ModEM: A modular system for inversion of electromagnetic geophysical data

Anna Kelbert [a,*], Naser Meqbel [b,a], Gary D. Egbert [a], Kush Tandon [c,a]

[a] College of Earth, Ocean and Atmospheric Sciences, Oregon State University, 104 CEOAS Admin Bldg., Corvallis, OR 97331-5503, USA
[b] Helmholtz Centre Potsdam, GFZ German Research Centre for Geosciences, Potsdam, Germany
[c] Bluware Inc., Houston, TX 77063, USA

ARTICLE INFO

ABSTRACT

We describe implementation of a modular system of computer codes for inversion of electromagnetic geophysical data, referred to as ModEM. The system is constructed with a fine level of modular granularity, with basic components of the inversion – forward modeling, sensitivity computations, inversion search algorithms, model parametrization and regularization, data functionals – interchangeable, reusable and readily extensible. Modular sensitivity computations and generic interfaces to parallelized inversion algorithms provide a ready framework for rapid implementation of new applications or inversion algorithms. We illustrate the code's versatility and capabilities for code reuse through implementation of 3D magnetotelluric (MT) and controlled-source EM (CSEM) inversions, using essentially the same components.

1. Introduction

Egbert and Kelbert (2012) (hereinafter EK12) present a general discrete formulation for linearized electromagnetic (EM) inverse problems in the frequency domain, treating a range of EM geophysical techniques and inversion approaches in a unified notational framework. This framework highlights the elements, and dependencies between elements, that are common across applications and inversion algorithms, providing a basis for a general modular system of computer codes for EM geophysical inverse problems. EK12 provided a brief overview of such a system, which they referred to as ModEM: the Modular system for Electromagnetic inversion. Here we present a more detailed description of this software package, implemented in the Fortran 95 programming language.

Geophysical electromagnetic inversion approaches and parallel computer codes have developed rapidly in the past decade or so (notably, Newman and Alumbaugh, 2000; Zhdanov and Hursan, 2000; Haber et al., 2004; Newman and Boggs, 2004; Kelbert et al., 2008; Siripunvaraporn and Egbert, 2009; Avdeev and Avdeeva, 2009 and others; see also review papers Avdeev, 2005; Siripunvaraporn, 2011 and the overview in EK12). While most of these efforts focused on a specific EM inverse problem, solved with a particular computational approach, recent efforts in the context of joint inversion (e.g., Moorkamp et al., 2011; Commer and Newman, 2009; Stefano et al., 2011), have seen the development of more flexible inversion frameworks, which allow a single inversion algorithm to be applied to a range of different data types. Our focus here is on a finer granularity of modularization, with the goal of making the basic components of the inversion – forward modeling, sensitivity computation, inversion search algorithms, model parametrization and regularization, and data functionals – interchangeable, reusable and readily extensible. Thus, while ModEM is flexible enough to be applied to a range of EM data types, and indeed to joint inversion, it also provides a code base suitable for rapid development and prototyping of new parallel inversion algorithms, and for experimentation with different model parametrization and regularization schemes. To the extent possible (given limitations of Fortran 95; Akin, 2003) we have taken an object oriented approach to maximize code reuse, and to provide templates for rapid development of new applications. We also use the terminology of this approach in our discussion here.

To allow readers to more easily follow the general development, we provide a brief overview of the theory and notation of EK12 in

---

* Corresponding author.
 E-mail address: anna@coas.oregonstate.edu (A. Kelbert).

Section 2. However, as our focus is on program structure and logic, mathematical formulae required for implementation of specific routines are not repeated here; readers are referred to appropriate sections in EK12 for such details. ModEM can be roughly organized into three functional levels which we discuss in successive sections. Section 3 provides an overview of ModEM, and describes implementation of top level "generic" inversion and sensitivity routines, including a coarse grained general parallelization. These top-level routines, and the parallelization (discussed in Section 4) are intended to be directly reusable for a wide range of different problems. This generality requires that all basic objects manipulated by the inversion (forward operators, data functionals, model parameters, electric and/or magnetic field solution vectors, etc.) follow uniform conventions. A distinctive characteristic of our formulation is the capability to accommodate most modifications in the geophysical forward problem (details of the solver, types of observable, etc) and easily translate these into implementation of the needed numerical derivative. This uniformity is provided through the "interface" level, discussed in Section 5. In effect this level of ModEM provides specific instances of the abstract classes used by top-level routines. Actual computations for specific problems (e.g., forward problem solutions, evaluation of EM field components at specified data site locations) are implemented in the numerical discretization modules, described in Section 6. Section 7 provides specific examples for two 3D EM inverse problems: magnetotelluric (MT) and controlled-source electromagnetic (CSEM). The two methods, which differ with regard to sources and data types, are implemented through variations of the interface layer. This example illustrates that due to the fine grained modularity of our approach, not only the parallel inversion algorithms, but also the numerical discretization (including the finite-difference 3D EM forward solver) and, more importantly, the sensitivity computations may be effectively reused for these two very different EM techniques.

2. Preamble: theory and notation

The theory of geophysical inversion has been previously described in generic terms in Parker (1994), Zhdanov (2002), Tarantola (2005), among others, and the general principles of discrete sensitivity computations for nonlinear electromagnetic inverse problems in geophysics discussed, for example, in McGillivray et al. (1994), Newman and Hoversten (2000) and EK12.

Table 1
List of notation from Egbert and Kelbert (2012). Here, $\mathcal{M}$ indicates the model space, $\mathcal{D}$ the data space, and $\mathcal{S}_{P,D}$ stand for the spaces of EM fields defined on primary and dual grids, as defined in Egbert and Kelbert (2012).

Symbol	Represents
$\mathbf{m}, \mathbf{m}_0 \in \mathcal{M}$	Model parameter vectors
$\pi(\mathbf{m}) : \mathcal{M} \mapsto \mathcal{S}_{P,D}$	Mapping from model parameter to primary or dual grid
$\Pi_{\mathbf{m}_0} : \mathcal{M} \mapsto \mathcal{S}_{P,D}$	$\partial\pi/\partial\mathbf{m}$, evaluated at $\mathbf{m}_0$
$\mathbf{C_m} : \mathcal{M} \mapsto \mathcal{M}$	Model covariance
$\mathbf{e}, \mathbf{e}_0 \in \mathcal{S}_P$	Solution vectors on the primary grid
$\mathbf{d} \in \mathcal{D}$	Data vector
$\psi_j(\mathbf{e}(\mathbf{m}),\mathbf{m}) : \mathcal{S}_P \mapsto \mathbb{C}$	j'th data functional, in general non-linear
$\mathbf{l}_j \in \mathcal{S}_P$	$\partial\psi_j/\partial\mathbf{e}$, evaluated at $\mathbf{e}_0, \mathbf{m}_0$; j'th row of $\mathbf{L}$
$\mathbf{q}_j \in \mathcal{M}$	$\partial\psi_j/\partial\mathbf{m}$, evaluated at $\mathbf{e}_0, \mathbf{m}_0$; j'th row of $\mathbf{Q}$
$\mathbf{L} : \mathcal{S}_P \mapsto \mathcal{D}$	Sparse matrix constructed from $\mathbf{l}_j$
$\mathbf{Q} : \mathcal{M} \mapsto \mathcal{D}$	Sparse matrix constructed from $\mathbf{q}_j$
$\mathbf{S_m^{-1}} : \mathcal{S}_P \mapsto \mathcal{S}_P$	Forward solver; $\mathbf{S_m e} = \mathbf{b}$
$\mathbf{P} : \mathcal{M} \mapsto \mathcal{S}_P$	Operator $-\partial(\mathbf{S_m e}_0)/\partial\mathbf{m}$, evaluated at $\mathbf{m}_0$
$\mathbf{J} : \mathcal{M} \mapsto \mathcal{D}$	Full Jacobian $\partial\psi/\partial\mathbf{m}$; $\mathbf{J} = \mathbf{L S_{m_0}^{-1} P} + \mathbf{Q}$
$\mathbf{T} : \mathcal{S}_P \mapsto \mathcal{S}_D$	Linear mapping from the primary to dual grid
$\tilde{\mathbf{T}}_{\pi(\mathbf{m}_0),\mathbf{e}_0} : \mathcal{S}_{P,D} \mapsto \mathcal{S}_D$	$\partial(\mathbf{T}_{\pi(\mathbf{m})}\mathbf{e}_0)/\partial\pi$, evaluated at $\pi(\mathbf{m}_0)$
$\lambda^P \in \mathcal{S}_P$	Primary grid interpolation coefficients (sparse)
$\lambda^D \in \mathcal{S}_D$	Dual grid interpolation coefficients (sparse)

In our discussion of these ideas throughout this paper, we use the notation of EK12, which is outlined in Table 1. For completeness we summarize key points here. ModEM provides a general framework for solving regularized EM inverse problems, i.e., minimization of a penalty functional of the form

$$\Phi(\mathbf{m}, \mathbf{d}) = (\mathbf{d} - \mathbf{f}(\mathbf{m}))^T \mathbf{C_d^{-1}} (\mathbf{d} - \mathbf{f}(\mathbf{m})) + \nu(\mathbf{m} - \mathbf{m}_0)^T \mathbf{C_m^{-1}} (\mathbf{m} - \mathbf{m}_0) \quad (1)$$

to recover an Earth conductivity model parameter vector $\mathbf{m}$, which provides an adequate fit to a data vector $\mathbf{d}$. In (1), $\mathbf{C_d}$ is the covariance of data errors, $\mathbf{f}(\mathbf{m})$ defines the forward mapping, $\mathbf{m}_0$ is a prior or first guess model parameter, ν is a trade-off parameter, and $\mathbf{C_m}$ (or more properly $\nu^{-1}\mathbf{C_m}$) defines the model covariance or regularization term.

The forward mapping requires solution of the frequency domain EM partial differential equation (PDE), which in discrete form is written generically as

$$\mathbf{S_m e} = \mathbf{b}. \quad (2)$$

The subscript $\mathbf{m}$ here denotes the dependence of the PDE operator on a specific model parameter, and in the following is often omitted; $\mathbf{e}$ represents the discrete EM field solution; and $\mathbf{b}$ is the forcing (boundary conditions and/or source terms). Typically $\mathbf{e}$ will represent only the *primary* field (e.g., electric) that is actually solved for; the other *dual* field (e.g., magnetic) is then computed via a simple transformation operator $\mathbf{h} = \mathbf{Te}$. Simulated observations are computed from the solution $\mathbf{e}$(and possibly $\mathbf{m}$) via

$$d_j = f_j(\mathbf{m}) = \psi_j(\mathbf{e}(\mathbf{m}), \mathbf{m}). \quad (3)$$

Using the chain rule, a general expression for the Jacobian (or sensitivity matrix) $\mathbf{J}$ ($= \partial\mathbf{f}/\partial\mathbf{m}$) can be given, in the vector notation, as

$$\left.\frac{\partial f_j}{\partial\mathbf{m}}\right|_{\mathbf{m}=\mathbf{m}_0} = \left[\left.\frac{\partial\psi_j}{\partial\mathbf{e}}\right|_{\mathbf{e}=\mathbf{e}_0,\mathbf{m}=\mathbf{m}_0}\right]\left.\frac{\partial\mathbf{e}}{\partial\mathbf{m}}\right|_{\mathbf{m}=\mathbf{m}_0} + \left.\frac{\partial\psi_j}{\partial\mathbf{m}}\right|_{\mathbf{m}=\mathbf{m}_0} \quad (4)$$

where $\mathbf{e}_0$ is the solution to (2) for model parameter $\mathbf{m}_0$. A simple calculation shows

$$\left.\frac{\partial\mathbf{e}}{\partial\mathbf{m}}\right|_{\mathbf{m}=\mathbf{m}_0} = -\mathbf{S_{m_0}^{-1}}\left[\frac{\partial}{\partial\mathbf{m}}(\mathbf{S_m e}_0)\right]_{\mathbf{m}=\mathbf{m}_0} = \mathbf{S_{m_0}^{-1} P}. \quad (5)$$

So, in matrix notation the Jacobian can be expressed as

$$\mathbf{J} = \mathbf{L S_{m_0}^{-1} P} + \mathbf{Q}. \quad (6)$$

The matrix $\mathbf{P}$ depends on details of the numerical model implementation and the conductivity parametrization. An explicit formula for $\mathbf{P}$ can be given, assuming the forward operator can be written

$$\mathbf{S_m e} \equiv \mathbf{S}_0 \mathbf{e} + \mathbf{U}(\pi(\mathbf{m}) \circ \mathbf{Ve}), \quad (7)$$

where $\mathbf{S}_0$, $\mathbf{U}$ and $\mathbf{V}$ are some linear operators that do not depend on the model parameter vector $\mathbf{m}$, $\pi(\mathbf{m})$ is a (possibly non-linear) mapping from the model parameter space to the numerical grid, and ($\circ$) denotes the component-wise multiplication of vectors. Then we have

$$\mathbf{P} = -\mathbf{U}\,diag(\mathbf{Ve}_0)\Pi_{\mathbf{m}_0}, \quad (8)$$

where $\Pi_{\mathbf{m}_0}$ is the Jacobian of the model parameter mapping $\pi(\mathbf{m})$ evaluated at the background model parameter $\mathbf{m}_0$. All of the usual forms for 2D and 3D EM induction operators can be expressed as in (7); e.g., for the second-order 3D staggered-grid finite difference equation for the electric field (with possible source term $\mathbf{j}_s$)

$$\nabla \times \nabla \times \mathbf{E} + i\omega\mu\sigma\mathbf{E} = \mathbf{j}_s \quad (9)$$

$\mathbf{S}_0$ corresponds to the discrete curl–curl operator, $\mathbf{U} \equiv i\omega\mu\mathbf{I}$, $\mathbf{V} \equiv \mathbf{I}$, and $\pi(\mathbf{m}) \equiv \sigma(\mathbf{m})$ is a mapping from the model parameter space to the cell edges, where electric field components are defined.

The matrix $\mathbf{L}$ in (6) represents the linearized data functionals. This can be decomposed into two sparse matrices as $\mathbf{L} = \mathbf{A}^T \Lambda^T$, where columns of Λ are sparse vectors which represent evaluation functionals for point observations of the electric and magnetic fields. The operator $\mathbf{T}$ is generally involved in evaluation of dual field components. The matrix $\mathbf{A}$ depends on details of the (generally non-linear) observation functionals (e.g., impedance, apparent resistivity), which may combine magnetic and electric measurements from one or more locations. More explicitly, each row of $\mathbf{L}$ takes the form

$$(\mathbf{l}_j)^T = \left(\left.\frac{\partial \psi_j}{\partial \mathbf{e}}\right|_{\mathbf{e}_0,\mathbf{m}_0}\right)^T = \sum_{k=1}^{K_P} a_{jk}^P \lambda_k^P + \sum_{k=1}^{K_D} a_{jk}^D [\mathbf{T}^T \lambda_k^D] \quad (10)$$

where λ_k^P and λ_k^D are sparse vectors defined on the primary and dual grids (see EK12, Section 5.2.3), and $\mathbf{l}_j$ is a primary grid sparse vector.

When either the evaluation functionals, or the field transformation operator $\mathbf{T}$ have an explicit dependence on the model parameter there is an additional term in the sensitivity matrix, which we have denoted $\mathbf{Q}$ in (6). Rows of this matrix can be effectively computed using the expression

$$(\mathbf{q}_j)^T = \left(\left.\frac{\partial \psi_j}{\partial \mathbf{m}}\right|_{\mathbf{e}_0,\mathbf{m}_0}\right)^T = \Pi_{\mathbf{m}_0}^T \left[\sum_{k=1}^{K_D} a_{jk}^D \tilde{\mathbf{T}}_{\pi(\mathbf{m}_0),\mathbf{e}_0}^T \lambda_k^D\right], \quad (11)$$

following the notation of Table 1. Note that the expression in the square bracket in (11) is also a sparse vector on the primary or dual grid, depending on where the electrical conductivity (or resistivity) $\pi(\mathbf{m})$ is defined.

The Jacobian represents a linear mapping, giving the perturbation to the data resulting from a model parameter perturbation ($\delta \mathbf{d} = \mathbf{J} \delta \mathbf{m}$). A wide range of gradient-based inversion algorithms make use of this operator, along with the transpose or adjoint ($\delta \mathbf{m} = \mathbf{J}^T \delta \mathbf{d}$). ModEM does not necessarily (or even typically) form the Jacobian, or the component matrices in (6), but rather implements the solver for the discrete system $\mathbf{S}_{\mathbf{m}}^{-1}$, the operators $\mathbf{P}$, $\mathbf{L}$, $\mathbf{Q}$, and the compound Jacobian operator $\mathbf{J}$, together with adjoints. These operators, together with model and data covariances, were then used to implement a range of gradient-based inversion algorithms. As discussed in EK12, and in detail below, in most EM inverse problems there is significant structure to the data vector, implied by the multiplicity of transmitters and receivers. This structure is also reflected in the Jacobian and the component matrices, and thus in the organization of ModEM.

3. Overview of the ModEM program

The organization of ModEM is summarized in Fig. 1, where we distinguish three general levels. At the top are generic components which implement parallelized inversion algorithms and sensitivity computations which can be applied to a wide range of EM inverse problems. At the lowest level are components which define the
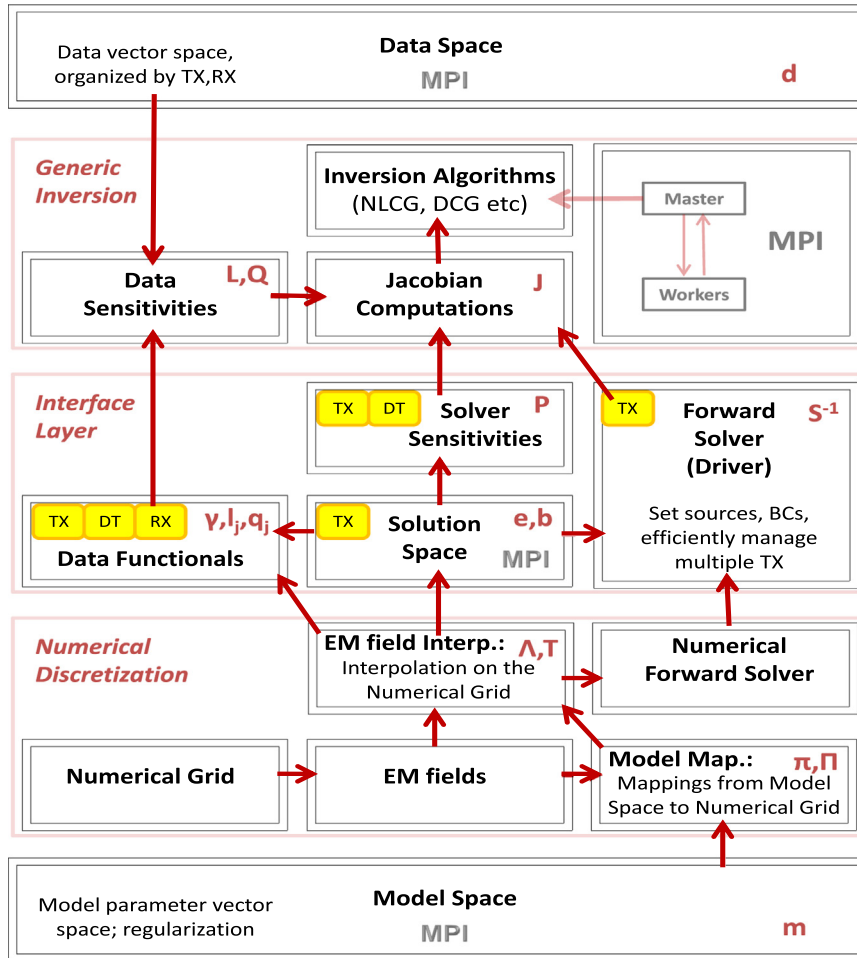
Fig. 1. Schematic overview of ModEM. Conceptual modules, which are described in Sections 3–6, are denoted by boxes, with dependencies indicated by arrows. Some boxes are also marked with symbols to indicate the vectors and operators from Egbert and Kelbert (2012) which they implement (see also Table 1). The shaded small boxes indicate which dictionaries are used in each module. The (optional) Message Passing Interface (MPI) module used for the parallel implementation is shown in lighter colors. Modules marked "MPI" also require additional subroutines to allow transmission of derived data types defined within these modules between processors.

basic discretization and numerical solution approach used for the forward problem. These routines might be used for a variety of frequency domain EM problems, with varying source and receiver configurations. The middle layer provides an interface which hides specific details of the numerical implementation from the generic inversion modules, and defines problem specific details (e.g., source and receiver configurations). In this section, we guide the reader through the overall organization of ModEM code, and discuss the top level, which is the core of ModEM. The interface layer will be further discussed in Section 5, and the numerical discretization layer in Section 6.

As summarized in Section 2, the basic data objects which are manipulated in any inversion scheme include model ($\mathbf{m}$) and data ($\mathbf{d}$) vectors, and EM solution and source fields ($\mathbf{e}$ and $\mathbf{b}$). Except for the data vector these are treated in all top level routines as essentially abstract classes, with no specific implementation details referenced by the generic inversion or sensitivity routines. These are implemented in Fortran 95 as data structures with standardized type names and interfaces, allowing implementations for different problems to be used interchangeably within the inversion system. For each of these data objects a standard series of methods must be defined (creation, destruction, vector space methods, dot products, etc.), again with standardized interfaces. The model parameter $\mathbf{m}$ interacts more directly with the numerical discretization level, and will be discussed further in Section 6. EM solution ($\mathbf{e}$) and source field ($\mathbf{b}$) objects are application specific, and actual implementations of these abstract classes are defined in the interface layer, discussed in Section 5. Note that we also make extensive use of a sparse vector representation for elements of the EM solution space, to allow efficient representation of observation functionals.

In contrast to the other basic data objects, data vectors $\mathbf{d}$ have been implemented with a fixed structure which is accessible to, and heavily used by, the top level inversion and sensitivity routines. We do this to allow efficient treatment of the multi-transmitter/multi-receiver data sets that are common to EM methods. For example, for 3D MT there are multiple frequencies, each requiring separate forward solutions. For each frequency there will be multiple sites, and at each site 8 real components in the impedance tensor, which require solutions for two independent source polarizations for their evaluation. Controlled source problems have a multiplicity of sources and receivers, which may interact in different ways. Data vectors are thus organized according to three attributes which we refer to as *transmitter*, *data type*, and *receiver*. The *transmitter* attribute uniquely defines the forward problem that must be solved, including both the specific partial differential equation (in general this will depend on frequency) and the sources and boundary conditions. The *receiver* attribute is used to define, in conjunction with *data type*, the measurement process that must be applied to the forward solution to allow comparison between measured and predicted data. These attributes are used to define a natural organization of the full data vector $\mathbf{d}$, which at the coarsest level consists of an array of structures corresponding to different transmitters TX. Each of these in turn contains one or more sub-structures for different data types DT, which store all components for all receivers for that TX/DT pair.

The three data attributes TX, DT, RX are treated abstractly at the level of the inversion and sensitivity modules. This is achieved by storing the actual information associated with these attributes as lists, which we call *dictionaries*. Thus, the *transmitter* (TX) dictionary has an entry for each unique forward problem, providing any data such as the frequency or geometry of the source required to set up and solve the forward problem. Entries in the *data type* (DT) dictionary define data functional types included in the inversion, such as impedance, vertical field transfer function,

phase tensor, and apparent resistivity. The *receiver* (RX) dictionary provides information about site locations, and if appropriate, orientation/configuration of the observing system. The dictionaries are only employed by the interface level modules (their use is denoted by small corner boxes in Fig. 1). This separation of the data values from their accompanying meta data information makes the data vector structure completely generic, allows mixing data of different types, and greatly simplifies addition of new data types. At the same time, tagging components of $\mathbf{d}$ with these generic attributes provides enough information about the transmitter/receiver structure so that forward modeling and sensitivity computations can be organized efficiently. For example, the transmitter attribute can be used to ensure that each required forward problem is solved once (and only once), and then used to compute predicted data (or implement appropriate sensitivity calculations) for all necessary receivers and data types.

A key feature of ModEM is the ability to easily implement any linearized inversion scheme that can be expressed in terms of the basic data objects $\mathbf{d}$, $\mathbf{m}$, $\mathbf{e}$ and $\mathbf{b}$, together with the forward mapping $\mathbf{f(m)}$, Jacobian $\mathbf{J}$, and data and model covariances $\mathbf{C_m}$ and $\mathbf{C_d}$. From the perspective of the inversion algorithms only a small number of conceptually simple operations need to be implemented for the covariance operators: multiplication of model parameter objects by $\mathbf{C_m}$, $\mathbf{C_m^{1/2}}$, and perhaps $\mathbf{C_m^{-1/2}}$, and multiplication of data vector objects by $\mathbf{C_d^{1/2}}$ and $\mathbf{C_d^{-1/2}}$. Interfaces for these symmetric covariance operators are simple, with both inputs and outputs being model parameters or data vectors, as appropriate. So far we have only implemented simple diagonal data error covariances within the data vector class. Model covariances, which are generally more complicated, are discussed further below in conjunction with the model parameter class. General implementations of high level routines for forward and Jacobian operators are implemented in the sensitivity module, following conventions given in Table 2.

Using these components we have so far implemented several inversion algorithms including non-linear conjugate gradients (NLCG; e.g., Rodi and Mackie, 2001), data space conjugate gradients (DCG; Siripunvaraporn and Egbert, 2007), and the multi-transmitter hybrid CG-Occam scheme of Egbert (2012). Example pseudo-code and further details are given in Appendix A.

Jacobian routines are further modularized, using the general decomposition of (6) (or its transpose) to implement multiplication by $\mathbf{J}$ (or $\mathbf{J}^T$), using the solver for the discrete system $\mathbf{S_m^{-1}}$, and the operators $\mathbf{P}$, $\mathbf{L}$, $\mathbf{Q}$ (or their adjoints). For efficiency, and to simplify parallelization, these computations are organized by transmitter. Thus, generic routines in the sensitivity module (see Table 2) implement multiplication by $\mathbf{L}$ and $\mathbf{Q}$ for a single transmitter. Each row of these operators ($\mathbf{l}_j$ and $\mathbf{q}_j$ in Table 1) corresponds to a single data type/receiver and defines the derivative of a single (generally non-linear and multi-component) data functional ψ_j with respect to the EM solution $\mathbf{e(l}_j)$ and, if necessary, the model parameter $\mathbf{m(q}_j)$. Individual rows, which can generally (but not always) be represented by sparse vectors in the appropriate space, are application dependent and are constructed by routines in the interface layer. $\mathbf{L}$ and $\mathbf{Q}$ (or the corresponding adjoints) are called, along with the solver and the operator $\mathbf{P}$ (or $\mathbf{P}^T$; see Section 5) to complete Jacobian calculations for a single transmitter. Higher level routines (Jmult, JmultT in Table 2) then loop over transmitters to complete the multiplication by the full Jacobian.

The sensitivity module also provides routines (fwdPred) that implement the full forward problem $\mathbf{d} = \mathbf{f(m)}$, and for calculation of the full Jacobian (calcJ). Note that fwdPred optionally returns the array of EM solutions objects $\mathbf{e}$ computed for all unique transmitters, so that these can be saved (e.g., after evaluation of data misfit) and used for subsequent sensitivity calculations (e.g., to evaluate the gradient of the penalty functional).

Table 2
Public Routines used for sensitivity computations. Conventions used in calling arguments: iDt=data type; iRx=receiver; iTx=transmitter. Other symbols are as in Table 1, with subscript zeros denoting background model parameters and solution vectors used for linearization, and for the template data vector. The template data vector $\mathbf{d}_0$ gets overwritten with the computed vector $\mathbf{d}$ on output, where appropriate. $\mathbf{Z}$ denotes an array of real responses. Optional inputs or outputs are in parentheses; $\mathbf{m}'$ is the optional "imaginary" component of the model parameter output by PmultT.

Routine name	Inputs	Outputs	Used for	Module
initSolver	iTx,$\mathbf{m}$	$\mathbf{e}_0$ ($\mathbf{e}$,$\mathbf{b}$)	fwdPred, Jmult, JmultT, calcJ	Forward solver (driver)
exitSolver	$\mathbf{e}_0$ ($\mathbf{e}$,$\mathbf{b}$)		fwdPred, Jmult, JmultT, calcJ	Forward solver (driver)
fwdSolver	iTx (FWDorADJ,$\mathbf{b}$)	$\mathbf{e}$	fwdPred, Jmult, JmultT, calcJ	Forward solver (driver)
Pmult	$\mathbf{e}_0$,$\mathbf{m}_0$,$\mathbf{m}$	$\mathbf{b}$	Jmult	Solver sensitivities
PmultT	$\mathbf{e}_0$,$\mathbf{m}_0$,$\mathbf{e}$	$\mathbf{m}$($\mathbf{m}'$)	JmultT, calcJ	Solver sensitivities
Qmult	$\mathbf{e}_0$,$\mathbf{m}_0$,$\mathbf{d}_0$,$\mathbf{m}$	$\mathbf{d}$	Jmult	Data sensitivities
QmultT	$\mathbf{e}_0$,$\mathbf{m}_0$,$\mathbf{d}$	$\mathbf{m}$($\mathbf{m}'$)	JmultT	Data sensitivities
Lmult	$\mathbf{e}_0$,$\mathbf{m}_0$,$\mathbf{d}_0$,$\mathbf{e}$	$\mathbf{d}$	Jmult	Data sensitivities
LmultT	$\mathbf{e}_0$,$\mathbf{m}_0$,$\mathbf{d}$	$\mathbf{b}$	JmultT	Data sensitivities
Qrows	$\mathbf{e}_0$,$\mathbf{m}_0$,iDt,iRx	$\mathbf{m}$	Qmult, QmultT, calcJ	Data functionals
Lrows	$\mathbf{e}_0$,$\mathbf{m}_0$,iDt,iRx	$\mathbf{l}$	Lmult, LmultT, calcJ	Data functionals
dataResp	$\mathbf{e}$,$\mathbf{m}$,iDt,iRx	$\mathbf{Z}$	fwdPred	Data functionals
fwdPred	$\mathbf{m}$, $\mathbf{d}_0$	$\mathbf{d}$($\mathbf{e}$)	Parallel inversion algorithms	Jacobian computations
calcJ	$\mathbf{m}_0$, $\mathbf{d}_0$	$\mathbf{J}$	Parallel inversion algorithms	Jacobian computations
Jmult	$\delta\mathbf{m}$, $\mathbf{m}_0$, $\mathbf{d}_0$ ($\mathbf{e}$)	$\delta\mathbf{d}$	Parallel inversion algorithms	Jacobian computations
JmultT	$\mathbf{m}_0$, $\delta\mathbf{d}$ ($\mathbf{e}$)	$\delta\mathbf{m}$	Parallel inversion algorithms	Jacobian computations

The organization of sensitivity computations we have described so far is simple and general, but may not be efficient for all problems. As discussed in EK12, in some cases computations with the Jacobian can be "factored" for efficiency into components that depend on the receiver and on the transmitter. The simplest example is the controlled source cross-well imaging problem considered by Newman and Alumbaugh (1997), with N_T transmitters and N_R receivers. The full Jacobian for this problem can be constructed from forward solutions for each transmitter, and adjoint solutions for each receiver. Thus if $N_T \approx N_R$ the most efficient way to implement a Gauss-Newton scheme is to pre-compute and save these $N_T + N_R$ solutions and use these to implement multiplication by $\mathbf{J}$ and $\mathbf{J}^T$ (Newman and Alumbaugh, 1997). Implementation of such a scheme in ModEM is also straightforward – the basic implementation of routines `Jmult` and `JmultT` described above are simply replaced, using an implementation that pre-computes and saves the appropriate forward and adjoint solutions. All of the computations in these variants are readily implemented using virtually the same components required for the basic versions outlined above.

In summary, routines in the sensitivity module manage interactions among the components of the sensitivity calculation (data functionals $\mathbf{L}$ and $\mathbf{Q}$; solver $\mathbf{S_m^{-1}}$; $\mathbf{P}$), using the structure of the data vector to ensure efficiency. As we discuss next, a coarse parallelization (over transmitters, or unique forward problems, similar to the approach used in Siripunvaraporn and Egbert, 2009) is also implemented at the level of the sensitivity module, and is thus completely isolated from details of the specific EM inverse problem that ModEM is applied to.

4. Parallelization

ModEM incorporates a flexible coarse-grained parallelization over forward problems following the scheme of Meqbel (2009), a variant of the master–worker parallelization method. The scheme minimizes communication between workers and master, and reduces memory requirements, by having workers store results of forward computation for reuse in the solution of the adjoint problem. Given the data vector transmitter/receiver structure, this can be implemented quite easily, through parallelized versions of the upper level routines which organize Jacobian calculations (`Jmult`, `JmultT` and `calcJ` in Table 2), also modified to allow for the bookkeeping necessary to maintain storage efficiencies in the worker nodes. Parallel versions of these routines are contained in a separate module.

The scheme is implemented using calls to the standard Massage Passing Interface (MPI) communication library, with only complications arising from efforts to minimize interaction of the parallelization with the rest of the ModEM system. For example, messages passed between processors using MPI are restricted to standard data types in the programming language used (i.e., in Fortran 95: Integer, Real, etc.), while ModEM makes extensive use of derived data types, e.g., to store $\mathbf{m}$, $\mathbf{e}$ and $\mathbf{d}$ as abstract encapsulated objects. Thus, to allow a generic implementation of the parallelization which can pass such objects around without reference to internal details of the actual data structure, it is necessary to implement specialized utility routines for each specific instance of the Model Space, Solution Space and Data Space modules. These are routines with generic names and interfaces that create an MPI structured data type from the Fortran derived data type, thus allowing communication routines in the MPI module to also treat these objects abstractly while sending and receiving. The application specific MPI source code for these utility routines is kept in separate files, which are included in the appropriate serial modules when compiling for parallel execution. Finally note that relatively trivial modifications to each specific inversion routine is required, essentially to route calls to forward modeling and Jacobian calculations through the parallel or serial versions of these routines, as appropriate. To avoid having two copies of these routines this is accomplished through compiler directives.

The general relationship of the MPI Main module to the rest of ModEM is illustrated in Fig. 1, and an overview of our MPI implementation is given in Fig. 2. Briefly, after processor initialization, one processor is assigned as the master, and the rest as workers. The workers enter a queue (in routine `Worker_Job`, say) and await messages from the master, while the master steps through the actual inversion algorithm, until a step requiring solution of the forward or adjoint problem is reached. At this point the master distributes messages to all workers, indicating which task to perform, and providing any necessary input data. The worker executes the requested job, typically for a single transmitter, by calling the appropriate sensitivity or interface layer subroutines–the same code that would be executed in a serial implementation. Thus, except for data type conversion, essentially all MPI specific routines are hosted inside a single MPI module.

Our parallel scheme is designed to minimize communication between processors. Consider for example the matrix-vector multiplication implemented in `JmultT`. As noted in Table 2,

a

```
MPI_Main
  ┌─────────────────────────────────────────────────────────────┐
  │ Master:                                                       │
  │                                     :                         │
  │                                     :                         │
  │ Subroutine Master_Job_fwdPred (m0,d,e0)                       │
  │     1- Task = 'Distribute Model Parameters'                   │
  │         -Send a copy of model parameters to all workers.      │
  │     2- Task = 'Solve Forward Problem '                        │
  │         - Send the index of one transmitter to each worker    │
  │         - Receive the predicted data                          │
  │ End subroutine                                                │
  │                                     :                         │
  └─────────────────────────────────────────────────────────────┘
  ┌─────────────────────────────────────────────────────────────┐
  │ Worker:                                                       │
  │ Do                                                            │
  │                 Receive task from Master                      │
  │                                                               │
  │     if task = 'Distribute Model Parameters'  then             │
  │         Receive a copy of model parameters from Master        │
  │                                                               │
  │     if task = 'Solve Forward Problem '  then                  │
  │         Receive the index of one transmitter from Master      │
  │           Call fwdPred_TX (m0,d,e0)                           │
  │         Send the predicted data for one transmitter to Master │
  │                                                               │
  │     if task = 'Stop' then                                     │
  │         EXIT loop                                             │
  │                                                               │
  │ Loop                                                          │
  └─────────────────────────────────────────────────────────────┘
```

b

```
Serial version: i.e. inside NLCG
            :
            :
            :
            :
            :
            :
    Call fwdPred  (m0,d,eAll)
            :
            :
            :
```

c

```
Serial/Parallel version:  i.e. inside NLCG
            :
            :
            :
#if def=MPI then
   Call Master_Job_fwdPred (m0,d,eAll)
#else
   Call fwdPred  (m0,d,eAll)
#end if
            :
            :
            :
```
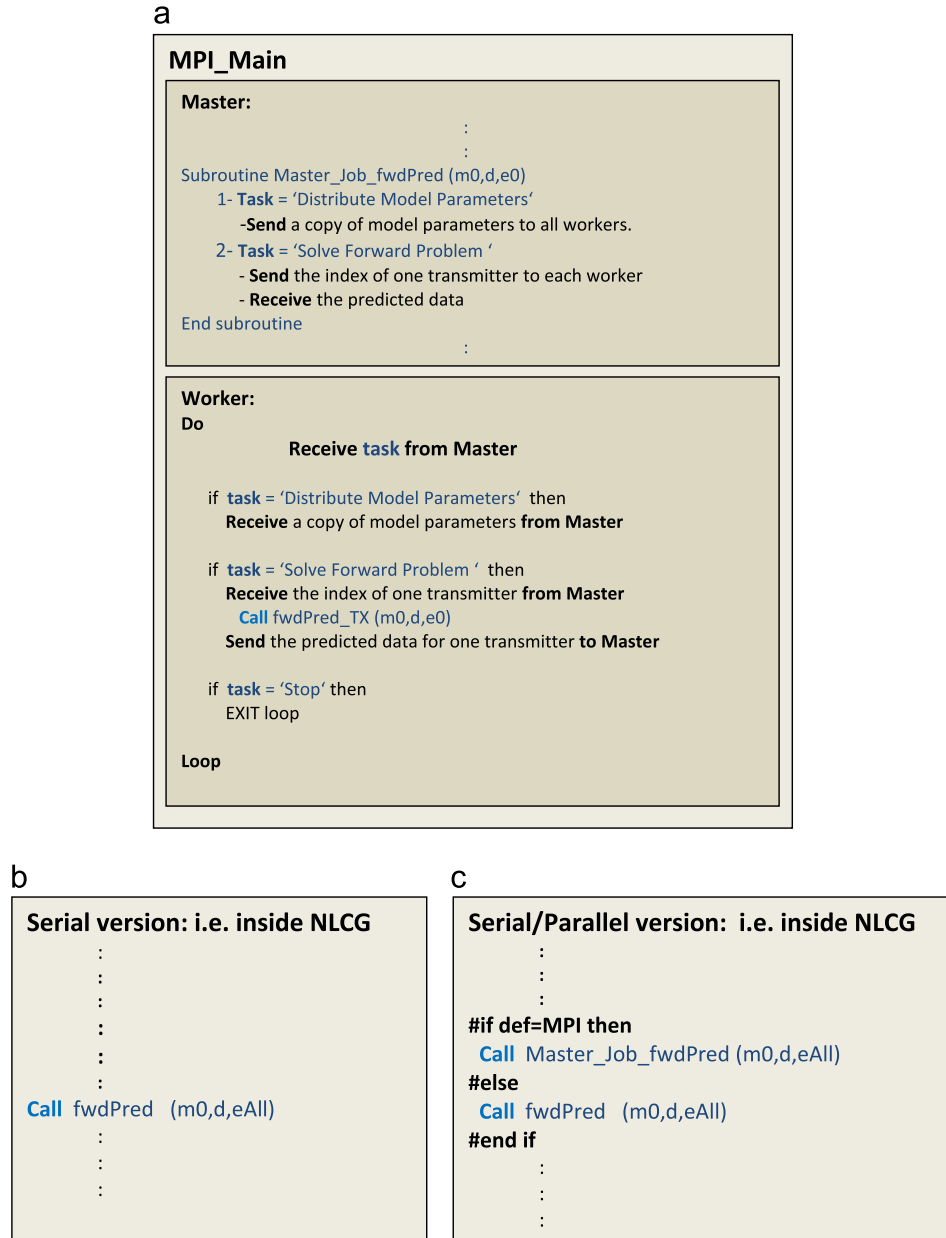
Fig. 2. Pseudo-code for the Message Passing Interface (MPI) modular implementation.

this routine takes the background EM solution (which typically will already have been computed to evaluate model misfit) as an optional input. We keep the solution for a particular transmitter on the processor used to compute it, and assign the same transmitter to this processor for the task of computing the product $\mathbf{J}^T\mathbf{d}$ (for a single transmitter). Thus it is not necessary to gather EM solutions for all transmitters on the master processor and then re-distribute these to the workers. The master sends only a transmitter index and the data vector ($\mathbf{d}$) to each worker to perform the multiplication in parallel. Of course the resulting model parameters must still be returned to the master to be summed to complete computation of $\mathbf{J}^T\mathbf{d}$ for the full model vector.

5. Interface layer

The central grouping in Fig. 1 provides an interface between the generic sensitivity and inversion modules discussed in the previous section, and the implementation specific details of the

numerical discretization modules. In particular, the EM solution and source terms $\mathbf{e}$ and $\mathbf{b}$ are defined at this level in the solution space module, as is the abstracted forward solver (which maps $\mathbf{b}$ to $\mathbf{e}$), the problem specific data functionals (which map $\mathbf{e}$ to $\mathbf{d}$), and the operator $\mathbf{P}$, which defines the sensitivity of the forward solver to the model parameter. All of these objects are used extensively by the generic sensitivity and inversion routines, and thus must have standardized names and interfaces, as in Table 2.

From the perspective of object oriented programming, the interface layer defines specific instances of abstract classes, which implement the methods needed by the generic top-level sensitivity routines. In addition to hiding details of the actual numerical implementation from generic inversion routines, the interface layer is where source and receiver details for specific EM applications are defined. As we shall see in Section 7, inversions for different EM methods (e.g., MT and CSEM) may be developed using the same base of numerical discretization modules and the same generic inversion modules through modifications to the interface layer.

Central to the interface layer is the solution space module, which defines derived data types used to represent the *solution vector* **e** and *source vector* **b** of the discrete forward problem (2) for a single transmitter. These two distinct data types represent objects which are effectively in the same space, but we have found it useful to distinguish between vectors used for sources and solutions. The *sparse vector* data type is also used to represent elements of this same space, in this case to provide a storage efficient representation of the sparse data functionals which constitute rows of the operator **L**. Basic methods for manipulating all of these objects, including creation, destruction, I/O, copying, linear algebra and dot-product operations, are used extensively throughout the Jacobian computations and thus must be implemented with standardized names and interfaces.

Note that the full solution vector employed in an inverse problem will in general be an array of *solution vector* objects, one for each transmitter. Elements of this array could in principal be quite different, for example in the case of joint inversion. For instance, joint inversion of 3D MT and CSEM could be easily accommodated by storing a solution for the MT problem in **e**(1) (encapsulating solutions for two source polarizations for one period), while **e**(2) could represent a solution for a single controlled source transmitter. And, for the 2D MT problem, the array of *solution vector* objects would generally contain both TE mode (electric field) and TM mode (magnetic field) forward solutions. The interface layer hides such complications from the generic sensitivity and inversion routines, though obviously these details are critical to implementation of actual computations.

5.1. Forward solver

The purpose of this module is to provide a uniform interface between forward modeling routines implemented in the numerical discretization layer, and the generic inversion and sensitivity routines of Section 3. The key public routines in this module include initialization (`initSolver`), deallocation and clean up (`exitSolver`), and the actual solver (`fwdSolver`), with interfaces as defined in Table 2. Internal functioning of these routines is application dependent, and controlled by the index into the *transmitter* dictionary, *iTx*. For example, in the 3D MT case (see Section 7) the output of `fwdSolver` **e** consists of solutions for two source polarizations, requiring separate calls to the actual numerical solver with different boundary conditions, for the frequency defined by *iTx*. In the 2D MT case only a single solution is required, but now there are two possibilities: either a TE or a TM solver might be required, depending on the mode, which again would be defined (along with the frequency) with reference to the *transmitter* dictionary. While applications may differ in internal function (a loop over polarizations for the 3D MT implementation vs. a case statement for the 2D case), routine names, interfaces, and abstract functionality (i.e., "*compute* **e** for *transmitter iTx*") are standardized.

Initialization (`initSolver`) is explicitly separated from actual solution to enable more efficient computational strategies. For example, if a direct matrix LU factorization is practical, this can be implemented in the initialization routine, and executed only if the coefficient matrix has changed from the previous solver call. More generally, repetition of operator setup steps can also be minimized by keeping track of attributes (frequency, conductivity parameter) used for the previous solution. The responsibility for these efficiencies lies with the initialization routine – higher level routines that require solutions of the governing PDE always call the initialization routine first. The cleanup routine `exitSolver` is only called when it is desired to deallocate all operator arrays and return solver module data to the state it had before the first call to `initSolver`. Finally, `fwdSolver` implements the general solver,

allowing arbitrary forcing and boundary conditions, and solutions for both the usual forward problem and its transpose, or adjoint. By default, forcing for the forward problem is computed internally, with reference to the *transmitter* dictionary. For sensitivity calculations the forcing is provided as an optional argument, along with an optional switch that specifies execution of the adjoint or forward solver.

5.2. Data functionals

This module implements the data functionals ($\psi_j(\mathbf{e}, \mathbf{m})$ in Table 1), as well as the linearizations with respect to **e** ($\mathbf{l}_j$; rows of **L**) and, if necessary, **m**($\mathbf{q}_j$; rows of **Q**). As shown explicitly in EK12, these functionals can be expressed in terms of (1) a mapping which converts the primary field (e.g., electric) to the dual field (e.g., magnetic); (2) interpolation operators for the primary and dual fields, which simulate the process of measuring EM components at specific locations; (3) functionals of the measured field components, e.g., an impedance or apparent resistivity (see Eqs. (10) and (11)). The first two components are tied closely to the numerical grid, and are implemented at the level of the numerical discretization, to be discussed in Section 6. These decompositions of the linearized data functionals into a set of elementary mappings allow for a completely streamlined modular implementation of the data functionals, making it especially easy to write the sensitivity derivations for any general data functional in terms of pre-existing elementary operators.

Three public routines with standardized names and interfaces (see Table 2) implement data functionals for use in sensitivity and inversion computations. `dataResp` evaluates the (possibly non-linear) data functionals $\psi_j(\mathbf{e}, \mathbf{m})$ for a single arbitrary receiver/data type. Information about the data type (e.g., an impedance, apparent resistivity or phase) is obtained by reference to the *data type* dictionary (indexed by *iDt*) while meta-data such as the site location is obtained from the *receiver* dictionary (indexed by *iRt*). Through calls to lower level (numerical implementation specific) routines all necessary interpolation operators are created, and magnetic and electric components at the observation location are evaluated. Any additional computations (e.g., impedance, amplitude, phase) are completed at the interface layer. Note that in general multiple components (e.g., all elements of an impedance tensor) may be returned for some data types. Note also that all data functional routines implement computations (selected by a case statement) for the full range of data types and transmitters.

`Lrows` implements the linearization of the data functional with respect to variations in the EM solution **e**. In the simplest case, where the data is just a measured field component (as for example in CSEM methods) and $\psi_j(\mathbf{e}, \mathbf{m})$ is already linear in **e**, this routine simply computes the appropriate evaluation functional (exactly as in `dataResp`), and returns this, packaged as a sparse vector in the EM solution space. For the more general case the linearized functionals $\mathbf{l}_j$ are linear combinations of those used to represent basic field component evaluation, with coefficients that depend on the background EM solution $\mathbf{e}_0$ (see Eq. (10). These sparse vectors are used in `Lmult` (Table 2) to form the dot products with **e** required to assemble the *data vector* object $\mathbf{d} = \mathbf{Le}$. `LmultT` multiplies these vectors by the appropriate data components, and sums to form the forcing for the adjoint system $\mathbf{L}^T\mathbf{d}$. See Section 7 for a specific example (3D MT).

The function of `Qrows` is similar, but instead of data functionals this routine returns one or more *model parameters*, $\mathbf{q}_j$. A general expression for these rows of **Q** is given in (11), with further details provided in EK12, Section 5.2.3. Here we note only that each of these model parameters can be expressed as a product of a sparse vector in the EM solution space (formed as for $\mathbf{l}_j$ with reference to the *data type* and *receiver* dictionaries), and the (transposed)

linearized model parameter mapping $\Pi_{\mathbf{m}_0}^T$ (see Section 6). Although in many cases the result will be sparse in the model parameter space, there are exceptions, so ModEM represents the $\mathbf{q}_j$ as full model parameters. Also, note that while model parameters are assumed to be real, the sparse vectors in the EM solution space that they multiply are complex. As discussed in EK12, in some situations (e.g., computation of the full Jacobian) both real and imaginary parts are required, while for other situations (e.g., computation of the data misfit derivative) only the real part is required. In Qmult (Table 2), dot products are formed with the input model parameter and assembled into the *data vector* object $\mathbf{d} = \mathbf{Q}\mathbf{m}$. For the transpose QmultT, the model parameters $\mathbf{q}_j$ are multiplied by the respective data components and summed to obtain $\mathbf{m} = \mathbf{Q}^T\mathbf{d}$.

5.3. Solver sensitivity

The final component of the interface layer implements $\mathbf{P}$, which effectively defines the sensitivity of the forward problem coefficient matrix $\mathbf{S}_{\mathbf{m}}$ to the model parametrization (see Eq. (5)). As seen from (8) this operator can be divided into two components: $\mathbf{U}\,diag(\mathbf{Ve}_0)$, which depends explicitly on the problem formulation (i.e., the form of the PDE used) and the numerical implementation, but not the specific model parametrization, and $\Pi_{\mathbf{m}_0}$, the linearized mapping of the model parameter to the grid. Pmult computes the product $\mathbf{P}\delta\mathbf{m}$ by first applying $\Pi_{\mathbf{m}_0}$ to model parameter object $\delta\mathbf{m}$, and then applying $(\mathbf{U}\,diag(\mathbf{Ve}_0))$ to the result. This routine returns a *source vector* object, of the proper type to provide forcing for the solver in a computation such as $\mathbf{J}\delta\mathbf{m} = \mathbf{LS}^{-1}\mathbf{P}\delta\mathbf{m}$. PmultT multiplies a *solution vector* object by $\mathbf{P}^T$, reversing these steps, and resulting in a model parameter vector. As with QmultT, the output of PmultT is intrinsically complex, and in some cases (e.g., computing the full sensitivity matrix) both real and imaginary parts are saved.

Therefore, the decomposition of the solver sensitivity operator $\mathbf{P}$ in Eq. (8) into the elementary components of the forward problem, identified in Eq. (7), simplifies implementation of sensitivity computations for a wide range of EM problems. Once the forward solver is theoretically decomposed into a combination of elementary discrete operators, the solver sensitivities may be immediately expressed in terms of these elementary components and plugged into the inversion scheme to provide the correct discrete numerical derivative.

6. Numerical discretization

The final group of modules (Fig. 1) provide the basic numerical grid, structures which represent primary and secondary fields, the actual numerical forward solver, and a basic set of interpolation functionals for point evaluation of electric and magnetic fields. For purposes of our discussion here we also include the model parameter module in this group, although, as discussed in Section 3, these can also be considered higher level objects, which are manipulated directly by inversion and sensitivity routines, and thus must conform to generic ModEM interface standards. However, even though the internal structure of $\mathbf{m}$ (and the model covariance) can be quite independent of the grid and other numerical implementation details, the model parameter does play a critical role in defining the discrete numerical operator and thus must interact quite strongly with other numerical discretization components.

The remaining numerical discretization modules have no direct interaction with the generic inversion and sensitivity routines, so a wide range of specific implementations can be accommodated through modifications at the interface layer. It is thus most useful to discuss these modules in the context of a specific example. We do this in Section 7, after offering some general guidelines on those features, conformance to which ensures the functionality required by more generic components, and an overview of how modules can be organized (e.g., as outlined in Fig. 1) to simplify interfacing with the rest of ModEM.

The basic numerical discretization data objects are the grid, and structures which represent the discretized electric and/or magnetic fields. In the terminology of EK12, the latter are elements of the primary and dual spaces, $\mathcal{S}_P$ and $\mathcal{S}_D$. Although not strictly required, explicitly defining an *EM field* data type (here the type name is unimportant), along with the related linear algebra and other basic data object operations, greatly simplifies construction of the required encapsulated *solution vector* and *source vector* objects ($\mathbf{e}$ and $\mathbf{b}$) discussed in Section 5. Similarly, defining sparse vector representations for elements of $\mathcal{S}_P$ simplifies construction of data functionals. An example of an *EM field* module, including relevant interactions with higher level modules, is given in Section 7. An encapsulated data structure which defines the numerical grid does have to be defined, although there are no specific requirements on the form that this takes. Some routines in the sensitivity module pass pointers to the grid, e.g., to allow data site locations to be referenced to the numerical solution grid.

Forward modeling is of course the core of the inversion. In principal an existing forward code can be used, but to be useful for ModEM several conditions must be met. First, the solver should be general, in the sense that the solution can be computed correctly for arbitrary sources and boundary data, even if the forcings encountered for the forward problem are more restricted. For example, for the usual MT forward problem forcing is restricted to the boundaries, but more general source terms must be allowed for to compute sensitivities. Second, capability to solve the transposed system is required. Because the EM equations are essentially self-adjoint, forward codes can generally be easily amended to implement adjoint calculations (e.g., see EK12 (Section 3), and Section 7), although there can be some subtle issues (e.g., Kelbert et al., 2008). Finally, the solver should have a clean interface that allows interaction with higher level routines. At a minimum these must include initialization of the solver, and updating the PDE coefficients (which depend on the model parameter, and the frequency), as well as actually solving the forward or transposed equations for a particular set of sources and boundary conditions.

Developing a separate module with standardized functionality for the interpolation functionals used to evaluate electric and magnetic fields at an arbitrary point within the model domain (e.g., *EM field interpolation* in Fig. 1) greatly simplifies implementation of data functionals at the interface level (see Section 5). The essential requirement is for routines which compute interpolation functionals λ for both primary and secondary field components, and return these as sparse vectors in the primary field space $\mathcal{S}_P$. The case of the primary (e.g., electric) field is clear; the non-zero components of λ are the local interpolation weights. For the dual field (e.g., the magnetic field $\mathbf{h} = \mathbf{Te}$), λ must incorporate the transformation operator ($\mathbf{T}$) used to compute the dual field, so that the functional can still be applied directly to the primary field vector. Note that it is not necessary to form the full transformation operator, as only a few components are needed to form the dual field evaluation functional for any location. If either interpolation functionals, or the transformation operator, depend explicitly on the model parameter some additional sparse vectors are required for assembly of rows of $\mathbf{Q}$. Further details are provided in EK12 (Section 5.2.3).

The sparse vectors returned by EM field interpolation routines are the basic building blocks for the non-linear data functionals ψ_j (e.g., impedances), as well as the corresponding linearizations – the rows of $\mathbf{L}$ and $\mathbf{Q}$ discussed in Section 5. The evaluation

functionals of course depend critically on details of the numerical grid, while computation of something like an impedance or apparent resistivity from the interpolated electric and magnetic field components does not. Thus, a different numerical discretization of the problem (e.g., finite elements with a non-structured grid) would require different interpolation routines. However, if these were again represented as sparse vectors in the appropriate discrete EM field spaces, higher level data functional routines (e.g., for calculation of impedance elements), or construction of the corresponding linearized data functionals, would remain unchanged. Conversely, to add new data types (e.g., inter-station magnetic transfer functions) there is no need to revisit the interpolation aspects of the problem.

The model parameter module consists of three clearly distinguishable components: (1) an encapsulated data structure which represents **m**, together with methods to implement vector space operators, including dot products; (2) the model parameter covariance or regularization operator; (3) mappings between the model parameter and the numerical grid. Note that to some extent these components can be modified independently – e.g., a different covariance implementation can be used without changes to the other two components, and if the numerical discretization were changed, only the mapping component would have to be modified. Key aspects of the first two components have been sketched in Section 3. The third component defines three routines that interact with the interface layer. The first defines $\pi(\mathbf{m})$, the model parameter mapping which specifies resistivity or conductivity on the numerical grid, as required to define the forward operator. This will generally be called by the forward solver initialization routine (`initSolver`). The second and third implement multiplication by the Jacobian of the model parameter mapping $\Pi = \partial\pi/\partial\mathbf{m}$, and its adjoint Π^T. These are required to implement multiplication by **P** and **Q**, and their adjoints. To the extent that the basic interpolation functionals depend on the model parameter, it is also useful to define the restriction of $\pi(\mathbf{m})$ to specific grid elements.

7. Example: 3D EM

7.1. Numerical discretization

As specific examples, we consider application of ModEM to 3D MT and CSEM inverse problems. Both are implemented using the same numerical discretization modules, based on finite difference (FD) solution of the quasi-static Maxwell's equations in the frequency domain, in the form given by (9) on a staggered-grid (e.g., Yee, 1966; Siripunvaraporn et al., 2002).

We begin with a summary of the discretization of the equations, following the notation of EK12 where further details are provided. Electric fields are defined on cell edges (see Fig. 3); the (primary) space of such discrete vector fields is denoted $\mathcal{S}_P$. The dual space, denoted by $\mathcal{S}_D$ consists of vector fields with components defined on cell faces, representing magnetic fields. Eq. (9) may then be written in discrete form as

$$[\mathbf{C}^\dagger\mathbf{C}+diag(i\omega\mu\sigma(\mathbf{m}))]\mathbf{e}=\mathbf{j}_s \qquad (12)$$

$$\mathbf{Be}=\mathbf{e}_b, \qquad (13)$$

where $\mathbf{e}\in\mathcal{S}_P$, $\mathbf{j}_s\in\mathcal{S}_P$ is the optional interior forcing, $\mathbf{C}:\mathcal{S}_P\mapsto\mathcal{S}_D$ is the discrete approximation of the curl operator mapping cell edge vectors (including boundaries) to interior cell face vectors, $\mathbf{C}^\dagger:\mathcal{S}_D\mapsto\mathcal{S}_P$ is the discrete curl mapping interior cell face vectors to interior cell edges, **B** extracts boundary edges, and $\mathbf{e}_b$ gives the specified boundary data. The interior forcing on the right-hand side of (12) is applicable to CSEM but not MT, and is defined on the interior primary cell edges only.
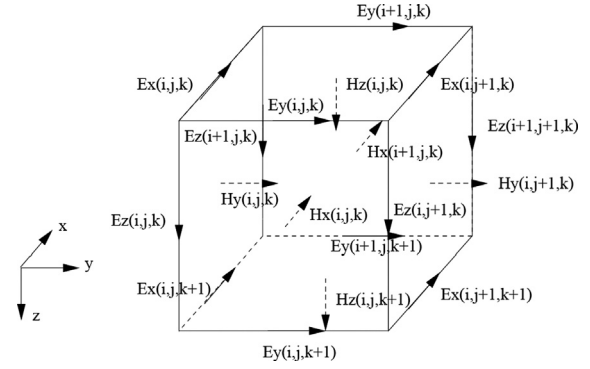

Fig. 3. Staggered finite difference grid for the 3D MT forward problem. Electric field components defined on cell edges are the primary EM field component, which the PDE is formulated in terms of. The magnetic field components can be defined naturally on the cell faces; these are the secondary EM field components in this numerical formulation.

The dependence of the operator coefficients on the model parameter is made explicit in (12) through the mapping $\sigma:\mathcal{M}\mapsto\mathcal{S}_P$, where $\mathcal{M}$ is the model parameter space. Magnetic fields corresponding to an electric field solution **e** can be expressed as

$$\mathbf{h}=(-i\omega\mu)^{-1}\mathbf{Ce}=\mathbf{Te}. \qquad (14)$$

To implement numerical discretization modules based on this formulation, we define the basic *EM field* object as a derived data type containing three separate 3D arrays (*x*, *y*, and *z* components), together with a pointer to the underlying basic grid (which defines the geometry and georeference of the staggered grid), and a tag to indicate if the vector field is defined on the primary (edges) or dual (faces) grid. Several variants on these basic complex vector objects are also defined, including scalar fields (defined either on cell centers or nodes), and allowing for real as well as complex versions of both scalars and vectors. For example, complex scalars are used to represent electric potentials (used in the solver for the divergence correction; Smith, 1996); a real vector can be used to represent electrical conductivity defined on cell edges, i.e., $\sigma(\mathbf{m})$ in (9). We also define a sparse vector representation for elements of $\mathcal{S}_P$ and $\mathcal{S}_D$, to allow efficient representation and storage of measurement functionals. For all of the variants of the basic vector and scalar fields there are routines for creation, deallocation, linear algebra, dot products, and point-wise multiplication. Algebraic routines for relevant combinations of types (e.g., dot products of sparse and full vectors, used for evaluation of field component functionals; point-wise multiplication of complex and real vector fields, required to compute $i\omega\mu\sigma\mathbf{E}$) are also included. Finally, we also define a specialized data structure to store data for boundary conditions.

The forward operator of (9) is implemented using a matrix-free approach, and the system of linear equations is solved iteratively with a quasi-minimum residual (QMR) scheme, with a level-1 incomplete LU decomposition for pre-conditioning. As in Smith (1996) and Siripunvaraporn et al. (2002), a divergence correction is applied periodically, with the Poisson-like equation solved with pre-conditioned conjugate gradients. Everything is coded in an object-oriented manner, so that, for example, solvers and pre-conditioners are standalone modules which can be easily replaced with alternative algorithms. Implementation of the transposed solver is straightforward. Following EK12 (Section 3 and Appendix D) the differential operator $\mathbf{S_m}=\mathbf{C}^\dagger\mathbf{C}+diag(i\omega\mu\sigma(\mathbf{m}))$ satisfies $\mathbf{S_m^T}=\mathbf{VS_mV}^{-1}$ where **V** is the diagonal matrix of integration volume elements for $\mathcal{S}_P$. This means that $\mathbf{VS_m}=\mathbf{S_m^T V}$ is symmetric, and to improve performance of the QMR solver we transform to this modified system. Thus, to solve $\mathbf{S_m e}=\mathbf{b}$ we multiply **b** by **V** before calling the solver; for solutions to $\mathbf{S_m^T e}=\mathbf{b}$ this preliminary step is replaced by multiplying the output of the solver by $\mathbf{V}^{-1}$.

Interpolation functionals are based on tri-linear splines. Separate routines compute interpolation coefficients for vectors defined on edges and faces, returning sparse vectors $\lambda^P \in \mathcal{S}_P$ and $\lambda^D \in \mathcal{S}_D$ respectively. A third routine combines the coefficients of λ^D with appropriate rows of the transformation operator of (14) to construct the sparse magnetic field interpolation functional, represented as a sparse vector in $\mathcal{S}_P$.

For an initial implementation of the model parameter module, we have used a very simple, and classical, scheme: $\mathcal{M}$ consists of the space of log conductivities defined independently on each of the cells in the underlying numerical grid. Thus, as discussed in EK12, a physically consistent (current conserving) form for the mapping $\sigma : \mathcal{M} \mapsto \mathcal{S}_P$ is given by $\sigma(\mathbf{m}) = \mathbf{W}\exp(\mathbf{m})$, where $\mathbf{W}$ is the matrix representing the operator that assigns the weighted average of the four surrounding cells to each edge. The Jacobian of this operator evaluated at the background conductivity $\mathbf{m}_0$ is thus

$$\Pi_{\mathbf{m}_0} = \mathbf{W}[diag(\exp(\mathbf{m}_0))]. \tag{15}$$

We also require the transpose $\Pi_{\mathbf{m}_0}^T$, a mapping from cell edges to cells, computed as a weighted sum of contributions from all edges that bound a cell.

Our basic model covariance is somewhat more novel: we have implemented a recursive autoregressive covariance smoother, loosely based on Purser et al. (2003a,b), which penalizes deviations from a specified prior model conductivity. Horizontal and vertical smoothings may be different; moreover, the extent of horizontal smoothing may vary with depth. The configuration file also allows the user to divide the model domain into disjoint pieces, allowing smoothing to be turned off across domain boundaries, and also allowing model parameters within a domain to be frozen. It is worth stressing again that the model parametrization and covariance are completely independent of the rest of ModEM. In practice, we envision supporting a menu of model parametrization and covariance choices for 3D EM problems.

7.2. Interface layer

The interface layer is slightly different for MT and CSEM implementations, starting with differing requirements for the *solution vector* $\mathbf{e}$. As discussed in detail in EK12, to allow evaluation of the basic 3D MT data type, the impedance tensor, $\mathbf{e}$ must contain a pair of complex vector *EM field* structures, corresponding to solutions for two independent source polarizations. Similarly, the source term $\mathbf{b}$ in (2) must define boundary conditions for two forward problems, and the linearized data functionals (e.g., $\mathbf{l}_j$) must represent a pair of sparse elements of $\mathcal{S}_P$, since calculating a predicted impedance requires sampling solutions for both polarizations. On the other hand, for most CSEM problems each observation will correspond to a single known source, so all interface layer objects (i.e., $\mathbf{e}$, $\mathbf{b}$, $\mathbf{l}_j$) will contain only a single *EM field* structure. To accommodate both of these cases we have defined *solution vector* and related data types for 3D EM problems to allow for any number of coupled source polarizations, defined by a variable (e.g., $nPol$) in the data structure. This makes treatment of a joint MT/CSEM problem straightforward – e.g., the array of *solution vectors* $\mathbf{e}$ can contain solutions for both sorts of EM data, with some having $nPol = 1$, and some $nPol = 2$, with the appropriate case selected by reference to the *transmitter* dictionary.

The interface layer Forward Solver module manages forward and adjoint solutions, including setting up sources and/or boundary conditions for the forward problem. This module thus also must provide slightly different functionality for the MT and CSEM problems. For the MT forward problem, each call to fwdSolver requires two calls to the FD solver with boundary conditions for two source polarizations (e.g., computed by solving 2D forward problems for the appropriate frequency, defined by *iTx*). Adjoint solutions similarly

require a pair of FD solver calls (now boundary conditions in both cases are homogeneous, but sources will differ). Of course only a single FD solver call will generally be required in the CSEM case, both for forward and for adjoint problems. A more significant difference is that for many CSEM problems (e.g., with point dipole sources) it is natural to use a secondary field formulation (e.g., Alumbaugh et al., 1996) for the forward problem. This can be quite simply implemented with changes only to the interface level, using a support module which solves the forward problem for a 1D background conductivity semi-analytically. This is used in the CSEM version of the Forward Solver module, where routine initSolver sets up the background and anomalous conductivity and does preliminary computations for the 1D solution (using the 1D CSEM forward solver of Key (2009)), with all results stored as private saved variables. Then, on each call to fwdSolver the appropriate location and frequency is extracted from the *transmitter* dictionary, and the 1D background solution for this transmitter, and then the forcing for the secondary field solution, are computed. The FD solver is then called to compute the 3D secondary field, which is added to the 1D background solution. The total field solution is returned by fwdSolver. All of these differences in implementation are hidden, both from higher level sensitivity and inversion routines, and from the lower level FD codes. Again, it is straightforward to allow for joint inversion, combining the functionality required for MT and CSEM into single module.

For a simple CSEM problem, where data are direct observations of field components, implementation of data functionals is trivial at the interface level, as the essential functionality is already provided by the basic interpolation functionals. In this case Lrows is just a wrapper which calls the appropriate interpolation functional (determined by the *receiver* dictionary), and packages the output as a *sparse vector*. The forward routine dataResp is also simple, but goes one step further to form the dot product between the *sparse vector* and the *solution vector*, returning predicted data. For 3D MT all data types are functions of impedance tensor elements, which are simple non-linear functions of the raw field components. Routine dataResp thus first computes the basic interpolation functionals required to evaluate E_x, E_y, H_x, and H_y at the site, and applies these to solutions for both polarizations. Then the predicted impedance tensor can be computed, followed (if needed) by transformations required for a particular data type (e.g., computation of an apparent resistivity or phase). Multiple data types are supported, with appropriate code selected by a case statement. Lrows implements linearizations of these calculations, forming linear combinations of the basic interpolation functionals, following specific formulae given in EK12. Neither the tri-linear spline interpolation coefficients (λ^P and λ^D as in Appendix 3) nor the transformation operator $\mathbf{T}$ defined in (14) depend directly on $\mathbf{m}$, so $\mathbf{Q} \equiv 0$. Qrows thus simply outputs $\mathbf{q}_j = 0$, for the MT and CSEM implementations discussed here.

Finally, the form for $\mathbf{P}$ can be easily derived using results from EK12, summarized in Section 2. Comparing (12) with (7), and using (8) and (15)), with the identifications $\mathbf{S}_0 \equiv \mathbf{C}^\dagger \mathbf{C}$, $\mathbf{U} \equiv i\omega\mu\mathbf{I}$, $\mathbf{V} \equiv \mathbf{I}$ we find the formal expression

$$\mathbf{P} = diag(-i\omega\mu\mathbf{e}_0)\Pi_{\mathbf{m}_0} = diag(-i\omega\mu\mathbf{e}_0)\mathbf{W}[diag(\exp(\mathbf{m}_0))], \tag{16}$$

where $\mathbf{W}$ maps from the model parameter to the primary grid. Note however, that for the MT case the background *solution vector* $\mathbf{e}_0 = (\mathbf{e}_{0,1}, \mathbf{e}_{0,2})$ contains FD solutions for two source polarizations, and the product $\mathbf{P}\delta\mathbf{m}$ computed by Pmult actually results in a *source vector* object which encapsulates the forcing for a pair of forward problems, computed by first evaluating $\mathbf{W}[diag(\exp(\mathbf{m}_0))]\delta\mathbf{m}$ followed by pointwise multiplication by $-i\omega\mu\mathbf{e}_{0,k}, k = 1, 2$. For PmultT, the input is a *solution vector* $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$. Now we first compute $i\omega\mu(diag[\mathbf{e}_{0,1}]\mathbf{e}_1 + diag[\mathbf{e}_{0,2}]\mathbf{e}_2)$, and then apply the adjoint model parameter mapping $\Pi_{\mathbf{m}_0}^T$, resulting in a model parameter object. Implementation of these Solver Sensitivity

routines is basically the same for the CSEM case, but with the simplification of a single polarization in *source* and *solution vectors*.

7.3. Application to 3D MT and CSEM problems

We illustrate some basic capabilities of ModEM by using the same Numerical Discretization modules to set up a single synthetic example for 3D MT and 3D CSEM problems and illustrate the relative resolution of the two methods.

Our synthetic model for land MT and CSEM is loosely inspired by the Commer and Newman (2009) marine synthetic example in that it is a vertical superposition of horizontally folded structures, designed to showcase the relative strengths and weaknesses of the two methods. It also mimics the kind of contrasts encountered in a real land electromagnetic survey setting. In this 3D model, the synthetic upper and lower conductive folds extend along the strike, while the thin resistive structure is fully three-dimensional, hosted in the upper fold with a length of 3 km along the strike.

The model domain is discretized into $50 \times 50 \times 50$ cells, with horizontal cell dimensions of 500 m^2, logarithmically increasing towards the edges. Cell thicknesses are 50 m in the upper 1000 m of the model, then increasing logarithmically downwards by a factor of 1.2, to the depth of 160 km.

The full MT impedance tensor is sampled at 182 sites (Fig. 4; black, yellow and green circles) distributed on a regular 2D array with site spacing of 1 km in both x and y directions. The data are
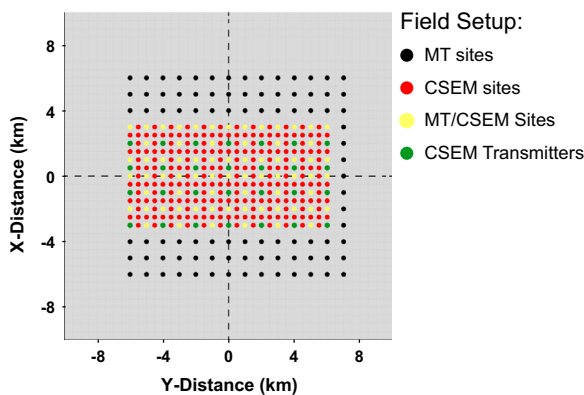

Fig. 4. The field setups used to compute the synthetic data. Black dots correspond to MT sites and red dots to CSEM receiver locations (RX), placed in the vicinity of the shallow resistive block, yellow dots serve as both MT and CSEM receivers. Green dots correspond to the 28 electrical point dipole transmitters (TX) used to compute the CSEM model responses; they also double as MT sites and CSEM receivers. There are 182 MT sites and 325 CSEM receivers, total. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

computed at 17 periods, distributed logarithmically between 0.1 s and 1000 s, with 4 periods per decade. An error floor of 2% of $|Z_{xy} Z_{yx}|^{1/2}$ is assigned to the full impedance tensor.

A dense receiver spacing is assumed for controlled-source EM (Fig. 4; 500 m spacing, denoted by red circles), including a subset of the MT sites (yellow circles) located in the vicinity of the shallow resistive block which also serve as receivers (RX) for CSEM.

The CSEM data are computed using 28 electrical point-dipole transmitters (TX; denoted by green circles in Fig. 4) operated at 5 periods (0.1, 0.5, 1, 5 and 10 s). The in-line (E_y) component of the electric field is computed at each of the receivers. An error floor of 2% of $|E_y|$ is assigned to the noise free E_y data.

For 3D inversion of the MT data, we used the ModEM version of NLCG (Fig. A1). A homogenous model with 100 Ωm resistivity was used as prior and starting model. In total, 25 NLCG iterations were required to decrease the nRMS value (normalized Root Mean Square) from ~ 23 down to 1.01. The computation was parallelized over 35 Intel processors, resulting in the run time for each NLCG iteration (comprising 2 forward modeling and 1 gradient computation) of ~ 2 min.

The 3D CSEM inversion also used a homogeneous 100 Ωm halfspace for prior and starting models. We found that 325 NLCG iterations were required to decrease the nRMS from ~ 101 to ~ 2.5, using 141 processors in parallel.

Results for the 2 cases are presented in Fig. 5, clearly illustrating the relative inability of the MT method to resolve this resistive layer. The shallow resistor is much clearer in the CSEM inversion. Conversely, MT is better at imaging the shape of the deeper conductor even in the middle of the array, where the MT site locations are sparser compared to the CSEM receiver coverage. We conclude that a joint MT-CSEM inversion would be advantageous to harness the relative strengths of the two methods.

As can be seen from the convergence summary plots (Fig. 6), even when the same synthetic model is employed and the same NLCG inversion procedure applied for MT and for CSEM configurations, convergence behavior is very different for these two problems. This illustrative example shows that in a practical joint MT and CSEM inversion, some care is require to balance fits to the two data types. Although implementation of joint MT and CSEM inversion is in principle straightforward in ModEM (Meqbel and Ritter, 2013), it is outside the scope of this paper.

8. Discussion and conclusions

ModEM has already proved to be a capable and efficient program for practical 3D MT inversion (Kelbert et al., 2012; Tietze and Ritter, 2013; Meqbel et al., 2014). However, the real power of ModEM lies in its potential as a tool for research, and for rapid development of new
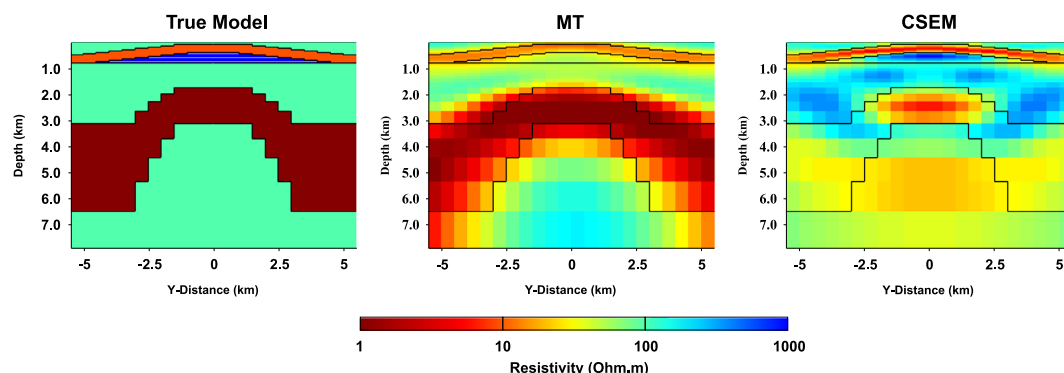

Fig. 5. Center cross-sections of the inversion results of fitting the MT (middle column) and CSEM (right column) synthetic data. For comparison purposes the true model is plotted on the left.
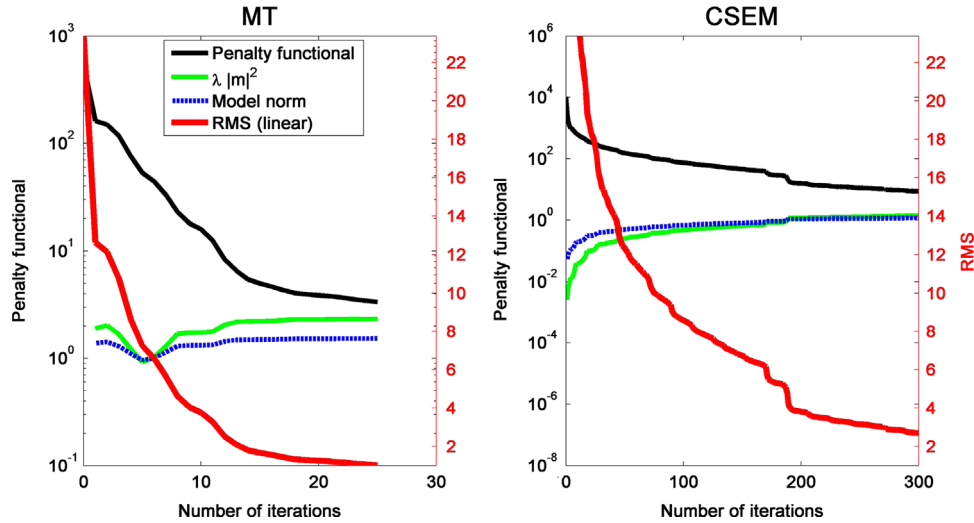
Fig. 6. NLCG algorithm convergence for MT and CSEM configurations. Both the total penalty functional (logarithmic scale in black) and the RMS value (linear scale in red) are plotted. Note the differences in both the run times and magnitudes, and overall behavior (steeper and smoother for the MT problem, likely because the same regularization has a stronger effect on MT than on CSEM). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

applications. Translating pseudo-code of the sort shown in Figs. A1 and A2 into a ModEM inversion module is simple, since all required operators and data objects are readily available. Thus, it would be easy to replace the NLCG scheme we have implemented with quasi-Newton (e.g., Newman and Boggs, 2004), or standard Gauss–Newton variants such as OCCAM (Constable et al., 1987; Siripunvaraporn et al., 2005), as well as more novel schemes such as the multi-frequency hybrid inversion of Egbert (2012). ModEM thus provides a natural test-bed for development, refinement, and comparison of inversion search algorithms. Similarly, ModEM offers potentially great flexibility in defining the model parametrization and regularization, simplifying new developments and extensions in this critical direction, e.g., to allow bounds on conductivities (Avdeev and Avdeeva, 2009), to allow simultaneous inversion for near-surface distortion parameters (e.g., DeGroot-Hedlin and Constable, 2004), or to decouple the model parameter and numerical solver grids (e.g., Commer and Newman, 2009).

In some respects ModEM has been designed specifically for frequency domain EM geophysical problems – e.g., the structure of data vectors reflects the transmitter/receiver structure common to such methods. However, ModEM should be more broadly useful, in particular providing a good starting point for development of joint inversion schemes. For example, ModEM's data space structure already supports multiple data types, each of which may require solution of a different forward problem – e.g., as for the 2D MT problem discussed in EK12, where TE and TM mode data are inverted simultaneously. Joint inversion of EM and other sorts of geophysical data (e.g., seismic or gravity) could be accommodated in a similar manner. In this case there might be two separate sets of numerical discretization modules, with possibly different grids, numerics, interpolation functionals, etc (see also Section 7). The interface routines would then merge and encapsulate these, hiding the details, and even the multiplicity of distinct physics involved. The two sorts of data would of course have to be coupled through the model parameter, either through some sort of constitutive relation, or through the covariance.

Of course, there will always be limits to the generality of any specific implementation. For example, the "generic" inversion algorithms implemented so far assume that the model regularization can be represented as a norm on the model space, defined by a quadratic form $\mathbf{C_m}$. More general regularization schemes would require some modification to these inversion modules, although these could be implemented rather simply, making use of lower level components. Modularity and code reuse in ModEM thus functions at multiple levels, from the numerical discretization to implementation of specific inversion schemes.

Appendix A. Modular linearized inversion algorithms

A wide range of gradient based inversion algorithms can be implemented using the components described above. As examples we summarize two schemes we have implemented: non-linear conjugate gradients (NLCG, e.g., Press et al., 1992), and the data-space conjugate gradient scheme (DCG) of Siripunvaraporn and Egbert (2007).

For NLCG, we adopted the standard Polak–Ribière scheme (see Fig. A1 for pseudo-code). For the line search, which is based on Press et al. (1992) and Nocedal and Wright (2006, Chapter 3), we first interpolate using a quadratic approximation; if the solution does not satisfy the sufficient decrease (Armijo) condition (ignoring the curvature condition), we backtrack using cubic interpolation. This strategy only requires one gradient evaluation and is efficient when such computations are expensive. The initial step size is set outside of the line search, and adjusted automatically based on the first gradient computation; it can also be specified by the user. We have also implemented an automatic criterion to decrease the damping parameter (ν, see EK12) when convergence of the inversion stalls. User control of the inversion includes specifying the initial damping parameter, the sufficient decrease condition, and the stopping criterion.

The DCG scheme of Siripunvaraporn and Egbert (2007) is an iterative Gauss–Newton algorithm. Briefly, the penalty functional is linearized about a trial value of the model parameter $\mathbf{m}_n$, and the minimum of the linearized functional is sought. This leads to a system of normal equations involving cross-products of $\mathbf{J}$ solved for $\mathbf{b}_n$ with conjugate gradients (without actually forming the normal equations). In the data-space variant we consider here,

$\mathbf{m}_{prior}$ = prior model
ν = initial damping parameter
a = initial line search step α scaled by model norm
$n = 0$
$\tilde{\mathbf{m}}_0 = 0$
$\mathbf{m}_0 = \mathbf{C_m}^{1/2}\tilde{\mathbf{m}}_0 + \mathbf{m}_{prior}$ = starting model
Evaluate $\mathcal{P}_0 = (\mathbf{d} - f(\mathbf{m}_0))^T\mathbf{C_d}^{-1}(\mathbf{d} - f(\mathbf{m}_0)) + \nu\tilde{\mathbf{m}}_0^T\tilde{\mathbf{m}}_0$
Evaluate $\mathcal{P}'(\tilde{\mathbf{m}})|_{\tilde{\mathbf{m}}_0} = -2\mathbf{C_m}^{1/2}\mathbf{J}^T\mathbf{C_d}^{-1}(\mathbf{d} - f(\mathbf{m}_0)) + 2\nu\tilde{\mathbf{m}}_0$
$\mathbf{g}_0 = -\mathcal{P}'(\tilde{\mathbf{m}})|_{\tilde{\mathbf{m}}_0}$
$\mathbf{h}_0 = \mathbf{g}_0$
Set $\alpha_0 = a/\mathbf{g}_0^T\mathbf{g}_0$
While $(\mathbf{d} - f(\mathbf{m}_n))^T\mathbf{C_d}^{-1}(\mathbf{d} - f(\mathbf{m}_n)) > T$ do
        $n = n + 1$
        Line search: choose $\hat{\alpha}$ to minimize $\mathcal{P}(\mathbf{C_m}^{1/2}(\tilde{\mathbf{m}}_{n-1} + \alpha\mathbf{h}_{n-1}) + \mathbf{m}_{prior})$
        $\tilde{\mathbf{m}}_n = \tilde{\mathbf{m}}_{n-1} + \hat{\alpha}\mathbf{h}_{n-1}$
        $\mathcal{P}_n = (\mathbf{d} - f(\mathbf{m}_n))^T\mathbf{C_d}^{-1}(\mathbf{d} - f(\mathbf{m}_n)) + \nu\tilde{\mathbf{m}}_n^T\tilde{\mathbf{m}}_n$
        $\mathbf{g}_n = -\mathcal{P}'(\tilde{\mathbf{m}})|_{\tilde{\mathbf{m}}_n}$
        $\alpha = 2(\mathcal{P}_n - \mathcal{P}_{n-1})/\mathbf{g}_{n-1}^T\mathbf{h}_{n-1}$
        Adjust α for super-linear convergence: $\alpha = 1.01*\alpha$
        If α too small
                $\nu = 0.1*\nu$
                $\alpha = a/\mathbf{g}_n^T\mathbf{g}_n$
                Restart: $\mathbf{h}_n = \mathbf{g}_n$
                Cycle do loop
        End if
        $\beta = \mathbf{g}_n^T(\mathbf{g}_n - \mathbf{g}_{n-1})/\mathbf{g}_{n-1}^T\mathbf{g}_{n-1}$
        If $(\mathbf{g}_n^T\mathbf{g}_n + \beta\mathbf{g}_n^T\mathbf{h}_{n-1} > 0)$
                $\mathbf{h}_n = \mathbf{g}_n + \beta\mathbf{h}_{n-1}$
        Else
                Restart to restore orthogonality: $\mathbf{h}_n = \mathbf{g}_n$
        End if
End do

Fig. A1. Pseudo-code for non-linear conjugate gradient (NLCG) algorithm. Notation: $f(\mathbf{m})$ represents responses obtained from forward modeling; $\mathcal{P}_n$ and $\mathcal{P}'_n$ are values of the penalty functional and its derivative at nth iteration; $\mathbf{m}_n$, $\mathbf{h}_n$, $\mathbf{g}_n$ are vectors in the model space; α and β represent real scalars; other symbols are as in the text.

$\mathbf{m}_{prior}$ = prior model
$\mathbf{m}_0$ = starting model
Outer loop: For $n = 0, 1, 2, \ldots$
        $\hat{\mathbf{d}}_n = \mathbf{d} - f(\mathbf{m}_n) + \mathbf{J}[\mathbf{m}_n - \mathbf{m}_{prior}]$
        Solve: $[\mathbf{C_d}^{-1/2}\mathbf{J}\mathbf{C_m}\mathbf{J}^T\mathbf{C_d}^{-1/2} + \nu\mathbf{I}]\mathbf{b}_n = \hat{\mathbf{d}}_n$ using CG
        $\mathbf{x}_0 = 0$
        $\mathbf{r}_0 = \mathbf{p}_0 = \hat{\mathbf{d}}_n$
        Inner loop (CG): For $k = 1, 2, \ldots$
                $\mathbf{y}_k = \mathbf{C_d}^{-1/2}\mathbf{J}\mathbf{C_m}\mathbf{J}^T\mathbf{C_d}^{-1/2}\mathbf{p}_k + \nu\mathbf{p}_k$
                $\alpha_k = \mathbf{r}_k^T\mathbf{r}_k/\mathbf{p}_k^T\mathbf{y}_k$
                $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$
                $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{y}_k$
                If $||\mathbf{r}_{k+1}|| < r_{tol}$
                        $\mathbf{b}_n = \mathbf{x}_{k+1}$
                        Exit
                Else
                        $\beta_k = ||\mathbf{r}_{k+1}||/||\mathbf{r}_k||$
                        $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} - \beta_k\mathbf{p}_k$
                End if
        End inner loop
        $\mathbf{m}_{n+1} = \mathbf{m}_{prior} + \mathbf{C_m}\mathbf{J}^T\mathbf{C_d}^{-1/2}\mathbf{b}_n$
        If $(\mathbf{d} - f(\mathbf{m}_{n+1}))^T\mathbf{C_d}^{-1}(\mathbf{d} - f(\mathbf{m}_{n+1})) < T$ exit
End outer loop

Fig. A2. Pseudo-code for data space conjugate gradient (DCG) algorithm, with normal equations solved in the data space. Notation: $f(\mathbf{m})$ represents responses obtained from forward modeling; $\hat{\mathbf{d}}_n, \mathbf{b}_n, \mathbf{x}_k, \mathbf{y}_k, \mathbf{p}_k$ and $\mathbf{r}_k$ are all vectors in the data space; α_k and β_k represent real scalars; other symbols are as in the text.

these equations are

$$(\mathbf{J}\mathbf{C_m}\mathbf{J}^T + \nu\mathbf{I})\mathbf{b}_n = \hat{\mathbf{d}}_n \qquad (A.1)$$

$$\mathbf{m}_{n+1} = \mathbf{C_m}\mathbf{J}^T\mathbf{b}_n. \qquad (A.2)$$

In the DCG algorithm Eq. (A.1) is solved for $\mathbf{b}_n$ using conjugate gradients, and the model update is computed as in (A.2). The whole procedure is iterated over n to achieve the desired level of misfit. Pseudo-code for this scheme is provided in Fig. A2. Choice of the regularization parameter, and stopping criteria for the conjugate gradient iterations are discussed in Siripunvaraporn and Egbert (2007). For sensitivity computations we use multiplication routines `Jmult` and `JmultT`, described in Table 2.

References

Akin, J.E., 2003. Object-Oriented Programming via Fortran 90/95 Volume 1317, Issue 1. Cambridge University Press, Cambridge, UK

Alumbaugh, D.L., Newman, G.A., Prevost, L., Shadid, J.N., 1996. Three-dimensional wide band electromagnetic modeling on massively parallel computers. Radio Sci. 33, 1–23.

Avdeev, D.B., 2005. Three-dimensional electromagnetic modelling and inversion from theory to application. Surv. Geophys. 26 (November (6)), 767–799.

Avdeev, D.B., Avdeeva, A., 2009. 3D magnetotelluric inversion using a limited-memory quasi-Newton optimization. Geophysics 74 (3), F45–F57.

Commer, M., Newman, G.A., 2009. Three-dimensional controlled-source electromagnetic and magnetotelluric joint inversion. Geophys. J. Int. 178 (September (3)), 1305–1316, url: ⟨http://doi.wiley.com/10.1111/j.1365-246X.2009.04216.x⟩.

Constable, S.C., Parker, R.L., Constable, C.G., 1987. Occam's inversion: a practical algorithm for generating smooth models from electromagnetic sounding data. Geophysics 52 (March (3)), 289–300.

DeGroot-Hedlin, C., Constable, S.C., 2004. Inversion of magnetotelluric data for 2D structure with sharp resistivity contrasts. Geophysics 69 (January (1)), 78, url: ⟨http://geophysics.geoscienceworld.org/cgi/content/full/69/1/78⟩.

Egbert, G.D., 2012. Hybrid conjugate gradient-Occam algorithms for inversion of multi-frequency and multi-transmitter EM data. Geophys. J. Int. 190, 255–266.

Egbert, G.D., Kelbert, A., 2012. Computational recipes for electromagnetic inverse problems. Geophys. J. Int. 189, 167–251.

Haber, E., Ascher, U.M., Oldenburg, D.W., 2004. Inversion of {3D} electromagnetic data in frequency and time domain using an inexact all-at-once approach. Geophysics 69 (5), 1216–1228.

Kelbert, A., Egbert, G.D., DeGroot-Hedlin, C., 2012. Crust and upper mantle electrical conductivity beneath the Yellowstone Hotspot Track. Geology 40 (March (5)), 447–450, url: ⟨http://geology.gsapubs.org/cgi/doi/10.1130/G32655.1⟩.

Kelbert, A., Egbert, G.D., Schultz, A., 2008. Non-linear conjugate gradient inversion for global EM induction: resolution studies. Geophys. J. Int. 173 (2), 365–381.

Key, K., 2009. 1D inversion of multicomponent, multifrequency marine CSEM data: methodology and synthetic studies for resolving thin resistive layers. Geophysics 74 (2), F9–F20.

McGillivray, P.R., Oldenburg, D.W., Ellis, R.G., Habashy, T.M., 1994. Calculation of sensitivities for the frequency-domain electromagnetic problem. Geophys. J. Int. 116 (January (1)), 1–4, url: ⟨http://doi.wiley.com/10.1111/j.1365-246X.1994.tb02121.x⟩.

Meqbel, N., November 2009. The Electrical Conductivity Structure of the Dead Sea Basin Derived from 2D and 3D Inversion of Magnetotelluric Data (Ph.D. thesis). Free University of Berlin, Berlin, Germany.

Meqbel, N., G. D. Egbert, P. E. Wannamaker, A. Kelbert and A. Schultz (2014), "Deep electrical resistivity structure of the Pacific Northwestern U. S. derived from 3-D inversion of USArray Magnetotelluric data", Earth and Planetary Science Letters, Available online 22 January 2014, ISSN 0012-821X, http://dx.doi.org/10.1016/j.epsl.2013.12.026.

Meqbel, N., Ritter, O., 2013. New Advances for a joint 3D inversion of multiple EM methods. In: Extended Abstracts for the 5th International Symposium on Three-Dimensional Electromagnetics, May 7–9, 2013, Sapporo, Japan.

Moorkamp, M., Heincke, B., Jegen, M., Roberts, A.W., Hobbs, R.W., 2011. A framework for 3-D joint inversion of MT, gravity and seismic refraction data. Geophys. J. Int. 184 (January (1)), 477–493, url: ⟨http://doi.wiley.com/10.1111/j.1365-246X.2010.04856.x⟩.

Newman, G.A., Alumbaugh, D.L., 1997. Three-dimensional massively parallel electromagnetic inversion – I. Theory. Geophys. J. Int. 128 (February), 345–354.

Newman, G.A., Alumbaugh, D.L., 2000. Three-dimensional magnetotelluric inversion using non-linear conjugate gradients. Geophys. J. Int. 140, 410–424.

Newman, G.A., Boggs, P.T., 2004. Solution accelerators for large-scale three-dimensional electromagnetic inverse problems. Inverse Probl. 20 (December), 151–170.

Newman, G.A., Hoversten, G.M., 2000. Solution strategies for two- and three-dimensional electromagnetic inverse problems. Inverse Probl. 16, 1357–1375.

Nocedal, J., Wright, S.J., 2006. Numerical Optimization. Springer, New York, USA

Parker, R.L., 1994. Geophysical Inverse Theory. Princeton University Press, Princeton, New Jersey

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., 1992. Numerical Recipes in Fortran 77 – The Art of Scientific Computing, 2nd edition Cambridge University Press, Cambridge, UK

Purser, R.J., Wu, W.-S., Parrish, D.F., Roberts, N.M., 2003a. Numerical aspects of the application of recursive filters to variational statistical analysis. Part I: spatially

homogeneous and isotropic Gaussian covariances. Mon. Weather Rev. 131 (8), 1524–1535.

Purser, R.J., Wu, W.-S., Parrish, D.F., Roberts, N.M., 2003b. Numerical aspects of the application of recursive filters to variational statistical analysis. Part II: spatially inhomogeneous and anisotropic general covariances. Mon. Weather Rev. 131 (8), 1536–1548.

Rodi, W.L., Mackie, R.L., 2001. Nonlinear conjugate gradients algorithm for 2-D magnetotelluric inversion. Geophysics 66 (1), 174–187, url: ⟨http://link.aip.org/link/?GPY/66/174/1⟩.

Siripunvaraporn, W., 2011. Three-dimensional magnetotelluric inversion: an introductory guide for developers and users. Surv. Geophys. 33 (May (1)), 5–27, url: ⟨http://www.springerlink.com/index/10.1007/s10712-011-9122-6⟩.

Siripunvaraporn, W., Egbert, G.D., 2007. Data space conjugate gradient inversion for 2-D magnetotelluric data. Geophys. J. Int. 170 (September), 986–994.

Siripunvaraporn, W., Egbert, G.D., 2009. WSINV3DMT: vertical magnetic field transfer function inversion and parallel implementation. Phys. Earth Planet. Inter. 173 (April (3–4)), 317–329, url: ⟨http://linkinghub.elsevier.com/retrieve/pii/S0031920109000144⟩.

Siripunvaraporn, W., Egbert, G.D., Lenbury, Y., 2002. Numerical accuracy of magnetotelluric modeling: a comparison of finite difference approximations. Earth Planets Space 54 (6), 721–725.

Siripunvaraporn, W., Egbert, G.D., Lenbury, Y., Uyeshima, M., 2005. Three-dimensional magnetotelluric inversion: data-space method. Phys. Earth Planet. Inter. 150 (1–3), 3–14.

Smith, J.T., 1996. Conservative modeling of 3-D electromagnetic fields: 2. biconjugate gradient solution and an accelerator. Geophysics 61 (5), 1319–1324.

Stefano, M.D., Andreasi, G.F., Re, S., Virgilio, M., Snyder, F.F., 2011. Multiple-domain, simultaneous joint inversion of geophysical data with application to subsalt imaging. Geophysics 76 (3), R69–R80.

Tarantola, A., 2005. Inverse Problem Theory and Methods for Model Parameter Estimation. SIAM, Philadelphia, PA, USA

Tietze K. and Ritter, O.. Three-dimensional magnetotelluric inversion in practice–the electrical conductivity structure of the San Andreas Fault in Central California. Geophys. J. Int. 195 (1), 2013, 130-147.

Yee, K., May 1966. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. IEEE Trans. Antennas Propag. 14, 302–307.

Zhdanov, M., Hursan, G., 2000. 3D electromagnetic inversion based on quasi-analytical approximation. Inverse Probl. 16, 1297–1322.

Zhdanov, M.S., 2002. Geophysical Inverse Theory and Regularization Problems (Methods in Geochemistry and Geophysics). Elsevier Science, Amsterdam, The Netherlands