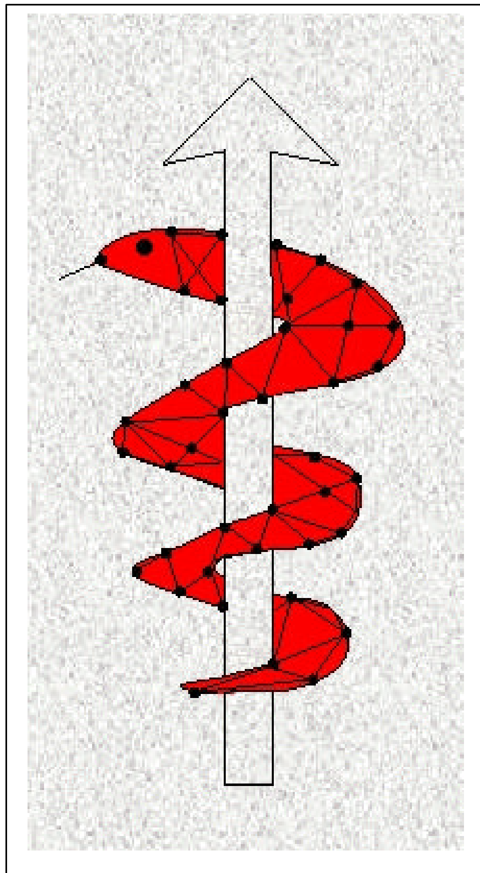
The IST Programme
Project No. 10378

SimBio

SimBio - A Generic Environment for Bio-numerical Simulation

http://www.simbio.de



D4.1a: Inverse Problem Methodology Design Report

Status:	Final
Version:	5.0
Security:	Public
Responsible:	A.N.T Software
Authoring Partners:	A.N.T Software MPI of Cognitive Neuroscience

Release History

Version	Date
1.0	07.27.00
2.0	03.08.00
3.0	22.08.00
4.0	18.09.00
5.0	29.09.00

The SimBio Consortium :

NEC Europe Ltd. – UK
A.N.T. Software – The Netherlands
K.U. Leuven R&D – Belgium
ESI Group – France
Smith & Nephew - UK
MPI of Cognitive Neuroscience – Germany
Biomagnetisches Zentrum Jena – Germany
CNRS-DR18 – France
Sheffield University – UK

© 2000 by the **SimBio** Consortium

Contents:

1. Introduction 3

2. Inverse methods 3

2.1 General classification 3

2.2 Inverse Methods using a continuous parameter space 4

2.2.1 Source Models: Single and multiple dipoles 4

2.2.2 How to measure the goodness of the model: Goal functions 4

2.2.3 How to find optimal parameters: Non-linear optimization procedures 5

2.3 Inverse methods using a discrete parameter space 5

2.3.1 Source models: Distributed sources 5

2.3.2 Spatial weighting 5

2.3.3 Regularization 6

2.3.4 Smoothed L2-Norm 6

2.3.5 Non-linear L1-Norm 6

2.4 Scanning methods 7

2.5 Discrete dipole fit methods 7

2.6 Simulators 8

2.7 Additional methods 8

3. Software design of generic toolbox 8

3.1 General guidelines 8

3.2 Class interface 9

3.3 Access to methods of the inverse toolbox 14

3.4 User interface I 15

3.5 User interface II 16

3.6 User interface III 17

3.7 Adding user scenarios 18

3.8 Interaction of the S-Cauchy Fortran 77 code in the inverse toolbox 18

3.9 Interaction with WP3 19

3.10 Forward simulator adapter class 20

3.11 Specification of graphical user interface 21

4. Error estimation package 24

4.1 General Description 24

4.2 Class interface 26

5. Software quality management 28

5.1 Software version control 28

5.2 Software test protocols 29

5.3 Bug database 30

5.4 Backup protocol 32

References 33

Appendix A: Commands and parameters for user interfaces 35

Appendix B: Examples for the creation of new user scenarios 42

Appendix C: Class definitions 48

SimBio: A Generic environment for bio-numerical simulation

Workpackage 4 Subtask 4.1

Detailed design report

1. Introduction

Localizing the sources of electrical activity of the brain by inverse methods has an important impact for surgical planning, either to identify regions which generate activity due to pathologic alterations of nerve cells, like in epilepsy, or to delineate regions with functions of high impact, that should strictly be avoided to be excised. But inverse methods are not restricted to clinical use. They allow insight in the localization of neuronal processes by a high time resolution, which can not be achieved by imaging methods like functional MRI. Thus they can be applied for the exploration of a wide variety of aspects in neuroscience. Results of WP2, which provides individual conductivity maps, combined with FEM methods of WP3 will improve the performance of inverse methods beyond the current status.

The objective of WP 4 Subtask 4.1 will be the generation of a generic inverse toolbox containing a large variety of inverse problem solvers and an error estimation package, which will include methods to estimate the sensitivity of the results of source reconstruction to inaccuracies in forward modeling. Software design should assure a generic and modular implementation of the inverse methods. Thus there should be a simple interface to add new methods and to reuse already implemented algorithms.

In this design report the methods that will be included in the inverse toolbox are defined. Furthermore this report gives a guideline for the implementation of these methods. This is done by describing the methods, giving references to detailed descriptions of the methods and defining the software environment, e.g. used platforms, compilers, class interfaces, user interfaces, file interfaces and software quality management. This report should also give workgroups of other workpackages, (especially WP3 of the SimBio consortium) a complete and plain description how to integrate algorithms to the inverse toolbox.

2. Inverse methods

2.1 General classification

A variety of methods have been developed for the solution of the electromagnetic inverse problem. All these methods object to gain insight into the source distribution that is responsible for measured magnetic fields or electric potentials (MEG/EEG/ECoG). Since this problem is in general non-unique, further assumptions have to be made. On the basis of reasonable assumptions a number of standard and application tailored algorithms will be part of the generic toolbox, which will be realized in ST4.1.

Generally, inverse methods can be divided into two classes. A first category of methods is characterized by a *continuous search space* for parameters to be determined. Such parameters could include e.g. source position, orientation etc.. For a given search space, parameters have to be optimized with respect to a *goal function*. The goal function determines a goodness of fit value between measured data and predicted data generated by estimated sources. For the determination of the predicted data a *simulator* is needed, which computes for a given source paramters the potentials or magnetic fields for a sensor configuration (electrodes or gradiometers). For the optimization of the source parameters, *non-linear optimization* methods are used.

The parameters of the second set of methods have a discrete search space. For example a finite set of positions of the sources is assumed. This can be either a small set of point-like sources the positions of which are known a priori, for example by anatomical knowledge, or an entire source region, divided into small voxels, each of them represented by a source. Generally the search space can be described by a *grid*, which defines the positions of possible sources. Also methods with a discrete parameter space need a *simulator*, which determines the influence of the sources on measured data.

The following chapters will give a more detailed insight into the mentioned approaches and supplementary algorithms.

2.2 Inverse Methods using a continuous parameter space

If the extent of the neuronal activity responsible for the measured data can be assumed to be restricted to a focal area, it is possible to describe the sources by single point-like dipolar current sources. Parameters of these dipolar sources e.g. positions, orientations, etc. are determined in a continuous parameter space.

2.2.1 Source Models: Single and multiple dipoles

Several classes of dipoles can be derived depending on the assumptions on their behavior in space and time: 1. Fixed dipoles: with one fixed position and orientation shared by all time samples. Only the strength of the dipole is variable over time. 2. Rotating Dipoles: with a fixed positions for all time points but separate directions for the each time steps. 3. Moving dipoles: the parameters are determined independently for each time sample. A non-linear optimization procedure is necessary to find the positions (and possibly other parameters) of the dipoles. This method is referred to as non-linear approach or dipole localization [1],[2].

2.2.2 How to measure the goodness of the model: Goal functions

To obtain a measure for the goodness of a model and to optimize the parameters of a model a *goal function* is needed. It performs the estimation of the goodness of fit between measured and predicted data generated by estimated sources. For the estimation of the similarity of measured and predicted data three different types of criteria will be realized in the inverse toolbox:

1. Minimum square error, which minimizes the square of the difference between measured data and predicted data by the estimated sources.
2. Maximum entropy, which favors solutions, which require minimal additional information [3].
3. Maximum probability, which maximizes the probability of a solution under preassumption of given measured data and additional constraints [3].

The goal functions will be provided with additional information to obtain reasonable results. First a *search space* for the parameters has to be defined. For position parameters a search volume will be defined, which is in the most simple case an infinite search volume. For more realistic scenarios boundaries will be introduced to restrict the parameter space. To prevent parameters outside the search volume, the goal function uses a penalty.

A further application of penalties is for example the suppression of sources with a radial direction in case of MEG measurements.

2.2.3 How to find optimal parameters: Non-linear optimization procedures

The inverse algorithms need a strategy to obtain optimized parameters. Standard optimization procedures for non-linear problems can be used for dipole parameter determination. Two different algorithms, which are known to be very effective, will be implemented in the inverse toolbox for the continuous parameter space:

1. Simplex algorithm [4].
2. Levenberg Marquardt algorithm [4],[5].

2.3 Inverse methods using a discrete parameter space

2.3.1 Source models: Distributed sources

Distributed source models are based a on set of dipoles with known positions. The positions and orientations of the dipoles are provided as a grid. Grid configurations in the inverse toolbox will be either regular 3-D grids and grids, which are provided by realistic head models. These can be either 2-D surface grids or 3-D volume grids. Grid generators for FEM (Finite Element Method) 3-D grids will be provided by WP 3.

For the description of the relationship between the sources and signal strength at the sensors the concept of the lead field matrix is introduced:

$$F = L \cdot Q$$

F is the matrix with measured values, Q consists of the source strengths and L describes the sensitivity of measurement sensors with respect to each source strength. Every row of F and L is associated with a particular measuring channel. The columns of F and Q represent the time samples. Each column of the leadfield matrix L represents the (normalized) contribution of one particular source to the data. Generally, there is no unique solution of this equation for the determination of the sources Q, because normally there are many more source reconstruction points than measurement channels. Thus, additional assumptions have to be made. If one uses the Moore-Penrose pseudoinverse to determine Q, one obtains the solution with the smallest quadratic norm (L2-norm) of the sources. This method is called the minimum norm least square (MNLS) estimate [6].

2.3.2 Spatial weighting

The equation for the determination of the sources can be extended by introducing a spatial weighting matrix:

$$Q = G \cdot L^T \cdot (LGL^T)^{-1} \cdot F.$$

One reason for spatial weighting is that MNLS methods prefers dipoles that are close to the sensors. This can be prevented by normalizing the lead field matrix [7],[8], using weighting factors for each column of the lead field matrix, which stress or damp the components of the source vectors. Thus, no locus will be favored in the result.

The “Loreta” (Low Resolution Electromagnetic Tomography) algorithm [9] determines the solution with the smallest 2nd spatial derivative. This can be achieved by a weighting matrix, which is not a diagonal matrix.

2.3.3 Regularization:

Generally the solution of the current density reconstruction is ill posed. To obtain stable results, that do not much depend on small alterations of measured data, the matrix to be inverted has to be regularized. Especially noise which is part of every measurement should not influence the results of the source reconstruction.

Two algorithms for the regularization of the matrix will be implemented in the inverse toolbox. The first algorithm is called Tikhonov-Philips [10] regularization. Therefore the above introduced equation is extended by adding $\lambda \cdot I$ to the matrix to be inverted.

$$Q=G\cdot L^T\cdot(LGL^T+\lambda\cdot I)^{-1}\cdot F.$$

Here λ is the *regularization parameter* or *Lagrangian multiplier* and I the unary matrix. The inverse toolbox will include two strategies for its determination. The first strategy uses the property of λ, that it has to lie within a certain statistical range [11], [12]. The inverse toolbox can iterate λ until the statistical range depending on the measured data is reached. Then an inverse reconstruction will be computed. This function is called *lambda iteration.* A second method to determine the optimal λ-value will be implemented in the *convergence test* algorithm. A convergence test is a sequence of inverse solutions with different user defined values for λ. This option is useful for determining the most appropriate λ. For a sufficient number of data points, it allows plotting an L-shape curve. The best λ can be found on the curve at the point with the maximum curvature.

The second method used for regularization inside the inverse toolbox is the truncated singular value decomposition. The singular value decomposition decomposes a matrix A in intro three matrices:

$$A = U\cdot W\cdot V$$

The matrix W is a diagonal matrix containing the singular values of the matrix A. Small singular values (compared to the maximum singular value) of the matrix to be inverted are responsible for large values in the result. Additionally small singular values are responsible for a high sensitivity of the result to noise. To prevent these influence of small singular values on the result, small singular values are set to zero. This method is called truncated singular value decomposition (TSVD).

2.3.4 Smoothed L2-Norm

The L2-norm algorithm will be implemented using different degrees of smoothness of the model term. It can be discrete or continuous with respect to the values or gradients of the distribution. The continuous case is implemented using an integral formulation of the FEM functions. The degree of smoothness can be of zero, first and second order. The higher the order, the smoother and the more coherent will be the current reconstruction. The implemented L2 algorithm resembles the LORETA algorithm [11]. Order zero takes into account only (incoherent) „peak activity“, first order accounts for node values of adjacent nodes and finally second order uses the gradient of the distribution, which results in a smooth, coherent reconstruction. The smoothed L2-norm represents a linear inverse problem solver and is highly advantageous with regard to the computation time.

2.3.5 Non-linear L1-Norm

Beside the L2-Norm regularization, the inverse toolbox will contain the L1-Norm current source reconstruction. The L2 Norm regularization creates a smooth distribution of the potential values because it tends to minimize the sum of the squares of all individual activity. The L1-norm minimizes the sum of the activity and it accounts better for the fact that measured data grows with the amount of individual activity and not with the square of it.
The L1-norm minimizes $\Sigma\ |J|$.

Non-linear equation systems cannot be solved with linear algebra, so non-linear solvers have to be used. The common disadvantage of non-linear problems is the highly increased computation time. The computation time increases for smaller λ values because of a worse condition number and matrices close to singularity for very small values. The inverse toolbox provides a discrete CG (conjugate gradient) solver for the inverse computation using the L1 Norm.

2.4 Scanning Methods

For scanning methods the parameter space is discretized and at every scan point a goal function value is computed. This way, global and local extrema of the goal function can be found very reliably. For reasons of computational costs they can only be applied, if there are very few non-linear parameters. In the inverse toolbox this kind of algorithms will be represented by the "Goal Function Scan" (GFS) [1] and "MUltiple SIgnal Characterization" (MUSIC) [13] algorithms. GFS is suitable in cases with one dipolar source. If more sources have to be determined it exceeds in most cases computational resources. The MUSIC algorithm has been developed to deal with such cases. Inside the MUSIC algorithm a probe source is scanned through the region of interest. At each test location the optimal dipole is computed and its array manifold (the curve of all outputs in the measurement space that can be produced by varying the magnitude of this dipole) is projected onto the signal subspace. This projection gives a probability measure which peaks at true source locations.

2.5 Discrete Dipole Fit Methods

One possibility for the inverse current reconstruction is the search for a limited number of focal current sources to explain the measured EEG/MEG signals. The search space for the dipole locations can be defined on the volume or surface grid. The selected influence node space either contains every node of a grid or just a selection of the nodes of special interest. The nodes in the influence space are marked in the grid representation by an influence node label. Separated influence spaces can be defined for different dipoles. In this case, the influence space incorporates several distributed domains. If besides of the cortical activation a further activation within the brain (for example thalamus) is assumed or if the origin of electrical activity is unclear a volume search is the adequate method. A volume search contains possible source nodes within the brain (as a volume).

For the reconstruction of focal activity with only a few electrically active dipoles, a Simulated Annealing (SA) algorithm is provided. Simulated Annealing utilizes the concepts of combinatorial optimization for finding the subset of all conceivable dipole locations, which best matches, the measured data [3],[14],[15],[16]. This dipole fit method in the discrete parameter space searches for a global minimum of the function. The procedure to find the best dipole configuration is split in two stages. This is due to the property that the potentials depend non-linearly on the dipole location but linearly on dipole strength and orientation. The algorithm first selects a subset of nodes from the influence nodes. In a second step linear least squares methods compute the best fit to the measured signals with respect to the noise in the data. When only one dipole is assumed a goal function scan is the quickest way to the solution.

For a surface search, additionally a normality constraint can be applied. Apart from the model aspect the normal constraint option reduces the degree of freedom at a possible source node to one, which implies that the computation time for the lead-field matrix is reduced by a factor three. For some applications it is absolutely necessary to use a fine representation of the cortex containing for example sulcus structures. A quasi-free search can be led by generating a free surface within the brain. The local curvature of the brain can be taken into account by modeling a free surface representing the brain's surface. The surface can be arbitrary and independent in shape, element size, element number and placement within the head.

2.6 Simulators

To obtain reference values for the goodness of fit for the described methods one has to predict EEG/MEG data at the sensor positions. Therefore a forward calculations has to be performed on the basis of the estimated sources, which needs a model of the electric/magnetic properties of the head. From simple to more realistic models of the head, following models will be realized in the inverse toolbox:

1. For a first approximation the head can be considered as a set of concentric spheres. The single spheres represent the properties of the different tissues of the head: scalp, skull, cerebro-spinal fluid (CSF), and the brain. Thus the head model is defined by the radii of the spheres and conductivity values of the tissues inside the spheres.

2. A refined model of the head uses boundaries based on the main tissue boundaries of the individual head. These boundaries are the scalp surface, inside and outside boundaries of the skull, surface of the brain and possibly the ventricles. The shape of the boundary is approximated by a set of geometric elements. Therefore these models are referred as boundary element models (BEM) [17]. Planar triangles are typically used for the approximation of the boundaries.

3. In order to reproduce the complex inhomogeneitiy and anisotropy of the conductivity of the head tissue, the head volume can be divided into volume elements of regular shape. For each of the elements the field equations are solved with a different conductivity sensor. This approach is called finite element method (FEM) [11]. Up to now a major drawback of FEM methods are high computational costs. Inside the SimBio Project FEM methods will be performed by WP 3 using high performance computers. For these methods a software interface to the inverse toolbox will be implemented.

2.7 Additional Methods

Supplementary to the above described inverse methods basic methods will be provided. These will be template classes for vectors, matrices and 3-D blocks, which provide basic operations. More complex matrix operations needed by the inverse algorithms will be implemented and will be generally available.

3. Software design of generic toolbox

3.1 General Guidelines

In agreement with the general objectives of the SimBio project the inverse toolbox will be designed in a highly generic, modular and portable way. This will be achieved by using ANSI C++ [18], which is implemented for nearly all computer platforms and operating systems and allows for multiple inheritance of classes as well as abstract classes. In addition, algorithms needing high computational power can be efficiently implemented in ANSI C++.

Software developed for the inverse toolbox will be tested on a Windows platform in a complete test environment using the Microsoft Visual C++ software development environment and on a Unix (Linux) system using the Gnu C++-Compiler. Software documentation will be provided platform-independent by using HTML.

To obtain a software design which is highly reusable, a sophisticated class structure will be implemented. Abstract classes will be used to minimize the number of interfaces, to get a simple access to methods and to keep implementation details out of interfaces. Thus simple interfaces for the implementation of algorithms, which will either directly be implemented as a part of WP4 ST41 or adapted from S-Cauchy, will be achieved. How to interface methods of S-Cauchy written in Fortran 90 to methods using the C++ class interface is described in [19].

3.2 Class Interface

The current version of the class interface will be available by the access restricted document part of the SimBio WWW-pages [20].

According to the separation of methods with either a discrete or continuos parameter space the class structure will be divided into two branches.

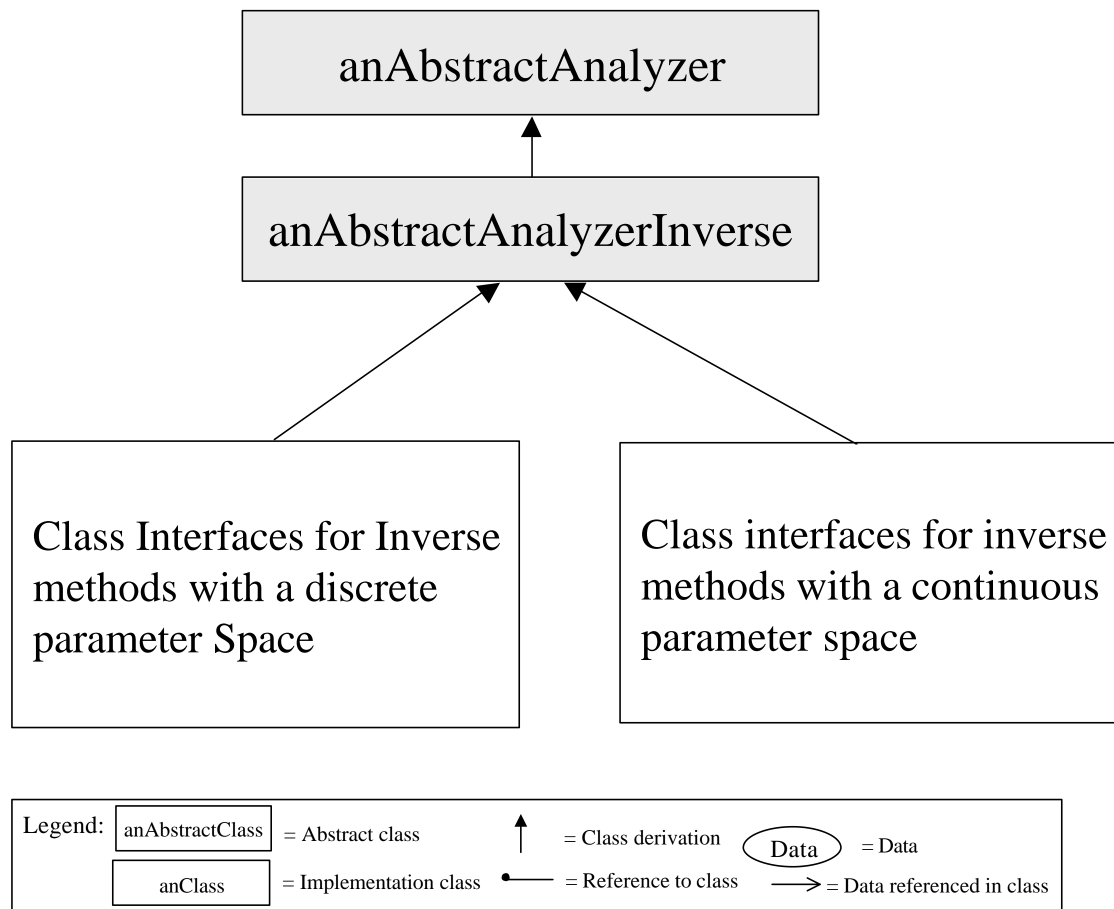


Fig 3.1 Overview about class structures of the inverse toolbox. According to the two different approaches of source modeling, two groups of classes are derived from the universal abstract inverse analyzer class.

All kinds of analyzer algorithms will be derived from the abstract class “anAbstractAnalyzer”. Respectively all inverse algorithms will be derived from the abstract class “anAbstractAnalyzerInverse.

For the class of source model methods using a discrete parameter space following class structure will be realized (Fig 3.2, flow diagram: Fig 3.4, Fig. 3.5):

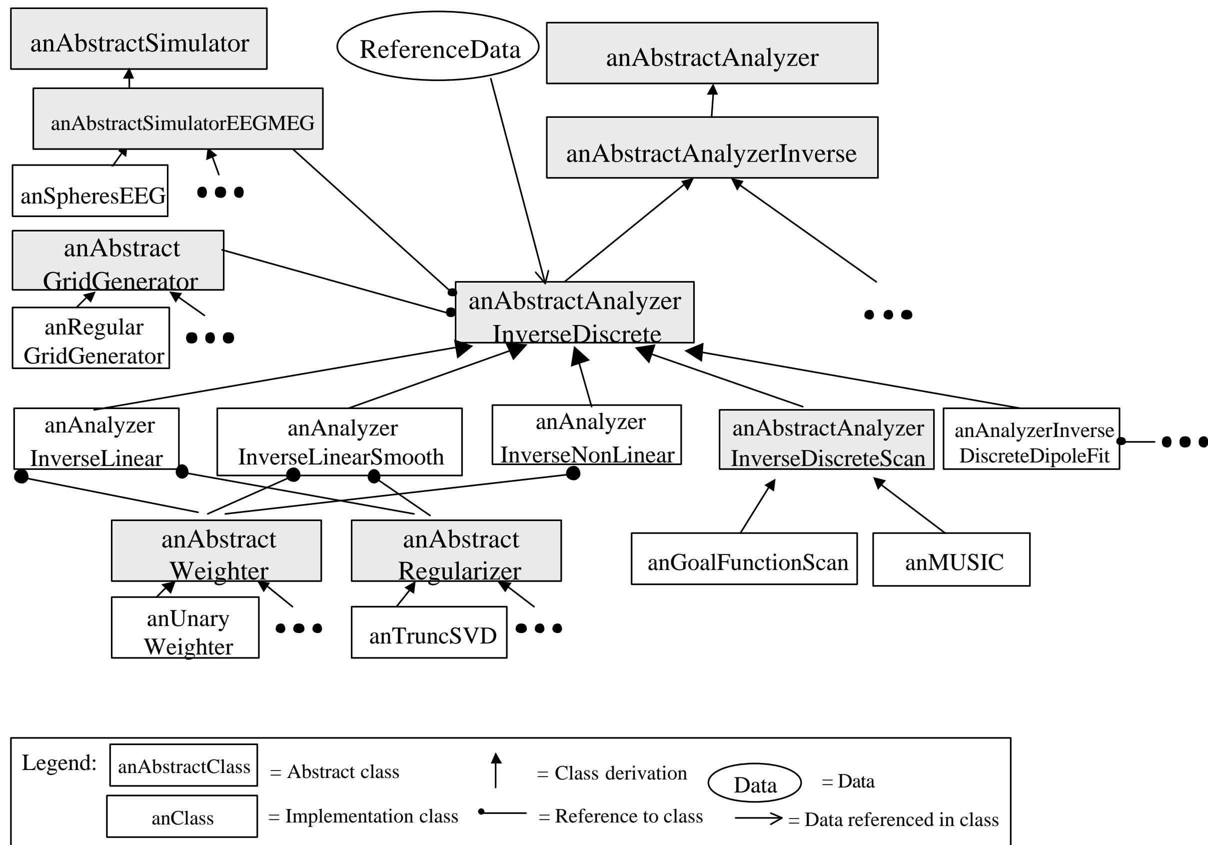


Fig 3.2 Class structure for the implementation of methods using a discrete search space for their parameters.

Common to all these algorithms is the need of a simulator, which performs forward calculations, and a grid generator, which generates the discrete source (parameter) space. Additionally they need the measured reference data. These components are referenced in the abstract class definition of "anAbstractAnalyzerInverseDiscrete". Methods which use linear estimation, smoothed linear estimation, or nonlinear procedures can additionally be provided with a weighting procedure. Since not all weighting procedures are appropriate for each of these methods, user interface will take care for reasonable combinations of weighting algorithms and inverse analyzers. Inverse linear algorithm will be provided with a reference to a regularizer. Scanning algorithms will be derived from the abstract class "anAbstractAnalyzerInverseDiscreteScan". Dipole fit methods using a discrete search space are derived from "anAbstractAnalyzerInverseDiscrete". The complete class structure is shown in fig. 3.3. The data flow is similar to the data flow for dipole fit methods for the continuous parameter space, which is shown in fig. 3.7. The class "anAnalyzerDiscreteDipoleFit" has a reference to the class "anAbstractOptimizerDiscrete". This class is derived from a general optimizer class "anAbstractOptimizer", which has an abstract goal function embedded. For the special classes of goal functions used for source modeling of EEG/MEG measurements a further abstract class interface is introduced. It has references to a simulator and reference data. To avoid a mismatch with the simulator and the reference data embedded in the "anAbstractAnalyzerInverseDiscrete" class, the methods getSimulator(), setSimulator(), getReferencedata and setReferencedata() of the latter class will be overloaded by methods of the "anAnalyzerInverseDiscreteDipoleFit" class, which uses links to the "anAbstractGoalFunctionEEGMEG" class. The goal function for EEG/MEG measurements has a further reference to a class "anAbstractCiteria" which returns a similarity measure for example between simulated and measured data. The embedded search volume can be used to restrict the search space for parameters.

Class Diagram Inverse Toolbox: Dipole Fit Discrete Parameter Space

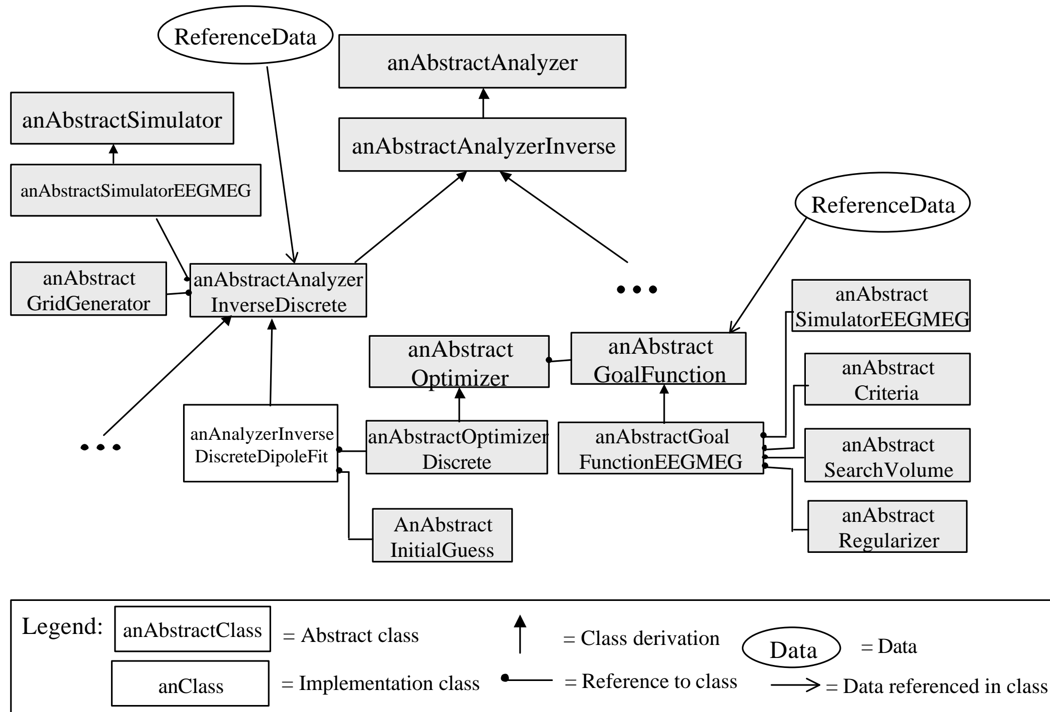


Fig. 3.3 Class structure for the implementation of dipole fit methods using a discrete search space for their parameters.

Flow Diagram Inverse Toolbox: Linear Estimation Methods

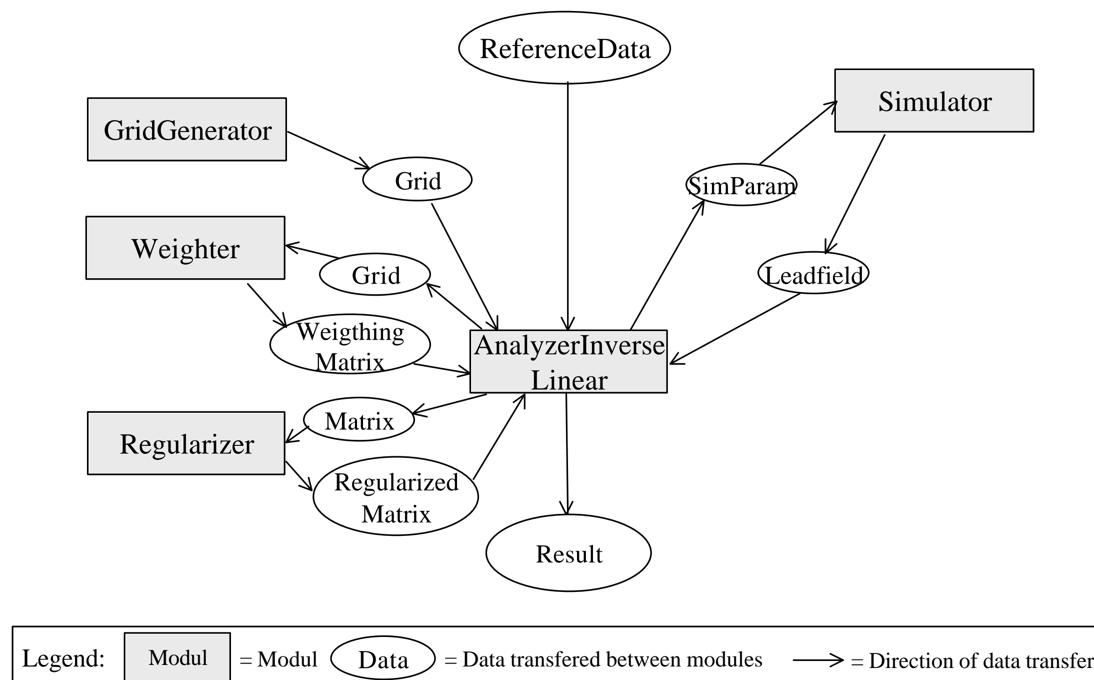


Fig 3.4 Flow diagram for source models with a discrete parameter space (linear estimation) giving an overview about the exchange of data between modules.

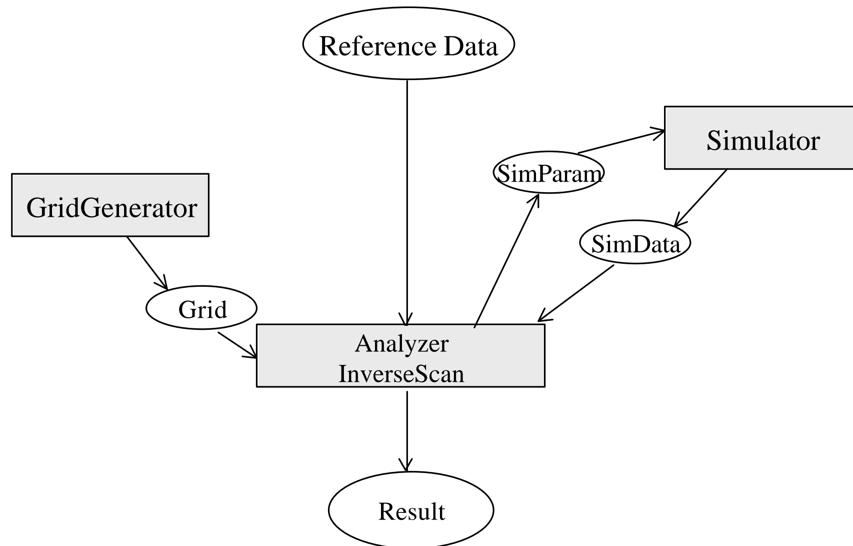


Fig 3.5 Flow diagram for source models with a discrete parameter space (scanning methods) giving an overview about the exchange of data between modules.

The abstract class for the all methods using a continuos parameter space "anAbstractAnalyzerInverseContinuous" (Fig. 3.6, Fig 3.7) has a reference to the abstract class interface for initial guesses, which provides initial parameters. On the other hand it has a reference to the abstract class interface for all kind of continuous optimization procedures, where it sends the initial parameters and receives the optimal parameters. The class definition "anAbstractOptimizerContinuous" is derived from a general class "anAbstractOptimizer". Optimization procedures need a goal function. As it is described for the dipole fit using a discrete search space, the abstract optimizer class has a reference to a goal function. All kinds of goal functions are represented by the class interface "anAbstractGoalFunction". Parameters are send to the goal function and it returns a goodness of fit value. For the special classes of goal functions used for source modeling of EEG/MEG measurements a further abstract class interface is introduced. It has references to the simulator, reference data and criteria, which return a similarity measure for example between simulated and measured data. To obtain the complete goodness of fit it has a reference to an abstract class interface, which defines the search volume. From the abstract class "anAbstractGoalFunctionEEGMEG" the classes with implementations for the different kinds of dipoles, e.g. for example rotating dipoles, are derived.

Since source reconstruction procedures are time consuming and a user may wish to watch the status of the process and even may wish to stop the algorithm and restart it with different parameters an "anAbstrcatStatus" class allows to update the status and inquire the status of a method.

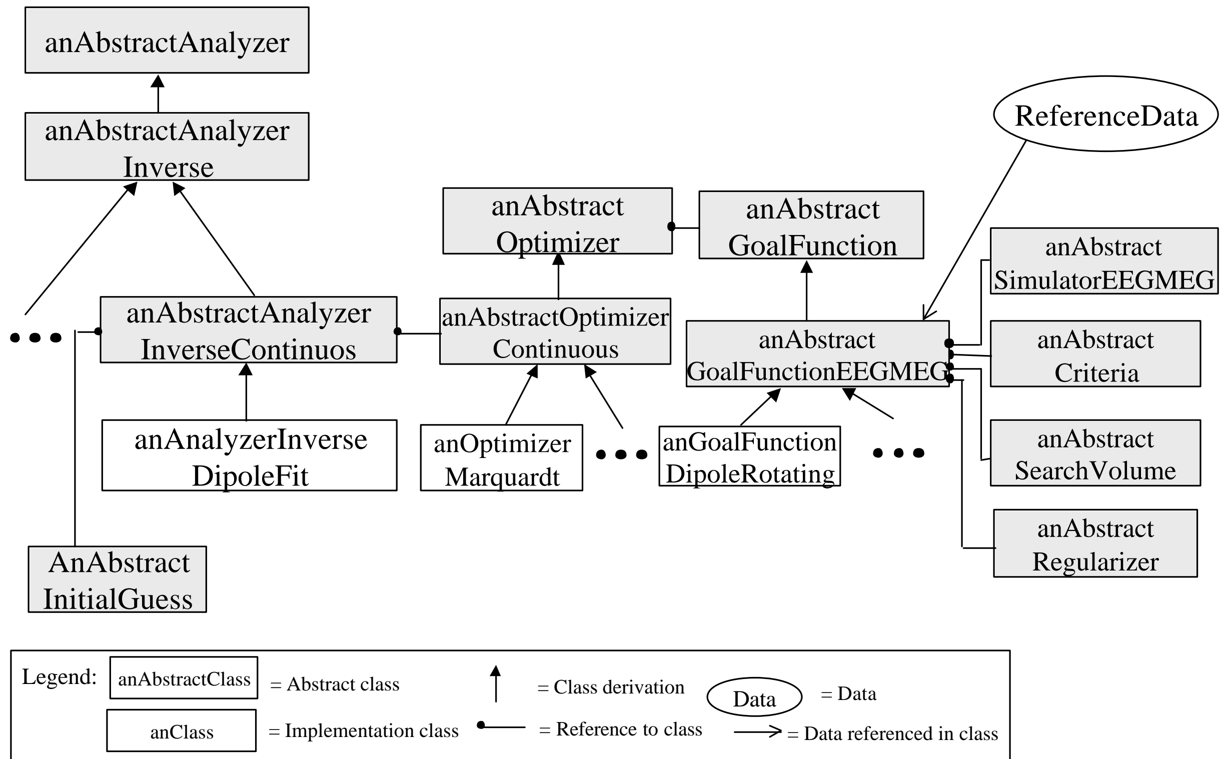


Fig 3.6 Class structure for the implementation of methods using a continuous search space for their parameters.

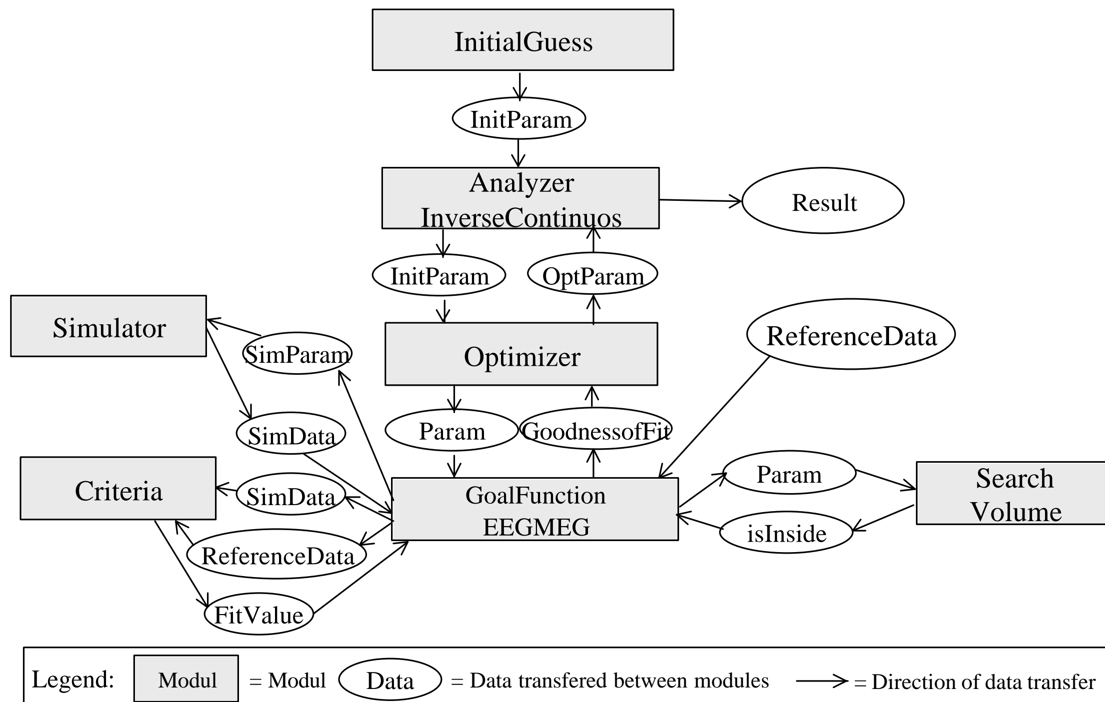


Fig 3.7 Flow diagram for source models with a continuous parameter search space giving an overview about the exchange of data between modules.

3.3 Access to methods of the inverse toolbox

The inverse toolbox provides a wide variety of combinations of methods. To give access to methods incorporated in the inverse toolbox a multi layer interface will be implemented. These user interfaces provide user scenarios. A user scenario consists of a reasonable combinations of methods and contains a complete source modeling procedure. When new methods are added to the inverse toolbox, this set of combinations can be augmented.

.

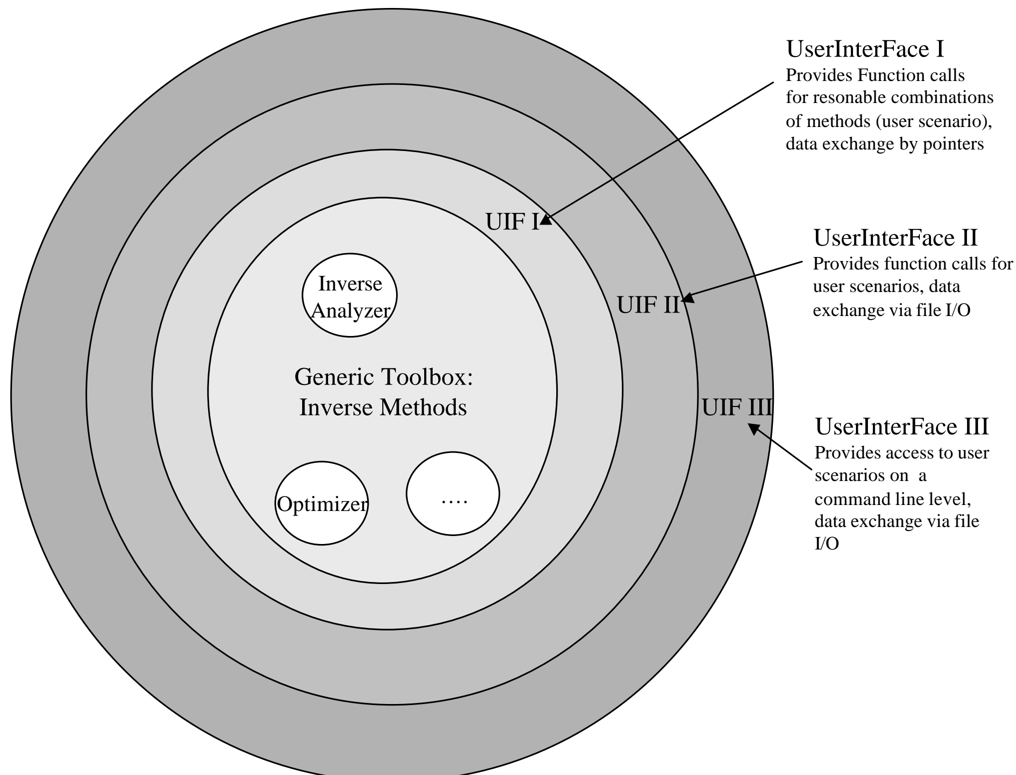


Fig.3.6 Three shell user interface

1. In the center is the generic inverse toolbox providing a variety of inverse methods.
2. The first shell provides a set of user scenarios to give access to the methods of the toolbox (UIF I). Since not all combinations of the methods inside the toolbox are reasonable, reasonable combinations are defined on this level, as well as for the other levels.
3. The next level (UIF II) gives acces to the user scenarios with additional file input/output facilities.
4. The outer level (UIF III) will allow to contsruct a user scenario on a command line level.

The three user interfaces are isomorph. A method or command of the two outer levels consist besides of functions specific to its level of just one call of a method of the next deeper layer. The three user interfaces can be easily extended, as it is described in chapter 3.6.

The user interfaces will be implemented on a Unix (Linux) system using the Gnu C++-Compiler. Testing of the user interfaces will be done by comparing results to reference solutions (see chapter 5.2) and by displaying results with the SimBio visualisation module. Visualization of results allows for example the checking of reasonable source positions and magnitudes.

3.4 User interface I

User interface I shall give the opportunity to use the methods of the inverse toolbox in a complete application, which allows data preselection, viewing capabilities, user interaction, etc.. User interface I is implemented as a class. For each general approach of source modeling (linear estimation, non-linear estimation, goal function scan, MUSIC, moving dipoles, rotating dipoles, fixed dipoles) separate methods are available. For each method with a discrete parameter space there are in addition methods for three different discrete search spaces: on cortex, on brain surface, in brain volume. Thus the class contains the following methods:

Linear Estimation:

uif1_linear_estimation_oncortex(parameter_1, parameter_2, …, parameter_n)
uif1_linear_estimation_onbrainsurface(parameter_1, parameter_2, …, parameter_n)
uif1_linear_estimation_inbrainvolume(parameter_1, parameter_2, …, parameter_n)

Non-Linear Estimation:

uif1_non_linear_estimation_oncortex(parameter_1, parameter_2, …, parameter_n)
uif1_ non_linear_estimation_onbrainsurface(parameter_1, parameter_2, …, parameter_n)
uif1_ non_linear_estimation_inbrainvolume(parameter_1, parameter_2, …, parameter_n)

Goal Function Scan:

uif1_goalfunctionscan_oncortex(parameter_1, parameter_2, …, parameter_n)
uif1_goalfunctionscan_onbrainsurface (parameter_1, parameter_2, …, parameter_n)
uif1_goalfunctionscan_inbrainvolume(parameter_1, parameter_2, …, parameter_n)

MUSIC:

uif1_MUSIC_oncortex(parameter_1, parameter_2, …, parameter_n)
uif1_MUSIC_onbrainsurface(parameter_1, parameter_2, …, parameter_n)
uif1_MUSIC_inbrainvolume(parameter_1, parameter_2, …, parameter_n)

Discrete Dipole Scan:

uif1_discrete_dipole_scan_oncortex(parameter_1, parameter_2, …, parameter_n)
uif1_discrete_dipole_scan_onbrainsurface (parameter_1, parameter_2, …, parameter_n)
uif1_discrete_dipole_scan_inbrainvolume(parameter_1, parameter_2, …, parameter_n)

Dipole Fit:

uif1_movingdipolfit(parameter_1, parameter_2, …, parameter_n)
uif1_rotatingdipolfit(parameter_1, parameter_2, …, parameter_n)
uif1_fixeddipolfit(parameter_1, parameter_2, …, parameter_n)

Parameters that have to be set invoking a method can be divided into three groups:

1. Parameters, which supply the methods with measured data like MRI, measured EEG/MEG signals and measurement conditions like electrode positions and SQUID magnetometer descriptions.
2. The result of the source modeling.
3. Parameters, which are used as switches, to select between combinations of algorithms. For the implemented methods it is insured that only reasonable combinations of algorithms can be selected. Whenever possible, default parameters are set, which can be omitted, when invoking the method.

4. For dipole fit methods, the number of dipoles can be set as an additional parameter.

Table 3.1 gives an overview about possible combinations of algorithms, which can be composed on the level of user interface I. An exact parameter definition for the methods is given in the appendix part A.

	Forward – type	Linear-inverse type	Non-linear inv. type	Inverter-type (Regu-larizer)	Sensor-type	Optimizer	Criteria	Search-volume	Intial Guess
Linear estima-tion	BEM FEM	L2 Loreta Continuos-L2 Continuous-GradientL2	-	TSVD Tikhonow	EEG MEG	-	-	Cortex-grid Brain-surface-grid Brain-volume-grid	-
Non-Linear estima-tion	BEM FEM	-	L1 -	TSVD Tikhonow	EEG MEG	Conjugate Gradient	-	Cortex-grid Brain-surface-grid Brain-volume-grid	-
Goal functio n scan	BEM FEM	-	-	-	EEG MEG	-	-	Cortex-grid Brain-surface-grid Brain-volume-grid	-
MUSIC	BEM FEM	-	-	-	EEG MEG	-	-	Cortex-grid Brain-surface-grid Brain-volume-grid	-
Discret edipole-scan	Spheres BEM FEM	-	-	TSVD Tikhonow	EEG MEG	Simulated Annealing	Minumum Square Error Maximum Probablity Maximum Entropie	Entire Brain Influence nodes	Stand-ard Dis-crete
Dipole-fit	BEM FEM	-	-	TSVD Tikhonow	EEG MEG	Simplex Marquardt	Minumum Square Error Maximum Probablity Maximum Entropie	Entire Brain	Stand-ard

Table 3.1 Overview about combinations of algorithms, which are provided by user interfaces

3.5 User interface II

User interface II shall give the opportunity to use algorithms of the inverse toolbox in an environment, which is not closed and complete in a sense that, for example, viewing facilities are not integrated in

the application. Inside SimBio these facilities can be provided by applications of other workpackes. These applications are for example viewing applications and applications performing segmentation with subsequent FEM grid generation. User interface II provides methods which can read files generated by these applications and will produce output files that can be read by subsequent applications. Thus, user Interface II enables distributed processing on a file level, which is one general objective of the SimBio project.

Each Method of user interface II corresponds to one method of user interface I. File input and output operations are wrapped around the call of the corresponding method of user interface I. Parameters for these methods of user interface II are not classes or arrays in combination with switches but filenames in combination with switches.

Files that can be read by the methods of UIF II contain:

reference data	(Vista file format [21]),
sensor configuration	(Vista file format),
grids	(Vista file format),
leadfield matrix	(Vista file format).

Reference data are used for the comparison of measured data and predicted data by the estimated sources. The sensor configuration contains sensor positions and for MEG measurements a description of the gradiometer device. Grids are used for the definition of the search space, forward computations and for the generation of spatial weigthing algorithms.

The output of results of the inverse toolbox will be written into a file in Vista format. This may be dipole parameters or a scan space description and the respective scan metric values.

Names of methods of user interface II start with “uif2”. An exact description of the parameters of the methods of UIF II is available in the appendix part A.

3.6 User interface III:

The third user interface level provides access to the methods of the inverse toolbox on a command line level. Commands are invoked with a set of arguments, to specify the execution of the command. On UNIX computers a sequence of commands can easily be created by using a shell script. In addition these commands can be invoked by an integrated user interface, which will be provided by WP 6 of the SimBio project. User interface III provides the same set of inverse methods as the above described user interfaces, despite user interface III is restricted to scenarios which generate data objects from files. Commands of user interface III have identical properties to methods of the inner levels by calling a method of UIF II which in turn calls a method of UIF I.

Commands of user interface III look like:

uif3_inverse_method_run argument1 argument2 argument3

Arguments for commands define the type of source modeling, filenames of input and output files and switches, to select between combinations of algorithms. Again it is insured that only reasonable combinations of algorithms can be selected and if possible default parameters are set, when parameters are omitted. For dipole fit methods the number of dipoles can be set.

As for the preceding user interfaces exact definitions of the commands are given in part A of the appendix.

3.7 Adding user scenarios

Adding new methods to the inverse toolbox or the wish to create new combinations of already implemented methods, evoke the need of creating new user scenarios. This can be done adding new user scenarios to the existing user interfaces. This chapter shall give a brief guideline to implement a new scenario for the three interface levels.

User interface I is defined as a class. Thus in the corresponding include file a new method has to be added. To provide an example a part of the class definition can be found in the appendix part B. The method must have arguments specifying input and output parameters as for example reference data, sensor configurations and result data. Additional switches are possible to allow a greater variability of combinations of methods. Inside the implementation of the methods a constructor for all classes which are necessary for the determination of the source parameters have to be invoked. Then the run() method for the parameter determination can be called, followed by a call of getResult(), which delivers a matrix with the estimated source parameters.

User interface II is realized as a class. Methods of user interface II create data matrices and classes from files before calling the method of user interface I. The results can be written in a file subsequent to the call of the method of UIF I. An example is given inside appendix B.

User Interface III uses the argument argc (argument count) and the array of arguments argv (argument value) of the main function of a program. Thus the values of argv can be used to identify the name of the scenario and to define names of input and output files as well as switches. To ensure that the array argv contains reasonable values, their contents should be checked before calling a method of user interface II, like it is done in the example of appendix B.

3.8 Interaction of the S-Cauchy Fortran 77 code in the inverse toolbox

The basic language of the inverse toolbox is ANSI C++. This makes it possible to use dynamic memory allocation and to build a C++ class interface with all the advantages of object orientated software. In the past most finite element (FEM) software, like S-CAUCHY[1] software, was developed in FORTRAN 77. The S-CAUCHY software was well tested and has proven to be useful, so we want to integrate the FEM modules from this software in the new C++ class structure. The FORTRAN 77 code of these functions will be wrapped in a C++ interface, using shared data with the C++ functions. Some of the issues that need attention in the data exchange and the calling conventions are described in [19]. The development of the Fortran functions will use the GNU F77-Compiler and will be linked to the C++ classes.

The use of the S-CAUCHY FEM modules asks for special attention on the data arrangement in the C++ classes in order to avoid time consuming reallocation of data fields. A first step in the finite element method is the grid generator. The resulting finite element mesh represents the geometric and electric properties the human head. The relevant compartments are: skin, skull, liquor, brain and ventricular system. The FE method allows assigning an individual conductivity to each finite element of the head. Geddes and Baker [22] and Haueisen et al. [23] investigated these values. In our model we go a step further and we will assign a conductivity tensor to each finite element. A strong anisotropy is known for the skull and the brain (white and gray matter). The conductivity tensor in the white matter will be measured using the diffusion tensor imaging technique. The inverse toolbox can take into account anisotropy for each element. The FE-grid will be generated by ST1.2 and used as input data in the inverse toolbox. Inside the inverse toolbox, the grid generator class will be used to read the FE-grid (stored in VISTA or S-CAUCHY format) and to represent it in the grid structure which can be used by the C++ functions and by FORTRAN 77 functions. The existing FEM-functions are written in

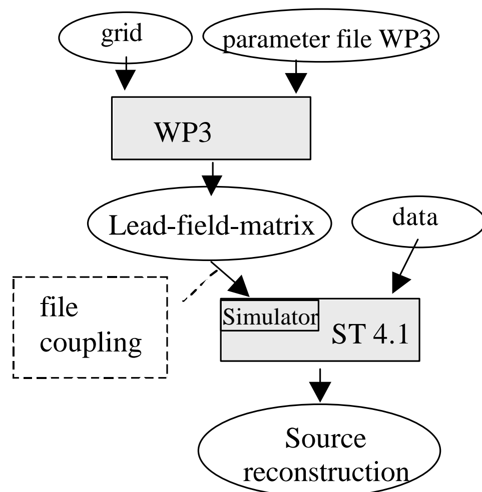
[1] S-CAUCHY is the SIMBIO version of the CAUCHY 1.8.4 (1997) software developed under co-ordination of Prof. Helmut Buchner at the RWTH Aachen
http://www.rwth-aachen.de/neurologie/Ww/Neurologie/cauchy/CauchyFunctionality.htm

FORTRAN 77, so we adapt the grid representation to the S-CAUCHY data structures and develop a compact representation of the grid structure. Details are given in the Annex.

3.9 Interaction with WP3

The call to the inverse toolbox can be divided in two cases using a previously created leadfield matrix or not. The leadfield matrix is the main computation job for inverse reconstruction in the discrete parameter space. This leadfield matrix can be computed on high performance computers used in WP3 and transferred via the Internet to the UIF shell of ST4.1, guided by WP6. The computation jobs of WP3 and ST4.1 can be executed on different computers. In this case the grid generator defines the influence nodes for the inverse computation. For all these influence nodes, WP3 calculates a source simulation (see Design Report WP3). This source simulation can be computed for all nodes in the three directions x, y, z or in the normal direction of the influence surface if the normal constraint is applied. This normal direction is provided as attribute of surface nodes by the grid generator computed as normal direction on the segmented image. Once the source simulation is computed for all nodes, it is saved in a binary VISTA file as a one-dimensional vector. This file containing the output vector will be read by a simulator class which is derived from the abstract simulator class. Thus, the results of computations of WP3 are available in the general framework of the inverse toolbox. This weak coupling between WP3 and ST 4.1 via files is illustrated on the left side of fig 3.7.

If the leadfield matrix is not provided, the UIF of the inverse toolbox can call a simulator class to compute the leadfield matrix. This needs a strong coupling between WP3 and ST4.1. All computation jobs will be executed on the same machine (parallel computer) and the inverse toolbox needs to communicate with WP3 by direct call of solver routines. This can not be handled on a file transfer level. In the continuous parameter space, the same strong coupling between WP3 and ST4.1 is necessary, because dipole parameters for each forward computation are only known after the respective optimization step.



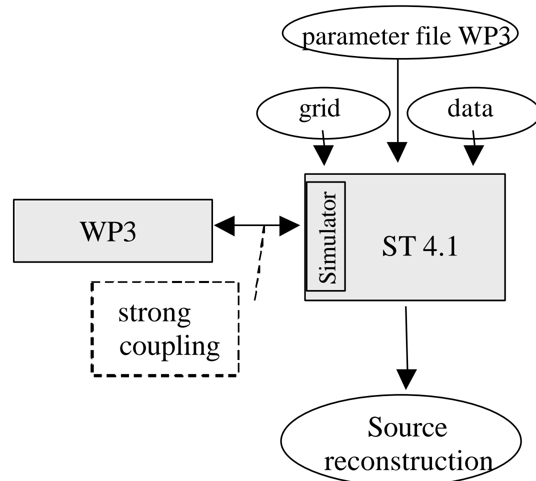


Fig. 3.7 Coupling between inverse toolbox of ST 4.1 and WP3

The strong coupling between the ST4.1 and WP3 will be implemented with an adapter class outside of the inverse toolbox. This makes it possible to compile and use the toolbox without the external module. The adapter class is derived from the abstract simulator class. After computation of the FEM stiffness matrix by WP3, the matrix will be stored in the adapter class of ST4.1. For each dipole position in the continuous space computed by the inverse algorithm, the solver of WP3 will compute a

fast forward solution. For this a pointer to the FEM stiffness matrix and the dipole position is passed to the solver routine. The solver returns the potential distribution on the FEM nodes. The parameter file for the solver will be passed to the UIF of ST4.1, which passes it to the fast solver. In this case Wp3 is called only by ST4.1 and not directly by WP6.

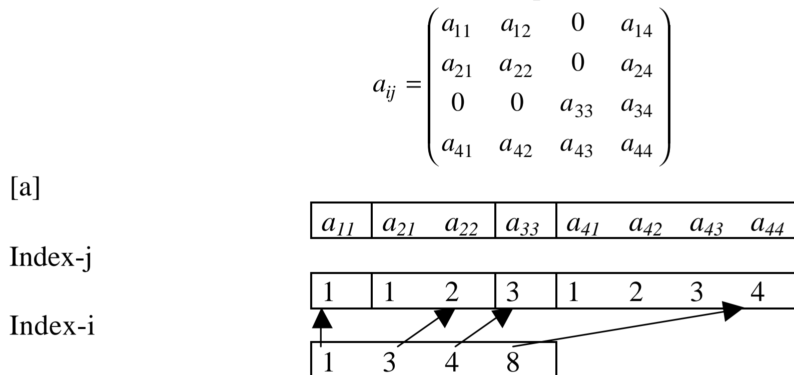
3.10 Forward simulator adapter class

An essential part of the inverse problem toolbox is the forward simulator. The fast solvers for the forward solution will be compiled in a library, which will be called from the forward simulator adapter class. The finite element method is used to simulate the potential values for a source configuration. The electromagnetic partial differential equation (PDE) is solved on the discrete finite element grid of the head. The solution of the PDE is approximated by piecewise continuous polynomials. For the FE numerical integration the function value is evaluated at 'support points' within an element. These support points are called 'Gaussian points'. The **'degree of integration'** defines the number of Gauss integration points for the element related numerical integration. A value of 2 means two integration points per edge; this yields to 8 Gaussian points for a brick element and 4 Gaussian points for a tetrahedron. A degree of integration of 1 is much quicker, but can lead to singular system matrix when the number of nodes exceeds the number of elements; this means a calculation error and is in most cases less exact. The FE integration split the PDE in a finite number of algebraic equations expressed in a system of linear equations:

$a_{ij}\, p_j + b_i = 0_i,$ with the **symmetric system matrix** $a_{ij}=a_{ji}$.

Without any boundary condition, the stiffness matrix (system matrix) is singular, and the equation can not be solved. A boundary condition is introduced with the node representing the reference electrode containing the value **zero**. The **'boundary condition'** can contain more than one (also nonzero) entry. The inverse toolbox includes **Dirichlet** and **Neumann** boundary conditions.

The system matrix contains almost only zero elements due to the small width of the FEM test function. A position in the matrix a_{ij} is only non-zero, if the node with the number i and the node with the number j lie in the same element. This sparseness must be used by the FEM solver to reduce the time and memory consumption for the solution. Only the elements in the matrix different zero from the left lower part of the symmetric positive definite matrix will be stored in a line-vector. Two other vectors will store the information about the line and column position as shown in the figure below.



Index-i stores the location of the diagonal element of row i. Since the matrix is symmetric, the rows only up to the diagonal element are stored, so that entries of row i lie on the left of the diagonal element. The elements are ordered in ascending column number in Index-j for each column. This is not really essential (except for the position of the diagonal element), but a nice (straightforward) way, that makes some things easier. The index information depends only on the topology of the grid and represents the neighborhood of the nodes. This packed format of the FE-matrix is near to the Compressed Sparse Row (CSR) format. The adapter class will store the FEM stiffness matrix, which is used in each call to the fast solver of the FEM system provided by WP3. The interface with modern fast parallel solvers from WP3 needs storage of the system matrix not in this symmetric format, but in a complete CSR format. The vector field will be constructed in a similar way to the one described in

the figure. This doubles the size of the exchanged files. A comparison of the speed of the different solvers for the source simulation can be found in Wolters [24].

3.11 Specification of graphical user interface

Commands defined on the level of user interface III can be invoked from a graphical user interface to provide an easy access to the methods of the inverse toolbox. The following paragraphs give a guideline for the development of a user interface, which can be a front end for the SimBio environment. The realization of a graphical user interface will not be an integral part of ST 4.1.

The purpose of the graphical user interface is to specify the inverse method. The selection of files (reference data, search space grid, head grid and sensor configuration) and checking of their existence should be done by the integrated SimBio environment without the need of interaction. Thus, there is no need for file selections dialogues in the graphical user interface.

The interface should have two levels to specify the method. On the first level a preselection of the method should be available to define main properties of the inverse method. The second level allows an refinement of the properties of the method. On this level default values are provided wherever possible.

Methods that can be selected on the first level are of four different categories. These categories and their subtypes are shown in table 3.2. together with the respective command line argument. On the first level one type of method has to be chosen exclusively.

Main Category	Subcategory	Command line argument
Linear Estimation	On Cortex	uif3_linear_estimation_oncortex
	On Brain Surface	uif3_linear_estimation_onbrainsurface
	In Brain Volume	uif3_linear_estimation_inbrainvolume
Non Linear Estimation	On Cortex	uif3_non_linear_estimation_oncortex
	On Brain Surface	uif3_non_linear_estimation_onbrainsurface
	In Brain Volume	uif3_non_linear_estimation_inbrainvolume
Goal Function Scan	On Cortex	uif3_goalfunctionscan_oncortex
	On Brain Surface	uif3_goalfunctionscan_onbrainsurface
	In Brain Volume	uif3_goalfunctionscan_inbrainvolume
Discrete Dipole Fit	On Cortex	uif3_discrete_dipole_scan_oncortex
	On Brain Surface	uif3_discrete_dipole_scan_onbrainsurface
	In Brain Volume	uif3_discrete_dipole_scan_inbrainvolume
MUSIC	On Cortex	uif3_MUSIC_oncortex
	On Brain Surface	uif3_MUSIC_onbrainsurface
	In Brain Volume	uif3_MUSIC_inbrainvolume
Dipole Fit	Moving Dipole	uif3_movingdipolfit
	Rotating Dipole	uif3_rotatingdipolfit
	Fixed Dipole	uif3_fixeddipolfit

Table 3.2 Methods accessible via the graphic user interface: Main category, subcategory, command line argument.

On the second level arguments allow further selections to specify the method. They have to be set exclusively. For each main category of methods these arguments are identical.

Categories of arguments and argument values for linear estimation methods are given in table 3.3

Linear Estimation	Command line argument
Head Model	**BEM**
	FEM
Inverse Type	**L2**
	Loreta
	ContinuousL2
	ContinuousGradientL2
Inverter Type	**TruncatedSVD**
	Tikhonow

Table 3.3 Categories and values for the refined definition of linear estimation methods (default values are printed in a bold font).

For non-linear estimation methods the arguments are given in table 3.4

Non Linear Estimation	Command line argument
Head Model	**BEM**
	FEM
Non-linear Inverse Type	**L1**
Inverter Type	**TruncatedSVD**
	Tikhonow

Table 3.4 Arguments for the non-linear estimation methods (default values are printed in a bold font).

For Scanning Methods only the type of forward model can be chosen (table 3.5).

Goal function scan/ MUSIC	Command line argument
Head Model	**BEM**
	FEM

Table 3.5 Categories and values for the refined definition of scanning methods (default values are printed in a bold font).

Discrete dipole scan methods need the number of dipoles (1…10). The following arguments further specify the method (table 3.6)

Discrete Dipole Scan	Command line argument
Head Model	Sphere
	BEM
	FEM
Inverter Type	**TruncatedSVD**
	Tikhonow
Optimizer Type	**Simulated Annealing**
Search Volume Type	**Influence nodes**

Table 3.6 Categories and possible values for the refined definition of discrete dipole scan methods (default values are printed in a bold font).

Continuous dipole fit methods have the number of dipoles as the second command line argument. Following arguments further specify the method (table 3.7)

Dipole Fit	Command line argument
Head Model	Sphere
	BEM
	FEM
Inverter Type	**TruncatedSVD**
	Tikhonow
Optimizer Type	**Simplex**
	Marquardt
Criteria Type	**MinimumSquareError**
	MaximumEntropy
	MaximumProbability
Search Volume Type	**InEntireBrain**
Intial Guess Type	**Standard**

Table 3.7 Categories and possible values for the refined definition of dipole fit methods (default values are printed in a bold font).

Subsequent to the selection of a method and their possible refined definition a button should be available to start the inverse calculations by sending a command according to user interface III with the argument values specified by the user. Following example presents a possible command that can be started by the graphical user interface:

uif3_inverse_method_run uif3_linear_estimation_onbrainsurface ReferenceDataFilename CortexGridFileName HeadGridFileName SensorConfigurationFileName ResultFilename FEM L2 TruncatedSVD

Figure 3.8 shows a possible realization of a user interface to define and start methods of the inverse toolbox.

Figure 3.8 shows a possible realization of a user interface to define and start a subsection of methods of the inverse toolbox.

4. Error estimation package

4.1 General Description

The objective of the error estimation package is to estimate the sensitivity of inverse current source reconstructions to errors, inaccuracies, and simplifications in the forward model in cooperation with WP 7 ST 7.1. The sensitivity of source reconstruction results due to errors in the estimation of tissue conductivity is an example of parameters to be investigated. This has for example implications for presurgical magnetic source imaging for tumor patients undergoing neurosurgery, since it is known that tumors are associated with changes in the tissue resistivity profile.

A starting point for sensitivity analysis will be a simple volume conductor. This will be a multiple layered sphere model with anisotropic but homogenous material behavior in every layer. In a second step realistic head models with individual conductivity tensor values will be used.

To obtain insight in the sensitivity of different inverse source reconstruction methods to changes in the forward model, different inverse methods can be used for inverse calculations. This is provided by the modularity of the inverse toolbox.

The accuracy of inverse reconstruction methods does not only depend on the model of forward calculations and the chosen inverse method but also on characteristics of the sources themselves. Characteristics of the sources which have effect on the accuracy of the inverse method are the number of sources, source positions, source orientations, and source magnitudes. Therefore these source parameters can be varied systematically inside the error estimation package.

The forward model parameters will be varied in the innermost layer of sensitivity analysis. The resulting sequence of parameter or model variations inside the error estimation package is shown in figure 4.1.

One iteration of the sequence concerning the variation of one parameter of the forward model consist of the following steps (fig. 4.2). Starting point is one set of source parameters. The forward problem is solved for this set of source parameters. The resulting potentials or magnetic fields at sensor positions serve in the next step as input values for an inverse analysis. The inverse procedure also needs a forward simulator. For this forward simulator one parameter the sensitivity of which is currently investigated is changed. The estimated sources by the inverse method are then compared with the given source configuration. Generally the simulators for the initial forward computation with and for the inverse algorithm are identical. In Addition, the framework of the error estimation package allows to use different simulators to compare results between different kinds of simulators.

For single dipoles one can compare distances of the positions of the dipoles (x, y, z), differences of angles between source directions, and differences of source magnitudes between original and reconstructed sources. Following similarity measures can be determined. For source models using a discrete parameter space: mean and maximum difference of the amplitude, mean and largest angle between moments, correlation between magnitudes of sources.

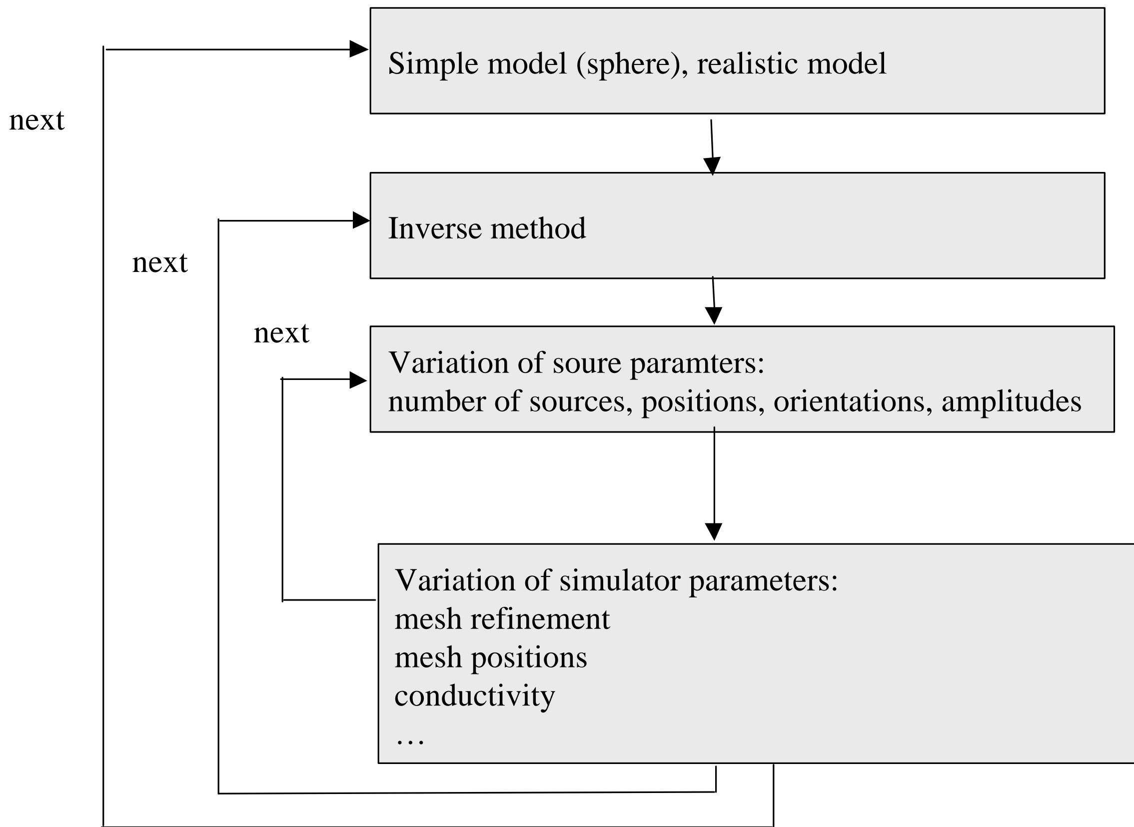


Fig. 4.1 Sequence for the variation of parameters or models in the error estimation package.

Flow DiagramError Estimation Package: Variation of Simulator Parameters

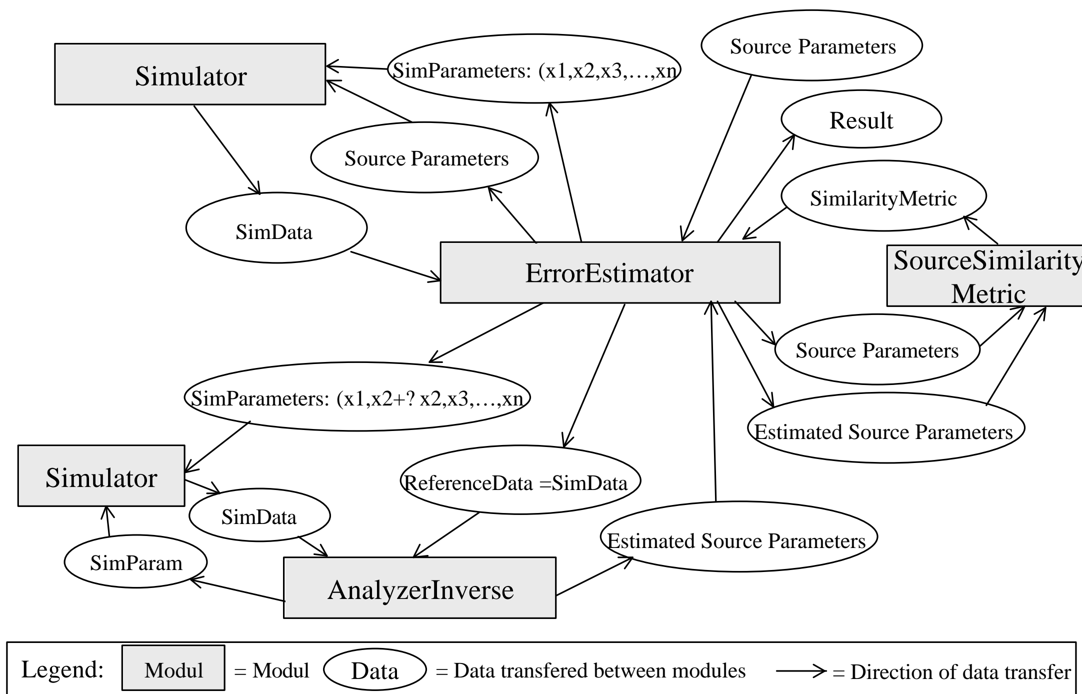


Fig 4.2 Flow Diagram for the sensitivity analysis regarding one parameter of the forward model (simulator)

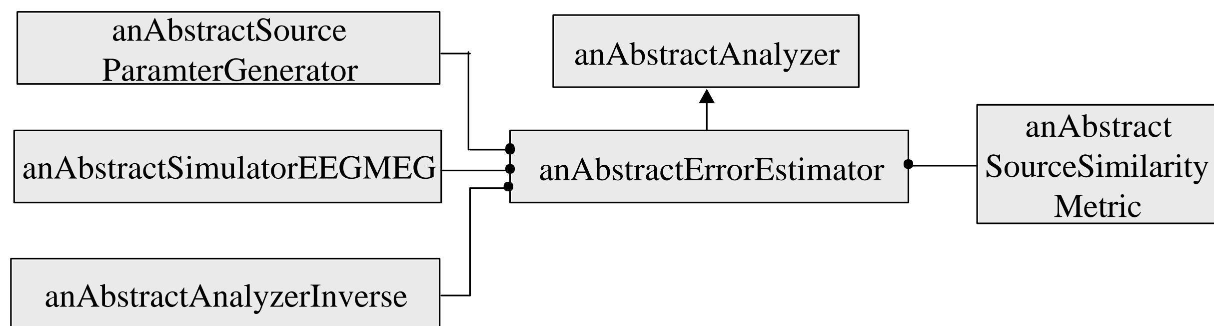
But also other combinations are possible. The original source can be for example a single dipole and the inverse method uses a linear estimation method. In this case one can compare position, orientation, and magnitude of the reconstructed source with the largest moment.
.

4.2 Class interface

To obtain a general framework for the analysis of sensitivities of source reconstructions due to changes in the forward model the modular class structure of the inverse toolbox can be extended. Fig. 4.3 shows the class interface of the error estimation package.

In the center of the class structure is the abstract class “anAbstractErrorEstimator” which defines general properties of error estimators. It is derived from the abstract class “anAbstractAnalyzer”. The error estimator class contains references to several other abstract classes. First there is a reference to the “anAbstractSourceParameterGenerator” class. This class provides a systematic variation of source parameters. It has a second reference to the class “anAbstractSimulatorEEGMEG”. This simulator is used for forward calculations. The “anAbstractErrorEstimator” class has a further reference to the “anAbstractAnalyzerInverse” class for the reconstruction of the sources by an inverse method. Using the abstract class interface all inverse methods which are realized in the inverse toolbox are available in the error estimation package. The simulator for the inverse calculations is used with one parameter varied for the sensitivity analysis. For this reason the parameters of the simulator can be requested and set using a general set of methods to access the simulator parameters.

Class Diagram Inverse Toolbox: Error Estimation Package



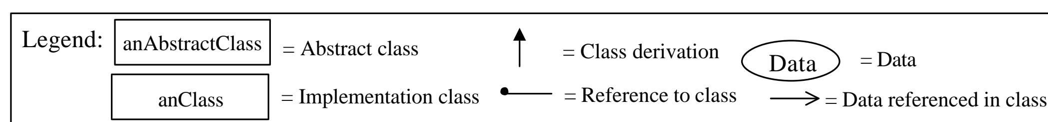


Fig. 4.3 Class diagram of the error estimation package.

There is one method within the simulator to return the number of variable forward parameters: getNumberofParameters(). Each parameter can be described by a name, a unit (e.g. mm), maximum and minimum values, default and current values. Table 4.1 shows the methods to get and set parameter values for single parameters. First one can get a name or a description of a parameter. The next method allows to get the units of the respective parameter. For the sensitivity analysis standard values, and minimum and maximum values for the deviation can be requested and set. Finally there are methods to set and get the current value of a parameter. Thus inside the implementation of error estimation classes one can request properties of parameters and change them systematically to determine the sensitivity of source reconstructions due to changes of parameters of the forward model.

Name	Return type	Description
ParameterNgetName(int ParameterNumber, string outName)	bool	Get name or description of a parameter
ParameterNgetUnit(int ParameterNumber, string outUnit)	bool	Get unit of a parameter
ParameterNsetMinimum(int ParameterNumber, double inMinimum)	bool	Set minimum value of a parameter. Minimum is used as lower deviation in sensitivity analysis.
ParameterNgetMinimum(int ParameterNumber, double outMinimum)	bool	Get minimum value of a parameter. Minimum is used as lower deviation in sensitivity analysis.
ParameterNsetMaximum(int ParameterNumber, double inMaximum)	bool	Set Maximum value of a parameter. Maximum is used as upper deviation in sensitivity analysis.
ParameterNgetMaximum(int ParameterNumber, double outMaximum)	bool	Get Maximum value of a parameter. Maximum is used as upper deviation in sensitivity analysis.
ParameterNsetDefault(int ParameterNumber, double inDefault)	bool	Set default/standard value of a parameter
ParameterNgetDefault(int ParameterNumber, double outDefault)	bool	Get default/standard value of a parameter
ParameterNsetValue(int ParameterNumber, double inValue)	bool	Set value of a parameter.
ParameterNgetValue(int ParameterNumber, double outValue)	bool	Get value of a parameter.

Table 4.1 Methods to request and set values of parameters of a forward simlator.

To compare the results between the original source parameters and the parameters of the reconstructed parameters sources the “anAbstractErrorEstimator” class has a reference to the “anAbstractSourceSimilarityMetric” class, which provides methods to compare source positions, directions and magnitudes. The method computesimilaritymeasure() provides besides a matrix with similarity values a description of the contents of the array.

To obtain a complete description of parameter variations and deviations of the reconstructed sources the getResult() method of the “anAbstractErrorEstimator” class provides following information:

1. Parameter name/description,
2. Parameter unit
3. Default/standard value of parameter
4. Changed simulator parameter for inverse analysis
5. Original source parameters

6. Reconstructed source parameters
7. Description of the values of similarity metric
8. Matrix with values of similarity metric

Thus, by abstract class definitions and modular design a huge number of parameters can be varied inside the error estimation package using one general approach. Additional descriptions of parameters and results, which are necessary for an interpretation and discussion of results, are incorporated in the definition of simulator classes and classes defining a similarity measure and can be extended by the user.

5. Software quality management

5.1 Software version control

To keep track of changes during software development and to allow distributed software development version control will be used during the generation of the inverse toolbox. New versions will be set up after implementing significant alterations and subsequent testing. Software which is not generated at A.N.T. Software b.v. will be merged to a common new version at A.N.T. Software b.v.. For each version a directory will be set up containing the implementation and a document describing the changes compared to the last version. Software versions will be placed on the network server at A.N.T. Software b.v. in the directory:

…\simbio\simbio_ipm_software_versions

The names of the directories for the individual versions will have the following naming convention including the date of their creation:

simbio_ipm_ver_yyyy_mm_dd.

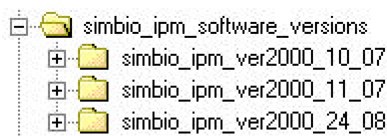


Fig. 5.1 Directory structure for different versions of the inverse toolbox software

On file level each change has to be documented in the file header by inserting a new line. This line should contain the date of change the person who performed the change and a description of the alterations inside the code.

```
// $2   23.08.2000  Alfred A.      added  index fields for FEM-node maps
// $1   11.07.2000  Matthias D.    created

#ifndef __anAbstractGridGenerator_c_H__
#define __anAbstractGridGenerator_c_H__
…
```

5.2 Software test protocols

Inside the inverse toolbox complex algorithms will be realized. Thus their implementation will be thoroughly tested and testing will be documented in a standardized way. The documentation will reside on the network server at A.N.T. Software b.v. in the directory:

…\simbio\softwarequalitymanagement\testprotocols

Test protocols will include following general information: date, person, version, compiler, tested method. Then the test procedure will be described, followed by their results, which can be numerical values, verbal descriptions or pictures. The details of a test procedure are defined in parallel to their implementation and the implementation will contain a reference to the test procedure document.

Methods of the inverse toolbox will be tested inside a complete test environment implemented for Microsoft Windows 95-98-NT. Besides data import, viewing and analysis features, this environment incorporates an open interface for adding new methods.:

Test procedures for methods inside the inverse toolbox can be divided into three different categories

1. Comparing the results of a method with a solution which can be derived analytically.
2. Real data can be used for methods which are implemented inside the ASA software packages of A.N.T. Software b.v. or the original Cauchy software . For these software packages reference results exist for specified data sets, which can be compared to results of methods implemented in the inverse toolbox. Once reference solutions are established for the inverse toolbox, they replace the reference solutions of the ASA package.

Initial testing of a method covers an extensive testing of their complete functionality. Whenever possible results are compared to analytically derived results. This testing has to be repeated each time changes are performed to this method.

If a new version of the inverse toolbox is created every method of the inverse toolbox available at the current stage of development will be covered by at least one test procedure. For example, testing of criteria and optimizer will be included in the dipole fit test. This test procedures will consist in the comparison of results to reference results.

Testing of methods may have some problems inherent to the used methods. If nonlinear search algorithms are used for the determination of source parameters they may not find the same optimal set of parameters as a reference solution, without being incorrect. In addition small differences to references solutions may be due to numerical inaccuracies or implementation details.

Generally it should be checked if results of methods are reasonable, for example if sources have plausible positions or magnitudes.

Table 5.1 contains a list how the methods of the inverse toolbox will be tested.

Method	Comparison with analytical solution	Comparison with reference data (criterion for comparison)
Dipoles with fixed positions for all time points	-	Distance to reference solution
Dipoles with rotating directions and fixed positions	-	Distance to reference solution
Moving dipoles	-	Maximum distance to

		reference solution
Criterion: Minimum square error	Computation for example matrices	-
Criterion: Maximum entropy	Computation for example matrices	-
Criterion: Maximum probability	Computation for example matrices	-
Parameter optimization: Simplex algorithm	Determination of parameters of non-linear functions	-
Parameter optimization: Levenberg Marquardt algorithm	Determination of parameters of non-linear functions	-
Parameter optimization: Simulated annealing	-	Maximum distance to reference solution
Loreta	-	Comparison to reference results (original "Loreta" software)
Truncated singular value decomposition	Computation for example matrices	-
Tikhonov-Philips-regularization	Computation for example matrices	-
Linear estimation	-	Correlation, relative maximum difference magnitudes, largest angle between moments
"Goal Function Scan" (GFS)	-	Correlation
"Multiple Signal Characterization" (MUSIC)	-	Correlation
Spheres	-	Included in dipole fit
BEM	-	Included in dipole fit, linear estimation, GFS and MUSIC test procedure
FEM	-	Correlation, relative maximum difference magnitudes, largest angle between moments

Table 5.1 List of test procedures of the methods inside the inverse toolbox.

5.3 Bug database

To have an overview about software bugs and unsolved problems a database will be used to keep information about bugs, their current status, and how they were probably solved. Thus every bug has to be registered. The database will consist of one table containing the fields shown in table 5.2.

Name	Type	Description
number	number	Assigned 4 digit bug number
status	text	Can be **bad** if unsolved or **ok** if the bug has been fixed
reported (date)	date	Date of the registration of this bug.
by whom	text	Person who registrates the bug
description	memo	Short description of the problem.
reproduction information	memo	Description how the bug can be reproduced. If necessary leave some sample data in a directory that starts with the bug number.

category	text	This can be **bug**, **deficiency**, or **wish.**
due	date	Date when bug has to be solved
priority	text	assign any of these: **very high**, **high, regular, low, very low**
solved (date)	date	Date when bug is solved.
solved by	text	Person who solved the bug
files	memo	Files changed to fix the bug
remarks	memo	Remarks concerning the bug or how it was solved

Table 5.2 Fields of bug database.

The bug database will reside on the network server at A.N.T. Software b.v. in the directory:

…\simbio\softwarequalitymanagement\bugdatabase

The database will be realized using the Microsoft Access 97 database. Bugs registered at other locations than A.N.T. Software B.V. will be merged to the database. If SimBio projects partners need data of the bug database, they will receive a current copy of the database or an exported sheet containing the information in a nonproprietary file format. In addition a bug report containing unsolved bugs will be sent to st41@simbio.de in HTML format once a week.

Fig. 5.2 Entry form for bug database.

5.4 Backup protocol

The software of the inverse toolbox will be written on a CD once a week at A.N.T. Software b.v.. This will include all documents concerning SimBio which are resident at A.N.T. Software b.v.. To have an overview about of the backups of software and documents they will be registered in a spreadsheet. The spreadsheet will reside on the network server at A.N.T. Software b.v. in the directory:

…\simbio\softwarequalitymanagement\backuphistory.

Fields of the spreadsheet are listed in table 5.3.

Name	Description
Date scheduled	Date, when the backup is scheduled
Directories	Directories, which have to be backed up
CD-ID	Identification of backup cd
Responsible	Person, who is responsible for the backup
Date done	Date, when the backup is done
Performed by	Person, who performed the backup

Table 5.3 Fields of the spreadsheet, which is used for the backup history.

References:

[1] Knoesche T.R., Solutions of the neuroelectromagnetic inverse problem- an evaluation study. Ph.D. thesis, University of Twente Enschede, The Netherlands, ISBN 9036509734, 1997

[2] Scherg M., Fundamentals of dipole source potential analysis, in: Auditory Evoked Magentic Fields and Electric Potentials, Ed. Grandori F et al., Karger, Basel, 40 -69

[3] Press W.H., Flannery B., Teukolsky S.A., Vetterling, W.T. Numerical Recipes, The Art of Sceintific Computing. Cambridge University Press, 1993

[4] Wagner M., Rekonstruktion neuronaler Stroeme aus bioelektrishen und biomagnetischen Messungen auf der aus MR-Bildern segmentierten Hirnrinde, Ph.D. thesis, University of Hamburg, Shaker Verlag, Aachen, ISBN 3-8265-4293-2, 1998

[5] Marquardt D. An algorithm for least squares estimation of nonlinear parameters. SIAM J. Appl. Math., 1963, 11:431-441

[6] Hämäläinen M.S., Illmomniemi R.J., Interpreting measured magnetic fields of the brain: Estimates of current distributions. Technical Report TKK-F-A559, Helsinki University of Technology, 1984

[7] Köhler T., Lösungen des biomagnetischen inversen Problems, PhD Thesis, University of Hamburg, Germany (1998).

[8] M. Fuchs H.-A., Wischmann M. Wagner, Generalized minimum norm least squares reconstruction algorithms, ISBET Newsletter No. 5 (1994): 8-1 1.

[9] Pascual-Marqui R.D., Low Resolution Brain Electromagnetic Tomography. Brain Topography, Vol. 7, 180

[10] Tikhonov A.N., Arsenim V.Y., Solutions of Ill-posed Problems. Wiley, New York 1977

[11] Buchner H., Knoll G., Fuchs M., Rienäcker A., Beckmann R., Wagner M., Silny J., Pesch J., Inverse localization of electric dipole current sources in finite element models of the human head. Electroenceph. Clin. Neurophysiol. 1997

[12] Rienäcker A., Buchner H., Knoll G., Comparison of Regularized Inverse EEG Analyses in Finite Element Models of the Individual Anatomy. Third International Hans Berger Congress 1996, accepted for publication 1997

[13] Mosher J.C., Lewis P.S., and Leahy R.M., Multiple dipole modeling and localization from spatio-temporal MEG data. IEEE Transactions on Biomedical Engineering, 1992, 39(6):541-557

[14] Gerson J., V. A. Cardenas, and G. Fein (1994), Equivalent dipole parameter estimation using simulated annealing, Electroenceph. din. NeurophysioL 92:161-168.

[15] Haneishi H., Ohyama N., Sekihara K., Honda T., Multiple current dipole estimation by simulated annealing. IEEE Transactions on Biomedical Engineering Vol.41, No.11, pp.1004-1009 (1994.11)

[16] Wolters C.H., Beckmann R.F., Rienacker A., Buchner H., Comparing regularized and non-regularized nonlinear dipole fit methods: a study in a simulated sulcus structure, Brain Topogr 1999 Fall;12(1):3-18

[17] Zanow F. Realistically shaped models of the head and their application to EEG and MEG. Ph.D. thesis, University of Twente Enschede, The Netherlands, ISBN 9036509416, 1997

[18] Stroustrup B., The C++ Programming Language, Special Edition, Addison-Wesley, 2000

[19] Anwander A, Wolters C., Interfacing S-CAUCHY FORTRAN 77 functions with C++ in ST4.1 and WP3, internal report, 27-07-2000.

[20] SimBio Docments: http://www.ccrl-nece.technopark.gmd.de/simbio/reports.html

[21] Vista: http://www.cs.ubc.ca/nest/lci/vista/vista.html

[22] Geddes L., Baker L., The Specific Resistance of Biological Material, Med. Biol. Eng. 5:271 (1967)

[23] Haueisen J., Ramon C., Eiselt M., Brauer H. et al., Influence of tissue resistivities on neuromagnetic fields and electric potentials studied with a finite element model of the head. IEEE Trans. Biomed. Eng. 44(8), 727-735 (1997).

[24] Wolters C., Reitzinger S., Basermann A., Burkhardt S., Hartmann U., Kruggel F., Anwander A., Improved tissue modelling and fast solver methods for high resolution FE-modelling in EEG/MEG-source localization. Proc. of the 10th Int. Conf. of Biomagnetism; BIOMAG 2000, Helsinky, August 2000, submitted.

Appendix A: Commands and parameters for user interfaces

List of methods and parameters of user interface I

1. Inverse methods:

Linear Estimation:
uif1_linear_estimation_oncortex(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType, inInverseType, inInverterType)
uif1_linear_estimation_onbrainsurface(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType, inInverseType inInverterType)
uif1_linear_estimation_inbrainvolume(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType, inInverseType, inInverterType)
Non-Linear Estimation:
uif1_non_linear_estimation_oncortex(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType, inNonLinearInverseType, inInverterType)
uif1_non_linear_estimation_ onbrainsurface (inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType, inNonLinearInverseType, inInverterType)
uif1_non_linear_estimation_ inbrainvolume (inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType, inNonLinearInverseType, inInverterType)
Goal Function Scan:
uif1_goalfunctionscan_oncortex(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType)
uif1_goalfunctionscan_onbrainsurface(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType)
uif1_goalfunctionscan_inbrainvolume(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType)
Music:
uif1_MUSIC_oncortex(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType)
uif1_MUSIC_onbrainsurface(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType);
uif1_MUSIC_inbrainvolume(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, inForwardType)
Discrete Dipole Fit:
uif1_discrete_dipole_fit_oncortex(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, nDipoles, inForwardType)
uif1_discrete_dipole_fit_ onbrainsurface (inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, nDipoles, inForwardType)
uif1_discrete_dipole_fit_ inbrainvolume (inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, nDipoles, inForwardType)
Dipole Fit:
uif1_movingdipolfit(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, nDipoles, inForwardType, inOptimizerType, inCriteriaType, inInverterType inSearchVolumeType, inIntialGuessType)
uif1_rotatingdipolfit(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, nDipoles, inForwardType, inOptimizerType, inCriteriaType, inInverterType inSearchVolumeType, inIntialGuessType)
uif1_fixeddipolfit(inReferenceData, inMRI, inSensorConfiguration, inSensorType, outResult, nDipoles, inForwardType, inOptimizerType, inCriteriaType, inInverterType inSearchVolumeType, inIntialGuessType)

All methods return a boolean value, indicating either a successful execution or a failure.

2. Parameters I

Parameter	Type	Description
inReferenceData	utMatrix_t<double>	Matrix containing the reference data
inMRI	MRItype	class containing the mri
inSensorConfiguration	SensorConfiguration	class containing the description of the sensor configuration
inSensorType	sensortype_e {sensortype_EEG, sensortype_MEG}	type of sensors: EEG, MEG
outResult	utMatrix_t<double>	Matrix containing the result
nDipoles	int	Number of dipoles for dipole fit

3. Parameters II (switches)

Parameter	**Possible Values**	**Default**
inForwardType	forwardtype_Sphere forwardtype_BEM forwardtype_FEM	forwardtype_BEM
inInverseType	linearinversetype_L2 linearinversetype_Loreta linearinversetype_ContinuousL2 linearinversetype_ContinuousL2-Gradient	linearinversetype_L2
inNonLinearInverse-Type	nonlinearinversetype_L1	nonlinearinversetype_L1
inInverterType	invertertype_Tikhonow, invertertype_TruncatedSVD	invertertype_TruncatedSVD
inOptimizerType	optimizertype_SimplexOptimizer optimizertype_MarquardtOptimizer optimizertype_SimulatedAnnealing	optimizertype_Marquardt Optimizer
inCriteriaType	criteriatype_MinimumSquareError criteriatype_MaximumEntropy criteriatype_MaximumProbability	criteriatype_MinimumSquare Error
inSearchVolumeType	searchvolumetype_InEntireBrain	searchvolumetype_InEntireBrain
inIntialGuessType	intialguesstype_Standard	intialguesstype_Standard

List of methods and parameters of user interface II

1. Inverse Methods:

Linear Estimation:
uif2_linear_estimation_oncortex(inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType, inInverseType, inInverterType)
uif2_linear_estimation_onbrainsurface(inReferenceDataFileName, inBrainSurfaceGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType, inInverseType, inInverterType)
uif2_linear_estimation_inbrainvolume(inReferenceDataFileName, inBrainVolumeGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType, inInverseType, inInverterType)

Non-Linear Estimation:
uif2_non_linear_estimation_oncortex(inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType inNonLinearInverseType, inInverterType)
uif2_non_linear_estimation_onbrainsurface(inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType, inNonLinearInverseType, inInverterType)
uif2_non_linear_estimation_ inbrainvolume (inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType, inNonLinearInverseType, inInverterType)
Goal Function Scan:
uif2_goalfunctionscan_oncortex(inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType)
uif2_goalfunctionscan_onbrainsurface(inReferenceDataFileName, inBrainSurfaceGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType)
uif2_goalfunctionscan_inbrainvolume(inReferenceDataFileName, inBrainVolumeGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType)
Music:
uif2_MUSIC_oncortex (inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType)
uif2_MUSIC_onbrainsurface(inReferenceDataFileName, inBrainSurfaceGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType)
uif2_MUSIC_inbrainvolume(inReferenceDataFileName, inBrainVolumeGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, inForwardType)
Discrete Dipole Fit:
uif2_discrete_dipole_fit_oncortex(inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, nDipoles, inForwardType)
uif2_discrete_dipole_fit_onbrainsurface(inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, nDipoles, inForwardType)
uif2_discrete_dipole_fit_ inbrainvolume(inReferenceDataFileName, inCortexGridFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, nDipoles, inForwardType)
Dipole Fit:
uif2_movingdipolfit(inReferenceDataFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, nDipoles, inForwardType, inOptimizerType, inCriteriaType, inInverterType, inSearchVolumeType, inIntialGuessType)
uif2_rotatingdipolfit(inReferenceDataFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, nDipoles, inForwardType, inOptimizerType, inCriteriaType, inInverterType, inSearchVolumeType, inIntialGuessType)
uif2_fixeddipolfit(inReferenceDataFileName, inHeadGridFileName, inSensorConfigurationFileName, outResultFilename, nDipoles, inForwardType, inOptimizerType, inCriteriaType, inInverterType, inSearchVolumeType, inIntialGuessType);

Additional methods for file I/O

uif2_read_ReferenceData(const string inReferenceDataFileName, utMatrix_t<double> inReferenceData)
uif2_read_SensorConfiguration(const string inSensorConfigurationFileName, SensorConfiguration inSensorConfiguration, sensortype_e inSensorType)
uif2_write_ResultData(const string outResultFilename, utMatrix_t<double> outResult)

2. Parameters I

Parameter	Type	Description	File Format
inReferenceDataFileName	string	Filename of file containing the reference data	Vista
inCortexGridFileName	string	Filename of file containing the description of the cortex grid	Vista
inBrainSurfaceGridFileName	string	Filename of file containing the description of the brain surface grid	Vista
inBrainVolumeGridFileName	string	Filename of file containing the description of the brain volume grid	Vista
inHeadGridFileName	string	Filename of file containing the description of the head grid (needed for simulator)	Vista
inSensorConfigurationFileName	string	Filename of file containing the reference data	Vista
outResultFilename	string	Filename of file containing the results	Vista

3. Parameters II (switches)

Parameter	**Possible Values**	**Default**
inForwardType	forwardtype_Sphere forwardtype_BEM forwardtype_FEM	forwardtype_BEM
inInverseType	linearinversetype_L2 linearinversetype_Loreta linearinversetype_ContinuousL2 linearinversetype_ContinuousL2-Gradient	linearinversetype_L2
inNonLinearInverse-Type	nonlinearinversetype_L1	nonlinearinversetype_L1
inInverterType	invertertype_Tikhonow, invertertype_TruncatedSVD	invertertype_TruncatedSVD
inOptimizerType	optimizertype_SimplexOptimizer optimizertype_MarquardtOptimizer optimizertype_SimulatedAnnealing	optimizertype_Marquardt Optimizer
inCriteriaType	criteriatype_MinimumSquareError criteriatype_MaximumEntropy criteriatype_MaximumProbability	criteriatype_MinimumSquare Error
inSearchVolumeType	searchvolumetype_InEntireBrain	searchvolumetype_InEntireBrain
inIntialGuessType	intialguesstype_Standard	intialguesstype_Standard

List of command arguments of user interface III

1. Inverse Methods:

Linear Estimation:
uif3_linear_estimation_oncortex_ with_leadfield inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType inInverseType inInverterType

uif3_linear_estimation_oncortex inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType inInverseType inInverterType
uif3_linear_estimation_onbrainsurface _with_leadfield inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType inInverseType inInverterType
uif3_linear_estimation_onbrainsurface inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType inInverseType inInverterType
uif3_linear_estimation_inbrainvolume _with_leadfield inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType inInverseType inInverterType
uif3_linear_estimation_inbrainvolume inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType inInverseType inInverterType
Non-Linear Estimation:
uif3_non_linear_estimation_oncortex_with_leadfield inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType inNonLinearInverseType inInverterType
uif3_non_linear_estimation_oncortex inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType inNonLinearInverseType inInverterType
uif3_non_linear_estimation_onbrainsurface_with_leadfield inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType inNonLinearInverseType inInverterType
uif3_non_linear_estimation_onbrainsurface inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType inNonLinearInverseType inInverterType
uif3_non_linear_estimation_inbrainvolume_with_leadfield inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType inNonLinearInverseType inInverterType
uif3_non_linear_estimation_onbrainsurface inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType inNonLinearInverseType inInverterType
Goal Function Scan:
uif3_goalfunctionscan_oncortex _with_leadfield inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType
uif3_goalfunctionscan_oncortex inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType
uif3_goalfunctionscan_onbrainsurface _with_leadfield inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType
uif3_goalfunctionscan_onbrainsurface inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType
uif3_goalfunctionscan_inbrainvolume _with_leadfield inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType
uif3_goalfunctionscan_inbrainvolume inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType
Music:
uif3_MUSIC_oncortex_with_leadfield inReferenceDataFileName inCortexGridFileName

inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType
uif3_MUSIC_oncortex inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType
uif3_MUSIC_onbrainsurface_with_leadfield inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType
uif3_MUSIC_onbrainsurface inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType
uif3_MUSIC_inbrainvolume_with_leadfield inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename inForwardType
uif3_MUSIC_inbrainvolume inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename inForwardType
Discrete Dipole Fit:
uif3_discrete_dipole_fit_oncortex_with_leadfield inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename nDipoles inForwardType
uif3_discrete_dipole_fit_oncortex inReferenceDataFileName inCortexGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename nDipoles inForwardType
uif3_discrete_dipole_fit _onbrainsurface_with_leadfield inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename nDipoles inForwardType
uif3_discrete_dipole_fit _onbrainsurface inReferenceDataFileName inBrainSurfaceGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename nDipoles inForwardType
uif3_discrete_dipole_fit_inbrainvolume_with_leadfield inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName inLeadFieldFileName outResultFilename nDipoles inForwardType
uif3_discrete_dipole_fit_inbrainvolume inReferenceDataFileName inBrainVolumeGridFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename nDipoles inForwardType
Dipole Fit:
uif3_movingdipolfit inReferenceDataFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename nDipoles inForwardType inOptimizerType inCriteriaType inInverterType inSearchVolumeType inIntialGuessType
uif3_rotatingdipolfit inReferenceDataFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename nDipoles inForwardType inOptimizerType inCriteriaType inInverterType inSearchVolumeType inIntialGuessType
uif3_fixeddipolfit inReferenceDataFileName inHeadGridFileName inSensorConfigurationFileName outResultFilename nDipoles inForwardType inOptimizerType inCriteriaType inInverterType inSearchVolumeType inIntialGuessType

2. Values for arguments used as switches

Parameter	Possible Values	Default
inForwardType	Sphere BEM FEM	BEM
inInverseType	L2 Loreta ContinuousL2	L2

	ContinuousL2Gradient	
inNonLinearInverseType	L1	L1
inInverterType	Tikhonow TruncatedSVD	TruncatedSVD
inOptimizerType	Simplex Marquardt SimulatedAnnealing	Marquardt
inCriteriaType	MinimumSquareError MaximumEntropy MaximumProbability	MinimumSquareError
inSearchVolumeType	InEntireBrain InInfluenceNodes	InEntireBrain
inIntialGuessType	Standard	Standard

Appendix B: Examples for the creation of new user scenarios

Implementation example of user interface I:

Class definition

```
// UIF1.h: interface for the UIF1 class.
//
……..
//
// A great variety of further options can be chosen, by setting parameters,
// when calling a method (see definition of enumarators).
//////////////////////////////////////////////////////////////////////


#include <utilities/include/utvector_t.h>
#include <utilities/include/utmatrix_t.h>
#include <utilities/include/utblock_t.h>
///// enumerators are used for switches to determine details of the methods

enum forwardtype_e {forwardtype_Sphere, forwardtype_BEM,forwardtype_FEM};
enum linearinversetype_e {linearinversetype_L2, linearinversetype_Loreta};
enum invertertype_e {invertertype_Tikhonow, invertertype_TruncatedSVD};
enum sensortype_e {sensortype_EEG, sensortype_MEG};
enum optimizertype_e {optimizertype_SimplexOptimizer, optimizertype_MarquardtOptimizer,
optimizertype_SimulatedAnnealing};
enum criteriatype_e {criteriatype_MinimumSquareError, criteriatype_MaximumEntropie,
criteriatype_MaximumProbability};
enum searchvolumetype_e {searchvolumetype_InEntireBrain};
enum intialguesstype_e {intialguesstype_Standard}

class UIF1
{
public:
        UIF1();
        virtual ~UIF1();

//  Discrete Parameter Space: UserFunctions for Linear Estimation

bool uif1_linear_estimation_oncortex
    (const utMatrix_t<double>& inReferenceData,              // Matrix containing reference (measured) data
     MRItype inMRI,                                          // Description of the segmented MRI
     SensorConfiguration inSensorConfiguration,              // Configurstion of the sensors (electrodes,
                                                             //  MEG-gradiometer)
     sensortype_e inSensorType,                              // Switch for the selecetion of the sensortype
     utMatrix_t<double> outResult,                           // Resultmatrix
     forwardtype_e inForwardType = forwardtype_BEM,          // Switch for the selecetion of forward model
     linearinversetype_e inInverseType = linearinversetype_L2, // Type of linear estimation,
                                                             // Default: L2-Norm
    invertertype_e inInverterType  = invertertype_TruncatedSVD); // Type of regularization,
                                                             //  Default: Truncated Singular
                                                             //  Value Decomposition
……}
```

Class implementation

```
//                                        Implementation of uif1 for linear estimation methods
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

bool UIF1::uif1_linear_estimation_oncortex
     (const utMatrix_t<double>& inReferenceData,      // Matrix containing reference (measured) data
      MRItype inMRI,                                  // Description of the segmented MRI
      SensorConfiguration inSensorConfiguration,      // Configurstion of the sensors (electrodes,
                                                      // MEG-gradiometer)
       sensortype_e inSensorType,                     // Switch for the selecetion of the sensortype
       const utMatrix_t<double> outResult,            // Resultmatrix of Linear Estimation
       forwardtype_e inForwardType,                   // Switch for the selecetion of forward model
       linearinversetype_e inInverseType,             // Type of linear estimation, Default: L2-Norm
       invertertype_e inInverterType)                 // Type of regularization,    Default: Truncated
                                                      //  Singular Value Decomposition

{

// Intialization of Grid on Cortex and for Head Model

if (!searchspacegrid_is_generated) anCorticalSurfaceGridGenerator_c  cortex_grid(inMRI);
if (!headgrid_is_generated)           anHeadModelGridGenerator_c         head_grid(inMRI);


// Selection of Type of forward model
anAbstractSimulatorEEGMEG_c                              *sim=NULL;

switch(inSensorType)
{
case sensortype_EEG:
            switch (inForwardType)
                      {
                         case forwardtype_Sphere:
                                        return false;

                          case forwardtype_BEM:
                                    sim= new
                                         anForwardSimulatorBEMEEG_c(inSensorConfiguration,head_grid);
                                        break;

                          case forwardtype_FEM:
                                        sim= new
                              anForwardSimulatorCauchyFEMEEG_c(inSensorConfiguration,head_grid);
                                        break;

                          default:
                                        return false;
                      }

    break;

case sensortype_MEG:
            switch (inForwardType)
                      {
                         case forwardtype_Sphere:
                                        return false ;

                          case forwardtype_BEM:
                                         sim= neanForwardSimulatorBEMMEG_c(inSensorConfiguration, head_grid);

                                        break;

                          case forwardtype_FEM:
                                        sim= new anForwardSimulatorCauchyFEMMEG_c(inSensorConfiguration,
                                                                                  head_grid);
                                        break;
```

```
                default:
                        return false;

                }
    break;
}

// Selection of Linear Inverse Type

 anAbstractWeighter_c                              *weigth=NULL;

switch (inInverseType)
{
        case linearinversetype_L2:
                weigth = new anL2weighter_c(sim);
                break;

        case linearinversetype_Loreta:
                weight = new anLoretaweighter_c(sim);
                break;
         default:
                delete sim;
                return false;


}

// Selection of Regularization
anAbstractRegularizer_c      *reg=NULL;

switch (inInverterType)
{
        case invertertype_Tikhonow:
                reg = new anRegularizerTikhonow_c;
                break;
        case invertertype_TruncatedSVD:
                reg = new anRegularizerTruncatedSVD_c;
                break;
        default:
                delete sim;
                delete weigth;
                return false;

}


//AnalyzerInverseLinear

anAnalyzerInversLinear_c          linest(inReferenceData, sim, cortex_grid, weight, reg);

linest.run();
linest.getResult(outResult);

delete cortex_grid;
delete head_grid;
delete sim;
delete weight;
delete reg;
return true;
}
```

Implementation example of user interface II:

Class definition

```
// UIF2.h: interface for the UIF2 class. Access to results probably generated on different computers via files
//
// UIF2 for the access of methodes from the inverse toolbx
……
//
// A great variety of further options can be chosen, by setting parameters,
// when calling a method (see definition of enumarators).
//////////////////////////////////////////////////////////////////////


#include <utilities/include/utvector_t.h>
#include <utilities/include/utmatrix_t.h>
#include <utilities/include/utblock_t.h>

#include <uif1_c.h>

#include <MRItype.h>
#include <SensorConfiguration.h>

///// enumerators are used for switches to determine details of the methods

enum forwardtype_e {forwardtype_Sphere, forwardtype_BEM,forwardtype_FEM};
enum linearinversetype_e {linearinversetype_L2, linearinversetype_Loreta};
enum invertertype_e {invertertype_Tikhonow, invertertype_TruncatedSVD};
enum sensortype_e {sensortype_EEG, sensortype_MEG};
enum optimizertype_e {optimizertype_SimplexOptimizer, optimizertype_MarquardtOptimizer,
optimizertype_SimulatedAnnealing};
enum criteriatype_e {criteriatype_MinimumSquareError, criteriatype_MaximumEntropie,
criteriatype_MaximumProbability};
enum searchvolumetype_e {searchvolumetype_InEntireBrain};
enum intialguesstype_e {intialguesstype_Standard}

class UIF2
{
public:
        UIF2();
        virtual ~UIF2();

//  Discrete Parameter Space: UserFunctions for Linear Estimation


   bool uif2_linear_estimation_oncortex
        (string inReferenceDataFileName,            // Inputfilename: File containing reference (measured) data
         string inCortexGridFileName,               //  Inputfilename: File with description of cortex grid
         string inHeadGridFileName,                 // Inputfilename: File with description of the head grid
         string inSensorConfigurationFileName,      // InputFile:Configurstion of the sensors (electrodes, MEG–
                                                    //  gradiometer)
         string outResultFilename,                  //  OutputFilenam: File with   Resultmatrix
         forwardtype_e inForwardType = forwardtype_BEM,            // Switch for the selecetion of
                                                                   // forward model
         linearinversetype_e inInverseType = linearinversetype_L2,     // Type of linear estimation,
                                                                       //   Default: L2-Norm
        invertertype_e inInverterType  = invertertype_TruncatedSVD);   // Type of regularization,    Default:
                                                          // Truncated Singular Value Decomposition

…}
```

Class implementation

```
//                                    Implementation of uif2 for linear estimation methods
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


bool UIF2::uif2_linear_estimation_oncortex
        (string inReferenceDataFileName,          // Inputfilename: File containing reference (measured) data
         string inCortexGridFileName,             // Inputfilename: File with description of cortex grid
         string inHeadGridFileName,               // Inputfilename: File with description of the head grid
         string inSensorConfigurationFileName,    // InputFile:Configurstion of the sensors (electrodes,
                                                  //MEG-gradiometer)
        string outResultFilename,                 // OutputFilenam: File with   Resultmatrix
        forwardtype_e inForwardType,              // Switch for the selecetion of forward model
        linearinversetype_e inInverseType,        // Type of linear estimation, Default: L2-Norm
        invertertype_e inInverterType);           // Type of regularization,    Default:
                                                  //Truncated Singular Value Decomposition

{

MRItype inMRI = NULL;   // MRI is not needed since grids are already available in files

// Intialization of Grid on Cortex and for Head Model

anCorticalSurfaceGridGeneratorfromFile_c   cortex_grid(inCortexGridFileName);
searchspacegrid_is_generated = true;
anHeadModelGridGeneratorfromFile_c         head_grid (inHeadGridFileName);
headgrid_is_generated = true;

// Read SensorConfiguration from File

uif2_read_SensorConfiguration(inSensorConfigurationFileName, inSensorConfiguration, inSensorType)

// Read Referencedata

uif2_read_ReferenceData(inReferenceDataFileName, inReferenceData);

UIF1::uif1_linear_estimation_oncortex(inReferenceData,        // Matrix containing reference (measured) data
                                      inMRI,                  // Description of the segmented MRI
                                      inSensorConfiguration,  // Configurstion of the sensors (electrodes,
                                                              //  MEG-gradiometer)
                                      inSensorType,           // Switch for the selecetion of the sensortype
                                      outResult,              // Resultmatrix of Linear Estimation
                                      inForwardType,          // Switch for the selecetion of forward model
                                      inInverseType,          // Type of linear estimation, Default: L2-Norm
                                      inInverterType)         // Type of regularization,    Default:
                                                              //Truncated Singular Value Decomposition



uif2_write_ResultData(outResultFilename, outResult);


}
```

Implementation example of user interface III:

```
int main(int argc,char* argv[])
{

forwardtype_e        inForwardType ;
linearinversetype_e  inInverseType ;
invertertype_e                    inInverterType;

bool correctparameter;

if (argc > 3)
{
 if ((argv[1]== "uif3_linear_estimation_oncortex") && (argc > 6))
            {
             // Start with default values, which are replaced if correct arguments are present
             inForwardType = forwardtype_BEM;
             inInverseType = linearinversetype_L2;
             inInverterType = invertertype_TruncatedSVD;

    if (argc > 7)
                                 {correctparameter = false;
                                  if (argv[7] == "Sphere") ;
                                  if (argv[7] == "BEM") {inForwardType = forwardtype_BEM; correctparameter = true;}
                                  if (argv[7] == "FEM") {inForwardType = forwardtype_BEM; correctparameter = true;}
                                  if (!correctparameter) std::cout << "wrong forward type, instead default is used"<<'\n';
    if (argc >8)
                                 {correctparameter = false;
                                  if (argv[8] == "L2")     {inInverseType = linearinversetype_L2; correctparameter = true;}
                                  if (argv[8] == "Loreta") {inInverseType = linearinversetype_Loreta; correctparameter =
                                                                                                          true;}
                                  if (!correctparameter) std::cout << "wrong inversetype, instead default is used"<<'\n';
                                 }

   correctparameter = false;
   if (argv[9] == "Tikhonow")     {inInverterType = invertertype_Tikhonow; correctparameter = true;}
   if (argv[9] == "TruncatedSVD") {inInverterType = invertertype_TruncatedSVD; correctparameter = true;}
   if (!correctparameter) std::cout << "wrong inverter type, instead default is used"<<'\n';

  UIF2::uif2_linear_estimation_oncortex(argv[2],       // Inputfilename: File containing reference (measured) data
                                        argv[3],       // Inputfilename: File with description of cortex grid
                                        argv[4],       // Inputfilename: File with description of the head grid
                                        argv[5],       // InputFile:Configurstion of the sensors (electrodes,
                                                       // MEG-gradiometer)
                                        argv[6],       // OutputFilenam: File with   Resultmatrix
                                        inForwardType,         // Switch for the selecetion of forward model
                                        inInverseType,          // Type of linear estimation, Default: L2-Norm
                                        inInverterType);

    }
….
}
```

Appendix C: Class definitions (Data description)

Grid definition : anabtsractgridgenerator.h

```
// $2    23.08.2000  Alfred Anwander added the index fields
// $1    11.07.2000  Matthias D.  created
#ifndef __anAbstractGridGenerator_c_H__
#define __anAbstractGridGenerator_c_H__

#include <utilities/include/utvector_t.h>
#include <utilities/include/utmatrix_t.h>
#include <utilities/include/utblock_t.h>

#include <analysis/include/analysisdef.h>



// class anAbstractGridGenerator_c serves as a base class for all kinds of grid generaotrs


enum comptartementype_e {oncortex, onbrainsurface, inbrainvolume, head};

class Grid_c
{
public:
   Grid_c();
   ~Grid_c();

   bool              isSurfaceGrid;                          // true, if surfacegrid
   bool              isIsomorphGrid;                         // true, if all elements are of the same type
   bool              is3_D_Grid;                             // true, if 3-D-grid
   comptartementype_e  Grid_Compartement;                    // Compartement
   int               n_GridNodes;                            // Number of Grid Nodes
   int               n_GridElements;                         // Number of Grid Elements

private:
   utMatrix_t<double>  GridNodePosition;                     // Position of grid node x,y,z : 3 Rows,
                                                             // n_GridNodes Columns

   utMatrix_t<double>  GridNodeSourceDirections;             // 3 rows: 1 Sourcedirections: vector1: x,y,z;
                                                             // 6 rows: 2 Sourcedirections: vector1: x,y,z;
                                                             //                              vector2: x,y,z
                                                             // 9 rows: 3 Sourcedirections: vector1: x,y,z;
                                                             //                  vector2: x,y,z; vector3: x,y,z
                                                             // n_GridNodes Columns

   utVector_t<int>     GridElementTyp;                      // type of the element : cube,... in cauchy code
                                                            // for a grid with only one element type, the lengh of
                                                            // this vector is 1
   utVector_t<int>     GridElementNumberNodes;              // Number of nodes per element
                                                            // for a grid with only one element type, the lengh of
                                                            // this vector is 1
   utVector_t<int>     GridElementNodes;                    // Node-Numbers for the element:
                                                            // nodes of the first element followed by the nodes of
                                                            // the second element...
   utVector_t<int>     GridElementNodesIndex;               // index of the nodes for the element

   utMatrix_t<double>  GridElementConductivity;              // Conductivity of the grid element : 1 or 6 Rows,
                                                             //  n_GridNodes Columns

   utVector_t<double>  GridNodeDirichletPotential;           // Boundary potential value of the volume nodes

   utVector_t<int>     GridNodeMapDiagonalIndexSymetric;     // Index of diagonal positions in the symetric
                                                             // FEM-node map

   utVector_t<int>     GridNodeMapIndexSymetric;             // Index of the positions in the symetric
                                                             // FEM-node map
```

```
   utVector_t<int>     GridNodeMapDiagonalIndexAsymetric;   // Index of diagonal positions in the FEM-node
                                                            // map ignoring symetrie

   utVector_t<int>     GridNodeMapIndexAsymetric;           // Index of the positions in the FEM-node map
                                                            //  ignoring symetrie
};

#endif // __anAbstractGridGenerator_c_H__
```

MRI definition: mritype.h

```
// $1 21.07.2000 Matthias D. created

// MRItype.h: interface for MRItype class.
//


#include <utilities/include/utvector_t.h>
#include <utilities/include/utmatrix_t.h>
#include <utilities/include/utblock_t.h>


enum orientation_e {axial, coronal, sagittal};           // Relation of image slices (the x-y plane) to the body plane
enum image_body_symmetry_e {natural, radiologic};   // Relation of image and body w.r.t. the body symmetry
                                                     //axis
                                                    // natural the left image side corresponds to the left body
                                                    // side,  radiologic: inverse


class MRI_Identification
{
public:
        MRI_Identification();
        ~MRI_Identification();
        string        SubjectName;            // Subject (Name of patient or volunteer)
        string        Study_Date_Time;        // Date and time of MRI Study
};



class MRI_Data_Description
{
public:
        MRI_Data_Description();
         ~MRI_Data_Description();
        int        MRIDimensions[3];      // Number of voxels: x-Direction,  y-Direction. Number of images:
                                          // z-Directon

        float      voxel[3];              // real world dimensions of a voxel in mm along the x-y-z axes
        orientation_e        MRIOrientation;        // Relation of image slices (the x-y plane) to the body plane
        image_body_symmetry_e  MRI_Image_Body_Symmetry;  // natural the left image side corresponds to
                                                          // the left body side, radiologic: inverse

};



class MRItype
{
public:
        MRItype();
        ~MRItype();

        bool set_MRI_Identification(MRI_Identification& inMRI_Identification);
        bool get_MRI_Identification(MRI_Identification& outMRI_Identification);

        bool set_MRI_Data_Description(MRI_Data_Description& inMRI_Data_Description);
        bool get_MRI_Data_Description(MRI_Data_Description& outMRI_Data_Description);
```

```
            bool set_MRI_Data_BLock(utBlock_t<unsigned short>&  inMRI_Data);
            bool get_MRI_Data_Block(utBlock_t<unsigned short>&  outMRI_Data);

            bool get_MRI_DATA_xy_plane(int planenumber, utMatrix_t<unsigned short>&  outMRI_xy_plane);
            bool get_MRI_DATA_xz_plane(int planenumber, utMatrix_t<unsigned short>&  outMRI_xz_plane);
            bool get_MRI_DATA_yz_plane(int planenumber, utMatrix_t<unsigned short>&  outMRI_yz_plane);


protected:
   MRI_Identification              m_MRI_Ident;     //  Identification of the MRI
   MRI_Data_Description            m_MRI_Desc;      //  Description of the MRI Data
   utBlock_t<unsigned short>       m_MRI_Data;      //  Block containing the values of the voxels

}
```

Sensorconfiguration: ansensorconfiguration_c.h

```
// ansensorconfiguration_c.h: interface for Sensorconfiguration class.
//


#include <utilities/include/utvector_t.h>
#include <utilities/include/utmatrix_t.h>
#include <utilities/include/utblock_t.h>


enum sensortype_e {sensortype_EEG, sensortype_MEG, sensortype_EEGMEG};

// Identification

struct SensorConfiguraton_Identification
            {
                        string          SubjectName;            // Subject (Name of patient or volunteer)
                        string          Study_Date_Time;            // Date and time of MRI Study
            };

// EEG

class EEG_Electrode_Description_c
            {
            public:
              EEG_Electrode_Description_c();
              ~EEG_Electrode_Description_c();

             string   Label;                                  // Channel Label (Fp1, Fp2,...)
             utVector_t<float>    ElectrodePosition;          // Electorde Positions x,y,z in mm
            };


// MEG

class CoilDescription_c
 {
public:
            CoilDescription_c();
            ~CoilDescription_c();
            int                       NumberCoils;
            utMatrix_t<float>     Pos;                        // (global) x,y,z-coordinates of coils
            utMatrix_t<float>     Dir;                        // (global) x,y,z-coordinates of coil directions
            utVector_t<float>     Sense;                      // number and sense of windings (seen in direction)
            utVector_t<float>     Area;                       // coil areas
            string                UnitArea;                   // unit of area (for exampel mm x mm)
};
```

```
class SensorConfiguration_c
{

public:
        SensorConfiguration_c();
        ~SensorConfiguration_c();


        SensorConfiguraton_Identification   SensSor_Ident;     //        Identification of SensorConfiguration

        sensortype_e            SensorType;             //   Type of sensor EEG, MEG or EEG+MEG
                                                        //    EEG

        int                                nEEGElectrodes;    //   Number of EEG Electrodes

        vector<EEG_Electrode_Description_c>   EEGElectrodes;     //   Electrode labels and positions

        string                             RefLabel;          //   Label of reference electrode

        // MEG

        int                     nMEGPositions;          //   Number of gradiometer postiions

        vector<string>          Labels;                 //   MEG Position Labels

        vector<CoilDescription_c>  MEGCoils;            //   Description of MEG Coils

        utMatrix_t<int>         Inpoints;               //   local (x,y) co-ordinates of each seven
                                                        //   integration points per coil

        utMatrix_t<double>      Weigths;                //   each seven values per coil enabling a
                                                        // weighted surface integration

        utMatrix_t<double>      Mat;                            //   magnetic matrix

        int                     NumberVertices;    //   number of nodes for the realistic model

        string                  MeasType;          //   Weber or Tesla

}
```