Computers & Geosciences 85 (2015) 142–154

Contents lists available at ScienceDirect

Computers & Geosciences

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

Review article

SimPEG: An open source framework for simulation and gradient based parameter estimation in geophysical applications

Rowan Cockett [a,*], Seogi Kang [a], Lindsey J. Heagy [a], Adam Pidlisecky [b], Douglas W. Oldenburg [a]

[a] Geophysical Inversion Facility, University of British Columbia, Canada
[b] University of Calgary, Canada

ARTICLE INFO

Article history:
Received 14 January 2015
Received in revised form
11 August 2015
Accepted 23 September 2015
Available online 26 September 2015

Keywords:
Geophysics
Numerical modeling
Inversion
Electromagnetics
Sensitivities
Object-oriented programming

ABSTRACT

Inverse modeling is a powerful tool for extracting information about the subsurface from geophysical data. Geophysical inverse problems are inherently multidisciplinary, requiring elements from the relevant physics, numerical simulation, and optimization, as well as knowledge of the geologic setting, and a comprehension of the interplay between all of these elements. The development and advancement of inversion methodologies can be enabled by a framework that supports experimentation, is flexible and extensible, and allows the knowledge generated to be captured and shared. The goal of this paper is to propose a framework that supports many different types of geophysical forward simulations and deterministic inverse problems. Additionally, we provide an open source implementation of this framework in Python called SimPEG (Simulation and Parameter Estimation in Geophysics, http://simpeg.xyz). Included in SimPEG are staggered grid, mimetic finite volume discretizations on a number of structured and semi-structured meshes, convex optimization programs, inversion routines, model parameterizations, useful utility codes, and interfaces to standard numerical solver packages. The framework and implementation are modular, allowing the user to explore, experiment with, and iterate over a variety of approaches to the inverse problem. SimPEG provides an extensible, documented, and well-tested framework for inverting many types of geophysical data and thereby helping to answer questions in geoscience applications. Throughout the paper we use a generic direct current resistivity problem to illustrate the framework and functionality of SimPEG.

Contents

* Corresponding author.
E-mail address: rcockett@eos.ubc.ca (R. Cockett).

http://dx.doi.org/10.1016/j.cageo.2015.09.015

1. Introduction and motivation

Geophysical surveys can be used to obtain information about the subsurface as the responses that are measured depend on the physical properties and contrasts in the earth. Inversions provide a mathematical framework for constructing physical property models consistent with the data collected by these surveys. The data collected are finite in number while the physical property distribution of the earth is continuous. Thus, inverting for a physical property model from geophysical data is an ill-posed problem, meaning that no unique solution explains the data. Furthermore, the data may be contaminated with noise. Therefore, the goal of a deterministic inversion is not only to find a model consistent with the data, but must be to find the 'best' model that is consistent with the data.[1] The definition of 'best' requires the incorporation of assumptions and *a priori* information, often in the form of an understanding of the particular geologic setting or structures (Constable et al., 1987; Oldenburg and Li, 2005; Lelièvre et al., 2009). Solving the inverse problem involves many moving pieces that must work together, including physical simulations, optimization, linear algebra, and incorporation of geology. Deterministic geophysical inversions have been extensively studied, and many components and methodologies have become standard practice. With increases in computational power and instrumentation quality, there is a greater drive to extract more information from the geophysical data. Additionally, geophysical surveys are being applied in progressively more challenging environments. As a result, the geosciences are moving towards the integration of geological, geophysical, and hydrological information to better characterize the subsurface (e.g. Haber and Oldenburg, 1997; Doetsch et al., 2010; Gao et al., 2012). This is a scientifically and practically challenging task (Li and Oldenburg, 2000b; Lelièvre et al., 2009). These challenges, compounded with inconsistencies between different data sets, often makes the integration and implementation complicated and/or non-reproducible. The development of new methodologies to address these challenges will build upon, as well as augment, standard practices; this presupposes that researchers have access to consistent, well-tested tools that can be extended, adapted and combined.

There are many proprietary codes available that focus on efficient algorithms and are optimized for a specific geophysical application (e.g. Kelbert et al., 2014; Li and Key, 2007; Li and Oldenburg, 1996b, 1998). These packages are effective for their intended application, for example, a domain specific large-scale geophysical inversion or a tailored industry workflow. However, many of these packages are 'black-box' algorithms, that is, they cannot easily be interrogated or extended. As researchers, we require the ability to interrogate and extend ideas; this must be afforded by the tools that we use. Accessibility and extensibility are the primary motivators for this work. Other disciplines have approached the development of these tools through open source initiatives using interpreted languages, such as Python, for example, Astropy in astronomy (Astropy Collaboration et al., 2013) and SciPy in numerical computing (Jones et al., 2001). Interpreted languages facilitate interactive development using scripting, visualization, testing, and interoperability with code in compiled languages and existing libraries. Furthermore, many open source initiatives have led to communities with hundreds of researchers contributing and collaborating using social coding platforms, such as GitHub (https://github.com). There are also initiatives in the geophysical forward and inverse modeling community targeting specific geophysical applications (cf. Hansen et al., 2013; Hewett and Demanet, 2013; Uieda et al., 2014; Kelbert et al., 2014; Harbaugh, 2005). We are interested in creating a community around geophysical simulations and gradient based inversions. To create a foundation on which to build a community, we require a comprehensive framework that is applicable across domains and upon which researchers can readily develop their own tools and methodologies. To support these goals, this framework must be modular and its implementation must be easily extensible by researchers.

The goal of this paper is to present a comprehensive framework for simulation and gradient based parameter estimation in geophysics. The core ideas from a variety of geophysical inverse problems have been distilled to create this framework. We also provide an open source library written in Python called SimPEG (Simulation and Parameter Estimation in Geophysics, http://github.com/simpeg/simpeg). Our implementation has core dependencies on SciPy, NumPy, and Matplotlib, which are standard scientific computing packages in Python (Jones et al., 2001; Van Rossum and Drake, 1995; Oliphant, 2007; Hunter, 2007). SimPEG includes staggered grid, mimetic finite volume discretizations on structured and semi-structured meshes. It interfaces to standard numerical solver packages, convex optimization algorithms, model parameterizations, and visualization routines. We make use of Python's object-oriented paradigm leading to modular code that is extensible through inheritance and subtype polymorphism. SimPEG follows a fully open source development paradigm (Feller and Fitzgerald, 2000), and uses the permissive MIT license. Throughout its development, we have focused on modularity, usability, documentation, and extensive unit-testing (Wilson et al., 2014; Holscher et al., 2010; Kalderimis and Meyer, 2011; Merwin et al., 2015). See the website http://simpeg.xyz for up-to-date code,

[1] Alternatively, the inverse problem can be formulated in a probabilistic framework, see for example Tarantola (2005) and Tarantola and Valette (1982). In this paper we will focus our attention on the deterministic approach.

examples and documentation of this package; in addition see Kang et al. (2014, 2015a,b), Kang and Oldenburg (2015), Heagy (2014), and Heagy et al. (2015) for examples of research and use cases throughout a variety of geophysical applications. We hope that the organization, modularity, minimal dependencies, documentation, and testing in SimPEG will encourage reproducible research, cooperation, and communication to help tackle some of the inherently multidisciplinary geophysical problems.

To guide the discussion, we start this paper by outlining gradient based inversion methodology in Section 2. The inversion methodology directly motivates the construction of the SimPEG framework, terminology, and software implementation which we discuss in Section 3. We weave an example of Direct Current (DC) resistivity throughout the discussion of the SimPEG framework to provide context for the choices made and highlight many of the features of SimPEG.

2. Inversion methodology

Geophysical inverse problems are motivated by the desire to extract information about the earth from measured data. A typical geophysical datum can be written as

$$F_i[\mathbf{m}] + \epsilon_i = d_i, \tag{1}$$

where F is a forward simulation operator that incorporates details of the relevant physical equations, sources, and survey design, $\mathbf{m}$ is a generic symbol for the inversion model, ϵ_i is the noise that is often assumed to have known statistics, and d_i is the observed datum. In a typical geophysical survey we are provided with the data d_i, $i = 1 \ldots N$ and some estimate of their uncertainties. The goal is to recover the model, $\mathbf{m}$, which is often a physical property. The data provide only a finite number of inaccurate constraints upon the sought model. Finding a model from the data alone is an ill-posed problem since there is no unique model that explains the data. Additional information must be included using prior information and assumptions, for example, downhole property logs, structural orientation information, or known interfaces (Fullagar et al., 2008; Li and Oldenburg, 2000a; Lelièvre et al., 2009). This prior knowledge is crucial if we are to obtain an appropriate representation of the earth, and will be discussed in more detail in Section 2.1.

Defining a well-posed inverse problem and solving it is a complex task that requires many components that must interact. It is helpful to view this task as a workflow in which various elements are explicitly identified and integrated. Fig. 1 outlines the inversion methodology that consists of inputs, implementation, and evaluation. The inputs are composed of the geophysical data, the equations which are a mathematical description of the governing physics, and prior knowledge or assumptions about the setting. The implementation consists of two broad categories: the forward simulation and the inversion. The forward simulation is the means by which we solve the governing equations given a model and the inversion components evaluate and update this model. We are considering a gradient based approach, which updates the model through an optimization routine. The output of this implementation is a model, which, prior to interpretation, must be evaluated. This requires considering, and often re-assessing, the choices and assumptions made in both the input and implementation stages. In this paper we are primarily concerned with the implementation component, that is, how the forward simulation and inversion are carried out numerically. As a prelude to discussing how the SimPEG software is implemented, we step through the elements in Fig. 1 considering a Tikhonov-style inversion.

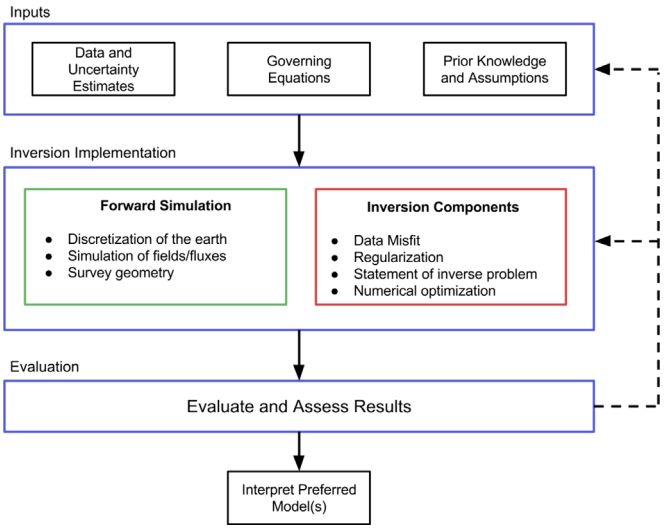

Fig. 1. Inversion methodology. Including inputs, implementation, evaluation and interpretation.

2.1. Inputs

Three sources of input are required prior to performing an inversion: (1) the geophysical data and uncertainty estimates, (2) the governing equations that connect the sought model with the observations, and (3) prior knowledge about the model and the context of the problem.

2.1.1. Data and uncertainty estimates

At the heart of the inversion are the geophysical data that consist of measurements over the earth. These data depend on the type of survey, the physical property distribution, and the type and location of the measurements. The details about the survey, for example, the location, orientation and waveform of a source, and which component of a particular wavefield is measured at a receiver, must be known. The data are contaminated with additive noise which can sometimes be estimated by taking multiple realizations of the data. However, standard deviations of those realizations only provide a lower bound for the noise. For the inverse problem, the uncertainty in the data must include not only this additive noise, but also any discrepancy between the true earth experiment and our mathematical representation of the data. This requires accounting for mislocation of receivers and sources, poor control of the transmitter waveform, electronic gains or filtering applied to signals entering the receivers, incorrect dimensionality in our mathematical model (e.g. working in 2D instead of 3D), neglect of physics in our mathematical equations by introducing assumptions (e.g. using a straight ray tomography vs. a full waveform simulation in seismic), and discretization errors of our mathematical equations.

2.1.2. Governing equations

The governing equations provide the connection between the physical properties of the subsurface and the data we observe. Most frequently, these are sets of partial differential equations with specific boundary conditions. The governing equations, with specified source terms, can be solved through numerical discretization using finite volume, finite element, or integral equation techniques. Alternatively, they may also be solved through evaluations of analytic functions. Whichever approach is taken, it is crucial that there exists some way to simulate the data response given a model.

2.1.3. Prior knowledge

If there is one model that acceptably fits the data then there are infinitely many. Additional information is therefore required to reduce the non-uniqueness. This can be geologic information, petrophysical knowledge about the various rock types, borehole logs, additional geophysical data sets, or inversion results. This prior information can be used to construct reference models for the inversion and also characterize features of the model, such as whether it is best described by a smooth function or if it is discontinuous across interfaces. Physical property measurements can be used to assign upper and lower bounds for a physical property model at points in a volume or in various regions within our 3D volume. The various types of information that are relevant to the geologic and geophysical questions to be addressed must be combined and translated into useful information for the inversion (Lelièvre et al., 2009; Li et al., 2010).

2.2. Implementation

In this section we outline the components necessary to formulate a well-posed inverse problem and solve it numerically. Two major abilities are critical to running the inversion: (1) the ability to simulate data and (2) the ability to assess and update the model (Fig. 1).

2.2.1. Forward simulation

The ability to carry out an inversion presupposes the ability to run a forward simulation and create predicted data given a physical property model. In forward simulation, we wish to compute $F[\mathbf{m}] = \mathbf{d}_{\text{pred}}$. The operator F simulates the specific measurements taken in a geophysical survey using the governing equations. The survey refers to all details regarding the field experiment that are needed to simulate the data. The forward simulation of DC resistivity data requires knowledge of the topography, the resistivity of the earth, and the survey details including locations of the current and potential electrodes, the source waveform, and the units of the observations. To complete the simulation, we need to solve our governing equations using the physical property model, $\mathbf{m}$, that is provided. In the DC resistivity experiment, our partial differential equation with supplied boundary conditions is solved with an appropriate numerical method, for example, finite volumes, finite elements, integral equations, or semi-analytic methods for 1D problems. In any case, we must discretize the earth onto a numerical forward simulation mesh ($mesh_F$) that is appropriate. The size of the cells will depend upon the structure of the physical property model, topography, as well as the distance between sources and receivers. Cells in $mesh_F$ must be small enough, and the domain large enough, so that sufficient numerical accuracy is achieved. Proper mesh design is crucial so that numerical modeling errors are below a prescribed threshold value (cf. Haber, 2015).

In general, we can write our governing equations in the form of

$$C(\mathbf{m}, \mathbf{u}) = 0, \tag{2}$$

where $\mathbf{m}$ is the modeled physical property, $\mathbf{u}$ are the fields and/or fluxes. C is often given by a partial differential equation or a set of partial differential equations. Information about the sources and appropriate boundary conditions are included in C. This system is solved for $\mathbf{u}$ and the predicted data are extracted from $\mathbf{u}$ via a projection (or functional), $\mathbf{d}_{\text{pred}} = P[\mathbf{u}]$. The ability to simulate the geophysical problem and generate predicted data is a crucial building block. Accuracy and efficiency are essential since many forward problems must be evaluated when carrying out any inversion.

2.2.2. Inversion elements

In the inverse problem, the first step is to specify how we parameterize the earth model. Finding a distributed physical property can be done by discretizing the 3D earth into voxels, each of which has a constant but unknown value. It is convenient to refer to the domain on which this model is discretized as the inversion mesh, $mesh_I$. The choice of discretization involves an assessment of the expected dimensionality of the earth model. If the physical property varies only with depth, then the cells in $mesh_I$ can be layers and a 1D inverse problem can be formulated. A more complex earth may require 2D or 3D discretizations. The choice of discretization depends on the spatial distribution and resolution of the data and the expected complexity of the geologic setting. We note that the inversion mesh has different design criteria and constraints than the forward simulation mesh. For convenience, many inverse problems have historically been solved with $mesh_I = mesh_F$ so that only one discretization is needed for the inversion. There is a growing body of work that investigates combinations of inversion discretizations and forward modeling meshes that are geared towards problem specific formulations as well as efficiency in large-scale problems (Haber and Schwarzbach, 2014; Yang et al., 2014; Haber and Heldmann, 2007). In any formulation there exists a mapping between $mesh_I$ and $mesh_F$ such that the parameterization chosen can be used to simulate data in a forward simulation.

To formulate a mathematical statement of the inverse problem, there are two essential elements:

1. *data misfit*: a metric that measures the misfit between the observed and predicted data; and
2. *regularization*: a metric that is constructed to evaluate the model's agreement with assumptions and prior knowledge.

The data misfit requires an assessment of the error in each datum. These errors result from anything that yields a discrepancy between the mathematical modeling and the true value. It includes additive noise, errors in the description of survey parameters (e.g. receiver locations, transmitter waveforms, etc.), incorrect choice of governing equations, and numerical errors arising from the simulation technique. Clearly, quantifying the noise for each datum is a challenge.

The data misfit is a measure of how well the data predicted by a given model reproduce the observed data. To assess *goodness of fit*, we select a norm which evaluates the 'size' of the misfit. This metric must include an uncertainty estimate for each datum. Often we assume that the data errors are Gaussian and uncorrelated and then estimate the standard deviation for each datum. The most common norm, and one that is compatible with Gaussian statistics, has the form

$$\phi_d(\mathbf{m}) = \frac{1}{2} \| \mathbf{W}_{\text{d}}(F[\mathbf{m}] - \mathbf{d}_{\text{obs}}) \|_2^2. \tag{3}$$

Here $F[\mathbf{m}]$ is a forward modeling that produces predicted data $\mathbf{d}_{\text{pred}}$, as in Eq. (1). $\mathbf{W}_{\text{d}}$ is a diagonal matrix whose elements are equal to $\mathbf{W}_{d_{ii}} = 1/\epsilon_i$ where ϵ_i is an estimated standard deviation of the ith datum. It is important to give careful thought into assignment of these. A good option is to assign a $\epsilon_i = floor + \%|d_i|$. Percentages are generally required when there is a large dynamic range of the data. A percentage alone can cause great difficulty for the inversion if a particular datum acquires a value close to zero, and therefore we include a floor.

In addition to a metric that evaluates the size of the misfit, it is also required that we have a tolerance, ϕ_d^*; models satisfying $\phi_d(\mathbf{m}) \leq \phi_d^*$ are considered to adequately fit the data (Parker, 1994). If the data errors are Gaussian and we have assigned the correct standard deviations, then the expected value of $\phi_d^* \approx N$, where N is

the number of data. Finding a model that has a misfit substantially lower than this will result in a solution that has excessive and erroneous structure, that is, we are fitting the noise. Finding a model that has a misfit substantially larger than this will yield a model that is missing structure that could have been extracted from the data (see Oldenburg and Li, 2005 for a tutorial).

The choice of misfit in Eq. (3) is not the only possibility for a misfit measure. If data errors are correlated, then $\mathbf{W}_\mathrm{d}$ is the square root of the data covariance matrix and it will have off-diagonal terms. Often useful in practice is recognizing if the noise statistics are non-Gaussian. Incorporating robust statical measures, like l_p norms with $p \approx 1$, are useful for handling outliers (Ekblom, 1973; Farquharson and Oldenburg, 1998).

The second essential inversion element is defining the regularization functional. If there is one model that has a misfit that equals the desired tolerance, then there are infinitely many other models which can fit to the same degree. The challenge is to find that model which has the desired characteristics and is compatible with *a priori* information. A single model can be selected from an infinite ensemble by measuring the length, or norm, of each model. Then a smallest, or sometimes largest, member can be isolated. Our goal is to design a norm that embodies our prior knowledge and, when minimized, yields a realistic candidate for the solution of our problem. The norm can penalize variation from a reference model, spatial derivatives of the model, or some combination of these. We denote this norm by ϕ_m and write it in a matrix form, for example,

$$\phi_m(\mathbf{m}) = \frac{1}{2} \parallel \mathbf{W}_\mathrm{m}(\mathbf{m} - \mathbf{m}_\mathrm{ref}) \parallel_2^2. \tag{4}$$

$\mathbf{W_m}$ is a matrix and $\mathbf{m}_\mathrm{ref}$ is a reference model (which could be zero). The matrix $\mathbf{W_m}$ can be a stacked combination of matrices weighted by α_*:

$$\mathbf{W_m} = [\alpha_s I, \quad \alpha_x W_x^\top, \quad \alpha_y W_y^\top, \quad \alpha_z W_z^\top]^\top. \tag{5}$$

Here $\mathbf{W}_\mathrm{m}$ is a combination of smallness and first-order smoothness in the x, y, and z directions. Each of the $\mathbf{W}$ matrices is, in fact, a discrete representation of an integral (cf. Oldenburg and Li, 2005). The final regularization $\mathbf{W}_\mathrm{m}$ can be a weighted sum of these, with α_* being applied as scalars or diagonal matrices with varying weights for each cell or cell face (cf. Oldenburg and Li, 2005; Haber, 2015). Additional weightings can also be incorporated through $\mathbf{W}_\mathrm{m}$, such as depth weighting, which is important in potential field inversions (such as magnetics and gravity), or sensitivity weightings to prevent model structure from concentrating close to sources or receivers (Li and Oldenburg, 1996a, 2000c). The regularization functionals addressed provide constraints on the model in a weak form, that is, a single number is used to characterize the entire model. Strong constraints that work within each cell can often be provided in the form of upper and lower bounds; these will be incorporated in the following section. The l_2 norms referred to above are appropriate for many problems, however models norms, such as l_p-norms, total variation, minimum support stabilizing functionals, or rotated smoothness operators that favor different character and/or include additional information can also be designed (cf. Oldenburg, 1984; Oldenburg and Li, 2005; Claerbout and Muir, 1973; Strong and Chan, 2003; Zhdanov, 2002; Li and Oldenburg, 2000a). The potential to have different norms tailored to a specific problem, with the additional functionality of localized weightings and reference models, provides the user with the capability of designing a regularization that favors a solution that is compatible with existing knowledge about the model. This task is not trivial, requires careful thought, and must often be re-evaluated and adjusted as the geophysicist iterates over the inversion procedure (Fig. 1).

2.2.3. Statement of the inverse problem

The purpose of this section is to pose our inverse problem in a mathematically precise way and to provide a methodology by which a solution can be achieved. In the work that follows, we outline a specific methodology that we will later demonstrate. We formulate the inverse problem as a problem in optimization, where we define an objective function based on the data misfit and model regularization and aim to find a model which sufficiently minimizes it. Many variants of this are possible.

At this stage of the workflow, all of the necessary components for formulating the inverse problem as an optimization problem are at hand. We have the capability to forward model and generate predicted data, assess the data misfit using ϕ_d, and a tolerance on the data misfit has been specified. A regularization functional ϕ_m and additional strong constraints on the model have been identified, such as upper and lower bounds: $\mathbf{m}_i^L \le \mathbf{m}_i \le \mathbf{m}_i^H$. The sought model is one that minimizes ϕ_m and also reduces the data misfit to some tolerance ϕ_d^*. However, a reduction in data misfit requires that the model have increased structure, which typically is at odds with the assumptions we impose in the construction of ϕ_m, meaning the ϕ_d and ϕ_m are antagonistic. To address this and still pose the inversion as an optimization problem, we design a composite objective function:

$$\phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m}), \tag{6}$$

where β is a positive constant. It is often referred to as the trade-off parameter, regression parameter, regularization parameter or Tikhonov parameter (Tikhonov and Arsenin, 1977). When β is very large, the minimization of $\phi(\mathbf{m})$ produces a model that minimizes the regularization term and yields a large $\phi_d(\mathbf{m})$. Alternatively, when β is very small, minimization of $\phi(\mathbf{m})$ produces a model that fits the data very well but is contaminated with excessive structure so that $\phi_m(\mathbf{m})$ is large. The inverse problem is posed as

$$\begin{aligned} &\underset{\mathbf{m}}{\text{minimize}} \quad \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \beta\phi_m(\mathbf{m}) \\ &\text{such that} \quad \phi_d \le \phi_d^*, \quad \mathbf{m}_i^L \le \mathbf{m}_i \le \mathbf{m}_i^H. \end{aligned} \tag{7}$$

Since the value of β is not known *a priori*, the above optimization problem can be solved at many values of β to produce a trade-off, or Tikhonov, curve (cf. Parker, 1994; Hansen, 1998). An optimum value, β^*, can be found so that minimizing Eq. (6) with β^* produces a model with misfit ϕ_d^*. The value of β^* can be found via cooling techniques where the β is progressively reduced from some high value and the process stopped when the tolerance is reached, or using two-stage methods as advocated by Parker (1977). There are other strategies for selecting the trade-off parameter including: the L-curve technique (Hansen, 1992), which attempts to find the point of greatest curvature in the Tikhonov curve; and Generalized Cross Validation (Wahba, 1990; Golub et al., 1979; Golub and Von Matt, 1997; Haber and Oldenburg, 2000; Oldenburg and Li, 2005; Farquharson and Oldenburg, 2004).

The optimization posed in Eq. (7) is almost always non-linear. It is linear only in the special case that the forward mapping is a linear functional of the model, ϕ_m and ϕ_d are written as l_2 norms, β is known, and that there are no imposed bound constraints. This rarely happens in practice, requiring that iterative optimization methods be employed to find a solution. Gradient-based methods are commonly used and the reader is referred to Nocedal and Wright (1999), for background and introductions to the relevant literature. For geophysical problems, Gauss–Newton techniques have proven to be valuable and practical. For l_2 norms we write the objective function as

$$\phi(\mathbf{m}) = \frac{1}{2} \parallel \mathbf{W}_\mathrm{d}(F[\mathbf{m}] - \mathbf{d}_\mathrm{obs}) \parallel_2^2 + \frac{1}{2}\beta \parallel \mathbf{W}_\mathrm{m}(\mathbf{m} - \mathbf{m}_\mathrm{ref}) \parallel_2^2. \tag{8}$$

The gradient is given by

$$g(\mathbf{m}) = J[\mathbf{m}]^{\top}\mathbf{W}_d^{\top}\mathbf{W}_d(F[\mathbf{m}] - \mathbf{d}_{obs}) + \beta\mathbf{W}_m^{\top}\mathbf{W}_m(\mathbf{m} - \mathbf{m}_{ref}), \tag{9}$$

where $J[\mathbf{m}]$ is the sensitivity or Jacobian. The components $J[\mathbf{m}]_{ij}$ specify how the ith datum changes with respect to the jth model parameter; this will be discussed in more detail in the next section. At the kth iteration, beginning with a model $\mathbf{m}^k$, we search for a perturbation $\delta\mathbf{m}$ that reduces the objective function. Linearizing the forward simulation by

$$F[\mathbf{m}^k + \delta\mathbf{m}] \approx F[\mathbf{m}^k] + J[\mathbf{m}^k]\delta\mathbf{m} \tag{10}$$

and setting the gradient equal to zero yields the standard Gauss–Newton equations to be solved for the perturbation $\delta\mathbf{m}$:

$$(J[\mathbf{m}]^{\top}\mathbf{W}_d^{\top}\mathbf{W}_d J[\mathbf{m}] + \beta\mathbf{W}_m^{\top}\mathbf{W}_m)\delta\mathbf{m} = -g(\mathbf{m}). \tag{11}$$

The updated model is given by

$$\mathbf{m}^{k+1} = \mathbf{m}^k + \gamma\delta\mathbf{m}, \tag{12}$$

where $\gamma \in (0, 1]$ is a coefficient that can be found by a line search. Setting $\gamma = 1$ is the default and a line search is necessary if $\phi(\mathbf{m}^{k+1}) \geq \phi(\mathbf{m}^k)$.

The iterative optimization process is continued until a suitable stopping criterion is reached. Completion of this iterative process yields a minimization for particular value of the trade-off parameter, β. If we are invoking a cooling schedule, and if the desired misfit tolerance is not yet achieved, β is reduced and the iterative numerical optimization procedure is repeated.

2.2.4. Sensitivities

A central element in the above approach is the computation of the sensitivities. The sensitivity functional is defined by

$$J[\mathbf{m}] = \frac{dF[\mathbf{m}]}{d\mathbf{m}} = \mathbf{P}\left(\frac{d\mathbf{u}}{d\mathbf{m}}\right), \tag{13}$$

where $\mathbf{P}$ is a linear projection, and $d\cdot$ indicates total difference. There are numerous approaches to computing the sensitivity but the chosen methodologies are dictated by the size of the problem. The discrete sensitivity matrix, $\mathbf{J}$, is a dense $N \times M$ matrix, where N is the number of data and M is the number of model parameters. For some problems, $\mathbf{J}$ can be computed directly and stored. Ultimately this demands the solution of numerous forward problems (cf. Haber, 2015). Another approach is to factor $J[\mathbf{m}]$ in symbolic form. In the general case, we solve for the sensitivity implicitly by taking the derivative of $C(\mathbf{m}, \mathbf{u}) = 0$ (Eq. (2)) to yield:

$$\nabla_{\mathbf{m}} C(\mathbf{m}, \mathbf{u})\, d\mathbf{m} + \nabla_{\mathbf{u}} C(\mathbf{m}, \mathbf{u})\, d\mathbf{u} = 0, \tag{14}$$

where ∇ indicates partial difference, and both $\nabla_{\mathbf{m}} C(\mathbf{m}, \mathbf{u})$ and $\nabla_{\mathbf{u}} C(\mathbf{m}, \mathbf{u})$ are matrices. For a given model, $\nabla_{\mathbf{u}} C(\mathbf{m}, \mathbf{u})$ corresponds to the forward simulation operator, and if the forward problem is well-posed, then the matrix is invertible (Haber, 2015). Eq. (14) can be rearranged to

$$d\mathbf{u} = -\left(\nabla_{\mathbf{u}} C(\mathbf{m}, \mathbf{u})\right)^{-1} \nabla_{\mathbf{m}} C(\mathbf{m}, \mathbf{u})\, d\mathbf{m}, \tag{15}$$

and combined with Eq. (13) to obtain a formula for the sensitivity matrix. We note that this matrix is dense, often large, and need not actually be formed and stored.

2.2.5. Inversion as optimization

Once the inverse problem has been stated in an optimization framework (Eq. (7)), an appropriate optimization routine can be selected. For example, if bound constrains are incorporated, a projected Gauss–Newton algorithm can be used. In large-scale inversions, special attention may have to be given to ensuring a memory efficient optimization algorithm, however, the underlying mechanics of the algorithms often remain unchanged. In a geophysical inversion we require a model that is consistent with *a priori* information and known or assumed statistical distributions. As such, the stopping criteria of the inversion are often implemented differently than traditional optimization algorithms, or a series of incomplete optimization algorithms are invoked while changing the objective function (Oldenburg and Li, 2005; Haber, 2015; Haber et al., 2000).

The optimization of the stated inverse problem provides the machinery to obtain a mathematical solution. However, before the model is accepted as a viable candidate, there are numerous questions that should be investigated. For example, questions to be addressed might include: (a) How well does the recovered model fit the observed data? (b) Is there bias in the misfits between the observed and predicted data? (c) What was the path for the convergence? (d) Is there too much or too little structure? (e) Does the model fit with prior knowledge and other data sets? The final results and details about how the inversion algorithm has performed all provide clues as to whether the constructed model can be accepted, or if elements in our procedure or its numerical implementation need to be altered and the inversion rerun. This might include adjusting the assigned uncertainties in the misfit function, altering the model regularization, or changing aspects of the numerical computations.

2.3. Evaluation/interpretation

In this section we return to the initial question posed for which the inversion was designed to help answer. Items of interest might include: (a) Are the interesting features supported by the data or are they artifacts? (b) Does the result make sense geologically and geophysically? (c) Are there interesting features that should be investigated further? Addressing these questions usually involves repeating the inversion process with appropriate modifications (cf. Oldenburg and Li, 2005; Pidlisecky et al., 2011; Lines et al., 1988). As such, we require an implementation that is inherently and unequivocally modular, with all pieces available to manipulation. Black-box software, where the implementations are hidden, obfuscated, or difficult to manipulate, do not promote experimentation and investigation. Exposing the details of the implementation to the geophysicist in a manner that promotes productivity and question-based interrogation is the goal of SimPEG and is the topic of the next section.

3. Modular implementation

There are an overwhelming amount of choices to be made as one works through the forward modeling and inversion process (Fig. 1). As a result, software implementations of this workflow often become complex and highly interdependent, making it difficult to interact with and to ask other scientists to pick up and change. Our approach to handling this complexity is to propose a framework, Fig. 2, that compartmentalizes the implementation of inversions into various units. We present it in this specific modular style, as each unit contains a targeted subset of choices crucial to the inversion process.

The aim of the SimPEG framework and implementation is to allow users to move between terminology, math, documentation and code with ease, such that there is potential for development in a scalable way. The SimPEG implementation provides a library that mimics the framework shown in Fig. 2 with each unit representing a base class. These base classes can be inherited in specific geophysical problems to accelerate development as well as to create code that is consistent between geophysical applications.

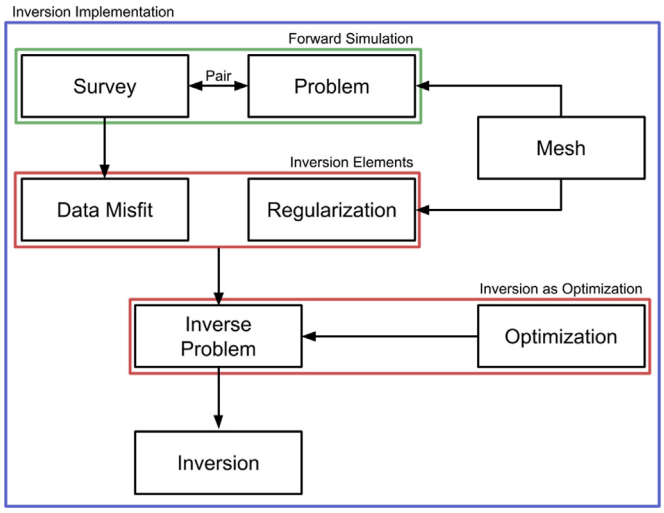

Fig. 2. SimPEG framework indicating the flow of information. In the implementation, each of these modules is a base class.

3.1. Implementation choices

We chose Python (Van Rossum and Drake, 1995) for the implementation of SimPEG. Python supports object-oriented practices, interactive coding, has extensive support for documentation, and has a large and growing open source scientific community (Lin, 2012). To enhance the dissemination of our work, we have released our work under the permissive MIT license for open source software. The MIT license does not force packages that use SimPEG to be open source nor does it restrict commercial use. We have also ensured that our practices with regard to version control, code-testing and documentation follow best practices (Wilson et al., 2014).

3.2. Overview

As discussed in the previous section, the process of obtaining an acceptable model from an inversion generally requires the geophysicist to perform several iterations of the inversion workflow, rethinking and redesigning each piece of the framework to ensure it is appropriate in the current context. Inversions are experimental and empirical by nature and our software package is designed to facilitate this iterative process. To accomplish this, we have divided the inversion methodology into eight major components (Fig. 2). The `Mesh` class handles the discretization of the earth and also provides numerical operators. The forward simulation is split into two classes, the `Survey` and the `Problem`. The `Survey` class handles the geometry of a geophysical problem as well as sources. The `Problem` class handles the simulation of the physics for the geophysical problem of interest. Although created independently, these two classes must be *paired* to form all of the components necessary for a geophysical forward simulation and calculation of the sensitivity. The `Problem` creates geophysical fields given a source from the `Survey`. The `Survey` interpolates these fields to the receiver locations and converts them to the appropriate data type, for example, by selecting only the measured components of the field. Each of these operations may have associated derivatives with respect to the model and the computed field; these are included in the calculation of the sensitivity. For the inversion, a `DataMisfit` is chosen to capture the *goodness of fit* of the predicted data and a `Regularization` is chosen to handle the non-uniqueness. These inversion elements and an `Optimization` routine are combined into an inverse problem class (`InvProblem`). `InvProblem` is the mathematical statement (i.e. similar to Eq. (7)) that will be numerically solved by running an `Inversion`. The `Inversion` class handles organization and dispatch of *directives* between all of the various pieces of the framework.

The arrows in the Fig. 2 indicate what each class takes as a primary argument. For example, both the `Problem` and `Regularization` classes take a `Mesh` class as an argument. The diagram does not show class inheritance, as each of the base classes outlined have many subtypes that can be interchanged. The `Mesh` class, for example, could be a regular Cartesian mesh or a cylindrical coordinate mesh, which have many properties in common. These common features, such as both meshes being created from tensor products, can be exploited through inheritance of base classes, and differences can be expressed through subtype polymorphism. We refer the reader to the online, up-to-date documentation (http://docs.simpeg.xyz) to look at the class inheritance structure in depth.

3.3. Motivating example

We will use the DC resistivity problem from geophysics to motivate and explain the various components of the SimPEG framework. This example will be referred to throughout this section; we will introduce it briefly here, and refer the reader to Pidlisecky et al. (2007) for a more in-depth discussion. The governing equations for DC resistivity are

$$\nabla\cdot(-\sigma\nabla\phi) = I(\delta(\vec{r} - \vec{r}_{s^+}) - \delta(\vec{r} - \vec{r}_{s^-})), \tag{16}$$

where σ is the electrical conductivity, ϕ is the electric potential, I is the input current at the positive and negative dipole locations $\vec{r}_{s^\pm}$, captured as Dirac delta functions. In DC resistivity surveys, differences in the potential field, ϕ, are sampled using dipole receivers to collect observed data. To simulate this partial differential equation PDE) (or set of PDEs, if there are multiple current injection locations), we must discretize the equation onto a computational mesh.

3.4. Mesh

Any numerical implementation requires the discretization of continuous functions into discrete approximations. These approximations are typically organized in a mesh, which defines boundaries, locations, and connectivity. Of specific interest to geophysical simulations, we require that averaging, interpolation and differential operators be defined for any mesh. In SimPEG, we have implemented a staggered mimetic finite volume approach (Hyman and Shashkov, 1999; Hyman et al., 2002). This approach requires the definitions of variables at either cell-centers, nodes, faces, or edges as described in Fig. 3. We will focus our attention on explaining our implementation of a tensor mesh. A more

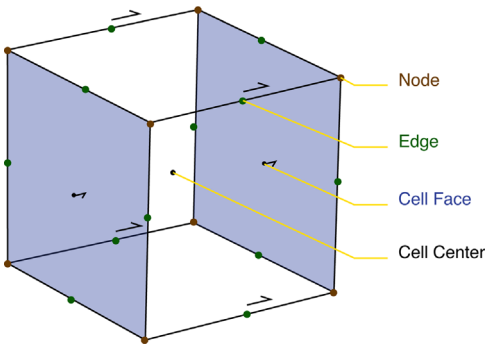

Fig. 3. Location of variable in a single voxel of a three dimensional finite volume discretization.

Table 1
Selected `Mesh` class properties with explanations.

Property or function	Explanation
dim	Dimension of the mesh
x0	Location of the origin
nC, nN, nF, nE	The number of cells, nodes, faces, or edges. (e.g. nC is the total number of cells)
vol, area, edge	Geometric measurements for the mesh
gridN, gridCC, etc.	Array of grid locations
nodalGrad	Gradient of a nodal variable → edge variable
faceDiv	Divergence of a face variable → cell – centered variable
edgeCurl	Curl of a edge variable → face variable
cellGrad	Gradient of a cell-centered variable → face variable
aveF2CC, aveN2CC, etc.	Averaging operators (e.g. F → CC, takes values on faces and averages them to cell-centers)
getInterpolationMat(loc)	Interpolation matrix for *xyz* locations

detailed explanation can be found in Haber (2015). To create a new `Mesh` instance, a `TensorMesh` class can be selected from the SimPEG Mesh module and instantiated with a list of vectors:

```
from SimPEG import Mesh, Solver, Utils, np, sp
hx = np.ones(30)
hy = np.ones(30)
mesh = Mesh.TensorMesh([hx,hy])
```

Here, the SimPEG library is imported, as well as NumPy (`np`) and SciPy's sparse matrix package (`sp`) (Oliphant, 2007; Jones et al., 2001). The vectors `hx` and `hy` describe the cell size in each mesh dimension. The dimension of the mesh is defined by the length of the list, requiring very little change to switch mesh dimensions or type. Once an instance of a mesh is created, access to the properties and methods shown in Table 1 is possible. There are additional methods and visualization routines that are also included in the SimPEG `Mesh` classes. Of note in Table 1 are organizational properties (such as counting, and geometric properties), locations of mesh variables as Cartesian grids, differential and averaging operators, and interpolation matrices. The mesh implementation can be readily extended to other types of finite volume meshes, for example, OcTree (Haber and Heldmann, 2007), logically rectangular non-orthogonal meshes (Hyman et al., 2002), and unstructured meshes (Ollivier-Gooch and Van Altena, 2002). Additionally, this piece of the framework may be replaced by other methodologies such as finite elements.

The `Mesh` interface allows for lazy loading of properties, that is, all properties of the mesh are created on demand and then stored for later use. This is important as not all operators are useful in all problems and, as such, are not created. The implementation here is different from some other finite volume implementations as the operators are held in memory as matrices and are readily available for interrogation. We find this feature to be extremely beneficial for educational and research purposes as the discretization remains visually very close to the math, and the matrices can be manipulated, visually inspected, and combined readily.

With the differential operators readily accessible, simulation of a cell-centered discretization for conductivity, σ, in the DC resistivity problem is straight-forward. The discretized system of Eq. (16) can be written as

$$\mathbf{A}(\boldsymbol{\sigma})\mathbf{u} = \mathbf{D}(\mathbf{M}^{\mathbf{f}}_{1/\sigma})^{-1}\mathbf{G}\mathbf{u} = -\mathbf{q}, \tag{17}$$

where $\mathbf{D}$ and $\mathbf{G}$ are the divergence and gradient operators, respectively. The conductivity, σ, is harmonically averaged from cell-centers to cell-faces to create the matrix $(\mathbf{M}^{\mathbf{f}}_{1/\sigma})^{-1}$ (Pidlisecky et al., 2007). In SimPEG this is written as:

```
D = mesh.faceDiv
G = mesh.cellGrad
Msig = Utils.sdiag(1/(self.mesh.aveF2CC.T * (1/
   sigma)))
A = D*Msig * G
```

The code is easy to read, looks similar to the math, can be built interactively using tools such as IPython (Pérez and Granger, 2007), and is not dependent on the dimension of mesh used. Additionally, it is decoupled from the mesh type, for example, Fig. 4 is generated by solving a `DCProblem` for three different mesh types: `TensorMesh`, `TreeMesh` and `CurvilinearMesh`. Other than the specific mesh generation code, no other modifications to the DC problem were necessary (see the online examples provided in SimPEG). Given the electrode locations, a $\mathbf{q}$ can be constructed on each mesh and the system $\mathbf{A}(\boldsymbol{\sigma})\mathbf{u} = -\mathbf{q}$ solved. SimPEG comes with a few different types of `Solver` objects that provide a simple and common interface to direct and iterative solvers found across many different Python packages.

```
Ainv = Solver(A) # Create a solver object
u = Ainv * (- q )
mesh.plotImage(u)
```

The potential field can be projected onto the receiver electrode

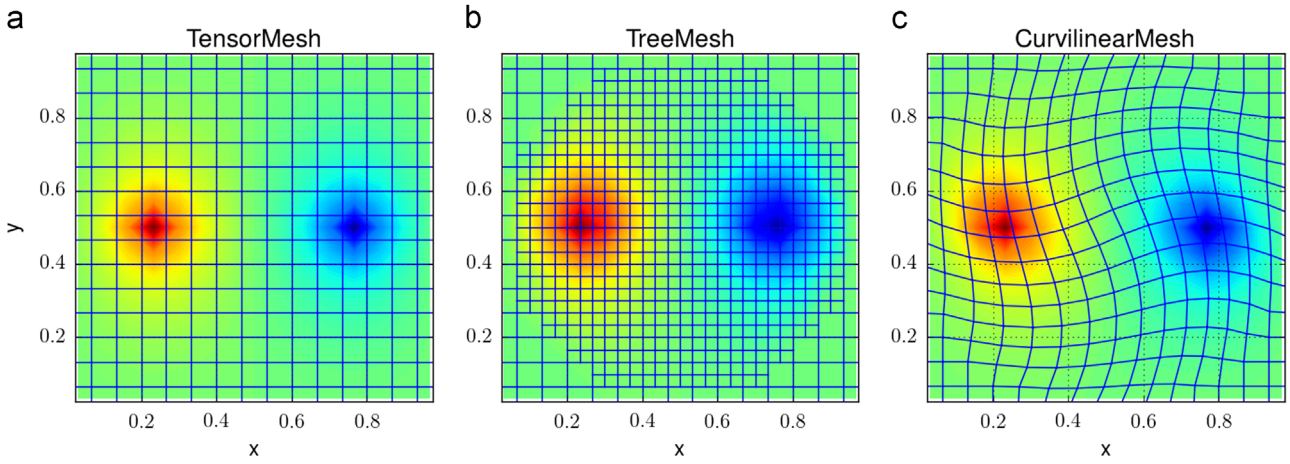

Fig. 4. Solving the DC resistivity problem for a dipole and using the meshes visualization routine for the potential, ϕ, for three different mesh types: (a) TensorMesh, (b) TreeMesh, and (c) CurvilinearMesh. The potential has been interpolated onto the tensor mesh for visualization.

locations through interpolation matrices constructed by the `Mesh` class. Additionally, there are multiple visualization routines that have been included in the `Mesh` class for rapid visualization and interrogation of geophysical fields and physical properties (Fig. 4). We note that these code snippets can be easily be combined in a script, highlighting the versatility and accessibility of the `Mesh` classes in SimPEG.

This script will be expanded upon and segmented into the various pieces of the framework in the following sections. We find that the development of geophysical codes is often iterative and requires 'scripting' of equations. Only after these are correct, as demonstrated by an appropriate test (e.g. `Tests.checkDerivative`), do we formalize and segment our script to enable a geophysical inversion to be run. The toolbox that SimPEG provides promotes this interactive and iterative style of development.

3.5. Forward simulation

The forward simulation in SimPEG is broken up into a `Survey` class and a `Problem` class. The `Problem` class contains the information and code that capture the physics used to describe the connection between a physical property distribution and the fields/fluxes that are measured in a geophysical survey. The `Survey` class contains information about the observed data and the geometry of how to collect the data (e.g. locations and types of receivers and sources) given a `Problem` that simulates fields. The `Problem` and the `Survey` must be *paired* together to simulate predicted data. We decided on this separation of the code because it is possible to have multiple mathematical descriptions, of varying complexities, that explain the same observed data. For example, a seismic simulation could have multiple approximations to the physics which increase in complexity and accuracy, from straight-ray tomography, Eikonal tomography, to full waveform simulation. Additionally, there are often multiple types of geophysical surveys that could be simulated from the same `Problem` class.

The crucial aspects of the `Problem` class are shown in Table 2, and the properties and methods of the `Survey` class are shown in Table 3. We note that each of the sub-classes of `Problem` will implement fields and sensitivities in a different way, likely with additional methods and properties. Furthermore, the choice of terminology becomes clearer when these classes are inherited and used in a specific geophysical method (e.g. a `DCProblem` or `EMProblem`). For the `DCProblem`, the `fields` can be created by constructing $\mathbf{A}(\mathbf{m})$ and solving with the source terms, $\mathbf{Q}$, which will be provided by the `DCSurvey`'s source list (`srcList`). Each source has at least one receiver associated with it; the receivers can create a matrix, $\mathbf{P}$, that project the fields, $\mathbf{u}$, onto the data-space. For example, in the DC problem, a dipole receiver samples the potential at each electrode location and computes the difference to give a datum. We note that the process of computing a datum may be more involved and have derivatives with respect the computed fields and possibly the model. Much of the organizational bottlenecks are taken care of through general receiver and source classes, which can be inherited and tailored to the specific application. The `mapping` in the `Problem` provides a transform

Table 2
Base `Problem` class properties with explanations.

Property or function	Explanation
fields(m)	Calculation of the fields given a model
Jvec(m, v)	Sensitivity times of a vector
Jtvec(m, v)	Adjoint sensitivity times a vector
mapping	Maps the model to a physical property

Table 3
Selected `Survey` class properties with explanations.

Property or function	Explanation
dobs, nD	$\mathbf{d}_{\text{obs}}$, number of data
std	Estimated standard deviations
srcList	List of sources with associated receivers
dpred(m)	Predicted data given a model, $\mathbf{d}_{\text{pred}}(m)$
projectFields(m, u)	Projects the fields, $P(m, u)$
projectFieldsDeriv(m, u)	Derivative of the projection, $\frac{dP(m,u)}{dm}$
residual(m)	$\mathbf{d}_{\text{pred}}(m) - \mathbf{d}_{\text{obs}}$

from an arbitrary model to a discretized grid function of physical properties. For example, log-conductivity is often used in the inverse problem for DC resistivity rather than parameterizing directly in terms of conductivity. If this choice is made for the model, an appropriate map (i.e. the exponential) must be provided to transform from the model space to the physical property space (cf. Heagy, 2014).

3.6. DC resistivity forward simulation

A simple DC-resistivity survey is presented to demonstrate some of the components of SimPEG in action. A set of Schlumberger arrays is used to complete a vertical sounding. In this example, we have taken our scripts from the previous section describing the forward simulation and combined them in a package called `simpegDC` (http://simpeg.xyz). We use the 3D tensor `mesh` to run the forward simulation for the data of this problem.

```
import simpegDC as DC
survey = DC.SurveyDC(srcList)
problem = DC.ProblemDC(mesh)
problem.pair(survey)
data = survey.dpred(sigma)
```

Here the `srcList` is a list of dipole sources (`DC.SrcDipole`), each of which contains a single receiver (`DC.RxDipole`). Similar to the illustration in Fig. 2, the `Problem` and the `Survey` must be paired for either to be used to simulate fields and/or data. These elements represent the major pieces of any forward simulation in geophysics, and are crucial to have in place and well tested for accuracy and efficiency before any attempt is made at setting up the inverse problem.

3.7. Sensitivities

The sensitivity and adjoint will be used in the optimization routine of the inversion. Inefficient or inaccurate calculation of the sensitivities can lead to an extremely slow inversion. This is critical in large-scale inversions where the dense sensitivity matrix may be too large to hold in memory directly. As discussed in the methodology section, the sensitivity matrix need not be explicitly created when using an iterative optimization algorithm such as Gauss–Newton (11) solved with a conjugate gradient approach. The calculation of vector products with the sensitivity matrices is an important aspect of SimPEG, which has many tools to make construction and testing of these matrices modular and simple. For the DC resistivity example, the discretized governing equations are written as $C(\mathbf{m}, \mathbf{u}) = \mathbf{A}(\mathbf{m})\mathbf{u} - \mathbf{q} = \mathbf{0}$. We can implement the sensitivity equations (13) and (15) to yield:

$$\mathbf{J} = -\mathbf{P}(\mathbf{A}(\mathbf{m})^{-1}\nabla_{\mathbf{m}} C(\mathbf{m}, \mathbf{u})), \tag{18}$$

where $\nabla_{\mathbf{m}} C(\mathbf{m}, \mathbf{u})$ is a known sparse matrix, $\mathbf{A}(m)$ is the forward

```
def Jvec(self, m, v, u=None):
    # Set current model; clear dependent property A(m)
    self.curModel = m
    sigma = self.curModel.transform # σ = M(m)
    if u is None:
        # Run forward simulation if u not provided
        u = self.fields(self.curModel)
    else:
        shp = (self.mesh.nC, self.survey.nTx)
        u = u.reshape(shp, order='F')

    D = self.mesh.faceDiv
    G = self.mesh.cellGrad
    # Derivative of model transform, ∂σ/∂m
    dsigdm_x_v = self.curModel.transformDeriv * v

    # Take derivative of C(m,u) w.r.t. m
    dCdm_x_v = np.empty_like(u)
    # loop over fields for each transmitter
    for i in range(self.survey.nTx):
        # Derivative of inner product, (M^f_{1/σ})^{-1}
        dAdsig         = D * self.dMdsig( G * u[:,i] )
        dCdm_x_v[:, i] = dAdsig *  dsigdm_x_v

    # Take derivative of C(m,u) w.r.t. u
    dCdu = self.A
    # Solve for ∂u/∂m
    dCdu_inv = self.Solver(dCdu, **self.solverOpts)
    P        = self.survey.getP(self.mesh)
    J_x_v    = - P * mkvc( dCdu_inv * dCdm_x_v )
    return J_x_v
```

Fig. 5. Sensitivity times of a vector method for the `DCProblem`.

operator and is equivalent to $\nabla_{\mathbf{u}} C(\mathbf{m}, \mathbf{u})$, and $\mathbf{P}$ is a projection matrix (cf. Pidlisecky et al., 2007). Each matrix in this expression is sparse and can be explicitly formed, however, the product is dense and it may not be possible to hold it in memory. If an iterative solver is used in the optimization, only matrix vector products are necessary and the sensitivity need not be explicitly calculated or stored. Fig. 5 outlines the calculation of `Jvec` given a model, `m`, the fields, `u`, and a vector to multiply, `v`. In Fig. 5, we draw the distinction between the model, `m`, and the conductivity, `sig`, which are connected through a mapping, $\sigma = \mathcal{M}(\mathbf{m})$, and associated derivatives. The matrix $\nabla_{\mathbf{m}} C(\mathbf{m}, \mathbf{u})$ is denoted `dCdm` and formed by looping over each source in the DC resistivity survey.

3.8. Inversion elements

As indicated in the methodology section, there are two key elements needed for a geophysical inversion: `DataMisfit` and `Regularization`. The `DataMisfit` must have a way to calculate predicted data, as such, it takes a paired survey as an initial argument, allowing forward simulations to be completed. `DataMisfit` and `Regularization` have similar interfaces which are shown in Table 4. The `DataMisfit` class also has a property, `targetMisfit`, for the target misfit, which can be checked by an `InversionDirective` and used as a stopping criteria. As discussed in the methodology section, the `Regularization` is defined independently from the forward simulation. The regularization is with respect to the model, which may or may not be on the same mesh as the forward simulation (i.e. $mesh_I \neq mesh_F$). In this case, a mapping of a model to a physical property on the forward simulation mesh is necessary for the `Problem`. The `Regularization` class also has a `mapping` property allowing a wide variety of regularizations to be implemented (e.g. an active cell map used to ignore air cells). As such, the `Regularization` `mapping` is often independent from the `mapping` in the `Problem` class, which outputs a physical property. Included in the SimPEG package are basic Tikhonov regularization routines and simple l_2 norms for both `Regularization` and `DataMisfit` classes. Each of these classes has properties for the appropriate model and data weightings discussed in the previous section (e.g. $\mathbf{W}_\mathrm{m}$ and $\mathbf{W}_\mathrm{d}$). These classes are readily extensible such that they can be customized to specific problems and applications, such as considering l_1 or l_p norms or customized regularizations.

Table 4
Common functions for the `Regularization`, `DataMisfit`, `InvProblem` classes.

Function	Explanation
eval(m)	Evaluate the functional given a model
evalDeriv(m)	First derivative returns a vector
eval2Deriv(m, v)	Second derivative as an implicit operator

3.9. Inverse problem and optimization

The `InvProblem` combines the `DataMisfit` and `Regularization` classes by introducing a trade-off parameter, β. In addition to the trade-off parameter, there are methods that evaluate the objective function and its derivatives (Table 4). Additional methods can save fields such that information is not lost between evaluation of the objective function and the derivatives. The `InvProblem` may also include bounds on the model properties so they can be used in the optimization routine. If one considers a joint or integrated inversion, multiple data misfit functions, employing different physics, and multiple types of regularization functionals may be summed together, possibly with relative weightings, to define the `InvProblem` (cf. Lines et al., 1988; Holtham and Oldenburg, 2010; Heagy, 2014). Once the `InvProblem` can be evaluated to a scalar and has associated derivatives, an `Optimization` can either be chosen among the ones included in SimPEG or provided by an external package. Optimization routines in SimPEG include steepest descent, L-BFGS, and Inexact Gauss–Newton (cf. Nocedal and Wright, 1999). The components are relatively simple to hook up to external optimization packages, for example, with the optimization package in SciPy (Jones et al., 2001).

3.10. Inversion

The `Inversion` conducts all communication between the various components of the framework and is instantiated with an `InvProblem` object. The `Inversion` has very few external methods, but contains the list of directives that are executed throughout the inversion. Each `InversionDirective` has access to the components of the inversion framework and can thus access and change any of these components as the inversion is running. A simple directive may print optimization progress or save models to a database. More complicated directives may change or compute parameters such as β, reference models, data weights, or model weights. These directives are often guided by heuristics, but versions can often be formalized, see for example the iterative Tikhonov style inversion (Tikhonov and Arsenin, 1977; Parker, 1994; Oldenburg and Li, 2005). There are many computational shortcuts that may be investigated, such as how many inner and outer CG iterations to complete in the inexact Gauss–Newton optimization and whether this should change as the algorithm converges to the optimal model. The `directiveList` in the `Inversion` encourages heuristics that geophysicists often complete 'by hand' to be codified, combined, and shared via a plug-in style framework.

3.11. DC resistivity inversion

We will build on the example presented in Section 3.6, which has a survey setup that only provides enough information for a vertical sounding. As such, we will decouple our 3D forward mesh

and 1D inversion mesh and connect them through a mapping (cf. Kang et al., 2015a). Additionally, since electrical conductivity is a log-varying parameter, we will also construct a model space that is optimized in log space. Both of these model transformations will be handled with a single map, $\mathcal{M}$, where $\boldsymbol{\sigma} = \mathcal{M}(\mathbf{m})$.

```
from SimPEG import Maps
mapping = Maps.ExpMap(mesh) * Maps.Vertical1DMap
   (mesh)
sigma=mapping * model
```

We have provided a number of common mapping transformations in the `SimPEG.Maps` package, and these can be easily combined through a multiplication symbol. Additionally, when using these maps, the derivatives are calculated using the chain rule allowing them to be easily included in the sensitivity calculation (cf. Fig. 5, line 15). Fig. 7 demonstrates this mapping visually. The 1D model is in $\log(\boldsymbol{\sigma})$, shown in Fig. 6(a) as a black solid line and the transformation produces a 3D `sigma` vector which was plotted in Fig. 6(b). We can now use the same simulation machinery as discussed in Section 3.6, with a single change:

problem = DC. ProblemDC(mesh, mapping = mapping)

Synthetic data, $\mathbf{d}_{\text{obs}}$, are created using the 1D log-conductivity model and adding 1% Gaussian noise. When creating the regularization inversion element, again we note that the `mapping` parameter can be used to regularize in the space which makes the most sense. In this case we will regularize on a 1D mesh in log-conductivity space, as such, we supply only a 1D tensor `mesh` to the regularization. An inversion is run by combining the tools described above. Fig. 2 illustrates how the components are put together.

```
mesh1D=Mesh.TensorMesh([mesh.hz])
dmis=DataMisfit.l2_DataMisfit(survey)
reg=Regularization.Tikhonov(mesh1D)
opt=Optimization.InexactGaussNewton()
invProb=InvProblem.BaseInvProblem(dmis, reg,
   opt)
inv=Inversion.BaseInversion(invProb)
mopt=inv.run(m0)
```

We note that there are many options and inputs that can enhance the inversion; refer to the online up-to-date documentation (http://docs.simpeg.xyz). The result of this inversion can be seen in Fig. 7(a) and (b) for the predicted data and model, respectively.

3.12. Development practices

Throughout the development of SimPEG, we have focused on building both a framework (Fig. 2) and a toolbox that is flexible and extensible. The toolbox includes utilities that we use repeatedly in our research, for example, visualization routines, mesh generation, and synthetic modeling. Additionally, when writing new code for differential operators and PDE systems, several functions are available to help test and verify results including: (a) checking derivatives and expected order of convergence, (b) comparing to analytics, and (c) adjoint tests for the sensitivity operators (cf. Haber, 2015). These tests can be written in-line in an interactive development paradigm and then rapidly transferred and incorporated as unit-tests to ensure future code changes do not change functionality. Currently the entire SimPEG project has upwards of 80% test coverage, with all core functionality (e.g. discretization and optimization) being extensively tested. All changes are combined using the practice of continuous integration, supported by freely available tools for open source projects such as TravisCI and GitHub. Additionally, we have focused on writing and maintaining documentation for core functionality (http://docs.simpeg.xyz). The documentation is also practicing continuous integration and is updated with all changes to SimPEG and is built and hosted through ReadTheDocs (Holscher et al., 2010). We are hopeful that our efforts and adherence to best practices in open source code development (cf. Wilson et al., 2014) will encourage a community to exercise reproducible research around these scientific tools (cf. Fomel and Claerbout, 2009).

4. Conclusions

Producing an interpretation from geophysical data through an inversion is an iterative process with many moving pieces. A number of inversion components, techniques and methodologies have become standard practice. The development of new methodologies to address the evolving challenges in the geosciences will build upon and extend these standard practices, requiring experimentation with and recombination of existing techniques. To facilitate this combinatorial experimentation, we have organized the components of geophysical inverse problems in a comprehensive, modular framework. Our implementation of this framework, SimPEG (http://www.simpeg.xyz), provides an extensible, well-tested toolbox and infrastructure that supports problems including electromagnetics, fluid flow, seismic, and potential fields. As SimPEG is formulated with the inverse problem as its core focus, many design choices have been made to ensure that sensitivities are efficient to compute and are readily available; we

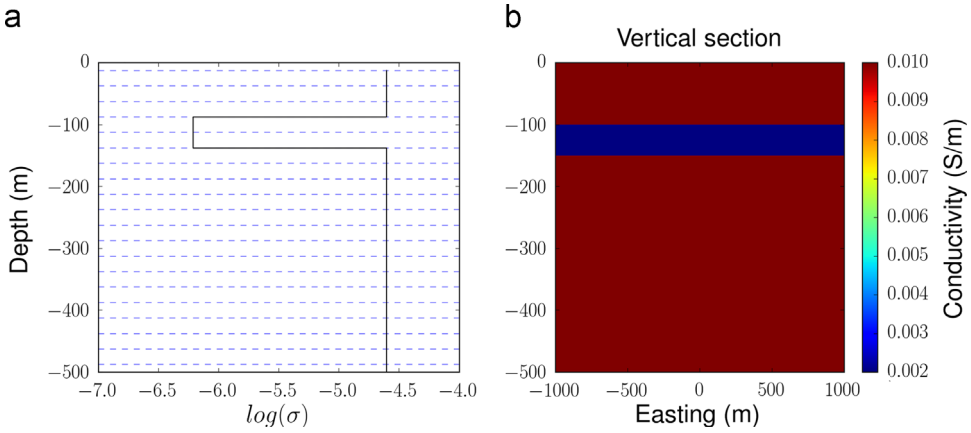

Fig. 6. Illustration of mapping in DC inversion. (a) 1D log conductivity model. (b) 3D conductivity model.

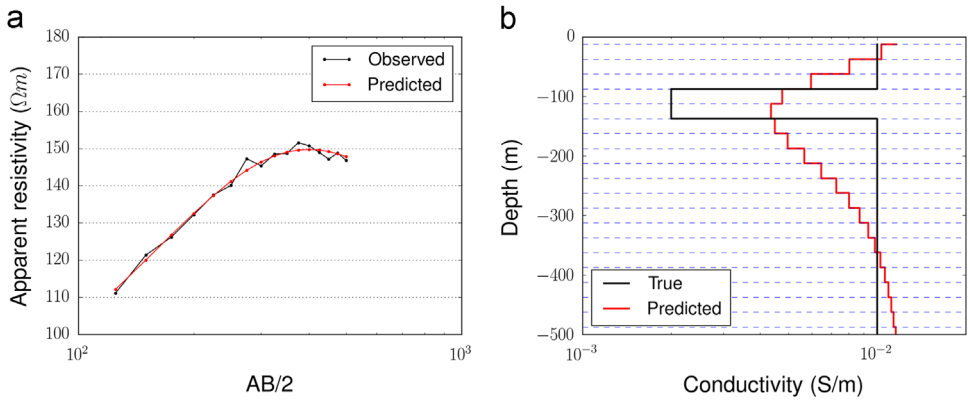

Fig. 7. (a) Observed (black line) and predicted (red line) apparent resistivity values. (b) True and recovered 1D conductivity model. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

presume this will be advantageous for integrated geophysical inversions. The modular framework we suggest splits the code into components that are motivated directly by geophysical methodology and terminology. This allows each piece to be improved by specialists whilst promoting quantitative communication between researchers.

To accelerate the dissemination and adoption of SimPEG in the wider community, we have made the entire project open source under the permissive MIT License. The usability of this framework has been a focus of SimPEG, and we strive to use best practices of continuous integration, documentation (http://docs.simpeg.xyz), unit-testing, and version-control. These practices are key to have in place as more modules and packages are created by the community.

Acknowledgments

We would like to thank Dr. Eldad Haber for his invaluable guidance and advice in the development process of our framework and SimPEG implementation. Dr. Dave Marchant, Dr. Lars Ruthotto, Luz Angelica Caudillo-Mata, Gudni Rosenkjaer, and Dr. Brendan Smithyman have contributed to the SimPEG code base and are always willing to talk through problems and ideas. The funding for this work is provided through the Vanier Canada Graduate Scholarships Program, and Grants through The University of British Columbia (NSERC 22R47082) and the University of Calgary (NSERC Discovery Grant for A. Pidlisecky). We also acknowledge and thank three reviewers, Peter Lelièvre, Thomas Hansen, and an anonymous reviewer for their thoughtful and constructive comments on this paper.

References

Astropy CollaborationRobitaille, T.P., Tollerud, E.J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A.M., Kerzendorf, W. E., Conley, A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M.M., Nair, P.H., Unther, H.M., Deil, C., Woillez, J., Conseil, S., Kramer, R., Turner, J.E.H., Singer, L., Fox, R., Weaver, B.A., Zabalza, V., Edwards, Z.I., Azalee Bostroem, K., Burke, D.J., Casey, A.R., Crawford, S.M., Dencheva, N., Ely, J., Jenness, T., Labrie, K., Lim, P.L., Pierfederici, F., Pontzen, A., Ptak, A., Refsdal, B., Servillat, M., Streicher, O., 2013. Astropy: a community python package for astronomy. Astron. Astrophys. 558 (October), A33.

Claerbout, J.F., Muir, F., 1973. Robust modeling with erratic data. Geophysics 38 (5), 826–844.

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 (3), 289–300, URL ⟨http://link.aip.org/link/?GPY/52/289/1⟩.

Doetsch, J., Linde, N., Coscia, I., Greenhalgh, S.A., Green, A.G., 2010. Zonation for 3D aquifer characterization based on joint inversions of multimethod crosshole geophysical data. Geophysics 75 (6), G53–G64.

Ekblom, H.k., 1973. Calculation of linear bestL p-approximations. BIT Numer. Math. 13 (3), 292–300.

Farquharson, C.G., Oldenburg, D.W., 1998. Non-linear inversion using general measures of data misfit and model structure. Geophys. J. Int. 134, 213–227.

Farquharson, C.G., Oldenburg, D.W., 2004. A comparison of automatic techniques for estimating the regularization parameter in non-linear inverse problems. Geophys. J. Int. 156 (March (3)), 411–425, URL ⟨http://gji.oxfordjournals.org/content/156/3/411.abstract ⟩.

Feller, J., Fitzgerald, B., 2000. A framework analysis of the open source software development paradigm. In: Proceedings of the 21st International Conference on Information Systems. ICIS '00. Association for Information Systems, Atlanta, GA, USA, pp. 58–69. URL ⟨http://dl.acm.org/citation.cfm?id=359640.359723⟩.

Fomel, S., Claerbout, J.F., 2009. Guest editors' introduction: reproducible research. Comput. Sci. Eng. 11 (January (1)), 5–7, URL ⟨http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4720217⟩.

Fullagar, P.K., Pears, G.A., McMonnies, B., 2008. Constrained inversion of geologic surfaces pushing the boundaries. Lead. Edge 27 (1), 98–105. http://dx.doi.org/10.1190/1.2831686.

Gao, G., Abubakar, A., Habashy, T.M., 2012. Joint petrophysical inversion of electromagnetic and full-waveform seismic data. Geophysics 77 (3), WA3–WA18.

Golub, G.H., Heath, M., Wahba, G., 1979. Generalized cross-validation as a method for choosing a good ridge parameter. Technometrics 21 (2), 215–223.

Golub, G.H., Von Matt, U., 1997. Generalized cross-validation for large-scale problems. J. Comput. Graph. Stat. 6 (1), 1–34.

Haber, E., 2015. Computational Methods in Geophysical Electromagnetics. Mathematics in Industry. URL ⟨http://books.google.ca/books?id=favjoQEACAAJ⟩.

Haber, E., Ascher, U.M., Oldenburg, D., 2000. On optimization techniques for solving nonlinear inverse problems. Inverse Probl. 16 (5), 1263.

Haber, E., Heldmann, S., 2007. An OcTree multigrid method for quasi-static Maxwell's equations with highly discontinuous coefficients. J. Comput. Phys. 65, 324–337.

Haber, E., Oldenburg, D., 1997. Joint inversion a structural approach. Inverse Probl. 13, 63–67.

Haber, E., Oldenburg, D.W., 2000. A gcv based method for nonlinear ill-posed problems. Comput. Geosci. 4 (1), 41–63.

Haber, E., Schwarzbach, C., 2014. Parallel inversion of large-scale airborne time-domain electromagnetic data with multiple OcTree meshes. Inverse Probl. 30 (5), 55011, URL ⟨http://stacks.iop.org/0266-5611/30/i=5/a=055011⟩.

Hansen, P.C., 1992. Analysis of discrete ill-posed problems by means of the l-curve. SIAM Rev. 34 (4), 561–580.

Hansen, P.C., 1998. Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion, vol. 4. SIAM.

Hansen, T.M., Cordua, K.S., Looms, M.C., Mosegaard, K., 2013. SIPPI: a Matlab toolbox for sampling the solution to inverse problems with complex prior information. Comput. Geosci. 52 (March), 481–492, URL ⟨http://linkinghub.elsevier.com/retrieve/pii/S0098300412003408⟩.

Harbaugh, A., 2005. The U.S. geological survey modular ground-water model the ground-water flow process. In: U.S. Geological Survey Techniques and Methods (6), A16.

Heagy, L.J., Cockett, R., Oldenburg, D.W., 2014. Parametrized Inversion Framework for Proppant Volume in a Hydraulically Fractured Reservoir.

Heagy, L.J., Cockett, R., Oldenburg, D.W., Wilt, M., 2015. Modelling Electromagnetic Problems in the Presence of Cased Wells, pp. 2–6.

Hewett, R., Demanet, L., the PySIT Team, 2013. PySIT: Python Seismic Imaging Toolbox v0.5. Release 0.5. URL ⟨http://www.pysit.org⟩.

Holscher, E., Leifer, C., Grace, B., 2010. Read the Docs. URL ⟨https://docs.readthedocs.org⟩.

Holtham, E., Oldenburg, D.W., 2010. Three-dimensional inversion of MT and ZTEM data. SEG Denver 2010 Annual Meeting (2), pp. 655–659.
Hunter, J.D., 2007. Matplotlib: a 2D graphics environment. Comput. Sci. Eng. 9 (3), 90–95.
Hyman, J., Morel, J., Shashkov, M., Steinberg, S., 2002. Mimetic finite difference methods for diffusion equations. Comput. Geosci. 6 (3-4), 333–352.
Hyman, J.M., Shashkov, M., 1999. Mimetic Discretizations for Maxwell's Equations, vol. 909, pp. 881–909.
Jones, E., Oliphant, T., Peterson, P., Others, 2001. {SciPy}: open source scientific tools for {Python}. URL ⟨http://www.scipy.org/⟩.
Kalderimis, J., Meyer, M., 2011. Travis ci gmbh. URL ⟨https://travis-ci.org⟩.
Kang, S., Cockett, R., Heagy, L.J., Oldenburg, D.W., 2015a. Moving between dimensions in electromagnetic inversions. SEG Technical Program Expanded Abstracts, 5000–5004. http://dx.doi.org/10.1190/segam2015-5930379.1.
Kang, S., Oldenburg, D.W., 2015. Recovering IP information in airborne-time domain electromagnetic data. ASEG Ext. Abstr. 2015 (January (1)), 1–4, URL ⟨http://www.publish.csiro.au/paper/ASEG2015ab102⟩.
Kang, S., Oldenburg, D.W., McMillan, M.S., 2015b. 3D IP inversion of airborne EM data at Tli Kwi Cho. ASEG Ext. Abstr. 2015 (January (1)), 1–4, URL ⟨http://www.publish.csiro.au/paper/ASEG2015ab274⟩.
Kang, S., Oldenburg, D.W., Yang, D., Marchant, D., 2014. On Recovering Induced Polarization Information from Airborne Time Domain EM Data, pp. 1785–1789 (Chapter 341). URL ⟨http://library.seg.org/doi/abs/10.1190/segam2014-1381.1⟩.
Kelbert, A., Meqbel, N., Egbert, G.D., Tandon, K., 2014. ModEM: a modular system for inversion of electromagnetic geophysical data. Comput. Geosci. 66 (May), 40–53, URL ⟨http://linkinghub.elsevier.com/retrieve/pii/S0098300414000211⟩.
Lelièvre, P.G., Oldenburg, D.W., Williams, N.C., 2009. Integrating geological and geophysical data through advanced constrained inversions. Explor. Geophys. 40 (4), 334–341.
Li, M., Abubakar, A., Habashy, T.M., Zhang, Y., 2010. Inversion of controlled-source electromagnetic data using a model-based approach. Geophys. Prospect. 58 (3), 455–467, URL ⟨http://dx.doi.org/10.1111/j.1365-2478.2009.00824.x⟩.
Li, Y., Key, K., 2007. 2d marine controlled-source electromagnetic modeling: Part 1 – an adaptive finite-element algorithm. Geophysics 72 (2), WA51–WA62, URL ⟨http://dx.doi.org/10.1190/1.2432262⟩.
Li, Y., Oldenburg, D., 2000a. Incorporating geological dip information into geophysical inversions. Geophysics 65 (1), 148–157, URL ⟨http://dx.doi.org/10.1190/1.1444705⟩.
Li, Y., Oldenburg, D.W., 1996a. 3-d inversion of magnetic data. Geophysics 61 (2), 394–408.
Li, Y., Oldenburg, D.W., 1996b. 3D inversion of magnetic data. Geophysics 61, 394–408.
Li, Y., Oldenburg, D.W., 1998. 3-D inversion of gravity data. Geophysics 63 (January (1)), 109–119, URL ⟨http://library.seg.org/doi/abs/10.1190/1.1444302⟩.
Li, Y., Oldenburg, D.W., 2000b. Incorporating geological dip information into geophysical inversions. Geophysics 65 (January (1)), 148–157, URL ⟨http://library.seg.org/doi/abs/10.1190/1.1444705⟩.
Li, Y., Oldenburg, D.W., 2000c. Joint inversion of surface and three-component borehole magnetic data. Geophysics 65 (2), 540–552.
Lin, J.W.-B., 2012. Why Python is the next wave in earth sciences computing. Bull. Am. Meteorol. Soc. 93 (December (12)), 1823–1824, URL ⟨http://journals.ametsoc.org/doi/abs/10.1175/BAMS-D-12-00148.1⟩.
Lines, L.R., Schultz, A.K., Treitel, S., 1988. Cooperative inversion of geophysical data. Geophysics 53 (1), 8–20.
Merwin, N., Donahoe, L., Mcangus, E., 2015. Coveralls. URL ⟨https://coveralls.io⟩.
Nocedal, J., Wright, S.J., 1999. Numerical Optimization. Springer, New York, NY.
Oldenburg, D.W., 1984. An introduction to linear inverse theory. IEEE Trans. Geosci. Remote Sens. (6), 665–674.
Oldenburg, D.W., Li, Y., 2005. 5. Inversion for Applied Geophysics: A Tutorial, pp. 89–150 (Chapter 5). URL ⟨http://library.seg.org/doi/abs/10.1190/1.9781560801719.ch5⟩.
Oliphant, T.E., 2007. Python for scientific computing. Comput. Sci. Eng. 9 (3).
Ollivier-Gooch, C., Van Altena, M., 2002. A high-order-accurate unstructured mesh finite-volume scheme for the advection–diffusion equation. J. Comput. Phys. 181 (September (2)), 729–752, URL ⟨http://linkinghub.elsevier.com/retrieve/pii/S0021999102971597⟩.
Parker, R.L., May 1977. Understanding inverse theory. Ann. Rev. Earth Planet. Sci. 5 (1), 35–64, URL ⟨http://www.annualreviews.org/doi/abs/10.1146/annurev.ea.05.050177.000343⟩.
Parker, R.L., 1994. Geophysical Inverse Theory. Princeton University Press.
Pérez, F., Granger, B.E., 2007. IPython: a system for interactive scientific computing. Comput. Sci. Eng. 9 (May 3), 21–29, URL ⟨http://ipython.org⟩.
Pidlisecky, A., Haber, E., Knight, R., 2007. RESINVM3D: a 3D resistivity inversion package. Geophysics 72 (2), H1–H10.
Pidlisecky, A., Singha, K., Day-Lewis, F.D., 2011. A distribution-based parametrization for improved tomographic imaging of solute plumes. Geophys. J. Int. 187, 214–224.
Strong, D., Chan, T., 2003. Edge-preserving and scale-dependent properties of total variation regularization. Inverse Probl. 19 (6), S165.
Tarantola, A., 2005. Inverse Problem Theory and Methods for Model Parameter Estimation. Society for Industrial and Applied Mathematics. URL ⟨http://epubs.siam.org/doi/abs/10.1137/1.9780898717921⟩.
Tarantola, A., Valette, B., 1982. Generalized nonlinear inverse problems solved using the least squares criterion. Rev. Geophys. 20 (2), 219–232. http://dx.doi.org/10.1029/RG020i002p00219.
Tikhonov, A., Arsenin, V.Y., 1977. Solutions of Ill-Posed Problems. W.H. Winston and Sons.
Uieda, Leonardo; Oliveira Jr, Vanderlei C.; Ferreira, André; Santos, Henrique Bueno dos; Jr., José Fernando Caparica (2014): Fatiando a Terra: a Python package for modeling and inversion in geophysics.
Van Rossum, G., Drake, Jr., F.L., 1995. Python Reference Manual. Centrum voor Wiskunde en Informatica, Amsterdam.
Wahba, G., 1990. Spline Models for Observational Data, vol. 59. SIAM.
Wilson, G., Aruliah, D.a., Brown, C.T., Chue Hong, N.P., Davis, M., Guy, R.T., Haddock, S.H.D., Huff, K.D., Mitchell, I.M., Plumbley, M.D., Waugh, B., White, E.P., Wilson, P., 2014. Best practices for scientific computing. PLoS Biol. 12 (January (1)), e1001745, URL ⟨http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3886731&tool=pmcentrez&rendertype=abstract⟩.
Yang, D., Oldenburg, D.W., Haber, E., 2014. 3-D inversion of airborne electromagnetic data parallelized and accelerated by local mesh and adaptive soundings. Geophys. J. Int. 196 (March (3)), 1492–1507, URL ⟨http://gji.oxfordjournals.org/content/196/3/1492.abstract⟩.
Zhdanov, M.S., 2002. Preface. In: Zhdanov, M.S. (Ed.), Geophysical Inverse Theory and Regularization Problems. Methods in Geochemistry and Geophysics vol. 36. Elsevier, pp. XIX–XXIII, URL http://www.sciencedirect.com/science/article/pii/S0076689502800373⟩.