Edge-elements Formulation of 3D CSEM in Geophysics: A Parallel Approach

Octavio Castillo Reyes

Department of Computer Architecture

Polytechnic University of Catalonia

This dissertation is submitted for the degree of

Doctor of Philosophy in Computer Architecture

José María Cela Espín

Josep de la Puente

September 2017

Abstract

Electromagnetic methods (EM) are an invaluable research tool in geophysics whose relevance has increased rapidly in recent years due to its wide industrial adoption. In particular, the forward modelling of three-dimensional marine controlled-source electromagnetics (3D CSEM FM) has become an important technique for reducing ambiguities in the interpretation of geophysical datasets through mapping conductivity variations in the subsurface. As a consequence, the 3D CSEM FM has real application in many areas such as hydrocarbon and mineral exploration, reservoir monitoring, CO_2 storage characterization, geothermal reservoir imaging and many others due to there quantities often displaying conductivity contrasts with respect to their surrounding sediments. However, the 3D CSEM FM at real scale implies a numerical challenge that requires an important computational effort, often too high for modest multicore computing architectures, especially if it fuels an inversion process. In this regard, the 3D CSEM FM is a key application that can benefit strongly from algorithmic and computational improvements.

On the other hand, although the High-performance Computing (HPC) code development is dominated by compiled languages, the popularity of high-level languages for scientific computations has increased considerably. Among all of them, Python is probably the language that has shown more interest, mainly because of flexibility and its simple and clean syntax. However, its use for HPC geophysical applications is still limited, which suggests a path for research, development and improvement. Therefore, this thesis reports the attempts at designing and implementing a methodology that has not been systematically applied for solving 3D CSEM FM with an HPC application baked upon Python. The net contribution of this effort is the development and documentation of a new open-source modelling code for 3D CSEM FM in geophysics, namely, the Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (*PETGEM*). The importance of having this modelling tools lies in the fact that they provide synthetic results that can be compared with real data which has a practical use both in the industry and academia. Still, available 3D CSEM FM codes are usually written in low-level languages (Fortran, C-like) whose implemented

methods are often innaccessible to the scientific community since they are commercial.

PETGEM is written mostly in Python and relies on *mpi4py* and *petsc4py* packages for parallel computations. Other scientific Python packages used include *Numpy* and *Scipy*. This code is designed to cope with the main challenges encountered within the numerical simulation of the problem under consideration: tackle realistic problems with accuracy, efficiency and flexibility. It uses the Nédélec Edge Finite Element Method (EFEM) as discretisation technique because its divergence-free basis is very well suited for solving Maxwell's equations. Furthermore, it supports completely unstructured tetrahedral meshes which allows the representation of complex geometries and local refinement, positively impacting the accuracy of the solution. The parallel implementation of the code using shared-memory and distributed-memory architectures is investigated and described throughout this document. In addition, an extensive analysis of the parallel performance factors has proved that *PETGEM* is highly scalable (hundreds of CPUs) allowing simulation of large cases for 3D CSEM FM (tens of millions of degrees of freedom).

In addition, the thesis deals with the numerical and physical challenges of the 3D CSEM FM problem. Through this work, frequency-domain Maxwell's equations have been discretised using EFEM and validated by comparison with analytical solutions and published data, proving that modelling results are highly accurate. Furthermore, this work discusses an automatic mesh adaptation strategy for a given frequency and a specific source position. The results show that adaptive mesh solutions have a factor of savings of up to four times in runtime solution compared to those computed without the use of the meshing technique. Moreover, a convergence study of the parallel Krylov subspace iterative solvers that are widely used in the literature for solving the EM problem is presented. Test results show that the GMRES method in combination with SOR is the best choice for the EM problem under consideration since it shows a better convergence rate and requires less computation time.

In summary, this thesis shows that it is possible to integrate Python with HPC for the solution of the 3D CSEM FM problem at large scale in an effective way. The resulting modelling tool is easy to use and extend and the adopted algorithms are not only accurate and efficient but also have the possibility to easily add or remove components without having to rewrite large sections of the code.

Keywords: 3D electromagnetic modelling, geophysics, Python, high-performance computing, edge finite elements.

I dedicate this thesis to my parents, Antonia Reyes Banda and Sergio Castillo, to my sisters, Esmeralda and Atalia, to my beloved wife, Magaly Basurto López, and in memory of my grandmother. I love you all dearly.

Acknowledgements

I wish to express my sincere appreciation to those who have contributed to this thesis and supported me in one way or the other during this amazing journey.

I am deeply thankful to my advisor, José María Cela Espín for accepting me in his research team, the *Computer Applications in Science & Engineering department (CASE)* in the *Barcelona Supercomputing Center-Centro Nacional de Supercomputación (BSC-CNS)*. I am very grateful for his support, guidance, encouragements and for countless time-intensive discussions during my research work; for bringing me to the project I worked on and giving enough freedom to implement and test my ideas.

My special and sincere gratitude goes out to my co-advisor, Josep de la Puente from the *BSC-Repsol Research Center*. He has provided crucial guidance and support at every moment during my research path. For comments and ideas that helped me to improve my papers as well as this thesis, and all discussions during which we were always on the same level and I never felt being rigorously supervised, but rather constructively guided. Through numerous research meetings with him, I learned not only geophysical-electromagnetic insights and wisdom but also how to communicate scientific ideas.

I express my thanks to all the people I had as friends and as colleagues from the *BSC-Repsol Research Center*: Mauricio Hanzich, Natalia Gutiérrez, Juan Esteban Rodríguez, Albert Farres, Otilio Rojas, Samuel Rodríguez and Miguel Ferrer. I express my special thanks to David Modesto for proofreading my dissertation and papers, and for his constructive remarks and invaluable discussions in which he genuinely shared he knowledge, for all questions he patiently answered. In the same way, I am deeply grateful to Claudia Rosas for her time and advice in the incredible world of methodologies for scalability prediction and the analysis of fundamental performance factors for HPC applications.

I am grateful to all of my colleagues from *CASE* of *BSC-CNS*. I am deeply thankful to Mariano Vásquez for his invaluable comments and constructive criticism that absolutely improved this manuscript. A deep thanks to Xevi Roca, Abel Gargallo and Miguel Zavala for all the fruitful discussions on numerical modelling, finite elements

and mesh generation techniques.

I wish also to thank all members of *BSC-CNS*. Above all, I want to thank Mateo Valero for his support and encouragement. His vision for an excellent and relevant science is reflected in every corridor of the *BSC-CNS*. For me, it has been an honor to be part of this research center that is full of interesting people who constitute a warm and welcoming environment for the development of quality science and research. Thanks for opening the doors of *BSC-CNS* and for showing me the extraordinary and magnificient supercomputing architecture that prides Spain: *Marenostrum*. I would also like to express my gratitude to Ulises Cortés for his advice and for the experiences organizing the *Jornadas de Cooperación CONACyT-Catalunya*. At the same time, I am greatly indebted to the Administrative Staff and Support Team of the *BSC-CNS* for all their management and technical support to carry out my research.

Beyond the borders of the *BSC-CNS*, several other people also deserve my thanks for their help. In November 2015 and March 2017, I visited the *Maguique 3D* research group at the *National Institute for Computer Science and Applied Mathematics (INRIA)* in Pau, France. I am extremely grateful to Hélène Barucq, Julien Diaz and Victor Péron for many useful discussions about the core of my research work and their hospitality during my visits. Thank you for your invaluable comments and for sharing your knowledge of geophysical electromagnetic modelling. In October-December 2016 I visited the *Pretroleum and Geosystems Engineering* research group at the *University of Texas (UT)* in Austin, USA. I am thankful to the Prof. Carlos Torres-Verdin for his willingness to be my host and for helping me to improve my work by sharing his broad experience with kindness. During my stay at *UT* I was able to learn about modelling tools in industry and some advanced programming practices which are carried out in one of the most powerful supercomputers in the world: *Stampede*. Also, I would like to express deep gratitude to Prof. Yonghyun Chung from *Seoul National University (SNU)* in South Korea, for his kindness and hospitality during my visit in July 2016. Our flash meetings enriched not only my research work but also my knowledge about the exciting Korean culture.

I would also like to thank colleagues and friends in my beloved Mexico, mainly those from *Universidad Veracruzana*. First, none of this would have happened without the support of Dr. Mario Miguel Ojeda, who believed in me enough to offer me my first laboral opportunity and, some years after, for give me his recommendation which opened me a scientific world with endless opportunities. I am very grateful for his guidance, encouragements and for countless discussions about my PhD project and my academic carrer. I wish also to thank Dr. Carlos Manuel Welsh Rodríguez for

sharing his scientific vision with me and for provided me a good platform to present and openly discuss ongoing work. I also thank to MSc. Rubén González Benítez for his support, advice and friendship that has lasted since he directed my master's thesis. My gratitude extends to other academic and research institutes in Mexico for their kindness and hospitability: *Centro de Investigación en Computación-(CIC-IPN)* and *Centro de Investigación y de Estudios Avanzados-(CINVESTAV-IPN).*

I would like to express my sincere gratitude to the *Mexican National Council for Science and Technology (CONACyT)* for the financial support to develop my thesis. I am convinced of the value of the *CONACyT* scholarship program and although investment in science remains a long and difficult route, the progress is irreversible. I wish also thanks to the *European Union's Horizon 2020* research and innovation programme for his financial support under Marie Sklodowska-Curie grant agreement No. 644202 which allowed me to carry out 3 research stays that improved my work and my scientific background.

I am indebted to all my friends in Barcelona who opened their homes to me during my time at *BSC-CNS* and who were always so helpful in numerous ways. Special thanks to Benítez Domínguez family: Luisa, Fernando, Gemma, Alba, Sira and Leo. Thank you for your hospitality and for sharing wonderful and unforgettable moments, as well as for the long and interesting conversations that extended my vision and knowledge about Spanish culture. Their friendship is one of my greatest non-academic achievements of which I am proud.

I would like to express my love and gratitude to one of the dearest hopes of my life, my parents, Sergio Castillo and Antonia Reyes Banda. I will never forget your gestures of protection, of affection and of indulgence, all of them will remain sculpted in my heart forever. You have forged my soul with discretion under the magnifying glass of your example and the little that is good in me, carries your emblem. I always thank to my sisters Esmeralda and Atalia for his emotional support and for taking care of my parents during my long absence, as well as for always following me with your thoughts and for sharing my dreams. I am deeply thankful to my uncle Mauricio Reyes Banda for his support and for his continuous calls that kept us close despite the physical distance between us. I also express my deep gratitude to my parents-in-law for their support and prayer for my wife Magaly Basurto López and me.

Finally, my gratitude goes to Maggie, who has been by my side throughout this PhD, living every minute of it, and without whom, I would not have had the courage to carry it to a successful conclusion. She had faith in me and my intellect even when I felt like digging hole and crawling into one because I did not have faith in

myself. I am grateful to her for so many unforgettable moments and for being an inexhaustible source of joy and strength, as well as being a compass that helped me to reorganize my priorities and efforts. I think we have learned a lot about life and this stage strengthened our commitment and determination to live to the fullest. Thank you Maggie for sharing with me your happiness for life, thank you for exploring the city and the world at my side, as well as for our long and interesting conversations on topics beyond computer architecture and science. I have always felt happy to share my life with you.

Contents

List of figures

List of tables

Chapter 1

Introduction

The science of exploration geophysics applies the principles of physics to study the Earth. Geophysical surveys of the interior of the Earth involve taking measurements at or near the Earth's surface that are influenced by the internal distribution of physical properties. Analysis of these measurements can reveal how the physical properties of the Earth's interior vary vertically and laterally. By working at different scales, geophysical prospecting (GP) methods may be applied to a wide range of investigations from studies of the entire Earth (Kearey et al., 1996) to exploration of a localized region of the upper crust for engineering or other purposes of great societal value such as environmental surveys (e.g. Gibson et al., 1996; Marchetti et al., 2002; Gajewski et al., 2005), water prospecting (e.g. Perttu and Wikberg, 2005), geothermal energy applications (e.g. Koon and Ufondu, 2015) and oil & gas sector (e.g. Davydycheva et al., 2003; Strack and Aziz, 2012; Puzyrev et al., 2013; Chung et al., 2014).

An alternative method of investigating subsurface geology is, of course, by drilling boreholes, but these techniques are invasive and expensive and provide information only at discrete locations. GP, although sometimes prone to major ambiguities or uncertainties of interpretation, provides a relatively rapid and cost-effective means of deriving a really distributed information on subsurface geology. In the exploration for subsurface resources the methods are capable of detecting and delineating local features of potential interest that could not be discovered by any realistic drilling technique. Geophysical surveying does not dispense with the need for drilling but, properly applied, it can optimize exploration surveys by maximizing the rate of ground coverage and minimizing the drilling requirement.

Regardless of the scope or scale of the surveys, exploration geophysics methods are based on studying the propagation of the different physical fields within the Earth. In the context of geophysical exploration, the main target of these methods is to

build a constrained model of geology, lithology and fluid properties based upon which commercial decisions about reservoir exploration, development and management can be made (Koldan, 2013). Nowadays, the three main technologies in applied geophysics are: seismic methods, potential field methods (magnetic and gravity approaches) and electromagnetic methods. Each of these methods processes a set of data within an integrated framework, so that the resultant Earth model is coherent with all data used in its construction.

In the oil & gas sector, the seismic methods have become a standard technique for obtaining high-resolution images of structure and stratigraphy of the Earth. However, seismic data have extremely poor sensitivity to changes in the type of fluids, such as water, brine, oil & gas. It is the main reason why in some scenarios it is difficult, if not impossible, to determine fluid properties from seismic data. The drawback of the seismic method of determining the presence of oil in a formation, encouraged the development of new methods aimed to strengthen the models and reduce uncertainty. In this sense, the electromagnetic methods (EM) have received special attention from industry and academia.

On top of that, the last decade has been a period of rapid growth for EM in geophysics, mostly because of their industrial adoption. In particular, the 3D marine controlled-source electromagnetic (3D CSEM) method has become an important technique for reducing ambiguities in data interpretation in hydrocarbon exploration. Furthermore, in order to be able to predict the EM signature of a given geological structure, modelling tools provide us with synthetic results which we can then compare to real data. In particular, if the geology is structurally complex, one might need to use methods able to cope with such complexity in a natural way by means of, e.g., an unstructured mesh representing its geometry. Among the modelling methods for EM based upon 3D unstructured meshes, the Nédélec Edge Finite Element Method (EFEM) offers a good trade-off between accuracy and number of degrees of freedom (DOFs), i.e. size of the problem. Furthermore, its divergence-free basis is very well suited for solving Maxwell's equations.

This thesis provides the numerical formulation of EFEM 3D CSEM forward modelling (FM) in geophysics and its implementation on massivelly parallel computers using an interpreted language, namely, Python. It enables the possibility to specify edge-based variational forms of $H(curl)$ for the simulation of electromagnetic fields in real 3D CSEM FM surveys with high flexibility accuracy. The new modelling tool is based on two main contributions controlling the structure of the document: firstly, the integration of EFEM and cutting-edge High-performance Computing (HPC) technolo-

gies for efficient 3D CSEM FM. Secondly, the study of the scopes of the computational tool through real-scale modelling. Its solution is evaluated by comparison to well-established 3D CSEM models. The flexibility, accuracy and modularity of the code makes it a competitive tool to simulate real scenarios of 3D CSEM FM in geophysics.

The purpose of this Chapter is to give a brief review of the 3D CSEM FM in exploration geophysics. Next, thesis contributions are introduced with emphasis on state-of-art modelling challenges. Finally, the thesis objectives are presented.

1.1 3D CSEM FM in geophysics

EM are an established tool in geophysics, with application in many areas such as hydrocarbon and mineral exploration, reservoir monitoring, CO_2 storage characterization, geothermal reservoir imaging and many others. Presumably the most successful application for oil prospecting has being well logging which was introduced by the Schlumberger brothers (Johnson, 1962). Jishan and Lizhi (1999), Tang et al. (2007), Xue et al. (2008) and Constable (2010) summarized and reviewed the development of EM survey techniques in terms of instrument, acquisition, processing and interpretation, numerical simulation, and application respectively.

For the diversity of remote sensing techniques that were deployed in the past century, EM seem to be of less importance compared to seismic techniques in the oil & gas sector. However, the activity in EM for exploration has not been absent, and in the 1970's and 1980's, improved equipment and increasing data-processing power led to extensive development. Nowadays, EM is a fundamental tool in the oil & gas industry because of the hope that the application of such methods would eventually lead to the direct detection of hydrocarbons through their insulating properties.

The electromagnetic properties of a medium are described by three elements: the electric permittivity or dielectric constant ϵ_r (Fm^{-1}), magnetic permeability μ (Hm^{-1}) and electric conductivity σ (Sm^{-1}) or its reciprocal called electric resistivity $1/\sigma$ (Ωm). Since the Earth is conductive, the attenuation of propagating waves becomes more severe as the signal frequency increases (Løseth, 2007). Hence, the aim of the most EM in geophysics is to measure the resistivity of the Earth materials. A strong description of the different methods of resistivity measurement and different possibilities to vary the source type (e.g. electric or magnetic dipole) and source signal (e.g. time-harmonic, direct current, or transient) can be found in (Nabighian, 1988).

Since the electric conductivity measure in a region could describe the properties and distribution of fluids in this area, EM for prospecting have become a standard

technique in oil & gas industry for mapping variations in the subsurface. In (Eidesmo et al., 2002; Edwards, 2005; Sheard et al., 2005; Srnka et al., 2006; Newman and Commer, 2009; Gray et al., 2012; Chung et al., 2014; Cai et al., 2017) can be found several examples which establish that the electrical conductivity of petroleum, gas or hydrate bearing sediments is based on the concept of an increased resistivity in hydrocarbon-rich zones. Because the ability to map significant contrast between electric conductivity, EM are very useful for detecting hydrocarbon locations.

3D CSEM FM techniques, also referred as seabed logging or marine 3D CSEM (Eidesmo et al., 2002), can be divided into two groups depending on the domain in which collected data is interpreted: time domain (TDEM) or frequency domain (FDEM). This thesis focuses in FDEM theory. In 3D CSEM FM, a deep-towed electric dipole transmitter is used to produce a low frequency electromagnetic signal (primary field) which interacts with the electrically conductive Earth and induces eddy currents that become sources of a new electromagnetic signal (secondary field). The two fields, the primary and the secondary one, add up to a resultant field, which is measured by remote receivers placed on the seabed. Since the electromagnetic field at low frequencies, for which displacement currents are negligible, depends mainly on the electric conductivity distribution of the ground, it is possible to detect thin resistive layers beneath the seabed by studying the received signal (Koldan, 2013). Operating frequencies of transmitters in 3D CSEM FM may range between 0.1 and 10 Hz, and the choice depends on the dimensions of a model. In most studies, typical frequencies vary from 0.25 to 3 Hz, which means that for source-receiver offsets of 10-12 km, the penetration depth of the method can extend to several kilometres below the seabed (Hanif et al., 2011; Koldan, 2013). The main disadvantage of 3D CSEM FM is its relatively low resolution compared to seismic imaging. Therefore, 3D CSEM FM is often used in conjunction with seismic surveying as the latter helps to constrain the resistivity model. Figure 1.1 depicts the marine CSEM. Although 3D CSEM FM is nowadays a well-known geophysical prospecting tool and his fundamental mathematical theory is well-established, the state-of-art shows a relative scarsity of robust, flexible, modular and open-source codes to simulate these problems on HPC platforms, which is crucial in the future goal of solving inverse problems. In this regard some examples of modelling tools for geophysical prospecting are (Mackie et al., 1994; Alumbaugh et al., 1996; Xiong et al., 2000; Zyserman and Santos, 2000; Badea et al., 2001; MacGregor et al., 2001; Fomenko and Mogi, 2002; Newman and Alumbaugh, 2002; Davydycheva et al., 2003; Key and Weiss, 2006; Franke et al., 2007; Kong et al., 2007; Li and Key, 2007; Li and Constable, 2007; Abubakar et al., 2008; Davydycheva and Rykhlinski,

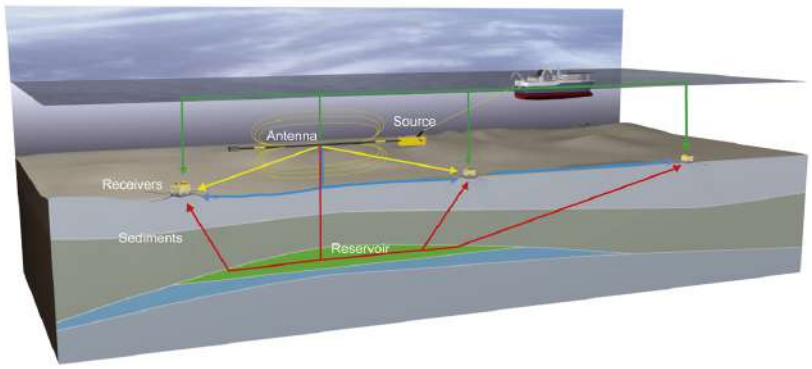


Fig. 1.1 Marine CSEM.

2011; Li and Dai, 2011; Puzyrev et al., 2013; Koldan, 2013). However, the tools that full fit needs for the solution of real models are commercial and often are inaccessible to the wider scientific community. Furthermore, due to the discretization method employed, not all codes that are affordable to community are capable of dealing with complex geometries and non-uniform bathymetries, which reduces or limits his use in situations which irregular and complicated geology has a significant influence on measurements. Additionally there are few parallel codes that are efficient, scalable and can deliver good performance.

This thesis proposes the development and documentation of a new parallel python code that meet the requirements of 3D CSEM FM at real-scale, namely, Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (*PETGEM*). It has been designed to cope with the various situations encountered within the numerical simulation of the 3D CSEM FM in geophysics. The code is based on the EFEM and pure Python language that allow users an easy adaptation to various 3D CSEM models and their fast execution on HPC arquitectures.

1.2 Present modelling challenges of 3D CSEM FM

3D CSEM FM maps resistive bodies such as carbonates, hydrocarbon filled sediments, volcanic rocks and salt from the seabed. Particularly in offshore hydrocarbon exploration, data regarding resistivity mappings beneath the seafloor is crucial and useful, namely, high resistivity of hydrocarbon filled rocks (30-500 Ωm) compared to bodies filled with saline formation water (0.5-2 Ωm) is usually a good indicator for the presence of oil & gas (Constable and Weiss, 2006; Constable and Srnka, 2007; Key, 2009). Because its capacity to detect, identify and characterize the target reservoir, the 3D CSEM modelling is an attractive and convincing method to conduct exploration cam-

paigns, thus increasing the drilling success rate as well as reducing associated cost and hazards.

The numerical properties of 3D CSEM FM algorithms in hydrocarbon exploration should be particularly sought for:

1. **Accuracy**. It is always desirable to obtain a numerical solution as accurate as possible, although the uncertainties associated with the domain discretisation, numerical operator and the spatial singularity at the source (electric dipole). This leads to the need for a method that offers a good trade-off between accuracy and number of degrees of freedom (DOFs), namely, the key issue is to design an algorithm whose accuracy is determined by the control of those uncertainties.

2. **Tackle realistic problems with efficiency**. Due to the tremendous progress of scientific computing geophysical imaging tackles more and more realistic models. However, not all numerical schemes are well suited for latest computing architectures or are well adapted to the problem. Furthermore, the actual execution of real-life scale models requires the use of HPC resources. For that reason, an architecture-aware design effort is often beneficial in order to ensure that a new method has capacity for large scale computations, thus competence to deal with real models.

3. **Adaptability, modularity and flexibility**. The adopted algorithms should cope with a variety of characteristic of the models with the possibility to easily add or remove components. The method should also run efficiently on a large variety of computer platforms without having to rewrite large parts of the code. It is also desirable to let to the user a minimum of parameters to tune. This can be satisfied with a correct use of third-party libraries which are usually optimized for the computer architecture being used.

This thesis proposes the use of state-of-art EFEM and cutting-edge HPC technologies to improve simultaneously the three key requirements already mentioned at real-scales.

1.2.1 Discretisation remarks

As principal discretisation techniques for 3D CSEM FM arises the Finite Difference Method (FDM), Finite Element Method (FEM) and Edge Finite Element Method (EFEM). The FDM, despite the disadvantage of not being able to precisely take into account complex geometries of subsurface structures, which in some cases may

significantly damage the quality of a solution, is still the most widely employed discretisation scheme (Koldan, 2013). There are many successful FDM implementations, but the most practical and highly efficient parallel code was developed by (Alumbaugh et al., 1996). However, the main disadvantage of FDM is his incapacity to work with unstructured grids. An example of later issue is an scenario with seabed bathymetry where an imprecise representation produces artefacts in images that can lead to false interpretations (Koldan, 2013). On the other hand, FEM supports completely unstructured meshes as well as mesh refinement, which enables the representation of complex geometries and thus improve the solution accuracy. Despite, FEM is still not as widely applied as FDM and a major obstacle for its broader adoption is that the nodal FEM does not correctly take into account all the physical aspects of the vector field functions. In fact, there are three main problems when nodal-based finite elements, obtained by interpolating the nodal values, are employed to represent vector fields (electric or magnetic). The first one is the occurrence of spurious solutions or non-physical solutions, which is generally attributed to lack of enforcement of the divergence condition. The second one is the inconvenience of imposing boundary conditions at material interfaces as well as at conducting surfaces. Finally, the third problem is the difficulty in treating conducting and dielectric edges and corners due to field singularities associated with these structures. Consequently, most of the researchers who have employed the FEM for 3D CSEM FM have been primarily focused on overcoming this problem, as well as on solving other physical and numerical challenges, in order to obtain a proper and accurate numerical solution, leaving aside the performance of the codes (Koldan, 2013). These drawbacks have encouraged the exploration of other approaches, namely, EFEM. This technique uses so-called vector basis functions that assign DOFs to the edges rather than to the nodes of each element.

This thesis focuses on EFEM as discretisation approach because it is based on unstructured meshes, which offer more convenient mesh adaptivity (refinement) and better fit to complex domains. Furthermore, the use of this meshes is more suitable because they allows place more grid points in areas where the solution error is large, avoiding the necessity to create a fine mesh over the whole domain. Examples of geophysical applications with FEM and EFEM can be found in (Ho-Le, 1988; Badea et al., 2001; Bespalov et al., 2007; Hanif et al., 2011; Koldan, 2013; Puzyrev et al., 2013; Cai et al., 2014; Chung et al., 2014; Koldan et al., 2014; Cai et al., 2017).

1.2.2 Computational remarks

The EFEM computational implementation is very similar to that which uses usual node-based elements. However, the main difference arises in the preparation of the input data. Since in EFEM the unknowns are associated with the edges of elements, new data structures are requiered, namely:

1. Elements connectivity matrix defined by their 3/6 in 2D/3D.

2. Edges matrix defined by their 2 nodes in 2D/3D.

3. Numbering strategy to ensure the tangential continuity of edges.

Since most of the FEM codes were developed for node-based finite elements, it is necessary to develop a small library to build up the previous data structures. Many strategies have been discussed in depth (Owen, 1998; Said et al., 1999; Chrisochoides, 2006; Zhang et al., 2008; Jamin et al., 2014; Knepley et al., 2015). However these approaches can be very time-consuming for meshes with a considerable number of elements, which is a normal requirement in real scenarios of 3D CSEM FM. Therefore, Chapter 2 of this thesis presents a new set of algorithms for EFEM computations.

Because of the extremely heavy 3D CSEM FM requeriments at real-scale, researchers aim at highly optimized implementations running on HPC architectures. Normally, this demands a need for developing tailored, hand-tuned codes in compiled languages: C-like and Fortran. However, there are not openly available and easily accesible ad-hoc codes to the public and the common geophysics researcher, e.g. the design is no user-friendly, and it is both challenging and time-consuming for a user to modify or extend the codes to satisfy their own needs. Latter issue is usually nature of codes written in low-level programming languages (compiled languages).

Over the last two decades the researchers have tended to move from low-level to high-level programming languages like Python, Matlab, R and Julia, among others. The experience is that implementations based on interpreted languages are faster to develop, easier to test and mantain, hardware-independent, and they can reach a much wider audience because the codes are readable and compact. The drawback of interpreted languages has been the decreased computational performance and, in particular, their lack of an efficient support for HPC architectures. However, a lot of progress has been made in theory as well as practice in order to reduce the cost of accessing to parallel environments through interpreted languages. From the long list of this kind of languages, Python is the option that has gained most popularity in the

parallel scientific computing context. In fact, there are already several examples of succesfull use of Python for HPC applications:

1. **FEniCS** (Alnæs et al., 2015). Computational tool for solving partial differential equations written mainly in C++, but most application developers are writting directly in Python. For large scale applications the developed Python solvers are usually equally fast as their C++ counterparts, because most of the computing time is spent within the low-level wrapped C++ functions that perform the costly linear algebra operation (Mortensen and Langtangen, 2016).

2. **Petsc4py** (Dalcín et al., 2011). Open-source, public-domain software project that provides access to the Portable, Extensible Toolkit for Scientific Computation (*PETSc* (Balay et al., 2016)) libraries within the Python programming language. *Petsc4py* is a general-purpose and full-featured package. Its facilities allow sequential and parallel Python applications to exploit state-of-art algorithms and data structures readily available in *PETSc*.

3. **Mpi4py** (Dalcín et al., 2008). Open-source, public-domain software project that provides bindings of the Message Passing Interface (MPI) standard for the Python programming language. Its facilities allow parallel Python programs to easily exploit multiple processors.

4. **GPAW** (Mortensen et al., 2005). Code devoted to electronic structure calculations, written as a combination of Python and C.

5. **PyClaw** (Ketcheson et al., 2012). Python-based interface to the algorithms of Clawpack and SharpClaw for wave propagation problems. It also contains the PetClaw package, which adds parallelism through *PETSc*.

6. **SfePy** (Cimrman, 2014). Python code for solving systems of coupled partial differential equations by the FEM in 1D, 2D and 3D. This code is useful for building custom applications.

7. **pyGIMLi** (Coscia et al., 2011). Open-source multi-method library for solving inverse and forward tasks related to geophysical problems. Written in C++ and Python, it allow users build robust inversion applications in the geophysical context.

Python has potential for providing short and quick implementations to compete with tailored codes in low-level programming languages up to thousands of processors.

However, this fact is not well known in the geophysics context and the number of end-user parallel applications is still limited. Therefore, this thesis reports a novel HPC python implementation for 3D CSEM FM in geophysics aimed at a wide audience.

1.3 Summary and thesis objectives

The 3D CSEM FM is well established and widely used in industry and academy to define and characterize bodies by its electric resistivity, which help us to conduct exploration campaigns with a significant reduction of costs and risks (Osseyran and Giles, 2015). On the other hand, simulation and modelling tools help us to formalize and simplify the complexity we observe in nature. This simplification together to HPC advances allow us to render natural phenomena treatable and testable.

Although some contributions have been made to the development of algorithms and tools for 3D CSEM FM, such as parallel codes developed by (Alumbaugh et al., 1996) and (Koldan, 2013), the knowledge on this subject is still limited in the context of interpreted languages, with plenty of room for improvement. As main example, most codes that meet the requirements at real-scale modelling are based on compiled languages, are commercial and often inaccessible to the wider scientific community, aspects that can all hamper advancements in the field. Furthermore, these codes are not user-friendly which reduce the potential for a user to modify or extend them to satisfy their own needs. Therefore, this thesis is focused in the development of an approach that efficiently deals with this scenario.

The main goal is develop and document a new open-source modelling tool for 3D CSEM FM in geophysics using HPC architectures and Python as programming language, namely, Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (*PETGEM*). This thesis considers Python as a glue language for interconnecting different modules of codes written in compiled languages. By exploiting this methodology, complex scientific modelling codes can take advantage of the best attributes of both worlds: the efficient high-level data structures and a simple but effective approach to object-oriented programming of Python, and the well-know efficiency of compiled languages for numerically intensive computations. The code must tackle realistic problems with accuracy, efficiency and flexibility. For this purpose, the following particular objectives are considered:

1. **Efficient implementation of 3D CSEM FM on HPC**. In literature few related works can be found dealing with codes for the numerical solution of 3D CSEM FM using EFEM. Most of them are based on FEM or FDM and written

with compiled languages, hence the use of interpreted languages in this context is a plenty room to investigate and improve. The first part of Chapter 2 is dedicated to the numerical formulation of EFEM for 3D CSEM FM and its implementation on state-of-art HPC architectures using Python programming techniques. A Matlab prototype for the solution of the problem under consideration is described in Appendix C. The algorithms developed seek to exploit the flexibility of the numerical method. Here, the HPC code is validated against the quasi analytical results of canonical models. In all cases the numerical solutions obtained were found in good agreement with reference models.

2. **Performance evaluation of the code**. Since some portions of the code are interpreted and because there is some calling overhead for Python functions, the performance is traditionally deteriorated. The second part of Chapter 2 is devoted to the presentation of the code performance analysis. This is based on strong scalability studies and hardware counters analysis for the models of previous particular objective.

3. **3D CSEM FM at real-scale**. An objetive of this thesis is the 3D CSEM FM at real-life scale which requires the inclusion of real geometries, such as bathymetry, as well as a considerable number of DOFs. Since these models requires using HPC resources, an architecture-aware design effort is desirable in order to ensure that a new method has capacity for large scale computations. Chapter 3 presents a set of 3D CSEM FM models at real-scale. According to the results, flexibility, ease and accuracy of *PETGEM* makes it a competitive and attractive tool to simulate real scenarios of 3D CSEM FM in geophysics.

Chapter 2

HPC python code for 3D CSEM FM

Nowadays, the electromagnetic methods are an established tool in geophysics, with application in many areas such as hydrocarbon and mineral exploration, reservoir monitoring, CO_2 storage characterization, geothermal reservoir imaging and many others. In particular, the 3D CSEM FM has become an important technique for reducing ambiguities in data interpretation for hydrocarbon exploration. In order to be able to predict the electromagnetic signature of a given geological structure, modelling tools provide us with synthetic results which we can then compare to real data. Additionally, in the multi-core and many-core era, parallelization is a crucial issue. Edge finite element method (EFEM) offer good scalability potential. Its low degrees of freedom (DOFs) number after primary/secondary field decomposition make them potentially fast, which is crucial in the future goal of solving inverse problems which might involve over 100,000 realizations (Osseyran and Giles, 2015).

As consequence, in past 2 decades the modelling tools have become one of the pillars for the simulation of numerical methods which main goal is elucidating the fundamental mechanisms behind simplified abstractions of complex phenomena in different areas. The 3D CSEM FM in geophysics is no exception and the scientific community has developed important contributions in this field. In this regard some examples of modelling tools for geophysical prospecting are (Mackie et al., 1994; Alumbaugh et al., 1996; Xiong et al., 2000; Zyserman and Santos, 2000; Badea et al., 2001; MacGregor et al., 2001; Dogru et al., 2002; Fomenko and Mogi, 2002; Newman and Alumbaugh, 2002; Collins et al., 2003; Davydycheva et al., 2003; Cao et al., 2005; Key and Weiss, 2006; Weiss and Constable, 2006; Franke et al., 2007; Kong et al., 2007; Li and Key, 2007; Li and Constable, 2007; Operto et al., 2007; Abubakar et al., 2008; Davydycheva

and Rykhlinski, 2011; Li and Dai, 2011; Puzyrev et al., 2013; Koldan, 2013; Koldan et al., 2014). However, the tools that full fit needs for the solution of real models are commercial and often are inaccessible to the wider scientific community. Due to the discretization method employed, not all codes that are affordable to community are capable of dealing with complex geometries such as models including bathymetries. Additionally there are few parallel codes that are efficient, scalable and can deliver good performance.

On top of that, this thesis is focused in the development and documentation of a new open-source modelling code for 3D CSEM FM in geophysics using interpreted languages and its parallel and vectorized techniques on HPC platforms. An advantage of interpreted languages over compiled languages lies in the fact that it is much easier to make changes and test those modifications in a rapid way. On the other hand, the most mentioned disadvantage of interpreted languages, compared to compiled languages, is the performance (Dalcín et al., 2011). However, since computer hardware is increasing in speed rapidly, the language performance factor is less and less critical. Furthermore, for most scientific computing applications the time-critical portion of the code that requires the efficiency of a compiled language, is confined to small set of functions. Implementing the remaining part of the code using an interpreted language has many advantages without a considerable performance degradation.

Within the variety of interpreted languages, we have decided to use Python 3.x to develop a Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (*PETGEM*). We chose Python 3.x language not only because is an highly flexible and open source language but also because is the easiest and natural way to migrate the Matlab prototype that we developed in Appendix C. In addition, Python 3.x offers numerous third-party modules that make possible a rapid development.

Among others, *PETGEM* aims to solve a relative scarcity of robust edge-based codes to simulate these problems on HPC architectures. *PETGEM* is implemented in current state-of-art platforms such as Intel Xeon Platinum, Intel Haswell and Intel Xeon Phi processors, which offer high-performance and flexibility. Nevertheless, *PETGEM* support older architectures such as SandyBridge, for the sake of usability and to be able to compare performance.

In this Chapter, the numerical formulation for the modelling of 3D CSEM using EFEM is presented. In addition, the algorithmic implementation and *PETGEM* features are surveyed. Furthermore, parallel strategies in *PETGEM* are described in depth. At the end of the Chapter we presented a *PETGEM* performance analysis on a set of bechmarks.

2.1 EFEM formulation for 3D CSEM FM

Electromagnetic (EM) problems that arise in geophysics when using 3D CSEM FM generally deal with a resultant EM field wich appears as response of the electrically conductive Earth to an impressed (primary) field generated by a source (Koldan, 2013). The primary field gives rise to a secondary field, and the resultant field is the sum of both fields.

3D CSEM FM involves the numerical solution of Maxwell's equations in stationary regimes for heterogeneous anisotropic electrically conductive domains in order to compute the components of the EM field. A broad description of these fundamental equations is included in Appendix A. 3D CSEM FM generally works with low frequency EM fields (0.1 *Hz* to 3 *Hz*) because the electric conductivity of the geological structures is much larger than their dielectric permittivity (Koldan, 2013). In consequence, Maxwell's equations are simplify and reduce to their diffusive form (Zhdanov, 2009; Koldan, 2013; Cai et al., 2014)

$$\boldsymbol{\nabla} \times \mathbf{E} = i\omega\mu_0\mathbf{H}, \tag{2.1}$$

$$\boldsymbol{\nabla} \times \mathbf{H} = \mathbf{J}_s + \tilde{\sigma}\mathbf{E}, \tag{2.2}$$

where we considered the harmonic time dependence $e^{-i\omega t}$, ω is the angular frequency, μ_0 is the free space magnetic permeability, $\mathbf{J}_s$ is the distribution of source current, $\tilde{\sigma}$ is the electric conductivity and $\mathbf{E}$ is the induced current in the conductive Earth. In isotropic domains, $\tilde{\sigma}$ is an scalar value that is function of all three spatial coordinates. On the other hand, in anisotropic domains, $\tilde{\sigma}$ is a 3×3 tensor defined as

$$\tilde{\sigma} = \begin{bmatrix} \sigma_x & 0 & 0 \\ 0 & \sigma_y & 0 \\ 0 & 0 & \sigma_z \end{bmatrix}, \tag{2.3}$$

where σ_x, σ_y and σ_z are function of the spatial coordinates.

In 3D CSEM FM, the most used formulations for the field $\mathbf{E}$ are those based on total field or primary/secondary field (electric field decomposition). In a total field formulation a slightly larger computational domain is required in order to discretize the source properly. Moreover, the rapid change of the primary current demands a dense mesh refinement near to the source. To overcome these difficulties, many authors use the primary/secondary field formulation, where the primary field corresponds to the solution in a layered earth. This formulation is desirable because the primary

field is much smoother than the source current, avoiding singularities near to the source and limiting the size of the computational domain. Therefore we have used a primary/secondary field approach.

Following the formulation by (Zhdanov, 2009; Cai et al., 2014), we decomposed the total electric field ($\mathbf{E}_t$) into primary field ($\mathbf{E}_p$) and secondary field ($\mathbf{E}_s$) as

$$\mathbf{E}_t = \mathbf{E}_p + \mathbf{E}_s, \tag{2.4}$$

$$\sigma = \sigma_s + \Delta\sigma. \tag{2.5}$$

Based on this decomposition, we derive the following expression for the secondary field

$$\boldsymbol{\nabla} \times \boldsymbol{\nabla} \times \mathbf{E}_s + i\omega\mu\sigma\mathbf{E}_s = -i\omega\mu\Delta\sigma\mathbf{E}_p, \tag{2.6}$$

where the source term is $\mathbf{E}_p$. For a general layered earth model, $\mathbf{E}_p$ can be computed semi-analytically by using Hankel transform filters. Therefore, the source term is given by a dipole able to work in each spatial coordinate (Nabighian, 1988), namely, a x-directed dipole is defined as

$$E_x = \Upsilon \cdot \left[\frac{d_x^2}{r^2}\right] \cdot \Psi + k^2 r^2 - ikr - 1, \tag{2.7a}$$

$$E_y = \Upsilon \cdot \left[\frac{d_x \cdot d_y}{r^2}\right] \cdot \Psi, \tag{2.7b}$$

$$E_z = \Upsilon \cdot \left[\frac{d_x \cdot d_z}{r^2}\right] \cdot \Psi, \tag{2.7c}$$

a y-directed dipole is defined as

$$E_x = \Upsilon \cdot \left[\frac{d_x \cdot d_y}{r^2}\right] \cdot \Psi, \tag{2.8a}$$

$$E_y = \Upsilon \cdot \left[\frac{d_y^2}{r^2}\right] \cdot \Psi + k^2 r^2 - ikr - 1, \tag{2.8b}$$

$$E_z = \Upsilon \cdot \left[\frac{d_y \cdot d_z}{r^2}\right] \cdot \Psi, \tag{2.8c}$$

and finally, a z-directed dipole is determined by

$$E_x = \Upsilon \cdot \left[\frac{d_x \cdot d_z}{r^2}\right] \cdot \Psi, \tag{2.9a}$$

$$E_y = \Upsilon \cdot \left[\frac{d_z \cdot d_y}{r^2}\right] \cdot \Psi, \tag{2.9b}$$

$$E_z = \Upsilon \cdot \left[\frac{d_z^2}{r^2}\right] \cdot \Psi + k^2 r^2 - ikr - 1, \tag{2.9c}$$

with Υ and Ψ defined as

$$\Upsilon = \frac{I \cdot dS}{4\pi\sigma r^3} \cdot e^{-ikr},$$
$$\Psi = -k^2 \cdot r^2 + 3ikr + 3,$$

where I is the dipole current, dS is the dipole length, σ is the background conductivity, k is the propagation parameter (wavenumber), r is the distance between source position and the evaluation point position, and (d_x, d_y, d_z) are the components of the vector connecting source position and evaluation point position.

Equation (2.6) can be solved by using integral equation method (Chew et al., 2008; Kythe and Puri, 2011), FDM (Newman and Alumbaugh, 2002; Newman et al., 2010), FEM (Key and Weiss, 2006; Key and Ovall, 2011) or EFEM (Cai et al., 2014; Chung et al., 2014; Cai et al., 2017). As already said, our formulation is based on EFEM which uses vector basis functions defined on the edges of the corresponding tetrahedral elements. Therefore, if the tangential components of the electric field $\mathbf{E}$ are assigned to each edge, the components of $\mathbf{E}$ inside the tetrahedral elements can be expressed as

$$E_x^e = \sum_{i=1}^{6} N_{xi}^e E_{xi}^e, \qquad E_y^e = \sum_{i=1}^{6} N_{yi}^e E_{yi}^e, \qquad E_z^e = \sum_{i=1}^{6} N_{zi}^e E_{zi}^e, \tag{2.10}$$

where $\mathbf{N}_i^e$ are the vector edge basis functions defined by (Jin, 2002). Despite its wide use, the literature about the computation of these basis is not easy to perveice. Therefore, in Appendix B.2 we included a detailed description in order to fill this relative scarcity. The compact form of these basis functions is defined as

$$\mathbf{E}^e = \sum_{i=1}^{6} \mathbf{N}_i^e \mathbf{E}_i^e. \tag{2.11}$$

Vector edge basis functions are divergence free but not curl free and continuous at the

element boundaries. Taking into account the EFEM background of Appendix B.2, and by substituting expression (2.11) into (2.6), and using Galerkin's method, the weak form of the original differential equation is given by

$$Q_i = \int_\Omega \mathbf{N}_i \cdot [\nabla \times \nabla \times \mathbf{E}_s - i\omega\mu\tilde{\sigma}\mathbf{E}_s + i\omega\mu\Delta\tilde{\sigma}\mathbf{E}_p]dV. \tag{2.12}$$

The discretized form of (2.12) for each tetrahedral element in the domain is obtained after applying the Green's theorem

$$Q_i = \int_\Omega \mathbf{N}_i \cdot [\nabla \times \nabla \times \mathbf{E}_s - i\omega\mu\tilde{\sigma}\mathbf{E}_s + i\omega\mu\Delta\tilde{\sigma}\mathbf{E}_p]dV, \tag{2.13}$$

where K^e and M^e are the elemental stiffness and mass matrices defined as

$$K^e_{ij} = \iiint_{V^e} (\boldsymbol{\nabla} \times \mathbf{N}^e_i S^e_i) \cdot (\boldsymbol{\nabla} \times \mathbf{N}^e_j S^e_j)\, dV, \tag{2.14}$$

$$M^e_{ij} = \iiint_{V^e} (\mathbf{N}^e_i S^e_i) \cdot (\mathbf{N}^e_j S^e_j)\, dV, \tag{2.15}$$

where S^e_i are coefficients equal 1 or -1 depending on the local and global direction of the *i-th* edge in the element. Expression (2.13) can be written compactly as $[A]\cdot\{\phi\} = \{b\}$ where matrix $[A]$ is assembled from elemental matrices K^e and M^e, and similarly, the vector $\{b\}$ from elemental vectors $\{b^e\}$, namely

$$[K^e_{jk} + i\omega\tilde{\sigma}_e M^e_{jk}] \cdot \{E_{sk}\} = -i\omega\mu\Delta\tilde{\sigma}_e R^e_k, \tag{2.16}$$

where R^e_k is the right hand side which requires numerical integration.

Proper boundary conditions need to be added in order to obtain a unique solution for system (2.16). For this purpose, we considered the homogeneous dirichlet boundary conditions in the EFEM formulation because these holds approximately for the secondary $\mathbf{E}$ at a distance from the domain with the anomalous conductivity (Jin, 2002)

$$e\,|_{\partial\Omega} = 0. \tag{2.17}$$

For the numerical simulation, the distance, where boundary conditions (2.17) hold, can be determined based on the skin depth of $\mathbf{E}$. Fundamental references about that are (Plessix et al., 2007; da Silva et al., 2012; Puzyrev et al., 2013).

2.2 Field interpolation with EFEM

To compute the $\mathbf{E}_s$ in interested points, such as receivers position, is necessary to perform an interpolation function. Based on the approach by (Jin, 2002; Zhdanov, 2009; Cai et al., 2014) and by using the vector basis functions (2.11), the $\mathbf{E}_s$ for a point r contained in element e, can be obtained by

$$E_s^r = \int \mathbf{N}_i^e \cdot \mathbf{E}_{si}^e \cdot S_i^e \qquad i = 1 \dots 6, \tag{2.18}$$

where $\mathbf{E}_{si}^e$ is the secondary field assigned to the *i-th* edge of element e and S_i^e are coefficients equal 1 or -1 depending on the relative local orientation of the *i-th* edge in the element e with respect to the global orientation of the *i-th* edge in the mesh, fully explained in Subsection 2.3 through expression (2.19).

2.3 Algorithms for EFEM

The implementation of a code based on EFEM is very similar to that which uses usual FEM. The main difference arises in the preparation of the input data, namely a nodal-based FEM standard code requires the data structures included in table 2.1, where N, E and F are the number of nodes, elements and faces respectively. Since

Name	Dimensions	Description
nodes2coord	$2/3 \times N$	Nodes defined by their 2/3 coordinates in 2D/3D
faces2nodes	$3 \times F$	Faces defined by their 3 nodes in 3D
elements2nodes	$3/4 \times E$	Elements defined by their 3/4 nodes in 2D/3D
elements2faces	$4 \times E$	Elements defined by their 4 faces in 3D

Table 2.1 Main FEM data structures for linear triangular/tetrahedral nodal elements.

for the EFEM, the unknowns are associated to element edges, instead of the nodes, one needs a matrix to represent every element by their edges and another matrix to describe every edge by their two nodes, as show table 2.2 where $\mathcal{E}$ is the number of edges. Because most of the FEM codes were developed for nodal-based FEM, it is necessary to develop a set of algorithms to build up the data structures of table 2.2, namely, convert node numbering into edge numbering. Traditional approach consists in go through element by element to check if all the edges of the i-th element have been numbered. If an edge has not been numbered, number it; otherwise, skip it (Jin,

Name	Dimensions	Description
elems2edges	$3/6 \times E$	Elements defined by their 3/6 edges in 2D/3D
edges2nodes	$2 \times \mathcal{E}$	Edges defined by their 2 nodes in 2D/3D

Table 2.2 Main EFEM data structures.

2002). However, former technique can be very time-consuming for meshes with a considerable number of elements, which is a normal feature in real applications not just in the geophysics field, but also in others fields.

Many edge numbering strategies have been discussed in depth (Owen, 1998; Said et al., 1999; Chrisochoides, 2006; Zhang et al., 2008; Jamin et al., 2014; Knepley et al., 2015). This thesis focuses on the use and improvement of two of the most common techniques. In sake of simplicity, both algorithms are applied to a FE mesh in 2D which connectivity is presented in table 2.3. The first strategy start with the computation

e	nodes(1,e)	nodes(2,e)	nodes(3,e)
1	2	4	1
2	2	4	5
3	2	3	5
4	3	5	6
5	4	5	7
6	5	7	8
7	5	6	8
8	6	9	8

Table 2.3 Element to nodes connectivity array in 2D.

of an edge *id*, which is given by simple operations such as $(i_1 \times i_2)$, $(i_1/i_2 + i_2/i_1)$ or $(i_1 \log i_2 + i_2 \log i_1)$, where i_1 and i_2 are the end-points of j-th edge. Its operation is applied to all elements in a nodal-based FE connectivity to list all edges in an array. Its array, should list the smaller nodal number first and should store the number of element. The ascending order of the end-points for all edges determines their global direction in the mesh, which is a critical aspect in the representation of vector fields such as the electric field ($\mathbf{E}$) and the magnetic field ($\mathbf{H}$). Once the edge *id* array is computed, it should be rearrange according to the *id*. Therefore, for the nodal-based FE connectivity of table 2.3, the edge *id* array is given by table 2.4.

Next step is number the edges of table 2.4 from top to bottom in order to produce the edge to node/element array, it approach works as follows. If *id* has a new value,

number the associated edge as a new edge. If the id has the same value as preceding one, compare i_1 with i_2 to determine whether this is the same edge as the preceding one. If yes, skip it and set the associated element number behind the preceding edge; if not, number it as a new edge. The resulting array is shown in table 2.5.

Finally, the element to edge connectivity array can be generated by the use of information of table 2.3 and table 2.5. Therefore, the edge connectivity array for this mesh is shown in table 2.6. The efficiency of this approach derives from the efficient sorting algorithm.

The second strategy does not require the use of a sorting algorithm. In this approach, the first step consist in generate a node to elements array such as table 2.7. After that, the element to edge connectivity array is initialized with zeros an set a counter, to be used to assign the edge id, to 1, and the array is filled as follows. Visit first element and examine each of its three edges. If the entry is non-zero, this edge was already numbered, then go to the next edge. If the entry is equal to zero, this is a new edge whose id is defined by the value of the counter. With information of tables 2.3 and 2.7, is possible check if this edge is also shared by other elements, namely, comparing the element numbers for the end-points of the edge. If the edge is also shared by other elements, set the value of the counter to the corresponding edges of these elements. After that, increase the counter by one and proceed to the next edge. Table 2.8 is a new version of element to edge array with edge to node/element array defined by table 2.9.

An important conclusion is that since the numbering of edges is not unique, their resultant arrays are different. However, both approaches can be applied to any nodal-based FE mesh (2D and 3D). Regardless the numbering edges technique, adopting local and global numbering conventions and then using these consistently is absolutely essential. Furthermore, since the basis functions of EFEM are vectors, they have directions in addition to magnitudes. To ensure the tangential continuity and considering that nodal numbering is assigned by the mesher, a unique global edge direction should be defined (Jin, 2002; Davidson, 2010; Rylander et al., 2012). This issue can to some extent be avoided on structured meshes of squares or cubes. For unstructured meshes, however, it is necessary to have efficient and reliable techniques. Therefore, the reference direction or global direction is usually based on the global node numbers at the end-points of the edge under consideration, i.e., the vector basis function $\mathbf{N}_i^e$ of e-th element is directed from the i_1 node to i_2 node when the coefficient for the vector basis function is positive. However, one or several of the vector basis functions on the local elements that share an edge may be defined in the reverse

id	i_1	i_2	e
2	1	2	1
4	1	4	1
6	2	3	3
8	2	4	1
8	2	4	2
10	2	5	2
10	2	5	3
15	3	5	3
15	3	5	4
18	3	6	4
20	4	5	2
20	4	5	5
28	4	7	5
30	5	6	4
30	5	6	7
35	5	7	5
35	5	7	6
40	5	8	6
40	5	8	7
48	6	8	7
48	6	8	8
54	6	9	8
56	7	8	6
72	8	9	8

Table 2.4 Edge id computation and sorting.

edge	i_1	i_2	e
1	1	2	1
2	1	4	1
3	2	3	3
4	2	4	1, 2
5	2	5	2, 3
6	3	5	3, 4
7	3	6	4
8	4	5	2, 5
9	4	7	5
10	5	6	4, 7
11	5	7	5, 6
12	5	8	6, 7
13	6	8	7, 8
14	6	9	8
15	7	8	6
16	8	9	8

Table 2.5 Edge to node/element connectivity - first approach.

e	edge 1	edge 2	edge 3
1	1	2	4
2	4	5	8
3	3	5	6
4	6	7	10
5	8	9	11
6	11	12	15
7	10	12	13
8	13	14	16

Table 2.6 Element to edges connectivity array in 2D - first approach.

node	e
1	1
2	1, 2, 3
3	3, 4
4	1, 2, 5
5	2, 3, 4, 5, 6, 7
6	4, 7, 8
7	5, 6
8	6, 7, 8
9	8

Table 2.7 Node to elements connectivity array in 2D - first approach.

e	edge 1	edge 2	edge 3
1	1	2	3
2	1	4	5
3	6	7	5
4	7	8	9
5	4	10	11
6	10	12	13
7	8	14	13
8	15	16	14

Table 2.8 Element to edges connectivity array in 2D - second approach.

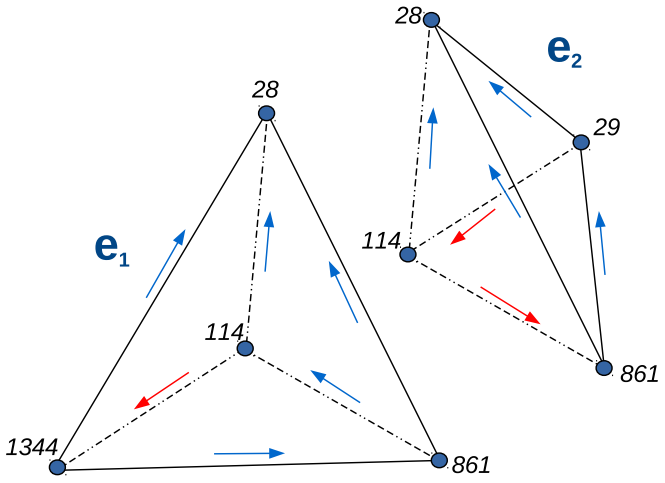


Fig. 2.1 Global and local edge direction of two elements sharing three edges. Blue arrow depicts edges whose local direction is inverse to global direction.

edge	i_1	i_2	e
1	2	4	1, 2
2	1	4	1
3	1	2	1
4	4	5	2, 5
5	2	5	2, 3
6	2	3	3
7	3	5	3, 4
8	5	6	4, 7
9	3	6	4
10	5	7	5, 6
11	4	7	5
12	7	8	6
13	5	8	6, 7
14	6	8	7, 8
15	6	9	8
16	8	9	8

Table 2.9 Edge to node/element connectivity - second approach.

direction. One way to deal with this problem is to multiply all local vector basis functions with reverser direction by -1, i.e., the local vector basis function $\mathbf{N}_i^e$ relates to the global vector basis function as $\mathbf{N}_i^e = -\mathbf{N}_i^e$.

Previous strategy can be summarize as follows. If an edge adjoins two nodes n_i and n_j, the direction of the edge as going from node n_i to node n_j if $i < j$. This simple algorithm gives a unique orientation of each edge in the mesh. On the other hand, the local orientation of edges within each element can be determined by his nodes indexes. For instance, the local edge direction for the elements in figure 2.1 with nodal and edge connectivity defined by table 2.10 and table 2.11 respectively, is given by the vectorial function

$$S_i^e = \frac{node_{i2}^e - node_{i1}^e}{|node_{i2}^e - node_{i1}^e|} \qquad i = 1 \dots 6, \tag{2.19}$$

where i is the edge index within e-th element that adjoins $node_{i1}^e$ with $node_{i2}^e$. The main advantage of function (2.19) is that it allow work with node numbering based on a clockwise or counter-clockwise in order to meet some conditions of FEM formulations such as element's volume computation, which must be positive in any case. Therefore, the relation between the local direction and global direction of edges in figure 2.1 is given by table 2.12, which means that vector basis functions $\mathbf{N}_4^1, \mathbf{N}_5^1, \mathbf{N}_5^2$ must be

e	nodes(1,e)	nodes(2,e)	nodes(3,e)	nodes(4,e)
e_1	28	114	861	1344
e_2	28	114	29	861

Table 2.10 Nodal connectivity for local/global edge direction.

e	edge 1	edge 2	edge 3	edge 4	edge 5	edge 6
e_1	158	157	160	165	708	167
e_2	158	160	162	708	709	5780

Table 2.11 Elements to edges connectivity for local/global edge direction.

reversed. On the other hand, discretisation with EFEM based on tetrahedrons yields to

e	edge 1	edge 2	edge 3	edge 4	edge 5	edge 6
e_1	1	1	1	1	-1	1
e_2	1	1	1	-1	-1	1

Table 2.12 Elements to edges connectivity for local/global edge direction.

more unknowns (about twice) than when using the nodal-based FEM on tetrahedrons. However, the higher number of unknowns is balanced by lower connectivity between edges or a greater sparsity of the nodal-based FEM matrices. As result, the memory demand for both kind of methods is about the same if only the nonzero entries are counted (Jin, 2002; Mukherjee and Everett, 2011; Cai et al., 2014).

2.4 PETGEM

The Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (*PETGEM*) is a Python code for the scalable solution of 3D CSEM FM on tetrahedral meshes, as these are the easiest to scale-up to very large domains or arbitrary shape. It is written mostly in Python 3.5.2 and relies on the scientific Python software stack with heavy use of *mpi4py* and *petsc4py* packages for parallel computations. Other scientific Python packages used include: *H5py* for binary data format support, *Numpy* for efficient array manipulation and *Scipy* algorithms. *PETGEM* allow users the simulation of electromagnetic fields in real 3D CSEM FM on shared-memory/distributed-memory

HPC platforms. Among others, the key drivers for the *PETGEM* development are the following:

1. Solve a relative scarcity of robust edge-based codes for 3D CSEM FM to reduce ambiguities in data interpretation for hydrocarbon exploration.
2. Provide synthetic results which can then compare to real data.
3. Simulate real scenarios, i.e. support for geologies structurally complex with a good trade-off between accuracy and number of DOFs.
4. The integration of Python, EFEM and geophysical methods such as 3D CSEM FM is still limited, with pently room for improvement. These concepts has been systematically applied in *PETGEM* for running simulations using HPC resources.

Although there are specialised modelling tools for geophysical prospecting, details of their implemented methods are generally hidden behind a black box, which could lead to a situation in which the formulation could be unknown. Furthermore, not all numerical schemes are well suited for latest computing architectures or are well adapted to the problem. *PETGEM* is developed as open-source at Computer Applications in Science & Engineering (CASE) of the Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS). Many features have gradually been included, such as modules for EFEM data structures and a set of Python wrappers for the use of efficient solvers and preconditioners suitable for the resulting matrix system. *PETGEM* is now a complete package particularly suited for the 3D CSEM FM aiming to foster our understanding about EM in geophysics and its coupling with HPC technologies. Since it was intended tackle realistic problems, its data structures were designed to cope simultaneously three key requirements: accuracy, flexibility and efficiency. In addition, the adopted algorithms has the posibility to easily add or remove components without having to rewrite large parts of the code. This approach leads to optimal performance in terms of development and computation time. In other words, *PETGEM* was written based on an architecture-aware design effort in order to ensure a good capacity for large scale computations, thus competence to deal with real models without losing versatility offered by the programming language.

2.4.1 Code workflow

The 3D CSEM FM is composed of four main tasks: discretisation of the geometry, elemental matrices computation and global system assembly, solving the resulting

system and post-processing the solution (for a general purpose case, these stages are described in Appendices B and C). In addition, the problem decomposition into independent modules is important because each region make use of methods that belong to different branches of mathematics. For instance, the design of algorithms for 3D meshing and iterative solvers for large scale modelling require knowledge going beyond the scope of this research. Therefore, this thesis rely on well-known tools for domain discretisation and solving systems of linear equations and focuses on the kernel of 3D CSEM FM, namely, the core of *PETGEM*.

Mesh formats supported are those generated by *Gambit*, *Netgen* and *Gmsh*. It task runs independently of main *PETGEM* workflow. The *PETGEM* kernel is based on a relatively straightforward high level parallelization, that makes use of the code capability to integrate the best attributes of compiled and interpreted languages. Several processes of the kernel can be spawned, each responsible for its own subdomain, so that the whole domain is covered. Each process then assembles its local contributions to the global linear system that is solved. For this purpose, *PETGEM* use the *PETSc* library and its large collection of data structures and parallel iterative solvers and preconditioners, that can be used in Python through the *petsc4py* and *mpi4py* packages.

The *PETGEM* modularity policy is preserving because discretisation tools and the *PETSc* library are coupled in such a way that they can be replaced easily by other packages at the cost of minimal changes. An outline of the overall *PETGEM* workflow is depicted in figure 2.2.

2.4.2 Software stack overview

PETGEM use a code structure for the EFEM that emphasizes good parallel scalability, which is crucial in the multi-core era. Furthermore, it's modularity should simplify the process of reaching the best possible performance in terms of percentage of the peak amount of floating point operations provided by the architecture.

An outline of the primary groups of modules in *PETGEM* design is given in figure 2.3. A more detailed explanation is the following:

1. Modular and extensible EFEM kernel. *PETGEM* kernel is extensible in any direction. Therefore, the possibility of adding new features such as new boundary conditions, numerical algorithms, analysis modules, among others.

2. Independent of problem formulation, numerical solution, and data storage. *PETGEM* kernel provides the independent abstractions for 3D CSEM FM, numerical

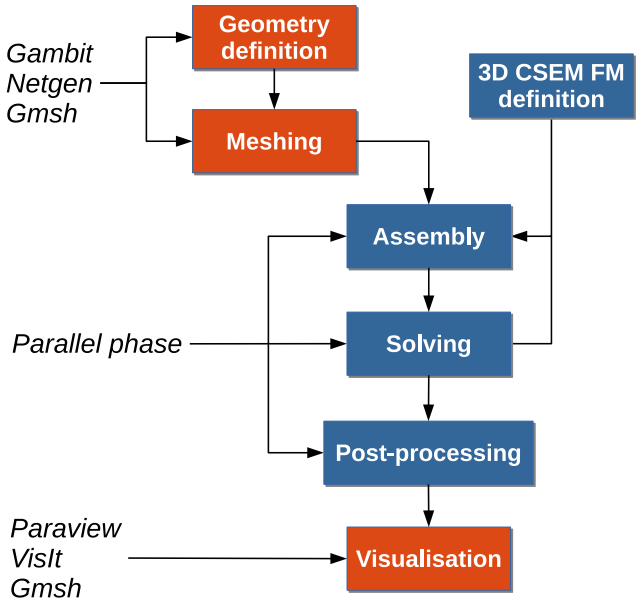


Fig. 2.2 Outline of the overall *PETGEM* workflow. On shared-memory architectures, the parallel phase is based on the Multiprocessing package. On the other hand, the *petsc4py* and *mpi4py* packages are used to provide parallel support on distributed-memory architectures.

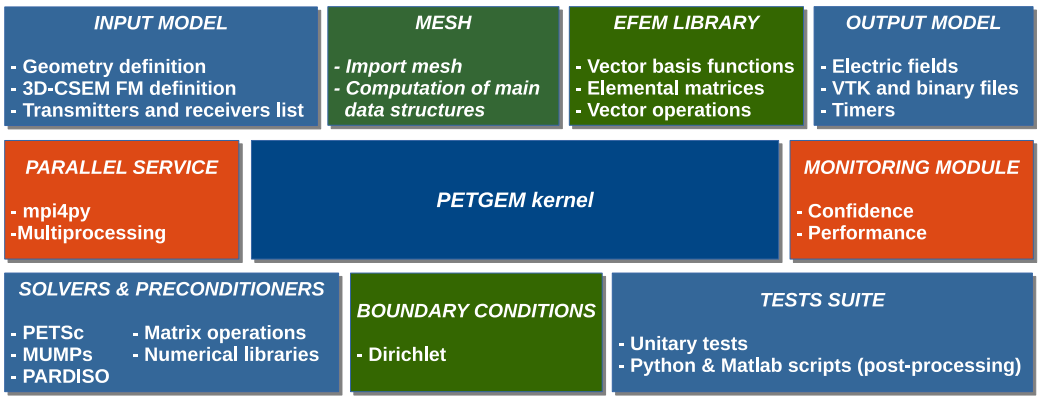


Fig. 2.3 Upper view of *PETGEM* software stack.

methods, data storage and analysis.

3. Parallel processing support. Based on an shared-memory parallelism (Multiprocessing package), distributed-memory parallelism (*mpi4py* and *petsc4py*) and static load balancing.

4. Confidence and performance monitoring. Based on an intensive error checking module and an automatic profiling module.

5. Efficient solvers & preconditioners. Direct as well as iterative solvers and preconditioners are supported through *petsc4py* package.

6. Interface to mesh generators. Not dependent on a specific mesh generator. Because most of the FEM codes were developed for nodal-based formulations, it is necessary to develop a module to compute edge-based data structures. As a result, different mesh formats are supported (*Gambit*, *Netgen*, and *Gmsh*).

7. EFEM library. Edge-based discretisations, vector basis functions, their geometry description, and generalized integration rules provides a generic support for implementing EFEM solution algorithms.

8. Linear systems library. Support to Compressed Row Storage (CSR) format and other formats for sparse matrices and their easy and efficient parallel assembly on shared/distributed-memory platforms.

9. 3D CSEM FM module. Ad-hoc design to meet specific requirements to simulate 3D CSEM FM surveys, namely, conductivity model, physical parameters, transmitter and receiver lists.

10. Test suite. Sample output for many examples are included. Furthermore, a set of matlab and Python functions to data analysis are included.

11. Post-processing. Export to binary files, HDF5 and VTK format are supported, allowing the analysis not just in different visualization tools but also on different platforms. It also gives timing values in order to evaluate the performance.

2.4.3 Programming language

PETGEM is written mostly in Python programming language because:

1. It is open source, cross-platform and functional on a wide number of platforms, including HPC environments.

2. It uses a high level and very expressive language.

3. It uses a good body of bindings to common tools needed in scientific computing such as plotting, numerical libraries, debugging and testing.

The code structure is modular, simple and flexible which allows exploiting not just *PETGEM* modules but also third party libraries. Therefore, the software stack includes interfaces to external suites of data structures and libraries that contain most of the necessary building blocks needed for programming large scale numerical applications, i.e. meshing, sparse matrices, vectors, iterative and direct solvers. As result, the code is compact and eliminates the need to write such libraries and thus speeds up development time by orders of magnitude.

2.4.4 Target architectures

The HPC goal of *PETGEM* involves using cutting-edge architectures. To that goal, the code is implemented in current state-of-the-art platforms such as Intel Xeon Platinum, Intel Haswell and Intel Xeon Phi processors, which offer high-performance, flexibility and power efficiency. Nevertheless, *PETGEM* support older architectures such as SandyBridge, for the sake of usability and to be able to compare performance.

2.4.5 Requeriments

Requirements packages for using *PETGEM*:

1. *Gambit*, *Netgen* or *Gmsh* for mesh generation.

2. Python3.

3. *Scipy* for numerical operations.

4. *Numpy* for arrays manipulation.

5. *H5py* for HDF5 files manipulation.

6. Python Multiprocessing package for parallel computations on shared-memory platforms.

7. Sharedmem Python package for arrays manipulation on shared-memory platforms.

8. *Petsc4py* and *mpi4py* for parallel computations on distributed memory platforms.

9. Paraview or VisIt for visualization of *PETGEM* output files.

10. *Sphinx* and LaTeX to (re)generate code documentation.

PETGEM can be used without any installation by running the kernel from the top-level directory of the distribution.

2.4.6 Coding style

PETGEM coding style is based on PEP-0008 guidelines. Main guidelines are the following:

1. 79 characteres per line.

2. 4 spaces per indentation level.

3. Variables are lower case meanwhile constants are upper case.

4. Comments convention for functions is as follows:
   def function(arg1, arg2):
   '''This is a function.
   :param int arg1: array of dimensions ...
   :param str arg2: string that ...
   '''

5. The use of inline comments is sparingly.

6. Use of lowercase to name functions. Furthermore, functions names have following form: <action>_<subject>(), e.g. compute_matrix().

7. Use of whole words instead abbreviations, examples:
   Yes: solve_system(), compute_edges(), compute_matrix().
   No: solve(), compedges(), compmatrix().

2.4.7 Python 3.x compatibility

Since first Python release each new version had always been backward compatibility. This rule had been broken with the advent of Python 3.x. It does it on purpose, so new set of features can be implemented. In order to ensure forward compatibility, this

thesis is based on Python 3.5.2 which is the latest version at the moment of writting. However, all theoretical concepts apply equally well also to Python 2.x.

On the other hand, and following the studies by (Cuni, 2010), *PETGEM* has been developed under an awareness approach in three areas: language programming, physical-numerical requirements and the computational architecture.

2.4.8 Code availability

The code of *PETGEM* is available at *http://petgem.bsc.es* or by requesting the author (*octavio.castillo@bsc.es*, *ocastilloreyes@gmail.com*). The code is supplied in a manner to ease the immediate execution under Linux platforms. User's manual and technical documentation (developer's guide) are provided in the *PETGEM* archive as well.

2.5 Parallel strategies

In FEM or EFEM simulations, the most time-consuming tasks are the assembly and solving. Hence, the need for efficient algorithms may be crucial especially when the DOFs is considerably large. For the solution of the system of equations there are parallel libraries that have shown their stability and flexibility, e.g. *PETSc*, *MUMPS* or *PARDISO*. In real scenarios of 3D CSEM FM the solution phase, which asymtotically dominates in large-scale computing, remains a critical portion of the code. Therefore, the *PETGEM* kernel is based on the *PETSc* solvers because these are well integrated in Python through the *petsc4py* package. In this way it is possible to take advantage of the capacity of stable and well-known tools in the field of scientific computing.

On the other hand, the classical assembly in FEM or EFEM programming is based on a loop over the elements as is described by (Zienkiewicz et al., 1977). Different techniques and algorithms for this purpose are presented by (Duff et al., 1986; Chen, 2008; Langtangen and Cai, 2008; Chen, 2011; Hannukainen and Juntunen, 2012; Anjam and Valdman, 2015). This characteristic is quite useful because it allows performing these computations at the same time, i.e., to compute them in parallel. However, parallel programming is not a trivial task in most programming languages, and demands a strong theoretical knowledge about the hardware architecture. Fortunately, Python presents two friendly solutions for parallel computations, namely, Multiprocessing and *mpi4py*. Both schemes are described in the following lines.

2.5.1 Parallelism on shared-memory platforms

The Multiprocessing package focuses on shared-memory platforms and similarly to Python threading module it allows the programmer to access the various processors in the platform using sub-processes instead of threads. This module allows you write parallelized codes using processes in relatively simple way. By leveraging system processes instead of threads, Multiprocessing package lets you avoid issues like the Global Interpreter Lock (GIL), which is a mutex that prevents multiple native threads from executing Python bytecodes at once. GIL is necessary mainly because CPython's memory management is not thread-safe, therefore CPython extensions must be GIL-aware in order to avoid defeating threads. As result, Multiprocessing package takes advantage of multiple cores (CPUs) by using child interruptible processes.

On top of that, parallelism on shared-memory platforms in *PETGEM* is based on Multiprocessing package since the assembly phase is a embarrasingly parallel task which does not require an explicit synchronization or communication between those related computational processes.

For the parallel assembly of matrix A, *PETGEM* is based on the CSR function $sparse(I_g, J_g, K_g, m, m)$ which returns a $m \times m$ sparse matrix where vectors I_g, J_g and K_g have the same length given by $m^2 \times$ number of elements, where m is the element order. The zero elements of K are not taken into account and the elements of K_g having the same indices in I_g and J_g are summed. Therefore, the main idea consist in the parallel popullation of the three global 1d-arrays I_g, J_g and K_g which store the position of their elements in the global matrix as well as the local matrices as show in figure 2.4. Using I_k, J_k and K_k and a parallel loop over elements, one may calculate the global arrays I_g, J_g and K_g. In order to avoid unnecessary data copies, we build the global arrays I_g, J_g and K_g using a shared-memory approach through the sharemem Python package (Feng, 2016). Hence, the global indices in which the computed terms must be added by each process are given by $idDominio \times Chunksize + idLocal$, where $idDominio$ is the id process in the parallel pool, $Chunksize$ is the number of elements to be computed within $idDominio$ and $idLocal$ is the local indice of the element computed within $idDominio$. In sake of an adequate workload balancing, the $Chunksize$ value is obtained by dividing the total number of elements in the mesh by the number of CPUs in the parallel pool. On top of that, figure 2.5 depicts a global view of the parallel scheme adopted for assembly in *PETGEM* on shared-memory architectures.

Previous strategy is also applied to assembly the right hand side vector. Because arrays I_g and J_g have already been populated, each process must compute local con-

tributions for all elements within his own domain as depict figure 2.4. Once array Q_g have been computed, the final right hand side can be obtained using the CSR sparse function $sparse(I_g, L_g, Q_g, m, 1)$, where I_g store the global row indices associated to the elements in Q_g, L_g is an auxiliar vector with all values equal to 0 (which define a column matrix) and Q_g store the coefficients of the elemental vectors.

After global assembly, the solution is computed in parallel through *Scipy* solvers or MKL library by Intel. The post-processing stage is done following the numerical scheme described in Section 2.2. As main features of this version arises the simplicity, flexibility and an acceptable efficiency. However, due to memory needs, this version of the code is limited to the solution of small 3D CSEM FM models.

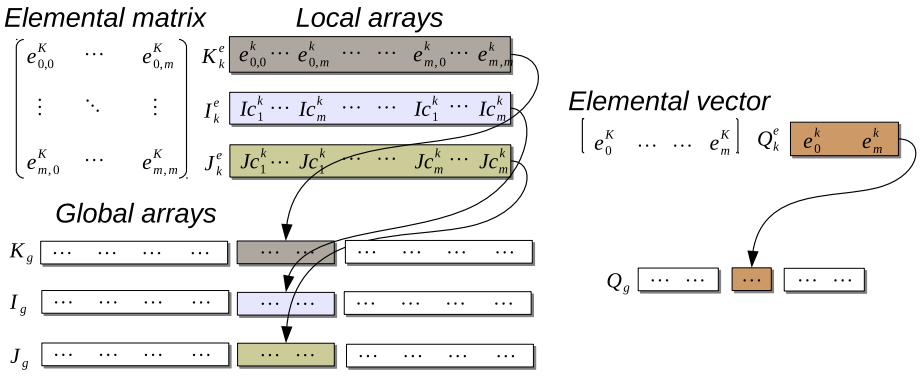


Fig. 2.4 Popullation of arrays associated with the global matrix assembly in *PETGEM* using shared-memory architectures. Here, K_k^e store coefficients of the elemental matrix in a row-wise format, I_k^e store the global row indices associated to the elements in K_k^e and J_k^e contain the global column indices associated to the elements in K_k^e. On the other hand, Q_k^e store coefficients of the elemental vector to populate the global vector or right hand side. Here the element order is represented by m.

2.5.2 Parallelism on distributed-memory platforms

The solution of 3D CSEM FM at real-scale requires greater computing resources than that provided by a shared-memory code whose scalability could limit the potential of the modelling tool. To overcome this situation, *PETGEM* offers support for distributed-memory architectures through the Message Passing Interface (MPI).

MPI is a standarized, portable message-passing system that defines a set of library routines and allows users to write portable codes in the main programming languages (C-like and Fortran). MPI defines a high-level abstraction for fast and portable process communication which is especially suited for distributed memory platforms (Dalcín et al., 2011).

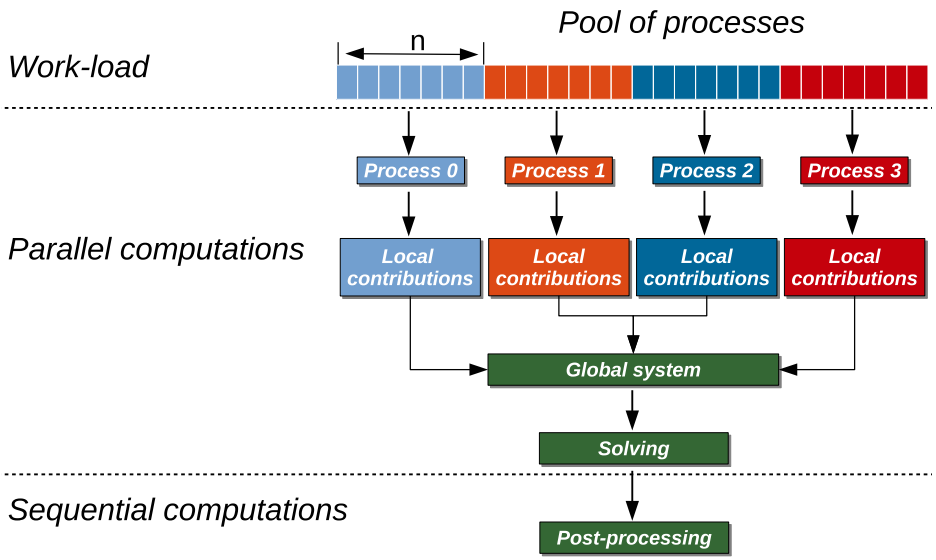


Fig. 2.5 Parallel scheme for global matrix assembly and solution in *PETGEM* using a pool with 4 processes. Here the master process manage a pool of multiple processes which kept ready to be used depending on the demand of their service. The chunksize (number of elements) per process is represented by n. At the end, global results are collected from each process.

The MPICH (MPICH2-Team, 2016) and Open MPI (Gabriel et al., 2004) implementations are available from vendors of high-performance computers. Furthermore, there are many Python implementations: OOMPI (Squyres et al., 2016), Pypar (Nielsen, 2016), pyMPI (Miller, 2016), Scientific Python (Hinsen, 2014), Numarray (Greenfield et al., 2016) and Pyfort (Dubois, 2016). However all of them lack from interface conformance with the MPI standard, in other words, they offers rather minimal Python interfaces to MPI because there is no support for communicators or process topologies, there is no support for direct array communication and they does not permit interactive parallel runs.

Previous attemps have encouraged the development of the *mpi4py* package, an open source software that provides bindings of the MPI standard for the Python programming language, allowing any Python code to exploit multiple processors architectures. The *mpi4py* code is constructed on top of the MPI specifications and provides support to point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communications of any picklable Python object, as well as optimized communications of Python object exposing the single-segment buffer interface (*NumPy* arrays, builtin bytes/string/array objects) (Dalcín et al., 2005, 2008, 2011).

Moreover, matrices and vectors are extremely important in parallel scientific com-

puting. For this reason, this thesis focuses on the maintenance of these data structures within the *PETSc* framework through the *petsc4py* package. As result, *PETGEM* exploits the distributed parallel layer of *PETSc*, with support for data types such as vectors and sparse matrices, as required by *PETGEM* and methods for manipulating them. Furthermore, there is significant support for direct and iterative solvers. Additionally, the mesh partitioner *Metis* and its parallel variant *Parmetis* can also be called from *petsc4py*.

On top of that, the parallel approach on distributed-memory architectures reuse conceptual basis of the method applied in the shared-memory scheme. Figure 2.6 shows an upper view of the matrix assembly and solution using the *mpi4py* and *petsc4py* packages in *PETGEM*. Implementation details of *petsc4py* classes and methods in *PETGEM* are documented in Appendix D. Besides, this thesis is based on *PETSc* documentation (Balay et al., 2016). The first step is to partition the work-

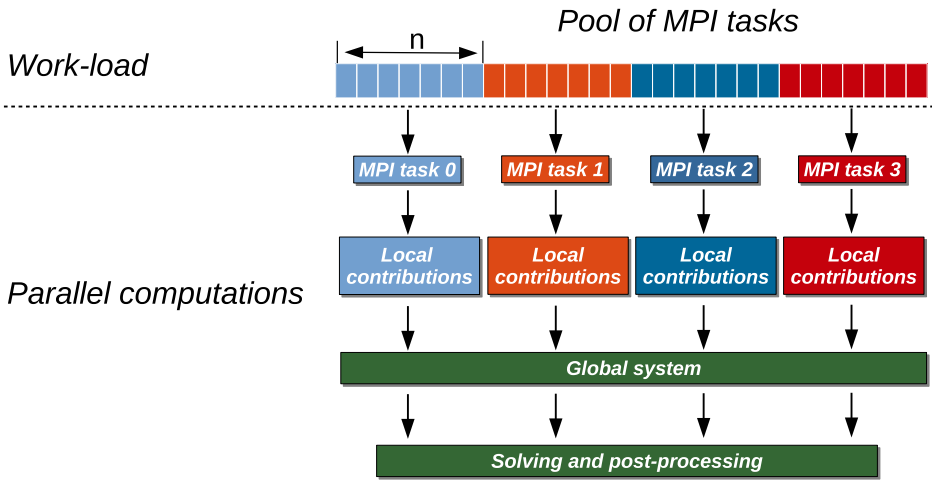


Fig. 2.6 Parallel scheme for assembly and solution in *PETGEM* using 4 MPI tasks. Here the elemental matrices computation is done in parallel. After calculations the global system is built and solved in parallel using the *petsc4py* and *mpi4py* packages.

load into subdomains. This task can be done by Metis library, which makes load over processes balanced. After domain partition, subdomains are read and assigned to MPI tasks and the elemental matrices are calculated concurrently. These local contributions are then accumulated into the global matrix system. To contain global matrix A and other data such as adjacency matrices, *PETGEM* use *petsc4py* calls to create a parallel matrix object, namely, *Mat().createAIJ()* class. Because matrix A is a sparse symmetric matrix, AIJ format (CSR) is used to store it. Since dynamic memory allocation and copying between old and new storage are very expensive in sparse format, it is critical to preallocate the memory needed for the matrix A. This preallocation of

memory is very important for achieving good performance during matrix assembly, as this reduces the number of allocations and copies required. *PETGEM* pre-processing can determine the nonzero structure for a given EFEM mesh (see Appendix D for more details).

After the matrix A has been created, the contributions must be inserted. Following the *PETSc* strategy, in *PETGEM* each MPI task loops the elements in its local domain, computes the local contributions and assembles them into global matrix A, without regard to which process eventually stores them. This task can be done in two ways with *petsc4py*, by either inserting a single value or inserting an array of values. In sake of performance, *PETGEM* implemented the second approach because it reduce the number of *PETSc* calls. This task is done by calling the *Mat.setValues()* method. Also, there are similar procedures to create vectors and insert values into them. *Petsc4py* currently provides two basic vector types: sequential and parallel vector. The created vector is distributed over all CPUs. Any CPUs can set any components of the vector and *PETSc* insures that they are automatically stored in the appropriate locations. After the matrix/vector elements have been inserted or added, they must be processed before the solve can be performed. The *petsc4py* methods for matrix/vector processing are *Mat.assemblyBegin()*, *Mat.assemblyEnd()*, *Vec.assemblyBegin()* and *Vec.assemblyEnd()*.

Subsequently, the system is ready to be solved. *PETGEM* uses the Krylov Subspace Package (KSP) from *PETSc* through the *petsc4py* package. The object KSP provides an easy-to-use interface to the combination of a parallel Krylov iterative method and a preconditioner (PC) or a sequential direct solver. As result, *PETGEM* users can set various solver options and preconditioner options at runtime via the *PETSc* options database. Details about KSP and PC context creation and its tunning are described in the *PETGEM* documentation (Appendix D).

Since *PETGEM* kernel knows which portions of the matrix and vectors are locally owned by each CPUs, the post-processing task is also completed in parallel following the numerical scheme described in Section 2.2.

All *petsc4py* classes and methods are called from the *PETGEM* kernel in a manner that allows a parallel matrix and parallel vectors to be created automatically when the code is run on many CPUs. Similarly, if only one CPUs is specified the code will run in a sequential mode. Although *petsc4py* allows control the way in which the matrices and vectors to be split across the CPUs on the architecture, *PETGEM* simply let *petsc4py* decide the local sizes in sake of computational flexibility. However, this can be modified in an easy way without any extra coding required.

2.6 Scalability tests

The scalability of the code has been tested on shared-memory and distributed-memory architectures by running the same problem for different number of CPUs working in parallel. In this set of experiments the most time-consuming sections have been considered, namely, assembly and solving tasks. All simulations have been carried out on version III of the *Marenostrum* supercomputer at *Barcelona Supercomputing Center-Centro Nacional de Supercomputación (BSC-CNS)*.

Marenostrum III (MNIII) is a supercomputer based on Intel SandyBridge processors, iDataPlex Compute Racks, a Linux Operating System and an Infiniband interconnection. Its has 48,896 Intel SandyBridge-EP E5–2670 cores at 2.6 *GHz* grouped into 3,056 computing nodes, 103.5 *TB* of main memory (128 nodes with 128 *Gb*, 128 nodes 64 *Gb* and 2880 nodes with 32 *Gb*) as well as 1.9 *PB* of GPFS disk storage. Its peak perfomance is 1.1 Petaflops. Each computing node has two 8-core Intel Xeon processors E5-2670 with a frequency of 2.6 *GHz* and 20 *MB* cache memory.

The following tests are based on the canonical model described in Section 3.1. Its mesh has been created with *Gmsh* and has $4,451,735$ elements, $723,586$ nodes and $5,223,449$ edges (mesh of first-level). Since real models normally have more DOFs we applied an uniform refinement to this mesh, namely, we used the refinement functions included in *PETGEM*. The second-level of the automatic mesh refinement has created a big mesh that has $35,613,880$ elements, $5,947,035$ nodes and $41,753,430$ edges. By exploiting this methodology, the scalability of the code can be evaluated in a way that is relevant for real-scale modelling. First-level mesh have been simulated using the shared-memory *PETGEM* version. Since second-level mesh has huge memory requirements, these tests have been performed using the distributed-memory *PETGEM* version on 32, 64, 128, 256, 512 and 1024 CPUs, using all 16 CPUs per node.

2.6.1 Shared-memory tests

First set of tests has been carried out using the shared-memory version of *PETGEM* on 1 node with 64 *Gb* of memory. Its are based on parallel approach described in Subsection 2.5.1. The performance is measured based on the wall clock times for the matrix assembly and solving tasks.

Figure 2.7 show the assembly speed-ups obtained for up 16 CPUs of MNIII for the first-level mesh ($4,451,735$ elements, $723,586$ nodes and $5,223,449$ DOFs). The achieved scalability is almost linear for up 8 CPUs. From this number on, the scalability stops its near-linear growth and slowly begins to saturate since the execution

becomes dominated by mainly three events: 1) creating and starting the processes, 2) passing the function and the arguments over to them and, 3) waiting for process termination. However, the speed-ups keep growing constantly until achieve $\approx 14x$ from the ideal $16x$. It is important to emphasise that profiling tools do not offer support for Multiprocessing Python applications (at least at the time of writing this document), thus a more insights into what happens when the number of CPUs is increased cannot be presented in this thesis. Table 2.13 shows the runtime, speed-up and parallel

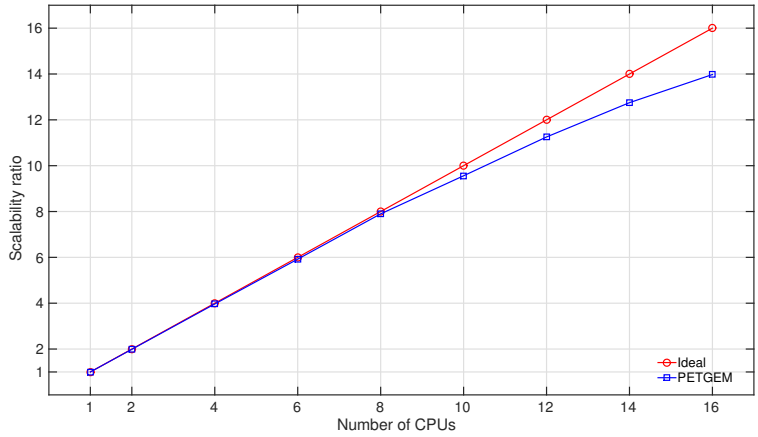


Fig. 2.7 Scalability tests for the first-level mesh (3, 793, 356 DOFs) on MNIII.

efficiency that processes have spent on performing computations. Analysing these results, it is easy to see that the computation time has been reduced by increasing the number of processes (around 14 times when increasing the number of CPUs from 1 to 16. Finally, although Multiprocessing package is efficient for the parallel assembly

# CPUs	1	2	4	6	8	10	12	14	16
Runtime (Min)	232.48	116.95	58.61	39.25	29.43	24.32	20.66	18.23	16.63
Speed-up	-	1.98	3.96	5.92	7.89	9.55	11.25	12.75	13.98
Efficiency	-	.99	.99	.98	.98	.95	.93	.91	.87

Table 2.13 Execution results for different number of CPUs on shared-memory architectures

task it does not allow to exploit the parallelism offered by the solution of the system. Therefore, the solution of the system has been done in parallel using the MKL library by Intel.

2.6.2 Distributed-memory tests

Second set of tests has been carried out using the distributed-memory version of *PETGEM* on 32, 64, 128, 256, 512 and 1024 CPUs, using all 16 CPUs per node. Its are based on the parallel approach described in Subsection 2.5.2. These experiments are relevant because although matrix assembly quickly disappears with the increase of the number of CPUs on shared-memory architectures, this represents only a portion of the total execution time and commonly the iterative solver is the most dominant and expensive region of the code. Furthermore, parallelism on distributed-memory platforms offer greater flexibility and capacity for large-scale computations such as the 3D CSEM FM.

Figure 2.8 show speed-ups obtained for up 1024 CPUs of MNIII for the second-level mesh (35, 613, 880 elements, 5, 947, 035 nodes and 41, 753, 430 DOFs). The achieved scalability is almost linear for up to 256. From this number on, the scalability stops its near-linear growth and slowly begins to saturate since the execution becomes dominated by exchange of messages between MPI tasks. However, the speed-ups keep growing constantly and significant reductions in runtime for more than thousand CPUs has been observed. Table 2.14 shows the runtime, speed-up and parallel efficiency that

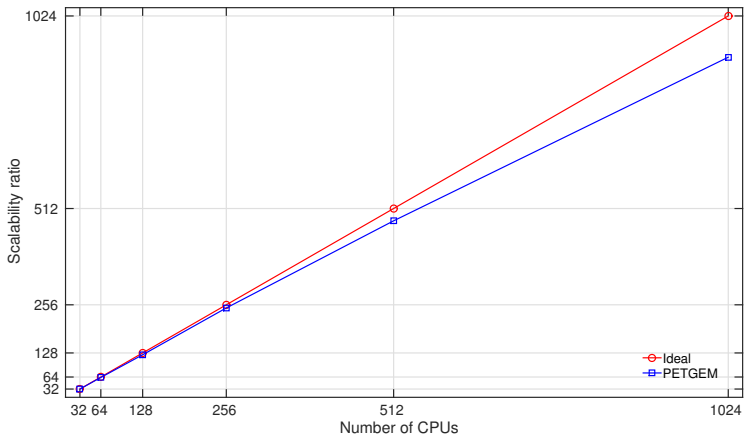


Fig. 2.8 Scalability tests for the second-level mesh (41, 753, 430 DOFs) on MNIII.

processes have spent on performing the modelling. Analysing these results, it is easy to see that the computation time has been reduced by increasing the number of processes (around 26 times when increasing the number of CPUs from 32 to 1024). In order to perform a more thorough analysis of the MPI parallelism within *PETGEM*, we have carried out a set of simulations that has been been analyzed using the performance tools developed at BSC-CNS: Paraver and Dimemas. These tools responds to the basic need to have a qualitative global perception of the application behaviour

# CPUs	32	64	128	256	512	1024
Runtime (Min)	945.30	482.14	246.09	122.96	63.08	36.92
Speed-up	-	1.96	3.84	7.68	14.98	25.60
Efficiency	-	.98	.96	.96	.94	.80

Table 2.14 Execution results for different number of CPUs on distributed-memory architectures

by visual inspection and then to be able to focus on the detailed quantitative analysis of the problems. Performance tools generate information that directly improves the decisions in whether and where to invert the programming effort to optimize an application. The result is a reduction of the development time as well as the minimization of the hardware resources required for it (Heroux et al., 2006).

The analysis methodology begins with a set of Paraver traces for a number of MPI tasks, obtained from executing an instrumented version of *PETGEM*. Then, from a visual analysis of the traces clean cuts are generated in order to identify the main computational phases. This allows identify the different environments of the code, namely, is possible to separate main computational regions from communication intensive stages, e.g. elemental matrix computations or solving phase from MPI calls. Furthermore, in this step, additional information such as useful computational duration and number of MPI calls can be measured. The net result of this analysis phase is depicted in figure 2.9, where the color represents the duration of computation burst (useful duration). This view gives a good perception of where are the major computation phases, and their balance across processors. As result, in figure 2.9 is easy to see that assembly and solver phases are the main computational regions. Once the code structure has been identified, we analize Paraver traces of *PETGEM* using 16, 32 and 64 CPUs in order to measure times for the aforementioned main computational regions. The main advantage of this strategy is that trace sizes are smaller and more manageable, in addition, all the effects that appear with the increase of the CPUs can be noticed much earlier.

The scalability ratio for these experiments is show in figure 2.10, where is easy to observe a quasi-linear ratio for up to 64 CPUs. Furthermore, the assembly task is slightly more efficient than solving because it is an embarrassingly parallel task, or task where does not exists dependency (or communication) between those parallel task. In sake of clarity, we analize the number of solver iterations for each Paraver trace a partir . Thus, we cut a representative region of the solver phase and we

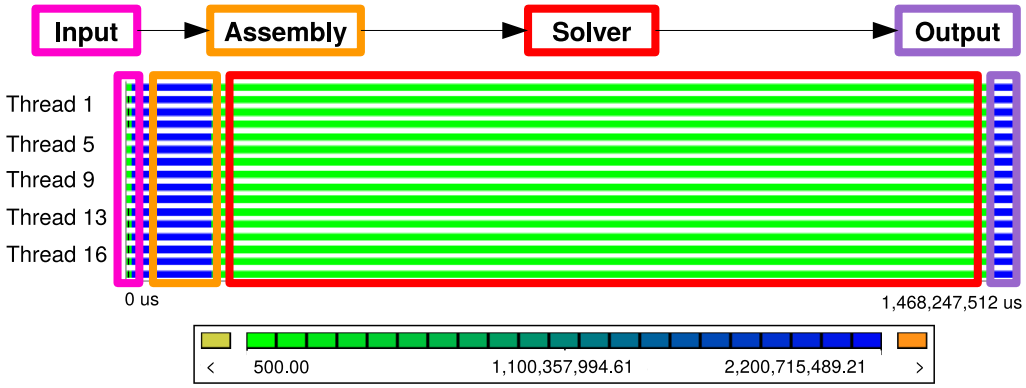


Fig. 2.9 Main computational phases in *PETGEM*. Here, the useful duration is plotted for each CPU in function of time.

measured the time (window size) for a number of iterations in the trace with 16 MPI tasks. Then, we count the number of iterations that fit in the same window size for remaining Paraver traces (32 and 64 CPUs). In our experiments, we fixed the initial number of solver iterations to 10, which produced a window size equal to 38,104,404 microseconds (us). Results of this analysis is depicted in figure 2.11, were is easy to see an acceptable performance in number of iterations when increasing number of CPUs.

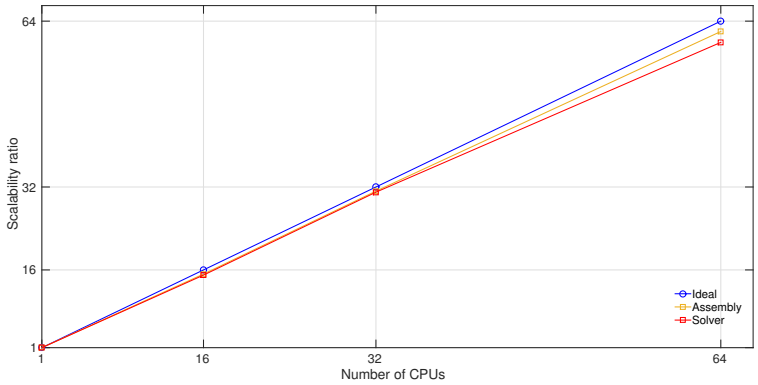


Fig. 2.10 Scalability ratio of main computational phases in *PETGEM*.

The conclusion is, what has been demonstrated with examples, that *PETGEM* offers an acceptable perfomance.

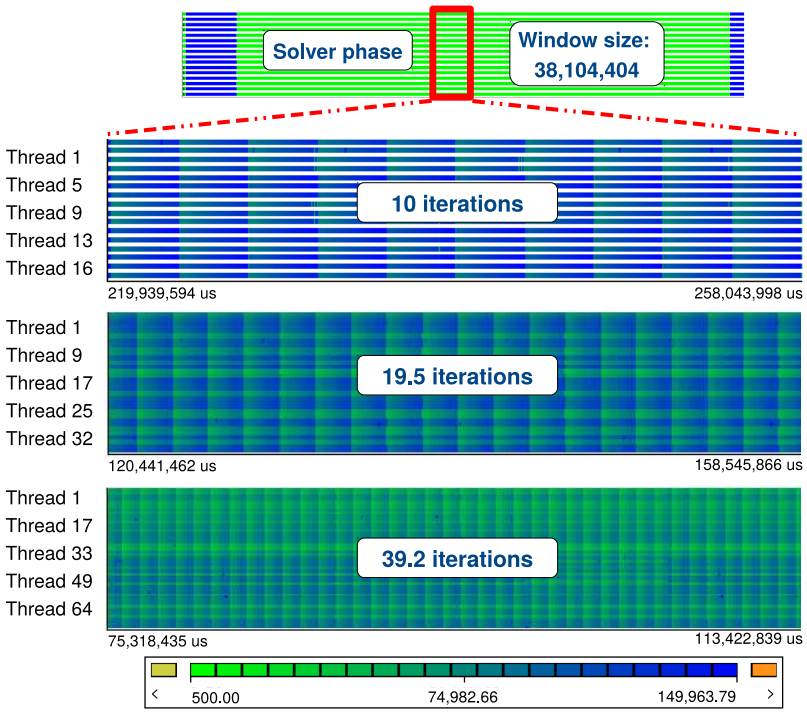


Fig. 2.11 Solver scalability analysis using Paraver.

Chapter 3

Use cases of 3D CSEM FM

This chapter is devoted to the numerical simulation of 3D CSEM FM cases using the parallel python code introduced in the previous chapter. All models that we have used in the tests demonstrated here can be found in the literature. Futhermore, an automatic mesh adaptation strategy and the performance of the solvers offered by *PETSc* for the solution of the EM problem are studied.

3.1 Canonical model of an off-shore hydrocarbon reservoir

The first model is a canonical modelling by (Constable and Weiss, 2006) which consists in four-layers: 1,000 m thick seawater (3.3 Sm^{-1}), 1,000 m thick sediments (1 Sm^{-1}), 100 m thick oil (0.01 Sm^{-1}) and 1,400 m thick sediments (1 Sm^{-1}). The computational domain is a $[0, 3500]^3$ m cube which was discretized into $4,984,767$ tetrahedral elements, resulting in $951,728$ nodes and $6,106,217$ DOFs. Figure 3.1 shows a 3D view of the model with its unstructured tetrahedral mesh for the halfspace y>1,750 m, where the color scale represents the electrical conductivity (σ) for each layer.

The electromagnetic field is excited by an horizontal electric dipole, described in Section 2.1, oriented in the x direction with a moment of 1 Am and located in the seawater with the coordinates ($x = 1750$, $y = 1750$, $z = -975$) m, which is 25 m above seafloor. The frequency of the harmonic electric source is 2 Hz. The receivers are placed in-line to the source position and along its orientation, directly above the seafloor ($z = -990$ m).

Figure 3.2a and 3.2a shows a comparison of $\mathbf{E}_x$ measurements between *PETGEM* solution and those obtained with the WHAM tool (Key, 2009). Figure 3.2a depicts

the amplitude ratio on receivers where it is easy to observe the effect of the imperfect absorbing boundaries (dirichlet boundary conditions), which can be mitigated by enlargening the domain with element sizes increasing logarithmically outwards from the zone of interest. In addition, figure 3.2b reveals only small phase ratio differences on receivers close to boundaries.

Figures 3.3a and 3.3b shows that amplitude and phase fields components for inline receivers far to boundaries are below the requested 5% error tolerance. For receivers close to boundaries, the error exceeds this threshold because low absorption capacity of the boundary conditions. However, these results demostrated the validity of the *PETGEM* solutions for this model. The mean runtime for this model is 74.62 minutes (11.94 for assembly and 62.68 for solving) using 64 MPI tasks and required less than 78.7 *Gb*. In order to extends this test and to illustrate the code's effectiveness, we have

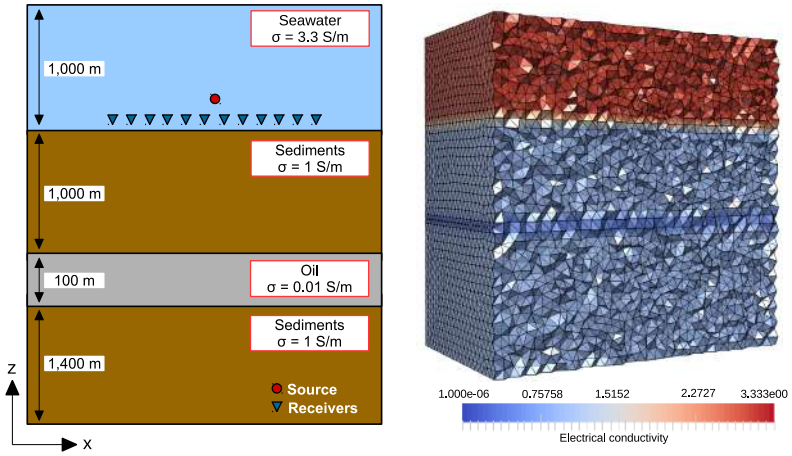
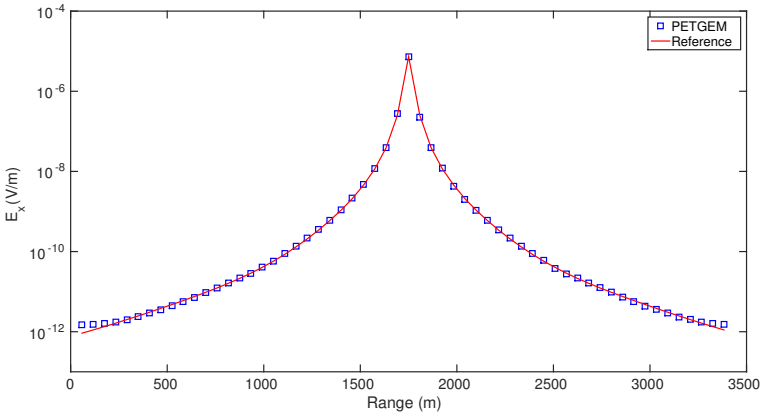
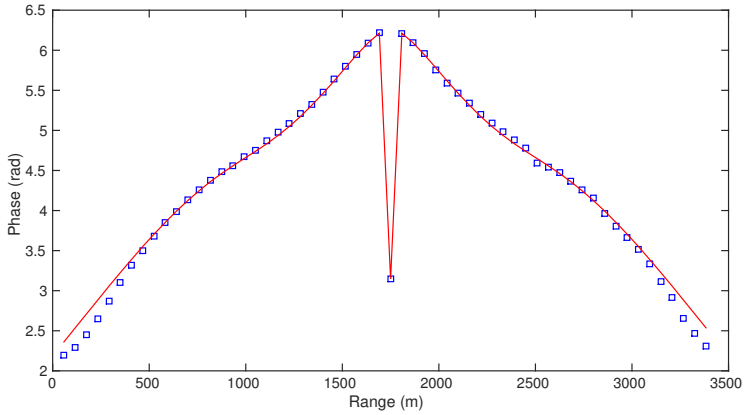


Fig. 3.1 In-line canonical off-shore hydrocarbon model with its unstructured tetrahedral mesh for y>1,750 m. The color scale represents the electrical conductivity σ for each layer.

prepared a set of hierarchically refined meshes so that we could verify the convergence of the obtained solution. For all cases the mesh has been locally refined around the source region. Using WHAM as reference solution and excluding those receivers closest to the boundaries, we have quantified the errors in *PETGEM* resulting electric fields by means of the L^1, L^2 and L^{inf} for the set of meshes, as plotted in figure 3.4. DOFs, mesh spacing and errors for each mesh are depicted in table 3.1, which also shows the expected linear convergence of the numerical scheme for all error norms and mesh sizes. Finally, table 3.1 also include some information about the algorithmic effort using a Biconjugate Gradient Stabilized Method (BiCGSTAB) solver for all cases. In general, the numerically *PETGEM* results are remarkably similar to those from quasi-analytical results in canonical models. Furthermore, the numerical results also

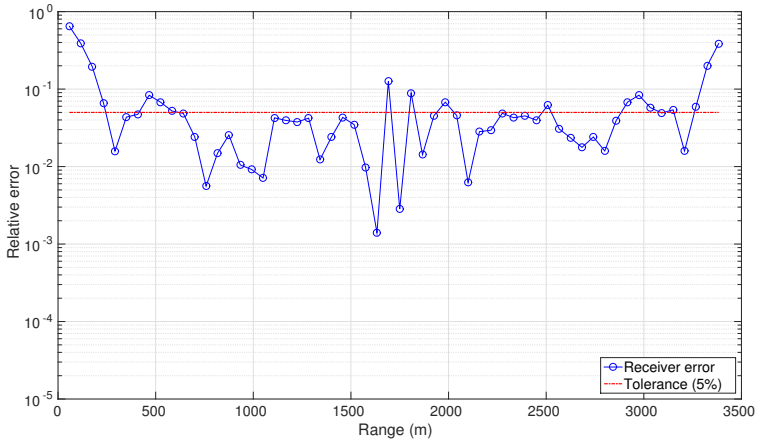


(a) Amplitude field comparison.

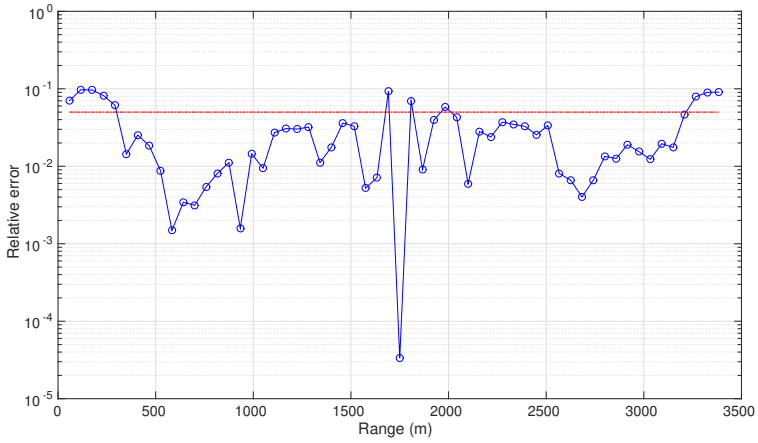


(b) Phase field comparison.

Fig. 3.2 Amplitude and phase comparison for $\mathbf{E}_x$ between *PETGEM* and the analytical solution of canonical model in figure 3.1.



(a) Amplitude errors on receivers.



(b) Phase errors on receivers.

Fig. 3.3 Relative errors for amplitude and phase on $\mathbf{E}_x$ shown in figures 3.2a and 3.2b.

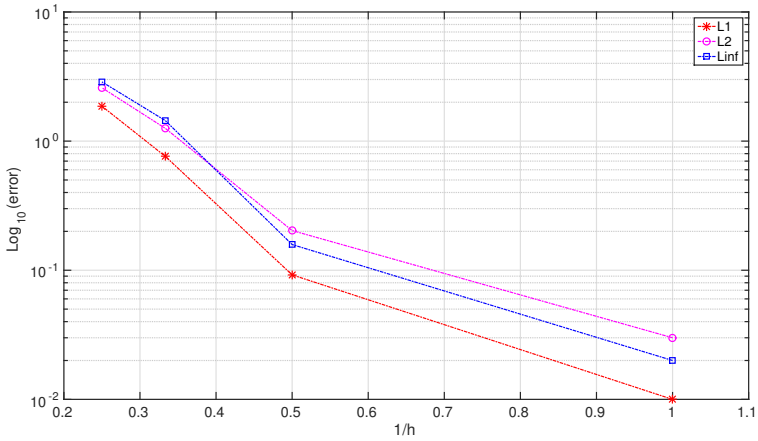


Fig. 3.4 Convergence order in L^1, L^2 and L^{inf} norm for model in figure 3.1. These norms were calculated using a mesh with ≈ 12 millions of DOFs.

demonstrate convergence to the reference solution.

Mesh	DOFs	$h(m)$	L^1	L^2	L^{inf}	Iterations	$\|\|r\|\|$
1	6.17e04	20e01	2.8447e-07	2.5059e-07	2.4939e-07	1776	9.9771e-07
2	4.36e05	10e01	1.6652e-07	1.0365e-07	9.0489e-08	3468	9.9703e-07
3	3.43e06	5e01	1.2859e-07	8.9955e-08	7.3234e-08	5512	9.7994e-07
4	11.9e06	3.3e01	1.5615e-08	8.3129e-08	7.0474e-08	8986	1.8174e-06

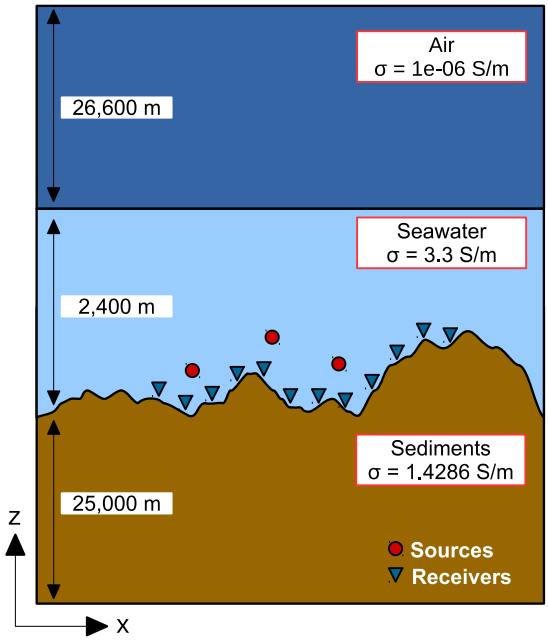
Table 3.1 Summary of results for convergence test and BiCGSTAB solver.

3.2 3D CSEM FM with bathymetry

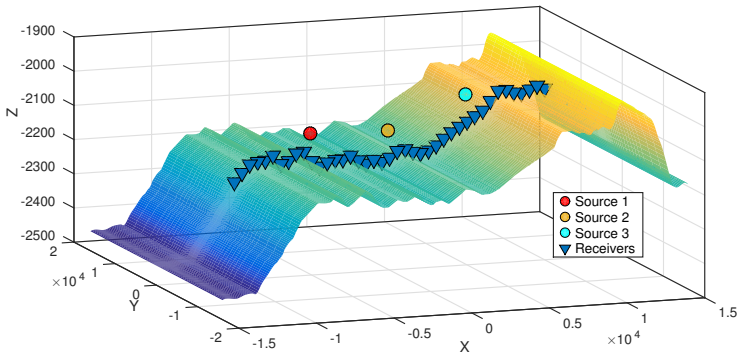
The second test involved a 3D CSEM FM with bathymetry. This model is especially interesting because a primary advantage of the EFEM over other techniques like the Finite Difference Method (FDM) is the fact that the EFEM allow precise and efficient representations of arbitrarily complex geological structures such as seafloor bathymetry without critically increasing the number of DOFs. Furthermore, if not taken into account bathymetry effects can produce large anomalies on the measured electric fields.

The model consists of a 26,600 m thick air layer (1e-6 Sm^{-1}), a 2,400 m thick seawater (3.3 Sm^{-1}) and a 25,000 m thick sediments (1.4286 Sm^{-1}). The computational domain is defined by $[30, 32, 54]$ km as show figure 3.5a. The model was discretized into $3,879,007$ tetrahedral elements, resulting in $649,028$ nodes and $4,535,045$ DOFs. The reference dataset of this model was provided by (Chung et al., 2014). Furthermore, a nodal FEM solution of this model is described in (Um et al., 2013).

The three transmitters are x-oriented electric dipole sources, with moment of 200 Am and frequency of 0.25 Hz, located at points with the coordinates ($x = -5000$, $y = 0$, $z = -2086$) m, ($x = 0$, $y = 0$, $z = -2096$) m, ($x = 5000$, $y = 0$, $z = -2001$) m. The 41 receivers are placed in-line to the source position and alongs its orientation, directly above the seafloor as show figure 3.5. The bathymetry in the area is very rough, and adequate spatial sampling can only be achieved by allowing receiver deployment in slopes or trenches. Several steep trenches go through the area from the shallow eastern part to the deeper western part. For this model, the resulting system of equations was solved with a Generalized Minimal Residual (GMRES) solver which has been preconditioned using a Symmetric succesive over-relaxation method (SOR).



(a) Model overview.



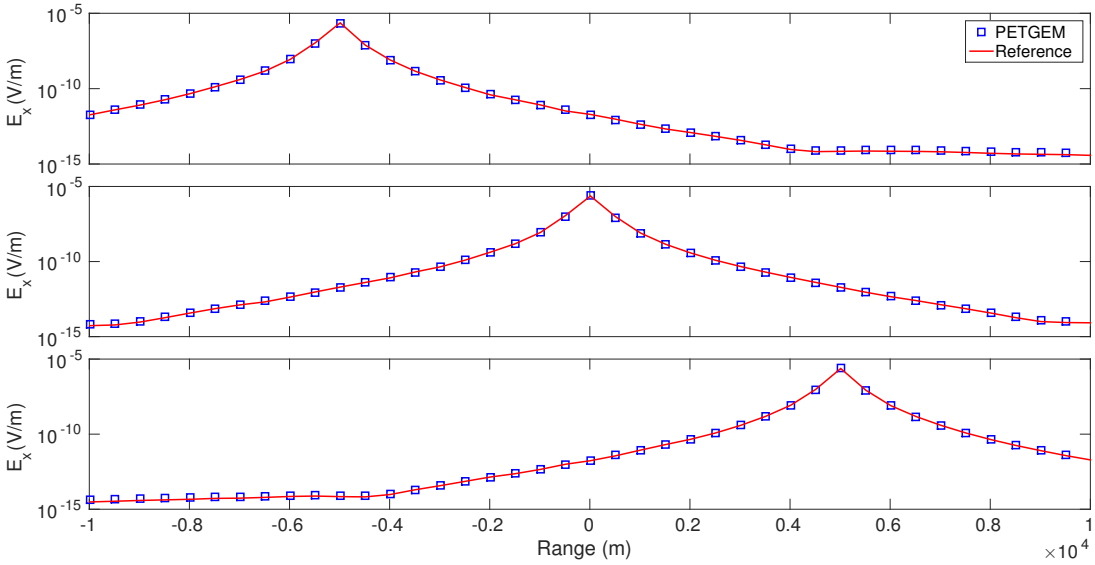
(b) 3D view of the seabed bathymetry.

Fig. 3.5 Bathymetry model description.

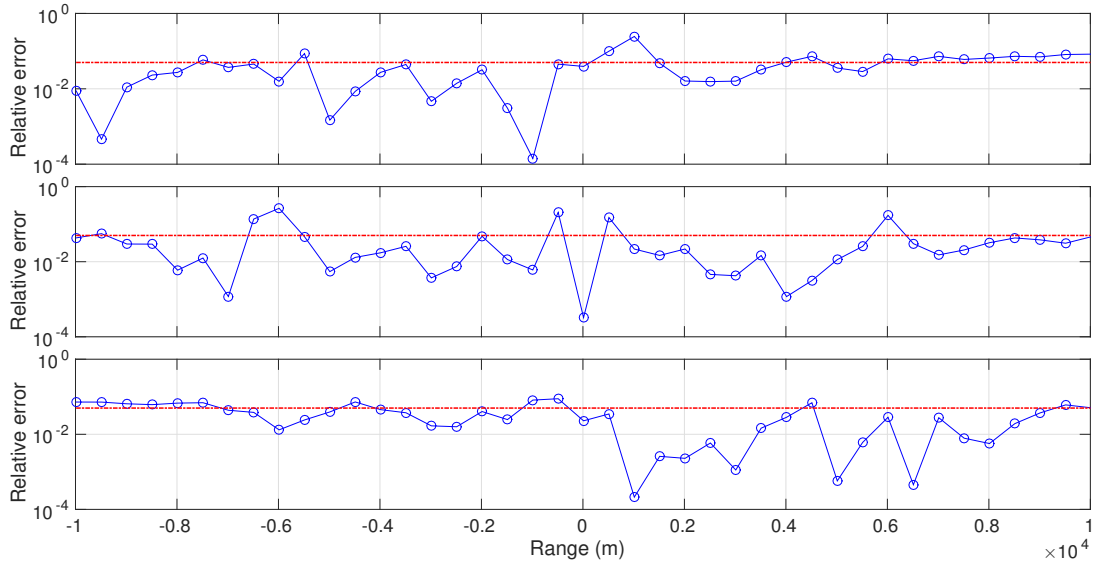
Figures 3.6a and 3.6b compares the $\mathbf{E}_x$ components and its phase obtained from *PETGEM* to those produced by (Chung et al., 2014). Most of the current successful applications of 3D CSEM FM are offshore because the water strongly attenuates anthropogenic and natural noise. However, one of the most significant problems in this environment is the air-wave effect when the water layer is shallow. The air-wave is the secondary EM field refracted from the air-water interface which dominates the recorded signal at large offsets. Therefore, the hydrocarbon detection ability of the 3D CSEM FM is weakened because the airwave is independent of the subsurface properties. Solutions to the airwave problem have been offered by (Eidesmo et al., 2002; Weiss and Constable, 2006; Newman et al., 2010; Everett, 2012). Although this test is not focused on the solution of this problem, in the figures 3.6a and 3.6b is easy to see the airwave effect in the receivers near the boundaries which demonstrates the physically meaningful of the results. Furthermore, the amplitude and phase errors are show in figures 3.7a and 3.7b, respectively. In general, the *PETGEM* results show a good overall agreement with the reference. In terms of performance, in (Chung et al., 2014) the solution for the model was computed using EFEM over an hexahedral mesh with 265 x 64 x 73 cells and $3,796,596$ DOFs. Furthermore, the authors reported that the computation took 64.53 minutes and required less than 88.5 *Gb* of memory on one node equipped with two Intel quad-core Xeon processors (resulting in eight cores) at 2.53 *GHz* sharing 96 *Gb* of memory. In our experiments, *PETGEM* simulations were executed with 64 MPI tasks. The mean runtime for this model is 86.17 minutes (6.89 for assembly and 79.27 for solving) and required less than 64 *Gb*.

3.3 Synthetic model with real target

The third model focuses on a synthetic case of 3D CSEM FM with a non-infinite target (reservoir). The model is composed by three flat-layers: 8,000 m thick air (1e-08 Sm^{-1}), 2,000 m thick seawater (3.3 Sm^{-1}) and 10,000 m thick sediments (1 Sm^{-1}). A resevoir (2e-02 Sm^{-1}), with the size of [3000, 3000, 50] m, is embedded in the marine sediment with the center at ($x = 0$, $y = 0$, $z = -2600$) m. The computational domain is defined by $[37, 37, 20]$ km. Figure 3.8 shows a 3D view of the model with its unstructured tetrahedral mesh for the halfspace y>18,500 m, where the color scale represents the electrical conductivity (σ) for each layer. This mesh contains $7,043,899$ tetrahedral elements, resulting in $1,207,452$ nodes and $8,261,676$ DOFs. The resulting size of the sparse matrix is $8,261,676$ x $8,261,676$. From figure 3.8, one can see that the mesh is refined in the areas of the sources, reservoir domain, and the

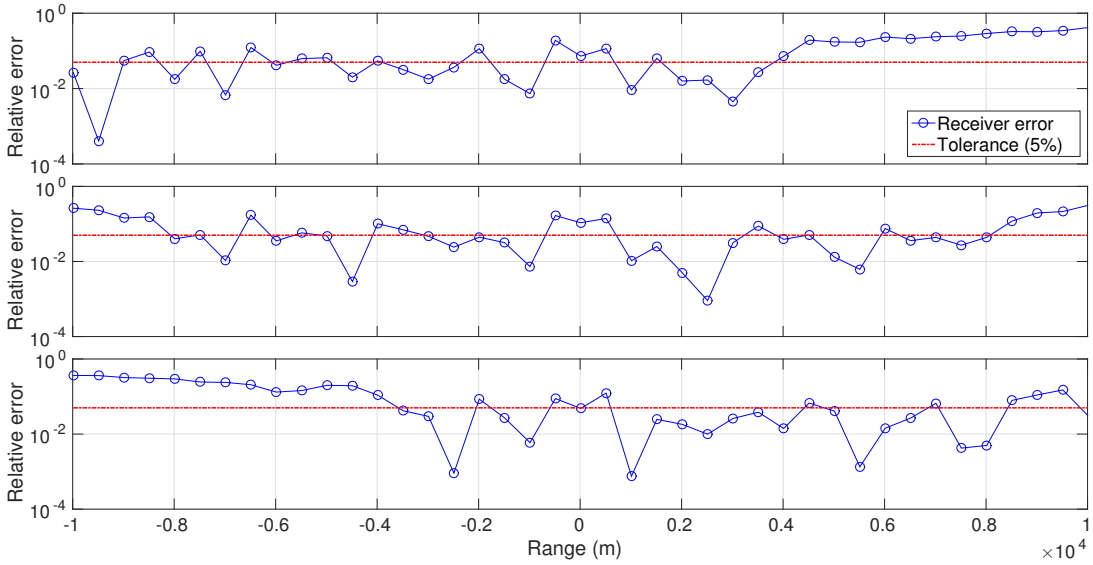


(a) Amplitude field comparison. The sources are plotted on the top (source 1), middle (source 2) and bottom (source 3).

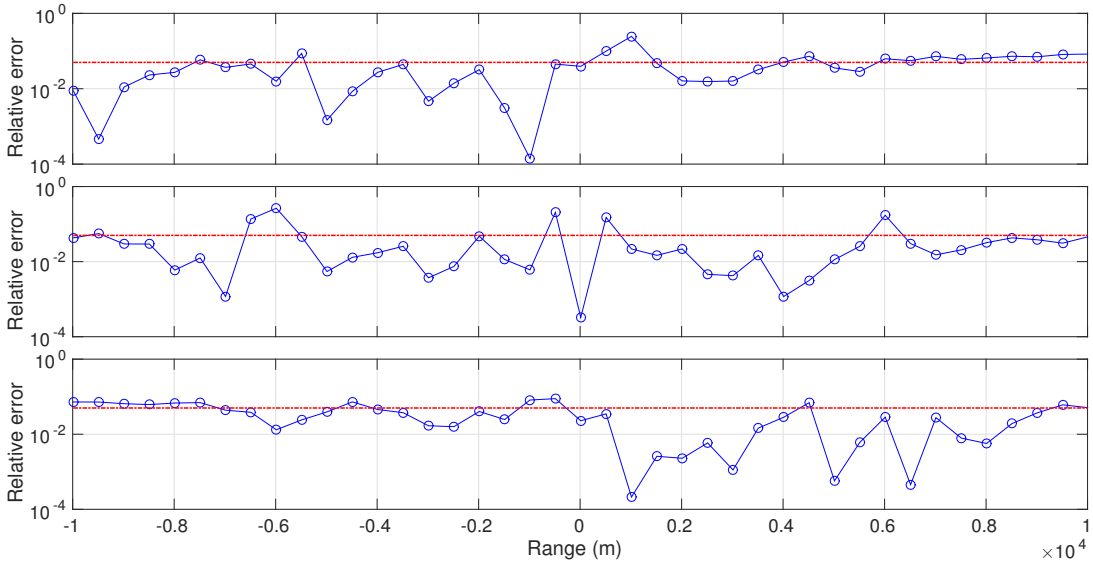


(b) Phase field comparison.

Fig. 3.6 Amplitude and phase comparison for $\mathbf{E}_x$ between *PETGEM* and those obtained by (Chung et al., 2014) for the model in figure 3.5a.



(a) Amplitude errors on receivers. The sources are plotted on the top (source 1), middle (source 2) and bottom (source 3).



(b) Phase errors on receivers. Considering the intrinsic errors of the numerical method, the differences are reasonable.

Fig. 3.7 Relative errors for amplitude and phase on $\mathbf{E}_x$ solution show in figures 3.6a. and 3.6b.

seafloor, where the data are measured by the receivers. In order to investigate the

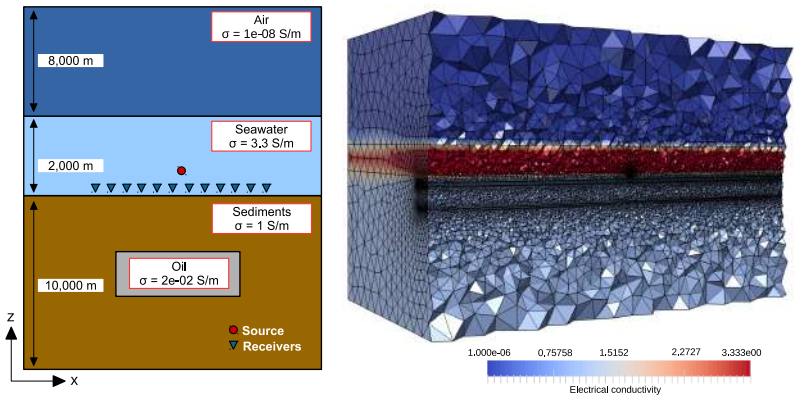


Fig. 3.8 Synthetic 3D CSEM model with a target with its unstructured tetrahedral mesh for y>15,500 m. The color scale represents the electrical conductivity σ for each layer.

source position effect, we used four x-oriented electric dipoles located 25 m above the seafloor at points with the coordinates ($x = -1500$, $y = 0$, $z = -1975$) m, ($x = -500$, $y = 0$, $z = -1975$) m, ($x = 500$, $y = 0$, $z = -1975$) m and ($x = 1500$, $y = 0$, $z = -1975$) m. The frequency of excitation current is 3 Hz with a moment of 1 Am. The 91 receivers are placed in-line to the source positions and along its orientation, directly above the seafloor ($z = -1999$ m).

The average runtime for this model is 92.47 minutes (7.39 for assembly and 85.07 for solving) using 128 MPI tasks. The mean memory usage is 76.6 Gb. The numerical results obtained by *PETGEM* was compared to those computed with the Barcelona Subsurface Imaging Tools (BSIT).

For typical 3D CSEM FM survey, the target response is much smaller than the background field. It is critical for the FM to accurately simulate the target effect. Therefore, two set of simulations have been executed. First, by setting the conductivity of the target to that of the sediments, the EM responses were computed. Next, the conductivity of the reservoir was setted and the EM responses for this model were calculated.

Figure 3.9 shows an amplitude comparison of $\mathbf{E}_x$ component for both models and for all sources along the profile at $y = 0$ m. A comparison between left panels and right panels of figure 3.9 shows that the field is distorted significantly by the target. The anomaly was observed around $x = -1550$ to $x = 1550$ m, which corresponds with the reservoir location. For the remaining offset, the total electric field is almost the same. The phase comparison of these responses are plotted in figure 3.10. Again, for this

model the results are practically the same because the normalized misfit in amplitude and phase is 0.5% as shows figures 3.11 and 3.12. In conclusion, the numerical results

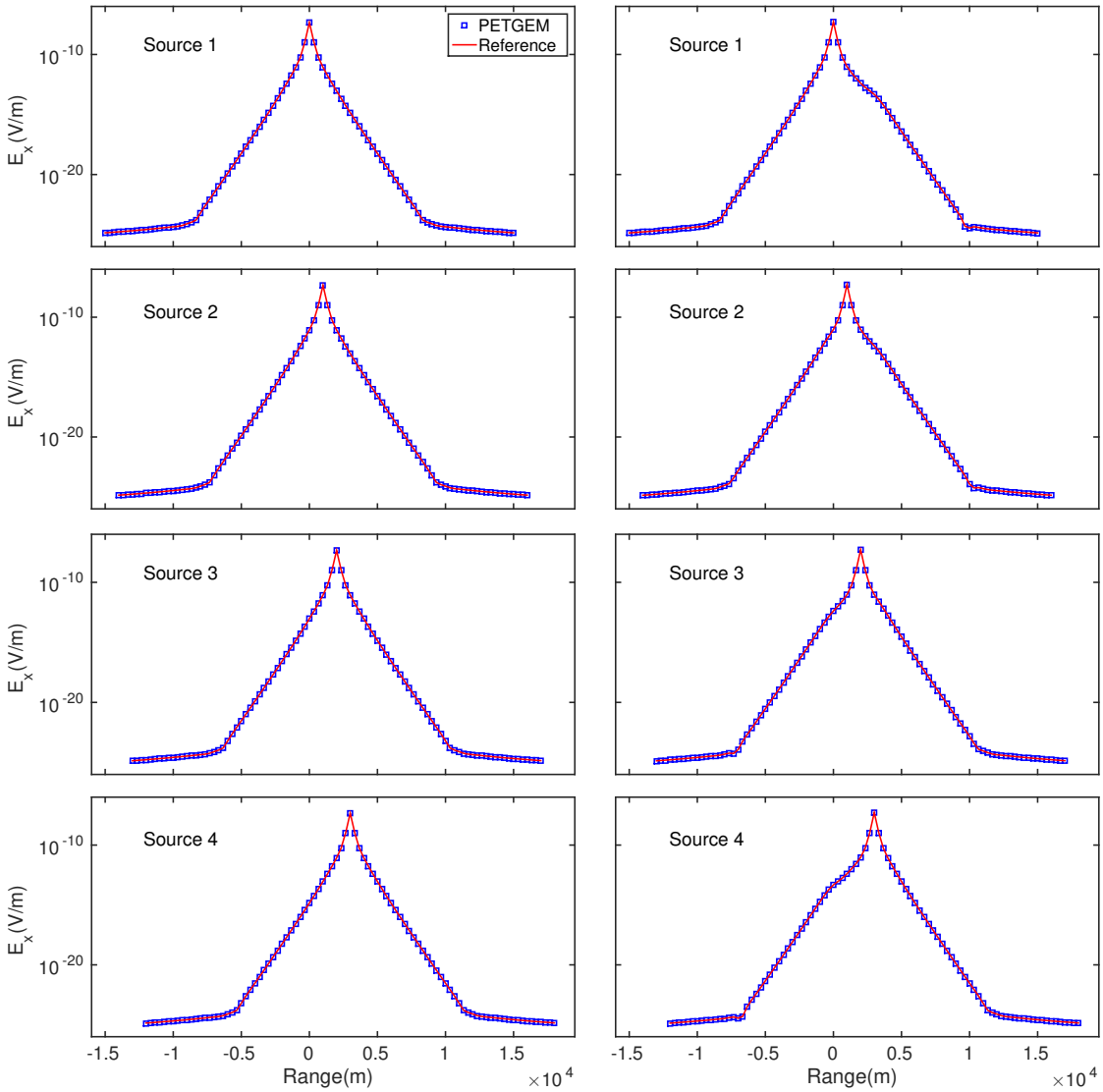


Fig. 3.9 Amplitude comparison for $\mathbf{E}_x$ between *PETGEM* and those obtained by BSIT. Left panels correspond to the model without target while the right panels correspond to the model with reservoir.

produced by *PETGEM* coincide well with the reference solution. Thus, modelling results confirm ones again the quantitative value of 3D CSEM FM data interpretation to reliable definition and characterization of bodies electric resistivity.

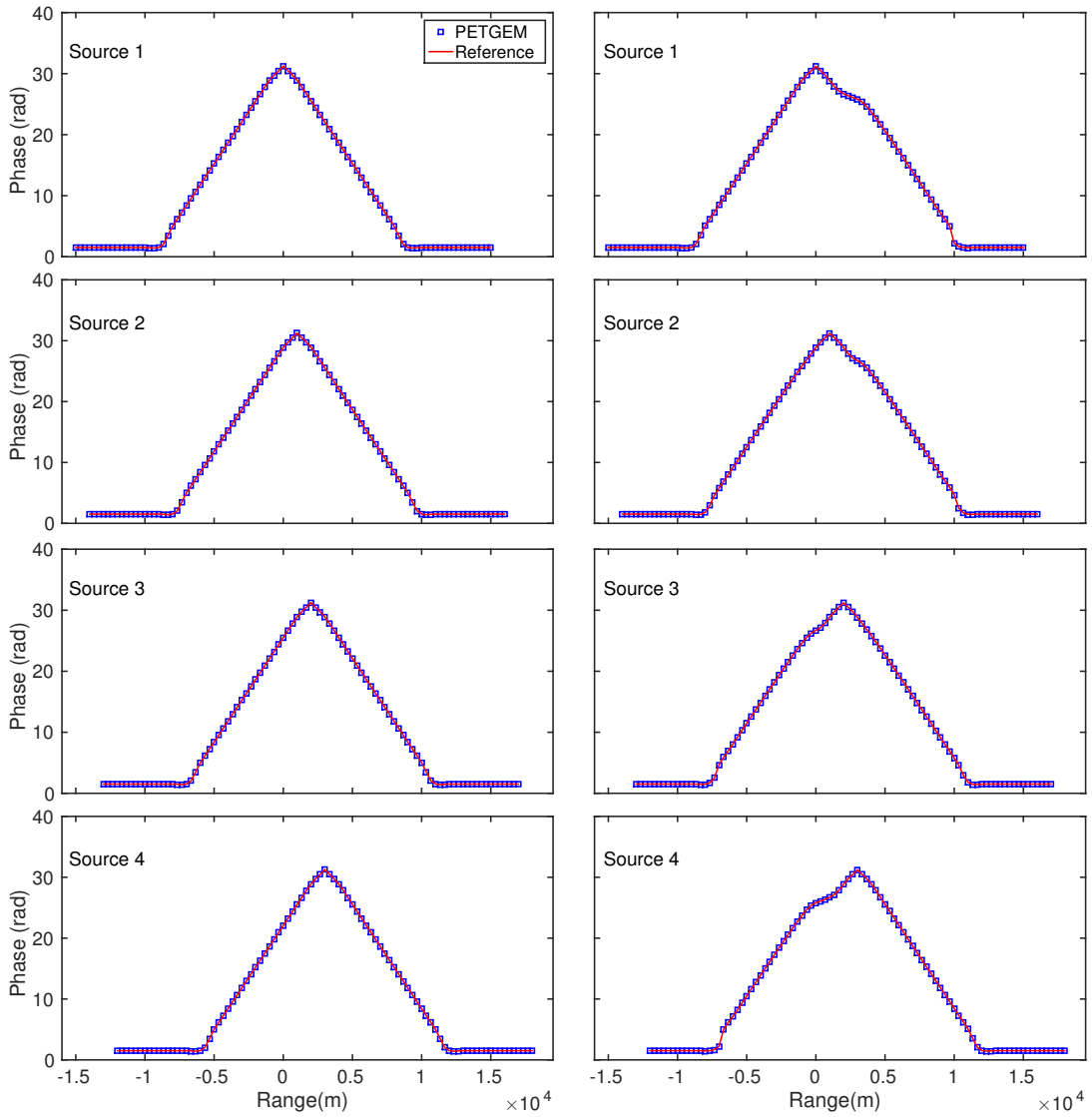


Fig. 3.10 Phase comparison for $\mathbf{E}_x$ between *PETGEM* and those obtained by BSIT. Left panels correspond to the model without target while the right panels correspond to the model with reservoir.

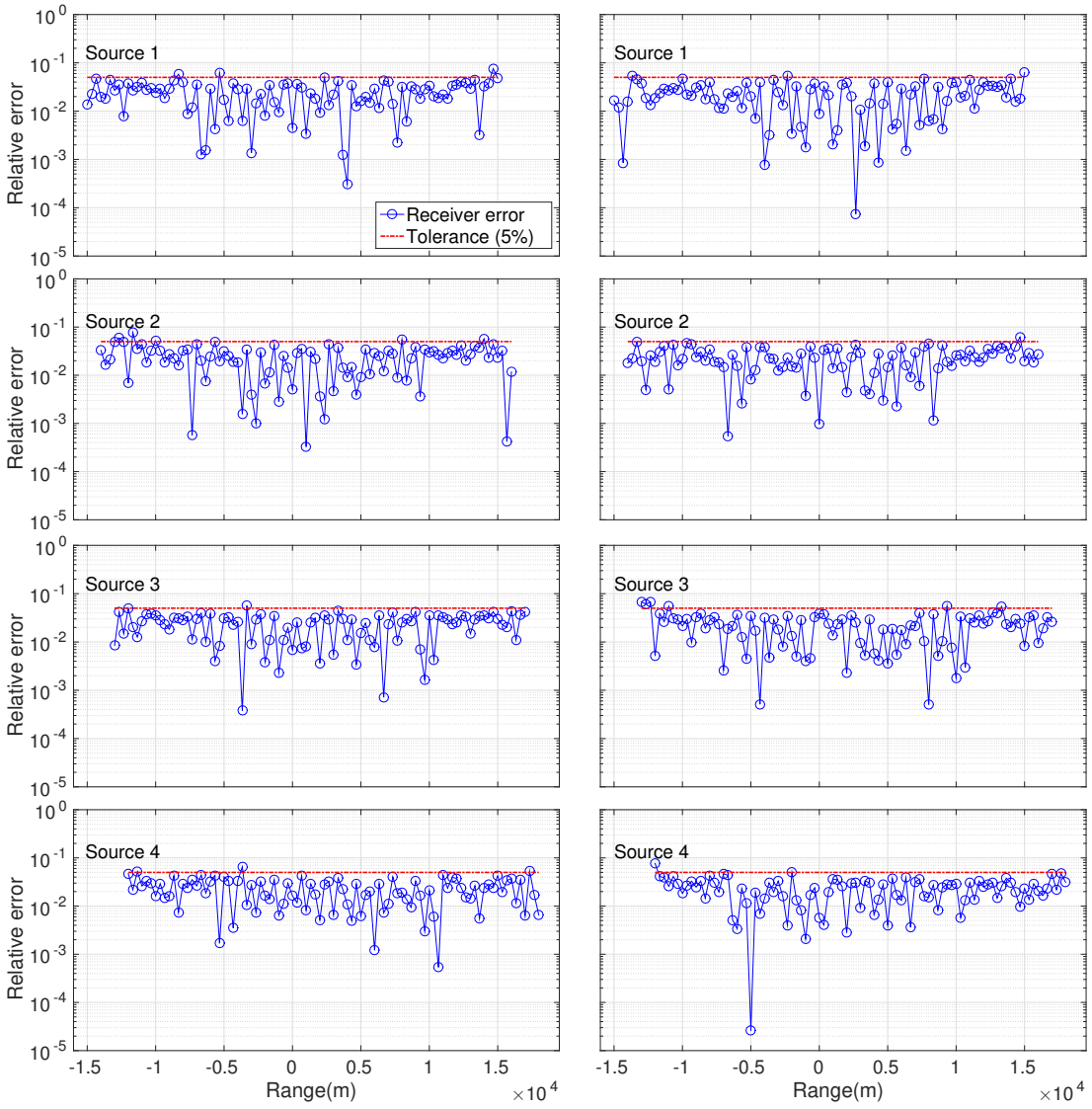


Fig. 3.11 Relative amplitude errors (left panels for model without target, rigth panels for model with reservoir) for EM responses shown in figure 3.9.

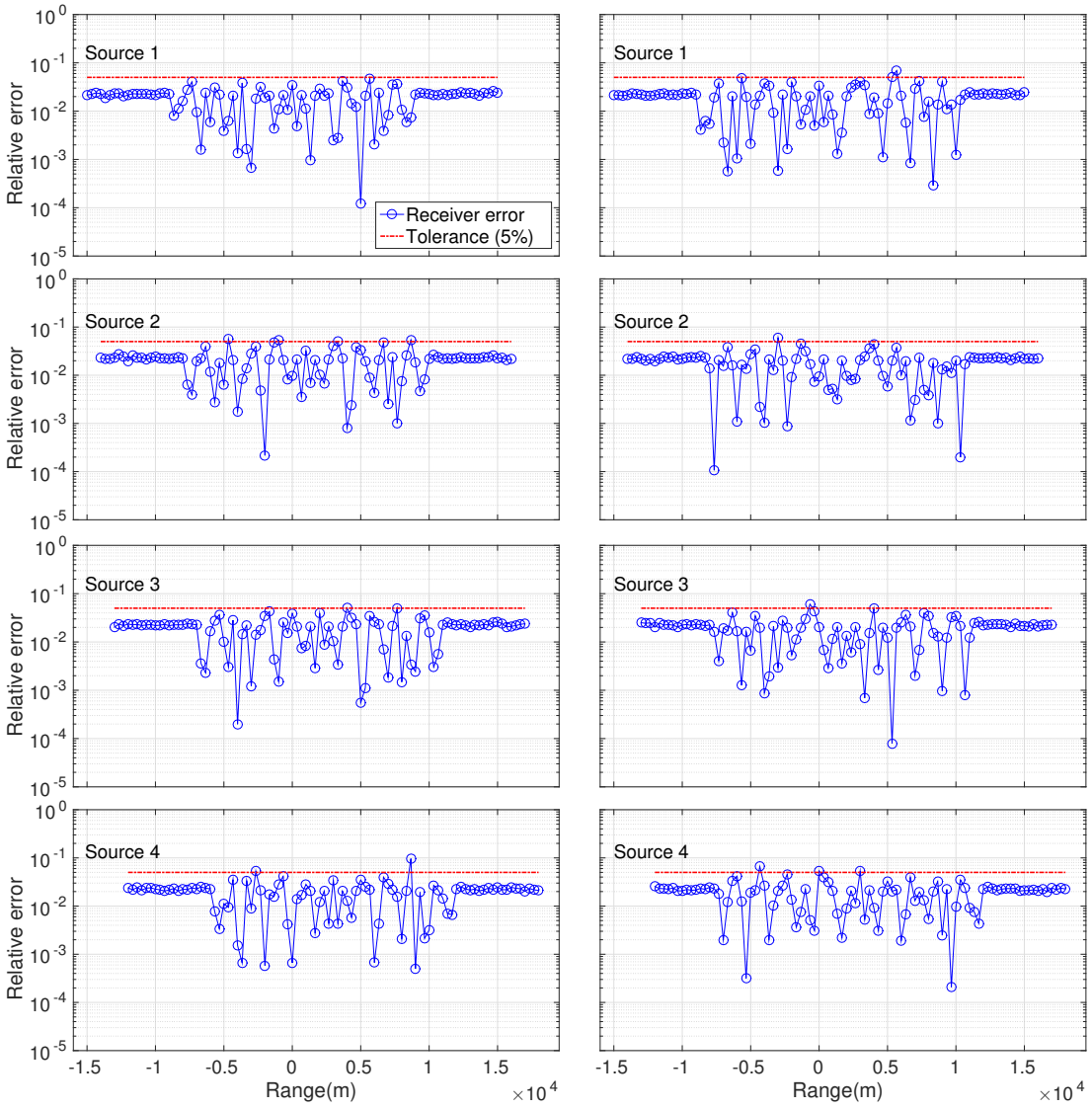


Fig. 3.12 Relative phase errors (left panels for model without target, rigth panels for model with reservoir) for electric fields shown in figure 3.14.

3.4 Automatic mesh adaptation

Nowadays, in the field of numerical simulations based on FEM and EFEM, automatic mesh adaptation has largely proved its efficiency for improving the accuracy of the numerical solution and capturing the behavior of physical phenomena by exploiting local mesh refinement. In principle, this technique allows substantially reducing the number of DOFs, thus favorably impacting CPU times, and achieving a desired accuracy on computed solutions. Although the iterative solver is rather efficient even on oversampled grid simulations, memory requirements can be reduced if the computational grid is adapted to the source location and to the frequency. However, when the source and receivers depth and the bathymetry are varying, it will be too cumbersome and impractical to ask the user to define a model per source/receivers and per frequency.

This test is devoted to the analysis of the automatic mesh adaptation approach developed by (Plessix et al., 2007). This approach ensures, for a given frequency and a given source position, that the computational domain is consistent with the discretization of the EM equations. Its core is based on the skin-depth (δ), defined as the effective depth of penetration of EM energy in a conducting medium, where the amplitude of a plane wave in a whole space has been attenuated to $1/e$ or 37% (Sheriff, 2002). Its formal definition is the following

$$\delta = \sqrt{\frac{2}{\mu_0 \omega \sigma}} \approx 503 \sqrt{\frac{1}{f\sigma}}, \tag{3.1}$$

where μ_0 is the free space magnetic permeability (Hm^{-1}), ω is the angular frequency ($2\pi f$), σ is the electric conductivity (Sm^{-1}) and f is the frequency (Hz). According to formulation by (Plessix et al., 2007), equation 3.1 gives a rule to automatically determine the spacing $d_\delta(f)$ at a frequency f

$$d_\delta(f) = \frac{\delta_{min}(f)}{r_\delta}, \tag{3.2}$$

where δ_{min} is the minimum skin-depth and r_δ is a number between two and three. In 3D CSEM FM surveys, δ_{min} occurs in the water layer where $\sigma \approx 3.3\ Sm^{-1}$ and $\delta_{min} \approx 275/\sqrt{f}$. In order to obtain better approximations around source and receivers, we define the spacing d_s as follows

$$d_s = min\left(\frac{L_s}{r_s}, d_\delta(f)\right), \tag{3.3}$$

where L_s is the source dipole length and r_s a number between ten and fifteen. The value for r_s is different to those described in (Plessix et al., 2007) (between two and four) because the authors used a finite-integration approach. However, in the test described below, we observe that this difference does not imply a significant increase in the computational cost.

We estimate the mesh dimensions from the average skin-depth δ_{ave}, that generally corresponds to the skin-depth in the sediment areas, i.e., $\delta_{ave} \approx 5 \times 10^3/\sqrt{f}$ for a conductivity of 0.01 (Sm^{-1}). The computational domain, decomposed into a core domain and extra boundary layers, was defined as follows

$$[x_s - r_x\delta_{ave}, x_s + r_x\delta_{ave}] \times [y_s - r_y\delta_{ave}, y_s + r_y\delta_{ave}] \times [z_{air}, z_s + r_z\delta_{ave}], \tag{3.4}$$

where r_x and r_y are numbers between four and eight (depending on the location of the receivers), r_z a number around four, z_{air} the depth of the air-water interface, and $x_s = [x_s, y_s, z_s]$ the source position. In order to reduce boundary reflections (as can be seen in Section 3.1 and Section 3.2), we add extra boundary layers with a thickness of $t_b = r_b\delta_{ave}$, where r_b is a number around four. The values for r_x, r_y, r_z and r_b where chosen in the same way as in (Plessix et al., 2007), where reflections of EM fields are reduced by 98% over four skin-depths, and by 99.9% over eight skin-depths.

For this test, the core of the computational domain is centered at x_s and the number of points is limited by the following power-law stretching

$$x_i = x_{i-1} + min(s_c^i d_s, d_\delta(f)), \tag{3.5}$$

where s_c is the stretching parameter equal to 1.04. Similarly, the boundary layers were stretched with a power-law defined as

$$x_i = x_{i-1} + s_b(x_{i-1} - x_{i-2}), \tag{3.6}$$

where s_b is equal to 1.1. Finally, we have defined a constant conductivity value for each element of the computational domain.

To evaluate whether the aforementioned approach is satisfactory to model 3D CSEM FM surveys, we carried out several *PETGEM* simulations based on the model described in Section 3.1 for the following frequencies: 0.25 *Hz*, 0.5 *Hz*, 0.75 *Hz*, 1 *Hz*, 1.25 *Hz*, 1.5 *Hz*, 1.75 *Hz* and 2 *Hz*. For each frequency, the computational mesh generation with *Gmsh* was controlled by $r_\delta = 3$, $r_s = 13$, $r_x = r_y = 8$, $r_z = 4$ and $r_b = 4$. A summary of the resulting meshes from this process is described in table 3.2.

All tests have been solved in sequential using the Symmetric Quasi-Minimal Residual (SQMR) solver with a Successive Over-relaxation (SOR) method as preconditioner. After executing each test separately (adapted tests) and aiming to investigate the ef-

Id	Frequency (Hz)	Elements	Nodes	Edges	DOFs
A	0.25	554, 262	92, 397	656, 186	627, 602
B	0.5	982, 094	163, 705	1, 163, 108	1, 111, 178
C	0.75	1, 463, 445	243, 729	1, 733, 203	1, 655, 113
D	1	1, 962, 870	325, 916	2, 322, 275	2, 221, 805
E	1.25	2, 447, 342	405, 880	2, 894, 085	2, 771, 493
F	1.5	2, 960, 034	489, 757	3, 497, 666	3, 354, 038
G	1.75	3, 576, 958	590, 519	4, 222, 968	4, 056, 492
H	2	4, 310, 303	709, 184	5, 082, 432	4, 893, 594

Table 3.2 Summary of resulting meshes based on automatic mesh adaptation

fect of an oversampled mesh, we solved each frequency using the mesh adapted to 2 Hz (oversampled tests).

Figures 3.13 and 3.14 compares the amplitude and phase components of electric fields between *PETGEM* and WHAM tool (Key, 2009). Here both results are better than those presented in Section 3.1, mainly on receivers whose position is close to the boundaries thanks to the inclusion of extra layers. However simulation results with the oversampled mesh show higher differences with respect to reference. Since these discrepancies have greater presence in the boundaries and around to source position, this negative effect is related to the quality of the elements on such mesh regions. The relative differences in amplitudes and phases of the responses obtained on both meshes, adapted and oversampled, are show in figures 3.15 and 3.16, respectively. The best results are obtained when the automatic mesh adaptation is carried out, e.g. the errors associated to modelling results at 2 Hz are smaller than their similar ones show in figures 3.3a and 3.3b. This numerical examples depict the usefulness of this approach for 3D CSEM FM design and scenario studies.

To illustrate the efficiency of this technique, we have measured the times for the assembly and solving tasks. Table 3.3 lists the number of iterations, assembly and solver times. Those results are obtained on a single CPU. The solution of tests based on oversampled meshes required much more iterations than tests with adapted meshes, as consequence needed considerably more CPU time. On average, the adapted scheme was about four times faster in these examples. In figure 3.17, the assembly and solver times for both set of tests are displayed.

Previous results show the relevance of mesh adaptation for survey design in the 3D CSEM FM context. The responses are very similar, the amplitude differences are a few percent, and the phase differences are around 0.4 rad. Notice that the results shown here are a direct consequence of using unstructured tetrahedral meshes, which can be generated fully automatically given a proper geometry and spacing rules. This is different from conforming hexahedral grids that often require manual interaction to properly honor arbitrary spacings and geometries (Owen, 1998). Furthermore, the computational time can be significantly reduced with an adapted mesh to the frequency and to the source position. In summary, this illustrates that a code with a careful mesh adaptation, included in *PETGEM*, is a competitive tool for EM modelling in the geophysics context.

Frequency	Assembly		Solver		Iterations	
(Hz)	A	B	A	B	A	B
0.25	1.77	11.8	17.52	222.4	5,950	11,382
0.5	2.99	11.9	35.78	222.3	6,630	12,519
0.75	4.49	11.8	45.35	213.7	5,625	9,685
1	5.72	11.6	91.70	206.7	8,245	9,275
1.25	6.95	11.8	95.70	203.4	6,825	8,670
1.5	8.32	11.6	104.71	174.9	6,145	7,540
1.75	10.38	11.8	137.02	161.1	7,290	7,230
2	12.52	11.8	155.76	153.6	6,935	6,935

Table 3.3 Summary of results for mesh adapted tests (A) and oversampled mesh tests (B). Times are expressed in minutes.

3.5 Convergence of solvers

In this section, we study the iterative solvers that were widely used for solving the EM problem under consideration (Freund and Nachtigal, 1991; Axelsson, 1996; Badea et al., 2001; Saad, 2003; Puzyrev et al., 2013; Cai et al., 2014; Koldan et al., 2014). Although there are other techniques, such as domain decomposition methods or geometric multigrid solvers, the purpose of these experiments focuses on exploiting the flexibility and simplicity of the solvers commonly used by researchers. Therefore, in order to determine which one is the best choice, the analysis include the *PETSc* implementation of the GMRES and BiCGSTAB methods.

These solvers can compete in terms of computational cost, convergence rate and

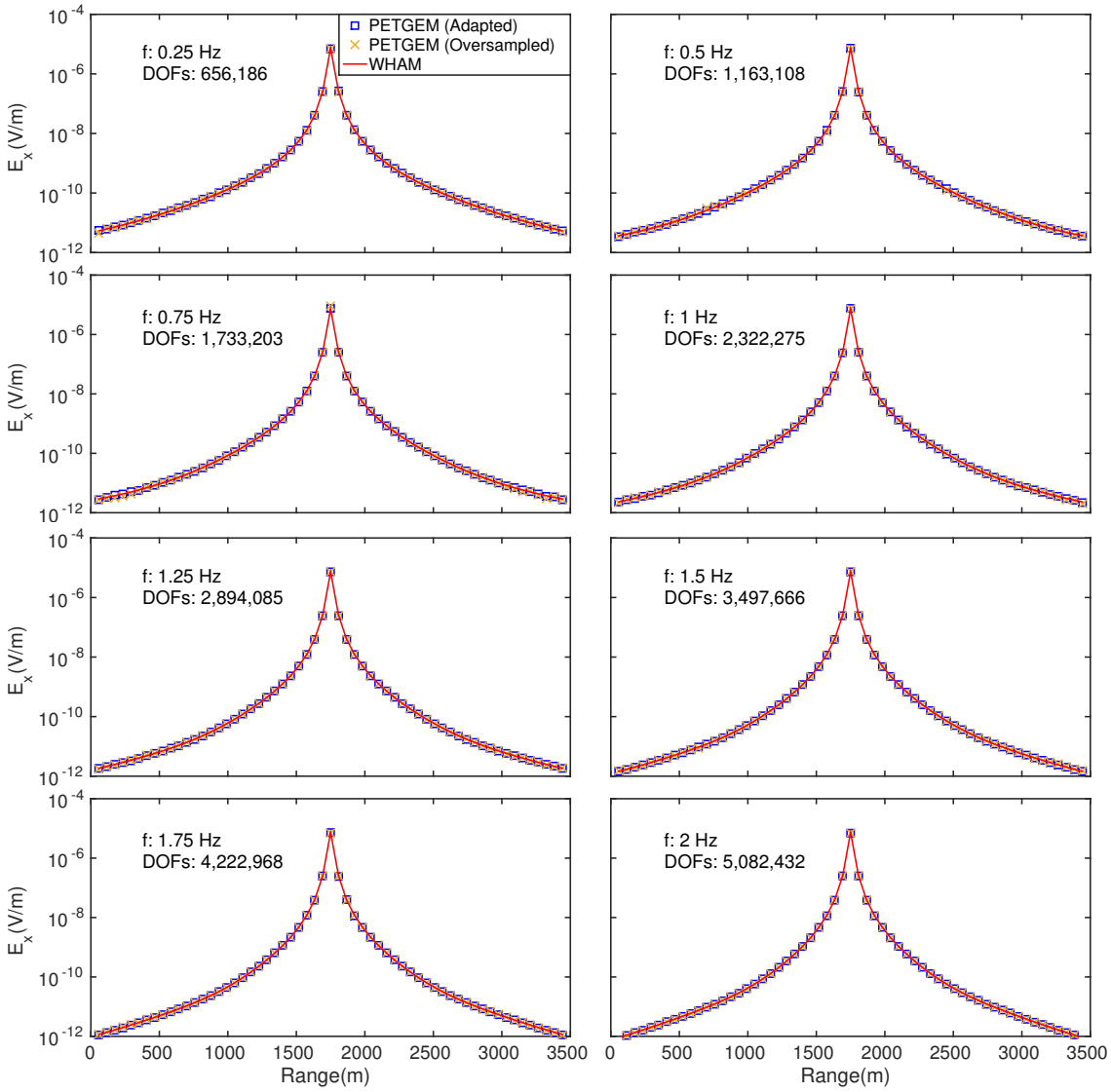


Fig. 3.13 Amplitude comparison for $\mathbf{E}_x$ between *PETGEM* and those obtained by WHAM tool (Key, 2009)

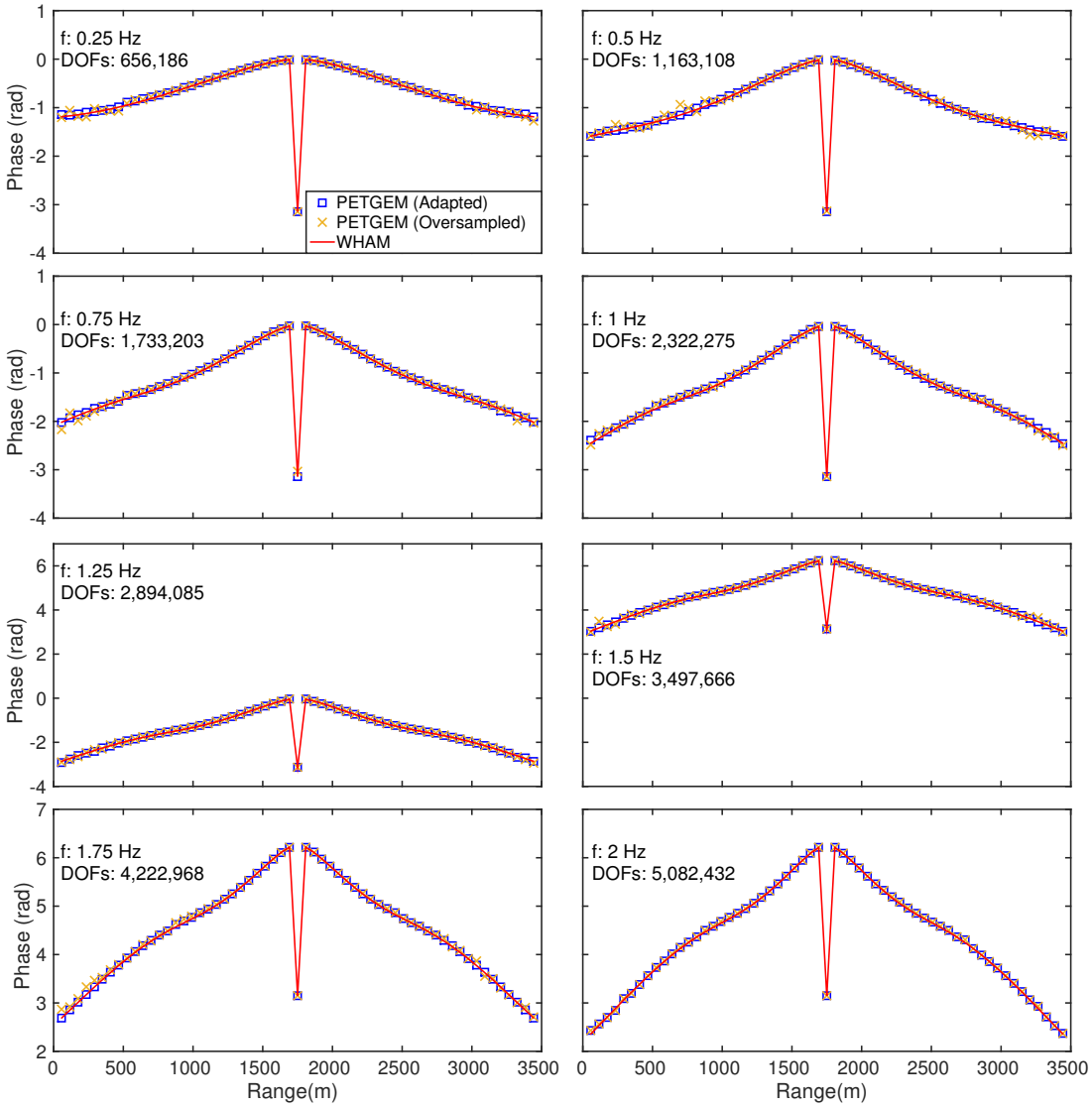


Fig. 3.14 Phase comparison for $\mathbf{E}_x$ between *PETGEM* and those obtained by WHAM tool (Key, 2009)

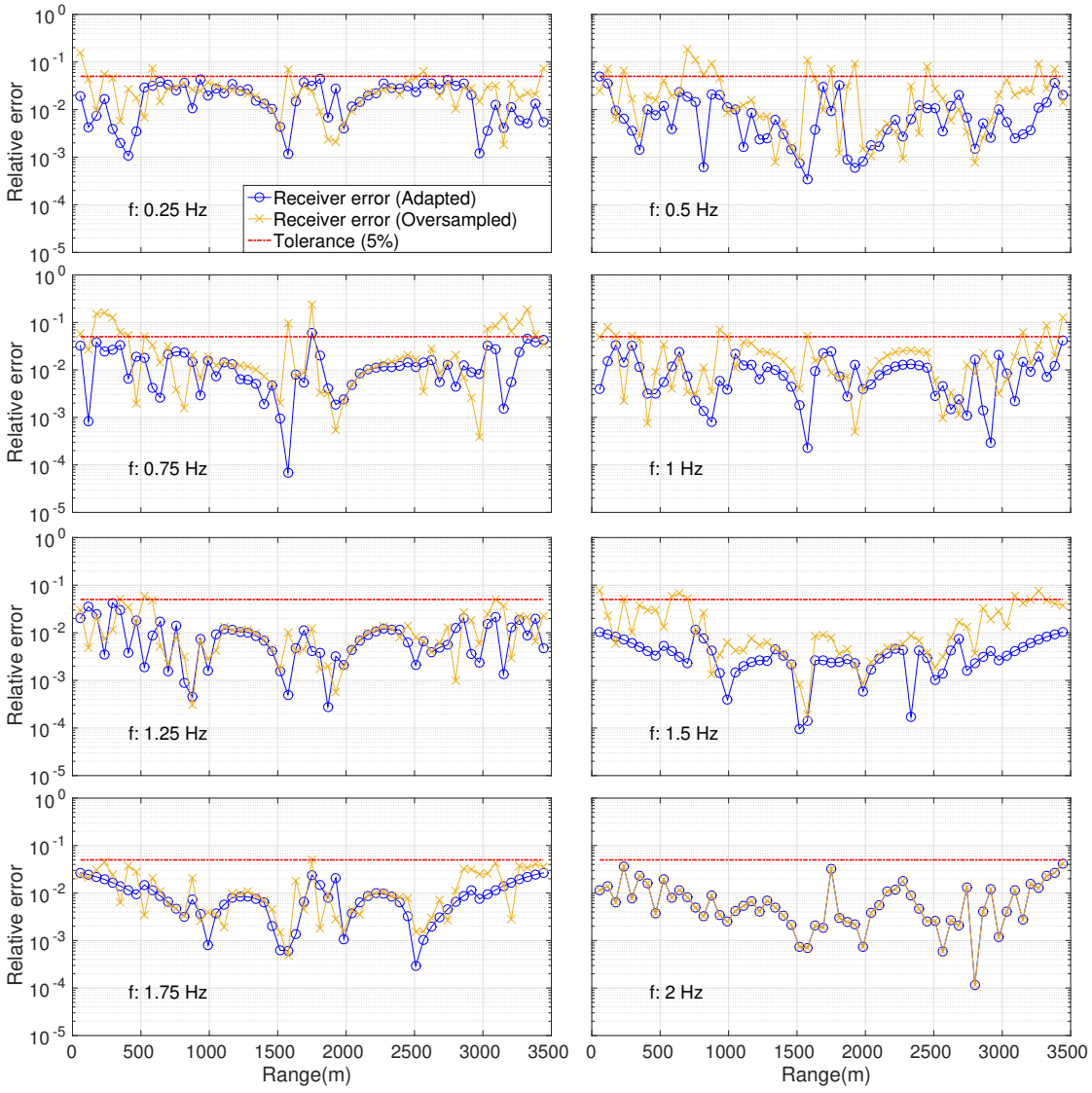


Fig. 3.15 Relative amplitude errors for EM responses show in figure 3.13

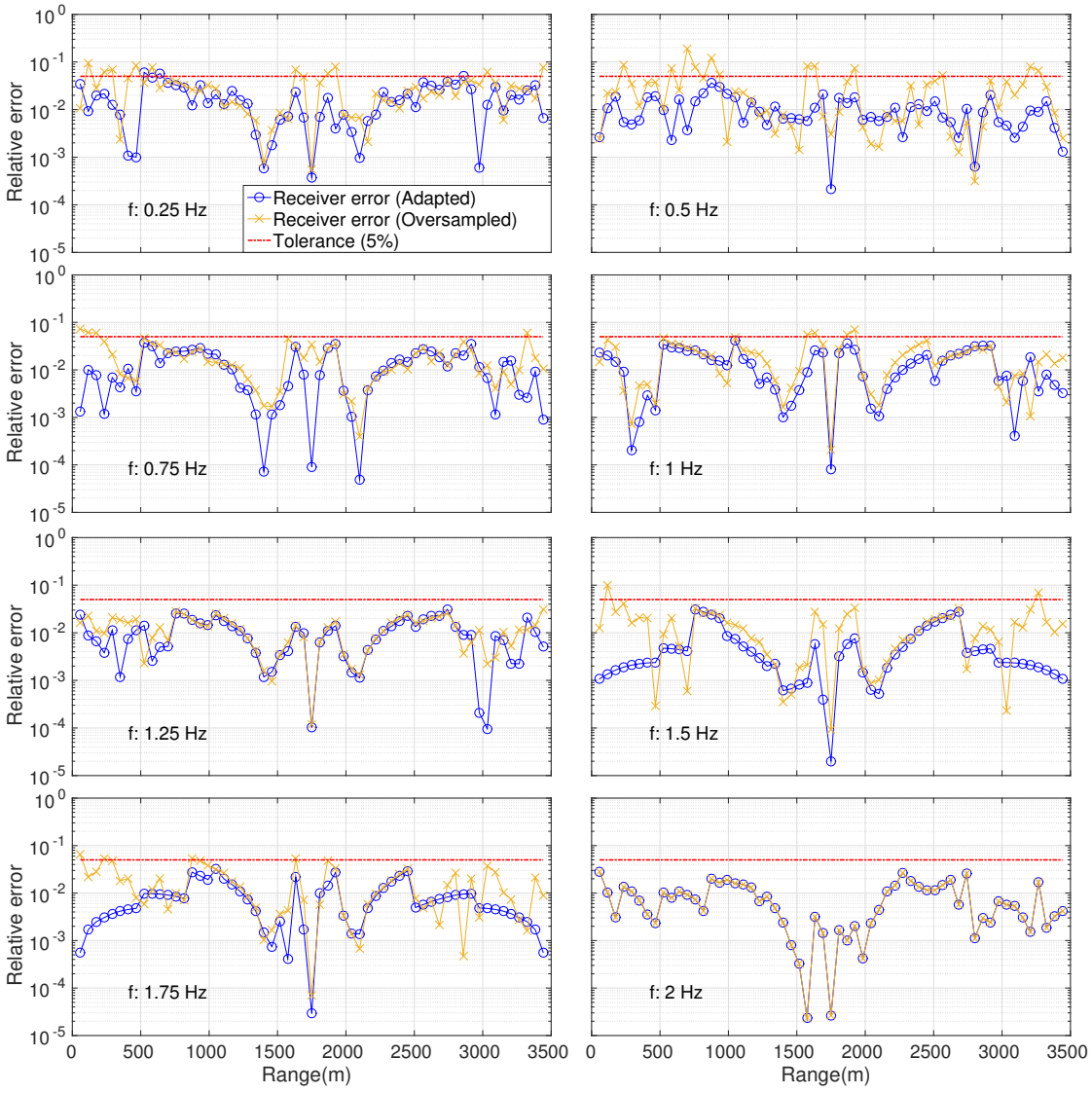


Fig. 3.16 Relative phase errors for EM responses show in figure 3.14

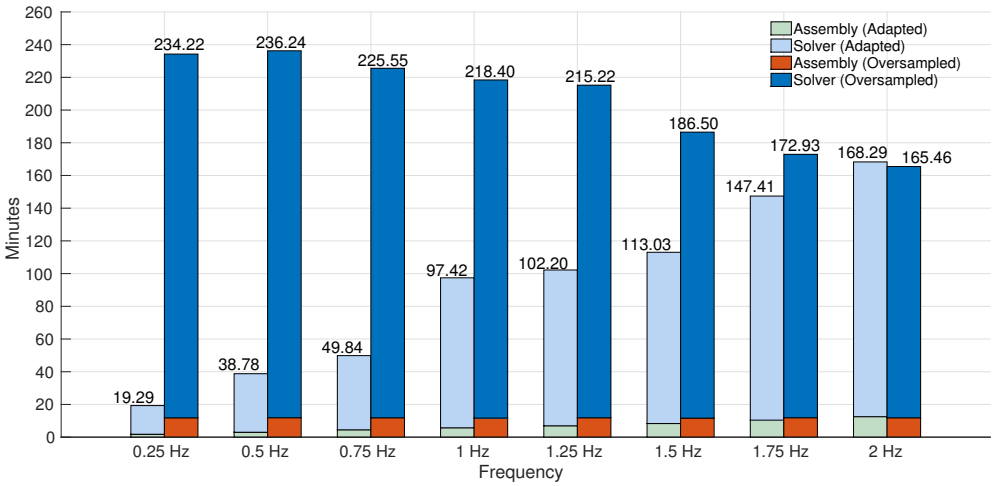


Fig. 3.17 Times comparison between adapted and oversampled meshes. Assembly and solving times are considered.

memory requirements. GMRES is an Arnoldi-based approach which only requires one matrix-vector multiplication for every iteration. However, the memory demands is large because it needs all the previously generated Arnoldi vectors to be saved (Puzyrev et al., 2013). BiCGSTAB is a Lanczos-based method that require two matrix-vector multiplications in every iteration, but the memory requirements for this approach is less than GMRES (Puzyrev et al., 2013). Furthermore, the convergence rate of Krylov subspace based iterative solvers strongly depends on the condition number of the matrix. A preconditioning technique can reduce the condition number of the matrix, thus the computation time for solving the linear system of equations can be scaled down. There are many options of preconditioners but among these, the Jacobian and SOR are the simplest ones because do not require extra computation (Axelsson, 1996). More advanced preconditioners such as Multigrid methods or Domain decomposition methods can be used to speed up the solvers. However, in these experiments we have adopted the Jacobian and SOR methods as preconditioners for simplicity and because they provided an adequate result for demonstration of the 3D CSEM FM proposed in this thesis.

To perform the convergence tests, we have used the 3D CSEM model presented in Section 3.3 (thick air: 1e-6 Sm^{-1}, thick seawater: 3.3 Sm^{-1}, reservoir: 1e-2 Sm^{-1}, thick sediments: 1 Sm^{-1}), and the mesh creation tecnique described in Section 3.4. Four frequencies have been studied, namely 0.5 Hz, 1 Hz, 2 Hz and 3 Hz. Table 3.4 provides quantitative information about the tetrahedal meshes that were generated for these tests. In all executions, the convergence criterion was a reduction of the relative residual norm (rtol) to a value in the order of 10^{-8}. The absolute tolerance (atol)

and divergence tolerance (dtol) values have been setted to 10^{-50} and 10^{5} respectively, meanwhile the number of iterations (maxits) has been limited by the maximum value of 10^{4}. Also, all tests have been carried out using 128 MPI tasks. Figure 3.18 shows

Frequency (Hz)	Elements	Nodes	Edges	DOFs
0.5	982,094	163,705	1,163,108	1,111,178
1	1,962,870	325,916	2,322,275	2,221,805
2	4,310,303	709,184	5,082,432	4,893,594
3	6,871,137	1,407,238	9,524,183	9,193,021

Table 3.4 Summary of meshes for solver convergence tests

the residual norms generated by GMRES and BiCGSTAB with versus the iteration number. In order to compare its convergence behaviour, figure 3.18 depicts the norms of both no-preconditioned and preconditioned iterative solvers. Regarding residual norms without preconditiong, we can see in figure 3.18 that neither solver was able to achieve the desired accuracy (both for low and high frecuencies). Also, the overall convergence of results with preconditioning is faster and it can reach lower residual norms, however the BiCGSTAB results has more dramatic oscillations that those obtained with GMRES. Furthermore, the obtained convergence rate of both techniques for tests at 0.25 Hz is quite poor since neither methods can reach given precision of 10^{-8} within 10,000 iterations. Namely, due to big sizes of the computational domain, its discretisation and singular values of resulting system of equations (diagonal values are quite near to zero at low frequencies and when air-layer is considered), the condition number of the matrix is huge, which results in bad convergence rate. In addition, the achieved precision for tests at 1 Hz was 10^{-8} in after 10,000 iterations meanwhile for tests at 2 Hz was 10^{-10} in less than 10,000 iterations (GMRES-Jacobi: 9,685, GMRES-SOR: 6,543, BiCGSTAB-Jacobi: 10,000, BiCGSTAB-SOR: 7,306). Finally, tests at 3 Hz showed the best convergence rate with precision of 10^{-10} after less than 7,000 iterations (GMRES-Jacobi: 6,487, GMRES-SOR: 4,343, BiCGSTAB-Jacobi: 4,095, BiCGSTAB-SOR: 3,003). The differences in precision and number of iterations are quite common situation in practice because the 3D CSEM FM is frequency dependent. Table 3.5 lists solver times (minutes) for each test.

The results have show that BiCGSTAB produces residual norms that oscillate significantly and its values are practically equal than ones of GMRES. Furthermore, the SOR preconditioner greatly improves both the convergence and the execution time for all tested cases. Therefore, we suggest using GMRES in combination with

SOR for the solution of the EM problem under consideration. The 3D CSEM model under consideration includes high conductivity contrasts (thick air: 1e-6 Sm^{-1}, thick seawater: 3.3 Sm^{-1}, reservoir: 1e-2 Sm^{-1}, thick sediments: 1 Sm^{-1}). In the described examples, the system of equations that have been solved have between 1 and 9 million unknowns.

Frequency (Hz)	0.25	1	2	3
GMRES	19.89	24.93	35.94	54.14
GMRES-Jacobi	19.91	21.15	28.70	36.49
GMRES-SOR	20.64	20.05	26.31	41.27
BiCSTAB	16.73	19.23	38.08	27.17
BiCSTAB-Jacobi	14.88	22.21	31.26	40.38
BiCSTAB-SOR	18.77	21.78	29.53	38.27

Table 3.5 Summary of runtime for solver convergence tests. Times are expressed in minutes.

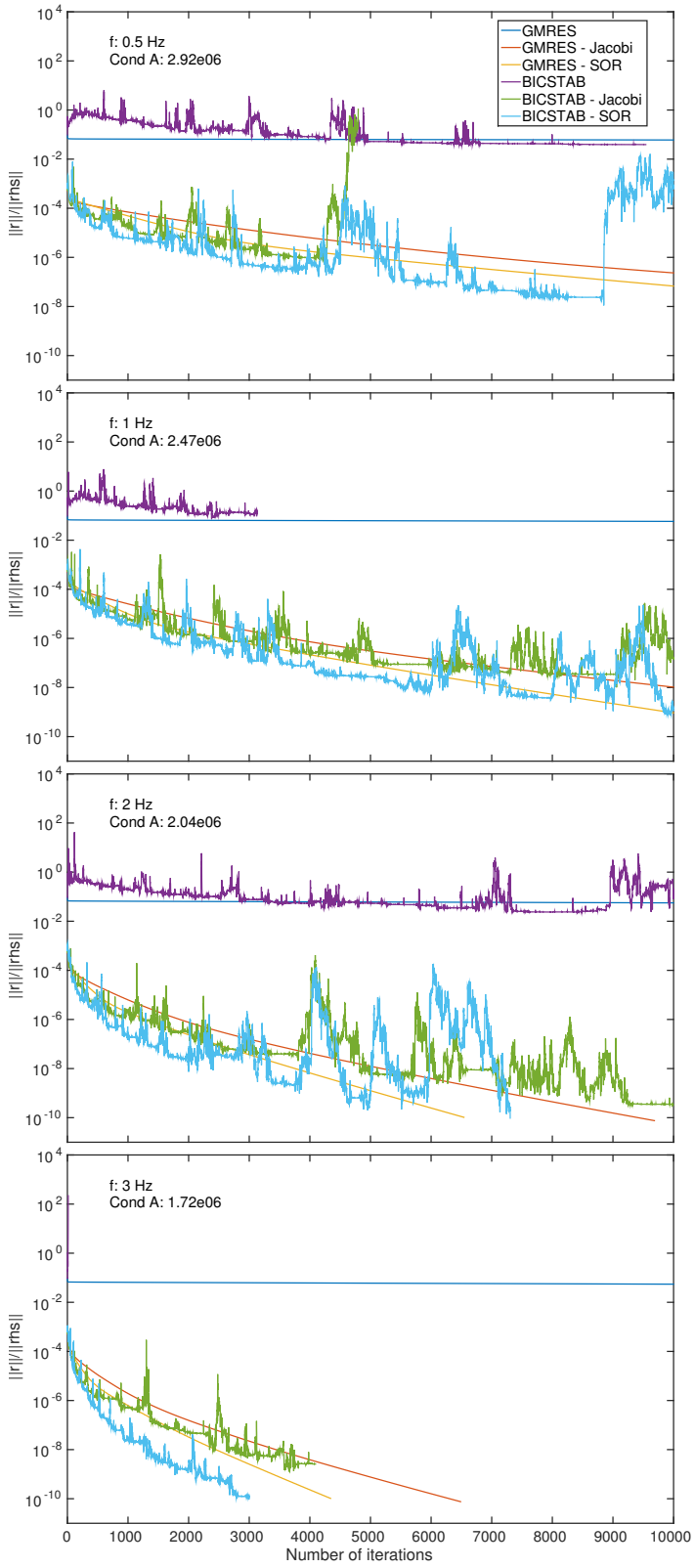


Fig. 3.18 Convergence rate of GMRES and BiCGSTAB solvers which have been preconditioned with the Jacobi and SOR methods for the meshes of table 3.4.

Chapter 4

Conclusions and future work

4.1 Conclusions

In this thesis we have developed a parallel python code for the 3D controlled-source electromagnetic forward modelling (3D CSEM FM) in geophysics using the edge finite element method (EFEM), namely, the Parallel Edge-based Tool for Geophysical Electromagnetic Modelling (*PETGEM*). The main design intent behind this code is reliable definition and characterization of bodies electric resistivity in the geophysical context within an acceptable runtime. This in turn can aid to conduct exploration campaigns with a significant reduction of costs and risks. There were three main themes addresed in this thesis.

The first theme is the modelling accuracy of currently used equations for the estimation of the electromagnetic (EM) signature of a given geological structure. Chapter 1 gave reasons why numerical simulation is a neccesary tool, compared to empirical or semiempirical approaches, for solving this problem. Furthermore, an overview of 3D CSEM FM (Section 1.1) and its state-of-art challenges were provided (Section 1.2). The employed EFEM formulation, in terms of primary/secondary field decomposition, has been developed and validated through numerical tests in Appendices B and C. Because EFEM divergence-free basis, this approach offers a good trade-off between accuracy and number of degrees of freedom (DOFs), i.e. size of the problem. Furthermore, EFEM supports unstructured meshes which is relevant if the geology is structurally complex. Thanks to this, the code has a very broad range of applicability for real 3D CSEM FM surveys.

Chapter 2 centered on the second theme, the computational implementation and its details. Nowadays, the role of this modelling tools in industry or academy is critical since they provide us synthetic results which we can then compare to real data. This

approach together to High-performance Computing (HPC) advances allow us to render natural phenomena treatable and testable. However, the tools that full fit needs for the solution of real models are commercial and often are inaccessible to the wider scientific community. On top of that, Chapter 2 is devoted to the development and documentation of *PETGEM* as a new open-source tool for the scalable solution of 3D CSEM FM on tetrahedral meshes, as these are the easiest to scale-up to very large domains or arbitrary shape. It is written mostly in Python 3.5.2 and relies on the scientific Python software stack with use of *mpi4py* and *petsc4py* packages for parallel computations. Other scientific Python packages used include: *H5py* for binary data format support, *Numpy* for efficient array manipulation and *Scipy* algorithms.

Although there are specialised modelling tools for geophysical prospecting such as the parallel codes developed by (Alumbaugh et al., 1996) and (Koldan, 2013), details of their implemented methods are generally hidden behind a black box, which could lead to a situation where the formulation could be unknown. Furthermore, not all numerical schemes are well suited for latest computing architectures or are well adapted to the problem. Aforementioned ideas are ones of the main drivers for developing the *PETGEM* code.

Many features have gradually been included, such as modules for EFEM data structures and a set of Python wrappers for the use of efficient solvers and preconditioners suitable for the resulting matrix system. *PETGEM* is now a complete package particularly suited for the 3D CSEM FM aiming to foster the understanding about EM in geophysics and its coupling with HPC technologies. Since it was intended tackle realistic problems, its data structure was designed to cope simultaneously three key requirements: accuracy, flexibility and efficiency. In addition, the adopted algorithms has the posibility to easily add or remove components without having to rewrite large parts of the code. This approach leads to optimal performance in terms of development and computation time without losing versatility offered by the programming language. In summary, Sections 2.1, 2.3 and 2.4 provided details about algorithms for EFEM, code workflow, software stack, target architectures, coding style, code availability and parallel Python strategies (shared-memory and distributed-memory architectures) within *PETGEM*. The Section 2.5 is dedicated to the study of the code scalability. For this purpose, the Paraver and Dimemas tools were used to identify the main computational phases and to collect the fundamental performance factors. The results obtained show an acceptable performance on both architectures.

The third theme is the verification stage of code testing to establish the reliability of the code programming and its numerical implementation. Three different cases of

3D CSEM FM were considered in Chapter 3. The first model, a canonical test of an off-shore hydrocarbon reservoir (Section 3.1), demonstrated the convergence behaviour of the numerical method and its good agreement with the reference solution. Here an effect of the imperfect absorbing boundaries was observed which can be mitigated by enlargening the domain with element sizes increasing logarithmically outwards from the zone of interest.

In the second model (Section 3.2), the code's capability to handle bathymetry has been tested. This model is especially interesting because it highlights one of the benefits of using an unstructured tetrahedral mesh, i.e. honoring complex geological structures without critically increasing the problem size. Furthermore, if not taken into account bathymetry effects can produce large anomalies on the measured electric fields. The air-wave effect when the water layer is shallow was also tested with this problem and shown to work well. For this test, the code was shown to provide solutions in good agreement with the reference.

The third model (Section 3.3) concerned simulations of a synthetic 3D CSEM model where the target (reservoir) is contained inside a sediments layer. In order to study the target effect, two set of simulations have been executed. First, by setting the conductivity of the target to that of the sediments, the EM responses to were computed. Next, the conductivity of the reservoir was setted and the EM responses for this model were calculated. As expected, the electric field is distorsed significant by reservoir. The main attraction of this complementary data (resistivity) is that, combined with other geophysical attributes like seismic information, can better de-risk the presence of high saturation of useful substances. Furthermore, the two solutions show a good overall agreement with the reference.

Additionally, an automatic mesh adaptation technique was considered (Section 3.4). This approach ensures, for a given frequency and a given source position, that the computational domain is consistent with the discretization of the EM equations. For this simulation it was observed that unnecessary and excessive refinement (oversampled) examples required much more iterations than tests with adapted meshes, as consequence needed considerably more CPU time. The adaptive mesh solutions had a factor of savings of up to four in time and storage compared to the uniform mesh result. Furthermore, the responses are very similar, the amplitude differences are a 5% percent, and the phase differences are around 0.4 rad, which showed a better agreement in comparison with previous numerical test depicted in Section 3.1.

The last experiments in Chapter 3 were focused on the study of parallel Krylov subspace solvers (Section 3.4) for matrices that arise due to the EFEM discretisation

of the EM problem that has been implemented. The tests, carried out to evaluate their convergence rate and required time, have show that the BiCGSTAB produces residual norms that oscillate significantly and its values are practically equal than ones of GMRES method. The experiments have also show that, for the problem under consideration, the SOR preconditioner greatly improves both the convergence rate and the execution time for all tested cases.

In summary, the conclusion is, what has been demonstrated with examples, that *PETGEM* is a competitive tool for EM modelling in the geophysics context. The code allow users the simulation of EM responses in real 3D CSEM FM on shared-memory/distributed-memory HPC platforms. The proposed modelling tool uses an approach based on the integration of 3D CSEM FM, EFEM, Python and parallel techniques, which is the first time this kind of methodology has been systematically applied for running simulations of this type on HPC architectures. As main properties of this tool are accuracy, modularity, efficiency and flexibility, that allow users an easy adaptation to real-life 3D CSEM FM cases.

4.2 Future directions

This thesis presents a new modelling tool for the solution of the 3D CSEM FM problem in geophysics, namely, *PETGEM*. The relevance of this development lies in its capacity to maps resistive bodies such as carbonates, hydrocarbon filled sediments, volvanic rocks and salt from the seabed. Particularly in offshore hydrocarbon exploration, data regarding resistivity mappings beneath the seafloor is crucial and useful, i.e. high resistivity of hydrocarbon filled rocks (30-500 Ωm) compared to bodies filled with saline formation water (0.5-2 Ωm) is usually a good indicator for the presence of oil and gas. Because its faculty to detect, identify and characterize the target reservoir, the 3D CSEM FM is an attractive and convincing method to conduct exploration campaigns, thus increasing the drilling success rate as well as reducing associated cost and hazards.

As consequence, in past 2 decades the modelling tools have become one of the pillars for the simulation of numerical methods which main goal is elucidating the fundamental mechanisms behind simplified abstractions of complex phenomena in different areas. The 3D CSEM FM in geophysics is no exception and the scientific community has developed important contributions in this field. Additionally, in the multi-core era, parallelization is a crucial issue especially if it fuels an inversion process which might involve over 100,000 realizations.

On top of that, *PETGEM* has been designed to cope with the various situations encountered within the numerical simulation of the 3D CSEM FM. Its was written based on an architecture-aware design effort in order to ensure a good capacity for large scale computations, thus competence to deal with real models. Furthermore, the software stack of this new tool has been composed to maximize simultaneously the key requirements: accuracy, tackle realistic problems with flexibility, efficiency and adaptability. In this thesis the code is provided as open-source so that it can be used, modified and redistributed freely with the aims of fostering reproducibility and encouraging investigations about 3D CSEM FM in geophysics and the HPC field.

The code validation process has led to a better insight into its accuracy and limitations and has helped identify a number of areas and challenges with the code that can be improved in the future. Some of them are trivial, i.e. the boundary conditions behaved as expected. Nevertheless, the following future lines could be addressed:

1. Experiments in this thesis showed that it is necessary the inclusion of more accurate boundary conditions in order to avoid the reflection effect close to domain boundaries. For this reason, it was decided to enlarging the domain with element sizes increasing logarithmically outwards from the zone of interest. However, adding better boundary conditions like PML, the robustness of the code can be increased and hence, better solutions can be obtained. Moreover, the use of high-order edge elements can also be explored. With these ideas in mind, a modular and flexible implementation have been developed.

2. The algorithms within *PETGEM* were adapted to work with an automatic mesh adaptation technique. Its is able to optimally ensures, for a given frequency and a given source position, that the computational domain is consistent with the discretization of the EM equations. However, it is important to take into account that a modification of the model representation may change the responses because this thesis only considered isotropic modelling. Taking advantage of this capability for multi-frequency, two related aims have been fixed. Firstly, study the behavior of the code when simulating anisotropic models. Secondly, determine the suitable parameter values for the anisotropic mesh generation.

3. The parallel implementation seems to perform well given the inherently serial and I/O portion of the code, but still could be more efficient. Further work might thus first focus on reducing the time of EFEM data structures computation without compromising the basic numerical methodology. It is possible because through this work has been concluded that there are not any restrictions in employing

parallel Python techniques for paralellisation of the presented EFEM solver. Furthermore, an advantage of Python or interpreted languages over compiled languages lies in the fact that it is much easier to make changes and test those modifications in a rapid way. These considerations combined with the always increasing performances of the computers and the mathematical modelling allow to envisage the simulation of more complex 3D CSEM FM cases.

4. The analysis of the *PETGEM* possibility to exploit the computing power provided by GPU cards is an interesting next development step. Currently, the most efficient distribution for GPU programming is the software library CUDA which has been developed by NVIDIA. Its provides PyCUDA as Python wrap. From an algorithmic point of view, this future research line offers a rich path, i.e. the need to create a scientific methodology for evaluating the algorithms performance proposed in this thesis in a different computational architecture.

Chapter 5

Papers from the thesis

Journals

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2017). PETGEM v1.0: Parallel code for 3D CSEM forward modelling in geophysics using edge finite elements. *Computers & Geosciences. Elsevier*, Submitted.

- Castillo-Reyes, O., de la Puente, Cela, J. M. (2017). Three-Dimensional CSEM modelling on unstructured tetrahedral meshes using edge finite elements. In: Barrios Hernández C., Gitler I., Klapp J. (eds) High Performance Computing. CARLA 2016. Communications in Computer and Information Science, Vol. 697: 247-256. ISBN 978-3-319-57971-9, Springer, Cham.

- Castillo-Reyes, O., de la Puente, J., Modesto, D., Puzyrev, V., and Cela, J. M. (2016). Parallel tool for numerical approximation of 3D electromagnetic surveys in geophysics. In *Computación y Sistemas, Thematic issue: Topic Trends in Computing Research in Catalonia*, Vol. 20, No.1, pp. 29-39, ISSN 2007-9737. National Polytechnic Institute. Mexico, D.F.

Proceedings

- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M. (2016). Edge-based parallel framework for the simulation of 3D CSEM surveys. In *ICE - Barcelona: International Conference & Exhibition 2016*. ISSN-ISSN: 2159-6832. AAPG-SEG. Barcelona, Spain.

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2016). Improving edge

finite element assembly for geophysical electromagnetic modelling on shared-memory architectures. In *Proceedings of the 7th Annual Ubiquitous Computing, Electronics Mobile Communication Conference*. ISBN 978-1-5090-1496-5. IEEE. New York, USA.

- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M. (2015). Edge-based electric field formulation in 3D CSEM simulations: a parallel approach. In *Proceedings of the 6th International Conference and Workshop on Computing and Communication*. ISBN 978-1-4799-6908-1. IEEE. Vancouver, Canada.

- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M. (2015). Parallel and numerical issues of the edge finite element method for 3D controlled-source electromagnetic surveys. In *Proceedings of the International Conference on Computing Systems and Telematics*. ISBN 978-1-4799-7639-3. IEEE. Xalapa, Veracruz, Mexico.

- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M. (2015). Assessment of edge-based finite element technique for geophysical electromagnetic problems: efficiency, accuracy and reliability. In *Proceedings of the 1st Pan-American Congress on Computational Mechanics and XI Argentine Congress on Computational Mechanics*, Vol. 1, No. 1, pp. 984-995, ISBN 978-84-943928-2-5. CIMNE, Buenos Aires, Argentine.

Conferences

- Queralt, P., Ledo, J., Marcuello, A., Castillo-Reyes, O., Modesto, D. (2017). Overview of numerical techniques and applications for CSEM/MT geophysical surveys. In *SIAM Conference on Mathematical and Computational Issues in the Geosciences*. Friedrich-Alexander University Erlangen-Nürnberg. Erlangen, Germany.

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2017). PETGEM: potential of 3D CSEM modelling using a new HPC tool for exploration geophysics. In *10th International Marine Electromagnetics conference - MARELEC*. University of Liverpool. Liverpool, England.

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2017). Python for HPC geophysical applications. In *GeoPython 2017*. Institute of Geomatics Engineer-

ing at University of Applied Sciences and Arts Northwestern Switzerland. Basel, Switzerland.

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2017). PETGEM: Parallel Edge-based Tool for Geophysical Electromagnetic Modelling. In *Congreso de Métodos Numéricos en Ingeniería - CMN*. Technical University of Valencia. Valencia, Spain.

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2017). Python code for CSEM modelling in geophysics and HPC architectures: advances and challenges. In *Innovation Match MX 2016-2017*. Innovation Match Association. Mexico, D.F.

- Castillo-Reyes, O. (2017). See underneath. High performance computing, geophysics and electromagnetic methods. In *Interdisciplinary Meeting of Predoctoral Researchers - JIPI 2017*. University of Barcelona. Barcelona, Spain.

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2017). Python for HPC geophysical electromagnetic applications: experiences and perspectives. In *4th BSC Doctoral Symposium*. Barcelona Supercomputing Center. Barcelona, Spain.

- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M. (2016). Parallel and vectorized code for CSEM surveys in geophysics: An edge-based approach. In *European Congress on Computational Methods in Applied Sciences and Engineering - ECCOMAS*. Crete Island, Greece.

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2016). High performance computing, geophysics and numerical methods: a symbiotic relation. In *Innovation Match MX 2015-2016*. Innovation Match Association. Guadalajara, Mexico.

- Castillo-Reyes, O., de la Puente, J., Barucq, H., Diaz, J., and Cela, J. M. (2016). Edge-based parallel code for CSEM surveys in geophysics: performance and accuracy improvements. In *WCCM XII and APCOM VI*. Korean Society for Computational Mechanics and International Association for Computational Mechanics. Seoul, Republic of Korea.

- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M. (2015). HPC and edge elements for geophysical electromagnetic problems: an overview. In

2nd BSC Doctoral Symposium. Barcelona Supercomputing Center. Barcelona, Spain.

- Castillo-Reyes, O. (2014). Soluciones HPC para el sector energético: Desafíos y oportunidades. In *IV Simposio Becarios CONACyT en Europa*. Casa Universitaria Franco-Mexicana - Consejo Nacional de Ciencia y Tecnología de México. Strasbourg, France.

Workshops

- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2017). High performance computing using Python: advances in geophysical electromagnetic modelling. In *Computing and Electromagnetics International Workshop - CEM*. Polytechnic University of Catalonia. Barcelona, Spain.
- Castillo-Reyes, O. (2016). Towards an HPC tool for 3D CSEM forward modelling in geophysics. In Fourth International Congress on Multiphysics, Multiscale, and Optimization problems. GEAGAM Network. Bilbao, Spain.
- Castillo-Reyes, O. (2016). Supercómputo y geofísica electromagnética: avances y desafíos. In *Seminarios internos de Ciencias de la Tierra y Medio Ambiente*. Centro de Ciencias de la Tierra. University of Veracruz. Xalapa, Veracruz, Mexico.
- Castillo-Reyes, O. (2016). Scientific computing on massively parallel computers. In *Master in Telematic Engineering*. School of Accounting and Management. University of Veracruz. Xalapa, Veracruz, Mexico.
- Castillo-Reyes, O., de la Puente, J., and Cela, J. M. (2016). Towards an HPC tool for simulation of 3D CSEM surveys: an edge-based approach. In *PRACE-days16*. Partnership for Advanced Computing in Europe. Czech Republic.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M. (2015). Edge-elements for geophysical electromagnetic problems: a new implementation challenge. In *PRACEdays15*. Partnership for Advanced Computing in Europe. Dublin, Ireland.
- Castillo-Reyes, O. (2014). Soluciones HPC en el campo de la geofísica. In *Seminarios internos de Ciencias de la Tierra y Medio Ambiente*. Centro de Ciencias de la Tierra. University of Veracruz. Xalapa, Veracruz, Mexico.

Papers from the thesis

- Castillo-Reyes, O. (2014). High performance computing, science and engineering. In *Master in Telematic Engineering*. School of Accounting and Management. University of Veracruz. Xalapa, Veracruz, Mexico.

References

Abubakar, A., Habashy, T., Druskin, V., Knizhnerman, L., and Alumbaugh, D. (2008). 2.5D forward and inverse modeling for interpreting low-frequency electromagnetic measurements. *Geophysics*, 73(4):F165–F177.

Ahrens, J., Geveci, B., and Law, C. (2005). Paraview: An end-user tool for large-data visualization. *The Visualization Handbook*, page 717.

Alnæs, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N. (2015). The fenics project version 1.5. *Archive of Numerical Software*, 3(100):9–23.

Alumbaugh, D. L., Newman, G. A., Prevost, L., and Shadid, J. N. (1996). Three-dimensional wideband electromagnetic modeling on massively parallel computers. *Radio Science*, 31(1):1–23.

Anjam, I. and Valdman, J. (2015). Fast MATLAB assembly of fem matrices in 2D and 3D: Edge elements. *Applied Mathematics and Computation*.

Axelsson, O. (1996). *Iterative solution methods*. Cambridge university press.

Badea, E. A., Everett, M. E., Newman, G. A., and Biro, O. (2001). Finite-element analysis of controlled-source electromagnetic induction using coulomb-gauged potentials. *Geophysics*, 66(3):786–799.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2016). PETSc Web page. http://www.mcs.anl.gov/petsc.

Barton, M. and Cendes, Z. (1987). New vector finite elements for three-dimensional magnetic field computation. *Journal of Applied Physics*, 61(8):3919–3921.

Beck, R. and Hiptmair, R. (1999). Multilevel solution of the time-harmonic maxwell's equations based on edge elements. *Internat. J. Numer. Methods Engrg.*, 45(7):901–920.

Berenger, J.-P. (1994). A perfectly matched layer for the absorption of electromagnetic waves. *Journal of computational physics*, 114(2):185–200.

Bespalov, A. et al. (2007). Simulation of electromagnetic well-logging tools by the nédélec edge finite elements. In *2007 SEG Annual Meeting*. Society of Exploration Geophysicists.

References

Bhogeswara, R. and Killough, J. (1994). Parallel linear solvers for reservoir simulation: A generic approach for existing and emerging computer architectures. *SPE Computer Applications (Society of Petroleum Engineers);(United States)*.

Biro, O., Preis, K., and Richter, K. R. (1996). On the use of the magnetic vector potential in the nodal and edge finite element analysis of 3D magnetostatic problems. *IEEE Transactions on Magnetics*, 32(3):651–654.

Bossavit, A. and Verite, J.-C. (1982). A mixed fem-biem method to solve 3-D eddy-current problems. *IEEE Transactions on Magnetics*, 18(2):431–435.

Burnett, D. (1987). *Finite element analysis: from concepts to applications*. Addison-Wesley Pub. Co., Massachusetts.

Cai, H., Hu, X., Li, J., Endo, M., and Xiong, B. (2017). Parallelized 3D CSEM modeling using edge-based finite element with total field formulation and unstructured mesh. *Computers & Geosciences*, 99:125–134.

Cai, H., Xiong, B., Han, M., and Zhdanov, M. (2014). 3D controlled-source electromagnetic modeling in anisotropic medium using edge-based finite element method. *Computers & Geosciences*, 73:164–176.

Cao, H., Tchelepi, H. A., Wallis, J. R., Yardumian, H. E., et al. (2005). Parallel scalable unstructured cpr-type linear solver for reservoir simulation. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers.

Chen, L. (2008). iFEM: an innovative finite element methods package in MATLAB. *Preprint, University of Maryland*.

Chen, L. (2011). Programming of finite element methods in MATLAB. *Preprint, University of California Irvine*.

Chew, W., Tong, M.-S., et al. (2008). *Integral equation methods for electromagnetic and elastic waves*. Morgan & Claypool Publishers.

Childs, H., Brugger, E., Bonnell, K., Meredith, J., Miller, M., Whitlock, B., and Max, N. (2005). A contract based system for large data visualization. In *Visualization, 2005. VIS 05. IEEE*, pages 191–198. IEEE.

Chrisochoides, N. (2006). Parallel mesh generation. In *Numerical solution of partial differential equations on parallel computers*, pages 237–264. Springer.

Chung, Y., Son, J.-S., Lee, T. J., Kim, H. J., and Shin, C. (2014). Three-dimensional modelling of controlled-source electromagnetic surveys using an edge finite-element method with a direct solver. *Geophysical Prospecting*, 62(6):1468–1483.

Cimrman, R. (2014). SfePy - write your own FE application. In *Proceedings of the 6th European Conference on python in Science (EuroSciPy 2013)*, pages 65–70. http://arxiv.org/abs/1404.6391.

Collins, D. A., Grabenstetter, J. E., Sammon, P. H., et al. (2003). A shared-memory parallel black-oil simulator with a parallel ilu linear solver. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers.

Constable, S. (2010). Ten years of marine CSEM for hydrocarbon exploration. *Geophysics*, 75(5):75A67–75A81.

Constable, S. and Srnka, L. J. (2007). An introduction to marine controlled-source electromagnetic methods for hydrocarbon exploration. *Geophysics*, 72(2):WA3–WA12.

Constable, S. and Weiss, C. J. (2006). Mapping thin resistors and hydrocarbons with marine EM methods: Insights from 1D modeling. *Geophysics*, 71(2):G43–G51.

Coscia, I., Greenhalg, S., Linde, N., Doetsch, J., Marescot, L., Günther, T., and Green, A. (2011). 3D crosshole apparent resistivity static inversion and monitoring of a coupled river-aquifer system. *Geophysics*, 76(2):G49–G59.

Crowley, C., Silvester, P., and Hurwitz Jr, H. (1988). Covariant projection elements for 3D vector field problems. *IEEE Transactions on Magnetics*, 24(1):397–400.

Cuni, A. (2010). *High performance implementation of python for CLI/.NET with JIT compiler generation for dynamic languages*. PhD thesis, Università di Genova.

da Silva, N. V., Morgan, J. V., MacGregor, L., and Warner, M. (2012). A finite element multifrontal method for 3D CSEM modeling in the frequency domain. *Geophysics*, 77(2):E101–E115.

Dalcín, L., Paz, R., Kler, P., and Cosimo, A. (2011). Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139.

Dalcín, L., Paz, R., and Storti, M. (2005). MPI for python. *Journal of Parallel and Distributed Computing*, 65(9):1108–1115.

Dalcín, L., Paz, R., Storti, M., and D'Elia, J. (2008). MPI for python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662.

Davidson, D. (2010). *Computational Electromagnetics for RF and Microwave Engineering*. Cambridge University Press.

Davydycheva, S., Druskin, V., and Habashy, T. (2003). An efficient finite-difference scheme for electromagnetic logging in 3D anisotropic inhomogeneous media. *Geophysics*, 68(5):1525–1536.

Davydycheva, S. and Rykhlinski, N. (2011). Focused-source electromagnetic survey versus standard CSEM: 3D modeling in complex geometries. *Geophysics*, 76(1):F27–F41.

Dogru, A., Sunaidi, H., Fung, L., Habiballah, W., Al-Zamel, N., Li, K., et al. (2002). A parallel reservoir simulator for large-scale reservoir simulation. *SPE Reservoir Evaluation & Engineering*, 5(01):11–23.

Dubois, F. (2016). Pyfort: The python-fortran connection tool.

Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). *Direct methods for sparse matrices*. Clarendon press Oxford.

References

Edwards, N. (2005). Marine controlled source electromagnetics: principles, methodologies, future commercial applications. *Surveys in Geophysics*, 26(6):675–700.

Eidesmo, T., Ellingsrud, S., MacGregor, L., Constable, S., Sinha, M., Johansen, S., Kong, F., and Westerdahl, H. (2002). Sea bed logging (SBL), a new method for remote and direct identification of hydrocarbon filled layers in deepwater areas. *First break*, 20(3):144–152.

Epov, M., Shurina, E., and Nechaev, O. (2007). 3D forward modeling of vector field for induction logging problems. *Russian Geology and Geophysics*, 48(9):770–774.

Everett, M. E. (2012). Theoretical developments in electromagnetic induction geophysics with selected applications in the near surface. *Surveys in geophysics*, 33(1):29–63.

Feng, X. (1999). Absorbing boundary conditions for electromagnetic wave propagation. *Math. Comp*, 68:145–168.

Feng, Y. (2016). Sharedmem: Dispatch your trivially parallizable jobs with python. https://pypi.python.org/pypi/sharedmem/0.3.

Filippone, S. and Colajanni, M. (2000). PSBLAS: A library for parallel linear algebra computation on sparse matrices. *ACM Transactions on Mathematical Software (TOMS)*, 26(4):527–550.

Fomenko, E. Y. and Mogi, T. (2002). A new computation method for a staggered grid of 3D EM field conservative modeling. *Earth, planets and space*, 54(5):499–509.

Franke, A., Börner, R.-U., and Spitzer, K. (2007). Adaptive unstructured grid finite element simulation of two-dimensional magnetotelluric fields for arbitrary surface and seafloor topography. *Geophysical Journal International*, 171(1):71–86.

Freund, R. W. and Nachtigal, N. M. (1991). QMR: a quasi-minimal residual method for non-hermitian linear systems. *Numerische mathematik*, 60(1):315–339.

Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. (2004). Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104.

Gajewski, A., Szczypa, S., and Wójcicki, A. (2005). Geophysical mapping for structural geology, prospecting and environment protection purposes. *Przeglad Geologiczny*, 53(10):141–146.

Geuzaine, C. and Remacle, J.-F. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331.

Gibson, P., Lyle, P., and George, D. M. (1996). Environmental applications of magnetometry profiling. *Environmental Geology*, 27(3):178–183.

Gill, P., Murray, W., and Wright, M. (1991). *Numerical linear algebra and optimization*. Number v. 1 in Numerical Linear Algebra and Optimization. Addison-Wesley Pub. Co., Advanced Book Program.

Gray, D. A., Majorowicz, J., and Unsworth, M. (2012). Investigation of the geothermal state of sedimentary basins using oil industry thermal data: case study from northern alberta exhibiting the need to systematically remove biased data. *Journal of Geophysics and Engineering*, 9(5):534.

Greenfield, P., Miller, T., White, R., and Hsu, J. (2016). Numarray: a new scientific array package for python.

Grund, F. (1999). *Direct linear solvers for vector and parallel computers*. Springer.

Hanif, N. H. H. M., Hussain, N., Yahya, N., Daud, H., Yahya, N., and Noh, M. (2011). 1D modeling of controlled-source electromagnetic (CSEM) data using finite element method for hydrocarbon detection in shallow water. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*.

Hannukainen, A. and Juntunen, M. (2012). Implementing the finite element assembly in interpreted languages. *Preprint, Aalto University, http://users. tkk. fi/˜ mojuntun/preprints/matvecSISC. pdf*.

Hano, M. (1984). Finite-element analysis of dielectric-loaded waveguides. *IEEE Transactions on Microwave Theory and Techniques*, 32(10):1275–1279.

Harrington, R. (1961). *Time-harmonic electromagnetic fields*. McGraw-Hill.

Heroux, M., Raghavan, P., and Simon, H. (2006). *Parallel Processing for Scientific Computing*. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104).

Hinsen, K. (2014). Scientific python: various python modules for scientific computing.

Hiptmair, R. (2015). Maxwell's equations: Continuous and discrete. Technical Report 2015-18, Seminar for Applied Mathematics, ETH Zürich. To appear in A. Bermdez de Castro, A. Valli (eds.), Computational Electromagnetism, Springer Lecture Notes in Mathematics 2148.

Ho-Le, K. (1988). Finite element mesh generation methods: a review and classification. *Computer-aided design*, 20(1):27–38.

Jamin, C., Alliez, P., Yvinec, M., and Boissonnat, J.-D. (2014). CGALmesh: A generic framework for delaunay mesh generation. *ACM Transactions on Mathematical Software*.

Jin, J. (2002). *The Finite Element Method in Electromagnetics*. Wiley, New York, second edition.

Jishan, H. and Lizhi, B. (1999). The situation and progress of marine electromagnetic method research. *Progress in Geophysics*, 1:001.

Johnson, H. M. (1962). A history of well logging. *Geophysics*, 27(4):507–527.

Kameari, A. (1988). Three dimensional eddy current calculation using edge elements for magnetic vector potential. *Appl. Electromagnetics in Materials*, pages 225–236.

Kearey, P., Klepeis, K., and Vine, F. (1996). *Global Tectonics*. Wiley-Blackwell, New York, third edition.

Ketcheson, D. I., Mandli, K. T., Ahmadia, A. J., Alghamdi, A., Quezada de Luna, M., Parsani, M., Knepley, M. G., and Emmett, M. (2012). PyClaw: Accessible, Extensible, Scalable Tools for Wave Propagation Problems. *SIAM Journal on Scientific Computing*, 34(4):C210–C231.

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

Key, K. and Ovall, J. (2011). A parallel goal-oriented adaptive finite element method for 2.5-D electromagnetic modelling. *Geophysical Journal International*, 186(1):137–154.

Key, K. and Weiss, C. (2006). Adaptive finite-element modeling using unstructured grids: The 2D magnetotelluric example. *Geophysics*, 71(6):G291–G299.

Knepley, M. G., Lange, M., and Gorman, G. J. (2015). Unstructured overlapping mesh distribution in parallel. *arXiv preprint arXiv:1506.06194*.

Koldan, J. (2013). *Numerical solution of 3-D electromagnetic problems in exploration geophysics and its implementation on massively parallel computers*. PhD thesis, Polytechnic University of Catalonia.

Koldan, J., Puzyrev, V., de la Puente, J., Houzeaux, G., and Cela, J. M. (2014). Algebraic multigrid preconditioning within parallel finite-element solvers for 3-D electromagnetic modelling problems in geophysics. *Geophysical J. Int.*, Accepted.

Kong, F. (2007). Hankel transform filters for dipole antenna radiation in a conductive medium. *Geophysical Prospecting*, 55(1):83–89.

Kong, F., Johnstad, S., Røsten, T., and Westerdahl, H. (2007). A 2.5D finite-element-modeling difference method for marine CSEM modeling in stratified anisotropic media. *Geophysics*, 73(1):F9–F19.

Koon, R. and Ufondu, L. (2015). Geothermal energy prospecting of the Yeoman and Winnipeg formations within Estevan, Canada. *Geothermal Reservoir Engineering*.

Kythe, P. and Puri, P. (2011). *Computational methods for linear integral equations*. Springer Science & Business Media.

Langtangen, H. P. and Cai, X. (2008). On the efficiency of python for high-performance computing: A case study involving stencil updates for partial differential equations.

Li, Y. and Constable, S. (2007). 2D marine controlled-source electromagnetic modeling: Part 2—the effect of bathymetry. *Geophysics*, 72(2):WA63–WA71.

Li, Y. and Dai, S. (2011). Finite element modelling of marine controlled-source electromagnetic responses in two-dimensional dipping anisotropic conductivity structures. *Geophysical Journal International*, 185(2):622–636.

Li, Y. and Key, K. (2007). 2D marine controlled-source electromagnetic modeling: Part 1—an adaptive finite-element algorithm. *Geophysics*, 72(2):WA51–WA62.

Løseth, L. O. (2007). *Modelling of Controlled Source Electromagnetic Data*. PhD thesis, Norwegian University of Science and Technology.

MacGregor, L., Sinha, M., and Constable, S. (2001). Electrical resistivity structure of the valu fa ridge, lau basin, from marine controlled-source electromagnetic sounding. *Geophysical Journal International*, 146(1):217–236.

Mackie, R. L., Smith, J. T., and Madden, T. R. (1994). Three-dimensional electromagnetic modeling using finite difference equations: The magnetotelluric example. *Radio Science*, 29(4):923–935.

Marchetti, M., Cafarella, L., Di Mauro, D., and Zirizzotti, A. (2002). Ground magnetometric surveys and integrated geophysical methods for solid buried waste detection: a case of study. *Annals of Geophysics*, 45(3):67–78.

Marchuk, G. I. and Ruzicka, J. (1975). *Methods of numerical mathematics*. Springer-Verlag New York.

Miller, P. (2016). pyMPI: Putting the py in MPI. http://pympi.sourceforge.net/.

Monk, P. (2003). *Finite element methods for Maxwell's equations*. Clarendon Press Oxford, first edition.

Mortensen, J. J., Hansen, L. B., and Jacobsen, K. W. (2005). Real-space grid implementation of the projector augmented wave method. *Physical review B*, 71(3):035109.

Mortensen, M. and Langtangen, H. P. (2016). High performance python for direct numerical simulations of turbulent flows. *Computer Physics Communications*, 203:53–65.

MPICH2-Team (2003–2016). MPICH2: a portable implementation of MPI. In *MPICH2*.

Mukherjee, S. and Everett, M. E. (2011). 3D controlled-source electromagnetic edge-based finite element modeling of conductive and permeable heterogeneities. *Geophysics*, 76(4):F215–F226.

Mur, G. (1981). Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations. *IEEE Transactions on Electromagnetic Compatibility*, 17(4):377–382.

Mur, G. and De Hoop, A. (1985). A finite-element method for computing three-dimensional electromagnetic fields in inhomogeneous media. *IEEE Transactions on Magnetics*, 21(6):2188–2191.

References

Nabighian, M. (1988). *Electromagnetic Methods in Applied Geophysics: Theory*. Society of Exploration Geophysics.

Nédélec, J.-C. (1980). Mixed finite elements in R3. *Numerische Mathematik*, 35(3):315–341.

Newman, G. A. and Alumbaugh, D. L. (2002). Three-dimensional induction logging problems, part 2: A finite-difference solution. *Geophysics*, 67(2):484–491.

Newman, G. A. and Commer, M. (2009). Massively parallel electrical conductivity imaging of the subsurface: Applications to hydrocarbon exploration. In *Journal of Physics: Conference Series*.

Newman, G. A., Commer, M., and Carazzone, J. J. (2010). Imaging CSEM data in the presence of electrical anisotropy. *Geophysics*, 75(2):F51–F61.

Nguyen, T. (2006). *Finite Element Methods: Parallel-Sparse Statics and Eigen-Solutions*. Springer.

Nielsen, O. (2016). Pypar home page. https://github.com/daleroberts/pypar.

Operto, S., Virieux, J., Amestoy, P., L'Excellent, J.-Y., Giraud, L., and Ali, H. B. H. (2007). 3D finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver: A feasibility study. *Geophysics*, 72(5):SM195–SM211.

Osseyran, A. and Giles, M. (2015). *Industrial Applications of High-Performance Computing: Best Global Practices*. Chapman & Hall/CRC Computational Science. CRC Press, first edition.

Owen, S. J. (1998). A survey of unstructured mesh generation technology. In *IMR*, pages 239–267.

Perttu, N. and Wikberg, L. (2005). Tools for groundwater prospecting and geophysical prospecting for water in Ocotal, Nicaragua. Master's thesis, Luleå University of Technology.

Plessix, R.-E., Darnet, M., and Mulder, W. (2007). An approach for 3D multisource, multifrequency CSEM modeling. *Geophysics*, 72(5):SM177–SM184.

Puzyrev, V., Koldan, J., de la Puente, J., Houzeaux, G., Vázquez, M., and Cela, J. M. (2013). A parallel finite-element method for three-dimensional controlled-source electromagnetic forward modelling. *Geophysical Journal International*, page ggt027.

Rognes, M. E., Kirby, R. C., and Logg, A. (2009). Efficient assembly of H(div) and H(curl) conforming finite elements. *SIAM Journal on Scientific Computing*, 31(6):4130–4151.

Rylander, T., Ingelström, P., and Bondeson, A. (2012). *Computational Electromagnetics*. Texts in Applied Mathematics. Springer.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems: Second Edition*. EngineeringPro collection. Society for Industrial and Applied Mathematics.

Said, R., Weatherill, N., Morgan, K., and Verhoeven, N. (1999). Distributed parallel delaunay mesh generation. *Computer methods in applied mechanics and engineering*, 177(1):109–125.

Sarrate Ramos, J., Huerta, A., et al. (2000). Efficient unstructured quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*.

Schwarzbach, C., Börner, R.-U., and Spitzer, K. (2011). Three-dimensional adaptive higher order finite element simulation for geo-electromagnetics—a marine CSEM example. *Geophysical Journal International*, 187(1):63–74.

Sheard, S., Ritchie, T., Christopherson, K. R., and Brand, E. (2005). Mining, environmental, petroleum, and engineering industry applications of electromagnetic techniques in geophysics. *Surveys in Geophysics*, 26(5):653–669.

Sheriff, R. E. (2002). *Encyclopedic dictionary of applied geophysics*. Society of exploration geophysicists.

Shewchuk, J. R. (2002). Delaunay refinement algorithms for triangular mesh generation. *Computational geometry*, 22(1):21–74.

Squyres, J., Willcock, J., McCandless, B., Rijks, P., and Lumsdaine, A. (2016). OOMPI. http://www.osl.iu.edu/research/oompi/.

Srnka, L. J., Carazzone, J. J., Ephron, M. S., and Eriksen, E. A. (2006). Remote reservoir resistivity mapping. *The Leading Edge*, 25(8):972–975.

Strack, K. and Aziz, A. (2012). Full field array electromagnetics-advanced EM from the surface to the borehole, exploration to reservoir monitoring. In *74th EAGE Conference and Exhibition incorporating EUROPEC 2012*.

Tang, J., Ren, Z., and Hua, X. (2007). The forward modeling and inversion in geophysical electromagnetic field [j]. *Progress in Geophysics*, 4:024.

Tomov, S., Nath, R., Ltaief, H., and Dongarra, J. (2010). Dense linear algebra solvers for multicore with gpu accelerators. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE.

Um, E. S., Commer, M., and Newman, G. A. (2013). Efficient pre-conditioned iterative solution strategies for the electromagnetic diffusion in the earth: finite-element frequency-domain approach. *Geophysical Journal International*, page ggt071.

van Welij, J. (1985). Calculation of eddy currents in terms of H on hexahedra. *IEEE Transactions on Magnetics*, 21(6):2239–2241.

Webb, J. (1993). Edge elements and what they can do for you. *IEEE Transactions on Magnetics*, 29(2):1460–1465.

Weiss, C. J. and Constable, S. (2006). Mapping thin resistors and hydrocarbons with marine EM methods, part ii—modeling and analysis in 3D. *Geophysics*, 71(6):G321–G332.

Whitney, H. (1957). *Geometric integration theory*, volume 21. Princeton]University Press.

Xiong, Z., Raiche, A., and Sugeng, F. (2000). Efficient solution of full domain 3D electromagnetic modelling problems. *Exploration Geophysics*, 31(1/2):158–161.

Xue, G., Li, X., and Di, Q. (2008). Research progress in tem forward modeling and inversion calculation [j]. *Progress in Geophysics*, 4:023.

Zhang, Y., Hughes, T. J., and Bajaj, C. L. (2008). Automatic 3D mesh generation for a domain with multiple materials. In *Proceedings of the 16th international meshing roundtable*, pages 367–386. Springer.

Zhdanov, M. S. (2009). *Geophysical electromagnetic theory and methods*, volume 43. Elsevier.

Zienkiewicz, O. C., Taylor, R. L., Zienkiewicz, O. C., and Taylor, R. L. (1977). *The finite element method*, volume 3. McGraw-hill London.

Zyserman, F. I. and Santos, J. E. (2000). Parallel finite element algorithm with domain decomposition for three-dimensional magnetotelluric modelling. *Journal of Applied Geophysics*, 44(4):337–351.

Appendix A

Maxwell's equations theory

Maxwell's equations are a set of fundamental equations that includes the four differential equations relating the electric vector field, **E**, and the magnetic vector field **B**, which governs all electromagnetic phenomena. These equations can be written in both integral forms and differential forms, which are described in the following lines.

Integral form of Maxwell's equations

For general time varying fields, Maxwell's equations in the integral form are given by

$$\oint_C \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \iint_S \mathbf{B} \cdot d\mathbf{s} \qquad \text{Faraday's law,} \qquad \text{(A.1)}$$

$$\oint_C \mathbf{H} \cdot d\mathbf{l} = \frac{d}{dt} \iint_S \mathbf{D} \cdot ds + \iint_S \mathbf{J} \cdot d\mathbf{s} \qquad \text{Ampère law,} \qquad \text{(A.2)}$$

$$\oiint_S \mathbf{D} \cdot d\mathbf{s} = \iiint_V \rho dv \qquad \text{Gauss's law,} \qquad \text{(A.3)}$$

$$\oiint_S \mathbf{B} \cdot d\mathbf{s} = 0 \qquad \text{Gauss's law – magnetic,} \qquad \text{(A.4)}$$

$$\oiint_S \mathbf{J} \cdot d\mathbf{s} = -\frac{d}{dt} \iiint_V \rho dv \qquad \text{Equation of continuity,} \qquad \text{(A.5)}$$

where **E** is the electric field intensity (Vm^{-1}), **B** is the magnetic flux density Wbm^{-2}, **D** is the electric flux density Cm^{-2}, **J** is the electric current density Am^{-2}, **H** is the magnetic field intensity Am^{-1} and ρ is the electric charge Cm^{-3}.

In equations (A.1) and (A.2), S represents an arbitrary open surface bounded by contour C, whereas in equations (A.3), (A.4) and (A.5), S is a closed surface enclosing volume V. Equations (A.1) – (A.5) are valid in all circumstances regardless of the medium and the shape of the integration volume, surface, and contour (Jin,

2002).

Differential form of Maxwell's equations

Maxwell's equations in their general differential form can be obtained from (A.1) – (A.5). To do that, the first step is consider a point in space where the field quantities and their derivatives are continuous, the second step is the application of Stokes's and Gauss's theorems to (A.1)– (A.5), which yields

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \qquad \text{Faraday's law,} \tag{A.6}$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J} \qquad \text{Ampère law,} \tag{A.7}$$

$$\nabla \cdot \mathbf{D} = \rho \qquad \text{Gauss's law,} \tag{A.8}$$

$$\nabla \cdot \mathbf{B} = 0 \qquad \text{Gauss's law – magnetic,} \tag{A.9}$$

$$\nabla \cdot \mathbf{J} = -\frac{\partial \rho}{\partial t} \qquad \text{Equation of continuity.} \tag{A.10}$$

Among equations (A.6)– (A.10), only three are independent for the case of time varying fields. Either the first three equations, (A.6)– (A.8), or the first two, (A.6) and (A.7), with A.10 can be chosen as independent equations. The other two equations, (A.9) and (A.10) or (A.8) and (A.9), can be derived from the independent equations.

Time harmonic electromagnetic fields

Time harmonic fields in Maxwell's equations are harmonically oscillating functions with a single frequency. Following the process defined by (Harrington, 1961), (A.6), (A.7) and (A.10) can be written in a simple notation as

$$\nabla \times \mathbf{E} = -i\omega\mathbf{B}, \tag{A.11}$$

$$\nabla \times \mathbf{H} = i\omega\mathbf{D} + \mathbf{J}, \tag{A.12}$$

$$\nabla \cdot \mathbf{J} = -i\omega\rho, \tag{A.13}$$

where the time convention $e^{i\omega t}$ is used and suppressed and ω is angular frequency. Furthermore, any time varying field can be expressed in terms of time harmonic com-

ponents via the Fourier transforms

$$\mathbf{E}(t) = \int_{-\infty}^{\infty} \mathbf{E}(\omega) e^{i\omega t} d\omega, \tag{A.14}$$

$$\mathbf{E}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathbf{E}(t) e^{-i\omega t} dt. \tag{A.15}$$

Is evident that if a time harmonic field is known for any ω, its counterpart in the time domain can be obtained by evaluating (A.14). It is noteworthy that equations (A.11)–(A.13) are complex-valued fields even in lossless media.

Static electromagnetic fields

In the static case, with no time dependency, the Maxwell's equations decouple into an electrostatic system

$$\nabla \times \mathbf{E} = 0, \tag{A.16}$$

$$\nabla \cdot \mathbf{D} = \rho, \tag{A.17}$$

and a magnetostatic system

$$\nabla \times \mathbf{H} = \mathbf{J}, \tag{A.18}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{A.19}$$

Equations (A.16) – (A.17) are commonly modeled by introducing a scalar electric potential which results in a Poisson problem for its potential (Harrington, 1961). On the other hand, magnetostatic problem (A.18) – (A.19) is not easily treated. It arise as a special case of the quasi-static applications (low frequency applications) and includes the complexity of magnetic properties of the medium, resulting in discontinuities and singularities in the field components. Other common approach is introduce a potential which becomes a vector potential for the magnetic flux.

Constitutive relations

The three independent equations among (A.6)– (A.10) are indefinite form since the number of equations is less than the number of unknowns ($\mathbf{E}$, $\mathbf{B}$, $\mathbf{H}$, $\mathbf{D}$, $\mathbf{J}$). However, Maxwell's equations become definite when constitutive relations between the field quantities are specified (Jin, 2002). Therefore, the constitutive relations for a simple

medium are given by

$$\mathbf{D} = \epsilon \mathbf{E}, \tag{A.20}$$

$$\mathbf{B} = \mu \mathbf{H}, \tag{A.21}$$

$$\mathbf{J} = \sigma \mathbf{E}, \tag{A.22}$$

where ϵ is the permittivity (Fm^{-1}), μ is the permeability (Hm^{-1}) and σ is the conductivity Sm^{-1}. These constitutive parameters, depending on the medium, are tensors for anisotropic media and scalars for isotropic media. For inhomogeneous media, they are functions of position, whereas for homogeneous media they are not.

It is very important to carefully choose the form of the constitutive relations that is suitable to describe the Earth in the problem that we want to solve. For example, in problems that arise in 3D CSEM FM, it is normally assumed that the Earth is heterogeneous, anisotropic and with electromagnetic parameters that are independent of temperature, time and pressure (Koldan, 2013).

Appendix B

Numerical techniques in electromagnetics

This appendix is based on two premises: the first one is that experiments have showed that all electromagnetic phenomena are governed by empirical Maxwell's equations, which are uncoupled first-order linear Partial Differential Equations (PDE). The second one is that in dealing with electromagnetic fields for oil & gas prospecting works, numerical techniques are considered an important component. Therefore, in order to obtain a numerical solution to PDE, is necessary to discretize the Maxwell's equations, which are, by nature, continuous, using some discretisation method such as Finite Difference Method (FDM), Finite Element Method (FEM) and Edge Finite Element Method (EFEM), among others.

FEM has long been used in solid mechanics, heat transfer, fluid mechanics, acoustics, and other fields. In geophysics prospecting, however, it has been employed for only a few decades, examples of FEM implementations for electromagnetic modelling are (Zyserman and Santos, 2000; Badea et al., 2001; MacGregor et al., 2001; Key and Weiss, 2006; Kong et al., 2007; Li and Key, 2007; Li and Constable, 2007; Franke et al., 2007; Li and Dai, 2011; Puzyrev et al., 2013; Koldan, 2013). Examples of FDM in geophysical prospecting can be found in (Mackie et al., 1994; Alumbaugh et al., 1996; Xiong et al., 2000; Fomenko and Mogi, 2002; Newman and Alumbaugh, 2002; Davydycheva et al., 2003; Kong, 2007; Abubakar et al., 2008; Davydycheva and Rykhlinski, 2011).

However, FEM is still not as widely applied as FDM and a major obstacle for its broader adoption is that the standard FEM does not correctly take into account all the physical aspects of the vector field functions. In fact, there are three main problems when nodal-based finite elements, obtained by interpolating the nodal values, are

employed to represent vector electric or magnetic fields. The first one is the occurrence of spurious solutions or non-physical solutions, which is generally attributed to lack of enforcement of the divergence condition. The second one is the inconvenience of imposing boundary conditions at material interfaces as well as at conducting surfaces. Finally, the third problem is the difficulty in treating conducting and dielectric edges and corners due to field singularities associated with these structures. These drawbacks have encouraged the exploration of other possibilities or other approaches. This approach uses so-called vector basis functions that assign degrees of freedom (DOFs) to the edges rather than to the nodes of each element, hence these elements are called edge elements or Nédélec Elements. Since, edge elements are free of all the previously mentioned shortcomings, they are chosen as discretisation method for this thesis. Both techniques, nodal-based approach and edge-based approach are described in this appendix.

B.1 Finite Element Method (FEM)

FEM is a numerical technique for obtaining approximate solutions to boundary value problems of mathematical physics. The method has a history of about 50 years. The principle of FEM is to replace an entire continuous domain by a number of sub-domains in which the unknown function is represented by simple interpolation functions with unknown coefficients. Thus, the original boundary-value problem with an infinite number of DOF sis converted into a problem with a finite number of DOFs, namely, the solution of the entire system is approximated by a finite number of unknown coefficients. Then a system of algebraic equations is obtained by applying the Ritz variational or Galerkin procedure (Marchuk and Ruzicka, 1975; Burnett, 1987; Jin, 2002). The Ritz method formulates the boundary value problem in terms of a variational expression, called functional. Galerkin's approach belongs to the family of weighted residual techniques. Finally, solution of the boundary value problem is achieved by solving the system of equations. To best introduce the FEM, this section defines boundary-value problems and then review two classical methods for their solution which are the roots of the modern FEM.

Boundary-value problems

Boundary-value problems (BVP) arise in the mathematical modelling of physical systems and their solution has long been a major topic in mathematical physics (Jin,

2002). In general, a boundary-value problem can be described by a governing differential equation in a domain Ω, whose standard form may be

$$\Psi\phi = f, \tag{B.1}$$

together with the boundary conditions (BC) on the boundary Γ that encloses the domain. In (B.1) Ψ is a differential operator, f is the forcing term, and ϕ is the unknown value. In electromagnetic problems, the form of governing differential equation ranges from simple Poisson equations, to complicated vector wave equations. The BC also range from the simple Dirichlet (Ω_D) and Neumann conditions (Ω_N) (Burnett, 1987), to complicate, elegant and efficient BC such as Absorbing Boundary Conditions (ABC) (Mur, 1981; Burnett, 1987; Berenger, 1994; Feng, 1999). Normally, the entire boundary Ω is divided into a Dirichlet boundary and a Neumann boundary

$$\Omega = \Omega_D + \Omega_N. \tag{B.2}$$

Analytical solution can be obtained for only a few special problems. Many other problems of societal and engineering value do not have an analytical solution. To overcome this difficulty, many approximate techniques have been developed, and among them the Ritz and Galerkin methods have been used most widely (Burnett, 1987; Jin, 2002).

Ritz method

The Ritz method is a direct method to find an approximate solution for BVP. In Ritz method, the BVP is formulated in terms of a variational expression or functional. The minimum of this functional corresponds to the governing differential equation under the given BC. The approximate solution is then obtained by minimizing the functional with respect to variables that define a certain approximation to the solution. For instance, for a inner product defined as

$$\langle \phi, \varphi \rangle = \int_{\Omega} \phi \varphi^* d\Omega, \tag{B.3}$$

where φ^* denotes the complex conjugate. If the differential operator Ψ in (B.1) is self-adjoint and also positive definite, respectively, that is

$$\langle \Psi\phi, \varphi \rangle = \langle \phi, \Psi\varphi \rangle, \tag{B.4}$$

$$\langle \Psi\phi, \phi \rangle = \begin{cases} > 0 & \phi \neq 0 \\ = 0 & \phi = 0 \end{cases}, \tag{B.5}$$

then the solution to (B.1) can be obtained by minimizing the variational expression with respect to $\tilde{\phi}$, given by

$$F(\tilde{\phi}) = \frac{1}{2}(\Psi\tilde{\phi}, \tilde{\phi}) - \frac{1}{2}(\tilde{\phi}, f) - \frac{1}{2}(f, \tilde{\phi}), \tag{B.6}$$

where $\tilde{\phi}$ denotes a trial function (Burnett, 1987). For a real-valued problem, the trial function can be approximated by the expression (Jin, 2002)

$$\tilde{\phi} = \sum_{i=1}^{N} c_j v_j = \{c\}^T\{v\} = \{v\}^T\{c\}, \tag{B.7}$$

where the v_j are the chosen expansion functions defined over all domain Ω, c_j are constant coefficients to be determined, $\{\cdot\}$ denotes a column vector and the superscript T his transpose. Substituting (B.7) into (B.6)

$$F = \frac{1}{2}\{c\}^T \int_\Omega \{v\}\Psi\{v\}^T \, d\Omega\{c\} - \{c\}^T \int_\Omega \{v\} f \, d\Omega. \tag{B.8}$$

In order to minimize $F(\tilde{\phi})$, is necessary force its partial derivatives with respect to c_i. This yields a set of linear algebraic equations

$$\begin{aligned} \frac{\partial F}{\partial c_i} &= \frac{1}{2}\int_\Omega v_i \Psi\{v\}^T \, d\Omega\{c\} + \frac{1}{2}\{c\}^T \int_\Omega \{v\}\Psi v_i \, d\Omega - \int_\Omega v_i f \, d\Omega \\ &= \frac{1}{2}\sum_{j=1}^{N} c_j \int_\Omega (v_i \Psi v_j + v_j \Psi v_i) \, d\Omega - \int_\Omega v_i f \, d\Omega \\ &= 0, \qquad\qquad i = 1, 2, 3, \ldots, N. \end{aligned} \tag{B.9}$$

The linear algebraic equations (B.9) can be written as the standard matrix equation

$$[A] \cdot \{x\} = \{b\}, \tag{B.10}$$

where the elements of the matrix $[A]$ and the vector $\{b\}$ are given by

$$A_{ij} = \frac{1}{2}\int_{\Omega}(v_i \Psi v_j + v_j \Psi v_i)\, d\Omega, \tag{B.11}$$

$$b_i = \int_{\Omega} v_i f\, d\Omega. \tag{B.12}$$

In electromagnetic simulations, and particularly in geophysical prospecting through EM such as 3D CSEM FM, matrix $[A]$ is large, sparse, complex and symmetric. Vector $\{x\}$ contains the unknowns coefficients. Then (B.7) is an approximation for (B.1), where the x_i are obtained by solving the matrix system (B.10) through a sequential or parallel linear equation solver (Gill et al., 1991; Bhogeswara and Killough, 1994; Grund, 1999; Filippone and Colajanni, 2000; Tomov et al., 2010). In addition, there are ad-hoc implementations of solvers for electromagnetic geophysical prospecting developed by (Dogru et al., 2002; Collins et al., 2003; Cao et al., 2005; Weiss and Constable, 2006; Li and Key, 2007; Operto et al., 2007; Koldan et al., 2014).

Galerkin method

Galerkin method, belonging to the family of Methods of Weighted Residuals (MWR), is a class of method for converting a continuous operator problem to a discrete problem (Burnett, 1987). In principle, is the equivalent of applying the technique of variation of parameters to a function space, by converting the equation to a weak formulation. Namely, Galerkin method seek the solution by weighting the residual of the differential equation.

The method assumes that $\tilde{\phi}$ is an approximate solution to a given BVP, for instance, (B.1). Substitution of $\tilde{\phi}$ for ϕ in (B.1) would then result in a nonzero residual

$$r = \Psi\tilde{\phi} - f \neq 0. \tag{B.13}$$

The aim is obtain an approximation for $\tilde{\phi}$ that reduces the residual r to the least value at all points of the domain Ω. Therefore, Galerkin method enforce the condition (Burnett, 1987; Jin, 2002)

$$R_i = \int_{\Omega} w_i r\, d\Omega = 0, \tag{B.14}$$

where R_i denote the weighted residual integrals and w_i are chosen weighting functions. In order to obtain the most accurate solution, in MWR the weighted functions are

selected to be the same as those used for expansion of the approximate solution. Then, for a solution represented as in (B.7), the weighting functions are selected as

$$w_i = v_i, \qquad i = 1, 2, 3, \ldots, N, \tag{B.15}$$

therefore (B.14) becomes

$$R_i = \int_\Omega (v_i \Psi\{v\}^T\{c\} - v_i f)\, d\Omega = 0, \qquad i = 1, 2, 3, \ldots, N. \tag{B.16}$$

This method again leads to the matrix system (B.10), although the matrix $[A]$ is not necessarily symmetric unless the operator Ψ is self-adjoint. Since in MWR is possible choose different expansion functions, there are different formulations such as point collocation method, subdomain collocation method and least squares method (Burnett, 1987).

General steps of Finite Element Method

FEM is a powerful and versatile numerical technique for geophysicists because it includes flexibility and a capability to handle complex structures, through unstructured meshes, that often appear in the real heterogeneous subsurface geology (shapes of orebodies, cylindrical wells, topography, seabed bathymetry, etc.). In addition, since the complexity of domains in geophysical prospecting simulations varies considerably from one area of the domain to another, FEM supports local refinement which is an efficient way to provide higher solution accuracy only at places where is desired (around transmitters, receivers, target locations as well as large conductivity contrasts). This is an important characteristic since imprecise modelling of complex shapes may result in misleading artefacts in images (Koldan, 2013). In addition, the systematic generality of the method makes it possible to construct general-purpose computer programs for solving a wide range of problems. Consequently, programs developed for a particular discipline have been applied successfully to solve problems in a different field with little or no modification.

A FEM analysis of BVP should include the following basic steps:

1. Discretize or subdivide the domain.
2. Select the interpolation functions.
3. Determine the elemental properties or elemental equations.
4. Assemble the system of equations.

5. Imposition of BC and solve the global equation system.
6. Compute output in an suitable format to visualise results (Post-processing).

Two-dimensional Finite Element Method

In order to explain previous steps, the following 2D BVP is considered

$$-\frac{\partial}{\partial x}\left(\alpha_x \frac{\partial \phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\alpha_y \frac{\partial \phi}{\partial y}\right) + \beta\phi = f \qquad x, y \in \Omega, \tag{B.17}$$

where ϕ is the unknown function, $\alpha_x, \alpha_y, \beta$ are physical known parameters associated with the domain Ω, and f is a source or excitation function. Special forms of (B.17) are the ordinary 2D Laplace equation, Poisson equation and Helmholtz equation (Jin, 2002). BC for (B.17) are given by

$$\phi = p \qquad \text{on } \Gamma_1, \tag{B.18}$$

$$\left(\alpha_x \frac{\partial \phi}{\partial x}\hat{x} + \alpha_y \frac{\partial \phi}{\partial y}\hat{y}\right) \cdot \hat{n} + \upsilon\phi = q \qquad \text{on } \Gamma_2, \tag{B.19}$$

where $\Gamma = \Gamma_1 + \Gamma 2$ denotes the boundary enclosing the domain Ω, $\hat{n}$ is its outward normal unit vector, and υ, p, and q are physical known parameters associated with Ω.

Domain discretisation. Discretisation of the domain involves dividing the entire domain into a number of small sub-domains, denoted as Ω^e $(e = 1, 2, 3, ..., N)$, where N denotes the total number of sub-domains or *elements*. There are different kinds of elements, each one with their own properties, advantages and disadvantages. A complete description of that can be found in (Burnett, 1987). For instance, for a one-dimension (1D) domain the elements commonly are short line segments. For two-dimension (2D) domains, the elements are often triangles since these are the best suited for discretizing irregular regions. In three-dimension (3D) cases, the domain is preferably subdivided into tetrahedrons.

In FEM solutions, the problem is formulated of the unknown function, commonly called ϕ, at nodes or vertex associated with each element. Therefore, 1D linear line element has two nodes (one at each endpoint), 2D linear triangular element has three nodes (one at each vextex), and similarly 3D linear tetrahedron has four nodes (Burnett, 1987). For implementation purposes, a complete description of these nodes is necessary, namely, the elements and nodes can be labeled with separate sets of integers for identification. Since each element is related to several nodes, three nodes in

2D cases with linear triangular elements as show figure B.1b, a node can be assigned a local label in the associated element in addition to its global number relative to the global system. The relation between number of element, local node number and global node number is stored in a $3 \times N$ array, commonly called connectivity array. The connectivity array is defined by $nodes(i, e)$, where $i = 1, 2, 3$, $e = 1, 2, 3, \ldots, N$, and N is the total number of elements. The array $nodes(i, e)$ contains the global node numbers indexed by the local node number i and the element e. Table B.1 is an example of connectivity array for the 2D domain of the figure B.1a . Connectivity array numbering is not unique. For instance, is possible number the three nodes of the first element as $1, 2, 4$ or $4, 1, 2$ since they are consistent with local numbering show in figure B.1a. In addition to connectivity array defined by the table B.1, some other

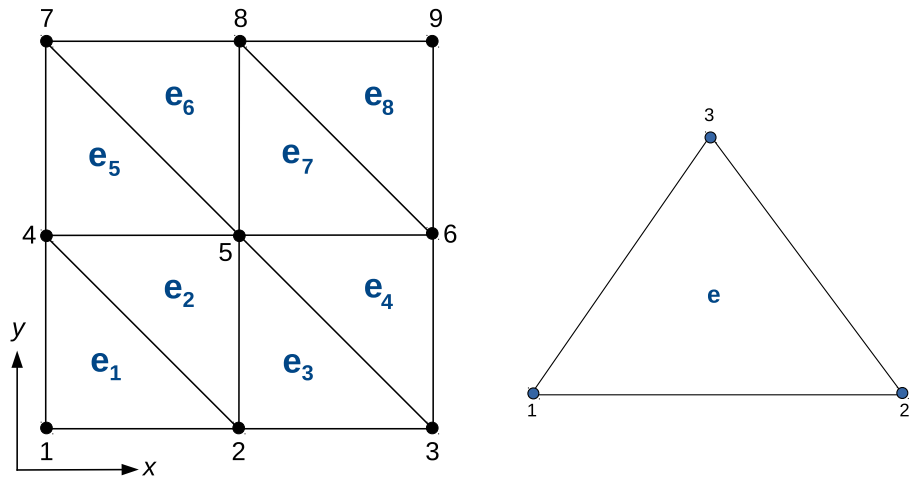
e	nodes(1,e)	nodes(2,e)	nodes(3,e)
1	2	4	1
2	2	4	5
3	2	3	5
4	3	5	6
5	4	5	7
6	5	7	8
7	5	6	8
8	6	9	8

Table B.1 Element to nodes connectivity array in 2D.

data are also necessary in the FEM, which are:

1. A vector coordinates of the nodes (x_i, y_i) for $i = 1, 2, 3, \ldots, n$, where n is the total number of nodes.
2. The physical parameters for each element, namely the values of α_x, α_y, β and the source f.
3. The values of boundary nodes, in other words the value of p for nodes residing on Γ_1 and the value of υ and q for nodes on Γ_2.

The domain discretisation is an important stage in FEM formulations since it has a direct impact in the computer storage requirements, the computation time and the accuracy of the numerical results. Therefore, the domain subdivision is treated as a preprocessing task to the FEM provided by a specific software such as *Netgen*, *Gambit*, *Gmsh*, *Tetgen*, among others. Examples of performance assessments of discretisation



(a) 2D domain (b) Linear triangular element

Fig. B.1 Discretisation in 2D.

methods or meshing techniques and its computational implementation are (Ho-Le, 1988; Sarrate Ramos et al., 2000; Shewchuk, 2002; Geuzaine and Remacle, 2009).

Interpolation functions. After domain discretisation stage, the unknown function ϕ is approximated within each element. Therefore, the unknown function ϕ within the linear triangular element of figure B.1b can be approximated by (Jin, 2002)

$$\phi^e(x, y) = a^e + b^e x + c^e y, \tag{B.20}$$

where a^e, b^e, and c^e are coefficients to be determined and e denotes the number of element in the entire system. Enforcing B.20 at the three nodes of element *e-th* yields

$$\phi_1^e = a^e + b^e x_1^e + c^e y_1^e, \tag{B.21}$$
$$\phi_2^e = a^e + b^e x_2^e + c^e y_2^e, \tag{B.22}$$
$$\phi_3^e = a^e + b^e x_3^e + c^e y_3^e, \tag{B.23}$$

where x_j^e and y_j^e, for $j = 1, 2, 3$, denote the coordinate values of the *j-th* node in the *e-th*. Solving for the constant coefficients in terms of ϕ_j^e and substituting them back into B.20 yields (Jin, 2002)

$$\phi^e(x, y) = \sum_{j=1}^{3} N_j^e(x, y)\phi_j^e, \tag{B.24}$$

where N_j^e are the interpolation functions given by

$$N_j^e(x,y) = \frac{1}{2\Delta^e}(a_j^e + b_j^e x + c_j^e y) \qquad\qquad j = 1,2,3, \tag{B.25}$$

with coefficients defined by

$$\begin{aligned} a_1^e &= x_2^e y_3^e - y_2^e x_3^e, & b_1^e &= y_2^e - y_3^e, & c_1^e &= x_3^e - x_2^e, \\ a_2^e &= x_3^e y_1^e - y_3^e x_1^e, & b_2^e &= y_3^e - y_1^e, & c_2^e &= x_1^e - x_3^e, \\ a_3^e &= x_1^e y_2^e - y_1^e x_2^e, & b_3^e &= y_1^e - y_2^e, & c_3^e &= x_2^e - x_1^e, \end{aligned}$$

and the area of element e-th defined by

$$\Delta^e = \frac{1}{2}\begin{vmatrix} 1 & x_1^e & y_1^e \\ 1 & x_2^e & y_2^e \\ 1 & x_3^e & y_3^e \end{vmatrix} = \frac{1}{2}(b_1^e c_2^e - b_2^e c_1^e). \tag{B.26}$$

Interpolation functions (B.25) posses the requisite interpolation property given by:

$$N_j^e(x_j, y_j) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \tag{B.27}$$

Therefore, ϕ^e in (B.24) reduces to its nodal value ϕ_i^e at node i. In addition, $N_j^e(x,y)$ vanishes when the observation point (x,y) is on the element side opposite the j-th node (Burnett, 1987; Jin, 2002).

Elemental equations. As already mentioned, in FEM formulations the elemental properties or elemental equations can be obtained using the Ritz or Galerkin method. In sake of simplicity, here is only considering the Ritz method applied to expression (B.17) with BC defined by expression (B.19). Thus, the functional given by (B.7) can be written as

$$H(\phi) = \sum_{e=1}^{N} H^e(\phi^e), \tag{B.28}$$

where N is the total number of elements and H^e is the sub-functional for the *e-th*

element given by

$$H^e(\phi^e) = \frac{1}{2}\iint_{\Omega^e}\left[\alpha_x\left(\frac{\partial\phi^e}{\partial x}\right)^2 + \alpha_y\left(\frac{\partial\phi^e}{\partial y}\right)^2 - \beta\left(\phi^e\right)^2\right] d\Omega - \iint_{\Omega^e} f\phi^e\, d\Omega, \tag{B.29}$$

where Ω^e is the elemental domain of element *e-th*. Introducing the Lagrange functional given by expression (B.24) and differentiating H^e with respect to ϕ_i^e yields to a system equations whose matrix form is

$$\left[\frac{\partial H^e}{\partial \phi^e}\right] = [A^e]\{\phi^e\} - \{b^e\}, \tag{B.30}$$

where

$$\left[\frac{\partial H^e}{\partial \phi^e}\right] = \begin{bmatrix}\frac{\partial H^e}{\partial \phi_1^e} & \frac{\partial H^e}{\partial \phi_2^e} & \frac{\partial H^e}{\partial \phi_3^e}\end{bmatrix}^T,$$
$$\{\phi^e\} = \begin{bmatrix}\phi_1^e & \phi_2^e & \phi_3^e\end{bmatrix}^T.$$

The elements of the matrix $[A^e]$ and the elements of the vector $\{b^e\}$ from equation (B.30) obey the general form of expression (B.10) and are respectively given by

$$A_{ij}^e = \iint_{\Omega^e}\left(\alpha_x\frac{\partial N_i^e}{\partial x}\frac{\partial N_j^e}{\partial x} + \alpha_y\frac{\partial N_i^e}{\partial y}\frac{\partial N_j^e}{\partial y} + \beta N_i^e N_j^e\right) dx\, dy, \tag{B.31}$$

$$b_i^e = \iint_{\Omega^e} f N_i^e\, dx\, dy, \qquad i,j = 1,2,3. \tag{B.32}$$

For constant values of α_x, α_y, α_z and source f within each element, expressions (B.31) and (B.32) become

$$A_{ij}^e = \frac{\alpha_x^e b_i^e b_j^e + \alpha_y^e c_i^e c_j^e}{4\Delta^e} + \frac{\Delta^e\beta^e(1+\delta_{ij})}{12}, \tag{B.33}$$

$$b_i^e = \frac{\Delta^e}{3} f^e \qquad i,j = 1,2,3. \tag{B.34}$$

Assemble of system of equations. In order to find the global equation system for the whole solution domain Ω, is necessary assemble all the element equations (B.30) and impose the BC. Therefore, the global equation system can be defined as

$$\left[\frac{\partial H}{\partial \phi}\right] = \sum_{e=1}^{N}\left[\frac{\partial H^e}{\partial \phi^e}\right] = \sum_{e=1}^{N}[A^e]\{\phi^e\} - \{b^e\} = 0, \tag{B.35}$$

which can be written compactly as $[A] \cdot \{\phi\} = \{b\}$ where matrix $[A]$ is assembled from elemental matrices $[A^e]$, and similarly, the vector $\{b\}$ from elemental vectors $\{b^e\}$ (Burnett, 1987). The dimensions of the matrix $[A]$ (commonly initialized as a null matrix) are given by the number of nodes in the domain Ω. Therefore, for the problem of figure B.1a the dimensions of the matrix $[A]$ are 9×9.

Since, the final target of assembly process is to adding each element of $[A^e]$ to the appropriate element of $[A]$, the connectivity array from the discretisation step has a fundamental role because it contains the relation between local nodes and global nodes. Then, the assembly process obey the general rule of add A^e_{ij} to $[A_{nodes(i,e),nodes(j,e)}]$. A similar assembly procedure for vector $\{b\}$ is used, namely each $\{b^e_i\}$ is added to $\{b_{nodes(i,e)}\}$. Hence, after the addition of all nine elements of A^1 to $[A]$ and $\{b^1_i\}$ to $\{b\}$, the following partial matrix system is obtained

$$\begin{bmatrix} A^1_{33} & A^1_{31} & 0 & A^1_{32} & 0 & 0 & 0 & 0 & 0 \\ A^1_{13} & A^1_{11} & 0 & A^1_{12} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A^1_{23} & A^1_{21} & 0 & A^1_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi^5 \\ \phi^6 \\ \phi^7 \\ \phi^8 \\ \phi^9 \end{Bmatrix} = \begin{Bmatrix} b^1_3 \\ b^1_1 \\ 0 \\ b^1_2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}.$$

After the addition of all A^e to A, and all $\{b^e\}$ to $\{b\}$, the global matrix is is given by

$$\begin{bmatrix} \gamma_1 & \gamma_2 & 0 & \gamma_3 & 0 & 0 & 0 & 0 & 0 \\ \gamma_4 & \gamma_5 & \gamma_6 & \gamma_7 & \gamma_8 & 0 & 0 & 0 & 0 \\ 0 & \gamma_9 & \gamma_{10} & 0 & \gamma_{11} & \gamma_{12} & 0 & 0 & 0 \\ \gamma_{13} & \gamma_{14} & 0 & \gamma_{15} & \gamma_{16} & 0 & \gamma_{17} & 0 & 0 \\ 0 & \gamma_{18} & \gamma_{19} & \gamma_{20} & \gamma_{21} & \gamma_{22} & \gamma_{23} & \gamma_{24} & 0 \\ 0 & 0 & \gamma_{25} & 0 & \gamma_{26} & \gamma_{27} & 0 & \gamma_{28} & \gamma_{29} \\ 0 & 0 & 0 & \gamma_{30} & \gamma_{31} & 0 & \gamma_{32} & \gamma_{33} & 0 \\ 0 & 0 & 0 & 0 & \gamma_{34} & \gamma_{35} & \gamma_{36} & \gamma_{37} & \gamma_{38} \\ 0 & 0 & 0 & 0 & 0 & \gamma_{39} & 0 & \gamma_{40} & \gamma_{41} \end{bmatrix} \begin{Bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi^5 \\ \phi^6 \\ \phi^7 \\ \phi^8 \\ \phi^9 \end{Bmatrix} = \begin{Bmatrix} b^1_3 \\ b^1_1 + b^2_1 + b^3_1 \\ b^3_2 + b^4_1 \\ b^1_2 + b^2_2 + b^5_1 \\ b^2_3 + b^3_3 + b^4_2 + b^5_2 + b^6_1 + b^7_1 \\ b^4_3 + b^7_2 + b^8_1 \\ b^5_3 + b^6_2 \\ b^6_3 + b^7_3 + b^8_3 \\ b^8_2 \end{Bmatrix},$$

where

$$
\begin{aligned}
\gamma_1 &= A^1_{33}, & \gamma_{15} &= A^1_{22} + A^2_{22} + A^5_{11}, & \gamma_{29} &= A^8_{12},\\
\gamma_2 &= A^1_{31}, & \gamma_{16} &= A^2_{23} + A^5_{12}, & \gamma_{30} &= A^5_{31},\\
\gamma_3 &= A^1_{32}, & \gamma_{17} &= A^5_{13}, & \gamma_{31} &= A^5_{32} + A^6_{21},\\
\gamma_4 &= A^1_{13} + A^2_{13}, & \gamma_{18} &= A^2_{31} + A^3_{31}, & \gamma_{32} &= A^5_{33} + A^6_{22},\\
\gamma_5 &= A^1_{11} + A^2_{11} + A^3_{11}, & \gamma_{19} &= A^4_{21}, & \gamma_{33} &= A^6_{23},\\
\gamma_6 &= A^3_{12}, & \gamma_{20} &= A^2_{32} + A^3_{32} + A^5_{21}, & \gamma_{34} &= A^6_{31} + A^7_{31},\\
\gamma_7 &= A^1_{12} + A^2_{12}, & \gamma_{21} &= A^2_{33} + A^3_{33} + A^4_{22} + A^5_{22} + A^6_{11} + A^7_{11}, & \gamma_{35} &= A^7_{32} + A^8_{31},\\
\gamma_8 &= A^3_{13}, & \gamma_{22} &= A^4_{23} + A^7_{12}, & \gamma_{36} &= A^6_{32},\\
\gamma_9 &= A^3_{21}, & \gamma_{23} &= A^5_{23} + A^6_{12}, & \gamma_{37} &= A^6_{33} + A^7_{33} + A^8_{33},\\
\gamma_{10} &= A^3_{22} + A^4_{11}, & \gamma_{24} &= A^6_{13} + A^7_{13}, & \gamma_{38} &= A^8_{32},\\
\gamma_{11} &= A^3_{23} + A^4_{12}, & \gamma_{25} &= A^4_{31}, & \gamma_{39} &= A^8_{21},\\
\gamma_{12} &= A^4_{13}, & \gamma_{26} &= A^4_{32} + A^7_{21}, & \gamma_{40} &= A^8_{23},\\
\gamma_{13} &= A^1_{23}, & \gamma_{27} &= A^4_{33} + A^7_{22} + A^8_{11}, & \gamma_{41} &= A^8_{22},\\
\gamma_{14} &= A^1_{21} + A^2_{21}, & \gamma_{28} &= A^7_{23} + A^8_{13}.
\end{aligned}
$$

Imposition of BC and solve the global equation system. Before the system of equations is ready to solve, the imposition of BC must be applied on Γ_1. Many approaches has been deployed to impose BC in an efficient way, the most common is the following. For a general problem having n nodes residing on Γ_1, imposition of BC can be accomplished simply by setting

$$\{b_{ind(i)}\} = v(i), \qquad A_{ind(i),ind(i)} = 1, \qquad A_{ind(i),j} = 0, \qquad \text{for } j \neq ind(i), \tag{B.36}$$

and

$$\{b_j\} = \{b_j\} - A_{j,ind(i)} \cdot v(i), \qquad A_{j,ind(i)} = 0, \qquad \text{for } j \neq ind(i), \tag{B.37}$$

where $ind(i)$ is a vector that store the global node indexes of nodes residing on Γ_1, and $v(i)$ is a vector that contains the prescribed values of $\{\phi\}$. Different techniques are described in (Burnett, 1987; Jin, 2002; Nguyen, 2006).

Once the BC conditions are applied, the system can be solve for unknowns $\{\phi_i\}$. In real applications, the matrix A is extremely large, complex and normally very sparse since it is a result of the discretisation of a differential operator. Consequently, solving

the large-scale linear system is the most important and expensive part of the overall numerical method. Normally, it takes up to 90% of the whole execution time (Koldan, 2013). To solve the linear system there are different methods that can be classified into two groups: direct methods and iterative methods. Both groups have certain advantages and disadvantages, and the choice of a given method is generally problem depend. The main advantage of iterative methods is their low storage requirement, which resolves the memory issue of direct methods. In addition, there is another very important benefit thanks to which iterative methods can cope with huge computational demands more readily then direct techniques. Namely, iterative methods are much easier to implement efficiently on high-performance parallel computers than direct solvers. Currently, the most common group of iterative techniques used in applications are Krylov subspace methods (Saad, 2003; Koldan, 2013).

Post-processing. This is the last step in a FEM analysis. Results obtained in the previous step are usually in the form of raw data and difficult to interpret. In the post-processing stage, a Computer-aided Design (CAD) program is utilized to manipulate and display the solution in tabular, graphical, or pictorial form. A graphical representation of the results is very useful in understanding behaviour of the solution, for instance, to estimate the error approximation in terms of the quantities of interest. Examples of visualisation tools are Paraview (Ahrens et al., 2005) and VisIt (Childs et al., 2005).

Three-dimensional Finite Element Method

The FE formulation for 3D problems is very similar to the 2D case. To explain that, the following 3D BVP is considered

$$-\frac{\partial}{\partial x}\left(\alpha_x \frac{\partial \phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(\alpha_y \frac{\partial \phi}{\partial y}\right) - \frac{\partial}{\partial z}\left(\alpha_z \frac{\partial \phi}{\partial z}\right) + \beta\phi = f \qquad x, y, z \in V, \quad \text{(B.38)}$$

with BC determined by

$$\phi = p \qquad \text{on } S_1, \quad \text{(B.39)}$$

$$\left(\alpha_x \frac{\partial \phi}{\partial x}\hat{x} + \alpha_y \frac{\partial \phi}{\partial y}\hat{y} + \alpha_z \frac{\partial \phi}{\partial z}\hat{z}\right) \cdot \hat{n} + \upsilon\phi = q \qquad \text{on } S_2, \quad \text{(B.40)}$$

where $S = S_1 + S_2$ denotes the boundary enclosing the volume V, $\hat{n}$ is its outward normal unit vector, and υ, p, and q are physical known parameters associated with

volume V.

Domain discretisation. As already mentioned, the first step of the FE analysis is the domain subdivision. In three-dimension cases, the volume V is preferably subdivided into tetrahedrons. Similarly to the 2D case, all elements and their nodes must be labeled with a set of integers. The element index, local node index and global node index are stored in a $4 \times N$ connectivity array denoted by $nodes(i, e)$, where $i = 1, 2, 3, 4$, $e = 1, 2, 3, \ldots, N$, and N is the total number of elements. The array $nodes(i, e)$ contains the global node numbers indexed by the local node number i and the element e. Just as in the 2D case, other data that are needed include the spatial coordinates of the nodes (x_i, y_i, z_i), the value of $\alpha_x, \alpha_y, \alpha_z$ and f for each element, the prescribed value of ϕ for each node residing on S_1, and the value of v and q for each surface triangular element on S_2. Figure B.2 depicts a linear tetrahedral element.

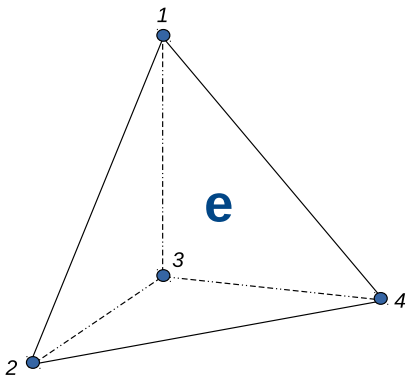


Fig. B.2 Tetrahedral nodal element.

Interpolation functions. Once domain discretisation, the unknown function ϕ can be approximated within each element as (Jin, 2002)

$$\phi^e(x, y, z) = a^e + b^e x + c^e y + d^e z. \tag{B.41}$$

Denoting the value of ϕ at the i-th node as ϕ_i^e

$$\phi_1^e = a^e + b^e x_1^e + c^e y_1^e + d^e z_1^e, \tag{B.42a}$$
$$\phi_2^e = a^e + b^e x_2^e + c^e y_2^e + d^e z_2^e, \tag{B.42b}$$
$$\phi_3^e = a^e + b^e x_3^e + c^e y_3^e + d^e z_3^e, \tag{B.42c}$$
$$\phi_4^e = a^e + b^e x_4^e + c^e y_4^e + d^e z_4^e, \tag{B.42d}$$

and the volume of the element as

$$V^e = \frac{1}{6}\begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix}, \tag{B.43}$$

the coefficients a_i^e, b_i^e, c_i^e and d_i^e can be determined by

$$a^e = \frac{1}{6V^e}\begin{vmatrix} \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix} = \frac{1}{6V^e}(a_1^e\phi_1^e + a_2^e\phi_2^e + a_3^e\phi_3^e + a_4^e\phi_4^e), \tag{B.44}$$

$$b^e = \frac{1}{6V^e}\begin{vmatrix} 1 & 1 & 1 & 1 \\ \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix} = \frac{1}{6V^e}(b_1^e\phi_1^e + b_2^e\phi_2^e + b_3^e\phi_3^e + b_4^e\phi_4^e), \tag{B.45}$$

$$c^e = \frac{1}{6V^e}\begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \\ z_1^e & z_2^e & z_3^e & z_4^e \end{vmatrix} = \frac{1}{6V^e}(c_1^e\phi_1^e + c_2^e\phi_2^e + c_3^e\phi_3^e + c_4^e\phi_4^e), \tag{B.46}$$

$$d^e = \frac{1}{6V^e}\begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e & x_4^e \\ y_1^e & y_2^e & y_3^e & y_4^e \\ \phi_1^e & \phi_2^e & \phi_3^e & \phi_4^e \end{vmatrix} = \frac{1}{6V^e}(d_1^e\phi_1^e + d_2^e\phi_2^e + d_3^e\phi_3^e + d_4^e\phi_4^e). \tag{B.47}$$

If i is running in a cyclic permutation $(1, 2, 3, 4)$, each coefficient can be determined by

$$a_1^e = \begin{vmatrix} x_2^e & x_3^e & x_4^e \\ y_2^e & y_3^e & y_4^e \\ z_2^e & z_3^e & z_4^e \end{vmatrix}, \qquad a_2^e = -\begin{vmatrix} x_3^e & x_4^e & x_1^e \\ y_3^e & y_4^e & y_1^e \\ z_3^e & z_4^e & z_1^e \end{vmatrix},$$

$$a_3^e = \begin{vmatrix} x_4^e & x_1^e & x_2^e \\ y_4^e & y_1^e & y_2^e \\ z_4^e & z_1^e & z_2^e \end{vmatrix}, \qquad a_4^e = -\begin{vmatrix} x_1^e & x_2^e & x_3^e \\ y_1^e & y_2^e & y_3^e \\ z_1^e & z_2^e & z_3^e \end{vmatrix},$$

$$b_1^e = -\begin{vmatrix} 1 & 1 & 1 \\ y_2^e & y_3^e & y_4^e \\ z_2^e & z_3^e & z_4^e \end{vmatrix}, \qquad b_2^e = \begin{vmatrix} 1 & 1 & 1 \\ y_3^e & y_4^e & y_1^e \\ z_3^e & z_4^e & z_1^e \end{vmatrix},$$

$$b_3^e = -\begin{vmatrix} 1 & 1 & 1 \\ y_4^e & y_1^e & y_2^e \\ z_4^e & z_1^e & z_2^e \end{vmatrix}, \qquad b_4^e = \begin{vmatrix} 1 & 1 & 1 \\ y_1^e & y_2^e & y_3^e \\ z_1^e & z_2^e & z_3^e \end{vmatrix},$$

$$c_1^e = \begin{vmatrix} 1 & 1 & 1 \\ x_2^e & x_3^e & x_4^e \\ z_2^e & z_3^e & z_4^e \end{vmatrix}, \qquad c_2^e = -\begin{vmatrix} 1 & 1 & 1 \\ x_3^e & x_4^e & x_1^e \\ z_3^e & z_4^e & z_1^e \end{vmatrix},$$

$$c_3^e = \begin{vmatrix} 1 & 1 & 1 \\ x_4^e & x_1^e & x_2^e \\ z_4^e & z_1^e & z_2^e \end{vmatrix}, \qquad c_4^e = -\begin{vmatrix} 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e \\ z_1^e & z_2^e & z_3^e \end{vmatrix},$$

$$d_1^e = -\begin{vmatrix} 1 & 1 & 1 \\ x_2^e & x_3^e & x_4^e \\ y_2^e & y_3^e & y_4^e \end{vmatrix}, \qquad d_2^e = \begin{vmatrix} 1 & 1 & 1 \\ x_3^e & x_4^e & x_1^e \\ y_3^e & y_4^e & y_1^e \end{vmatrix},$$

$$d_3^e = -\begin{vmatrix} 1 & 1 & 1 \\ x_4^e & x_1^e & x_2^e \\ y_4^e & y_1^e & y_2^e \end{vmatrix}, \qquad d_4^e = \begin{vmatrix} 1 & 1 & 1 \\ x_1^e & x_2^e & x_3^e \\ y_1^e & y_2^e & y_3^e \end{vmatrix}.$$

These can be written uniformly as

$$a_i^e = \begin{vmatrix} x_{i+1}^e & x_{i+2}^e & x_{i+3}^e \\ y_{i+1}^e & y_{i+2}^e & y_{i+3}^e \\ z_{i+1}^e & z_{i+2}^e & z_{i+3}^e \end{vmatrix}, \tag{B.48}$$

$$b_i^e = \begin{vmatrix} 1 & 1 & 1 \\ y_{i+1}^e & y_{i+2}^e & y_{i+3}^e \\ z_{i+1}^e & z_{i+2}^e & z_{i+3}^e \end{vmatrix}, \tag{B.49}$$

$$c_i^e = \begin{vmatrix} 1 & 1 & 1 \\ x_{i+1}^e & x_{i+2}^e & x_{i+3}^e \\ z_{i+1}^e & z_{i+2}^e & z_{i+3}^e \end{vmatrix}, \tag{B.50}$$

$$d_i^e = \begin{vmatrix} 1 & 1 & 1 \\ x_{i+1}^e & x_{i+2}^e & x_{i+3}^e \\ y_{i+1}^e & y_{i+2}^e & y_{i+3}^e \end{vmatrix}. \tag{B.51}$$

Substituting the expressions (B.42) back into (B.41) the following approximation is obtained

$$\phi^e(x, y, z) = \sum_{i=1}^{4} N_i^e(x, y, z)\phi_i^e, \tag{B.52}$$

where the interpolation functions $N_i^e(x, y, z)$ are given by

$$N_i^e(x, y, z) = \frac{1}{6V^e}(a_i^e + b_i^e x + c_i^e y + d_i^e z), \tag{B.53}$$

Similarly to the 2D case, basis functions (B.53) have the property

$$N_j^e(x_j, y_j, z_j) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}, \tag{B.54}$$

Elemental equations. Next step is to formulate the problem in terms of the unknowns at the nodes. In sake of simplicity, here is only considering the Ritz method applied to problem of (B.40) with $\upsilon = q = 0$. Therefore, functional given by (B.7) can be written as

$$H(\phi) = \sum_{e=1}^{N} H^e(\phi^e), \tag{B.55}$$

where N is the total number of volume elements and H^e is given by

$$H^e(\phi^e) = \frac{1}{2}\iiint_{\Omega^e}\left[\alpha_x\left(\frac{\partial\phi^e}{\partial x}\right)^2 + \alpha_y\left(\frac{\partial\phi^e}{\partial y}\right)^2 + \alpha_z\left(\frac{\partial\phi^e}{\partial z}\right)^2 - \beta\left(\phi^e\right)^2\right] dV - \iiint_{V^e} f\phi^e\, dV, \tag{B.56}$$

where V^e denotes the volume of the e-th element. Substituting (B.52) into (B.56) and taking the partial derivative of H^e with respect to ϕ_i^e, the following expression is obtained

$$\frac{\partial H^e}{\partial \phi_i^e} = \sum_{j=1}^{4}\phi_j^e \iiint_{V^e}\left(\alpha_x\frac{\partial N_i^e}{\partial_x}\frac{\partial N_j^e}{\partial_x} + \alpha_y\frac{\partial N_i^e}{\partial_y}\frac{\partial N_j^e}{\partial_y} + \alpha_z\frac{\partial N_i^e}{\partial_z}\frac{\partial N_j^e}{\partial_z} + \beta N_i^e N_j^e\right) dV - \iiint_{V^e} f N_i^e\, dV \qquad i = 1,2,3,4. \tag{B.57}$$

This can be written as expression (B.30) where

$$A_{ij}^e = \iiint_{V^e}\left(\alpha_x\frac{\partial N_i^e}{\partial_x}\frac{\partial N_j^e}{\partial_x} + \alpha_y\frac{\partial N_i^e}{\partial_y}\frac{\partial N_j^e}{\partial_y} + \alpha_z\frac{\partial N_i^e}{\partial_z}\frac{\partial N_j^e}{\partial_z} + \beta N_i^e N_j^e\right) dV, \tag{B.58}$$

$$b_i^e = \iiint_{V^e} f N_i^e\, dV. \tag{B.59}$$

Similarly to the 2D case, for constant values $\alpha_x, \alpha_y, \alpha z, \beta$ and source f within each element, expressions (B.58) and (B.59) become (Jin, 2002)

$$A_{ij}^e = \frac{1}{36V^e}(\alpha_x^e b_i^e b_j^e + \alpha_y^e b_i^e b_j^e + \alpha_z^e b_i^e b_j^e) + \frac{V^e}{20}\beta^e(1+\delta_{ij}), \tag{B.60}$$

$$b_i^e = \frac{V^e}{4}f^e. \tag{B.61}$$

Assemble of system of equations. The process of assembly is very similar to that for the 2D case. Indeed, just as in the 2D case, it amounts to adding each A_{ij}^e to $[A_{nodes(i,e),nodes(j,e)}]$ and $\{b_i^e\}$ to $\{b_{nodes(i,e)}\}$. This process produces a global system that obeys the expression (B.35).

Imposition of BC and solve the global equation system. The imposition of BC can be accomplished just as in the 2D case through expressions (B.36) and (B.37). Like the 2D case, for solving the system of equations there are different methods with the same properties and challenges. In (Saad, 2003) can be found a deep description of Krylov subspace methods such as GMRES (Generalized Minimum Resid-

ual), BiCGSTAB (Biconjugate Gradient Stabilized), QMR (Quasi Minimal Residual), TFQMR (Transpose-free QMR), and MINRES (Minimal Residual), among others.

Post-processing. The postprocessing stage is similar to the 2D case, therefore, this operation displays the solution to the system equations in tabular, graphical, or pictorial form. Other physical meaningful quantities might be derived from the solution and also displayed.

B.2 Edge Finite Element Method (EFEM)

As already said above, nodal-based finite elements can not be used for electromagnetic problems formulated in terms of the electric field (**E**) and/or magnetic field (**H**) functions, which is a natural and physically meaningful problem formulation. This is the main reason why traditionally, the FEM has not been applied directly to electromagnetic problems (first order form of Maxwell's equations (A.6)– (A.10)). Instead the problem could be rewritten as a second order problem, either by eliminating one of the fields, or by introducing scalar and vector potentials through gauge conditions. For the last case, the most common types of these conditions are the Coulomb gauge and Lorentz gauge. In this sense a coupled vector-scalar potential functions, v_H and v_E which represents vector magnetic potential and scalar electric potential respectively, are continuous across the interfaces between different materials, which solves the problem of discontinuity. In order to prevent the appearance of spurious solutions, it is necessary to apply an additional condition or penalty, the Coulomb gauge condition, $\nabla \cdot v_H = 0$, that enforces zero divergence of the vector potential function at element level. A parallel version of this potential-based formulation for geophysical prospecting has been applied in (Koldan, 2013).

Other solution are Nédélec Elements, more commonly called edge elements (Jin, 2002), which uses vector basis functions instead scalar basis functions. Namely, the DOFs are assigned to the edges rather than the nodes of each element. Although this kind of discretisation were described in (Whitney, 1957), it's use and importance in the electromagnetic area was not realized until recently. In (Nédélec, 1980) is discussed the construction of edge elements on tetrahedrons and rectangular bricks, tetrahedral edge elements to 3D eddy-current problems were applied in (Bossavit and Verite, 1982; van Welij, 1985; Kameari, 1988; Barton and Cendes, 1987), rectangular edge elements for the analysis of dielectric-loaded waveguides (Hano, 1984), electromagnetic fields in inhomogeneous media (Mur and De Hoop, 1985) and the developed of a more sophisti-

cated element type, called covariant projection elements, which permit elements with curved edges (Crowley et al., 1988). In addition, edge elements on hexahedrons (Bespalov et al., 2007; Epov et al., 2007; da Silva et al., 2012; Cai et al., 2014; Chung et al., 2014) and on tetrahedrons (Mukherjee and Everett, 2011; Schwarzbach et al., 2011) are used in some recent approaches to 3D CSEM FM.

In all of these works, edge elements have been shown to be free of all the shortcomings of nodal-based finite elements formulations if an appropriate selection of the interpolation functions or basis functions is made. Edge Finite Element Method (EFEM) use the same principle of FEM which consist in replacing an entire continuous domain by a number of sub-domains in which the unknown function is represented by simple interpolation functions with unknown coefficients. Thus, the solution of the entire system is approximated by a finite number of unknown coefficients. Then a system of algebraic equations is obtained and finally, solution of the BVP is achieved by solving the system of equations. Since EFEM has the same general steps of FEM, this section define the main features of edge elements for implementation purposes on unstructured tetrahedral meshes. Namely, a strong emphasis is dedicated to the basis functions, the elemental equations formulation and some computational aspects. The type of discretisation used in this thesis obeys to the lack of formulations and codes with tetrahedral elements and because it enables the representation of complex geological structures and allows local refinement in order to improve the solution's accuracy.

Two-dimensional Edge Finite Element Method

As already mentioned, the main disadvantage of rectangular elements is that they are restricted to a limited kind of geometries, therefore for problems that deal with irregular geometries is necessary the use of triangular elements. Therefore the formulation for the triangular edge elements is considered here.

Interpolation functions. For triangular edge elements of lowest order, their DOFs are associated with edges, which are sub-entities of dimension 1 as show in figure B.3. Their edge basis functions or interpolation functions are based on the area coordinates defined by expression (B.25) which here is renamed as $\lambda_i^e = N_i^e$, where $i = 1, 2, 3$. Namely, the basis functions for a triangular edge Element can be expressed in terms of first order of node-based Finite Elements (Lagrange Elements) (Jin, 2002). For the triangular element of figure B.3 with edges defined by table B.2, their basis functions

edge	i_1	i_2
1	1	2
2	2	3
3	3	1

Table B.2 Edges definition: triangular element

can be expressed by the following vectorial functions

$$\mathbf{N}_1^e = \mathbf{W}_{12}\ell_1^e = (\lambda_1^e \boldsymbol{\nabla}\lambda_2^e - \lambda_2^e \boldsymbol{\nabla}\lambda_1^e)\ell_1^e, \tag{B.62}$$

$$\mathbf{N}_2^e = \mathbf{W}_{23}\ell_2^e = (\lambda_2^e \boldsymbol{\nabla}\lambda_3^e - \lambda_3^e \boldsymbol{\nabla}\lambda_2^e)\ell_2^e, \tag{B.63}$$

$$\mathbf{N}_3^e = \mathbf{W}_{31}\ell_3^e, = (\lambda_3^e \boldsymbol{\nabla}\lambda_1^e - \lambda_1^e \boldsymbol{\nabla}\lambda_3^e)\ell_3^e \tag{B.64}$$

where ℓ_i^e denotes the length of the i-th edge and $\mathbf{W}_{ij}$ is a vectorial function that connect i-th node to j-th node. Therefore, the vector field within the element can be defined as

$$\mathbf{E}^e = \sum_{i=1}^{3} \mathbf{N}_i^e E_i^e, \tag{B.65}$$

where E_i^e denotes the tangential scalar field along the i-th edge. The gradients $\boldsymbol{\nabla}\lambda_i^e$

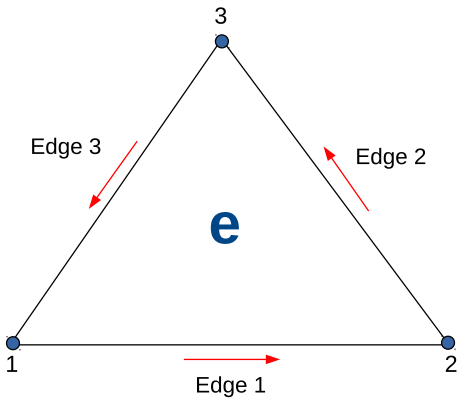


Fig. B.3 Triangular edge element.

can be expanded as

$$\boldsymbol{\nabla}\lambda_1^e = \frac{1}{2\Delta^e}(a_1^e + b_1^e x + c_1^e y) = \frac{1}{2\Delta^e}\begin{bmatrix} b_1^e \cdot \hat{i} \\ c_1^e \cdot \hat{j} \end{bmatrix},$$
$$\boldsymbol{\nabla}\lambda_2^e = \frac{1}{2\Delta^e}(a_2^e + b_2^e x + c_2^e y) = \frac{1}{2\Delta^e}\begin{bmatrix} b_2^e \cdot \hat{i} \\ c_2^e \cdot \hat{j} \end{bmatrix},$$
$$\boldsymbol{\nabla}\lambda_3^e = \frac{1}{2\Delta^e}(a_3^e + b_3^e x + c_3^e y) = \frac{1}{2\Delta^e}\begin{bmatrix} b_3^e \cdot \hat{i} \\ c_3^e \cdot \hat{j} \end{bmatrix}.$$

Therefore, the basis functions for triangular edge element are given by

$$\mathbf{W}_{12}\ell_1^e = \frac{1}{(2\Delta^e)^2}\begin{bmatrix} b_2^e(a_1^e + b_1^e x + c_1^e y) - b_1^e(a_2^e + b_2^e x + c_2^e y) \cdot \hat{i} \\ c_2^e(a_1^e + b_1^e x + c_1^e y) - c_1^e(a_2^e + b_2^e x + c_2^e y) \cdot \hat{j} \end{bmatrix} \ell_1^e, \tag{B.66}$$
$$\mathbf{W}_{23}\ell_2^e = \frac{1}{(2\Delta^e)^2}\begin{bmatrix} b_3^e(a_2^e + b_2^e x + c_2^e y) - b_2^e(a_3^e + b_3^e x + c_3^e y) \cdot \hat{i} \\ c_3^e(a_2^e + b_2^e x + c_2^e y) - c_2^e(a_3^e + b_3^e x + c_3^e y) \cdot \hat{j} \end{bmatrix} \ell_2^e, \tag{B.67}$$
$$\mathbf{W}_{31}\ell_3^e = \frac{1}{(2\Delta^e)^2}\begin{bmatrix} b_1^e(a_3^e + b_3^e x + c_3^e y) - b_3^e(a_1^e + b_1^e x + c_1^e y) \cdot \hat{i} \\ c_1^e(a_3^e + b_3^e x + c_3^e y) - c_3^e(a_1^e + b_1^e x + c_1^e y) \cdot \hat{j} \end{bmatrix} \ell_3^e. \tag{B.68}$$

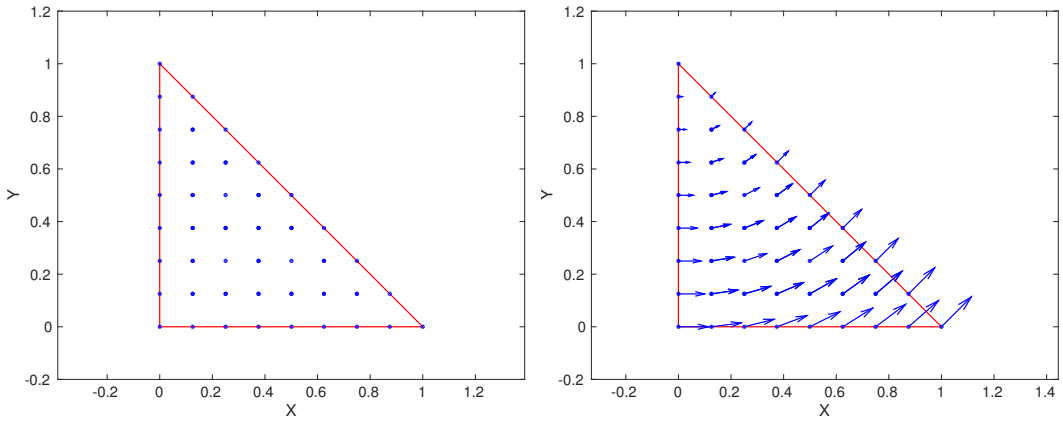
If $i = j = 1, 2, 3$, expressions (B.66) - (B.68) can be written uniformly as

$$\mathbf{W}_{ij}\ell_i^e = \frac{1}{(2\Delta^e)^2}\begin{bmatrix} b_j^e(a_i^e + b_i^e x + c_i^e y) - b_i^e(a_j^e + b_j^e x + c_j^e y) \cdot \hat{i} \\ c_j^e(a_i^e + b_i^e x + c_i^e y) - c_i^e(a_j^e + b_j^e x + c_j^e y) \cdot \hat{j} \end{bmatrix} \ell_i^e. \tag{B.69}$$

Following the notation of table B.2, vector basis functions (B.69) are plotted in figure B.4. From basis functions (B.69) is not difficult to see that they are divergence free but not curl free

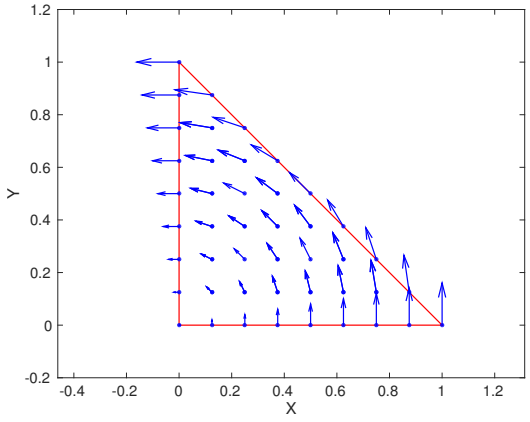
$$\nabla \cdot \mathbf{W}_{ij} = \nabla \cdot (\lambda_1^e \nabla \lambda_2^e) - \nabla \cdot (\lambda_2^e \nabla \lambda_1^e) = 0, \tag{B.70}$$
$$\boldsymbol{\nabla} \times \mathbf{W}_{ij} = \boldsymbol{\nabla} \times (\lambda_1^e \boldsymbol{\nabla} \lambda_2^e) - \boldsymbol{\nabla} \times (\lambda_2^e \boldsymbol{\nabla} \lambda_1^e) = 2\nabla\lambda_1^e \times \lambda_2^e. \tag{B.71}$$

For instance, basis $\mathbf{W}_{12}$ has a constant tangential component and discontinuous normal component along the edge 1. Further, since λ_1^e vanishes along the edge 2 and λ_2^e vanishes along edge 3, $\mathbf{W}_{12}$ has no tangential component along these two edges. Thus, $\mathbf{W}_{12}$ possesses all the necessary properties for being a vector basis function for the edge field assoiated with edge 1. Former properties, which are valid for the remaining edges, are depicted in figure B.5.

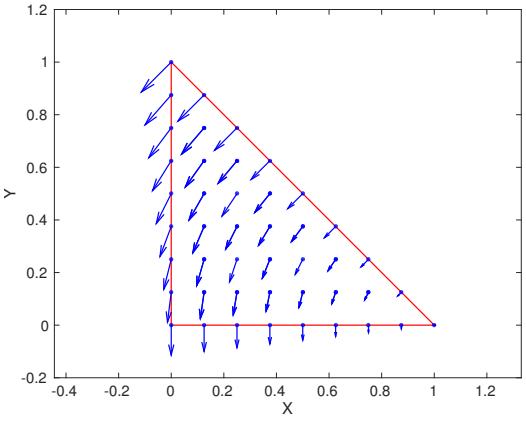


(a) Evaluation points

(b) $\mathbf{W}_{12}$

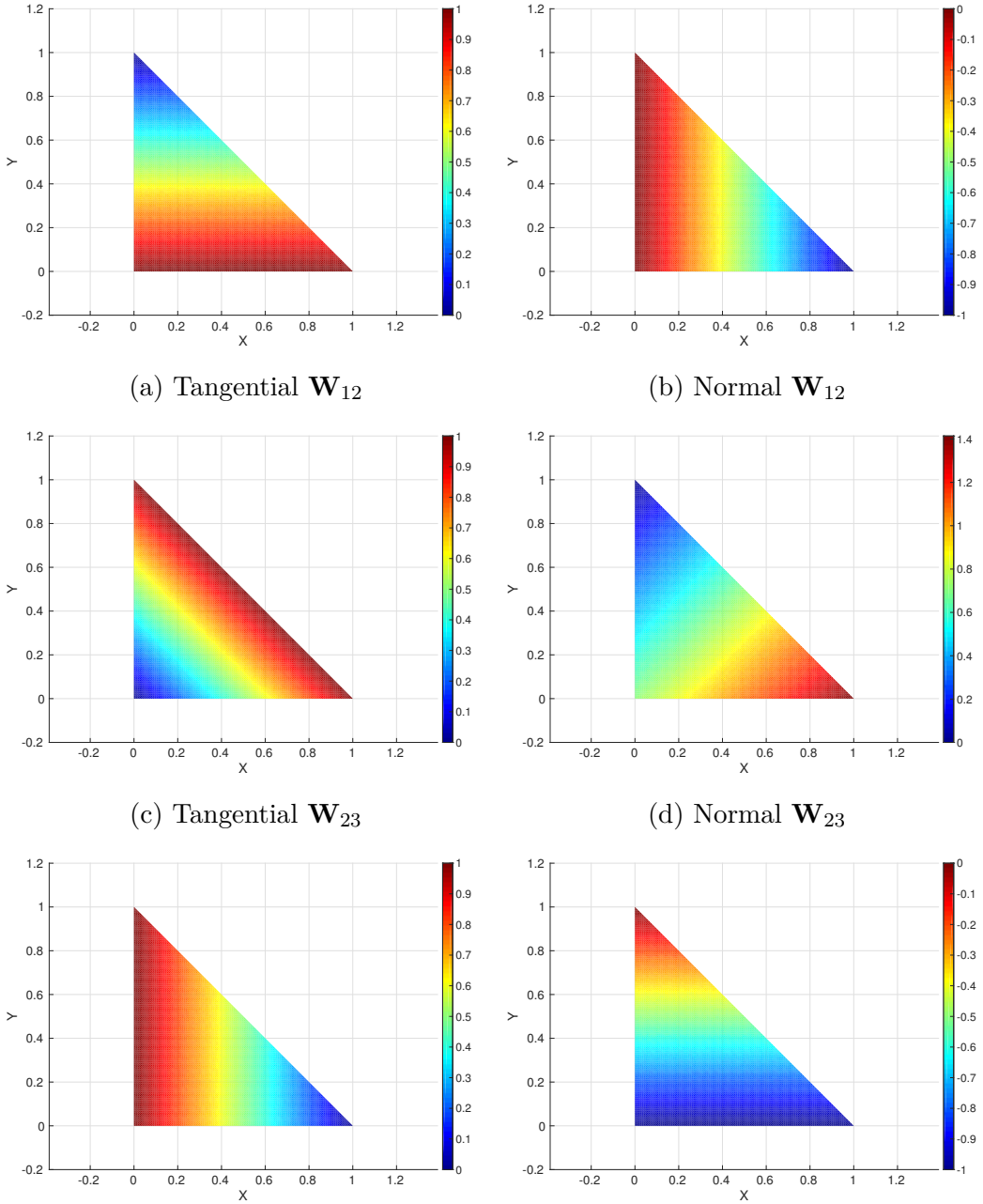


(c) $\mathbf{W}_{23}$



(d) $\mathbf{W}_{31}$

Fig. B.4 Vector basis functions for unitary triangular edge element.



(a) Tangential $\mathbf{W}_{12}$

(b) Normal $\mathbf{W}_{12}$

(c) Tangential $\mathbf{W}_{23}$

(d) Normal $\mathbf{W}_{23}$

(e) Tangential $\mathbf{W}_{31}$

(f) Normal $\mathbf{W}_{31}$

Fig. B.5 Tangential and normal components for triangular edge basis functions.

Elemental equations. When vector basis functions as (B.69) are employed to represent a vector field or potential in FE formulations of vector wave equations, the resulting elemental matrices contain analytical integrals of the following two forms

$$K_{ij}^e = \iint_{\Omega^e} (\boldsymbol{\nabla} \times \mathbf{N}_i^e) \cdot (\boldsymbol{\nabla} \times \mathbf{N}_j^e)\, d\Omega, \tag{B.72}$$

$$M_{ij}^e = \iint_{\Omega^e} \mathbf{N}_i^e \cdot \mathbf{N}_j^e\, d\Omega. \tag{B.73}$$

Integral (B.72) can be reduced to the following expression

$$K_{ij}^e = \frac{\ell_i^e \cdot \ell_j^e}{\Delta^e},$$

where Δ^e denotes the area of the e-th element. On the other hand, integral (B.73) can be defined by

$$\begin{aligned}
M_{11}^e &= \frac{(\ell_1^e)^2}{24\Delta^e} = (m_{22} - m_{12} + m_{11}),\\
M_{21}^e = M_{12}^e &= \frac{\ell_1^e \cdot \ell_2^e}{48\Delta^e} = (m_{23} - m_{22} - 2m_{13} + m_{12}),\\
M_{31}^e = M_{13}^e &= \frac{\ell_1^e \cdot \ell_3^e}{48\Delta^e} = (m_{21} - 2m_{23} - m_{11} + m_{13}),\\
M_{22}^e &= \frac{(\ell_2^e)^2}{24\Delta^e} = (m_{33} - m_{23} + m_{22}),\\
M_{32}^e = M_{23}^e &= \frac{\ell_2^e \cdot \ell_3^e}{48\Delta^e} = (m_{31} - m_{33} - 2m_{21} + m_{23}),\\
M_{33}^e &= \frac{(\ell_3^e)^2}{24\Delta^e} = (m_{11} - m_{13} + m_{33}),
\end{aligned}$$

where $m_{ij} = b_i^e b_j^e + c_i^e c_j^e$, with b_i^e and c_i^e defined in the formulation of (B.25).

Three-dimensional Edge Finite Element Method

The formulation of edge elements for 2D cases can be extended to 3D cases in a straightforward manner. Similarly to nodal FEM in 3D, in the EFEM the volume V is preferably subdivided into tetrahedrons since this enables the representation of complex geological structures and also allows local refinement in order to improve the solution's accuracy. Therefore, the formulation for the tetrahedral edge elements is considered here.

Interpolation functions. In concordance with the essence of EFEM, the DOFs

for tetrahedral edge elements of lowest order are associated with their edges as show in figure B.6. Their edge basis functions are based on the barycentric coordinates defined by expression (B.53) which here is renamed as $\boldsymbol{\lambda}_i^e = \mathbf{N}_i^e$, therefore the edge basis functions for tetrahedral edge elements can be expressed in terms of first order of node-based Finite Elements (Jin, 2002). For the tetrahedral element of figure B.6

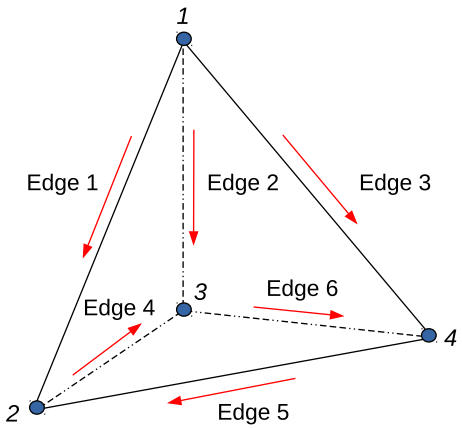


Fig. B.6 Tetrahedral edge element.

with edges defined by table B.3, their basis functions can be expressed by the following vectorial functions:

$$\mathbf{N}_1^e = \mathbf{W}_{12}\ell_1^e = (\lambda_1^e\boldsymbol{\nabla}\lambda_2^e - \lambda_2^e\boldsymbol{\nabla}\lambda_1^e)\ell_1^e, \tag{B.74}$$

$$\mathbf{N}_2^e = \mathbf{W}_{13}\ell_2^e = (\lambda_1^e\boldsymbol{\nabla}\lambda_3^e - \lambda_3^e\boldsymbol{\nabla}\lambda_1^e)\ell_2^e, \tag{B.75}$$

$$\mathbf{N}_3^e = \mathbf{W}_{14}\ell_3^e = (\lambda_1^e\boldsymbol{\nabla}\lambda_4^e - \lambda_4^e\boldsymbol{\nabla}\lambda_1^e)\ell_3^e,, \tag{B.76}$$

$$\mathbf{N}_4^e = \mathbf{W}_{23}\ell_4^e = (\lambda_2^e\boldsymbol{\nabla}\lambda_3^e - \lambda_3^e\boldsymbol{\nabla}\lambda_2^e)\ell_4^e, \tag{B.77}$$

$$\mathbf{N}_5^e = \mathbf{W}_{42}\ell_5^e = (\lambda_4^e\boldsymbol{\nabla}\lambda_2^e - \lambda_2^e\boldsymbol{\nabla}\lambda_4^e)\ell_5^e, \tag{B.78}$$

$$\mathbf{N}_6^e = \mathbf{W}_{34}\ell_6^e = (\lambda_3^e\boldsymbol{\nabla}\lambda_4^e - \lambda_4^e\boldsymbol{\nabla}\lambda_3^e)\ell_6^e. \tag{B.79}$$

The gradients $\boldsymbol{\nabla}\lambda_i^e$ can be expanded as

edge	i_1	i_2
1	1	2
2	1	3
3	1	4
4	2	3
5	4	2
6	3	4

Table B.3 Edges definition: tetrahedral element

$$\boldsymbol{\nabla}\lambda_1^e = \frac{1}{6V^e}\begin{bmatrix} b_1^e \cdot \hat{i} \\ c_1^e \cdot \hat{j} \\ d_1^e \cdot \hat{k} \end{bmatrix}, \qquad \boldsymbol{\nabla}\lambda_2^e = \frac{1}{6V^e}\begin{bmatrix} b_2^e \cdot \hat{i} \\ c_2^e \cdot \hat{j} \\ d_2^e \cdot \hat{k} \end{bmatrix},$$

$$\boldsymbol{\nabla}\lambda_3^e = \frac{1}{6V^e}\begin{bmatrix} b_3^e \cdot \hat{i} \\ c_3^e \cdot \hat{j} \\ d_3^e \cdot \hat{k} \end{bmatrix}, \qquad \boldsymbol{\nabla}\lambda_4^e = \frac{1}{6V^e}\begin{bmatrix} b_4^e \cdot \hat{i} \\ c_4^e \cdot \hat{j} \\ d_4^e \cdot \hat{k} \end{bmatrix},$$

$$\boldsymbol{\nabla}\lambda_5^e = \frac{1}{6V^e}\begin{bmatrix} b_5^e \cdot \hat{i} \\ c_5^e \cdot \hat{j} \\ d_5^e \cdot \hat{k} \end{bmatrix}, \qquad \boldsymbol{\nabla}\lambda_6^e = \frac{1}{6V^e}\begin{bmatrix} b_6^e \cdot \hat{i} \\ c_6^e \cdot \hat{j} \\ d_6^e \cdot \hat{k} \end{bmatrix}.$$

Therefore, the expanded form of the edge basis functions for tetrahedral edge elements is the following

$$\mathbf{W}_{12}\ell_1^e = \frac{1}{(6V^e)^2}\begin{bmatrix} b_2^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - b_1^e(a_2^e + b_2^e x + c_2^e y + d_2^e z)\cdot\hat{i} \\ c_2^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - c_1^e(a_2^e + b_2^e x + c_2^e y + d_2^e z)\cdot\hat{j} \\ d_2^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - d_1^e(a_2^e + b_2^e x + c_2^e y + d_2^e z)\cdot\hat{k} \end{bmatrix}\ell_1, \quad \text{(B.80)}$$

$$\mathbf{W}_{13}\ell_2^e = \frac{1}{(6V^e)^2}\begin{bmatrix} b_3^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - b_1^e(a_3^e + b_3^e x + c_3^e y + d_3^e z)\cdot\hat{i} \\ c_3^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - c_1^e(a_3^e + b_3^e x + c_3^e y + d_3^e z)\cdot\hat{j} \\ d_3^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - d_1^e(a_3^e + b_3^e x + c_3^e y + d_3^e z)\cdot\hat{k} \end{bmatrix}\ell_2, \quad \text{(B.81)}$$

$$\mathbf{W}_{14}\ell_3^e = \frac{1}{(6V^e)^2}\begin{bmatrix} b_4^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - b_1^e(a_4^e + b_4^e x + c_4^e y + d_4^e z)\cdot\hat{i} \\ c_4^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - c_1^e(a_4^e + b_4^e x + c_4^e y + d_4^e z)\cdot\hat{j} \\ d_4^e(a_1^e + b_1^e x + c_1^e y + d_1^e z) - d_1^e(a_4^e + b_4^e x + c_4^e y + d_4^e z)\cdot\hat{k} \end{bmatrix}\ell_3, \quad \text{(B.82)}$$

$$\mathbf{W}_{23}\ell_4^e = \frac{1}{(6V^e)^2}\begin{bmatrix} b_3^e(a_2^e + b_2^e x + c_2^e y + d_2^e z) - b_2^e(a_3^e + b_3^e x + c_3^e y + d_3^e z)\cdot\hat{i} \\ c_3^e(a_2^e + b_2^e x + c_2^e y + d_2^e z) - c_2^e(a_3^e + b_3^e x + c_3^e y + d_3^e z)\cdot\hat{j} \\ d_3^e(a_2^e + b_2^e x + c_2^e y + d_2^e z) - d_2^e(a_3^e + b_3^e x + c_3^e y + d_3^e z)\cdot\hat{k} \end{bmatrix}\ell_4, \quad \text{(B.83)}$$

$$\mathbf{W}_{42}\ell_5^e = \frac{1}{(6V^e)^2}\begin{bmatrix} b_2^e(a_4^e + b_4^e x + c_4^e y + d_4^e z) - b_4^e(a_2^e + b_2^e x + c_2^e y + d_2^e z)\cdot\hat{i} \\ c_2^e(a_4^e + b_4^e x + c_4^e y + d_4^e z) - c_4^e(a_2^e + b_2^e x + c_2^e y + d_2^e z)\cdot\hat{j} \\ d_2^e(a_4^e + b_4^e x + c_4^e y + d_4^e z) - d_4^e(a_2^e + b_2^e x + c_2^e y + d_2^e z)\cdot\hat{k} \end{bmatrix}\ell_5, \quad \text{(B.84)}$$

$$\mathbf{W}_{34}\ell_6^e = \frac{1}{(6V^e)^2}\begin{bmatrix} b_4^e(a_3^e + b_3^e x + c_3^e y + d_3^e z) - b_3^e(a_4^e + b_4^e x + c_4^e y + d_4^e z)\cdot\hat{i} \\ c_4^e(a_3^e + b_3^e x + c_3^e y + d_3^e z) - c_3^e(a_4^e + b_4^e x + c_4^e y + d_4^e z)\cdot\hat{j} \\ d_4^e(a_3^e + b_3^e x + c_3^e y + d_3^e z) - d_3^e(a_4^e + b_4^e x + c_4^e y + d_4^e z)\cdot\hat{k} \end{bmatrix}\ell_6. \quad \text{(B.85)}$$

Edge basis functions (B.80) - (B.85) are divergence free but not curl free (Jin, 2002).

Elemental equations. When edge basis functions (B.80) - (B.85) are employed to represent a vector field in FE formulations of vector wave equations, the resulting elemental matrices contain analytical integrals of the following two forms (Jin, 2002)

$$K_{ij}^e = \iiint_{V^e} (\boldsymbol{\nabla}\times\mathbf{N}_i^e)\cdot(\boldsymbol{\nabla}\times\mathbf{N}_j^e)\,dV, \quad \text{(B.86)}$$

$$M_{ij}^e = \iiint_{V^e} \mathbf{N}_i^e\cdot\mathbf{N}_j^e\,dV. \quad \text{(B.87)}$$

According with the numerical formulation by (Jin, 2002), the integral (B.86) can be reduced to the following expression

$$\begin{aligned} K_{ij}^e = \frac{4\ell_i^e \ell_j^e V^e}{(6V^e)^4}[&(c_{i1}^e d_{i2}^e - d_{i1}^e c_{i2}^e)(c_{j1}^e d_{j2}^e - d_{j1}^e c_{j2}^e) \\ &+ (d_{i1}^e b_{i2}^e - b_{i1}^e d_{i2}^e)(d_{j1}^e b_{j2}^e - b_{j1}^e d_{j2}^e) \\ &+ (b_{i1}^e c_{i2}^e - c_{i1}^e b_{i2}^e)(b_{j1}^e c_{j2}^e - c_{j1}^e b_{j2}^e)], \end{aligned} \tag{B.88}$$

where V^e denotes the volume of the element e-th, and the coefficients b_{ij}^e, c_{ij}^e and d_{ij}^e are defined in (B.49), (B.50) and (B.51), respectively . On the other hand, integral (B.87) is given by

$$\begin{aligned} M_{11}^e &= \frac{(\ell_1^e)^2}{360V^e}(m_{22} - m_{12} + m_{11}), \\ M_{21}^e = M_{12}^e &= \frac{\ell_1^e \ell_2^e}{720V^e}(2m_{23} - m_{12} - m_{13} + m_{11}), \\ M_{31}^e = M_{13}^e &= \frac{\ell_1^e \ell_3^e}{720V^e}(2m_{24} - m_{12} - m_{14} + m_{11}), \\ M_{41}^e = M_{14}^e &= \frac{\ell_1^e \ell_4^e}{720V^e}(m_{23} - m_{22} - 2m_{13} + m_{12}), \\ M_{51}^e = M_{15}^e &= \frac{\ell_1^e \ell_5^e}{720V^e}(m_{22} - m_{24} - m_{12} + 2m_{14}), \\ M_{61}^e = M_{16}^e &= \frac{\ell_1^e \ell_6^e}{720V^e}(m_{24} - m_{23} - m_{14} + m_{13}), \end{aligned}$$

$$\begin{aligned} M_{22}^e &= \frac{(\ell_2^e)^2}{360V^e}(m_{33} - m_{13} + m_{11}), \\ M_{32}^e = M_{23}^e &= \frac{\ell_2^e \ell_3^e}{720V^e}(2m_{34} - m_{13} - m_{14} + m_{11}), \\ M_{42}^e = M_{24}^e &= \frac{\ell_2^e \ell_4^e}{720V^e}(m_{33} - m_{23} - m_{13} + 2m_{12}), \\ M_{52}^e = M_{25}^e &= \frac{\ell_2^e \ell_5^e}{720V^e}(m_{23} - m_{34} - m_{12} + m_{14}), \\ M_{62}^e = M_{26}^e &= \frac{\ell_2^e \ell_6^e}{720V^e}(m_{13} - m_{33} - 2m_{14} + m_{34}), \end{aligned}$$

$$M^e_{33} = \frac{(\ell^e_3)^2}{360V^e}(m_{44} - m_{14} + m_{11}),$$
$$M^e_{43} = M^e_{34} = \frac{\ell^e_3\ell^e_4}{720V^e}(m_{34} - m_{24} - m_{13} + m_{12}),$$
$$M^e_{53} = M^e_{35} = \frac{\ell^e_3\ell^e_5}{720V^e}(m_{24} - m_{44} - 2m_{12} + m_{14}),$$
$$M^e_{63} = M^e_{36} = \frac{\ell^e_3\ell^e_6}{720V^e}(m_{44} - m_{34} - m_{14} + 2m_{13}),$$

$$M^e_{44} = \frac{(\ell^e_4)^2}{360V^e}(m_{33} - m_{23} + m_{22}),$$
$$M^e_{54} = M^e_{45} = \frac{\ell^e_4\ell^e_5}{720V^e}(m_{23} - 2m_{34} - m_{22} + m_{24}),$$
$$M^e_{64} = M^e_{46} = \frac{\ell^e_4\ell^e_6}{720V^e}(m_{34} - m_{33} - 2m_{24} + m_{23}),$$

$$M^e_{55} = \frac{(\ell^e_5)^2}{360V^e}(m_{22} - m_{24} + m_{44}),$$
$$M^e_{65} = M^e_{56} = \frac{\ell^e_5\ell^e_6}{720V^e}(m_{24} - 2m_{23} - m_{44} + m_{34}),$$

$$M^e_{66} = \frac{(\ell^e_6)^2}{360V^e}(m_{44} - m_{34} + m_{33}),$$

where $m_{ij} = b^e_i b^e_j + c^e_i c^e_j + d^e_i d^e_j$ with b^e_i, c^e_i and d^e_i defined by (B.49), (B.50) and (B.51) respectively.

B.3 Test case of EFEM

In this section, we study the properties, performance and accuracy of edge elements. To do that, we implemented a simple, flexible and parallel Fortran prototype for edge elements applied to simple problems. We focus in assembly time, solving time and convergence order. The mathematical formulation is based on ideas from (Rognes

et al., 2009; Anjam and Valdman, 2015). The experiments were performed on one node of the Marenostrum supercomputer with two 8-core Intel Xeon processors $E5-2670$ at 2.6 GHz. The BLAS library was compiled and linked with the code in order to use the SGESV subroutine as solver.

Experiments of edge elements for eddy-current

As already said, in a standard BVP, the domain Ω can be split into two sections $\partial\Omega = \Omega_D + \Omega_N$ such that $\Omega_D \cap \Omega_N = \emptyset$. Therefore, the 2D eddy-current problem can be expressed as

$$\nabla \times \mu^{-1}\nabla \times E + \kappa E = F \text{ on } \Omega, \tag{B.89}$$

$$E \times n = 0 \text{ on } \Omega_D, \tag{B.90}$$

$$\mu^{-1}\nabla \times E = 0 \text{ on } \Omega_N, \tag{B.91}$$

for searched function $E \in H_{\Omega_D}(\nabla, \Omega) = v \in H(\nabla, \Omega)|v \times n = 0 \text{ on } \Omega_D$, where n denotes the outward unit normal to the boundary $\partial\Omega$. Here the right hand side $F \in L^2(\Omega, \Re^2)$, and the material parameters μ and κ are given.

According with generalized formulation of expressions (B.89) - (B.91) by (Anjam and Valdman, 2015), we choose the unit square $\Omega = [0,1]^2$ with material parameters $\mu = \kappa = 1.5$. The domain Ω is divided into two sections $\Omega_1 = x \in \Omega | x_1 > x_2$ and $\Omega_2 = \Omega \backslash \bar{\Omega_1}$ in order to define the following discontinuous exact solution

$$E_{\Omega_1}(x) = \begin{bmatrix} \sin(2\pi x_1) + 2\pi\cos(2\pi x_1)(x_1 - x_2) \\ \sin((x_1 - x_2)^2(x_1 - 1)^2 x_2) - \sin(2\pi x_1) \end{bmatrix},$$
$$E_{\Omega_2}(x) = 0.$$

Because the exact solution has a continuous tangential component at $x_1 = x_2$, $E \in (\nabla, \Omega)$. Namely, the exact solution satisfies full Neumann BC (Anjam and Valdman, 2015), therefore, $\Omega_D = \emptyset$ and $\Omega_N = \partial\Omega$.

All tests are based on a regular triangular mesh (figure B.1) with uniform refinement. This strategy produces 4 times more elements in the domain. Table B.4 depicts the performance of the code with 16 OpenMP threads. First column of table B.4 represents the number of elements, second column represents the number of edges in the domain Ω, third and fourth column expresses the assembly and solver time in seconds respectively; fifth column represents the mesh spacing, sixth and seventh column represents the L^2 error and the convergence order $\mathcal{O}_{L^2}$ respectively. The L^2

e	edges	Assembly	Solver	h	L^2	$\mathcal{O}_{L^2}$
128	208	7.66×10^{-3}	6.76×10^{-3}	3.3×10^{-1}	3.9×10^{-1}	–
512	800	6.75×10^{-3}	7.72×10^{-3}	2.0×10^{-1}	9.8×10^{-2}	2.016
2,048	3,136	2.22×10^{-2}	4.47×10^{-2}	1.1×10^{-1}	2.4×10^{-2}	2.017
8,192	12,416	1.82×10^{-1}	2.08×10^{-1}	5.8×10^{-2}	6.0×10^{-3}	1.999
32,768	49,408	4.27×10^{-1}	6.68×10^{-1}	3.0×10^{-2}	1.5×10^{-3}	2.004
131,072	197,120	0.151×10^{1}	0.227×10^{1}	1.5×10^{-2}	3.7×10^{-4}	2.002
524,288	787,456	0.583×10^{1}	0.671×10^{1}	7.8×10^{-3}	9.3×10^{-5}	2.002
2,097,152	3,147,776	0.213×10^{2}	0.247×10^{2}	3.9×10^{-3}	2.3×10^{-5}	2.001

Table B.4 Summary of results for 2D eddy-current problem

error of table B.4 is plotted versus mesh spacing h in figure B.7. On the other hand, figure B.8 depicts the discrete solution of eddy-current problem for a mesh with 32,768 triangular elements which produces 49,408 edges. Is not difficult to see that vector

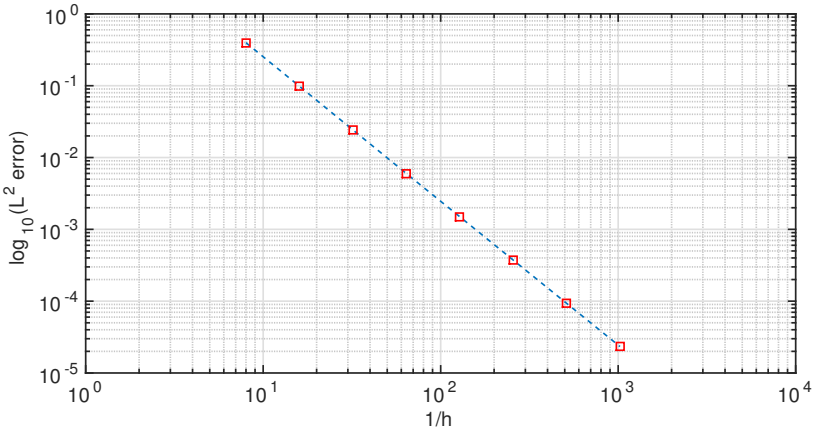
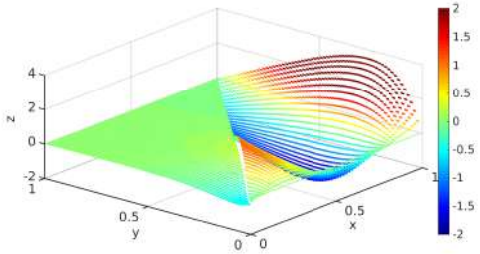
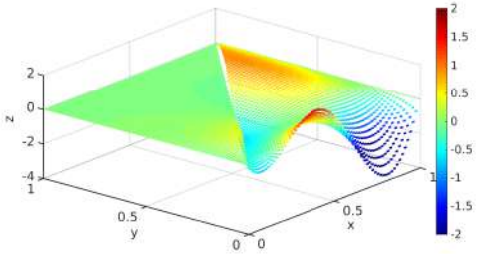


Fig. B.7 Convergence results of edge elements (lowest order) for eddy current in 2D. The L^2 error is plotted versus the mesh spacing h.

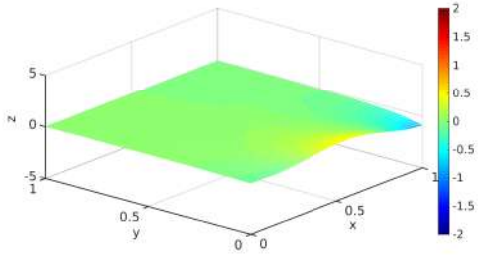
fields in figure B.8 satisfies the divergence condition (B.70) and curl condition (B.71). Different approaches to solve eddy-current problems with edge elements are described in (Beck and Hiptmair, 1999; Hiptmair, 2015). Furthermore, in (Webb, 1993; Biro et al., 1996; Monk, 2003) the authors explain the capacity of edge elements to avoid spurious solutions in an ample range of applications.



(a) x-component



(b) y-component



(c) ∇E

Fig. B.8 Discrete solution for eddy-current problem in 2D. Mesh with 32,768 triangular elements and 49,408 edges.

Appendix C

Prototyping and validation with synthetic test

As we mentioned in Appendix B, nodal-based finite elements can not be used directly for electromagnetic problems formulated in terms of the electric field (**E**) and/or magnetic field (**H**) functions, which is a natural and physically meaningful problem formulation. Although there are other schemes to solve the problem, for instance vector-scalar electromagnetic potential functions by (Koldan, 2013), we decide formulate the physical problems in terms of electric field decomposition (Nabighian, 1988; Zhdanov, 2009; Cai et al., 2014) for a 3D CSEM FM using EFEM scheme. This approach solves not only the already mentioned problems of FEM, but also the problem of having singularities introduced by sources. As consequence, it is numerically very stable.

On top on that, in this appendix we study the key aspects for the numerical solution of equation systems that arises from the 3D CSEM FM, namely, elemental matrices computations, vector basis implementation, global system assembly and its solution.

C.1 Prototype for 3D CSEM modelling

Before to design the HPC implementation, we implemented a modular, flexible and parallel (OpenMP) Matlab prototype for the numerical solutions of electromagnetic fields in 3D CSEM FM surveys, whose software stack is show in figure C.1. Since the aim of this part of our research was dedicated to the analysis of the numerical formulation instead the code performance issues, all experiments were performed on a modest cluster with 24 Intel Xeon processors E5-2620 at 2.10 *GHz*. Furthermore, all results were obtained by using a direct solver.

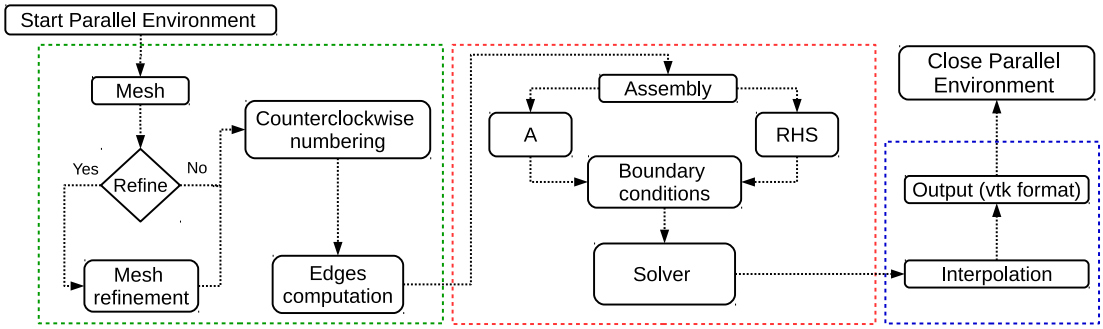


Fig. C.1 Upper view of software stack. Green dashed: Pre-processing stage, Red dashed: Forward modelling, Blue dashed: Post-processing stage.

C.1.1 Synthetic test for mass matrix

In order to ensure the tangential continuity of the fields and to obtain a consistent assembly of the system under consideration, stiffness matrix (B.86) and mass matrix (B.87) must be redefined as follows

$$K_{ij}^{e} = \iiint_{V^e} (\boldsymbol{\nabla} \times \mathbf{N}_i^e S_i^e) \cdot (\boldsymbol{\nabla} \times \mathbf{N}_j^e S_j^e)\, dV, \tag{C.1}$$

$$M_{ij}^{e} = \iiint_{V^e} (\mathbf{N}_i^e S_i^e) \cdot (\mathbf{N}_j^e S_j^e)\, dV, \tag{C.2}$$

where, again, S_i^e are coefficients equal 1 or -1 depending on the local and global direction of the *i-th* edge in the element e. Latter is an important consideration for the assembly of elemental matrices, it does not matter if these are evaluated analytically or numerically.

In order to verify the correct computation of mass matrices (C.2), we solve an equation system of the form $[A] \cdot \{\phi\} = \{b\}$, where matrix $[A]$ is assembling through the sum of elemental matrices $[A^e = M^e]$, where M^e is the mass matrix from expression (C.2), and vector $\{b\}$, or right hand side (RHS) is populate by the sum of elemental contributions for all tetrahedral in the mesh (Burnett, 1987; Jin, 2002). The field to be tested is defined for real and complex polynomial functions as follows

Function 1: Real of 1st order

$$\mathbf{F} = \begin{bmatrix} (ax+b)\cdot\hat{i} \\ (cy+d)\cdot\hat{j} \\ (ez+f)\cdot\hat{k} \end{bmatrix}, \tag{C.3}$$

Function 2: Complex of 1st order

$$\mathbf{F} = \begin{bmatrix} (ax + bi) \cdot \hat{i} \\ (cy + di) \cdot \hat{j} \\ (ez + fi) \cdot \hat{k} \end{bmatrix}, \tag{C.4}$$

Function 3: Real of 2nd order

$$\mathbf{F} = \begin{bmatrix} (ax^2 + bx + c + dy^2 + ey + fxy) \cdot \hat{i} \\ (cx^2 + ax + b + ey^2 + fy + dxy) \cdot \hat{j} \\ (dx^2 + ex + a + fy^2 + by + cxy) \cdot \hat{k} \end{bmatrix}, \tag{C.5}$$

Function 4: Complex of 2nd order

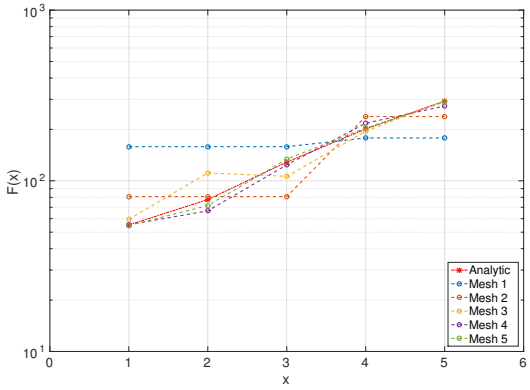
$$\mathbf{F} = \begin{bmatrix} (ax^2 + bx + ci + dy^2 + eyi + fxyi) \cdot \hat{i} \\ (cx^2 + ax + bi + ey^2 + fyi + dxyi) \cdot \hat{j} \\ (dx^2 + ex + ai + fy^2 + byi + cxyi) \cdot \hat{k} \end{bmatrix}, \tag{C.6}$$

where coefficient values are determined as $a = 2.5$, $b = 7$, $c = 1.7$, $d = -2.1$, $e = 2$ and $f = 5$. For all tests we used an uniform refinement technique in order to study the convergence of the solution, the result of this process are five levels of meshes with detailed information in table C.1. After numerical computation of mass matrices (C.2), assembling, solving and interpolation process, we obtained a convergence order equal to 1 for all tests which is consistent with the lowest order of edge elements. Errors per component in norm L^1, L^2 and L^{inf} are depicted in table C.2. The convergence orders associated to each error of table C.2 are show in table C.3.

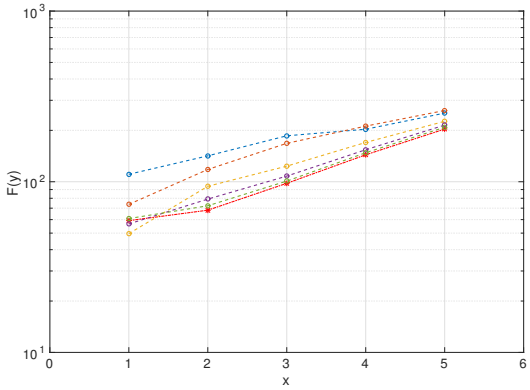
In sake of clarity, figure C.2 shows the field traces of function (C.6) for each mesh. Is easy to see the importance of mesh refinement in order to capture the rapid change of the field. As result, mesh 5 represent the best approximation. Convergence orders per component associated with figure C.2 are plotted versus mesh spacing h in figure C.3. Results show that computation of mass matrices and their assembly is correct.

C.1.2 Synthetic test for stiffness matrix

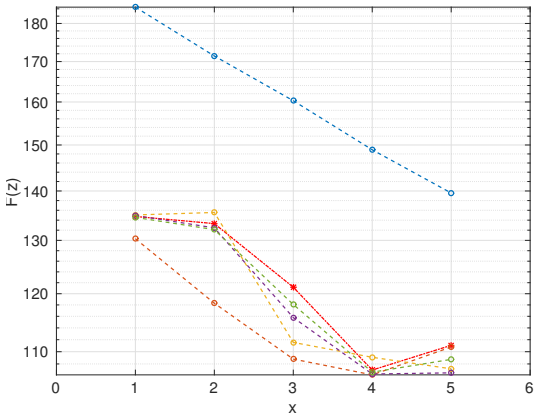
The process to verify the computation and assembly of stiffness matrix is similar to previous one. Hence we solved an equation system of the form $[A] \cdot \{\phi\} = \{b\}$, where matrix $[A]$ is assembling through the sum of elemental matrices $[A^e = K^e + M^e]$, where K^e is the stiffness matrix from expression (C.1) and M^e is the mass matrix from equation (C.2) and vector $\{b\}$ is assembly by the same way than previous one.



(a) x-component

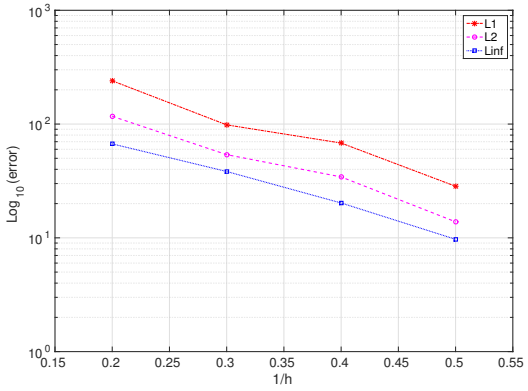


(b) y-component

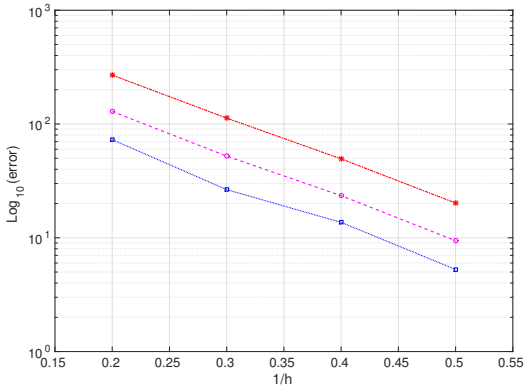


(c) z-component

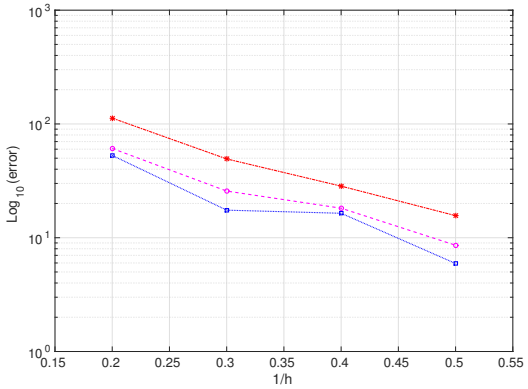
Fig. C.2 Mass matrix: trace fields for test equation C.6.



(a) x-component



(b) y-component



(c) z-component

Fig. C.3 Mass matrix: convergence order for test equation C.6.

Mesh level	Nodes	Elements	Edges
1	8	6	19
2	27	48	98
3	125	384	604
4	729	3,072	4,184
5	4,913	24,576	31,024

Table C.1 Levels of meshes

Therefore, the field to be tested is redefined from (C.6) as follows

Function 1: Complex of 2nd order

$$\mathbf{G} = \boldsymbol{\nabla} \times \boldsymbol{\nabla} \times \mathbf{F} + \mathbf{F} = \begin{bmatrix} (-2d + di + ax^2 + bx + ci + dy^2 + eyi + fxyi) \cdot \hat{i} \\ (-2c + fi + cx^2 + ax + bi + ey^2 + fyi + dxyi) \cdot \hat{j} \\ (-2d - 2f + dx^2 + ex + ai + fy^2 + byi + cxyi) \cdot \hat{k} \end{bmatrix}, \tag{C.7}$$

where coefficient values are determined as $a = 1.7$, $b = 4$, $c = -2.8$, $d = 8.5$, $e = -1.6$ and $f = -2.3$. Stiffness matrix test is also based on meshes from table C.1. Similar to mass matrices tests, we obtained a convergence order equal to 1. Errors and convergence orders are depicted in table C.4 and table C.5 respectively. Similar to mass matrices tests, figure C.4 shows the field traces of function (C.7) for each mesh and again, the mesh refinement plays an important role in the solution accuracy, hence the best approximation was obtained by using mesh 5. Convergence orders per component associated with results of figure C.4 are plotted versus mesh spacing h in figure C.5. Results show that computation of stiffness matrices and their assembly is correct.

Function 1									
Mesh	L^1			L^2			L^{inf}		
	x	y	z	x	y	z	x	y	z
1	1.91×10^1	1.68×10^1	1.44×10^1	1.06×10^1	0.78×10^1	0.72×10^1	0.91×10^1	0.45×10^1	0.53×10^1
2	1.54×10^1	1.83×10^1	2.05×10^1	0.80×10^1	0.83×10^1	0.93×10^1	0.54×10^1	0.43×10^1	0.51×10^1
3	0.801×10^1	1.07×10^1	1.12×10^1	0.433×10^1	0.482×10^1	0.502×10^1	0.291×10^1	0.238×10^1	0.249×10^1
4	0.438×10^1	0.545×10^1	0.610×10^1	0.216×10^1	0.244×10^1	0.273×10^1	0.155×10^1	0.119×10^1	0.125×10^1
5	0.188×10^1	0.274×10^1	0.304×10^1	0.098×10^1	0.122×10^1	0.136×10^1	0.078×10^1	0.058×10^1	0.062
Function 2									
Mesh	L^1			L^2			L^{inf}		
	x	y	z	x	y	z	x	y	z
1	1.915×10^1	1.683×10^1	1.448×10^1	1.068×10^1	0.786×10^1	0.720×10^1	0.915×10^1	0.458×10^1	0.533×10^1
2	1.546×10^1	1.830×10^1	2.051×10^1	0.806×10^1	0.830×10^1	0.936×10^1	0.546×10^1	0.431×10^1	0.519×10^1
3	0.805×10^1	1.072×10^1	1.126×10^1	0.431×10^1	0.483×10^1	0.505×10^1	0.299×10^1	0.238×10^1	0.249×10^1
4	0.438×10^1	0.545×10^1	0.610×10^1	0.216×10^1	0.244×10^1	0.273×10^1	0.155×10^1	0.119×10^1	0.125×10^1
5	0.188×10^1	0.274×10^1	0.304×10^1	0.098×10^1	0.122×10^1	0.136×10^1	0.078×10^1	0.058×10^1	0.062×10^1
Function 3									
Mesh	L^1			L^2			L^{inf}		
	x	y	z	x	y	z	x	y	z
1	62.53×10^1	24.42×10^1	35.97×10^1	32.45×10^1	13.21×10^1	16.38×10^1	22.52×10^1	9.610×10^1	9.488×10^1
2	33.05×10^1	25.47×10^1	43.80×10^1	16.13×10^1	12.94×10^1	27.75×10^1	9.258×10^1	8.804×10^1	20.74×10^1
3	14.30×10^1	11.33×10^1	16.98×10^1	8.07×10^1	5.27×10^1	9.81×10^1	6.124×10^1	2.884×10^1	6.893×10^1
4	9.40×10^1	5.56×10^1	15.62×10^1	4.71×10^1	2.86×10^1	7.28×10^1	2.719×10^1	2.081×10^1	3.793×10^1
5	3.65×10^1	3.34×10^1	4.56×10^1	1.79×10^1	1.52×10^1	2.63×10^1	1.25×10^1	0.816×10^1	1.863×10^1
Function 4									
Mesh	L^1			L^2			L^{inf}		
	x	y	z	x	y	z	x	y	z
1	44.67×10^1	28.15×10^1	30.80×10^1	23.20×10^1	13.04×10^1	14.50×10^1	16.02×10^1	7.70×10^1	9.925×10^1
2	23.90×10^1	28.18×10^1	39.17×10^1	11.65×10^1	13.41×10^1	22.69×10^1	6.67×10^1	7.52×10^1	16.682×10^1
3	13.00×10^1	10.49×10^1	15.01×10^1	5.76×10^1	5.05×10^1	7.95×10^1	4.33×10^1	2.85×10^1	5.466×10^1
4	6.82×10^1	5.26×10^1	12.67×10^1	3.42×10^1	2.56×10^1	5.74×10^1	2.01×10^1	1.56×10^1	2.902×10^1
5	2.65×10^1	2.88×10^1	3.94×10^1	1.28×10^1	1.32×10^1	2.07×10^1	0.887×10^1	0.751×10^1	1.396×10^1

Table C.2 Mass matrix: errors per field component

Function 1									
Mesh	$\mathcal{O}_{L^1}$			$\mathcal{O}_{L^2}$			$\mathcal{O}_{L^{inf}}$		
	x	y	z	x	y	z	x	y	z
1	—	—	—	—	—	—	—	—	—
2	0.015×10^1	-0.006×10^1	-0.025×10^1	0.020×10^1	-0.004×10^1	-0.018×10^1	0.037×10^1	0.004×10^1	0.001×10^1
3	0.094×10^1	0.077×10^1	0.086×10^1	0.090×10^1	0.078×10^1	0.089×10^1	0.084×10^1	0.085×10^1	0.105×10^1
4	0.087×10^1	0.097×10^1	0.088×10^1	0.099×10^1	0.098×10^1	0.088×10^1	0.094×10^1	0.099×10^1	0.099×10^1
5	0.121×10^1	0.099×10^1	0.100×10^1	0.113×10^1	0.099×10^1	0.100×10^1	0.099×10^1	0.103×10^1	0.100×10^1
Function 2									
Mesh	$\mathcal{O}_{L^1}$			$\mathcal{O}_{L^2}$			$\mathcal{O}_{L^{inf}}$		
	x	y	z	x	y	z	x	y	z
1	—	—	—	—	—	—	—	—	—
2	0.015×10^1	-0.006×10^1	-0.025×10^1	0.020×10^1	-0.004×10^1	-0.018×10^1	0.037×10^1	0.004×10^1	0.001×10^1
3	0.094×10^1	0.077×10^1	0.086×10^1	0.090×10^1	0.078×10^1	0.089×10^1	0.086×10^1	0.085×10^1	0.105×10^1
4	0.087×10^1	0.097×10^1	0.088×10^1	0.099×10^1	0.098×10^1	0.088×10^1	0.094×10^1	0.099×10^1	0.099×10^1
5	0.121×10^1	0.099×10^1	0.100×10^1	0.113×10^1	0.099×10^1	0.100×10^1	0.099×10^1	0.103×10^1	0.100×10^1
Function 3									
Mesh	$\mathcal{O}_{L^1}$			$\mathcal{O}_{L^2}$			$\mathcal{O}_{L^{inf}}$		
	x	y	z	x	y	z	x	y	z
1	—	—	—	—	—	—	—	—	—
2	0.045×10^1	-0.003×10^1	-0.014×10^1	0.050×10^1	0.001×10^1	-0.038×10^1	0.064×10^1	0.006×10^1	-0.056×10^1
3	0.120×10^1	0.116×10^1	0.136×10^1	0.099×10^1	0.129×10^1	0.151×10^1	0.059×10^1	0.160×10^1	0.158×10^1
4	0.060×10^1	0.102×10^1	0.012×10^1	0.077×10^1	0.088×10^1	0.042×10^1	0.117×10^1	0.047×10^1	0.086×10^1
5	0.136×10^1	0.073×10^1	0.177×10^1	0.139×10^1	0.091×10^1	0.146×10^1	0.111×10^1	0.134×10^1	0.102×10^1
Function 4									
Mesh	$\mathcal{O}_{L^1}$			$\mathcal{O}_{L^2}$			$\mathcal{O}_{L^{inf}}$		
	x	y	z	x	y	z	x	y	z
1	—	—	—	—	—	—	—	—	—
2	0.045×10^1	-0.001×10^1	-0.017×10^1	0.049×10^1	-0.002×10^1	-0.032×10^1	0.063×10^1	0.001×10^1	-0.037×10^1
3	0.121×10^1	0.142×10^1	0.138×10^1	0.101×10^1	0.140×10^1	0.151×10^1	0.062×10^1	0.139×10^1	0.160×10^1
4	0.059×10^1	0.099×10^1	0.024×10^1	0.075×10^1	0.097×10^1	0.046×10^1	0.110×10^1	0.086×10^1	0.091×10^1
5	0.136×10^1	0.086×10^1	0.168×10^1	0.141×10^1	0.095×10^1	0.146×10^1	0.118×10^1	0.106×10^1	0.105×10^1

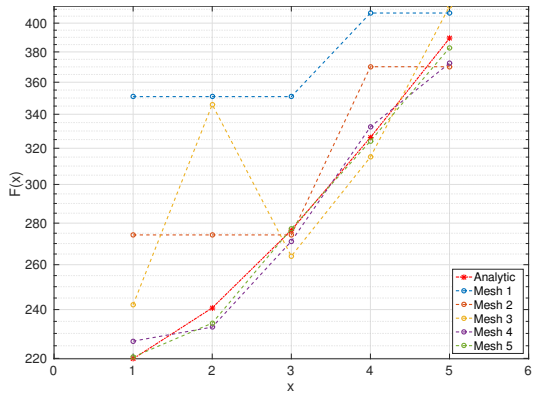
Table C.3 Mass matrix: convergence orders per field component

Function 1										
Mesh	L^1			L^2			L^{inf}			
	x	y	z	x	y	z	x	y	z	
1	152.5×10^1	135.3×10^1	123.5×10^1	72.31×10^1	64.82×10^1	56.22×10^1	47.05×10^1	44.38×10^1	30.25×10^1	
2	81.2×10^1	78.26×10^1	83.71×10^1	38.71×10^1	37.49×10^1	31.46×10^1	27.81×10^1	28.43×10^1	20.22×10^1	
3	50.43×10^1	32.75×10^1	57.19×10^1	19.76×10^1	16.28×10^1	18.75×10^1	18.06×10^1	16.55×10^1	12.11×10^1	
4	31.34	14.73	28.46×10^1	10.17×10^1	9.43×10^1	10.29×10^1	9.85×10^1	9.24×10^1	7.72×10^1	
5	15.47×10^1	6.47×10^1	18.42×10^1	5.90×10^1	5.83×10^1	6.40×10^1	5.78×10^1	5.51×10^1	4.83×10^1	

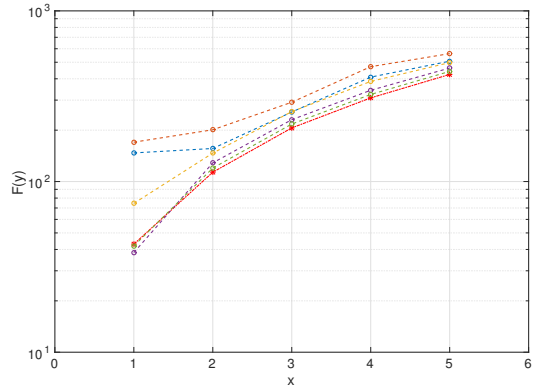
Table C.4 Stiffness matrix: errors per field analysis

Function 1										
Mesh	$\mathcal{O}_{L^1}$			$\mathcal{O}_{L^2}$			$\mathcal{O}_{L^{inf}}$			
	x	y	z	x	y	z	x	y	z	
1	—	—	—	—	—	—	—	—	—	
2	0.045×10^1	0.039×10^1	0.028×10^1	0.045×10^1	0.039×10^1	0.041×10^1	0.037×10^1	0.032×10^1	0.029×10^1	
3	0.068×10^1	0.125×10^1	0.054×10^1	0.096×10^1	0.120×10^1	0.074×10^1	0.062×10^1	0.078×10^1	0.074×10^1	
4	0.068×10^1	0.115×10^1	0.100×10^1	0.095×10^1	0.078×10^1	0.087×10^1	0.088×10^1	0.084×10^1	0.064×10^1	
5	0.101×10^1	0.118×10^1	0.062×10^1	0.076×10^1	0.069×10^1	0.068×10^1	0.076×10^1	0.074×10^1	0.067×10^1	

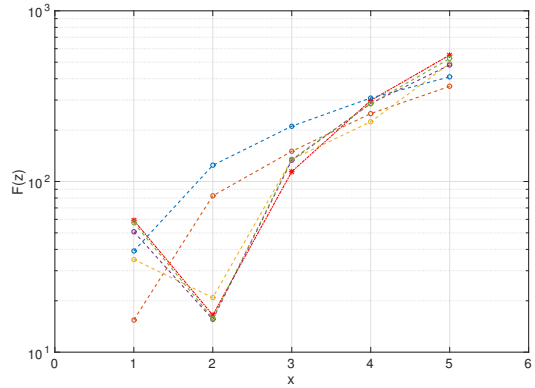
Table C.5 Stiffness matrix: convergence orders per field component



(a) x-component

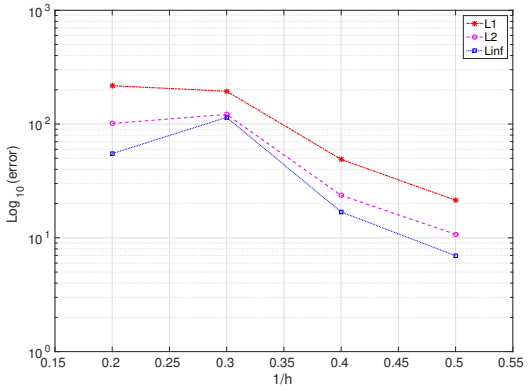


(b) y-component

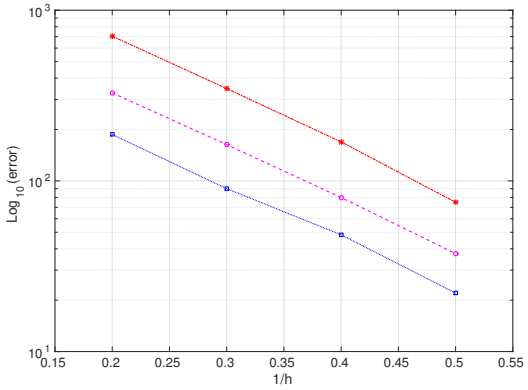


(c) z-component

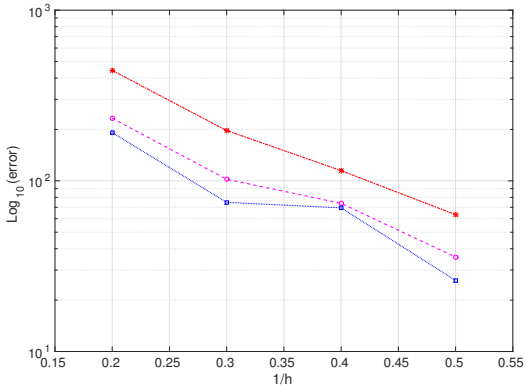
Fig. C.4 Stiffness matrix: trace fields for test equation C.7.



(a) x-component



(b) y-component



(c) z-component

Fig. C.5 Stiffness matrix: convergence order for test equation C.7.

Appendix D

PETGEM documentation

PETGEM Documentation

Release 1.0

Octavio Castillo Reyes

September 14, 2017

CHAPTER

ONE

WHAT IS PETGEM?

Electromagnetic methods (EM) are an established tool in geophysics, with application in many areas such as hydrocarbon and mineral exploration, reservoir monitoring, CO_2 storage characterization, geothermal reservoir imaging and many others. In particular, the marine Controlled-Source ElectroMagnetic method (CSEM) has become an important technique for reducing ambiguities in data interpretation for hydrocarbon exploration. In order to be able to predict the EM signature of a given geological structure, modelling tools provide us with synthetic results which we can then compare to real data. In particular, if the geology is structurally complex, one might need to use methods able to cope with such complexity in a natural way by means of, e.g., an unstructured mesh representing its geometry. Among the modelling methods for EM based upon 3D unstructured meshes, the Nédélec Finite Elements (FE), a type of Edge Elements, offer a good trade-off between accuracy and number of degrees of freedom, i.e. size of the problem.

In the multi-core and many-core era, parallelization is a crucial issue. Nédélec FE offer good scalability potential. Its low DOF number after primary/secondary field decomposition make them potentially fast, which is crucial in the future goal of solving inverse problems which might involve over 100,000 realizations. However, the state of the art shows a relative scarcity of robust edge-based codes to simulate these problems.

On top of that, **Parallel Edge-based Tool for Geophysical Electromagnetic Modelling** (PETGEM) is a Python tool for the scalable solution of EM on tetrahedral meshes, as these are the easiest to scale-up to very large domains or arbitrary shape. It supports distributed-memory paralelism through mpi4py and petsc4py packages.

As a result, PETGEM tool allow users the simulation of electromagnetic fields in real CSEM surveys with high accuracy, reliability and efficiency.

PETGEM code is developed as `open-source GPLv3` at Computer Applications in Science & Engineering (CASE) of the Barcelona Supercomputing Center (BSC). Requests and contributions are welcome.

CHAPTER

TWO

FEATURES

PETGEM use a code structure for the Nédélec FE algorithm that emphasizes good parallel scalability, which is crucial in the multi-core era. Furthermore, it's modularity should simplify the process of reaching the best possible performance in terms of percentage of the peak amount of floating point operations provided by the architecture.

Software stack

An outline of the primary groups of modules in PETGEM design is given in *Figure 3.1*.

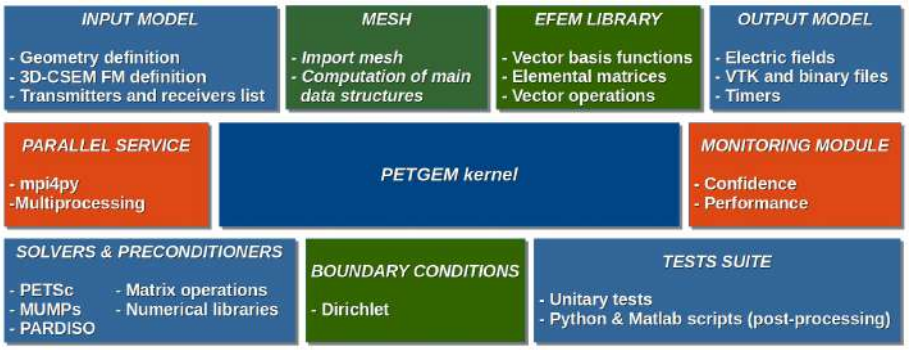


Fig. 2.1: Figure 3.1. Upper view of PETGEM software stack.

A more detailed explanation is the following:

- Modular and extensible EFEM kernel – The kernel is extensible in any direction. Therefore, the possibility of adding new features such as new boundary conditions, numerical algorithms, analysis modules, among others.
- Independent of problem formulation, numerical solution, and data storage – The kernel provides the independent abstractions for modeling, numerical methods, data storage and analysis.
- Parallel processing support – Based on an shared-memory parallelism (Multiprocessing package), distributed-memory parallelism (mpi4py and petsc4py) and static load balancing.
- Confidence and performance monitoring – Based on an intensive error checking module and an automatic profiling module.
- Efficient solvers & preconditioners – Direct as well as iterative solvers and preconditioners are supported through petsc4py package.

Direct as well as iterative solvers and preconditioners are supported through an interface to third party libraries (PETSc, MUMPs, PARDISO).

- Interface to mesh generators – Not dependent on a specific mesh generator. Because most of the FEM codes were developed for node-based formulations, it is necessary to develop a module to compute edge-based data structures. As a result, different mesh formats are supported (Gambit, Netgen, Gmsh).
- Edge FEM library – Edge-based discretisations, vector basis functions, their geometry description, and generalized integration rules provides a generic support for implementing edge-based solution algorithms.
- Linear systems library – Support to Compressed Row Storage (CSR) format for sparse matrices and their easy and efficient parallel assembly on shared/distributed-memory platforms.
- CSEM module – Ad-hoc design to meet specific requirements to simulate 3D CSEM surveys, namely, conductivity model, physical parameters, transmitter and receiver lists.
- Tests suite – Sample output for many examples are included. Furthermore, a set of matlab functions to data analysis are included.
- Post-processing – Export to binary files, HDF5 and VTK format are supported, allowing the analysis not just in different visualization tools but also on different platforms. It also gives timing values in order to evaluate the performance.

Programming language

PETGEM is based on Python language programming because:

- It is open source, cross-platform and functional on a wide number of platforms, including HPC environments.
- It uses a high level and very expressive language.
- It is based on a sophisticated array manipulation in a Fortran-like manner.
- It uses a good body of bindings to common tools needed in scientific computing: plotting, numerical libraries, debugging and testing.

The code structure is modular, simple and flexible which allows exploiting not just our own modules but also third party libraries. Therefore, the software stack includes interfaces to external suites of data structures and libraries that contain most of the necessary building blocks needed for programming large scale numerical applications, i.e. sparse matrices, vectors, iterative and direct solvers or domain decomposition. As a result, the code is compact and eliminates the need to write such libraries and thus speeds up development time by orders of magnitude [1], [2], [3] and [5].

In order to meet the high computational cost of the modeling, two parallel approaches are supported: shared-memory parallelism (Python Multiprocessing package) and distributed-memory parallelism (Petsc4py and mpi4py packages) [1], [2], [3] and [4].

Target architecture

The HPC goal of the PETGEM involves using cutting-edge architectures. To that goal, the code is implemented in current state-of-the-art platforms such as Intel Haswell and Intel Xeon Phi processors, which offer high performance,

[1] Bangerth, W., Burstedde, C., Heister, T., and Kronbichler, M. (2011). Algorithms and data structures for massively parallel generic adaptive finite element codes. ACM Transactions on Mathematical Software (TOMS), 38(2):14.

[2] Heister, T., Kronbichler, M., and Bangerth, W. (2010). Massively parallel finite element programming. Recent Advances in the Message Passing Interface, pages 122–131.

[3] Key, K. and Ovall, J. (2011). A parallel goal-oriented adaptive finite element method for 2.5-d electromagnetic modelling. Geophysical Journal International, 186(1):137– 154.

[5] Osseyran, A. and Giles, M. (2015). Industrial Applications of High-Performance Computing: Best Global Practices. Chapman & Hall/CRC Computational Science. CRC Press, first edition.

[4] Logg, A. (2007). Automating the finite element method. Archives of Computational Methods in Engineering, 14(2):93–138.

flexibility and power efficiency. Nevertheless, PETGEM support older architectures such as SandyBridge, for the sake of usability and to be able to compare performance.

CHAPTER

THREE

CSEM FORWARD MODELLING & EDGE ELEMENTS FORMULATION

The last decade has been a period of rapid growth for electromagnetic methods (EM) in geophysics, mostly because of their industrial adoption. In particular, the marine controlled-source electromagnetic (CSEM) method has become an important technique for reducing ambiguities in data interpretation in hydrocarbon exploration. In order to be able to predict the EM signature of a given geological structure, modelling tools provide us with synthetic results which we can then compare to real data. In particular, if the geology is structurally complex, one might need to use methods able to cope with such complexity in a natural way by means of, e.g., an unstructured mesh representing its geometry. Among the modelling methods for EM based upon 3D unstructured meshes, the Nédélec Edge Finite Element Method (EFEM) offers a good trade-off between accuracy and number of degrees of freedom, i.e. size of the problem. Furthermore, its divergence-free basis is very well suited for solving Maxwell's equation. On top of that, we choose to support tetrahedral meshes, as these are the easiest to use for very large domains or complex geometries. We present the numerical formulation and results of 3D CSEM forward modelling (FM) using tetrahedral EFEM on unstructured meshes.

CSEM problem

3D CSEM FM is typically solved in frequency domain, which involves the numerical solution of Maxwell's equations in stationary regimes for heterogeneous anisotropic electrically conductive domains. Furthermore, CSEM surveys generally work with low frequency electromagnetic fields ($\sim$ 1 Hz) because the electric conductivity of the geological structures is much larger than their dielectric permittivity. As a consequence, in an unbound domain Γ, the electric field can be obtained by solving Maxwell's equations in their diffusive form:

$$\begin{aligned} \nabla \times \mathbf{E} &= i\omega\mu_0 \mathbf{H} \\ \nabla \times \mathbf{H} &= \mathbf{J}_s + \tilde{\sigma}\mathbf{E} \end{aligned} \tag{3.1}$$

where the harmonic time dependence $e^{-i\omega t}$ is omitted, with ω is the angular frequency, μ_0 the free space magnetic permeability, $\mathbf{J}_s$ the distribution of source current, $\tilde{\sigma}\mathbf{E}$ the induced current in the conductive Earth and $\tilde{\sigma}$ the electrical conductivity which is assumed isotropic for simplicity.

In numerical approximations of EM fields there are two main drawbacks. The first one is the inevitable spatial singularity at the source. The second is the grid refinement requirements in order to capture the rapid change of the primary field [1]. In order to mitigate these issues, PETGEM used a secondary field approach where the total electric field $\mathbf{E}$ is obtained as:

$$\begin{aligned} \mathbf{E} &= \mathbf{E}_p + \mathbf{E}_s \\ \tilde{\sigma} &= \tilde{\sigma_s} + \Delta\tilde{\sigma} \end{aligned} \tag{3.2}$$

where subscripts p and s represent a primary field and secondary field respectively. For a general layered Earth model, $\mathbf{E}_p$ can be computed semi-analytically by using Hankel transform filters. Based on this decomposition and following

[1] Cai, H., Xiong, B., Han, M. and Zhdanov, M. (2014). 3D controlled-source electromagnetic modeling in anisotropic medium using edge-based finite element method. Computers & Geosciences, 73, 164–176.

the work by [6] the equation system to solve $\mathbf{E}_s$ is:

$$\nabla \times \nabla \times \mathbf{E}_s + i\omega\mu\tilde{\sigma}\mathbf{E}_s = -i\omega\mu\Delta\sigma\mathbf{E}_p \tag{3.3}$$

where the electrical conductivity σ is a function of position that is allowed to vary in 3D, whereas the vacuum permeability μ is set to the free space value μ_0. Homogeneous Dirichlet boundary conditions, $\mathbf{E}_s = 0$ on $\partial\Gamma$, are applied. The range of applicability of this conditions can be determined based on the skin depth of the electric field [7].

Edge finite element formulation

For the computation of $\mathbf{E}_s$, PETGEM implemented the Nédélec EFEM which uses vector basis functions defined on the edges of the corresponding elements. Its basis functions are divergence-free but not curl-free [4]. Thus, EFEM naturally ensures tangential continuity and allows normal discontinuity of $\mathbf{E}_s$ at material interfaces. PETGEM used unstructured tetrahedral meshes because of their ability to represent complex geological structures such as bathymetry or reservoirs as well as the local refinement capability in order to improve the solution accuracy. *Figure 4.1* shows the tetrahedral Nédélec elements (lowest order) together with their node and edge indexing.

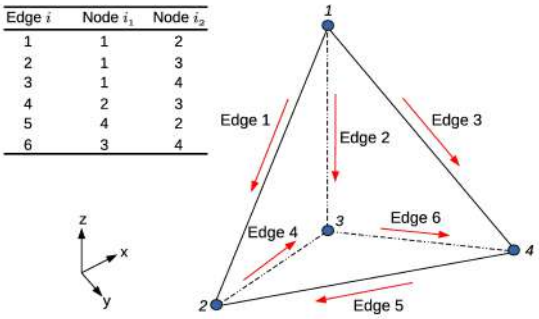


Fig. 3.1: Figure 4.1. Tetrahedral Nédélec edge element with node/edge indexing.

In PETGEM workflow, the tangential component of the secondary electric field is assigned to the edges in the mesh. Therefore, all components of the electric field at a point $\mathbf{x}$ located inside a tetrahedral element e can be obtained as follows:

$$\mathbf{E}^e(\mathbf{x}) = \sum_{i=1}^{6} \mathbf{N}_i^e(\mathbf{x}) E_i^e \tag{3.4}$$

where $\mathbf{N}_i^e$ are the vector basis functions associated to each edge i and E_i^e their degrees of freedom. Considering the node and edge indexing in *Figure 4.1*, the vector basis functions can be expressed as follows:

$$\mathbf{N}_i^e = (\lambda_{i1}^e \nabla\lambda_{i2}^e - \lambda_{i2}^e \nabla\lambda_{i1}^e)\ell_i^e \tag{3.5}$$

where subscripts $i1$ and $i2$ are the first and second nodes linked to the i-th edge, λ_i^e are the linear nodal basis functions, and ℓ_i^e is the length of the i-th edge of the element e.

By substituting equation (3.4) into (3.3), and using Galerkin's approach, the weak form of the original differential equation becomes:

$$Q_i = \int_\Omega \mathbf{N}_i \cdot [\nabla \times \nabla \times \mathbf{E}_s - i\omega\mu\tilde{\sigma}\mathbf{E}_s + i\omega\mu\Delta\tilde{\sigma}\mathbf{E}_p] dV \tag{3.6}$$

[6] Newman, G.A. and Alumbaugh, D.L. (2002). Three-dimensional induction logging problems, Part 2: A finite difference solution. Geophysics, 67(2), 484–491.

[7] Puzyrev, V., Koldan, J., de la Puente, J., Houzeaux, G., Vázquez, M. and Cela, J.M. (2013). A parallel finite-element method for three-dimensional controlled-source electromagnetic forward modelling. Geophysical Journal International, ggt027.

[4] Jin, J. (2002). The Finite Element Method in Electromagnetics. Wiley, New York, second edn.

The compact discretized form of (3.6) is obtained after applying the Green's theorem:

$$[K_{jk}^{e} + i\omega\tilde{\sigma}_{e}M_{jk}^{e}] \cdot \{E_{sk}\} = -i\omega\mu\Delta\tilde{\sigma}_{e}R_{k}^{e} \tag{3.7}$$

where K^e and M^e are the elemental stiffness and mass matrices which can be calculated analytically or numerically [4], and R_k^e is the right hand side which requires numerical integration.

CHAPTER

FOUR

INSTALLATION

This section describe the platforms supported by PETGEM and the requirements to install it.

Platforms

PETGEM is known to run on various flavors of Linux clusters.

Requirements

Requirements packages for using PETGEM:

- Python3.
- Scipy for numerical operations.
- Numpy for arrays manipulation.
- H5py for HDF5 files manipulation.
- Python Multiprocessing package for parallel computations on shared-memory platforms.
- Sharedmem Python package for arrays manipulation on shared memory platforms.
- Petsc4py and mpi4py for parallel computations on distributed-memory platforms. It allows the use of parallel direct/iterative solvers from PETSc.
- Paraview for visualization of PETGEM output files.
- Sphinx and LaTeX to (re)generate documentation.

On Linux, consult the package manager of your preference. PETGEM can be used without any installation by running the kernel from the top-level directory of the distribution.

Install PETGEM

Please, look at `config_site_template.py` and follow the instructions therein.

There are 3 ways to install PETGEM:

- In-place use:

```
$ python3 setup.py build_ext --inplace
```

- For installation (system-wide):

```
$ python3 setup.py build
```

or

```
$ python3 setup.py install
```

This option may require root privileges.

- Local installation:

```
$ python3 setup.py install --root=<install_prefix>
```

This option requires write privileges to `--root=<install_prefix>`. Finally, the command to clean a PETGEM installation is:

```
$ python3 setup.py clean
```

Install documentation

PETGEM is documented using Sphinx and LaTeX. The documentation source are in the doc directory.

The following steps summarize how to generate PETGEM documentation.

1. Install Sphinx and LaTeX.
2. If is necessary, edit the rst files in doc/ directory using your favorite text editor. Nothing fancy is needed since ReST format is really simple.
3. (Re) generate the PETGEM documentation as follows:

- By using setup.py script:

```
$ python3 setup.py output_format
```

since PETGEM actually support html and pdf formats, valid options for `output_format = [pdfdocs, htmldocs]`.

- By using Sphinx commands directly:

```
$ cd doc
$ make output_format
```

where `output_format = [html, latexpdf]`.

CHAPTER

FIVE

TUTORIAL

The PETGEM tutorial contains a collection of programs which demonstrate various aspects of the PETGEM work-flow. Each such example has the following structure:

1. Introduction. What the program does, including the mathematical model.
2. The commented program. An extensively documented of the source code.
3. Results. The output of the program, with comments and interpretation.

The programs are in the examples/ directory of local PETGEM folder. This tutorial focuses on the distributed-memory PETGEM version. Please refer to the *Download* section for details about the shared-memory version.

Basic notions

The start point of using PETGEM to solve a CSEM forward modelling is through its definition in a parameters file description, also referred to as **input file**. In such file, the modelling is described using several keywords that allow one to define physical parameters, mesh format, solvers, parallel specifications, and so on. See Parameters file description in *Manual* section for a full explanation of those keywords.

The syntax of the **input file** is very simple yet powerful. For a general CSEM forward modelling, the PETGEM work-flow can be summarize as follows:

1. The kernel (kernel.py) reads an **input file**.
2. Following the contents of the **input file**, a problem instance is created.
3. The problem sets up its domain, sub-domains, source, solver. This stage include the data structures computation related with the edges in the mesh.
4. Parallel assemblig of $Ax = b$. See *CSEM problem* and *Edge finite element formulation* sections for a detail mathematical background of this equation system.
5. The solution is obtained in parallel by calling `ksp()` PETSc object.
6. Interpolation & post-processing parallel stage.
7. Finally the solution can be stored by calling `save_solution()` function. Current version support binary files, HDF5 and VTK format.

Pre-processing

The `petgem_preprocessing.m` Matlab script provides functions to change input file formats into a representation that is more suitable for PETGEM. It transforms the mesh files into a PETSc binary format, which allow

parallel computations in a simple way. This step is hence mandatory for any modelling. This script is included in `utils/Preprocessing/petgem_preprocessing.m` of the PETGEM source tree.

From a Matlab terminal, `petgem_preprocessing.m` is invoked has follows:

```
$ petgem_preprocessing('mesh_file', 'mesh_format', sigma_domain_per_subdomain, 'receivers_file')
```

See *Utilities* section for more details about `petgem_preprocessing.m`.

Running a simulation

This section introduces the basics of running PETGEM on the command line. The $ represents the command prompt of the terminal. The following commands should be run in the top-level directory of the PETGEM source tree.

PETGEM kernel is invoked as follows:

```
$ mpirun -n <# of MPI processes> python3 kernel.py -options_file path/petsc.opts path/input_file.py
```

where *petsc.opts* is the PETSc options file and *input_file.py* is the PETGEM parameters file, which describes the modelling to be solved in terms PETGEM can understand.

Parameters file template

By definition, any model should include: physical parameters, a mesh file, source and receivers files, computational issues (solver type, domain decomposition) and an output file format.

In sake of simplicity and in order to avoid a specific parser, the PETGEM parameters file is defined as a Pyhton dictionary. As consequence, the dictionary name and his key names MUST NOT BE changed.

A template of this file is included in `examples/params_file_template.py` of the PETGEM source tree. A glance of this file is the following:

```
modelling = {
  # ------------------------------
  # ---------- General ----------
  # ------------------------------

  # ------------------------------
  # ----- Pyshical parameters -----
  # ------------------------------
  # Source
  # Source frequency. Type: float
  # Optional: NO
  'freq': 1.0,

  # Source position(x, y, z). Type: float
  # Optional: NO
  'src_pos': [1750.0, 1750.0, -975.0],

  # Source orientation. Type: int
  # 1 = X-directed source
  # 2 = Y-directed source
  # 3 = Z-directed source
  # Optional: NO
  'src_direc': 1,
```

```
  # Source current. Type: float
  # Optional: NO
  'src_current': 1.0,

  # Source length. Type: float
  # Optional: NO
  'src_length': 1.0,

  # Conductivity model. Type: str
  # Optional: NO
  'sigma_file': 'examples/WHAM/elemsSigma.dat',

  # Background conductivity. Type: float
  # Optional: NO
  'sigma_background': 3.33,

  # --------------------------------
  # ------- Mesh information ------
  # --------------------------------
  # Mesh files

  # Nodes spatial coordinates. Type: str
  # Optional: NO
  'nodes_file': 'examples/WHAM/nodes.dat',

  # Elements-nodes connectivity. Type: str
  # Optional: NO
  'elemsN_file': 'examples/WHAM/elemsN.dat',

  # Elements-edges connectivity. Type: str
  # Optional: NO
  'elemsE_file': 'examples/WHAM/elemsE.dat',

  # Edges-nodes connectivity. Type: str
  # Optional: NO
  'edgesN_file': 'examples/WHAM/edgesN.dat',

  # nnz for matrix allocation (PETSc)
  'nnz_file': 'examples/WHAM/nnz.dat',

  # Boundary-edges. Type: str
  # Optional: NO
  'bEdges_file': 'examples/WHAM/bEdges.dat',

  # --------------------------------
  # ------------ Solver ------------
  # --------------------------------
  # Solver options must be set in
  # petsc_solver.opts file

  # --------------------------------
  # ------------ Output ------------
  # --------------------------------
  # Name of the file that contains the receivers position. Type: str
  # Optional: NO
  'receivers_file': 'examples/WHAM/receivers.dat',
}
```

In section *Parameters file description* is included a deep explanation about this file is included.

Visualization of results

Once a solution of CSEM forward modelling has been obtained, it should be post-processed by using a visualization program. PETGEM does not do the visualization by itself, but it generates output files (binary format) with the final results which can be exported to other format by using the Post-processing PETGEM scripts. It also gives timing values in order to evaluate the performance. It requires task Paraview.

CHAPTER

SIX

MANUAL

Table of contents

This manual provides reference documentation to PETGEM from a user's and developer's perspective.

How generate documentation

PETGEM is documented using Sphinx and LaTeX. The documentation source are in the doc directory.

The following steps summarize how generate PETGEM documentation.

1. Install Sphinx and LaTeX.
2. If is necessary, edit the rst files in doc/ directory using your favorite text editor. Nothing fancy is needed since ReST format is really simple.
3. (Re) generate the PETGEM documentation as follows:

- By using setup.py script:

```
$ python3 setup.py output_format
```

since PETGEM actually support html and pdf formats, valid options for `output_format = [pdfdocs, htmldocs]`.

- By using Sphinx commands directly:

```
$ cd doc
$ make output_format
```

where `output_format = [html, latexpdf]`.

Coding style

PETGEM's coding style is based on PEP-0008 guidelines. Main guidelines are the following:

- 79 characteres per line.
- 4 spaces per indentation level.
- Variables are lower case meanwhile constants are upper case.
- Comments convention for functions is as follows:

```
def function(arg1, arg2):
    ''' This is a function.

        :param int arg1: array of dimensions ...
        :param str arg2: string that ...
    '''
```

- The use of inline comments is sparingly.
- Use of lowercase to name functions. Furthermore, functions names have following form: `<action>_<subject>()`, e.g. `compute_matrix()`.
- Use of whole words instead abbreviations, examples:
  - Yes: `solve_system()`, `compute_edges()`, `compute_matrix()`.
  - No: `solve()`, `compedges()`, `compmatrix()`.

PETGEM design

PETGEM use a code structure for the Nédelec FE algorithm that emphasizes good parallel scalability, which is crucial in the multi-core era. Furthermore, it's modularity should simplify the process of reaching the best possible performance in terms of percentage of the peak amount of floating point operations provided by the architecture. The code structure is modular, simple and flexible which allows exploiting not just our own modules but also third party libraries. Therefore, the software stack includes interfaces to external suites of data structures and libraries that contain most of the necessary building blocks needed for programming large scale numerical applications, i.e. sparse matrices, vectors, iterative and direct solvers or domain decomposition. As a result, the code is compact and flexible whose main UML diagrams are described as follows.

Component diagram

Main components of PETGEM are described in *Figure 7.1*. Pre-processing, processing and post-processing phases are included in *Figure 7.1*.

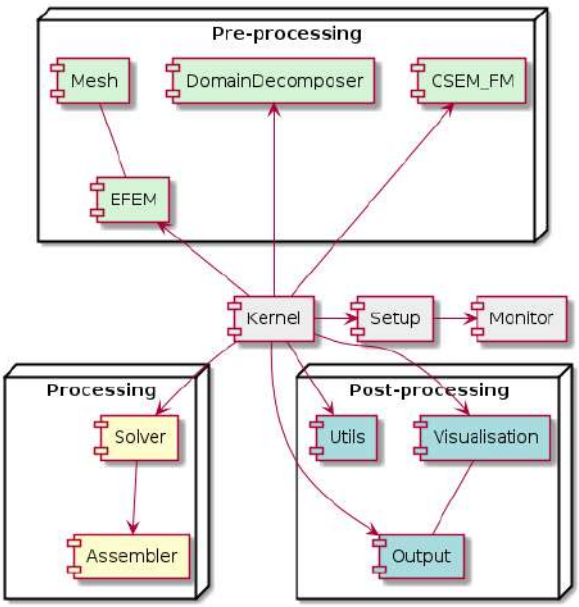


Fig. 6.1: Figure 7.1. PETGEM: class diagram.

Class diagram

Main classes for PETGEM are described in *Figure 7.2*. Among all, the `kernel` class is the most important since it manage and control the others classes, as consequence, `kernel` class is the start point for any modelling.

Sequence diagram

Figure 7.3 describe the sequence for a standard CSEM forward modelling.

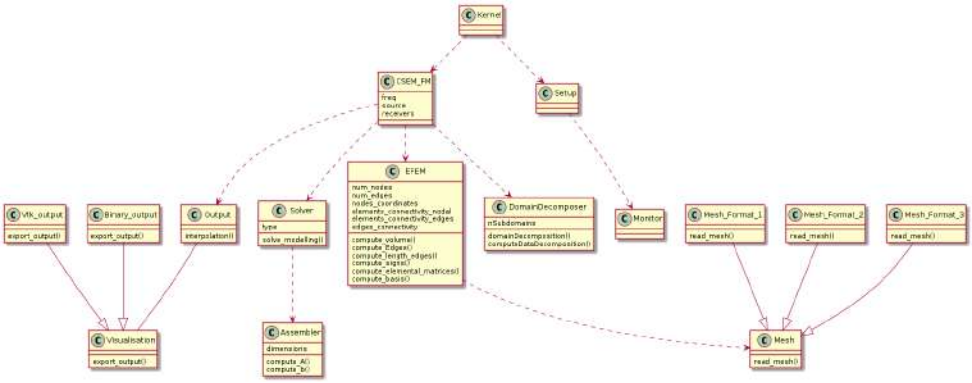


Fig. 6.2: PETGEM: class diagram.

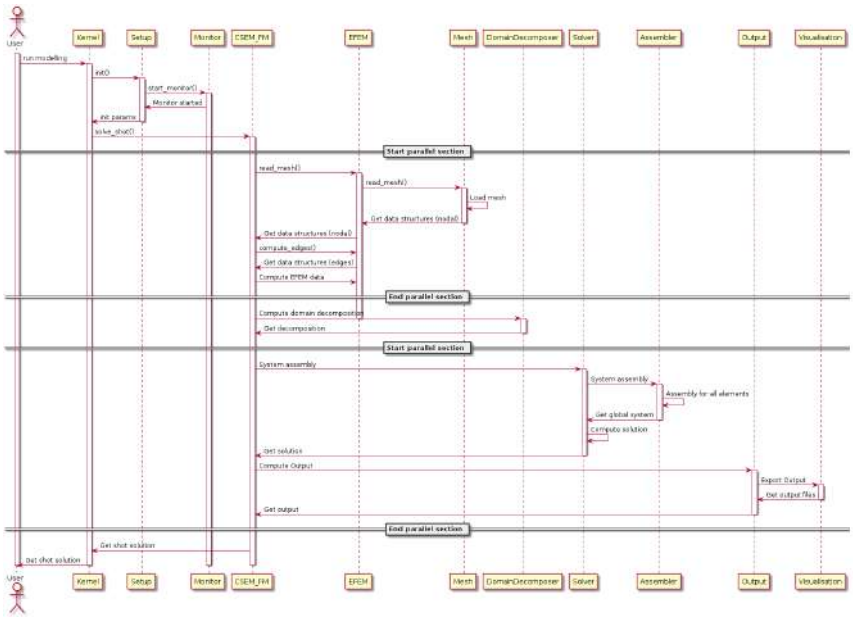


Fig. 6.3: PETGEM: sequence diagram.

PETGEM directory structure

This subsection is dedicated to list and describe the PETGEM directory structure.

Table 6.1: Top directory structure

Name	Description
doc/	Source files for PETGEM documentation
examples/	CSEM FM examples directory
petgem/	Source code
tests/	PETGEM tests directory
utils/	Set of Python/Matlab scripts for pre-processing and post-processing
AUTHORS	Authors file
builder.py	Build helper for PETGEM setup script. The original version of this script was adapted from NiPy project
config_site.py	PETGEM configuration file
INSTALL	Installation and configuration instructions
kernel.py	Heart of the code. This file contains the entire work-flow for a CSEM FM
LICENSE	License file
Makefile	Makefile with main PETGEM setup commands
MANIFEST.in	File with exact list of files to include in PETGEM distribution
README	Readme file
setup.py	Main PETGEM setup script, it is based on Python setup-module
VERSION	File with PETGEM version

The PETGEM source code is *petgem/*, which has the following contents:

Table 6.2: *petgem/* directory structure

Name	Description
base/	Common classes and functions for init process.
decomposition/	Modules for domain decomposition (required for parallel execution of PETGEM)
efem/	General modules and classes for describing a CSEM FM by using EFEM of lowest order in tetrahedral meshes
mesh/	Interface to import mesh files
monitoring/	Modules for PETGEM performance monitoring
parallel/	Modules for supporting parallel assembling and solution of CSEM FM
postprocessing/	Modules for postprocessing stage
solver/	Interface to various internal/external solvers

Running a simulation

The following command should be run in the top-level directory of the PETGEM source tree. PETGEM kernel is invoked as follows:

```
$ mpirun -n <# of MPI processes> python3 kernel.py -options_file path/petsc.opts path/input_file.py
```

where *petsc.opts* is the PETSc options file and *input_file.py* is the PETGEM parameters file, whose main structure is explained in the following section.

Parameters file description

By definition, any model should include: physical parameters, a mesh file, source and receivers files, computational issues (solver type, solver tolerance, preconditioner) and an output file format.

In sake of simplicity and in order to avoid a specific parser, the PETGEM parameters file is defined as a Pyhton dictionary. As consequence, the dictionary name and his key names MUST NOT BE changed. The content of this file is the following:

```
modelling = {
 # --------------------------------
 # ----------- General ------------
 # --------------------------------

 # --------------------------------
 # ----- Pyshical parameters -----
 # --------------------------------
 # Source
 # Source frequency. Type: float
 # Optional: NO
 'freq': 1.0,

 # Source position(x, y, z). Type: float
 # Optional: NO
 'src_pos': [1750.0, 1750.0, -975.0],

 # Source orientation. Type: int
 # 1 = X-directed source
 # 2 = Y-directed source
 # 3 = Z-directed source
 # Optional: NO
 'src_direc': 1,

 # Source current. Type: float
 # Optional: NO
 'src_current': 1.0,

 # Source length. Type: float
 # Optional: NO
 'src_length': 1.0,

 # Conductivity model. Type: str
 # Optional: NO
 'sigma_file': 'examples/WHAM/elemsSigma.dat',

 # Background conductivity. Type: float
 # Optional: NO
 'sigma_background': 3.33,

 # --------------------------------
 # ------- Mesh information ------
 # --------------------------------
 # Mesh files

 # Nodes spatial coordinates. Type: str
 # Optional: NO
 'nodes_file': 'examples/WHAM/nodes.dat',
```

```
  # Elements-nodes connectivity. Type: str
  # Optional: NO
  'elemsN_file': 'examples/WHAM/elemsN.dat',

  # Elements-edges connectivity. Type: str
  # Optional: NO
  'elemsE_file': 'examples/WHAM/elemsE.dat',

  # Edges-nodes connectivity. Type: str
  # Optional: NO
  'edgesN_file': 'examples/WHAM/edgesN.dat',

  # nnz for matrix allocation (PETSc)
  'nnz_file': 'examples/WHAM/nnz.dat',

  # Boundary-edges. Type: str
  # Optional: NO
  'bEdges_file': 'examples/WHAM/bEdges.dat',

  # --------------------------------
  # ------------ Solver ------------
  # --------------------------------
  # Solver options must be set in
  # examples/WHAM/petsc_solver.opts

  # --------------------------------
  # ------------ Output ------------
  # --------------------------------
  # Name of the file that contains the receivers position. Type: str
  # Optional: NO
  'receivers_file': 'examples/WHAM/receivers.dat',
}
```

The PETGEM parameters file is divided into 4 sections, namely, physical parameters (frequency, source position, source current, source length, conductivity model, background conductivity), mesh information (file path of nodal spatial coordinates, nodal element connectivity, edge element connectivity, edges nodes connectivity, and sparsity structure for PETSc matrix allocation), solver parameters (solver type, tolerance, maximum number of iterations), results information (receivers position file path).

Mesh formats

PETGEM support following mesh formats:

- STL format from NETGEN.
- STL format from Gambit.
- Msh format from Gmsh.
- Matlab format: matrix of nodal coordinates and element connectivity.
- Files in Alya format (dom.dat and geo.dat).

Aforementioned formats must be pre-processed using the `petgem_preprocessing.m` Matlab script. This script is included in `utils/Preprocessing/petgem_preprocessing.m` of the PETGEM source tree. From a Matlab terminal, `petgem_preprocessing.m` is invoked has follows:

```
$ petgem_preprocessing('mesh_file', 'mesh_format', sigma_domain_per_subdomain, 'receivers_file')
```

See *Utilities* section for more details about `petgem_preprocessing.m`.

Available solvers

This section describes solvers available in PETGEM from user's perspective. Direct as well as iterative solvers and preconditioners are supported through an interface to third party libraries (PETSc, MUMPs).

PETGEM uses petsc4py package in order to support the Krylov Subspace Package (KSP) from PETSc. The object KSP provides an easy-to-use interface to the combination of a parallel Krylov iterative method and a preconditioner (PC) or a sequential direct solver. As result, PETGEM users can set various solver options and preconditioner options at runtime via the PETSc options database.

Simulations in parallel

In FEM or EFEM simulations, the need for efficient algorithms for assembling the matrices may be crucial, especially when the DOF is considerably large. This is the case for real scenarios of 3D CSEM forward modelling because the required accuracy. In such situation, assembly process remains a critical portion of the code optimization since solution of linear systems which asymptotically dominates in large-scale computing, could be done with linear solvers such as PETSc, MUMPs, PARDISO). In fact, in PETGEM V1.0, the system assembly takes around $40%$ of the total time.

The classical assembly in FEM or EFEM programming is based on a loop over the elements. Different techniques and algorithms have been proposed and nowadays is possible performing these computations at the same time, i.e., to compute them in parallel. However, parallel programming is not a trivial task in most programming languages, and demands a strong theoretical knowledge about the hardware architecture. Fortunately, Python presents user friendly solutions for parallel computations, namely, mpi4py. The mpi4py package is an open source software that provides bindings of the MPI standard for the Python programming language, allowing any Python code to exploit multiple processors architectures.

On top of that, *Figure 7.4* depicts shows an upper view of the matrix assembly and solution using the mpi4py and petsc4py package in PETGEM. The first step is to partition the work-load into subdomains. This task can be done by Metis library, which makes load over processes balanced. After domain partition, subdomains are read and assigned to MPI tasks and the elemental matrices are calculated concurrently. These local contributions are then accumulated into the global matrix system. The process for global vector assembly is similar.

Subsequently, the system is ready to be solved. PETGEM uses the Krylov Subspace Package (KSP) from PETSc through the petsc4py package. The object KSP provides an easy-to-use interface to the combination of a parallel Krylov iterative method and a preconditioner (PC) or a sequential direct solver. As result, PETGEM users can set various solver options and preconditioner options at runtime via the PETSc options database. Since PETSc knows which portions of the matrix and vectors are locally owned by each processor, the post-processing task is also completed in parallel following the numerical scheme described in *CSEM problem* section.

All petsc4py classes and methods are called from the PETGEM kernel in a manner that allows a parallel matrix and parallel vectors to be created automatically when the code is run on many processors. Similarly, if only one processor is specified the code will run in a sequential mode. Although petsc4py allows control the way in which the matrices and vectors to be split across the processors on the architecture, PETGEM simply let petsc4py decide the local sizes in sake of computational flexibility. However, this can be modified in an easy way without any extra coding required.

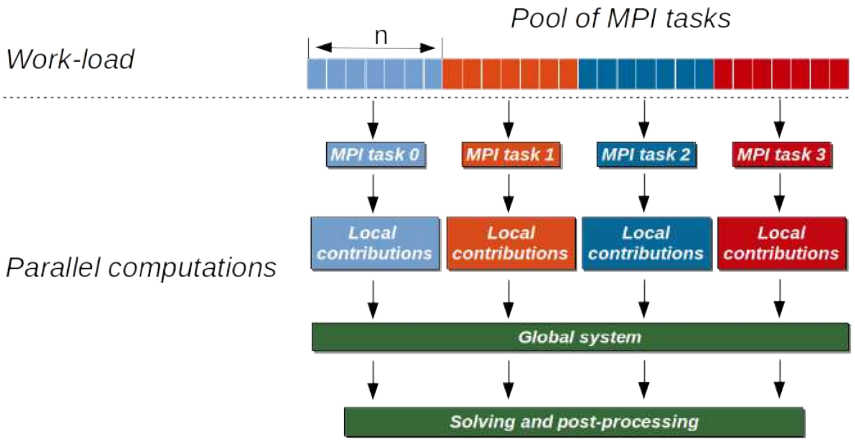


Fig. 6.4: Parallel scheme for assembly and solution in PETGEM using 4 MPI tasks. Here the elemental matrices computation is done in parallel. After calculations the global system is built and solved in parallel using the petsc4py and mpi4py packages.

Visualization of results

Once a solution of CSEM forward modelling has been obtained, it should be post-processed by using a visualization program. PETGEM does not do the visualization by itself, but it generates output files (vtk format) with the final results. *Figure 7.5* shows an example of PETGEM output for the modelling described in *Examples* section. *Figure 7.5* was obtained using Paraview.

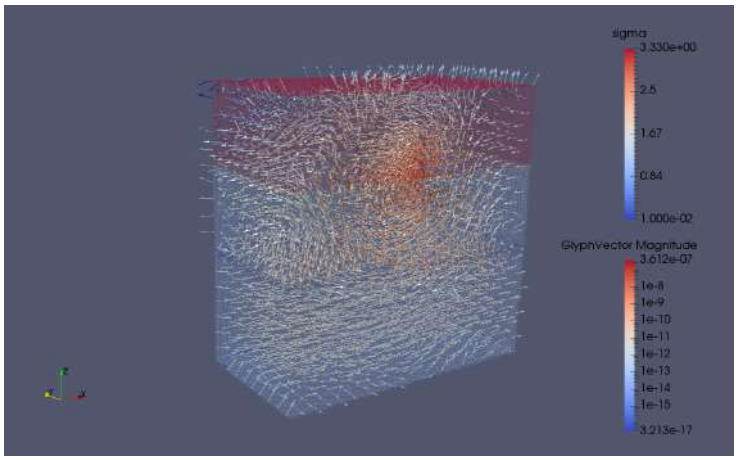


Fig. 6.5: PETGEM vtk output.

Utilities

A set of Matlab functions to preprocessing and postprocessing included. These scripts are located in `utils/Preprocessing/` and `utils/Postprocessing/` respectively. A more detailed explanation for Preprocessing is the following:

Table 6.3: Preprocessing

Name	Description	Parameters
export_matrix_hdf5.m	Export a matrix to a hdf5 file	Matrix to be exported and out file name
petgem_preprocessing.m	Build input files for PETGEM (PETSc format)	File mesh to be loaded, mesh format, conductivity model (layer model), receivers file name

On the other hand, the Postprocessing utils are the following:

Table 6.4: Postprocessing

Name	Description	Parameters
petgem_import.m	Import PETGEM output to Matlab	PETGEM output file name. This file contains 3 arrays: Primary field (Ep), Secondary field (Es) and Total field (Et) for each receiver.

Examples

This section includes a step-by-step walk-through of the process to solve a simple CSEM forward modelling. The typical process to solve a problem using PETGEM is followed: a model is meshed, PETGEM input files are created, a parameters file is drafted, PETGEM is run to solve the modelling and finally the results of the analysis are visualised.

Problem statement: Isotropic model

In order to explain the CSEM forward modelling using PETGEM, here is considered the canonical model by Weiss2006 which consists in four-layers: 1000 m thick seawater (3.3 S/m), 1000 m thick sediments (1 S/m), 100 m thick oil (0.01 S/m) and 1400 m thick sediments (1 S/m). The computational domain is a $[0, 3500]^3$ m cube. For this model, a 1 Hz x-directed dipole source is located at $z = 975$, $x = 1750$, $y = 1750$. The receivers are placed in-line to the source position and along its orientation, directly above the seafloor ($z = 990$).

Meshing

PETGEM V1.0 is based on tetrahedral meshes of lowest order. Therefore, *Figure 7.6* shows a 3D view of the model with its unstructured tetrahedral mesh for the halfspace $y > 1750$, with the color scale representing the electrical conductivity σ for each layer. Mesh in *Figure 7.6* have been obtained using Gambit.

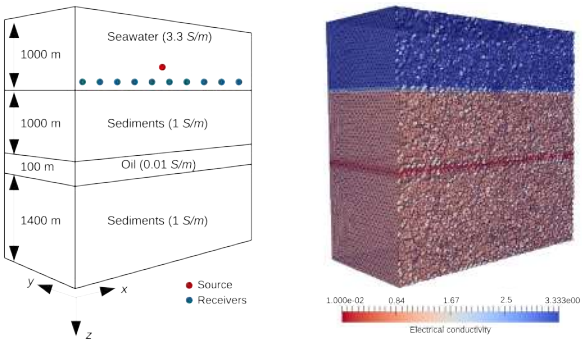


Fig. 6.6: In-line canonical off-shore hydrocarbon model with its unstructured tetrahedral mesh for $y > 1750$. The color scale represents the electrical conductivity σ for each layer.

PETGEM preprocessing

The next step in the process is the PETGEM input files construction. These files can be created using the `utils/Preprocessing/petgem_preprocessing.m` Matlab script. For this modelling, `petgem_preprocessing.m` should be invoked has follows:

```
$ petgem_preprocessing('examples/WHAM/Mesh_WHAM.neu', 'Gambit', [1/.3 1 1/100 1], 'Receivers_WHAM')
```

`petgem_preprocessing.m` will create 8 files in binary PETSc format: nodes.dat (nodal spatial coordinates), elemsN.dat (nodal elements connectivity), elemsE.dat (edges elements connectivity), edgesN.dat (edges nodes connectivity), bEdges.dat (edges boundary array), nnz.dat (sparsity pattern for matrix allocation), receivers.dat (receivers position).

Parameters file for PETGEM

The parameters file structure for PETGEM is well documented in *Parameters file description* section. The parameters file used for this example follows:

```
modelling = {
  # -------------------------------
  # ----------- General -----------
  # -------------------------------

  # -------------------------------
  # ----- Pyshical parameters -----
  # -------------------------------
  # Source
  # Source frequency. Type: float
  # Optional: NO
  'freq': 1.0,

  # Source position(x, y, z). Type: float
  # Optional: NO
  'src_pos': [1750.0, 1750.0, -975.0],

  # Source orientation. Type: int
  # 1 = X-directed source
  # 2 = Y-directed source
  # 3 = Z-directed source
  # Optional: NO
  'src_direc': 1,

  # Source current. Type: float
  # Optional: NO
  'src_current': 1.0,

  # Source length. Type: float
  # Optional: NO
  'src_length': 1.0,

  # Conductivity model. Type: str
  # Optional: NO
  'sigma_file': 'examples/WHAM/elemsSigma.dat',

  # Background conductivity. Type: float
  # Optional: NO
  'sigma_background': 3.33,
```

```
  # -------------------------------
  # ------- Mesh information ------
  # -------------------------------
  # Mesh files

  # Nodes spatial coordinates. Type: str
  # Optional: NO
  'nodes_file': 'examples/WHAM/nodes.dat',

  # Elements-nodes connectivity. Type: str
  # Optional: NO
  'elemsN_file': 'examples/WHAM/elemsN.dat',

  # Elements-edges connectivity. Type: str
  # Optional: NO
  'elemsE_file': 'examples/WHAM/elemsE.dat',

  # Edges-nodes connectivity. Type: str
  # Optional: NO
  'edgesN_file': 'examples/WHAM/edgesN.dat',

  # nnz for matrix allocation (PETSc)
  'nnz_file': 'examples/WHAM/nnz.dat',

  # Boundary-edges. Type: str
  # Optional: NO
  'bEdges_file': 'examples/WHAM/bEdges.dat',

  # -------------------------------
  # ------------ Solver -----------
  # -------------------------------
  # Solver options must be set in
  # examples/WHAM/petsc_solver.opts

  # -------------------------------
  # ------------ Output -----------
  # -------------------------------
  # Name of the file that contains the receivers position. Type: str
  # Optional: NO
  'receivers_file': 'examples/WHAM/receivers.dat',
}
```

Note that you may wish to change the location of the input files to somewhere on your drive. By default PETGEM will create the output directory at same level where the parameters file is located. For this example and following the PETSc documentation, the solver options have been configured in the file as petsc.opts follows:

```
# Solver options for PETSc
-ksp_type gmres
-pc_type sor
-ksp_rtol 1e-8
-ksp_converged_reason
-log_summary
```

That's it, we are now ready to solve the modelling.

Running PETGEM

To run the simulation, the following command should be run in the top-level directory of the PETGEM source tree:

```
$ mpirun -n 16 python3 kernel.py -options_file examples/WHAM/petsc.opts examples/WHAM/params_file.py
```

PETGEM solves the problem and outputs the solution to the output path (`examples/WHAM/Output/`). The output files will be PETSc binary format. By default PETGEM save the electric field components in different files:

- EpX.dat, EpY.dat, EpZ.dat: primary electric field components.
- EsX.dat, EsY.dat, EsZ.dat: secondary electric field components.
- EtX.dat, EtY.dat, EtZ.dat: total electric field components.

PETGEM postprocessing

Once the simulation has ended, PETGEM output can be imported to Matlab by using the script `utils/Postprocessing/petgem_import.m`. Hence, in order to import the x-component of the total electric field for this modelling, `petgem_preprocessing.m` should be invoked has follows:

```
$ [EtX] = petgem_import('examples/WHAM/Output/EtX.dat')
```

The dimension of arrays of EtX is determined by the number of receivers. Once the arrays are imported, feel free to handling and ploting. *Figure 7.7* shows a comparison of the x-component of total electric field between PETGEM results and the quasi-analytical solution obtained with the WHAM tool. In *Figure 7.7* it is easy to see the effect of the imperfect absorbing boundaries which can be mitigated by enlargening the domain with element sizes increasing logarithmically outwards from the zone of interest. The total electric field in *Figure 7.7* was calculated using a mesh with $\approx$ 12 millions of edges (degrees of freedom).

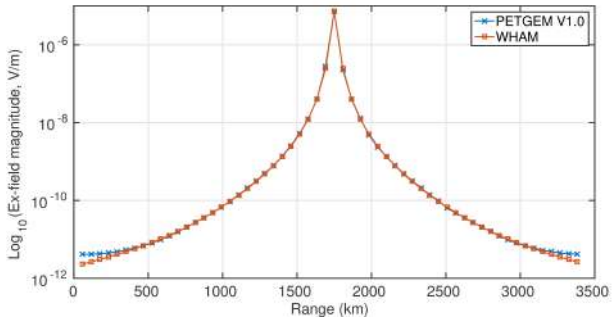


Fig. 6.7: Total electric field comparative for x-component between PETGEM V1.0 and WHAM. The effect of our imperfect absorbing boundaries can be mitigated by enlargening the computational domain with element sizes increasing logarithmically outwards from the zone of interest.

Code documentation

Following sub-sections are dedicated to code documentation of PETGEM.

Setup & install scripts

setup.py

setup.py is the main PETGEM setup script, it is based on python setup-module.

setup.**check_versions**(*show_only=False*)
Check all requires packages for **PETGEM** installation.

setup.**configuration**(*parent_package=''*, *top_path=None*)
Configure a local environment.

Parameters

- **parent_package** (*str*) – initial parent package of PETGEM
- **top_path** (*str*) – top path of PETGEM directory

Returns config structure

Return type list

setup.**setup_package**()
Setup a package for a specific installation.

builder.py

Build helper for setup script. It includes dependency checks and monkey-patch in order to extend or modify supporting system software locally (affecting only the running instance of the program).

Note: The original version of this script was adapted from NiPy project.

class builder.**Clean**(*dist*)
Distutils command class to clean. Enhanced to clean also files generated during *python setup.py build_ext –inplace* process.

run()
Run clean process recursively for directories: petgem, examples, utils, tests, doc and root directory.

class builder.**NoOptionsDocs**(*dist*)
Handler class for no options docs

finalize_options()
Finalize options for NoOptionsDocs class.

initialize_options()
Init options for NoOptionsDocs class.

user_options = [('None', None, 'this command has no options')]

class builder.**SphinxHTMLDocs**(*dist*)
Generate html docs by Sphinx.

run()
Run SphinxHTMLDocs class.

class builder.**SphinxPDFDocs**(*dist*)
Generate pdf docs by Sphinx

run()
Run SphinxPDFDocs class.

builder.**get_sphinx_make_command**()
Get make command for Sphinx documentation.

builder.**package_check**(*pkg_name*, *version=None*, *optional=False*, *checker=<class 'distutils.version.LooseVersion'>*, *version_getter=None*, *messages=None*, *show_only=False*)
Check if package *pkg_name* is present, and in correct version.

Parameters

- **pkg_name** (*str, list*) – the name of the package as imported into python. Alternative names (e.g. for different versions) may be given in a list.
- **version** (*str*) – the minimum version of the package that is required. If not given, the version is not checked.
- **optional** (*bool*) – if False, raise error for absent package or wrong version, otherwise warning.
- **checker** (*callalble*) – if given, the callable with which to return a comparable thing from a version string. The default is `distutils.version.LooseVersion`.
- **version_getter** (*callable*) – if given, the callable that takes *pkg_name* as argument, and returns the package version string as in:

  ```
  version = version_getter(pkg_name)
  ```

  The default is equivalent to:

  ```
  mod = __import__(pkg_name); version = mod.__version__
  ```

- **messages** (*dict*) – if given, the dictionary providing (some of) output messages.
- **show_only** (*bool*) – if True, do not raise exceptions, only show the package name and version information.

builder.**recursive_glob**(*top_dir*, *pattern*)
Finds all the pathnames matching a specific pattern according to the rules used by Unix shell. `recursive_glob` works like `glob.glob()`, but in working recursively.

Parameters

- **top_dir** (*str*) – the top-level directory
- **pattern** (*str, list*) – the pattern or list of patterns to match

builder.**unitary_test**()
Unitary test for builder.py script. Check if version, top_dir and in_source are present in INFO PETGEM object.

config.py

Setup config object for **PETGEM**, namely: python version, system, flags and links for C extension (if C dependencies exist).

class petgem.config.**Config**
Setup config object for **PETGEM**

compile_flags()
Setup flags for C compilation (is C code exist).

debug_flags()
Setup debug flags.

is_release()
Setup a release version of **PETGEM**.

link_flags()
Setup links for flags.

numpydoc_path()
Setup numpydoc path

python_include()
Setup python include.

python_version()
Setup python version.

system()
Setup system.

petgem.config.**has_attr**(*obj*, *attr*)

setup.py (petgem)

Setup main path directories for **PETGEM**.

petgem.setup.**configuration**(*parent_package=''*, *top_path=None*)
Setup parent package and child packages.

version.py

Setup package versions require for **PETGEM**.

petgem.version.**get_basic_info**(*version='2016.1'*)
Return PETGEM installation directory information. Append current git commit hash to *version*.

setup.py (base directory)

Setup path for base directory into petgem parent directory.

petgem.base.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config base directory.

setup.py (decomposition directory)

Setup path for decomposition directory into petgem parent directory.

petgem.decomposition.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config decomposition directory.

setup.py (efem directory)

Setup path for efem directory into petgem parent directory.

petgem.efem.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config efem directory.

setup.py (mesh directory)

Setup path for mesh directory into petgem parent directory.

petgem.mesh.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config mesh directory.

setup.py (monitoring directory)

Setup path for monitoring directory into petgem parent directory.

petgem.monitoring.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config monitor directory.

setup.py (parallel directory)

Setup path for parallel directory into petgem parent directory.

petgem.parallel.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config parallel directory.

setup.py (postprocessing directory)

Setup path for postprocessing directory into petgem parent directory.

petgem.postprocessing.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config postprocessing directory.

setup.py (solver directory)

Setup path for solver directory into petgem parent directory.

petgem.solver.setup.**configuration**(*parent_package=''*, *top_path=None*)
Config solver directory.

kernel

kernel.py

PETGEM kernel. It solve a CSEM forward modelling according to parameters into a **PETGEM** parameters file format.

Basis scripts

base.py

Define base operations for **PETGEM** such as: check init params, data types, import files and timers.

petgem.base.base.**checkDictionaryConsistencyMaster**(*rank*, *in_dict*, *file_name*, *dir_name*)
Check if dictionary consistency match with **PETGEM** requirements. (master task)

Params int rank MPI rank.

Params dict in_dict input dictionary to be tested.

Params str file_name parameters file name.

Params str dir_name parent directory of file_name

Returns csem modelling dictionary after test.

Return type csem_modelling dictionary.

petgem.base.base.**checkDictionaryConsistencySlave**(*rank*, *in_dict*, *file_name*, *dir_name*)
Check if dictionary consistency match with **PETGEM** requirements. (slave task)

Params int rank MPI rank.

Params dict in_dict input dictionary to be tested.

Params str file_name parameters file name.

Params str dir_name parent directory of file_name

Returns csem modelling dictionary after test.

Return type csem_modelling dictionary.

petgem.base.base.**checkDirectoryPath**(*in_directory_path*)
Determine if exists a directory.

Params str in_directory_path directory name to be checked.

Returns success.

Return type bool

petgem.base.base.**checkFilePath**(*in_file_path*)
Determine if exists a file.

Params str in_file_path file name to be checked.

Returns success.

Return type bool

petgem.base.base.**checkNumberParams**(*init_params*)
Check number of initial kernel parameters.

Parameters **init_params** (*list*) – list of initial kernel parameters.

Returns a parameters file name.

Return type str.

Note: if the number of init_params is different to 2, **PETGEM** kernel will stop.

---

petgem.base.base.**readUserParams**(*input_params*, *rank*)
Read a kernel input, namely a parameters file name.

Params list input_params user input parameters.

Parameters **rank** (*int*) – MPI rank.

Returns a modelling dictionary.

Return type dict of type modelling.

petgem.base.base.**unitary_test**()
Unitary test for base.py script.

styles.py

Define styles for **PETGEM** screen-output such as: string formats, headers, and footers.

petgem.base.styles.**petgemFooter**()
Setup the **PETGEM** footer to be printed in screen.

petgem.base.styles.**petgemHeader**()
Setup the **PETGEM** header to be printed in screen.

petgem.base.styles.**set_str_format**(*in_string*, *FORMAT=None*)
Setup a string with a specific format

Parameters

- **in_string** (*str*) – string to be formated.
- **format** (*str*) – format to be applied.

Returns formated string.

Return type str.

Note: valid formats are: Warning, Error, OkGreen and OkBlue

petgem.base.styles.**test_footer**(*pass_test*)
Print the footer of a unitary test.

Parameters **pass_test** (*bool*) – boolean that express if a test is, or not, passed.

petgem.base.styles.**test_header**(*caller*)
Print the header of a unitary test.

Parameters **caller** (*str*) – name of caller (test owner).

petgem.base.styles.**unitary_test**()
Unitary test for styles.py script.

modelling.py

Define the csem_modelling dictionary. csem_modelling dictionary contain the main initial parameters for a CSEM FM modelling such as: frequency, source position, conductivity model and mesh information.

petgem.base.modelling.**CSEM_MODELLING**(*rank*, *freq*, *src_pos*, *src_direc*, *src_current*, *src_length*, *sigma_background*, *sigma_file*, *nodes_file*, *elemsN_file*, *elemsE_file*, *edgesN_file*, *nnz_file*, *bEdges_file*, *receivers_file*, *dir_name*)
csem_modelling dictionary with main parameters for CSEM FM.

Parameters

- **rank** (*int*) – MPI rank.
- **freq** (*int, float*) – frequency.
- **src_pos** (*list*) – source position.
- **src_dir** (*int, float*) – source orientation.
- **src_current** (*int, float*) – source current.
- **src_length** (*int, float*) – source length.

- **sigma_background** (*int, float*) – background conductivity.
- **sigma_file** (*str*) – file name of conductivity model.
- **nodes_file** (*str*) – file name of node spatial coordinates.
- **elemsN_file** (*str*) – file name of elements-nodes connectivity.
- **elemsE_file** (*str*) – file name of elements-edges connectivity.
- **edgesN_file** (*str*) – file name of edges-nodes connectivity.
- **nnz_file** (*str*) – file name of nnz for matrix allocation.
- **bEdges_file** (*str*) – file name of boundary edges.
- **receivers_file** (*str*) – file name or receivers position.
- **dir_name** (*str*) – parent directory of sigma_file, nodes_file

and elemsN_file.

petgem.base.modelling.**printModellingData**(*input_modelling*)
Print the content of a csem_modelling dictionary. :param dictionary: input_modelling.

petgem.base.modelling.**unitary_test**()
Unitary test for modelling.py script.

Mesh scripts

mesh.py

Define functions for mesh handling.

petgem.mesh.mesh.**get_nNodes_nElems**(*matrixNodal*, *matrixElems*)
Compute the number of elements and the number of nodes of a given mesh.

Params list matrixNodal nodal coordinates.

Params list matrixElems elemsN connectivity.

petgem.mesh.mesh.**printMeshInfo**(*nElems*, *nEdges*, *dofs*, *bEdges*)
Print data mesh.

Parameters

- **nElems** (*int*) – number of elements.
- **nEdges** (*int*) – number of edges.
- **dofs** (*int*) – number of degrees of freedom.
- **bEdges** (*int*) – number of boundary edges.

Returns none

petgem.mesh.mesh.**readHdf5**(*file_name*, *DATA=None*)
Read an hdf5 file.

Parameters

- **file_name** (*str*) – file name to be readed.
- **DATA** (*str*) – type of DATA to be readed: nodes, elemsN or elemsS.

Returns nodes, elemsN or elemsSigma.

Return type ndarray.

petgem.mesh.mesh.**readMesh**(*input_modelling*)
Read a tetrahedral mesh defined by two HDF5 files: nodes description and elements connectivity description.

Parameters **input_modelling** (*dictionay*) – csem_modelling dictionary.

Returns arrays with nodes and elements connectivity.

Return type ndarray.

petgem.mesh.mesh.**unitary_test**()
Unitary test for mesh.py script.

EFEM scripts

efem.py

Define the classes, methods and functions for Edge Finite Element Method (EFEM) of lowest order in tetrahedral meshes, namely, Nedelec elements.

petgem.efem.efem.**compute_boundary_edges**(*edgesN*, *bfacesN*)
Compute boundary edges of a tetrahedral mesh.

Parameters

- **elemsN** (*ndarray*) – edges-nodes connectivity.
- **bfacesN** (*ndarray*) – boundary-faces-nodes connectivity.

Returns boundary-edges connectivity.

Return type ndarray

petgem.efem.efem.**compute_boundary_faces**(*elemsF*, *facesN*)
Compute boundary faces of a tetrahedral mesh.

Parameters

- **elemsF** (*ndarray*) – elements-face connectivity.
- **facesN** (*ndarray*) – faces-nodes connectivity.

Returns boundary-faces connectivity.

Return type ndarray

petgem.efem.efem.**compute_edges**(*elemsN*)
Compute edges of a 3D tetrahedral mesh.

Parameters **elemsN** (*ndarray*) – elements-nodes connectivity.

Returns element/edges connectivity and edges/nodes connectivity.

Return type ndarray

petgem.efem.efem.**compute_faces**(*elemsN*)
Compute the element's faces of a 3D tetrahedral mesh.

Parameters **matrix** (*ndarray*) – elements-nodes connectivity.

Returns element/faces connectivity.

Return type ndarray

Note: References:

Rognes, Marie E., Robert Cndarray. Kirby, and Anders Logg. "Efficient assembly of H(div) and H(curl) conforming finite elements." SIAM Journal on Scientific Computing 31.6 (2009): 4130-4151.

petgem.efem.efem.**unitary_test**()
Unitary test for efem.py script.

fem.py

Define the classes, methods and functions for Finite Element Method (FEM) of lowest order in tetrahedral meshes.

petgem.efem.fem.**gauss_points_tetrahedron**(*polyOrder*)
Compute the quadrature points X and the weights W for the integration over the unit tetrahedra whose nodes are (0,0,0), (1,0,0), (0,1,0) and (0,0,1).

Parameters **polyOrder** (*int*) – degree of polynominal

Returns quadrature Gauss points and Gauss weights.

Return type ndarray.

Note: References:

P Keast, Moderate degree tetrahedral quadrature formulas, CMAME 55: 339-348 (1986).

O.C. Zienkiewicz, The Finite Element Method, Sixth Edition.

petgem.efem.fem.**tetraXiEtaZeta2XYZ**(*eleNodes*, *XiEtaZetaPoints*)
Map a set of points in XiEtaZeta coordinates to XYZ coordinates.

Parameters

- **eleNodes** (*ndarray*) – nodal spatial coordinates of the tetrahedral element.
- **XiEtaZetaPoints** (*ndarray*) – set of points in XiEtaZeta coordinates.

Returns new spatial coordinates of XiEtaZetaPoints.

Return type ndarray.

petgem.efem.fem.**unitary_test**()
Unitary test for fem.py script.

general_functions.py

Define general and common functions for edge finite element method (EFEM) of lowest order in tetrahedral meshes, namely, Nedele elements.

petgem.efem.general_functions.**compute_items**(*elemsE*, *elemsF*, *bEdges*, *nodes*)
Compute the number of elements, number of nodes, number of edges and number of boundary edges of a tetrahedral mesh in the Edge Finite Element Method.

Parameters

- **elemsE** (*ndarray*) – element/edges connectivity.
- **elemsF** (*ndarray*) – element/faces connectivity.

- **bEdges** (*ndarray*) – boundary-edges connectivity.
- **nodes** (*ndarray*) – nodal coordinates.

Returns None.

petgem.efem.general_functions.**unitary_test**()
Unitary test for general.py script.

nedelec_elements.py

Define functions for Nedelec elements of lowest order.

petgem.efem.nedelec_elements.**compute_analytic_element_matrix**(*eleNodes*, *eleVol*, *lengthE*, *signs*)
Compute tetrahedral elemental matrices in an analytic manner.

Parameters

- **eleNodes** (*ndarray*) – nodal spatial coordinates of the element.
- **eleVol** (*float*) – element's volume.
- **lengthE** (*ndarray*) – element's edges defined by their length.
- **signs** (*ndarray*) – local edge signs.

Returns stiffness matrix and mass matrix.

Return type ndarray.

petgem.efem.nedelec_elements.**compute_signs_edges**(*indx_nodes*)
Compute the local direction of element's-th edges with respect to global direction of edges within the mesh. This data is needed in order to use linear Nedelec elements in 3D. The edge signs can be easily deduced from the mesh data itself by directly using the data structure which represents the elements (tetrahedrons) by their node indexes. As consequence, the signs are obtained with minimal matrix operations in a vectorized manner.

signs(i) is the sign related to i'th edge of the element-th.

edge1 = [1 2] edge2 = [1 3] edge3 = [1 4] edge4 = [2 3] edge5 = [4 2] edge6 = [3 4]

Parameters **indx_nodes** (*ndarray*) – nodal indexes of the element.

Note: References:

Jin, Jian-Ming. The finite element method in electromagnetics. John Wiley & Sons, 2002.

---

petgem.efem.nedelec_elements.**nedelec_basis_iterative**(*eleNodes*, *points*, *eleVol*, *lengthEdges*)
Compute the basis Nedelec functions in an iterative way for a set of points in a given element.

Parameters

- **eleNodes** (*ndarray*) – nodal spatial coordinates of the element.
- **points** (*ndarray*) – spatial coordinates of the evaluation points.
- **eleVol** (*float*) – element's volume.
- **lengthEdges** (*ndarray*) – element's edges defined by their length.

Returns values of Nedelec functions.

Return type ndarray.

petgem.efem.nedelec_elements.**nedelec_basis_vectorized**(*eleNodes*, *points*, *eleVol*, *lengthEdges*)

Compute the basis Nedelec functions in a vectorized way for a set of points in a given element.

Parameters

- **eleNodes** (*ndarray*) – nodal spatial coordinates of the element.
- **points** (*ndarray*) – spatial coordinates of the evaluation points.
- **eleVol** (*float*) – element's volume.
- **lengthEdges** (*ndarray*) – element's edges defined by their length.

Returns values of Nedelec functions.

Return type ndarray.

petgem.efem.nedelec_elements.**unitary_test**()

Unitary test for nedelec_elements.py script.

vector_matrix_functions.py

Define standard vector and matrix functions.

petgem.efem.vector_matrix_functions.**delete_duplicate_rows**(*matrix*)

Delete duplicate rows in a matrix.

Parameters **matrix** (*ndarray*) – input matrix to be processed.

Returns matrix without duplicate rows

Return type ndarray

petgem.efem.vector_matrix_functions.**find_unique_rows**(*array*, *return_index=False*, *return_inverse=False*)

Find unique rows of a two-dimensional numpy array.

Parameters

- **ndarray** – array to be processed.
- **return_index** (*bool*) – the indices of array that result in the unique array.
- **return_inverse** (*bool*) – indices of the unique array that can be used to reconstruct array.

petgem.efem.vector_matrix_functions.**unitary_test**()

Unitary test for vector_matrix_functions.py script.

Solver

assembler.py

Define functions for assembly of sparse linear systems in Edge Finite Element Method (EFEM) of lowest order in tetrahedral meshes.

petgem.solver.assembler.**computeElementalContributionsMPI**(*modelling*, *coordEle*, *nodesEle*, *sigmaEle*)

Compute the elemental contributions of matrix A (LHS) and right hand side (RHS) in a parallel-vectorized manner for CSEM surveys by EFEM. Here, all necessary arrays are populated (Distributed-memory approach).

Parameters

- **modelling** (*int*) – CSEM modelling with physical parameters.
- **coordEle** (*ndarray*) – array with nodal coordinates of element.
- **nodesEle** (*ndarray*) – array with nodal indexes of element.
- **sigmaEle** (*int*) – element conductiviy.

Returns Ae, be.

Return type complex.

petgem.solver.assembler.**unitary_test**()
Unitary test for assembler.py script.

solver.py

Define functions to find the solution of a sparse linear system of the format Ax = b, in Edge Finite Element Method (EFEM) of lowest order in tetrahedral meshes.

petgem.solver.solver.**direct_scipy**(*A*, *b*)
Interface to direct sparse scipy solver.

Parameters

- **A** (*ndarray*) – sparse and complex coefficients matrix in a CSR format.
- **b** (*ndarray*) – rigth hand side.

Returns solution to the system.

Return type ndarray of complex coefficients.

petgem.solver.solver.**iterative_scipy**(*A*, *b*, *type_solver*, *tol*, *maxiter*)
Interface to iterative sparse scipy solver.

Parameters

- **A** (*ndarray*) – sparse and complex coefficients matrix in a CSR format.
- **b** (*ndarray*) – rigth hand side.
- **type_solver** (*str*) – solver type. Available solvers are: bicgstab.
- **tol** (*float*) – solver tolerance.
- **maxiter** (*int*) – maximum number of solver iterations.

Returns solution to the system.

Return type ndarray of complex coefficients.

petgem.solver.solver.**set_dirichlet_boundaries**(*A*, *b*, *in_boundaries*)
Compute the list of degrees of freedom.

Parameters

- **A** (*ndarray*) – sparse and complex coefficients matrix in a CSR format.
- **b** (*ndarray*) – rigth hand side.
- **in_boundaries** (*ndarray*) – boundary-edges array.

Returns equation system after applied Dirichlet BC.

Return type ndarray.

petgem.solver.solver.**solver**(*A*, *b*, *boundaries*, *model*)
Solve a matrix system of the form Ax = b.

Parameters

- **A** (*ndarray*) – sparse and complex coefficients matrix in a CSR format.
- **b** (*ndarray*) – rigth hand side.
- **boundaries** (*ndarray*) – boundary-edges array.
- **model** (*object_modelling*) – CSEM modelling with physical parameters.

Returns solution to the system.

Return type ndarray of complex coefficients.

petgem.solver.solver.**unitary_test**()
Unitary test for solver.py script.

Parallel

parallel.py

Define parallel functions for Edge Finite Element Method (EFEM) of lowest order in tetrahedral meshes, namely, Nedelec elements.

petgem.parallel.parallel.**func_start_as_many**(*func_items_args*)
Compute function as many arguments.

Parameters **func_item_args** (*str*) – function arguments.

Returns arguments.

Return type arguments.

Note: Equivalent to:

func = func_item_args[0]

items = func_item_args[1]

args = func_item_args[2:]

return func(items[0],items[1],...,args[0],args[1],...)

---

petgem.parallel.parallel.**func_start_as_single**(*func_item_args*)
Compute function as single argument.

Parameters **func_item_args** (*str*) – function arguments.

Returns arguments.

Return type arguments.

Note: Equivalent to:

func = func_item_args[0]

item = func_item_args[1]

args = func_item_args[2:]

return func(item,args[0],args[1],...)

petgem.parallel.parallel.**get_num_processors**(*num_proc*)

Compute the number of processors that had been insert by the user against available processors in the platform.

Params int num_proc number of processors to set parallel environment.

Returns number of processors to set parallel environment.

Return type int

Note: if num_proc > available_processors then nProcesses=available_processors

petgem.parallel.parallel.**parallel_map**(*func*, *iterable*, **args*, ***kwargs*)

Parallel mapping of a function. Equivalent to:

return [function(x, args[0], args[1],...) for x in iterable]

Parameters

- **func** (*str*) – function name.
- **iterable** (*int*) – iterable variable.
- **args** (*argument*) – function arguments.
- **kwargs** (*argument*) – keyword arguments.

Keyword arguments are defined as follows:

•parallel = True/False: Force parallelization on/off

•chunksize = see multiprocessing.Pool().map

•pool = multiprocessing.Pool() Pass an existing pool.

•processes = see multiprocessing.Pool() processes argument

Note: The original version of this function was adapted from J.F. Sebastian and from parmap python module.

petgem.parallel.parallel.**parallel_map_async**(*func*, *iterable*, **args*, ***kwargs*)

This function is the multiprocessing.Pool.map_async version that supports multiple arguments.

```
>>> [function(x, args[0], args[1],...) for x in iterable]
```

Parameters

- **parallel** (*bool*) – Force parallelization on/off.
- **chunksize** (*int*) – see *multiprocessing.pool.Pool*.
- **callback** (*function*) – see *multiprocessing.pool.Pool*.
- **error_callback** (*function*) – see *multiprocessing.pool.Pool*.
- **pool** (*multiprocessing.pool.Pool*) – Pass an existing pool.
- **processes** (*int*) – Number of processes to use in the pool.

petgem.parallel.parallel.**parallel_starmap**(*func*, *iterables*, **args*, ***kwargs*)
Parallel start_mapping of a function. Equivalent to:

return [function(x1,x2,x3,..., args[0], args[1],...) for (x1,x2,x3...) in iterable].

Parameters

- **func** (*str*) – function name.
- **iterables** (*int*) – iterable variables.
- **args** (*argument*) – function arguments.
- **kwargs** (*argument*) – keyword arguments.

Keyword arguments are defined as follows:

•parallel = True/False: Force parallelization on/off

•chunksize = see multiprocessing.Pool().map

•pool = multiprocessing.Pool() Pass an existing pool.

•processes = see multiprocessing.Pool() processes argument

Note: The original version of this function was adapted from J.F. Sebastian and from parmap python module.

petgem.parallel.parallel.**unitary_test**()
Unitary test for parallel.py script.

Postprocessing

postprocessing.py

Define the functions for post-processing stage.

petgem.postprocessing.postprocessing.**EpReceiverComputation**(*model*, *point*)
Compute the primary electric field for an array of point (receivers).

Parameters

- **model** (*object_modelling*) – CSEM modelling with physical parameters.
- **point** (*ndarray*) – receiver spatial coordinates.

Returns primary electric field on receivers.

Return type ndarray.

petgem.postprocessing.postprocessing.**EsReceiverComputation**(*field*, *coordEle*, *coordReceiver*, *nodesEle*)
Compute the secondary electric field on receivers.

Parameters

- **field** (*ndarray*) – secondary field to be interpolated.
- **coordElement** (*ndarray*) – element spatial coordinates.
- **coordReceiver** (*ndarray*) – receiver spatial coordinates.
- **nodesEle** (*ndarray*) – nodal indices of element (element container).

Returns secondary electric field on receivers.

Return type ndarray.

petgem.postprocessing.postprocessing.**EtReceiverComputation**(*primary_field*, *secondary_field*)

Compute the total electric field on receivers.

Parameters

- **primary_field** (*ndarray*) – primary electric field on receiver.
- **secondary_field** (*ndarray*) – secondary electric field on receiver.

Returns total electric field on receivers.

Return type ndarray.

petgem.postprocessing.postprocessing.**computeFieldsReceiver**(*modelling*, *coordReceiver*, *coordElement*, *nodesElement*, *x_field*)

Compute the CSEM modelling output: primary electric field, secondary electric field and total electric field on receivers position.

Parameters

- **model** (*object_modelling*) – CSEM modelling with physical parameters.
- **coordReceiver** (*ndarray*) – receiver spatial coordinates.
- **coordElement** (*ndarray*) – element spatial coordinates.
- **nodesElement** (*ndarray*) – nodal indices of element (element container).
- **x_field** (*ndarray*) – vector solution for receiver.

Returns primary, secondary and total electric field

Return type ndarray

petgem.postprocessing.postprocessing.**create_directory_output**(*out_dir*)

petgem.postprocessing.postprocessing.**export_output**(*primary_field*, *secondary_field*, *total_field*, *out_prefix*, *out_dir*)

Export the results of CSEM FM modelling by using EFEM in a HDF5 format, namely, this function export the primary field (Ep), secondary field (Es) and total field (Et).

Parameters

- **primary_field** (*ndarray*) – primary electric field on receivers.
- **secondary_field** (*ndarray*) – secondary electric field on receivers.
- **total_field** (*ndarray*) – total electric field on receivers.

Returns None

petgem.postprocessing.postprocessing.**postprocessing**(*model*, *field*, *mesh*, *edgesN*, *elemsE*)

Compute the CSEM modelling output: primary electric field, secondary electric field and total electric field on receivers position.

Parameters

- **model** (*object_modelling*) – CSEM modelling with physical parameters.
- **field** (*ndarray*) – secondary field to be interpolated.

- **mesh** (*mesh_object*) – mesh object with element/nodes connectivity, element/nodes connectivity, nodal coordinates and element/sigma values.
- **edgesN** (*ndarray*) – edges/nodes connectivity.
- **elemsE** (*ndarray*) – element/edges connectivity.

petgem.postprocessing.postprocessing.**read_receivers_file**(*file_name*)
Read a receivers file in a hdf5 format. It file contain receivers spatial coordinates.

Parameters **receivers_file** (*str*) – file name to be readed.

Returns receivers spatial coordinates.

Return type ndarray.

petgem.postprocessing.postprocessing.**unitary_test**()
Unitary test for post_processing.py script.

Monitoring

monitoring.py

Define functions for performance monitoring such as timers and HW counters.

petgem.monitoring.monitoring.**printTimes**(*times_array*)
Print, in a pretty manner, a times array.

Parameters **times_array** (*list*) – array of times for each **PETGEM** phase.

petgem.monitoring.monitoring.**timing**(*name_function*)
Measure spent time in a given function.

Parameters **name_function** (*str*) – name function to be measured.

petgem.monitoring.monitoring.**unitary_test**()
Unitary test for mesh.py script.

Examples

params_file_template.py

Parameters file template for CSEM forward modelling (FM).

By definition, any model should include: physical parameters, mesh file, source and receivers files, computational issues (solver type, domain decomposition) and output file format.

In order to avoid a specific parser, this file is imported by **PETGEM** as a Python dictionary. As consequence, the dictionary name and his key names MUST NOT BE changed.

All file paths should consider as absolute.

Next, each key is described.

CHAPTER

SEVEN

PUBLICATIONS

Papers:

- Castillo-Reyes, O., de la Puente, Cela, J. M. Three-Dimensional CSEM modelling on unstructured tetrahedral meshes using edge finite elements. In: Barrios Hernández C., Gitler I., Klapp J. (eds) High Performance Computing. CARLA 2016. Communications in Computer and Information Science, vol 697: 247-256. ISBN 978-3-319-57971-9 Springer, Cham
- Castillo-Reyes, O., de la Puente, Cela, J. M. Improving edge finite element assembly for geophysical electromagnetic modelling on shared-memory architectures. 7th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference – UEMCON. 2016.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Edge-based parallel framework for the simulation of 3D CSEM surveys. ICE Barcelona-AAPG/SEG International Conference & Exhibition. 2016.
- Castillo-Reyes, O., de la Puente, J., Barucq, H., Diaz, J., and Cela, J. M., Parallel and vectorized code for CSEM surveys in geophysics: An edge-based approach. ECCOMAS. 2016.
- Castillo-Reyes, O., de la Puente, J., Modesto, D., Puzyrev, V., and Cela, J. M., A parallel tool for numerical approximation of 3D electromagnetic surveys in geophysics. Computación y Sistemas: Topic trends in computing research, vol. 20, no. 1, pp 29-39. 2016.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Towards an HPC tool for simulation of 3D CSEM surveys: an edge-based approach. PRACEdays16 Conference. 2016.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Assessment of edge-based finite element technique for geophysical electromagnetic problems: efficiency, accuracy and reliability. Proceedings of the 1st Pan-American Congress on Computational Mechanics and XI Argentine Congress on Computational Mechanics. CIMNE, pp. 984-995, 2015.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Edge-based electric field formulation in 3D CSEM simulations: a parallel approach. Proceedings of the 6th International Conference and Workshop on Computing and Communication. IEEE, 2015.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Edge-elements for geophysical electromagnetic problems: a new implementation challenge. PRACEdays15 Conference. 2015.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., HPC and edge elements for geophysical electromagnetic problems: an overview. BSC Doctoral Symposium (2nd: 2015: Barcelona). 2015.
- Castillo-Reyes, O., de la Puente, J., Puzyrev, V., and Cela, J. M., Parallel and numerical issues of the edge finite element method for 3D controlled-source electromagnetic surveys. Proceedings of the International Conference on Computing Systems and Telematics. IEEE, 2015.

Conferences:

- Castillo-Reyes, O., de la Puente, Cela, J. M. PETGEM: potential of 3D CSEM modelling using a new HPC tool for exploration geophysics. Marelec 2017. University of Liverpool. Liverpool, United Kingdom.

- Castillo-Reyes, O., de la Puente, Cela, J. M. High performance computing using python: advances in geophysical electromagnetic modelling. Computing and Electromagnetics International Workshop. Polytechnic University of Catalonia. Barcelona, Spain.
- Castillo-Reyes, O. Python code for CSEM modelling in geophysics and HPC architectures: advances and challenges. 2do Foro Internacional de Talento Mexicano – Innovation Match MX 2016-2017. Mexico, D.F.
- Castillo-Reyes, O., de la Puente, Cela, J. M. Python for HPC geophysical applications. GeoPython 2017. University of Applied Sciences and Arts Northwestern Switzerland. Basel, Switzerland.
- Castillo-Reyes, O. Python for HPC geophysical electromagnetic applications: experiences and perspectives. 4th BSC International Doctoral Symposium. Barcelona, Spain.
- Castillo-Reyes, O. See underneath. High Performance Computing, geophysics and electromagnetic methods. Interdisciplinary Meeting of Predoctoral Researchers – JIPI 2017. University of Barcelona. Barcelona, Spain. February 2017.
- Castillo-Reyes, O. Supercomputing and electromagnetic modelling in geophysics: advances and challenges. Centro de Ciencias de la tierra. University of Veracruz. Xalapa, Veracruz, Mexico. December 2016.
- Castillo-Reyes, O. Improving edge finite element assembly for geophysical electromagnetic modelling on shared-memory architectures. 7th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference – UEMCON 2016. New York, USA. October 2016.
- Castillo-Reyes, O. Three-dimensional CSEM modelling on unstructured tetrahedral meshes using edge finite elements. Latin American High Performance Computing Conference – CARLA 2016. Mexico, D.F. August 2016.
- Castillo-Reyes, O. Edge-based parallel code for CSEM surveys in geophysics: performance and accuracy improvements. 12th World Congress on Computational Mechanics – WCCM XII. Seúl, Corea. July 2016.
- Castillo-Reyes, O. Towards an HPC tool for 3D CSEM forward modelling in geophysics. Fourth International Congress on Multiphysics, Multiscale, and Optimization problems. Bilbao, España. May 2016
- Castillo-Reyes, O. High performance computing, geophysics and numerical methods: a symbiotic relation. 1er Foro Internacional de Talento Mexicano – Innovation Match MX 2015-2016. Guadalajara, Jalisco, México. April 2016.
- Castillo-Reyes, O. Edge-based electric field formulation in 3D CSEM simulations: a parallel approach. 6th International Conference and Workshop on Computing and Communication – IEMCON – 2015. University of British Columbia. Vancouver, Canada. October 2015.
- Castillo-Reyes, O. High Performance Computing and electromagnetic modeling in geophysics: from concepts to application. Research Center in Computing. National Polytechnic Institute. Mexico, D.F. October 2015.
- Castillo-Reyes, O. Parallel and numerical issues of the edge finite element method for 3D controlled-source electromagnetic surveys. IEEE International Conference on Computing Systems and Telematics. University of Veracruz. Xalapa, Veracruz, Mexico. October 2015.
- Castillo-Reyes, O. "Your Thesis in 3 Minutes" (3TM) with the topic: Edge-elements formulation of CSEM in geophysics: a parallel approach. Jornadas de Cooperación CONACyT – Cataluña 2015. Polytechnic University of Catalonia – National Council of Science and Technology of Mexico. Barcelona, Spain. June 2015.
- Castillo-Reyes, O. Edge-elements for geophysical electromagnetic problems: A new implementation challenge. PRACE Scientific and Industrial Conference 2015 – PRACEDays15. Dublin, Ireland. April 2015.
- Castillo-Reyes, O. HPC and edge elements for geophysical electromagnetic problems: an overview. 2nd BSC International Doctoral Symposium. Barcelona, Spain. April 2015.
- Castillo-Reyes, O. Assessment of edge-based finite element technique for geophysical electromagnetic problems: efficiency, accuracy and reliability. 1st. Pan-American Congress on Computational Mechanics – PANACM 2015. IACM. Buenos Aires, Argentina. April 2015.

- Castillo-Reyes, O. HPC solutions for oil industry: trends and challenges. Centro de Ciencias de la Tierra. University of Veracruz. Xalapa, Veracruz, Mexico. December 2014.
- Castillo-Reyes, O. High Performance Computing, Science and Engineering. Master in Telematic. School of Accounting and Management. University of Veracruz. Xalapa, Veracruz, Mexico. December 2014.
- Castillo-Reyes, O. HPC solutions for oil industry: trends and challenges. IV Simposio de Becarios CONACyT en Europa. Strasbourg, France. November 2014.

CHAPTER

EIGHT

SUPPORT

Work on PETGEM has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 644202. The research leading to these results has received funding from the European Union's Horizon 2020 Programme (2014-2020) and from Brazilian Ministry of Science, Technology and Innovation through Rede Nacional de Pesquisa (RNP) under the HPC4E Project , grant agreement No. 689772.

Castillo-Reyes expresses his gratitude to the Mexican National Council for Science and Technology (CONACyT) for his support.

CHAPTER

NINE

DOWNLOAD

PETGEM is developed as `open-source GPLv3`.

PETGEM distributed-memory release version 1.0

- `PETGEM_distributed_memory.zip`
- `PETGEM_distributed_memory.tar.gz`

PETGEM release version 1.0

- `PETGEM_shared_memory.zip`
- `PETGEM_shared_memory.tar.gz`

All files includes current patches, documentation and examples.

CHAPTER

TEN

CONTACT

Octavio Castillo Reyes
Nexus II - Planta 3
C/ Jordi Girona, 29
Barcelona
08034
+34 934137992
email: octavio.castillo@bsc.es

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

b

e

p

s