Parallelizing the 2.5D Airborne Electromagnetic Inversion Program ArjunAir

by

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Science

Department of Earth Sciences

Memorial University of Newfoundland

June 2014

St. John's Newfoundland and Labrador

Abstract

This thesis describes the development of a parallel version of the 2.5D airborne electromagnetic modelling and inversion program ArjunAir. The program uses a finite-element scheme to model the response of an earth with a 2D conductivity structure to a 3D electromagnetic source. The program uses a Gauss-Newton like iterative inversion algorithm, stabilized by singular value damping, to estimate the conductivity of a 2D depth section of the earth beneath an airborne electromagnetic survey line. The forward modelling code was parallelized and whenever possible, bottleneck routines were replaced by more efficient versions. Shared and distributed memory parallel versions of the ArjunAir forward solver were developed, with the shared memory version being incorporated into a modified ArjunAir inversion program based on the Levenberg-Marquardt algorithm. The new shared memory parallel ArjunAir inversion algorithm ran up to 8 times faster than the original algorithm when running with 8 threads, with speedup due both to parallelization and the use of more efficient sequential routines.

Acknowledgements

First and foremost I would like to thank my supervisors Dr. Colin Farquharson and Dr. Ronald Haynes for their continuous support and encouragement throughout my entire time here at MUN. They guided my research and held me to a high academic standard. Without our many stimulating discussions and their deep knowledge I would not have been able to complete this work. As my senior supervisor, Dr. Farquharson's door was always open and he was always ready to help with a smile on his face, whether it be with some technical issue, or advice on my academic future.

This project was orginally the idea of Marc Vallée. I am most grateful to him for providing the idea to Dr. Farquharson and giving me such a stimulating research project. I am also indebted to Glenn Wilson, Art Raiche and Fred Sugeng, the developers of ArjunAir. Their efficient and well documented software gave me a strong foundation on which to build my work. The software was initially developed as part of project p223, which was funded by the AMIRA consortium and CSIRO, the Australian research organization. Their decision to release the source code of all p223 software made my M.Sc. research possible.

Finally, I would never have developed into the person I am today or made it through my M.Sc. without the deep love and support of my family and friends back home in New Brunswick, far away in Vancouver, and here in St. John's. Thank you everyone. I could not have done it without you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Airborne electromagnetic (EM) surveying methods, which are used to map the electrical conductivity of the subsurface using an aircraft-mounted detection system, form an important class of geophysical techniques. They have been employed by the mineral exploration industry since the mid 20^{th} century to find metallic sulphide ores and other conductive mineral deposits (Palacky and West, 1991). More recently, airborne EM techniques have been used to map groundwater resources (e.g. Kirkegaard et al., 2011).

Software capable of rigorously estimating the true three-dimensional (3D) conductivity structure of the earth based on airborne EM survey data is beginning to be used commercially but is extremely computationally intensive and can only be used by experts with access to supercomputing resources (e.g. Cox et al., 2010; Oldenburg et al., 2013). Assuming the earth's structure to vary in only two dimensions and estimating the conductivity on a 2D slice of the earth is more computationally tractable than the full 3D problem. The

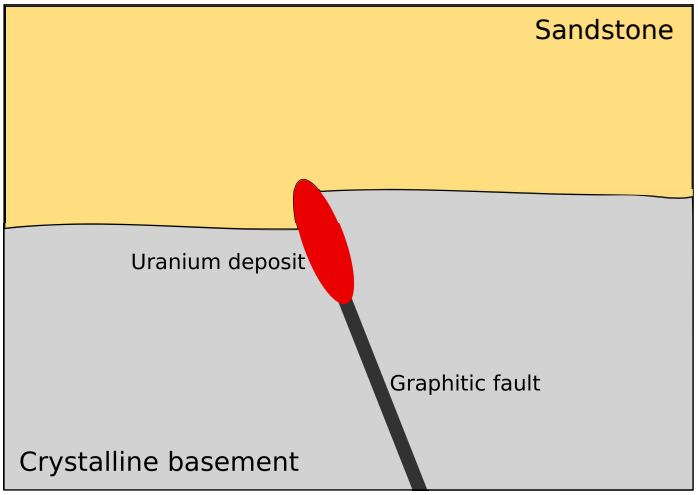


Figure 1.1: Unconformity style uranium deposit. Adapted from a figure by Long Harbour Exploration (2014).

2D approximation is valid in many real-life geological scenarios such as Athabasca style unconformity hosted uranium deposits. These deposits often occur at the bottom of sedimentary basins at the locations of conductive graphitic faults. These faults can be detected by EM methods (Powell et al., 2007) and tend to have long strike length. EM surveys are conducted using flight lines perpendicular to the strike direction. A highly simplified diagram of an Athabasca style uranium deposit is shown in Figure 1.1. The geology will be relatively constant along the direction perpendicular to the page.

The energy sources used in airborne EM produce electromagnetic fields that vary in 3D, even for a 2D earth. The process of modelling these 3D fields using a 2D discretization of the earth is known as 2.5D modelling.

Airborne geophysics industry contacts of my supervisor Dr. Colin Farquharson had been using a 2.5D software package called ArjunAir (Wilson et al., 2006) to estimate the conductivity of 2D slices of the earth from airborne EM survey data. ArjunAir was originally developed by an industrially funded group at the Australian national research organi-

zation CSIRO (Commonwealth Scientific and Industrial Research Organisation). Its source code is now freely available. It is the only production quality software package capable of inverting airborne EM data to estimate a general two-dimensional (2D) model of subsurface conductivity. Dr. Farquharson's industry contacts found that ArjunAir produced useful results but was too computationally intensive to be used routinely on large datasets. CPU and memory costs limited the program to performing inversions over compact targets of interest, rather than over large sections of survey lines.

The experience of these colleagues motivated this project. The goal was to improve the performance of ArjunAir by replacing its core computational routines by more efficient and, whenever possible, parallel versions, while maintaining its capabilities and user interface. There are several obvious sources of parallelism in the computations performed by ArjunAir, and opportunities to reduce its memory use. Making such improvements has allowed more detailed inversions to be carried out on larger datasets in practical lengths of time (e.g. a few hours or less). The project was able to reduce inversion runtimes by more than a factor of 10 on a multicore workstation with the potential for speedups of over 100 on a cluster.

This thesis will describe the principles underlying ArjunAir's main algorithms, the modifications undertaken to improve its performance and the results achieved. The remainder of this introductory chapter will give an overview of the principles of airborne EM surveying methods and discuss the major approaches to modelling and inverting airborne EM data, which will motivate the 2.5D approach. The following chapter will describe the theoretical basis for ArjunAir's main algorithms. The third and fourth chapters will discuss the computational techniques used in the forward modelling and inversion components of the program, respectively. Those discussions will be interwoven with the presentation of

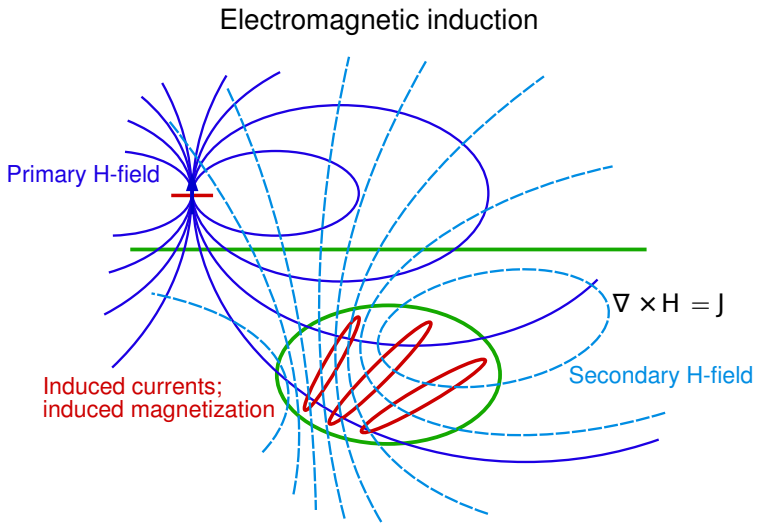


Figure 1.2: Conceptual illustration of EM induction in the earth. Adapted from Farquharson (personal communication).

results. The final chapter presents some brief concluding remarks.

1.2 Principles of Airborne EM

All airborne EM methods are based on Faraday's principle of electromagnetic induction, which states that a time varying magnetic field will induce the creation of an electric field (Telford et al., 1990). An airborne EM system generally consists of two main components, the transmitter and the receiver, which are both circular coils of wire. A schematic of a generic system is shown in Figure 1.2. Magnetic fields are generated by electric currents, according to Ampere's law. An EM transmitter generates a time varying magnetic field, called the primary field, by running a time-varying current through a wire coil. By the principle of induction, this will induce electric fields throughout space. These applied electric fields will cause electric currents to flow in any subsurface bodies with sufficient electrical conductivity. Conductivity is a material property that can often be diagnostic of

subsurface geology (Telford et al., 1990). The exact relationship between applied electric fields and the electric currents they produce can vary from material to material but for most everyday substances it can be very well approximated by Ohm's law, which states that an applied electric field in a material will produce an electric current in the direction of—and with magnitude proportional to—the applied field. The constant of proportionality is the electrical conductivity.

The electric currents in the subsurface are called secondary currents. By Ampere's law, they will generate their own time varying magnetic fields, called secondary fields (Telford et al., 1990). The secondary magnetic fields will induce their own electric fields, causing electric currents to flow in the EM measurement system's receiver coil. In an EM survey, strong measured secondary fields will indicate areas of elevated subsurface conductivity.

There are two main categories of airborne EM surveying systems, frequency-domain and time-domain (Palacky and West, 1991). Frequency domain systems contain up to six pairs of transmitter and receiver coils. Each transmitter emits a continuous magnetic field varying sinusoidally in time and its corresponding receiver is tuned to the frequency of that sinusoid. In a time-domain system, a short pulse of current is run through the transmitter coil, generating a short lived primary field. The secondary fields will rapidly decay after the transmitter is turned off. The receiver measures that decay. High quality time-domain EM datasets arguably contain more information than frequency-domain datasets but are more difficult to acquire (Nabighian and Macnae, 1991). Due to the technical difficulty of acquiring good time-domain data, frequency-domain surveying was the dominant technique in the early days of geophysical EM surveying. Both types are widely used today. ArjunAir can work with datasets from both survey types.

1.3 Introduction to EM modelling and inversion

1.3.1 Forward modelling

In a typical time or frequency-domain survey, an aircraft will fly over the survey area along a set of parallel lines, measuring the steady-state total fields at fixed time intervals along each line. The secondary fields are then computed by removing the known primary field. The behaviour of the electromagnetic fields generated in an EM survey are described mathematically by Maxwell's equations, the fundamental laws of classical electromagnetism. Maxwell's equations may be formulated as a set of coupled linear partial differential equations (PDEs) for the electric and magnetic fields. Electrical conductivity enters these equations as a coefficient. Solving Maxwell's equations for the fields that would be produced by an EM surveying system, given a model of the earth's conductivity, is known as forward modelling. Given a mathematical description of the transmitter and appropriate boundary conditions, solving for the electric and magnetic fields is a well-posed problem (Hohmann, 1987).

In their canonical form, Maxwell's equations are hyperbolic PDEs in space and time (Jackson, 1999). However, if one assumes a sinusoidal time dependence for the fields, as in frequency-domain surveying, they may be formulated as complex-valued elliptic PDEs. These are known as Maxwell's equations in the frequency-domain. Even for time-domain modelling, it is common to solve Maxwell's equations in the frequency-domain at a range of different frequencies and then recover the time-domain fields by inverse Fourier transformation. This is the approach taken in ArjunAir for time-domain modelling.

The difficulty of solving the forward modelling problem depends on the complexity of the transmitter and the assumed conductivity model. Airborne EM transmitters are mod-

elled as ideal magnetic dipoles. An exact solution in terms of elementary functions exists for a magnetic dipole source above a homogeneous earth with a flat surface. If the earth's conductivity is assumed to be laterally constant and to vary only with depth, then a solution is known expressing the electromagnetic fields in terms of Hankel transform integrals (Ward and Hohmann, 1987). Exact solutions also exist for some simple geometrical shapes embedded in homogeneous halfspaces (Telford et al., 1990). Maxwell's equations have not been solved exactly for a general 3D conductivity distribution but reliable numerical techniques are well established (e.g. Börner, 2010)

3D numerical solutions are generally robust and accurate but computing them requires extensive CPU and memory resources. Two-dimensional modelling offers a compromise between the computational difficulties of modelling in three dimensions and the oversimplification of the 1D layered earth approximation. The 2D approximation also makes sense given that airborne survey data are normally collected along straight lines. A two-dimensional inversion will seek to estimate the lateral and depth variations of conductivity along a survey line, while assuming that conductivity does not vary in the direction perpendicular to the line direction. The approximation will be valid when surveying conductors of long strike length, with flight lines perpendicular to strike direction. A conceptual illustration of a survey over a body of long strike length is shown in Figure 1.3. Such geological targets occur frequently enough in the field to make 2D modelling a useful tool.

Unfortunately, the 3D nature of the primary fields emitted by airborne EM transmitters makes pure 2D modelling impossible. The transmitters used in most if not all airborne EM surveys can be modelled as magnetic dipoles (Telford et al., 1990). Such transmitters create primary and secondary fields that vary in three dimensions, regardless of conductivity structure. However, if a 2D conductivity model is assumed, the 3D forward problem may

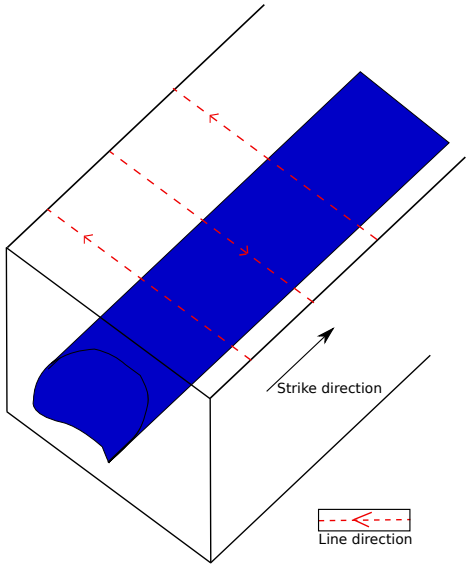


Figure 1.3: Airborne EM survey over a target of infinite strike length. Based on Figure 1.3 in Yu (2012).

be decomposed into a set of independent subproblems that may be solved numerically by discretization (e.g. finite element, finite volume) over a 2D domain. This decomposition is achieved by setting the geological strike direction to be one of the three Cartesian coordinates and then Fourier transforming Maxwell's equations with respect to that chosen coordinate. Once these 2D subproblems have been solved in the Fourier domain, the results may be (inverse) transformed to give the secondary fields along a survey line. This approach to modelling an EM system with a 3D source and 2D earth model is called 2.5D modelling.

Stoyer and Greenfield (1976) published the first 2.5D forward modelling algorithm. They presented a frequency-domain finite-difference solution for the electric and magnetic fields due to a 2D earth excited by a 3D magnetic dipole source. Lee and Morrison (1985) presented a finite-element solution to the same problem. Unsworth (1991) and Everett and Edwards (1992) published finite-element solutions for electric dipole sources, which are

used in marine EM surveying. ArjunAir's forward modelling routine is based on the finite element approach taken by Sugeng et al. (1993). Fred Sugeng and Art Raiche, two of the authors of that paper, were two of the three principle developers of ArjunAir. A finite-volume solution for the 2.5D airborne EM problem was recently formulated by Yu (2012).

Aside from Yu, recent work on 2.5D EM has been focused on the marine problem. Mitsuhata (2000) published a finite-element solution for electric dipole sources using a very similar approach to the one taken by ArjunAir. Abubakar et al. (2008) presented a finite-difference solution using the optimal grid technique of Ingerman et al. (2000) to reduce the number of cells required to achieve an accurate solution. Key and Ovall (2011) formulated an adaptive finite-element solution using hierarchical basis functions on triangular unstructured meshes.

Progress on the marine EM problem has been impressive. However, ArjunAir remains the only widely available 2.5D code for airborne EM modelling. Improving its performance will allow it to be used on a wider range of problems and fill an important niche in EM modelling and inversion.

1.3.2 Inversion

1.3.2.1 Overview

In an airborne EM survey, the main problem to be solved is the inverse of forward modelling. The electromagnetic fields are measured and the goal is to recover the subsurface conductivity as a function of position. Attempting to rigorously estimate physical properties of the subsurface, such as electrical conductivity, from geophysical survey data is known as geophysical inversion (e.g. Hohmann and Raiche, 1987; Aster et al., 2013). In-

version is a data fitting problem.

In the case of airborne EM, conductivity is a continuously varying function of position, which the geophysicist must try to recover from a finite number of noisy data. This makes the inverse problem inherently ill posed (Parker, 1994). There will be an infinite number of conductivity models that may fit a given dataset equally well. Any practical inversion procedure is therefore subjective since some assumptions about the subsurface conductivity structure must be made in order to create a deterministic inversion algorithm. The problem is complicated by the fact that the relationship between the observed fields and the conductivity of the earth is non-linear.

Compared to the number of published forward solutions, there has been little work on the 2.5D inverse problem. Early inversion algorithms (for both EM and other types of geophysical data) assumed very simple models of subsurface conductivity such as layered earths with a small number of layers or simple geometrical shapes embedded in homogeneous halfspaces. Non-linear least squares methods were used to estimate the parameters of these simple earth models (e.g. Glenn et al., 1973). Limiting possible earth models to those with a smaller number of parameters than the number of observed data is a very restrictive assumption that is unreasonable for most airborne EM applications. The next generation of inversion algorithms divided the subsurface into a large number of thin horizontal layers of fixed thickness, with the number of layers potentially much greater than the number of observed data. The goal of the inversion is then to find the conductivity (or other physical property) of each layer in the model. The geophysicist must make additional assumptions about the desired characteristics of the recovered model in order to define a problem with a unique solution that may be solved by non-linear optimization methods.

The simplest assumption is to choose the model that is as close as possible to an a priori

reference model. Such a solution may be found by regularized least squares methods such as the Levenburg-Marquardt algorithm or zeroeth order Tikhonov regularization (Aster et al., 2013). The major downside of this approach is that it tends to fail if a reference model reasonably close to the true model cannot be chosen a priori. This procedure may be made more robust and even independent of any a priori model by imposing additional constraints on the characteristics of the desired model through more complex forms of Tikhonov regularization. The two most common examples of this approach are choosing the most spatially smooth model that fits the data, known as Occam, or minimum structure, inversion (e.g. Constable et al., 1987; Haber et al., 2007; Key, 2012), and choosing the model whose physical property variations are the most spatially compact (e.g. Last and Kubik, 1983; Cox et al., 2010). Despite their ability to robustly find physical property models that fit observed data, these approaches bias inversions toward certain types of models that may or may not reflect geology. For example, minimum structure inversions cannot recover sharp physical property transitions, which occur commonly in the earth. One rationale for the minimum structure approach in EM is that conductive features will only be included in the model if they are absolutely required by the data.

1.3.2.2 Computational issues

Theoretically, any of the approaches discussed in the last paragraph may be readily extended to 2D and 3D conductivity models. In the 2D EM case, conductivity is assumed to be constant in one horizontal direction (the strike direction) and is allowed to vary with depth and the other horizontal direction. A 2D section of the earth is divided into small polygonal cells of constant conductivity. The goal of the inversion is to recover the conductivity of each cell. In 3D the earth is divided into polyhedra of constant conductivity.

The main difficulty in implementing 2D and 3D EM inversions is computational. Non-linear Tikhonov regularized least-squares EM inversion algorithms are iterative in nature and generally involve a linearization of the problem at each iteration. Solving the linearized inverse problem requires computing its Jacobian, or sensitivity, matrix. Computing the Jacobian is very expensive and the computational cost grows strongly with the number of unknown parameters in the inversion. Storing the Jacobian in memory is generally impossible for 3D problems. Fortunately, the inverse problem may normally be posed such that the Jacobian need not be stored explicitly, as long as its action on a vector can be computed (Haber et al., 2000). Computing the action of the Jacobian normally requires one or more forward modellings to be performed. Additionally, forward modelling must be performed at least once more per iteration to compute misfit between the observed data and the theoretically predicted data computed by forward modelling with the current earth model estimate.

The cost of computing the Jacobian (or its action on a vector) and estimating the data misfit are significant constraints that limit the number of parameters that may be estimated by a practical inversion code. The number of parameters grows quickly, of course, with the dimensionality of the model. As mentioned above, for a 1D earth, the forward solution is known in terms of a Hankel transform integral. These integrals may be numerically evaluated very efficiently (e.g. Christensen, 1990). On a modern consumer grade PC a 1D forward solution may be computed in much less than a second (e.g. Ray and Key, 2012) and a full inversion will take no more than a few seconds. For 3D conductivity models, where numerical techniques such as finite element or finite difference methods must be used to solve Maxwell's equations, inversions may take hours or days to run for practical datasets. As alluded to in the first section of this chapter, 3D EM inversion is still in its

infancy. A couple of software packages exist but they are too computationally intensive for widespread use, likely requiring supercomputing resources to run inversions on large datasets (Cox et al., 2010; Oldenburg et al., 2013).

ArjunAir takes the approach of finding a 2D conductivity model that is as close as possible to a reference model, with no constraints on the smoothness or compactness of conductivity variations. Since solving the forward problem with a 2D model requires the 2.5D method, controlled source electromagnetic inversion methods using 2D earth models are known as 2.5D inversion methods. To my knowledge, the only 2.5D inversion program aside from ArjunAir capable of handling airborne EM data is the one developed by Yu (2012). It was coded in Matlab and to my knowledge was not released to the public. That makes ArjunAir the only production quality 2.5D airborne EM modelling and inversion code.

Three 2.5D marine EM inversion programs (to my knowledge) have recently been published in the geophysical literature (e.g. Abubakar et al., 2008; Ramananjaona and MacGregor, 2010; Key, 2012). The main difference in the implementation of airborne and marine EM programs is in the type of transmitter used. Marine surveying systems normally use electric dipole transmitters.

Given that several subproblems on the full 2D mesh must be solved in order to construct a 2D conductivity model, one may wonder if 2.5D inversion is much more efficient than full 3D inversion. It is, and the key reason is that the subproblems are completely independent so the amount of work required scales linearly with the number of subproblems. The number of subproblems required normally depends on the distance in the along-strike direction the fields take to decay to a negligible level. That same factor controls the number of cells in the along-strike direction required for 3D modelling and inversion of long

strike length conductors. However, amount of work required for a 3D inversion scales as the cube of the number of cells in the along-strike direction. This reasoning was given in the paper on 2.5D forward modelling by Sugeng et al. (1993). For conductors of limited strike length, where boundary effects in the strike direction are important, 2.5D modelling will be inaccurate, compared to full 3D modelling. This is more likely to be a problem at lower frequencies, for which the fields decay more slowly as a function of distance from the transmitter. Another limitation is that even if the geology is 2D the strike direction must be known and the survey lines oriented perpendicular to strike if the 2.5D approach is to be useful.

1.4 Summary

This project has taken a program that fills an important niche in EM data interpretation software and made it practical to run on much larger datasets than was previously possible, probably allowing full airborne surveys to be inverted line by line. Additionally, the project proved that it is possible to take a piece of legacy geophysics software and modernize it using parallel programming and high performance mathematical software libraries.

Chapter 2

Theory

This chapter will review the theory behind the forward modelling and inversion capabilities of ArjunAir. It will start by describing the fundamentals of classical electromagnetism and the formulation of Maxwell's equations solved in ArjunAir. The second main section of the chapter describes ArjunAir's original inversion algorithm and the modified version developed for this thesis.

2.1 Forward modelling

2.1.1 Fundamentals of classical electromagnetism

Geophysical electromagnetic forward modelling involves mathematically modelling the behaviour of macroscopic electromagnetic (EM) fields in the earth. Given a source of electric charge or current and a model of the electromagnetic physical properties of the earth, forward modelling seeks to calculate the resulting electric and magnetic fields. In airborne EM, sources are usually small loops of current that can be modelled as magnetic

dipoles (i.e. point sources) (Palacky and West, 1991).

Macroscopic EM fields are governed by Maxwell's equations. Maxwell's equations can be written as a set of coupled, vector-valued, linear partial differential equations (PDEs) for the fields in space and time. The electromagnetic physical properties of the earth enter the equations as coefficients. The task of forward modelling consists, conceptually, of constructing a physical property model, in order to define the coefficients, and then solving Maxwell's equations. In canonical differential form, Maxwell's equations are (Jackson, 1999)

$$\begin{aligned}
\nabla \times \mathbf{e} + \frac{\partial \mathbf{b}}{\partial t} &= 0 && \text{(Faraday's law)}, \\
\nabla \times \mathbf{h} - \frac{\partial \mathbf{d}}{\partial t} &= \mathbf{j} && \text{(Ampere's law, corrected)}, \\
\nabla \cdot \mathbf{d} &= \frac{\rho_{\text{free}}}{\epsilon} && \text{(Gauss's law)}, \\
\nabla \cdot \mathbf{h} &= 0 && \text{(un-named)},
\end{aligned} \tag{2.1}$$

where $\mathbf{e}$ is the electric field and $\mathbf{b}$ is called the magnetic induction. The variable $\mathbf{h}$ is the magnetic field intensity, $\mathbf{d}$ is the electric displacement, $\mathbf{j}$ is electric current, and ρ_{free} is free electric charge density.

The magnetic induction is related to the magnetic field intensity, and the electric field to the electrical displacement, through constitutive relations. The electromagnetic constitutive relations are a set of empirical relations that define the relationship between electromagnetic fields in a given substance. They describe how bulk materials react to applied $\mathbf{e}$ and $\mathbf{b}$ fields (Jackson, 1999). Thus, $\mathbf{b}$ can be thought of as an externally applied magnetic field, and $\mathbf{h}$ as the total magnetic field that includes $\mathbf{b}$ as well as fields generated inside materials by $\mathbf{b}$. $\mathbf{d}$ describes an electric field stimulated in a material due to an applied electric field. Maxwell's equations are macroscopic approximations of the behaviour of EM fields which,

fundamentally, originate from the behaviour of microscopic electric charges and are governed by the laws of quantum electrodynamics (Jackson, 1999). The constitutive relations provide a way to model the bulk electromagnetic behaviour of materials without having to consider the microscopic origins of the behaviour.

Additionally, $\mathbf{e}$ and $\mathbf{j}$ are related by a third constitutive law. Electrical current density, $\mathbf{j}$ represents moving electric charge. In geophysical contexts, electrical currents are almost always the result of applied electric fields (Ward and Hohmann, 1987). The third EM constitutive relation describes how current will flow in response to a given electric field.

In general, constitutive relations might depend on the position, orientation, frequency, and strength of the fields. In mineral exploration applications, the constitutive relations are most often taken to be linear and isotropic (Ward and Hohmann, 1987):

$$\mathbf{d} = \epsilon \mathbf{e} \tag{2.2}$$

$$\mathbf{b} = \mu \mathbf{h} \tag{2.3}$$

$$\mathbf{j} = \sigma \mathbf{e}, \tag{2.4}$$

where the scalars ϵ, μ, and σ are known as the dielectric permittivity, magnetic permeability, and electrical conductivity, respectively. In the linear isotropic case, they may still depend on frequency, position, and other parameters. In most mineral exploration scenarios ϵ and μ are set to their vacuum values ϵ_0 and μ_0 but σ is free to vary over several orders of magnitude. ArjunAir allows all three parameters to vary fully but only vacuum values of μ and ϵ were used in this work.

It is possible to remove the time dependence of Maxwell's equations by Fourier transforming them with respect to time. This is equivalent to assuming an $e^{i\omega t}$ time dependence for the fields. Let the Fourier transform with respect to time be defined by the following

pair of integrals (Osgood, 2007):

$$F(x,y,z,\omega) = \int_{-\infty}^{\infty} f(x,y,z,t)e^{-i\omega t}\,\mathrm{d}t,$$
$$f(x,y,z,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(x,y,z,\omega)e^{i\omega t}\,\mathrm{d}\omega. \tag{2.5}$$

EM fields encountered in geophysics are often harmonic and even when they are not, working in the frequency-domain can often be more convenient, computationally, than solving Maxwell's equations by direct time-stepping methods (Sugeng et al., 1993). Fourier transforming and invoking the constitutive relations, Faraday's and Ampere's laws in the frequency-domain can be written as

$$\begin{aligned} \nabla \times \mathbf{E} + i\omega\mu\mathbf{H} &= 0 \\ \nabla \times \mathbf{H} - (\sigma + i\epsilon\omega)\mathbf{E} &= \mathbf{J}_{\text{source}}, \end{aligned} \tag{2.6}$$

where $\mathbf{J}_{\text{source}}$ is an applied current, often called the source current. They are now elliptic rather than hyperbolic differential equations.

Frequency-domain EM systems generate harmonic fields and for such surveys, solutions to Maxwell's equations in the frequency-domain constitute the full solution of the forward problem. Time domain surveys produce anharmonic fields. The evolution in time of anharmonic EM fields may be approximated by solving equations (2.6) at several fixed frequencies. Numerical inverse Fourier transformation of the frequency-domain fields will yield the approximate time-domain behaviour (Sugeng et al., 1993). ArjunAir takes this approach and does not solve Maxwell's equations by direct time-stepping methods.

Another advantage of working in the frequency-domain is the ability to model induced polarization (IP) effects (Wilson et al., 2006). In the Cole-Cole model, the most common way IP effects are understood in geophysics (Telford et al., 1990), IP effects are modelled

in terms of a frequency dependent conductivity. ArjunAir uses the Cole-Cole model, and defines conductivity using the formula

$$\sigma(\omega) = \sigma_0 \frac{1 + (i\omega\tau)^c}{1 + (1-m)(i\omega\tau)^c} + i\omega\epsilon, \tag{2.7}$$

where τ, m, and c are user chosen empirical IP parameters and σ_0 is the direct current (DC) conductivity. ArjunAir allows forward modelling of full IP effects but can only invert for DC conductivity. IP effects were not considered in this work. Using this expression for σ, Ampere's law can be written compactly as

$$\nabla \times \mathbf{H} - \sigma(\omega)\mathbf{E} = \mathbf{J}_{\text{source}}. \tag{2.8}$$

It is important to note here that ArjunAir uses the full-wave version of Maxwell's equations. It does not employ the quasi-static approximation. The non-quasi-static permittivity term is included as part of the conductivity expression in (2.7) and therefore does not appear explicitly in (2.8).

2.1.2 Primary-secondary field separation

Airborne EM systems transmit EM fields into the ground by running a time varying electric current through a loop that is small in comparison with its height above the ground. Such a transmitter can be idealized as a perfect magnetic dipole (Palacky and West, 1991). ArjunAir models airborne EM transmitting antennas as magnetic dipoles, which are represented mathematically by spatial delta functions:

$$\mathbf{J}_{\text{source}} = \mathbf{m}\delta(\mathbf{x}) \tag{2.9}$$

where $\mathbf{m}$ is the dipole moment of the transmitter. In order to avoid modelling the delta function directly, and to separate fields generated in the earth from those propagating directly

from the transmitter to the receiver, ArjunAir uses a primary-secondary field separation.

Following Hohmann (1987), I will go over the basics of primary-secondary field separation. The total electric and magnetic fields in an EM survey measurement can be divided into components due to fields generated in the ground by regions of anomalous conductivity (the secondary field) and those due to a source field in a background earth model (the primary field). The background conductivity structure is normally a simple one such as a layered earth, for which the fields can be computed directly by integration.

Consider background conductivity and permeability models σ_b and μ_b. Maxwell's equations for the primary fields with background physical properties and source $\mathbf{J}_{\text{source}}$ are

$$\begin{aligned} \nabla \times \mathbf{E}^p + i\omega\mu_b\mathbf{H}^p &= 0 \\ \nabla \times \mathbf{H}^p - \sigma_b\mathbf{E}^p &= \mathbf{J}_{\text{source}}. \end{aligned} \tag{2.10}$$

The secondary fields are given by subtracting the primary fields, $\mathbf{E}^p$ and $\mathbf{H}^p$, from the total fields:

$$\begin{aligned} \mathbf{E}^s &= \mathbf{E} - \mathbf{E}^p \\ \mathbf{H}^s &= \mathbf{H} - \mathbf{H}^p. \end{aligned} \tag{2.11}$$

A set of PDEs for the secondary fields can be easily derived by subtracting equations (2.10) from the total field equations, (2.6), and invoking the linearity of the curl operator. The secondary field equations are:

$$\begin{aligned} \nabla \times \mathbf{E}^s + i\omega(\mu\mathbf{H}^s + \mu_a\mathbf{H}^p) &= 0 \\ \nabla \times \mathbf{H}^s - \sigma\mathbf{E}^s &= \sigma_a\mathbf{E}^p, \end{aligned} \tag{2.12}$$

where σ_a and μ_a are the anomalous conductivity and permeability, respectively. $\mu_a = 0$ in most mineral exploration EM applications and it will always be small. Although ArjunAir allows the total permittivity μ to be set arbitrarily, it assumes μ_a will be small and ignores

it in the secondary field equations. μ_a will be ignored in the remainder of this discussion. Ignoring μ_a eliminates the need to compute $\mathbf{H}^p$ and simplifies the secondary field equations to

$$\begin{aligned} \nabla \times \mathbf{E}^s + i\omega\mu\mathbf{H}^s &= 0 \\ \nabla \times \mathbf{H}^s - \sigma\mathbf{E}^s &= \sigma_a\mathbf{E}^p. \end{aligned} \tag{2.13}$$

In the secondary field equations, the source term $\mathbf{J}_{\text{source}} = \mathbf{m}\delta(\mathbf{x})$ is replaced by the primary field term $\sigma_a\mathbf{E}^p$. If $\mathbf{E}^p$ is known, the Cartesian components of equations (2.13) form a set of six coupled scalar PDEs for the components of $\mathbf{E}^s$ and $\mathbf{H}^s$. Only Cartesian coordinate systems will be considered in this thesis. The difficulty of computing $\mathbf{E}^p$ depends on the complexity of σ_b. It is normally chosen to be simple enough for $\mathbf{E}^p$ to be computed directly by, at worst, the numerical evaluation of an integral—see e.g. Sugeng et al. (1993).

ArjunAir defines the primary field to be that due to a magnetic dipole in free space. Thus in equations (2.13) the anomalous conductivity, σ_a, is equal to the total conductivity of the earth, σ. In the frequency and time domains, the primary field is known in terms of a closed form elementary algebraic expression (Ward and Hohmann, 1987).

2.1.3 The 2.5D problem

The secondary field equations, as stated in (2.13), are valid for an arbitrary 3D conductivity distribution. In this thesis I am interested in computing the response of a 2D earth to an airborne EM system (i.e. compute $\mathbf{H}_s$ and $\mathbf{E}_s$ or $\mathbf{h}_s$ and $\mathbf{e}_s$) on a closed 2D domain in the x-z Cartesian plane. Let y be the geoelectric strike direction and assume that conductivity and magnetic permeability are constant in the y-direction. Even with 2D physical properties, it is impossible to solve equations (2.13) purely on a 2D domain for magnetic dipole sources using standard numerical methods for differential equations.

In the secondary field equations, the 3D nature of the fields, imparted by the source, is manifested by the primary field, $\mathbf{E}_p$. In ArjunAir, $\mathbf{E}_p$ is the field due to a harmonic magnetic dipole of frequency ω and unit magnetization, oriented in the x-z plane at an angle θ from the z axis. It is given in Cartesian coordinates by

$$\mathbf{E}_p(x, y, z) = \frac{i\omega\mu}{4\pi r^3} \left(y \cos\theta \hat{\mathbf{x}} + (z \sin\theta - x \cos\theta)\hat{\mathbf{y}} + y \sin\theta \hat{\mathbf{z}} \right), \tag{2.14}$$

where $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are unit vectors; x, y, and z are the distances from the dipole in the three Cartesian directions, and $r = \sqrt{x^2 + y^2 + z^2}$. The field obviously depends on y in a non-negligible way. Solutions to equation (2.13) will depend on y and the y-derivative terms cannot be eliminated from the equation. Therefore it is not possible to solve the equations using a 2D discretization.

However, since conductivity and the rest of the geo-electric physical parameters are constant with respect to (w.r.t.) the y coordinate, they remain unchanged under a Fourier transform (FT) w.r.t. y. Performing that FT on the scalar components of equation (2.13) allows a set of two coupled 2D PDEs to be derived for the along-strike components of the secondary electric and magnetic fields in the spatial wavenumber domain, $\tilde{E}^s_y, \tilde{H}^s_y = \tilde{E}^s_y, \tilde{H}^s_y(x, z, \omega, k_y)$, where k_y is the y-direction wavenumber. For a given frequency, the wavenumber domain equations may be solved at a number of constant values of k_y. Frequency domain behaviour can then be recovered by numerical inverse Fourier transform of the k_y-domain solutions.

The coupled set of wavenumber domain PDEs will now be derived from equations (2.13) following Hohmann (1987). The Fourier transform w.r.t. y is defined in the same

manner as equation (2.5):

$$\mathfrak{F}_y\{f(x,y,z,\omega)\} = \int_{-\infty}^{\infty} f(x,y,z,\omega)e^{-ik_y y}\,\mathrm{d}y,$$
$$f(x,y,z,\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathfrak{F}_y(x,k_y,z,\omega)e^{ik_y y}\,\mathrm{d}k_y. \tag{2.15}$$

Recall the formula for the Fourier transform of a derivative (Osgood, 2007)

$$\mathfrak{F}_y\left\{\partial_y f(y)\right\} = ik_y f(k_y).$$

The Fourier transforms of the components of (2.13) are

$$ik_y\tilde{E}^s_z - \partial_z\tilde{E}^s_y + i\omega\mu\tilde{H}^s_x = 0, \qquad ik_y\tilde{H}^s_z - \partial_z\tilde{H}^s_y - \sigma\tilde{E}^s_x = \sigma_a\tilde{E}^p_x, \tag{2.16a}$$

$$\partial_z\tilde{E}^s_x - \partial_x\tilde{E}^s_z + i\omega\mu\tilde{H}^s_y = 0, \qquad \partial_z\tilde{H}^s_x - \partial_x\tilde{H}^s_z - \sigma\tilde{E}^s_y = \sigma_a\tilde{E}^p_y, \tag{2.16b}$$

$$\partial_x\tilde{E}^s_y - ik_y\tilde{E}^s_x + i\omega\mu\tilde{H}^s_z = 0, \qquad \partial_x\tilde{H}^s_y - ik_y\tilde{H}^s_x - \sigma\tilde{E}^s_z = \sigma_a\tilde{E}^p_z. \tag{2.16c}$$

where the tildes denote wavenumber domain quantities. Equations (2.16a) and (2.16c) can be combined to form expressions for $\tilde{E}^s_x$, $\tilde{E}^s_z$, $\tilde{H}^s_x$, and $\tilde{H}^s_z$ in terms of $\tilde{E}^s_y$, $\tilde{H}^s_y$, and the primary fields. Substituting those expressions into equations (2.16b) and assuming $\tilde{\mathbf{E}}^p$ to be known gives a system of two complex-valued linear inhomogeneous PDEs for $\tilde{E}^s_y$ and $\tilde{H}^s_y$:

$$\nabla\cdot(\frac{\sigma}{k_e^2}\nabla\tilde{E}^s_y) - ik_y\nabla\cdot(\mathbf{A}\nabla\tilde{H}^s_y) - \sigma\tilde{E}^s_y = \sigma_a\tilde{E}^p_y - ik_y\nabla\cdot\left[\frac{\sigma_a}{k_e^2}\begin{pmatrix}\tilde{E}^p_x\\ \\ \tilde{E}^p_z\end{pmatrix}\right], \tag{2.17}$$

$$\nabla\cdot(\frac{1}{k_e^2}\nabla\tilde{H}^s_y) + \frac{k_y}{\omega\mu}\nabla\cdot(\mathbf{A}\nabla\tilde{E}^s_y) - \tilde{H}^s_y = \partial_x\left(\frac{\sigma_a}{k_e^2}\tilde{E}^p_z\right) - \partial_z\left(\frac{\sigma_a}{k_e^2}\tilde{E}^p_x\right), \tag{2.18}$$

where the gradient is defined in 2D, $k_e^2 = k_y^2 + i\omega\mu\sigma$ and

$$\mathbf{A} = \frac{1}{k_e^2} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

These are equations 107 and 108 in Hohmann (1987). Other components of $\tilde{\mathbf{E}}^s$ and $\tilde{\mathbf{H}}^s$ may be recovered by numerical differentiation. The primary reason for choosing to solve for the y-components of the wavenumber domain secondary fields is that they are continuous in x and z. Electromagnetic fields are always continuous inside homogeneous materials. All magnetic field components and the tangential component of the electric field are continuous at boundaries between media of differing conductivity. The normal component of the electric field will be discontinuous across conductivity boundaries (Ward and Hohmann, 1987). Since $\tilde{E}_y^s$ is tangential to all conductivity boundaries in a 2D model, $\tilde{E}_y^s$ and $\tilde{H}_y^s$ will always be continuous in 2D.

ArjunAir solves the system of PDEs (2.17) and (2.18) using an isoparametric finite-element method. Continuity of $\tilde{E}_y^s$ and $\tilde{H}_y^s$ allows for the use of node based finite elements in solving the equations, avoiding the complication of using vector elements (Jin, 2002). Equations (2.17) and (2.18) may be written more compactly as a single vector PDE. The compact vector form will be useful in deriving the finite element approximation to the boundary value problem (BVP) defined by the PDEs (2.17) and (2.18) and appropriate

boundary conditions. Note the following definitions:

$$\mathbf{u} = \begin{pmatrix} \tilde{H}_y^s \\ \tilde{E}_y^s \end{pmatrix} \qquad \nabla\mathbf{u} = \begin{pmatrix} \partial_x \tilde{H}_y^s & \partial_z \tilde{H}_y^s \\ \partial_x \tilde{E}_y^s & \partial_z \tilde{E}_y^s \end{pmatrix} \tag{2.19}$$

$$\mathbf{K}_1 = \frac{1}{k_e^2} \begin{pmatrix} 1 & 0 \\ 0 & \sigma \end{pmatrix} \qquad \mathbf{K}_2 = \frac{1}{k_e^2} \begin{pmatrix} 0 & \frac{k_y}{\omega\mu} \\ -ik_y & 0 \end{pmatrix} \tag{2.20}$$

$$\mathbf{P} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \qquad \mathbf{K}_3 = \begin{pmatrix} 1 & 0 \\ 0 & \sigma \end{pmatrix} \tag{2.21}$$

$$\mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}, \tag{2.22}$$

where, f_1 and f_2 are the right hand sides of equations (2.18) and (2.17), respectively. Additionally, for any matrix $\mathbf{X} \in \mathbb{C}^{2\times 2}$, $\nabla \cdot \mathbf{X}$ is defined to be the vector who's i^{th} element is the divergence of the i^{th} row of $\mathbf{X}$. Using those definitions, equations (2.17) and (2.18) can be written as the vector PDE

$$\nabla \cdot (\mathbf{K}_1 \nabla \mathbf{u}) + \nabla \cdot (\mathbf{K}_2 \nabla \mathbf{u}\, \mathbf{P}) - \mathbf{K}_3 \mathbf{u} = \mathbf{f}. \tag{2.23}$$

For a set of frequencies, ω, and wavenumbers, k_y, ArjunAir solves the boundary value problem

$$\nabla \cdot (\mathbf{K}_1 \nabla \mathbf{u}) + \nabla \cdot (\mathbf{K}_2 \nabla \mathbf{u}\, \mathbf{P}) - \mathbf{K}_3 \mathbf{u} = \mathbf{f} \ \text{ in } \ \Omega, \tag{2.24a}$$

$$\mathbf{u} = 0 \ \text{ on } \ \partial\Omega, \tag{2.24b}$$

$$\Omega = \left\{ (x, z) \in \mathbb{R}^2 : x_1 < x < x_2, \ \ z_1 < z < z_2 \right\}, \tag{2.24c}$$

where $\partial\Omega$ is the boundary of Ω. x_1 and x_2 are the lateral limits of the domain, and z_1 and z_2 are the vertical limits. The field amplitudes decay asymptotically toward zero far from the

source location. Therefore, Ω must be taken large enough for the fields to be approximately zero on the boundary.

Having described how to compute the secondary EM fields in the k_y wavenumber domain, the entire forward modelling process may now be summarized. The secondary fields at all nodes in the finite element mesh in the wavenumber domain are required in computing the sensitivity matrix in the ArjunAir inverse problem. Frequency and (possibly) time-domain fields only need to be known at so-called observation locations, the locations of EM receivers in a survey. Consequently, transformation of the wavenumber domain fields to the frequency and time domains only needs to be performed at the observation locations. The conceptual structure and workflow of the forward modelling process is outlined in Algorithm 2.1. This conceptual algorithm corresponds to solving the forward problem

Algorithm 2.1 Conceptual ArjunAir forward solve procedure.

Specify an earth model $\sigma(x, z, \omega)$, $\mu(x, z)$
Specify a source current $\mathbf{J} = \mathbf{m}(t)\delta(x - x_0, z - z_0)$
Choose set of n_f frequencies: $\{\omega_i\}$
Choose set of n_y wavenumbers: $\{k_j^y\}$
for $i = 1$ to n_f **do**
    for $j = 1$ to n_y **do**
        Compute primary electric field $\tilde{\mathbf{E}}^p$, given $\mathbf{J}$
        Solve equations (2.17) and (2.18) with $\omega = \omega_i$, $k^y = k_j^y$
        Find other field components by numerical differentiation
        Find field values at observation locations and store them
    end for
    Spline and interpolate observation location fields as functions of k^y
    Use interpolated k^y domain fields in numerical inverse FT to recover frequency-domain fields for ω_i
end for
Spline and interpolate observation location fields as functions of ω
Use set of interpolated frequency-domain fields in numerical inverse FT to recover time-domain fields

for a single source. In a typical airborne EM survey there are 1-3 observations per source location, with an entire survey comprising observations taken at many locations. However, the ArjunAir forward solver is implemented such that solving for many source locations requires very little computation, relative to the cost of solving for one location.

In addition to ArjunAir, several other 2.5D EM software packages based on equations (2.17) and (2.18) have been developed. Everett and Edwards (1992), Kong et al. (2008), and Key and Ovall (2011) developed 2.5D marine controlled-source EM codes, using electric dipole sources. Mitsuhata (2000) solved a system of equations having the same left hand side structure as (2.17) and (2.18) but modelled sources directly as pseudo delta functions. He also did not consider magnetic dipole sources. However, adapting any of these codes to use magnetic sources would be simple. Stoyer and Greenfield (1976) published the first numerical solution of equations (2.17) and (2.18). They considered both electric and magnetic sources.

2.1.4 Solving Maxwell's equations in the k_y domain

2.1.4.1 Galerkin's method and the weak form of a BVP

As mentioned above, ArjunAir uses an isoparametric finite-element method to solve the BVP (2.24). This section will provide an overview of the finite element method, as employed by ArjunAir. Derivation of the finite element equations is based on the approach to the Galerkin finite element method taken by Gockenbach (2006), with reference to the book by Brenner and Scott (2008).

Galerkin's method approximates the solution $u \in W$—for some function space W to be specified later—of an elliptic PDE or system of PDEs by finding the projection of u onto a

finite-dimensional subspace of W, which will be denoted L for now. This reduces the problem to solving a system of linear algebraic equations. The finite element method consists of applying Galerkin's method using a subspace L with a basis of piecewise polynomials.

The first step in deriving the system of finite element equations for the BVP (2.24) is to write it in its weak form. Consider the general BVP

$$\mathcal{L}\{u\} = f \ \text{ in } \ \Omega_g, \tag{2.25a}$$

$$u = 0 \ \text{ on } \ \partial\Omega_g, \tag{2.25b}$$

where $\mathcal{L}$ is an elliptic differential operator and Ω_g is a bounded domain in $\mathbb{R}^2$. Following Gockenbach (2006), next define

$$L^2(\Omega) = \left\{ v : \int_\Omega v v^* \,\mathrm{d}\Omega < \infty \right\}, \tag{2.26}$$

where v^* is the complex conjugate of v. This is the space of complex valued functions that are square integrable over Ω. The Sobolev space, H_0^1 can then be defined as

$$H_0^1 = \left\{ u \in L^2(\Omega) : \partial_x u, \partial_y u \in L^2(\Omega), u = 0 \ \text{ on } \ \partial\Omega \right\} \tag{2.27}$$

with the partial derivatives defined in the weak sense (see Gockenbach (2006, pg. 24) for a definition of the weak partial derivative). The weak form of the BVP (2.25) is

$$\text{find } u \in H_0^1(\Omega) : \int_\Omega \mathcal{L}(u) v \,\mathrm{d}\Omega = \int_\Omega f v \,\mathrm{d}\Omega \qquad \forall v \in H_0^1. \tag{2.28}$$

Since (2.28) must hold for *all* v in H_0^1, any u that solves it must also solve (2.25)—this is proven by Gockenbach (2006, pg. 20).

The weak form of the BVP (2.24) is to find $\mathbf{u} \in V$ such that

$$\int_\Omega [\nabla \cdot (\mathbf{K}_1 \nabla \mathbf{u}) + \nabla \cdot (\mathbf{K}_2 \nabla \mathbf{u}\, \mathbf{P}) - \mathbf{K}_3 \mathbf{u}] \cdot \mathbf{v} = \int_\Omega \mathbf{f} \cdot \mathbf{v} \qquad \forall \mathbf{v} \in V, \tag{2.29}$$

where $\mathrm{d}\Omega$ has been omitted from the integrals to save space and

$$V = \left\{ \mathbf{v} = (v_1, v_2) : v_1, v_2 \in H_0^1(\Omega) \right\}.$$

For use in the Galerkin method, the left hand side of (2.29) will be re-written as a symmetric bilinear form, $a(\mathbf{u}, \mathbf{v})$. Using the identity

$$(\nabla \cdot \sigma) \cdot \mathbf{v} = \nabla \cdot (\sigma^T \mathbf{v}) - \sigma \cdot \nabla \mathbf{v}, \tag{2.30}$$

which holds for any $\sigma \in \mathbb{C}^{2\times 2}$ and $\mathbf{v} \in \mathbb{C}^2$ (Gockenbach, 2006, pg. 18), the first term in (2.29) can be written

$$\int_\Omega [\nabla \cdot (\mathbf{K}_1 \nabla \mathbf{u})] \cdot \mathbf{v} = \int_\Omega \nabla \cdot [(\mathbf{K}_1 \nabla \mathbf{u})^T \mathbf{v}] - \int_\Omega (\mathbf{K}_1 \nabla \mathbf{u}) \cdot \nabla \mathbf{v}.$$

The first term on the right hand side of this equation is the integral of the divergence of a vector so by the divergence theorem

$$\int_\Omega \nabla \cdot [(\mathbf{K}_1 \nabla \mathbf{u})^T \mathbf{v}] = \int_{\partial\Omega} [(\mathbf{K}_1 \nabla \mathbf{u})^T \mathbf{v}] \cdot \hat{\mathbf{n}},$$

where $\hat{\mathbf{n}}$ is a unit vector normal to $\partial\Omega$. Since $\mathbf{u} = 0$ on $\partial\Omega$, this term vanishes, leaving

$$\int_\Omega [\nabla \cdot (\mathbf{K}_1 \nabla \mathbf{u})] \cdot \mathbf{v} = -\int_\Omega (\mathbf{K}_1 \nabla \mathbf{u}) \cdot \nabla \mathbf{v}.$$

Using the same reasoning on the second term, equation (2.29) can now be written in the form

$$a(\mathbf{u}, \mathbf{v}) = \ell(\mathbf{v}) \qquad \forall \mathbf{v} \in V, \tag{2.31}$$

with

$$a(\mathbf{u}, \mathbf{v}) = -\int_\Omega (\mathbf{K}_1 \nabla \mathbf{u}) \cdot \nabla \mathbf{v} + (\mathbf{K}_2 \nabla \mathbf{u}\, \mathbf{P}) \cdot \nabla \mathbf{v} + (\mathbf{K}_3 \mathbf{u}) \cdot \mathbf{v}\, \mathrm{d}\Omega \tag{2.32}$$

and

$$\ell(\mathbf{v}) = \int_\Omega \mathbf{f} \cdot \mathbf{v}\, \mathrm{d}\Omega. \tag{2.33}$$

Existence and uniqueness of the solution to (2.31) will not be shown here. Its derivation from the strong BVP, (2.24), was not shown in any of the publications relating to ArjunAir or in its documentation. The developers simply stated that they used a Galerkin finite element method with isoparametric quadrilateral elements, gave the form of the quadratic basis functions, and noted that the local element stiffness matrices are complex symmetric. Thus, it is not known how the original developers derived the finite element system of linear equations solved in ArjunAir. The derivation in this section leads to the same system of algebraic equations solved by the program. This was determined by inspection of the source code.

It is also not known whether or not the ArjunAir developers rigorously determined the existence and uniqueness of exact solutions to (2.31), or if they attempted to derive a bound on the error of the finite element solutions. They tested the accuracy of ArjunAir modelling results by comparing them with those from 1D and 3D airborne EM modelling software (Wilson et al., 2006). Their claims of accuracy were lightly tested for this study by comparing ArjunAir modelling results to those of a 1D modelling package for homogeneous halfspaces of varying resistivity. Further tests comparing ArjunAir with 3D modelling of long strike-length targets was carried out by Miensopust et al. (2013). Miensopust (personal communication, 2013) also tested ArjunAir on several homogeneous halfspaces and compared the results to a 1D code, getting results matching my tests.

As mentioned above, Key and Ovall (2011) developed a 2.5D marine CSEM modelling program that solves (2.24) using a finite element method. They used a different source but the left hand side of the PDE is the same. They were able to prove existence and uniqueness of a solution to the weak form of the BVP. Their proof relied on setting relative dielectric permittivity to 1 (i.e. they used the quasi-static approximation) so it is not technically valid

for ArjunAir. However, relative permittivity will be set very close to, if not exactly equal to 1 for most ArjunAir applications. Therefore, Key and Ovall's results provide confidence that the ArjunAir finite-element scheme is sound in principle.

For a rigorous discussion of existence/uniqueness issues and error estimation in finite element solutions to Maxwell's equations, see the book by Monk (2003, *esp*. chap. 2). It is mainly concerned with high-frequency EM fields, which behave somewhat distinctly from geophysical EM fields but many of the results apply generally to Maxwell's equations, geophysical fields included.

Now, the finite-dimensional Galerkin approximate solution of the weak BVP (2.31) will be derived. Consider a finite dimensional subspace $V_h \subset V$. Using the terminology of Brenner and Scott (2008), the Galerkin approximation corresponding to (2.31) is to find $\mathbf{u}_h \in V_h$ such that

$$a(\mathbf{u}_h, \mathbf{v}) = \ell(\mathbf{v}) \qquad \forall \mathbf{v} \in V_h. \tag{2.34}$$

Since V_h is finite-dimensional, it has a basis $\{\boldsymbol{\beta}_i\}$, which allows $\mathbf{u}_h$ to be written as a linear combination of the basis functions,

$$\mathbf{u}_h = \sum_i u_i \boldsymbol{\beta}_i, \tag{2.35}$$

$u_i \in \mathbb{C}$. Substituting this expression for $\mathbf{u}_h$ into (2.34) and appealing to the linearity of a in its first argument gives

$$\sum_i u_i \, a(\boldsymbol{\beta}_i, \mathbf{v}) = \ell(\mathbf{v}) \qquad \forall \mathbf{v} \in V_h. \tag{2.36}$$

Choosing $\mathbf{v} = \boldsymbol{\beta}_j$ the following holds:

$$\sum_i u_i \, a(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j) = \ell(\boldsymbol{\beta}_j). \tag{2.37}$$

The coefficients $\{u_i\}$ can now be found by solving a square system of linear equations $KU = F$ where

$$K_{ij} = a(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j), \qquad U_i = u_i, \qquad F_i = \ell(\boldsymbol{\beta}_j). \tag{2.38}$$

Since a is a symmetric bilinear form, the matrix K is complex symmetric. Its numerical properties will be discussed further in Chapter 3.

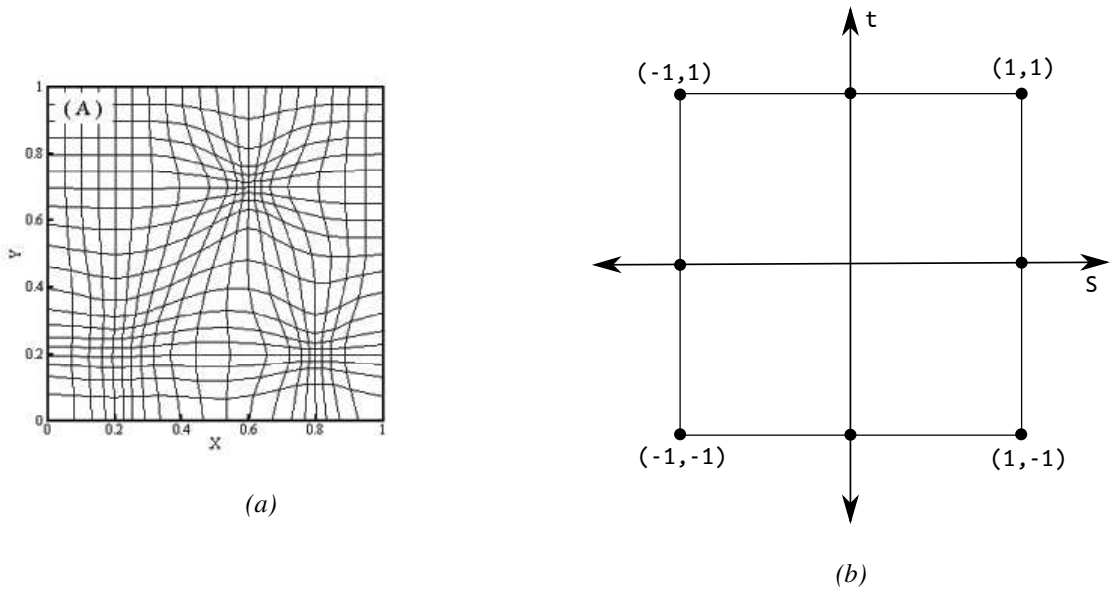
2.1.4.2 The finite element method in ArjunAir

A Galerkin finite element method consists—to paraphrase Gockenbach (2006)—of using Galerkin's method with V_h being a space of piecewise polynomials. Before discussing the details of the finite element method implemented in ArjunAir, recall that it is concerned with a vector BVP. The finite element solution, $\mathbf{u}_h$, is a vector quantity, $\mathbf{u}_h = (\tilde{H}_y^{sh}, \tilde{E}_y^{sh})$. Let $P_h^0(\Omega) \subset H_0^1(\Omega)$ be the set of piecewise polynomials of degree h that are continuous and weakly differentiable on Ω and zero on $\partial\Omega$. Let the set of scalars $\{\psi_i\}$ be a basis for $P_h^0(\Omega)$. V_h may now be defined more specifically as

$$V_h = \left\{ \mathbf{v}_h = (v_{1h}, v_{2h}) : v_{1h}, v_{2h} \in P_h^0(\Omega) \right\}. \tag{2.39}$$

In ArjunAir, Ω is divided into a set of isoparametric quadrilateral regions, or elements. These elements have four sides and four vertices each, but their sides may be curved. Additionally, the mesh must be conforming, meaning that the elements must be non-overlapping and that no vertex may lie on the edge of an adjacent element at a position that is not a vertex of the adjacent element. An example of a mesh of isoparametric quadrilaterals on a rectangular domain is shown in Figure 2.1a. Inside each element, each component of $\mathbf{u}_h$ is represented by a quadratic polynomial of the form

$$a_0 + a_1 x + a_2 x^2 + a_3 z + a_4 z^2 + a_5 xz + a_6 xz^2 + a_7 x^2 z. \tag{2.40}$$



(a) *(b)*

Figure 2.1: a) Example of meshing a rectangular domain with isoparametric quadrilaterals (Bono and Awruch, 2008). b) Eight node isoparametric quadrilateral reference element.

This corresponds to taking $h = 2$ in equation (2.39). To insure each component of the finite element solution, $\mathbf{u}_h$, is continuous over all of Ω, the polynomials on adjacent elements must agree on the boundary joining the elements. Thus the ArjunAir finite element method approximates the solution of the weak BVP (2.31) by a vector function, $\mathbf{u}_h$, with each component in the space P_2^0—i.e. a function that is continuous and weakly differentiable on Ω and representable by a function of the form (2.40) within each element.

To find the finite element solution using Galerkin's method, a basis, $\{\psi_i\}$, for P_2^0 must be constructed. The two components of $\mathbf{u}_h$ will be represented by linear combinations of scalar basis functions,

$$\tilde{H}_y^{sh} = \sum_{i=1}^{n_f} H_i \psi_i, \qquad \tilde{E}_y^{sh} = \sum_{i=1}^{n_f} E_i \psi_i, \tag{2.41}$$

where n_f is the dimensionality of the basis set, equal to the total number of interior nodes in the finite element mesh. The full solution $\mathbf{u}_h \in V_h$ may then be formed by a linear

combination of vector basis functions

$$\mathbf{u}_h = \sum_{i=1}^{2n_f} u_i \boldsymbol{\beta}_i, \tag{2.42}$$

where

$$\{\boldsymbol{\beta}_i : \boldsymbol{\beta}_i = (\psi_i, 0) \text{ for } 1 \leq i \leq n_f, \boldsymbol{\beta}_i = (0, \psi_i) \text{ for } n_f + 1 \leq i \leq 2n_f\}. \tag{2.43}$$

ArjunAir uses a nodal basis for $\{\psi_i\}$. A node is placed at each vertex in the mesh and at the midpoint of each element edge, as in Figure (2.1b). The homogeneous Dirichlet boundary conditions mean that $\tilde{H}_y^{sh}$ and $\tilde{E}_y^{sh}$ are zero on all nodes on $\partial\Omega$. When homogeneous Dirichlet boundary conditions are used, it can be shown (Gockenbach, 2006) that a function in P_2^0 can be fully determined by its value at the interior nodes of the finite element mesh. This suggests using a basis with the following property:

$$\psi_i = \begin{cases} 1 & : \text{ at node } i \\ 0 & : \text{ at all other nodes.} \end{cases} \tag{2.44}$$

The basis functions must also have the property that $\psi_i \neq 0$ only in elements that include node i.

At this point, recall that Galerkin's method leads to the linear system of equations $KU = F$, with $K_{ij} = a(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j)$, and $F = \ell(\boldsymbol{\beta}_j)$. a and ℓ are integrals over Ω. They can be written as the sum of integrals over each element that makes up Ω:

$$\begin{aligned} a(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j) &= -\sum_{e_i=1}^{n_e} \int_{\Omega_{e_i}} (\mathbf{K}_1 \nabla \boldsymbol{\beta}_i) \cdot \nabla \boldsymbol{\beta}_j + (\mathbf{K}_2 \nabla \boldsymbol{\beta}_i \, \mathbf{P}) \cdot \nabla \boldsymbol{\beta}_j + (\mathbf{K}_3 \boldsymbol{\beta}_i) \cdot \boldsymbol{\beta}_j \, \mathrm{d}\Omega., \\ \ell(\boldsymbol{\beta}_j) &= \sum_{e_i=1}^{n_e} \int_{\Omega_{e_i}} \mathbf{F} \cdot \boldsymbol{\beta}_j \, \mathrm{d}\Omega. \end{aligned} \tag{2.45}$$

It is clearly seen that the first integral will be zero over all elements except those that include both node i and node j. Additionally, for nodes i and j that never occur in the same

element, $a(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j)$ will be zero over all of Ω. Thus, for a given $\boldsymbol{\beta}_i$, $a(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j)$ will only be non-zero for a small number of $\boldsymbol{\beta}_j$, leading to a sparse matrix, K.

In ArjunAir, as in most practical finite element codes, K and F are assembled element by element, as the sum of submatrices and subvectors K^{e_i} and F^{e_i}, associated with each element, e_i. The entries of the submatrix for element e_i are the integrals $a(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j)$ over e_i, for each pair of basis functions $\boldsymbol{\beta}_i$ and $\boldsymbol{\beta}_j$ that are non-zero in e_i. There are eight nodes per element and each node is associated with two basis functions, one of the form $(\psi_i, 0)$ and the other of the form $(0, \psi_i)$. Thus, each element matrix is 16×16.

The vector nature of the basis functions imparts further structure to the element matrices. They can be written as 2×2 block matrices

$$K^{e_i} = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}. \tag{2.46}$$

Entries of K_{11} have the form $a^{e_i}[(\psi_i, 0), (\psi_j, 0)]$. The entries of K_{12}, K_{21}, and K_{22} have the respective forms $a^{e_i}[(\psi_i, 0), (0, \psi_j)]$, $a^{e_i}[(0, \psi_i), (\psi_j, 0)]$, and $a^{e_i}[(0, \psi_i), (0, \psi_j)]$. K_{11} and K_{22} are symmetric, while $K_{12} = K_{21}^T$. General forms of the entries of each submatrix may be found by substituting the corresponding forms of $\boldsymbol{\beta}_i$ and $\boldsymbol{\beta}_j$ into equation (2.45). For example, entries of K_{11} will have the form

$$a^{e_i}(\boldsymbol{\beta}_i, \boldsymbol{\beta}_j) = a^{e_i}[(\psi_i, 0), (\psi_j, 0)] = \int_{\Omega_{e_i}} \frac{1}{k_e^2} \nabla\psi_i \cdot \nabla\psi_j + \psi_i\psi_j \, \mathrm{d}\Omega. \tag{2.47}$$

The core task of computing the entries of the global stiffness matrix K is to compute integrals of the form (2.47) and the similar integrals for the entries of K_{12} and K_{22}.

The missing ingredients required to compute these integrals are expressions for the ψ_i inside an arbitrary element. On a rectangular domain divided into rectangular elements with sides aligned with those of the domain, bi-quadratic basis functions of the same form

as the element polynomials (equation 2.40) may be chosen. Unfortunately, for elements with edges not aligned with the domain axes, or edges that may be curved, a linear combination of bi-quadratic functions will not necessarily determine a piecewise continuous function on Ω. However, the basis functions within a given element may be taken to be images of a bi-quadratic function on a rectangular reference element under a quadratic mapping (Gockenbach, 2006). The reference element is shown in Figure 2.1b. The mapping of a point (s, t) in the reference element to a point (x, z) in the true domain, is given by

$$\begin{aligned} x &= a_0 + a_1 s + a_2 s^2 + a_3 t + a_4 t^2 + a_5 st + a_6 st^2 + a_7 s^2 t \\ z &= b_0 + b_1 s + b_2 s^2 + b_3 t + b_4 t^2 + b_5 st + b_6 st^2 + b_7 s^2 t. \end{aligned} \tag{2.48}$$

This can be written as a vector function,

$$\mathbf{w} = (x, z) = \mathbf{g}(s, t). \tag{2.49}$$

The bi-quadratic basis functions associated with each node of the reference element may be constructed by inspection using the Kronecker delta requirement of the basis functions. The reference element has vertices (-1,1), (0,1), (1,1), (1,0), (1,-1), (0,-1), (-1,-1), and (-1,0) in the reference s-t coordinate system. The basis functions corresponding to the reference

nodes, ordered as in the previous sentence, are

$$\begin{aligned}
\zeta_1 &= -\frac{1}{4}(1-s)(1+t)(1+s-t) \\
\zeta_2 &= \frac{1}{2}(1+s)(1-t^2) \\
\zeta_3 &= -\frac{1}{4}(1+s)(1+t)(1-s-t) \\
\zeta_4 &= \frac{1}{2}(1+s)(1-t^2) \\
\zeta_5 &= -\frac{1}{4}(1+s)(1-t)(1-s+t) \\
\zeta_6 &= \frac{1}{2}(1-s^2)(1-t) \\
\zeta_7 &= -\frac{1}{4}(1-s)(1-t)(1+s+t) \\
\zeta_8 &= \frac{1}{2}(1-s)(1-t^2).
\end{aligned} \tag{2.50}$$

Rather than transforming these reference basis functions to each real element in order to compute the integrals (2.47), the integrals may be performed over the reference element by making an appropriate change of variables. According to the rules of multivariable calculus, as cited in Gockenbach (2006), the formula for change of variables in a multiple integral is

$$\int_{\Omega_{e_i}} f(x,z)\,\mathrm{d}x\mathrm{d}z = \int_{e_r} f(g(s,t))|\mathrm{det}[\mathbf{J}(\mathbf{g})]|\,\mathrm{d}s\mathrm{d}t, \tag{2.51}$$

where e_r denotes the reference element and $\mathbf{J}(g)$ is the Jacobian of the transformation from the reference coordinates to the real coordinates:

$$\mathbf{J}(\mathbf{g}) = \begin{pmatrix} \partial_s x(s,t) & \partial_t x(s,t) \\ \partial_s z(s,t) & \partial_t z(s,t) \end{pmatrix}. \tag{2.52}$$

Using the change of variable rule, equation (2.47) is transformed to

$$a^{e_i}[(\psi_i,0),(\psi_j,0)] = \int_{e_r} \left[\frac{1}{k_e^2}(\mathbf{J}^{-T}\nabla\zeta_i)\cdot(\mathbf{J}^{-T}\nabla\zeta_j) + \zeta_i\zeta_j\right] |\det(\mathbf{J})|\,\mathrm{d}s\mathrm{d}t, \qquad (2.53)$$

where $\mathbf{J}^{-T}$ is the transpose of $\mathbf{J}^{-1}$. k_e, which includes the electrical conductivity, is assumed to be constant in each element. The integral formulas for the other entries of the element matrices and the entries of the element load vectors, F_{e_i}, may be transformed by the same rule. The integrands will not, in general, be polynomials and cannot normally be integrated analytically. In ArjunAir, the integrals are computed approximately by a 9-point Gaussian quadrature rule. Once all these integrals have been computed and the global system of equations $KU = F$ has been assembled, the system may be solved using the techniques of sparse numerical linear algebra. The methods used to solve $KU = F$ in ArjunAir will be discussed in the next chapter. Computing the integrals (2.53) is very efficient and takes less than 1% of the total forward solution time.

2.1.5 Computing the primary field

Recall that the inhomogeneous, or source, term of the ArjunAir wavenumber domain BVP is a function of the primary electric field. The primary field is the electric field due to a harmonic magnetic dipole transmitter in free-space. It is oriented in the x-z plane at an angle θ from the z-axis. In the frequency-domain, the field is given by the simple expression

$$\mathbf{E}_p(x,y,z) = \frac{i\omega\mu}{4\pi r^3}\left(y\cos\theta\hat{\mathbf{x}} + (z\sin\theta - x\cos\theta)\hat{\mathbf{y}} + y\sin\theta\hat{\mathbf{z}}\right), \qquad (2.54)$$

where $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ are unit vectors, x, y, and z are the distances from the dipole in the three Cartesian directions, and $r = \sqrt{x^2+y^2+z^2}$. ArjunAir needs to compute the value of the Fourier transform with respect to y of all three components of $\mathbf{E}_p$ at each subsurface node

in the finite element mesh. The values at nodes in the air are not required since every time a component of $\mathbf{E_p}$ appears in the wavenumber domain BVP, it is multiplied by the model's total conductivity. For the purposes of the primary field calculation the total conductivity in the air is taken to be zero, making the source term in the secondary field BVP zero in the air. The wavenumber domain representation of $\mathbf{E}_p$ must still be computed at all subsurface mesh nodes. Unfortunately, a closed form expression for $\mathbf{E}_p$ in the y-wavenumber domain does not exist, so it must be computed by numerical Fourier transformation.

ArjunAir uses the digital filtering technique to compute the transforms. Digital filtering converts a Fourier or Hankel transform of a function f to a weighted sum of the values of f at a discrete set of sample points. The fast Fourier transform (FFT) is the most common technique for computing numerical approximate FTs. It also converts an FT to a weighted sum, however, it requires equally spaced sample points. The $\frac{y}{r^3}$ and $\frac{1}{r^3}$ y-dependence of the components of $\mathbf{E}_p$ make the FFT technique inefficient for the 2.5D geophysical EM problem. Sample spacing must be very tight to capture the rapid decay of the field at small y, leading to a very large number of samples being required to capture the behaviour at large y. Therefore, a technique that allows for logarithmic sample spacing is preferred.

Digital filtering meets the requirement of logarithmic sample spacing. It has been applied more often in EM geophysics to the computation of Hankel transforms, which are required in 1D geophysical EM modelling (e.g. Christensen, 1990; Farquharson and Oldenburg, 2000), but is equally applicable to the computation of Fourier transforms (e.g. Johansen and Sørensen, 1979; Anderson, 1983). The ArjunAir developers wrote their own digital filtering routines, with the help of routines developed by Christensen (1990) and based on his work. This section of the thesis will give some brief theoretical background information on the digital filtering technique, while technical implementation details and

modifications to the original routines performed by me will be described in the next chapter.

The Fourier transform can be written as the sum of sine and cosine transforms

$$\mathfrak{F}_y\left\{f(y)\right\} = \int_{-\infty}^{\infty} f(y)e^{-ik_y y}\,\mathrm{d}y = \int_{-\infty}^{\infty} f(y)\cos(k_y y)\,\mathrm{d}y - i\int_{-\infty}^{\infty} f(y)\sin(k_y y)\,\mathrm{d}y. \tag{2.55}$$

The x and z components of $\mathbf{E}_p$ are anti-symmetric with respect to y and the y component is symmetric. Therefore, the cosine term of the FT is zero for the x and z components and the sine term is zero for the y component. Letting, $\rho^2 = x^2 + z^2$, this gives

$$\begin{aligned}\mathfrak{F}\left\{E_{px}\right\} = \tilde{E}_{px} &= \frac{\omega\mu\cos\theta}{2\pi}\int_0^{\infty}\frac{y}{\left(\rho^2+y^2\right)^{3/2}}\sin(k_y y)\,\mathrm{d}y,\\ \mathfrak{F}\left\{E_{py}\right\} = \tilde{E}_{py} &= (z\sin\theta - x\cos\theta)\frac{i\omega\mu}{2\pi}\int_0^{\infty}\frac{\cos(k_y y)}{\left(\rho^2+y^2\right)^{3/2}}\,\mathrm{d}y,\end{aligned} \tag{2.56}$$

and

$$\mathfrak{F}\left\{E_{pz}\right\} = \tilde{E}_{pz} = \frac{\omega\mu\sin\theta}{2\pi}\int_0^{\infty}\frac{y}{\left(\rho^2+y^2\right)^{3/2}}\sin(k_y y)\,\mathrm{d}y.$$

Digital filtering is used to compute the two integrals

$$\int_0^{\infty}\frac{y}{\left(\rho^2+y^2\right)^{3/2}}\sin(k_y y)\,\mathrm{d}y \quad \text{and} \quad \int_0^{\infty}\frac{\cos(k_y y)}{\left(\rho^2+y^2\right)^{3/2}}\,\mathrm{d}y \tag{2.57}$$

that occur in $\tilde{E}_{px}$, $\tilde{E}_{py}$, and $\tilde{E}_{pz}$.

The routines used for digital filtering in ArjunAir were developed for the computation of Hankel transforms. A Hankel transform is an integral transformation whose kernel J_ν is a Bessel function of the first kind with order $\nu > -1$. The Hankel transformation of a function, $f(y)$ is

$$g(k_y) = \int_0^{\infty} f(y)\,y J_\nu(k_y y)\,\mathrm{d}y. \tag{2.58}$$

The cosine and sine transforms in equations (2.57) may be converted to Hankel transforms

by using the identities

$$\cos(k_y y) = \sqrt{\frac{\pi k_y y}{2}} J_{-1/2}, \qquad \sin(k_y y) = \sqrt{\frac{\pi k_y y}{2}} J_{1/2}.$$

Using those identities, equations (2.57) become the Hankel transforms

$$\sqrt{\frac{\pi k_y}{2}} \int_0^\infty \frac{\sqrt{y}}{(\rho^2 + y^2)^{3/2}}\, y J_{1/2}\, \mathrm{d}y, \qquad \sqrt{\frac{\pi k_y}{2}} \int_0^\infty \frac{y^{-1/2}}{(\rho^2 + y^2)^{3/2}}\, y J_{-1/2}\, \mathrm{d}y. \tag{2.59}$$

These integrals may now be evaluated approximately using the Hankel transform technique of Christensen (1990). His work is an improvement of the method developed by Johansen and Sørensen (1979). They were able to show that a Hankel integral of the form (2.58) may be approximated by a weighted sum

$$g(k_y) = \sum_{j=a}^{b} f(\rho, e^{y_j}) H_j, \tag{2.60}$$

where the function evaluation points y_j and the coefficients H_j are computed in advance and should be valid for any reasonably well behaved function $f(y)$ that decays asymptotically toward zero quickly enough as $y \to \infty$. The evaluation points y_j must be equally spaced but the fact that $f(e^{y_j})$ rather than $f(y_j)$ is evaluated gives the desired logarithmic sample spacing. ArjunAir uses filter coefficients computed using the software of Christensen (1990). The coefficients and evaluation points are hard coded into the software.

The digital filtering technique computes the integrals (2.59) for a fixed value of ρ. Therefore, in order to compute $\tilde{\mathbf{E}}_p$ as accurately as possible at each subsurface node in the mesh, the transform integrals must be computed separately at each node and for each transmitter position. This was done in the original ArjunAir software. However, since the integrals depend only on the distance from the transmitter, ρ, and not on x and z separately, the task of computing $\tilde{\mathbf{E}}_p$ may be simplified considerably by computing the integrals at a range of values of ρ, computing ρ at each mesh node, then interpolating over ρ to find the

approximate values of the integrals at the nodes. Such an interpolation scheme was implemented as part of the work for this thesis. Implementation details will be described in the next chapter.

2.2 Inversion

2.2.1 Overview

The goal of ArjunAir is to compute an estimated earth model that matches a set of observed data. The observations are values of the secondary magnetic field along a straight flight line above the earth. The observations may be taken at different heights above the surface of the earth but their lateral positions must lie along a straight line in the x-y plane. In practice, flight lines will not be perfectly straight. To account for this, ArjunAir requires the user to input a starting point and direction for each flight line and all observations are assumed to fall on that line. An error message is generated and the program stops if the true observation points are too far from the idealized flight line. At each observation location, the secondary magnetic field may be measured in steady state as a function of frequency (for frequency-domain EM systems) or as a time decay after a pulse of source current through the transmitter for time-domain systems. From the point of view of the inverse problem, the fields at each frequency or time at a given location are independent observations.

An earth model is defined here as the electrical conductivity as a function of position on a 2D section of the earth below the flight line. Conductivity is assumed to be constant in each element of a user defined region of the 2D finite element mesh. The goal of the

computational inverse problem in ArjunAir is to recover the conductivity of each element. Recall that ArjunAir uses a multi-parameter frequency dependent complex conductivity for forward modelling. Only one component of the complex conductivity, the DC conductivity, may be inverted for. The other components may be set arbitrarily by the user but will remain fixed during inversion. DC conductivity may vary over several orders of magnitude. It is strictly positive and typically less than 1. Because of the positivity and large variation, ArjunAir's inversion algorithm uses the natural logarithms of the inverses of the cell conductivities, rather than the actual conductivities, as model parameters.

The inverse problem is posed as a non-linear least squares optimization problem,

$$\underset{\mathbf{m}}{\text{minimize}} \quad \Phi(\mathbf{d}, \mathbf{m}) = \|\mathbf{d} - \mathbf{f}(\mathbf{m})\|_W^2. \tag{2.61}$$

Here $\mathbf{d}$ is the vector of observed data, $\mathbf{m}$ is the vector of model parameters. $\mathbf{d}$ has dimension n_d and $\mathbf{m}$ has dimension n_m. ArjunAir is capable of handling both overdetermined ($n_d > n_m$) and underdetermined ($n_d < n_m$) problems. Only underdetermined problems are discussed here. $\mathbf{f}$ is the forward modelling operator. $\mathbf{f}(\mathbf{m})$ represents the predicted secondary fields computed by forward modelling for an earth with conductivity structure corresponding to $\mathbf{m}$. The W norm is a close cousin of the standard L_2 norm. For an arbitrary vector $\mathbf{a} \in \mathbb{R}^{n_d}$,

$$\|\mathbf{a}\|_W^2 = \mathbf{a}^T W \mathbf{a}, \tag{2.62}$$

for some real, symmetric positive-definite matrix W. In ArjunAir W will be a diagonal data-weighting matrix, with entries

$$W_{ii} = \frac{1}{\frac{n_d^2}{2}(d_i^2 + f_i^2)}, \tag{2.63}$$

where d_i and f_i are components of the observed and predicted data vectors. Stated in

words, the inverse problem is to find the model that minimizes the discrepancy between the observed data and the predicted secondary fields calculated by forward modelling.

2.2.2 The damped eigenparameter algorithm as implemented in ArjunAir

The forward modelling operator is non-linear in the model parameters so the inverse problem may not be solved directly by linear least-squares methods. The algorithm employed by ArjunAir to solve the non-linear minimization problem (2.61) is iterative and is based on the damped eigenparameter method of Jupp and Vozoff (1975). An overview of the method will now be given, following Jupp and Vozoff's original paper. To start, the forward modelling operator is linearized about a user supplied initial model $\mathbf{m}_0$. A multi-dimensional Taylor expansion of $\mathbf{f}$ about $\mathbf{m}_0$ gives

$$\mathbf{f}(\mathbf{m}_0 + \boldsymbol{\delta}_m) = \mathbf{f}(\mathbf{m}_0) + \mathbf{J}\boldsymbol{\delta}_m + \mathbf{R}, \tag{2.64}$$

where $\boldsymbol{\delta}_m$ is a small model perturbation and $\mathbf{J}$ is the Jacobian matrix of $\mathbf{f}$ at $\mathbf{m}_0$. The Jacobian is normally defined to be the $n_d \times n_m$ matrix with entries

$$J_{ij} = \frac{\partial f_i(\mathbf{m})}{\partial m_j} \tag{2.65}$$

where $f_i(\mathbf{m})$ is the predicted value of the i^{th} datum and m_j is the j^{th} model parameter. The data and model parameters may vary over widely different scales. To account for this ArjunAir computes a scaled version of the Jacobian, which measures the size of changes in the EM response due to fractional changes in model parameters (Wilson et al., 2006):

$$J_{ij} = \frac{m_j}{f_i(\mathbf{m})} \frac{\partial f_i(\mathbf{m})}{\partial m_j}. \tag{2.66}$$

Using this definition of the Jacobian will not otherwise affect the analysis presented here.

$\mathbf{R}$ in equation (2.64) is a remainder term, whose magnitude is assumed to be $\mathrm{O}\big(\|\boldsymbol{\delta}_m\|^2\big)$. For small model perturbations, $\boldsymbol{\delta}_m$, $\mathbf{R}$ may be ignored, giving

$$\mathbf{f}(\mathbf{m}_0 + \boldsymbol{\delta}_m) \approx \mathbf{f}(\mathbf{m}_0) + \mathbf{J}\boldsymbol{\delta}_m. \tag{2.67}$$

Replacing $\mathbf{f}$ by its linearization in the non-linear problem (2.61) gives the linear least squares problem

$$\underset{\boldsymbol{\delta}_m}{\text{minimize}} \quad \|\mathbf{d} - \mathbf{f}(\mathbf{m}_0) - \mathbf{J}\boldsymbol{\delta}_m\|_W^2 = \|\boldsymbol{\varepsilon} - \mathbf{J}\boldsymbol{\delta}_m\|_W^2, \tag{2.68}$$

where $\boldsymbol{\varepsilon} = \mathbf{d} - \mathbf{f}(\mathbf{m}_0)$. This may be converted to a least-squares problem in the L_2 norm by a change of variables in $\boldsymbol{\varepsilon}$ and $\mathbf{J}$ (Jackson, 1972). Let $\mathbf{W} = \mathbf{Z}^T\mathbf{Z}$, where $\mathbf{W}$ is the same as in equation (2.62). Then write

$$\mathbf{ZJ} = \mathbf{J}', \quad \text{and } \mathbf{Z}\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}'. \tag{2.69}$$

The minimization problem (2.68) can then be written

$$\underset{\boldsymbol{\delta}_m}{\text{minimize}} \quad \|\boldsymbol{\varepsilon}' - \mathbf{J}'\boldsymbol{\delta}_m\|_2^2. \tag{2.70}$$

The value of $\boldsymbol{\delta}_m$ is unaffected by the change of variables. The Jacobian and residual error are scaled in ArjunAir and computations proceed as if solving the L_2 norm minimization problem. Primes and specific norm designations are suppressed for the remainder of this chapter, on the understanding that doing so causes no loss of generality (Jackson, 1972; Jupp and Vozoff, 1975).

The full non-linear problem (2.61) is solved by an iterative process. At each iteration, let the current model be $\mathbf{m}_i$, the linear problem (2.68) is solved at $\mathbf{m}_i$ to find a model perturbation $\boldsymbol{\delta}_m$. The model is then updated by the rule $\mathbf{m}_{i+1} = \mathbf{m}_i + \boldsymbol{\delta}_m$. The problem is then

linearized about $\mathbf{m}_{i+1}$ and the process is repeated until the discrepancy between observed and predicted data is sufficiently small, a maximum number of iterations is reached, or until no further improvement may be made due to the algorithm reaching a local minimum in the model-space. The inversion procedure is laid out in Algorithm (2.2).

Algorithm 2.2 ArjunAir inversion algorithm

Specify an initial model $\mathbf{m}_0$
Solve forward problem to compute ε
if $\|\varepsilon\| \leq$ (a user specified tolerance) **then**
    End
end if
while $\|\varepsilon\| >$ (a user specified tolerance) **do**
    Linearize $\mathbf{f}$ about $\mathbf{m}_0$
    Solve (2.68) with model $\mathbf{m}_0$, to find model update $\boldsymbol{\delta}_m$
    $\mathbf{m}_0 = \mathbf{m}_0 + \boldsymbol{\delta}_m$
    Solve forward problem with new $\mathbf{m}_0$ to compute ε
end while

One can see from the algorithm that there are three main computational tasks involved in solving the ArjunAir non-linear inverse problem. First, one must be able to solve the forward problem in order to compute the data misfit, ε. Secondly, one must be able to compute the Jacobian of the forward modelling operator about an arbitrary model, $\mathbf{m}$. Finally, one must be able to solve the linear least squares problem (2.68) in order to compute the model update $\boldsymbol{\delta}_m$.

The algorithm used to solve the forward problem was described in the first section of this chapter. The Jacobian was calculated using the adjoint-operator method, as described in McGillivray et al. (1994). The algorithm was not altered for this study and will not be described here except to say that it requires the computation of adjoint electric fields. The

adjoint electric field for a given transmitter-receiver pair is the field due to a hypothetical transmitter at the receiver location. This means, that during inversion, the forward problem must be solved for the adjoint transmitters, in addition to the actual transmitters.

The final main inversion task, the solution of the linear least-squares problem (2.70) for the model perturbation $\boldsymbol{\delta}_m$, will now be described. Solving the linear problem presents two main challenges. First, there are normally more unknown model parameters than there are data, making the problem underdetermined. Additionally, the data have a very weak but non-zero dependence on some parameters, which makes the linear problem ill-conditioned and can lead to instability in the non-linear minimization.

To deal with the underdeterminedness that comes from having more model parameters than data points, ArjunAir finds the solution to (2.70) with minimum L_2 norm. In principle, that solution may be found by the matrix vector multiplication

$$\boldsymbol{\delta}_m = \mathbf{J}^\dagger \boldsymbol{\varepsilon}, \tag{2.71}$$

where $\mathbf{J}^\dagger$ is the Moore-Penrose pseudoinverse of $\mathbf{J}$ (Jupp and Vozoff, 1975). The Moore-Penrose pseudoinverse of an arbitrary matrix, $\mathbf{A}$, is a matrix $\mathbf{A}^\dagger$ that meets the following four conditions

1. $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$
2. $\mathbf{A}^\dagger\mathbf{A}\mathbf{A}^\dagger = \mathbf{A}^\dagger$
3. $(\mathbf{A}^\dagger\mathbf{A})^T = \mathbf{A}^\dagger\mathbf{A}$
4. $(\mathbf{A}\mathbf{A}^\dagger)^T = \mathbf{A}\mathbf{A}^\dagger$.

If one can compute the singular value decomposition (SVD) of $\mathbf{A}$, $\mathbf{A}^\dagger$ may be trivially

computed. Returning to the inverse problem, recall that $\mathbf{J} \in \mathbb{R}^{n_d \times n_m}$ ($n_d < n_m$) and let

$$\mathbf{J} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{2.72}$$

be its singular value decomposition, where $\mathbf{U} \in \mathbb{R}^{n_d \times n_d}$ and $\mathbf{V} \in \mathbb{R}^{n_m \times n_m}$ are orthogonal matrices. $\mathbf{\Sigma}$ is a diagonal matrix in $\mathbb{R}^{n_d \times n_m}$, whose diagonal entries, σ_i, come in non-increasing order and are called the singular values of $\mathbf{J}$. $\mathbf{\Sigma}$ has a maximum rank of n_d but may have a lower rank. The number of non-zero singular values of a matrix is equal to its rank. For a matrix with rank p, the singular values obey

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p > \sigma_{p+1} = \sigma_{p+2} = \cdots = \sigma_{n_d} = 0. \tag{2.73}$$

Using the SVD, $\mathbf{J}^\dagger$ is given by

$$\mathbf{J}^\dagger = \mathbf{V}\mathbf{\Sigma}^\dagger\mathbf{U}^T, \tag{2.74}$$

where $\mathbf{\Sigma}^\dagger \in \mathbb{R}^{n_m \times n_d}$ is a diagonal matrix whose diagonal entries, s_i, are defined by the rule

$$s_i = \begin{cases} \frac{1}{\sigma_i} & : \sigma_i > 0 \\ 0 & : \sigma_i = 0. \end{cases} \tag{2.75}$$

One may verify by direct substitution that $\mathbf{J}^\dagger$ meets the criteria for being a Moore-Penrose pseudoinverse. ArjunAir calculates the SVD of $\mathbf{J}$ using a routine based on the Golub-Reinsch algorithm (Golub and Reinsch, 1970).

Unfortunately, computing $\boldsymbol{\delta}_m$ directly from equation (2.71) causes instability in the iterative non-linear minimization algorithm, of which computing $\boldsymbol{\delta}_m$ is a part. Roundoff error in very small but non-zero singular values will be greatly magnified when their reciprocals are computed in $\mathbf{\Sigma}^\dagger$. Even in exact arithmetic, small singular values pose a serious problem since observed data are never exact. To see why this is the case, consider the vector of what Jupp and Vozoff (1975) call the eigenparameters of the inverse problem,

$\boldsymbol{\delta}_p = \sigma_1 \mathbf{V}^T \boldsymbol{\delta}_m$. Also, let $k_i = \sigma_i/\sigma_1$ for all non-zero σ_i and zero otherwise. Jupp and Vozoff showed that if the data are perturbed by an amount $\Delta\mathbf{d}$, satisfying $\|\Delta\mathbf{d}\| < q$, then

$$|\delta_{pi}| \leq \frac{q}{k_i} \quad \text{for } i = 1, p \tag{2.76}$$

where p is the number of non-zero singular values. That expression shows that a small change in the observed data can lead to a large change in a model eigenparameter if its corresponding singular value is small, relative to σ_1. To show that such changes are manifested in the actual model parameters, note that since $\mathbf{V}$ is an orthogonal matrix, taking the norm of the definition of $\boldsymbol{\delta}_p$ gives

$$\|\boldsymbol{\delta}_p\| = \sigma_1 \|\boldsymbol{\delta}_m\|, \tag{2.77}$$

showing that small changes in the data can lead to large changes in the model when $\mathbf{J}$ has small enough non-zero singular values. Put another way, small singular values correspond to what Jupp and Vozoff call unimportant model parameters. The observed data are not very sensitive to changes in unimportant parameters but the small singular values they create may create large entries in the model update vector. This may lead to model update vectors that violate the linear approximation of the forward modelling operator about the current model and lead to wild and unstable changes in the model from iteration to iteration.

A stable inversion may be achieved by damping the effect of small singular values, thus avoiding wild changes in model parameters from iteration to iteration. The simplest method of damping is singular value truncation, in which singular values with magnitudes below some threshold, relative to σ_1, are set to zero. In the approach taken by ArjunAir, small singular values are damped. The model update is computed as

$$\boldsymbol{\delta}_m = \mathbf{V}\mathbf{T}\boldsymbol{\Sigma}^{\dagger}\mathbf{U}^T\boldsymbol{\varepsilon} \tag{2.78}$$

where $\mathbf{T}$ is the diagonal damping matrix with diagonal entries

$$t_i = \begin{cases} \frac{k_i^{2N}}{k_i^{2N} + (\frac{\nu}{\sigma_1})^{2N}} & : \sigma_i > 0 \\ 0 & : \sigma_i = 0. \end{cases} \tag{2.79}$$

for some natural number N. As before, $k_i = \sigma_i/\sigma_1$. ν is known as the damping parameter. In practice it is not set explicitly, only the relative threshold $\nu/\sigma_1 = \mu$ is adjusted. ArjunAir uses $N = 2$. μ is set heuristically. It is set to 0.1 at the start of the inversion. At any iteration, if the decrease in data misfit from the current model, $\mathbf{m}$, to the new model $\mathbf{m}+\boldsymbol{\delta}_m$ is large enough, then the model update is accepted and μ is divided by two. If the misfit decrease is inadequate, the model update is rejected, μ is multiplied by two and the update is re-calculated with the new damping parameter. If no adequate decrease in misfit can be achieved after a set number of iterations, the inversion is abandoned. The overall strategy here, as described by Jupp and Vozoff (1975) and the ArjunAir developers (Wilson et al., 2006) is to start the inversion with heavy damping, so that only the more important model parameters are updated—and in a smooth gradual fashion. Then, as the fit is improved, damping is reduced to allow a finer fit to the data and an approach to the true minimum of the non-linear problem (2.61).

2.2.3 The Levenberg-Marquardt algorithm

As alluded to in the introduction, a major drawback of the damped least squares inversion procedure described above is that the results are highly dependent on the choice of initial model. Another limitation is that $\mathbf{J}$ and its SVD are expensive to compute and store. Addressing the dependence on the initial model would require implementing a completely new inversion algorithm, such as a minimum structure method. Such a modification is beyond

the scope of this thesis. Additionally, it would require adjusting the user interface of ArjunAir, which would make it more difficult for existing users to use—especially those who access ArjunAir through a plugin in the commercial graphical EM processing and analysis software package Maxwell (EMIT, 2014).

However, it was possible to eliminate the need to compute the SVD while maintaining a roughly equivalent inversion algorithm and ArjunAir's current interface. ArjunAir's original inversion algorithm is a slightly modified version of the Levenberg Marquardt algorithm (Wilson et al., 2006). The modified algorithm produced for this thesis implemented the actual Levenberg-Marquardt algorithm and did it in such a way as to avoid computing the SVD of $\mathbf{J}$ and allow for the possibility of using a sparse $\mathbf{J}$. The Levenberg-Marquardt algorithm solves the non-linear least squares problem (2.61) iteratively by linearizing about an initial model and computing model updates by solving a damped linear least squares problem. The linear problem for the model update is

$$\underset{\boldsymbol{\delta}_\mathbf{m}}{\text{minimize}} \quad \|\boldsymbol{\varepsilon} - \mathbf{J}\boldsymbol{\delta}_m\|^2 + \nu^2\|\boldsymbol{\delta}_m\|^2, \tag{2.80}$$

where ν is the same as in equation (2.79). This is equivalent to computing the model update by equation (2.78) but taking $N = 1$ in the damping parameter definition, (2.79)—(rather than $N = 2$ as in the original algorithm).

The difference in eigenparameter damping between $N = 1$ and $N = 2$ is illustrated in Figure 2.2. Large singular values remain essentially undamped and very small singular values are almost entirely damped. The difference lies in the length of the transition from undamped to damped singular values. The damping threshold is higher in the original ArjunAir algorithm and the transition to essentially full damping is faster than in the new algorithm. In practice, both damping schemes produced very similar non-linear inversion

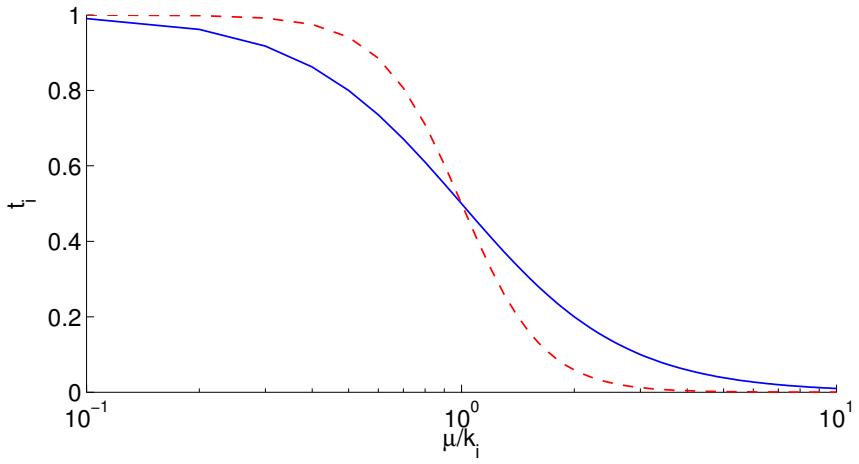


Figure 2.2: Damping factors plotted as functions of the ratio of the relative damping parameter μ to the relative singular value magnitude k_i. The curve for $N = 1$ (Levenberg-Marquardt damping) is shown as the solid blue line and $N = 2$ (Arjunair damping) as the dashed red line.

results, which will be discussed in Chapter 4.

The solution to the new linear least squares problem (2.80) is the $\boldsymbol{\delta}_m$ that satisfies the system of linear equations

$$(\mathbf{J}^T\mathbf{J} + \nu^2\mathbf{I})\boldsymbol{\delta}_m = \mathbf{J}^T\boldsymbol{\varepsilon}, \tag{2.81}$$

where $\mathbf{I}$ is the identity matrix. The system may now be solved by standard methods of numerical linear algebra. $\mathbf{J}$ is a dense matrix. Recall that its entries are $J_{ij} = \frac{m_j}{f_i(\mathbf{m})}\frac{\partial f_i(\mathbf{m})}{\partial m_j}$.

Whether $\mathbf{J}$ is dense or sparse, it may be solved by an iterative Krylov subspace method, eliminating the need to factor $(\mathbf{J}^T\mathbf{J} + \nu^2\mathbf{I})$. The updated inversion algorithm implemented in ArjunAir for this project finds the model update vector, $\boldsymbol{\delta}_m$, by solving (2.81) using the Krylov solver LSQR, due to Paige and Saunders (1982). LSQR is specifically designed to stably and efficiently solve systems of equations of the form (2.81). Multiple heuristic techniques for adjusting the damping parameter were tested. They will be described in Chapter 4.

Chapter 3

Computational Methods and Results I: Forward Modelling

This chapter will describe the numerical methods used and the results achieved in decreasing ArjunAir forward solve runtimes without sacrificing solution accuracy. The main approach taken was to replace the most time consuming computations with efficient parallel routines. A brief overview of parallel computer architectures and programming models will be given. A goal of this project was to develop a version of ArjunAir suitable for multi-core desktop computers and another version suitable for large scale computer clusters. The general approaches taken to develop both versions will be discussed.

The remainder of the chapter will describe the specific methods used to achieve parallelism and to increase the efficiency of each bottleneck computation in the forward solver. Computing multiple 2D wavenumber domain subproblems concurrently is the most obvious and coarsest grained source of parallelism in the forward solver. Parallelizing over 2D subproblems was trivial to implement and produced excellent parallel speedup. However,

for inversions on large domains with small cells, memory constraints limited the number of 2D solves that could be performed simultaneously. It was therefore important to look for inefficiencies and parallelizable computations within each subproblem. The two main bottlenecks were the solution of the wavenumber domain linear systems of finite-element equations and the computation of the wavenumber domain primary electric fields.

Three main approaches were taken to solving the finite-element equations. First, the existing solver, coded by the ArjunAir developers, was modified to run as a distributed memory parallel code. That solver was then replaced with the distributed memory parallel sparse direct solver MuMPS (Amestoy et al., 2001). Finally, the shared memory parallel solver Pardiso (Schenk and Gartner, 2004) was tested. Both MuMPS and Pardiso performed significantly better than the original ArjunAir solver and had slightly disappointing but acceptable parallel scaling. A simple modification of the primary field computations was able to yield dramatic speedups without parallelization. This came at the cost of a small loss in the accuracy of the computed primary fields but that did not seem to affect the accuracy of the final solutions.

3.1 Parallel architectures and programming paradigms

Flynn's taxonomy provides a useful method for classifying parallel computers (Pacheco, 2011). It is shown in tabular form in Table 3.1. It divides computers into four categories based on two criteria: whether a computer can execute multiple instructions simultaneously, and whether it can operate on multiple data simultaneously. Sequential, or serial computers fall into the single instruction, single data (SISD) category. They can only execute one instruction on one datum at a time. Graphical processing units and vector

	Single instruction	**Multiple instruction**
Single data	SISD	MISD
Multiple data	SIMD	MIMD

Table 3.1: Flynn's taxonomy.

processors fall into the single instruction, multiple data (SIMD) category. They can have many processing units processing different data simultaneously but all the units must execute identical instructions. Multiple instruction, single data computers have almost never been built in practice. Today, most general purpose parallel computers may be classified as multiple input, multiple data (MIMD) machines (Pacheco, 2011). The processing units of an MIMD machine may operate asynchronously on separate streams of instructions and data. Writing parallel programs for such machines is most often achieved using the single program, multiple data (SPMD) approach. In SPMD, a single program is written and conditional branching is be used to assign different instructions or data to different processing elements.

Another way to classify parallel computers is by memory architecture. The two main classes are shared and distributed memory. MIMD machines and SPMD programming approaches exist for both architectures. In a shared memory system, all the processors share a common global memory address space. Each processor may still have its own local cache memory. When a processor on a shared memory system writes a set of data to the global memory, those data may be accessed by all the other processors immediately. OpenMP (OpenMP Architecture Review Board, 2008) is the most common application programming interface (API) for shared memory computing. Generally, a shared memory program is launched as a single process. At any time that process may launch multiple

threads that run in parallel. Creating and destroying threads can be done much more quickly than creating and destroying processes. It is common for a shared memory program to start out in serial, spawn a set of threads for some parallel task and then destroy the threads and continue running sequentially, to perform tasks that cannot be done in parallel.

Most desktop computers produced in recent years have come equipped with multicore processors and high end desktop machines with 16 or more cores are commonly available (e.g. HP, 2014). Shared memory programming techniques can improve the performance of programs run on such machines. By contrast, computer clusters and most modern supercomputers use either what is known as distributed memory architecture, or some hybrid of distributed and shared memory. In a distributed memory system, each processor has its own memory address space and processors may only communicate with one another by passing messages to one another. On distributed memory systems using the SPMD paradigm, each processor core launches a separate instance of the same program. All these instances are separate processes with their own memory address space. They may only interact by passing messages to one another. Although each process launches the same program, through conditional branching they may operate on different data and/or execute different instructions. MPI (Gabriel et al., 2004), which stands for message passing interface, is the most common API for distributed memory programming. MPI defines a standard interface and functionality for a set of libraries that facilitate passing messages between the different processes of a distributed memory program.

The near ubiquity of multicore processors in recent years has led to a decrease in purely distributed memory systems and an increase in hybrid systems (Chow and Hysom, 2001). A common hybrid architecture is a cluster of shared memory computers. In hybrid programming, the large tasks of a program are often divided between the nodes of a cluster and

each node performs its task using shared memory parallelism. I was interested in exploring the shared memory, distributed memory, and hybrid parallel programming approaches in ArjunAir. I had access to a 504 core (42 nodes, 12 cores per node) Linux cluster at Memorial University, called Torngat, which provided a platform for testing the scaling of a distributed memory code. The ability to run ArjunAir over multiple nodes of the cluster also provided access to a large amount of memory, with each node having 24GB of RAM. However, many potential industrial users of a parallel version of ArjunAir would not necessarily have access to a cluster computing platform. Message passing programs may still be run on a shared memory multicore workstation but programs written specifically for shared memory will likely provide better performance on such a machine.

3.2 Approaches to developing a parallel ArjunAir forward solver

My supervisors and I were interested in developing a version of ArjunAir that would scale well enough to take advantage of the substantial computing resources of the Torngat cluster, motivating the development of a distributed memory version of ArjunAir. Potential industrial users would be more likely to want to run ArjunAir on high end shared memory workstations. That practical consideration, as well as the desire to compare the parallel scaling of the distributed and shared memory approaches, motivated the desire to write a shared memory code.

The cluster and workstation architectures allow for different approaches to developing a parallel forward solver. On a workstation, memory constraints limit the ability to

perform multiple wavenumber domain 2D BVP solves concurrently for large problems. Thus, for the workstation version of ArjunAir, the main focus was on achieving parallelism (and higher performance generally) within each 2D wavenumber domain boundary value problem (BVP) solve.

On a cluster, memory constraints are less severe, meaning that multiple wavenumber domain BVPs may be solved concurrently. In the simplest implementation, that would be the only parallelism, solving as many 2D problems concurrently as memory and CPU resources allow. In a more sophisticated implementation, a solver with nested levels of parallelism could be constructed. Nested parallelism makes sense on a hybrid distributed memory/shared memory cluster like Torngat. Recall that the cluster is composed of a series of nodes that communicate by distributed memory message passing. Each node is a shared memory multicore computer. In a nested forward solver, the 2D BVP solves are divided among a set of nodes on the cluster, with one MPI process per node. Within each node, the individual BVPs may be solved in parallel using shared memory techniques, implemented with OpenMP. From a software engineering perspective, this type of hybrid code has the advantage that the code for solving the individual 2D BVPs is the same in the workstation and cluster versions. The only modification for the distributed memory version is the code used to assign BVPs to nodes and collect the final results on the master MPI process.

Such nested parallelism can also be achieved purely using MPI distributed memory programming. In such a scheme, one top-level MPI process would be assigned to each cluster node. Each top level process would in turn spawn a set of second level processes on all the cores of its node. Insuring that each process is mapped to the correct physical core would be a non-trivial difficulty in implementing a two-level MPI forward solver.

Developing a two-level MPI code was not attempted. Pure MPI and pure OpenMP

versions of ArjunAir were completed and an MPI/OpenMP hybrid code was partially developed. The hybrid code produces correct results but no shared memory parallel speedup on top of the speedup due to solving multiple BVPs concurrently. That unacceptable performance was likely caused by a cluster configuration issue that, unfortunately, could not be resolved in time for meaningful hybrid results to be included in this thesis.

Initial testing of ArjunAir was performed on two consumer grade workstations with 4 and 8 processor cores, respectively. However, all performance results quoted in the remainder of this thesis are for runs on the Torngat cluster. The purely distributed memory code was tested for correctness using the Intel® ifort compiler and GNU's gfortran compiler. The shared memory version of the code relied on the Intel® Math Kernel Library (MKL) (Intel, 2014). It is possible to link MKL to a program compiled with gfortran but this was not attempted. The shared memory version of ArjunAir was only tested with the ifort compiler.

3.3 Solving the finite element equations

3.3.1 Sparse-direct methods for $\mathbf{KU} = \mathbf{F}$

For each frequency the 2D wavenumber domain boundary value problem (2.24) is solved at a set of 21 logarithmically spaced wavenumbers ranging from 1×10^{-5} to 0.1. The wavenumbers were chosen empirically by the ArjunAir developers. They are hardwired into the code and cannot be modified by the user.

As described in Chapter 2, the finite-element method converts the boundary value problem (2.24) for a given transmitter location into a system of linear algebraic equations

$\mathbf{KU} = \mathbf{F}$, where the vector $\mathbf{F}$ depends on the location and character of the transmitter. In a typical ArjunAir inversion, data from several hundred transmitter locations may be included. The coefficient matrix $\mathbf{K}$, which is large and sparse, does not depend on the source location. This creates a situation where the same coefficient matrix must be solved against a great many right hand sides. That consideration led the developers of ArjunAir to use a direct method to factor $\mathbf{K}$ rather than solving $\mathbf{KU} = \mathbf{F}$ by an iterative Krylov subspace method.

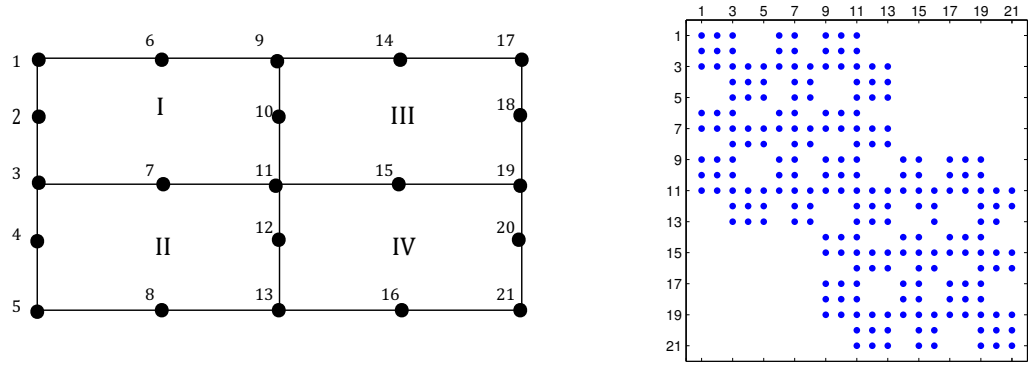
For most sparse matrices encountered in common applications, solving a linear system against a single right hand side with an iterative solver is normally much more efficient (in terms of both CPU cycles and memory use) than solving it with even the best sparse direct methods (Gould et al., 2007). Unfortunately, an iterative solver must start from scratch with each right hand side. A direct solver on the other hand, will factor $\mathbf{K}$ to the form $\mathbf{PKP}^T = \mathbf{LDL}^T$ (for the complex symmetric matrices in ArjunAir), where $\mathbf{L}$ is a lower triangular matrix, $\mathbf{D}$ is a diagonal matrix, and $\mathbf{P}$ is a permutation matrix. Once the factorization is known, the solution for any right hand side $\mathbf{F}$ may be found by solving the two triangular systems $\mathbf{Ly} = \mathbf{b}$ and $\mathbf{DL}^T\mathbf{U} = \mathbf{y}$ in succession. Solving the first system is called forward-substitution and solving the second is called back-substitution. The substitution process is extremely fast relative to factorization, taking approximately 1% of the factorization time in the case of ArjunAir's original solver. Sparse direct methods may also beat iterative methods for ill-conditioned systems, for which iterative methods may converge very slowly (Gould et al., 2007). Due to the large number of right hand sides that must be solved for each coefficient matrix, sparse direct methods are ideal for airborne EM modelling when there is enough memory available to store the factorization.

3.3.2 The frontal method of sparse matrix factorization

3.3.2.1 The original frontal method

ArjunAir solves $\mathbf{KU} = \mathbf{F}$ using an implementation of the frontal method of sparse matrix factorization written by the developers themselves. The frontal method was developed by Irons (1970) for use in finite-element modelling of structural problems in engineering. Modern implementations of the frontal method and its extension, the multifrontal method, may be applied to general sparse matrices (Duff, 1996). However, the original method, which ArjunAir implements, relies on the specific structure of finite-element stiffness matrices and the method in which they are assembled from submatrices corresponding to individual elements in the finite-element mesh.

Before describing the frontal method it is important to note that it was originally developed for use on real symmetric positive-definite matrices, for which $\mathbf{LDL}^T$ factorization is stable without partial pivoting. For complex symmetric matrices, such as the ArjunAir stiffness matrices, $\mathbf{LDL}^T$ factorization without partial pivoting is not guaranteed to be stable in general but will be if the real and imaginary parts of the matrix are symmetric positive-definite (Higham, 1998). Partial pivoting for stability is not performed by ArjunAir. The real and imaginary parts of the stiffness matrix were not proven to be symmetric positive definite but the diagonal entries will never be zero and they did tend to have higher magnitudes than the off diagonal entries. The effect of pivoting on solution accuracy will be discussed in Section 3.3.4. It was studied by comparing the ArjunAir solver solutions with those from the professionally developed sparse direct multifrontal solver MuMPS (Amestoy et al., 2001). For the remainder of this discussion of the original frontal method, as implemented in ArjunAir, pivoting for stability will be ignored.



(a) *(b)*

Figure 3.1: a) Small example finite element mesh and b) associated stiffness matrix sparsity pattern. Each row and column of the matrix corresponds to a node in the mesh.

In $\mathbf{LDL}^T$ factorization row operations on $\mathbf{K}$ are used to compute the factorization matrices $\mathbf{L}$ and $\mathbf{D}$. The elements of $\mathbf{L}$ and $\mathbf{D}$ are given by the formulas (Burden and Faires, 2000)

$$D_{ii} = K_{ii} - \sum_{j=1}^{i-1} L_{ij}^2 D_{jj}, \qquad L_{ij} = K_{ij} - \sum_{k=1}^{i-1} \frac{L_{jk} L_{ik} D_{kk}}{D_{ii}}. \tag{3.1}$$

The frontal method makes use of the key fact that the terms in the sums in the last two equations may be subtracted from the relevant entries of $\mathbf{K}$ in any order. Also, recall that $\mathbf{K}$ is the sum of entries from small dense matrices associated with each element in the finite-element mesh. Adding contributions to a K_{ij} from new submatrices may be done concurrently with subtracting terms for the factorization. The ordering of assembly and factorization is determined by the ordering of the elements in the mesh. Each row and column of $\mathbf{K}$ is associated with one node in the mesh. Consider a row, $\mathbf{a}^T$, of $\mathbf{K}$, associated with node i. Let $\{e\}$ be the set of elements that node i is part of. The non-zero entries of $\mathbf{a}^T$ are the entries in columns associated with nodes in $\{e\}$. Consider the toy mesh and associated matrix sparsity pattern in Figure 3.1. Node 1 is only a part of element I,

meaning it is only connected to nodes in element I. Therefore the first row of the stiffness matrix only has non-zero elements in columns associated with those nodes. Node 11, on the other hand, is part of all four elements so it is connected to all nodes and the stiffness matrix is completely dense in its row. Recall that ArjunAir uses isoparametric quadrilateral elements like the ones in Figure 3.1. If the elements are ordered column-wise, as in the figure, then a node that is part of an element of column i may only be connected to nodes in elements from, at most, columns $i-1$, i, and $i+1$.

The pieces are now in place to describe the frontal method. Start by assembling the element matrix for the first element in the mesh. This is the first frontal matrix. Determine which nodes in that first element appear in no further nodes in the mesh. Say there are n such nodes. The rows and columns of the frontal matrix associated with those nodes are called fully summed. Next, perform row and column interchanges on the frontal matrix so that the fully summed rows and columns are its first n rows and columns. Perform factorization steps to compute the entries of $\mathbf{L}$ and $\mathbf{D}$ associated with the fully summed rows. Use those entries to perform the subtractions in equation (3.1) from the required entries in the frontal matrix. At this point, the fully summed rows and columns will no longer be used so they may be removed from the frontal matrix and set aside for later use in forward and back-substitution.

Now move on to the next element and assemble its matrix. Entries of this matrix that are part of a K_{ij} already in the frontal matrix will be added to the appropriate entry. Entries of the element 2 matrix corresponding to nodes in element 2 that were not nodes of element 1 will not yet have positions in the frontal matrix. The rows and columns of the element 2 matrix associated with those nodes will form new rows and columns of the frontal matrix.

Now find all the nodes in element 2 that appear in no further elements. The rows and

columns of the new frontal matrix associated with those nodes are now fully summed. Permute the new fully summed rows and columns so that they are now the first rows and columns of the frontal matrix. Perform factorization taking pivots from the fully summed rows and then remove the fully summed rows and columns from the frontal matrix.

After that, move on to element 3 and repeat the process. Continue repeating until all the elements have been covered, alternating assembly and elimination. Forward-substitution is carried out in parallel with the factorization so $\mathbf{D}$ is not stored. The right hand sides are overwritten with the solution to the forward substitution problem $\mathbf{Ly} = \mathbf{F}$. The amount of arithmetic involved in factorization depends on the size of the frontal matrix at each step. For ArjunAir meshes, that is determined by the number of elements in each column of the mesh.

3.3.2.2 Parallelizing the frontal method by domain decomposition

There are two main approaches to modifying the frontal method to a parallelizable form. The first method is a form of domain decomposition. Within each subdomain frontal elimination occurs as described above. The subdomain partial solutions are stitched together using the Schur complement method. The second approach, known as the multifrontal method (Duff and Reid, 1983), moves the frontal method beyond finite element problems to general matrices and involves using graph partitioning tools to reorder the coefficient matrix into quasi-independent blocks. I took the first approach in writing my own parallel version of the original ArjunAir solver. The second approach is the one taken by modern professional multifrontal software packages such as MuMPS and the Harwell sparse direct factorization codes (Gould et al., 2007).

In the domain decomposition approach the finite-element mesh is broken into subdo-

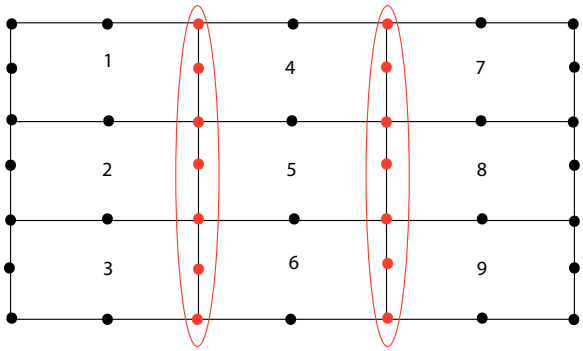


Figure 3.2: Example ArjunAir finite-element mesh divided column-wise into three subdomains

mains and frontal elimination is carried out on each subdomain. In ArjunAir, the domain is divided column-wise, as in Figure 3.2. The first column (elements 1, 2, and 3) makes up the first subdomain. The next three elements make up the second, and the last column (elements 7, 8, and 9) makes up the last subdomain. The boundary nodes are shown in red. To use the Schur complement method (Soria Guerrero, 2000, chap. 6), the full finite element system of equations $\mathbf{KU} = \mathbf{F}$ is written in the block matrix form:

$$\begin{bmatrix} K_{1,1} & 0 & \cdots & 0 & K_{1,s} \\ 0 & K_{2,2} & \cdots & 0 & K_{2,s} \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & & K_{n,n} & K_{n,s} \\ K_{s,1} & K_{s,2} & \cdots & & K_{s,s} \end{bmatrix} \begin{bmatrix} x_1 \\ \\ \vdots \\ x_n \\ x_s \end{bmatrix} = \begin{bmatrix} b_1 \\ \\ \vdots \\ b_n \\ b_s \end{bmatrix}. \tag{3.2}$$

The matrices $K_{i,i}$ on the diagonal represent the interior of each subdomain. The $K_{s,i}$ and $K_{i,s}$ matrices represent the boundaries. Frontal elimination is used to factor the $K_{i,i}$ to triangular form and zero the $K_{s,i}$. These factorizations are all completely independent and may be performed in parallel. However, each factorization will cause a new matrix to be subtracted from $K_{s,s}$. When all the interior factorizations have been performed, $\mathbf{K}$ will have been transformed to a triangular form except in the $K_{s,s}$ block, which will be transformed

to the Schur complement matrix:

$$\mathbf{\Gamma} = K_{s,s} - \sum_{i=1}^{n} K_{s,i} K_{i,i}^{-1} K_{i,s}. \tag{3.3}$$

This matrix may also be factored in parallel using the frontal technique. Considering the subdomain factorizations from the frontal perspective, rows and columns of the frontal matrices associated with interior nodes of the subdomains will be eliminated but those corresponding to the boundary nodes will never become fully summed. When all the rows and columns corresponding to interior nodes have been eliminated in the subdomain factorizations, rather than forming the full Schur complement matrix, the frontal matrices from adjacent subdomains may be combined. Consider again the example mesh in Figure 3.2. When the final frontal matrices from subdomains 1 and 2 are combined, all the rows and columns corresponding to the nodes on the first boundary become fully summed and may be eliminated from the factorization. Then the frontal matrix that remains after that process can be combined with the final subdomain 3 frontal matrix and the last rows and columns, corresponding to the nodes on the second boundary, may be eliminated. For the three subdomain example, eliminating the boundary node rows and columns is done sequentially; in the case of a greater number of subdomains, multiple boundary node factorizations may be carried out simultaneously.

3.3.3 Implementing a parallel domain decomposition frontal method

A Schur complement parallel frontal solver based on the original ArjunAir solver was implemented for this project. ArjunAir was written entirely in Fortran 90. The frontal solver implemented in the program used no external libraries. It was written entirely by the ArjunAir developers. My supervisors and I decided to parallelize using a distributed memory

approach in order to make the code fully scalable. MPI was used to implement message passing. All computations were performed in single precision complex arithmetic, with machine precision $\epsilon = 1.19209290 \times 10^{-7}$. For simplicity, the first parallel version of the code was limited to two subdomains. This version achieved good parallel speedup. After testing the two subdomain code, a general version able to handle an arbitrary number of subdomains was written. This version scaled very poorly and may have produced incorrect results. However, work on the code was abandoned in favour of using the professional multifrontal solver MuMPS, which will be discussed in Section 3.3.4. Implementation of the two subdomain solver will now be described.

There are three main housekeeping tasks required in the ArjunAir frontal method. First, when each element matrix is added to the frontal matrix, the program must know which rows and columns have become fully summed and thus ready for elimination from the frontal matrix. Secondly, the eliminated rows must be stored for later use in back-substitution. Thirdly, the row interchanges in the frontal matrix are obviously not truly performed by rearranging how its entries are stored in computer memory. The program must keep track of the correspondence between the mathematical ordering of the rows and columns of the frontal matrix and where the entries are actually stored in memory.

That last task remains unchanged in the Schur complement version of the code. The first two must be adapted for the Schur complement approach. First the boundary nodes are determined and a subdomain is assigned to each MPI process. Then each process finds the element in which each of the nodes in its subdomain will appear for the last time. For each of its interior nodes, each process will loop through all its elements, noting the elements in which that node appears. The number of the last element it appears in is stored. The final result is an array that lists the rows that may be eliminated from the frontal matrix at

each element. Each process stores its own array of eliminated rows. When all interior node rows and columns have been eliminated in both subdomains, the final frontal matrices are combined to form the Schur complement matrix, written using the notation of equations (3.3) as

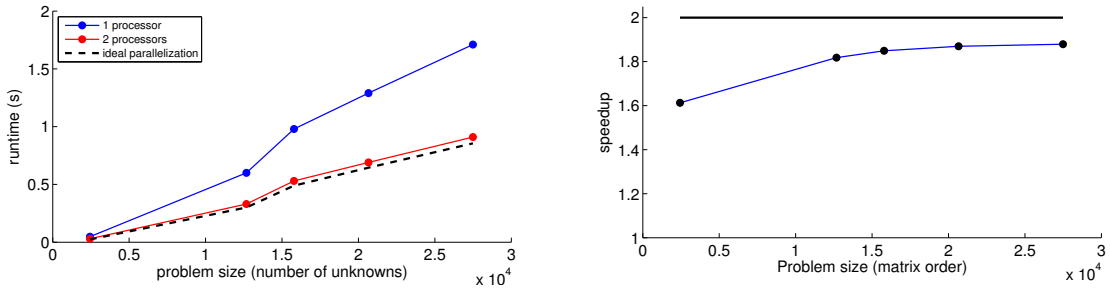
$$\mathbf{\Gamma}_2 = K_{s,s} - K_{s,1}K_{1,1}^{-1}K_{1,s} - K_{s,1}K_{1,1}^{-1}K_{1,s}. \tag{3.4}$$

$\mathbf{\Gamma}_2$ is factored on the host process and back-substitution is performed to solve for the values of the boundary node variables. Those values are then passed back to the other process and both processes perform back-substitution in parallel to solve for the interior node variables. The final results from the second process are then passed back to the host. Forward substitution is performed on each right hand side over the course of the factorization so $\mathbf{D}$ is not stored explicitly.

Performance of the two subdomain code on a range of matrix sizes is shown in Figure 3.3. All test matrices were actual ArjunAir finite-element matrices from forward modelling runs. All times used in assessing performance of the ArjunAir sparse direct solver (and the other sparse linear solvers that were tested) were averaged over all wavenumbers and all frequencies of at least two full forward modelling runs. Figure 3.3b shows parallel performance using the speedup metric, which is defined as the runtime for the sequential version of an algorithm, divided by the parallel runtime:

$$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}. \tag{3.5}$$

For a parallel program running on n processors, the maximum theoretical speedup is n. If the work is divided among n processors, the runtime should ideally be divided by n. This is called linear speedup (Pacheco, 2011). Super-linear speedups may be observed on some problems when splitting the problem up into chunks happens to speedup the rate at



(a) *(b)*

Figure 3.3: Two subdomain frontal solver performance. a) Runtime with ideal times being sequential times divided by two. b) speedup, with perfect speedup shown on horizontal black line.

which computations are performed, e.g. by lowering cache miss rates. For two processes, the ideal speedup is 2. In reality, the need for processes to send data to one another and the fact that most computations cannot be completely parallelized means that close to ideal speedups are not often achieved.

Obviously a code limited to two processes cannot be tested for parallel speedup as a function of the number of processes. It can be tested as a function of problem size. The MPI two subdomain ArjunAir frontal solver does seem to scale well with problem size. The effect of parallel overheads is reduced as problem size grows. The main reason for this is that the parallelizable section of the computation (subdomain interior factorization and back-substitution) grows at a much faster rate with problem size than the Schur complement problem. Speedup seems to be asymptoting toward approximately 1.9 for large problems. Unfortunately, this code uses more memory than the original sequential solver. For very large problems (more than 2×10^5 unknowns), speedup deteriorates due to memory bottlenecks.

As mentioned above, efforts to generalize the two subdomain code to an arbitrary number of subdomains was not completed. It was decided that time would be better spent

integrating a professional quality solver into ArjunAir. Sparse direct solvers have seen great advances in the last fifteen years (Gould et al., 2007) and their performance depends heavily on complicated implementation details. It is extremely unlikely that I would be able match the performance of such a solver.

3.3.4 MuMPS: a professional distributed memory solver

3.3.4.1 Overview

The multifrontal method, first presented by Duff and Reid (1983), provides a way to extend and parallelize the frontal method that is much more general than domain decomposition. It (along with other sparse direct matrix factorization techniques) uses graph partitioning to analyze the sparsity pattern of the coefficient matrix. This allows its rows and columns to be reordered in such a way as to divide the matrix into a set of almost independent regions. Connections between the regions may be represented by a tree structure.

Consider the tree in Figure 3.4. Frontal elimination may be performed concurrently on regions of the matrix represented by the vertices on the bottom row of the tree. Rows that can be fully summed without input from other parts of the tree may be eliminated in the same manner as the standard frontal method. Once all such variables have been eliminated, the matrices from adjacent vertices on the bottom row of the tree are combined, forming a new set of frontal matrices, represented by the second layer from the bottom of the elimination tree. Frontal elimination continues and variables that can be fully summed inside these new frontal matrices are eliminated. The process continues until only one frontal matrix remains. This final matrix is then factored using standard dense linear algebra methods. Graph partitioning is also used to choose an optimal pivot sequence for the frontal elim-

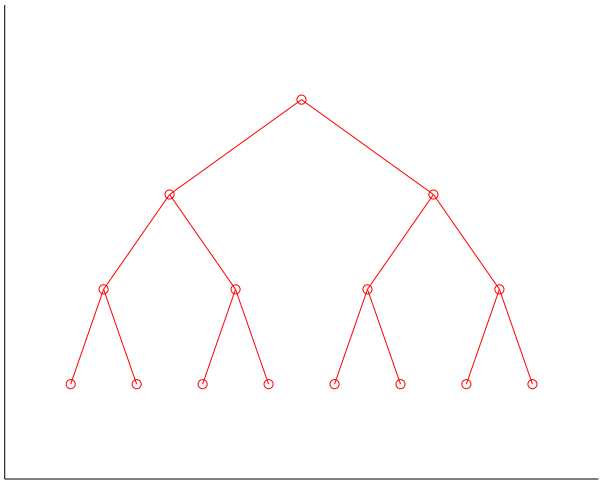


Figure 3.4: Multifrontal factorization elimination tree.

ination steps. The main goal in choosing a pivot sequence is to minimize fill-in. Fill-in refers to the destruction of sparsity during the factorization. Elimination steps will cause zero entries in the initial matrix to become non-zero, increasing the storage requirements of the factorization and the number of floating point operations required. Some fill-in is unavoidable but the quality of the fill-reducing pivot ordering used by a sparse direct solver can have a dramatic impact on performance (Amestoy et al., 2001). Determining the pivot sequence that is guaranteed to minimize fill-in is an NP-hard problem (Ng and Peyton, 1993) and all the widely used fill reducing pivot ordering techniques are heuristic.

The multifrontal method is used by the solver MuMPS. MuMPS (Amestoy et al., 2001) is an open source, MPI based code written in Fortran 90. It has access to multiple external packages for matrix reordering. In my version of ArjunAir, the METIS nested dissection reordering routine was used (Karypis and Kumar, 1999). MuMPS also relies heavily on the basic linear algebra subprograms (BLAS) to achieve high performance. The BLAS present a standard API for basic linear algebra computations such as dot products and

matrix multiplication. Very high performance on these tasks can be achieved by using BLAS implementations optimized for a particular computer architecture. For example, matrix multiplication can be performed by a block partitioning algorithm with the block size optimized to minimize cache misses on a certain type of CPU. Since the BLAS have a standardized interface, an application program that calls BLAS routines can achieve excellent linear algebra performance on any machine that has an efficient implementation of the libraries installed. Initial testing of MuMPS was performed on a desktop computer using the ATLAS BLAS (Whaley et al., 2001). Later tests on the Torngat computing cluster at MUN used the BLAS in the Intel® Math Kernel Library (Intel, 2014).

MuMPS was chosen because it is a distributed memory code, because it offers high performance matched by only a few other solvers (Gould et al., 2007), and, uniquely to my knowledge among widely distributed sparse direct solvers, it does not require the user to fully assemble the coefficient matrix. The element submatrices and how their entries map to the global matrix may be input directly to the program, which considerably simplified the task of using MuMPS within ArjunAir.

3.3.4.2 Accuracy of solutions

The accuracy of MuMPS solutions to $\mathbf{KU} = \mathbf{F}$ was compared to the accuracy of the original ArjunAir frontal solver. The single precision complex arithmetic version of MuMPS was used. Additionally, the effect of numerical pivoting was examined by comparing the accuracy of solutions with and without pivoting. Accuracy was studied using the concept of sparse backward error, developed by Arioli et al. (1989). Their paper will now be followed in briefly describing sparse backward error. Consider the problem of solving the general system of linear equations $\mathbf{Ax} = \mathbf{b}$ in floating point arithmetic using Gaussian

elimination or $\mathbf{LDL}^T$ factorization with partial pivoting. The computed solution $\bar{\mathbf{x}}$ will be the exact solution of the perturbed problem $(\mathbf{A} + \delta\mathbf{A})\bar{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b}$, where the norms of the perturbations are bounded:

$$\|\delta\mathbf{A}\| \leq \gamma\|\mathbf{A}\|, \qquad \|\delta\mathbf{b}\| = \gamma\|\mathbf{b}\|. \tag{3.6}$$

This model is not ideal for sparse matrices since the perturbations may be dense, even for sparse matrices. The zero entries of a sparse matrix are known exactly. Sparse backward error recognizes that fact. It seeks to find a bound on perturbations to entries of $\mathbf{A}$ and $\mathbf{b}$, i.e. it seeks the bound ω such that $\bar{\mathbf{x}}$ is the exact solution to $(\mathbf{A} + \delta\mathbf{A})\bar{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b}$ with

$$|\delta a_{ij}| \leq \omega|a_{ij}|, \qquad |\delta b_i| \leq \omega|b_i| \tag{3.7}$$

for all a_{ij} in $\mathbf{A}$ and b_i in $\mathbf{b}$. This bounds perturbations of structural zeroes at zero. ω may be easily computed by evaluating the formula

$$\omega = \max_i \frac{|\mathbf{A}\bar{\mathbf{x}} - \mathbf{b}|_i}{(|\mathbf{A}||\bar{\mathbf{x}}| + |\mathbf{b}|)_i}, \tag{3.8}$$

where $|\mathbf{A}|$ is the entry-wise absolute value of $\mathbf{A}$. ω is related to the relative error in the computed solution $\bar{\mathbf{x}}$ through the inequality

$$\frac{\|\bar{\mathbf{x}} - \mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \leq \frac{\omega\,\kappa(\mathbf{A}, \mathbf{b})}{1 - \omega\,\kappa_A(\mathbf{A})}, \tag{3.9}$$

where $\kappa(\mathbf{A}, \mathbf{b})$ and $\kappa_A(\mathbf{A})$ are both conditions numbers and the ∞ symbol refers to the maximum norm. The condition numbers are defined as

$$\kappa(\mathbf{A}, \mathbf{b}) = \frac{\| |\mathbf{A}^{-1}||\mathbf{A}||\mathbf{x}| + |\mathbf{A}^{-1}||\mathbf{b}| \|_\infty}{\|\mathbf{x}\|_\infty} \tag{3.10}$$

and

$$\kappa_A(\mathbf{A}) = \| |\mathbf{A}^{-1}||\mathbf{A}| \|_\infty. \tag{3.11}$$

For sufficiently small ω the relative error in $\bar{\mathbf{x}}$ is approximately

$$\frac{\|\bar{\mathbf{x}} - \mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} \leq \omega\, \kappa(\mathbf{A}, \mathbf{b}). \tag{3.12}$$

Arioli et al. (1989) describe a method of estimating $\kappa(\mathbf{A}, \mathbf{b})$.

$\kappa(\mathbf{A}, \mathbf{b})$ measures the conditioning of the problem for given $\mathbf{A}$ and $\mathbf{b}$. ω is a measure of the stability of an algorithm that computes approximate solutions of $\mathbf{Ax} = \mathbf{b}$. Clearly both quantities are needed in order to estimate the relative error in the solution. MuMPS has the ability to compute both quantities. I wrote code to compute ω for the ArjunAir original frontal solver. Comparing ω for the two solvers will provide a way to compare their stability.

Using different MuMPS pivot thresholds was also tested. Consider a fully summed row of a frontal matrix $\mathbf{a}_k^T$ with a_{kk} ready to be used as a pivot. In threshold pivoting, it will only be used as a pivot if it satisfies what Duff and Reid (1983) call a threshold criterion. The criterion is

$$|a_{kk}| > u \cdot \max_j |a_{kj}| \tag{3.13}$$

for all columns j in the frontal matrix. $0 \leq u \leq 1$ is called the pivot threshold. If the threshold criterion is not met, either another pivot is chosen from the diagonal of a fully summed row or 2×2 block pivoting is used (Amestoy et al., 2001) to select an off diagonal pivot while affecting sparsity as little as possible. The default value of u in MuMPS is 0.01. Values of 0, 0.01, and the maximum value, 0.5 were tested.

Table 3.2 shows the average value of ω for the ArjunAir solver and MuMPS on two test problems using the same mesh of 30 m wide by 10 m deep rectangular elements. The ω values were averaged over all wavenumbers and three frequencies (380 Hz, 5500 Hz, 56 kHz) for a transmitter located at the centre of the mesh. The first test problem was a homoge-

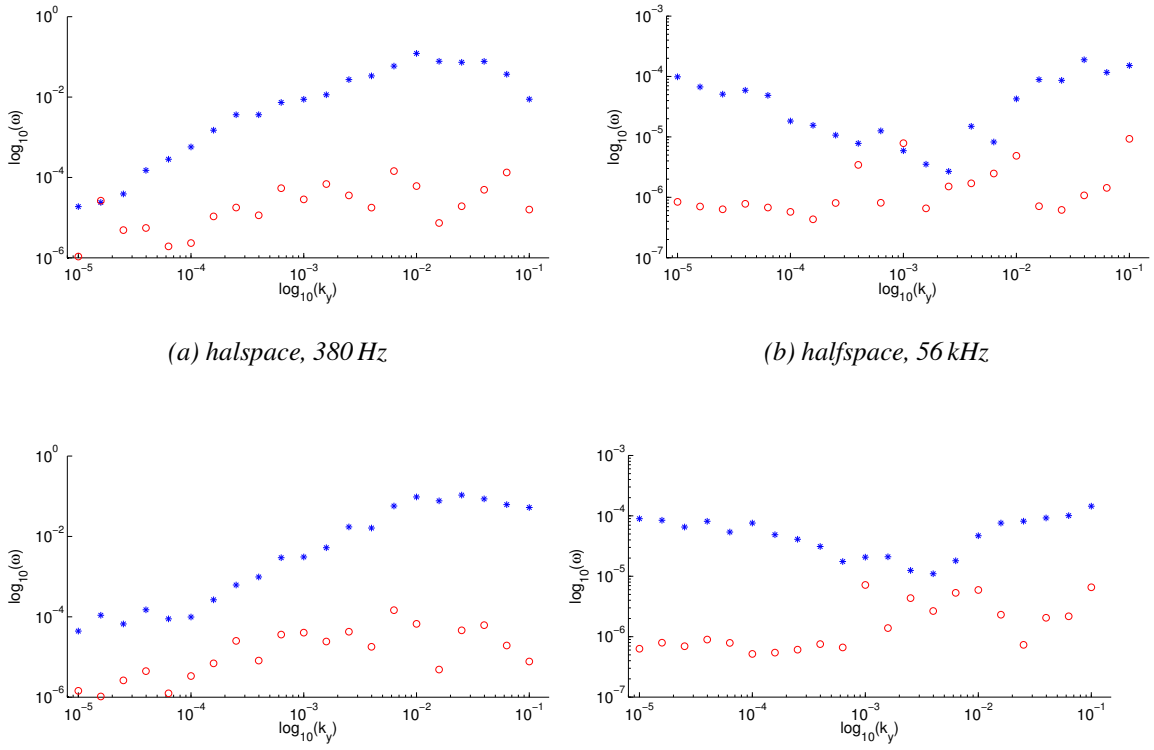
	ArjunAir solver	MuMPS, $u = 0$	MuMPS, $u = 0.01$	MuMPS, $u = 0.5$
Halfspace	0.0089	1.09×10^{-4}	1.38×10^{-5}	1.38×10^{-5}
Block	0.0096	1.04×10^{-4}	1.11×10^{-5}	1.17×10^{-5}

Table 3.2: Sparse backward error bounds ω for two test problems. Both problems use the same mesh of 7875 elements—23 811 unknowns.

neous halfspace of resistivity 500 Ωm and the second one was a 1200 m wide and 190 m thick block of resistivity 0.1 Ωm buried 30 m below the surface in a homogeneous halfspace of resistivity 800 Ωm. Tests on smaller problems and different transmitter locations gave similar results. The results in the table show that the MuMPS solutions are much more accurate overall than those from the ArjunAir solver, even without numerical pivoting. The difference is likely due to the different pivot sequence used by MuMPS, which likely led to less fill in than with the ArjunAir frontal solver, and possibly due to less careful coding of the elimination steps. However, the differences between the MuMPS solutions with and without pivoting clearly show that pivoting is important. The default pivoting level seems to be adequate, with ω being roughly equal for $u = 0.01$ and $u = 0.5$. $u = 0.01$ was chosen as the final value since choosing larger values led to longer runtimes and significantly higher memory usage.

Table 3.2 paints a somewhat damning portrait of the ArjunAir solver's accuracy but it does not tell the whole story. Figure 3.5 shows plots of $\log_{10}(\omega)$ versus $\log_{10}(k_y)$ for two frequencies, for both of the models represented in Table 3.2. The plots show a strong dependence of the ArjunAir solver ω values on k_y. Although the ArjunAir backward error bounds were almost always much higher than the MuMPS bounds they were generally much lower for small k_y. There was also a strong frequency dependence. High frequency

solutions tended to have much smaller backward error than low frequency solutions. The MuMPS ω values showed a much weaker dependence on k_y than the ArjunAir solutions. At this point it is important to remember that these solutions of the finite element equations



(a) halspace, 380 Hz *(b) halfspace, 56 kHz*

(c) block, 380 Hz *(d) block, 56 kHz*

Figure 3.5: ω vs. k_y log-log plots for a transmitter in the middle of the mesh 30 m above a homogeneous halfspace of resistivity 500 Ωm. ArjunAir original solver values are represented by blue stars and MuMPS solutions with $u = 0.01$ by red circles.

represent the along strike electric and magnetic fields in the wavenumber domain at each node in the finite element mesh. The main output of the forward solver will be one or more components of the magnetic field at the transmitter locations in the frequency or time domain. Thus to go from finite element solutions to the final forward modelling

output, cross-strike and vertical components of the field must be computed by numerical differentiation and then inverse Fourier transformed to the frequency and then possibly time domain. It is important to assess the accuracy of the solution of the finite element equations but doing so does not amount to assessing the accuracy of the forward solver. As Table 3.2 and Figure 3.5 clearly show, there were discrepancies between the overall results from the MuMPS and original ArjunAir sparse linear solvers. However, the relative discrepancies between the MuMPS and ArjunAir computed solutions were generally much smaller (on the order of machine precision) than the maximum discrepancy

$$\frac{\|\bar{\mathbf{x}}_{\text{aa}} - \bar{\mathbf{x}}_{\text{mps}}\|_\infty}{\|\bar{\mathbf{x}}_{\text{mps}}\|_\infty}. \tag{3.14}$$

To test the accuracy of the final forward modelling results, frequency domain secondary magnetic fields were computed using the original and MuMPS solvers on several homogeneous halfspace models and compared with results from the 1D airborne EM modelling software package EM1DFMFWD (Farquharson et al., 2003). For a one-dimensional earth model, the airborne EM forward modelling problem may be reduced to computing a Hankel transform integral. EM1DFMFWD computes such integrals using the digital filtering routines of Anderson (1982). Its forward modelling results can be considered correct, relative to the finite element/Fourier transform derived results of ArjunAir. Differences between the ArjunAir results using MuMPS and using the original solver were always negligible compared to discrepancies with the EM1DFMFWD solutions. Table 3.3 shows the component of the secondary field in the direction of the receiver dipole moment at the receiver locations (as computed by the 1D solver, MuMPS, and the original solver) for the homogeneous halfspace model used in the Table 3.2 and Figure 3.5 computations. The MuMPS and original ArjunAir solutions always agreed to at least 4 significant figures. Similar

	380 Hz IP	5500 Hz IP	56 kHz IP	380 Hz Q	5500 Hz Q	56 kHz Q
Orig. solver	1.0900	9.0186	197.1389	11.8097	32.9397	294.3021
MuMPS	1.0900	9.0184	197.1401	11.8100	32.9396	294.3000
1D sol.	1.0880	8.9759	200.2500	11.8000	31.7860	295.4700

Table 3.3: Secondary fields measured 30 m above a 500 Ωm homogeneous halfspace. Transmitter to receiver separation was 6.3 m at 56 kHz and 8.1 m at all other frequencies. Model cells were 10 m deep by 30 m wide. The units are parts per million (normalized by strength of primary field). IP means in-phase and Q means quadrature.

agreement was seen between the original ArjunAir solver and MuMPS solutions for block in a halfspace models of varying conductivity contrasts. Agreement with the 1D solutions is acceptable. The main points here are that inaccuracy in the ArjunAir frontal solver due to instability does not seem to be the cause of the discrepancies with the 1D solutions and that replacing the original solver with MuMPS did not significantly affect the final forward modelling results. Comparison with 1D results still does not quite tell the whole story. The secondary electric fields at all subsurface mesh nodes are required in computing the Jacobian matrix for inversion. Using MuMPS had a negligible effect on inversion results.

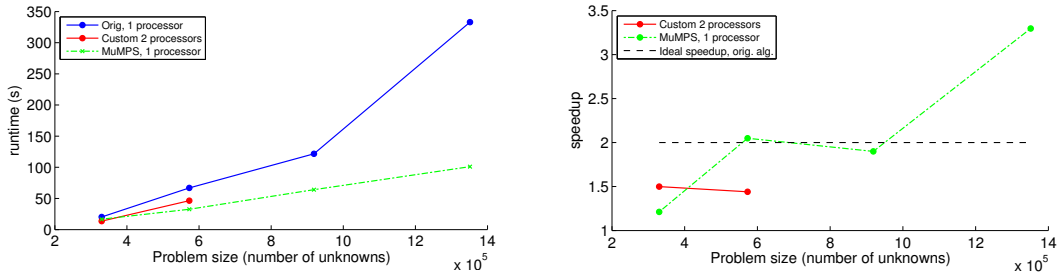
ArjunAir is already an established code so I did not extensively test its accuracy. Improving performance while maintaining the capabilities of the code was the focus. Thus, the key test of correctness was that any modifications should produce results matching the original code. Miensopust et al. (2013) recently compared the accuracy of the ArjunAir forward solver with several 3D modelling codes on models of conductive prisms of long strike length embedded in resistive halfspaces. ArjunAir agreed well with the 3D codes at frequencies of 8 kHz or greater but diverged from 3D solutions at low frequencies. This

is likely an artifact of the 2.5D approximation itself rather than a deficiency in ArjunAir's implementation of it. Low frequency fields attenuate over longer distances than higher frequency fields. It is possible that for the blocks used by Miensopust et al. (2013), effects at the ends of the conductive prism were important at low frequencies.

3.3.4.3 Performance

Even without the benefit of parallel speedup, MuMPS provided a large performance increase over the original ArjunAir frontal solver. Figure 3.6 shows average forward solve runtimes for the original ArjunAir solver, my parallel domain decomposition version, and the sequential version of MuMPS on a range of problem sizes. When running ArjunAir for standalone forward modelling the range of problem sizes shown in the figure is representative of the range of size that might be encountered in practice. In inverse modelling, linear systems with as many as 10^6 unkowns may still be encountered but the memory required to store the Jacobian matrix often limits the size of problems that can be effectively inverted to those containing smaller finite-element linear systems—e.g. with 5×10^5 or fewer unknowns.

The finite-element shape functions and their derivatives are computed in a separate routine and are the same for all frequencies and wavenumbers. Reordering the stiffness matrix to choose a pivot ordering depends only on the sparsity pattern of the matrix and not its numerical values. It therefore is the same for all stiffness matrices. Computing the reordering is done only once in a forward modelling run and is not included in the listed MuMPS runtimes. Computing the actual entries of the stiffness matrix element submatrices requires adding together products of shape function derivatives and electromagnetic constants. Those calculations must obviously be performed for each frequency and

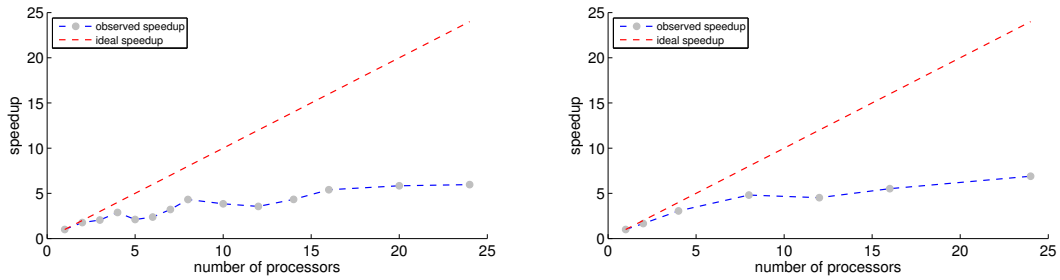


(a) *(b)*

Figure 3.6: a) Runtimes for MuMPS solver in sequential mode, along with times for original solver and my two subdomain decomposition code. b) Speedups for MuMPS and the two subdomain solver, relative to the original solver. The dashed black line shows speedup for an ideal parallelization of the original solver. Both data points for the two subdomain solver are for larger problems than any of the runs in Figure 3.3.

wavenumber. They are included in the listed linear solve runtimes. The two subdomain frontal solver crashed due to insufficient memory on the larger problems shown Figure 3.6. MuMPS showed superior performance on all problems tested. The improvements were small for small problems but the inefficiencies in the original algorithm become more costly as problem size increases, with the original solver taking, on average, 3.4 times longer than MuMPS to solve the finite element equations on the largest problem tested.

The parallel scaling of MuMPS was less impressive. It scaled acceptably up to 8 processors but using more than that yielded rapidly diminishing returns, with a speedup higher than 6 never being observed. Figure 3.7 shows speedup relative to a sequential MuMPS solve as a function of the number of MPI processes for two different problem sizes. As mentioned above the Torngat nodes all have 12 processors. When launching an MPI program, a full node is used up before assigning processes to cores on the second node. ArjunAir with MuMPS was tested on up to two full nodes, or 24 cores. Messages were passed within a node using shared memory and by infiniband interconnect between nodes. For all



(a) *(b)*

Figure 3.7: MuMPS speedup on Torngat cluster for matrix with a) 572872 unknowns and b) 1.35×10^6 unknowns.

problem sizes, the speedup curved flattened before reaching a full node, so slow message passing between nodes cannot explain the poor scaling. The scaling was likely limited by the structure of the coefficient matrices generated by the ArjunAir finite-element problem.

When choosing an appropriate parallel elimination tree structure (i.e. a way to distribute chunks of the stiffness matrix to be factored to each processor), there is often a tradeoff between load balancing and fill-in (Schenk, 2000), which limits parallel speedup. Another possible factor limiting speedup is that the problem size is too small, meaning that the overhead of communication and the non-parallelizable parts of the calculation are limiting speedup. These overheads should become less significant with problem size. I did not, however, observe a meaningful increase in speedup with increasing problem size.

Assembly of the element stiffness matrices was not parallelized, occurring only on the host processor. However, time to compute the matrix entries was almost negligible compared to time spent in MuMPS routines, taking from 2-4.5% of the combined sequential factorization and triangular substitution time, depending on the problem size. Speedup of the MuMPS computations were measured separately from the assembly process and showed slightly better speedup. Since MuMPS, the original ArjunAir solver and Pardiso,

the shared memory solver that will be described in the next section, all handle matrix assembly differently it was judged that assembly time should be included in any measure of solver performance.

3.3.5 MKL Pardiso: a professional shared memory solver

In the shared memory version of ArjunAir, The sparse direct solver Pardiso (Schenk and Gartner, 2004) was used to solve the finite-element equations. As well as offering shared memory parallelism, using Pardiso offered a chance to compare the performance of MuMPS against another very well regarded professional sparse direct code. Pardiso was chosen from among other shared memory solvers because it is very highly regarded and performed best in a comparison study of sequential sparse direct solvers—or parallel solvers run sequentially—(Gould et al., 2007). The version of Pardiso used in ArjunAir was the one included in version 10.3 of the Intel® Math Kernel Library (Intel, 2014). Having the code readily available as part of MKL was another reason for choosing Pardiso. The MKL version is not the most recent version but it was the only one available at MUN. The latest version may be acquired from the Institute of Computational Science at USI Lugano, Switzerland.

Pardiso uses supernodal factorization to factor the ArjunAir finite-element stiffness matrices. An overview of supernodal factorization can be found in Ng and Peyton (1993). As in any sparse direct factorization method, the rows and columns of the coefficient matrix are reordered to eliminate fill-in during the factorization and so that $\mathbf{LDL}^T$ factorization can be carried out independently in subregions of the coefficient matrix. Like MuMPS, MKL Pardiso uses the METIS graph partitioning package for reordering. After reordering, numeri-

cal factorization is carried out inside regions of the coefficient matrix called supernodes. A supernode is a group of contiguous columns in the reordered matrix in which factorization may be performed independently. Pardiso performs a hybrid of left and right looking block $\mathbf{LDL}^T$ factorization within each supernode. Parallelization is achieved by reordering the coefficient matrix in such a way that the supernodes may be evenly distributed among processor cores. As with MuMPS, reordering in Pardiso requires compromise between fill-in minimization and parallel load balancing.

Supernodal factorization is closely related to the multifrontal method. In fact, the multifrontal method can be thought of as a supernodal technique that uses right-looking block $\mathbf{LDL}^T$ factorization within each supernode. MKL Pardiso requires the fully assembled stiffness matrix and its sparsity pattern as input. Like MuMPS, Pardiso has the capability to perform the matrix analysis and choose a column reordering only once at the beginning of the forward solve process, using the same ordering for each subsequent solve. However, a bug in MKL 10.3 caused very large memory leaks over successive numerical factorization and solve steps, large enough to make the code completely unusable. The problem was fixed in a subsequent version of the library but I only had access to version 10.3. In ArjunAir, a fresh instance of the Pardiso solver was called for each stiffness matrix, with the reordering being performed each and every time. Reordering was fast enough, and Pardiso's performance strong enough that Pardiso outperformed MuMPS, even with all the extra reordering computations.

Assembly of the global stiffness matrix from the element submatrices was parallelized using OpenMP. When running sequentially, the assembly proceeds as a loop over elements in the mesh. The elements are numbered column-wise. Each element matrix is assembled and its entries are added to the appropriate entries of the global matrix. The loop over

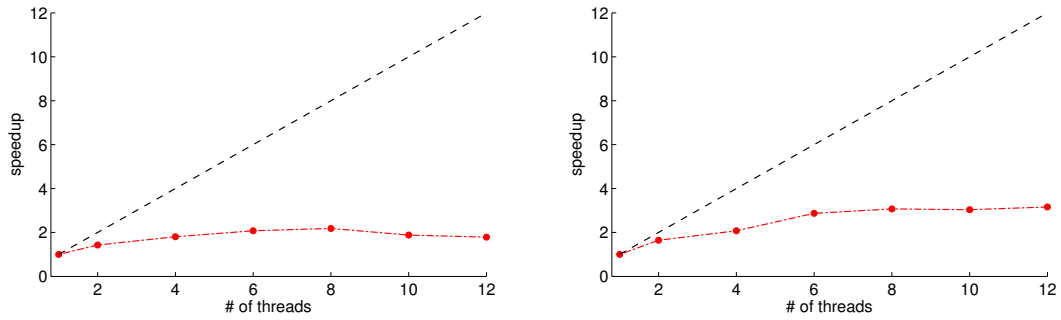
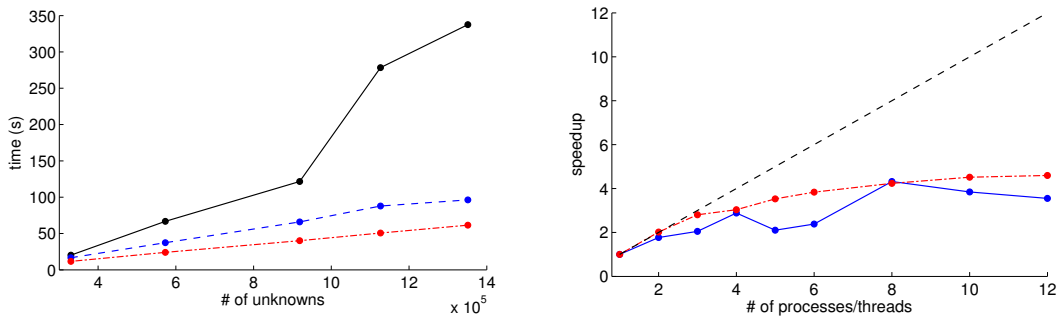


Figure 3.8: Global stiffness matrix assembly speedup: a) 27490 unknowns, 52 right hand sides, b) 572872 unknowns, 93 right hand sides.

elements was parallelized. A set number of columns of elements was assigned to each OpenMP thread. The computations of each thread are independent except for assembly of global matrix entries corresponding to nodes in the mesh on the boundaries between columns of elements assigned to each thread. Contributions to these entries are stored in temporary arrays. Once all element matrices have been assembled and summed, the boundary node temporary array entries are added to the global matrix in an OpenMP critical section, insuring that when each thread adds a boundary entry contribution to the global matrix, it adds it to an up-to-date global matrix value.

The structured nature of the ArjunAir meshes allowed very efficient assembly of the global matrix, with no searching required. In sequential mode, stiffness matrix assembly took 2.3-3.2% of the combined time spent in Pardiso reordering, factorization, and triangular substitution, depending on problem size. Synchronization of the boundary node entries limited the parallel speedup in matrix assembly. Assembly speedup on two matrices, representative of small and large ArjunAir forward problems, respectively, is shown in Figure 3.8. Because assembly took little time relative to the rest of the finite-element linear system solution time, the poor scaling had a very small limiting effect on the overall paral-



(a) *(b)*

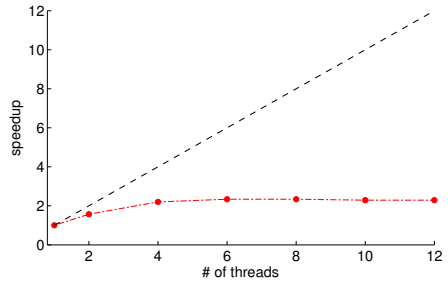
Figure 3.9: Comparison of MuMPS and Pardiso performance. The scaling results in b) are for a matrix with 572872 unknowns and 93 right hand sides. The solid black line is the original ArjunAir solver. The dashed blue line in a) shows the MuMPS runtimes, while the dot-dashed red line shows the Pardiso runtimes. In b) MuMPS speedups are shown on the solid blue line, with Pardiso speedups on the dot-dashed red line. Ideal speedup is shown on the dashed black line.

lel scaling of the total solution time. The total solution time includes assembly, reordering, factorization and triangular substitution.

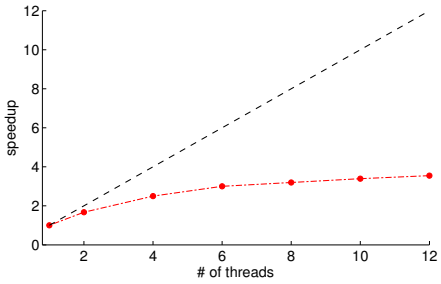
Pardiso's absolute performance was better than MuMPS and its scaling was similar. Figure 3.9a shows total sequential solution time for a number of matrix sizes, all solved against 92 right hand sides. All Pardiso runtimes are lower than the MuMPS times, even when considering matrix assembly. Pardiso also scales slightly better, as shown in Figure 3.9b. As with MuMPS, the speedup curved flattened after 4-5 threads. Scaling was tested using up to twelve threads—a full node on Torngat. It should be noted that although the representative large problem is indeed large in terms of ArjunAir inverse problems, it is not very large for a sparse linear system that might be solved by a program like Pardiso. The largest matrix tested (1.35×10^6 unknowns) yielded almost identical speedup results as the representative problem with 572872 unknowns.

Pardiso allows for more detailed performance analysis than MuMPS. It measures run-

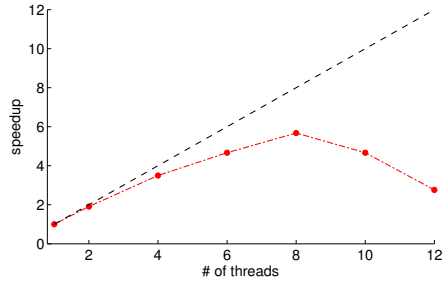
times for the individual phases of the solution procedure. Figure 3.10 shows speedup for the three main phases of the solution process and the total runtime, for the same representative large and small problems as discussed above. The small problem is likely small enough that parallel overheads simply overwhelm the amount of work that can be done in parallel. The average sequential full solve runtime for the small problem is only 0.62 seconds. Speedup is still observed up to 8 processors but beyond that scaling deteriorates significantly, likely because of synchronization overheads. For both problems, it is clear that reordering limits scaling, while factorization scales best of the three phases of Pardiso. However, poor scaling of the reordering phase cannot fully account for the poor full solution scaling, relative to the factorization and substitution scaling. The sum of the runtimes of all three phases has a maximum speedup of 5.6 on 12 threads, compared to 4.55 for the full solution. Other auxiliary calculations at the beginning and end of each call to Pardiso are not parallelizable and account for the rest of the drop in speedup. These are timed by Pardiso and included in a general category called "additional calculations."



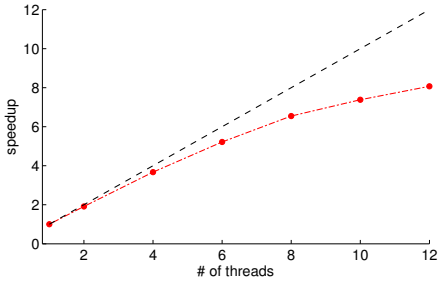
(a) reordering, small matrix



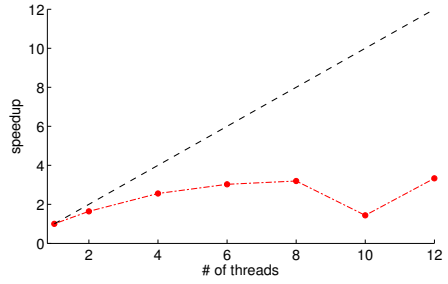
(b) reordering, large matrix



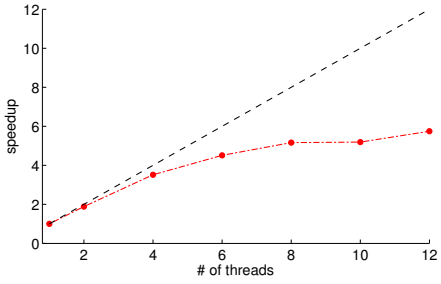
(c) factorization, small matrix



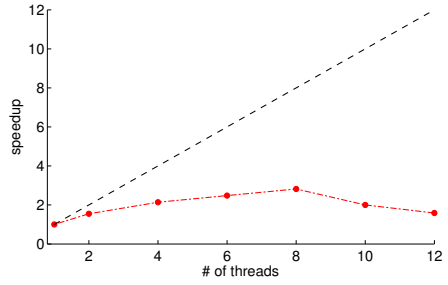
(d) factorization, large matrix



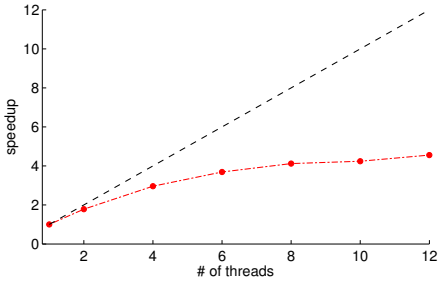
(e) substitution, small matrix



(f) substitution, large matrix



(g) full solve, small matrix



(h) full solve, large matrix

Figure 3.10: Total and individual phase speedup of Pardiso, for representative small and large linear systems.

For a fixed matrix size, Pardiso performance was also tested as a function of the number of right hand sides and compared to the ArjunAir original solver. Results for a matrix with 142442 unknowns are shown in Figure 3.11. Pardiso was run sequentially for these tests. Aside from demonstrating the superior absolute performance of Pardiso, the plot once again shows how the inefficiencies in the original ArjunAir code become more pronounced for large problems. The improved performance in Pardiso is likely the result of its performing fewer floating point operations due to reduced fill-in during factorization, as well as having better memory access patterns than ArjunAir.

Increasing the number of right hand sides, relative to the size of the coefficient matrix, also tended to increase speedup. This makes sense. The runtimes of the non-parallelizable parts of Pardiso and the poorly scaling reordering phase grow with increasing coefficient matrix size but not with increasing number of right hand sides. The best observed Pardiso speedup was on a problem with 142442 unknowns and 362 right hand sides. Speedup is shown in Figure 3.12. The problem comes from a mesh of 10 m$\times$10 m cells along a 8 km line with 20 m spacing between observation locations. The representative large problem discussed above corresponded to a fine mesh (2 m$\times$5 m cells) over a 16 km long survey

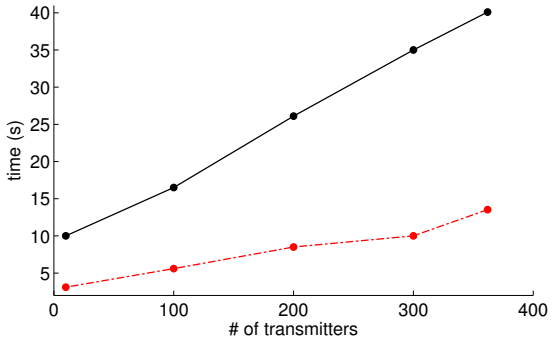


Figure 3.11: Pardiso (dashed red line) and ArjunAir original solver (solid black line) runtimes vs. number of right hand sides for a coefficient matrix with 142442 unknowns.

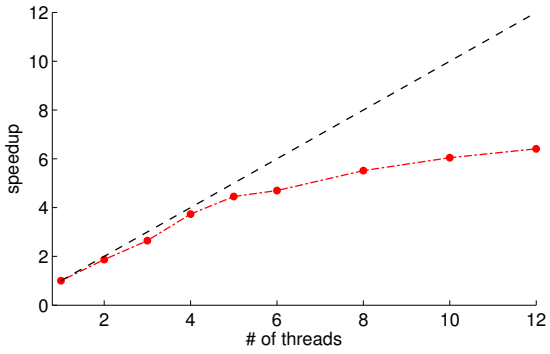


Figure 3.12: Pardiso speedup, 142442 unknowns, 362 right hand sides.

line with observation locations spaced at 175 m intervals. Users interested in detailed forward modelling would likely encounter models with scaling characteristics like the representative large problem. Those interested in preliminary inversion of large sections of an airborne EM survey would be more likely to encounter models like the one used to generate Figure 3.12. Since that second user group is likely to be significantly larger than the first, it is a satisfactory outcome that they are likely to encounter better parallel scaling than users in the first group.

3.4 Computing the primary electric field

To solve the 2D wavenumber domain BVPs that arise in ArjunAir forward modelling, the primary electric field must be computed at each subsurface node of the mesh, for each transmitter position. Recall that the primary field is the field of a dipole in free space, making anomalous conductivity equal to total conductivity. In the wavenumber domain secondary field equations that form the 2D BVPs, the primary field always appears multiplied by the anomalous conductivity. In the 2D finite-element modelling, the conductivity of the air is set to 1×10^{-10} S/m. For the purposes of the primary field computation, conductivity is set

to zero in the air, meaning the primary field is always multiplied by zero at nodes in the air and must therefore only be computed in the subsurface.

Recall that the primary fields are computed by digital filtering. Refer to Section 2.1.5 for a full mathematical description of the calculation. I note here that the primary fields in the wavenumber domain are given by the expressions

$$\tilde{E}_{px} = \frac{\omega\mu\cos\theta}{2\pi}\int_0^\infty \frac{y}{(\rho^2+y^2)^{3/2}}\sin(k_y y)\,\mathrm{d}y$$

$$\tilde{E}_{py} = (z\sin\theta - x\cos\theta)\frac{i\omega\mu}{2\pi}\int_0^\infty \frac{\cos(k_y y)}{(\rho^2+y^2)^{3/2}}\,\mathrm{d}y$$

$$\tilde{E}_{pz} = \frac{\omega\mu\sin\theta}{2\pi}\int_0^\infty \frac{y}{(\rho^2+y^2)^{3/2}}\sin(k_y y)\,\mathrm{d}y,$$

where $\rho^2 = x^2+y^2+z^2$. Digital filtering approximates the two integrals

$$g_1 = \int_0^\infty \frac{y}{(\rho^2+y^2)^{3/2}}\sin(k_y y)\,\mathrm{d}y, \qquad g_2 = \int_0^\infty \frac{\cos(k_y y)}{(\rho^2+y^2)^{3/2}}\,\mathrm{d}y \tag{3.15}$$

by sums of the form

$$g(k_y) = \sum_{j=a}^{b} f(\rho, e^{y_j})H_j. \tag{3.16}$$

At each mesh node, ArjunAir finds ρ and then computes the integrals by digital filtering. Note that the integrals do not depend on the frequency of the fields, ω. Therefore, if transmitters of different frequencies are located in the same position, the above integrals need only be computed once for each wavenumber and transmitter position. The primary fields

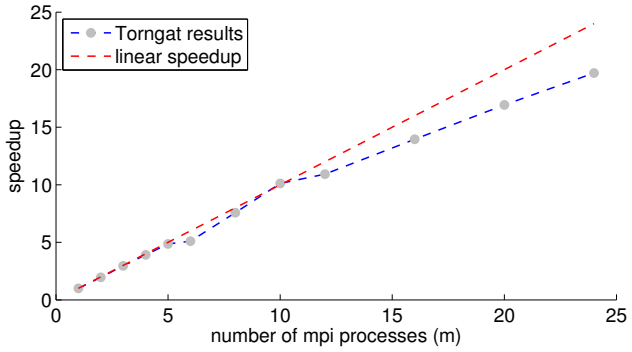


Figure 3.13: Primary field computation speedup on a mesh with 6.75×10^5 unknowns and 92 transmitters

for each frequency are computed by multiplying the values of the integrals by the constant terms in equation (3.15).

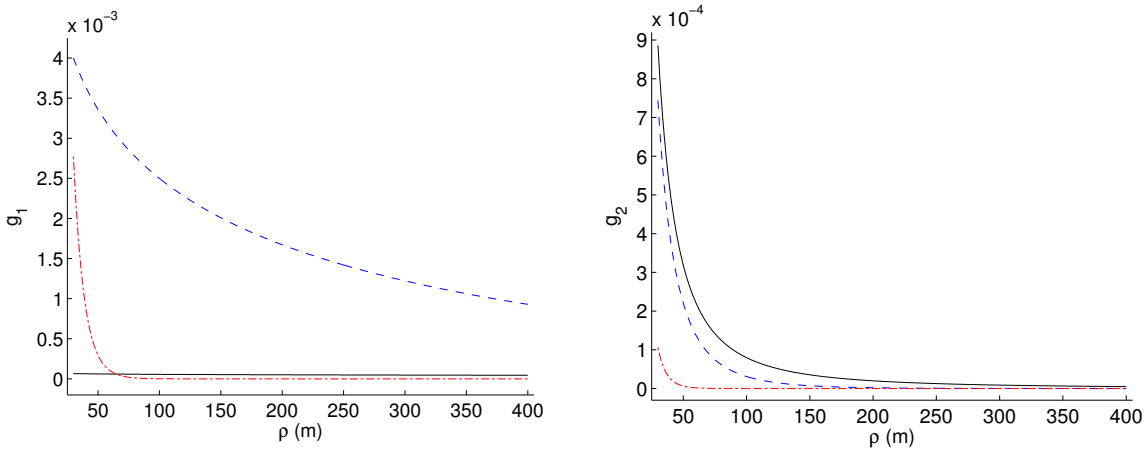
The computations at each node are completely independent, making the computation of the primary fields embarrassingly parallel. That computation was parallelized using MPI, as part of the distributed memory version of ArjunAir. The mesh nodes were evenly divided among MPI processes in a column-wise fashion and the primary fields computed in parallel. At the end of the computation, the results are passed back to the master process to be used in assembling the right hand sides of the finite element equations. Speedup results on up to two full nodes of the Torngat cluster on a mesh with 6.75×10^5 unknowns and 92 transmitters is shown in Figure 3.13. The mesh corresponds to a system of finite element equations with 1.35×10^6 unknowns—two unknowns per node, one electric field component and one magnetic field component. Speedup is almost perfect up to ten processes, as would be expected for an embarrassingly parallel calculation. Speedup deteriorates a bit after 10 processes but is still quite good up to a full 24. Recall that there are 12 processor cores, and therefore MPI processes, per node. The slight flattening of the speedup curve after 10 processes is due to the cost of having to collect results from each process on the

master at the end of the computation.

That parallel overhead from gathering intermediate results on the host process demonstrates one of the difficulties of incrementally parallelizing a sequential code using a distributed memory approach. MuMPS required that the right hand sides of the finite-element equations be assembled on the master process. Even if that were not the case, extensive modification of the code outside of the computations benefiting from parallelization would have to have been completed to avoid having to gather the primary field values at all mesh nodes on the master process. By contrast, in a shared memory approach, the primary fields may be stored in a single array that all threads can read from and write to. The downside of that approach is that conflicts between the threads in accessing that single array can slow down memory access and destroy parallel speedup in some cases.

A brute force shared memory parallelization of the primary field digital filtering integral computations was not attempted. Instead, a very efficient approximate method of computing the integrals that required a very small number of digital filtering calls was implemented. The integrals in equation (3.15) are well behaved as functions of ρ. That observation led to the idea that they could be computed once at a range of values of ρ and then interpolated to the exact node locations. Figure 3.14 shows the the value of the primary field integrals, g_1 and g_2, plotted against distance from the transmitter ρ for three representative k_y values. The figure shows that ρ dependence varies significantly as a function of k_y and between the two integrals but it is always smooth. To generate the curves, the two integrals were computed at 0.5 m ρ intervals, from $\rho = 1$ m to $\rho = 4000$ m.

ArjunAir makes use of cubic spline routines to, for example, perform the inverse Fourier transformation of the wavenumber domain modelling results to the frequency domain. The existing cubic spline routines were used to implement an interpolation scheme for comput-



(a) *(b)*

Figure 3.14: a) g_1 vs. k_y. Solid black line is $k_y = 1 \times 10^{-5}$. Dashed blue line is $k_y = 0.0016$. Red dash-dotted line is $k_y = 0.1$. g_2 vs. k_y. Solid black line is $k_y = 1 \times 10^{-5}$. Dashed blue line is $k_y = 0.0158$. Red dash-dotted line is $k_y = 0.1$. Note that the vertical axes and k_y values are different in each plot.

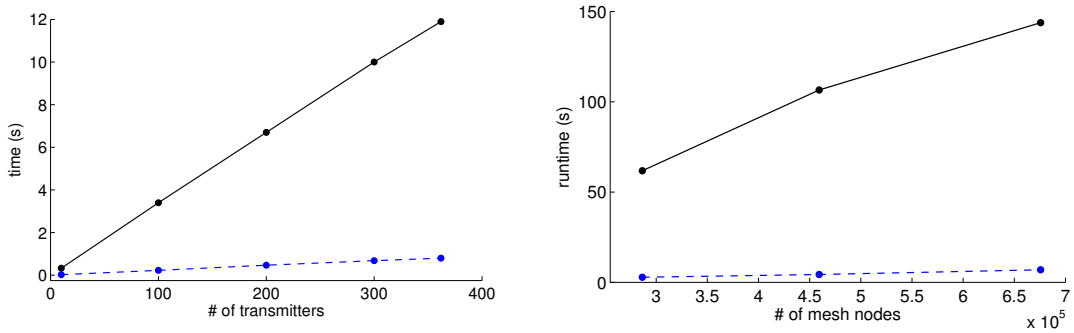
ing the primary field integrals.

For each wavenumber, g_1 and g_2 were computed at a range of values of ρ from the minimum transmitter to ground separation distance to 2 km. The computed values of g_1 and g_2 were then input to a cubic spline routine which solves a tridiagonal system of linear equations to compute the spline coefficients, which are stored in memory. Then, for each transmitter location, the values of g_1 and g_2 at each node were computed as follows. First, the distance ρ separating the transmitter and node is computed. The value of ρ and the spline coefficients are then input to a cubic spline evaluation routine. The routine finds the interval of the spline in which the input value of ρ lies and then evaluates the spline to interpolate g_1 and g_2 at the requested value of ρ. The mesh nodes are traversed in a nested loop, moving from top to bottom and then left to right across the mesh. This means that the code moves gradually between intervals of the spline, limiting the cost of searching for

the correct interval. The spline evaluation routine starts the search at each node from the interval of the previous node. The ArjunAir spline routines were developed from examples in the book *A Practical Guide to Splines* by De Boor (1978).

The ρ spacing of spline knots (and thus the length of intervals) was found empirically. A spacing of 0.5 m was used from the minimum ρ to 1 km. From 1-1.5 km a spacing of 2 m was used, and from 1.5-2 km a spacing of 10 m was used. For all k_y values, the integrals decayed to negligible levels by $\rho = 2$ km. The spline representation of g_1 and g_2 with that knot spacing was sufficient to guarantee a maximum relative error in the computed primary field of less than 1×10^{-4}. That precision was in turn enough to guarantee that negligible error was introduced in the final forward modelling and inversion results.

Evaluating the integrals by digital filtering to compute the spline representation requires approximately 4600 calls to the digital filtering routine, with the exact number depending on the minimum transmitter to ground separation distance. By contrast, the representative large problem from section 3.3.5 required approximately 2×10^8 digital filtering calls to compute the primary field at all nodes, for all transmitters, for a single value of k_y when using the original code. Evaluating a cubic spline is much cheaper than digital filtering. Even without parallelization, the interpolation method provided a huge performance improvement over using digital filtering at all nodes, with the interpolation technique taking 5-10% of the runtime of the original code primary field computation routine on all test problems. Figure 3.15 illustrates the interpolation method's performance.



(a) *(b)*

Figure 3.15: Primary field computation time, interpolation method shown on dashed blue line and original code on solid black line: a) plotted vs. number of transmitters for a fixed mesh size of 71221 nodes; b) plotted vs. mesh size for 92 transmitters on three large meshes.

3.5 Parallelization over wavenumbers

The final modification of the ArjunAir forward solver was to parallelize the loop over full wavenumber BVP solves. The initial goal was to combine this coarse grained parallelism, achieved using MPI, with the shared memory parallel implementation of the wavenumber domain BVP solver, yielding a hybrid distributed/shared memory version of ArjunAir. Excellent speedup was achieved on the Torngat cluster by running one sequential BVP solve per node on Torngat but for some unknown reason, when running over MPI, no parallel speedup was observed within each 2D BVP solve. Future work should address that problem.

Figure 3.16 shows speedup for parallelization over full 2D solves, using one MPI process per Torngat node on representative small and large problems. The timings used to compute speedup are for full forward modelling runs. Speedup is impressive but not quite as close to linear as expected. Two main factors limited the speedup. First, initial data needed for all 2D solves is computed on the master process and must be broadcast to all

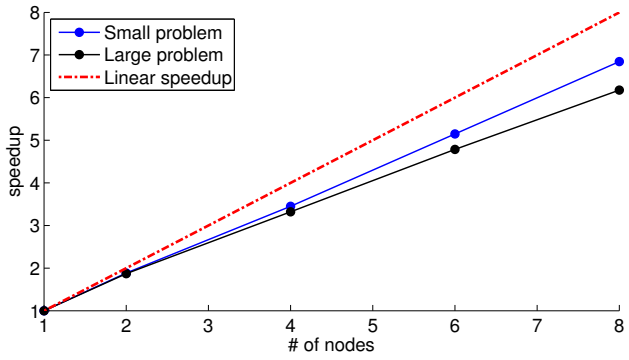


Figure 3.16: Primary field computation speedup. The small problem has a 92 transmitter locations and a mesh with 71221 nodes. The large problem has 92 transmitter locations and 6.75×10^5 mesh nodes.

the other processes before they can begin to work in parallel. The other, larger factor, is load balancing. Recall that 21 2D wavenumber domain BVPs must be solved in a forward modelling run. Thus, some processors will always have more BVPs to solve than others if the number of processes is not a divisor of 21. Each BVP requires the same amount of work to solve. Collection of final results on the master process should not be a significant limiting factor since only the fields at the observation locations need to be gathered on the master.

Parallelization over frequencies is equivalent to parallelizing over wavenumbers, up to loop ordering However, since primary fields computed for a given wavenumber may be used for all frequencies, it makes sense to group all frequencies by wavenumber. Thus only parallelization over wavenumbers was considered in this thesis and not direct parallelization over frequencies.

3.6 Summary

Overall, although parallel scaling of the solution of the linear system of finite-element equations was lower than expected, the modifications to the ArjunAir forward solver described in this chapter were able to significantly improve its performance, especially in the shared memory version using Pardiso and computing the primary field by interpolation. Speedup on a full forward solve for the shared memory solver running sequentially, relative to the original code, ranged from 2.5 on small problems where ArjunAir is efficient, to as much as 5.2 on one large problem. When running the shared memory solver on 12 cores—the most available to me for shared memory runs—speedup over the original ArjunAir ranged between 6.8 and 22.2 depending on the problem. Since one of the main reasons for improving ArjunAir's performance is to make it practical to run on larger problems than was previously possible, the results are encouraging.

Given the unique structure of the ArjunAir forward problem, a hybrid distributed/shared memory code would seem to have the potential for very large speedups, relative to the original code. The MPI solver parallelizing over full 2D problems achieved a speedup of up to 6.8 on eight nodes. If that performance could be maintained while also getting full performance from the shared memory code that solves each 2D problem, a speedup of 130 over the original ArjunAir could be achieved. Hopefully future work will be able to reach that level of performance.

Chapter 4

Computational Methods and Results II: Inversion

This chapter will describe efforts to modify the ArjunAir inversion algorithm to use the Levenberg-Marquardt method, solving iteratively for the model update. The algorithm was similar but not equivalent to ArjunAir's original damped-eigenparameter algorithm. Inversion results were comparable. The new algorithm was faster and used less memory than the original code.

4.1 Implementation of the original inversion algorithm

Section 2.2.2 of this thesis explained the ArjunAir inversion algorithm at a mathematical level. However, it omitted a detailed discussion of convergence criteria and criteria for setting the singular-value damping parameter. The algorithm may terminate in one of three ways. First, a maximum number of iterations set by the user may be reached without

convergence. Secondly, the algorithm will terminate if the misfit between observed and predicted data is less than a user specified level. Finally, the algorithm will also terminate if it judges that it has reached a local minimum in model space. If a local minimum is reached while misfit is still too large, the only recourse is to run another inversion with a different starting model.

ArjunAir's methods of adjusting the singular-value damping parameter and determining that a local minimum has been reached rely on access to the right singular-vectors of the Jacobian matrix. Therefore, new methods had to be used in the updated inversion algorithm since it does not compute the singular-value decomposition (SVD) of the Jacobian. The original convergence and damping adjustment criteria will now be discussed before describing the criteria in the modified algorithm.

Misfit between the observed and predicted data was measured using symmetric root mean squared error (RMS). Symmetric RMS was computed using the formula (Wilson et al., 2006)

$$\xi = \frac{1}{n_d}\left\{\sum_{i=1}^{n_d}\left(\frac{d_i - f_i}{\left[\frac{1}{2}(d_i^2 + f_i^2)\right]^{1/2}}\right)^2\right\}^{1/2}, \tag{4.1}$$

where n_d is the number of data points. The d_i and f_i are the individual observed and predicted data points. Symmetric RMS can be seen as the norm of the residual error

$$\xi = \|\boldsymbol{\varepsilon}\|_W = \sqrt{\boldsymbol{\varepsilon}^T \mathbf{W} \boldsymbol{\varepsilon}}, \tag{4.2}$$

where $\mathbf{W}$ is a diagonal matrix with entries

$$W_{ii} = \frac{1}{\frac{n_d^2}{2}(d_i^2 + f_i^2)}. \tag{4.3}$$

The residual error is of course $\boldsymbol{\varepsilon} = \mathbf{d} - \mathbf{f}(\mathbf{m})$, where $\mathbf{d}$ is the vector of observed data and $\mathbf{f}(\mathbf{m})$ is the vector of synthetic data computed by forward modelling with conductivity

model $\mathbf{m}$. ArjunAir inversion will terminate when the symmetric RMS drops below a user set value. The user should set the stopping value based on his or her interpretation of the dataset's noise level (Wilson et al., 2006).

Before describing the singular-value damping adjustment, some key points from Section 2.2.2 will be restated. Recall that at each main iteration of the inversion algorithm ArjunAir finds a regularized solution of the linear least squares problem

$$\underset{\boldsymbol{\delta}_m}{\text{minimize}} \quad \|\boldsymbol{\varepsilon} - \mathbf{J}\boldsymbol{\delta}_m\|_W^2, \tag{4.4}$$

where $\mathbf{J}$ is the Jacobian matrix, which has the singular value decomposition $\mathbf{J} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$. As discussed in section 2.2.1, the problem can be solved by exactly the same method in the L_2 norm, given a proper change of variables. The model perturbation $\boldsymbol{\delta}_m$, the solution to the linear least-squares problem, is computed by the formula

$$\boldsymbol{\delta}_m = \mathbf{V}\mathbf{T}\boldsymbol{\Sigma}^{\dagger}\mathbf{U}^T\boldsymbol{\varepsilon} \tag{4.5}$$

where $\mathbf{T}$ is the diagonal damping matrix with diagonal entries

$$t_i = \begin{cases} \frac{k_i^{2N}}{k_i^{2N} + (\frac{\nu}{\sigma_1})^{2N}} & : \sigma_i > 0 \\ 0 & : \sigma_i = 0 \end{cases} \tag{4.6}$$

and ν is the positive, real-valued, damping parameter.

ArjunAir does not manipulate ν directly. Rather, it adjusts the so-called relative damping parameter $\mu = \nu/\sigma_1$. All inversions start with $\mu = 0.1$ (this is hard-wired into the source code) and subsequently adjust μ heuristically based on a comparison of the actual misfit decrease achieved by a model update $\boldsymbol{\delta}_m$ and an estimate of the predicted decrease in misfit that would be achieved if the forward modelling operator was actually linear.

Let $\mathbf{r} = \mathbf{U}^T\boldsymbol{\varepsilon}$ and let r_i be the i^{th} element of $\mathbf{r}$. Jupp and Vozoff (1975) (on whom's work the ArjunAir inversion algorithm is based) define a measure of predicted residual decrease:

$$\delta L^2 = \sum_{i=1}^{p} t_i r_i^2, \tag{4.7}$$

where p is the number of non-zero singular values. This measure is used to control the value of μ. Also, when δL^2 becomes small relative to the current misfit, it indicates that the algorithm is near a local minimum, implying that significant decrease in the misfit is no longer possible. ArjunAir inversion will terminate if $\delta L^2 < 0.01\|\boldsymbol{\varepsilon}\|$.

Adjustment of μ is based on the ratio of δL^2 to the actual decrease in misfit achieved by a given model perturbation. Let the current model and residual be $\mathbf{m}_i$ and $\boldsymbol{\varepsilon}_i$, respectively. Based on those values and a given value of μ, ArjunAir computes a model update $\boldsymbol{\delta}_m$ using equation (4.5). The model is then updated as $\mathbf{m}_j = \mathbf{m}_i + \boldsymbol{\delta}_m$. A new misfit $\boldsymbol{\varepsilon}_j$ is then calculated by forward modelling with $\mathbf{m}_j$. The misfit decrease g may then be computed as

$$g = \|\boldsymbol{\varepsilon}_i\|_W^2 - \|\boldsymbol{\varepsilon}_j\|_W^2. \tag{4.8}$$

Knowing g and δL^2, adjustment of μ is very simple. If $g > (\sqrt{1 \times 10^{-7}})\delta L^2$ then the updated model $\mathbf{m}_j$ is accepted, μ is divided by 2 and the inversion proceeds to its next iteration, recomputing the Jacobian at the new model. However, if g is below the required threshold, the updated model is rejected, μ is increased (multiplied by 2) and $\boldsymbol{\delta}_m$ is recomputed using equation (4.5) with the new, larger value of μ. Larger values of μ cause small singular values to be more heavily damped, leading to smaller adjustments in less important eigenparameters. As the misfit is reduced, damping may generally be reduced also, allowing for fine tuning of more model parameters.

All the pieces are now in place to describe the full ArjunAir inversion algorithm. It is shown in Algorithm 4.1.

Algorithm 4.1 ArjunAir inversion algorithm

Specify an initial model $\mathbf{m}_0$

Set $\mathbf{m}_i = \mathbf{m}_0$

for ($i = 1$ to max # of iterations) **do**

Solve forward problem to compute the misfit $\boldsymbol{\varepsilon}_i$

Compute Jacobian **J**

Compute SVD of **J**

for ($j = 1$ to 6) **do**

Compute δL^2

if ($\delta L^2 < 0.01\|\boldsymbol{\varepsilon}\|$) **then**

Terminate inversion (At local minimum, inversion cannot be improved)

end if

Compute model update $\boldsymbol{\delta}_m$ using equation 4.5

Set $\mathbf{m}_j = \mathbf{m}_i + \boldsymbol{\delta}_m$

Solve forward problem with model $\mathbf{m}_j$ to compute $\boldsymbol{\varepsilon}_j$

if ($\|\boldsymbol{\varepsilon}_j\| \leq$ [a user specified tolerance]) **then**

Terminate inversion (Misfit reduced to acceptable level)

end if

Set $g = \|\boldsymbol{\varepsilon}_i\|_W - \|\boldsymbol{\varepsilon}_j\|_W$

if ($g > (\sqrt{1 \times 10^{-7}})\delta L^2$) **then**

Set $\mu = \mu/2$

Set $\mu =$ Max(μ,minimum value)

Set $\mathbf{m}_i = \mathbf{m}_j$

Break out of j loop (Accept model update and move on to next main loop iteration)

else

$\mu = 2\mu$

end if

if (j is equal to 6) **then**

Terminate inversion (Could not find a $\boldsymbol{\delta}_m$ that lowered misfit)

end if

end for

end for

4.2 Implementing the Levenberg-Marquardt algorithm

4.2.1 Overview

The main iterative approach to the ArjunAir inverse problem was not changed for the work described in this thesis. The solution of the linearized inverse problem was modified. This new algorithm gives similar results to the original algorithm but uses quite different computational techniques. Additionally, the computation of the Jacobian matrix was parallelized using OpenMP. The Jacobian is computed in the wavenumber domain at each wavenumber, then the frequency-domain Jacobian is formed by (inverse) Fourier transformation of the wavenumber-domain results. Computation of each entry of the wavenumber domain Jacobians is independent, meaning the computation of each one can be trivially parallelized. This was performed using OpenMP. Memory access conflicts limited speedup to an extent but a speedup of 8-8.5 on 12 threads was observed over a range of different sized test problems. Static loop scheduling was used in the OpenMP parallelization in order to limit false-sharing and other memory access conflicts but all parallel overheads could not be completely eliminated. In the context of ArjunAir, false-sharing refers to a situation in shared memory computing where different threads are working on independent elements in the same block of an array that is being stored in the local memory of multiple cores. When one core updates its element of the array, the entire block of the array must be re-written and the memory management system pauses all threads using that block of the array until it has been updated.

As discussed in Section 2.2.3, if $N = 1$, rather than $N = 2$ is taken in equation (4.6) then computing the model update with equation (4.5) is equivalent (in exact arithmetic) to

solving the system of equations

$$(\mathbf{J}^T\mathbf{J} + \nu\mathbf{I})\boldsymbol{\delta}_m = \mathbf{J}^T\boldsymbol{\varepsilon}, \tag{4.9}$$

where $\mathbf{I}$ is the identity matrix.

This brings up two new challenges. First and foremost, the system (4.9) needs to be solved. Secondly, methods for computing the damping parameter ν and detecting local minima are needed since performing both those tasks in the original algorithm required having access to the SVD of the Jacobian. In order to keep the results of the new algorithm similar to those of the original ArjunAir, the damping parameter was adjusted using a similar heuristic scheme. The relative damping parameter μ was set to 0.1 at the start of each inversion. At each iteration, the largest singular-value of the Jacobian, σ_1, is computed. Since no singular-vectors and no other singular-values are required, σ_1 can be computed cheaply using a very fast and memory efficient sparse technique.

Once σ_1 was computed, ν was computed as $\nu = \mu\sigma_1$. As in the original algorithm, the adjustment of μ was based on the ratio of actual to predicted misfit decrease. It was not possible to compute predicted misfit decrease in same way as the original algorithm, since singular vectors were no longer being computed. The ratio of actual to predicted misfit decrease can be written as (Moré, 1978)

$$\rho = \frac{\|\mathbf{d} - \mathbf{f}(\mathbf{m})\|_W^2 - \|\mathbf{d} - \mathbf{f}(\mathbf{m} + \boldsymbol{\delta}_m)\|_W^2}{\|\mathbf{d} - \mathbf{f}(\mathbf{m})\|_W^2 - \|\mathbf{d} - \mathbf{f}(\mathbf{m}) + \mathbf{J}\boldsymbol{\delta}_m\|_W^2}, \tag{4.10}$$

where the W-norm is defined as in chapter 2. Due to superior numerical properties, ρ was actually computed by the equivalent expression:

$$\rho = \frac{1 - \frac{\|\mathbf{d}-\mathbf{f}(\mathbf{m}+\boldsymbol{\delta}_m)\|_W^2}{\|\mathbf{d}-\mathbf{f}(\mathbf{m})\|_W^2}}{\frac{\|\mathbf{J}\boldsymbol{\delta}_m\|_W^2}{\|\mathbf{d}-\mathbf{f}(\mathbf{m})\|_W^2} + 2\nu\frac{\|\boldsymbol{\delta}_m\|_2^2}{\|\mathbf{d}-\mathbf{f}(\mathbf{m})\|_W^2}}, \tag{4.11}$$

as recommended by Moré (1978). The quantity ρ is analogous to the quantity g in the original algorithm. When ρ is greater than some upper threshold value ρ_{high}, the current model update is accepted, μ is divided by two and the inversion moves on to the next iteration. If ρ is less than some lower threshold value ρ_{low}, the current update is rejected, μ is multiplied by two and the model update is recomputed. The main qualitative difference between this approach and that of the original algorithm occurs when $\rho_{\text{low}} \leq \rho \leq \rho_{\text{high}}$. When ρ is in that interval, the model update is accepted and the inversion moves on to the next iteration but μ remains unchanged. This approach to adjusting the damping parameter was taken by Wright and Holt (1985), from whose work my modified ArjunAir algorithm was adapted. Multiple threshold values were tested for this study. The choices $\rho_{\text{low}} = 0.001$ and $\rho_{\text{high}} = 0.5$ gave the lowest RMS error in the final model, for all test problems.

As with the original algorithm, the approach of local minima was detected by comparing the predicted misfit decrease with the current misfit. An inversion would be abandoned if the predicted decrease became less than 1% of the misfit. Predicted decrease was computed explicitly by adding $\mathbf{J}\boldsymbol{\delta}_m$ to the current predicted data and computing

$$\delta L^2 = \|\mathbf{d} - \mathbf{f}(\mathbf{m}) - \mathbf{J}\boldsymbol{\delta}_m\|_W^2. \tag{4.12}$$

$\mathbf{J}\boldsymbol{\delta}_m$ was computed using the MKL BLAS matrix multiplication routine.

The software package PROPACK (Larsen, 2000) was used to compute σ_1. PROPACK uses Lanczos bidiagonalization with partial reorthogonalization to compute the singular values (and optionally the singular vectors) of sparse matrices. It has the advantage over dense techniques such as the Golub Reinsch algorithm (Golub and Reinsch, 1970)—used by the original ArjunAir algorithm—that its memory and CPU costs depend on the number of singular values required. That makes it quite cheap to solve for only σ_1. For example,

on a sample inverse problem on a large mesh where computing the full SVD of $\mathbf{J}$ took 132 seconds, computing σ_1 using PROPACK took 0.76 seconds. On one small problem PROPACK took 4×10^{-3} s, while the full SVD took 3.3 s. PROPACK requires user defined routines to multiply $\mathbf{J}$ and $\mathbf{J}^T$ by vectors. Those computations are carried out using the highly optimized Intel® MKL BLAS matrix multiplication routine, which is awful fast. Additionally, the threaded MKL BLAS libraries are used, offering some parallel speedup. I tested that PROPACK was working correctly inside ArjunAir by comparing its singular values with the ones computed by ArjunAir using the Golub-Reinsch algorithm. They agreed to machine precision.

4.2.2 Computing model updates with LSQR

The LSQR algorithm (Paige and Saunders, 1982) was used to solve the model update system of equations (4.9). LSQR is an iterative algorithm for solving systems of equations $\mathbf{Ax} = \mathbf{b}$ as well as damped least-squares problems of the form

$$\underset{\mathbf{x}}{\text{minimize}} \quad \|\mathbf{Ax} - \mathbf{b}\|_2^2 + \nu\|\mathbf{x}\|_2^2. \tag{4.13}$$

It is specifically designed to have strong numerical performance on least-squares problems. The algorithm, to quote Paige and Saunders' original paper, "is based on the bidiagonalization procedure of Golub and Kahan" (Golub and Kahan, 1965). For least-squares problems in exact arithmetic LSQR is equivalent to performing the conjugate gradient method on the normal equations but has better numerical properties. It does not use the coefficient matrix directly. It requires two user defined functions that will output the matrix vector products $\mathbf{Aw}$ and $\mathbf{A}^T\mathbf{v}$, respectively, for vectors $\mathbf{w}$ and $\mathbf{v}$. Since $\mathbf{J}$ is a dense matrix, the MKL BLAS general dense matrix multiplication routine was used to compute the matrix vector

products. Stopping tolerance is set by LSQR based on user input values for the relative error in the entries of $\mathbf{A}$ and $\mathbf{b}$.

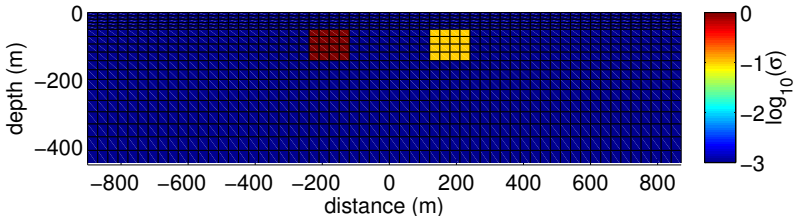
The reference Fortran implementation of LSQR was used. It is maintained by Saunders and is freely available from the Stanford Systems Optimization Laboratory webpage. Dot product and norm computations performed by LSQR are computed using BLAS routines. Solution time varies depending on the conditioning of $\mathbf{J}$ and the magnitude of ν. Larger damping parameters and better conditioned matrices will both tend to decrease solution time. Even for the slowest LSQR run measured (relative to SVD computation time), solving for $\boldsymbol{\delta}_m$ using LSQR took 0.8% of the time needed to compute the SVD of the Jacobian. In the original algorithm the SVD only needed to be computed once per main iteration of the inversion. After computing it, testing different values of the damping parameter was extremely efficient, costing only one dense matrix-vector multiplication and one vector scaling. However, computing the SVD is so expensive that it is still more efficient to use LSQR. LSQR may be called a maximum of six times per main inversion iteration. Since LSQR is clearly much more than six times faster than computing the SVD and inversions using the LSQR based Levenberg-Marquardt algorithm usually took about the same number of main iterations as the original algorithm, the new algorithm is much faster than the original.

4.3 Comparison of original and Levenberg-Marquardt inversion results

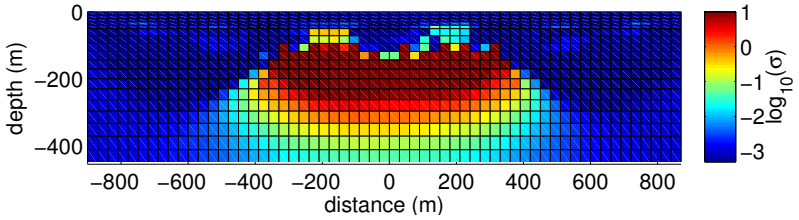
4.3.1 Quality of results

As mentioned above, the Levenberg-Marquardt algorithm is very closely related but not identical to the original ArjunAir inversion algorithm. It produced similar inversion results, as shown in Figure 4.1, which plots results from inversion of a synthetic dataset generated by ArjunAir forward modelling with a model consisting of two conductive blocks in a resistive halfspace. Synthetic data were generated from the true model for a typical frequency-domain surveying system with five frequencies. The transmitters and receivers were vertically directed dipoles at 380 Hz, 7200 Hz and 56 kHz, and dipoles directed along the survey line direction at 900 Hz and 5500 Hz. All transmitters and receivers were 30 m above the earth's surface.

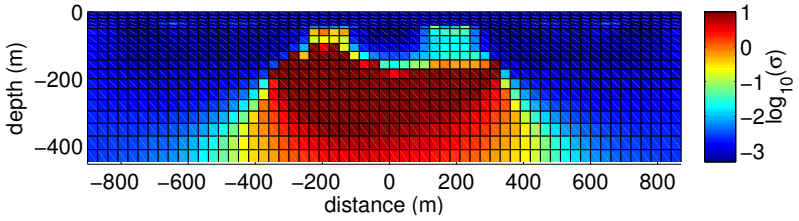
Quality of the results was heavily dependent on the starting model but that was true for the original and modified algorithms. The figure shows inversion results for both algorithms, starting from a homogeneous halfspace model. Both algorithms terminated due to a small predicted misfit decrease (i.e. they hit local minima), rather than reaching a desired RMS value. The observed and predicted values of the in-phase (real) and quadrature (imaginary) parts of the components of the secondary magnetic field in the direction of the receiver dipole moment are included in Figures 4.2 and 4.3. The modified inversion algorithm clearly results in a model that fits the data better than the one generated by the original algorithm. Qualitatively, both algorithms recovered the shapes of the tops of the conductive blocks well but left their conductivities far too low, spreading conductivity out



(a) True model



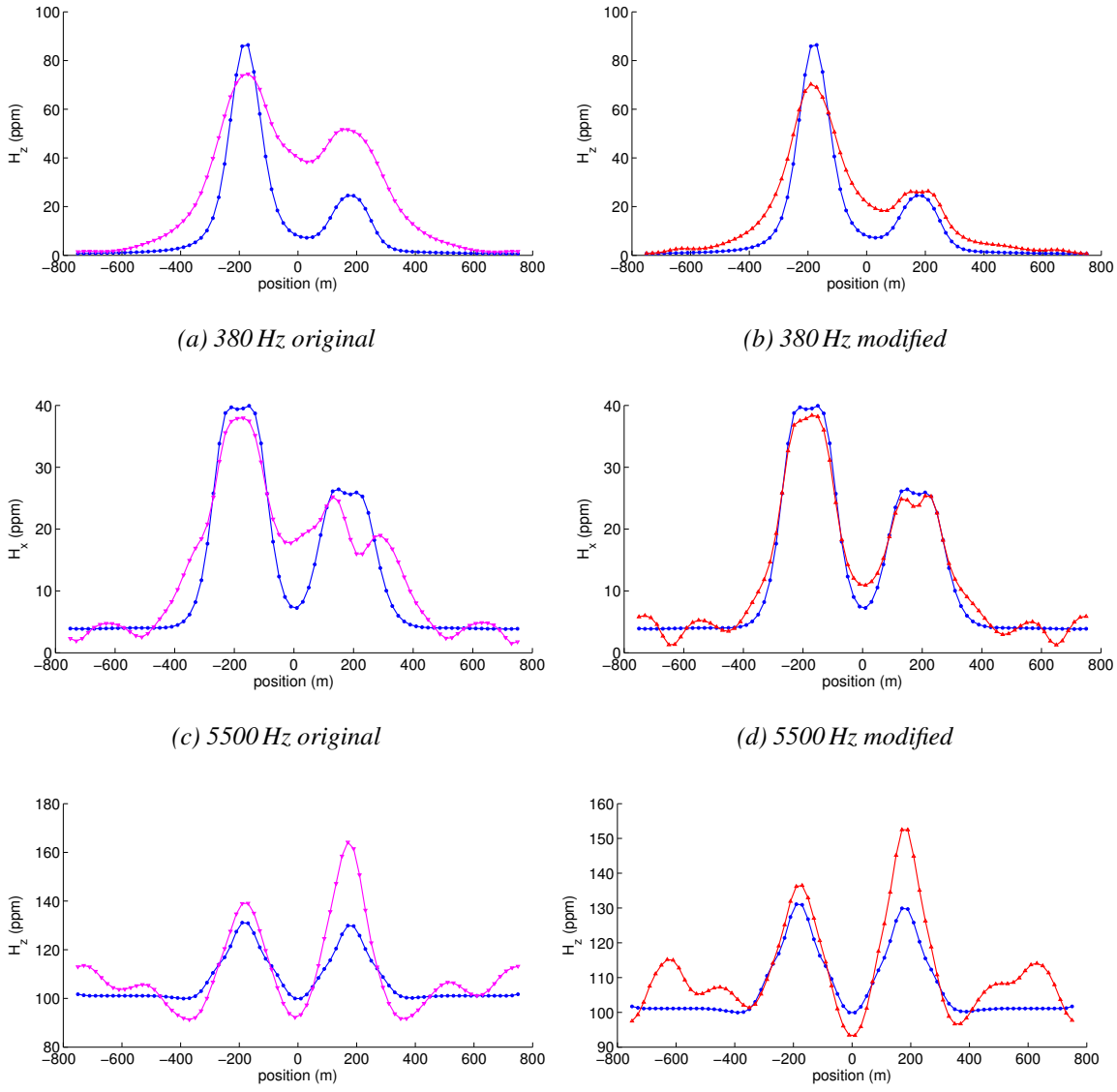
(b) Original algorithm



(c) Modified algorithm

Figure 4.1: True model shown with inversion results. The true model was a homogeneous halfspace with conductivity 1×10^{-3} S/m with two embedded conductive blocks. The left one has conductivity 1 S/m and the right 0.1 S/m. The final RMS was 42.48% for the original algorithm and 32.02% for the modified algorithm.

to greater depth. Models generated by the modified algorithm generally tended to produce broader conductive regions than the original algorithm.



(a) 380 Hz original

(b) 380 Hz modified

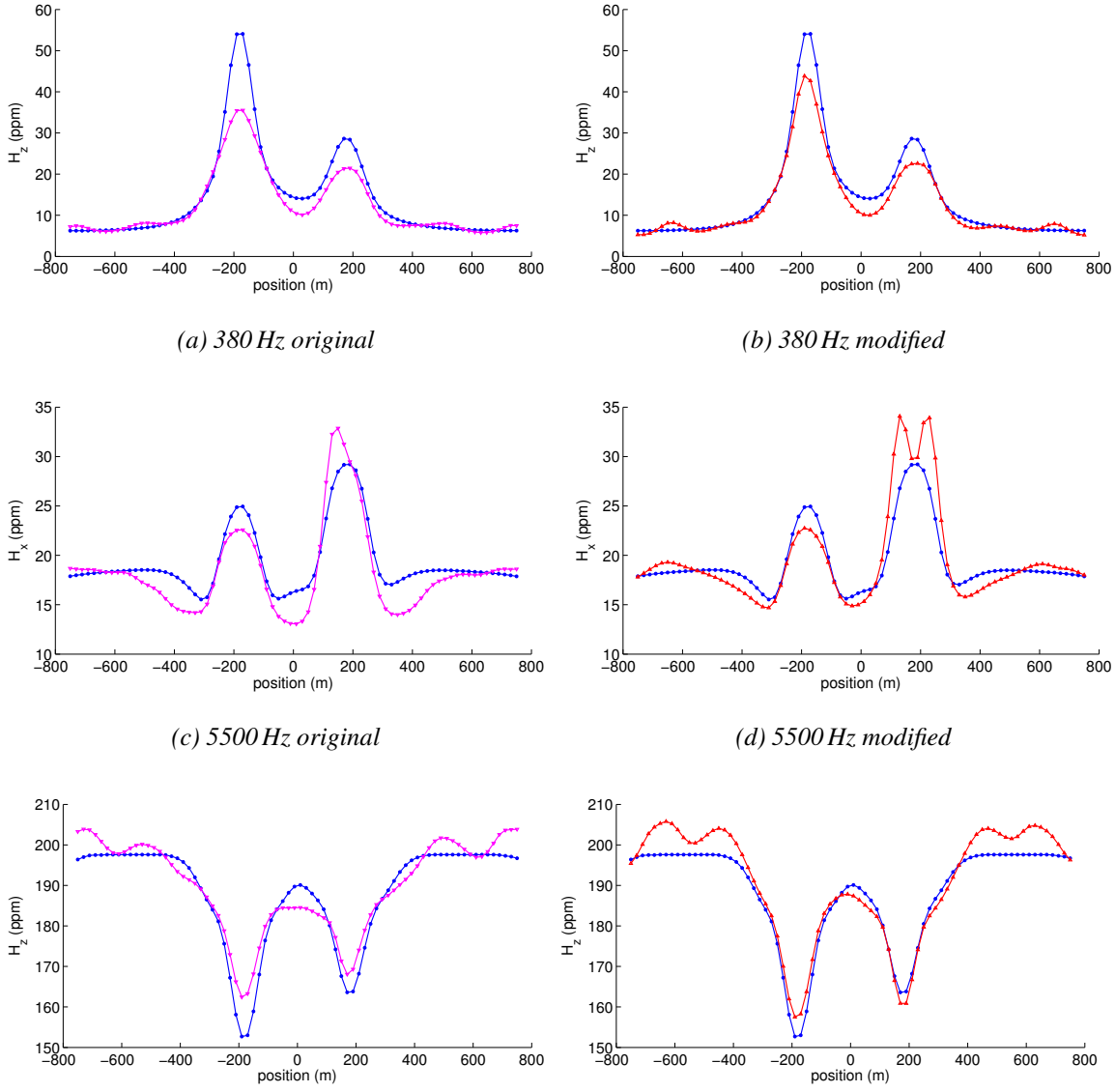
(c) 5500 Hz original

(d) 5500 Hz modified

(e) 56 kHz original

(f) 56 kHz, modified

Figure 4.2: In-phase observed and predicted data. Synthetic data from the true model shown in blue with circular data points. Predicted data is shown with triangular points , in magenta for the original inversion and red for the modified algorithm



(a) 380 Hz original *(b) 380 Hz modified*

(c) 5500 Hz original *(d) 5500 Hz modified*

(e) 56 kHz original *(f) 56 kHz, modified*

Figure 4.3: Quadrature observed and predicted data. Synthetic data from the true model shown in blue with circular data points. Predicted data is shown with triangular points , in magenta for the original inversion and red for the modified algorithm.

4.3.2 Overall performance

The inversions using the modified algorithm were incorporated into the shared memory version of ArjunAir, using the Pardiso based forward solver. I did not produce a code that used the new inversion algorithm with the original forward solver so, strictly speaking, I did not measure the speedup due solely to changing the inversion algorithm. However, comparing the combined runtimes for computing σ_1 and the model update with the cost of computing the SVD of the Jacobian indicates that even without the benefit of faster forward solves, the new inversion algorithm is faster than the old one.

For the inversion example given above, the original algorithm completed in 1926 seconds, after 8 iterations. The modified algorithm used 9 iterations and took 779 seconds to complete when running single-threaded. Convergence curves for the two algorithms, starting with the RMS from the initial model, are shown in Figure 4.4. The modified algorithm with the fast forward solver gave a speedup of 2.5 without parallelization. When running on 8 cores, the modified algorithm took 282 seconds to complete—a speedup of 2.8 relative to the single-threaded modified inversion and 6.8 relative to the original code. The parallel scaling of the modified algorithm was not particularly impressive but the speedup over the original algorithm was substantial.

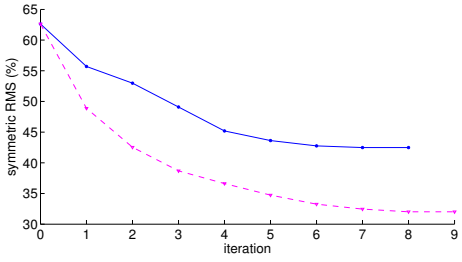


Figure 4.4: Convergence curves. Solid blue line shows the original algorithm and the dashed magenta line shows the modified algorithm.

Chapter 5

Conclusions

The forward modelling portion of this project was successful in significantly improving ArjunAir's performance without sacrificing accuracy. A robust and efficient OpenMP based shared memory solver was developed and is currently ready for production use. Prospects for a cluster oriented shared/distributed memory hybrid solver are bright. Improving the forward solver and parallelizing computation of the Jacobian gave significant speedup for ArjunAir inversions, relative to the original code. The largest forward solve speedup observed relative to the original code (22.2) was on a problem too large to be inverted, due to memory limitations.

Although the shared memory forward modelling code using MKL Pardiso to solve the finite-element equations proved to be the best option for production use, experimentation with MuMPS and with developing my own parallel sparse-direct solver yielded valuable information on how the accuracy with which the finite-element equations are solved affects the accuracy of the full forward modelling process.

Memory constraints limited the sizes of inversions that could be run to much smaller

meshes than those for the largest forward problems that were tested. Factors that affected inversion speedup included forward solver speedup, parallelization of the computation of the wavenumber domain Jacobian matrix, not having to compute the SVD of the Jacobian, as well as potential variability in the number of iterations performed by the new inversion algorithm. Speedup as high as 8 was observed for a full inversion using the new algorithm with the fast shared memory forward solver, relative to the original version of ArjunAir.

The quality of ArjunAir inversion results is highly dependent on the starting model—for both the original and modified algorithms. ArjunAir was not successful in recovering conductive anomalies starting from a homogeneous halfspace model. An improved inversion algorithm could use the ArjunAir forward solver in a more robust inversion approach such as minimum structure. For those interested in using the ArjunAir inversion algorithm as is, the modified version, using the OpenMP forward solver, offers much improved performance over the original code while providing results of equivalent quality. For those who desire results that are exactly equivalent to the original ArjunAir, the speedup in the forward solver and parallelization of the computation of the Jacobian provide much faster inversion than the original code was capable of. All of these modifications serve the overall goal of making it practical to run 2.5D inversions of airborne EM data on larger datasets than was possible before.

Bibliography

Abubakar, A., T. Habashy, V. Druskin, L. Knizhnerman, and D. Alumbaugh, 2008, 2.5D forward and inverse modeling for interpreting low-frequency electromagnetic measurements : Geophysics, **73**, F165–F177.

Amestoy, P., I. Duff, J. Koster, and J.-Y. L'Excellent, 2001, A fully asynchronous multifrontal solver using distributed dynamic scheduling: SIAM Journal of Matrix Analysis and Applications, **23**, 15–41.

Anderson, W., 1982, Fast Hankel transforms using related and lagged convolutions: ACM Transactions on Mathematical Software, **8**, 344–368.

———, 1983, Fourier cosine and sine transforms using lagged convolutions in double-precision, Open-File Report 83-320: Technical report, U.S. Department of the Interior, Geological Survey.

Arioli, M., J. Demmel, and I. Duff, 1989, Solving sparse linear systems with sparse backward error: SIAM Journal of Matrix Analysis and Applications, **10**, 165–190.

Aster, R. C., B. Borchers, and C. H. Thurber, 2013, Parameter Estimation and Inverse Problems: Elsevier.

Bono, G., and A. M. Awruch, 2008, An adaptive mesh strategy for high compressible flows based on nodal re-allocation: Journal of the Brazilian Society of Mechanical Sciences

and Engineering, **30**.

Börner, R.-U., 2010, Numerical modelling in Geo-Electromagnetics: Advances and Challenges: Surveys in Geophysics, **31**, 225–245.

Brenner, S. C., and L. R. Scott, 2008, The Mathematical Theory of Finite Element Methods, 3rd ed.: Springer.

Burden, R. L., and J. D. Faires, 2000, Numerical Analysis, 7th ed.: Brooks/Cole.

Chow, E., and D. Hysom, 2001, Assessing Performance of Hybrid MPI/OpenMP Programs on SMP Clusters: Technical report, Lawrence Livermore National Laboratory Technical Report UCRL-JC-143957.

Christensen, N. B., 1990, Optimized fast Hankel transform filters: Geophysical Prospecting, **38**, 545–568.

Constable, S., R. L. Parker, and C. Constable, 1987, Occam's inversion: A practical algorithm for generating smooth models from electromagnetic sounding data: Geophysics, **52**, 289–300.

Cox, L., G. Wilson, and M. Zhdanov, 2010, 3D inversion of airborne electromagnetic data using a moving footprint: Exploration Geophysics, **41**, 250–259.

De Boor, C., 1978, A Practical Guide to Splines: Springer.

Duff, I. S., 1996, A review of frontal methods for solving linear systems: Computer Physics Communications, **97**, 45–52.

Duff, I. S., and J. Reid, 1983, The multifrontal solution of indefinite sparse symmetric linear equations: ACM Transactions on Mathematical Software, **9**, 302–325.

EMIT, 2014, Maxwell webpage. (http://www.electromag.com.au/maxwell.php, accessed June 2, 2014).

Everett, M., and R. Edwards, 1992, Transient marine electromagnetics: the 2.5-D forward

problem: Geophysical Journal International, **113**, 545–561.

Farquharson, C., and D. Oldenburg, 2000, Simultaneous one-dimensional inversion of electromagnetic loop-loop data for both magnetic susceptibility and electrical conductivity. (GeoCanada2000).

Farquharson, C., D. Oldenburg, and P. Routh, 2003, Simultaneous one-dimensional inversion of loop-loop electromagnetic data for magnetic susceptibility and electrical conductivity: Geophysics, **68**, 1857–1869.

Gabriel, E., et al., 2004, Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation: Presented at the Proceedings, 11th European PVM/MPI Users' Group Meeting.

Glenn, W., J. Ryu, W. S.H., W. Peeples, and R. Phillips, 1973, The inversion of vertical magnetic dipole sounding data: Geophysics, **38**, 1109–1129.

Gockenbach, M. S., 2006, Understanding and Implementing the Finite Element Method: SIAM.

Golub, G., and W. Kahan, 1965, Calculating the singular values and pseudoinverse of a matrix: SIAM Journal of Numerical Analysis, **2**, 205–224.

Golub, G., and C. Reinsch, 1970, Singular value decomposition and least squares solutions: Numerical Mathematics, **14**, 403–420.

Gould, N. I., J. A. Scott, and Y. Hu, 2007, A numerical evaluation of sparse direct solvers for the solution of large sparse symmetric linear systems of equations: ACM Transactions on Mathematical Software, **33**.

Haber, E., D. Oldenburg, and U. Ascher, 2000, On optimization techniques for solving nonlinear inverse problems: Inverse Problems, **16**, 1263–1280.

Haber, E., D. W. Oldenburg, and R. Shekhtman, 2007, Inversion of time domain three-

dimensional electromagnetic data: Geophysical Journal International, **171**, 550–564.

Higham, N. J., 1998, Factorizing complex symmetric matrices with positive definite real and imaginary parts: Mathematics of Computation, **67**, 1591–1599.

Hohmann, G. W., 1987, Numerical Modeling for Electromagnetic Methods of Geophysics, *in* Electromagnetic Methods in Applied Geophysics, Volume 1, Theory: Society of Exploration Geophysicists, 313–364.

Hohmann, G. W., and A. P. Raiche, 1987, Inversion of controlled-source electromagnetic data, *in* Electromagnetic Methods in Applied Geophysics, Volume 1, Theory: Society of Exploration Geophysicists, 469–503.

HP, 2014, HP workstation webpage. (http://www8.hp.com/ca/en/campaigns/workstations/benefits.html, accessed June 15th, 2014).

Ingerman, D., V. Druskin, and L. Knizhnerman, 2000, Optimal finite difference grids and rational approximations of the square root, I: Elliptic problems: Communications on Pure and Applied Mathematics, **53**, 1039–1066.

Intel, 2014, Intel Math Kernel Library. (http://software.intel.com/en-us/intel-mkl, accessed June 9th, 2014).

Irons, B. M., 1970, A frontal solution program for finite element analysis: International Journal for Numerical Methods in Engineering, **2**, 5–32.

Jackson, D. D., 1972, Interpretation of inaccurate, insufficient, and inconsistent data: Geophysical Journal of the Royal astronomical Society, **28**, 97–109.

Jackson, J. D., 1999, Classical Electrodynamics, 3rd ed.: Wiley.

Jin, J., 2002, The Finite Element Method in Electromagnetics: Wiley-Interscience.

Johansen, H., and K. Sørensen, 1979, Fast hankel transforms : Geophysical Prospecting, **27**, 876–901.

Jupp, D., and K. Vozoff, 1975, Stable iterative methods for the inversion of geophysical data: Geophysical Journal of the Royal astronomical Society, **42**, 957–976.

Karypis, G., and V. Kumar, 1999, A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs: SIAM Journal on Scientific Computing, **20**, 359–392.

Key, K., 2012, 116, *in* Marine EM inversion using unstructured grids: a 2D parallel adaptive finite element algorithm: 1–5.

Key, K., and J. Ovall, 2011, A parallel goal-oriented adaptive finite element method for 2.5-D electromagnetic modelling: Geophysical Journal International, **186**, 137–154.

Kirkegaard, C., T. O. Sonnenberg, E. Auken, and F. Jørgensen, 2011, Salinity Distribution in Heterogeneous Coastal Aquifers Mapped by Airborne Electromagnetics: Vadose Zone Journal, **10**, 125–135.

Kong, F., S. Johnstad, T. Røsten, Westerdahl, and H., 2008, A 2.5D finite-element-modeling difference method for marine CSEM modeling in stratified anisotropic media : Geophysics, **73**, F9–F19.

Larsen, R., 2000, Computing the SVD for Large and Sparse Matrices: Technical report, SCCM, Stanford University. (A presentation given on June 16th, 2000. Available at http://sun.stanford.edu/ rmunk/PROPACK/index.html).

Last, B., and K. Kubik, 1983, Compact gravity inversion: Geophysics, **48**, 713–721.

Lee, K., and H. Morrison, 1985, N, A numerical solution for the electromagnetic scattering by a two-dimensional inhomogeneity: Geophysics, **50**, 466–472.

Long Harbour Exploration, 2014, About Uranium. (http://www.longharbourexploration.com/about_uranium/ accessed June24th, 2014).

McGillivray, P., D. Oldenburg, R. Ellis, and T. Habashy, 1994, Calculation of sensititivities for the frequency-domain electromagnetic problem: Geophysical Journal International,

116, 1–4.

Miensopust, M., B. Siemon, R. Börner, and S. Ansari, 2013, Multi-dimensional forward modeling of frequency-domain helicopter-borne electromagnetic data: AGU Fall Meeting Abstracts, A1584.

Mitsuhata, Y., 2000, 2-D electromagnetic modeling by finite-element method with a dipole source and topography : Geophysics, **65**, 465–475.

Monk, P., 2003, Finite Element Methods for Maxwell's Equations: Oxford University Press.

Moré, J., 1978, The Levenberg-Marquardt algorithm: Implementation and theory, *in* Lecture Notes in Mathematics, No. 630–Numerical Analysis: Springer-Verlag, 105–116.

Nabighian, M. N., and J. C. Macnae, 1991, Time domain electromagnetic prospecting methods, *in* Electromagnetic Methods in Applied Geophysics, Volume 2, Applications: Society of Exploration Geophysicists, 427–520.

Ng, E. G., and B. W. Peyton, 1993, Block sparse Cholesky algorithms on advanced uniprocessor computers: SIAM Journal on Scientific Computing, **14**, 1034–1056.

Oldenburg, D., E. Haber, and R. Shekhtman, 2013, Three dimensional inversion of multisource time domain electromagnetic data: Geophysics, **78**, E47–E57.

OpenMP Architecture Review Board, 2008, OpenMP Application Program Interface Version 3.0: Technical report, OpenMP Architecture Review Board.

Osgood, B., 2007, The Fourier Transform and its Applications (Lecture notes for Stanford course electrical engineering 261).

Pacheco, P. S., 2011, An Introduction to Parallel Programming: Elsevier.

Paige, C., and M. Saunders, 1982, LSQR: An algorithm for sparse linear equations and sparse least squares: ACM Transactions on Mathematical Software, **8**, 43–71.

Palacky, G., and G. West, 1991, Airborne Electromagnetic Methods, *in* Electromagnetic Methods in Applied Geophysics, Volume 2, Applications: Society of Exploration Geophysicists.

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

Powell, B., G. Wood, and L. Bzdel, 2007, Advances in Geophysical Exploration for Uranium Deposits in the Athabasca Basin : Proceedings of Exploration 07: Fifth Decennial International Conference on Mineral Exploration, 771–790.

Ramananjaona, C., and L. MacGregor, 2010, 2.5D inversion of CSEM data in a vertically anisotropic earth: Presented at the Workshop on Electromagnetic Inverse Problems.

Ray, A., and K. Key, 2012, Bayesian inversion of marine CSEM data with a trans-dimensional self parametrizing algorithm : Geophysical Journal International, **191**, 1135–1151.

Schenk, O., 2000, Scalable Parallel Sparse LU Factorization Methods on Shared Memory Multiprocessors: PhD thesis, Swiss Federal Institute of Technology, Zurich.

Schenk, O., and K. Gartner, 2004, Solving unsymmetric sparse systems of linear equations with PARDISO: Journal of Future Generation Computer Systems, **20**, 475–487.

Soria Guerrero, M., 2000, Parallel multigrid algorithms for computational fluid dynamics and heat transfer: PhD thesis, Universitat Politècnica de Catalunya.

Stoyer, C., and R. J. Greenfield, 1976, Numerical solutions of the response of a two-dimensional earth to an oscillating magnetic dipole source: Geophysics, **41**, 519–530.

Sugeng, F., A. Raiche, and L. Rijo, 1993, Comparing the time-domain em response of 2-d and elongated 3-d conductors excited by a rectangular loop source: Journal of Geomagnetism and Geolectricity, **45**, 873–885.

Telford, W., L. Geldart, and R. Sheriff, 1990, Applied Geophysics, second ed.: Cambridge

University Press.

Unsworth, M., 1991, Electromagnetic exploration of the oceanic crust with controlled sources: PhD thesis, University of Cambridge.

Ward, S. H., and G. W. Hohmann, 1987, Electromagnetic theory for geophysical applications, *in* Electromagnetic Methods in Applied Geophysics, Volume 1, Theory: Society of Exploration Geophysicists, 131–311.

Whaley, R. C., A. Petitet, and J. J. Dongarra, 2001, Automated empirical optimization of software and the ATLAS project: Parallel Computing, **27**, 3–35.

Wilson, G., A. Raiche, and F. Sugeng, 2006, 2.5D inversion of airborne electromagnetic data: Exploration Geophysics, **37**, 363–371.

Wright, S., and J. Holt, 1985, An inexact Levenberg-Marquardt method for large sparse nonlinear least squares: Journal of the Australian Mathematical Society Series B, **26**, 387–403.

Yu, W. W., 2012, Inversion of Airborne Electromagnetic data in 2.5D : Master's thesis, University of British Columbia.