COMPUTATIONAL METHODS in Geophysical Electromagnetics

• MATHEMATICS IN INDUSTRY •

Series Volumes

Eldad Haber, *Computational Methods in Geophysical Electromagnetics*

COMPUTATIONAL METHODS in

Geophysical Electromagnetics

Eldad Haber
University of British Columbia
Vancouver, British Columbia
Canada

siam
Society for Industrial and Applied Mathematics
Philadelphia

10 9 8 7 6 5 4 3 2 1

Publisher	David Marshall
Acquisitions Editor	Elizabeth Greenspan
Developmental Editor	Gina Rinelli
Managing Editor	Kelly Thomas
Production Editor	Ann Manning Allen
Copy Editor	Ann Manning Allen
Production Manager	Donna Witzleben
Production Coordinator	Cally Shrader
Compositor	Scott Boswell
Graphic Designer	Lois Sellers

Library of Congress Cataloging-in-Publication Data

Haber, Eldad.
Computational methods in geophysical electromagnetics / Eldad Haber, University of British Columbia, Vancouver, British Columbia, Canada.
pages cm. -- (Mathematics in industry)
Includes bibliographical references and index.
ISBN 978-1-611973-79-2
1. Electromagnetism--Mathematics. 2. Geophysics--Mathematics. I. Title.
QC760.4.M37H33 2015
538'.70151--dc23

2014035243

For Noa, Neve, and Ori

It was difficult to write this book with you
but I could not have done it without you.

Contents

Preface

This book started as a set of lectures in computational science for inverse problems where electromagnetics was used as an example. Talking with many graduate students and other faculty, I realized that there is much "know-how" that is based on practical experience but is not covered in any course or book.

Such experience is crucial if we desire to solve realistic problems, write our own computer programs, and work with field data sets.

The book attempts to bridge the gap between more theoretical papers and books on inverse and forward problems and the complete applied material that gives very little detail on how to do things in practice. I have tried to stay away from formal mathematics and added computer codes that enable the reader to solve realistic problems. Working through the codes that are attached to this book (see www.siam.org/books/mn01) the user should be able to solve moderate-size electromagnetic (EM) forward and inverse problems on a stretched rectangular mesh.

The book is by no means complete. I do not cover finite element methods, Bayesian techniques, uncertainty estimation, and many other important topics. On the other hand, the book is mostly self-contained. It enables the reader to program a forward and an inverse EM problem from start to finish. I believe that issues such as uncertainty can be best served by gaining some first-hand experience with the forward and inverse problems.

The book can be used by students who wish to study EM forward and inverse problems as well as students interested in applied inverse problems with partial differential equations as constraints. The MATLAB software that accompanies this book can be used to solve realistic inverse EM problems and can be easily modified to solve other problems such as medical EIT or DOT. I hope that students as well as practitioners will find it useful.

Eldad Haber
Vancouver - Kochav-Michael, 2014

Chapter 1

Maxwell's Equations and Their Geophysical Applications

Maxwell's equations are a set of differential equations that describe the connection between the electric fields and fluxes and the magnetic fields and fluxes. These equations are the basis for the propagation of electromagnetic waves, particularly in heterogeneous and anisotropic media.

In this chapter, we discuss different representations of the equations and their application to geophysical exploration. Much of the material presented here can be found in classical textbooks such as [49, 55, 71, 78, 79], and we summarize the topics that are needed to understand the numerical methods and the applications discussed in this book.

1.1 ▪ Different representations of Maxwell's equations

1.1.1 ▪ Maxwell's equations in time and frequency

Let $\vec{E}$ be the electric field and $\vec{H}$ the magnetic field. Let $\vec{J}$ be the electric flux, $\vec{D}$ the displacement flux, and $\vec{B}$ the magnetic flux. Maxwell's equations read

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E}, \qquad \text{Faraday equation,} \tag{1.1a}$$

$$\frac{\partial \vec{D}}{\partial t} = \nabla \times \vec{H} - \vec{J} - \vec{s}, \qquad \text{Ampère's law,} \tag{1.1b}$$

where $\vec{s}$ is the source. The equations need to be equipped with some initial conditions on $\vec{B}$ and $\vec{D}$ as well as some boundary conditions on $\vec{E}$ or $\vec{H}$.

The system (1.1) connects the fields $\vec{E}$ and $\vec{H}$ to the fluxes $\vec{B}$, $\vec{J}$, and $\vec{D}$.

By using Stokes's theorem we see that Maxwell's equations have a geometrical interpretation.

$$\frac{\partial}{\partial t} \int_S \vec{B} \cdot dS = -\oint_{\partial S} \vec{E} \cdot d\ell, \tag{1.2a}$$

$$\frac{\partial}{\partial t} \int_S \vec{D} \cdot dS = \oint_{\partial S} \vec{H} \cdot d\ell - \int_S (\vec{J} + \vec{s}) \cdot dS. \tag{1.2b}$$

The geometrical connection between $\vec{E}$ and $\vec{B}$ is observed when considering any closed surface, S, and the line that encloses this surface, ∂S, as demonstrated in Figure 1.1. The

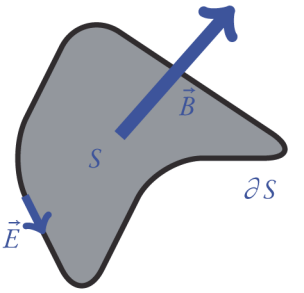

Figure 1.1. *Geometrical interpretation of Maxwell's equations.*

geometrical interpretation is important not only for theoretical purposes; as we see next, this interpretation is also the key for "good" numerical techniques.[1]

The system is not complete since it involves five (vector) unknowns and only two (vector) equations. To close the system, we need relationships between the fields and fluxes, namely, constitutive relations. The most common constitutive relations are linear,

$$\vec{J} = \sigma \vec{E}, \tag{1.3a}$$

$$\vec{B} = \mu \vec{H}, \tag{1.3b}$$

$$\vec{D} = \epsilon \vec{E}, \tag{1.3c}$$

where σ is the conductivity, μ is the magnetic permeability, and ϵ is the dielectric permittivity. The parameter functions σ, μ, ϵ are media dependent and can characterize different geological targets and environments. It is important to emphasize that while Maxwell's equations are fundamental, material properties are a gross macroscopic description of heterogeneous media. As we explore further in this chapter, other relations between the fields and fluxes are often considered.

Using the constitutive relations, the system (1.1) can be written in terms of the electric field and the magnetic flux as

$$\frac{\partial \vec{B}}{\partial t} = -\nabla \times \vec{E}, \tag{1.4a}$$

$$\epsilon \frac{\partial \vec{E}}{\partial t} = \nabla \times (\mu^{-1} \vec{B}) - \sigma \vec{E} - \vec{s}. \tag{1.4b}$$

Although many books use the fields $\vec{E}$ and $\vec{H}$ as primary variables, here we generally use the variables $\vec{E}$ and $\vec{B}$. This is because we take the view that Maxwell's equations are a system that connects fields to fluxes. As we see next, this is important for the discretization as we will consider the discretization of fields ($\vec{E}$) and a flux ($\vec{B}$). We will be using the relation $\vec{B} = \mu \vec{H}$ when we discuss the magnetic field at some points in this text.

One more condition "hides" within Maxwell's equations. Taking the divergence of Faraday's equations (1.1a), we obtain

$$\nabla \cdot \vec{B}_t = 0.$$

[1]The meaning of "good" numerical methods is explained in the next sections.

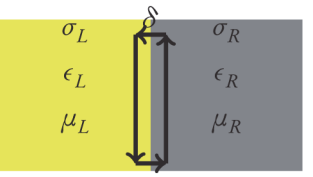

Figure 1.2. *Integration of Maxwell's equations along a path that goes through material discontinuity.*

Therefore, if $\vec{B}(0,\mathbf{x}) = 0$, that is, there are no magnetic charges, then we also have

$$\nabla \cdot \vec{B} = 0.$$

This is sometimes referred to as Gauss's law for magnetism.

In many cases, sources are periodic, and therefore it is easier to work with Maxwell's equations in the frequency domain. Taking the Fourier transform, we obtain

$$i\omega\widehat{\vec{B}} = -\nabla\times\widehat{\vec{E}}, \tag{1.5a}$$

$$i\omega\epsilon\widehat{\vec{E}} = \nabla\times(\mu^{-1}\widehat{\vec{B}}) - \sigma\widehat{\vec{E}} - \widehat{\vec{s}}, \tag{1.5b}$$

where $\widehat{\cdot}$ implies the Fourier transform of the quantity.

Mathematically, the fields $\vec{E}$ and $\vec{B}$ belong to Hilbert spaces known as $\mathscr{H}(\text{curl})$ and $\mathscr{H}(\text{div})$ [55]. These spaces are defined as spaces of functions with integrable square curl and divergence. While it is possible to develop many of the results in this work from a rigorous mathematical formulation that uses these spaces, we choose to use a more intuitive notion of fields and fluxes. We refer the reader to [55] for a mathematically rigorous treatment of these spaces.

1.1.2 ▪ Integral form of Maxwell's equations and interface conditions

The integral form of the equations (1.4) can be used to study the continuity behavior of the fields when material interfaces are present. Consider the case of a discontinuous interface where on the left we have material properties $\mu_L, \sigma_L, \epsilon_L$ and on the right the material properties are $\mu_R, \sigma_R, \epsilon_R$. Suppose a tall, thin rectangular surface that crosses the interface of width δ (see Figure 1.2).

We can integrate equation (1.2a) over the rectangular path and shrink δ to 0 to obtain

$$0 = \vec{E}_L^{\parallel} - \vec{E}_R^{\parallel} = (\vec{B}^{\perp})_R - (\vec{B}^{\perp})_L,$$

where $(\cdot)^{\parallel}$ is the field parallel or tangential to the interface, and $(\cdot)^{\perp}$ is the field normal to the interface. That is, the normal components of $\mu\vec{H}$ are continuous. This implies that if μ is discontinuous across an interface, so are the normal components of $\vec{H}$. On the other hand, the tangential components of $\vec{E}$ are continuous along the interface. By using the same rationale on equation (1.2b), we obtain that the tangential components of $\vec{H}$ are continuous along the interface, but the normal components of $\vec{E}$ obey

$$(\partial_t\vec{D}^{\perp} + \vec{J}^{\perp})_L = (\partial_t\vec{D}^{\perp} + \vec{J}^{\perp})_R,$$

and this implies that $\vec{E}$ is discontinuous in the normal direction to an interface.

The interface conditions have some immediate, practical implications on geophysical data. Consider for a moment the measurement of the electric field on the surface. Clearly, the surface is an interface between the air, with a conductivity of 10^{-8} S/m (not on a rainy day in Vancouver), and the ground, with a much higher conductivity of, say, 10^{-2} S/m. This implies that the vertical component of the electric field is discontinuous and there is no reason to measure it (as it has two very different values across this interface). On the other hand, since for most earth materials the magnetic permeability may not be significantly different than that of the air, it makes sense to measure three components of the magnetic field.

1.1.3 ▪ The equations in weak form

A useful way to write Maxwell's equations is by using the so-called weak form. Here, we will make use of this form when considering the discretization of Maxwell's equations. The idea behind the weak form is to integrate the equations with an appropriate class of functions and then reduce the requirement about differentiability of the fields. To start, we write the system in terms of the electric field, $\vec{E}$, and the magnetic flux, $\vec{B}$, assuming that the term $\epsilon\partial_t\vec{E}$ can be dropped (see section 1.2).

$$\frac{\partial\vec{B}}{\partial t}+\nabla\times\vec{E}=0, \tag{1.6a}$$

$$\nabla\times\mu^{-1}\vec{B}-\sigma\vec{E}=\vec{s}. \tag{1.6b}$$

Note that the $\nabla\times$ maps a field to a flux, and the product with μ^{-1} maps a flux to a field. Similarly, the product with σ maps from a field to a flux.

Let $\vec{F}$ be some flux function that is in the same functional space as $\vec{B}$, and let $\vec{W}$ be some field function in the same space as $\vec{E}$. Finally, let $(\cdot,\cdot)$ be the usual inner product defined as

$$(\vec{A},\vec{G})=\int_\Omega \vec{A}\cdot\vec{G}\,d\mathbf{x}$$

for any vector fields or fluxes $\vec{A},\vec{G}$.

To obtain the weak form we take the inner products of the first equation in (1.6) with $\vec{F}$ and those of the second with $\vec{W}$, and, choosing $\vec{F}$ and $\vec{W}$ to be time invariants, we obtain

$$\frac{\partial}{\partial t}(\vec{B},\vec{F})+(\nabla\times\vec{E},\vec{F})=0,$$
$$(\nabla\times\mu^{-1}\vec{B},\vec{W})-(\sigma\vec{E},\vec{W})=(\vec{s},\vec{W}).$$

Integrating the term $(\nabla\times\mu^{-1}\vec{B},\vec{W})$ by parts, we obtain

$$(\nabla\times\mu^{-1}\vec{B},\vec{W})=(\mu^{-1}\vec{B},\nabla\times\vec{W})+\int_{\partial\Omega}\mu^{-1}\vec{W}\cdot(\vec{B}\times\vec{n})\,dS,$$

and assuming that the boundary conditions on $\vec{B}$ are $\vec{B}\times\vec{n}=0|_{\partial\Omega}$, then the boundary

term drops, and we obtain

$$\frac{\partial}{\partial t}(\vec{B},\vec{F})+(\nabla\times\vec{E},\vec{F})=0, \tag{1.7a}$$

$$(\mu^{-1}\vec{B},\nabla\times\vec{W})-(\sigma\vec{E},\vec{W})=(\vec{s},\vec{W}). \tag{1.7b}$$

The advantage of representing the system in weak form is that we relax the demand of taking the **curl** $\mu^{-1}\vec{B}$. Since $\vec{W}$ and $\vec{E}$ belong to the same functional space, we require only the **curl** of fields from the space of $\vec{E}$. As we see in the next chapters, this fact can be used in order to discretize the system and obtain discrete symmetric linear systems.

1.2 ▪ Dimensional analysis and geophysical experiments

To better understand the overall behavior of the equations, we now use classical dimensional analysis. As we see next, different asymptotic regimes lead to different behavior of the solution and hence to different geophysical experiments.

Analyzing the terms in the system (1.4) or (1.5), we assume that the derivative terms behave as h^{-1}, where h is some characteristic length scale. We can now analyze a number of scenarios that depend on the importance of the different terms in Maxwell's equations and their interaction with the source.

1.2.1 ▪ Elliptic case—statics with magnetic sources: Magnetics

Assume that the angular frequency, ω, is sufficiently small such that $\mu\omega h \ll 1$ and also $\epsilon\omega h \ll 1$. This is equivalent to very slowly changing fields, that is, $\partial/\partial t \approx 0$. In this case, we can neglect the frequency dependent part of Maxwell's equations, (1.1), and using the relation $\vec{H}=\mu^{-1}\vec{B}$, we obtain the system

$$0=\nabla\times\vec{E}, \tag{1.8a}$$

$$\vec{s}=\nabla\times\vec{H}-\sigma\vec{E}. \tag{1.8b}$$

The identity $\nabla\times\nabla=0$ implies that the first equation can be solved by setting $\vec{E}=\nabla\phi$, where ϕ is some potential function. Recalling that $\nabla\cdot\mu\vec{H}=0$, we obtain the system

$$\nabla\times\vec{H}-\sigma\nabla\phi=\vec{s}, \tag{1.9a}$$

$$\nabla\cdot\mu\vec{H}=0. \tag{1.9b}$$

We can further divide the problem into two subproblems based on the properties of the source. Assume that the source is divergence-free; that is, $\nabla\cdot\vec{s}=0$. We refer to a divergence-free source as a *magnetic source*.

Taking divergence of the first equation, we obtain

$$\nabla\cdot\sigma\nabla\phi=-\nabla\cdot\vec{s}=0,$$

and since the boundary conditions at infinity for ϕ are homogeneous, $\phi|_{\partial\Omega}=0$, it turns out that $\phi=0$ everywhere. That is, there is no electric field if the source is static and divergence-free. This implies that

$$\nabla\times\vec{H}=\vec{s}, \tag{1.10a}$$

$$\nabla\cdot\mu\vec{H}=0. \tag{1.10b}$$

This is the full system that describes the static magnetic response of the earth due to a magnetic source.

The system (1.10) is an *elliptic partial differential equation*. Loosely speaking, elliptic differential equations are a generalization of the Poisson equation. They require only boundary conditions and no initial conditions. The solution of such equations is smoothing; that is, if the right-hand side has high frequencies, these frequencies are attenuated at the solution. For a more rigorous treatment and definition see [26].

The equations (1.10) are typically simplified with a number of assumptions about the sources and the magnetic susceptibility. In particular, we separate the problem into three cases.

Case I: Small perturbation in μ within earth's magnetic field

The simplest case is that the source is $\vec{s} = 0$ and that the boundary conditions are the earth's magnetic field, $\vec{H}_0$. In this case the equations can be written as

$$\nabla \times \vec{H} = 0, \tag{1.11a}$$

$$\nabla \cdot \mu \vec{H} = 0, \tag{1.11b}$$

$$\vec{H} \cdot \vec{n} = \vec{H}_0 \cdot \vec{n}, \quad \vec{H} \in \partial\Omega. \tag{1.11c}$$

The equation $\nabla \times \vec{H} = 0$ implies that $\vec{H} = \nabla\psi$, and therefore, we obtain a Poisson equation for ψ:

$$\begin{aligned} \nabla \cdot \mu \nabla \psi &= 0, \quad \psi \in \Omega, \\ \nabla \psi \cdot \vec{n} &= \vec{H}_0 \cdot \vec{n}, \quad \psi \in \partial\Omega. \end{aligned} \tag{1.12}$$

Furthermore, in most of earth's magnetic materials, μ exhibits only small variations, and therefore it is common to write

$$\mu = \mu_0(1 + \varkappa(\mathbf{x})),$$

where $\varkappa(\mathbf{x})$ is small. Separate the potential field into $\psi = \psi_0(\mathbf{x}) + \psi_1(\mathbf{x})$, where ψ_0 is the solution to the problem with constant $\mu = \mu_0$; that is, ψ_0 solves the problem

$$\begin{aligned} \mu_0 \Delta \psi_0 &= 0, \quad \psi_0 \in \Omega, \\ \nabla \psi_0 \cdot \vec{n} &= \vec{H}_0 \cdot \vec{n}, \quad \psi_0 \in \partial\Omega. \end{aligned}$$

Substituting $\mu = \mu_0(1 + \varkappa(\mathbf{x}))$ and $\psi = \psi_0 + \psi_1$ into the system (1.12) and dropping the high order term of the form $\varkappa \nabla \psi_1$, we obtain the following equation for ψ_1:

$$\begin{aligned} \Delta \psi_1 &= -\nabla \varkappa \cdot \nabla \psi_0, \quad \psi_1 \in \Omega, \\ \nabla \psi_1 \cdot \vec{n} &= 0, \quad \psi \in \partial\Omega. \end{aligned} \tag{1.13}$$

If we further assume that the magnetic field, $\vec{H}_0$, is uniform within the domain of interest, then we can further simplify the equation to

$$\begin{aligned} \Delta \psi_1 &= -\nabla \varkappa \cdot \vec{H}_0, \quad \psi_1 \in \Omega, \\ \nabla \psi_1 \cdot \vec{n} &= 0, \quad \psi \in \partial\Omega. \end{aligned} \tag{1.14}$$

Since the left-hand side is a simple Laplace equation with boundary conditions at infinity, we can use Green's function for the solution of the system.

$$\psi_1 = -\frac{1}{4\pi}\int_\Omega \frac{\nabla\varkappa(\mathbf{x}')\cdot\vec{H}_0}{|\mathbf{x}-\mathbf{x}'|}\,d\mathbf{x}',$$

which implies that we can write the anomalous magnetic field as

$$\vec{H}_1 = -\frac{1}{4\pi}\int_\Omega \nabla\left(\frac{\nabla\varkappa(\mathbf{x}')\cdot\vec{H}_0}{|\mathbf{x}-\mathbf{x}'|}\right)d\mathbf{x}'.$$

This formula can be used for the computation of the magnetic field due to a *small* perturbation in the magnetic susceptibility μ. For more details on the derivation of potential fields in magnetics, the reader is referred to [11].

Case II: Large perturbation in μ and self-demagnetization

If we assume that there is no source other than the earth's magnetic field and we avoid the assumption of small perturbation in μ, then we are left with the equation

$$\begin{aligned} &\nabla\cdot\mu\nabla\psi = 0, \qquad \psi\in\Omega,\\ &\nabla\psi\cdot\vec{n} = \vec{H}_0\cdot\vec{n}, \qquad \psi\in\partial\Omega. \end{aligned} \tag{1.15}$$

This implies that the field cannot be simply separated into induced field (the earth's field) and the secondary field. This effect is often referred to as self-magnetization. When the effect is strong, the secondary field points to a different direction from the earth's (primary) field.

Case III: Remanent magnetization

The case of remanent magnetization occurs when rocks have magnetic fields that are locked within them, even without any induced fields. This implies that the source $\vec{s}$ in the system (1.10) is not 0. Nonetheless, since $\vec{s}$ is a magnetic source, it must be divergence-free; that is, $\nabla\cdot\vec{s} = 0$ (no magnetic particles).

In this case, one cannot simply transform the vector equations into scalar equations, and we need to solve the vector system (1.10) directly. To continue, note that the equation $\nabla\cdot\mu\vec{H} = 0$ implies that

$$\vec{B} = \mu\vec{H} = \nabla\times\vec{A},$$

where $\vec{A}$ is a vector potential. Since this condition does not determine $\vec{A}$ uniquely, we use the Coulomb gauge condition

$$\nabla\cdot\vec{A} = 0.$$

Substituting $\vec{H}$ into the **curl** equation, we obtain

$$\nabla\times\mu^{-1}\nabla\times\vec{A} = \vec{s}. \tag{1.16}$$

This is a singular system because for any $\vec{A}$ that solves the system, $\vec{A}+\nabla\psi$ also solves the system. However, the system is consistent since the divergence of $\vec{s}$ vanishes.

One can continue similarly to the previous methods by assuming a small perturbation in μ. Nonetheless, while it is relatively easy to solve the forward problem for any μ and

$\vec{s}$, it is difficult to solve the inverse problem, as it requires the evaluation of the source as well as the magnetic susceptibility from the data.

One approach that was suggested is to linearize the problem using $\mu = \mu_0(1+\varkappa)$ and $\vec{A} = \vec{A}_0 + \vec{A}_1$, where $\vec{A}_0$ is the solution of the usual magnetic problem (without the source but with nonhomogeneous boundary conditions) to obtain

$$\nabla\times(1-\varkappa)\nabla\times(\vec{A}_0+\vec{A}_1) = \nabla\times\nabla\times\vec{A}_1 - \nabla\times\varkappa\nabla\times\vec{A}_0 = \mu_0\vec{s}. \tag{1.17}$$

After some manipulations this yields an equation for the secondary magnetic field,

$$\vec{H}_1 = \frac{\mu_0}{4\pi}\int_\Omega \nabla\times\frac{\nabla\times\varkappa\vec{H}_0+\vec{s}}{|\mathbf{x}-\mathbf{x}'|}\,d\mathbf{x}'. \tag{1.18}$$

One idea is to avoid the inversion for $\varkappa$ altogether and introduce the vector quantity $\vec{M}$,

$$\vec{M} = \nabla\times\varkappa\vec{H}_0+\vec{s}.$$

The problem of evaluating $\varkappa$ and $\vec{s}$ is then replaced with that of evaluating $\vec{M}$.

1.2.2 ▪ Statics with electric sources: Direct current and magnetometric resistivity

A very different scenario occurs when the source, $\vec{s}$, is not divergence-free. A nondivergent free source is an *electric source* as it "pumps" electrons into the earth. In this case, the electric field is nonzero, and one needs to solve the system (1.9) to compute the electric potential and the magnetic fields.

A simple approach to decoupling the system is to take the divergence of the first equation in (1.9) to obtain an equation for the potential ϕ:

$$\nabla\cdot\sigma\nabla\phi = -\nabla\cdot\vec{s}. \tag{1.19}$$

Solving for ϕ, we can obtain the electric field $\vec{E} = \nabla\phi$.

Note that ϕ depends on the conductivity alone, and therefore, by measuring the electric field, given a static electric source, it is possible to infer the conductivity of the earth, σ. This is the basis behind the method known as direct current resistivity (DCR), in which one measures the potential field that results from a direct current that is injected into the ground. A cartoon of the method is presented in Figure 1.3.

Assume that the earth has a highly conductive overburden. In this case, the current tends to channel through the overburden and not sample the earth below the overburden. Nonetheless, the magnetic field easily penetrates the overburden, and this can be utilized to study the media under the overburden. This is the idea behind the magnetometric resistivity method (MMR), where magnetic fields are used to recover the conductivity under the conductive overburden.

Assuming ϕ is obtained by solving (1.19), one can recover $\vec{H}$ by solving the system

$$\nabla\times\vec{H} = \vec{s} - \sigma\nabla\phi, \tag{1.20a}$$

$$\nabla\cdot\mu\vec{H} = 0. \tag{1.20b}$$

This is again a magnetostatic equation but with a modified source. It can be solved by the magnetic potential method discussed above (that is, (1.16)). Assuming $\mu \approx \mu_0$ the

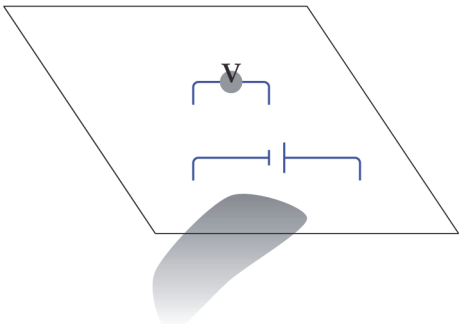

Figure 1.3. *An electric source is used to inject direct current into the earth, and the response of the earth is measured in terms of voltage.*

magnetic field is affected mainly by σ and thus can be recovered from recording magnetic data.

Note that while magnetic fields are nonindicative of the conductivity when the source is divergence-free, the magnetic field is strongly influenced by the conductivity when the source is not divergence-free. This demonstrates the large effect that the source has on the data and our ability to recover specific properties of the media. It also highlights the importance of design when performing a geophysical experiment. Inappropriate design of an experiment can lead to data that are not indicative of the property that one attempts to recover.

1.2.3 ▪ Parabolic approximations: Quasi-static regime and electromagnetic induction

In the previous two sections, we explored static experiments with magnetic and electric sources. In this section, we discuss the second regime in electromagnetic phenomena: induction.

Assume now that the frequency is low, that is, $\omega\mu h \approx 1$. Assume also that $\omega\epsilon h \ll 1$. This is a likely scenario for frequencies ranging roughly from 1–10^4Hz. Since $\mu_0 = 4\pi \times 10^{-7}$ but $\epsilon_0 \approx 8.86 \times 10^{-12}$, at this range of frequencies the term $\omega\epsilon\widehat{\vec{E}}$ or $\epsilon\vec{E}_t$ can be safely neglected, and we obtain a diffusion equation to be solved for the electric and magnetic fields. We write the equations in time and frequency as

$$\begin{aligned} \vec{B}_t &= -\nabla\times\vec{E}, & i\omega\vec{B} &= -\nabla\times\widehat{\vec{E}}, \\ 0 &= \nabla\times\mu^{-1}\vec{B} - \sigma\vec{E} - \vec{s}, & 0 &= \nabla\times\mu^{-1}\widehat{\vec{B}} - \sigma\widehat{\vec{E}} - \widehat{\vec{s}}. \end{aligned} \tag{1.21a}$$

These systems are often referred to as the quasi-static approximation to Maxwell's equations. It is possible to eliminate the magnetic field in frequency $\widehat{\vec{B}}$ or in time $\vec{B}$ from the system and obtain a system for the electric field:

$$\nabla\times\mu^{-1}\nabla\times\vec{E} + \sigma\vec{E}_t = -\vec{s}_t, \qquad \nabla\times\mu^{-1}\nabla\times\vec{E} + i\omega\sigma\vec{E} = -i\omega\vec{s}.$$

It is straightforward to show that, qualitatively, the term $\nabla\times\mu^{-1}\nabla\times$ behaves similarly to $-\mu\vec{\Delta}$, and this demonstrates the qualitative behavior of the system as a heat equation for $\vec{E}$.

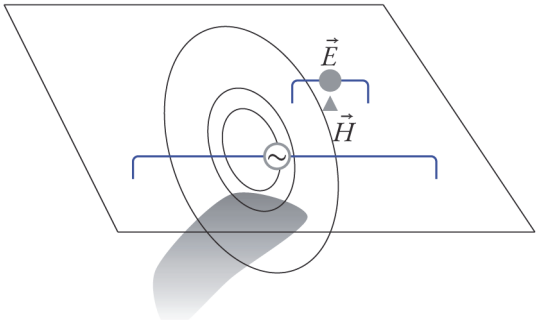

Figure 1.4. *An alternating electric source injects currents into the earth, and the response of the earth is measured in terms of voltage and magnetic fields.*

The system (1.21) in time is an example of a *parabolic differential equation*. Loosely speaking, parabolic equations are partial differential equations that involve a spatial elliptic operator and a first order time derivative. We again refer the reader to [26] for a more precise definition. The solution of the system (assuming no source) decays in time, with high spatial frequencies decaying faster than lower frequencies.

A number of geophysical methods can be described by this set of equations, and the behavior of the solution strongly depends on the type of sources. We now briefly review these techniques.

Controlled source audio magnetotelluric (CSAMT)

An extension to the DCR problem is the controlled source audio magnetotelluric (CSAMT) problem. In this method, the geometry of the experiment is similar to that in DCR; however, the source is alternating rather than direct (see Figure 1.4). This implies that the source serves as a current source as well as an antenna that transmits electromagnetic radiation.

In this experiment, a number of frequencies are transmitted, typically ranging from a few Hz to a few thousands of Hz. The receivers can measure the voltage between grounded points (the electric fields) or the magnetic fields.

Seabed electromagnetic logging

A straightforward implementation of CSAMT in an ocean environment is the seabed logging experiment. A boat drags a line source that transmits an electromagnetic (EM) wave into the water (see Figure 1.5). Receivers are placed on the ocean floor and record electric and magnetic fields. From the basic principles this experiment shares the same properties as CSAMT. The main differences are that the transmitted frequencies are lower (to be able to penetrate highly conductive sea water) and that the number of sources is very large due to the ability to easily move the source.

Large loop and airborne electromagnetics

A different geophysical experiment is carried out using inductive, that is, magnetic, sources that are divergence-free. In this case no current is injected into the earth, and the currents are generated by induction. There are a number of such experiments with different geometries but with similar physical attributes.

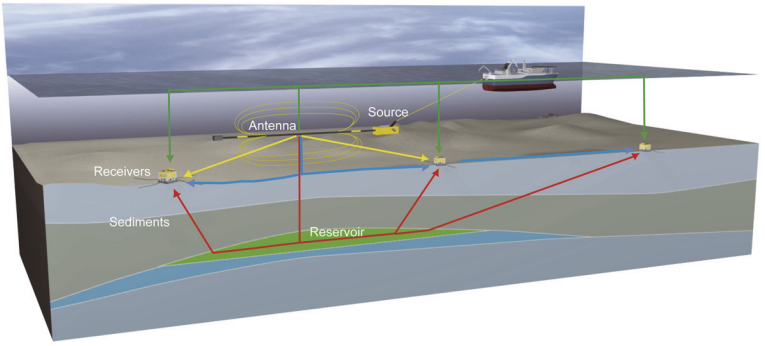

Figure 1.5. *Seabed logging experiment (image from NOCA). A line source is dragged by a boat and transmits low frequency electromagnetic waves. Receivers are placed on the ocean floor, and electric and magnetic fields are measured.*

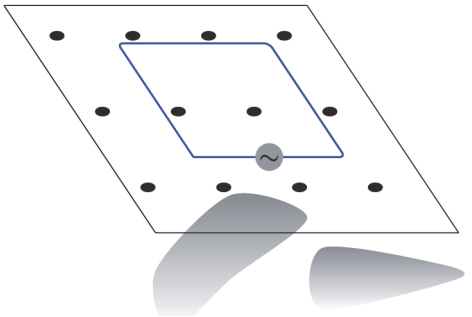

Figure 1.6. *A magnetic source is generated by using an alternating current in a loop.*

The basic experiment involves a loop above the earth that carries some nonstationary currents. The magnetic source generates inductive currents within the earth that in turn interact with the earth's structure. The secondary magnetic fields are typically measured.

Two such experiments are commonly done. In the first, a large loop is laid on top of the earth, and a number of magnetic receivers are placed on top of the earth, as well (see Figure 1.6). Typically, some time dependent signal is transmitted through the loop, and the magnetic field is recorded. One common signal to be transmitted is a step-off function, and the measurements are done mainly in the off-time. The advantages of a large loop experiment are that it can energize a large, deep volume and that the measurements can be done at very late times, allowing for the identification of large conductors that are buried deep.

A second, more common, experiment is performed with a small loop, typically carried by an airplane or a helicopter. A single receiver is typically carried by the same platform, and either it records in the center of the transmitter approximating a coincident loop (that is, the transmitting loop is also the receiving loop) or it is carried behind the moving platform. A signal is transmitted every few meters and is recorded by the receiver. An example of such a survey is in Figure 1.7. These surveys cover an extensive amount of ground and contain millions of transmitter-receiver combinations. The scale and the amount of data make these surveys very difficult to deal with using standard inversion

Figure 1.7. *Airborne electromagnetic survey (image from GeoTech web site).*

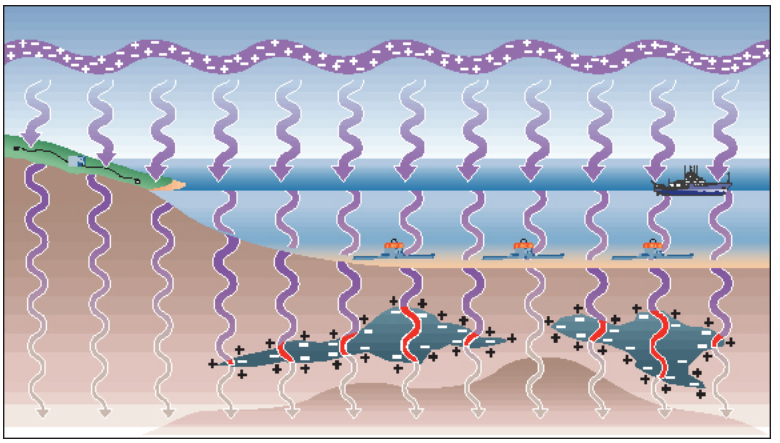

Figure 1.8. *Magnetotelluric survey (image from SIO publication Explorations, v4, 1997).*

techniques.

Natural sources: Magnetotelluric and ZTEM

One of the oldest techniques in electromagnetics is the magnetotelluric (MT) method. A related, more recent, method that uses the same physical principles is the airborne ZTEM method. In both methods, the sources are natural electromagnetic fields that can be modeled as plane waves. Since the waves can be of very low frequency and with high energy, it is possible to study very deep targets using these techniques. A sketch of the method is plotted in Figure 1.8.

The difficulty with natural source methods is that the source strength is unknown. This can be overcome by noting that since the equations are linear, the source simply scales all components by a constant number. Therefore, rather than inverting the fields, it is common to invert the ratio of the fields. For example, in classical one-dimensional (1D) processing, assuming that the data result from an electric wave that points to the

x-direction, it is possible to measure $\vec{E}_x$ as well as $\vec{B}_y$, and the data are simply the ratio between the fields.

In the ZTEM method, the magnetic field is recorded in the z-direction in the air. In this case the data are the ratios $\vec{B}_z/\vec{B}_x$ and $\vec{B}_z/\vec{B}_y$.

1.2.4 ▪ Hyperbolic regime: Ground penetrating radar and radio imaging

So far, we have discussed methods that evolve from statics, that is, neglecting the time dependent terms from the equations, and quasi-static, that is, neglecting the terms associated with $\epsilon\partial_t$ from the equations. However, for quickly changing fields, the term $\epsilon\omega^2$ is similar in magnitude to the term $\mu\omega\sigma$ and thus cannot be neglected. This typically occurs for frequencies ranging from a few kHz and above. In this case, the appropriate system to solve is the full Maxwell system (1.1). The equations can be rewritten in terms of the electric field as

$$\begin{aligned} &\nabla\times\mu^{-1}\nabla\times\vec{E}+\sigma\vec{E}_t+\epsilon\vec{E}_{tt}=-\vec{s}_t, \\ &\nabla\times\mu^{-1}\nabla\times\widehat{\vec{E}}+i\omega\sigma\widehat{\vec{E}}-\omega^2\epsilon\widehat{\vec{E}}=-i\omega\vec{s}. \end{aligned} \tag{1.22}$$

This is a wave equation with a lossy term that is governed by the conductivity.

The time dependent version of the system is an example of a *hyperbolic differential equation*. Purely hyperbolic PDEs (that is, without diffusion) are characterized by conservation, that is, unlike elliptic and parabolic equations, where high frequencies decay, purely hyperbolic PDEs conserve and redistribute them. The lack of smoothing of the solution makes the numerical treatment of such problems very different from the elliptic and parabolic cases and much more difficult.

For sufficiently high frequencies, the equation behaves as a wave equation where the main influence on the data is the dielectric permittivity ϵ. This is the basis for ground penetrating radar (GPR) and for radio imaging (RIM). In both experiments, a high frequency signal is transmitted and the electromagnetic echo is recorded. Although the problem is of interest, its characters are rather different from the rest of the problems covered here, and we point the reader to the book [70] and references within.

1.2.5 ▪ Frequency dependent conductivity and induced polarization

Maxwell's equations are stated as relations between the electric and magnetic fields, $\vec{E}$ and $\vec{H}$, to the magnetic and electric fluxes, $\vec{B}$ and $\vec{J}$. The system is closed by considering the constitutive relations (1.3).

However, it has been observed that in many cases, these constitutive relations are not adequate to describe the earth's response to the fields. In particular, it has been observed that Ohm's law $\widehat{\vec{J}}=\sigma\widehat{\vec{E}}$ should be replaced. The physical phenomenon that is often observed is charge accumulation in rocks that serve as capacitors. The charge then decays, and this introduces an effective change to Ohm's law that does not assume capacitance.

A number of phenomenological formulas have been proposed to describe this behavior. Perhaps the most commonly used is the Cole–Cole model [18] that states that we measure a frequency dependent conductivity. The equation is typically expressed as a function of the resistivity $\rho=\sigma^{-1}$,

$$\rho(\omega)=\rho_0\left(1-\eta\left(1-\frac{1}{1+(i\omega\tau)^c}\right)\right), \tag{1.23}$$

where ρ_0 is the resistivity at 0 frequency. The parameter η is the chargeability, τ is a time constant, and c helps to control the shape of the conductivity-frequency curve. The main parameter of interest is the chargeability, η, as it has been documented that it is highly indicative of some mineralizations.

Transforming the equation into the time domain, Ohm's law that involves a product of $\rho(\omega)$ and $\widehat{\vec{J}}$ becomes a convolution in time. This complicates the system, as the relation between $\vec{E}$ and $\vec{J}$ in time is an integral over all times.

However, the situation simplified significantly when $c = 1$, the so-called Debye model [68]. In this case, we can rearrange Ohm's law into

$$\rho_0\left(1-\frac{i\omega\tau\eta}{1+i\omega\tau}\right)\vec{J}=\vec{E},$$

$$(1+i\omega\tau(1-\eta))\vec{J}=\sigma_0(1+i\omega\tau)\vec{E},$$

and then, recalling that a product with $i\omega$ is differentiation in time, we obtain

$$\vec{J}+\tau(1-\eta)\vec{J}_t=\sigma_0\left(\vec{E}+\tau\vec{E}_t\right). \tag{1.24}$$

This is a local relation (in time) between $\vec{E}$ and $\vec{J}$ and therefore is much easier to work with. Furthermore, it has been shown [68, sect. 7.5] that the Cole–Cole effect with any c can be obtained by a linear combination of Debye models. This makes the use of the Debye model preferable from a computational point of view.

In order to measure chargeability, it is common to slightly modify the DCR experiment and introduce a step function as the source to the DCR. To explain how this source can be used to approximately determine the chargeability, we rewrite Maxwell's equations in frequency in terms of the electric field

$$\nabla\times\mu^{-1}\nabla\times\widehat{\vec{E}}+i\omega\sigma(\omega)\widehat{\vec{E}}=-i\omega\vec{s}.$$

A useful tool to study the electric field is the Helmholtz decomposition. We decompose $\widehat{\vec{E}}$ into a *curl*-free component and a *divergence*-free component, that is,

$$\widehat{\vec{E}}=\vec{A}+\nabla\phi,\qquad \nabla\cdot\vec{A}=0.$$

The condition $\nabla\cdot\vec{A}=0$ is known as the Coulomb gage condition. It can be easily shown that, given appropriate boundary conditions for the vector potential, $\vec{A}$, and the scalar potential, ϕ, this decomposition is unique. Substituting the decomposition into the equations, we obtain

$$\nabla\times\mu^{-1}\nabla\times\vec{A}+i\omega\sigma(\omega)\vec{A}+i\omega\sigma(\omega)\nabla\phi=-i\omega\vec{s}. \tag{1.25}$$

Assume now that ω is small. The term $\omega\sigma\vec{A}$ is a zeroth order differential operator, the term $\omega\sigma(\omega)\nabla\phi$ is a first order term, and finally, the term $\nabla\times\mu^{-1}\nabla\times\vec{A}$ is a second order term. If the source is magnetic, that is, divergence-free, then at first order $\vec{A}$ solves the system

$$\nabla\times\mu^{-1}\nabla\times\vec{A}=\vec{s}.$$

This implies that for a divergence-free source, the vector potential $\vec{A}$ and the fields are mainly determined by the source and not by the conductivity. If the source is not divergence-free, then the equation $\nabla\times\mu^{-1}\nabla\times\vec{A}=\vec{s}$ is inconsistent, and the next term in the equation is needed. Dropping the zeroth order term, we obtain

$$\nabla\times\mu^{-1}\nabla\times\vec{A}+i\omega\sigma(\omega)\nabla\phi=-i\omega\vec{s}. \tag{1.26}$$

Note the similarity of this system to the magnetostatic case (1.20). Indeed, noting that

$$i\omega\mu\widehat{\vec{H}}=-\nabla\times\widehat{\vec{E}}=-\nabla\times\vec{A},$$

and recalling that there are no magnetic charges, we can write the equation as a system in $\widehat{\vec{H}}$ and ϕ,

$$\begin{aligned}&\nabla\times\widehat{\vec{H}}-\sigma(\omega)\nabla\phi=\vec{s},\\&\nabla\cdot\mu\widehat{\vec{H}}=0,\end{aligned} \tag{1.27}$$

which is equivalent to the static system (1.20) but with frequency dependent conductivity.

The observation that for slowly changing fields with electric sources one can drop the vector potential $\vec{A}$ has important implications. Taking the *divergence* of (1.27), we obtain the electrostatic equation

$$\nabla\cdot\sigma(\omega)\nabla\phi=-\nabla\cdot\vec{s}. \tag{1.28}$$

Many data collection systems collect the data in time. We can use the Debye model and obtain an equivalent equation in time.

$$\begin{aligned}&\nabla\cdot\vec{J}=-\nabla\cdot\vec{s},\\&\vec{J}+\tau(1-\eta)\vec{J}_t=\sigma_0(\nabla\phi+\tau\nabla\phi_t).\end{aligned} \tag{1.29}$$

This equation can be used in order to model direct current induced polarization (DCIP) decay curves.

The Debye model works well for $c=1$. When $c\neq 1$ one cannot simply transform the equation into time. Reorganizing Ohm's law for $c\neq 1$, we obtain

$$(1+(i\omega\tau)^c)\widehat{\vec{J}}=\sigma_0(1-\eta)(1-\eta+(i\omega\tau)^c)\widehat{\vec{E}}, \tag{1.30}$$

and we can transform the equation into time by using fractional derivatives:

$$\mathscr{F}^{-1}(i\omega\tau)^c\widehat{\vec{J}}=\frac{\tau^c}{\Gamma(1-c)}\partial_t\int_0^t\frac{\vec{J}(\xi)}{(t-\xi)^c}\,d\xi. \tag{1.31}$$

An altogether different approach is to replace the Debye decay with a stretched exponential function. The stretched exponential has been proposed as an alternative to the Cole–Cole model [82, 1, 42, 53]. The stretched exponential has the form

$$f(t)=\exp\left(\frac{-t^\beta}{\tau}\right),$$

where $0 < \beta \leq 1$ is a constant parameter. The associated differential equation is

$$f_t + \tau^{-1}\beta t^{\beta-1} f = 0.$$

Note that this differential equation is similar to the Debye model with time dependent coefficients. Thus, to approximate the decay of $\vec{E}$ and $\vec{J}$ we replace the differential equation generated by the Debye model with the differential equation that is generated using the stretched exponential. This leads to the ordinary differential equation

$$(1-\eta)\vec{J}_t + \tau^{-1}\beta t^{\beta-1}\vec{J} = \sigma(\vec{E}_t + \tau^{-1}\beta t^{\beta-1}\vec{E}), \tag{1.32}$$

which can be coupled to Maxwell's equations

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \quad \text{and} \quad \nabla \times \mu^{-1}\vec{B} - \vec{J} = \vec{s},$$

obtaining a system for $\vec{E}, \vec{B}$, and $\vec{J}$.

The different parameter regimes discussed above are known in geophysical electromagnetic methods by different names. First, if $\omega \to 0$, then $\sigma(\omega) \to \sigma_0$, and the problem is considered to be static. Second, if $\omega \gg 0$ but is sufficiently small, then one can solve (1.28) and ignore the vector potential $\vec{A}$. This is referred to as decoupled EM signals. Finally, if ω is larger, then one cannot decouple the EM signal into a scalar and a vector potential, and this is referred to as EM coupling.

The advantage of the decoupled system is that we can use the simpler system to evaluate the chargeability. Assume that we perform two experiments for two different near-by low frequencies ω_1 and ω_2, obtaining potentials ϕ_1 and ϕ_2. Assuming that $\sigma(\omega_2) = \sigma(\omega_1)(1+\gamma)$, we obtain that

$$\nabla \cdot \sigma(\omega_1)\nabla(\phi_2 - \phi_1) \approx -\nabla(\gamma\sigma(\omega_1)) \cdot \nabla\phi_1, \tag{1.33}$$

and therefore, if we have an estimate for σ_1, we can estimate γ, which is a function of the chargeability parameters. This is the basic physics behind the induced polarization method (IP). As we see, the method strongly depends on our ability to decouple the electric field, and therefore, much attention has been given to the so-called EM coupling problem. The derivation above highlights the fact that for high frequencies the problem cannot be decoupled, and any treatment of the problem requires the solution of the coupled system.

1.3 ▪ Electromagnetic fields, sources, and the EM coupling problem

In the above geophysical techniques, we can differentiate between two types of sources. First, sources can be inductive, that is, divergence-free, and second, the source can be galvanic, that is, have a significant divergence. From a physical point of view, inductive sources do not inject current into the media, while galvanic sources do.

The fields that correspond to these two different sources, especially at low frequencies, have fundamentally different characters. Consider again the Helmholtz decomposition of the electric field

$$\vec{E} = \vec{A} + \nabla\phi, \quad \nabla \cdot \vec{A} = 0.$$

The Coulomb gauge condition, $\nabla \cdot \vec{A} = 0$, implies that $\vec{A}$ does not have components in the null space of the curl, while $\nabla\phi$ does not have any components in the active space of the curl. Maxwell's equation for low frequencies can be written as

$$\nabla \times \mu^{-1} \nabla \times \vec{A} + i\omega\sigma\vec{A} + i\omega\sigma\nabla\phi = -i\omega\vec{s},$$

and its divergence is

$$\nabla \cdot \sigma\vec{A} + \nabla \cdot \sigma\nabla\phi = -\nabla \cdot \vec{s}.$$

Assume for a moment that the source is inductive; that is, $\nabla \cdot \vec{s} = 0$. We obtain that

$$\nabla \cdot \sigma\nabla\phi = -\nabla \cdot \sigma\vec{A},$$

and since the operator on the left is second order, while the operator on the right is only first order, the contribution of $\vec{A}$ to the electric field is much larger than the contribution of ϕ. This implies that the electric field is mainly inductive; that is, it is not affected much by charge accumulation.

On the other hand, assume for a moment that $\nabla \cdot \vec{s}$ is large. In this case the main contributor to the electric field is $\nabla\phi$, which implies that the electric field is mainly generated by charge accumulation.

In some cases, $\nabla \cdot \vec{s}$ is large but the term $\sigma\vec{A}$ is significant. This clearly occurs when σ has large gradients. In this case the field is influenced by both charge accumulation and induction. This scenario leads to the infamous EM coupling problem where static approximations to Maxwell's equations fail.

1.4 ▪ Some useful analytic solutions

We now briefly discuss some analytical solutions to the problem. These solutions can be used to test numerical methods and thus serve as important "quality control" and benchmarks to many numerical codes. A wonderful collection of analytic solutions can be found in [78]. Here we collect a small number that we find useful for testing our simulation codes.

1.4.1 ▪ DCR over a uniform earth

Assume that the DCR experiment is performed with a single pole for a source. If the earth has uniform conductivity σ, then the field is the solution of the Poisson equation in a half-space

$$\sigma\Delta\phi = \delta(\mathbf{x} - \mathbf{x}_0),$$

and the solution is

$$\phi = \frac{1}{2\pi\sigma|\mathbf{x} - \mathbf{x}_0|}. \tag{1.34}$$

Since the problem is linear with respect to the source, one can use this basic solution to build the response of a uniform earth with respect to a dipole or other more exotic arrangement of dipoles.

1.4.2 ▪ Magnetotelluric response of a half-space

Another easy problem is the 1D magnetotelluric problem. In this case, the system reduces to

$$\begin{aligned} &i\omega\hat{\vec{B}}_y = -\partial_z \vec{E}_x, \\ &\partial_z(\mu^{-1}\vec{B}_y) + \sigma(z)\vec{E}_x = 0, \\ &\vec{E}_x(-\infty) = 0, \quad \vec{E}_x(0) = 1. \end{aligned}$$

Eliminating $\vec{B}_y$ and assuming μ is constant, we obtain

$$\partial_{zz}\vec{E}_x - i\omega\mu\sigma(z)\vec{E}_x = 0.$$

Now assume that the conductivity σ is

$$\sigma(z) = \begin{cases} 0, & -L \le z \le 0, \\ \sigma_0 & \text{otherwise.} \end{cases}$$

The solution in the air is therefore linear (with a constant 1 for the boundary condition),

$$\vec{E}_x = az + 1,$$

and the solution in the earth is

$$\vec{E}_x = c\exp(ikz),$$

with $k = \sqrt{-i\omega\mu\sigma}$. The coefficients a and c are matched to have continuous fields and continuous derivatives obtaining

$$a = \frac{ik}{1+ikL}, \qquad c = \frac{\exp(ikL)}{1+ikL}.$$

The response of a 1D earth to a plane wave has some fundamental features that need to be discussed. In particular, note that the solution is of the form

$$\exp(ikz) = \exp(k_i z)\exp(ik_r z),$$

where we decompose the complex k into real and imaginary parts $k = k_r + ik_i$. This implies that the solution possesses two main properties. First, it decays, and second, it oscillates. Assuming $\mu = \mu_0$, it is easy to show that the amplitude of the wave becomes $\exp(-1)$ of its initial amplitude when

$$z = s_d \approx 500\sqrt{(\sigma\omega)^{-1}}. \tag{1.35}$$

The depth, s_d, is known as skin depth, and it plays a major role in the understanding of diffused electromagnetic signals. As we will see later, this depth can be used to design the mesh that the problem is solved on as well as in understanding the data.

1.4.3 ▪ The frequency response of a half-space to a magnetic dipole

Another analytic solution can be obtained for the magnetic field that results from a vertical magnetic dipole over a uniform earth, assuming that the air has 0 conductivity. In this case, the z component of the magnetic field on the earth's surface is

$$H_z = \frac{m}{2\pi k^2 \rho^5}\left(9-(9+9ik\rho-4k^2\rho^2-ik^3\rho^3)\exp(-ik\rho)\right), \tag{1.36}$$

where $k=\sqrt{i\omega\mu\sigma}$ and ρ is the distance to the loop. This is a nontrivial expression that can be used to test frequency dependent codes.

There are a number of interesting features that can be observed when looking at the analytical solution.

Consider the case of low frequencies $\omega \ll 1$ and hence $k \ll 1$. We can expand the exponent as

$$\exp(-ik\rho) \approx 1-ik\rho,$$

and neglecting higher order terms we can approximate the field as

$$H_z \approx \frac{m}{2\pi k^2 \rho^5}\left(9-9(1+ik\rho)(1-ik\rho)\right).$$

Opening brackets and rearranging, we obtain

$$H_z \approx \frac{9m}{2\pi\rho^3}.$$

That is, for low frequency, the first order term is real and does not depend on the frequency or conductivity. This should not come as a surprise as the response of Maxwell's equations for 0 frequency does not depend on the conductivity.

1.4.4 ▪ The frequency response of full and half-space to an electric dipole

Another important scenario is the case of electric dipole of length δs pointing in the x-direction, imbedded in a conductive media with conductivity σ. In this case the electric field is

$$\vec{E} = \frac{I\delta s \exp(ikr)}{4\pi\sigma r^3}\left((-k^2r^2+3ikr+3)\begin{pmatrix}\frac{x^2}{r^2}\\ \frac{xy}{r^2}\\ \frac{xz}{r^2}\end{pmatrix}+(k^2r^2-ikr-1)\begin{pmatrix}1\\0\\0\end{pmatrix}\right), \tag{1.37}$$

where r is the distance from the dipole and $k=\sqrt{i\omega\mu\sigma}$.

In the case of a half-space there is no closed expression for the solution everywhere. However, given an x-direction dipole on the surface of the earth, we can compute the x-directed electric field on the earth to be

$$\vec{E}_x = \frac{I\delta s}{2\pi\sigma\rho^3}\left(-2+(ik\rho+1)\exp(-ik\rho)+\frac{3x^2}{\rho^2}\right). \tag{1.38}$$

1.5 ▪ Exercises

1. Prove the skin depth formula (1.35).
2. Use (1.36) to analyze the behavior of the magnetic field at low frequency ($\omega \ll 1$). How do the real and imaginary parts of the field depend on the conductivity?
3. Use (1.38) to analyze the behavior of the electric field at low frequency ($\omega \ll 1$). How do the real and imaginary parts of the field depend on the conductivity? How does their behavior compare to the behavior of the magnetic field at low frequencies?

Chapter 2

The Finite Volume Method in 1D and Its Application to Maxwell's Equations

In this chapter, we discuss a finite volume discretization for some simple problems. As we see next, the 1D is in the core of two-dimensional (2D) and three-dimensional (3D) computations, and therefore we give much attention to the 1D problem, where things are much simpler. Our derivation follows mimetic finite volume methods presented in [47, 48, 46, 45] which can be thought of as an extension to the famous Yee discretization [83] used in most finite difference/volume discretizations of Maxwell's equations.

At this point we would like to note that although finite element methods (FEM) are also common and can be used for the discretization, we choose to not discuss them in this exposition for the sake of brevity. We would like to note that in spite of the (sometimes) heated discussion about the advantages and disadvantages of FEM compared with the finite volume method, the methods are very similar, especially if they are both derived from the weak form. We do believe not that one technique is superior to the other but that this is really a question of personal taste.

2.1 ▪ Deriving the discretization

Maxwell's equations in the 1D medium can be written as

$$\begin{aligned} i\omega b &= -\partial_z e, \\ s &= \partial_z(\mu^{-1} b) - \sigma(z) e, \end{aligned}$$

where $e = \widehat{\vec{E}}_x$ and $b = \widehat{\vec{B}}_y$.

We rewrite the equations in weak form as

$$i\omega(b, f) = -(\partial_z e, f), \tag{2.1a}$$

$$(s, w) = -(\mu^{-1} b, \partial_z w) - (\sigma(z) e, w), \tag{2.1b}$$

where f is in the same functional space as b and w is in the same functional space as e. Note the minus sign in the inner product $(\mu^{-1} b, \partial_z w)$ that results from the integration by parts.

To discretize the system, we consider a simple 1D mesh plotted in Figure 2.1. The functions e and b are discretized by grid functions $\mathbf{e}$ and $\mathbf{b}$. Here we face our first dilemma. At which points should we discretize the fields? To answer this question, consider the inner product on a cell with uniform material properties, that is, an interval $[x_{k-\frac{1}{2}}, x_{k+\frac{1}{2}}]$.

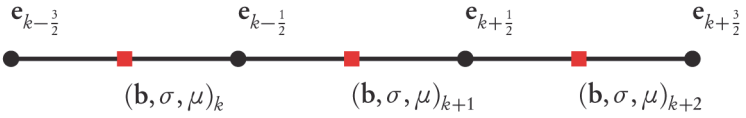

Figure 2.1. *1D grid for nodal discretization.*

A natural way to discretize the derivative of a quantity u that is discretized on the nodes is

$$(\partial_z u)_{x_k} \approx h_k^{-1}(\mathbf{u}_{k+\frac{1}{2}} - \mathbf{u}_{k-\frac{1}{2}}) + \mathcal{O}(h^2).$$

Order: We say that v_h converges to u in order h^p if $|v_h - u| = Ch^p$, where C is independent of h.

Since the derivatives in the weak form operate on the electric field, e, it makes sense to discretize e on the nodes of our mesh, and since $\partial_z e$ "lives" in the same place where the magnetic flux, b, "lives," this implies that b should be discretized on the cell center of our mesh, leading to the well-known *staggered grid*.

To complete the discretization we note that we need to compute two inner products. One involves a cell centered variable, and the other involves a nodal variable. The derivation of inner products for variables that live in the cell center is straightforward. Consider two grid functions $\mathbf{b}$ and $\mathbf{f}$ discretized on cell centers. Using the midpoint rule, we obtain

$$(b,f) \approx \sum_k h_k \mathbf{b}_k \mathbf{f}_k + \mathcal{O}(h^2).$$

For the discretization of inner products that involve with nodal variables, one must be a little bit more careful. Consider the inner product $(\sigma e, w)$ and a nodal discretization of the functions by grid functions $\mathbf{e}$ and $\mathbf{w}$. A second order accurate discretization of this product can be written as

$$(\sigma e, w) \approx \sum_k \frac{h_k \sigma_k}{4} (\mathbf{e}_{k+\frac{1}{2}} + \mathbf{e}_{k+\frac{1}{2}})(\mathbf{w}_{k+\frac{1}{2}} + \mathbf{w}_{k+\frac{1}{2}}) + \mathcal{O}(h^2).$$

That is, we average e and w to the cell center and then compute the inner product. However, this inner product is *inappropriate* to use. When considering an inner product we need to note that the inner product is not only converging to the true value but also that it does not contain nontrivial null spaces. Consider the inner product $(\sigma w, w)$ for a nontrivial w and a positive σ. Clearly, this product must be greater than zero. However, the above inner product does not yield this property! Consider the vector $\mathbf{w} = (-1, 1, -1, \ldots)$. Then clearly this approximation yields $(\sigma w, w) \approx 0$, and this is clearly a wrong approximation for a highly oscillating function. An inner product with the same accuracy but with no null space can be obtained by considering the approximation

$$(\sigma e, w) \approx \sum_k \frac{h_k \sigma_k}{4} (\mathbf{e}_{k+\frac{1}{2}} \mathbf{w}_{k+\frac{1}{2}} + \mathbf{e}_{k+\frac{1}{2}} \mathbf{w}_{k+\frac{1}{2}}) + \mathcal{O}(h^2). \tag{2.2}$$

This phenomenon is a fundamental observation in discretization techniques, and it leads to the following principle for many numerical methods:

When some function $f(w)$ needs to be averaged/interpolated and grid function $\mathbf{w}$ is used for the discretization, it is better to first compute (pointwise) $f(\mathbf{w})$ and only then average/interpolate.

This principle will be used later when we discretize the inverse problem as well.

2.2 ▪ Matrix representation

Matrix representation of the equations allows us to rewrite the equations in a much simpler and more apparent form. To this end, we consider first the discretization of the gradient of the nodal function e. The matrix form of the gradient is simply

$$\mathsf{GRAD}\,\mathbf{e} = \begin{pmatrix} h_1^{-1} & & & \\ & h_2^{-1} & & \\ & & \ddots & \\ & & & h_n^{-1} \end{pmatrix} \begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{e}_1 \\ \vdots \\ \\ \mathbf{e}_{n+1} \end{pmatrix},$$

which can be written in matrix form as

$$e_z \approx \mathbf{L}^{-1}\mathbf{G}\mathbf{e},$$

where $\mathbf{L} = \mathrm{diag}(\mathbf{h})$ contains the length of each edge and $\mathbf{G}$ is a matrix with values of ± 1 that represents the topology of our mesh. As we will see later, this representation is common for 3D as well as 1D and therefore is very useful.

Next, consider the inner products. It is straightforward to verify that the cell center inner products have the form

$$(b,f) \approx \mathbf{b}^\top \mathbf{M}^f \mathbf{f} \quad \text{and} \quad (\mu^{-1}b,f) \approx \mathbf{b}^\top \mathbf{M}^f_\mu \mathbf{f},$$

where $\mathbf{M}^f_\mu = \mathrm{diag}(\mathbf{h} \odot \mu^{-1})$ and $\mathbf{M}^f = \mathrm{diag}(\mathbf{h})$. To express the nodal averaging in matrix form we note that

$$\sum_k \frac{h_k \sigma_k}{4}(\mathbf{e}_{k+\frac{1}{2}}\mathbf{w}_{k+\frac{1}{2}} + \mathbf{e}_{k+\frac{1}{2}}\mathbf{w}_{k+\frac{1}{2}}) = (\mathbf{h}\odot\sigma)^\top(\mathbf{A}_v(\mathbf{e}\odot\mathbf{w})) = \mathbf{w}^\top \mathrm{diag}(A_v^\top(\mathbf{h}\odot\sigma))\mathbf{e},$$

where $\odot$ is the pointwise Hadamard product and $\mathbf{A}_v$ is an averaging matrix from nodes to cell center:

$$\mathbf{A}_v = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & & \\ & \ddots & \ddots & \\ & & \frac{1}{2} & \frac{1}{2} \end{pmatrix}.$$

We define the matrix

$$\mathbf{M}^e_\sigma = \mathrm{diag}(A_v^\top(\mathbf{h}\odot\sigma)).$$

Using the above matrices, we obtain the following discretization for 1D Maxwell's equations in weak form:

$$\begin{aligned} i\omega\,\mathbf{f}^\top\mathbf{M}^f\mathbf{b} &= -\mathbf{f}^\top\mathbf{M}^f\mathbf{L}^{-1}\mathbf{G}\mathbf{e}, \\ \mathbf{w}^\top\mathbf{M}^e\mathbf{s} &= -\mathbf{w}^\top\mathbf{G}^\top\mathbf{L}^{-1}\mathbf{M}^f_\mu\mathbf{b} - \mathbf{w}^\top\mathbf{M}^e_\sigma\mathbf{e}. \end{aligned}$$

These equations are discretized for arbitrary grid functions $\mathbf{w}$ and $\mathbf{f}$, and therefore we must have that

$$i\omega\,\mathbf{b} = -\mathbf{L}^{-1}\mathbf{G}\mathbf{e}, \tag{2.3a}$$

$$\mathbf{M}^e\mathbf{s} = -\mathbf{G}^\top\mathbf{L}^{-1}\mathbf{M}^f_\mu\mathbf{b} - \mathbf{M}^e_\sigma\mathbf{e}. \tag{2.3b}$$

We can now eliminate $\mathbf{b}$ from the equations, obtaining an equation for $\mathbf{e}$ only:

$$(\mathbf{G}^\top\mathbf{L}^{-1}\mathbf{M}^f_\mu\mathbf{L}^{-1}\mathbf{G} + i\omega\mathbf{M}^e_\sigma)\mathbf{e} = -i\omega\mathbf{M}^e\mathbf{s}. \tag{2.4}$$

2.3 ▪ Implementation of boundary conditions

In the above we did not discuss the implementation of boundary conditions. Homogeneous boundary conditions for b are incorporated into the discretization. This can be seen from the inner product

$$(\partial_z\mu^{-1}b, w) = -(\mu^{-1}b, \partial_z w) + (\mu^{-1}bw)|_0^L,$$

where we have assumed that the domain of integration is $[0, L]$. Previously, we have dropped the boundary conditions implying $b = 0$ at the ends of the interval. In the case that the boundary conditions are not homogeneous, we need to incorporate the conditions into the discretization. In matrix-vector form, this condition can be written as

$$(\mu^{-1}bw)|_0^L = \mathbf{b}_{\text{bc}}^\top(\mathbf{B}\mathbf{w}),$$

where the matrix $\mathbf{B}$ extracts the boundary elements in $\mathbf{w}$ and $\mathbf{b}_{\text{bc}}$ are the known boundary conditions. For the simple 1D case here we have

$$\mathbf{B} = \begin{pmatrix} -1 & 0 \\ \vdots & \vdots \\ 0 & 1 \end{pmatrix}.$$

This implies that equation (2.1b) in the weak form is changed to

$$\mathbf{M}^e\mathbf{s} = -\mathbf{G}^\top\mathbf{L}^{-1}\mathbf{M}^f_\mu\mathbf{b} - \mathbf{M}^e_\mu\mathbf{e} + \mathbf{B}^\top\mathbf{b}_{\text{bc}}$$

and that the boundary conditions play the role of a source.

Next we discuss the implementation of the boundary conditions on e, which are straightforward to implement. We rewrite the system that was obtained without boundary conditions on $\mathbf{e}$ as

$$\mathbf{A}\mathbf{e} = \mathbf{q}.$$

The boundary condition on $\mathbf{e}$ implies that we know two elements in $\mathbf{e}$, $\mathbf{e}_1$ and $\mathbf{e}_{\text{end}}$. We can partition the unknowns as $\mathbf{e} = [\mathbf{e}_I^\top, \mathbf{e}_B^\top]^\top$, where $\mathbf{e}_B$ are the boundary unknowns and $\mathbf{e}_I$ are the internal nodes. We can further partition the matrix as

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{II} & \mathbf{A}_{IB} \\ \mathbf{A}_{BI} & \mathbf{A}_{BB} \end{pmatrix}.$$

Since the boundary conditions are known, the values of $\mathbf{e}_B$ are known, and we can reduce the system to obtain the equation for the known boundary conditions

$$\mathbf{A}_{II}\mathbf{e}_I = \mathbf{q}_I - \mathbf{A}_{IB}\mathbf{e}_B.$$

Finally, boundary conditions of the form $\alpha e + b = c$ can be applied, and this is left as a problem for the reader.

2.4 ▪ 1D in practice

We now give our attention to the actual implementation of the 1D problem using MATLAB. Notice that the following matrices need to be generated:

- A difference matrix from cells to nodes, $\mathbf{G}$: nodes $\rightarrow$ cell centers.
- Averaging matrices from nodes to cells, $\mathbf{M}^f$: nodes $\rightarrow$ cell center.
- Diagonal matrices with edge length.

To start, the following code generates the 1D difference from nodes to cell centers on a mesh with n cells:

生成带状（稀疏）对角矩阵

```
function[G] = ddx(n)
% [G] = ddx(n)
%
G = spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);
```

Next, the diagonal matrix with the volumes on its diagonal can be generated by the simple command

```
function[V] = sdiag(v)
% [V] = sdiag(n)
%
V= spdiags(v(:),0,numel(v),numel(v));
```

Finally, the averaging matrix can be obtained by using a structure similar to that of the derivative matrix:

```
function[Av] = ave(n)
% Av = ave(n)
%
Av = spdiags(ones(n+1,1)*[1/2 1/2],[0,1],n,n+1);
```

Using these building blocks, we can now quickly write a program that discretizes Maxwell's equations in 1D. We now discuss testing our code by showing how to set up a problem.

2.5 ▪ Programmer note: Testing the code—the method of fictitious sources

The first rule in any code writing is that there must be bugs somewhere, and therefore testing the code is crucial. In fact, we should not consider the code "correct" unless it was tested and retested. In many cases, developing a "good enough" test can be as complicated as writing the code; nonetheless, testing is necessary if we want to use the code and make some conclusions from the results.

In our case here, we have developed a second order discretization for sufficiently smooth problems. Assuming no bugs exist in the theoretical development of the discretization, we would like to verify that the code behaves like the theory predicts.

The method to test the code is referred to as the method of fictitious sources. The idea is to choose functions $e(x)$, $b(x)$, $\mu(x)$, and $\sigma(x)$ and then "plug" them into the PDE, obtaining some nontrivial right-hand side. The right-hand side can be thought of as a fictitious source. Next, we discretize the PDE and the fictitious source on a mesh and

try to recover the known solution. The "trick" is to choose the fields $e(x)$ and $b(x)$ and the parameter functions $\sigma(x)$ and $\mu(x)$ such that the test is indicative of our numerical method.

Here we use a rather simple test. We choose smooth functions for σ, μ, e, and b on the interval $[0,1]$. We make sure that $b(0) = b(1) = 0$, which are the simple boundary conditions we test. To make the test somewhat challenging we use a random mesh for our discretization. We then solve a sequence of problem with a decreasing mesh-size and set up the linear system on each of these meshes, solving the problem and recording the solution error.

The code for setting up the problem is as follows:

```
% test 1D EM
clear all;
close all;

%% set up the analytic example
omega = 1;
sigma = @(x) x.^2+1;
mu    = @(x) cos(x) + 2;

b     = @(x) x.*(x-1).*mu(x);  % note b(0) = b(1) = 0;
e     = @(x) cos(2*pi*x);

% The system
% 1i*w*b + e'           = s1
% (1/mu * b)' - sigma*e = s2

% fictitious source
s1 = @(x) 1i*omega*b(x) - 2*pi*sin(2*pi*x);
s2 = @(x) 2*x-1 - sigma(x).*e(x);
```

Using the functions we can now run the following test:

Table 2.1. *Errors in the recovered solutions of b and e.*

Mesh points	L2 b	L2 e	L_∞ b	L_∞ e
8	1.01e-02	3.79e-02	3.80e-02	5.42e-02
16	1.78e-03	6.38e-03	1.18e-02	1.37e-02
32	2.30e-04	9.39e-04	2.79e-03	3.19e-03
64	3.29e-05	2.01e-04	6.58e-04	9.97e-04
128	1.00e-05	3.71e-05	2.24e-04	2.26e-04
256	1.90e-06	7.54e-06	6.82e-05	6.09e-05
512	3.37e-07	1.47e-06	1.58e-05	1.53e-05
1024	5.64e-08	2.47e-07	4.15e-06	4.04e-06
2048	1.00e-08	4.28e-08	9.96e-07	9.99e-07

```
%% Run numerical setting

for n = [8  16  32  64  128  256 512  1024   2048]
  % set up a random nodal mesh
  h = rand(n,1)*0+1; L = sum(h); h = h/L;
  x = [0; cumsum(h)];
  % cell-centered mesh
  xc = x(1:end-1) + diff(x)/2;

  % the sources
  sh = s1(xc);
  se = s2(x);

  % the linear system
  nc = length(sigma(xc));

  G    = ddx(nc);
  Av   = ave(nc);
  Linv = sdiag(1./h);
  Mmu  = sdiag(h./mu(xc));
  Msig = sdiag(Av'*(sigma(xc).*h));
  M    = sdiag(Av'*h);

  % set the matrix system
  % [1i*w         d/dz] [b] - [s1]
  % [1/mu d/dz  -sigma] [e] - [s2]

  A = [1i*omega*speye(n)  Linv*G; ...
       -G'*Linv'*Mmu      -Msig];
  bb = [sh; M*se];


  eh = A\bb;
  subplot(2,1,1)
  plot(xc,real(eh(1:n)),xc,real(b(xc)),'r');
  subplot(2,1,2)
  plot(x,real(eh(1+n:end)),x,real(e(x)),'r');

  fprintf('L2 error  %3.2e %3.2e  Linf error %3.2e  %3.2e\n', ...
          norm(Linv\(real(eh(1:n) -b(xc)))), ...
          norm(Linv\(0.5*real(eh(1+n:end-1)+eh(2+n:end))-e(xc))), ...
          norm(real(eh(1:n) -b(xc)),'inf'), ...
          norm(real(eh(1+n:end)-e(x)),'inf'));

  pause(0.5)

end
```

The convergence is presented in Table 2.1. Note that even though the mesh is random, the overall convergence is $\mathcal{O}(h^2)$, where h is the averaged mesh size.

For every code that is written, similar (and even more robust) tests must be performed before we proceed to the stage of trying to solve a realistic problem. Unfortunately, far-reaching conclusions on issues such as global warming and nuclear storage have been reached in the past based on "gently" tested codes. One should always be a bit skeptical

about the code and continue to evaluate it. Remember, the usual phrase is

the only code without bugs is a code that is not being used.

2.6 ▪ The 1D MT problem

Finally, we use our discretization to solve the 1D MT problem

$$i\omega b = -\partial_z e, \quad 0 = \partial_z(\mu^{-1} b) - \sigma(z)e, \quad b(0) = 1, \quad b(-\infty) = 0.$$

The problem we face when discretizing the differential equation is the boundary condition at $-\infty$. Since the discretized problem can only be expressed on a finite domain, we need to limit the computations to an interval $[-L, 0]$, and the question is "How should we choose L?" There are two approaches to resolve this issue. First, it is possible to discretize the problem on a large enough domain $[-L, 0]$ such that $b(-L) \approx 0$. This approach can be highly inefficient since L can be large. A better approach is to note that, assuming that σ and μ are constant for $z < -L$, the solution has the form

$$e = c\exp(ikz), \quad \text{where} \quad k = -\sqrt{i\omega\mu\sigma} \quad \text{and} \quad b = -ikc\omega^{-1}\exp(ikz). \tag{2.5}$$

Therefore, for $z < -L$ the equation $i\omega b + e_z = 0$ implies that

$$\omega b - ke = 0.$$

This replaces the boundary condition at $-\infty$ with a boundary condition at $z = -L$, and we can use a much smaller domain for the discretization. To deal with this boundary condition one needs to use ghost points (see Exercises). In the script below we avoid ghost points and set the boundary conditions at a depth of three times the skin depth. We compute the electric field on the surface and extract its absolute value and phase as a function of frequency.

```
% set up and test the MT problem in 1D
clear all;
close all;

omega = logspace(1,4,32);
mu    = 4*pi*1e-7;

% set up the basic mesh
sig0 = 1e-2;
skin = sqrt(2./omega/mu/sig0(end));


for j = 1:length(omega)

  % set up a deep enough mesh
  L  = 3*skin(j);

  h = diff(linspace(0,2*skin(j),2049));
  h = h(:);
  while 1,
      h = [h;h(end)*1.1];
      if sum(h) > L, break; end
  end

  n   = length(h);
  sig = ones(n,1) * sig0;

  % mesh
  z = [0; cumsum(h)];
  % cell-centered mesh
  zc = z(1:end-1) + diff(z)/2;

  % the linear system

  G    = ddx(n);
  Av   = ave(n);
  Linv = sdiag(1./h);
  Mmu  = sdiag(h./mu);
  Msig = sdiag(Av'*(sig.*h));
  M    = sdiag(Av'*h);

  % set the matrix system
  % [1i*w         d/dz] [b] - [s1]
  % [1/mu d/dz  -sigma] [e] - [s2]

  A = [1i*omega(j)*speye(n)  Linv*G; ...
       -G'*Linv'*Mmu      -Msig];

  % set up boundary conditions b(0) = 1
  bb = -A(2:end,1);
  A  = A(2:end,2:end);

  % solve
  be = A\bb;
  b = [1;be(1:n-1)];
  e = be(n+1:end);

  % extract MT data
  d(j) = e(1);


end

subplot(2,1,1)
semilogx(omega,1./omega(:)*mu .* abs(d(:)).^2)
title('apparent resistivity')
subplot(2,1,2)
semilogx(omega,angle(d))
title('phase')
```

The MT problem in 1D has an analytical solution (see Chapter 1). Thus, the code can be verified and checked for accuracy. The MT data are the electric field in the x-direction divided by the magnetic field in the y-direction on the surface. If we define $b = \vec{B}_y$ on the surface as 1, then the data are simply $e = \vec{E}_x$. It is possible to use the ratio between the electric field and the magnetic flux in order to estimate the conductivity. Using the

analytic solution (2.5), the apparent resistivity is defined as

$$\rho_a = \frac{\mu}{\omega}\left|\frac{e}{b}\right|^2.$$

A first test for any MT code is that if one uses a homogeneous space, then the apparent resistivity is retrieved. This can be seen in the example above.

2.7 ▪ Exercises

1. Derive the discretization for the mixed boundary condition $\alpha e + b = c$. Hint: Use the inner product
$$(\partial_z b, w) = -(b, \partial_z w) + (bw)_0^{\text{end}},$$
and
$$b_0^{\text{end}} = c_0^{\text{end}} - \alpha e_0^{\text{end}}.$$
2. Discretize the MT problem using your mixed boundary condition. Report on the accuracy when simulating data.
3. Build your own example for the fictitious sources. Use a solution that has sharper derivatives, and comment on the accuracy obtained.

Chapter 3
Finite Volume Discretization in 3D

In this chapter we expand the 1D discussion into 2D and 3D. As we will see, most of the discussion is a straightforward extension of the 1D case. We start by discussing the discretization of differential operators followed by the inner products on an orthogonal tensor mesh. We then discuss the discretization of the operators and inner products on a logically orthogonal mesh. Such meshes give greater flexibility when working on problems with complex geometry.

The first part of this chapter is a straightforward description of the Yee method [83], which has also been explained in [34]. The second part of this chapter is more advanced and is based on [47, 48, 46, 45].

Just as in 1D, the discretization on any mesh (regular or irregular) is made from two parts. First, the discretization of the differential operators is needed, and second, the discretization of inner products is approximated. As we will see, the discretization of the differential operators is straightforward, while the discretization of inner products is easy on regular mesh and requires some attention on any other mesh.

3.1 ▪ Discretization of differential operators on a tensor mesh

The extension of the staggered discretization proposed in 1D to 2D and 3D is slightly more involved. In principle, we have four types of variables (see Figure 3.1). First, scalar fields can be placed on cell centers or on nodes. Next, we need to consider faces and edge variables that "live" on faces and edges, respectively. While face variables are assumed to be normal to the cell face, edge variables are aligned with the cell edge.

The physical interpretation of face variables are fluxes, since they represent a flux of some sort into or out of a cell. The edge variables are fields, as they represent some path around the surface. The physical meaning of these variables explains our representation of Maxwell's equations in terms of a field $\vec{E}$ (that is discretized on the edges) and a flux $\vec{B}$ (that is discretized on the faces). As we see next, cell center and nodal scalars also have continuous variables that correspond to their physical meaning. A typical mesh in 3D is plotted in Figure 3.1.

We now use simple integration to discretize the differential operators on a staggered mesh. The basic idea is to use the integral definition of each differential operator. These

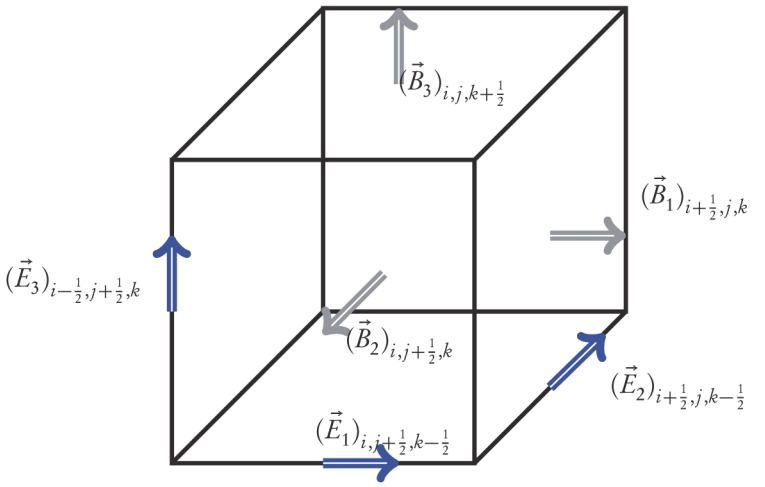

Figure 3.1. *Staggered discretization in 3D.*

definitions are

$$\nabla \cdot \vec{B} = \lim_{\Omega \to 0} \frac{\int_{\partial\Omega} \vec{B} \cdot \vec{n}\, dS}{\int_{\Omega} dV}, \tag{3.1a}$$

$$(\nabla \times \vec{E}) \cdot \vec{n} = \lim_{S \to 0} \frac{\oint_{\partial S} \vec{E} \cdot d\vec{\ell}}{\int_{S} da}, \tag{3.1b}$$

$$(\nabla \phi) \cdot (\vec{x}_2 - \vec{x}_1) = \lim_{|\vec{x}_2 - \vec{x}_1| \to 0} \frac{\phi(\vec{x}_2) - \phi(\vec{x}_1)}{|\vec{x}_2 - \vec{x}_1|}. \tag{3.1c}$$

The DIV

To derive a discretization of the divergence we simply apply the integral form to obtain

$$\nabla \cdot \vec{B} \approx \frac{1}{V_{ijk}} \int_{\Omega_{ijk}} \nabla \cdot \vec{B}\, dV = \frac{1}{\Omega_{ijk}} \int_{\partial\Omega_{ijk}} \vec{B} \cdot \vec{n}\, ds \tag{3.2}$$

$$\approx V^{-1} \left(\sum_{\text{in/out flux}} (\vec{B} \cdot \vec{n}_{\text{in}}) \Delta s_{\text{in}} - (\vec{B} \cdot \vec{n}_{\text{out}}) \Delta s_{\text{out}} \right).$$

Assuming a tensor mesh where $\vec{B}$ is on its faces using a grid function $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3)$ and denoting the area of each face as a face vector as $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$,

$$\begin{aligned} \nabla \cdot \vec{B} \approx V_{ijk}^{-1} \Big(& (\mathbf{a}_1 \mathbf{b}_1)_{i+\frac{1}{2},j,k} - (\mathbf{a}_1 \mathbf{b}_1)_{i-\frac{1}{2},j,k} \\ & + (\mathbf{a}_2 \mathbf{b}_2)_{i,j+\frac{1}{2},k} - (\mathbf{a}_2 \mathbf{b}_2)_{i,j-\frac{1}{2},k} + (\mathbf{a}_3 \mathbf{b}_3)_{i,j,k+\frac{1}{2}} - (\mathbf{a}_3 \mathbf{b}_3)_{i,j,k-\frac{1}{2}} \Big). \end{aligned} \tag{3.3}$$

Similar to the 1D case, we separate the operator into a part that contains the geometry and a part that contains the mesh parameters. We set

$$\mathbf{F} = \text{diag}(\mathbf{a}) \quad \text{and} \quad \mathbf{V} = \text{diag}(\mathbf{v}),$$

where **a** and **v** are vectors of face sizes and cell volumes; then we obtain that

$$\mathsf{DIV} = \mathbf{V}^{-1}\mathbf{D}\mathbf{F},$$

where **D** is a matrix of ± 1 that has the mesh topology.

Note that we somewhat abuse notation to express the vectors **v**, **a**, and **b** both as 3D grid functions and vectors; that is, we do not distinguish between **v** and vec(**v**), and the meaning should be understood from the context. We continue using this notation next.

The CURL

To discretize the curl we use the Gauss rule

$$\int_{\mathscr{S}} \nabla \times \vec{E} \cdot dS = \oint_{\Gamma} \vec{E} \cdot d\ell,$$

where $\mathscr{S}$ is any surface closed with a path Γ. Using the tensor mesh and Gauss's rule, for example, on the x faces (i.e., faces whose normal is parallel to the x axis), we obtain the first component of the **curl**,

$$\begin{aligned}(\nabla \times \vec{E})_1 &\approx \frac{1}{|S|}\int_{\mathscr{S}} \nabla \times \vec{E} \cdot dS \\ &= \frac{1}{\mathbf{a}_{i+\frac{1}{2},j,k}}\Big((\ell_3\mathbf{e}_3)_{i+\frac{1}{2},j+\frac{1}{2},k} - (\ell_3\mathbf{e}_3)_{i+\frac{1}{2},j-\frac{1}{2},k} + (\ell_2\mathbf{e}_2)_{i+\frac{1}{2},j,k-\frac{1}{2}} - (\ell_2\mathbf{e}_2)_{i+\frac{1}{2},j,k+\frac{1}{2}}\Big),\end{aligned}$$

where $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ is the discretization of the edge electric field and ℓ is the length of the edges. Similar expressions are obtained by integrating on the y and z faces.

Separating the operator into a part that contains the geometry and a part that contains the mesh parameters, we set

$$\mathbf{F} = \mathrm{diag}(\mathbf{a}) \quad \text{and} \quad \mathbf{L} = \mathrm{diag}(\ell),$$

where **a** and ℓ are vectors of face area and edge length, and we let **C** be the matrix with ± 1 on its diagonals; then we obtain that

$$\mathsf{CURL} = \mathbf{F}^{-1}\mathbf{C}\mathbf{L}.$$

The GRAD

In a similar way we can discretize the gradient of a nodal function. Let ϕ be such a function; then we have that $\nabla\phi$ can be approximated on cell edges by the following short differences:

$$\begin{aligned}(\phi_x)_{i,j+\frac{1}{2},k+\frac{1}{2}} &\approx \frac{\phi_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - \phi_{i-\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}}{\ell_{i,j+\frac{1}{2},k+\frac{1}{2}}}, \\ (\phi_y)_{i+\frac{1}{2},j,k+\frac{1}{2}} &\approx \frac{\phi_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - \phi_{i+\frac{1}{2},j-\frac{1}{2},k+\frac{1}{2}}}{\ell_{i+\frac{1}{2},j,k+\frac{1}{2}}}, \\ (\phi_z)_{i+\frac{1}{2},j+\frac{1}{2},k} &\approx \frac{\phi_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} - \phi_{i+\frac{1}{2},j+\frac{1}{2},k-\frac{1}{2}}}{\ell_{i+\frac{1}{2},j+\frac{1}{2},k}}.\end{aligned}$$

Separating the discretization of the operator into a part that contains the geometry and a part that contains the mesh parameters, we obtain

$$\mathsf{GRAD} = \mathbf{L}^{-1}\mathbf{G}, \tag{3.4}$$

where again $\mathbf{G}$ is a matrix that contains the ± 1 and the mesh topology.

Properties of the discrete operators

It is possible to verify that for the matrices that contain the geometry of the mesh we have

$$\mathbf{CG} = 0, \quad \mathbf{DC} = 0.$$

It is also possible to verify that $\mathbf{G}$ spans the nontrivial null space of $\mathbf{C}$.

This observation has important consequences known as the mimetic properties of the discrete linear operators. The CURL is an edge-to-face **curl** operator, the GRAD is a nodal gradient-to-edge operator, and DIV is a face-to-cell-center divergence operator. It is straightforward to show (involving a bunch of indices) that

$$\begin{aligned}
&\mathsf{CURL}\ \mathsf{GRAD} = 0,\\
&\mathsf{DIV}\ \mathsf{CURL} = 0,\\
&\mathsf{CURL}^\top\ \mathsf{CURL} + \mathsf{GRAD}\ \mathsf{GRAD}^\top = -\vec{\Delta}_{\text{Edges}},\\
&\mathsf{CURL}\ \mathsf{CURL}^\top + \mathsf{DIV}^\top\ \mathsf{DIV} = -\vec{\Delta}_{\text{Faces}},
\end{aligned}$$

where $\vec{\Delta}$ is the discretization of the vector Laplacian on the Face/Edge variables.

These properties are often referred to as *mimicking* properties, as they mimic the properties of the continuous differential operators. In many physical simulations such as fluid flow and electromagnetics, these properties are essential as they keep some of the conservation properties of the underling PDE. For example, the above discretization applied to Maxwell's equations keeps the magnetic field divergence-free, which can be crucial in many physical simulations.

3.2 ▪ Inner products on tensor mesh

Our discretization is based on Maxwell's equations in weak form. The equations are presented in weak form, and therefore we need to discretize three different weak forms, of cell center, face, and edge variables. We now discuss the discretization of each inner product.

3.2.1 ▪ Cell center inner product

Maybe the simplest discretization is of the cell center inner product. Using a midpoint approximation, we obtain that if u and q are functions discretized in cell centers by the vectors $\mathbf{u}$ and $\mathbf{q}$, then

$$(u,q) = \int_\Omega u(\vec{x})q(\vec{x})\,dV \approx \sum(\mathbf{vuq})_{i,j,k} = \mathbf{u}^\top\mathbf{Vq}.$$

3.2.2 ▪ Face variable inner product

For face variable inner products we note that we discretize a vector inner product. More specifically,

$$(\vec{B},\vec{F}) = \int_\Omega \vec{B}(\vec{x})^\top\vec{F}(\vec{x})\,dV = \int_\Omega \vec{B}_1(\vec{x})\vec{F}_1(\vec{x}) + \vec{B}_2(\vec{x})\vec{F}_2(\vec{x}) + \vec{B}_3(\vec{x})\vec{F}_3(\vec{x})\,dV.$$

To compute the product in one of the $\vec{x}_1$ components, we note that this is equivalent to the 1D case, and therefore

$$\int_\Omega \vec{B}_1(\vec{x})\vec{F}_1(\vec{x})\,dV \approx \sum \frac{\mathbf{v}_{i,j,k}}{2}\left((\mathbf{b}_1\mathbf{f}_1)_{i+\frac{1}{2},j,k} + (\mathbf{b}_1\mathbf{f}_1)_{i-\frac{1}{2},j,k}\right).$$

This can be expressed in matrix form as

$$\int_\Omega \vec{B}_1(\vec{x})\vec{F}_1(\vec{x})\,dV \approx \mathbf{v}^\top \mathbf{A}_{f1}(\mathbf{b}_1 \odot \mathbf{f}_1) = \mathbf{f}_1^\top \operatorname{diag}(\mathbf{A}_{f1}^\top \mathbf{v})\mathbf{b}_1,$$

where $\mathbf{b}_1$ is the discretization of $\vec{B}_1$ on the face in the $\vec{x}_1$-direction. Similarly,

$$\int_\Omega \vec{B}_2(\vec{x})\vec{F}_2(\vec{x})\,dV \approx \mathbf{f}_2^\top \operatorname{diag}(\mathbf{A}_{f2}^\top \mathbf{v})\mathbf{b}_2 \quad \text{and} \quad \int_\Omega \vec{B}_3(\vec{x})\vec{F}_3(\vec{x})\,dV \approx \mathbf{f}_3^\top \operatorname{diag}(\mathbf{A}_{f3}^\top \mathbf{v})\mathbf{b}_3.$$

The sparse matrices $\mathbf{A}_{fj}$ average face variables in the jth direction into the cell centers and contain only $\vec{H}f$ where the faces are averaged.

Finally, we put it all together to obtain

$$(\vec{B}, \vec{F}) \approx \mathbf{f}^\top \mathbf{M}_f \mathbf{b}, \tag{3.5}$$

where

$$\mathbf{M}_f = \operatorname{diag}\begin{pmatrix} \mathbf{A}_{f1}^\top \mathbf{v} \\ \mathbf{A}_{f2}^\top \mathbf{v} \\ \mathbf{A}_{f3}^\top \mathbf{v} \end{pmatrix}.$$

Note that if we need to compute an inner product with weighting of material property, it is straightforward:

$$(\mu^{-1}\vec{B}, \vec{F}) \approx \mathbf{f}^\top \mathbf{M}_{\mu^{-1}f} \mathbf{b}, \tag{3.6}$$

where

$$\mathbf{M}_{\mu^{-1}f} = \operatorname{diag}\begin{pmatrix} \mathbf{A}_{f1}^\top (\mathbf{v} \odot \mu^{-1}) \\ \mathbf{A}_{f2}^\top (\mathbf{v} \odot \mu^{-1}) \\ \mathbf{A}_{f3}^\top (\mathbf{v} \odot \mu^{-1}) \end{pmatrix}.$$

3.2.3 ▪ Edge variable inner product

It is straightforward to derive the discretization for the edge variable inner product in a similar way to the face variable inner products. The main difference is that each cell in the mesh has four edges in each direction, and therefore the averaging is done twice, yielding weights of $\frac{1}{4}$ on each edge. This yields the following approximation to the inner product of an edge function:

$$(\vec{E}, \vec{W}) \approx \mathbf{w}^\top \mathbf{M}_f \mathbf{e}, \tag{3.7}$$

where

$$\mathbf{M}_e = \operatorname{diag}\begin{pmatrix} \mathbf{A}_{e1}^\top \mathbf{v} \\ \mathbf{A}_{e2}^\top \mathbf{v} \\ \mathbf{A}_{e3}^\top \mathbf{v} \end{pmatrix},$$

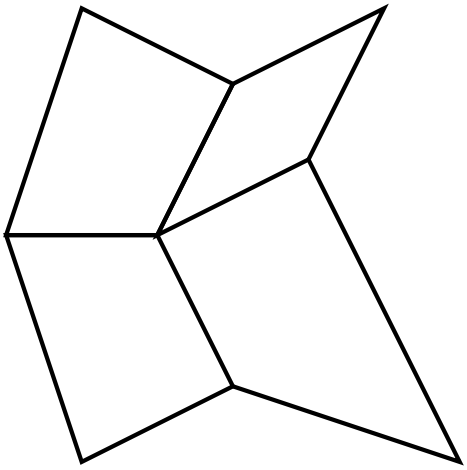

Figure 3.2. *Logically orthogonal mesh.*

where the matrix $\mathbf{A}_{ej}$ averages the edges in the jth direction into cell centers.

Similarly weighted inner products are easy to compute:

$$(\sigma\vec{E}, \vec{W}) \approx \mathbf{e}^\top \mathbf{M}_{\sigma e} \mathbf{w}, \tag{3.8}$$

where

$$\mathbf{M}_{\sigma e} = \operatorname{diag}\begin{pmatrix} \mathbf{A}_{e1}^\top(\mathbf{v} \odot \sigma) \\ \mathbf{A}_{e2}^\top(\mathbf{v} \odot \sigma) \\ \mathbf{A}_{e3}^\top(\mathbf{v} \odot \sigma) \end{pmatrix}.$$

3.3 ▪ Discretization on logically orthogonal grids

A logically orthogonal mesh can be thought of as a regular mesh that has been distorted. In 2D, the mesh is a collection of $(n_1 + 1) \times (n_2 + 1)$ points that define $n_1 \times n_2$ cells. Each point on the mesh has four connected neighbors. A 2D logically orthogonal mesh is plotted in Figure 3.2. In 3D, each point is connected to six other points.

The discretization on this mesh follows the same principles of the orthogonal mesh, and the main complication is the computation of the geometrical equivalents to those discussed on the regular mesh. We now discuss each of the components at some depth. Before we do that we recall some basic vector and geometry identities and refer the reader to Figure 3.3.

- Two vectors, $\mathbf{a}_1$ and $\mathbf{a}_2$ in 3D space generate a plane (as long as they are not parallel to each other). An orthogonal line to the plane, that is, the normal, can be found by the cross product

$$\mathbf{n} = \mathbf{a}_1 \times \mathbf{a}_2.$$

- The parallelogram obtained by $\mathbf{a}_1$ and $\mathbf{a}_2$ has the area of

$$a = |\mathbf{a}_1 \times \mathbf{a}_2|.$$

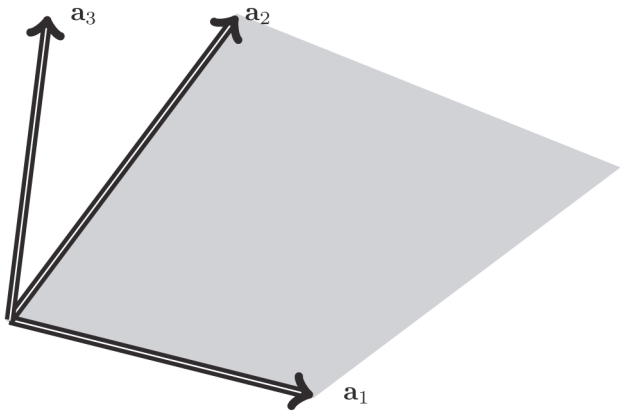

Figure 3.3. *The area of a parallelogram and the volume of a parallelepiped.*

- Given three vectors $\mathbf{a}_1, \mathbf{a}_2$, and $\mathbf{a}_3$ in 3D space, the volume of the parallelepiped that is generated by these vectors is

$$v = |(\mathbf{a}_1 \times \mathbf{a}_2) \cdot \mathbf{a}_3|.$$

Using these simple geometrical formulas, we can now discuss the discretization on the logically orthogonal mesh. To this end, we assume that the mesh is defined by three 3D arrays $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$ such that the nodes of each deformed cube on the mesh correspond to the entries in these arrays. We note the (i, j, k) node by $\mathbf{a}_{i,j,k} = (\mathbf{X}_{i,j,k}, \mathbf{Y}_{i,j,k}, \mathbf{Z}_{i,j,k})$. The volume $v_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}$ is located between the nodes

$$\{\mathbf{a}_{i,j,k}, \mathbf{a}_{i+1,j,k}, \mathbf{a}_{i,j+1,k}, \mathbf{a}_{i+1,j+1,k}, \mathbf{a}_{i,j,k+1}, \mathbf{a}_{i+1,j,k+1}, \mathbf{a}_{i,j+1,k+1}, \mathbf{a}_{i+1,j+1,k+1}\}.$$

Unlike the orthogonal mesh, the discretization of the different components is a delicate matter. Just like in the case of the orthogonal mesh, we can think of nodal variables, edge variables, face variables, and cell center variables.

Nodal variables. Nodal variables are a discretization of a scalar field, say, potential, onto the nodes of the grid; that is, given a function ϕ and its grid function $\boldsymbol{\phi}$, we have that $\boldsymbol{\phi} = \{\phi(\mathbf{a}_{i,j,k}), i = 1, \ldots, (n_1 + 1), j = 1, \ldots, (n_2 + 1), k = 1, \ldots, (n_3 + 1)\}$.

Edge variables. To define edge variables we first define the edges. Each volume has 12 edges. The coordinates of the edges can be obtained by using the nodes of the mesh. We define the edge coordinate $\mathbf{a}_{i+\frac{1}{2},j,k}$ as

$$\mathbf{a}_{i+\frac{1}{2},j,k} = \mathbf{a}_{i+1,j,k} - \mathbf{a}_{i,j,k}$$

and the unit vector as

$$\hat{\mathbf{a}}_{i+\frac{1}{2},j,k} = \frac{\mathbf{a}_{i+\frac{1}{2},j,k}}{|\mathbf{a}_{i+\frac{1}{2},j,k}|}.$$

Now, assume we discretize a field $\vec{E}$ on the edges of our nonorthogonal grid and let $\mathbf{e}$ be the grid function that represents $\vec{E}$. The field $\vec{E}$ is a vector field, and therefore, its

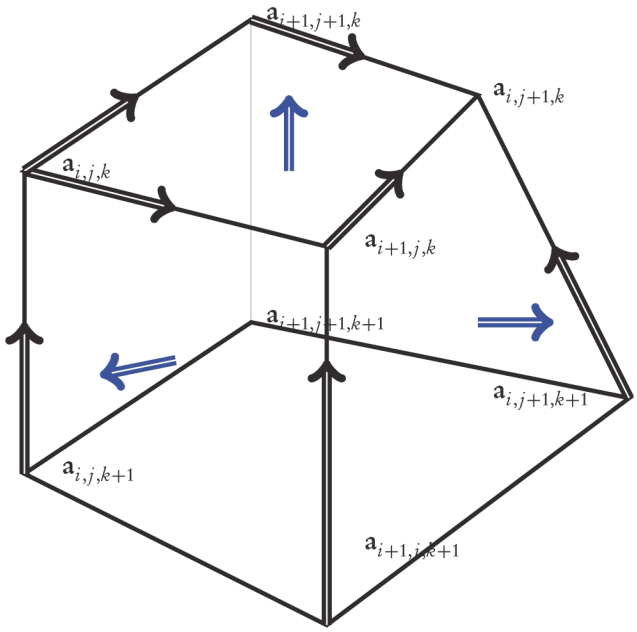

Figure 3.4. *A control volume.*

discretization $\mathbf{e}$ is a projection of $\vec{E}$ onto the discretization point in the direction of the edge. For example, the field $\mathbf{e}(\mathbf{a}_{i+\frac{1}{2},j,k})$ is defined as

$$\mathbf{e}(\mathbf{a}_{i+\frac{1}{2},j,k}) = \vec{E} \cdot \hat{\mathbf{a}}_{i+\frac{1}{2},j,k},$$

that is, the field on the $i+\frac{1}{2},j,k$ edge projected onto the edge direction. The vector field $\mathbf{e}$ is therefore the collection of the projection of fields onto the edges

$$\mathbf{e} = [\mathbf{e}_1^\top, \mathbf{e}_2^\top, \mathbf{e}_3^\top]^\top,$$

where

$$\mathbf{e}_1 = \{\vec{E} \cdot \hat{\mathbf{a}}_{i+\frac{1}{2},j,k}, i = 1,\ldots,n_1, j = 1,\ldots,(n_2+1), k = 1,\ldots,(n_3+1)\},$$
$$\mathbf{e}_2 = \{\vec{E} \cdot \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}, i = 1,\ldots,(n_1+1), j = 1,\ldots,n_2, k = 1,\ldots,(n_3+1)\},$$
$$\mathbf{e}_3 = \{\vec{E} \cdot \hat{\mathbf{a}}_{i,j,k+\frac{1}{2}}, i = 1,\ldots,(n_1+1), j = 1,\ldots,(n_2+1), k = 1,\ldots,n_3\}.$$

Face variables. A more complicated issue is the discretization of face variables. The main problem here is that in general, there is no linear surface that goes through four points. Therefore, there is neither a unique point to define the face variable nor a unique normal to the surface. To define the center of the face we use the average of the four nodes that define the face; that is, the face location $\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k}$ is defined as

$$\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{1}{4}(\mathbf{a}_{i,j,k} + \mathbf{a}_{i+1,j,k} + \mathbf{a}_{i,j+1,k} + \mathbf{a}_{i+1,j+1,k}).$$

The direction of the face is defined as an average direction of normals for each of the corners. First, we consider the four normalized edges that define the face

$$\hat{\mathbf{a}}_{i+\frac{1}{2},j,k}, \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}, \hat{\mathbf{a}}_{i+\frac{1}{2},j+1,k}, \hat{\mathbf{a}}_{i+1,j+\frac{1}{2},k}.$$

Every two edges generate a plane. Thus we consider four different planes, each with its own normal that represents the normal at the corner of each node. Averaging these normals, we obtain an average normal to the face

$$\begin{aligned}\mathbf{n}_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{1}{4}\big(&\hat{\mathbf{a}}_{i+\frac{1}{2},j,k} \times \hat{\mathbf{a}}_{i,j+\frac{1}{2},k} + \hat{\mathbf{a}}_{i+\frac{1}{2},j,k} \times \hat{\mathbf{a}}_{i+1,j+\frac{1}{2},k} \\ &+ \hat{\mathbf{a}}_{i+\frac{1}{2},j+1,k} \times \hat{\mathbf{a}}_{i+1,j+\frac{1}{2},k} + \hat{\mathbf{a}}_{i+\frac{1}{2},j+1,k} \times \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}\big).\end{aligned} \tag{3.9}$$

Similar to the electric field, if a magnetic flux, $\vec{B}$, is discretized on the face of the mesh, then we infer that

$$\mathbf{b}(\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k}) = \vec{B} \cdot \mathbf{n}_{i+\frac{1}{2},j+\frac{1}{2},k}.$$

Cell centered variables. Finally, a cell centered variable $\boldsymbol{\phi}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}$ is a discretization of scalar field located at the point

$$\mathbf{a}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} = \frac{1}{8}\sum_{m,n,p=0}^{1} \mathbf{a}_{i+m,j+n,k+p}.$$

3.3.1 ▪ Differential operators

The discretization in this case follows the same lines as that on the orthogonal mesh, and we use the approximations

$$\nabla \cdot \vec{B} = \frac{\int_{\partial V} \vec{B} \cdot \vec{n}\, dS}{\int_V dV} \approx \mathbf{V}^{-1}\mathbf{D}\mathbf{F}\mathbf{b} = \mathsf{DIV}\mathbf{b}, \tag{3.10a}$$

$$(\nabla \times \vec{E}) \cdot \vec{n} = \frac{\int_{\partial S} \vec{E} \cdot d\ell}{\int_S dS} \approx \mathbf{F}^{-1}\mathbf{C}\mathbf{L}\mathbf{e} = \mathsf{CURL}\mathbf{e}, \tag{3.10b}$$

where, just as in the case of the regular mesh, $\mathbf{D}$ and $\mathbf{C}$ are matrices that have entries of ± 1 and $\mathbf{F}$, $\mathbf{V}$, and $\mathbf{L}$ contain the approximate face area, volume, and edge length of each face, cell, and edge. Since the mesh topology is identical to the tensor mesh, the matrices $\mathbf{D}$ and $\mathbf{C}$ are identical to those obtained on a regular mesh. The main difference between a logical orthogonal mesh and a tensor mesh, and the main difficulty when working with a logically orthogonal mesh, is the computation of the face area, volume, and edge length. We now discuss the computation of the edge length, face area, and cell volume in detail.

Computation of edge length

The computation of edge length is straightforward. Consider two points on a cube $\mathbf{a}_{i,j,k} = (\mathbf{X}_{i,j,k}, \mathbf{Y}_{i,j,k}, \mathbf{Z}_{i,j,k})$ and $\mathbf{a}_{i+1,j,k} = (\mathbf{X}_{i+1,j,k}, \mathbf{Y}_{i+1,j,k}, \mathbf{Z}_{i,j,k})$. The length of the edge between these two points can be computed as

$$\ell_{i+\frac{1}{2},j,k} = \|\mathbf{a}_{i+1,j,k} - \mathbf{a}_{i,j,k}\|.$$

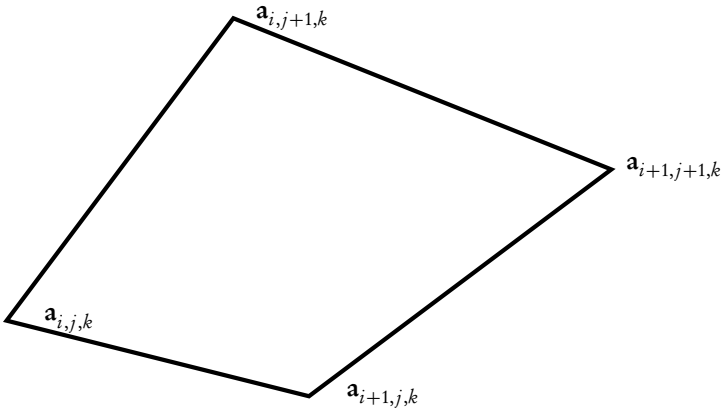

Figure 3.5. *A plane in 3D.*

Computation of face area

Similar to the discussion on face variables, the computation of the area of each face is *not* a straightforward calculation; this is because, given four points in space, they do not generally lie in the same plane.

One way to overcome the difficulty is to determine a unique plane by, for example, seeking the one with minimal curvature. A different approach that we have used to define the face variables is to use an "average" plane to represent the face and its area. Consider the four points in Figure 3.5. It is possible to define (at least) four different planes using these points. We define the plane $P_{i,j,k}$ by the points $\mathbf{a}_{i+1,j,k}, \mathbf{a}_{i,j,k}, \mathbf{a}_{i,j+1,k}$. Similarly, the plane $P_{i+1,j,k}$ is defined by the points $\mathbf{a}_{i,j,k}, \mathbf{a}_{i+1,j,k}, \mathbf{a}_{i+1,j+1,k}$. The plane $P_{i,j+1,k}$ is defined by $\mathbf{a}_{i,j,k}, \mathbf{a}_{i,j+,k}, \mathbf{a}_{i+1,j+1,k}$, and finally, the plane $P_{i+1,j+1,k}$ is defined by $\mathbf{a}_{i+1,j,k}, \mathbf{a}_{i+1,j+,k}, \mathbf{a}_{i,j+1,k}$. To compute the area of the face we average the four areas obtained by the four planes:

$$f_{i+\frac{1}{2},j+\frac{1}{2},k} = \frac{1}{4}\Big(|\mathbf{a}_{i+\frac{1}{2},j,k} \times \mathbf{a}_{i,j+\frac{1}{2},k}| + |\mathbf{a}_{i+\frac{1}{2},j,k} \times \mathbf{a}_{i+1,j+\frac{1}{2},k}| + |\mathbf{a}_{i+\frac{1}{2},j+1,k} \times \mathbf{a}_{i+1,j+\frac{1}{2},k}| + |\mathbf{a}_{i+\frac{1}{2},j+1,k} \times \mathbf{a}_{i,j+\frac{1}{2},k}|\Big).$$

Computation of cell volume

Since cell faces are not uniquely defined, the volume of each cell is not well defined either. Once again, it is possible to define it using some definitions of the surfaces; however, it is also possible to compute it as the average of volumes. Indeed, we can consider every corner of the deformed cube as a corner of a parallelepiped that is defined by the three edges that emerge from that corner. Defining the volume around the (i,j,k) node as

$$v_{i,j,k} = |(\mathbf{a}_{i+\frac{1}{2},j,k} \times \mathbf{a}_{i,j+\frac{1}{2},k}) \cdot \mathbf{a}_{i,j,k+\frac{1}{2}}|,$$

we can approximate the volume as

$$V_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \approx \frac{1}{8} \sum_{\ell,n,m=0,1} v_{i+\ell,j+n,k+m}.$$

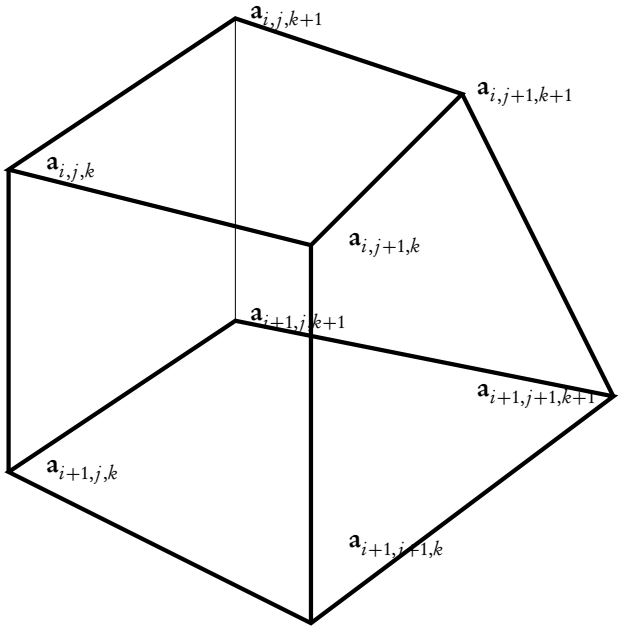

Figure 3.6. *A control volume.*

3.3.2 ▪ Computation of inner products

The computation of inner products is the most involved aspect of the discretization on nonorthogonal grids. Before we go into detail, we review the general idea behind the computation of such a product. Consider the case that the vector fields $\vec{W}$ and $\vec{E}$ are given in nonorthogonal coordinates, and we need to compute the inner product

$$(\vec{W},\vec{E}) = \int_\Omega \vec{E}\cdot\vec{W}\, dv = \int_\Omega (\vec{E}_1\vec{W}_1 + \vec{E}_2\vec{W}_2 + \vec{E}_3\vec{W}_3)\, dv.$$

Clearly, if we are given orthogonal components of the fields, we can use the formula above in a straightforward manner. If, on the other hand, the components are not orthogonal, then we need to do an extra step. Consider the components of the nonorthogonal coordinate system that is given by

$$\hat{\vec{E}}_1 = \mathbf{a}_1\cdot\vec{E}, \quad \hat{\vec{E}}_2 = \mathbf{a}_2\cdot\vec{E}, \quad \hat{\vec{E}}_3 = \mathbf{a}_3\cdot\vec{E},$$

where $\mathbf{a}_i$ are direction vectors that transform the field from the orthogonal system to the nonorthogonal system. This can be written as

$$\begin{pmatrix} \hat{\vec{E}}_1 \\ \hat{\vec{E}}_2 \\ \hat{\vec{E}}_3. \end{pmatrix} = \begin{pmatrix} - & \mathbf{a}_1^\top & - \\ - & \mathbf{a}_2^\top & - \\ - & \mathbf{a}_3^\top & - \end{pmatrix} \begin{pmatrix} \vec{E}_1 \\ \vec{E}_2 \\ \vec{E}_3 \end{pmatrix} \quad \text{or} \quad \hat{\vec{E}} = \mathbf{A}\vec{E}.$$

The above implies that if we are given $\vec{H}at\vec{E}$, we can obtain $\vec{E}$ by simply inverting $\mathbf{A}$ or

$$\vec{E} = \mathbf{A}^{-1}\hat{\vec{E}} \quad \text{and} \quad \vec{W} = \mathbf{A}^{-1}\hat{\vec{W}}.$$

The inner product can therefore be written as

$$(\vec{E},\vec{W}) = \int_\Omega \hat{\vec{W}}^\top \mathbf{A}^{-\top}\mathbf{A}^{-1}\hat{\vec{E}}\, dv.$$

Equipped with this machinery, we can now discuss the discretization of edge and face inner products.

The discretization of edge inner products

To discuss the discretization of the edge inner product we use the notation in Figure 3.7. Consider the (i,j,k) node and the three edges that emerge from this node, noted as $(i+\frac{1}{2},j,k),(i,j+\frac{1}{2},k),(i,j,k+\frac{1}{2})$. These three edges form a nonorthogonal coordinate system with the unit vectors $\vec{Hat}\mathbf{a}_{i+\frac{1}{2},j,k}, \hat{\mathbf{a}}_{i,j+\frac{1}{2},k}, \hat{\mathbf{a}}_{i,j,k+\frac{1}{2}}$. The discretization of the vector fields $\vec{E}$ and $\vec{W}$ on these edges is such that

$$\mathbf{e}_{i+\frac{1}{2},j,k} = \vec{E}_{i+\frac{1}{2},j,k}\cdot\hat{\mathbf{a}}_{i+\frac{1}{2},j,k}, \quad \mathbf{e}_{i,j+\frac{1}{2},k} = \vec{E}_{i,j+\frac{1}{2},k}\cdot\hat{\mathbf{a}}_{i,j+\frac{1}{2},k}, \quad \mathbf{e}_{i+\frac{1}{2},j,k} = \vec{E}_{i+\frac{1}{2},j,k}\cdot\hat{\mathbf{a}}_{i+\frac{1}{2},j,k}.$$

We can approximate the fields in Cartesian coordinates around the (i,j,k) nodes as

$$\begin{pmatrix}(\mathbf{e}_{i,j,k})_x\\(\mathbf{e}_{i,j,k})_y\\(\mathbf{e}_{i,j,k})_z\end{pmatrix} = \begin{pmatrix}- & \hat{\mathbf{a}}_{i+\frac{1}{2},j,k} & -\\ - & \hat{\mathbf{a}}_{i,j+\frac{1}{2},k} & -\\ - & \hat{\mathbf{a}}_{i,j,k+\frac{1}{2}} & -\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{e}_{i+\frac{1}{2},j,k}\\ \mathbf{e}_{i,j+\frac{1}{2},k}\\ \mathbf{e}_{i,j,k+\frac{1}{2}}\end{pmatrix}.$$

Similarly,

$$\begin{pmatrix}(\mathbf{w}_{i,j,k})_x\\(\mathbf{w}_{i,j,k})_y\\(\mathbf{w}_{i,j,k})_z\end{pmatrix} = \begin{pmatrix}- & \hat{\mathbf{a}}^\top_{i+\frac{1}{2},j,k} & -\\ - & \hat{\mathbf{a}}^\top_{i,j+\frac{1}{2},k} & -\\ - & \hat{\mathbf{a}}^\top_{i,j,k+\frac{1}{2}} & -\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{w}_{i+\frac{1}{2},j,k}\\ \mathbf{w}_{i,j+\frac{1}{2},k}\\ \mathbf{w}_{i,j,k+\frac{1}{2}}\end{pmatrix}.$$

With some abuse of notation we set

$$\mathbf{e}^{i,j,k} = \begin{pmatrix}\mathbf{e}_{i+\frac{1}{2},j,k} & \mathbf{e}_{i,j+\frac{1}{2},k} & \mathbf{e}_{i,j,k+\frac{1}{2}}\end{pmatrix}, \quad \mathbf{w}^{i,j,k} = \begin{pmatrix}\mathbf{w}_{i+\frac{1}{2},j,k} & \mathbf{w}_{i,j+\frac{1}{2},k} & \mathbf{w}_{i,j,k+\frac{1}{2}}\end{pmatrix},$$

and

$$\mathbf{A}^{i,j,k} = \begin{pmatrix}- & \hat{\mathbf{a}}^\top_{i+\frac{1}{2},j,k} & -\\ - & \hat{\mathbf{a}}^\top_{i,j+\frac{1}{2},k} & -\\ - & \hat{\mathbf{a}}^\top_{i,j,k+\frac{1}{2}} & -\end{pmatrix}.$$

Using this notation, we can approximate the dot product $\vec{E}\cdot\vec{W}$ around the (i,j,k) node as

$$(\vec{E}\cdot\vec{W})_{i,j,k} \approx (\mathbf{w}^{i,j,k})^\top(\mathbf{A}^{i,j,k})^{-\top}(\mathbf{A}^{i,j,k})^{-1}\mathbf{e}^{i,j,k}.$$

Similar approximations can be obtained around every node in our deformed cube; thus we obtain eight approximations to the dot product $\vec{E}\cdot\vec{W}$, each around a different node. To compute the inner product, that is, the integral over the volume of the dot product, we average the eight corners and multiply by the volume:

$$(\vec{E},\vec{W}) = \int_V \vec{E}\cdot\vec{W}\, dv \tag{3.11}$$

$$\approx \frac{v_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}}{8}\sum_{n,m,\ell=0,1}(\mathbf{w}^{i+n,j+m,k+\ell})^\top(\mathbf{A}^{i+n,j+m,k+\ell})^{-\top}(\mathbf{A}^{i+n,j+m,k+\ell})^{-1}\mathbf{e}^{i+n,j+m,k+\ell}.$$

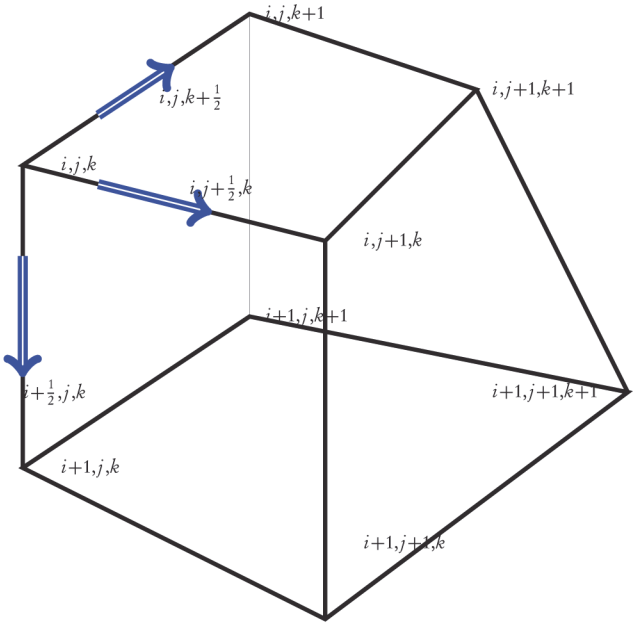

Figure 3.7. *Edges for the computation of an inner product.*

The discretization of face inner products

The face inner product is discretized in a very similar way to the edge inner product. Consider the normals $\hat{\mathbf{n}}_{i,j+\frac{1}{2},k+\frac{1}{2}}, \hat{\mathbf{n}}_{i+\frac{1}{2},j,k+\frac{1}{2}}, \hat{\mathbf{n}}_{i+\frac{1}{2},j+\frac{1}{2},k}$. We can associate them with the (i,j,k) node to approximate the dot product between a vector field $\vec{B}$ and $\vec{F}$ as

$$(\vec{B}\cdot\vec{F})_{i,j,k} \approx (\mathbf{b}^{i,j,k})^\top (\mathbf{N}^{i,j,k})^{-\top} (\mathbf{N}^{i,j,k})^{-1} \mathbf{f}^{i,j,k},$$

where

$$\mathbf{b}^{i,j,k} = \begin{pmatrix} \mathbf{b}_{i,j+\frac{1}{2},k+\frac{1}{2}} & \mathbf{b}_{i+\frac{1}{2},j,k+\frac{1}{2}} & \mathbf{b}_{i+\frac{1}{2},j+\frac{1}{2},k} \end{pmatrix}, \quad \mathbf{f}^{i,j,k} = \begin{pmatrix} \mathbf{f}_{i,j+\frac{1}{2},k+\frac{1}{2}} & \mathbf{f}_{i+\frac{1}{2},j,k+\frac{1}{2}} & \mathbf{f}_{i+\frac{1}{2},j+\frac{1}{2},k} \end{pmatrix},$$

and

$$\mathbf{N}^{i,j,k} = \begin{pmatrix} - & \hat{\mathbf{n}}^\top_{i,j+\frac{1}{2},k+\frac{1}{2}} & - \\ - & \hat{\mathbf{n}}^\top_{i+\frac{1}{2},j,k+\frac{1}{2}} & - \\ - & \hat{\mathbf{n}}^\top_{i+\frac{1}{2},j+\frac{1}{2},k} & - \end{pmatrix},$$

and the inner product is approximated as

$$(\vec{F},\vec{B}) = \int_V \vec{F}\cdot\vec{B}\,dv \tag{3.12}$$
$$\approx \frac{v_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}}}{8} \sum_{n,m,\ell=0,1} (\mathbf{f}^{i+n,j+m,k+\ell})^\top (\mathbf{N}^{i+n,j+m,k+\ell})^{-\top} (\mathbf{N}^{i+n,j+m,k+\ell})^{-1} \mathbf{b}^{i+n,j+m,k+\ell}.$$

Matrix representation of inner products

Similar to the case of the regular mesh, it is possible to represent the inner products in matrix operations. Here, we do not explain this in detail, and we only note that the inner product over the domain Ω can be approximated as

$$(\vec{E}, \vec{W}) \approx \mathbf{e}^\top \mathbf{M}_e \mathbf{w},$$

where

$$\mathbf{M}_e = \frac{1}{8} \sum_i \mathbf{P}_i^\top \mathbf{T}_i^\top \operatorname{diag}(\mathbf{v}, \mathbf{v}, \mathbf{v}) \mathbf{T}_i \mathbf{P}_i,$$

where $\mathbf{P}_i$ is a matrix that takes the fields that are associated with the ith corner and $\mathbf{T}_i$ contains the geometrical edge factors (the inverse of a block 3×3 matrix that transforms from one coordinate system to the other).

Similarly,

$$(\vec{B}, \vec{F}) \approx \mathbf{b}^\top \mathbf{M}_f \mathbf{f},$$

where

$$\mathbf{M}_f = \frac{1}{8} \sum_i \mathbf{Q}_i^\top \mathbf{N}_i^\top \operatorname{diag}(\mathbf{v}, \mathbf{v}, \mathbf{v}) \mathbf{N}_i \mathbf{Q}_i,$$

and again $\mathbf{Q}_i$ extracts the fields associated with the nodes while $\mathbf{N}_i$ contains the face geometry.

Extension of the discretization to anisotropy

One of the nicer consequences of this approach is the ease with which anisotropy can be handled. Assuming symmetric positive definite matrix functions, Σ and Ξ, we can compute inner products of the form

$$(\vec{E}, \Sigma \vec{W}) \quad \text{and} \quad (\vec{F}, \Xi \vec{B}).$$

Assuming discretization, Σ and Ξ are tensors that are defined at each cell; that is, the tensor Σ and Ξ are discretized on the mesh as

$$\Sigma_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} = \begin{pmatrix} \sigma^{11}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{12}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{13}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \\ \sigma^{12}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{22}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{23}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \\ \sigma^{13}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{23}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} & \sigma^{33}_{i+\frac{1}{2},j+\frac{1}{2},k+\frac{1}{2}} \end{pmatrix}.$$

It is straightforward to follow the arguments above and observe that the inner product in this case is changed into

$$\mathbf{M}_{e,\sigma} = \frac{1}{8} \sum_i \mathbf{P}_i^\top \mathbf{T}_i^\top \mathbf{V}_\sigma \mathbf{T}_i \mathbf{P}_i,$$

where

$$\mathbf{V}_\sigma = \operatorname{diag}(\mathbf{v}, \mathbf{v}, \mathbf{v})^{\frac{1}{2}} \begin{pmatrix} \operatorname{diag}(\boldsymbol{\sigma}^{11}) & \operatorname{diag}(\boldsymbol{\sigma}^{12}) & \operatorname{diag}(\sigma^{13}) \\ \operatorname{diag}(\boldsymbol{\sigma}^{12}) & \operatorname{diag}(\boldsymbol{\sigma}^{22}) & \operatorname{diag}(\sigma^{23}) \\ \operatorname{diag}(\boldsymbol{\sigma}^{13}) & \operatorname{diag}(\boldsymbol{\sigma}^{23}) & \operatorname{diag}(\sigma^{33}) \end{pmatrix} \operatorname{diag}(\mathbf{v}, \mathbf{v}, \mathbf{v})^{\frac{1}{2}}.$$

3.4 ▪ Programming the operators using Kronecker products

In order to program problems in 3D, it is useful to consider the matrix representation of the operators. As we see next, for tensor meshes the matrices can be obtained by looking at the 1D case. We assume that we discretize the functions on a mesh with spacing vectors $\mathbf{h}_1, \mathbf{h}_2$, and $\mathbf{h}_3$ and mesh size $n_1 \times n_1 \times n_3$. That is, our mesh is not uniform but orthogonal. We leave the code for the nonorthogonal mesh as an exercise for the reader.

A key property that we heavily use in our derivation is as follows. Let $\mathbf{U}$ be a 2D array, the discretization of the function $u(x,y)$ on the nodes of our mesh. We also define the vector

$$\mathbf{u} = \mathrm{vec}(\mathbf{U}).$$

That is, if $\mathbf{U}$ is an $(n_1+1)\times(n_2+1)$ matrix, then $\mathbf{u}$ is an $(n_1+1)\times(n_2+1)$ vector with entries

$$\mathbf{u}(k) = \mathbf{U}(i,j), \quad k = (i-1)\times(n_1+1)+j.$$

The following identity connects a linear operation on $\mathbf{U}$ to $\mathbf{u}$:

$$\mathrm{vec}(\mathbf{A}\,\mathbf{U}\mathbf{B}^\top) = (\mathbf{B}\otimes\mathbf{A})\mathbf{u}, \tag{3.13}$$

where $\otimes$ is the Kronecker product of matrices

$$\mathbf{A}\otimes\mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ & \ddots & \\ a_{n1}\mathbf{B} & \dots & a_{nn}\mathbf{B} \end{pmatrix}.$$

To see how this can be used to generate a matrix representation of, say, u_x, consider the 2D version of u, that is, the matrix $\mathbf{U}_{ij} = u(x_i, y_j)$. Let $\mathbf{D}_1$ be the 1D difference matrix in the x-direction:

$$\mathbf{D}_1 = \begin{pmatrix} -1 & 1 & & \\ & \ddots & & \\ & & -1 & 1 \end{pmatrix}.$$

Let us look at the product

$$\mathbf{D}_1\mathbf{U} = \frac{1}{h}\begin{pmatrix} -1 & 1 & & \\ & \ddots & & \\ & & -1 & 1 \end{pmatrix}\begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} & \dots & \\ \mathbf{U}_{21} & \ddots & & \\ & & \mathbf{U}_{n,n+1} & \mathbf{U}_{n+1,n+1} \end{pmatrix}$$
$$= \frac{1}{h}\begin{pmatrix} \mathbf{U}_{21}-\mathbf{U}_{11} & \mathbf{U}_{22}-\mathbf{U}_{12} & \dots & \\ \mathbf{U}_{31}-\mathbf{U}_{21} & \ddots & & \\ & & \mathbf{U}_{n+1,n}-\mathbf{U}_{n,n} & \mathbf{U}_{n+1,n+1}-\mathbf{U}_{n,n+1} \end{pmatrix}.$$

If we organize the directions such that x, y is the same as the i, j directions on the matrix, then we can write the difference matrix which corresponds to the (scaled) operator ∂_x in 2D as

$$\partial_x \approx \mathbf{D}_x = \mathbf{I}_2 \otimes \mathbf{D}_1,$$

where D_1 is a 1D $(n_1)\times(n_1+1)$ finite difference matrix and $\mathbf{I}_2$ is an $(n_2+1)\times(n_2+1)$ identity matrix. It is easy to verify that the difference matrix in 2D for the y-direction can be written as

$$\partial_y \approx \mathbf{D}_y = \mathbf{D}_2 \otimes \mathbf{I}_1,$$

where $\mathbf{D}_2$ is a 1D $n_2\times(n_2+1)$ finite difference matrix and $\mathbf{I}_1$ is an $(n_1+1)\times(n_1+1)$ identity matrix.

3.4.1 ▪ Programming linear differential operators

Combining derivatives in different directions, we obtain a discrete representation for the gradient in 2D on a uniform mesh with mesh size 1:

$$\nabla \approx \begin{pmatrix} \mathbf{I}_2 \otimes \mathbf{D}_1 \\ \mathbf{D}_2 \otimes \mathbf{I}_1 \end{pmatrix}.$$

Thus, if we can program the 1D difference matrix and we have a Kronecker product, we can easily obtain a matrix representation in 2D.

The extension to 3D is rather straightforward:

$$\nabla \approx \begin{pmatrix} \mathbf{I}_3 \otimes \mathbf{I}_2 \otimes \mathbf{D}_1 \\ \mathbf{I}_3 \otimes \mathbf{D}_2 \otimes \mathbf{I}_1 \\ \mathbf{D}_3 \otimes \mathbf{I}_2 \otimes \mathbf{I}_1 \end{pmatrix}.$$

To code this we use the code for the 1D case (from Chapter 2). This is a single MATLAB line which we insert into the code as an inline function. Given the 1D code, we can now quickly get the differential operators by combining the derivatives in the different directions and Kronecker products:

```
function[Div] = getFaceDivergenceMatrix(n1,n2,n3)

ddx = @(n) spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);

D1 = kron(speye(n3),kron(speye(n2),ddx(n1)));
D2 = kron(speye(n3),kron(ddx(n2),speye(n1)));
D3 = kron(ddx(n3),kron(speye(n2),speye(n1)));
% DIV from faces to cell-centers
Div = [D1 D2 D3];
```

Similarly, we can get the gradient on the nodes,

```
function[Grad] = getNodalGradientMatrix(n1,n2,n3)

ddx = @(n) spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);

G1 = kron(speye(n3+1),kron(speye(n2+1),ddx(n1)));
G2 = kron(speye(n3+1),kron(ddx(n2),speye(n1+1)));
G3 = kron(ddx(n3),kron(speye(n2+1),speye(n1+1)));
% grad on the nodes
Grad = [G1; G2; G3];
```

and the more complex **curl** matrix,

```
function[Curl] = getEdgeCurlMatrix(n1,n2,n3)

ddx = @(n) spdiags(ones(n+1,1)*[-1 1],[0,1],n,n+1);

nfx = (n1+1)*n2*n3; nfy = n1*(n2+1)*n3; nfz = n1*n2*(n3+1);
nex = n1*(n2+1)*(n3+1); ney = (n1+1)*n2*(n3+1); nez = (n1+1)*(n2+1)*n3;


Dyz = kron(ddx(n3),kron(speye(n2),speye(n1+1)));
Dzy = kron(speye(n3),kron(ddx(n2),speye(n1+1)));

Dxz = kron(ddx(n3),kron(speye(n2+1),speye(n1)));
Dzx = kron(speye(n3),kron(speye(n2+1),ddx(n1)));

Dxy = kron(speye(n3+1),kron(ddx(n2),speye(n1)));
Dyx = kron(speye(n3+1),kron(speye(n2),ddx(n1)));


% curl on the edges
Curl = [sparse(nfx,nex)          Dyz            -Dzy; ...
        -Dxz             sparse(nfy,ney)         Dzx; ...
        Dxy                      -Dyx     sparse(nfz,nez)];
```

The above operators are on a uniform mesh with mesh size 1. To use a general orthogonal mesh we need to generate the volumes, face areas, and edges of the mesh. This can also be obtained from the vector of edge lengths $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3$ by using Kronecker products as follows,

```
function[V,F,L] = getMeshGeometry(h1,h2,h3)

n1 = length(h1); n2 = length(h2); n3 = length(h3);
V = kron(sdiag(h3),kron(sdiag(h2),sdiag(h1)));

F = sdiag([diag(kron(sdiag(h3),kron(sdiag(h2),speye(n1+1)))); ...
           diag(kron(sdiag(h3),kron(speye(n2+1),sdiag(h1)))); ...
           diag(kron(speye(n3+1),kron(sdiag(h2),sdiag(h1))))]);

L = sdiag([diag(kron(speye(n3+1),kron(speye(n2+1),sdiag(h1)))); ...
          diag(kron(speye(n3+1),kron(sdiag(h2),speye(n1+1)))); ...
          diag(kron(sdiag(h3),kron(speye(n2+1),speye(n1+1))))]);
```

setting

```
Div = V\(Div*F); Grad = L\Grad; Curl = F\(Curl*L);
```

The mimetic properties of the operators can be verified by the following lines of code:

```
G = getNodalGradientMatrix(n1,n2,n3);
C = getEdgeCurlMatrix(n1,n2,n3);
figure(1); spy(C*G)
figure(2); spy(C'*C + G*G')
```

We can also check the mimetic properties of the formula

$$\nabla\times\ \nabla\times\ -\nabla\nabla\cdot\ = -\vec{\Delta}.$$

This is demonstrated in Figure 3.8.

Combining the operators, we can now generate simple discretizations for differential equations with constant coefficients. For example, to discretize the Helmholtz equation $\Delta u + k^2 u$ we use the following code:

```
G = getNodalGradientMatrix(n1,n2,n3);
H = G'*G + k^2*speye(size(G,2));
```

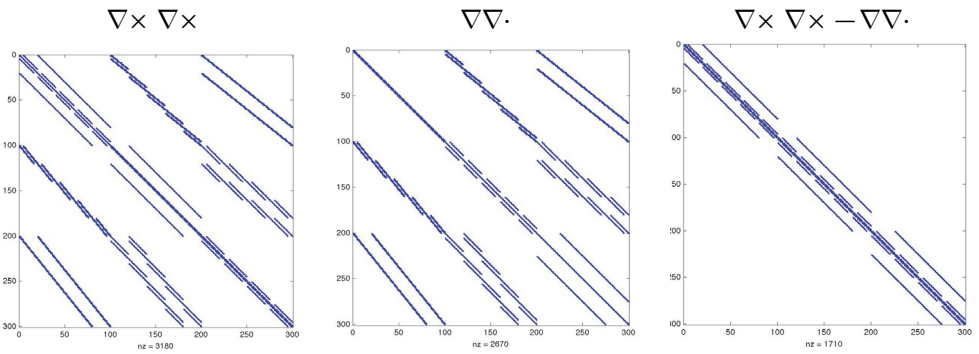

Figure 3.8. *The discrete* $\nabla\times\nabla\times - \nabla\nabla\cdot$.

3.4.2 ▪ Programming the inner product matrices

The discretization is not complete without the discretization of the inner products. Similarly to the 1D case, we use a combination of the trapezoidal and midpoint methods for the calculation of the integral. To do that we need to generate interpolation matrices from faces/edges/nodes to cell centers. We use the 1D interpolation matrix from nodes to cell centers from Chapter 2 as an inline function, and using this 1D discretization we can build an interpolation from faces to cell centers:

```
function[Afc] = getFaceToCellCenterMatrix(n1,n2,n3)

av  = @(n) spdiags(ones(n+1,1)*[0.5 0.5],[0,1],n,n+1);

A1 = kron(speye(n3),kron(speye(n2),av(n1)));
A2 = kron(speye(n3),kron(av(n2),speye(n1)));
A3 = kron(av(n3),kron(speye(n2),speye(n1)));
% average from faces to cell-centers
Afc = [A1 A2 A3];
```

Similarly, we interpolate from the edges to the cell centers,

```
function[Aec] = getEdgeToCellCenterMatrix(n1,n2,n3)

av  = @(n) spdiags(ones(n+1,1)*[0.5 0.5],[0,1],n,n+1);

A1 = kron(av(n3),kron(av(n2),speye(n1)));
A2 = kron(av(n3),kron(speye(n2),av(n1)));
A3 = kron(speye(n3),kron(av(n2),av(n1)));
% average from edge to cell-centers
Aec = [A1 A2 A3];
```

and from the nodes to the cell centers,

```
function[Anc] = getNodalToCellCenterMatrix(n1,n2,n3)

av  = @(n) spdiags(ones(n+1,1)*[0.5 0.5],[0,1],n,n+1);

Anc = kron(av(n3),kron(av(n2),av(n1)));
```

Averaging, we can now discretize the mass matrices in (3.6) and (3.8) directly as

```
% compute the cell volume
V = kron(sdiag(h3),kron(sdiag(h2),sdiag(h1)));
Afc = getFaceToCellCenterMatrix(n1,n2,n3);
Aec = getEdgeToCellCenterMatrix(n1,n2,n3);

Mf = sdiag(Afc'*(V*(1./mu(:))));
Me = sdiag(Aec'*(V*(sigma(:))));
```

The discretization of the mass matrices completes the building blocks needed to discretize Maxwell's equations (and many other equations as well). In the next chapter we discuss the discretization using the tools we build in this chapter.

In the above we have generated different computer codes for the discretization of the **div**, **grad**, **curl**, and the averaging operators. It is possible to "lump" the codes together into one file that computes all the differential operators together. We have done so in the file getLinearOps.m. In the rest of the book we either generate each operator separately or generate all the operators within a single call, depending on the context.

3.5 ▪ Exercises

1. The Stokes system from the pressure and velocity of fluid is

$$\vec{\Delta}\vec{u} + \nabla p = q, \qquad \nabla \cdot \vec{u} = 0.$$

   (a) Show that the system is equivalent to the system

$$\nabla \times \vec{w} + \nabla p = q, \qquad \nabla \times \vec{u} = \vec{w}, \qquad \nabla \cdot \vec{u} = 0,$$

   where $\vec{w}$ is the vorticity.

   (b) Assume that $\vec{u}$ is a field that is discretized on the edges. Where would $\vec{w}$ be naturally discretized? Where is p naturally discretized?

   (c) Suggest a discretization to the system using the tools established in this chapter.

2. Extend the discretization on the tensor mesh to include tensor conductivity. What is the order of the accuracy of your discretization?

Chapter 4
Discretization of Maxwell's Equations

In this chapter we use the machinery we developed in Chapter 3 to discretize Maxwell's equations in frequency and time. We discuss the discretization of sources and linear algebra techniques for the solution of the linear system. Many of the issues here can be found in [34, 4, 35, 13, 12, 6, 80, 37, 43].

4.1 ▪ Discretization of Maxwell's equations in frequency

Recall that Maxwell's equations in weak form are

$$i\omega(\vec{B},\vec{F}) + (\nabla\times\vec{E},\vec{F}) = 0, \tag{4.1a}$$
$$(\mu^{-1}\vec{B},\nabla\times\vec{W}) - (\sigma\vec{E},\vec{W}) = (\vec{s},\vec{W}), \tag{4.1b}$$

where $(\cdot,\cdot)$ is the standard inner product. For the discretization, we assume that the field, $\vec{E}$, is discretized by a grid function, $\mathbf{e}$, on cell edges, and the flux, $\mathbf{b}$, is discretized on cell faces. The products $(\vec{B},\vec{F})$, $(\nabla\times\vec{E},\vec{F})$, and $(\vec{B},\nabla\times\vec{W})$ are face products, and the products $(\sigma\vec{E},\vec{W})$ and $(\vec{s},\vec{W})$ are edge-based products.

Using the discretization of the **curl** and the inner products discussed in Chapter 3, we obtain the following discrete analogue to the system in weak form:

$$i\omega\mathbf{f}^\top\mathbf{M}_f\mathbf{b} + \mathbf{f}^\top\mathbf{M}_f\mathsf{CURL}\mathbf{e} = 0,$$
$$\mathbf{w}^\top\mathsf{CURL}^\top\mathbf{M}_{f\mu}\mathbf{b} - \mathbf{w}^\top\mathbf{M}_{e\sigma}\mathbf{e} = \mathbf{w}^\top\mathbf{M}_e\mathbf{s}.$$

Since this form applies to arbitrary vectors $\mathbf{f}$ and $\mathbf{w}$, we obtain the linear system

$$i\omega\mathbf{b} + \mathsf{CURL}\mathbf{e} = 0, \tag{4.2a}$$
$$\mathsf{CURL}^\top\mathbf{M}_{f\mu}\mathbf{b} - \mathbf{M}_{e\sigma}\mathbf{e} = \mathbf{M}_e\mathbf{s}. \tag{4.2b}$$

Note that the discrete identity

$$\mathsf{DIV}\,\mathsf{CURL} = 0$$

implies also that

$$\mathsf{DIV}\,\mathbf{b} = 0,$$

which means that the discrete $\mathbf{b}$ has a discrete divergence-free condition that mimics the continuous one for $\vec{B}$.

Although it is possible to solve the system, and this is discussed later, we first show how different problems in electromagnetics, in particular all static problems, can be formulated using the discrete system directly. This can be important if we need our code to solve both static and quasi-static problems.

4.2 ▪ Statics

We consider the static case and repeat the derivations done in Chapter 1 for the continuous variables $\vec{E}$ and $\vec{B}$ with the discrete variables $\mathbf{e}$ and $\mathbf{b}$. As we show next, one can carry over the calculations in continuous space to the discrete space directly due to the mimetic properties of the discretization.

In the case that $\omega = 0$, equation (4.2a) implies

$$\mathsf{CURL}\mathbf{e} = 0,$$

and since for the matrices CURL and GRAD the relationsip $\mathsf{CURL}\,\mathsf{GRAD} = 0$ holds, we have that

$$\mathbf{e} = \mathsf{GRAD}\boldsymbol{\phi}.$$

This yields the magnetostatic system

$$\mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathbf{b} - \mathbf{M}_{e\sigma}\,\mathsf{GRAD}\,\boldsymbol{\phi} = \mathbf{M}_e\mathbf{s}, \tag{4.3a}$$

$$\mathsf{DIV}\,\mathbf{b} = 0. \tag{4.3b}$$

As discussed in Chapter 1, there are two cases to consider. Multiplying the first equation by the $\mathsf{GRAD}^\top$ matrix and using the identity $\mathsf{GRAD}^\top\,\mathsf{CURL}^\top = 0$, we obtain that

$$\mathsf{GRAD}^\top \mathbf{M}_{e\sigma}\,\mathsf{GRAD}\,\boldsymbol{\phi} = -\mathsf{GRAD}^\top \mathbf{M}_e\mathbf{s}. \tag{4.4}$$

Equation (4.4) is the well-known DCR equation that can be solved for the discrete potential $\boldsymbol{\phi}$ in order to obtain the discrete electric field $\mathbf{e}$.

To solve for the (discrete) magnetic field $\mathbf{b}$ we use the identity

$$\mathsf{DIV}\,\mathsf{CURL} = 0$$

and set

$$\mathbf{b} = \mathsf{CURL}\,\mathbf{a} \quad \text{and} \quad \mathsf{GRAD}^\top \mathbf{a} = 0.$$

This is the discrete Helmholtz decomposition for $\mathbf{b}$. Note that the term $\mathsf{GRAD}^\top\mathbf{a} = 0$ serves as the discrete Coulomb gauge condition.

The Helmholtz decomposition yields the magnetostatic equations

$$\mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL}\mathbf{a} = \mathbf{M}_e\mathbf{s} + \mathbf{M}_{e\sigma}\,\mathsf{GRAD}\,\boldsymbol{\phi}.$$

This system is singular due to the null space of the CURL. To change it to positive definite we use the discrete Coulomb gauge condition and add the term $\mathsf{GRAD}\,\mathbf{M}_n\,\mathsf{GRAD}^\top\mathbf{a} = 0$ to the left side of the equation, obtaining

$$(\mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} + \mathsf{GRAD}\,\mathbf{M}_n\,\mathsf{GRAD}^\top)\mathbf{a} = \mathbf{M}_e\mathbf{s} + \mathbf{M}_{e\sigma}\,\mathsf{GRAD}\,\boldsymbol{\phi}. \tag{4.5}$$

It is important to note that the matrix $\mathbf{M}_n$ is any symmetric semipositive matrix that we are free to choose. One common option is to try to balance its entries with the matrix

$\mathbf{M}_{f\mu} = \text{diag}(\mathbf{A}_f^\top(\mathbf{v} \odot \mu))$. To do this we note that $\mathbf{M}_n$ operates on the nodes of our mesh, and therefore a reasonable choice is

$$\mathbf{M}_n = \text{diag}(\mathbf{A}_n^\top(\mathbf{v} \odot \mu)),$$

where $\mathbf{A}_n$ is a matrix that averages the nodes into cell centers.

Finally, we note that if the source is zero but the boundary conditions on $\mathbf{b}$ are not homogeneous, for example, setting

$$\mathbf{b}_0 = [\alpha_1 \mathbf{1}_1^\top, \alpha_2 \mathbf{1}_2^\top, \alpha_3 \mathbf{1}_3^\top]^\top,$$

where $\mathbf{1}$ is a vector of ones and the α's are arbitrary parameters, then we obtain that

$$\mathsf{DIV}\mathbf{b}_0 = \mathbf{q},$$

where $\mathbf{q} = 0$ for every cell inside the domain but is not zero at faces of the boundary of the domain. Assuming $\mathbf{b}_0$ and $\mathbf{b}$ share the same boundary conditions, we can demand that

$$\mathsf{DIV}\mathbf{b} = \mathbf{q}.$$

Using the identity $\mathsf{CURL}^\top \mathsf{DIV}^\top = 0$ and setting

$$\mathbf{b} = \mathbf{M}_{f\mu}^{-1} \mathsf{DIV}^\top \psi,$$

which satisfies the first equation in (4.3), yields a Poisson-like equation for the discrete magnetic scalar potential ψ:

$$\mathsf{DIV}\mathbf{M}_{f\mu}^{-1} \mathsf{DIV}^\top \psi = \mathbf{q}.$$

The above shows that discretizing Maxwell's equations using mimetic methods allows us to discretize (and later solve) virtually every scenario within the electromagnetic forward problems.

4.3 ▪ Programming Maxwell's equations

We now combine the "machinery" developed to generate a code that discretizes Maxwell's equations in 3D.

The code to generate the system is as follows:

```
% set up the Maxwell system with a random source
clear all; close all
n1 = 4; n2 = 5; n3 = 6;
h1 = 10*rand(4,1);
h2 = 10*rand(5,1);
h3 = 10*rand(6,1);

sigma = rand(n1,n2,n3)*0.1;
mu    = ones(n1,n2,n3)*4*pi*1e-7;

% freq
w = 1e1 * 2*pi;

% linear operators
[V,F,L] = getMeshGeometry(h1,h2,h3);
Afc = getFaceToCellCenterMatrix(n1,n2,n3);
Aec = getEdgeToCellCenterMatrix(n1,n2,n3);
Mu = sdiag(Afc'*(V*(1./mu(:))));
Msig = sdiag(Aec'*(V*(sigma(:))));


CURL = getEdgeCurlMatrix(n1,n2,n3);
DIV  = getFaceDivergenceMatrix(n1,n2,n3);
GRAD = getNodalGradientMatrix(n1,n2,n3);

DIV = V\(DIV*F); GRAD = L\GRAD; CURL = F\(CURL*L);

%% The full Maxwell system and sources
A = [-1i*w*Mu    CURL; CURL'     Msig];

% call for the source (random placeholder)
s = randn(size(CURL,2),1);
b = [zeros(size(CURL,1),1); s];
```

As previously discussed, it is easy to eliminate either of the discrete analogues to $\vec{E}$ and $\vec{H}$, **e** and **h**, from the discrete system.

The MATLAB script reads as follows:

```
Ae = CURL'* sdiag(1./diag(Mu)) * CURL - 1i*w*Msig;
be = -1i*w*s;
```

The linear system (either in first or second order form) can now be solved to obtain an approximation to the electric and magnetic fields. One can verify that the systems are equivalent:

```
%% Test equivalence
ee = Ae\be;
eh = A\b;
h = eh(1:size(CURL,1)); e = eh(size(CURL,1)+1:end);

fprintf('||e_full - e_red|| =  %3.2e\n',norm(ee-e)/norm(e));
```

The reader who actually experiments with the code can find that the difference between the solutions can be much larger than machine precision. This is due to the null space of the **curl** to be discussed later in this chapter.

4.4 ▪ Setting up sources

So far we have discussed the discretization of Maxwell's equations. However, the linear system is not complete without the discretization of the right-hand side. We now discuss a number of approaches to treating the right-hand side.

4.4.1 ▪ Mathematical models of the sources and receivers

Prior to discretization we need to have a mathematical description of the sources. Given a time domain, or a frequency domain source that we denote as $\vec{s}(t,\mathbf{x})$ or $\widehat{\vec{s}}(\omega,\mathbf{x})$, we assume that they have the general form

$$\vec{s}(t,\mathbf{x}) = \vec{v}(\mathbf{x})f(t) \quad \text{or} \quad \widehat{\vec{s}}(\omega,\mathbf{x}) = \vec{v}(\mathbf{x})\widehat{f}(\omega).$$

That is, we model the source as a time or frequency varying source located and oriented at the same location for all times and frequencies.

We now further investigate the properties of the function $\vec{v}(\mathbf{x})$. A uniform wire carrying one unit of current is modeled by a delta function following the path of the wire. Let the vector $\vec{\Gamma}(\mathbf{x})$ represent the path of the wire. We assume that $|\vec{\Gamma}(\mathbf{x})| = 1$ where the wire is present and 0 elsewhere. The components of $\vec{\Gamma}$ point to the direction of the flow of the current in the wire. Using $\vec{\Gamma}(\mathbf{x})$, the vector function $\vec{v}(\mathbf{x})$ is modeled as a delta function over that path, where the sign and orientation is determined by the projection of the path onto the coordinate system

$$\vec{v}(\mathbf{x}) = \delta(|\vec{\Gamma}(\mathbf{x})|) \sum_{j=1}^{3} (\vec{\Gamma}(\mathbf{x}), \hat{\mathbf{x}}_j)\hat{\mathbf{x}}_j, \tag{4.6}$$

where $(\cdot,\cdot)$ is the usual L_2 inner product, $\delta(\cdot)$ is the Dirac delta function, and $\vec{H}bfx_j$ are the canonical vectors in $\mathbb{R}^3$.

Consider now a loop or a wire $\vec{\Gamma}'(\mathbf{x})$ acting as the receiver. The loop/wire can be modeled in the same way:

$$\vec{v}'(\mathbf{x}) = \delta(|\vec{\Gamma}'(\mathbf{x})|) \sum_{j=1}^{3} (\vec{\Gamma}'(\mathbf{x}), \hat{\mathbf{x}}_j)\hat{\mathbf{x}}_j.$$

Given an electric field $\vec{E}$, and assuming a time domain measurement, the voltage induced in the receiver loop is

$$d(t) = \int_{\Omega} \vec{v}'(\mathbf{x}) \cdot \vec{E}(t,\mathbf{x})\, d\mathbf{x}.$$

Similarly, in frequency we have

$$\widehat{d}(\omega) = \int_{\Omega} \vec{v}'(\mathbf{x}) \cdot \widehat{\vec{E}}(\omega,\mathbf{x})\, d\mathbf{x}.$$

Thus, the discretization of the source can be accomplished by discretizing path dependent delta functions on the given mesh.

4.4.2 ▪ Direct discretization of the sources

The simplest way to discretize the sources and receivers is by directly discretizing the integrals of the delta functions on a grid. To this end, we recall that a delta function cannot be thought of as a usual function but needs to be considered as an integral.

Let $\vec{v}(\mathbf{x})$ be a simple line source, and assume for a moment that the source is between the points (x_s, y_s, z_s) and $(x_s + \Delta x, y_s, z_s)$ that coincide with the mesh. That is, the path $\Gamma(\mathbf{x})$ is

$$\Gamma(\mathbf{x}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

for $x_s \le x \le x_s + \Delta x$, $y = y_s$, and $z = z_s$.

Rewriting Ampère's law in weak form,

$$(\mu^{-1}\vec{B}, \nabla\times \vec{W}) + (\sigma\vec{E}, \vec{W}) = (\vec{s}, \vec{W}),$$

where $\vec{W}$ is some arbitrary edge function. We have discussed the integration of the left-hand side. For smooth right-hand sides one can simply use a midpoint method; however, for the delta function this approach does not work. Let us look at the inner product of $\vec{s}$ and $\vec{W}$ more closely:

$$(\vec{s}, \vec{W}) = \int_{\text{cell}} \vec{W} \cdot \vec{v}(\mathbf{x})\, d\mathbf{x}.$$

Assuming that the source is in the x-direction, we obtain

$$(\vec{s}, \vec{W}) = \int_{\text{cell}} \vec{W}_x \vec{v}_x(\mathbf{x})\, d\mathbf{x} = \vec{W}_x(\Gamma(\mathbf{x})).$$

That is, the value of the integral is the value of the function $\vec{W}$ over the path Γ.

To evaluate $\vec{W}$ over the path, linear interpolation *inside each cell* is used. For simplicity, assume that the source is on the yz plain and it changes in the x-direction, as plotted in Figure 4.1. In this case the interpolation weights yield

$$(\vec{s}, \vec{W}) \approx \frac{(\vec{W}_1)_{i,j-\frac{1}{2},k+\frac{1}{2}} L_1 + (\vec{W}_1)_{i,j+\frac{1}{2},k+\frac{1}{2}} L_2}{L_1 + L_2} L_3. \tag{4.7}$$

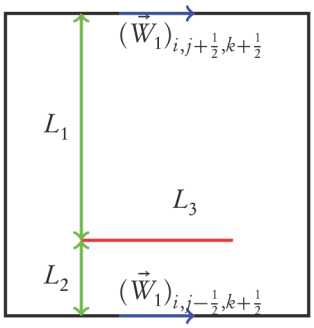

Figure 4.1. *Interpolation of $\vec{W}$ to approximate the integral with a delta function.*

Generally, when the source is completely inside the cell and parallel to one of the axes, the interpolation requires four edges. More generally, if the source is not parallel to the edges, interpolation of all components is required. In the simplest case, if the source is on the edge, this gives a 1 on the edge and zero elsewhere.

Collecting the nonzero terms in (4.7), we obtain that

$$(\vec{s}, \vec{W}) \approx \mathbf{w}^\top \mathbf{s},$$

where the vector $\mathbf{s}$ contains the interpolation weights of the delta function.

The discretization discussed above has one very important aspect. We can verify directly that, taking the discrete divergence of the source, we obtain zero everywhere but the end of any wire segment. Furthermore, for any closed loop the discrete divergence is zero everywhere. This character of the discrete source is extremely important. It does not allow for current leakage into the media from the wire unless it is open and it requires the current to exit the wire only from its two ends. One can verify using numerical experiments that if numerical current leakage does occur, then the fields become rather nonphysical.

We now discuss the implementation of this discretization. Here we use a rather simple source that is assumed to be made of line segments that are parallel to the axis.

```
function[q,qx,qy,qz] = getSourceFromLineSeg(P,x,y,z)
% [q,qx,qy,qz] = getSourceFromLineSeg(p,x,y,z)
% Generates a current source on the mesh.

% allocate memory for the source
nx = length(x); ny = length(y); nz = length(z);
qx = zeros(nx-1,ny,nz); qy = zeros(nx,ny-1,nz); qz = zeros(nx,ny,nz-1);

% discretize each line segment
for i=1:size(P,1)-1
    d = P(i+1,:) - P(i,:);
    dr = find(abs(d)>1e-5);
    if dr == 1,
        qt = interpolateLineSegment(P(i:i+1,:),x,y,z);
        qx = qx + sign(d(dr))*qt;
    elseif dr == 2,
        qt = interpolateLineSegment(P(i:i+1,:),x,y,z);
        qy = qy + sign(d(dr))*qt;
    elseif dr == 3,
        qt = interpolateLineSegment(P(i:i+1,:),x,y,z);
        qz = qz + sign(d(dr))*qt;
    end
end

q = [qx(:);qy(:); qz(:)];
```

The code requires the interpolation of line segments that is given in the following code:

```
function[q] = interpolateLineSegment(p,x,y,z)
% interpolation for different directions

nx = length(x); ny = length(y); nz = length(z);

% determine the direction of the line segment
d  = p(2,:) - p(1,:);
dr = find(abs(d)>1e-5);
if p(end,dr) < p(1,dr), p = flipud(p); end

c1 = findCellNumber(p(1,:),x,y,z);
c2 = findCellNumber(p(2,:),x,y,z);

P(1,:) = p(1,:);
if dr == 1,
    tt = x;
elseif dr == 2,
    tt = y;
elseif dr == 3,
    tt = z;
end
i = 1;
while tt(c1(dr)+i) <= tt(c2(dr))
    if dr == 1,
       P(i+1,:) = [tt(c1(1)+i),p(1,2),p(1,3)];
    elseif dr == 2,
       P(i+1,:) = [p(1,1),tt(c1(2)+i),p(1,3)];
    elseif dr == 3,
       P(i+1,:) = [p(1,1),p(1,2),tt(c1(3)+i)];
    end
    i = i+1;
end
if tt(c2(dr)) ~= p(2,dr)
  P(i+1,:) = p(2,:);
end

% interpolation weights
if     dr == 1, t1 = y; t2 = z; i1 = 2; i2 = 3;
elseif dr == 2, t1 = x; t2 = z; i1 = 1; i2 = 3;
elseif dr == 3, t1 = x; t2 = y; i1 = 1; i2 = 2;
end

d1(1) = t1(c1(i1)+1) - p(1,i1);
d1(2) = p(1,i1) - t1(c1(i1));
d2(1) = t2(c1(i2)+1) - p(1,i2);
d2(2) = p(1,i2) - t2(c1(i2));

a     = sum(d1)*sum(d2);
w1    = (d1(1)*d2(1))/a;
w2    = (d1(2)*d2(1))/a;
w3    = (d1(1)*d2(2))/a;
w4    = (d1(2)*d2(2))/a;

e = zeros(1,3); e(dr) = 1; q = zeros([nx,ny,nz]-e);

for i=1:size(P,1)-1
    L = P(i+1,dr) - P(i,dr);
    if dr == 1
        q(c1(1)+i-1,c1(2),c1(3))     = L*w1;
        q(c1(1)+i-1,c1(2)+1,c1(3))   = L*w2;
        q(c1(1)+i-1,c1(2),c1(3)+1)   = L*w3;
        q(c1(1)+i-1,c1(2)+1,c1(3)+1) = L*w4;
    elseif dr == 2
        q(c1(1),c1(2)+i-1,c1(3))     = L*w1;
        q(c1(1)+1,c1(2)+i-1,c1(3))   = L*w2;
        q(c1(1),c1(2)+i-1,c1(3)+1)   = L*w3;
        q(c1(1)+1,c1(2)+i-1,c1(3)+1) = L*w4;
    elseif dr == 3
        q(c1(1),c1(2),c1(3)+i-1)     = L*w1;
        q(c1(1),c1(2)+1,c1(3)+i-1)   = L*w2;
        q(c1(1)+1,c1(2),c1(3)+i-1)   = L*w3;
        q(c1(1)+1,c1(2)+1,c1(3)+i-1) = L*w4;
    end

end
```

Finally, the interpolation requires finding the cell in which the source resides.

```
function[cellnum] = findCellNumber(p,x,y,z)
%[cellnum] = findCellNumber(p,x,y,z)

[~,im] = min(abs(p(1)-x));
cellnum(1) = im - (p(1) - x(im) < 0);

[~,im] = min(abs(p(2)-y));
cellnum(2) = im - (p(2) - y(im) < 0);

[~,im] = min(abs(p(3)-z));
cellnum(3) = im - (p(3) - z(im) < 0);
```

The above code summarizes the direct discretization of the sources and receivers.

While direct discretization of the sources is straightforward, it has a few drawbacks. First, using direct discretization the resolution is limited to the mesh size. A second issue with the direct discretization is that the boundary of our domain is required to be sufficiently far away in order to minimize its effect on the solution.

While the above limitation makes sense, we now show that in some cases, it is possible to obtain better results by using a different approach.

4.4.3 ▪ Primary secondary discretization

The direct approach to discretizing sources can work well when the source is sufficiently large relative to the mesh. However, this discretization can run into difficulties when the source has some arbitrary shape that is smaller than the mesh. Also, the previous approach cannot deal with sources that are outside of the mesh.

A different way to discretize the source is to consider Ampère's law (neglecting displacement currents) that reads

$$\nabla \times \mu^{-1}\vec{B} - \sigma\vec{E} = \vec{s}.$$

Now assume that for some given source $\vec{s}$, a known conductivity σ_0, and a frequency ω, we can somehow compute the electric and magnetic fields, $\vec{E}_0$ and $\vec{B}_0$, to high accuracy. Such computation can be done analytically in some cases, and by using other codes, say, a 1D code. We now define the source as

$$\vec{s}_0 \stackrel{\text{def}}{=} \nabla \times \mu_0^{-1}\vec{B}_0 - \sigma_0\vec{E}_0.$$

Even though we have defined the source by using the fields, the source represents a quantity that is independent of the fields! The source is supposed to stay identical for all frequencies and conductivities. Any fields that solve Maxwell's equations for any frequency will do. This implies that we can set $\vec{s} = \vec{s}_0$ as the source.

The question that remains is how to choose σ_0 and ω_0 such that we can compute the analytic solution with relative ease for most wire sources. To do that we note that if $\vec{s}$ is a linear wire source, then we can compute the electric and magnetic static fields assuming some known conductivity and that $\omega = 0$. Thus, we can approximate the source as

$$\vec{s} = \sum_j \vec{\ell}_j,$$

where $\vec{\ell}_j$ are linear sources that do not in general coincide with the mesh.

Given $\vec{\ell}_j$, we compute $\vec{B}_0^j$ and $\vec{E}_0^j$ as the solution of the static Maxwell equations. A code to compute the magnetic field generated from a line segment is as follows:

```
function B = calcBFromLineSource(tx_loc,rx_loc)
%Calculates the analytic h-fields from an arbitrary transmitter
%Inputs: tx_loc -> a nSegments+1 by 3 matrix with the location of
%                  transmittor nodes
%        rx_loc -> nStations by 3 matrix of the locations where the
%                  fields should be calculated


nrx = size(rx_loc,1);
n_segments = size(tx_loc,1) - 1;
mu = 4*pi*1e-7;

B = zeros(nrx,3);

for ii = 1:n_segments

    a = tx_loc(ii,:);
    b = tx_loc(ii+1,:);

    ra = rx_loc - repmat(a,nrx,1);
    rb = rx_loc - repmat(b,nrx,1);

    nra = sqrt(ra(:,1).^2 + ra(:,2).^2 + ra(:,3).^2);
    nrb = sqrt(rb(:,1).^2 + rb(:,2).^2 + rb(:,3).^2);

    ba = b - a;
    nba = norm(ba);

    ra_dot_rb = dot(ra',rb')';

    denom = (nba.^2*nra.^2 - ( ra_dot_rb - nra.^2 ).^2 );
    denom(~denom) = eps;
    t1 = ( ra_dot_rb .* (1./nra + 1./nrb) - ( nra + nrb ) ) ./ denom;

    t2 = [(rb(:,2).*ra(:,3)-rb(:,3).*ra(:,2)), ...
          (rb(:,3).*ra(:,1)-rb(:,1).*ra(:,3)), ...
          (rb(:,1).*ra(:,2)-rb(:,2).*ra(:,1))].*t1(:,ones(1,3));

    B = B + t2;

end
B = B/mu;
```

To compute the electric field, we first evaluate the potential field ϕ. Let the ends of the wire be at points A and B; then

$$\boldsymbol{\phi}_j(\vec{r}) = \frac{1}{2\pi\sigma|\vec{r}_B - r|} - \frac{1}{2\pi\sigma|\vec{r}_A - r|}.$$

The total fields ϕ and $\vec{B}_0$ are a linear combination of the fields:

$$\vec{B}_0(\mathbf{x}) = \sum \vec{B}_0^j(\mathbf{x}), \qquad \phi(\mathbf{x}) = \sum \phi_j(\mathbf{x}).$$

Recall now that in order to discretize the source we need to evaluate the integral

$$(\vec{s}, \vec{W}) = (\mu^{-1}\vec{B}_0(\mathbf{x}), \nabla \times \vec{W}) - (\sigma \nabla \phi, \vec{W}).$$

If the source is outside of the mesh or much smaller than the mesh, then the fields $\vec{B}_0(\mathbf{x})$ and $\phi(\mathbf{x})$ are smooth. For these cases, assume that we have discretized $\vec{B}_0$ on the staggered mesh (that is, $\vec{B}_0$ is discretized on the faces of our mesh) and that ϕ is discretized on the nodes of our mesh. Let $\mathbf{b}_0$ and $\boldsymbol{\phi}_h$ be the discrete values of the field on the mesh. Then, it is a straightforward calculation to obtain that

$$\mathbf{s} = \mathsf{CURL}^\top \mathbf{M}_{f_\mu} \mathbf{b}_0 + \sigma \mathsf{GRAD} \boldsymbol{\phi}. \tag{4.8}$$

However, this discretization can yield unreasonable results if the source is close to the edges or, worse, on an edge. This is because the potential fields are singular on an edge.

In this case one must evaluate the integral of the fields (which is always bounded). Some analytical integration can be used, and, while this is not complicated, the expressions can be rather long. We leave this as an exercise for the reader.

The indirect discretization has a number of interesting properties. Assuming that the source is a closed loop implies that the electric field (for the zero frequency) is 0. Using the previous notation of the discrete **curl** and **grad**, we note that

$$\mathsf{GRAD}^\top \mathsf{CURL}^\top \mathbf{h}_0 = \mathsf{GRAD}^\top \mathbf{s} = 0;$$

that is, the source is divergence-free by construction. This implies that we are not generating "magnetic charges" by discretization errors and thus follow the physical meaning of Maxwell's equations faithfully.

4.5 ▪ Time stepping

So far we have discussed the discretization of Maxwell's equations in space. This allows us to solve problems in the frequency domain. To solve problems in the time domain we must discretize the system in time. This is discussed in this section, where we quickly review three time stepping techniques and then apply direct time discretization techniques. We then turn our attention to a fourth method of time integration that can be useful in some scenarios.

4.5.1 ▪ A (very) quick review of methods for time dependent ODEs

One version of Maxwell's equations in time (ignoring displacement currents) can be written as

$$\nabla\times \mu^{-1}\nabla\times \vec{E} + \sigma \vec{E}_t = -\vec{s}_t,$$

which is closely related to the heat equation[2]

$$\Delta \hat{u} - \sigma \hat{u}_t = 0,$$

with some initial condition on $\vec{H}at\,u$. This is analogous to the quasi-static Maxwell system, not including displacement currents.

For this simple problem we can take the Fourier transform of u in space obtaining the ODE

$$\xi^2 u + \sigma u_t = 0,$$

where ξ is the spacial frequency.

We now consider three simple discretizations for the problem:

$$\begin{aligned}
\text{EXP} \quad & (u^{n+1} - u^n) = -\sigma^{-1} k \xi^2 u^n, \\
\text{IMP} \quad & (u^{n+1} - u^n) = -\sigma^{-1} k \xi^2 u^{n+1}, \\
\text{MID} \quad & (u^{n+1} - u^n) = -\frac{1}{2}\sigma^{-1} k \xi^2 (u^n + u^{n+1}),
\end{aligned}$$

where $k = t_{n+1} - t_n$ is the time step. The first method, EXP, is an explicit forward Euler method. The solution at the next time step can be calculated explicitly:

$$u^{n+1} = (1 - k\sigma^{-1}\xi^2) u^n.$$

[2]Recall that $\nabla\times\nabla\times - \nabla\nabla\cdot = -\vec{\Delta}$.

The forward Euler method can be rather unstable. For stability we need to have

$$|1-k\sigma^{-1}\xi^2| \le 1$$

or (when the absolute value changes its sign)

$$k \le 2\sigma\xi^{-2}.$$

This step may be too small for many practical settings.

The second method is the backward Euler (BE), and it reads

$$u^{n+1} = (1+k\sigma^{-1}\xi^2)^{-1}u^n.$$

Clearly, the method is unconditionally stable, and there is no restriction on the step size for stability. This is not to be confused with accuracy. While the system may be stable, very large step sizes may lead to inaccurate results.

Finally, the third method, the midpoint method, reads

$$u^{n+1} = \frac{2-k\sigma^{-1}\xi^2}{2+k\sigma^{-1}\xi^2}\,u^n.$$

This is also an unconditionally stable method.

While the above methods discuss stability requirements, the question about accuracy still remains. In principle, while the BE method has only a first order accuracy, the midpoint method has second order accuracy. Since the cost of these methods is comparable, one may wonder why we do not simply use the midpoint method.

To see one of the main drawbacks of the midpoint method, we note that if k is large, then the denominator is roughly of the same magnitude as the numerator, and therefore

$$u^{n+1} \approx -u^n.$$

In this case the solution will be oscillating at every time step. This is one of the diagnostics of a too-large time step within the midpoint method that causes the loss of second order accuracy. To alleviate the problem one can either use smaller steps or use the BE method.

In the above discussion we note that the time stepping requirement was large enough to make the forward Euler method unstable. That is, if we wish to use the forward Euler method, then we use a small time step not for accuracy but for stability. This is one of the characteristics of a *stiff differential equation*. Thus, Maxwell's equations in the time domain, assuming the quasi-static approximation, is a stiff partial differential equation.

4.5.2 ▪ Discretization of Maxwell's equations in time

Choosing time stepping method and time steps

It is straightforward to apply either the midpoint method or the BE method to Maxwell's equations, and we shall do this in the next subsection. However, before simply applying the method, we need to rethink one important issue.

For the geophysical applications we have considered so far, we have neglected the displacement currents, that is, the part that contains $\epsilon\vec{E}_{tt}$. This part yields a wave equation that decays due to the conductivity. If we choose large time steps, then we would not be able to resolve the "wavy" behavior of Maxwell's equations at early times. This begs the question, Can we indeed do that? That is, can we use large time steps, skipping early wave

phenomena, or do we have to resolve the early behavior of Maxwell's equations in order to have a reasonable solution at later times? If we need to accurately integrate through early times, then time stepping can be a truly expensive process.

Luckily, this is not the case. Indeed, if we use the BE method, we can take large time steps, skipping the early, wavy part of the equation and still resolving the decaying part of the system.

We illustrate this point with the following example. We integrate the PDE

$$\epsilon_0 u_{tt} + \sigma u_t - \mu_0^{-1} \Delta u = 0,$$

with initial conditions $u(0,\mathbf{x}) = \delta(\mathbf{x})$ and $u_t(0,\mathbf{x}) = 0$. This PDE can be thought of as a simplification of Maxwell's equations to the scalar case, which occurs, for example, in a uniform media. To have a more realistic simulation, we choose $\epsilon_0 = 8.86 \times 10^{-12}$, $\mu_0 = 4\pi \times 10^{-7}$, and

$$\sigma = \begin{cases} 10^{-3}, & z < 0, \\ 10^{-8}, & z \geq 0. \end{cases}$$

We discretize the equation on a 2D regular mesh and use a standard (five point) discretization to the Laplacian on a uniform mesh size h, that is,

$$\Delta u \approx \frac{1}{4h^2}\left(\mathbf{u}_{i-1,j} + \mathbf{u}_{i+1,j} + \mathbf{u}_{i-1,j} + \mathbf{u}_{i,j+1} - 4\mathbf{u}_{ij}\right).$$

We first use a very small time step of 10^{-7} and use both the midpoint and the BE methods to discretize the PDE. We then use a much larger time step of 10^{-4} to discretize the PDE. We display the results in Figure 4.2.

As can be seen, the BE method with smaller time steps resolves the initial waves and then dumps the waves at later times due to its diffusive behavior. On the other hand, the BE method with long time steps ignores the wavy part altogether and skips to the diffusive part. Comparing the results and experimenting also with the midpoint method, one finds that large time steps for the midpoint method lead to the midpoint oscillations previously discussed, while the BE method keeps its first order accuracy.

To explain the results further, we note that the discrete operator Δ has eigenvalues that scale to h^{-2} (where h is the mesh size). We therefore analyze the ODE

$$\epsilon_0 u_{tt} + \sigma u_t + \mu_0^{-1} h^{-2} u = 0.$$

To do that we consider the equivalent first order system

$$\begin{pmatrix} u \\ v \end{pmatrix}_t = \begin{pmatrix} 0 & \sigma^{-1} \\ -\epsilon^{-1}\sigma & -\epsilon^{-1}\sigma h^{-2}\mu^{-1} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}.$$

We rewrite the system as

$$\dot{\mathbf{u}} = A\mathbf{u}.$$

Discretizing the system using the BE method, we obtain

$$\mathbf{u}^{n+1} = (I - \Delta t\, A)^{-1}\mathbf{u}^n = M_{BE}^n \mathbf{u}_0,$$

and using the midpoint method we obtain

$$\mathbf{u}^{n+1} = \left(I - \frac{1}{2}\Delta t\, A\right)^{-1}\left(I + \frac{1}{2}\Delta t\, A\right)\mathbf{u}^n = M_{MP}^n \mathbf{u}_0,$$

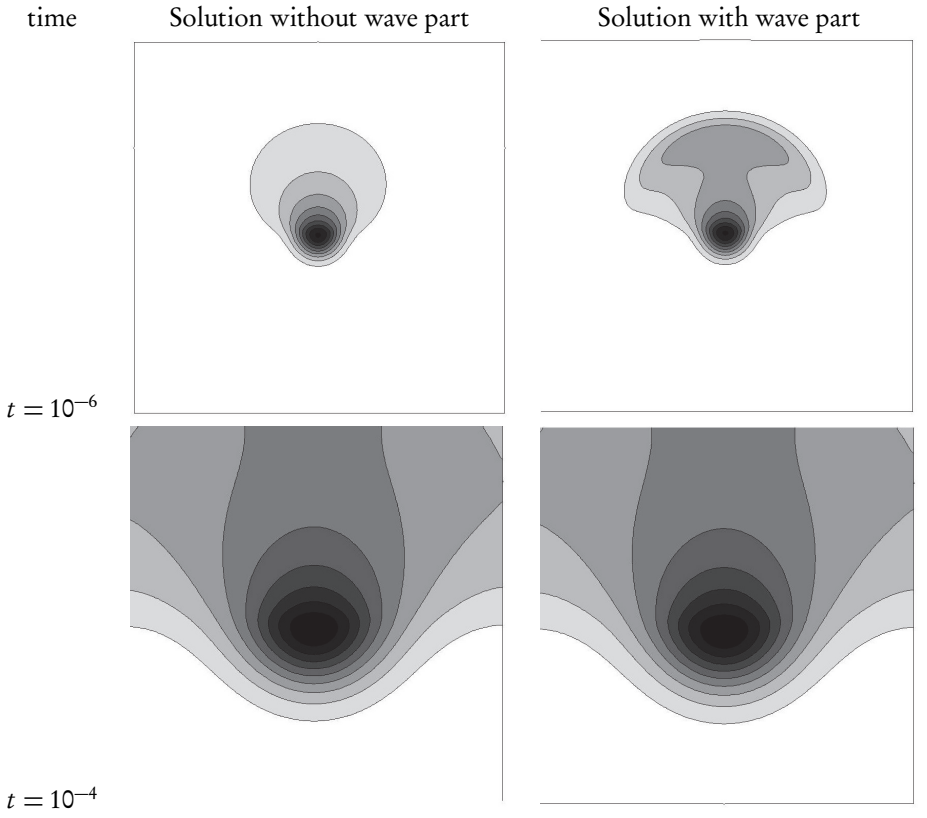

Figure 4.2. *Solution of the heat equation without (left) and with (right) a wave term. While there is a substantial difference at early times, there is no noticeable difference at later times.*

where $M_{BE} = I - \Delta t A$ and $M_{MP} = (I - \frac{1}{2}\Delta t A)^{-1}(I + \frac{1}{2}\Delta t A)$. For the parameters we choose here, we can compute the values of the solution at later times given either very small steps and using the midpoint method or larger time steps and using the BE method. Assume first a more accurate simulation using the midpoint method with a step size of 10^{-7}. We can then use 10^4 time steps to compute the solution at time 10^{-3}. A direct calculation shows that

$$\mathbf{u}(10^{-3}) = M_{MP}^{10^4}\mathbf{u}_0 = \begin{pmatrix} 4.9408 \times 10^{-1} & 4.3776 \times 10^{-10} \\ -3.4836 \times 10^{2} & -3.0864 \times 10^{-7} \end{pmatrix}\mathbf{u}_0,$$

and if we use the BE method with a time step of 10^{-4}, we obtain

$$\mathbf{u}(10^{-3}) = M_{BE}^{10}\mathbf{u}_0 = \begin{pmatrix} 5.0595 \times 10^{-1} & 4.4827 \times 10^{-10} \\ -3.5673 \times 10^{2} & -3.1606 \times 10^{-7} \end{pmatrix}\mathbf{u}_0.$$

Thus, as can be seen here even though our time step was 1000 times smaller in the midpoint method, we lose only roughly 2% accuracy for this simulation.

Using symbolic manipulations for general σ, ϵ, μ, and h, one can find that indeed this particular finding is more general and indeed one can skip early wave phenomena if the measurements are done later.

To conclude, if we are interested in the lossy behavior of Maxwell's equations at later times, then we recommend using the BE method, which allows for skipping early wavy phenomena. Furthermore, using the BE method with a long time step, we can neglect the term $\epsilon \vec{E}_t$ from the first order system. If, on the other hand, measurements are made in earlier times when the waves have not yet traveled and attenuated, then we recommend using the midpoint method, including the displacement current term $\epsilon \vec{E}_t$. Clearly, if we need to evaluate the solution at early as well as at late times, then one can combine methods, say, starting with small time steps and the midpoint method and switching to longer time steps with the BE method.

Backward Euler discretization of Maxwell's equations

Discretizing the system in space, ignoring displacement currents, we obtain the following ODE:

$$\begin{aligned} &\mathbf{b}_t = -\mathsf{CURL}\mathbf{e}, \\ &\mathsf{CURL}^\top \mathbf{M}_{f\mu}\mathbf{b} - \mathbf{M}_{\mathbf{e}\sigma}\mathbf{e} = \mathbf{s}, \\ &\mathbf{b}(0) = \mathbf{b}_0. \end{aligned} \tag{4.9}$$

Using the BE method, we obtain

$$\begin{aligned} &\frac{\mathbf{b}^n - \mathbf{b}^{n-1}}{\Delta t} = -\mathsf{CURL}\mathbf{e}^n, \\ &\mathsf{CURL}^\top \mathbf{M}_{f\mu}\mathbf{b}^n - \mathbf{M}_{e\sigma}\mathbf{e}^n = \mathbf{s}^n, \\ &\mathbf{b}(0) = \mathbf{b}_0. \end{aligned} \tag{4.10}$$

We can now eliminate either $\mathbf{e}^n$ or $\mathbf{b}^n$ from the system. To obtain a symmetric form when eliminating $\mathbf{e}^n$, we define the magnetic field

$$\mathbf{h}^n = \mathbf{M}_{f\mu}\mathbf{b}^n,$$

obtaining the following discrete systems:

$$(\mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} + \Delta t^{-1}\mathbf{M}_{e\sigma})\mathbf{e}^n = -\Delta t^{-1}\mathbf{s}^n + \mathsf{CURL}\mathbf{h}^{n-1}, \tag{4.11a}$$

$$(\mathsf{CURL}\,\mathbf{M}_{e\sigma}^{-1}\,\mathsf{CURL}^\top + \Delta t^{-1}\mathbf{M}_{f\mu}^{-1})\mathbf{h}^n = \Delta t^{-1}\mathbf{M}_{f\mu}^{-1}\mathbf{h}^{n-1} - \mathsf{CURL}\,\mathbf{M}_{e\sigma}^{-1}\mathbf{s}^n. \tag{4.11b}$$

These systems are then solved using some linear solver, and the solution is updated.

Although the discretization above uses a uniform mesh in time, this is clearly not necessary. In fact, one can use a variable time step at each iteration. However, we find that this strategy not useful in general. First, assume that Δt is constant for all time steps. Then the linear system to be solved is identical for all time steps, and one can utilize this when considering either direct methods or preconditioning for iterative solvers (see section 4.6). If the time interval spans many times, then using a single time step is unrealistic. In this case we recommend using, say, 5–20 equispaced time steps per decade or two before switching to a larger time steps. This allows for a compromise between efficiency of adaptive time steps and constant time steps, allowing for the reuse of iterative solvers.

4.5.3 ▪ Initialization of the magnetic fields and their divergence

One important issue when integrating the system (4.10) in time is the initialization of the fields. Regardless of the type of source we use, some calculation is needed in order to evaluate the initial fields over the mesh. Recall from Chapter 1 that the computations of the magnetic field involve solving the discrete system (1.20).

We rewrite the system using the vector potential $\vec{A}$, assuming the Coulomb gauge condition $\nabla \cdot \vec{A} = 0$:

$$\nabla \cdot \sigma \nabla \phi = -\nabla \cdot \vec{s}, \tag{4.12a}$$

$$\nabla \times \mu^{-1} \nabla \times \vec{A} - \nabla \mu^{-1} \nabla \cdot \vec{A} = -\vec{s} - \sigma \nabla \phi. \tag{4.12b}$$

This system is discretized on a staggered grid where $\vec{A}$ is on the edges and ϕ is on the nodes and solved to obtain a discrete vector potential $\mathbf{a}$ and a discrete scalar potential $\boldsymbol{\phi}$. The discretization of the initial conditions $\vec{B}_0$ is performed by using the CURL matrix, setting $\mathbf{b}_0 = .\mathsf{CURL}\mathbf{a}$.

An important aspect of this construction of the initial conditions is that the discrete analogue to $\vec{B}_0$, namely, $\mathbf{b}_0$, is discretely divergent-free; that is,

$$\mathsf{DIV}\,\mathbf{b}_0 = 0.$$

Since the right-hand side of (4.11b) is also divergence-free in exact arithmetic, we obtain

$$\mathsf{DIV}\,\mathbf{b}^k = 0$$

for all k.

4.5.4 ▪ Exponential integrators

In this subsection we discuss a more recent development within time stepping techniques applied to Maxwell's equations (see [32]). These are exponential integrators that can be very competitive for a few applications. To do this we eliminate $\mathbf{e}$ from (4.10) and assume that $\mu = \mu_0$, set $\mathbf{h} = \mathbf{M}_{f\mu}\mathbf{b}$, and obtain the ODE

$$\mathbf{h}_t = -\mu_0 \mathsf{CURL}\,\mathbf{M}_{e\sigma}^{-1}\,\mathsf{CURL}^\top \mathbf{h} + \mathbf{s} = \mathbf{A}\,\mathbf{h} + \mathbf{s}, \tag{4.13}$$

where $\mathbf{A} = -\mu_0 \mathsf{CURL}\,\mathbf{M}_{e\sigma}^{-1}\,\mathsf{CURL}^\top$. This is a first order linear ODE, and therefore it has an analytic solution

$$\mathbf{h}(t) = \exp(t\mathbf{A})\mathbf{h}_0 + \int_0^t \exp((\tau - t)\mathbf{A})\mathbf{s}(\tau)\, d\tau.$$

To understand the meaning of the above formula we recall that the exponent of a matrix can be defined using the Taylor series:

$$\exp(\mathbf{M}) = \mathbf{I} + \mathbf{M} + \frac{1}{2}\mathbf{M}^2 + \frac{1}{6}\mathbf{M}^3 + \ldots.$$

It is straightforward to show that if we can decompose $\mathbf{M}$ into its eigenvalues and vectors

$$\mathbf{M} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top,$$

where $\mathbf{V}\mathbf{V}^\top = \mathbf{V}^\top\mathbf{V} = \mathbf{I}$ and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, then

$$\exp(\mathbf{M}) = \mathbf{V}\exp(\mathbf{\Lambda})\mathbf{V}^\top,$$

where $\exp(\mathbf{\Lambda}) = \mathrm{diag}(\exp(\lambda)) = \mathrm{diag}(\exp(\lambda_1), \ldots, \exp(\lambda_n))$.

This suggests the following algorithm for the computation of $\exp(t\mathbf{A})\mathbf{h}_0$. First, compute the decomposition $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}V^\top$. Second, for every time t compute

$$\exp(t\mathbf{A})\mathbf{h}_0 = \mathbf{V}\Big(\exp(\lambda t) \odot (\mathbf{V}^\top\mathbf{h}_0)\Big).$$

This implies that, using a single decomposition, one can compute the whole time series, at least for a time dependent problem without a source.

Next, consider the case in which we have a time dependent source. Then, we need to compute the integral

$$\mathbf{g}(t) = \int_0^t \exp((\tau - t)\mathbf{A})\mathbf{s}(\tau)\, d\tau.$$

This can be approximated by some quadrature rule, say, for simplicity, the midpoint method:

$$\mathbf{g}(t) \approx \delta\tau \sum_{k=1}^{n_T} \exp((\tau_k - t)\mathbf{A})\mathbf{s}(\tau_k).$$

We see that we need to evaluate $\exp((\tau_k - t)\mathbf{A})\mathbf{s}(\tau_k)$ for n_T different vectors $\mathbf{s}(\tau_k)$. Luckily, this is not really the case. The source $\mathbf{s}$ is typically fixed in space and changes in time and therefore can be written as

$$\mathbf{s}(t) = \mathbf{q}\,\gamma(t),$$

where $\mathbf{q}$ is a fixed vector and $\gamma(t)$ is a scalar function. This implies that we can approximate $\mathbf{g}$ as

$$\mathbf{g}(t) \approx \delta\tau \sum_{k=1}^{n_T} \gamma(\tau_k)\, \exp((\tau_k - t)\mathbf{A})\mathbf{q} = \sum_{k=1}^{n_T} \gamma(\tau_k)\, \mathbf{V}\Big(\exp(\lambda(\tau_k - t)) \odot (\mathbf{V}^\top\mathbf{q})\Big),$$

which implies that we need only compute $\exp((\tau_k - t)\mathbf{A})\mathbf{q}$ for all times between 0 and t.

In the above we have assumed the Schur decomposition to eigenvalues and eigenvectors. This decomposition is rather expensive, and therefore other approximations can be used. One obvious idea is to use a similar approximation

$$\mathbf{A} \approx \mathbf{V}_j\mathbf{T}_j\mathbf{V}_j^\top,$$

where $\mathbf{V}_j$ is an $n \times j$ orthogonal matrix and $\mathbf{T}_j$ is a $j \times j$ matrix.

One such method is the shift and invert method. A full description can be found in [62]. The method is a Krylov space method that involves the inversion of the matrix $(I - \xi\mathbf{A})$, j times, for some choice of parameter ξ.

Using such an approximation, the matrix exponent is given by

$$\exp(t\mathbf{A})\mathbf{q} \approx \mathbf{V}_j \exp(t\mathbf{T}_j)(\mathbf{V}_j^\top\mathbf{q}).$$

The advantage of this approach is that since j is typically small, one may be able to compute an approximation to the exponential over many times using only j matrix inversions.

This is equivalent to taking j time steps in, say, a BE method. If j can indeed be small, then this method can be advantageous for the computation of time dependent signals.

Further advances can be made by generating the matrices $\mathbf{V}_j$ using a number of shift parameters, computing products of the form $(\mathbf{I}-\xi_j\mathbf{A})\mathbf{z}_j$. This method is referred to as the rational Krylov method [32]. Yet, in my experience, exponential integrators have two main disadvantages. First, when there is a nontrivial source, the computation requires many inversions of matrices of the above form, which lead to a cost similar to that of the BE method. Second, and more important, computing derivatives of the solution with respect to the conductivity is highly nontrivial and requires careful numerical treatment. Finally, when the number of times that is recorded is bigger than 16 or so, the computation of the derivatives does not pay off. For these reasons we will not continue to consider exponential integrators here but note that they are worthy of further research.

4.6 ▪ Solving the linear systems

The bottleneck of solving any forward problem is the solution of the linear system that results from the discretization of Maxwell's equations.

In the frequency domain the system has the form

$$\mathbf{A}\mathbf{e}_j = (\mathbf{K}+i\omega\mathbf{M}_\sigma)\mathbf{e}_j = \mathbf{q}_j, \tag{4.14}$$

where $\mathbf{K}$ is a symmetric positive semidefinite matrix that results from the discretization of the $\nabla\times\mu^{-1}\nabla\times$ operator and $\mathbf{M}_\sigma$ is a positive diagonal matrix (assuming σ is a scalar function). The fields $\mathbf{e}_j$, $j=1,\ldots,n_s$, correspond to the different sources $\mathbf{q}_j$, $j=1,\ldots,n_s$. This linear system is complex and symmetric.

In the time domain, assuming some implicit method, the linear system has the form

$$\mathbf{A}_k\mathbf{e}_j = (\mathbf{K}+\gamma_k\mathbf{M}_\sigma)\mathbf{e}_j = \mathbf{q}_j,$$

where $\gamma_k > 0$. This system is symmetric positive definite.

For both frequency and time domain formulations the systems are large and sparse. For many problems it can easily reach millions of degrees of freedom, and, as we explore next, is usually highly ill-conditioned. Therefore, the solution of this system is a serious challenge, and we discuss it next.

Recall: Condition number. The L_2 condition number of a square matrix $\mathbf{A}$ is defined as the ratio between the maximal eigenvalue and the minimal one (in modulus)

$$\varkappa(\mathbf{A}) = \frac{\max\lambda(\mathbf{A})}{\min\lambda(\mathbf{A})}.$$

4.6.1 ▪ The ill-conditioning of the discrete Maxwell system

We are interested in solving Maxwell's equations for low frequencies or late times. This implies that ω and γ are small. The matrix $\mathbf{M}_\sigma$ contains σ on its main diagonals and therefore is bounded between $\sigma_{\min}$ and $\sigma_{\max}$ (with volume scaling). The matrix $\mathbf{K}$ is a discretization of a differential operator. Therefore, as the mesh refines, its largest eigenvalue goes to infinity, while its lower eigenvalue stays constant (assuming volume scaling). This

ill-conditioning is common to all differential operators. However, in our case, there is an increased complexity. The **curl** operator has a nontrivial null space. Indeed, the discretization of the nodal gradient is in the null space of the discrete edge **curl**, and, as a result,

$$\mathbf{K}\,\mathsf{GRAD} = 0.$$

Since ω and γ are small, this implies that the system is close to singular and its condition number is very large.

We now discuss a number of strategies to solve this system that deal with this approximate singularity.

4.6.2 ▪ Iterative solvers and the discrete Helmholtz decomposition

To choose an iterative method we need to separate the frequency domain and the time domain cases. In the time domain the linear system is symmetric positive definite (SPD), and therefore one should use the preconditioned conjugate gradient (PCG).

For the frequency domain we need to deal with a complex symmetric system. The candidates for this problem are the generalized miminal residual (GMRES) and the biconjugate gradient stabilized (BiCGSTAB). In my experience BiCGSTAB performs better than GMRES, even though it lacks any real theory for convergence.

Regardless of the iterative technique that is used, these methods tend to be highly inefficient for ill-conditioned problems, where the eigenvalues are spread over many scales, and thus converge slowly when applied to our problems. To amend this difficulty one typically must use two complementary mechanisms. First, in many cases it is possible to reformulate the linear system such that it has much better numerical properties. Second, it is common to use a preconditioner; that is, rather than solving the original problem $\mathbf{Ae} = \mathbf{q}$, we multiply the matrix and the right-hand side by another matrix, $\mathbf{P}$, and solve the system

$$(\mathbf{PA})\mathbf{e} = \mathbf{Pq}.$$

The idea is that the product of $\mathbf{P}$ and a vector is relatively cheap, and the new system with the matrix $\mathbf{PA}$ is better conditioned compared with the original matrix $\mathbf{A}$.

"Standard" stationary preconditioners are Jacobi, symmetric Gauss–Seidel, and SSOR (see [64] for a discussion). These preconditioners are easy to apply and work well if the system is not poorly conditioned. Furthermore, even if the system is ill-conditioned, they can work fairly well and thus are very useful for preconditioning moderate scale problems. For very large scale problems more sophisticated preconditioners are needed. In particular, multigrid methods are popular. Nonetheless, while programming standard preconditioners takes a few minutes, programming multigrid methods is a significant effort. Furthermore, multigrid methods can be quite sensitive to jumping coefficients and changes in boundary conditions, and therefore, while they have been in the center of research, they are rarely used in industrial strength codes, where robustness is more important than speed.

Applying standard preconditioners to Maxwell's equations tends to lead to less than satisfying results. This is due to the null space of the CURL that leads to highly ill-conditioned systems. We now discuss how to reformulate the discrete Maxwell system such that we remove the issues associate with the null space of the CURL. The idea is to use the discrete Helmholtz decomposition, setting

$$\mathbf{e} = \mathbf{a} + \mathsf{GRAD}\,\boldsymbol{\phi}.$$

Note that $\mathsf{GRAD}\boldsymbol{\phi}$ is in the null space of $\mathbf{K}$. Since we introduce a new vector $\mathsf{GRAD}\boldsymbol{\phi}$, we have some degrees of freedom that can be chosen. To make sure that no parts of $\mathbf{a}$ are in the null space of $\mathbf{K}$, we demand that

$$\mathsf{GRAD}^\top \mathbf{a} = 0.$$

Substituting this decomposition, we obtain that

$$\begin{pmatrix} \mathbf{K} + i\omega\mathbf{M} & i\omega\mathbf{M}\mathsf{GRAD} \\ \mathsf{GRAD}^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ 0 \end{pmatrix}.$$

This is a saddle point system, and therefore it contains both positive and negative (real part) eigenvalues. The system can be solved considering its special structure using specialized methods [9, 30]. However, it is possible to obtain a simpler system that behaves better and is easier to solve using "standard" techniques.

The first step is to change the (1,1) block by adding a 0 term that eliminates the null space of $\mathbf{K}$, obtaining

$$(\mathbf{K} + \mathsf{GRAD}\mathbf{M}_n\mathsf{GRAD}^\top + i\omega\mathbf{M})\mathbf{a} + i\omega\mathbf{M}\mathsf{GRAD}\boldsymbol{\phi} = \mathbf{q},$$

where $\mathbf{M}_n$ is an arbitrary positive definite matrix. The matrix is typically chosen such that the resulting $\mathbf{K} + \mathsf{GRAD}\mathbf{M}_n\mathsf{GRAD}^\top$ is a discretization of $\nabla\times\,\mu^{-1}\nabla\times\, - \nabla\mu^{-1}\nabla\cdot$. Multiplying the system from the left with $\mathsf{GRAD}^\top$ using the relations $\mathsf{GRAD}^\top\mathbf{K} = 0$ and recalling that $\mathsf{GRAD}^\top\mathbf{a} = 0$, we obtain

$$\mathsf{GRAD}^\top\mathbf{M}\mathbf{a} + \mathsf{GRAD}^\top\mathbf{M}\mathsf{GRAD}\boldsymbol{\phi} = (i\omega)^{-1}\mathsf{GRAD}^\top\mathbf{q}.$$

Putting the two equations together, we obtain the so-called $\mathbf{a}, \boldsymbol{\phi}$ system

$$\begin{pmatrix} \mathbf{K} + \mathsf{GRAD}\mathbf{M}_n\mathsf{GRAD}^\top + i\omega\mathbf{M} & i\omega\mathbf{M}\mathsf{GRAD} \\ \mathsf{GRAD}^\top\mathbf{M} & \mathsf{GRAD}^\top\mathbf{M}\mathsf{GRAD} \end{pmatrix} \begin{pmatrix} \mathbf{a} \\ \boldsymbol{\phi} \end{pmatrix} = \begin{pmatrix} \mathbf{q} \\ \frac{1}{i\omega}\mathsf{GRAD}^\top\mathbf{q} \end{pmatrix}. \tag{4.15}$$

The system (4.15) is a well-behaved linear system, and therefore standard preconditioners and solvers can be used. For example, we had very good results using BiCGSTAB with symmetric Gauss–Seidel as a preconditioner. More sophysticated solvers and, in particular, multigrid, can also be applied to the system, yielding mesh independence convergence [3]. One should be warned that multigrid solvers are only "optimal" when the system is sufficiently large. Implementing a multigrid solver is not an easy task, and unless one intends to solve very large scale problems, one should avoid them and address other inefficiencies in the code first.

It is possible to solve the system (4.15) instead of (4.14). However, it is also possible to use system (4.15) or an approximation of the system as a preconditioner. To do that we note that we can write

$$\mathbf{e} = \begin{pmatrix} \mathbf{I} & \mathsf{GRAD} \end{pmatrix} \mathbf{A}_{\mathbf{a}\boldsymbol{\phi}}^{-1} \begin{pmatrix} \mathbf{I} \\ (i\omega)^{-1}\mathsf{GRAD}^\top \end{pmatrix} \mathbf{q},$$

where $\mathbf{A}_{\mathbf{a}\boldsymbol{\phi}}^{-1}$ is the $\mathbf{a}, \boldsymbol{\phi}$ system. Assume that $\widehat{\mathbf{A}}_{\mathbf{a}\boldsymbol{\phi}}^{-1}$ is an approximation to $\mathbf{A}_{\mathbf{a}\boldsymbol{\phi}}^{-1}$, obtained, for example, by symmetric Gauss–Seidel. More efficient preconditioners can be obtained by

using a multigrid cycle to approximate the action of $\mathbf{A}_{\mathbf{a}\phi}^{-1}$ on a vector. It is possible to set the preconditioner for the system (4.14) as

$$\mathbf{P} = \begin{pmatrix} \mathbf{I} & \mathsf{GRAD} \end{pmatrix} \widehat{\mathbf{A}}_{\mathbf{a}\phi}^{-1} \begin{pmatrix} \mathbf{I} \\ (i\omega)^{-1}\mathsf{GRAD}^{\top} \end{pmatrix}.$$

In our experience this preconditioner works well for most problems as long as the frequency, ω, is low.

4.6.3 ▪ Solution using direct solvers

While iterative solvers are efficient for large scale problems and typically use very little computer memory, they suffer from one main drawback. Most problems require the solution of the linear system many times for different sources/time steps and for the computation of the derivatives. For example, assuming a single frequency, the linear system has to be solved for at least $N_{\text{sources}} \times 2$, where N_{sources} is the number of sources, and in many cases to a much larger number than that (this is due, as we discuss in the next chapters, to the estimation of the gradients and Hessians). For many problems N_{sources} is rather large and could easily reach a few thousands or even millions. For such cases the linear systems are solved tens or even hundreds of thousands or millions of times! For these applications, direct factorization is, by far, the best way to deal with the linear system. In direct factorization we use some version of Gaussian elimination to compute the Cholesky (if $\mathbf{A}$ is symmetric) or the LU (if $\mathbf{A}$ is complex) decomposition, setting (in the LU case)

$$\mathbf{A} = \mathbf{LU},$$

where $\mathbf{L}$ is lower tridiagonal and $\mathbf{U}$ is upper tridiagonal. The matrices $\mathbf{L}$ and $\mathbf{U}$ are stored, and the problem can be solved repeatedly for all sources and derivatives.

For problems in 1D and for most problems in 2D direct factorization methods can be easily used with many packages; however, for problems in 3D and four dimensions (4D) (space-time) it is not always possible to factorize the system without specialized software and hardware. Recently, parallel direct solvers have been available—for example, MUMPS [2] and superLU [19]. Given the appropriate computational resources, these codes can solve problems with millions of unknowns by factorization. For problems in 3D such solvers are preferable, but in order for them to be effective for large scale problems some advanced computational hardware is required.

4.7 ▪ Example: Modeling a magnetic loop in conductive media

In this section we put all of the above mechanisms into a case study of modeling a magnetic loop in conductive media. This is a common geophysical experiment that is sketched in Figure 4.3.

The loop is assumed to be 15 meters in diameter, and we assume that it is 30 meters off the ground. The receiver is located in the center of the loop and records the $\vec{H}_z$ component. This is a classical airborne survey. We assume two types of experiments. First, we assume a time dependent experiment where the source is a step-off and that we measure at times of 10^{-4} to 10^{-2} seconds. Second, we assume a frequency domain experiment where we record at frequencies of 10^1 to 10^3 Hz.

Solving the forward problem is then done by the following steps:

- mesh generation and mesh quality testing,

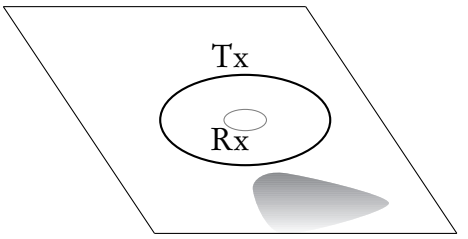

Figure 4.3. *A magnetic source (closed loop) is used to generate currents by induction, and the response of the earth is measured in terms of magnetic fields on the surface.*

- discretization of the conductivity on the mesh,
- discretization of the operators,
- discretization of the source terms,
- solution of the linear system,
- displaying the results and checking their validity.

We now discuss each of these steps in more detail.

Mesh generation

When solving most geophysical problems we can roughly divide the domain into two parts. First, there is the domain of interest. This is where the sources and receivers reside and where changes in the conductivity make a significant change to the data. Typically, one uses a regular, or almost regular, mesh in these regions. Second, there is the padding area. Since geophysical problems are solved in an unbounded domain, one typically needs to use a sufficiently large mesh to pad to "infinity." The grid that is stretched to infinity needs a reasonable aspect ratio in order to have appropriate approximation properties. Practical experience is that the grid should not stretched by more than a factor of 2; "urban legend" suggests that the "magic" number is 1.3. To compute how far we should stretch our grid we need to look at two physical phenomena. First, assuming a plain wave in conductive media, the decay of the wave is governed by the skin depth [81], where the amplitude of the wave decays to $e^{-1} \approx 0.36$ of its original amplitude and can be approximated as

$$\mathsf{sd} \approx 500\sqrt{\frac{1}{\sigma f}} \text{ in frequency} \quad \text{or} \quad \mathsf{sd} \approx 890\sqrt{\frac{2t}{\sigma}} \text{ in time,}$$

where σ is the conductivity, f is the angular frequency (in Hz), and t is the time. As an example, for a frequency of 10^3Hz and an average conductivity of 10^{-3}S/m, the skin depth is roughly 500 meters. A mesh that is stretched to, say, 2500 meters reduces its strength to less than 1% of its initial value. However, looking only at skin depth would be overly pessimistic. For example, if the frequency is 0.1Hz, then the same background conductivity has a skin depth of approximately 5×10^4 meters. If the source is a plain wave, then, indeed, one needs to consider conductors at depth. However, at the far field, any finite source behaves either as an electric dipole or as a magnetic dipole that decay as R^3, where R is the distance to the dipole. Therefore, even for low frequencies or long times,

the signal will decay rapidly, and we find by practical experience that for most problems, a few kilometers of padding will do.

Another important question is how small the smallest cell in the mesh should be. Unfortunately this is also not an easy question to answer. In principle the mesh has to be sufficiently small to model the conductivity at the scale that affects the data we are after. For most exploration problems the mesh is chosen between a few meters and a few hundred meters. When using a coarse mesh, care must be taken to verify that cell size is not at the same order as the skin depth.

The above is just a guideline. As we see next, when working with a concrete example, we generate a number of meshes and compare the field with the analytic response if possible and to finer meshes.

4.7.1 ▪ Source generation

The source for this problem is an impulse response; that is, we assume we have a static magnetic field that is being generated by a loop. This magnetic field is shut down at time 0, and the data are recorded afterward. In many cases, there is no instant shut-off, and the source contains some waveform. Here we do not model the waveform, though this can be easily done.

It is important to note that in the case of an inductive source, the magnetic field of a vanishing time derivative is independent of the conductivity. Assuming that the magnetic susceptibility is known and constant, the initial field can be computed in two ways. First, we can approximate the loop as a segment of wires and use the above code to compute the magnetic field of each wire. A second approach is to compute the magnetic field directly using the theoretical response of a loop. The code for the response of a loop is given below.

```
function A = magnetostaticsCurrentLoop(X, a, x0)
% A = magnetostaticsCurrentLoop(x, a, x0)
% Compute magnetic vector potential of horizontal circular current loop
% of radius a located at the origin.

n = size(X, 1);
if size(X, 2) ~= 3
  error('X must be an n x 3 array');
end

x = X(:, 1) - x0(1); y = X(:, 2) - x0(2); z = X(:, 3) - x0(3);


r = sqrt(x.^2 + y.^2);
m = (4 * a * r) ./ ((a + r).^2 + z.^2);
m(m > 1) = 1; % m might be slightly larger than 1 due to rounding errors
              % but ellipke requires 0 <= m <= 1

[K, E] = ellipke(m);

i = r > 0 & m < 1; % 1/r singular at r = 0 and K(m) singular at m = 1

Aphi = zeros(n, 1);
Ax   = zeros(n, 1);
Ay   = zeros(n, 1);
Az   = zeros(n, 1);

% Common factor is (mu * I) / pi with I = 1 and mu = 4e-7 * pi.
Aphi(i) = 4e-7 ./ sqrt(m(i)) .* sqrt(a ./ r(i)) .* ...
  ((1 - m(i) / 2) .* K(i) - E(i));
Ax(i)   = Aphi(i) .* (-y(i) ./ r(i));
Ay(i)   = Aphi(i) .* ( x(i) ./ r(i));

A = [Ax Ay Az];
```

Note that here we do not compute the magnetic flux. We rather compute a discrete approximation to the magnetic potential $\vec{A}$ and compute the initial magnetic field by tak-

ing the **discrete curl** of the discrete potential. This ensures that the discrete magnetic flux is divergence-free.

4.7.2 ▪ Calculation of the data

The next step in the simulation is to use the discretization and the source to compute the data. The following code computes the data and the electric field in space time.

```
function[d,e] = getTimeDomainData(m,param)
% [d,e] = getMisfit(m,param)
% Curl'*Mmuinv*Curl*en + 1/dt*Msig*en = 1/dt * Msig*eo

% magnetic permeability
mu = 4*pi*1e-7 * ones(prod(param.nc),1);
% extract matrices
Curl = param.Curl;  Grad = param.Grad;
Af   = param.Af;      Ae   = param.Ae;
An   = param.An;    Iact = param.Iact;
V    = param.V;
% inactive cells (earth+air)
sigma = param.mref + Iact*m;

% set up mass matrices
Mmuinv  = sdiag(Af'*(V*(1./mu)));
Msig    = sdiag(Ae'*(V*sigma));
Mmuinvn = sdiag(An'*V*(1./mu));

% initial conditions
b0   = param.b0; e0   = Msig\(Curl'*Mmuinv*b0);
dt   = param.dt;

e = zeros(size(Curl,2),length(param.dt)+1);
d = zeros(3,length(dt));
e(:,1) = e0;

% time step
for i=1:length(dt)

    % The linear system to be solved
    Ke = Curl'*Mmuinv*Curl + 1/dt(i)*Msig;
    % solve
    rhs = 1/dt(i)*Msig*e(:,i);
    e(:,i+1) = solveSystem(Ke,Msig,Mmuinvn,Grad,dt(i),rhs);
    % compute the data
    d(:,i) = param.obs*Curl*e(:,i+1);
end
```

The time dependent problem requires the solution of a linear system at each step. The code solveSystem uses the $\mathbf{a}, \phi$ decomposition discussed in (4.15). The linear solver is PCG with a simple Gauss–Seidel preconditioner applied to the $\mathbf{a}, \phi$ system. More sophisticated methods include applying a multigrid method for the $\mathbf{a}, \phi$ system (4.15); see [3]. However, we have found that for many of our problems the Gauss–Seidel preconditioner, while not "optimal," usually suffices.

```
function en = solveSystem(Ke,Msig,Mmuinvn,Grad,dt,rhs)

% set up preconditioner using Aphi system
STBa = Grad*Mmuinvn*Grad';
Aap = [Ke + STBa,          1/dt*Msig*Grad; ...
       1/dt*Grad'*Msig,    1/dt*Grad'*Msig*Grad];

Map = @(x) tril(Aap)\(diag(Aap).*(triu(Aap)\x));
P1  = [speye(size(Ke,2)); Grad'];
P2  = [speye(size(Ke,2)), Grad];

MM = @(x) P2*(Map(P1*x));
% The rhs

[en,~] = pcg(Ke,rhs, 1e-9,1000,MM);
```

4.7.3 ▪ The data from a block in a half-space

Assume that we would like to use the code in order to simulate the response of a conductive block in a half-space. Given the code and a problem, a few questions arise. What discretization in space and time should be used? How far should we stretch infinity? While in some cases one can answer these questions analytically, for most problems this has to be answered numerically. The following is a script for running the forward problem. We can now run this script choosing different padding and different discretizations in time and space. By running it a number of times with different padding and different discretizations in space and time we can develop a "feel" for the accuracy that we can obtain using these techniques.

To set up the problem we start by choosing a number of parameters and the mesh.

```
% set up EM problem

% loop size
aloop = 10;
% set up the mesh
npad = 10;   nin  = 4;
padxy = 1.2.^[1:npad];   padz  = 1.2.^[1:npad];
h0    = 100;
h1    = h0*[fliplr(padxy),ones(1,nin+1),padxy]; h1 = h1(:);
h2    = h0*[fliplr(padxy),ones(1,nin+1),padxy]; h2 = h2(:);
h3    = h0*[fliplr(padz),ones(1,nin),padz];          h3 = h3(:);
n1 = length(h1); n2 = length(h2); n3 = length(h3);
nc = [n1, n2, n3];

% loop center
x0 = [sum(h1)/2+h0/2,sum(h2)/2+h0/2,sum(h3)/2+h0/2];
```

Next, we generate the differential operators and set up the active part of the model. The active part is the part that is not air.

```
%% Compute differential operators
Curl = getEdgeCurlMatrix(n1,n2,n3);
Grad = getNodalGradientMatrix(n1,n2,n3);
Div  = getFaceDivergenceMatrix(n1,n2,n3);
Af   = getFaceToCellCenterMatrix(n1,n2,n3);
Ae   = getEdgeToCellCenterMatrix(n1,n2,n3);
An   = getNodalToCellCenterMatrix(n1,n2,n3);

[V,F,L] = getMeshGeometry(h1,h2,h3);

Div = V\(Div*F); Grad = L\Grad; Curl = F\(Curl*L);

%% set up active cells (cells that are in the earth)
ind  = reshape(1:n1*n2*n3,n1,n2,n3); ind  = ind(:,:,1:n3/2);
Iact = speye(n1*n2*n3); Iact = Iact(:,ind);
```

We now generate the sources. Note that we use two new codes, getEdgeGrid and getFaceGrid. These codes simply give the coordinates of the staggered grid.

```
% Compute the initial condition by computing magetic potential
[Xe1,Ye1,Ze1,Xe2,Ye2,Ze2,Xe3,Ye3,Ze3] = getEdgeGrid(h1,h2,h3);
A = magnetostaticsCurrentLoop([Xe1(:),Ye1(:),Ze1(:)], aloop, x0);  a1 = A(:,1);
A = magnetostaticsCurrentLoop([Xe2(:),Ye2(:),Ze2(:)], aloop, x0);  a2 = A(:,2);
A = magnetostaticsCurrentLoop([Xe3(:),Ye3(:),Ze3(:)], aloop, x0);  a3 = A(:,3);
a  = [a1;a2;a3];

b0 = Curl*a;
```

To obtain the value of the fields at the receiver we interpolate to the receiver location. This is done by the function interpmat.

```
xr = x0;
[Xf1,Yf1,Zf1,Xf2,Yf2,Zf2,Xf3,Yf3,Zf3] = getFaceGrid(h1,h2,h3);

Px = interpmat(unique(Xf1(:)),unique(Yf1(:)),unique(Zf1(:)),xr(1),xr(2),xr(3));
Py = interpmat(unique(Xf2(:)),unique(Yf2(:)),unique(Zf2(:)),xr(1),xr(2),xr(3));
Pz = interpmat(unique(Xf3(:)),unique(Yf3(:)),unique(Zf3(:)),xr(1),xr(2),xr(3));
P  = blkdiag(Px,Py,Pz);
```

Finally, we put it all together into a structure to be fed into the forward problem.

```
%% lump it all in a structure
param.Curl = Curl;
param.Grad = Grad;
param.Af   = Af;
param.Ae   = Ae;
param.An   = An;
param.V    = V;
param.b0   = b0;
param.Iact = Iact;
param.obs  = P;
param.dt   = ones(20,1)*5e-4;
param.mref = 1e-7;
param.nc   = nc;
param.nf   = prod(nc+[1,0,0]) + prod(nc+[0,1,0]) + prod(nc+[0,0,1]);

% set up conductivity
m = zeros(n1,n2,n3/2) + 1e-2;  % background
m(fix(n1/2)-3:fix(n1/2)+3,fix(n2/2)-3:fix(n2/2)+3, n3/2-5:n3/2-2) = 1; % block
m = m(:);
% get the data
[d,e] = getTimeDomainData(m,param);
```

A common strategy is to compute the data for a given discretization in space and time and then divide the space and time intervals in two and rerun the code. For each of these runs we compute the data, and the mesh is considered reasonable when the data does not change as the mesh refines or padding is added.

A different approach is to use an analytic example. For example, in Chapter 1 we have explored the field on the top of the earth due to a magnetic dipole. This expression can be used to verify the numerical results and help in choosing an appropriate mesh.

4.8 ▪ Exercises for Maxwell's simulation

1. Change the code getTimeDomainData.m to a frequency domain code getFreqDomainData.m.

2. Program an electric line source, and use the getFreqDomainData.m to compute data on the surface from a 1000m line source at frequencies of $0.1, 1, 10, 100$, and 1000 Hz. Plot the real and imaginary components for the different frequencies.

3. Use (1.24) and change the code getTimeDomainData.m to handle polarizable material.

Chapter 5
Sensitivity Calculation

5.1 ▪ The concept of sensitivities

In the previous chapters we assumed that we have a system of PDEs, in particular, Maxwell's equations, that after discretization can be written in general form as

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

Here $\mathbf{u}$ are the fields $\mathbf{e}$ and/or $\mathbf{b}$ (depending on the formulation we use), and $\mathbf{m}$ can be any parameters that control the solution, that is, the conductivity, σ, the magnetic susceptibility, μ, or the source term and boundary conditions. The term $\mathbf{m}$ is often referred to as the "model," and in the next chapter we discuss its estimation. However, before we attempt to estimate the model, there is one important task at hand. We need to evaluate the sensitivity of the fields with respect to changes in the model $\mathbf{m}$. This is the second basic ingredient in any parameter estimation algorithm for large scale problems (where direct methods cannot be used). The sensitivities are also very useful when trying to understand the importance of certain parameters in the simulation. Large sensitivities with respect to some parameters imply that changes in these parameters can significantly change the solution of the differential equation, while small sensitivities imply that the fields are not very sensitive with respect to the parameters.

In this chapter, we discuss in detail how to compute the sensitivities for different problems. While it is possible to treat the sensitivities in infinite dimensions as integral operators, we prefer to discuss the discrete sensitivities. This choice has one major advantage: Since the sensitivities are needed in order to evaluate the gradients in the inverse problem, computing the sensitivities by differentiating the *discrete* forward solution yields *exact discrete* gradients. Since gradient information is crucial to any gradient descent type method, having an exact gradient can be important in obtaining convergence. Also, the treatment in discrete space is much simpler compared to its continuous analogue. One disadvantage of the discrete approach is that one may lose some insight into the structure of the sensitivity, but, as we show, this insight can be obtained by interpreting the discrete quantities alongside their continuous analogues.

5.2 ▪ The basic sensitivity equation

Assume that the Maxwell system is written as $C(\mathbf{u}, \mathbf{m}) = 0$. Given the parameters $\mathbf{m}$, we can solve for $\mathbf{u}$ as a function of the parameters $\mathbf{m}$; that is, $\mathbf{u} = \mathbf{u}(\mathbf{m})$. In most cases, $\mathbf{u}$ is

not of interest everywhere, and given $\mathbf{u}(\mathbf{m})$ we typically obtain *data* by another operation. Thus, we assume that the data, $\mathbf{d}$, are given by an operation, that is,

$$\mathbf{d} = \mathbf{P}(\mathbf{u}(\mathbf{m})),$$

where $\mathbf{P}$ is some operator from the fields, $\mathbf{u}$, to the data, $\mathbf{d}$.

As an example, consider that the model is the log conductivity, $\mathbf{m} = \log(\sigma)$, and that the data are the magnetic fields at a number of observation points. In this case, the operator $\mathbf{P}$ is simply a matrix that interpolates the magnetic fields from their computational points (faces) to the measurement points.

A more complex operation is the case in which the data are given by the amplitude and phase of the magnetic field. In this case $\mathbf{P}$ is a nonlinear operation that involves the interpolation of the magnetic fields followed by the computation of the amplitude and phase at the interpolation points. In the following, we treat the case in which $\mathbf{P}$ is linear and discuss the nonlinear case at the end of this chapter.

To define the sensitivity, we treat $\mathbf{d}$ as a function of $\mathbf{m}$; that is, $\mathbf{d} = \mathbf{P}\mathbf{u}(\mathbf{m})$. The question is "How does $\mathbf{d}$ changes when we change $\mathbf{m}$?" To understand the local behavior of the function we look at Taylor's series that reads

$$\mathbf{P}\mathbf{u}(\mathbf{m} + h\mathbf{v}) = \mathbf{P}\mathbf{u}(\mathbf{m}) + h\mathbf{P}\frac{\partial \mathbf{u}}{\partial \mathbf{m}}\mathbf{v} + \mathcal{O}(h^2).$$

For small h, the nonlinear relation can be approximated by the linear one. We therefore use the linear relation and define the sensitivity matrix

$$\mathbf{J} = \mathbf{P}\frac{\partial \mathbf{u}}{\partial \mathbf{m}}.$$

The matrix gives an idea about the components in the model that, when perturbed, yield (at least locally) the largest changes in the data.

An important tool that allows us to analyze the sensitivities is the singular value decomposition (SVD). We assume that the number of data is N, which is smaller than the number of parameters M. The SVD of an $N \times M$ matrix $\mathbf{J}$ is a decomposition

$$\mathbf{J} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\top = \sum_{i=1}^{N} \lambda_i \mathbf{u}_i \mathbf{v}_i^\top.$$

Here $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_N]$ is an orthogonal $N \times N$ left singular vectors matrix ($\mathbf{U}\mathbf{U}^\top = \mathbf{U}^\top\mathbf{U} = \mathbf{I}_N$), $\mathbf{\Lambda} = \mathsf{diag}(\lambda_1, \ldots, \lambda_N)$ is an $N \times N$ diagonal matrix with the singular values $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$, and $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_N]$ is an $M \times N$ orthogonal matrix ($\mathbf{V}^\top\mathbf{V} = \mathbf{I}_N$) of the right singular values. Note that $\mathbf{V}$ is not a full basis, and therefore $\mathbf{V}\mathbf{V}^\top \neq \mathbf{I}$.

Assume now a perturbation in the model $\mathbf{w}$. Then we can decompose $\mathbf{w}$ into components

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{v}_i + \mathbf{w}^{\text{orth}} = \mathbf{V}\boldsymbol{\alpha} + \mathbf{w}^{\text{orth}}.$$

The component $\mathbf{w}^{\text{orth}}$ contains any components that are not spanned by $\mathbf{V}$, which implies that

$$\mathbf{J}\mathbf{w}^{\text{orth}} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\top\mathbf{w}^{\text{orth}} = 0.$$

We therefore see that we can change $\mathbf{m}$ in the direction $\mathbf{w}^{\text{orth}}$ without any change in the data; that is, the data are insensitive to changes in this direction.

Now consider the singular vectors that correspond to the large singular values. A small change in these vectors leads to a large change in the data; that is, the problem is sensitive to changes in these directions. While the sensitivity matrix is important for the inverse problem, it is also important for a better understanding of the forward problem. In many cases, one can learn about the important components in the model by looking at the sensitivities. We will demonstrate this in an example later in this chapter.

5.3 ▪ Computation of the sensitivities

Computing the sensitivities is rather straightforward. The forward problem reads

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

Differentiating the forward problem with respect to $\mathbf{m}$, we obtain

$$\nabla_{\mathbf{m}} C(\mathbf{u},\mathbf{m})\delta\mathbf{m} + \nabla_{\mathbf{u}} C(\mathbf{m},\mathbf{u})\delta\mathbf{u} = 0.$$

Note that $\nabla_{\mathbf{m}} C(\mathbf{u},\mathbf{m})$ and $\nabla_{\mathbf{u}} C(\mathbf{m},\mathbf{u})$ are matrices. Furthermore, assuming that the forward problem is solved, the matrix $\nabla_{\mathbf{u}} C(\mathbf{m},\mathbf{u})$ is the forward problem Jacobian, and assuming that the forward is well posed, it is invertible. Manipulating this expression, we obtain

$$\delta\mathbf{u} = -(\nabla_{\mathbf{u}} C(\mathbf{m},\mathbf{u}))^{-1}\nabla_{\mathbf{m}} C(\mathbf{u},\mathbf{m})\delta\mathbf{m},$$

and therefore we establish the formula for the sensitivities,

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

This is the fundamental sensitivity equation.

The computation of the sensitivities requires us to compute two matrices $\nabla_{\mathbf{u}} C(\mathbf{m},\mathbf{u})$ and $\nabla_{\mathbf{m}} C(\mathbf{u},\mathbf{m})$. We now discuss a number of problems and demonstrate how to compute these matrices in practice. The reader is advised to first read section 5.6 about differentiating linear algebra expressions and to brush up on linear algebra and multivariable calculus.

5.3.1 ▪ Computation of stationary linear forward problem

Consider the forward problem of the form

$$C(\mathbf{u},\mathbf{m}) = \mathbf{A}(\mathbf{m})\mathbf{u} - \mathbf{q}.$$

This corresponds to virtually all the stationary problems discussed in this book.

Differentiating with respect to $\mathbf{u}$, we obtain

$$\nabla_{\mathbf{u}}(\mathbf{A}(\mathbf{m})\mathbf{u} - \mathbf{q}) = \mathbf{A}(\mathbf{m}).$$

Differentiating with respect to $\mathbf{m}$, we obtain

$$\nabla_{\mathbf{m}}(\mathbf{A}(\mathbf{m})\mathbf{u} - \mathbf{q}) = \nabla_{\mathbf{m}}(\mathbf{A}(\mathbf{m})\mathbf{u}).$$

The computation of this matrix is problem dependent. Consider, for example, the case in which the forward problem is the DCR,

$$\mathbf{A}(\mathbf{m})\mathbf{u} - \mathbf{q} = \mathsf{GRAD}^{\top}\mathbf{M}(\mathbf{m})\mathsf{GRAD}\mathbf{u} - \mathbf{q} = 0,$$

where

$$\mathbf{M}(\mathbf{m}) = \mathsf{diag}(\mathbf{A}_v^\top(\mathbf{v} \odot \mathbf{m})),$$

where $\mathbf{A}_v$ is an average matrix and $\mathbf{v}$ is the cell volume. Manipulating, we obtain that

$$\begin{aligned}&\mathsf{GRAD}^\top \mathbf{M}(\mathbf{m})\mathsf{GRAD}\mathbf{u} = \mathsf{GRAD}^\top \mathsf{diag}(\mathbf{A}_v^\top(\mathbf{m} \odot \mathbf{v}))\mathsf{GRAD}\mathbf{u}\\ &= \mathsf{GRAD}^\top \mathsf{diag}(\mathsf{GRAD}\mathbf{u})\mathbf{A}_v^\top \mathsf{diag}(\mathbf{v})\mathbf{m},\end{aligned}$$

and therefore, in this case,

$$\nabla_{\mathbf{m}} C(\mathbf{u}, \mathbf{m}) = \mathsf{GRAD}^\top \mathsf{diag}(\mathsf{GRAD}\mathbf{u})\mathbf{A}_v^\top \mathsf{diag}(\mathbf{v}).$$

Using the expressions above, we obtain that the sensitivities are

$$\mathbf{J} = -\mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathsf{GRAD}^\top \mathsf{diag}(\mathsf{GRAD}\mathbf{u})\mathbf{A}_v^\top \mathsf{diag}(\mathbf{v}).$$

Similar calculation can be done for Maxwell's equations, where $\mathbf{m}$ plays the role of conductivity. We have that

$$C(\mathbf{u}, \mathbf{m}) = (\mathsf{CURL}^\top \mathbf{M}_{f\mu}\mathsf{CURL} + i\omega \mathbf{M}_{e\mathbf{m}})\mathbf{u} + i\omega \mathbf{q} = 0,$$

where

$$\mathbf{M}_{e\mathbf{m}} = \mathsf{diag}(\mathbf{A}_v^\top(\mathbf{m} \odot \mathbf{v})).$$

In this case, we obtain that

$$\nabla_{\mathbf{m}} C(\mathbf{u}, \mathbf{m}) = -\mathbf{P}(\mathsf{CURL}^\top \mathbf{M}_{f\mu}\mathsf{CURL} + i\omega \mathbf{M}_{e\mathbf{m}})^{-1}\mathsf{diag}(\mathbf{u})\mathbf{A}_v^\top \mathsf{diag}(\mathbf{v}).$$

Although the computation of each problem is slightly different, it is evident that in general, the calculation leads to the following formula:

$$\mathbf{J} = -\mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{G}(\mathbf{u}, \mathbf{m}), \tag{5.1}$$

where

$$\mathbf{G}(\mathbf{u}, \mathbf{m}) = \nabla_{\mathbf{m}}(\mathbf{A}(\mathbf{m})\mathbf{u}).$$

Thus to compute the sensitivities, one must compute the inverse of the forward problem matrix times either $\mathbf{G}$ or $\mathbf{P}^\top$. This may make the computation of the sensitivities difficult if not impossible for most problems that involve many data or a reasonable number of model parameters. Nonetheless, the formula (5.1) is highly useful, and we now show how to use it in order to work with the sensitivities in practice.

5.3.2 ▪ Computation for time dependent problems

Perhaps the most involved computation of sensitivities arises for time dependent problems. We now discuss the computation of the time domain derivatives for Maxwell's equations (4.11a) in their discrete form. We rewrite the system (4.11a) (assuming no source) as

$$\begin{aligned}&(\mathsf{CURL}^\top \mathbf{M}_{f\mu}\mathsf{CURL} + \Delta t^{-1}\mathbf{M}_{e\mathbf{m}})\mathbf{e}^{n+1} - \Delta t^{-1}\mathbf{M}_{e\sigma}\mathbf{e}^n = 0, \qquad (5.2)\\ &\mathbf{M}_{e\mathbf{m}}\mathbf{e}^0 = \mathsf{CURL}^\top \mathbf{b}^0,\end{aligned}$$

where $\mathbf{m}$ is the conductivity. Note that for the first time step, $\mathbf{e}^0$ is not used; we use only $\mathbf{M}_{e\mathbf{m}}\mathbf{e}^0$, which is *independent* of $\mathbf{m}$. Recall also that

$$\mathbf{M}_{e\mathbf{m}} = \mathsf{diag}(\mathbf{A}_v^\top(\mathbf{v}\odot\mathbf{m})).$$

The system (5.2) for all times can be written as

$$C(\mathbf{e},\mathbf{m}) = \begin{pmatrix} \mathbf{A}(\mathbf{m}) & & & \\ \mathbf{B}(\mathbf{m}) & \mathbf{A}(\mathbf{m}) & & \\ & \ddots & \ddots & \\ & & \mathbf{B}(\mathbf{m}) & \mathbf{A}(\mathbf{m}) \end{pmatrix} \begin{pmatrix} \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_n \end{pmatrix} - \begin{pmatrix} \mathsf{CURL}^\top \mathbf{b}^0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 0,$$

where $\mathbf{A}(\mathbf{m}) = \mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} + \Delta t^{-1}\mathbf{M}_{e\mathbf{m}}$ and $\mathbf{B}(\mathbf{m}) = \Delta t^{-1}\mathbf{M}_{e\sigma}$.

Consider now the vector $\mathbf{e} = (\mathbf{e}_1^\top, \ldots, \mathbf{e}_n^\top)^\top$. Then we have that

$$\nabla_{\mathbf{e}} C = \begin{pmatrix} \mathbf{A}(\mathbf{m}) & & & \\ \mathbf{B}(\mathbf{m}) & \mathbf{A}(\mathbf{m}) & & \\ & \ddots & \ddots & \\ & & \mathbf{B}(\mathbf{m}) & \mathbf{A}(\mathbf{m}) \end{pmatrix}.$$

Since $\mathbf{m}$ does not change in time, and the matrix $\mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL}$ is independent of $\mathbf{m}$, we rewrite the derivative of the forward problem as

$$\begin{aligned} \nabla_m C(\mathbf{e},\mathbf{m}) &= \nabla_{\mathbf{m}} \begin{pmatrix} \Delta t^{-1}\mathbf{M}(\mathbf{m})\mathbf{e}^1 \\ \Delta t^{-1}\mathbf{M}(\mathbf{m})(\mathbf{e}^2 - \mathbf{e}^1) \\ \vdots \\ \Delta t^{-1}\mathbf{M}(\mathbf{m})(\mathbf{e}^n - \mathbf{e}^{n-1}) \end{pmatrix} \\ &= \frac{1}{\Delta t}\nabla_{\mathbf{m}} \left(\begin{pmatrix} \mathsf{diag}(\mathbf{e}^1) \\ \mathsf{diag}(\mathbf{e}^2 - \mathbf{e}^1) \\ \vdots \\ \mathsf{diag}(\mathbf{e}^n - \mathbf{e}^{n-1}) \end{pmatrix} \mathbf{A}_v^\top \mathsf{diag}(\mathbf{v})\mathbf{m} \right) \\ &= \frac{1}{\Delta t} \begin{pmatrix} \mathsf{diag}(\mathbf{e}^1) \\ \mathsf{diag}(\mathbf{e}^2 - \mathbf{e}^1) \\ \vdots \\ \mathsf{diag}(\mathbf{e}^n - \mathbf{e}^{n-1}) \end{pmatrix} \mathbf{A}_v^\top \mathsf{diag}(\mathbf{v}), \end{aligned}$$

and the sensitivity is

$$\mathbf{J} = -\frac{1}{\Delta t} \begin{pmatrix} \mathbf{A}(\mathbf{m}) & & & \\ \mathbf{B}(\mathbf{m}) & \mathbf{A}(\mathbf{m}) & & \\ & \ddots & \ddots & \\ & & \mathbf{B}(\mathbf{m}) & \mathbf{A}(\mathbf{m}) \end{pmatrix}^{-1} \begin{pmatrix} \mathsf{diag}(\mathbf{e}^1) \\ \mathsf{diag}(\mathbf{e}^2 - \mathbf{e}^1) \\ \vdots \\ \mathsf{diag}(\mathbf{e}^n - \mathbf{e}^{n-1}) \end{pmatrix} \mathbf{A}_v^\top \mathsf{diag}(\mathbf{v}). \tag{5.3}$$

Although the computation is straightforward, there are some practical difficulties when computing time dependent sensitivities. In particular, note that we need to store the vector $\mathbf{e}$. For 3D time dependent problems this may become very expensive and may require a very large memory. One option to overcome this problem is to use checkpointing. The idea is to store only a small number of $\mathbf{e}_j$'s for a number of times and then to compute the rest on the fly. This trades the lack of memory for additional computations.

5.4 ▪ Sensitivities of complex variables

Frequency domain electromagnetic problems result in complex fields. Furthermore, it is possible to consider complex conductivity functions. Therefore, it is important to compute the sensitivities of the complex field with respect to the possibly complex conductivity.

Since derivatives with respect to complex variables can be ambiguous, we do not work with complex quantities but rather convert the system to real arithmetic. To be more specific, we consider the frequency domain discretization

$$(\mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} - i\omega\,\mathbf{M}_{e\mathbf{m}})\mathbf{e} = -i\omega\mathbf{s}, \tag{5.4}$$

where, as before, we have that $\mathbf{m}$ is the conductivity, and

$$\mathbf{M}_{e\mathbf{m}} = \mathsf{diag}(\mathbf{A}_v^\top(\mathbf{v}\odot\mathbf{m})).$$

Separating the system into real and imaginary parts,

$$\mathbf{e} = \mathbf{e}_r + i\mathbf{e}_i,$$

we can write

$$C(\mathbf{e}_r,\mathbf{e}_i,\mathbf{m}) = \begin{pmatrix} \mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} & \omega\,\mathbf{M}_{e\mathbf{m}} \\ -\omega\,\mathbf{M}_{e\mathbf{m}} & \mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} \end{pmatrix}\begin{pmatrix}\mathbf{e}_r\\ \mathbf{e}_i\end{pmatrix} = \begin{pmatrix}0\\ -i\omega\mathbf{s}\end{pmatrix}.$$

This is a real system, and therefore, we can differentiate it with respect to the conductivity, assuming that the conductivity is real.

$$\nabla_{\mathbf{m}} C(\mathbf{e}_r,\mathbf{e}_i,\mathbf{m}) = \omega\begin{pmatrix}\nabla_{\mathbf{m}}(\mathbf{M}_{e\mathbf{m}}\mathbf{e}_i)\\ -\nabla_{\mathbf{m}}(\mathbf{M}_{e\mathbf{m}}\mathbf{e}_r)\end{pmatrix} = \omega\begin{pmatrix}\mathsf{diag}(\mathbf{e}_i)\mathbf{A}_v^\top\mathbf{v}\\ -\mathsf{diag}(\mathbf{e}_r)\mathbf{A}_v^\top\mathbf{v}\end{pmatrix}. \tag{5.5}$$

Using the above, the sensitivity can be written as

$$\mathbf{J} = -\omega\begin{pmatrix} \mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} & \omega\,\mathbf{M}_{e\mathbf{m}} \\ -\omega\,\mathbf{M}_{e\mathbf{m}} & \mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} \end{pmatrix}^{-1}\begin{pmatrix}\mathsf{diag}(\mathbf{e}_i)\mathbf{A}_v^\top\mathbf{v}\\ -\mathsf{diag}(\mathbf{e}_r)\mathbf{A}_v^\top\mathbf{v}\end{pmatrix}.$$

The sensitivity maps the real conductivity vector $\mathbf{m}$ into the real and imaginary fields. With a bit of manipulation, this can be written as

$$\mathbf{J}_c = i\omega(\mathsf{CURL}^\top \mathbf{M}_{f\mu}\,\mathsf{CURL} - i\omega\,\mathbf{M}_{e\mathbf{m}})^{-1}\,\mathsf{diag}(\mathbf{e})\mathbf{A}_v^\top\mathbf{v},$$

where $\mathbf{J}_c$ is a complex sensitivity function.

5.5 ▪ Working with sensitivities in practice

The sensitivity matrix is large and composed of three matrices. The matrices $\mathbf{P}, \nabla_{\mathbf{m}} C$, and $\mathbf{A}(\mathbf{m})$ are sparse; however, the matrix $\mathbf{A}(\mathbf{m})^{-1}$ is almost always dense. Therefore, the sensitivity matrix is typically dense and very large. Computing the matrix in practice is impossible for most large scale problems. Nonetheless, *the actual computation of the sensitivity is not needed in most practical cases.* What are needed are matrix-vector products of the form $\mathbf{J}\mathbf{v}$ and $\mathbf{J}^\top\mathbf{w}$. For that, note that one can compute the product in three steps. To evaluate $\mathbf{J}\mathbf{v}$, we first multiply the vector $\mathbf{v}$ by $\nabla_{\mathbf{m}} C$. In the second step, we solve the linear system $\mathbf{A}(\mathbf{m})\mathbf{y} = (\nabla_{\mathbf{m}} C)\mathbf{v}$. In the last step, we set $\mathbf{J}\mathbf{v}$ to be the product $-\mathbf{P}\mathbf{y}$.

For the transpose, we start by computing $\mathbf{P}^\top \mathbf{w}$. We then solve the system $\mathbf{A}(\mathbf{m})^\top \mathbf{y} = \mathbf{P}^\top \mathbf{w}$. Finally, we set the product $\mathbf{J}^\top \mathbf{w}$ to $-(\nabla_{\mathbf{m}} C)^\top \mathbf{y}$.

Note that the calculation of the sensitivity matrix-vector product and its adjoint require solving a system equivalent to a linearized forward or transposed problem. This is not a trivial task. If the calculation is required many times, then investing in good preconditioners can be crucial. Assume for a moment that we are able to compute the LU factorization of the forward problem. In this case, the cost of each matrix-vector product is the cost of forward-backward substitution. If a single product is needed, then the cost of the factorization dominates the computation. However, if the number of sensitivity matrix-vector products is large, one may want to reconsider and compute either an exact or an approximate factorization if possible. We will discuss this detail in the next chapters.

Although computing sensitivities is almost never done in practice, there are some rare cases where, for one reason or another, the sensitivity matrix is desired. This is typically the case where either the size of the model is or the data are very small. It is important to remember that there are two ways to compute the matrix. Since $\mathbf{J} = -\mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\nabla_{\mathbf{m}} C$, one can compute it as

$$\mathbf{J} = -\mathbf{P}(\mathbf{A}(\mathbf{m})^{-1}\nabla_{\mathbf{m}} C).$$

That is, compute the matrix $\mathbf{A}(\mathbf{m})^{-1}\nabla_{\mathbf{m}} C$, and then multiply by $\mathbf{P}$. The second option is to compute

$$\mathbf{J} = -((\nabla_{\mathbf{m}} C)^\top (\mathbf{A}(\mathbf{m})^{-\top}\mathbf{P}^\top))^\top.$$

That is, compute the product $\mathbf{A}^{-\top}\mathbf{P}^\top$ first, multiply by $\nabla_{\mathbf{m}} C^\top$, and then transpose the result. The first approach is sometimes referred to as the forward calculation, and the second approach is referred to as the adjoint calculation. If the number of parameters $\mathbf{m}$ is very small, then the forward calculation is obviously preferable, and, on the other hand, if the number of data is small, then the adjoint evaluation is better.

When considering the linear algebra, it is straightforward to see that the adjoint ($\mathbf{A}^{-\top}$) comes naturally and is a simple consequence of the calculations. In the next chapters, we will see how the adjoint is used for the evaluation of the gradient of some common functions of the model.

Another important observation is to be made here. Note that the above derivation of the sensitivity also fits time dependent problems where the matrix $\mathbf{A}(\mathbf{m})$ is the space-time matrix in (5.3). When computing $\mathbf{J}\mathbf{v}$, we need to solve the space-time system, and this is done by forward stepping in time. For the computation of $\mathbf{J}^\top \mathbf{w}$, we need to solve the adjoint system. In this case, the adjoint system reads

$$\begin{pmatrix} \mathbf{A}(\mathbf{m})^\top & \mathbf{B}(\mathbf{m})^\top & & \\ & \mathbf{A}(\mathbf{m})^\top & \ddots & \\ & & \ddots & \mathbf{B}(\mathbf{m})^\top \\ & & & \mathbf{A}(\mathbf{m})^\top \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \vdots \\ \mathbf{y}_n \end{pmatrix} = \text{rhs}.$$

This system is solved *backward*; that is, we start from the last block and go back. This process is equivalent to a backward in time differential equation.

5.6 ▪ Differentiating linear algebra expressions

The computation of derivatives involves the differentiation of expressions that involve matrices and vectors. It is useful to be able to differentiate such expressions quickly. Although one can try to do it using indices, we find that it is easier to differentiate math-

ematical expressions in matrix form. To this end, we discuss the differentiation of some common linear algebra expressions and use them for the computation of the sensitivities.

We start with the linear multivariable function

$$f(\mathbf{x}) = \mathbf{y}^\top \mathbf{x} = \mathbf{x}^\top \mathbf{y} = \sum_i \mathbf{x}_i \mathbf{y}_i.$$

Differentiating with respect to $\mathbf{x}$, we obtain the vector

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial \mathbf{x}_1} & \cdots & \frac{\partial f}{\partial \mathbf{x}_M} \end{pmatrix} = \begin{pmatrix} \mathbf{y}_1 & \cdots & \mathbf{y}_M \end{pmatrix} = \mathbf{y}^\top.$$

Note that the derivative of a function $f(\mathbf{x})$ should be written as a *row vector* if we assume that $\mathbf{x}$ is a column vector. However, the convention is to store *the gradient of a scalar function as a column vector*. This convention is rather confusing and therefore noteworthy. This implies that we write $\nabla(\mathbf{x}^\top \mathbf{y}) = \mathbf{y}$ and not $\mathbf{y}^\top$. Bear in mind that the transpose is being used only for differentiating scalar functions!

We now use the result to differentiate the multivariable function

$$\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} = \begin{pmatrix} \mathbf{a}_1^\top \mathbf{x} \\ \vdots \\ \mathbf{a}_N^\top \mathbf{x} \end{pmatrix},$$

where $\mathbf{a}_i^\top = \mathbf{A}_{i,:}$. Note that in this case f is a vector function $\mathbf{f} = \begin{pmatrix} f_1 & \cdots & f_M \end{pmatrix}$. Each entry in the vector $\mathbf{f}$ is similar to that we just differentiated; thus

$$\nabla \mathbf{f} = \begin{pmatrix} \nabla f_1 & \cdots & \nabla f_M \end{pmatrix} = \begin{pmatrix} \mathbf{a}_1^\top \\ \vdots \\ \mathbf{a}_M^\top \end{pmatrix} = \mathbf{A}.$$

Note that we did not transpose each row in the matrix $\mathbf{A}$. Note also that if $\mathbf{A}$ were a $1 \times M$ matrix, then we would have transposed the result. Again, this is just a convention that can be confusing at times.

Using this result, we can now differentiate a quadratic form. Let

$$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}\mathbf{x}.$$

To differentiate, we use the product rule

$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^\top \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} = \left(\mathbf{x}^\top \frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}}\right)^\top + \frac{\partial \mathbf{x}^\top \mathbf{A}}{\partial \mathbf{x}} \mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{A}^\top \mathbf{x}.$$

If $\mathbf{A}$ is symmetric, then we obtain $2\mathbf{A}\mathbf{x}$.

We now look at another type of derivative generated by the Hadamard product. The Hadamard product of two vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$f = \mathbf{x} \odot \mathbf{y} = \begin{pmatrix} \mathbf{x}_1 \mathbf{y}_1 \\ \vdots \\ \mathbf{x}_M \mathbf{y}_M \end{pmatrix}.$$

It is easy to verify that

$$\mathbf{f} = \operatorname{diag}(\mathbf{y})\mathbf{x},$$

where diag($\mathbf{y}$) is a diagonal matrix with $\mathbf{y}$ on its main diagonal. Therefore,

$$\nabla \mathbf{f} = \mathrm{diag}(\mathbf{y}).$$

Using the above, we can now differentiate more complicated expressions. Consider the expression

$$\mathbf{f}(\mathbf{x}) = (\mathbf{A}\mathbf{x})^2 = (\mathbf{A}\mathbf{x}) \odot (\mathbf{A}\mathbf{x}).$$

Differentiating with respect to $\mathbf{x}$, we obtain

$$\nabla \mathbf{f} = \nabla((\mathbf{A}\mathbf{x}) \odot (\mathbf{A}\mathbf{x})) = \nabla(\mathrm{diag}(\mathbf{A}\mathbf{x})\mathbf{A}\mathbf{x}) = 2\,\mathrm{diag}(\mathbf{A}\mathbf{x})\mathbf{A}.$$

Let us make this slightly more complicated. Consider now the expression

$$f(\mathbf{x}) = \mathbf{v}^\top \sqrt{(\mathbf{A}\mathbf{x})^2 + 1}.$$

Similar expressions arise in the minimal surface problem. Since $\mathbf{v}$ is independent of $\mathbf{x}$, to differentiate we need to differentiate $\sqrt{(\mathbf{A}\mathbf{x})^2 + 1}$. Using the chain rule, we have

$$\nabla(\sqrt{(\mathbf{A}\mathbf{x})^2 + 1}) = \mathrm{diag}\left(\frac{1}{2\sqrt{(\mathbf{A}\mathbf{x})^2 + 1}}\right)\nabla((\mathbf{A}\mathbf{x})^2 + 1) = \mathrm{diag}\left(\frac{1}{\sqrt{(\mathbf{A}\mathbf{x})^2 + 1}}\right)\mathrm{diag}(\mathbf{A}\mathbf{x})\mathbf{A}.$$

Putting this together, we have

$$\mathbf{v}^\top \mathrm{diag}\left(\frac{1}{\sqrt{(\mathbf{A}\mathbf{x})^2 + 1}}\right)\mathrm{diag}(\mathbf{A}\mathbf{x})\mathbf{A};$$

note that this expression is a row vector. If we use column vectors, we have

$$\nabla f = \left(\mathbf{v}^\top \mathrm{diag}\left(\frac{1}{\sqrt{(\mathbf{A}\mathbf{x})^2 + 1}}\right)\mathrm{diag}(\mathbf{A}\mathbf{x})\mathbf{A}\right)^\top = \mathbf{A}^\top \mathrm{diag}\left(\frac{\mathbf{v}}{\sqrt{(\mathbf{A}\mathbf{x})^2 + 1}}\right)\mathbf{A}\mathbf{x}.$$

5.7 ▪ Programmer note

When programming the sensitivities, there are a number of pitfalls which, unfortunately, tend to be detrimental to making the code work. We now discuss two important tests to be done when programming the sensitivities. The first is the gradient test, and the second is the adjoint test.

5.7.1 ▪ Derivative test

The calculation of the sensitivities (or the sensitivities' matrix-vectors) is *crucial* for the solution of the inverse problem. It is therefore very important to verify that the correct sensitivity is computed. This verification can be obtained by the derivative test we now present. Assume that we have $C(\mathbf{u}, \mathbf{m}) = 0$ and that we have computed the Jacobians $\nabla_{\mathbf{u}} C$ and $\nabla_{\mathbf{m}} C$. Then, to test the Jacobians, we take a random vector $\mathbf{v}$ and generate the table of $||C(\mathbf{u} + h\mathbf{v}, \mathbf{m}) - C(\mathbf{u}, \mathbf{v})||$ versus h, where h is decreasing logarithmically, and a table of $||C(\mathbf{u} + h\mathbf{v}, \mathbf{m}) - C(\mathbf{u}, \mathbf{m}) - h\nabla_{\mathbf{u}} C\mathbf{v}||$ versus h. By Taylor's theorem the first difference converges to 0 linearly, while the second difference converges to 0 quadratically. If you do not get quadratic convergence, you have the wrong Jacobian! A similar calculation should be done for the Jacobian with respect to $\mathbf{m}$, $\nabla_{\mathbf{m}} C$.

Table 5.1. *The derivative test for the problem* (5.6).

1.00e-01	7.58e+02	7.01e+01
1.00e-02	7.61e+01	7.04e-01
1.00e-03	7.62e+00	7.04e-03
1.00e-04	7.62e-01	7.04e-05
1.00e-05	7.62e-02	7.04e-07
1.00e-06	7.62e-03	7.04e-09
1.00e-07	7.62e-04	7.05e-11
1.00e-08	7.62e-05	8.49e-12
1.00e-09	7.62e-06	9.54e-12
1.00e-10	7.62e-07	8.87e-12

To demonstrate, we consider the forward problem of the form

$$C(\mathbf{u},\mathbf{m}) = \mathbf{A}(\mathbf{m})\mathbf{u} - \mathbf{q} = \mathbf{D}^\top \operatorname{diag}(\mathbf{S}\exp(\mathbf{m}))\mathbf{D}\mathbf{u} - \mathbf{q}. \tag{5.6}$$

Computing the derivative with respect to $\mathbf{m}$, we obtain that

$$\nabla_{\mathbf{m}} C = \mathbf{D}^\top \operatorname{diag}(\mathbf{D}\mathbf{u})\mathbf{S}\operatorname{diag}(\exp(\mathbf{m})).$$

The following code tests the derivative:

```
% define function and derivatives
clear all; close all
n1 = 4; n2 = 5; n3 = 6;
D = getNodalGradientMatrix(n1,n2,n3);
S = getEdgeToCellCenterMatrix(n1,n2,n3);
q = randn((n1+1)*(n2+1)*(n3+1),1);

C       = @(u,m)(D'*sdiag(S'*exp(m))*D*u - q);
dCdm    = @(u,m)(D'*sdiag(D*u)*S'*sdiag(exp(m)));
dCdu    =  @(u,m)(D'*sdiag(S'*exp(m))*D);

% now test the derivatives
u = randn(size(D,2),1); m = randn(size(S,1),1);
v = randn(size(m));
f = C(u,m);
G = dCdm(u,m);
for i=1:10
    h = 10^(-i);
        fp = C(u,m+h*v);
        diff1 = norm(fp-f);
        diff2 = norm(fp-f - h*G*v);
        fprintf('%3.2e    %3.2e    %3.2e\n',h,diff1,diff2)
end
```

Table 5.1 was obtained using the above code (note that we use the randn function, so results may be slightly different for the reader).

Obviously, as long as h is not too small, where machine precision takes over, we see that reducing h by a factor of 10 reduces the linear approximation by a factor of 100.

Although the derivative test is meant to test the derivatives, it can also be used to learn something about the function at hand. For example, consider the nonlinear PDE

$$\nabla \cdot \rho(\mathbf{u})\nabla\mathbf{u} = \mathbf{m}$$

with $\rho(\mathbf{u}) = \frac{1}{\sqrt{|\mathbf{D}\mathbf{u}|^2+\eta}}$.

Table 5.2. *The derivative test for the problem* (5.7).

1.00e-01	3.62e+01	3.62e+01
1.00e-02	1.58e+01	1.58e+01
1.00e-03	1.37e-04	8.51e-05
1.00e-04	5.55e-06	3.80e-07
1.00e-05	5.20e-07	3.59e-09
1.00e-06	5.17e-08	3.51e-11
1.00e-07	5.17e-09	1.57e-12
1.00e-08	5.17e-10	1.45e-12
1.00e-09	5.17e-11	1.27e-12
1.00e-10	5.17e-12	1.20e-12

In this case, a simple discretization reads

$$C(\mathbf{u},\mathbf{m}) = \mathbf{D}^\top \operatorname{diag}\left(\frac{1}{\sqrt{|\mathbf{Du}|^2+\eta}}\right)\mathbf{Du}-\mathbf{m}, \tag{5.7}$$

and the derivative with respect to $\mathbf{u}$ can be computed as

$$\nabla_{\mathbf{u}} C = \mathbf{D}^\top \operatorname{diag}\left(\frac{1}{\sqrt{|\mathbf{Du}|^2+\eta}}\right)\mathbf{D} - \mathbf{D}^\top \operatorname{diag}\left(\frac{(\mathbf{Du})^2}{(|\mathbf{Du}|^2+\eta)^{\frac{3}{2}}}\right)\mathbf{D}.$$

The following MATLAB script tests this approximation:

```
t = 1e-8;
n1 = 4; n2 = 5; n3 = 6;
D = getNodalGradientMatrix(n1,n2,n3);
% define function and derivatives
C   = @(u,m)(D'*sdiag(1./sqrt((D*u).^2 + t))*D*u - m);
dCdu = @(u,m)(D'*sdiag(1./sqrt((D*u).^2 + t))*D - ...
                    D'*sdiag((D*u).^2./((D*u).^2 + t).^(3/2))*D);

% now test the derivatives
u = randn(size(D,2),1); m = randn(size(D,2),1);
v = randn(size(u));
f = C(u,m);
A = dCdu(u,m);
for i=1:10
        h = 10^(-i);
        fp = C(u+h*v,m);
        diff1 = norm(fp-f);
        diff2 = norm(fp-f - h*A*v);
        fprintf('%3.2e    %3.2e    %3.2e\n',h,diff1,diff2)
end
```

The results are presented in Table 5.2.

Table 5.2 is rather different from Table 5.1. Note that for a step size of $h = 10^{-2}$, no marked difference is observed between the first and second order approximations. This implies that, at least for the direction chosen here, the problem is nonlinear and requires very small steps for the linear approximation to be a good representative of the nonlinear problem. Thus, using the derivative test, we can learn not only about the correctness of the derivative but also about the nonlinearity of the problem at hand.

5.7.2 ▪ The adjoint test

A second test that must be performed is the so-called adjoint test. To explain, we recall that typically, one does not compute the sensitivities but rather has a code to compute $\mathbf{Jv}$ and $\mathbf{Jw}$, where $\mathbf{v}$ and $\mathbf{w}$ are any appropriate size vectors. If for some reason we have a bug in one of these, then any code that is based on $\mathbf{J}^\top$ will fail.

One simple way to test the codes for $\mathbf{J}$ and $\mathbf{J}^\top$ is to use the identity

$$\mathbf{w}^\top(\mathbf{Jv}) = \mathbf{v}^\top(\mathbf{J}^\top\mathbf{w}).$$

Assuming a sensitivity matrix that maps from $\mathbb{R}^n \to \mathbb{R}^k$ and two codes JmatVec and JTmatVec that compute the forward and adjoint products, we can test them as follows:

```
v = randn(n,1);  w = randn(k,1);
d = abs(v'*JTmatVec(w) - w'*JmatVec(v))
```

In principle, d should be 0. However, in practice, it will be small, which is typically because of the tolerance used by the linear solve in the sensitivity calculation.

5.8 ▪ Example: Sensitivities for the time domain model problem

We now give an example of the computation and the properties of the sensitivities of a conductive uniform media to a half-space coded in section 4.7. The codes for the sensitivities of the data with respect to the conductivity and their adjoint are as follows:

```
function[Jv] = getJMatVec(z,e,m,param)
% [v] = getJMatVec(u,m,param)
%

% magnetic permeability
mu = 4*pi*1e-7 * ones(prod(param.nc),1);
% extract matrices
Curl = param.Curl; Grad = param.Grad;
Af   = param.Af; Ae   = param.Ae;
An   = param.An;
Iact = param.Iact;
V    = param.V;
dt   = param.dt;

nt = length(dt); ne = size(Curl,2);

% inactive cells
sigma = param.mref + Iact*m;

% set up mass matrices
Mmuinv  = sdiag(Af'*(V*(1./mu)));
Msig    = sdiag(Ae'*(V*sigma));
Mmuinvn = sdiag(An'*V*(1./mu));

e  = reshape(e,[],nt+1);
% Solve linear system
lam = e*0;
Jv  = zeros(size(param.obs,1),nt);
for i=1:length(dt)

    % The linear system to be solved
    Ke = Curl'*Mmuinv*Curl + 1/dt(i)*Msig;
    Gzi = (1/dt(i))*sdiag(e(:,i+1)-(i>1)*e(:,i))*Ae'*V*Iact*z;
    rhs = Gzi + 1/dt(i)*Msig*lam(:,i);
    lam(:,i+1) = solveSystem(Ke,Msig,Mmuinvn,Grad,dt(i),rhs);

    % compute Jz
    Jv(:,i)  = -param.obs*Curl*lam(:,i+1);
end
```

The adjoint calculation can be done in a similar way:

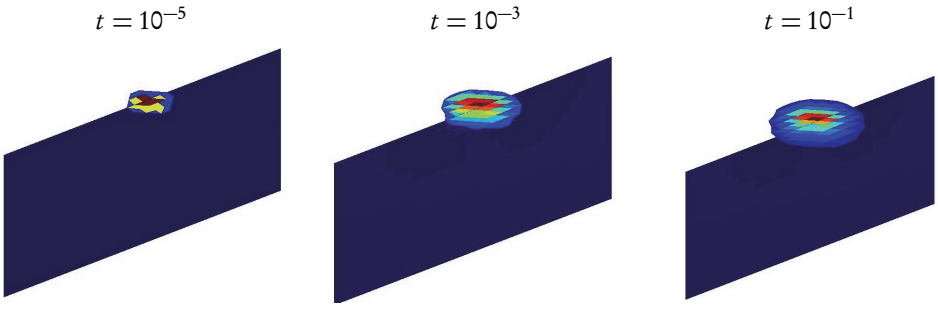

Figure 5.1. *The sensitivity of the data in the z-direction at times* $10^{-5}, 10^{-3}$, *and* 10^{-1} *seconds. Note the ring-like structure of the sensitivities.*

```
function[JTv] = getJTmatVec(z,e,m,param)
%[JTv] = getJTmatVec(z,e,m,param)
%

% magnetic permeability
mu = 4*pi*1e-7 * ones(prod(param.nc),1);
% extract matrices
Curl = param.Curl;   Grad = param.Grad;
Af   = param.Af; Ae   = param.Ae; An   = param.An;
Iact = param.Iact;
V    = param.V;
dt   = param.dt;

nt = length(dt); ne = size(Curl,2);

% inactive cells
sigma = param.mref + Iact*m;

% set up mass matrices
Mmuinv  = sdiag(Af'*(V*(1./mu)));
Msig    = sdiag(Ae'*(V*sigma));
Mmuinvn = sdiag(An'*V*(1./mu));

e   = reshape(e,[],nt+1);
lam = e*0;

s = Curl'*param.obs'*reshape(z,size(param.obs,1),[]);
JTv = 0;
for i=length(dt):-1:1
    Ke = Curl'*Mmuinv*Curl + 1/dt(i)*Msig;
    rhs = s(:,i) + 1/dt(i)*Msig*lam(:,i+1);
    lam(:,i) = solveSystem(Ke,Msig,Mmuinvn,Grad,dt(i),rhs);
    Gzi = (1/dt(i))*sdiag(e(:,i+1)-(i>1)*e(:,i))*Ae'*V*Iact;
    JTv = JTv - Gzi'*lam(:,i);
end
```

To understand the sensitivity of the problem, we use the code LoopEMsetup, compute the columns of $\mathbf{J}^\top$ that correspond to a magnetic data in the z-direction, and plot them in Figure 5.1.

The sensitivity has a distinct shape of a bowl with inner and outer nonzero values. Therefore, geological structures that have similar shapes are in the active space of the sensitivity, and it is easy to generate such shapes. One has to be aware that "ring-like" structures can be artifacts since they are very similar to the sensitivities.

5.9 ▪ Exercises for sensitivity calculation

1. The DCR problem is

$$\nabla \cdot \sigma \nabla u = q, \qquad \sigma > 0,$$

with some boundary conditions. Assume that a nodal discretization is used to obtain the discrete system of the form

$$\mathsf{GRAD}\left(\mathrm{diag}(\mathbf{A}_v^\top \boldsymbol{\sigma})\right)\mathsf{GRAD}\mathbf{u} = q.$$

(a) Program the forward problem on an $n_1 \times n_2$ tensor grid.

(b) Compute the derivatives with respect to $\mathbf{u}$ and $\boldsymbol{\sigma}$.

(c) Program the derivatives, and validate them using the derivative test.

(d) For a problem of size $32 \times 32 \times 16$ and $\sigma = 1$ compute an approximation to the sensitivity matrix assuming $\mathbf{Q} = \mathbf{I}$.

(e) Using the Lanczos method, compute the first 30 singular vectors of the sensitivity matrix. Plot the right singular vectors that correspond to $\lambda_1, \ldots, \lambda_{10}$, and view the singular vectors that correspond to smaller singular values. Make qualitative comments about the behavior of the singular vectors.

(f) In a semilog curve, plot the singular values of the problem. Can you comment on the effective rank of the sensitivities?

2. Repeat the calculations for the frequency domain electromagnetic data.

Chapter 6
Data Fitting, Regularization, and Inversion

In this chapter we present the components of the inverse problem. We assume to have some finite amount of measured *data*, $\mathbf{d}$, and that we have some forward model that yields a synthetic version of the data, which, for most geophysical problems can be written as

$$\mathbf{d} = \mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q} + \boldsymbol{\epsilon}. \tag{6.1}$$

We have discussed examples for $\mathbf{A}(\mathbf{m}), \mathbf{q}$, and $\mathbf{P}$ in the previous chapters. $\boldsymbol{\epsilon}$ is noise with some assumed statistics. The inverse problem can be defined as an attempt to *estimate* the model $\mathbf{m}$ given the observed data, $\mathbf{d}$, assuming the forward problem (6.1) and some noise statistics.

Given the fact that the data are noisy and finite, it is rather clear that there cannot be a unique solution to the inverse problem. Indeed, many estimators to the true model exist. In this chapter we discuss some of the more common estimators and explore some of their properties.

The treatment we give here is frequentist; that is, we do not assume that the a priori knowledge about the model is given in terms of a probability density function. We refer the reader to [50] for a Bayesian point of view and to [67] for a tutorial about the difference between the two.

6.1 ▪ The misfit

Consider the observed data $\mathbf{d}$ and the noise $\boldsymbol{\epsilon}$. Assuming that we have the "right" model $\mathbf{m}$ and that the forward modeling equation is correct, then

$$\boldsymbol{\epsilon} = \mathbf{d} - \mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q}.$$

One common way to model the noise is by assuming it is random. To this end we need to assume the distribution of the noise, $\pi(\boldsymbol{\epsilon})$. The simplest assumption is that $\boldsymbol{\epsilon}$ is Gaussian, with 0 mean, that is,

$$\pi(\boldsymbol{\epsilon}) \propto \exp\left(-\frac{1}{2}\boldsymbol{\epsilon}^\top \Sigma_d^{-1}\boldsymbol{\epsilon}\right),$$

where Σ_d is the covariance matrix of the measured data. Substituting, we obtain

$$\text{Likelihood}(\mathbf{m}) \propto \exp\left(-\frac{1}{2}(\mathbf{d} - \mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q})^\top \Sigma_d^{-1}(\mathbf{d} - \mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q})\right). \tag{6.2}$$

The expression above is proportional to the so-called *likelihood*. Assume a "candidate" model $\mathbf{m}^*$, and substitute $\mathbf{m}^*$ into the likelihood above. If we obtain a very small number, then the likelihood that the "true" noise lies in a small ball around the noise predicted by $\mathbf{m}^*$ is rather small. If, on the other hand, we obtain a large number, then the likelihood that the "true" noise is close to the one predicted by $\mathbf{m}^*$ is larger. Clearly, the largest likelihood is obtained by maximizing (6.2) or minimizing its negative log, and this leads to the definition of the misfit

$$\phi_d(\mathbf{m}) = \frac{1}{2}(\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q})^\top \Sigma_d^{-1}(\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q}). \tag{6.3}$$

Before we discuss the use of the misfit, in order to evaluate $\mathbf{m}$ we should acknowledge that it stems from our assumptions on the noise. If the noise is not Gaussian, then different misfit functions can be used. A few of the most common non-Gaussian distributions are obtained by assuming a "heavy tail" assumption. Heavy tail distributions are distributions that decay to zero slowly (more slowly than Gaussians), and this implies that it is possible to have large outliers in the data. For example, a very common distribution is

$$\text{Likelihood}(\mathbf{m}) \propto \exp\left(-\phi_d(\mathbf{w} \odot (\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q}))\right), \tag{6.4}$$

where $\mathbf{w}$ are (nonnegative) weights and

$$\phi_d(t) = \sum_i |t|_i.$$

This leads to the misfit function

$$\phi_d(\mathbf{m}) = \mathbf{w}^\top |\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q}|, \tag{6.5}$$

where the absolute value is taken elementwise. Distributions that can handle outliers in the data are often referred to as "heavy tail distributions" [44]. Other distributions such as the student-t have been successfully used to model outliers in the data [75]. Needless to say, the choice of the distribution should fit the type of data and their statistics. For data with many outliers, robust distributions work well. However, when working with data that naturally scale over many decades, such as time domain EM data, these techniques have to be used carefully. Robust statistics may consider very large (in absolute value) data points as outliers when in fact they represent reasonable data. The choice of the weights $\mathbf{w}$ or covariance Σ_d plays a major role in generating a reasonable fit.

Although the derivation above seems straightforward, it is actually rather difficult to implement in practice. The main reason is that it involves the covariance Σ_d or the weights $\mathbf{w}$. Typically, it is assumed that the data variance is known; however, in practice, this is rarely the case. The user of an inversion code will find that changing the assumptions on the covariance and the weights results in highly different solutions. We therefore try to provide a practical guide for choosing the weights.

Three common assumptions are used to estimate the covariance/weights.

- The data are not correlated; that is, $\Sigma_d = \text{diag}(\mathbf{w}^{-1})$, where $\mathbf{w}_i = \sigma_i^{-1}$ is the inverse standard deviation of each datum.
- Any physical instrument has some floor value, and the data are only as accurate as this floor value.
- The error in each datum is proportional to its magnitude.

This leads to the noise model

$$\sigma_i = \beta|\mathbf{d}_i| + \sigma_0 \quad \text{or} \quad \mathbf{w}_i = \frac{1}{\beta|\mathbf{d}_i| + \sigma_0}. \tag{6.6}$$

The numbers β and σ_0 are set by the understanding of the experiment. For example, for most electromagnetic problems, a measurement error of 1% is considered to be low, while σ_0 is determined by the instrumentation. It is also possible to use methods such as cross validation (CV) [77] to estimate β and σ_0, but the discussion about the use of CV in this application is beyond the scope of this work. It is important to stress that the above noise model is not "correct," and in the presence of more information different noise models should be built. For example, the assumption on uncorrelated data is very restrictive and does not apply in many scenarios. One has to assess each data set in order to obtain the noise model. Nonetheless, we have found that the above model works relatively well for many applications.

6.2 ▪ Overfitting and regularization techniques

The construction of the data misfit leads to the optimization problems where equations (6.3) or (6.5) are minimized. However, before one attempts to solve these problems, it is important to understand the properties of the solution. To analyze the solution we assume that the forward problem is linear; that is,

$$\mathbf{d} = \mathbf{J}\mathbf{m} + \boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon}$ is normal with covariance matrix $\sigma^2\mathbf{I}$. This leads to the likelihood

$$\text{Likelihood}(\mathbf{m}) \propto \exp\left(-\frac{1}{2\sigma^2}||\mathbf{J}\mathbf{m} - \mathbf{d}||^2\right)$$

and to the optimization problem

$$\widehat{\mathbf{m}} = \arg\min \frac{1}{2}||\mathbf{J}\mathbf{m} - \mathbf{d}||^2.$$

Assuming that $\mathbf{J}^\top\mathbf{J}$ is invertible, we obtain

$$\widehat{\mathbf{m}} = (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\mathbf{d}.$$

It is natural to ask what the error is between the true $\mathbf{m}$ and our estimator. By direct substitution we obtain that

$$\widehat{\mathbf{m}} - \mathbf{m} = (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top(\mathbf{J}\mathbf{m} + \boldsymbol{\epsilon}) - \mathbf{m} = (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\boldsymbol{\epsilon}.$$

The expected value of this estimator is

$$\mathbb{E}(\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\boldsymbol{\epsilon} = (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\mathbb{E}\boldsymbol{\epsilon} = 0$$

because we assumed that $\boldsymbol{\epsilon}$ has zero mean. This implies that the estimator $\widehat{\mathbf{m}}$ is an unbiased estimator of $\mathbf{m}$ and converges to $\mathbf{m}$ as the number of data approaches infinity (assuming that the eigenvalues of $(\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top$ are bounded independent of the number of data). Nonetheless, the variance of our estimator is not necessarily small since

$$\mathbb{E}||(\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top\boldsymbol{\epsilon}||^2 = \sigma^2\text{trace}[(\mathbf{J}^\top\mathbf{J})^{-1}]. \tag{6.7}$$

If $\mathbf{J}$ has very large and very small singular values, then the variance of our estimator can be very large indeed, making it useless. Therefore, estimators that maximize the likelihood, also called maximum likelihood estimators (MLEs) are useful only for well-posed problems.

Another way to understand the failure of the MLE is through the statistical properties of the residual. Assume that we have obtained a model $\widehat{\mathbf{m}}$ that solves the optimization problem (6.3). The first question to be asked is whether this model is "reasonable." To answer this question we look at the residual

$$\mathbf{r}(\widehat{\mathbf{m}}) = \mathbf{d} - \mathbf{P}\mathbf{A}(\widehat{\mathbf{m}})^{-1}\mathbf{q}.$$

Assuming, for simplicity, a uniform standard deviation $\sigma^2 = 1$ for all data, if we had the "true" model in our hands, then we would have the residual

$$\mathbf{r}^{\text{true}} = \mathbf{d} - \mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q}) = \boldsymbol{\epsilon}.$$

Assuming that $\boldsymbol{\epsilon}$ is Gaussian with 0 mean and a unit standard deviation, we have that its sum square is also a random variable,

$$\phi_d = \boldsymbol{\epsilon}^\top \boldsymbol{\epsilon} \sim \chi^2_n. \tag{6.8}$$

That is, ϕ_d is a random number that has a χ^2_n distribution. The expected value of ϕ_d is n, where n is the dimension of $\boldsymbol{\epsilon}$ and its standard deviation is $\sqrt{2n}$. Now assume that

$$\mathbf{r}(\widehat{\mathbf{m}})^\top \mathbf{r}(\widehat{\mathbf{m}}) \ll n - \sqrt{2n}.$$

In this case we have overfit the data. As a result we are fitting to noise and will obtain solutions that contain too much structure. Thus, simply minimizing the misfit often leads to unreasonable results.

Another problem with trying to minimize the misfit is that, as is straightforward to see, the solution to the optimization problem is nonunique. There are two main reasons for this nonuniqueness. The first is simple: while the number of data points is always finite, the number of model parameters is unlimited as $\mathbf{m}$ is a discretization of a continuous function $m(\vec{x})$. We therefore obtain a problem with more unknowns than equations, and the minimization problem does not admit a unique solution. A second reason for the nonuniqueness stems from the fact that the forward problem is a PDE. Considering its discrete analogue, the forward modeling matrix, $\mathbf{A}(\mathbf{m})$, has eigenvalues that cluster at infinity, and the eigenvectors that are associated with the large eigenvalues are typically highly oscillatory. This implies that its inverse $\mathbf{A}(\mathbf{m})^{-1}$ has eigenvalues that cluster at 0 and that the very small eigenvalues are associated with oscillatory eigenvectors. Therefore, if we try to fit the data well, small perturbations to the data lead to large perturbations in the recovered model.

The above discussion stresses the point that data fitting by itself is insufficient to obtain a reasonable estimate for the inverse problem. We now discuss a number of techniques that allow for the estimation of the model given the data. These techniques can be referred to as *regularization methods*. Roughly speaking, regularization techniques reduce the variance of the problem by adding terms to the optimization problem that restrict the solution of the MLE, adding bias and obtaining a solution that is (we hope) not far from the "true" solution. We now discuss a number of such techniques and discuss their properties.

6.2.1 ▪ Subspace and parametric techniques

The above discussion implies that, in order to obtain an estimator of the model, one needs to deal with the problem of overfitting. Maybe the oldest method to obtain a solution to the problem is using subspace or parametric techniques. Subspace and parametric techniques restrict the dimension of the model, $\mathbf{m}$, by parameterizing the space of feasible solutions, making the problem well-posed. For example, in some geophysical applications we may try to recover a model that changes only in 1D with only a few different layers; that is, $m(x,y,z) = m(z)$ with $m(z)$ being a piecewise constant function containing k layers. This reduces the number of parameters and in many cases allows for a unique recovery of a model.

A more general treatment of a subspace is to assume that $\mathbf{m}$ can be written as a linear combination of k vectors

$$\mathbf{m} = \mathbf{V}_k \mu,$$

where the matrix $\mathbf{V}_k$ is chosen by assuming some a priori information and k is small. If the subspace $\mathbf{V}_k$ is sufficiently small, the problem becomes well-posed, and the solution to the optimization problem (assuming Gaussian statistics) is

$$\mu^* = \arg\min_{\mu} (\mathbf{d} - \mathbf{PA}(\mathbf{V}_k \mu)^{-1}\mathbf{q})^\top \Sigma_d^{-1} (\mathbf{d} - \mathbf{PA}(\mathbf{V}_k \mu)^{-1}\mathbf{q}). \tag{6.9}$$

Clearly, making different choices for $\mathbf{V}_k$ yields very different solutions to the problem. Recently, a number of authors [66] have proposed choosing $\mathbf{V}_k$ by assuming that the model is a random variable and using an assumed known covariance. To this end, one needs to assume that $\mathbf{m}$ has a covariance matrix $\Sigma_{\mathbf{m}}$ that is chosen by, say, geostatistical tools. One can then compute or approximate the largest k eigenvalues and their associated eigenvectors and use them as the subspace vectors in the recovery process.

To understand the quality of such estimators we repeat the calculation above for a linear inverse problem to obtain (dropping the subscript k from $\mathbf{V}$)

$$\mathbf{m} - \widehat{\mathbf{m}} = \mathbf{m} - \mathbf{V}(\mathbf{V}^\top \mathbf{J}^\top \mathbf{J}\mathbf{V})^{-1}\mathbf{V}^\top \mathbf{J}^\top (\mathbf{J}\mathbf{m} + \boldsymbol{\epsilon}).$$

Assume that the true model can be written as

$$\mathbf{m} = \mathbf{V}\mu_t + \mathbf{V}^\perp \theta,$$

where $\mathbf{V}^\perp$ is orthogonal to $\mathbf{V}$ and μ_t is the true μ. Then

$$\begin{aligned} \mathbf{m} - \widehat{\mathbf{m}} &= \mathbf{V}\mu_t + \mathbf{V}^\perp\theta - \mathbf{V}(\mathbf{V}^\top \mathbf{J}^\top \mathbf{J}\mathbf{V})^{-1}\mathbf{V}^\top \mathbf{J}^\top (\mathbf{J}(\mathbf{V}\mu_t + \mathbf{V}^\perp\theta) + \boldsymbol{\epsilon}) \\ &= \mathbf{V}^\perp\theta - \mathbf{V}(\mathbf{V}^\top \mathbf{J}^\top \mathbf{J}\mathbf{V})^{-1}\mathbf{V}^\top \mathbf{J}^\top \boldsymbol{\epsilon}. \end{aligned}$$

In this case there is a systematic error in our estimator. Note that

$$\mathbb{E}(\mathbf{m} - \widehat{\mathbf{m}}) = \mathbf{V}^\perp\theta.$$

Thus, even if we collect infinite data, this estimator will not converge to the true model. The quantity

$$\mathsf{bias} = \mathbb{E}(\mathbf{m} - \widehat{\mathbf{m}})$$

is referred to as the bias. One may wonder why we use methods that have bias. To understand this we look at the mean square error of our estimator (assume that $\mathbf{V}$ is orthogonal)

$$\mathbb{E}(||\mathbf{m} - \widehat{\mathbf{m}}||^2) = ||\mathbf{V}^\perp\theta||^2 + \sigma^2 \mathsf{trace}((\mathbf{V}^\top \mathbf{J}^\top \mathbf{J}\mathbf{V})^{-1}) = ||\mathsf{bias}||^2 + \mathsf{variance}.$$

The variance term has now changed from (6.7). If the matrix $\mathbf{V}^\top\mathbf{J}^\top\mathbf{J}\mathbf{V}$ is well conditioned, then the variance term is small. This makes this estimator much more stable than the MLE. The hope for this estimator is that the bias term $\mathbf{V}^\perp\theta$ is small, and, in many cases, this can be justified by understanding the physics and geology of the problem and our expectations of the model. On the other hand, if we have made an inappropriate choice for our subspace, then the bias term may be large, and we may end up with a very rough estimation of the true model.

It can be shown that the decomposition of the mean square error into a bias term and a variance term is general for *any* estimator (linear or nonlinear). This stems from the following simple equality:

$$\begin{aligned}\mathsf{MSE} &= \mathbb{E}\|\widehat{\mathbf{m}}-\mathbf{m}\|^2 = \mathbb{E}\|\widehat{\mathbf{m}}-\mathbf{m}+\mathbb{E}\widehat{\mathbf{m}}-\mathbb{E}\widehat{\mathbf{m}}\|^2\\ &= \mathbb{E}\|\widehat{\mathbf{m}}-\mathbb{E}\widehat{\mathbf{m}}\|^2+\mathbb{E}\|\mathbf{m}-\mathbb{E}\widehat{\mathbf{m}}\|^2-2\mathbb{E}(\widehat{\mathbf{m}}-\mathbb{E}\widehat{\mathbf{m}})^\top(\mathbf{m}-\mathbb{E}\widehat{\mathbf{m}})\\ &= \underbrace{\mathbb{E}\|\widehat{\mathbf{m}}-\mathbb{E}\widehat{\mathbf{m}}\|^2}_{\mathsf{Var}}+\underbrace{\|\mathbf{m}-\mathbb{E}\widehat{\mathbf{m}}\|^2}_{\|\mathsf{bias}\|^2}.\end{aligned}$$

For nonlinear estimators, one cannot obtain a closed form expression of the bias and variance, and it has to be estimated numerically. However, the above analysis teaches us that this decomposition means that there is a compromise between the bias and the variance, also referred to as the bias-variance dilemma [28]. The "trick" in any good recovery technique that obtains a stable estimate of the model (that is, an estimator with a small variance) is to not introduce a large bias term.

Although linear subspaces are common, in many cases a nonlinear parameterization can be used. For example, in some cases one assumes that the model is an ellipsoid, with unknown depth, orientation, and axis, that is covered within a homogenous earth. Such simple assumptions have been used for unexploded ordinance detection [60]. The choice of the nonlinear parameterization is problem dependent. In this case we have that

$$\mathbf{m} = g(\mu),$$

where g is a nonlinear transformation $g : \mathbb{R}^k \to \mathbb{R}^n$. In order to use standard techniques the Jacobian of the transformation is also needed, and we assume that the $n \times k$ matrix $\mathbf{G} = \nabla_\mu g(\mu)$ can be computed. In some cases the nonlinear transformation is not differentiable and has to be smoothed. For example, a common transformation that reduces the space of possible solutions is based on level-set methods [59, 52, 22, 74]. In this case we parameterize the model as an indicator function:

$$\mathbf{m} = \begin{cases}\sigma_0 & \text{if } \mu < 0,\\ \sigma_1 & \text{otherwise.}\end{cases}$$

This parameterization does not make the problem well-posed, and additional regularization is needed to deal with the remaining ill-posedness. This regularization is not smooth, and therefore it is usually smoothed out by introducing the nonlinear smooth function

$$\mathbf{m} = \sigma_0 + \frac{1}{2}(1+\tanh(\alpha\mu))(\sigma_1-\sigma_0),$$

where α is chosen such that the function is smooth on the mesh used for $\mathbf{m}$.

There are two main difficulties when working with subspace and parametric methods. First, the choices of the subspace or parameter size, k, and the subspace vectors or nonlinear transformation to be used are not always easy, as it can be difficult to estimate what

subspace or parameterization is sufficient to span the true model. In many cases (such as the level-set example above), the resulting parameterization still yields an ill-posed problem. The second problem is that it can be difficult to ensure that the bias remains small. In particular, it is hard to impose a number of geological constraints. We therefore look at different regularization schemes that allow for higher degrees of flexibility.

6.2.2 ▪ L_2 based regularization

L_2 based regularizations are often used when assuming that the models are smooth or when smoothed solutions are sought. There are a number of ways to motivate this type of regularization. The approach we take here attempts to balance the data fitting term with a term that ensures regularity of the solution, thus increasing the bias while reducing the variance. This leads to the optimization problem

$$\mathbf{m}^* = \arg\min_{\mathbf{m}} \frac{1}{2}||\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q}||^2_{\Sigma_d^{-1}} + \frac{\alpha}{2}||\mathbf{L}(\mathbf{m} - \mathbf{m}_{\text{ref}})||^2. \tag{6.10}$$

The function above is composed of two parts. The first is the misfit discussed before, and the second is a regularization term that penalizes unwanted behavior of the solution. The regularization is built from the following components: $\mathbf{m}_{\text{ref}}$ is the reference model to be discussed next, α is a regularization parameter that aids in the tradeoff between the data fit and the regularization, and the matrix $\mathbf{L}$ can be freely chosen by the user, but a common choice is the discretization of the integral

$$||\mathbf{Lm}||^2 \approx \int_\Omega \gamma_0(m - m_{\text{ref}})^2 + \gamma_1(m - m_{\text{ref}})_x^2 + \gamma_2(m - m_{\text{ref}})_y^2 + \gamma_3(m - m_{\text{ref}})_z^2 \, d\vec{x}. \tag{6.11}$$

This approach is also known as the Tikhonov regularization [72].

The discretization of the integral (6.11) and the matrix $\mathbf{L}$ can be done using the finite volume techniques discussed previously. The choice of the parameters $\gamma_0, \ldots, \gamma_3$ and the reference model m_{ref} is problem dependent. In many cases γ_0 is chosen to be very small such that the resulting solution is very smooth. For many geophysical problems, one assumes that the earth is smoother in the x, y-direction than in the z-direction, and therefore γ_1 and γ_2 are chosen much larger than γ_3. For example, a ratio of 1:2 or 1:3 is common. It is important to note that there is no "magic" formula and that the choice of these parameters is problem dependent. The reference model m_{ref} is crucial as it biases the estimated model toward m_{ref}. There is no such thing as "no reference model." The choice $m_{\text{ref}} = 0$ is a choice just as $m_{\text{ref}} = 1$ or any other function is. In many cases m_{ref} is chosen to be a uniform background obtained from simple analysis of the data such as apparent conductivity or other considerations.

The regularization parameter α is usually chosen such that $\phi_d(\mathbf{m}(\alpha))$ approximately fits the Chi-squared (χ^2) misfit in (6.8). We discuss methods for the selection of α later in this chapter.

For linear problems it is possible to analyze the regularized estimator in the same manner as above, obtaining

$$\mathbb{E}(||\mathbf{m} - \widehat{\mathbf{m}}||^2) = \alpha^2(\mathbf{m} - \mathbf{m}_{\text{ref}})^\top \mathbf{L}^\top \mathbf{L}(\mathbf{m} - \mathbf{m}_{\text{ref}}) + \sigma^2 \text{trace}(\mathbf{J}^\top(\mathbf{J}^\top\mathbf{J} + \alpha\mathbf{L}^\top\mathbf{L})^{-1}\mathbf{J}).$$

This implies that if we choose $\mathbf{L}$ such that $\mathbf{L}(\mathbf{m} - \mathbf{m}_{\text{ref}})$ is small, then the bias term is small, and we obtain a reasonable reconstruction. For example, if the model is smooth

and m_{ref} is constant, then ∇m is small, and therefore its discrete analogue $\mathbf{Lm}$ is small as well, yielding a reasonable reconstruction. However, if the model contains discontinuities, then ∇m is unbounded and the bias is large, resulting in a rather large error in the reconstruction.

6.2.3 ▪ Total variation and anisotropic diffusion

While L_2 based solutions tend to give smooth looking models, in some cases one may want to obtain a solution with discontinuities. To understand why L_2 regularization cannot yield discontinuous solutions, assume for simplicity that m_{ref} is constant, and note that $\int_\Omega |\nabla m|^2 \, dv$ is not bounded for discontinuous m, as ∇m yields a delta function that is not square intergrable. On the other hand,

$$R(m) = \int_\Omega |\nabla m|^p \, d\vec{x}, \tag{6.12}$$

with $p \leq 1$ well defined and bounded, as the delta function is integrated to obtain a finite value. The case $p = 1$ is referred to as total variation (TV) regularization and is well studied and a popular choice for many problems.

Other functions of the gradient that can be used are the student-t function

$$\rho(t) = \frac{1}{1+t^2}$$

and other functions that have similar forms.

It should not come as a surprise that these functions are identical to the robust estimators (6.5) that are used in the presence of data outliers. A discontinuity in the model can be thought of as an outlier in its gradient, and therefore functions that are used for robust statistics for the data misfit are used here as well. The first use of such a regularization was reported in [17] in 1973 and then rediscovered independently for image processing in [63]. Further work in image processing [10, 61] has used more exotic regularization. The advantage of TV regularization is that it is convex and thus does not introduce local minima to the inverse problem. Regularization methods that are based on $p < 1$ and other functionals may introduce local minima, even if the original problem is convex. Nonetheless, in many cases, nonconvex regularization tends to give more discontinuous models and is sometimes preferable to some users.

When using nonquadratic regularization it is impossible to evaluate the bias and variance terms in a closed form. By using examples, it is possible to show that for models that contain discontinuities these methods indeed reduce the bias and therefore can be advantageous. Nonetheless, at this point, it is worth pausing for a moment and thinking about the meaning of such a regularization. Assume that $m(x)$ is a 1D function in the interval $[0, 1]$ and that we know m at two points only. If we are to estimate m given these two points, we may want to use linear interpolation, which does not give an edge. This of course does not mean that the true model does not contain an edge but simply that the data do not have information about such an edge and its location. Now suppose that we use some regularization technique and obtain an edge. Clearly, this edge is not backed by the data, and it is obtained by the choice of regularization. While this example is very simple, it is rather representative. In many geophysical applications, it is possible to generate a reasonable fit to the data using smooth models. Thus, in many cases, edges are generated by the regularization term and are not mandated by the data. One has to wonder if

the regularization puts the edge at the right place. If it does not, a smooth model may be better.

In the process of obtaining the solution to an inverse problem with a particular regularization, derivatives are needed. Differentiating the TV-like operator, we obtain that

$$\nabla_m R(m) = \nabla \cdot \rho(|\nabla m|)\nabla m,$$

where

$$\rho(t) = p|t|^{p-1}.$$

The differential operator $\nabla_m R(m)$ can be viewed as an anisotropic diffusion operator. Indeed, the operator $\nabla \cdot \nabla$ is the Laplacian, and the operator $\nabla \cdot \rho(|\nabla m|)\nabla$ can be seen as a diffusion operator that diffuses differently in different directions.

6.3 ▪ L_1 based regularization

A property of TV is that it tends to give piecewise constant solutions. That is, the gradient of the model is sparse; it has only a small number of nonzero entries.

A generalization of this idea is L_1 based regularization. It is rather well known that L_1 norm regularization yields solutions with many zeros and very few nonzeros. This observation was used extensively by geophysicists in the 1970s and 1980s [17, 65, 81] to obtain so-called spiky solutions to inverse problems. Recently, proofs regarding the amount of sparsity under some strict conditions have been provided [15], and this has generated a "hot" trend within the inverse problem community of trying to solve almost all inverse problems with sparse-like solutions. We now briefly review some of the techniques for sparse recovery and discuss some of the applications, their advantages, and limits.

Consider first the case of imaging a cluster of point conductors treated as "spikes." This is a simple case of "sparse solution." Sparse solution implies that most of the entries in $\mathbf{m}$ are zero. The appropriate problem that is used for this type of model is

$$\min_{\mathbf{m}} \ \frac{1}{2}||\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q}||^2_{\Sigma_d^{-1}} + \alpha||\mathbf{m}||_1, \tag{6.13}$$

where

$$||\mathbf{m}||_1 = \sum_j |\mathbf{m}_j|.$$

The case of scattered point conductors is mostly unrealistic; however, even if the model is not sparse, we assume that it can be expressed using a basis function,

$$\mathbf{m} = \mathbf{W}\mathbf{z},$$

where $\mathbf{W}$ is some basis and $\mathbf{z}$ are coefficients. The important assumption is that given an appropriate choice of $\mathbf{W}$, the model can be expressed using only a few of the basis vectors in $\mathbf{W}$, and therefore we can minimize

$$\min_{\mathbf{z}} \ \frac{1}{2}||\mathbf{d} - \mathbf{PA}(\mathbf{W}\mathbf{z})^{-1}\mathbf{q}||^2_{\Sigma_d^{-1}} + \alpha||\mathbf{z}||_1. \tag{6.14}$$

The choice of $\mathbf{W}$ is crucial. It is easy to see that it is possible to choose a $\mathbf{W}$ that yields sparse solutions without any advantage compared with the 2-norm solutions introduced

in the previous section. For example, for linear problems, if we choose $\mathbf{W} = \mathbf{V}$, where $\mathbf{V}$ is the right-hand singular vector matrix, then it is easy to verify that we simply obtain the truncated SVD solution. Choosing $\mathbf{W}$ judicially is problem dependent, and for many inverse problems an appropriate $\mathbf{W}$ is hard to find.

Another difficulty with L_1 regularization is that it is not quadratic and not differentiable. The question is how to effectively solve such problems. Here we discuss two main approaches.

First, it is possible to use iterative reweighted least squares (IRLS). IRLS has been used in the past for many problems with much success. IRLS is a simple strategy that linearly converges for the solution of the problem. Rather than solving the original nondifferentiable problem, we "regularize" the regularizer. Defining

$$||\mathbf{m}||_{1,\epsilon} = \sum_i \sqrt{\mathbf{m}_i^2 + \epsilon},$$

we replace the one norm with a differentiable function, minimizing

$$\min_{\mathbf{m}} \ \frac{1}{2}||\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q}||^2_{\Sigma_d^{-1}} + \alpha||\mathbf{m}||_{1,\epsilon}. \tag{6.15}$$

Next, we replace the problem with a sequence of quadratic problems of the form

$$\min_{\mathbf{m}_k} \ \frac{1}{2}||\mathbf{d} - \mathbf{PA}(\mathbf{m})^{-1}\mathbf{q} - \mathbf{J}(\mathbf{m}_k - \mathbf{m}_{k-1})||^2 + \frac{\alpha}{2}\mathbf{m}_k^\top \operatorname{diag}\left(\frac{1}{\sqrt{\mathbf{m}_{k-1}^2 + \epsilon}}\right)\mathbf{m}_k. \tag{6.16}$$

Hence the name is iterative reweighted least squares (IRLS). The advantage of this approach is that one can use tools developed for the quadratic problem. There are two disadvantages to IRLS. First, each iteration can be rather expensive, solving a linear system of equations to high accuracy when this may not be needed. Second, the choice of ϵ may pose more difficulty.

A second approach for the solution of the L_1 problem is to replace the nonsmooth problem by a smooth optimization problem with inequality constraints.

Setting $\mathbf{m} = \mathbf{p} - \mathbf{w}$ with both $\mathbf{p}, \mathbf{w} \geq 0$, the optimization problem (6.13) is equivalent to the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{p},\mathbf{w}} \frac{1}{2}||\mathbf{d} - \mathbf{PA}(\mathbf{p} - \mathbf{w})^{-1}\mathbf{q}||^2_{\Sigma_d^{-1}} + \alpha \sum_i \mathbf{p}_i + \mathbf{w}_i \\ &\text{s.t. } \mathbf{p}, \mathbf{w} \geq 0. \end{aligned} \tag{6.17}$$

A very effective method to solve optimization problems with simple inequality constraints that does not require matrix inversion is the gradient projection method. We will discuss this method in the next section.

6.3.1 ▪ Discretization of regularization operators

We now briefly discuss the discretization of the regularization operators and explore some of the possible pitfalls.

The regularization techniques involve the discretization of the gradient of a cell-centered quantity. Assuming we have a tensor mesh, a first order approximation can be obtained

by the simple difference

$$(\partial_x m)_{i+\frac{1}{2}} \approx \frac{2(\mathbf{m}_{i+1} - \mathbf{m}_i)}{\mathbf{h}_{i+1} + \mathbf{h}_i},$$

which can be expressed in the usual matrix form (3.4), obtaining that $\nabla m \approx \mathsf{GRAD}_c \mathbf{m}$, where GRAD_c is a cell-centered gradient matrix. The gradient of a cell-centered variable is naturally centered on cell faces. To compute an approximation to the integrals (6.11) and (6.12), we average the quantities to cell centers, multiply by the cell volume, and sum, obtaining

$$\int_\Omega (\nabla m)^2 \, dv \approx \mathbf{v}^\top \mathbf{A}_f (\mathsf{GRAD}_c \mathbf{m})^2 = \mathbf{m}^\top \mathsf{GRAD}_c^\top \mathsf{diag}(\mathbf{A}_f^\top \mathbf{v}) \mathsf{GRAD}_c \mathbf{m}, \tag{6.18}$$

$$\int_\Omega (\nabla m)^p \, dv \approx \mathbf{v}^\top \mathbf{A}_f |\mathsf{GRAD}_c \mathbf{m}|^p, \tag{6.19}$$

where the square and absolute value are taken pointwise.

6.3.2 ▪ Hard constraints

For many inverse problems (probably most of them), hard constraints on the model parameters are known—for example, the relative susceptibility $0 < \mu_r \leq \mu_{\max}$. Virtually any physical model has some sort of bound constraints, and therefore, without loss of generality, we assume that

$$\mathbf{m}_{\mathsf{low}} \leq \mathbf{m} \leq \mathbf{m}_{\mathsf{high}}. \tag{6.20}$$

While in some problems these constraints play a minor role, for many electromagnetic problems and especially for magnetic problems these constraints can yield much more reliable results. The incorporation of bound constraints into the inversion is discussed in the next section.

6.4 ▪ Inversion through optimization

In the previous sections we have motivated the evaluation of the model by solving the optimization problem

$$\min_{\mathbf{m}} \; \phi(\mathbf{m}) = \phi_d(\mathbf{m}) + \alpha R(\mathbf{m}), \tag{6.21}$$

where $R(m)$ is a regularization term and $\phi_d(\mathbf{m})$ is a misfit term. This problem is an unconstrained optimization problem. If the bound constraints (6.20) are added, then we obtain a bound constrained optimization problem. We now briefly discuss optimization techniques for this problem. We start with the unconstrained problem and then discuss an algorithm for the constrained problem. A more comprehensive discussion of optimization techniques can be found in [58, 51].

The methods discussed here assume that the electric field $\mathbf{e}$ can be expressed as $\mathbf{e}(\mathbf{m})$; that is, we solve the forward problem at each iteration. For completeness we note that other techniques that do not assume this relation can be used [36]. These techniques are referred to as "all at once methods." While they can be very effective, they are difficult to use when the number of sources and frequencies grows. Furthermore, since the forward problem is solved only approximately at each step, these methods are more involved when applied in practice. We refer the interested reader to [38] for further details.

6.4.1 ▪ First order methods

First order methods are methods that use only the objective function and gradient information. To this end, we note that the gradient of the least square misfit can be written as

$$\min_{\mathbf{m}} \nabla_{\mathbf{m}}\phi(\mathbf{m}) = \mathbf{J}^{\top}(\mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q} - \mathbf{d}) + \alpha\nabla_{\mathbf{m}}R(\mathbf{m}). \tag{6.22}$$

The sensitivity matrix, $\mathbf{J}$, need not be calculated explicitly, and we can compute $\mathbf{J}^{\top}$ times a vector by solving the adjoint problem. In the case of time dependent problems this involves the solution of the time dependent problem backward in time (see Chapter 4). Thus, the evaluation of the gradient requires the solution of the forward problem and the adjoint problems. For geophysical surveys that contain many sources and receivers this has to be done for each source.

Note that if the number of receivers is small, then it is possible to replace the sources with receivers. For example, assume we have many sources and a single receiver. Such a case should be considered when, for example, the receiver is placed on the surface and the source is moving. The mathematical model for this case is

$$\mathbf{d} = \mathbf{p}^{\top}\mathbf{A}(\mathbf{m})^{-1}\mathbf{Q} = (\mathbf{Q}^{\top}\mathbf{A}(\mathbf{m})^{-\top}\mathbf{p})^{\top}.$$

Thus, we replace the sources with receivers and solve the adjoint problem rather than the forward problem.

The idea of changing sources for receivers is often referred to as reciprocity. Reciprocity implies that the data obtained at a receiver from a source are identical by switching between the two. While it is possible to explain this in continuous space, the discretized version is simpler. Note that the data are a simple inner product between the field $\mathbf{e} = \mathbf{A}^{-1}\mathbf{q}$ and a discretized receiver $\mathbf{p}$. Therefore (assuming real arithmetic)

$$d = \mathbf{p}^{\top}\mathbf{e} = \mathbf{p}^{\top}\mathbf{A}^{-1}\mathbf{q} = \mathbf{q}^{\top}\mathbf{A}^{-\top}\mathbf{p}.$$

If $\mathbf{A} = \mathbf{A}^{\top}$, then switching sources and receivers with the same forward problem gives the same result. Note that in the presence of complex arithmetic this changes to

$$d = \mathbf{p}^{\top}\mathbf{e} = \mathbf{p}^{\top}\mathbf{A}^{-1}\mathbf{q} = (\mathbf{q}^{\top}\mathbf{A}^{-\top}\mathbf{p})^{*},$$

where $(*)$ is the complex conjugate. In the case that $\mathbf{A}$ is not symmetric one needs to use the adjoint problem (see, for example, [16]).

Steepest descent and preconditioned steepest descent

Maybe the simplest and most basic method to minimize a function is steepest descent. Using Taylor's series, we have that

$$\phi(\mathbf{m} + \delta\mathbf{m}) = \phi(\mathbf{m}) + \nabla_{\mathbf{m}}\phi(\mathbf{m})^{\top}\delta\mathbf{m} + \mathcal{O}||\delta\mathbf{m}||^2.$$

To have $\phi(\mathbf{m}+\delta\mathbf{m}) < \phi(\mathbf{m})$, we need to choose a direction $\delta\mathbf{m}$ such that $\nabla_{\mathbf{m}}\phi(\mathbf{m})^{\top}\delta\mathbf{m} < 0$ (assuming that $\nabla_{\mathbf{m}}\phi(\mathbf{m}) \neq 0$). An obvious choice is

$$\delta\mathbf{m} = -\nabla_{\mathbf{m}}\phi(\mathbf{m}),$$

which defines the steepest descent method. The method can be summarized as follows.

Algorithm: Steepest descent

- Initialize $\mathbf{m} = \mathbf{m}_0$
- While not converge
  1. Compute $\phi(\mathbf{m})$ and $\nabla_{\mathbf{m}}\phi$ (two forward problems)
  2. Set $\mathbf{m} \leftarrow \mathbf{m} - \mu\nabla_{\mathbf{m}}\phi$, where μ is set by a line search (see next)
  3. Test for convergence

Note that the cost of each step is at least two forward modeling problems at each step.

The line search guarantees that the function decreases. A simple line search is the Armijo line search. The idea is to start with an initial step size, test that the function decreases, and then decrease the line search parameter until ϕ is decreased. The algorithm is as follows.

Algorithm: Line search

- Given a direction $\mathbf{s}$ check that $\mathbf{s}^\top\nabla_{\mathbf{m}}\phi < 0$
- Choose a parameter $\gamma > 0$
- While $\phi(\mathbf{m} + \mu\mathbf{s}) < \phi(\mathbf{m}) + \gamma\mathbf{s}^\top\nabla_{\mathbf{m}}\phi$
  - Compute $\phi(\mathbf{m} + \mu\mathbf{s})$ (one forward problem)
  - Set $\mu \leftarrow \frac{1}{2}\mu$

The reason the term $\gamma\mathbf{s}^\top\nabla_{\mathbf{m}}\phi$ is added is to guarantee sufficient decrease of the objective function (see [58] for details). Typically, the parameter γ is set to be very small: $\gamma = 10^{-4}$. Although the algorithm is "guaranteed" to find an appropriate μ, this may not happen. There are a number of reasons for such a failure. First, this may happen only for very small μ, and we may not want to take very small steps as they practically leave us with the same model. A second reason for a potential failure is that the computation of the forward and gradient depends on the solution of linear systems. When using iterative methods, if the tolerance of the solver is not set to be sufficiently small, then we may not have a descent direction. Therefore, we usually set a maximum number of steps and terminate the line search if the objective function does not decrease.

Steepest descent is known to converge slowly. One simple way to accelerate its convergence is to choose an SPD matrix, $\mathbf{H}$, and choose a direction

$$\mathbf{s} = -\mathbf{H}^{-1}\nabla_{\mathbf{m}}\phi.$$

The direction is clearly a descent direction as $\mathbf{s}^\top\nabla_{\mathbf{m}}\phi < 0$. As we see next, the best choice is some approximation to the Hessian, $\mathbf{H} = \nabla^2_{\mathbf{m}}\phi$, but this choice can be rather expensive. For many inverse problems a natural selection is

$$\mathbf{H} = \nabla^2_{\mathbf{m}}R(\mathbf{m}).$$

This choice usually speeds convergence significantly. Other, more sophisticated, approaches can be used. For example, $\mathbf{H}$ can be a (fixed) matrix computed on a coarse mesh or some analytic expression for homogeneous media.

There are a number of other practical aspects for the implementation of steepest descent. The first relates to scaling. Unfortunately, steepest descent is very sensitive to scaling. For example, assume that we change the units of our data from volts to millivolts. This amounts to minimizing $c\phi(\mathbf{m})$ rather than $\phi(\mathbf{m})$, where c is a constant, in our example, 10^6. Clearly, the gradient is simply $c\nabla_{\mathbf{m}}\phi(\mathbf{m})$. If the typical value of $\nabla\phi$ is roughly 1, then the scaled objective function has a gradient with typical values of 10^6. This implies that many line search steps are needed to find a feasible step, and, in most cases, the code will quit before such a step is found. Therefore, assuming one uses steepest descent, making sure that the function is scaled right is essential.

A second important practical aspect of the line search is the choice of μ when entering into the line search. Usually, we assume that the direction is well scaled, and we pick $\mu = 1$ as an initial μ. This choice can be wasteful. Assume that at the previous iteration we chose $\mu = 10^{-3}$. It is unlikely that $\mu = 1$ is a reasonable step for the current iteration. A better strategy is to use the previous μ in order to initialize the current one. However, if one uses the previous μ blindly, we have a mechanism to make μ smaller but not bigger. Thus, the code will choose small steps where the function is highly nonlinear but will not adapt to areas where the function tends to be more linear. I have found that the following heuristic strategy works well.

Algorithm: Choice of μ

- Initialize $\mu = 1$ (assume that the problem is scaled)
- While not converge
  - If previous line search is successful in the first attempt, try $\mu_{k+1} = \min(1, 2\mu_k)$
  - Otherwise $\mu_{k+1} = \mu_k$

This strategy can be modified to be more conservative, increasing μ after a few successful attempts, but the basic idea is to use prior information about the line search parameter in order to not waste expensive function evaluations.

Quasi-Newton methods and the (L)BFGS method

Using Taylor's expansion, we have that

$$\phi(\mathbf{m}+\delta\mathbf{m}) = \phi(\mathbf{m}) + \nabla_{\mathbf{m}}\phi(\mathbf{m})^{\top}\delta\mathbf{m} + \frac{1}{2}\delta\mathbf{m}^{\top}\mathbf{H}(\mathbf{m})\delta\mathbf{m} + \mathcal{O}||\delta\mathbf{m}||^3,$$

where $\mathbf{H}(\mathbf{m}) = \nabla^2\phi(\mathbf{m})$ is the Hessian. Assuming for a moment that Hessian information is available, we can optimize the above with respect to $\delta\mathbf{m}$ to obtain the so-called Newton system

$$\mathbf{H}(\mathbf{m})\delta\mathbf{m} = -\nabla_{\mathbf{m}}\phi(\mathbf{m}). \tag{6.23}$$

As we see next, solving the Hessian system requires some nontrivial amount of extra work. Nonetheless, assuming we have used steepest descent, we have the history of the optimization, and this can be used to approximate the Hessian. At this point we note that by assuming that the Hessian $\mathbf{H}(\mathbf{m})$ is invertible, we may approximate its inverse directly.

Using Taylor's expansion again, we have that

$$\nabla_{\mathbf{m}}\phi(\mathbf{m}+\delta\mathbf{m}) \approx \nabla_{\mathbf{m}}\phi(\mathbf{m}) + \mathbf{H}\delta\mathbf{m},$$

which implies that

$$\mathbf{H}(\mathbf{m})^{-1}(\nabla_{\mathbf{m}}\phi(\mathbf{m}+\delta\mathbf{m})-\nabla_{\mathbf{m}}\phi(\mathbf{m}))=\mathbf{H}(\mathbf{m})^{-1}\delta\nabla_{\mathbf{m}}\phi\approx\delta\mathbf{m},$$

where $\delta\nabla_{\mathbf{m}}\phi=\nabla_{\mathbf{m}}\phi(\mathbf{m}+\delta\mathbf{m})-\nabla_{\mathbf{m}}\phi(\mathbf{m})$. This equation is referred to as the quasi-Newton equation, and it implies that, given the gradient at two points $\mathbf{m}$ and $\mathbf{m}+\delta\mathbf{m}$, the (inverse) Hessian should obey this equation. Note that the Hessian has $n^2/2$ numbers while the gradient has only n numbers, and therefore this equation does not define the Hessian uniquely. A number of different methods have been proposed to approximate the Hessian. Maybe the most successful are the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method and its limited memory version L-BFGS. The BFGS method attempts to find the minimum Frobenius norm correction to the Hessian. Assuming that we have a Hessian $\mathbf{H}_k$, we are looking for a correction $\delta\mathbf{H}$ such that the quasi-Newton equation holds. It is possible to show that this correction is

$$\delta\mathbf{H}=\frac{(\delta\mathbf{m}^\top\delta\nabla_{\mathbf{m}}\phi+\delta\nabla_{\mathbf{m}}\phi^\top\mathbf{H}_k^{-1}\delta\nabla_{\mathbf{m}}\phi)\delta\mathbf{m}\delta\mathbf{m}^\top}{(\delta\mathbf{m}^\top\delta\nabla_{\mathbf{m}}\phi)^2}-\frac{\mathbf{H}_k^{-1}\delta\nabla_{\mathbf{m}}\phi\delta\mathbf{m}^\top+\delta\mathbf{m}\delta\nabla_{\mathbf{m}}\phi^\top\mathbf{H}_k^{-1}}{\delta\mathbf{m}^\top\delta\nabla_{\mathbf{m}}\phi}.$$

The BFGS technique requires the storage of the dense matrix $\mathbf{H}^{-1}$, which is not feasible for most inverse problems. In the popular L-BFGS method, the matrix $\mathbf{H}^{-1}$ is never stored. Only a small number of $\delta\mathbf{m}$'s and $\delta\nabla_{\mathbf{m}}\phi$'s are stored together with the initial Hessian $\mathbf{H}_0^{-1}$. Assuming that $\mathbf{H}_0^{-1}$ can be stored or applied efficiently, this yields a method that can be easily used for large scale problems. In "typical" implementation the number of vectors to be stored changes from a few (say, 3–5) to a few tens (20–30).

A very detailed algorithm is given in [51] and in [58]. Here we refer to these references for the exact algorithm and its implementation. We do, however, discuss a number of important aspects that make the method work better in our context. Maybe the most important aspect is the choice of the initial Hessian. While standard optimization algorithms tend to pick the identity as the initial Hessian, this approximation tends to perform poorly in our context. Instead, the obvious option is to pick the regularization matrix $\nabla_{\mathbf{m}}R(\mathbf{m}_0)$. When R is quadratic, this matrix is constant. For nonquadratic regularization such an approach does not work well, and other quasi-Newton methods may be needed to approximate the Hessian [33]. A second and important aspect is that there is no need to compute the inverse initial Hessian. Indeed, only matrix-vector operations are required, and therefore all we need is an efficient solver for the system

$$\mathbf{H}_0\mathbf{v}=\mathbf{w}$$

for arbitrary $\mathbf{v}$ and $\mathbf{w}$. This can be done using the conjugate gradient method with some simple preconditioning [8].

The attraction of BFGS and its limited memory version is that no Hessian is computed. The cost of each iteration is therefore similar to steepest descent; however, in my experience its convergence is much better than that of steepest descent. It is interesting to note that L-BFGS usually outperforms nonlinear conjugate gradient (NLCG) methods that have similar cost; therefore, we do not discuss the NLCG method further.

Even though L-BFGS can be an efficient solver, I have observed that for most electromagnetic inverse problems the performance of L-BFGS is not sufficient, as the number of iterations to convergence is very large. Therefore, a more involved technique is needed.

6.4.2 ▪ Gauss–Newton and inexact Gauss–Newton

The Gauss–Newton method is by far the method of choice for most nonlinear inverse problems. If implemented efficiently, it can be relatively fast and usually outperforms most other optimization methods. We thus discuss this method and its implementation to nonlinear inverse problems in detail.

There are a number of ways to present the method. Here we discuss two such presentations. First, assuming a least square optimization problem, we have that

$$\phi_d(\mathbf{m}) = \frac{1}{2}\mathbf{r}(\mathbf{m})^\top \Sigma_d^{-1}\mathbf{r}(\mathbf{m}),$$

where

$$\mathbf{r}(\mathbf{m}) = \mathbf{P}\mathbf{A}(\mathbf{m})^{-1}\mathbf{q} - \mathbf{d}.$$

The gradient is therefore

$$\nabla_{\mathbf{m}}\phi_d = \mathbf{J}(\mathbf{m})^\top \Sigma_d^{-1}\mathbf{r}(\mathbf{m}),$$

and the Hessian of ϕ_d is

$$\nabla^2_{\mathbf{m}}\phi_d = \mathbf{J}(\mathbf{m})^\top \Sigma_d^{-1}\mathbf{J}(\mathbf{m}) + \frac{\partial(\mathbf{J}(\mathbf{m})^\top \Sigma_d^{-1}\mathbf{r}(\mathbf{m}^{\text{fix}}))}{\partial \mathbf{m}} = \mathbf{J}(\mathbf{m})^\top \Sigma_d^{-1}\mathbf{J}(\mathbf{m}) + \mathbf{S}(\mathbf{m}).$$

The Hessian has two parts. The first is a symmetric positive semidefinite matrix $\mathbf{J}(\mathbf{m})^\top \Sigma_d^{-1}\mathbf{J}(\mathbf{m})$, and the second, $\mathbf{S}(\mathbf{m})$, depends on the curvature of the objective function and therefore may not add positiveness to the Hessian. Furthermore, it is possible to show that $\mathbf{S}$ is small in two cases—first, for problems with small residual at the solution, and second, for problems that are not very nonlinear. The Gauss–Newton approximation drops the matrix $\mathbf{S}$ from the Hessian, yielding the following approximation to the Hessian of ϕ_d:

$$\nabla^2_{\mathbf{m}}\phi_d \approx \mathbf{J}(\mathbf{m})^\top \Sigma_d^{-1}\mathbf{J}(\mathbf{m}). \tag{6.24}$$

A different way to obtain this result is by expanding the residual

$$\mathbf{r}(\mathbf{m} + \delta\mathbf{m}) \approx \mathbf{r}(\mathbf{m}) + \mathbf{J}(\mathbf{m})\delta\mathbf{m}.$$

Rewriting ϕ_d (assuming for simplicity that $\Sigma_d = \mathbf{I}$), we have that

$$\begin{aligned}\phi_d(\mathbf{m} + \delta\mathbf{m}) &\approx \frac{1}{2}(\mathbf{r}(\mathbf{m}) + \mathbf{J}(\mathbf{m})\delta\mathbf{m})^\top(\mathbf{r}(\mathbf{m}) + \mathbf{J}(\mathbf{m})\delta\mathbf{m}) \\ &= \frac{1}{2}\left(\mathbf{r}^\top\mathbf{r} + 2\delta\mathbf{m}^\top\mathbf{J}^\top\mathbf{r} + \delta\mathbf{m}^\top\mathbf{J}^\top\mathbf{J}\delta\mathbf{m}\right).\end{aligned}$$

This implies that the Hessian for the inverse problem with the linearized forward problem is $\mathbf{J}^\top\mathbf{J}$.

The Gauss–Newton method can be generalized with respect to nonquadratic misfits. Consider the case of robust misfit functions where

$$\phi_d(\mathbf{m}) = \mathbf{w}^\top \rho(\mathbf{r}(\mathbf{m})),$$

where $\rho(t)$ is a semipositive function implemented pointwise. Then, we have that

$$\nabla_{\mathbf{m}}\phi_d(\mathbf{m}) = \mathbf{J}^\top \text{diag}(\mathbf{w})\rho'(\mathbf{r}(\mathbf{m})),$$

where ρ' is the pointwise derivative of ρ,

$$\rho'(t) = \frac{d\rho(t)}{dt}.$$

Using the same arguments, we have that the equivalent Gauss–Newton approximation is

$$\nabla^2_{\mathbf{m}} \phi_d(\mathbf{m}) \approx \mathbf{J}^\top \mathrm{diag}(\mathbf{w} \odot \rho''(\mathbf{r}(\mathbf{m})))\mathbf{J}, \tag{6.25}$$

where ρ'' is the second derivative of ρ. Note that this expression is semipositive only as long as the penalty function used is convex. Thus, when using nonconvex robust norms, the Gauss–Newton method may be much less effective and may lose much of its appeal unless we treat the possible negative eigenvalues. One way to do that is the Gauss–Newton conjugate gradient (CG) method, which uses CG to solve the linear system. The CG method has a built-in mechanism to approximate the eigenvalues of the problem and therefore can be stopped if a negative eigenvalue is hit. For a much more elaborate discussion on the implementation of Newton CG and Gauss–Newton CG, see [51].

Given the approximation to the misfit, the (quadratic) Gauss–Newton system to be solved at each iteration can be written as

$$(\mathbf{J}(\mathbf{m})^\top \Sigma_d^{-1} \mathbf{J}(\mathbf{m}) + \alpha R''(\mathbf{m}))\delta\mathbf{m} = -\nabla_{\mathbf{m}} \phi(\mathbf{m}). \tag{6.26}$$

Before we discuss the solution of the system, we quickly review the whole Gauss–Newton algorithm.

Algorithm: Gauss–Newton

- Initialize $\mathbf{m} = \mathbf{m}_0$
- While not converge
  1. Compute $\phi(\mathbf{m})$ and $\nabla_{\mathbf{m}} \phi$ (two forward problems)
  2. Solve the system (6.26) ($2 \times N_{\mathrm{cg}}$ forward modeling; see next)
  3. Set $\mathbf{m} \leftarrow \mathbf{m} + \mu\delta\mathbf{m}$, where μ is set by a line search
  4. Test for convergence

The Gauss–Newton method involves the solution of the linear system (6.26). This is the main computational bottleneck of the method, and therefore we discuss it in detail. Recall that the sensitivity matrix has the form

$$\mathbf{J} = -\mathbf{P}\mathbf{A}^{-1}\mathbf{G},$$

where $\mathbf{P}$ and $\mathbf{G}$ are sparse and $\mathbf{A}$ is the forward modeling problem (in space and possibly time). Computing the sensitivity matrix is therefore not practical for most problems (unless subspace methods are used and the subspace is very small). On the other hand, computing $\mathbf{J}$ times a vector and $\mathbf{J}^\top$ times a vector is readily available, as it amounts to solving one forward problem and one adjoint problem. Thus, we are set to use iterative techniques for the solution of the problem. Since the linear system (6.26) is SPD, the method of choice is preconditioned conjugate gradient (PCG). The theoretical background of the

CG method is one of the more elegant developments in numerical linear algebra, and the reader is referred to [41, 64] for further discussion. The algorithm is summarized as follows.

Algorithm: Preconditioned conjugate gradient for $\mathbf{Ax} = \mathbf{b}$

- Choose a preconditioning matrix $\mathbf{M}$
- Initialize $\mathbf{x} = 0, \mathbf{r} = \mathbf{b}, \mathbf{z} = \mathbf{M}^{-1}\mathbf{r}, \mathbf{p} = \mathbf{z}$
- While not converge
  - $\mathbf{q} = \mathbf{Ap}$ (two forward problems)
  - $\alpha = \mathbf{r}^\top \mathbf{z}/(\mathbf{p}^\top \mathbf{q})$
  - $\mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{p}$
  - $\mathbf{r}_{\text{new}} \leftarrow \mathbf{r} - \alpha \mathbf{q}$
  - $\mathbf{z}_{\text{new}} = \mathbf{M}^{-1}\mathbf{r}_{\text{new}}$
  - $\beta = \mathbf{z}_{\text{new}}^\top(\mathbf{r}_{\text{new}} - \mathbf{r})/(\mathbf{z}^\top \mathbf{r})$
  - $\mathbf{z} = \mathbf{z}_{\text{new}}, \mathbf{r} = \mathbf{r}_{\text{new}}$
  - $\mathbf{p} \leftarrow \mathbf{z} + \beta \mathbf{p}$
  - Test for convergence

The cost of each Gauss–Newton iteration can be divided into two parts. First, the forward and adjoint problems are solved for the misfit and the gradient. Second, the PCG system is approximately solved to obtain a step. The main computational work within the PCG iteration is done in the matrix-vector product step, where we need to multiply $\mathbf{J}^\top \Sigma_d^{-1} \mathbf{J}$ times a vector. This requires one forward and one transpose solve and therefore is the main cost of our algorithm. This cost can be reduced in a number of ways. First, if the forward modeling matrix is factorized, then the solution of every new system is, in essence, free, as most of the work is invested in the factorization step. Therefore, if the forward modeling matrix is not too large and can be factorized, then doing so may pay off. Note that the factorization is used for the solution of the forward problem, the transpose problem (for the computation of the gradient), and finally again for every PCG step.

A second way to reduce the cost of the system is to solve the system only approximately. Indeed, it turns out that linear convergence (which is not saying much) is guaranteed as long we reduce the residual of the Gauss–Newton system (6.26). For an elaborate discussion on inexact Newton methods, see [58, 51]. Therefore, it is common to use a rather high tolerance as a stopping criterion for the PCG method. In my experience, 10^{-2} is a sufficiently low tolerance and gives satisfactory results for many problems.

If one uses iterative methods, then potential savings can be obtained by solving the forward problem to a large tolerance at each iteration. While the theoretical aspects of this approach are rather sketchy [7, 31], practical experience suggests that the accuracy of the linear inner solver can be relaxed to, say, 10^{-4}. The main problem with this approach is that we no longer work with the matrix $\mathbf{J}$ but rather with an approximation $\widehat{\mathbf{J}} \approx \mathbf{J}$ and with a different approximation $\widehat{\mathbf{J}}^\top \approx \mathbf{J}^\top$. Furthermore, these approximations change from one iteration of PCG to the next. Under these conditions PCG may break, and therefore care must be taken to note that reasonable approximations are obtained. In some cases, the inner tolerance needs to be tightened in order to obtain a reasonable step.

One last point to be discussed regarding the Gauss–Newton method is the preconditioning of the system, that is, the choice of $\mathbf{M}$ in the PCG algorithm. One obvious option is to use the regularization matrix $\mathbf{M} = \nabla^2_{\mathbf{m}} R(\mathbf{m})$. This preconditioner gives reasonable results in many applications; however, in some cases, this preconditioner tends to be slow.

A different preconditioner can be built throughout the optimization process that captures some of the essence of $\mathbf{J}^\top\mathbf{J}$. Recall from the previous section the L-BFGS method. This method keeps a number of the previous gradients and model updates in order to approximate the Hessian. Thus, we have found that, in many cases, using the L-BFGS Hessian can give a better preconditioner than the regularization alone.

6.4.3 ▪ Bound constraints

We now briefly discuss the incorporation of bound constraints (6.20) into the inversion. Again, we refer the reader to [58, 51] for more details. One of the most important aspects of bound constraints is that the conditions for a minimum are

$$0 = \nabla_{\mathbf{m}}\phi(\mathbf{m}) \odot (\mathbf{m} - \mathbf{m}_{\text{low}}) \odot (\mathbf{m} - \mathbf{m}_{\text{high}}).$$

That is, either the gradient is zero or the model is on the bounds. Assuming we have obtained the solution $\widehat{\mathbf{m}}$, the set of indices that are equal to the bounds is referred to as the active set, and the set of indices where $\mathbf{m}_{\text{low}} < \mathbf{m} < \mathbf{m}_{\text{high}}$ is referred to as the inactive set. Note that if we somehow knew the active set, then we could solve an unconstrained optimization problem only for the inactive set. The "trick" is to identify the active set quickly.

We now discuss two methods to impose bound constraints by using projection. We first define the projection operator $\mathscr{P}$:

$$\mathscr{P}(t) = \begin{cases} t & \text{if} \quad t_{\text{low}} \le t \le t_{\text{high}}, \\ t_{\text{low}} & \text{if} \quad t < t_{\text{low}}, \\ t_{\text{high}} & \text{if} \quad t > t_{\text{high}}. \end{cases}$$

Using the projection operator, we define the projected steepest descent method. The method is identical to the steepest descent method, with the only modification being that the model update is obtained by

$$\mathbf{m} \leftarrow \mathscr{P}(\mathbf{m} - \mu\nabla_{\mathbf{m}}\phi(\mathbf{m})).$$

This method is extremely simple to implement and works well, that is, as long as the steepest descent method works well. It tends to identify the active set of the problem rather quickly and then continue to reduce the gradient to convergence. However, projected steepest descent is an inappropriate method for most inverse problems, where steepest descent converges rather slowly.

A simple, similar, extension to the Gauss–Newton method does not work (see [51] for an example); however, using a rather simple correction, the Gauss–Newton method can be made convergent. To this end, the variables are divided into the active set variables and the inactive set variables. A scaled steepest descent step is taken on the active variables (that lie on the bounds), and a Gauss–Newton direction is taken on the inactive set.

To this end, define the matrix $\mathbf{I}_i$ as an identity matrix with the active set indices set to 0. Then the step is computed by two different processes that are combined. First, we (approximately) solve the system

$$\mathbf{I}_i(\mathbf{J}(\mathbf{m})^\top\Sigma_d^{-1}\mathbf{J}(\mathbf{m}) + \alpha R''(\mathbf{m}))\mathbf{I}_i\delta\mathbf{m}_i = -\mathbf{I}_i\nabla_{\mathbf{m}}\phi(\mathbf{m}). \tag{6.27}$$

The system (6.27) can be solved again using PCG with the minor modification of adding the matrix $\mathbf{I}_i$ to the implicit calculation of matrix-vector products. Second, we compute the steepest descent direction

$$\delta \mathbf{m}_a = (\mathbf{I}_i - \mathbf{I})\nabla_{\mathbf{m}}\phi(\mathbf{m})$$

and combine the components to get

$$\delta \mathbf{m} = \delta \mathbf{m}_i + \eta \delta \mathbf{m}_a. \tag{6.28}$$

In my experience, the parameter η that is not discussed in most other work is crucial for a reasonable performance of the code. A reasonable choice of the parameter η is key to the successful implementation of the algorithm. The steepest descent method does not have natural scaling built into it, while the Gauss–Newton method does. The goal of η is to scale the whole step based on the Gauss–Newton scaling. A reasonable scaling is to use

$$\eta = \frac{||\delta \mathbf{m}_i||_\infty}{||\delta \mathbf{m}_a||_\infty},$$

ensuring that the entries of $\delta \mathbf{m}_a$ are not larger than the entries of $\delta \mathbf{m}_i$. If this is not done, we may reduce the step based on its steepest descent entries and obtain convergence speed that is as slow as steepest descent. Finally, an update is computed by using a projection

$$\mathbf{m} \leftarrow \mathcal{P}(\mathbf{m} + \mu \delta \mathbf{m}).$$

The algorithm for bound constraints is as follows.

Algorithm: Projected Gauss–Newton

- Initialize $\mathbf{m} = \mathbf{m}_0$
- While not converge
  - Compute $\phi(\mathbf{m})$ and $\nabla_{\mathbf{m}}\phi$ (two forward problems)
  - Identify inactive indices i
  - Solve the system (6.27) ($2 \times N_{\text{cg}}$ forward modeling)
  - Set up the step (6.28)
  - Set $\mathbf{m} \leftarrow \mathcal{P}(\mathbf{m} + \mu \delta \mathbf{m})$, where μ is set by a line search
  - Test for convergence

The attraction of projection methods (Gauss–Newton and steepest descent) is that they require a rather minor modification to the unconstrained version; however, surprisingly enough, these small changes work rather well for many inverse problems.

6.4.4 ▪ A comment about direct search methods

The techniques discussed above are based on gradient computation. These techniques work well when convergence to a local minimum is sufficient. In some cases the problem contains many local minima, and an attempt to obtain the global minima requires other techniques.

Direct search methods are geared toward achieving this goal. Most are based on either some random search direction or on a smart grid based search. It is important to realize that such methods are highly ineffective when the number of parameters is larger than (very generously) a few tens. Thus, these techniques are useful only for subspace and parametric methods, assuming that the number of parameters to be estimated is small.

Direct search methods require tens of thousands of forward modeling computations and therefore tend to be much less competitive compared with gradient based methods. In my experience, these methods are not very useful for large scale EM problems.

There are a few exceptions, especially when the model is parametric. Parametric models can be rather small and can give a very good initial guess to larger models and to gradient based methods. Using a direct search method in order to initialize the model by finding, for example, a background conductivity and a trend can be very useful in initializing more advanced inversions.

In some other cases some of the system parameters are unknown. For example, in many time domain legacy data sets the initial shut-off time of the current is unknown and therefore needs to be estimated. In these cases using a direct search method with parametric models can help in calibrating the system.

6.4.5 ▪ Optimization over complex variables

For frequency domain electromagnetics, the fields are complex, and therefore, as we have seen in the previous chapter, the sensitivities are complex as well. Care must be taken when computing the gradient and Hessian. The safe thing to do is to develop the gradients and Hessians, separating the fields into real and imaginary fields, that is, working with real numbers when deriving the derivatives. After the derivatives are computed one can recombine them into complex quantities for ease of computation. The reason we suggest using complex variables at all depends on the computational platform. Many scientific computing languages perform complex arithmetic much faster than an equivalent real arithmetic. This makes the computation in complex arithmetic preferable.

One can verify that for an L_2 misfit, where

$$\phi_d(\mathbf{m}) = \frac{1}{2}\mathbf{r}(\mathbf{m})^*\mathbf{r}(\mathbf{m})$$

with $\mathbf{r}$ maps from a real vector $\mathbf{m}$ to a complex vector, that is,

$$\mathbf{r} : \mathbb{R}^n \to \mathbb{C}^k,$$

the gradient of the misfit function is given by

$$\nabla\phi_d(\mathbf{m}) = \mathbf{Re}(\mathbf{J}_c^*\mathbf{r}(\mathbf{m})),$$

where $\mathbf{J}_c$ is the complex sensitivities and $*$ stands for the complex conjugate transpose. The Gauss–Newton approximation to the Hessian of the misfit is simply

$$\mathbf{H} \approx \mathbf{Re}(\mathbf{J}_c^*\mathbf{J}_c) = \mathbf{Re}(\mathbf{J}_c)^\top\mathbf{Re}(\mathbf{J}_c) + \mathbf{Im}(\mathbf{J}_c)^\top\mathbf{Im}(\mathbf{J}_c).$$

When nontrivial data covariance exists, using an equivalent complex system is a bit more involved. To that end, let Σ_{dr} be the covariance associated with the real part of the data, and let Σ_{di} be the covariance associated with the imaginary part of the data. The misfit to be considered is

$$\phi_d = \frac{1}{2}\mathbf{Re}(\mathbf{r})^\top\Sigma_{dr}^{-1}\mathbf{Re}(\mathbf{r}) + \frac{1}{2}\mathbf{Im}(\mathbf{r})^\top\Sigma_{di}^{-1}\mathbf{Im}(\mathbf{r}).$$

Differentiating with respect to real and imaginary parts, we obtain that

$$\nabla_{\mathbf{Re(r)}}\phi_d = \Sigma_{dr}^{-1}\mathbf{Re(r)} \quad \text{and} \quad \nabla_{\mathbf{Im(r)}}\phi_d = \Sigma_{di}^{-1}\mathbf{Im(r)},$$

which suggests that we can write the gradient with respect to $\mathbf{r}$ as

$$\nabla_{\mathbf{r}}\phi_d = \Sigma_{dr}\mathbf{Re(r)} + i\Sigma_{di}\mathbf{Im(r)}.$$

Using the chain rule, we obtain

$$\nabla_{\mathbf{m}}\phi_d = \mathbf{Re}\left(\mathbf{J}_c^*(\Sigma_{dr}\mathbf{Re(r)} + i\Sigma_{di}\mathbf{Im(r)})\right),$$

and the Gauss–Newton approximated to the Hessian is

$$\mathbf{H}_w \approx (\mathbf{Re}(\mathbf{J}_c))^\top \Sigma_{dr}\mathbf{Re}(\mathbf{J}_c) + (\mathbf{Im}(\mathbf{J}_c))^\top \Sigma_{di}\mathbf{Im}(\mathbf{J}_c).$$

The above formula can be expressed in a nicer form. Defining a complex product as

$$\mathbf{A} \circledast \mathbf{b} = \mathbf{Re(A)Re(b)} + i\mathbf{Im(A)Im(b)},$$

the complex covariance matrix

$$\Sigma_d = \Sigma_{dr} + i\Sigma_{di}.$$

The above is nothing but

$$\mathbf{H}_w = \mathbf{Re}\left(\mathbf{J}_c^* \circledast (\Sigma_d \circledast \mathbf{J}_c)\right).$$

To implement an efficient matrix-vector product for the gradient and Hessian, care must be taken. Consider the product of the weighted Hessian $\mathbf{H}_w$ and a real vector $\mathbf{s}$ (note that since we assume that $\mathbf{m}$ is real, there is no point in considering the product with a complex vector). The product is done in three stages and is summarized in the following algorithm.

Algorithm: Weighted Hessian product for complex variables

- Compute the complex variable $\mathbf{z} = \mathbf{Js}$
- Set $\mathbf{y} = \Sigma_{dr}\mathbf{Re(z)} + i\Sigma_{di}\mathbf{Im(z)}$
- Set $\mathbf{Hs} = \mathbf{Re}(\mathbf{J}^*\mathbf{y})$

6.4.6 ▪ Implementation

We now show our implementation of the projected Gauss–Newton method and discuss a number of points that are important in our code and can be different from generic optimization algorithms.

First, note that the code is rather generic. It calls the method solveForwardProblem and requires a model and parameter structure. It also requires the generic codes getJMatVec and getJTmatVec for the sensitivity times a vector and the transpose of the sensitivity times a vector. A few other generic codes are the calculation of misfit done in misfun and regularization done in regfun. The code is based on the work presented in [29, 76] that combines the CG direction within the optimization code. While the code is less modular, it has the advantage of making it easy to change parameters within the CG method (such as accuracy of the forward problem solver).

Most of the computational work is done in the codes that solve the forward problem and get the sensitivity times a vector and the transpose of the sensitivity times a vector. It is therefore desired to have control over these steps, in terms of, for example, setting the tolerances and allocating the storage needed. It is possible to change the parameter structure such that the solution to the forward problem when computing sensitivities times a vector is different for the forward and gradient calculation from the solution when using it in the calculation of the Hessian times a vector. This can save significant amounts of work; the reader is encouraged to play with the code.

Another important piece of this code is the sigfun code. This code takes the vector $\mathbf{m}$ and uses it to return a conductivity $\boldsymbol{\sigma}$ and its derivative. The conductivity $\boldsymbol{\sigma}$ is then used for the forward problem, and the derivative is used within the gradient and Hessian. Note that in particular, if we define $\boldsymbol{\sigma} = \mathbf{Vm}$ with $\mathbf{V}$ a "tall skinny" matrix and set the regularization parameter to 0, then the Gauss–Newton code is optimizing the model over a small subspace.

In the code below we use two termination criteria. First, an obvious criterion is that we exceed the number of predetermined iterations. A second criterion looks at the perturbation to the model. We terminate when the update to the model is small. The advantage of this criterion is that it it less sensitive to scaling of the gradient; however, it has one major disadvantage. If for some reason the Gauss–Newton direction is poor (and this can happen for very nonlinear problems), the code will terminate without reaching the desired solution. Other, more sophisticated, termination criteria can be used, and we refer the reader to [20] for further details. In my experience, if the line search fails or the step size is very small without solving the problem, then one should replace the optimization framework above with a more robust one such as a trust region method or a continuation method, both of which deal better with highly nonlinear problems [58].

```
function [mc,d] = projGNCG(mc,param)
%[mc,dpre] = projNewtonCG(mc,param)
%

maxIter     = param.maxIter;     %  Max. no. iterations.
maxCG       = param.maxCGiter;   %  Max cg iters.
cgTol       = param.cgTol;       %  CG stopping tolerance.
stepTol     = param.stepTol;     %  Step norm stopping tol.

low   = param.bounds.low;
high  = param.bounds.high;

regpar = param.regPar;
Wd     = param.Wd;

%-------------------------------------------------------------------
%  Initialization.
%-------------------------------------------------------------------
maxStepSize = param.maxStep;

Active = [mc<=low | mc>=high];  % Compute active set

%% evaluate function and derivatives
[sig,dsig] = param.sigFun(mc,param); Jxm = sdiag(dsig);
[d,e]      = param.solveForwardProblem(sig,param);
r          = Wd *(d(:) - param.dobs(:));
[F,dF]     = param.misFun(r,param);
dF         = Jxm'*param.JTmatVec(Wd'*dF,e,sig,param);

% compute model objective function
[R,dR,d2R] = param.regFun(mc,param);

% objective function
Jc  = F  + regpar*R;
gc  = dF + regpar*dR;

F0 = F; J0 = Jc;
```

```
%%  Outer iteration.

  iter = 0;
  outerFlag = 0;

  while outerFlag == 0;

    iter = iter + 1;
    fprintf('%3d.0    %3.2e   %3.2e   %3.2e   %3.2e   %3d\n',...
             iter,F/F0,R,regpar,Jc/J0,nnz(Active));

    %  Set up Hessian and preconditioner.
    Jx       = @(x) Wd*param.JmatVec(Jxm*x,e,sig,param);
    JTx      = @(x) Jxm*param.JTmatVec(Wd*x,e,sig,param);
    Hs       = @(x) JTx(Jx(x)) + param.regPar*(d2R*(x));
    Precond  = @(x) tril(d2R)\(diag(d2R).*(triu(d2R)\x));

    resid = (1-Active) .* gc;
    delm = projPCG(Hs,-resid,Active,Precond,cgTol,maxCG);
    % scale step for the safe size
    if max(abs(delm)) > maxStepSize, delm = delm/max(abs(delm))*maxStepSize; end

    % take gradient direction in the active cells
    ga = -gc(mc == low);
    if max(abs(ga)) > max(abs(delm)), ga = ga/max(abs(ga))*max(abs(delm)); end
    delm(mc == low) = ga;
    ga = -gc(mc == high);
    if max(abs(ga)) > max(abs(delm)), ga = ga/max(abs(ga))*max(abs(delm)); end
    delm(mc == high) = ga;
```

```
    %% Begin line search
    muLS = 1; lsIter = 1;
    while 1,
        mt = mc + muLS*delm;
        mt(mt<low) = low;
        mt(mt>high) = high;
        %% evaluate function and derivatives
        [sig,dsig] = param.sigFun(mt,param); Jxm = sdiag(dsig);
        [d,e]      = param.solveForwardProblem(sig,param);
        r          = Wd *(d(:) - param.dobs(:));
        [F,dF]     = param.misFun(r,param);

        % compute model objective function
        [R,dR] = param.regFun(mt,param);

        % objective function
        Jt  = F  + regpar*R;
        %%

        fprintf('%3d.%d   %3.2e   %3.2e   %3.2e   %3.2e\n',iter,lsIter,F/F0,R,regpar,Jt/J0);

        if Jt < Jc
            break;
        end
        muLS = muLS/2; lsIter = lsIter+1;
        if lsIter > 6
            disp('LSB '); keyboard
            return
        end
    end
    dF   = Jxm'*param.JTmatVec(Wd'*dF,e,sig,param);
    mnew = mt;
    Jc   = Jt;
    gc = dF + regpar*dR;
    %% End Line search
```

```
    %% Check for termination
    stepnorm = norm(mnew-mc);
    mc = mnew;
    Active = [mc<=low | mc>=high];

    %  Check stopping criteria for outer iteration.

    if iter >= maxIter
      fprintf(' \n\n\n  *** Max iterations emceeded ***\n\n\n\n');
      return
    elseif stepnorm < stepTol * norm(mc(:))
      fprintf('  \n\n\n    *** Step norm tolerance met ***\n');
      return
    end

  end
```

The code uses the PCG method implemented below.

```
function delm = projPCG(Hs,resid,Active,Precond,cgTol,maxCG)
% delm = projPCG(H,resid,Active)
% CG with projection to the active set

cgiter = 0;
normResid0 = norm(resid);
delm = zeros(length(resid),1);

while 1
    cgiter = cgiter + 1;
    dc = (1-Active).*(Precond(resid));
    rd = resid'*dc;

    %  Compute conjugate direction pc.
    if cgiter == 1,
      pc = dc;
    else
      betak = rd / rdlast;
      pc = dc + betak * pc;
    end

    %  Form product Hessian*pc.
    Hp = Hs(pc);
    Hp = (1-Active).*Hp;

    %  Update delm and residual.
    alphak = rd / (pc'*Hp);
    delm = delm + alphak*pc;
    resid = resid - alphak*Hp;
    rdlast = rd;

    if norm(resid)/normResid0 <= cgTol || cgiter == maxCG
       break;
    end
end
```

While the algorithms presented above are rather general, a few aspects are different when considering a parameter estimation problem. In such problems we are estimating both the electric fields $\mathbf{e}$ and the model $\mathbf{m}$ assuming $\mathbf{e} = \mathbf{e}(\mathbf{m})$. Consider now the evaluation of the data. In this case we first solve Maxwell's equation for $\mathbf{e}$ and then project $\mathbf{e}$ onto the data. For the computation of the gradient we need the fields, and therefore keeping the fields can save a forward modeling step when computing the gradient. Thus when considering steps (1) and (3) within the Gauss–Newton method or steps (1) and (2) within the preconditioned steepest descent, it is important to note that by storing $\mathbf{e}$ we decrease the number of forward modeling to be done in each iteration. In particular, when computing the step length in step (3) of the Gaus–Newton method (step (2) in steepest descent), one computes the fields that can be used again in step (1) for the calculation of the gradient, where the computation of the objective function value is trivial. These fields are used again within the Gauss–Newton method in step (2) within the PCG algorithm to

compute the sensitivity and its transpose times a vector. Thus, storing the fields reduces the cost of the solution significantly.

Nonetheless, in some cases, it is impossible to store the fields. If the number of sources and frequencies or times is large, then memory requirements can be too large, and one has to recompute quantities in order to be able to handle such problems. In these cases, the fields are not stored and the gradient that corresponds to the source and frequency is computed for every source/frequency immediately after each forward problem is solved, and the fields are overwritten for every source/frequency. In this case step (1) requires two forward modeling (rather than one) to re-evaluate the fields for each gradient. For the Gauss–Newton method having all the fields is essential since the product of sensitivities and its transpose times a vector requires the fields. If these cannot be stored, the Gauss–Newton method becomes too expensive unless some approximations are used. We explore such approximations in the next chapter.

6.4.7 ▪ Regularization parameter selection

When solving the optimization problem the regularization parameter is assumed to be known. We consider again the optimization problem and note that

$$\mathbf{m}(\alpha) = \arg\min \phi_d(\mathbf{m}) + \alpha R(\mathbf{m}).$$

Assume we start with a very large α, yielding $R(\mathbf{m}(\alpha)) \approx 0$ and $\mathbf{m}(\alpha) \approx \mathbf{m}_{\text{ref}}$. Our goal is to choose α such that $\phi_d(\mathbf{m}(\alpha)) \approx \tau$, where τ is some given tolerance. This yields an equation for α of the form

$$\phi_d(\mathbf{m}(\alpha)) - \tau = 0.$$

The equation can be solved using a secant method. At each step one computes the solution to the optimization problem for a different α. This is an expensive process since each iteration requires the solution of a nonlinear inverse problem. The solution of each problem can be obtained rather quickly if we start from the solution of the previous problem and do not take overly large steps in α. A commonly used technique is to "cool" α slowly, say, in a factor of 1.5 at each iteration, until the desired misfit is obtained.

While the above approach to obtain a reasonable regularization parameter is often discussed, I have found that it does not correspond to many practical situations. In fact, I have yet to see a field data set that is associated with a well-defined τ. In practice, one chooses a number of values for τ and recovers a few models. The models and the residuals are then interpreted to see if the data were not overfitted and the models have features that "make sense." Thus, rather than having a single model, we aim to obtain a sequence of models which gradually fit the data. This leads to different regularization techniques that are discussed next.

6.5 ▪ Inversion through inverse scale space methods

As previously discussed most EM inverse problems do not come with a well-known misfit, and in practice one is interested in a path from the reference model to a model that fits the data well. In many cases practitioners would like to examine the models that gradually fit the data in order to see whether some features are common to all models and whether some features appear only when we start to fit the data very well. In the previous section we discussed how this can be achieved by solving a number of optimization problems for different regularization parameters. In this section we discuss the inverse scale space and

the iterated Tikhonov regularization [40, 25] as well as "looser" techniques that lead to similar results [5].

To better understand and motivate the technique, assume again the linear problem

$$\mathbf{J}\mathbf{m} + \epsilon = \mathbf{d},$$

and for simplicity assume that $\mathbf{L} = \mathbf{I}$. The L_2 solution is given by

$$\mathbf{J}^\top(\mathbf{J}\mathbf{m} - \mathbf{d}) + \alpha(\mathbf{m} - \mathbf{m}_{\text{ref}}) = 0,$$

or, introducing $\delta\tau = \alpha$, we rewrite the solution as

$$\frac{\mathbf{m} - \mathbf{m}_{\text{ref}}}{\delta\tau} = \mathbf{J}^\top(\mathbf{d} - \mathbf{J}\mathbf{m}).$$

This equation should remind the reader of the differential equation

$$\dot{\mathbf{m}} = \mathbf{J}^\top(\mathbf{d} - \mathbf{J}\mathbf{m}) \quad \mathbf{m}_0 = \mathbf{m}_{\text{ref}}. \tag{6.29}$$

The solution to this ODE is

$$\mathbf{m}(\tau) = \exp(-\tau\mathbf{J}^\top\mathbf{J})\mathbf{m}_{\text{ref}} + \int_0^\tau \exp((s-\tau)\mathbf{J}^\top\mathbf{J})\mathbf{J}^\top\mathbf{d}\, ds,$$

which can be integrated analytically to obtain

$$\begin{aligned}\mathbf{m}(\tau) &= (\mathbf{I} - \exp(-\tau\mathbf{J}^\top\mathbf{J}))(\mathbf{J}\mathbf{J}^\top)^{-1}\mathbf{J}^\top\mathbf{d} + \exp(-\tau\mathbf{J}^\top\mathbf{J})\mathbf{m}_{\text{ref}} \\ &= (\mathbf{I} - \exp(-\tau\mathbf{J}^\top\mathbf{J}))\mathbf{J}^\dagger\mathbf{d} + \exp(-\tau\mathbf{J}^\top\mathbf{J})\mathbf{m}_{\text{ref}}.\end{aligned}$$

This equation describes a smooth path (in $\mathbf{m}(\tau)$) where for $\tau = 0$ one obtains $\mathbf{m} = \mathbf{m}_{\text{ref}}$ and when $\tau \to \infty$, $\mathbf{m} = \mathbf{J}^\dagger\mathbf{d}$ and the model fits the data as much as possible. This path is the inverse scale space path. The idea is that different scale features in the model appear as we move forward in time.

A number of methods are closely related to this approach, and they are related to the way we approximate the differential equation (6.29). For example, in the Landweber iteration [24], the model is updated by

$$\mathbf{m}_{k+1} = \mathbf{m}_k + \delta\tau\mathbf{J}^\top(\mathbf{d} - \mathbf{J}\mathbf{m}_k),$$

which can be thought of as a forward Euler solution to the problem. The iterated Tikhonov regularization [40] reads

$$\mathbf{m}_{k+1} = \mathbf{m}_k + (\mathbf{J}^\top\mathbf{J} + \delta\tau\mathbf{I})^{-1}\mathbf{J}^\top(\mathbf{d} - \mathbf{J}\mathbf{m}_k),$$

which can be thought of as a semi-implicit backward Euler method.

When a nontrivial regularization operator is introduced the iteration is changed to

$$\mathbf{m}_{k+1} = \mathbf{m}_k + (\mathbf{J}^\top\mathbf{J} + \delta\tau\nabla_{\mathbf{m}}^2 R(\mathbf{m}))^{-1}\mathbf{J}^\top(\mathbf{d} - \mathbf{J}\mathbf{m}_k). \tag{6.30}$$

In my experience, iterated Tikhonov and scale space methods give equivalent results (in terms of quality) to Tikhonov style regularization. Care must be taken that $\delta\tau$ is not too small. In this case the data overfits quickly, and the results are noisy. Nonetheless, the algorithm is less sensitive to the choice of large $\delta\tau$, and indeed one can iterate with a large $\delta\tau$ to achieve results similar to those obtained by iterating with a small $\delta\tau$.

6.6 ▪ Exercises

1. Prove (6.7). Hint: Use the identity $\mathbf{x}^\top \mathbf{A}\mathbf{x} = \mathrm{trace}(\mathbf{A}\mathbf{x}\mathbf{x}^\top)$.

2. Consider the optimization problem

$$\begin{aligned} &\min ||\mathbf{L}\mathbf{m}||_1 \\ &\text{s.t. } \mathbf{A}\mathbf{m} = \mathbf{b}, \end{aligned}$$

where $\mathbf{L}$ is an invertible matrix. Show that by writing $\mathbf{L}\mathbf{m} = \mathbf{p} - \mathbf{q}$ with $\mathbf{p}, \mathbf{q} \geq 0$ the problem can be cast as a linear programming problem.

3. Let $\mathscr{A} : \mathscr{H} \to \mathbb{R}^2$ be the sampling operator

$$\mathscr{A}_j m(x) = \int_0^1 \delta(x - x_j) m(x)\, dx, \quad j = 1,2, \quad 0 < x_j < 1.$$

Let $m(x)$ be any integrable function in $[0,1]$.

   (a) Find the *analytic* solution(s) to the problem

$$\min \int_0^1 |m_x| dx \quad \text{s.t. } \mathscr{A} m = b.$$

   (b) Find the *analytic* solution to the problem

$$\min \int_0^1 \frac{1}{2} |m_x|^2 dx \quad \text{s.t. } \mathscr{A} m = b.$$

   (c) Comment on the solutions.

4. Use the optimization code in this chapter and the direct current sensitivity code in the previous chapter and write your own DCR inversion code. Test it on a simple example, and comment on your experience.

Chapter 7
Large Scale Problems and Problems with Many Sources

With the help of recent years' technological improvements, large amounts of data over very large domains are being collected. At the same time, improvements in computational hardware enable computing with some parallel system. In this chapter, we explore the use of parallel computing in order to solve large scale problems with many sources in frequency and time domains. Since the topics in this chapter are still under investigation, we do not dive into the same level of detail as in previous chapters, and the interested reader should consult the references for a more complete understanding of the topics presented here.

7.1 ▪ Big data and big models

In the previous chapters, we considered solving Maxwell's equation with a single source or very few sources. In many important applications, the number of sources is very large. For example, in airborne electromagnetic surveys the transmitter is mounted on a flying platform of some sort, and a time dependent magnetic field is transmitted and received every 10–50 meters. Such a platform covers huge areas—thousands of square kilometers. A similar experiment is conducted in marine electromagnetics. In this case, the source is an electric dipole being towed by a ship, and the receivers are either towed behind or placed on the ocean floor (see Figure 7.1).

These kinds of surveys contain a huge amount of data. Indeed, regional airborne surveys may contain millions of source locations and a few tens of measurements in time. Marine electromagnetic surveys can be much larger. If the ship is towing n_d receivers that record in n_f frequencies and transmit in n_s source locations, then the number of data is $n_s \times n_d \times n_f$. For realistic surveys this number can easily reach 10^{10} complex-valued data points. Such problems have been coined as "big data" problems. There are a number of challenges when solving problems that arise from these types of experiments. First, even before we dive into complicated calculations, the question of data quality and data processing can be a main issue. For small scale problems, it is possible to examine each data point, detect the unreasonable data points, and delete them from the inversion. For large scale problems this is impossible. Data management, plotting, and visualizing in a way that makes sense can become an issue in itself. Here we do not discuss these issues but rather point out that if one attempts to solve such problems, then these topics have to be dealt with, and this is one of the aspects of "big data."

Our concern here is dedicated to the second challenge of dealing with big data, mod-

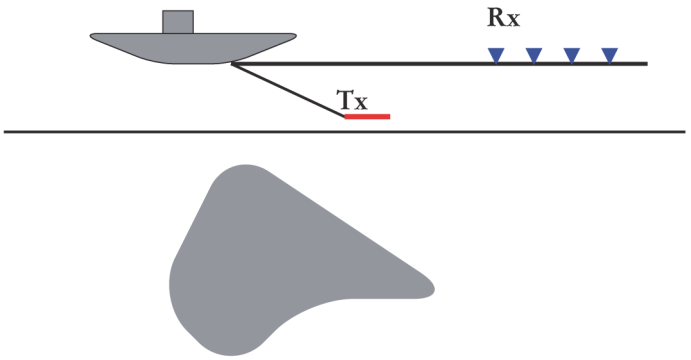

Figure 7.1. *A sketch of a marine electromagnetic survey where a moving boat tows a transmitter and receivers as measurements are made.*

eling and inversion. Clearly, big data requires big models. Working with big models is much more challenging than working with big data. Here, we also require solving a large number of systems of PDEs over very large domains in order to fit the data. Let us consider the problem in the frequency domain (the extension to the time domain is straightforward). The misfit for this problem can be written as

$$\phi_d(\mathbf{m}) = \frac{1}{2n_d} \sum_{i,j} ||\mathbf{W}_{ij}(\mathbf{P}_{i,j}\mathbf{A}_{ij}(\mathbf{m}, \omega_j)^{-1}\mathbf{q}_{ij} - \mathbf{d}_{ij})||^2. \tag{7.1}$$

Here $\mathbf{d}_{ij}$ is a vector that corresponds to the data from the ith source and jth frequency, and similarly $\mathbf{A}_{ij}(\mathbf{m}, \omega_j)$, $\mathbf{P}_{ij}$, $\mathbf{q}_{ij}$, and $\mathbf{W}_{ij}$ are the forward modeling matrix, the data projection matrix, the source vectors, and the (square root of the) inverse data covariance matrix for the ith source and jth frequency. Note that this approach is rather general and can include different sources and receiver types for each frequency. The inversion of different data types is sometimes referred to as cooperative inversion, and it is very common in many practical settings, where, for example, direct current and EM data are measured in the same domain. In practice, most large scale problems share the same source and receiver types for all frequencies. Note that we design the misfit in a way that the forward modeling matrix, $\mathbf{A}$ (the discretization of Maxwell's equations), can be different for each source and frequency. As we see next, this is an important point for solving such problems.

7.2 ▪ Mesh decoupling

To numerically solve the inverse problem, we need to compute the misfit. This is our first big challenge. The misfit implies solving a linear system for every source and time/frequency. If the number of sources is large, then one obtains a huge number of linear systems to be solved. One simple way to solve this problem is to generate a large mesh where all sources and frequencies can be represented. This has the advantage of small overhead and simplicity in programming. Once the mesh has been generated, we construct the forward modeling matrix for each frequency and proceed as in the previous chapters. In particular, one main advantage is that if direct methods are used for the solution of the forward problem, then a single matrix factorization is needed in order to compute all the fields from all the sources for a given frequency. For time domain problems, we can get

away with very few factorizations by using mostly equal time steps. However, this approach has one major disadvantage. The discretization of different forward problems on the same mesh implies that the mesh has to be fine where there are sources and receivers, and the cells have to be sufficiently small to capture the behavior of the fields at the highest frequency. For example, if we work on frequencies that range from 1Hz to 10^4Hz in a media with the conductivity of 10^{-2}S/m, then skin-depth calculations imply that the smallest cell for modeling the largest frequency must be no bigger than 15m, while we can choose cells size of 100m or more to work with the 1Hz. Using small cells for all frequencies is wasteful, especially if the number of sources is large. A similar issue arises when using sources with a small support on a large domain. Maybe the most obvious example is the airborne survey, where one collects many data points from a moving loop. In this case, accurate forward modeling requires fine discretization only close to the source. Economical forward modeling typically uses a coarse discretization far away from the source. Nonetheless, using a single mesh that works well for all sources requires a fine mesh everywhere.

This discussion leads to the concept of mesh decoupling. Rather than having a single mesh for the forward problem, we use different meshes for the forward modeling and the inversion. Assume that we have n_s sources and n_ω frequencies. Then, for each problem we build its own mesh $\mathcal{S}_{ij}$, $i = 1,\ldots,n_s$, $j = 1,\ldots,n_\omega$. Each mesh is designed specifically for its forward problem; that is, we adapt the mesh to the source-receiver-frequency configuration. A substantial savings can be obtained by this construction; however, using this approach, a number of difficult problems needs to be addressed. In particular, Maxwell's equations need to be discretized on different meshes. While at first this looks trivial, it leads to many practical complications. The main one is the discretization of the operators on the different meshes. To be more specific, there are two types of operators that are needed for the solution of the forward problem. The first type is differential operators that do not contain any information about material properties. These are the **div**, the **grad**, and the **curl** operators. Since they do not contain any information about the conductivity, one can simply rediscretize them on any sufficiently fine mesh. Thus, assuming that the mesh for the electric field, $\vec{E}$, is sufficiently fine, we can use the discretization developed in Chapter 3 for any of these operators.

The main difficulty arises when we need to compute the effective inner products $(\sigma\vec{E},\vec{W})$ that are needed in order for the discretization of the operator $\sigma(\cdot)$. To this end, assume that σ over the cell is not constant, and consider the approximation to the integral

$$(\sigma\vec{E},\vec{W}) = \int_{\Omega_j} \sigma(\vec{x})\vec{E}\cdot\vec{W}\,d\vec{x}, \tag{7.2}$$

where $\vec{E}$ and $\vec{W}$ are given on the edges of the mesh that is coarser than the mesh for σ. Recall that for the approximation derived earlier in Chapters 3 and 4, we assumed that σ is constant over the cell. That implies that the fields are smooth, and therefore, we could safely approximate the integral by a low order quadrature method. However, when $\sigma = \sigma(\vec{x})$ and assuming that $\sigma(\vec{x})$ may contain jumps, such an integration formula can lead to an $\mathcal{O}(1)$ error and therefore to grossly inaccurate results.

There are two related methods to deal with the problem. First, one can attempt to obtain a uniform or a smooth averaged conductivity over the cell to represent the heterogeneous conductivity. This process is often referred to as upscaling or homogenization. Homogenization theory is rather involved (see [23]), and there are many possible ways to obtain such an average. It is straightforward to see that simple averaging of the conductivity can fail for some very simple scenarios. Consider the averaging of a conductivity

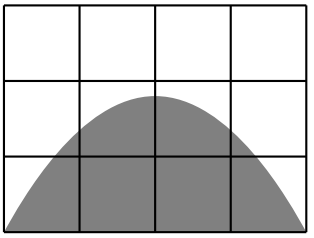

Figure 7.2. *Averaging conductivity due to topographic features. The mountain that goes through the mesh yields a nonsmooth conductivity over some cells.*

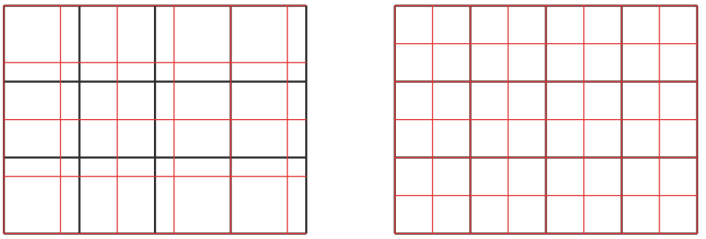

Figure 7.3. *Meshes for the model (red) and fields (black). Left, nonnested mesh. Right, nested mesh.*

that is made from a topographic feature, say, a mountain, with conductivity that is much larger than 0S/m and air that has a conductivity that is very close to 0S/m, as plotted in Figure 7.2.

Simple arithmetic averaging leads to a conductivity that is close to the value of the earth. Indeed, if we try to pass current in the (x,y)-direction through a cell that contains both mountain and air, the conductivity of the earth dominates, and the block that contains the earth and the air behaves more or less like the earth. On the other hand, if we try to pass current from the bottom to the top of the cell, then the air that is very resistive determines the behavior of the cell that behaves as very resistive material. Clearly, the homogenized cell presents anisotropy. One may use logarithmic averaging as suggested in some multigrid applications [73] to obtain some compromise between the two. It is possible to show [54] that any homogenization technique can fail. A more robust way to approach the problem is through multiscale finite volume [39]. Multiscale methods seek an appropriate interpolation for the fields inside each cell given the fields on the edge of the cell. The interpolation is computed by solving Maxwell's equations over each (coarse) cell locally. If the interpolation is $\mathcal{O}(h^p)$ (with some constant $p > 0$) accurate, then it is possible to approximate the integral (7.2) in a similar order, obtaining a consistent approximation to the inner products and to Maxwell's equations.

While the principles of upscaling or multiscale methods can be applied to any forward modeling mesh where the conductivity mesh is finer than the forward modeling mesh, a few simple conventions can make the process and code writing much easier. To this end, consider the mesh for the model (the conductivity mesh), $\mathcal{S}_{\mathbf{m}}$, and a mesh for the fields, $\mathcal{S}_{\mathbf{e}}$. In principle, these meshes can be arbitrarily different, as demonstrated in Figure 7.3 (left). However, if the meshes are not nested, then a partial volume effect needs to be dealt with. To avoid such complexity, we recommend that nested meshes be used as seen

in Figure 7.3 (right) whenever possible. Furthermore, to avoid complexity close to the sources and receivers, we typically choose the mesh for the fields around the sources and receivers to be either the same as or finer than the mesh for the model. This implies that no special integration is needed close to sources and receivers, and the effect of interpolation errors when computing the integral (7.2) is small. We use coarser meshes for the fields away from the sources and receivers, and this tends to reduce discretization errors.

To put the above discussion within the framework presented so far, we consider the upscaling process as a possibly nonlinear function of the conductivity and note the upscaled conductivity tensor Σ as

$$\Sigma = \mathsf{up}(\sigma).$$

As previously explained, the operator up can be chosen using many different techniques. Maybe the simplest is by using some linear scheme on the log conductivity [73], setting the log conductivity of the mesh $\mathcal{S}_{\mathbf{e}}$ (the mesh for the fields) as

$$\mathbf{m}_{\mathcal{S}_{\mathbf{e}}} = \mathbf{U}\mathbf{m},$$

where $\mathbf{U}$ is some upscaling averaging matrix (not necessarily with equal weights).

The advantage of linear averaging is that the forward problem for the ith source and jth frequency can be written as

$$\mathbf{d}_{ij} = \mathbf{P}_{i,j}\mathbf{A}_{ij}(\mathbf{U}_{ij}\mathbf{m}, \omega_j)^{-1}\mathbf{q}_{ij},$$

where $\mathbf{U}_{ij}$ is the averaging matrix for the log conductivity from the mesh $\mathcal{S}_{\mathbf{m}}$ to the mesh $\mathcal{S}_{\mathbf{e}_{ij}}$. This makes the computation of the forward problem easy and the computation of the sensitivities straightforward. The more general case of setting $\Sigma = \mathsf{up}(\sigma)$ (assuming an appropriate upscaling technique is given) can also be easily used for the solution of the forward problem. However, one should observe that it rather complicates the sensitivity calculation since derivatives of the upscaling function are needed as well.

One huge advantage of generating different forward matrices for different sources is that they can be small enough for the use of direct methods and that they can be solved on parallel systems. Indeed, given $n_s \times n_\omega$ linear systems, it is trivial to parallelize it over $n_s \times n_\omega$ processors. In our experience, solving many small problems pays off considerably compared to solving one large system, especially when using some parallel architecture. One important fact that we discuss next is that not all the systems are "equal." Some linear systems are larger or more ill conditioned, and therefore the solution of each linear system may take a different amount of time. As a result, it is difficult to synchronize the process, and this has to be taken into consideration. Asynchronous techniques can be used for the solution of the optimization problem, and we discuss some of these aspects next.

7.3 ▪ Stochastic methods for problems with many sources

For problems with many sources, lowering the cost of each forward problem by choosing an individual mesh is the first tool we have in our arsenal. In this section, we discuss a second mechanism that enables us to reduce the number of forward problems to be solved, at least if the required accuracy for the solution of the optimization problem is low. The mechanism uses stochastic programming methods in order to reduce the cost of the optimization. Stochastic programming is commonly used for problems that have a structure similar to ours, where the objective function is a sum of many functions. The technique is frequently used in machine learning (see, for example, [57]), and the extension

to inverse problems is straightforward. The idea stems from the observation that our problems can be written as a sum over subproblems,

$$\phi_d(\mathbf{m}) = \frac{1}{2n_d}\sum_{i,j} ||\mathbf{W}_{ij}(\mathbf{P}_{i,j}\mathbf{A}_{ij}(\mathbf{m},\omega_j)^{-1}\mathbf{q}_{ij} - \mathbf{d}_{ij})||^2 = \frac{1}{2n_d}\sum_{i,j}\phi_{d_{ij}}(\mathbf{m}), \quad (7.3)$$

where $\phi_{d_{ij}}$ is the misfit from the ith source and jth frequency. In many cases there is some redundancy in the data. For example, in practical airborne surveys, sources are transmitting close to each other and thus yield similar information. While it is difficult to analyze the redundancy exactly, it is possible to use it in order to obtain methods that use only a portion of the sources at each iteration.

A number of authors have been analyzing similar problems [27, 74] and for a variety of problems have used variations of the algorithms we present here. We now discuss these methods and outline the ideas behind them.

7.3.1 ▪ Stochastic Hessians

Recent improvement to the Gauss–Newton method for our problem, where the misfit can be decomposed into a sum of misfits, has been discussed in the context of machine learning [14].

Assume that we use a Gauss–Newton method for the solution of the inverse problem. The gradient of the misfit is given by

$$\nabla_{\mathbf{m}}\phi_d(\mathbf{m}) = \frac{1}{2n_d}\sum_{i,j}\nabla_{\mathbf{m}}\phi_{d_{ij}}(\mathbf{m}), \quad (7.4)$$

and the Jacobian is given by

$$\mathbf{J} = \frac{1}{n_d}\sum_{i,j}\mathbf{J}_{ij}(\mathbf{m},\mathbf{e}_{ij}). \quad (7.5)$$

Even though we do not compute the Jacobian explicitly, we require the storage of the electric fields $\mathbf{e}_{ij}$ that correspond to each frequency and source. For large scale problems we are required to store all the electric fields, and, in order to compute the product of the Jacobian or its transpose with a vector, we need to further solve Maxwell's equations for each source and frequency.

An important property of standard optimization techniques is that while the gradient and the objective function need to be estimated relatively accurately, only approximate Hessian information is required. This point is the basis of methods such as truncated Newton [58, 51], and we can use it here to reduce some of the computational burden.

To this end, we consider the following approximation to the Hessian:

$$\widehat{\mathbf{H}}(\mathbf{m}) = \frac{1}{n_{ds}}\widehat{\mathbf{J}}^\top\widehat{\mathbf{J}} + \alpha\nabla^2 R(\mathbf{m}),$$

where

$$\widehat{\mathbf{J}} = \frac{1}{n_{ds}}\sum_{i,j\in\mathscr{I}}\mathbf{J}_{ij}(\mathbf{m},\mathbf{e}_{ij})$$

and $\mathscr{I}$ is some index set that has only n_{ds} data that correspond to the choice of sources and frequencies rather than the full number of data, n_d. Though this is a very small

change to the algorithm, it has some far-reaching consequences. First, using all sources and frequencies within the Gauss–Newton method requires the storage of all the fields. For problems with many sources and fields, this storage becomes prohibitively large, and therefore one cannot use the Gauss–Newton method directly. Storing only a few of the fields, one computes the data, one field and frequency at a time, and overwrites the field for a new source or frequency. Only a selected, random, number of sources are then chosen and kept in memory, depending on the size of our memory. In our numerical experiments we have found that even with 1% of the fields, a rather reasonable approximation to the sensitivities is obtained. This is, of course, problem dependent. The more redundant information we have, the fewer samples can be used. We have found that by analyzing the sensitivities (in a half-space), we are able to obtain some indication of the sampling that "makes sense." Nonetheless, this analysis is rather problem and system dependent and requires some experimentation.

7.3.2 ▪ The stochastic gradient and Gauss–Newton

In the previous section, we assumed that the objective function and the gradient are calculated to a very good approximation, while the Hessians are computed only approximately using a stochastic choice of sources and frequencies. The advantage of this technique is that it has similar properties to the usual Gauss–Newton technique and requires very few changes. However, for problems with many sources and frequencies, the method still needs to loop over virtually all sources and frequencies in order to evaluate the objective function and the gradient. For problems with many sources and frequencies, for example, for regional airborne methods, such an approach is too expensive. Most computational platforms cannot deal with millions of sources, and thus the Gauss–Newton method with stochastic Hessians is impractical in these scenarios.

This leads to different methods that can be used, that is, stochastic gradient descent and the stochastic Gauss–Newton. The classical stochastic gradient descent reads as follows.

Algorithm: Stochastic gradient descent

- Initialize $\mathbf{m} = \mathbf{m}_0$ and choose a damping parameter μ
- While not converge
  1. Compute the estimate $\phi_{\mathscr{A}_k}(\mathbf{m}_k)$ and $\nabla_{\mathbf{m}_k,\mathscr{A}_k}\phi$
  2. Set $\widehat{\mathbf{m}}_{k+1} \leftarrow \mathbf{m}_k - \mu\nabla_{\mathbf{m},\mathscr{A}_k}\phi$
  3. Set $\mathbf{m}_{k+1} = \frac{1}{k+1}\left(\sum_{j=1}^{k}\mathbf{m}_j + \widehat{\mathbf{m}}_{k+1}\right)$
  4. Test for convergence

In this algorithm, the line search (or damping) parameter μ is fixed, and $\mathscr{A}_k$ is *any* unbiased subset of sources and frequencies. An extreme example is that we compute a single gradient with respect to one random source at each iteration. This basic algorithm dates to the 1950s (see [69] for details). Its attraction is in its simplicity. It never requires that the objective function or gradients be evaluated for the whole data set, and it can be used with as few sources as our computer memory can afford. However, while this algorithm and others that follow with similar flavor [56] seem very attractive, they tend to work poorly for our applications. This should not come as a surprise. Simple stochastic descent methods require very strong assumptions on the function to be minimized. For the problems we are attempting to solve, these assumptions are rarely (if at all) fulfilled.

Nonetheless, stochastic gradient methods have been the motivation for many algorithms that lack rigorous proofs of convergence but work very well in practice.

The first method is discussed in [74, 21]. The basic idea here is to use the Gauss–Newton method, subsampling the sources and adapting the number of sources that are randomly chosen as we descend. Consider the kth stochastic Gauss–Newton iteration, and assume that we are using n_k sources that are randomly chosen. We assume we have an approximation to the objective function $\phi_k(\mathbf{m}_k)$ and the gradient $\nabla_{\mathbf{m}}\phi_k(\mathbf{m}_k)$. We can compute and take a usual Gauss–Newton-like step for the n_k sources obtaining a *candidate model* $\widehat{\mathbf{m}}_{k+1}$. The reason that this is a candidate model is that while it decreases the objective function that uses the n_k random sources we choose, we do not know that it actually decreases the true objective function (and we have no way of knowing without using all sources). To evaluate the validity of the candidate model without computing the true objective function, we choose a different random set of ℓ_k sources and compute the objective function over these sources for the model $\mathbf{m}_k$ and for the model $\widehat{\mathbf{m}}_{k+1}$. If the objective function decreases over both sets, then we accept the model and set $\mathbf{m}_{k+1} = \widehat{\mathbf{m}}_{k+1}$. If, on the other hand, the objective function decreases over the n_k sources used for the computation of the Gauss–Newton step but increases for the ℓ_k sources used for validation, then this implies that the sample size is too small and that we should increase the sample size. In the experiments reported in the literature the sample size is typically doubled, or increased by 50%, and a step is computed again. In my experience, such methods work well for problems where the noise level is large. They tend to quickly and cheaply decrease the misfit, especially initially, using very few sources to achieve this goal. However, when trying to "push" the misfit, they tend to get "stuck," converging very slowly, and have a hard time reducing the misfit unless many sources are used.

In the above stochastic algorithms, we deal with a different subset of sources at each iteration. This has the advantage of our being able to use very few sources; however, the price to be paid is robustness. In my experience, such algorithms require careful tuning and can have many parameters to select. Their behavior may also be unpredictable. For example, the working set may grow significantly at times, to the point that makes stochastic methods inefficient. Careful examination can reveal that we are trying to oversolve the problem, fitting it below the noise level. Indeed, making such algorithms robust is not simple. We therefore discuss a second type of algorithm that can be more expensive but in turn is more robust. The base for such algorithms is the sample average approximation (SAA), which is summarized as follows.

Algorithm: Stochastic average approximation

- Initialize $\mathbf{m} = \mathbf{m}_0$
- For $i = 1, \ldots, \ell$
  1. Choose a random subset of sources/frequencies
  2. Solve the inverse problems over these sources (use any method), and obtain an estimate $\mathbf{m}_i$
- Use the models $\mathbf{m}_1, \ldots, \mathbf{m}_\ell$ and obtain their statistics
- Return $\mathbf{m} = \frac{1}{\ell}\sum \mathbf{m}_i$ and the standard deviation

There are a number of advantages to the SAA over stochastic descent methods. First, the method allows for any descent technique and decouples the sampling process from the

optimization process. This allows for the use of any "standard" optimization routine to be used and does not require special algorithms. Second, solving the problem a number of times using a different set of sources allows us to obtain some statistics. In particular, we obtain some estimate of the variance of the models (but not the bias) for free. This gives us some idea about the uncertainty of the results. Finally, while the algorithm seems much more expensive compared with stochastic descent methods, a few simple "tricks" can make the algorithm much cheaper. First, there is no need to initialize every inversion with $\mathbf{m}_0$ (unless the inversion of different subsets is done in parallel). In practice, one uses the parallel system to deal with sources and solve a number of problems in a sequential fashion. In this case, if the solutions to problems with different random sets are initialized with the solution of the first problem, then a solution is usually obtained within a very few iterations (potentially one or two). Furthermore, it is possible to dynamically choose the number of sources/frequencies to be used, similar to the techniques discussed above. The disadvantage of SAA-type approaches is that the methods are relatively slow compared to stochastic gradient descent. Nonetheless, these methods are much more robust and therefore should not be overlooked.

Chapter 8
Inversion in Practice

In this chapter we provide a practical guide to inversion by using a simple, yet semirealistic, example. One key component is that I have never seen a successful 3D inversion of field data that was fed into an inversion code without any preprocessing and run to obtain a reasonable model and data fit. Inversion of field data requires understanding the data and its errors, designing the mesh and the regularization, and, finally, setting the expectation from the inversion and understanding the inherent resolution of the method. In this chapter we discuss these issues and demonstrate the steps to be taken for the inversion of field data. We also supply and work through computer codes that enable the reader to repeat the experiments performed here, modify them, and obtain a general inversion code that can work for many electromagnetic inverse problems.

8.1 ▪ Electromagnetic data set

To ground ourselves with an example and show how inversion can work in practice we consider a controlled source electromagnetic (CSEM) data set. Similar data sets are collected in exploration for hydrocarbons. The example here is a toy example; that is, the model is very small (in size). We aim for the user to be able to run this example with rather modest computational resources. However, it is straightforward to scale up the example presented here by making the model larger and assuming a larger range of frequencies and more sources and receivers. We leave this as an assignment for the reader.

For the example here, there are 196 electrical dipole sources that are towed in the water roughly 150 meters above the sea floor. The 324 receivers are placed on the ocean floor. Here we assume we have three component electric receivers; that is, we record the fields in the x-, y-, and z-directions. We assume we have a single frequency of 10^{-1}Hz. In practice many more frequencies are collected; however, for ease of computation we assume we have only a single frequency. The locations of the sources and receivers are plotted in Figure 8.1.

Slices through the model are plotted in Figure 8.2. The model contains sea water with conductivity of 3S/m and some slightly more resistive earth layers. A salt dome with conductivity of 10^{-2}S/m and a thin reservoir with a resistivity of 10^{-3}S/m are the resistive targets within the conductive media.

To simulate the data we discretize the model on a rather modest grid of 30^3 mesh points. This mesh is too coarse, and we choose it only for ease of computation. Nonetheless, as we see next, even for this coarse mesh, the results that are obtained are reasonable.

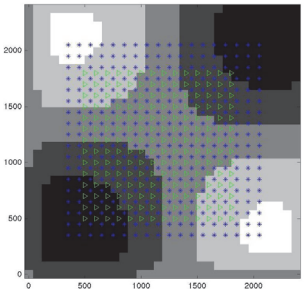

Figure 8.1. *Source (green) and receiver (blue) locations for the synthetic data.*

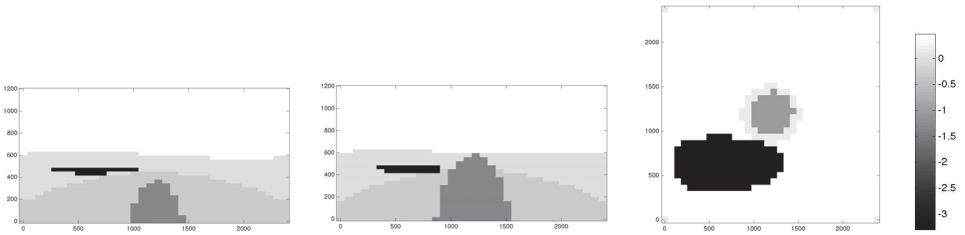

Figure 8.2. *The hydrocarbon model. The goal is to recover the thin resistor from the data collected on the ocean floor.*

The refinement of the mesh and more thorough computation is left as an assignment for the reader. Solving Maxwell's equations with the assigned source-receiver configuration yields 190,512 complex data points. This data is to be inverted next in order to obtain an estimate of the conductivity.

8.2 ▪ Understanding the data

Electromagnetic data do not have a unique standard. This may be a result of the different experiments that are done, different types of field data and systems, and a lack of agreement between vendors on some basic conventions such as units or coordinate systems. Therefore, the first step when approaching any field data set is the understanding of the units and the coordinate system. While this may look trivial, you may find that in some cases this cannot be done without some simple modeling. To this end, it is highly recommended to have a few simple modeling tools. In particular, full/half-space and 1D responses are essential to the understanding of the data.

Consider the CSEM example used here. To understand the data and see that it "makes sense," we plot the field obtained by one of the sources. Although our data are obtained from an electric source in heterogeneous media, we use either 1D or analytic solutions for a dipole in a full space of homogeneous conductivity. We choose a conductivity of 3S/m as the conductivity of the full space. While we expect the fields to be different (to second order), they should be very similar to first order. This is because the main impact on electromagnetic data is the geometric spread and background conductivity. In Figure 8.3 we compare the data from our model to the half-space data. We plot the absolute value

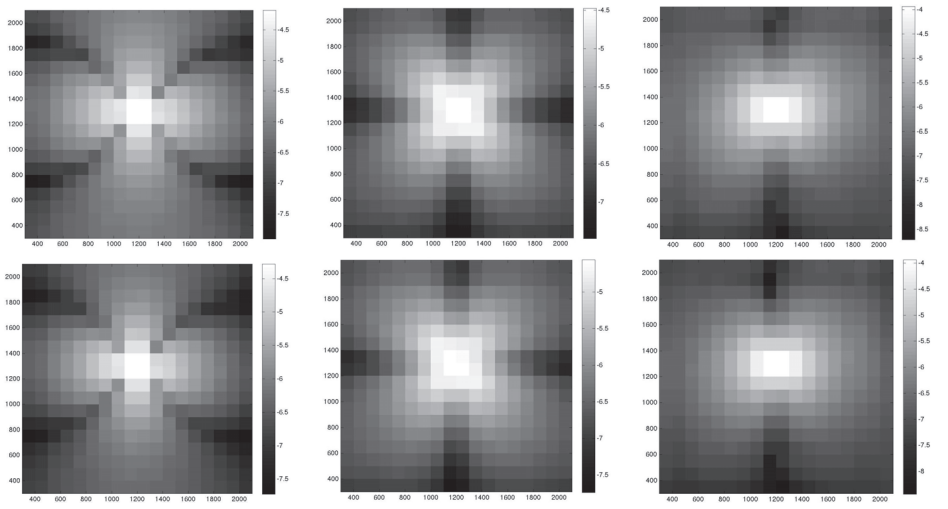

Figure 8.3. *A comparison between the data obtained in our experiment and the half-space data.*

of the data obtained by our model and the data obtained from a full space. As can be seen in the figure, the data indeed look similar, and this indicates that our model is well calibrated. A more thorough examination of the data shows a difference of roughly 10%. These 10% are to be fitted in order to estimate the conductivity variations.

The above example is a simple illustration of a data quality control (QC) process. Data QC is very important and should be done with various tools. A different tool that is very important is the apparent conductivity/resistivity. Apparent resistivity is the value of the conductivity, assuming the earth is a full space, that yields a particular datum, given the source-receiver combination. Recall from Chapter 1 that the electric field in a full space is given by

$$\vec{E} = G\frac{I\delta s \exp(ikr)}{4\pi\sigma r^3}, \tag{8.1}$$

where

$$G = \left((-k^2r^2 + 3ikr + 3)\begin{pmatrix} \frac{x^2}{r^2} \\ \frac{xy}{r^2} \\ \frac{xz}{r^2} \end{pmatrix} + (k^2r^2 - ikr - 1)\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}\right),$$

and therefore, given a measured $\vec{E}_x$ at some particular point, we can define the apparent resistivity as

$$\rho_a = \left|\frac{4\pi r^3 \vec{E}_x \exp(-ikr)}{IG_x\delta s}\right|.$$

The apparent conductivity should have "reasonable" values. For example, in our CSEM surveys we expect that the data yield an apparent resistivity in the range of $1-100\Omega$/m. If we detect that the data are not close to that range, then we may want to check the experimental data. For example, we should check that the current, I, is known and the

distance between source and receiver is indeed as reported. In many cases, the apparent resistivity calculations can aid in the detection of data outliers and in the estimation of the noise.

From apparent conductivity calculations to 1D and 2D inversion algorithms, the QC process, in which we understand the data and its limitations and decide about the standard deviation of each datum and its importance, is one of the most influential parts of the inversion process. In many cases, different choices of data variance lead to different inversion results. We have to remember that while regularization is our "invention" that allows for the solution of the inverse problem by introducing bias, the data are typically unbiased, and therefore, attempting to understand them should get a high priority in the inversion process.

As stated earlier, I have never seen a real data set that was inverted "out of the box." Each data set requires modeling, playing with, and inverting more than once. The process of understanding the data can lead to algorithmic developments. For example, after "playing" with the data, one may realize that a different misfit function is to be used for inversion. These understandings come often when experimenting with the data, and therefore I recommend that one not rush to invert the data but rather understand them first.

8.3 ▪ Solving the inverse problem: First pass

Inverting field data set on a fine mesh can be a computationally demanding task. A common "beginner" mistake is to set up the inversion on a fine mesh and let it run. The problem is that most first pass inversions do not work at first. Running the inversion using a fine mesh, we may "discover" that the inversion does not work only after many hours or days. We may want to change some parameters and rerun. Such an exercise is expensive and inefficient. Therefore, before the final inversion is run we *always* run preliminary inversions on a number of coarse meshes and in many cases with a subset of the data. This allows for a quick turnaround that leads to a quick fine tuning of the inversion parameters.

In this section we run such a process on the CSEM data set and show how such a process can help in choosing parameters within the inversion.

First, let us examine the preparation of the inversion. In the first step we prepare the mesh.

```
%% set up the mesh and the domain
h1 = [1.85.^[3:-1:1], ones(1,24),1.85.^[1:1:3]]*50;  h1 = h1(:);
h2 = [1.85.^[3:-1:1], ones(1,24),1.85.^[1:1:3]]*50;  h2 = h2(:);
h3 = [1.85.^[3:-1:1], ones(1,24),1.85.^[1:1:3]]*25;  h3 = h3(:);

Lx = sum(h1); Ly = sum(h2); Lz = sum(h3);

n1 = length(h1); n2 = length(h2); n3 = length(h3);
nc = n1*n2*n3;

[X,Y,Z] = getCellCenteredGrid(h1,h2,h3);
```

Next, we prepare a synthetic model.

```
% set up salt body
stp = @(t) 0.5*(tanh(100*(t-0.5))+1);
sx = 10^5; sy = 10^5; sz = 5*10^5;
s = stp(exp(-(X-1200).^2/sx - (Y-1200).^2/sy - (Z - 50).^2/sz));

% layered earth
m1 = 50*(sin(2*pi*X/Lx) .* sin(2*pi*Y/Ly)) + 600;
m2 = 100*(cos(2*pi*X/Lx) .* cos(2*pi*Y/Ly)) + 400;

sigWater = 3;
sigmaL = zeros(n1,n2,n3);
sigmaL(Z(:)< m1(:)) = 1;
sigmaL(Z(:)> m1(:)) = sigWater;
sigmaL(Z(:)< m2(:)) = 0.5;

% Reservoir
sx = 10^5; sy = 5*10^5; sz = 2e3;
r = stp(exp(-(X-800).^2/sx - (Y-800).^2/sy - (Z - 500).^2/sz));

% put it all together
sigma = sigmaL;
sigma(s(:)>0.99) = 5e-2;
sigma(r(:)>0.99) = 5e-4;
sigma(Z(:)> m1(:)) = sigWater;
```

We next set up the source-receiver location.

```
% set up source locations
[srcx,srcy] = meshgrid(500:100:1800);
srcx = srcx(:); srcy = srcy(:); srcz = srcx*0+700;

% set up receiver location
[recx,recy] = meshgrid(350:100:2050);
recx = recx(:); recy = recy(:);
recz = 50*(sin(2*pi*recx/Lx) .* sin(2*pi*recy/Ly)) + 600;
```

Using the source-receiver location and the mesh, we can define the source-receiver matrices that are to be used for the inversion. For the problem here we use a single mesh; however, more complex structures, such as a source-receiver fitted mesh, can be used.

```
%% Prepare the matrices for source receiver
ns = length(srcx); nr = length(recx);
% set up source dipoles
x = [0;cumsum(h1)];
y = [0;cumsum(h2)];
z = [0;cumsum(h3)];

nex = n1*(n2+1)*(n3+1); ney = n2*(n1+1)*(n3+1); nez = n3*(n2+1)*(n1+1);
ne = nex + ney + nez;
Q = zeros(ne,ns);
for i=1:length(srcx)
    pt = [srcx(i)-50  srcy(i)  srcz(i); srcx(i)+50  srcy(i)  srcz(i)];
    Q(:,i) = getDipoleFromPoints(pt,x,y,z);
end

% set up receivers (3 components)
P = zeros(ne,3*nr);
cnt = 1;
for i=1:length(recx)
    pt = [recx(i)-50  recy(i)  recz(i); recx(i)+50  recy(i)  recz(i)];
    P(:,cnt) = getDipoleFromPoints(pt,x,y,z);
    cnt = cnt+1;
end
for i=1:length(recx)
    pt = [recx(i)  recy(i)-50  recz(i); recx(i)  recy(i)+50  recz(i)];
    P(:,cnt) = getDipoleFromPoints(pt,x,y,z);
    cnt = cnt+1;
end
for i=1:length(recx)
    pt = [recx(i)  recy(i)  recz(i)-50; recx(i)  recy(i)  recz(i)+50];
    P(:,cnt) = getDipoleFromPoints(pt,x,y,z);
    cnt = cnt+1;
end
```

Finally, we put all this information in a structure to be used for the modeling of the data.

```
%% Prepare the param structure for forward modeling
param.w = 0.1*2*pi;

[Curl,Grad,Div,Af,Ae,An,V] = getLinearOps(h1,h2,h3);
param.Curl = Curl;
param.Grad = Grad;
param.Af   = Af;
param.Ae   = Ae;
param.An   = An;
param.V    = V;

% active cells
Iact = speye(nc);
ind = find(sigma~=sigWater);
Iact = Iact(:,ind);

param.Iact  = Iact;
param.nc    = nc;
param.mback = sigWater;

param.obs = P';
param.src = Q;

sigAct = Iact'*sigma(:);

%% Finally model
d = getFreqDomainData(sigAct,param);
```

Some of the data are plotted in Figure 8.3. These data are to be inverted. For the inversion we need to pick the standard deviation of the data. Observing the data, we see that they range over three orders of magnitude. Our noise model consists of a floor which is 10% of the low order values and a percentile of 5%.

The code for the inversion is divided into a few directories. The directory MaxFreq contains the main lower level codes. The code getFreqDomainData.m solves the forward problem, and the codes getJmatVec.m and getJTmatVec.m compute the products of the sensitivity matrix and its transpose with vectors. The optimization routines are located in the directory optim. The main optimization code is the projGNCG.m that uses the CG Gauss–Newton method. The codes call a regularization routine, regfun.m, that computes the regularizer and its derivative and a misfit function, misfun.m, that computes the misfit and its derivatives. Finally, the parameterization of the conductivity is given in sigfun.m. The rest of the codes for getting the linear operators, the sources, and the setting up of the problem are in the directories utils, EMsources, and CSEMsetup.

For the results presented here, we run the inversion code projGNCG.m with the set up that is presented above. The results are presented in Figure 8.4.

The results are very typical of EM inversions. One can see that the tops of the resistors are nicely identified; however, using a single frequency the bottom of the model is poorly resolved. Also, the images show that the overall resolution of EM methods is rather poor. We did not manage to recover a very accurate representation of the thin resistor. Therefore, if our goal is to find the thin resistor, then using the CSEM experiment with the setting above is the right tool; however, if we wish to accurately define the boundaries of the thin resistor, then our experiment does not provide this information. These results can be improved by using better regularization techniques that are fitted to such models; however, one has to understand that the overall resolution of the data is not as good as, say, that of seismic data.

Setting the expectation from the experiment is important, and therefore I recommend that a synthetic inversion be done before one attempts to invert and interpret any field

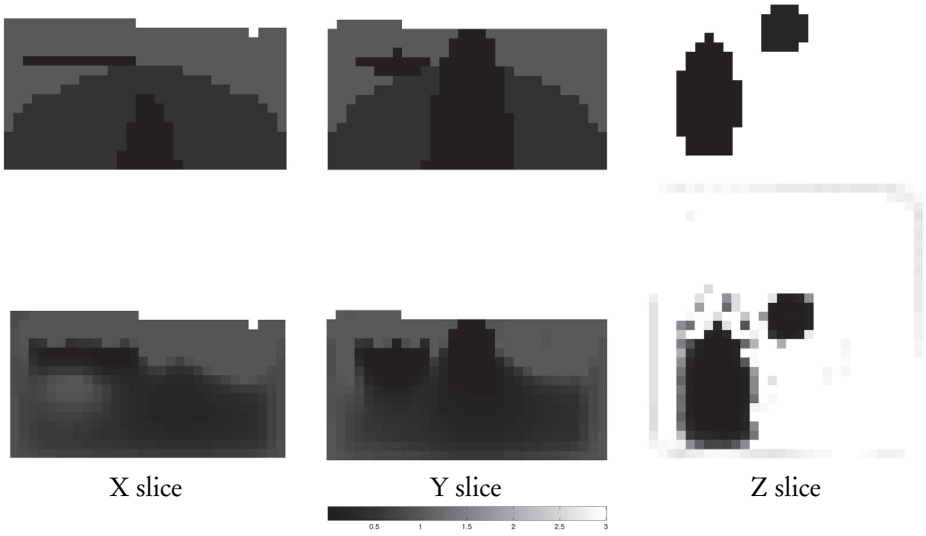

Figure 8.4. *The "true" model and the inversion result.*

data set. Synthetic inversions aid in the understanding of which features in the final model are real and which features are ignored in the inversion process. Synthetic inversions provide an intuitive tool for the understanding of the uncertainty of the inversion process and the bias introduced by the regularization. Currently, practical tools for uncertainty estimation are in their infancy, and therefore, inverting the data with different assumptions on the noise, using different regularizations, and playing with synthetic models are vital tools for understanding the inversion results. Interpreting the inversion and coming up with geological conclusions without understanding the inversion process and its limitations can lead to hazardous results. On the other hand, interpreting the results when the inversion process is well understood can lead to very useful and insightful discoveries.

Bibliography

[1] F. Alvarez, A. Alegria, and J. Colmenero. Relationship between the time-domain Kohlrausch-Williams-Watts and frequency-domain Havriliak-Negami relaxation functions. *Physical Review B*, 44:7306–7312, 1991. (Cited on p. 15)

[2] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl.*, 23:15–41, 2001. (Cited on p. 71)

[3] D. Aruliah and U.M. Ascher. Multigrid preconditioning for Krylov methods for time-harmonic Maxwell's equations in three dimensions. *SIAM J. Sci. Comput.*, 24:702–718, 2003. (Cited on pp. 70, 74)

[4] D.A. Aruliah. *Fast Solvers for Time-Harmonic Maxwell's Equations in* 3*D*. Ph.D. thesis, University of British Columbia, Vancouver, BC, 2001. (Cited on p. 51)

[5] U. Ascher, H. Huang, and K. van den Doel. Artificial time integration. *BIT*, 47:3–25, 2007. (Cited on p. 117)

[6] U.M. Ascher and L.R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM, Philadelphia, 1998. (Cited on p. 51)

[7] Z. Bai, M. Fahey, and G. Golub. Some large scale matrix computation problems. *J. Comput. Appl. Math.*, 74:71–89, 1996. (Cited on p. 108)

[8] R. Barrett, M.W. Berry, T.F. Chan, J. Demmel, , J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 1993. (Cited on p. 105)

[9] M. Benzi, G.H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005. (Cited on p. 70)

[10] D. Black, G. Sapiro, D. Marimont, and D. Heeger. Robust anisotropic diffusion. *IEEE Trans. Image Processing*, 7:442–449, 1998. (Cited on p. 98)

[11] R. Blakely. *Potential Theory in Gravity and Magnetic Applications*. Cambridge University Press, Cambridge, UK, 1996. (Cited on p. 7)

[12] A. Bossavit. *Computational Electromagnetism. Variational Formulation, Complementarity, Edge Elements*. Academic Press, New York, 1998. (Cited on p. 51)

[13] A. Bossavit and L. Kettunen. Yee-like schemes on staggered cellular grids: A synthesis between FIT and FEM approaches. *COMPUMAG*, 1999. Short contribution. (Cited on p. 51)

[14] R.H. Byrd, G.M. Chin, W. Neveitt, and J. Nocedal. On the use of stochastic Hessian information in optimization methods for machine learning. *SIAM J. Optim.*, 21:977–995, 2011. (Cited on p. 124)

[15] E.J. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Information Theory*, 52:489–509, 2006. (Cited on p. 99)

[16] J. Chen and D.W. Oldenburg. 3-D inversion of magnetic induced polarization data. In *3DEMIII Workshop, Adelaide*, 2003. (Cited on p. 102)

[17] J. Claerbout and F. Muir. Robust modeling with erratic data. *Geophysics*, 38:826–844, 1973. (Cited on pp. 98, 99)

[18] K.S. Cole and R.H. Cole. Dispersion and absorption in dielectrics. I. Alternating current field. *J. Chemical Physics*, 1:341–351, 1941. (Cited on p. 13)

[19] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, and J.W.H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20:720–755, 1999. (Cited on p. 71)

[20] J.E. Dennis, Jr., and R.B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, Philadelphia, 1996. (Cited on p. 113)

[21] Y. Dikun, D. Oldenburg, and E. Haber. 3-D inversion of airborne electromagnetic data parallelized and accelerated by local mesh and adaptive soundings. *Geophys. J. Int.*, 196:1492–1507, 2014. (Cited on p. 126)

[22] O. Dorn, E.L. Miller, and C.M. Rappaport. A shape reconstruction method for electromagnetic tomography using adjoint fields and level sets. *Inverse Problems*, 16:1119–1156, 2000. (Cited on p. 96)

[23] L. Durlofskey. Upscaling of geocellular models for reservoir flow simulation: A review of recent progress. In *International Forum on Reservoir Simulation, Baden-Baden, Germany*, June, 2003. (Cited on p. 121)

[24] H.W. Engl, M. Hanke, and A. Neubauer. *Tikhonov Regularization of Nonlinear Differential-Algebraic Equations*. Technical report, Institut für Mathematik, Linz, Austria, 1989. (Cited on p. 117)

[25] H.W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Kluwer, Dordrecht, The Netherlands, 1996. (Cited on p. 117)

[26] L.C. Evans. *Partial Differential Equations*. AMS, San Francisco, CA, 1998. (Cited on pp. 6, 10)

[27] M.P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM J. Sci. Comput.*, 34:A1380–A1405, 2012. (Cited on p. 124)

[28] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias variance dilemma. *Neural Computation*, 4:1–58, 1992. (Cited on p. 96)

[29] L. Gilles, C.R. Vogel, and J.M. Bardsley. Computational methods for a large-scale inverse problem arising in atmospheric optics. *Inverse Problems*, 18:237–252, 2002. (Cited on p. 112)

[30] G.H. Golub and C. Greif. On solving block-structured indefinite linear systems. *SIAM J. Sci. Comput.*, 24:2076–2092, 2003. (Cited on p. 70)

[31] G.H. Golub and Q. Ye. Inexact preconditioned conjugate gradient method with inner-outer iteration. *SIAM J. Sci. Comput.*, 21:1305–1320, 1999. (Cited on p. 108)

[32] S. Guttel. *Rational Krylov Methods for Operator Functions*. Thesis Dissertation, The University of Freiburg, Freiburg, Germany, 2010. (Cited on pp. 66, 68)

[33] E. Haber. Quasi-Newton methods for large scale electromagnetic inverse problems. *Inverse Problems*, 21:305–333, 2005. (Cited on p. 105)

[34] E. Haber and U.M. Ascher. Preconditioned all-at-once methods for large, sparse parameter estimation problems. *Inverse Problems*, 17:1847–1864, 2001. (Cited on pp. 31, 51)

[35] E. Haber, U.M. Ascher, D.A. Aruliah, and D.W. Oldenburg. Fast simulation of 3D electromagnetic problems using potentials. *J. Comput. Phys.*, 163:150–171, 2000. (Cited on p. 51)

[36] E. Haber, U.M. Ascher, and D. Oldenburg. On optimization techniques for solving nonlinear inverse problems. *Inverse Problems*, 16:1263–1280, 2000. (Cited on p. 101)

[37] E. Haber and S. Heldmann. An OcTree multigrid method for quasi-static Maxwell's equations with highly discontinuous coefficients. *J. Computational Physics*, 223:324–337, 2007. (Cited on p. 51)

[38] E. Haber, D. Oldenburg, and U. Ascher. Inversion of 3D electromagnetic data—a constrained optimization approach. In *SEG*, Calgary, August 2000. (Cited on p. 101)

[39] E. Haber and L. Ruthutto. A multiscale method for Maxwell's equations at low frequencies. *Geophys. J. Int.*, to appear, 2014. (Cited on p. 122)

[40] M. Hanke and C. Groetsch. Nonstationary iterated Tikhonov regularization. *J. Optim. Theory Appl.*, 98:37–53, 1998. (Cited on p. 117)

[41] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Research Nat. Bur. Standards*, 49:409–436, 1952. (Cited on p. 108)

[42] J. Hilfer. *H*-function representations for stretched exponential relaxation and non-Debye susceptibilities in glassy systems. *Physical Review E*, 65:061510, 2002. (Cited on p. 15)

[43] R. Hiptmair. *Multilevel Preconditioning for Mixed Problems in Three Dimensions*. Ph.D. thesis, University of Augsburg, Augsburg, Germany, 1996. (Cited on p. 51)

[44] P.J. Huber. Robust estimation of a location parameter. *Ann. Math. Statist.*, 35:73–101, 1964. (Cited on p. 92)

[45] J.M. Hyman and M. Shashkov. The adjoint operators for natural discretizations for the divergence, gradient and curl on logically rectangular grids. *IMACS J. Applied Numerical Math.*, 25:413–442, 1997. (Cited on pp. 21, 31)

[46] J.M. Hyman and M. Shashkov. *Mimetic Discretizations for Maxwell's Equations and Equations of Magnetic Diffusion*. Technical report, LA-UR-98-1032, Los Alamos National Laboratory, Los Alamos, NM, 1998. (Cited on pp. 21, 31)

[47] J.M. Hyman and M. Shashkov. Mimetic discretizations for Maxwell's equations. *J. Comput. Phys.*, 151:881–909, 1999. (Cited on pp. 21, 31)

[48] J.M. Hyman and M. Shashkov. The orthogonal decomposition theorems for mimetic finite difference methods. *SIAM J. Numer. Anal.*, 36:788–818, 1999. (Cited on pp. 21, 31)

[49] J. Jin. *The Finite Element Method in Electromagnetics*. John Wiley and Sons, New York, 1993. (Cited on p. 1)

[50] J. Kaipio and E. Somersalo. *Statistical and Computational Inverse Problems.* Springer-Verlag, New York, 2004. (Cited on p. 91)

[51] C.T. Kelley. *Iterative Methods for Optimization*. SIAM, Philadelphia, 1999. (Cited on pp. 101, 105, 107, 108, 109, 124)

[52] A. Leito and O. Scherzer. On the relation between constraint regularization, level sets, and shape optimization. *Inverse Problems*, 19:1–11, 2003. (Cited on p. 96)

[53] C.P. Lindsey and G.D. Patterson. Detailed comparison of the Williams-Watts and Cole-Davidson functions. *J. Chemical Physics*, 73:3348–3357, 1980. (Cited on p. 15)

[54] L.A. Caudillo Mata, L. Heagy, E. Haber, and D. Oldenburg. Homogenization and upscaling for Maxwell's equations. *SEG Technical Program Expanded Abstracts*, 2014. (Cited on p. 122)

[55] P. Monk. *Finite Element Methods for Maxwell's equations*. Oxford University Press, Oxford, UK, 2003. (Cited on pp. 1, 3)

[56] Y. Nesterov. *Gradient Methods for Minimizing Composite Objective Function*. Ph.D. dissertation, Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2007. CORE Discussion Paper 2007/76. (Cited on p. 125)

[57] F. Niu, B. Recht, C. Re, and S.J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, Granada, Spain, 2011. (Cited on p. 123)

[58] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, 1999. (Cited on pp. 101, 103, 105, 108, 109, 113, 124)

[59] S. Osher and R.P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, New York, 2003. (Cited on p. 96)

[60] L.R. Pasion and D. Oldenburg. A discrimination algorithm for UXO using time domain electromagnetics. *J. Engineering and Environmental Geophysics*, 28:91–102, 2001. (Cited on p. 96)

[61] P. Perona and J. Malik. Scale space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Machine Intell.*, 12:629–639, 1990. (Cited on p. 98)

[62] M. Popolizio and V. Simoncini. Acceleration techniques for approximating the matrix exponential operator. *SIAM J. Matrix Anal. Appl.*, 30:657–683, 2008. (Cited on p. 67)

[63] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. In *Proceedings of the Eleventh Annual International Conference of the Center for Nonlinear Studies on Experimental Mathematics: Computational Issues in Nonlinear Science*. Elsevier North-Holland, Amsterdam, 1992, pp. 259–268. (Cited on p. 98)

[64] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2nd ed., SIAM, Philadelphia, 2003. (Cited on pp. 69, 108)

[65] M.D. Sacchi and T.J. Ulrych. Improving resolution of Radon operators using a model re-weighted least squares procedure. *J. Seismic Exploration*, 4:315–328, 1995. (Cited on p. 99)

[66] P. Sarma, L.J. Durlofsky, K. Aziz, and W. Chen. A new approach to automatic history matching using kernel PCA. In *SPE Reservoir Simulation Symposium*, Houston, Texas, 2007. (Cited on p. 95)

[67] J. Scales and L. Tenorio. Prior information and uncertainty in inverse problems. *Geophysics*, 66:389–397, 2001. (Cited on p. 91)

[68] D. V. Schroeder. *An Introduction to Thermal Physics*. Addison–Wesley, San Francisco, CA, 2000. (Cited on p. 14)

[69] A. Shapiro, D. Dentcheva, and A. Ruszczyński. *Lectures on Stochastic Programming: Modeling and Theory*. SIAM, Philadelphia, 2009. (Cited on p. 125)

[70] A. Taflove. *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Artech House Publishers, Norwood, MA, 1995. (Cited on p. 13)

[71] Y. Tai. *Dyadic Green Functions in Electromagnetics*. Elsevier, New York, 1991. (Cited on p. 1)

[72] A.N. Tikhonov. Determination of the electrical characteristics of the deep strata of the earth's crust. *Doklady Akademii Nauk*, 73:295–297, 1950. (Cited on p. 97)

[73] U. Trottenberg, C. Oosterlee, and A. Schuller. *Multigrid*. Academic Press, New York, 2001. (Cited on pp. 122, 123)

[74] K. van den Doel and U.M. Ascher. *Adaptive and Stochastic Algorithms for EIT and DC Resistivity Problems with Piecewise Constant Solutions and Many Measurements*. Technical report, 2011. (Cited on pp. 96, 124, 126)

[75] T. van Leeuwen, A.Y. Aravkin, and F.J. Herrmann. Seismic waveform inversion by stochastic optimization. *Internat. J. Geophysics*, 2011:689041, 2011. (Cited on p. 92)

[76] C.R. Vogel. *Computational Methods for Inverse Problems*. SIAM, Philadelphia, 2002. (Cited on p. 112)

[77] G. Wahba. *Spline Models for Observational Data*. SIAM, Philadelphia, 1990. (Cited on p. 93)

[78] S.H. Ward and G.W. Hohmann. Electromagnetic theory for geophysical applications. In *Electromagnetic Methods in Applied Geophysics*, Society of Exploration Geophysicists, Tulsa, OK, 1988. (Cited on pp. 1, 17)

[79] J.T. Weaver. *Mathematical Methods for Geo-Electromagnetic Induction*. John Wiley and Sons, New York, 1994. (Cited on p. 1)

[80] C.J. Weiss and G.A. Newman. Electromagnetic induction in a generalized 3D anisotropic earth, Part 2: The LIN preconditioner. *Geophysics*, 68:922–930, 2003. (Cited on p. 51)

[81] K.P. Whittall and D.W. Oldenburg. *Inversion of Magnetotelluric Data for a One Dimensional Conductivity*, Geophysical Monograph Series 5, Society of Exploration Geophysicists, Tulsa, OK, 1992. (Cited on pp. 72, 99)

[82] G. Williams and D.C. Watts. Non-symmetrical dielectric relaxation behaviour arising from a simple empirical decay function. *Trans. Faraday Soc.*, 66:80–85, 1970. (Cited on p. 15)

[83] K.S. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Trans. Antennas and Propagation*, 14:302–307, 1966. (Cited on pp. 21, 31)

Index