Computers & Geosciences 104 (2017) 1–11

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Case study

A MATLAB based 3D modeling and inversion code for MT data

Arun Singh*, Rahul Dehiya, Pravin K. Gupta, M. Israil

Department of Earth Sciences, Indian Institute of Technology Roorkee, Roorkee 247667, India

ARTICLE INFO

Keywords:
Magnetotelluric
MATLAB
Numerical modeling
Inversion
Parallelization

ABSTRACT

The development of a MATLAB based computer code, *AP3DMT*, for modeling and inversion of 3D Magnetotelluric (MT) data is presented. The code comprises two independent components: grid generator code and modeling/inversion code. The grid generator code performs model discretization and acts as an interface by generating various I/O files. The inversion code performs core computations in modular form – forward modeling, data functionals, sensitivity computations and regularization. These modules can be readily extended to other similar inverse problems like Controlled-Source EM (CSEM). The modular structure of the code provides a framework useful for implementation of new applications and inversion algorithms. The use of MATLAB and its libraries makes it more compact and user friendly. The code has been validated on several published models. To demonstrate its versatility and capabilities the results of inversion for two complex models are presented.

1. Introduction

Several 3D electromagnetic data inversion approaches and parallel computer codes were developed during the first decade (notably, Newman and Alumbaugh (2000), Zhdanov and Hursan (2000), Haber et al. (2004), Newman and Boggs (2004), Siripunvaraporn and Egbert (2009), Avdeev and Avdeeva (2009), etc.). Most of these codes were focused on inversion with a particular computational approach. Recently, Egbert and Kelbert (2012) and Kelbert et al. (2014) presented a Fortran code, *ModEM*, with its emphasis on modular implementation of the basic components of inversion - forward modeling, model parametrization and regularization, data functionals, sensitivity computation and inversion algorithms which are reusable and readily extensible. This and most of the previous codes are written in Fortran programming language. Many of these algorithms permit interchangeability and re-usability of various subprograms thereby providing a code base for development of new inversion algorithms. However, it is difficult for a new researcher to make desired changes for experimentation. MATLAB provides a solution to this problem and we have developed the present code, *AP3DMT*, on MATLAB platform. *AP3DMT* provides a basic framework for 3D modeling and inversion which is flexible for rapid development and experimentation with different schemes of inversion, parametrization and regularization. This code will provide a basic framework to researchers who are conversant with MATLAB and willing to use or develop its capabilities for future applications.

In the present paper, main focus is on program structure with some basic mathematical formulae required for implementation. Section 2 provides the basic theory for forward modeling and inversion along with main components for sensitivity computations. For detailed discussion one can refer (Egbert and Kelbert, 2012). The program is divided into two parts. The first part includes grid generator and I/O subprograms (functions in MATLAB) while the second part includes modeling/inversion. Section 3 describes the overview of the program including grid generator for common models with cuboids and/or polyhedron shaped target bodies and forward/inverse modeling. The main feature of grid generator is its robustness in handling complex geological features needed to simulate responses for complex 3D structures. Such an efficient grid generator is useful when performing block inversion (Singh et al., 2014). A detailed description and implementation of inversion and sensitivity computations is also provided in this section. These subprograms, along with a coarse grained parallelization (Section 4), can be directly used for a wide range of problems. The key feature of *AP3DMT* is its capability to accommodate modifications in the forward problem (e.g., different forward solver, different types of responses, model parameters, etc.) and their easy implementation without any modifications in other subprograms like, inversion subprogram. In Section 5, we provide details about the actual computations for both forward problem (i.e. forward solutions, evaluation of EM field components, etc.) and sensitivity computations. In Section 6, we demonstrate the code versatility for grid generation and inversion with the help of two synthetic examples.

* Corresponding author.
E-mail address: arunsingh2626@gmail.com (A. Singh).

http://dx.doi.org/10.1016/j.cageo.2017.03.019
Received 19 February 2016; Received in revised form 22 March 2017; Accepted 24 March 2017
Available online 27 March 2017

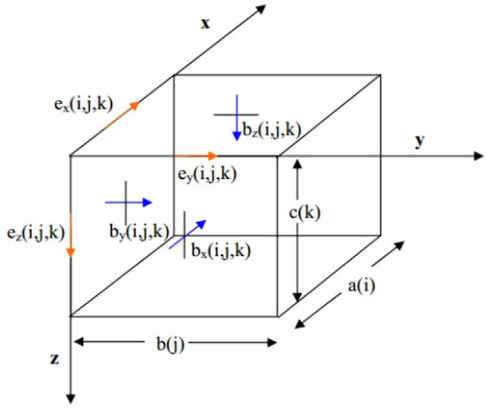

Fig. 1. Staggered finite difference grid for the 3D MT forward problem. Since the PDE is formulated in terms of electric field components, these are defined on cell edges and the magnetic field components are defined on center of the cell faces.

2. Theory

2.1. Forward modeling

The electric field **E** are simulated by solving the vector Helmholtz equation, given below, in frequency domain (assuming $e^{i\omega t}$ time dependence),

$$\nabla \times \nabla \times \mathbf{E} + i\omega\mu\sigma\mathbf{E} = 0, \tag{1}$$

where ω is the angular frequency, μ is the magnetic permeability and σ is the conductivity. Magnetic fields corresponding to an electric field solution **E** can be expressed as $\mathbf{H} = (-i\omega\mu)^{-1}\nabla \times \mathbf{E}$.

(1) is approximated on a staggered-grid (Yee, 1966), as shown in Fig. 1, using finite difference (FD) formulation. The linear system, for frequency *f*, resulting after symmetric scaling can be expressed as

$$\mathbf{A_m e} = \mathbf{s}, \tag{2}$$

where $\mathbf{A}_m$ is a frequency dependent $N_e \times N_e$ sparse symmetric complex matrix with 13 non-zero elements per row; **s** is the N_e dimensional source vector and **e** is the N_e dimensional vector representing electric fields at the N_e internal nodes. This sparse linear system is solved iteratively using bi-conjugate gradient stabilized (BiCGSTAB) scheme which belongs to a class of Krylov subspace techniques. The incomplete **LU** decomposition of the diagonal sub-block matrix (Mackie et al., 1994) is used for pre-conditioning.

For stable and accurate solutions of (2) at low frequencies, a divergence correction is periodically applied (Smith, 1996) by solving a Poisson-like equation with pre-conditioned conjugate gradients (CG). Smith (1996) has shown that the convergence is improved by applying divergence correction and it thereby significantly reduces the computational time needed for solution of (2).

2.2. Inversion

All the inversion algorithms aim at finding a meaningful model $\widetilde{\mathbf{m}}$, a *M*- dimensional model parameter vector, while fitting the data $\mathbf{d}^{obs}$ of dimension N_d to an acceptable level in a stable manner. We consider minimization of the penalty functional defined as

$$\phi(\widetilde{\mathbf{m}}, \mathbf{d}^{obs}) = (\mathbf{d}^{obs} - \widetilde{F}(\widetilde{\mathbf{m}}))^T \mathbf{C_d}^{-1}(\mathbf{d}^{obs} - \widetilde{F}(\widetilde{\mathbf{m}})) + \lambda(\widetilde{\mathbf{m}} - \mathbf{m_0})^T \mathbf{C_m}^{-1}(\widetilde{\mathbf{m}} - \mathbf{m_0}), \tag{3}$$

where $\widetilde{F}(\widetilde{\mathbf{m}})$ is the forward mapping, $\mathbf{C_d}$ is the data covariance matrix, $\mathbf{m_0}$ is the apriori model, $\mathbf{C_m}$ is the model covariance matrix or regularization term and λ is the trade-off parameter. $\mathbf{C_d}$ is generally diagonal hence, it can be eliminated from definition of penalty functional by simply rescaling of data and forward mapping. Both $\mathbf{m_0}$ and $\mathbf{C_m}$ can also be eliminated from (3) by setting $\mathbf{m} = \mathbf{C_m}^{-1/2}(\widetilde{\mathbf{m}} - \mathbf{m_0})$.

This transformation reduces (3) to

$$\phi(\mathbf{m}, \mathbf{d}^{obs}) = (\mathbf{d}^{obs} - F(\mathbf{m}))^T(\mathbf{d}^{obs} - F(\mathbf{m})) + \lambda\mathbf{m}^T\mathbf{m}, \tag{4}$$

where, $F(\mathbf{m}) = \widetilde{F}(\mathbf{C_m}^{1/2}\mathbf{m} + \mathbf{m_0})$. After minimizing (4) in transform domain the model parameters are transformed back into the space of the original model parameter $\widetilde{\mathbf{m}} = \mathbf{C_m}^{1/2}\mathbf{m} + \mathbf{m}_0$. For details one can refer to Kelbert et al. (2008). We have implemented both quasi-linear inversion and non-linear inversion using conjugate gradient for the minimization of (4).

2.2.1. Quasi-linear inversion

In this approach, the objective functional is first approximated by a Taylor series expansion. The quadratic approximation of the objective functional is then minimized to produce a series of the updated models. In the Gauss Newton (*GN*) method only the first derivative in the Hessian matrix of Newton's method is retained but the second-order derivative is discarded. This leads to an iterative sequence of approximate solutions as,

$$(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})\delta\mathbf{m}_n = \mathbf{J}^T\mathbf{r} - \lambda\mathbf{m}_n, \tag{5}$$

where $\mathbf{m}_n$ are the model parameters at the n^{th} iteration, **J** is Jacobian and $\mathbf{r} = \mathbf{d}^{obs} - F(\mathbf{m}_n)$ is the residual. (5) is solved for $\delta\mathbf{m}_n$ and the new updated model parameter vector is obtained as $\mathbf{m}_{n+1} = \mathbf{m}_n + \delta\mathbf{m}_n$. For stability this linearized scheme generally requires some form of step length damping (Marquardt, 1963; Rodi and Mackie, 2001). Alternatively, instead of solving for $\delta\mathbf{m}$ one can solve for $\mathbf{m}_{n+1}$ using Occam approach (Constable et al., 1987; Parker, 1994). In Occam's algorithm (5) is written as

$$(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})\mathbf{m}_{n+1} = \mathbf{J}^T\hat{\mathbf{d}}, \tag{6}$$

where $\hat{\mathbf{d}} = \mathbf{d}^{obs} - F(\mathbf{m}) + \mathbf{J}\mathbf{m}_n$. In data space, (Siripunvaraporn and Egbert, 2000; Siripunvaraporn et al., 2005) the solution of (6) is written as

$$\mathbf{m}_{n+1} = \mathbf{J}^T\mathbf{b}_n; \quad (\mathbf{J}\mathbf{J}^T + \lambda\mathbf{I})\mathbf{b}_n = \hat{\mathbf{d}}. \tag{7}$$

To avoid explicit computation and storage of **J**, (6) and (7) are solved with a memory efficient Krylov subspace iterative solver such as conjugate gradients (*CG*). In this approach, the product of matrix and an arbitrary vector such as $(\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})\mathbf{m}$ is computed and this can be performed at the cost of just two forward problems. Following Newman and Alumbaugh (1997), at the *n*th inversion iteration, the regularization parameter λ for *GN* is determined as $\lambda = r_{sum}/2^{n-1}$ where, r_{sum} is the largest row sum of *real* $(\mathbf{J}^T\mathbf{J})$.

2.2.2. Non-linear inversion using conjugate gradient

In this approach, (4) is directly minimized using a gradient based optimization technique like non-linear conjugate gradient (*NLCG*) (Rodi and Mackie, 2001; Newman and Boggs, 2004; Kelbert et al., 2014). Here, the gradient of (4) with respect to the variation in model parameter **m** is computed as,

$$\left.\frac{\partial\phi}{\partial\mathbf{m}}\right|_{\mathbf{m}_n} = -2\mathbf{J}^T\mathbf{r} + 2\lambda\mathbf{m}_n, \tag{8}$$

and it is used to a compute new 'conjugate' search direction. The 'line search' is used to minimize the penalty functional along this direction and it requires solving forward problem few times and the gradient is recomputed. Basic computational steps for *NLCG* include solving forward problem for model parameter $\mathbf{m}_n$ and multiplication of $\mathbf{J}^T$ by the residual **r**. However, for the regularization parameter approach described above, this scheme does not work because varying the regularization parameter would compromise the orthogonality of search directions (Egbert, 2012). Following Kelbert et al. (2008), NLCG iterations are performed for fixed value of λ and when misfit stalls i.e. difference between misfits of two previous iterations is less than a predefined threshold, λ is reduced by a predetermined factor (10