by LSQR are very similar to those produced by other methods. The implementation of LSQR allows damping to be used to smooth the noise in the reconstructed image. In addition, because of the iterative nature of LSQR, this method works fast and has very low resource and memory requirements. Most importantly, the LSQR implementation takes advantage of the matrix sparse structure. This leads to significant savings of memory. As with iterative method, LSQR requires the stopping criteria to terminate. Although many stopping criteria are implemented, in general, taking several iterations of this method is sufficient to obtain fair reconstructions. Similarly to Tikhonov regularization, for every choice of regularization parameter we have to solve the problem $Ax = b$. This can make the method expensive for the repeated usage. Nevertheless, due to the reasons stated above, LSQR is extremely well-suited for use for with large, sparse, well-conditioned or mild ill-conditioned problems. Applied to ill-conditioned problems, LSQR may fail to produce reasonable reconstructions.

As observed in numerical experiments, using the weighting matrix $W^{1/2}$ results in the smoother reconstructions. Since in reality the images are contaminated with noise, and our goal is to blur out the noise, the usage of

weighting is desirable.

Although this project offers robust reconstruction implementations, other numerical methods may produce very interesting results. For instance, the efficient implementation of a sparse iterative singular value decomposition will allow powerful analysis techniques to be applied to the large problems. Other reconstruction methods may also provide more effective blurring of the noise. In addition to new mathematical algorithms, the design of the software package can be also improved as described in chapters on programming and design considerations.

Chapter 8

Appendix A: Software Package User Manual

8.1 Introduction

This software package allows one to perform image reconstruction and analysis using the discussed numerical methods. The package consists of three main applications - LGEN, LINE2MAT^{8.1}, and INVERT^{8.2}.

The LGEN application guides through the first stage of the engineering experiment - the design of the sensor system - yielding the geometry profile. The geometry information can be also exported as a lines file, the major input of LINE2MAT.

LINE2MAT generates the matrices A and $W^{1/2}$ for the specific geometry modelled by LGEN; computes the SVD and forms the matrix $A^T A$ intended to be used in INVERT; generates simulated projected data; and allows the visualization of some output results.

INVERT is responsible for the interactive image reconstruction and analysis, implementing TSVD, Tikhonov regularization, and LSQR.

The descriptions of both physical and mathematical models, as well as

^{8.1}The initial IRIX version of this software designed for TSVD method was developed by Steven H. Izen.

^{8.2}The initial IRIX/ViewKit version of this software implementing TSVD reconstruction was developed by Steven H. Izen.

detailed explanation and analysis of numerical methods are presented in the preceding chapters of this thesis.

8.2 System Requirements

The binaries of this software package were tested on Red Hat Linux versions 7.1 and 7.2, as well as on IRIX 6.5 platforms. Since the compiler and the libraries used for this package are available for a great variety of platforms, the source codes of this package can be compiled for other platforms. More detailed information on compiler and libraries used can be found in the developer manual.

The applications `LGEM` and `INVERT` require the X windowing system. All except for visualization, the functions of `LINE2MAT` can be used without X.

8.3 Hardware Requirements

The hardware requirements are dependent on the sizes of the geometry describing the sensor system, as well as required resolution, and numerical method used.

The software package will minimally run on Pentium-I workstations with memory sufficient to work in the X environment. On the other hand, for large sensor systems (consisting of hundreds of sources and detectors), it will be necessary to use at least Pentium-III or Silicon Graphics (SGI) Octane workstations with a minimum of 256 Mb RAM. Although the size of the binaries is less than 3 Mb, it is recommended to have at least 2 Gb disk-space available to work with the software package efficiently.

8.4 Installation Package Contents

The full installation package consists of the following directories:

- ./**doc** – Documentation (Thesis, User Manual, Developer Manual);
- ./**doc/tex** – Sources of documentation including figures;
- ./**bin** – Binaries (Redhat Linux 7.1). *lgen* - LGEN; *line2mat* - LINE2MAT; *start* - LINE2MAT_INTERFACE; *invert* - INVERT; *data_sorter* - DATA SORTER (a small utility to sort “ir” and “v” data files when the user is switching from the old lines file to the sorted one). Run the program, paste the I/O filenames and click on “Create New Datafiles” button; *tiview* - TIVIEW (advanced TIFF image viewer) allows to manipulate center and width of the grayscale image, as well as change the magnification;
- ./**src** – Sources subdirectory;
- ./**src/lgen** – LGEN sources;
- ./**src/invert** – INVERT sources;
- ./**src/line2mat** – LINE2MAT sources;
- ./**src/line2mat_interface** – LINE2MAT_INTERFACE sources;
- ./**src/data_sorter** – DATA SORTER sources;
- ./**src/tiview** – TIVIEW sources;
- ./**examples** – Examples subdirectory;

- `./examples/lgen_projects` – Sample LGEN project;
- `./examples/lgen_pictures` – Sample output images of LGEN;
- `./examples/sample_images` – Sample images which can be used as simulated data (make sure that the image contains same number of pixels as the reconstruction grid);
- `./examples/lgen_lines_files` – Sample lines files;
- `./examples/line2mat_interface_projects` – Sample LINE2MAT_INTERFACE projects;
- `./examples/line2mat_output` – Sample output of LINE2MAT (matrices, singular value decomposition, images);
- `./examples/invert_projects` – Sample INVERT projects;
- `./examples/laboratory_data` – Laboratory data (right-hand side);
- `./examples/line2mat_output_rsv_images_noweight` – Sample images of right singular vectors when weighting was not used;
- `./examples/line2mat_output_rsv_images_weight` – Sample images of the right singular values with weighting;
- `./tmp` – Blank subdirectory;
- `./libs` – Libraries subdirectory;
- `./libs/sparselib_1_5d` – Sparse library (check online at <http://math.nist.gov/sparselib++/> for the newest version of the library).

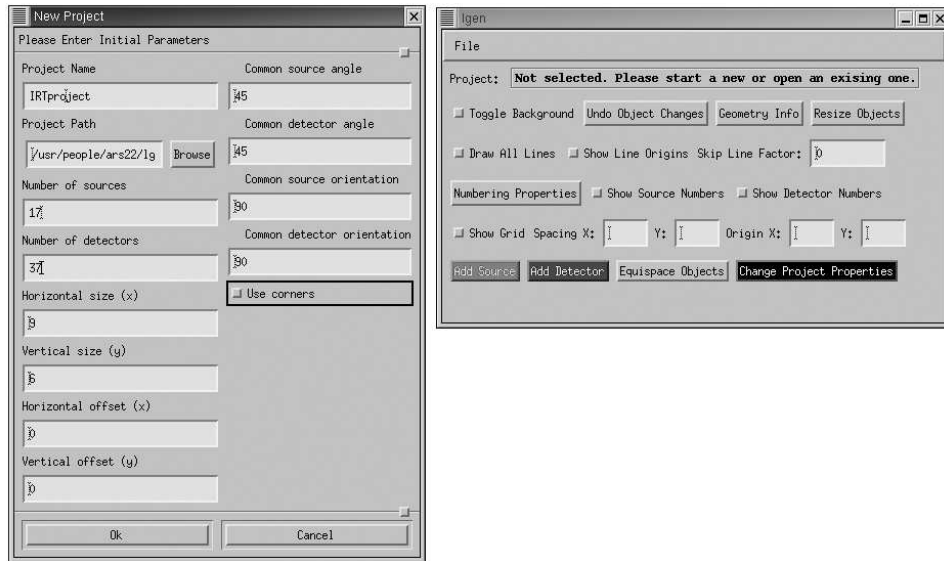


Figure 8.1: Starting a new LGEN project.

8.5 Using LGEN

8.5.1 Starting LGEN Session

To start LGEN, change the directory where this software package is installed and go to the subdirectory “*bin*”. Next, type “*./lgen*” in the command prompt.

8.5.2 Starting a New LGEN Project

To start a new LGEN project, first make sure that the current project (if any) is saved. Next, go to “File” menu and click on “New Project”. The “New Project” window will popup (See Figure 8.1).

Now you have to initialize the sensor system geometry. Enter a project name (if not specified, the project name will be “Untitled”). Enter the path to your project file (if not specified, the project path will be your current path). Enter the number of sources and detectors in the geometry (required

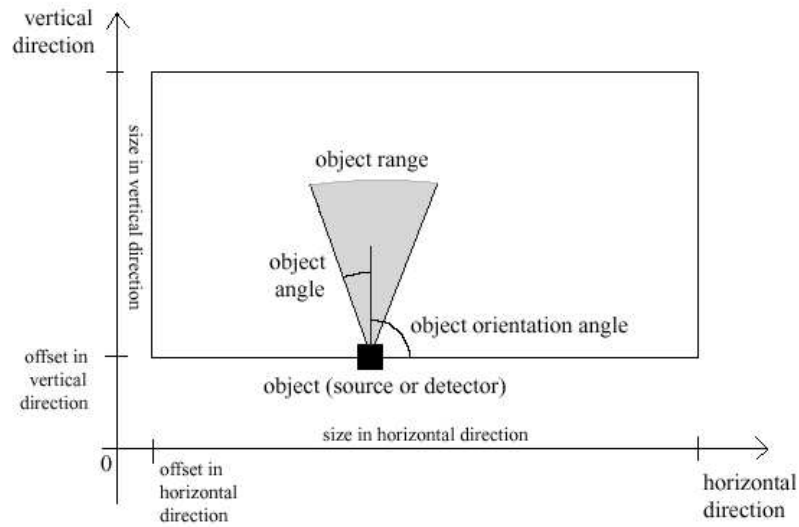


Figure 8.2: The illustration of major geometry initialization.

parameter; by default the maximum number of sources and the maximum number of detectors is 500). Refer to developer manual in order to change these maximum values. Enter the horizontal size of the frame (default is 9 units). Enter the vertical size of the frame (default is 6 units). Enter horizontal and vertical offsets of the frame (default offsets are 0). Enter the common source and detector angle in degrees (default angles are 45° , the selection range is $0^\circ - 90^\circ$). Enter the common source and detector orientation in degrees (default orientations are 90° , the selection range is $0^\circ - 180^\circ$). Finally, specify whether you allow placement of sources and detectors in the corners (See Figure 8.2).

Please keep in mind, that objects can be added and deleted, and all geometry properties can be changed later.

If you click “Ok”, LGEN will initialize the geometry, placing sources (red squares) and detectors (blue circles) regularly (with the same distance between

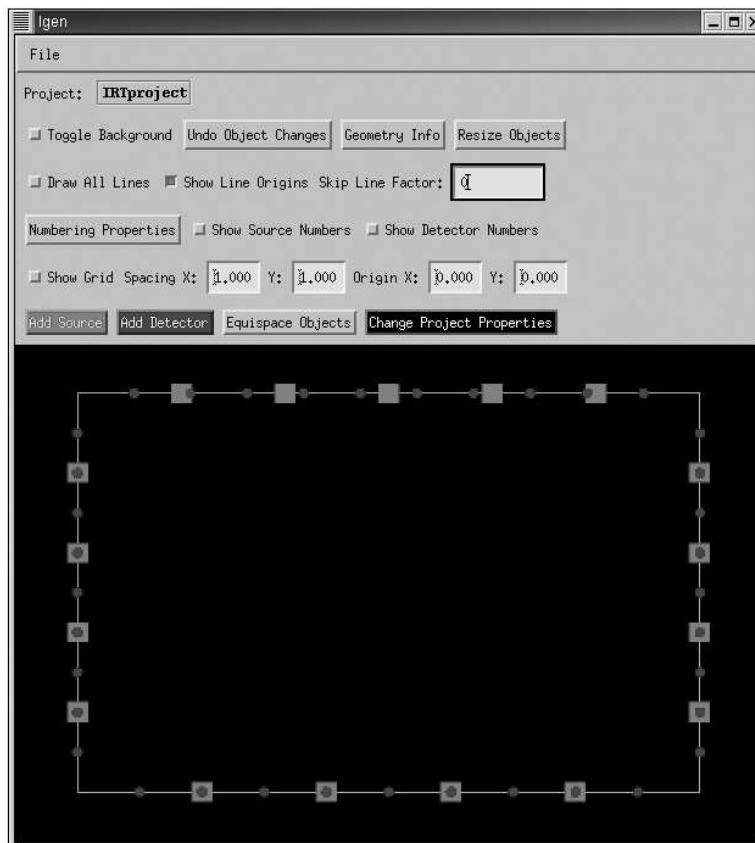


Figure 8.3: A new project is initialized.

the objects on each side) in the frame (See Figure 8.3).

8.5.3 Loading Existing LGEN Project

To open an existing LGEN project, first make sure that the current project (if any) is saved. Next, go to “File” menu and click on “Open Project”. Then browse a to project file with an “lpr” extension and open it. LGEN loads the project and displays the geometry including the selected (see the section on *showing lines for specific object*) lines (see *saving LGEN project* to understand what information is saved).

8.5.4 Drawing All Lines

In order to display all lines (green), including default (when the source and detector are not on the same wall and are in the range of each other - see Figure 1.3 in the thesis), and manually specified lines (see the sections on *connecting and disconnecting two objects* and *changing object line destinations*) check the toggle box “Draw All Lines”. To remove all lines (green) except for selected (magenta) (see the section on *showing lines for specific object*) uncheck the box “Draw All Lines”.

8.5.5 Showing Line Origins

The lines begin and end in the geometrical center of the objects represented either by a blue circle (detector) or a red square (source). By default, you can see the origins of lines. In order to deactivate the option of displaying line origins, uncheck the box “Show Line Origins”. Similarly, to display line origins, check this box. Notice, that you can see this feature only when the lines are shown on the screen.

8.5.6 Changing “Skip Line” Factor

While working with geometries with a large number of lines, it makes sense to reduce the number of displayed lines on the screen. In the text box “Skip Line Factor” you can enter the integer corresponding to the number of sources to skip while drawing lines on the screen. Press “Enter” to redraw the lines after you input the “skip line” factor.

The default “skip line” factor is 0 which corresponds to displaying all lines in geometry.

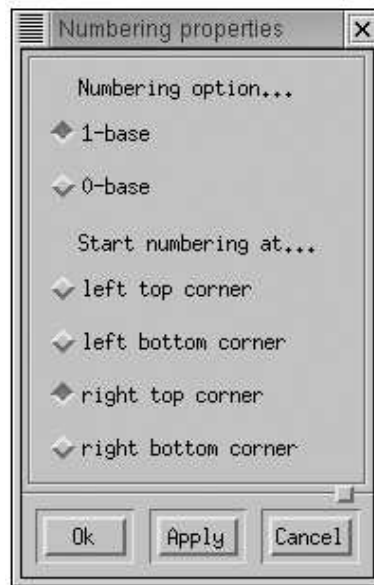


Figure 8.4: Specifying numbering properties.

8.5.7 Specifying Numbering Properties

For display purposes it is convenient to number sources and detectors. Click on “Numbering Properties” button, and the window with dialog menu will popup. You can select the option to start numbers either from 0 (0-based) or from 1 (1-based). Next, you can specify where to start numbering - either from the left top corner, or left bottom corner, or right top corner, or right bottom corner. The numbering will be done clockwise. Click “Apply” to number the objects, or click “Ok” to apply the changes and close the numbering dialog menu.

Refer to the section *showing source and detector numbers* to display the source and detector numbers according to the selected numbering properties.

See Figure 8.4.

8.5.8 Showing Source and Detector Numbers

To display source or/and detector numbers according to the numbering properties (see the section on *specifying numbering properties*), check the “Show Source Numbers” or/and “Show Detector Numbers” boxes. To remove object numbers from the screen, uncheck the corresponding box.

8.5.9 Showing Grid

To display the grid, check the “Show Grid” box. To remove the grid, uncheck that box. Refer to the sections on *changing grid spacing* and *changing grid origin* for additional grid display features.

8.5.10 Changing Grid Spacing

To change the grid X (horizontal) and Y (vertical) spacing, enter the required length of the grid interval in the floating-point format in the text fields “Spacing X ” and “ Y ” and then press “Enter” to redraw the grid. Make sure that the grid is displayed on the screen (refer to the section on *showing grid*).

8.5.11 Changing Grid Origin

To change the grid origin X (horizontal) and Y (vertical) coordinates, enter the new coordinates of the origin in the floating-point format in the text fields “Origin X ” and “ Y ” and then press “Enter” to redraw the grid. Make sure that the grid is displayed on the screen (refer to the section on *showing grid*).

8.5.12 Toggling Background

Sometimes it is more convenient to use a white background instead of black. Check the “Toggle background” box to select the white background, and uncheck it to select the black one. Notice, that this option affects the TIFF output (i.e. the TIFF image will have the same background as displayed on the screen) and does not affect the EPS output (EPS image will always have white background).

8.5.13 Selecting Object

To select a source or detector, click on it with a left button of the mouse. The selected object will be highlighted. The selection is required for moving the object along the frame, and connecting or disconnecting the object with another object (see corresponding sections for more information).

8.5.14 Moving Object Along the Frame

In order to change the position of the object, you can either move the selected object (refer to the section on *selecting object*) along the frame holding the left button of the mouse or specify its position in the “Object properties” popup dialog menu (see the section on *changing object properties*). In all cases the object coordinates should belong to the frame.

Notice, that two objects should never have the same coordinates. Otherwise, LGEN will produce an error message and return the object to the original position.

Refer to the section on *undoing object movement/removal* to undo the object movement.

8.5.15 Changing Object Properties

To change the basic properties of the object you have to right-click on it. The “Object properties” dialog menu will popup. You can input X and Y coordinates in the floating-point format. Similarly you can change the object orientation (ranging from 0° to 180° degrees) and angle (ranging from 0° to 90° degrees) as shown in the Figure 8.2. Moreover, you can display lines for the object (refer to the section on *showing lines for specific object*), remove it (see the section on *removing object*), or change its line destinations (see *changing object line destinations*).

Refer to the section on *undoing object movement/removal* to undo the object removal, movement, or change of the coordinates or angles.

Click “Apply” to apply the changes or click “Ok” to apply the changes and close the dialog menu.

See Figure 8.5.

8.5.16 Adding Object

To add a new source to the sensor system, click on the red button “Add Source”. To add a new detector, click on the blue button “Add Detector”. The new object will be placed in the middle of the right side of the frame (representing the 0° coordinate on the imaginary circle intersecting the frame in the corners). To remove, move, or change properties of the object, please refer to the corresponding sections.

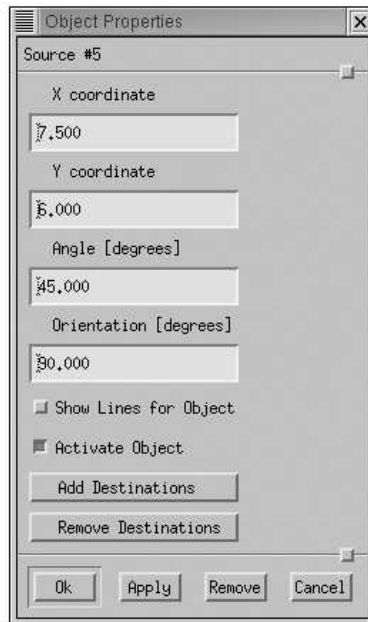


Figure 8.5: Changing object properties.

8.5.17 Removing Object

To remove the object, right-click on it. The “Object properties” dialog menu will popup. Click on “Remove” button to remove the object.

Refer to the section on *undoing object movement/removal* to undo the object removal.

8.5.18 Undoing Object Movement/Removal

To undo the latest movement/removal of the object, as well as changes of its angles or coordinates, click on the “Undo Object Changes” button. Notice, that there is no redo function implemented.

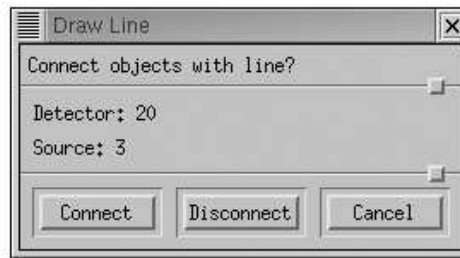


Figure 8.6: Connecting and disconnecting two objects.

8.5.19 Connecting and Disconnecting Two Objects

In order to manually connect or disconnect the selected object (refer to the section on *selecting object*) with the object of the different type (i.e. source with detector or vice versa) located not on the same side of the frame which is not necessary in the range of the first object, hold “Shift” key and left-click with a mouse on the second object. The dialog menu “Draw line” will popup. You can either click on “Connect” button to ensure that these objects are always connected regardless of their angles and orientations. Similarly, you can click on “Disconnect” button to disconnect these objects if they are already connected and to make sure that these objects will be never connected independent of their angles and orientations.

The line generated or removed by this procedure or as described in the section on *changing object line destinations* will be referred to as “manually specified” or “manually deleted”.

See Figure 8.6.



Figure 8.7: Changing object line destinations.

8.5.20 Changing Object Line Destinations

To specify a permanent connection or disconnection of the object with a range of objects of the different type located not on the same side of the frame independent of angles and orientations you have to right-click on the object.

The “Object properties” dialog menu will popup. To connect the object, select “Add Destinations” (see Figure 8.5). Then input the range of destination objects and either click “Ok” or “Apply”. Similarly, to disconnect the object, select “Remove Destinations” and repeat the procedure described above.

The lines generated or removed by this procedure or as described in the section on *connecting and disconnecting two objects* will be referred to as “manually specified” or “manually deleted”.

See Figure 8.7.

8.5.21 Showing Lines for Specific Object

To show lines for the specific object, right-click on the object. The “Object properties” dialog menu will popup. Check the “Show Lines for Object” box. Uncheck that box to undisplay the lines. These lines appear in magenta.

Notice, that this is just a visualization function. It does not actually connect or disconnect any objects. This function simply displays the existing connections.

This functions is especially useful if you have a large sensor system, and you want to see only the lines for specific objects (say, on one side of the frame - refer to the Figure 2.3 for the example). Uncheck the “Draw All Lines” box (otherwise the magenta lines will be drawn on top of green lines), and display lines for specific objects as described above.

The lines displayed in this way are referred to as “selected”.

8.5.22 Deactivating or Activating the Object

To deactivate or activate the object, right-click on it. The “Object properties” dialog menu will popup. Either check or uncheck the “Activate Object” box depending on whether you want to activate or deactivate the object. By default, all objects are activated. The fact that some object is deactivated means that it is not used for line connections.

8.5.23 Equispacing Objects

To equispace either sources or detectors (or both) on a side, click on “Equispace objects” button. The dialog menu “Equispace objects” will popup. Select whether you allow to place objects in the corners or not. Next, select the sources on specific sides of the frame you want to equispace. You can do similar for the detectors. Click “Apply” to apply the selected equispacing or click “Ok” to equispace and close the dialog menu.

See Figure 8.8.

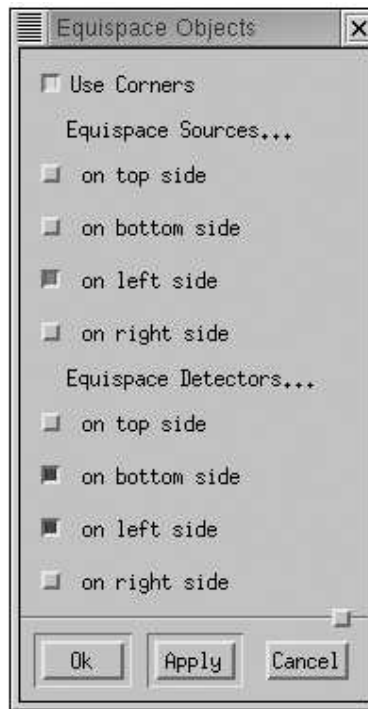


Figure 8.8: Equispacing objects.

8.5.24 Changing Project Properties

The project “global” properties input while creating a new project (see the section on *starting a new LGEN project*) can be changed any time during the LGEN session. To do this, click on the black button “Change Project Properties”. The “Project properties” window will popup. Make the necessary changes and click “Ok”.

Changing common sources and angles does not affect manually added and deleted connections.

See Figure 8.9.

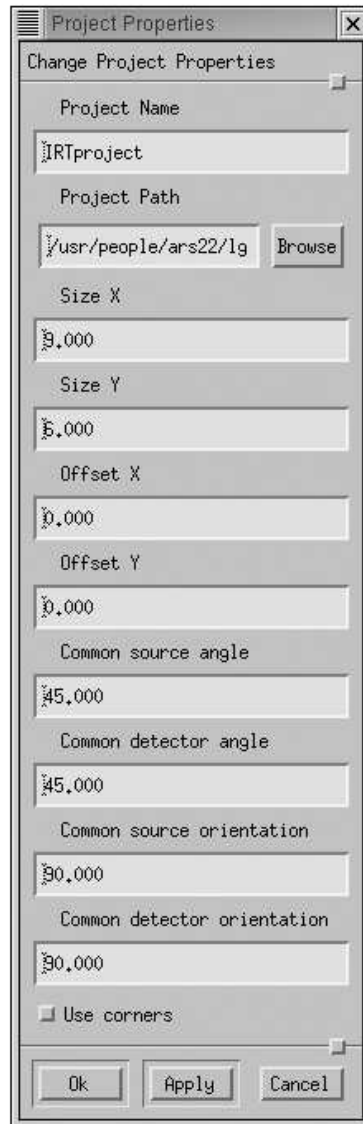


Figure 8.9: Changing project properties.

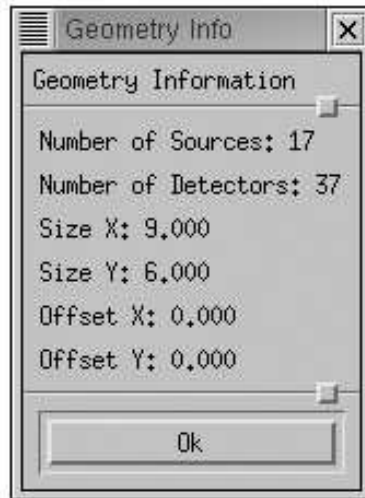


Figure 8.10: Viewing geometry information.

8.5.25 Viewing Geometry Information

To view the basic geometry information - the number of sources, the number of detectors, and the vertical and horizontal sizes and offsets of the frame, click on “Geometry Info” button. The corresponding dialog window will popup.

See Figure 8.10.

8.5.26 Resizing Objects

Sometimes the default sizes of the red squares (representing sources) and blue circles (representing detectors) are inconvenient. In order to change the sizes of the objects, click on “Resize Objects” button. The corresponding dialog menu will popup. Use the sliders to adjust the sizes of the objects, selecting their sizes in percents from the current sizes. Click “Apply” to do the resizing or click “Ok” to resize the objects and close the dialog menu.

See Figure 8.11.

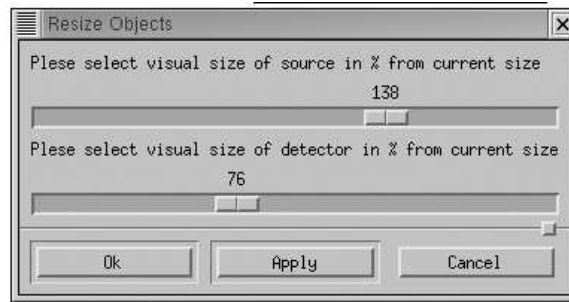


Figure 8.11: Resizing objects.

8.5.27 Resizing LGEN Window

The LGEN window like many other windows in X can be resized using standard techniques. Upon resizing, LGEN preserves the aspect ratio of the work area.

8.5.28 Saving the Lines File

The geometry line configuration can be saved for the future usage by LINE2MAT. Go to “File” menu and click on “Save Lines as...”. Select a lines file with a “txt” extension. Click “Ok”. The dialog menu “Save lines sorted by” will popup. Select the ordering of line records in the lines file. The lines can be either sorted by sources, detectors, distance from center of the frame (so-called “frame origin”), or angle. Refer to the chapter on mathematical model of the thesis for more information. Click “Ok” to save the lines file using the selected sorting mode.

The lines file is an ASCII file with lines consisting of four double-precision numbers (separated by spaces) which define the coordinates of the two objects connected by a line.

Notice, that all lines - default and manually specified (refer to the section on *drawing all lines*) - will be saved. If the object is deactivated (see

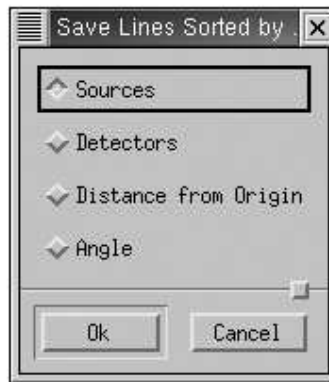


Figure 8.12: Saving lines file.

the section on *deactivating or activating the object*), no lines will connect it with the other objects; hence, this object will not “appear” in the lines file.

See Figure 8.12.

8.5.29 Saving TIFF File

A colored screen-shot of the work area of the screen containing the frame with objects can be saved in the TIFF image format with Macintosh RLE compression scheme (so-called “Packbits”). In order to have a “printer-friendly” image, it is recommended to toggle the work area background to the white (refer to the section on *toggling background*) before saving the image.

To save the TIFF image, go to “File” menu and click on “Save TIFF as...”. Select the image file with “tiff” extension. Click “Ok” to save the image.

Notice, that the resulting TIFF image will be a screen-shot (bitmap) of the drawing area of the screen. To use the maximum resolution available by

the output device (such as a printer),

save the image in postscript format. Refer to the section on *saving encapsulated postscript file*.

8.5.30 Saving Encapsulated Postscript File

The image of the work area of the screen containing frame with objects can be saved in encapsulated postscript format (EPS). EPS is a standard format for importing and exporting PostScript language files in all environments. It is a single page PostScript language program that describes an illustration using vector graphics. Using EPS, the resolution of the print-out or screen-image is defined by the user and is bounded by the maximum output device (such as printer) resolution. A detailed description of PostScript can be found in [1]. The EPS format is preferable to the bitmapped TIFF format (see the section on *saving TIFF file*).

To save the EPS image, go to “File” menu and click on “Save EPS as...”. Select the image file with “eps” extension. Click “Ok” to save the image.

8.5.31 Saving LGEN Project

The current LGEN project can be saved in the ASCII file with the “lpr” extension.

The project file contains complete state information for LGEN. Included are the version of LGEN used; file creation date and time; number of sources; number of detectors; horizontal (X) size of the frame; vertical (Y) size of the frame; horizontal (X) offset of the frame; vertical (Y) offset of the frame; flag

indicating whether the placement of objects in the corners is allowed or not; visual size (as seen on the screen) of source; visual size (as seen on the screen) of detector; common source angle; common detector angle; common source orientation; and common detector orientation (see Figure 8.2).

Then information about each source is listed in the following format: source X coordinate (including offset), source Y coordinate (including offset), flag indicating whether this source is active or not (refer to the section on *deactivating or activating the object*); source orientation; source angle; flag indicating whether the source angle was manually defined or not (see the section on *changing object properties*); and the flag indicating whether the source orientation was manually defined or not (see the section on *changing object properties*). The same information about detectors is listed after the source data.

Next, the file contains numbers of manually specified and deleted lines (see the sections on *changing object line destinations* and *connecting and disconnecting two objects*). After that the file defines these lines by four double-precision numbers.

In order to save the project with the same filename (with “lpr” extension) to the same (given or default) path, go to “File” menu and click on “Save Project”.

To save the project to a different file or path, go to “File” menu and click on “Save Project as...”. Select the file with “lpr” extension. Click “Ok” to save the project.

8.5.32 Ending LGEN Session

To end LGEN session, first make sure that you saved your project. Note that all unsaved changes to the project will be discarded. Then go to “File” menu and click on “Quit”.

8.6 Using LINE2MAT

8.6.1 General Principles

In order to run LINE2MAT, change directory to the where this software package is installed and go to the subdirectory “*bin*”. Next, type “*./line2mat*” and the options string (see below) in the command prompt.

There are two possible modes of running LINE2MAT - dense mode (referred to as “dense task”) and sparse mode (referred to as “sparse task”). The data is processed and stored in dense and sparse formats respectively. See Appendix C for more information on sparse storage formats. By default, LINE2MAT runs in the dense mode. Using the option **-sparse_task**, one can switch to the sparse mode. For each mode, the user can either create a new matrix A according to the specified lines file (see the sections on *starting a new “dense task”: required options* and *starting a new “sparse task”: required options*) or load the existing matrix (see the sections on *loading an existing “dense task”: required options* and *loading an existing “sparse task”: required options*) and work with it (see the sections on *additional options for a “dense task”* and *additional options for a “sparse task”*).

Although most functions in LINE2MAT do not require a GUI (hence can be used without X), the display functions require the X windowing system.

When working without X, make sure that you read carefully the description of the options to avoid usage of the GUI functions.

In the tables below “INT” refers to a positive integer number, “SINT” refers to a nonzero signed integer number, and “DBL” refers to a floating-point double-precision number.

8.6.2 Starting a New “Dense Task”: Required Options

To create a new matrix A in dense format, the user has to specify the lines file using the **-lines** option. Please refer to the preceding chapters of the thesis for the algorithm used for generating the matrix A .

The lines file is an ASCII file containing lines coordinates. The user can either use lines files created by LGEN or create them by other means.

In addition to the lines file, the user has to specify the horizontal and vertical sizes (in number of pixels) of the frame using **-sizex** and **-sizey** options respectively. Although often the values specified in **-sizex** and **-sizey** match the sizes of the frame (in units) used in the lines file, sometimes it is useful use a different number of pixels, thereby changing the resolution. In order to accomplish this, refer to the options **-pixelx** and **-pixely** described in the section on *additional options for a “dense task”*.

See Table 8.1.

8.6.3 Loading an Existing “Dense Task”: Required Options

To load an existing dense matrix A , the user has to use the option **-matrix** and specify the compressed dense matrix file (generated *a priori* by instance of

Usage	Option Description
-lines LINESFILE	An ASCII lines file containing line endpoints
-sizex INT	Integer number of pixels along the horizontal direction
-sizey INT	Integer number of pixels along the vertical direction

Table 8.1: Starting a new “dense task”. Required options.

Usage	Option Description
-matrix MATRIXFILE	Open an existing matrix A . Options -lines , -sizex , -sizey , -offsetx , -offsety , -pixelx , -pixely are neglected

Table 8.2: Loading an existing “dense task”. Required option.

LINE2MAT) containing matrix elements.

Notice, that since the matrix is already generated, all options **-lines**, **-sizex**, **-sizey**, **-offsetx**, **-offsety**, **-pixelx**, and **-pixely** will be ignored.

See Table 8.2.

8.6.4 Additional Options for a “Dense Task”

The options listed here have to be used together with the required options (see the sections on *starting a new “dense task”: required options* and *loading an existing “dense task”: required options*).

The options **-pixelx** and **-pixely** control the sizes of the pixels (floating point values are allowed) in the reconstruction basis. By default, 1 by 1 pixel is used.

Use the options **-offsetx** and **-offsety** to specify the horizontal and

vertical offsets of the frame respectively (floating point values are allowed).

In order to view the matrix A on the screen, use the option **-view**. The option **-mag** followed by a nonzero integer number specified the magnification. Positive magnification values result in scaling up the image, while negative values result in scaling down the image. Notice, that this is a GUI option, and X is required for it.

The internal viewer displays the images in grayscale (see Figure 8.13). The user has options to change the grayscale center and grayscale width parameters using the sliders. To return to the original values of center and width, click on “Reset” button. Each time the sliders are moved, the viewer updates the image. When the image is large or the computer is not fast enough, it is recommended to uncheck the “Auto-scale” box. This controls whether the image is updated synchronously with the motion of the slider (when checked), or only with the final position of the slider. Finally, the “Status” string represents the image pixel coordinates and the scaled value from 0 (black) to 255 (white).

To save the matrix A in compressed format, use the option **-save** followed by the filename without an extension. The “gz” extension will be automatically added to the filename. Notice, that this matrix can be later retrieved with the option **-matrix** (see the section on *loading an existing “dense task”: required options*).

To compute and save (for the later use in **INVERT**) the singular value decomposition (SVD) of the matrix A , use the option **-svd** followed by the base filename without extension. The singular value decomposition will be

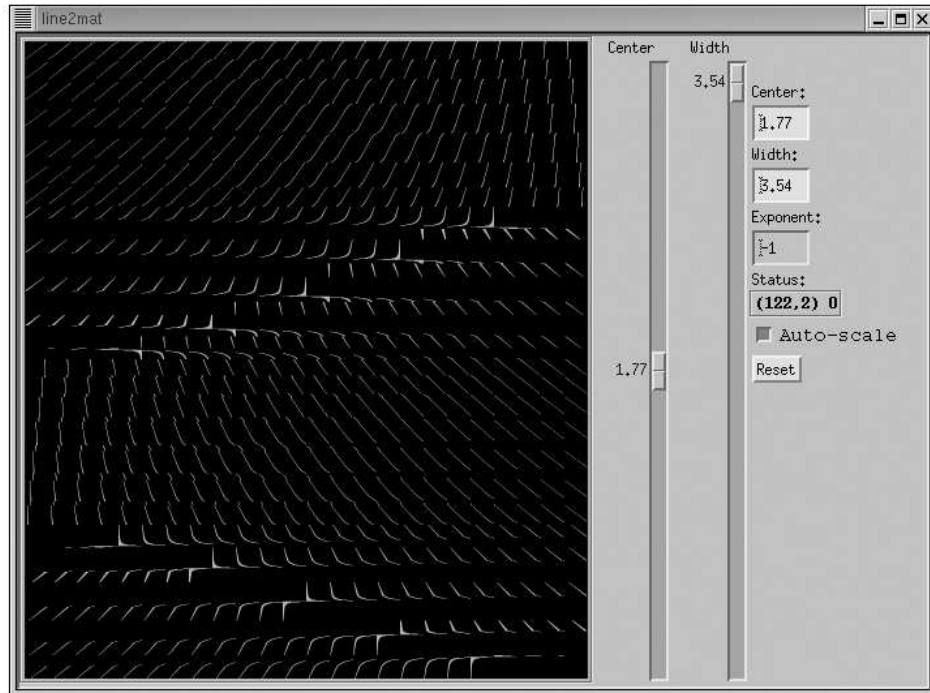


Figure 8.13: Viewing projection matrix A on the screen. Using internal viewer.

computed and stored in the files with the base filename and extensions “sv” (singular values in ASCII format), “rsv.gz” (right singular vectors in compressed binary format), and “lsv.gz” (left singular vectors in compressed binary format). To compute the SVD of the matrix $W^{1/2}A$ and save $U^T W^{1/2}$ instead of U^T (left singular vectors), use the option **-wsvd** together with **-svd**. Refer to the preceding chapter of the thesis on theory and implementations of singular value decomposition. Note that **-svd** is the most time consuming and memory intensive function of LINE2MAT.

To view a certain right singular vector as the grayscale image, use the option **-view_rsv** followed by number (starting with 0) of the singular vector. If no number is specified, the first right singular vector will be displayed. To change the magnification of the image, use **-mag_view_rsv** followed by a

positive integer magnification. The default magnification is 10. Make sure that you compute the singular value decomposition (i.e. use **-svd** option) while using this option. Notice, that X is required for this option.

Use the option **-save_rsv** together with the **-mag_save_rsv** option followed by a positive magnification (default magnification is 10) to save the right singular vectors in TIFF format. The TIFF files are saved with the base filename specified in the **-svd** option followed by the singular vector number and “tiff” extension. Make sure that you compute the singular value decomposition (i.e. use **-svd** option) while using this option.

In order to save the reciprocals of diagonal elements of the weighting matrix $W^{1/2}$ in compressed binary format, one can use the option **-wsave** followed by the filename without extension. The “gz” extension will be automatically added to the filename.

To load the reciprocals of the diagonal elements of the weighting matrix $W^{1/2}$, use the option **-wload** followed by the filename.

To view the matrix $A^T A$ on the screen in grayscale, use the option **-view_ata** together with the option **-mag_view_ata**. The default magnification is 1. Notice, that X is required for this option.

To save the matrix $A^T A$ as the TIFF grayscale image, use the option **-ata_filename** followed by the filename without an extension. The extension “tiff” will be automatically added to the filename. This option may be used with **-mag_save_ata** followed by a positive integer magnification. The default magnification is 1.

The matrix $A^T A$ can be also saved in the binary compressed format for

the later use in `INVERT`. Use the option `-ata_dump` followed by the filename without an extension. The extension “gz” will be automatically added to the filename.

`LINE2MAT` offers options for generation of simulated data. The input data can be supplied in TIFF file(s) and projected (i.e. the matrix-vector product - $A \times$ input data - is formed) to obtain the right-hand side(s) b for use with `INVERT`. Using of TIFF files is convenient, since we can visually compare the original image with the image reconstructed by `INVERT`, as well as compute the residuals (see user manual on `INVERT`).

To project a single TIFF file, use the option `-project_d_data` followed by the TIFF filename. The TIFF file should have the product of dimensions equal to the number of pixels in the reconstructed image (i.e. the product of values specified in the options `-sizex` and `-sizey`). The simulated right-hand side b will be saved to the regular binary file with “v” extension to the filename specified in the `-project_d_output` option.

To project two TIFF files simultaneously, use the option `-project_iw_ice` followed by the first TIFF filename (ice image) and `-project_iw_water` followed by the second TIFF filename (water image). Both TIFF files should have the product of dimensions equal to the number of pixels in the reconstructed image (i.e. the product of values specified in the options `-sizex` and `-sizey`). The simulated right-hand sides will be saved to binary files with “v” and “ir” extensions with the base filename specified in the `-project_iw_output` option.

Finally, while working in dense mode in `LINE2MAT`, one can also save

the matrix A in the sparse format using **-sparsify** option followed by the filename. The extension “gz” will be automatically added to that filename.

Please refer to the developer manual on the details of the file formats used.

See Table 8.3 and Table 8.4.

8.6.5 Starting a New “Sparse Task”: Required Options

To create a new matrix A in the sparse mode, use **-sparse_task** and specify the lines file using **-lines** option. Please refer to the preceding chapter of the thesis for the algorithm used for generation of the matrix A .

The lines file is an ASCII file containing lines coordinates. The user can either use a lines files created by LGEN or create one by other means.

Notice, that LINE2MAT will perform a sparse task if the matrix is indeed sparse. That is, an m by n matrix A is considered to be sparse if and only if its number of nonzero elements does not exceed the integer part of the number $1.2(mn)^{0.8}$. This limit is also used for memory allocation for the sparse matrix. The “magic” number $1.2(mn)^{0.8}$ was derived empirically based on more than 200 experiment sparsity values (corresponding to lines files generated by LGEN) analyzed in Matlab.

In addition to the lines file, the user has to specify the horizontal and vertical sizes (in number of pixels) of the frame using **-sizex** and **-sizey** options respectively. Although often the values specified in **-sizex** and **-sizey** match the sizes of the frame (in units) used in the lines file, sometimes it is useful use different number of pixels changing the resolution. In order to accomplish this, refer to the options **-pixelx** and **-pixely** described in the

Usage	Option Description
-pixelx DBL	Positive horizontal size DBL of the pixel (default=1)
-pixely DBL	Positive vertical size DBL of the pixel (default=1)
-offsetx DBL	Horizontal offset DBL of the frame (default=0)
-offsety DBL	Vertical offset DBL of the frame (default=0)
-view	View projection matrix A on the screen; X required
-mag SINT	Nonzero magnification SINT for viewing matrix A (default=1)
-save MATRIXFILE	Save matrix A in compressed format to the file MATRIXFILE.gz
-svd SVDFILE	Compute and save dense SVD of the matrix A
-wsvd	Compute and save SVD of the weighted matrix. $U^T W^{1/2}$ will be saved instead of U^T
-view_rsv INT	View a right singular vector (first vector is default) number INT (zero-based); SVD required; X required
-mag_view_rsv INT	Positive integer magnification INT for viewing the selected right singular vector (default=10)
-save_rsv	Save right singular vectors as TIFF files as SVDFILEXXXX.tiff (XXXX represent number); SVD required
-mag_save_rsv INT	Positive integer magnification INT for saving right singular vectors (default=10); SVD required
-wsave WEIGHTFILE	Save weighting matrix $W^{1/2}$ in compressed format to the file WEIGHTFILE.gz
-wload WEIGHTFILE	Load weighting matrix $W^{1/2}$ from the file WEIGHTFILE.gz
-view_ata	View matrix $A^T A$ on the screen; X required
-mag_view_ata SINT	Nonzero magnification SINT for viewing $A^T A$ (default=1)
-ata_filename ATAFILE	Save $A^T A$ in TIFF format to the file ATAFILE.tiff
-mag_save_ata INT	Positive magnification INT for saving $A^T A$ in TIFF format (default=1)
-ata_dump ATAFILE	Save matrix $A^T A$ in compressed format to the file ATAFILE.gz

Table 8.3: Additional options for a “dense task” (Part 1).

Usage	Option Description
<p>-project_d_data DATAFILE</p> <p>-project_d_output OUTPUTFILE1</p> <p>-project_iw_ice ICEFILE</p> <p>-project_iw_water WATERFILE</p> <p>-project_iw_output OUTPUTFILE2</p> <p>-sparsify SPARSEFILE</p>	<p>Project data: project matrix A on the TIFF image DATAFILE</p> <p>Project data: save projection output (matrix-vector product) to the file OUTPUTFILE1.v</p> <p>Project ice and water images: input ice TIFF file ICEFILE</p> <p>Project ice and water images: input water TIFF file WATERFILE</p> <p>Project ice and water images: output file base name OUTPUTFILE2 where matrix-vector products ($A \times$ input file data) will be saved</p> <p>Sparsify and create compressed Harwell-Boeing file SPARSEFILE.gz from the current dense matrix</p>

Table 8.4: Additional options for a “dense task” (Part 2).

Usage	Option Description
-sparse_task	Indicate the sparse task
-lines LINESFILE	An ASCII lines file containing line endpoints
-sizex INT	Integer number of pixels along the horizontal direction
-sizey INT	Integer number of pixels along the vertical direction

Table 8.5: Starting a new “sparse task”. Required options.

Usage	Option Description
-sparse_task	Indicate the sparse task
-matrix MATRIXFILE	Open an existing matrix A . Options -lines , -sizex , -sizey , -offsetx , -offsety , -pixelx , -pixely are neglected

Table 8.6: Loading an existing “sparse task”. Required option.

section on *additional options for a “sparse task”*.

See Table 8.5.

8.6.6 Loading an Existing “Sparse Task”: Required Options

To load an existing sparse matrix A , use **-sparse_task** together with **-matrix** and specify the compressed sparse matrix file (generated by LINE2MAT) containing matrix elements.

Notice, that since the matrix is already generated, the options **-lines**, **-sizex**, **-sizey**, **-offsetx**, **-offsety**, **-pixelx**, and **-pixely** will be ignored.

See Table 8.6.

8.6.7 Additional Options for a “Sparse Task”

The options listed here have to be used together with the required options (see the sections on *starting a new “sparse task”: required options* and *loading an existing “sparse task”: required options*).

The options **-pixelx** and **-pixely** control the sizes of the pixel (floating point values are allowed) - the basis of image reconstruction. By default a 1 by 1 pixel is used.

The options **-offsetx** and **-offsety** specify the horizontal and vertical offsets of the frame respectively (floating point values are allowed).

To save the matrix A in compressed sparse format, use **-save** followed by the filename without an extension. The “gz” extension will be automatically added to the filename. Notice, that this matrix can be later retrieved with **-matrix** (see the section on *loading an existing “sparse task”: required options*).

In order to save the weighting matrix $W^{1/2}$ in compressed binary format, use the option **-wsave** followed by the filename without an extension. The “gz” extension will be automatically added to the filename.

To load the weighting matrix $W^{1/2}$, use the option **-wload** followed by the filename.

LINE2MAT offers options for generating simulated data. The input data can be supplied in TIFF file(s) and projected (i.e. the matrix-vector product $A \times$ input data - is formed) to obtain right-hand side(s) b for use with **INVERT**. Using of TIFF files is convenient, since we can visually compare the original image with the image reconstructed by **INVERT**, as well as compute the

residuals (see user manual on `INVERT`).

To projection a single TIFF file, use **-project_d_data** followed by the TIFF filename. The TIFF file should have a product of dimensions equal to the number of pixels in the reconstructed image (i.e. the product of values specified in the options **-sizex** and **-sizey**). The simulated right-hand side b will be saved a regular binary file with “v” extension to the filename specified in the **-project_d_output** option.

To projection two TIFF files at the same time, use **-project_iw_ice** followed by the first TIFF filename (ice image) and **-project_iw_water** followed by the second TIFF filename (water image). Both TIFF files should have a product of dimensions equal to the number of pixels in the reconstructed image (i.e. the product of values specified in the options **-sizex** and **-sizey**). The simulated right-hand sides will be saved to regular binary files with the base filename specified in the **-project_iw_output** option and “v” and “ir” extensions.

Please refer to the developer manual for the details of the file formats used.

See Table 8.7.

8.6.8 LINE2MAT_INTERFACE

Although users may prefer to use command line interface for `LINE2MAT`, others may prefer a more convenient graphical user interface application, `LINE2MAT_INTERFACE`. This interface program performs elementary input verification and in case of incomplete or incorrect input produces an error message. The users can also easily save and load their tasks. Given the

Usage	Option Description
-pixelx DBL	Positive horizontal size DBL of the pixel (default=1)
-pixely DBL	Positive vertical size DBL of the pixel (default=1)
-offsetx DBL	Horizontal offset DBL of the frame (default=0)
-offsety DBL	Vertical offset DBL of the frame (default=0)
-save MATRIXFILE	Save matrix A in compressed format to the file MATRIXFILE.gz
-wsave WEIGHTFILE	Save weighting matrix $W^{1/2}$ in compressed format to the file WEIGHTFILE.gz
-wload WEIGHTFILE	Load weighting matrix $W^{1/2}$ from the file WEIGHTFILE.gz
-project_d_data DATAFILE	Project data: project matrix A on the TIFF image DATAFILE
-project_d_output OUTPUTFILE1	Project data: save projection output (matrix-vector product) to the file OUTPUTFILE1.v
-project_iw_ice ICEFILE	Project ice and water images: input ice TIFF file ICEFILE
-project_iw_water WATERFILE	Project ice and water images: input water TIFF file WATERFILE
-project_iw_output OUTPUTFILE2	Project ice and water images: output file base name OUTPUTFILE2 where matrix-vector products ($A \times$ input file data) will be saved

Table 8.7: Additional options for a “sparse task”.

consistent input, `LINE2MAT_INTERFACE` runs `LINE2MAT`. After the completion, the users receive an acknowledgement.

`textttLINE2MAT_INTERFACE` is located in the *bin* subdirectory in the root folder of this software package. It is started by typing `“./start”` or `“./start TASKFILE”` at a command prompt. In the second case, an existing task will be loaded (see information on task file below).

Since `LINE2MAT_INTERFACE` is a GUI application, it requires X.

Similarly to `LINE2MAT`, `LINE2MAT_INTERFACE` operates in two modes - “sparse task” and default task (“dense task”). The user can switch from one mode to another by checking or unchecking the “Sparse Task” box. The user does not have to think about what options are supported by a sparse mode (since `LINE2MAT` has limited support for the “sparse functions”) - the program takes care of it (compare the Figure 8.14 and Figure 8.15).

The user can either select an existing projection matrix A to work with (refer to the sections on *loading an existing “dense task”: required options* and *loading an existing “sparse task”: required options*) or generate a new projection matrix A (refer to the sections on *starting a new “dense task”: required options* and *starting a new “sparse task”: required options*) by selecting either “Use existing projection matrix” or “Compute new projection matrix A ”, respectively.

For an existing matrix, the user has to supply only the filename of the corresponding matrix file. For the new matrix, the user has to specify the lines file and horizontal (“Size X”) and vertical (“Y”) sizes of the frame. The required options for the generation of a new matrix are marked with red stars.

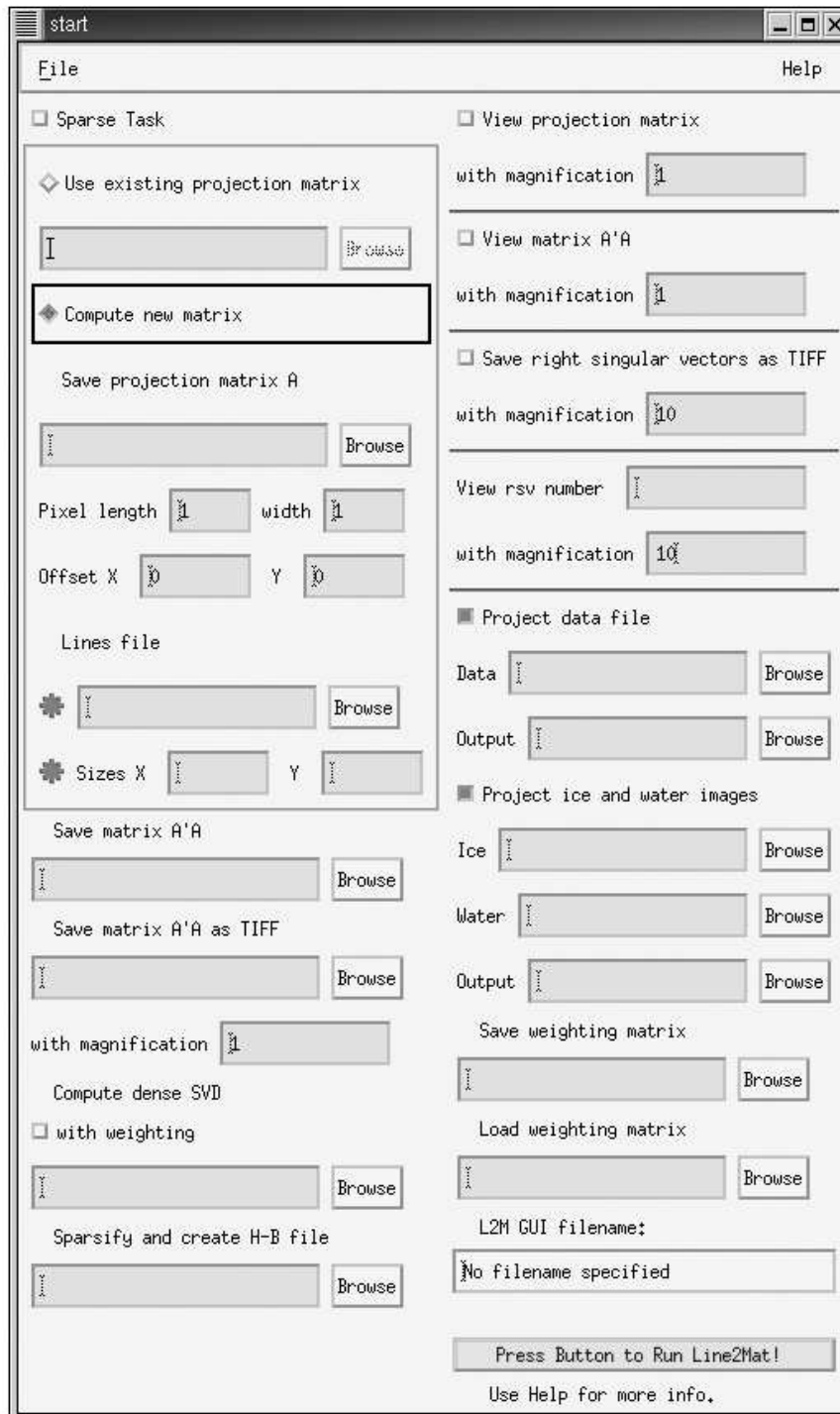


Figure 8.14: LINE2MAT_INTERFACE: Dense task options.

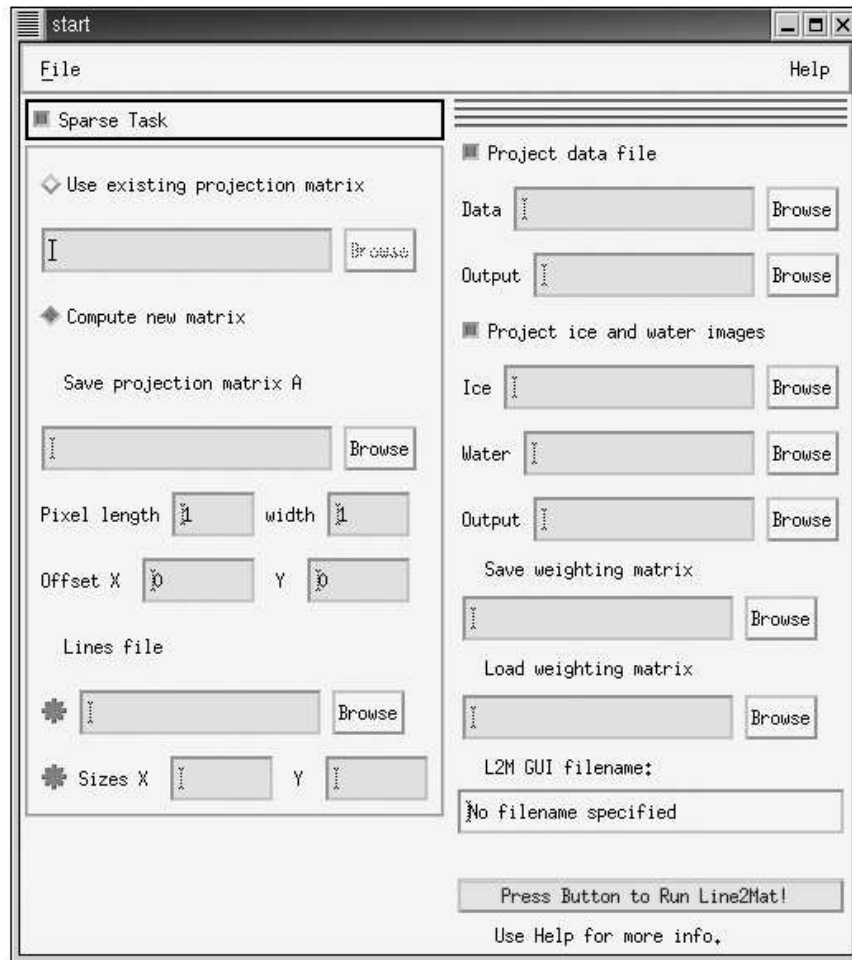


Figure 8.15: LINE2MAT_INTERFACE: Sparse task options.

The user can also save the projection matrix to a compressed binary file, specify offsets and pixel sizes (see the frame in the left part of the Figure 8.14 and Figure 8.15). Please refer to the documentation of LINE2MAT for the additional information.

While working in the dense mode, matrix $A^T A$ can be saved in both compressed binary or TIFF (with user-defined magnification) formats. Moreover, the singular value decomposition can be computed (with or without weighting), or save the matrix A in sparse formats. Please refer to the bottom left part of the Figure 8.14.

In the dense mode, the matrices A and $A^T A$, as well as a selected right singular vector (if SVD is computed) can be viewed on the screen with a user-selected magnification. All right singular vectors can be also saved in TIFF format with a user specified magnification (if SVD is computed). Please refer to the top right part of the Figure 8.14.

In both dense and sparse modes, data file can be projected by specifying a TIFF image as input, and specifying the output file without an extension. Ice and water images can be projected by specifying two TIFF images corresponding to the ice and water respectively, as well as the output file without an extension. Moreover, one can either load or save the weighting matrix. Please refer to the bottom right part of the Figure 8.14 and right part of the Figure 8.15.

The task can be saved (using the menu) to an ASCII file which can be edited by any text editor. A task file can be loaded either from the command string while starting LINE2MAT_INTERFACE, or using the menu. The current

task filename and creation date will be displayed in the bottom right part of the screen. If the task is not saved, the user will see “No filename specified” message in the corresponding field (see Figure 8.14 and Figure 8.15).

When the button “Press Button to Run Line2Mat!” is pressed, the program will check the input fields and launch `LINE2MAT`, if the input is consistent and complete.

The `LINE2MAT_INTERFACE` task file format is shown in the left column of the Table 8.8. The task file usually starts with several user comments lines beginning with “//” which are ignored. Each option is followed by a double-colon. The user has to specify the option value after the double-colon. For the options without value, specify either “TRUE” or “FALSE” (or do not specify the entire option string at all) in the value field to turn the option on or off. It is not necessary to specify options which are not used. The order in which options appear is unimportant.

The right column of the Table 8.8 shows `LINE2MAT` options and the equivalent in the task file.

If an existing matrix is specified in task file (i.e. the task file contains the line “matrix::MATRIXFILE”), `LINE2MAT_INTERFACE` will parse the matrix file header. For the dense mode (dense matrix), the information about the pixel sizes, offsets, sizes of the frame, and the employed lines file will be extracted. For the sparse mode (sparse matrix), the information about vertical and horizontal frame sizes will be extracted.

textttLINE2MAT_INTERFACE option:: value	Equivalent LINE2MAT option and value
sparse_task::TRUE	-sparse_task
save::MATRIXFILE	-save MATRIXFILE
matrix::MATRIXFILE	-matrix MATRIXFILE
pixelx::DBL	-pixelx DBL
pixely::DBL	-pixely DBL
offx::DBL	-offsetx DBL
offy::DBL	-offsety DBL
lines::LINESFILE	-lines LINESFILE
size::INT	-size INT
sizey::INT	-sizey INT
wsave::WEIGHTFILE	-wsave WEIGHTFILE
wload::WEIGHTFILE	-wload WEIGHTFILE
ata_dump::ATAFILE	-ata_dump ATAFILE
ata_filename::ATAFILE	-ata_filename ATAFILE
mag_save_ata::INT	-mag_save_ata INT
svd::SVDFILE	-svd SVDFILE
sparsify::SPARSEFILE	-sparsify SPARSEFILE
view::TRUE	-view
mag::SINT	-mag SINT
view_ata::TRUE	-view_ata
mag_view_ata::SINT	-mag_view_ata SINT
save_rsv::TRUE	-save_rsv
mag_save_rsv::INT	-mag_save_rsv INT
wsvd::TRUE	-wsvd
view_rsv::INT	-view_rsv INT
mag_view_rsv::INT	-mag_view_rsv INT
group1_flag::TRUE	Display text fields for projection of the data. GUI option. No equivalent in LINE2MAT.
project_d_data::DATAFILE	-project_d_data DATAFILE
project_d_output::OUTPUTFILE1	-project_d_output OUTPUTFILE1
group2_flag::TRUE	Display text fields for projection of the water and ice. GUI option. No equivalent in LINE2MAT.
project_iw_ice::ICEFILE	-project_iw_ice ICEFILE
project_iw_water::WATERFILE	-project_iw_water WATERFILE
project_iw_output::OUTPUTFILE2	-project_iw_output OUTPUT- FILE2

Table 8.8: LINE2MAT_INTERFACE task file format [left]. Equivalent LINE2MAT options [right].

8.7 Using INVERT

8.7.1 Starting INVERT, Task Manager Window

INVERT is located in *bin* subdirectory in the root folder of this software package. It is started by typing `./invert` or `./invert TASKFILE` at a command prompt. In the second case, the existing task will be loaded (see information on task file below).

Since INVERT is a graphical user interface application, it requires X.

The task file is an ASCII file containing all information about the input file paths and other input parameters (see Table 8.9 and Table 8.10). The user can create a new task (go to “File” menu and select “New Task”). The task manager window will popup (see Figure 8.16). Similarly, the user can open an existing task file (“File”, “Load Task”). While working in INVERT, the current task file can be edited any time by going to “File” menu and selecting “Change Task”. The task window with the current information will popup. Finally, task file can be saved by selecting “File”, “Save Task”.

The format of the task file is shown below (see Table 8.9 and Table 8.10). The comment lines start with `/**`. If the task file was generated by INVERT, the first three lines will contain creation date and time (in the commented-out lines). Each option value is specified after the double-colon.

In the Table 8.9 and Table 8.10 “INT” refers to a positive integer number, and “DBL” refers to a floating-point double-precision number.

The task window reflects the options saved in the task file. The method to be used is selected by the corresponding toggle button (“Use iterative method”, “Use SVD method” or “Use Tikhonov regularization”).

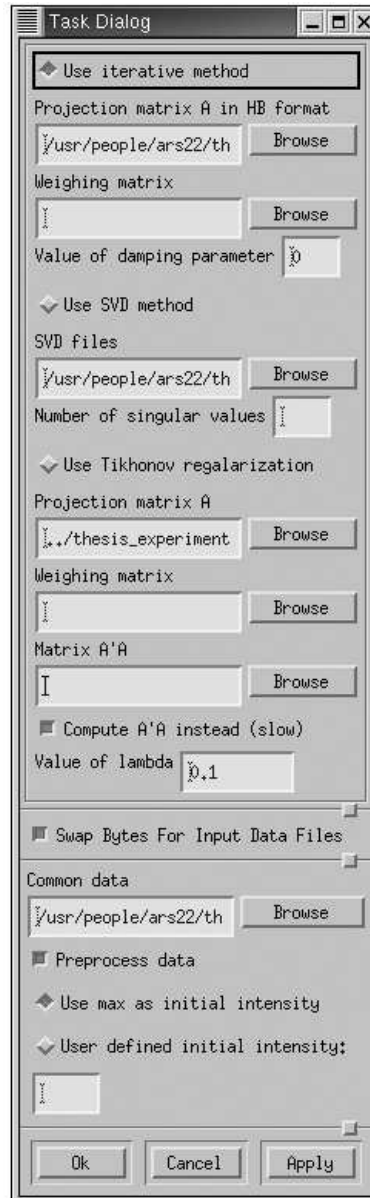


Figure 8.16: INVERT Task manager window.

Task file option:: value	Description
method::1	Selected method: singular value decomposition.
method::2	Selected method: Tikhonov regularization.
method::3	Selected method: LSQR.
Options for TSVD method	
svd::SVDFILE.sv	Location of the precomputed singular value decomposition (“sv” file with the singular values and “rsv.gz” and “lsv.gz” files (with the same base name) containing corresponding singular vectors).
sv::INT	Number of singular triplets used for the reconstruction (Optional. By default all singular values are used).
Options for Tikhonov regularization method	
a::MATRIXFILE.gz	Path to the dense matrix A stored in the compressed dense format.
ata::ATAFILE.gz	Path to the dense matrix $A^T A$ stored in the compressed dense format (Optional. The matrix $A^T A$ can be computed in the runtime which will be slower than loading an existing one).
compute_ata::TRUE	Compute $A^T A$ in the runtime (This option is neglected when the matrix $A^T A$ is loaded).
l::DBL	The initial value of regularization parameter. The default value is 0. This value may not work for ill-conditioned matrices (the matrix $A^T A$ may not be positive definite).
w.t::WEIGHTFILE.gz	The reciprocals of the diagonal elements of the weighting matrix $W^{1/2}$ in the binary compressed format (Optional).
Options for LSQR method	
ahb:: SPARSEFILE.gz	Matrix A in the sparse compressed format.
dmp::DBL	The initial value of the regularization parameter. The default value is 0. This value is not suggested to use for the ill-conditioned matrices.
w.i::WEIGHTFILE.gz	The reciprocals of the diagonal elements of the weighting matrix $W^{1/2}$ in the binary compressed format (Optional).

Table 8.9: Task-specific task file options.

Task file option:: value	Description
Common Options	
data::DATAFILE.ir	Common data used by all methods. The data is in binary format. If the file with the same name and “v” extension exists, INVERT will run in the double-data mode (i.e. two images will be reconstructed at the same time - see Figure 4.2 in the thesis). Otherwise, INVERT will run in the single data mode (see Figure 8.17, Figure 8.18, and Figure 8.19).
data::DATAFILE.v	Common data used by all methods. The data is in binary format. If the file with the same name and “ir” extension exists, INVERT will run in the double-data mode (i.e. two images will be reconstructed at the same time - see Figure 8 in the thesis). Otherwise, INVERT will run in the single data mode (see Figure 8.17, Figure 8.18, and Figure 8.19).
preprocess::DBL	Preprocess the data element with the value DBL; i.e. the negative logarithm of the ratio of the original data value over DBL will be used instead of input data.
preprocess::-1	Preprocess the data with the maximum data element value, i.e. the negative logarithm of the ratio of the original data over the value of the maximum data element will be used instead of input data.
preprocess::0	No data preprocessing will take place. The data “as-is” will be used for the reconstruction.
swap::TRUE	Swap the bytes in all input binary files. This option is used when one loads files generated on IRIX platform on the Linux machines.

Table 8.10: Common task file options.

For the iterative method (LSQR), the matrix A in sparse compressed format (“gz” file) must be specified; the weighting matrix (“gz” file - see Table 8.9 for more information) and the value of the damping parameter (default=0) are optional.

For the singular value decomposition method, the user has to specify the path to the singular value decomposition of (unweighted or weighted) matrix A stored in the files “sv” (singular values), “rsv.gz” (right singular vectors), and “lsv.gz” (left singular vectors). The user may specify the number of singular triplets to be used. By default all singular triplets are used.

For the Tikhonov regularization method, the user must specify the path to the matrix A stored in dense compressed binary format (“gz” file). Optional is the weighting matrix (“gz” file - see Table 8.9 for more information). If a precomputed matrix $A^T A$ (“gz” file) is not available, **INVERT** can compute it checking the box “Compute $A^T A$ instead (slow)”. If weighting of the matrix A is used, **INVERT** will be required to generate $A^T A$ because instead of the matrix $A^T A$, $A^T W A$ is needed. The value of the regularization parameter (default=0) may be specified. For ill-conditioned matrices, it is necessary to specify this value in order to ensure that the matrix $A^T A$ will become positive definite.

If the input binary files were generated on a big-endian platform (such as SGI) and these files are about to be used on a little-endian platform (such as Linux), the user is required to check the “Swap Bytes for Input Data Files” box. Be careful with this option, as inappropriate usage may cause a program crash. Notice, that all ASCII files, as well as the file containing the sparse

matrix A do not require byte-swapping.

The user is required to specify the data file (either “v” or “ir” file) used for all reconstruction methods. The data file is in binary format. If a file with the same name and “ir”/“v” extension exists, **INVERT** will run in the double-data mode (i.e. two images will be reconstructed at the same time - see Figure 4.2 in the thesis). Otherwise, **INVERT** will run in the single data mode (see Figure 8.17, Figure 8.18, and Figure 8.19).

The user can preprocess the data by checking the “Preprocess data” box (optional). If “Use max as initial intensity is selected, the maximum data element value will be used for I_0 (instead of the default $I_0 = 1000$). If “Use defined initial intensity” is selected, the specified I_0 will be used.

8.7.2 INVERT GUI Elements Common to All Methods

By default, the image is shown with magnification equal to 7. In order to change the magnification, use the “View” menu and select “Magnification”. Specify a new positive integer magnification.

Below the reconstructed image there are some tools to control the visualization (in order to display these tools, check the box “Graphics tools” and uncheck this box to remove them). Two sliding bars for each image, “Center” and “Width” control the grayscale center and width of the image respectively. When in double-data mode, the user can link the sliders for both images by checking the box “Link sliders”. The values of center and width are shown in the text area below the sliders. Use “Reset” to restore the original values of the center and width. To see the value of a certain pixel, move the mouse pointer over the pixel of interest, and its value will be displayed below

the image in double-precision next to the text “Pixel value”.

Normally distributed white noise can be added using the text box. The number “1.00” corresponds to “100%” of the noise. When a new value for noise is entered, and “Enter” is pressed, all information on the screen will be updated, and the image will be recomputed. Also the noise level can be changed by going to “Values”, selecting “Noise”, and inputting noise level there.

While working with simulated data, or when a reconstruction prototype is available, *INVERT* compute and display the corresponding residual. Go to “Values”, select “Water Residual” and “Add Water Residual” to load the TIFF file with the original image to be subtracted. The corresponding residual value will be displayed on the screen right from the noise value text area. While working with double-data, the residual for the second image can be computed by going to “Values”, “Ice Residual”, “Add Ice Residual”. In order to remove the residual display from the screen, go to “Values”, select corresponding residual submenu, and select either “No Ice Residual” or “No Water Residual”.

8.7.3 Using TSVD method

The screen shot of *INVERT* in TSVD mode is shown in the Figure 8.17. The reconstructed image (or images in the double-data mode) is shown on top of the screen.

The ratio of the current singular value to the largest one is displayed to the right of the text “Singular Value Ratio”. The number of singular values used in the reconstruction is also displayed.

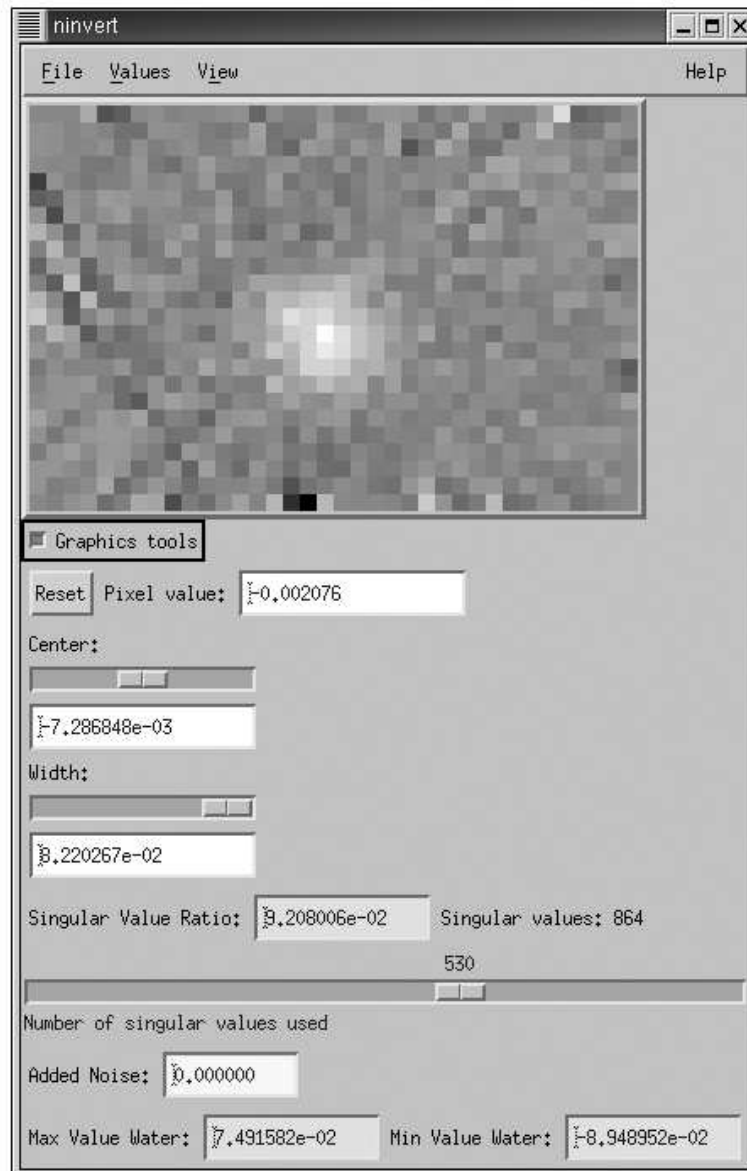


Figure 8.17: Using TSVD method.

The slider below controls the number of singular triplets used in the image reconstruction. Select all singular triplets by moving the slider bar to the right extreme, and select one singular triplet by moving the slider bar to the left extreme. After movement of the slider bar, the reconstructed image, and all other parameters shown on the screen are updated automatically.

One can compute an L-curve to determine the optimal number of singular triplets to use. Go to “Values” menu and select “Compute L-curve”. The L-curve implementation minimizes the sum $\log \|x\| + \log \|Ax - b\|$. A detailed description of L-curve criterion is presented in the chapter on mathematical methods of this thesis.

The information about maximum and minimum values of the reconstructed image is displayed at the bottom of the screen. When working in the double-data mode, this information will be displayed for each image.

8.7.4 Using Tikhonov Regularization Method

The screen shot of INVERT in Tikhonov regularization mode is shown in the Figure 8.18. The reconstructed image(s) is (are) shown on top of the screen.

The information about the backward and relative error estimates during the image reconstruction is displayed below the controls for visualization properties. An estimate of the relative condition number of the regularized matrix is displayed below the error estimates fields. The relative condition number is a reciprocal of the condition number of the matrix. These three estimates (backward and relative error estimates and estimate of the relative condition number) are very important since they provide information about the accuracy of the solution. If the relative condition number is of the

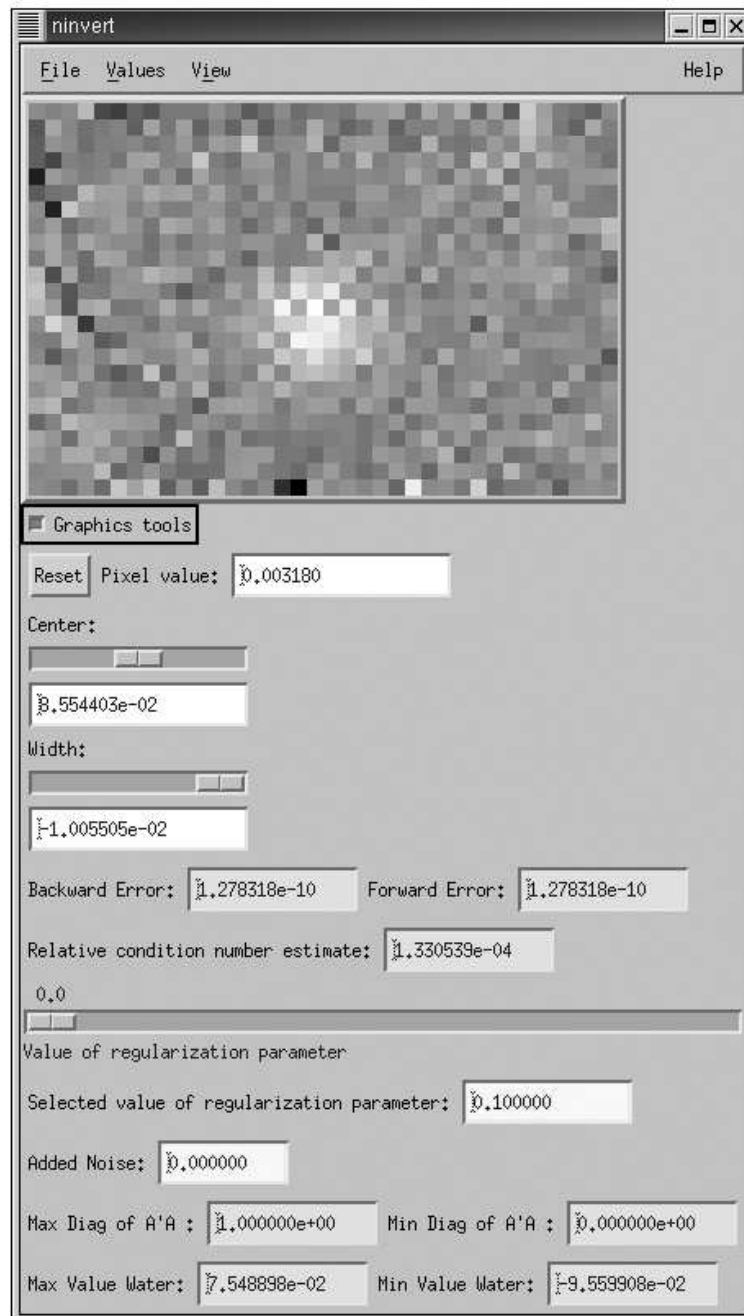


Figure 8.18: Using Tikhonov regularization method.

order of 10^{-n} , then we expect to have only at most $16 - n$ digits of accuracy in the solution.

The slider controls the value of the square of the regularization parameter, τ^2 . The slider bar allows the users to select τ^2 from 0 (not recommended, since the matrix $A^T A$ may not be positive definite) to the maximum eigenvalue λ_{max} of the matrix A which is determined iteratively (see chapter on computer implementations of this thesis). After the movement of the slider bar, the reconstructed image, as well as all other parameters shown on the screen are updated automatically.

The exact value of the regularization parameter, the required value of τ^2 may be input in the text field “Selected value of the regularization parameter” (this text field always displays the current value of τ^2). Press “Enter” to apply new value.

The information about the maximum and the minimal diagonal elements of $A^T A$ is displayed below. These can be used to select the regularization parameter τ .

The information about maximum and minimum values of the reconstructed image is displayed at the bottom of the screen. When working in the double-data mode, this information will be displayed for each image.

8.7.5 Using LSQR Method

The screen shot of INVERT in LSQR mode is shown in the Figure 8.19. The reconstructed image (or images in the double-data mode) is shown on top of the screen.

Below the controls for the visualization properties are the six text fields

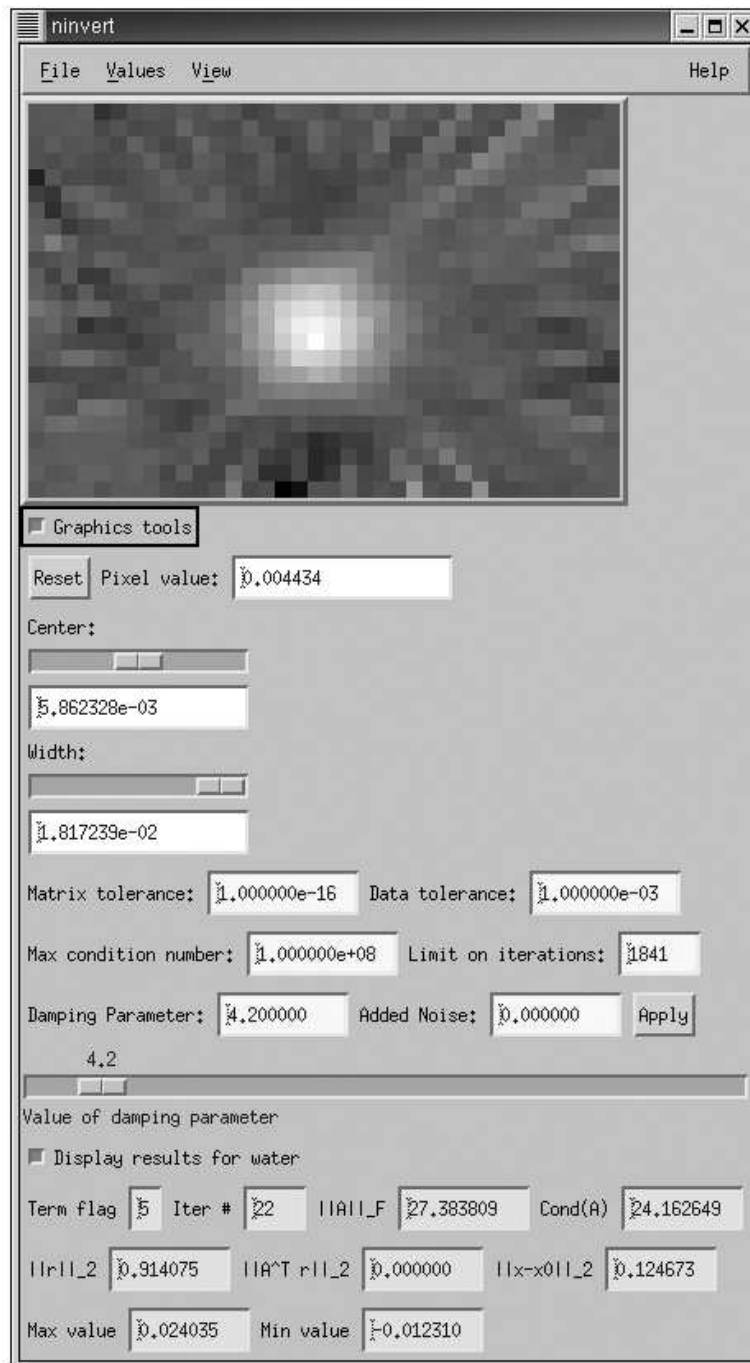


Figure 8.19: Using LSQR method.

with the input values.

The first four input values are the stopping criteria to terminate LSQR iterates: “Matrix tolerance” - terminate when maximum matrix tolerance is reached (default value is $1e - 16$, i.e. matrix is assumed to be exact); “Data tolerance” - terminate when maximum data tolerance is reached (default value is $1e - 3$); “Max condition number” - terminate when this condition number is reached (default value is $1e + 8$, i.e. bound the inaccuracies in the computation of solution); and “Limit on iterations” - terminate after this number of LSQR iterations (default is $m + n$, i.e. sum of dimensions of the original matrix). All four criteria are used simultaneously, i.e. whatever stopping condition is encountered first, terminates LSQR. A detailed description of the stopping criteria can be found in the earlier chapter of the thesis.

The fifth input field labelled by “Damping Parameter” corresponds to the nonnegative value of the damping (regularization) parameter τ (it is not recommended to select the regularization parameter equal to 0).

The last input field controls the value of the normally distributed noise as described above.

When the new values of any input parameter is entered, the user has to press “Enter”, and all information on the screen will be updated, and the image will be recomputed. If several input fields were change, it is recommended to press the button “Apply” instead to pressing “Enter” for each text field.

Below the input fields is the slider which controls the value of the regularization parameter τ . The regularization parameter can be selected in

the range from 0 (no regularization - not recommended) to the largest singular value of A which is equal to the 2-norm of the matrix. The text field “Damping Parameter” can be used to specify the desired regularization parameter.

Finally, the information about LSQR termination flag; the number of the last LSQR iterate taken; the matrix Frobenius norm $\|A\|_F$; the matrix A condition number estimate; the 2-norm of the residual $\|r\|_2 = \|Ax - b\|_2$; the norm of the residual projection $\|A^T r\|_2$; the difference between the initial guess and last iterate $\|x - x_{initial}\|_2$, as well as maximum and minimum values of the reconstructed image are displayed at the bottom of the screen. The termination flag values are shown in the Table 8.11. This information can be displayed by checking “Display results for water” box. If the user is working in the double-data mode, this information can be also displayed for the second image by checking “Display results for ice” box.

See Table 8.11.

Termination flag value	Description
0	The initial guess (zero vector) is the same as the exact solution.
1	The equations $Ax = b$ are probably compatible. The norm $\ Ax - b\ $ is sufficiently small, given the values of relative matrix and right-hand side errors.
2	The system $Ax = b$ is probably not compatible. A least-squares solution has been obtained that is sufficiently accurate, given the value of matrix relative error.
3	The estimate of the matrix condition number has exceeded specified condition number limit. The system $Ax = b$ appears to be ill-conditioned in that context.
4	The equations $Ax = b$ are probably compatible. The norm $\ Ax - b\ $ is as small as seems reasonable on this machine.
5	The system $Ax = b$ is probably not compatible. A least-squares solution has been obtained that is as accurate as seems reasonable on this machine.
6	The estimate of the matrix condition number seems to be so large that there is no point in doing further iterations, given the precision of this machine.
7	The specified iteration limit was reached.

Table 8.11: LSQR termination flag values.

Chapter 9

Appendix B: Software Package Developer Manual

9.1 Preface

The developer manual is intended to assist a future developer or advanced user of this software package in making both minor and major modifications to the software design, algorithms and data-structures. The package was written using C/C++, and it is understood that the reader of this manual knows the basics of computer programming and operation systems.

The package consists of three main applications - LGEN, LINE2MAT^{9.1}, and INVERT^{9.2}.

The design overview, data-structures, key methods description, and other information relevant for the potential developer is presented separately for each of the application.

^{9.1}The initial IRIX version of this software designed for TSVD method was developed by Steven H. Izen.

^{9.2}The initial IRIX/ViewKit version of this software implementing TSVD reconstruction was developed by Steven H. Izen.

9.2 Developing LGEN

9.2.1 Compiling the Program

LGEN requires the following computer libraries: X libraries (libXm, libXt, libXmu, libXp (Linux only), libXext, libX11), OpenGL and GLU libraries (libGL and libGLU respectively), and TIFF image library (libtiff).

The suggested compiler is gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-85) or gcc version 2.8.1 (IRIX).

Below is a sample Makefile (Linux version) for LGEN:

```
#!/usr/local/bin/gmake

OPT=-g
CXXFLAGS=$(OPT) -I/usr/X11R6/include
CXX=g++
LD_FLAGS=-v

lgen: lgen.o lg.o init_font.o object_list.o\
      draw_primitives.o menu.o geometry_init.o\
      draw_objects.o motion.o draw_lines.o\
      popup_object.o show_grid.o renumber_objects.o\
      show_numbers.o popup_info.o draw_lines_object.o\
      resize_objects.o add_object.o save_tiff.o\
      load_geometry.o equispace.o save_project.o\
      show_line_origins.o init_font.o expose.o\
```

```
resize_window.o events.o highlight_object.o\  
undo.o toggle_background.o project_properties.o\  
popup_line.o lines_database.o save_lines.o\  
generate_lines_database.o eprintf.o\  
check_popup.o save_ps.o popup_destinations.o\  
$(CXX) $(OPT) $(.ALLSRC) -o $@ $@.o lg.o\  
init_font.o object_list.o draw_primitives.o\  
menu.o geometry_init.o draw_objects.o motion.o\  
draw_lines.o popup_object.o show_grid.o \  
renumber_objects.o show_numbers.o popup_info.o\  
draw_lines_object.o resize_objects.o\  
add_object.o save_tiff.o load_geometry.o\  
equispace.o save_project.o show_line_origins.o\  
expose.o resize_window.o events.o\  
highlight_object.o undo.o toggle_background.o\  
project_properties.o popup_line.o\  
lines_database.o save_lines.o\  
generate_lines_database.o eprintf.o\  
check_popup.o popup_destinations.o save_ps.o\  
-L/usr/X11R6/lib \  
-lXm -lXt -lXmu -lXp -lXext -lX11 \  
-lGLU -lGL -ltiff
```

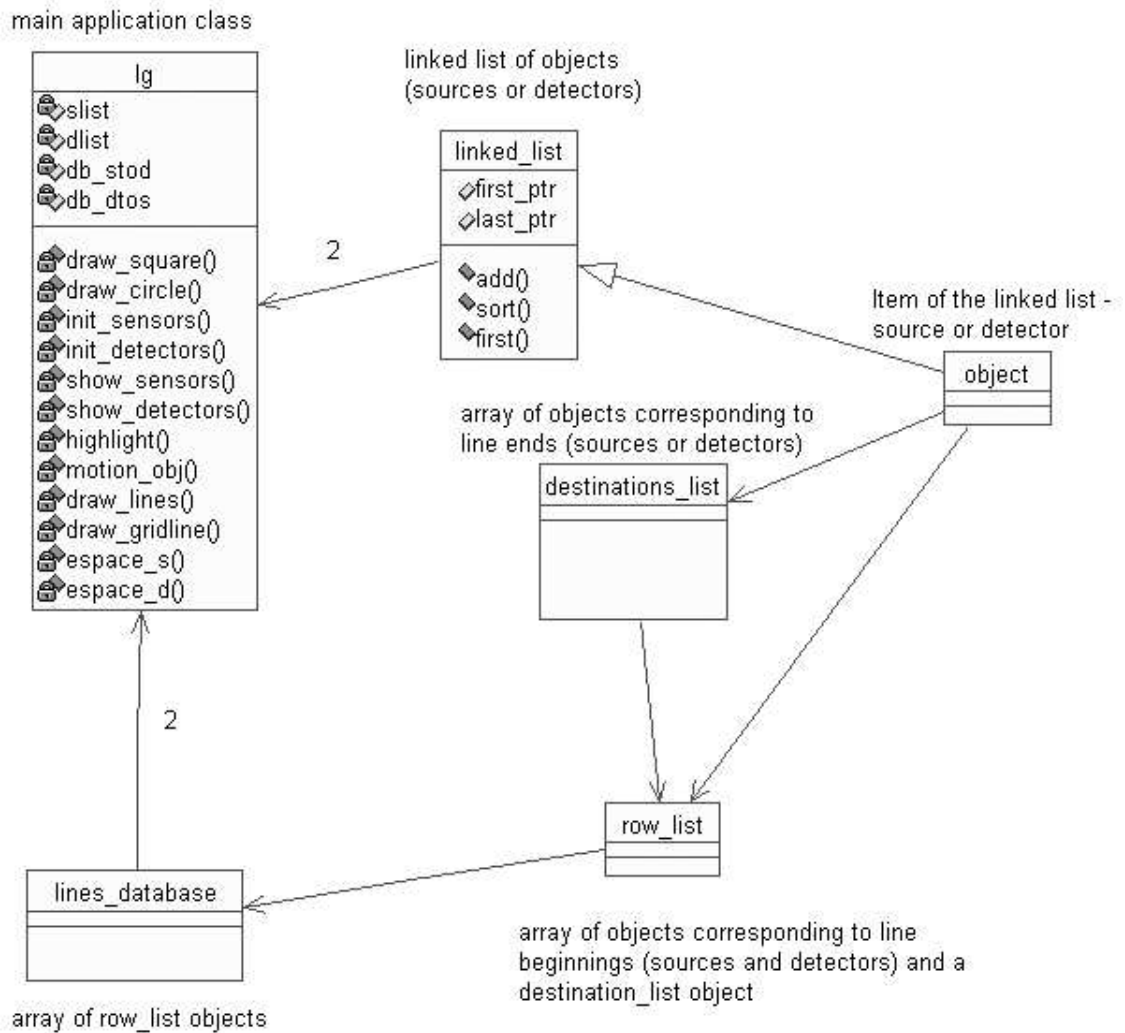


Figure 9.1: Sketch of the LGEN UML class diagram. Only a few methods and data structures are shown. The diagram was generated by Rational Rose.

9.2.2 Design Overview

A sketch of the UML class diagram of the LGEN application summarizing its class static relationships is shown in the Figure 9.1.

This application consists of the main class *lg* which is responsible for the Motif based graphical user interface (see the section on *Motif design*) and operations including events and callbacks. The attributes *slist* and *dlist* of this object are double circular linked lists (instances of the class *linked_list*). The linked lists include objects of the type *object* which in turn represents a source or a detector on the frame in the test section. There is no difference between sources and detectors from the programming perspective (see the section on *artificial data-structures*).

The class *lg* also has two databases *db_stod* (lines given by sources connected to detectors) and *db_dtos* (lines given by detectors connected to sources). See the section on *artificial data-structures* below for details of the database structure. The database objects are instances of the class *lines_database* which is related to the sources and detectors as shown in the diagram. Since the two databases include the same information on the line connections, one database is usually generated from the other. It is not problematic to program LGEN to use only one database. However, when the geometries have a large number of sources and detectors, the “reverse tracing” (showing lines starting at the given detectors versus show the lines starting at the given sources) of the database can be slow.

9.2.3 GUI Design

The Motif GUI consists of the main window which has three children widgets - *menu* (top menu bar), *glxarea* (drawing area widget), and *form* (a form for control buttons, input fields, toggles, etc - everything between the menu bar and the drawing area widget wrapped into multiple row column widgets).

Even though a Motif drawing area widget provides all required functionality for the application, its use is inconvenient since it requires a significant amount of low-level X-Windows programming to establish the OpenGL-Motif interface. A simpler approach is to consider an OpenGL-specific Widget. Unfortunately, this widget is not available on all platforms, and its installation is difficult. That is why LGEN was implemented using a Motif drawing area widget. A detailed description of integrating Motif with OpenGL can be found in [14].

The Motif drawing area widget provides an interface to interpret and display OpenGL statements inside the widget. Moreover, Motif provides a collection of callbacks and events which are registered to that widget (see the section on *key methods description*).

9.2.4 Constants and Limits

Each object corresponding to either a source or a detector has a status member *dobj_status* (see the section on artificial data-structures). The status member values are defined in the file *lg.h* as follows:

```
#define DEFAULT 1
#define ADDED 2
#define REMOVED 3
```

The user can also define a new status adding the following string to the file *lg.h*:

```
#define NEWSTATUS value
```

where *NEWSTATUS* is the status variable, and *value* is the status value. See the section on artificial data-structures for more information on access to object data including status information.

By default, the user can work with geometries containing up to 500 sources and 500 detectors. The developer can easily redefine these limits in the files *lg.h* and *lines_database.h*:

```
#define MAX_SENSORS 500
#define MAX_DETECTORS 500
```

9.2.5 Artificial Data-Structures

The application maintains the database of lines given by sources connected to detectors (*db_stod*) and the database of lines given by detectors connected to sources) (*db_dtos*). These two databases are instances of the same class *lines_database*.

It is better to interpret the database as the matrix consisting of some object elements. The matrix has $MAX_SENSORS + MAX_DETECTORS$ rows - instances of the class *row_list*. Each row consists of the pointer to the line origin object (source or detector), as well as the array of $MAX_SENSORS + MAX_DETECTORS$ pointers to the destination objects of the other type (detectors or sources respectively).

Instead of using regular objects of the type *object* (elements of the linked lists *slist* and *dlist*), the instances of the class *dest_list* are used to describe the line destinations. These objects contain the *object* as well as its line status information (i.e. whether the line is manually added, removed, or

default - see LGEN user manual for more information).

If one considers the line starting from the origin object number i to the destination object of the opposite type number j , the line origin object is given by $db_stod \rightarrow start[i] \rightarrow ending[j] \rightarrow dobj$ and $db_dtos \rightarrow start[i] \rightarrow ending[j] \rightarrow dobj$ depending on the employed database. The line origin object can be accessed by $db_stod \rightarrow start[i] \rightarrow beginning$ or $db_dtos \rightarrow start[i] \rightarrow beginning$. The line status information is available at $db_stod \rightarrow start[i] \rightarrow ending[j] \rightarrow dobj_status$ and $db_dtos \rightarrow start[i] \rightarrow ending[j] \rightarrow dobj_status$ respectively. See the section on *constants and limits* for the information about possible values of the line status variable *dobj_status*.

The second major artificial data-structure employed in LGEN is the double circular linked list of the objects on the frame (see the section on *design overview*). The sources linked list (class *linked_list*) is the object *slist* and the detectors linked list is the object *dlist*. Every element of the linked list is the object of the type *object*. This object encapsulates the states defining the basic properties of the object on the frame:

```
class object{
    ...
    class linked_list* myclass; // Pointer to the super-class.
    int active;                // Indicator of activity of the
                                // of the object (1 - activated,
                                // 0 - deactivated).
    int id;                    // Object ID.
```

```

int show_lines;           // Show lines for this object
                           // with magenta color (1) or not
                           // (0).

double x;                 // Net (offset is not considered)
                           // horizontal coordinate.

double y;                 // Net (offset is not considered)
                           // vertical coordinate.

double angle;            // Angle in radians.

double orientation;      // Orientation in radians.

int angle_manual;        // Manually defined angle in
                           // radians.

int orientation_manual;  // Manually defined orientation
                           // in radians.

object* next_ptr;        // Pointer to the next object.

object* prev_ptr;        // Pointer to the previous
                           // object.

...

};

```

9.2.6 Key Methods Description

See Table 9.1, Table 9.2, Table 9.3, and Table 9.4 for methods of the class *lg*.

See Table 9.5 for methods of the class *object*. See Table 9.6 for methods of the class *linked_list*. See Table 9.7 for methods of the class *linked_list*.

Method	Description
activate_object	Activate or deactivate the object.
adddetector	Add a new detector to the frame.
addsensor	Add a new source to the frame.
browse	Popup fileselection window when one clicks on the “Browse” button initializing the new project.
buttonpress	Method executed when mouse button is pressed.
buttonrelease	Method executed when mouse button is released.
cancel	Simple “Cancel” method which closes the window.
check_popup	Check that there is no more than one open window of the same type.
CreateMenuBar	Create Motif menu bar.
connect	Connect or disconnect two objects with a line.
defaultsave	Save the project file with the same name.
destinations	Connect or disconnect an object with the range of objects of the other type.
destinationspopup	Popup a window offering to connect or disconnect an object with the range of objects of the other type.
display	Expose the drawing area.
drawlines	Draw all lines specified in the lines database (if exists; otherwise - call database generation).
drawlines_object	The the flag of the object to draw the lines originating from it with magenta color.
drawlines_selected	Draw selected lines for all objects with magenta color.
draw_circle	Draw a circle at the given point with the specified radius.
draw_circle_select	Draw a border around the circle at the given point with the specified radius.
draw_gridline	Draw a grid line between two points in the drawing area.
draw_line	Draw a line of the specified color between two objects.

Table 9.1: Methods of the class *lg* (callback and event “drivers” are not listed). Part 1.

Method	Description
draw_lines_ generate_database_ draw_lines_ generate_database_ complimentary draw_lines_ generate_database_ complimentary_2 draw_square	Initialize and generate a lines database (“sources to detectors”). Generate “detectors to sources” database from the existing “sources to detectors” database. Generate “sources to detectors” database from the existing “detectors to sources” database. Draw a square at the given point with the specified half-side.
draw_square_select	Draw a border around the square at the given point with the specified half-side.
eprintf	Print an error message in the Motif dialog.
equispace	Equispace the objects on all sides (driver to <i>espace_d</i> and <i>espace_s</i>).
espace_d	Equispace detectors at the given side.
espace_s	Equispace sources at the given side.
font_init	Initialize X font.
getlines	Get lines starting at the given origin object - driver to the “2-angle” algorithm (method <i>twoangle</i>).
GetTopShell	Get top shell widget.
gprint	Print a text string at the given point of the drawing area with font of the specified color.
init_detectors	Initialize detector objects.
init_sensors	Initialize source objects.
lg	<i>lg</i> class constructor.
Loadgeometry	Load an existing project.
mcancel	One more “Cancel” method which closes the window.
motion	Mouse motion event function.
motion_obj	Move an object to the new coordinates.
motion_obj_put	Place an object to the new position (subfunction of <i>motion_obj</i>).

Table 9.2: Methods of the class *lg* (callback and event “drivers” are not listed). Part 2.

Method	Description
newtask	Popup the new task window (the corresponding callback is registered to the drawing area widget).
ok	Method executed when the user clicks on the “Ok” button while initializing a new project.
ok_popup	Method executed when the user clicks on the “Ok” button while editing object properties.
popup	Popup the object properties window on the right-click of the mouse on the object.
popupequispace	Popup the window offering to equispace objects on the sides.
popupinfo	Popup geometry information window.
popupLoadgeometry	Popup the window for loading the project.
popupSavegeometry	Popup the window for saving the project.
popupSavelines	Popup the window for saving a lines file.
popupSavePS	Popup the window for saving an Encapsulated Postscript file.
popupSaveTIFF	Popup the window for saving a TIFF file.
popup_line	Popup the window offering to connect or disconnect two selected objects with a line.
projectproperties	Popup the project properties window.
properties	Change the global project properties.
quitCallback	Quit.
remove	Remove the object from the frame.
renumberobjects	Label the objects on the frame.
renumberobjectspopup	Popup the window with options for labeling of the objects on the frame.
resize	Resize the window.
resize_o	Popup the window offering to resize the objects’ visual representaiion.
Savegeometry	Save the current project.
Savelines	Save the lines file dialog.

Table 9.3: Methods of the class *lg* (callback and event “drivers” are not listed). Part 3.

Method	Description
SaveLines_	Save the lines file.
SavePS	Save the drawing area image to the Encapsulated Postscript file.
SaveTIFF	Save the TIFF image.
scale	Resize the objects' visual representation.
showdetnumbers	Show detector numbers.
showgrid	Show the grid.
showlineorigins	Display/undisplay the line origins.
showsennumbers	Show source numbers.
show_detectors	Draw detector objects.
show_sensors	Draw source objects.
togglebackground	Toggle the background of the drawing area widget.
twoangle	“2-angle” algorithm - an algorithm to determine two angular coordinates on the frame corresponding to the intersections of the “range bounds” of the origin object with the frame.
updatefield	Update the text field after the file selection.
undo	Undo object movement/removal.
writeTiff	Save the drawing area image to a TIFF file.
~lg	<i>lg</i> class destructor.

Table 9.4: Methods of the class *lg* (callback and event “drivers” are not listed). Part 4.

Method	Description
getangle	Get the object angular coordinate on the frame.

Table 9.5: Method of the class *object*.

Method	Description
add	Add the object to the linked list.
first	Determine the most adjacent object to a given angular coordinate.
linked_list	Constructors (2) of the class <i>linked_list</i> .
sort_id	Sort object ID's.
~linked_list	Destructor of the class <i>linked_list</i> .

Table 9.6: Methods of the class *linked_list*.

Method	Description
lines_database	Constructor of the class <i>lines_database</i> .
~lines_database	Destructor of the class <i>lines_database</i> .

Table 9.7: Methods of the class *lines_database*.

9.3 Developing LINE2MAT

9.3.1 Compiling the Program

LINE2MAT requires the following computer libraries: X libraries (libXm, libXt, libXmu, libXp (Linux only), libXext, libX11) for the internal image viewer, regular mathematical library (libm), ZIP compression library (libz), BLAS library (libblas), LAPACK library (liblapack), TIFF image library (libtiff), Fortran 77 library (libF77), Fortran to C linkage library (libg2c), and Sparselib++ suit (including libsparse, libmv, libspblas). While working in IRIX environment, BLAS and LAPACK are substituted by the Scientific Computing Software Library (libscs). Moreover, there is no need to use Fortran to C linkage library (libg2c) on IRIX platforms when libscs is used.

The suggested compiler is gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-85) or gcc version 2.8.1 (IRIX).

Below is a sample Makefile (Linux version) for LINE2MAT:

```
#!/usr/local/bin/gmake

SPARSELIB=/usr/people/ars22/libs/sparselib
SPARSELIB_INCLUDE=-I$(SPARSELIB)/include
MV_INCLUDE=-I$(SPARSELIB)/mv/include
SPARSELIB_DIR=$(SPARSELIB)/lib
SPARSE_LIBS=-L$(SPARSELIB_DIR) -lsparse -lmv -lspblas

OPT=-g

CXXFLAGS=$(OPT) \
    $(SPARSELIB_INCLUDE)\
    $(MV_INCLUDE) \
    '-DCOMPLEX=complex<double>' -DCOMPLEX_SUPPORT\
    -I/usr/X11R6/include \

line2mat: line2mat.C line2mat.o line_item.o\
    line_list.o cl2m.o process_ata.o\
    svd.o line_cross.o wmatrix_save.o\
    matrix_save.o project.o\
    matrix_fill.o matrix_extra.o viewer.o\
    tiff_fio.o process_sv.o matrix_load.o\
    wmatrix_load.o
```

```

$(CXX) $(.ALLSRC) -o $@ $@.o \
line_item.o line_list.o cl2m.o process_ata.o\
svd.o line_cross.o wmatrix_save.o\
matrix_save.o matrix_fill.o matrix_extra.o\
viewer.o tiff_fio.o process_sv.o project.o\
matrix_load.o wmatrix_load.o\
-L/usr/lib \
-L/usr/X11R6/lib \
-lXm -lXt -lXmu -lXp -lXext -lX11 \
-lm -lz -lblas -llapack -ltiff \
-lF77 -lg2c\
$(SPARSE_LIBS)

```

9.3.2 Design Overview

Unlike other applications, LINE2MAT does not take much advantage of object oriented design. The sketch of the LINE2MAT UML class diagram is shown in the Figure 9.2. The methods of the main class which execute after parsing the options lines are responsible for all computation and functionality of LINE2MAT (see the section on *key methods description*). The purpose of the remaining three classes *line_descriptor*, *line_list*, and *endpoints* is to simplify the generation of the matrix A from the lines file.

There was no need to implement a more sophisticated design, since LINE2MAT mostly consists of single computational functions which do not have interactions between each other.

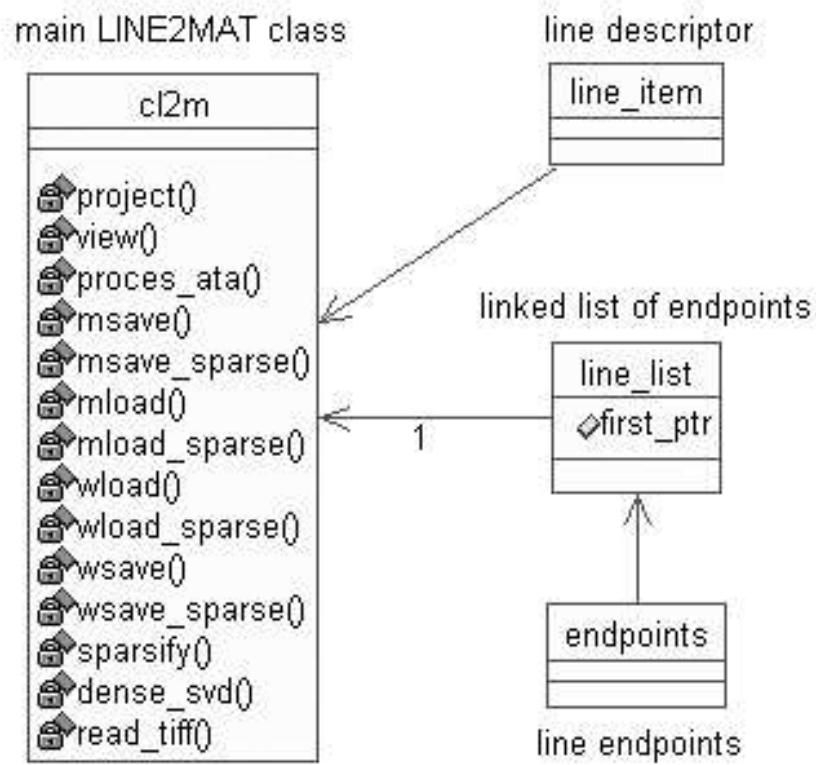


Figure 9.2: Sketch of the LINE2MAT UML class diagram. Only a few methods and data structures are shown.

9.3.3 GUI Design

LINE2MAT includes a simple Motif image viewer to display the matrix A , $A^T A$, or a selected right singular vector. The viewer is implemented as a single method *view* (see the section on *key methods description*) which is executed as the forked process.

The viewer consists of a scrolled window with the form which contains all the GUI elements - a frame widget with a drawing area widget and labels, buttons, and scroll bars to the right of the drawing area.

The image representing the array of doubles is displayed using standard X graphics functions, filling the image pixel by pixel. If a positive magnification k is used, the 1 by 1 pixel is replaced by a k by k block of pixels of the same color. If a negative magnification is used, the average value of the pixels inside a block is computed and assigned to a single pixel.

9.3.4 Artificial Data-Structures

The most important data-structure used in LINE2MAT is the compressed binary matrix storage format. The files of this type are created by the ZLIB compression library.

First, a header of 16 integers (first element - number of pixels, second element - number of lines, third element - number of pixels in horizontal dimension, fourth element - number of pixels in horizontal direction) is written to the file. The 16 integers header is followed by a 16 doubles header (first element - horizontal pixel size, second element - vertical pixel size, third element - horizontal size of the frame, fourth element - vertical size of the

frame). Next, a 512-char array containing user comments information (such as lines file name used to generate the matrix A) is written. Any of the unused elements in the headers can be left unassigned and are available for future use. All headers are followed by the data array of doubles of the size equal to the product of number of pixels and number of lines.

Below an example of creating a file using the procedure described above:

```
// Open file
void* zf;
zf=gzopen(outfilename,"wb");

// Write integers header
int header_int[16];
header_int[0]=num_pixels;
header_int[1]=num_lines;
header_int[2]=pixel_xdim;
header_int[3]=pixel_ydim;
gzwrite(zf, header_int, 16*sizeof(int));

// Write doubles header
double header_dbl[16];
header_dbl[0]=pixelx;
header_dbl[1]=pixely;
header_dbl[2]=llx;
```

```

header_dbl[3]=lly;
gzwrite(zf, header_dbl, 16*sizeof(double));

// Write chars header
gzwrite(zf, linesfilename, 512*sizeof(char));

// Write data
gzwrite(zf, matrix, num_pixels*num_lines*sizeof(double));

// Close file
gzclose(zf);

```

While saving a vector in this format, we assign 1 to “header_int[0]” and the length of vector to “header_int[1]” in the context of the example above.

It is important to emphasize that any changes in the described format should be reflected in the corresponding functions in `LINE2MAT_INTERFACE` and `INVERT`.

9.3.5 Constants and Limits

Notice, that `LINE2MAT` makes certain assumptions on matrix sparsity. Namely, the following definition is used in the program: an m by n matrix A is sparse if and only if its number of nonzero elements does not exceed the integer part of the number $1.2(mn)^{0.8}$. This definition is also used for the memory allocation for the sparse matrix. In order to modify constraints on matrix sparsity, edit the following lines in the file `cl2m.h`:

Method	Description
<code>linecross</code>	Driver to the algorithm to compute of the lengths of the intersections of a line with the pixel grid.
<code>line_intersection</code>	Compute of the lengths of the intersections of a line with the pixel grid when the line is not nearly (or exactly) horizontal or vertical.
<code>line_intersection_horizontal</code>	Compute of the lengths of the intersections of a line with the pixel grid when the line is horizontal (nearly or exactly).
<code>line_intersection_vertical</code>	Compute of the lengths of the intersections of a line with the pixel grid when the line is vertical (nearly or exactly).
<code>main</code>	Main method. Option parser.

Table 9.8: Main application methods.

```
double exp_nonz_d=(1.2)*pow(num_pixels*num_lines, 0.8);
int exp_nonz=(int)ceil(exp_nonz_d);
```

9.3.6 Key Methods Description

See Table 9.8 for main application methods. See Table 9.9 and Table 9.10 for the methods of the class *cl2m*. See Table 9.11 for the methods of the class *line_item*. See Table 9.12 for the methods of the class *line_list*. See Table 9.13 for the methods of the class *endpoints*.

9.3.7 Adding New Basis Functions

In order to implement a new basis (in addition to the implemented rectangular basis), the developer has to edit (or add new) methods *linecross*, *line_intersection*, *line_intersection_horizontal*, and

Method	Description
autoscale	Update the image when the slider bars are moved in the image viewer.
cl2m	Constructors (4). Depending on arguments can be used to starting a new matrix (dense or sparse), as well load the existing one.
dense_svd	Compute and save dense singular value decomposition.
density	Report on the density of the dense matrix.
expose	Expose the drawing area widget calling methods to “refill” the widget.
fillImage	Fill the image (the drawing area widget) pixel by pixel when positive magnification is used (image is enlarged).
fillImage2	Fill the image (the drawing area widget) pixel by pixel when negative magnification is used (image is shrunked).
line_fill	Fill the row of the matrix (process a line in the geometry) (dense version).
line_fill_sparse	Fill the row of the matrix (process a line in the geometry) (sparse version).
msave	Save the matrix A (dense version).
msave_sparse	Save the matrix A (sparse version).
process_ata	Driver to work with (generate, save, view, etc) the matrix $A^T A$ (dense version).
process_sv	Drive to save all right singular vector and/or view a selected one.
project	Methods (2) for generating simulating data by matrix projection on the original image (dense version).
project_s	Methods (2) for generating simulating data by matrix projection on the original image (sparse version).
read_tiff	Load an existing TIFF image in the array of doubles.
set_pixel	Set the matrix (array of doubles) element.

Table 9.9: Methods of the class *cl2m*. Part 1.

Method	Description
sparsify	Convert the dense matrix to the sparse format.
track	Track the position of the mouse.
view	Fork the image viewer for the array of doubles with the certain initial magnification.
write_tiff	Save a TIFF file for the given image (array of doubles).
wload	Load the vector of reciprocals of the diagonal elements of the matrix $W^{1/2}$ (dense version). Then apply the matrix $W^{1/2}$.
wload_sparse	Load the vector of reciprocals of the diagonal elements of the matrix $W^{1/2}$ (sparse version). Then apply the matrix $W^{1/2}$.
wsave	Save the vector of reciprocals of the diagonal elements of the matrix $W^{1/2}$ (dense version).
wsave_sparse	Save the vector of reciprocals of the diagonal elements of the matrix $W^{1/2}$ (sparse version).
~cl2m	Class destructor.

Table 9.10: Methods of the class *cl2m*. Part 2.

Method	Description
line_item	Constructors (2). Define a line and compute the angle of the line with the horizontal axis.

Table 9.11: Methods of the class *line_item*.

Method	Description
addlist	Add an item to the list.
line_list	Constructor of the class <i>line_list</i> .

Table 9.12: Methods of the class *line_list*.

Method	Description
endpoints	Constructor of the class <i>endpoints</i> .

Table 9.13: Method of the class *endpoints*.

line_intersection_vertical; although the most work should be done in the last three methods.

Next, the programmer has to adjust the call to the modified methods in the *line_fill* method of the class *cl2m*.

It is suggested that the developer will create two files, say *linecross2.h* and *linecross2.C* where the new methods will be implemented. In order to reduce the development work, one should consider a new basis as an object with the property to return its length of intersection with the line object. The pixel grid then can be represented by the list of pixel objects.

Also notice that the methods *linecross*, *line_intersection*, *line_intersection_horizontal*, and *line_intersection_vertical* do not belong to any class. This can be modified by creating a new class (say, *linecross*) - a subclass of *cl2m*. The suggested modification will simplify the future development.

9.4 Developing LINE2MAT Interface: LINE2MAT_INTERFACE

9.4.1 Compiling the Program

LINE2MAT_INTERFACE requires the following computer libraries: X libraries (libXm, libXt, libXmu, libXp (Linux only), libXext, libX11), regular mathematical library (libm), and ZIP compression library (libz).

The suggested compiler is gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-85) or gcc version 2.8.1 (IRIX).

Below is a sample Makefile (Linux version) for LINE2MAT_INTERFACE:

```
#!/usr/local/bin/gmake
```

```

OPT=-g
CXX=g++
CXXFLAGS=$(OPT)\
        -I/usr/X11R6/include \

start: start.o l2m.o sparse_task.o radio_button.o new_task.o\
        save_task.o open_task.o read_task.o init_task.o\
        browse_button.o execute_task.o
$(CXX) $(OPT) -o $@ $@.o l2m.o sparse_task.o\
        radio_button.o new_task.o save_task.o open_task.o\
        read_task.o init_task.o browse_button.o execute_task.o\
-L/usr/lib \
-L/usr/X11R6/lib \
-lXm -lXt -lXmu -lXp -lXext -lX11 \
-lm -lz\

```

9.4.2 GUI Design

The Motif GUI consists of the main window with the menu bar and a form. The two row column widgets *left_rc* and *right_rc* divide the form in two parts. Each of the parts (row column widget) contains visible GUI elements placed inside the another row column widget - a child of either *left_rc* or *right_rc*.

9.4.3 Updating this Program

If one decides to use `LINE2MAT_INTERFACE` to work with `LINE2MAT`, it is necessary to make modifications to `LINE2MAT_INTERFACE` corresponding to the changes in `LINE2MAT`.

For instance, if one adds a method to compute a new matrix decomposition in `LINE2MAT`, the labeled text field with the “Browse” button should be added to `LINE2MAT_INTERFACE`; the callback for the button controlling the execution of `LINE2MAT` should be changed as well; and the `LINE2MAT_INTERFACE` task file format should be modified (and implemented in all file I/O methods) accordingly.

9.4.4 Key Methods Description

See Table 9.14 for methods of the class `l2m`.

9.5 Developing INVERT

9.5.1 Compiling the Program

`INVERT` requires the following computer libraries: X libraries (`libXm`, `libXt`, `libXmu`, `libXp` (Linux only), `libXext`, `libX11`), regular mathematical library (`libm`), ZIP compression library (`libz`), BLAS library (`libblas`), LAPACK library (`liblapack`), TIFF image library (`libtiff`), Fortran 77 library (`libF77`), Fortran to C linkage library (`libg2c`), and Sparselib++ suit (including `libsparse`, `libmv`, `libspblas`). While working in IRIX environment, BLAS and LAPACK are substituted by the Scientific Computing Software Library (`libscs`). Moreover, there is no need to use Fortran to C linkage library (`libg2c`)

Method	Description
browse	Display the file selection dialog with the certain filter when the user clicks on any “Browse” button.
buttonF	Read the fields and execute <code>LINE2MAT</code> .
CreateFilePane	Create “File” pane.
CreateHelpPane	Create “Help” pane.
CreateMenuBar	Create menu bar.
dodefault	Initialize task descriptors.
errormessagedisplay	Display the error message in the dialog window.
group1arm	Manage/unmanage GUI elements when the user checks the “Project data file” box.
group2arm	Manage/unmanage GUI elements when the user checks the “Project ice and water images” box.
helpindexCallback	Help index dialog.
helpkeysandshortcuts Callback	Help on keys and shortcuts.
helpoverviewCallback	Help overview dialog.
helpproductinfoCallback	Help on product info.
newb	Start a new project.
openb	Load a project file (driver to <i>readbatch</i>).
popupopen	Popup menu offering to load a project file.
popupsave	Popup menu offering to save a project file.
quitCallback	Quit dialog.
radio	Manage/unmanage GUI elements when the user selects either a dense or a sparse task.
readbatch	Read a project file.
reallyquitCallback	Quit.
saveb	Save a project file.
sparsetask	Manage/unmanage GUI elements when the user checks the “Sparse task” box.
updatebrowse	After the user has selected a file clicking on “Browse” button, the application updates the corresponding text field.

Table 9.14: Method of the class *l2m* (the only class of `LINE2MAT_INTERFACE`). Callback “drivers” are not listed.

on IRIX platforms when libscs is used.

The suggested compiler is gcc version 2.96 20000731 (Red Hat Linux 7.1 2.96-85) or gcc version 2.8.1 (IRIX).

Below is a sample Makefile (Linux version) for INVERT:

```
#!/usr/local/bin/gmake

SPARSELIB=/usr/people/ars22/libs/sparselib
SPARSELIB_INCLUDE=-I$(SPARSELIB)/include
MV_INCLUDE=-I$(SPARSELIB)/mv/include
SPARSELIB_DIR=$(SPARSELIB)/lib
SPARSE_LIBS=-L$(SPARSELIB_DIR) -lsparse -lmv -lspblas

OPT=-g
CXX=g++
CXXFLAGS=$(OPT) \
    $(SPARSELIB_INCLUDE) \
    $(MV_INCLUDE) \
    '-DCOMPLEX=complex<double>' -DCOMPLEX_SUPPORT \
    -I/usr/X11R6/include

invert: invert.o svd.o tikh.o iter.o lsqr.o \
    svd_pane.o tikh_pane.o iter_pane.o \
```

```

viewer_pane.o menu_pane.o\
task_window.o task.o\
work_window.o menu_saver.o data_service.o
$(CXX) $(.ALLSRC) -o $@ $@.o svd.o tikh.o iter.o\
lsqr.o svd_pane.o viewer_pane.o iter_pane.o\
tikh_pane.o task_window.o menu_pane.o work_window.o\
menu_saver.o task.o data_service.o\
-L/usr/lib \
-L/usr/X11R6/lib \
-lXm -lXt -lXmu -lXp -lXext -lX11\
-lm -lz -lblas -llapack -ltiff\
-ltiff -lg2c\
-lF77 $(SPARSE_LIBS)

```

9.5.2 Design Overview

The schematic of the current design of INVERT is shown in the UML diagram in the Figure 9.3.

The responsibilities of the main class *work_window* include providing Motif graphical user interface and data flow to the computation engines. The class *work_window* has an object of type *task* which stores all information concerning the current task including the numerical method used, input file locations, and so on.

INVERT provides an interface to load ASCII, binary, zip-compressed, and sparse data using the functions *read_ascii*, *read_rhs*, *read_gz*, and *read_hb*, respectively. The application also allows the user to add noise using

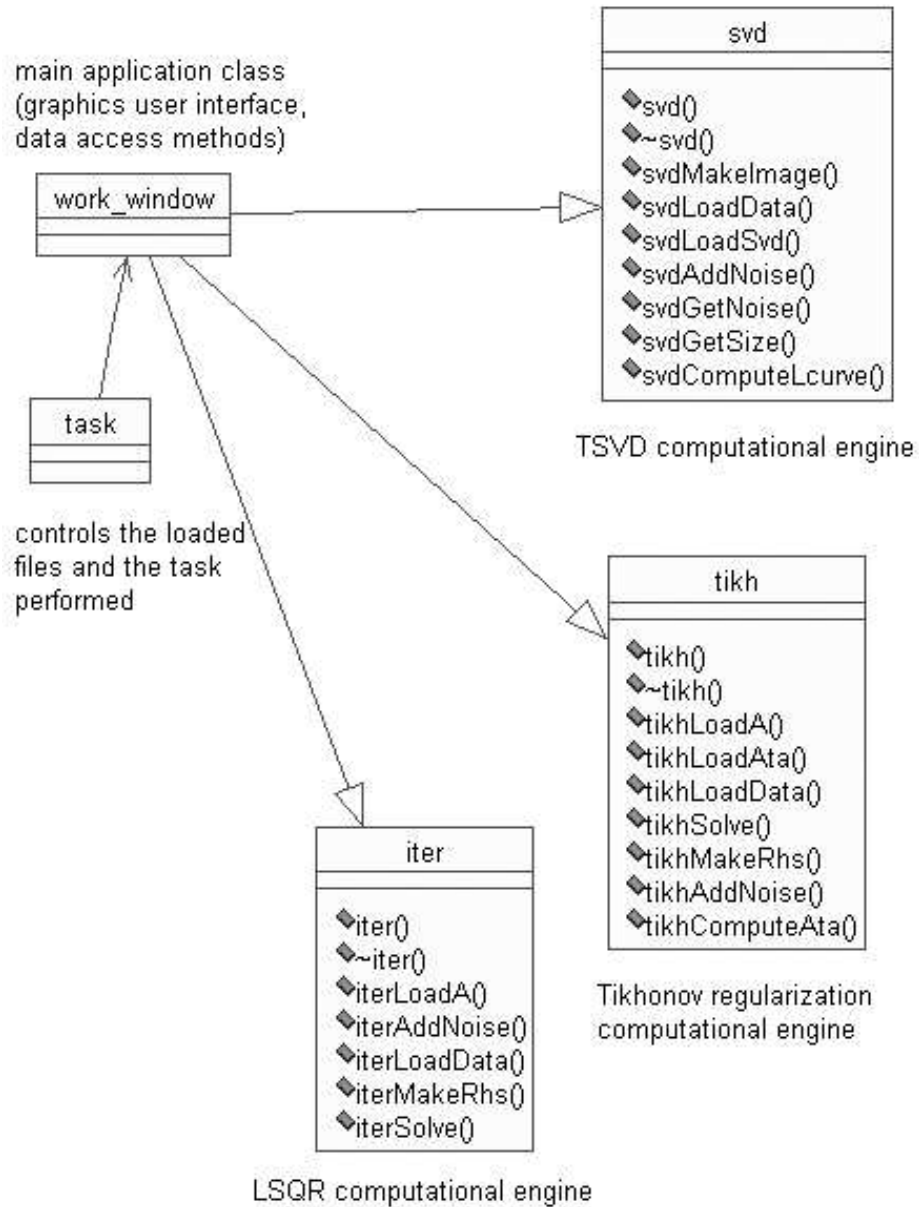


Figure 9.3: Sketch of the INVERT UML class diagram. Only a few methods and data structures are shown.

the method *add_noise*. The functions *read_ascii*, *read_rhs*, *read_gz*, *read_hb*, and *add_noise* do not belong to any class due to access problems caused by the existing software design.

The classes *iter*, *tikh*, and *tsvd* are computational engines implementing LSQR, Tikhonov regularization, and truncated singular value decomposition respectively. Each of the computational engines loads the data using a function from the set *read_ascii*, *read_rhs*, *read_gz*, and *read_hb*. The noise can be added to the right-hand side *b* via the *add_noise* method. Given the required inputs, each computational engine takes care of the reconstruction and returns the solution, as well as other information which helps to analyze the results, to the *work_window* object. The solution is displayed in the image viewer, area and the other results are shown in the method specific part of the reconstruction window.

9.5.3 GUI Design

The graphical interface consists of two major windows - the task window and the reconstruction window.

The task window is implemented in the *work_window* class as the paned window and communicates with the object of the type *task*. Changes in the reconstruction window data fields are reflected when the user clicks either “OK” or “Apply” button. This action calls a registered callback which “talks” to the computational engines.

The reconstruction window is the main window implemented in the constructor of the *work_window* class. It consists of three parts. Its first part is the menu pane. The second fragment (Motif form) is the reconstructed

image viewer. These two parts are common to the displays of all computational engines (except for several items in the menu pane). The third part (Motif form) represents the inputs and outputs specific for each implemented method. When the *work_window* class is instantiated, only the first two parts of the window are created, and only the menu pane is displayed. The third part (form) is created upon user request from the task window. If the user selects a numerical reconstruction method while working with another one, the third part of the reconstruction window is automatically “unmanaged”, and control is provided to the form appropriate to the newly selected reconstruction method.

9.5.4 Key Methods Description

See Table 9.15 for main application methods.

The main application also includes a collection auxiliary functions, data type definitions, and function prototypes for the iterative linear solver LSQR. See the files *lsqr.h* and *lsqr.C* for more information.

See Table 9.16, Table 9.17, and Table 9.18 for methods of the class *work_window*. See Table 9.19 for methods of the class *svd*. See Table 9.20 for methods of the class *tikh*. See Table 9.21 for methods of the class *iter*.

9.5.5 Adding a New Computational Engine

If the developer decides to add a new computational engine implementing the method *newmethod*, one has to follow the procedure outlines below.

First, create a class *newmethod* with the methods to load data and add noise (drivers to the functions in the main application domain - do not forget

Method	Description
add_noise	Add randomly distributed noise to the data.
byte_swap	Swap bytes in doubles when loading binary data from IRIX platforms on Linux machines.
byte_swap_i	Swap bytes in integers when loading binary data from IRIX platforms on Linux machines.
main	Main method.
multiply	Routine for matrix-vector products used by LSQR method.
preprocess_func	Preprocess data (see user manual for more information).
read_ascii	Load data vector stored in an ASCII file.
read_hb	Load a sparse compressed matrix from the file.
read_gz	Load binary data stored in a compressed gzip file.
read_rhs	Load right-hand side vector(s) - file(s) with extension "v" and/or "ir".

Table 9.15: Main application methods.

to include the file *data_services.h* in the class header file). Create methods to make the image (if necessary), weight the data, and perform the reconstruction. Make sure that the methods implement the preprocessing of the right-hand side data and byte swapping of all binary data. It is suggested that the methods should have the same inputs as in classes *svd*, *tikh*, and *iter*.

Second, change inheritance properties of the class *work_window* in the file *work_window.h* as follows:

```
class work_window: private svd, Tikh, Iter, newmethod
```

Then, the main class *work_window* will be able to access members and attributes of the *newmethod* class.

Third, edit the definition of the class *task* and add new members corresponding to required input information for the class *newmethod*.

Method	Description
apply	Run a LSQR reconstruction with the input parameters specified in the text fields in the LSQR pane (reconstruction window).
browse	Popup the fileselection dialog when the user clicks on the “Browse” button in the task window.
buttonCancelCallback	Close the dialog.
change_ice_center	Change the value of the ice image “center” when the corresponding slider in the image viewer pane of the reconstruction window is moved.
change_ice_width	Change the value of the ice image “width” when the corresponding slider in the image viewer pane of the reconstruction window is moved.
change_water_center	Change the value of the water image “center” when the corresponding slider in the image viewer pane of the reconstruction window is moved.
change_water_width	Change the value of the water image “width” when the corresponding slider in the image viewer pane of the reconstruction window is moved.
CreateFilePane	Create “File” menu pane.
CreateHelpPane	Create “Help” menu pane.
CreateMenuBar	Create menu bar.
CreateValuesPane	Create “Values” menu pane.
CreateViewPane	Create “Viewer” menu pane.
draw	Driver to fill the image with graphical context corresponding to the reconstructed image.
errormessagedisplay	Display the error message in the dialog.
fillImage	Fill the image with pixels.
GetTopShell	Get the top shell widget.
helpindex	Help index dialog.
helpkeysandshortcuts	Help keys and shortcuts dialog.
helpoverview	Help overview dialog.
helpproductinfo	Help product information dialog.

Table 9.16: Methods of the class *work_window* (callback and event “drivers” are not listed). Part 1.

Method	Description
hideIceResidual	Hide ice residual.
hideWaterResidual	Hide water residual.
it_constructor	“Constructor” of the LSQR pane.
lcurve	Call the method to compute L-curve.
loadIceResidual	Load, compute, and display ice residual.
loadWaterResidual	Load, compute, and display water residual.
magnification	Update the image magnification.
make_image	Make the image (driver function).
menuSaver	Popup a dialog to save the image as TIFF.
menuSaver_	Popup a fileselection dialog when the user selects to save image as TIFF.
new_constructor	“Constructor” of the Tikhonov regularization pane.
noise	Call the method to add the noise.
old_constructor	“Constructor” of the SVD pane.
popupIceFileSelectionBox	Popup a fileselection dialog to load ice residual.
popupLoadtask	Popup a fileselection dialog to load a task file.
popupSavetask	Popup a fileselection dialog to save a task file.
popupTaskbox	Popup the task window.
popupWaterFileSelectionBox	Popup a fileselection dialog to load water residual.
preprocchange	Manage and unmanage the group of widgets corresponding to data preprocessing in the task window.
quitCallback	Popup quit dialog.
reallyquit	Quit.

Table 9.17: Methods of the class *work_window* (callback and event “drivers” are not listed). Part 2.

Method	Description
resetSliders	Reset the sliders to the original values of “center” and “width” in the image viewer pane of the reconstruction window.
savetask	Save a task file.
saveTIFF	Save the image as TIFF.
showg	Show group of widgets controlling visualization properties in the image viewer pane of the reconstruction window.
taskApply	Apply the task.
taskCancel	Cancel the task (close the task window).
togglelink	Toggle the “center” and “width” sliders in the image viewer pane of the reconstruction window.
toggle_	Manage and unmanage the group of widgets corresponding to display of the information about the reconstruction (in the bottom of the reconstruction window) for LSQR method.
trackmouse	Track the mouse pointer and display the value of the pixel in the labeled text area.
updatebrowse	Update the text field when the user selects a file in the fileselection dialog.
updateImage	Image update.
updateSliders	Update the sliders in the image viewer pane.
updatethemagnification	Popup a dialog to update the image magnification.
updatethenoise	Popup a dialog to add the noise.
viewer_pane	Create an image viewer pane in the reconstruction window.
work_window	Constructors (2) of this class.
~work_window	Class destructor.

Table 9.18: Methods of the class *work_window* (callback and event “drivers” are not listed). Part 3.

Method	Description
svd	Default constructor.
svdAddNoise	Add noise (driver to the function <i>add_noise</i>).
svdComputeLcurve	Compute L-curve.
svdGetNoise	Return noise level value.
svdGetSize	Return the number of singular triplets used.
svdLoadSvd	Load the precomputed singular value decomposition (drivers to the functions <i>read_gz</i> , <i>byte_swap</i> , <i>byte_swap_i</i> , and <i>read_ascii</i>).
svdLoadData	Load the right-hand side data (drivers to the functions <i>read_rhs</i> , <i>byte_swap</i> , and <i>preprocess_func</i>).
svdMakeImage	Make image.
~svd	Destructor.

Table 9.19: Methods of the class *svd*.

Method	Description
tikh	Default constructor.
tikhAddNoise	Add noise (driver to the function <i>add_noise</i>).
tikhComputeAta	Compute the matrix $A^T A$.
tikhLoadA	Load the matrix A (drivers to the functions <i>read_gz</i> , <i>byte_swap</i> , and <i>byte_swap_i</i>).
tikhLoadAta	Load the matrix $A^T A$ (drivers to the functions <i>read_gz</i> , <i>byte_swap</i> , and <i>byte_swap_i</i>).
tikhLoadData	Load the right-hand side data (drivers to the functions <i>read_rhs</i> , <i>byte_swap</i> , and <i>preprocess_func</i>).
tikhLoadWeight	Load the reciprocals of the diagonal elements of the matrix $W^{1/2}$ (drivers to the functions <i>read_gz</i> , <i>byte_swap</i> , and <i>byte_swap_i</i>).
tikhMakeRhs	Make the right-hand side.
tikhSolve	Solve the Tikhonov regularization.
tikhWeightA	Weight the matrix A .
tikhWeightRhs	Weight the right-hand side.
~tikh	Destructor.

Table 9.20: Methods of the class *tikh*.

Method	Description
iter	Default constructor.
iterAddNoise	Add noise (driver to the function <i>add_noise</i>).
iterLoadA	Load the matrix <i>A</i> (driver to the function <i>read_hb</i>).
iterLoadData	Load the right-hand side data (drivers to the functions <i>read_rhs</i> , <i>byte_swap</i> , and <i>preprocess_func</i>).
iterLoadWeight	Load the reciprocals of the diagonal elements of the matrix $W^{1/2}$ (drivers to the functions <i>read_gz</i> , <i>byte_swap</i> , and <i>byte_swap-i</i>).
iterMakeRhs	Make the right-hand side.
iterWeightA	Weight the matrix <i>A</i> .
iterWeightData	Weight the right-hand side.
iterSolve	Solve LSQR.
~iter	Destructor.

Table 9.21: Methods of the class *iter*.

Fourth, edit the methods of the class *work_window* responsible for the task window. Make sure that you add necessary GUI elements, change procedure to load and save the task file according to the modifications of the task file format.

Fifth, create a method of the class *work_window* to construct a pane for the *newmethod*. The pane is based on Motif form with the collection of widgets controlling the reconstruction using *newmethod*. Register and implement all callbacks of the widgets of this pane.

Sixth, modify common methods of the class *work_window*, such as *updatethemagnification*, *updatethenoise*, *draw*, etc.

Chapter 10

Appendix C: Sparse Matrix Formats

To illustrate various matrix storage formats used in this project, consider a nonsymmetric sparse matrix:

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 & 3 \\ 4 & 5 & 6 & 0 & 0 \\ 0 & 7 & 8 & 0 & 9 \\ 0 & 0 & 0 & 10 & 0 \\ 11 & 0 & 0 & 0 & 12 \end{pmatrix}. \quad (10.1)$$

The most straightforward approach is to save the row and column coordinates of the nonzero elements. The matrix can be stored in the following arrays:

$$Values = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) \quad (10.2)$$

$$RowIndices = (0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 4, 4) \quad (10.3)$$

$$ColumnIndices = (0, 1, 4, 0, 1, 2, 1, 2, 4, 3, 0, 4). \quad (10.4)$$

We notice that for the sample matrix we have many repeating numbers corresponding to row indices. We want to avoid any repetitions and consider a more efficient compressed row storage scheme.

The compressed row storage format views the nonzero elements in each row as a sparse vector, storing pointers to the first element in each row, as well

as nonzero values and their associated column indices in the corresponding arrays. We add an additional element to the row array to represent the number of nonzero array elements:

$$Values = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) \quad (10.5)$$

$$RowPointers = (0, 3, 6, 9, 10, \mathbf{12}) \quad (10.6)$$

$$ColumnIndices = (0, 1, 4, 0, 1, 2, 1, 2, 4, 3, 0, 4). \quad (10.7)$$

Similarly one can implement compressed column storage scheme which saves column pointers instead of row pointers.

BIBLIOGRAPHY

- PostScript(R) Language Tutorial and Cookbook*. Addison Wesley, 1985.
- A Sparse Matrix Library in C++ for High Performance Architectures*, 1994. Proceedings of the Second Object Oriented Numerics Conference.
- E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, S. Ostrouchov, and D. Sorenson. *LAPACK User's Guide*. SIAM, Philadelphia, PA, 1992.
- L. Bergamaschi and G. Zilli. Quasi Newton methods for sparse systems of nonlinear equations on parallel platforms. *Recent Advances in PVM and MPI*, 1997.
- Michael Berry, Theresa Do, Gavin O'Brein, Vijay Krishna, and Sowmini Varadhan. SVDPACKC (version 1.0) user's guide. Technical Report CS-93-194, Department of Computer Science, University of Tennessee, Knoxville, TN, 1996.
- Åke Björck. *Numerical methods for least squares problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.
- L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: a linear algebra library for message-passing computers. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997)*, page 15 pp. (electronic), Philadelphia, PA, 1997. SIAM.
- James Demmel. On floating point errors in Cholesky. Technical report, Courant Institute, NY, 1989. LAPACK Working Note 14.
- James Demmel, Ming Gu, Stanley Eisenstat, Ivan Slapničar, Krešimir Veselić, and Zlatko Drmač. Computing the singular value decomposition with high relative accuracy. *Linear Algebra Appl.*, 299(1-3):21–80, 1999.
- Jean-Ioup Gailly and Mark Adler. ZLIB (version 1.1.3): General purpose compression library manual. <http://www.gzip.org/zlib/manual.html>, 1998.
- Per Christian Hansen. *Rank-deficient and discrete ill-posed problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1998.

- Dan Heller and Paula Ferguson. *Motif Programming Manual (The Definitive Guides to the X Window System, Volume 6A)*. O'Reilly and Associates, Inc, 1993.
- Steven H. Izen and Timothy J. Bencic. *Application of the Radon transform to calibration of the NASA-Glenn Icing Research Wind Tunnel*. Amer. Math. Soc., Providence, RI, 2001.
- Mark J. Kilgard. OpenGL and X, Part 3: Integrating OpenGL with Motif. <http://www.sgi.com/software/opengl/glandx/motif/motif.html>, 1994.
- C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Trans. Math. Software*, 5:308–323, 1979.
- A. Lumsdaine, R. Pozo, and K. Remington. Sparselib++ sparse matrix class library: User's guide. <ftp://gams.nist.gov/pub/pozo/docs/sparselib.ps.gz>, 1996.
- F. Natterer. *The mathematics of computerized tomography*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.
- Christopher C. Paige and Michael A. Saunders. Algorithm 583. LSQR: sparse linear equations and least squares problems. *ACM Trans. Math. Software*, 8(2):195–209, 1982.
- Christopher C. Paige and Michael A. Saunders. LSQR: an algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8(1):43–71, 1982.
- B. N. Parlett and D. S. Scott. The Lanczos algorithm with selective orthogonalization. *Math. Comp.*, 33(145):217–238, 1979.
- Ronald H. Soeder, David W. Sheldon, Charles R. Andracchio, Robert F. Ide, David A. Spera, and Nick M. Lalli. *NASA Lewis Icing Research Tunnel User Manual*. Cleveland, OH, 1996.
- Lloyd N. Trefethen and David Bau. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.
- Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison Wesley, 2001.
- Douglas Young and Doug Young. *The X Window System: Programming and Applications with XT, OSF/Motif*. Prentice Hall, 1994.