**Developing a 1D Magnetotelluric Inversion Web Application**

Report No.: GPM 9/15

By Stefan Spears

MSc. (Geophysics) Curtin

This report is presented as part of the requirement for the unit

ERTH4001 Geophysics Project – Thesis Parts A&B, totaling 75 credits

In the MSc Science – Geophysics Major from Curtin University.

The work is the result of supervised research; however, the report has been

prepared **by the student who is solely responsible for its contents**.

DEPARTMENT OF EXPLORATION GEOPHYSICS

Curtin University

November 2015

-----------------------------------------------------

~ Thus the heavens and the earth were completed in all their vast array.

Then God blessed the seventh day and made it holy, because on it he

rested from all the work of creating that he had done. (Genesis 2) ~

-------------------------------------------------------

**Acknowledgements**

I must say a heartfelt thank you to Dr. Andrew Pethick for his many varied roles throughout the life cycle of this project. His was one of guidance and mentoring during the process. I must also mention his patience and creativity in explaining the different ways the software could have been implemented. I must also say a heartfelt thank you to Dr. Brett Harris for his guidance, mentoring as well as his knowledge in the development of this software.

I wish to also thank my fellow MSc students in Geophysics for their help in many aspects during the course of our program, and for the memories and friendships I intend on keeping henceforth.

To my family, a special thank you for their understanding and unwavering support throughout this period in my life.

To my “Australian family” for making me feel home, away from home.

Finally to my close friends, for keeping me “on the straight and narrow.”

**Abstract**

The volume of geophysical data is growing at an exponential rate. This means geophysicists will need more advanced means of handling, processing and analyzing these large datasets. Currently, magnetotelluric applications are written for desktop only. Implementing cloud based web applications should overcome these modern limitations. To assess the feasibility and benefits of such systems a one-dimensional magnetotelluric web application was developed.

A web-based application utilizes cloud-computing technologies, providing many benefits to both software managers and geophysical users. Firstly, cloud based solutions are becoming widely adopted by businesses world wide because of cost benefits. Installation, support and maintenance costs are reduced because the application is hosted at a single virtual location. That is, simple installation, rapid update management, easy licensing and low hardware requirements for users due to the computation being performed within the cloud. Secondly, having the software on the cloud allow users to access the service from anywhere with any HTML5 compliant device.

The steps to create the software are presented. This includes, 1) developing magnetotelluric forward and inverse solvers in Python, 2) creating a generalized framework including geo-electrical and inversion parameters, 3) developing visualization capabilities for web use, 4) developing a web-based PHP engine to execute and handle output from the Python code and 5) designing and implementing a HTML web front end for user input and display. The complete application brings together a range of computer science and geophysical skillsets including the use of three programming languages. Python forms the basis of the geophysical computations while PHP and HTML are used to implement the web component. Altogether the web application comprises approximately 1000 lines of code that is split into various classes to allow for easy modification.

This research culminated with a working 1D magnetotelluric inversion web application. A major aim of this project was also to design the magnetotelluric platform to be extensible with further developments not limited to magnetotelluric methods. It is also my hope that the project will inspire the creation of other cloud-based geophysical methods.

Table of Contents

Chapter 1: Introduction 1

Chapter 2: Software Overview 5

2.1 Front End – User Input 8

2.2 Back End – Programming 12

2.3 Software Structure 13

2.4 Forward Modeling Derivation 14

Chapter 3: Benchmark Testing 18

3.1 Computation on the desktop 18

3.2 Computation on the Cray XC30 Supercomputer 18

Chapter 4: Case Study – Hydrological Investigations in Yanchep, Perth 20

Chapter 5: Discussion & Conclusion 24

Chapter 6: Executive Summary 26

Chapter 7: References 27

Chapter 8: Appendix 29

8.1 MT Global.py 30

8.2 MT Run.py 30

8.3 MT Inversion Parameters.py 32

8.4 MT Iteration.py 32

8.5 MT Invert.py 33

8.6 MT Earth.py 39

8.7 MT Data.py 40

8.8 Video of the MT 1Dnversion Web Application in use 43

**List of Figures**

Figure 2.1: A high level schematic of the web application 6

Figure 2-2: Example dataset showing proper format of data entry 9

Figure 2-3: Starting Earth Model example showing proper data input 10

Figure 2-4: “Invert” button used to initiate the inversion routine 11

Figure 2-5: Class Diagram of 1D MT Inversion Web Application 13

Figure 2-6: Schematic of a Geo-Electric Model 16

Figure 3-1: Compute time for 1500 Iterations 19

Figure 4-1: Case Study location showing position of Well and MT Station 20

Figure 4-2: Plot of Apparent Resistivity & Phase vs Frequency for well location 21

Figure 4-3: Final Earth Model for MT Station in Yanchep 22

Figure 4-4: Final Earth Model & plot of well data for case study location 23

# 

# Chapter 1: Introduction

The magnetotelluric (MT) method is a low frequency electromagnetic geophysical technique to detect subsurface conductivity distribution. The source of the magnetotelluric signal mainly originates from two places (i) the naturally occurring geomagnetic and geo-electric fields that occur at the ionosphere. These fields typically contain the lower frequency content below 1Hz ([Simpson and Bahr 2005](#_ENREF_17)) (ii) the electromagnetic waves generated by worldwide lightning activity. These fields contain high frequency MT content (i.e., 1Hz to 10 kHz) ([Simpson and Bahr 2005](#_ENREF_17)). These time varying magnetic fields that occur due to electric currents in the ionosphere, induce eddy currents in the earth. It is these fields that are observed on the earth’s surface ([Bailey 1970](#_ENREF_1)). Electrical resistivity variations are then determined from surface observations and are used to parameterize the earth into a conductivity distribution ([Pedersen and Hermance 1986](#_ENREF_13)). The magnetotelluric method can be performed to estimate electrical structures of deep sedimentary basins, lower crust and upper mantle ([Niwas, Gupta et al. 2005](#_ENREF_12)).

There are many advantages of using the telluric method, as in there is no current source which means no associated long leads, it combines flexibility, rapidity, is low cost and due to low frequency content has a greater depth of penetration than some other resistivity methods ([Cagniard 1953](#_ENREF_2)). In larger MT surveys, the volume of data collected can become significant. Due to the nature of electromagnetic inversion, as the volume and area of data increases, as does the computational expense. So far large 3D models consisting of hundreds of MT sites, inversion may require weeks on even the most powerful supercomputers ([Fraser, Vote et al. 2013](#_ENREF_4)).

Since it’s inception, the volume of data collected to run magnetotelluric inversion has been growing at an even-faster rate. Fraser, Vote et al. state that in the last five decades geoscientists in Australia have collected about 3 Petabytes of geoscience datasets ([2013](#_ENREF_4)). Demand for higher resolution, requirements for deeper exploration and improvements in technological progress have seen data being acquired at exponential rates. As such, time is running out for our desktop computers being an effective tool in analyzing these datasets. Already we are seeing insufficiencies in even the most modern small-scale computational platforms. It is therefore of paramount importance that we find an alternative solution. This project aims to design, test and benchmark a modern solution to shift the computation of MT inversion into the cloud. Cloud computing is one such solution as it provides scalable and cost effective compute resources ([Fraser, Vote et al. 2013](#_ENREF_4)). Server based applications can take advantage of cheap computational power to process and invert large volumes of geophysical data quickly.

The steps to recovering a sub-surface conductivity distribution from magnetotelluric data includes:

1. Processing the raw time series of electric and magnetic field measurements into spectral information.
2. Removing or weighting outliers and poorly resolved data points based on frequency for both TE and TM modes
3. Exporting the spectral data into apparent resistivity, phase and or impedance versus frequency for use in inversion applications.
4. Generating a reasonable starting geo-electrical model with accurate constraints based on prior geophysical or geological data.
5. Inverting data with appropriate inversion parameters.
6. Analyzing output and repeating appropriate steps.

Our research focuses on steps 4 to 6 and will be explained in further detail. Prior to step 4, quality control must be performed. This involves checking and processing the recorded MT frequency domain data. Outliers should be removed. This involves analyzing and consequentially the removal of data impacted by the MT dead bands resulting from either contamination due to sensor motion or lack of signal within a particular bandwidth. These include the AMT dead band (i.e., 1 – 5 kHz) ([Garcier 2008](#_ENREF_5)) and the MT dead band (0.2 – 5 Hz) ([Nichols, Morrision et al. 1988](#_ENREF_11)). Inversion is one of the final steps in obtaining an accurate representation of subsurface conductivity. It is therefore imperative to note that the quantitative approximation of boundaries of a layered earth model from magnetotelluric measurements in the field is an inverse problem ([Niwas, Gupta et al. 2005](#_ENREF_12)).

Inversion is “Deriving from field data a model to describe the subsurface that is consistent with the data; determining the cause from observational effects” ([Sheriff 1991](#_ENREF_16)). Essentially, given an electromagnetic MT field dataset, what sub-surface electrical distribution would result in the recorded results with the same survey geometry ([Bailey 1970](#_ENREF_1)). It is well known that geo-electrical equivalence is prevalent in magnetotelluric methods, that is, there may be many sub-surface electrical distributions that may result in the same response curves. One unconstrained solution is therefore insufficient. Niwas, Gupta et al. state that the process of inverting MT data is affected by the typical problem of non-uniqueness ([2005](#_ENREF_12)). Since inversion does not yield a unique solution imposing constraints on the models help stabilize the solution, reducing non-uniqueness ([Constable, Parker et al. 1987](#_ENREF_3)).

The aim of this project is to create new or repurpose existing inversion magnetotelluric code and integrate it into a cloud based web application framework. Web based applications are not new to geophysics, however, they are an inventive tactic to developing easily accessible geophysical routines ([Pethick and Harris 2015](#_ENREF_14)). At this present-day time, geophysical cloud computing usage and development is relatively low and the volume and maturity of software application programming interface (API’s) techniques and practices available pose significant challenges. Once this stage is completed, the boundary of geophysical computing would have been pushed a little further.

.

# Chapter 2: Software Overview

The purpose of the software is to image sub-surface conductivity changes from processed MT data all within a web-interface. The software needs to utilize different web and computation based technologies to work effectively. The steps for developing this application will be covered in greater detail, however, in short includes the development of:

1. Magnetotelluric forward and inverse solvers written in Python
2. A generalized framework including geo-electrical and inversion parameters
3. Code that facilitates visualization suitable for web use
4. A server side PHP engine to execute and handle output from the Python code
5. Designing and implementing a HTML web front end for user input and display

In addition to these steps each module must be connected with each other to form a seamless application, encapsulating both back end and web technologies from the user. The connection of the modules used in our application is best explained in the Figure below.

# 

### Figure 2.1: A high level schematic of the web application

There are three main components to the web application: 1) the HTML front end 2) the PHP handler & 3) the 1D MT Inversion Python code. The interactions and flow of the Information is shown by the arrow direction.

When users access the webpage, they invoke Hypertext Preprocessor (PHP) code that exists on the webserver. The resulting code that is delivered to the client post-PHP execution is Hypertext Modeling Language (HTML) script. HTML is what is interpreted by the browser and converted to a visual representation ([PHP\_5\_Tutorial 2015](#_ENREF_15)). With regard to the web application, when the “invert” button action is detected by the browser the PHP script accesses form data from the user and processes and pipes the output into the Python MT inversion code. Once executed the python code returns the result back to the PHP script that is executed again and shown on the webpage as HTML.

The steps to operate the software are as follows:

1. HTML - User inputs data
2. HTML - User enters a reasonable starting earth model
3. HTML - User clicks on “INVERT”
4. HTML - Form submits execution job to PHP script on server
5. PHP - Runs python script in the cloud
6. Python - Python script outputs data
7. PHP - Reads Python output
8. PHP – Restructure output and generate HTML

The steps listed above are a combination of both the front end and back end software structure. These will be explained further.

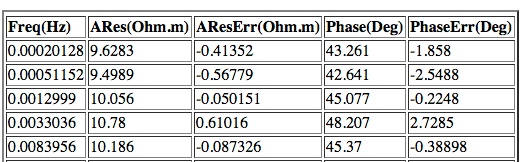
## 2.1 Front End – User Input

The web application’s user input requirements are considerably small, but necessary. We had two main objectives for user input:

1. Minimal ASCII text data entry
2. Easily performed data entry in all instances

Users must do three steps on the web portion of the application.

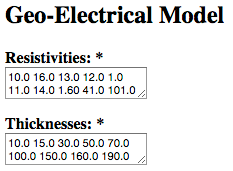
**Step 1**: User inputs data

****

### Figure 2-2: Example dataset showing proper format of data entry

The figure above is an example of the format required for proper data input. The application must receive data in the form of frequency, apparent resistivity, apparent resistivity error, phase and phase error in order for it to work effectively.

**Step 2**: User enters a reasonable starting earth model

****

### Figure 2-3: Starting Earth Model example showing proper data input

The figure above shows the two parameters that would make up the geo-electrical starting earth model: resistivities & thicknesses. It is important for users to input a reasonable starting model as it assists the application to give a solution that is close to the actual subsurface. The figure also shows the correct format for data entry (i.e., leaving a space between each entry).

**Step 3**: User clicks on “INVERT”



### Figure 2-4: “Invert” button used to initiate the inversion routine

When both steps one and two are complete, the program is ready to carry out the inversion. To initiate the inversion routine the user clicks on the “Invert” button as shown in the figure above. The remaining steps (4 to 8) to operate the application are completed on the back end. These are explained further.

## 2.2 Back End – Programming

As previously stated, the application was built using three programming languages: HTML, PHP & Python. The section of the application designed to compute forward models and inversion was written in Python. This language was chosen, as it was relatively simple to learn and use. As it is simple to learn and use, further development and maintenance of the application should also be simple.

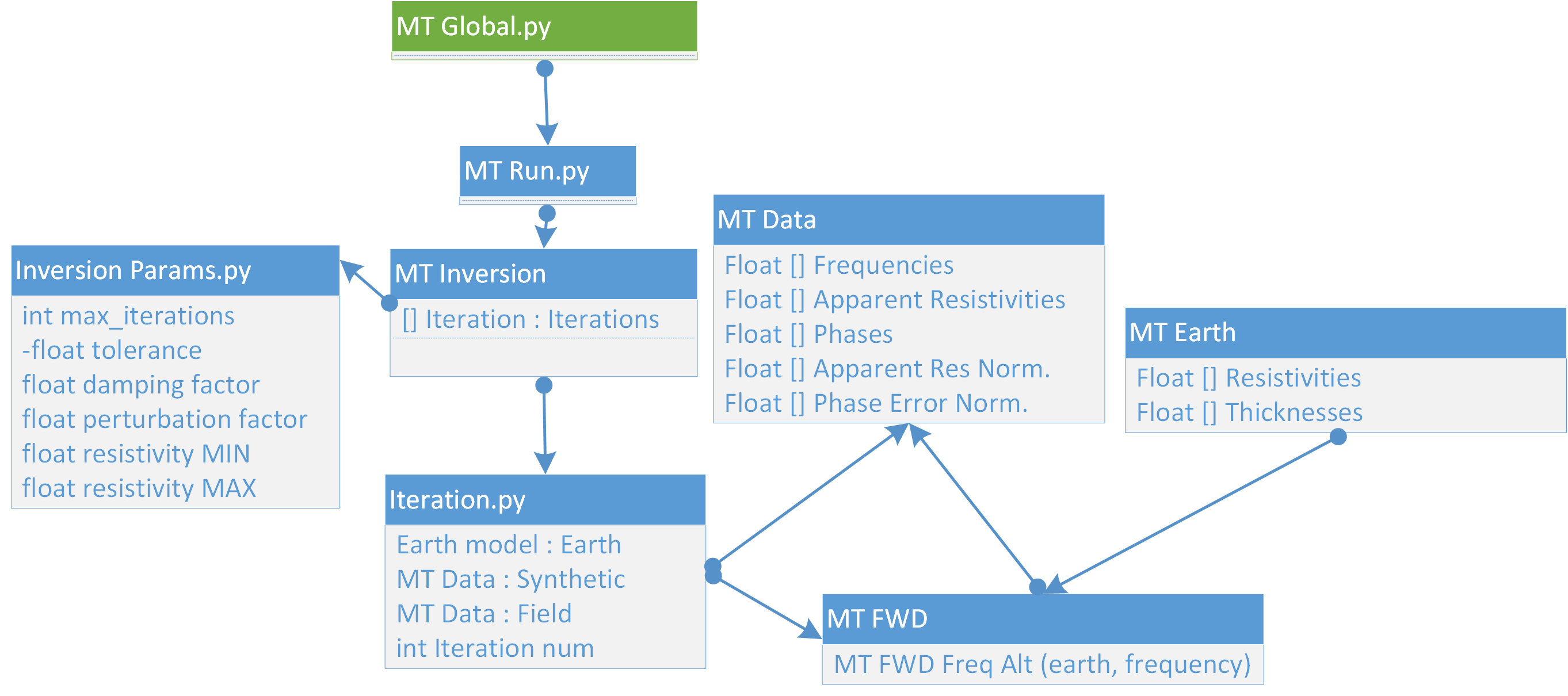
To aid in simplicity, the code was designed using Object Oriented Programming (OOP). OOP is achieved when code is designed as objects as opposed to action and data as opposed to line-by-line logic ([PHP\_5\_Tutorial 2015](#_ENREF_15)). This method was selected as it ensured the application could be modified easily. Ease of modification is accomplished by splitting up the various lines of code into classes, where a class is a program-code template for creating objects ([Mezini 1998](#_ENREF_10)).

The application was split up into classes based on four main headings:

1. Data
2. Earth
3. Inversion
4. Forward Model

Each of these main headings is further divided to make the application even simpler. The software structure highlighting the various classes is shown below. (See Figure 4)

## 2.3 Software Structure



### Figure 2-5: Class Diagram of 1D MT Inversion Web Application

The figure above shows the different classes created for the development of the software. The name of the class is highlighted in blue with the different objects in the grey thereafter. The name associated with the various classes represents what that specific class is responsible for in the application process. Once the individual classes were written, they were connected using the classes named “MT Global” and “MT Run”. These two main classes link the others to form the forward modeling and inversion Python module.

## 2.4 Forward Modeling Derivation

The code used for the forward modeling component was developed based on derivations from a few sources such as ([Niwas, Gupta et al. 2005](#_ENREF_12)) & ([Pedersen and Hermance 1986](#_ENREF_13)).

The surface transverse electric (TE) impedance is calculated and it can be thought of as a ratio of *Ex* to *Hy*

Where,

– Surface Impedance

ω - Angular Frequency (radians) 2**πf**

ρ - Resistivity (ohm.m)

μ0 – Magnetic Permeability (Henry/m) 4**π** x 10-7

*Ex* – Electric Field (V)

*Hy.* – Magnetic Field (T)

The halfspace impedance, Z, is initially calculated. In order to perform the calculation for layers above the halfspace we need to know about the energy that is reflected and transmitted at each layer boundary. This means that we must calculate reflection coefficient. However, prior to this step, the induction parameter, exponential factor and intrinsic impedance factors must be computed.

Induction Parameter = ϒj =

Exponential Factor = Ej =

Intrinsic Impedance = wj = ϒjρj

Where j represents the jth layer and is the conductivity of the layer, which is measured in Siemens/m.

Once these parameters are calculated we can now compute reflection coefficient.

Reflection Coefficient = Rj =

Impedance for each layer can now be computed from the reflection coefficient.

Layer Impedance = Zj =

Once impedance is calculated for each layer, impedance for layer 1 is used to determine apparent resistivity and phase.

Apparent Resistivity = ρa = |Z1|2

Phase = tan-1

30m

200m

Halfspace

60 ohm.m

120 ohm.m

300 ohm.m

N

N - 2

N - 1

### Figure 2-6: Schematic of a Geo-Electric Model

The application works by first calculating the impedance of the halfspace. Once this is calculated, the solution is passed to the layers above (one at a time) until calculation at the first layer is complete. Once complete calculations would have been completed for the number of layers – 1.

Since frequency is stored in an array, the application employs the use of a “for” loop to ensure each frequency is iterated over. Within the loop, the application would calculate induction parameter, exponential factor and intrinsic impedance. From these parameters, reflection coefficient will be computed, which will be used to determine apparent resistivity and phase. At this point the program will plot the results in the form of a graph of apparent resistivity vs. frequency & phase vs. frequency.

The algorithm for inversion, just as for forward modeling was developed from solutions formulated by ([Meju 1959](#_ENREF_9)).

# Chapter 3: Benchmark Testing

Benchmark testing can be described as the process of testing particular software or a component of a software package under measurable load, to determine the performance level of the component and/or the software ([Kanoun and Spainhower 2008](#_ENREF_7)). As benchmark tests are repeatable, it allows users to determine if changing code improves or reduces performance of the software.

The 1D inversion application was run on both a desktop computer as well as the Magnus supercomputer as a means of doing benchmark testing.

## 

## 3.1 Computation on the desktop

The application was run on a MacBook Pro with a 2.9GHz Intel Core i7 processor and 8GB memory. The application was allowed to complete 1500 iterations and the time to compute was recorded.

## 

## 3.2 Computation on the Cray XC30 Supercomputer

The application was run on the Magnus Cray XC40 Supercomputer. This computer has 12-core, 2.6GHz processors with 93 Terabytes of memory. Each node has 64GB of memory ([Magnus 2015](#_ENREF_8)). The application was again allowed to complete 1500 iterations on one node and the time to compute was recorded.

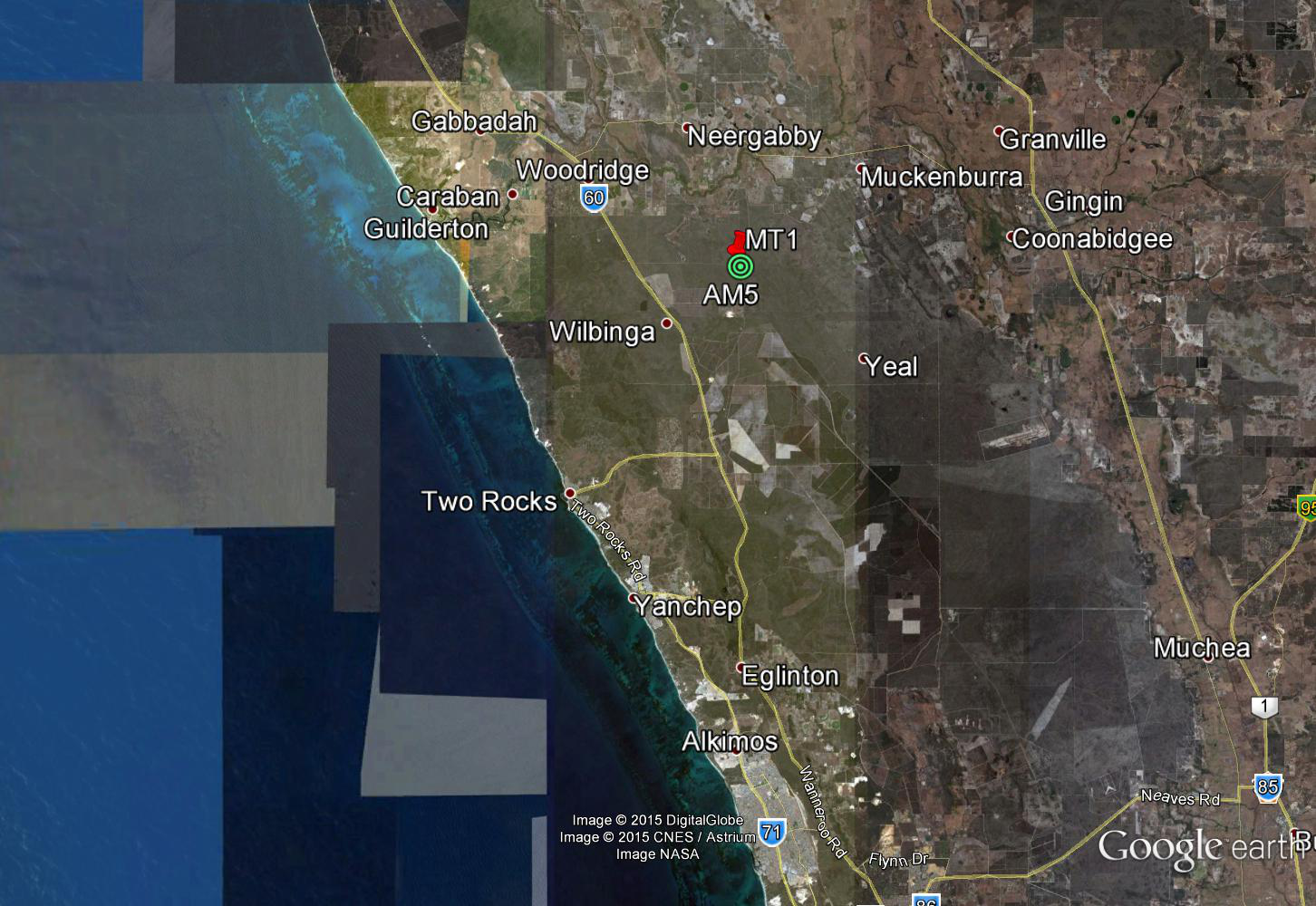
|  |  |  |  |
| --- | --- | --- | --- |
| Type | Num Iterations | Time in Sec | Time/Iteration |
| MacBook | 1500 | 352.4 | 0.234933333 |
| Magnus | 1500 | 167 | 0.111333333 |

### Figure 3-1: Compute time for 1500 Iterations

The results above give an indication of how much of a difference working on the cloud has over working on a desktop computer. The MacBook took approximately 5 minutes to compute 1500 iterations, whilst the Magnus supercomputer took on average 2 minutes.

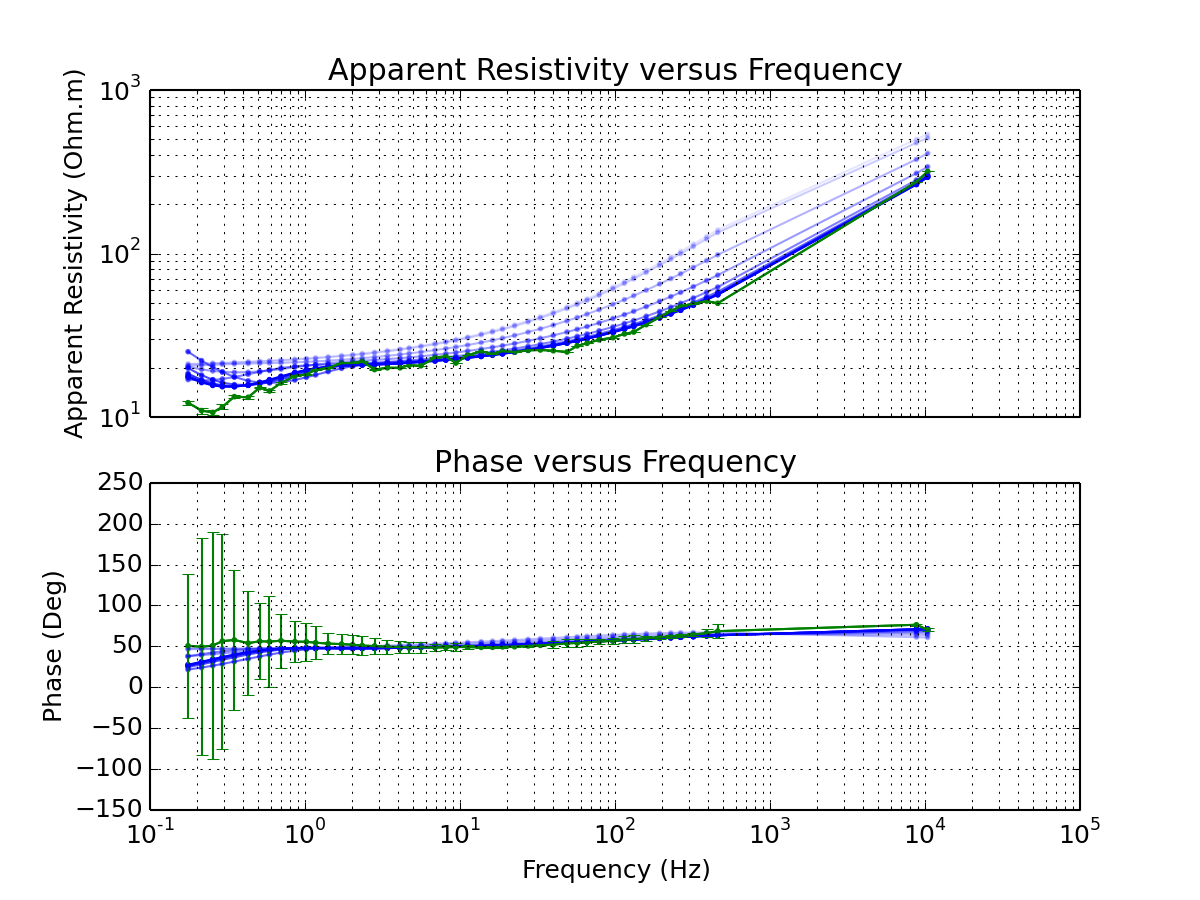
This means that the AusLamp survey that contains 2800 sites ([GeoscienceAustralia 2015](#_ENREF_6)) could theoretically be inverted in approximately 4 minutes. It is important to note that these computations were done unconstrained and the program was not optimized with Message Passing Interface (MPI). MPI assist the computer to perform parallel processing which speeds up compute times.

# Chapter 4: Case Study – Hydrological Investigations in Yanchep, Perth



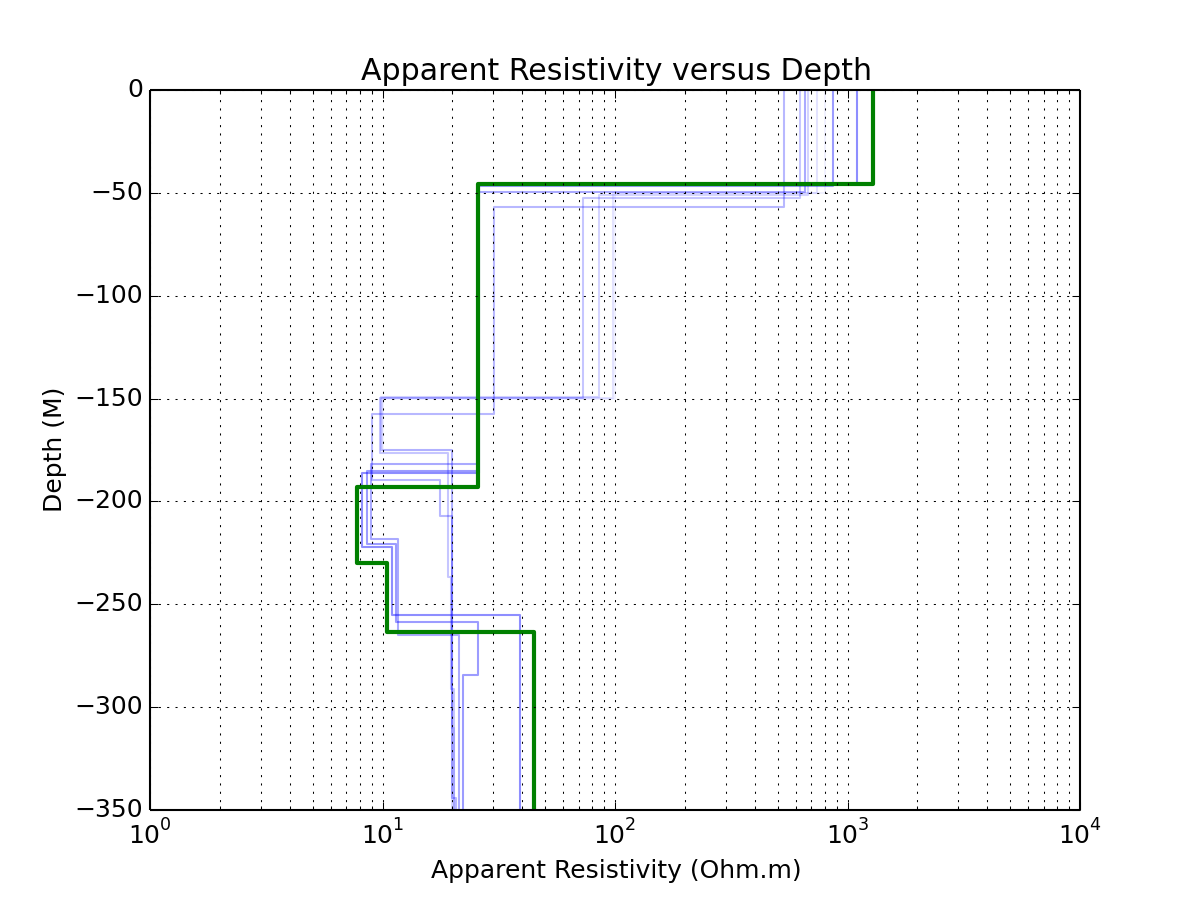
### Figure 4-1: Case Study location showing position of Well and MT Station

The application was tested using data acquired from the Yanchep area in Perth, WA. This area was selected as it contains well data that would be handy to compare final earth models.



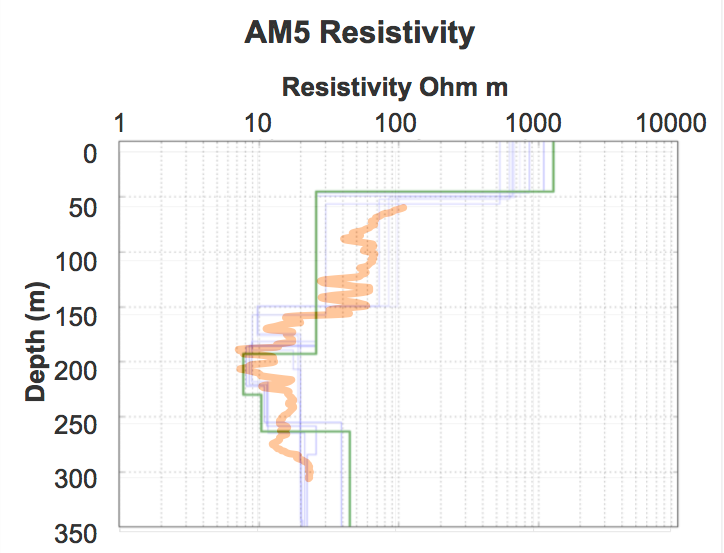
### Figure 4-2: Plot of Apparent Resistivity & Phase vs Frequency for well location

The figure above shows the result obtained when the data from the MT station was run on the web application. The application gives results in terms of apparent resistivity in (Ohm.m), Phase in (Degrees) and Frequency in (Hz). The lighter blue lines show “inversion tracking” which is a visualization technique native to this application. Inversion tracking is a plot of a model after each iteration is complete. It is a means of showing how far the program has come from the starting model.



### Figure 4-3: Final Earth Model for MT Station in Yanchep

The figure above shows the final earth model for the MT station after it was fun on the web application. This model is a representation of the subsurface of the area under the MT station. As stated previously, the model shows inversion tracking (in blue) and the final earth model in green.



### Figure 4-4: Final Earth Model & plot of well data for case study location

The figure above shows the final earth model overlain on the plot of the well data for the case study location, Yanchep. The well log (in orange) shows about 4 main resistivity interfaces. The MT web application is able to model these 4 transition zones as well. The final earth model from the MT web application is plotted in green and is able to show average locations of changes in resistivity values.

# Chapter 5: Discussion & Conclusion

There is a void in cloud based or even web based magnetotelluric applications. Currently these applications are written for desktop only. Implementing both cloud and web based technologies will help overcome this challenge.

The feasibility of a cloud based 1D magnetotelluric web application was tested. The structure (see chapter 2) and code (see appendix) necessary for implementation are presented. The application was tested using data acquired over an area of known electrical resistivity distribution.

We present a working 1D MT Inversion Web application.

The application provides:

1. Portability – works on any device that is HTML 5 compliant
2. Mobile Friendly – works on cell phones
3. Installation Free – no client installation is required
4. Security – Can be deployed on private servers
5. Rapid - Visual results in seconds
6. Multiple Users – Multiple users can concurrently invert data
7. New visualization techniques – Inversion Tracking
8. Custom models – Allows custom geo-electrical input starting models

Currently the application is housed locally on a private network and not distributed over the internet. This means that users must also be on the same network as the private server to access the application. The next logical step from this research is to set up a dedicated web server to run the application. This webserver would provide geophysicists access to rapid 1D magnetotelluric inversion. This industry and global exposure would improve our testing capability to find bugs and improve functionality. In short the cost factor associated with storing the application on a dedicated web server will be nothing compared to the amount of geophysicists who will be able to access and improve the application.

Since this 1D Magnetotelluric web application is the first known of it’s kind, the application can act as a platform for further development; namely 2D and 3D MT inversion web applications. One aspiration we had whilst building the application was that it would be further developed and enhanced with 2D and later 3D MT inversion capabilities. By completing the 1D version, we have created a good framework for future geophysicists who are interested in computer programming to work from.

In summary, we have created a 1D inversion web application that is ready for cloud deployment and industry alpha stage testing. We have also developed a magnetotelluric web framework that can be used for further development (2D and/or 3D inversion web applications.)

# Chapter 6: Executive Summary

This project has concluded with:

* A working 1D inversion web application ready for cloud deployment and industry alpha stage testing
* A magnetotelluric web framework for further development (2D, 3D inversion web applications)

# Chapter 7: References

Bailey, R. C. (1970). "Inversion of the geomagnetic induction problem." Proc. Roy. Soc, London **A. 315**: 185 - 194.

Cagniard, L. (1953). "Basic Theory of the Magneto-Telluric Method of Geophysical Prospecting."

Constable, S. C., R. L. Parker and C. G. Constable (1987). "Occam's inversion: A practical algorithm for generating smooth models from electromagnetic sounding data." Geophysics **52**(3): 289 - 300.

Fraser, R., J. Vote, R. Goh and S. Cox (2013). "Virtual Geophysics Laboratory ; Exploiting the Cloud and Empowering Geophysicsts." Geophysical Research Abstracts **15**.

Garcier, X. J. G. A. (2008). "Robust processing of magnetotelluric data in the AMT dead band using the continous wavelet transform." Geophysics **73.6**: F223-F234.

GeoscienceAustralia. (2015). "Geoscience Australia." from <http://www.ga.gov.au/>.

Kanoun, K. and L. Spainhower (2008). Dependability Benchmarking for Computer Systems. Hoboken, New Jersey, John Wiley & Sons, Inc.

Magnus. (2015). "Magnus." from <http://www.payser.org.au>.

Meju, A. M. (1959). Geophysical Data Analysis: Understanding Inverse Problem Theory and Practice.

Mezini, M. (1998). Variational Object-Oriented Programming beyond classes and Inheritance.

Nichols, E. A., H. F. Morrision and J. Clarke (1988). "Signals and noise in measurements of low-frequency geomagnetic fields." Journal of Geophysical Research **Solid Earth**(93.B11).

Niwas, S., P. K. Gupta and V. K. Gaur (2005). "Normalized impedance function and the straightforward inversion scheme for magnetotelluric data." Journal of Earth System Science **114**(5): 523-531.

Pedersen, J. and J. F. Hermance (1986). "Least Squares Inversion of One-Dimensional Magnetotelluric Data: An assessment of procedures employed by Brown University." Surveys in Geophysics **8**: 187-231.

Pethick, A. and B. Harris (2015). "1D Magnetotelluric Forward Modelling Web App." ASEG - PESA 2015.

PHP\_5\_Tutorial. (2015). "PHP 5 Tutorial." Retrieved July 12th, 2015, from <http://www.w3schools.com/php>.

Sheriff, E. R. (1991). Encyclopedic Dictionary of Exploration Geophysics, Society of Exploration Geophysics, 1991.

Simpson, F. and K. Bahr (2005). Practical Magnetotellurics. Cambridge, UK, Cambridge University Press.

# Chapter 8: Appendix

8.1 MT Global.py 30

8.2 MT Run.py 31

8.3 MT Inversion Parameters.py 32

8.4 MT Iteration.py 33

8.5 MT Invert.py 33

8.6 MT Earth.py 39

8.7 MT Data.py 41

8.8 Video of the MT 1Dnversion Web Application in use 43

## 8.1 MT Global.py

#Class that contains the global parameters of the program

-----------------------------------------------------------------

global verbosity #object that is declared as global

import time #imported library so that time can be assessed

verbosity = 3

#Parameter that dictates how much is printed to screen

dir = *'/Users/spearzo/Documents/Thesis/MT\_Benchmark/'*;

#Working directory pathway for the program

#this should be changed for each machine

current\_micro\_time = lambda: int(round(time.time() \* 1000000)); #current time in micro seconds

-----------------------------------------------------------------

## 8.2 MT Run.py

#Contains all the classes and libraries required to run program

-----------------------------------------------------------------

import matplotlib.pyplot as plt

import math

import cmath

import time

import mt1dfwd as mt

import mtdat as dat

import mtearth as earth;

import mtinvert as inv

import datetime

import time

import numpy as np

import mtglobal as mtglobal

import mtinvparameters as mtinvparam

#Lines that begin with ‘import’ contain either a class or library that is needed in the running of the program.

mtglobal.verbosity = 0; #assesses the parameter ‘verbosity’ from the class ‘mtglobal’ to determine how much to print to screen

if(mtglobal.verbosity >= 1) :

print(*'====================================='*);

print(*' 1DMT Inversion v1.0 '*);

print(*' Developed by '*);

print(*' Andrew Pethick '*);

print(*' and '*);

print(*' Stefan Spears '*);

print(*' July 2015 '*);

print(*'====================================='*);

print(*''*);

print(*'Starting Program'*);

#Prints the information above once the level of verbosity is greater than or equal to one.

###########SET GLOBAL PARAMETERS##############

mtglobal.dir = *'./tmp/'*; #Declares the working directory for the program

###########INITIALISE DATESTAMP##############

datestamp = datetime.datetime.fromtimestamp(time.time()).strftime(*'%Y%m%d%H%M%S'*)

##### INITIALISE EARTH MODEL##################

resistivities = np.array([10.0, 16.0,13.0,12.0,1.0,11.0,14.0,1.60,41.0,101.0]);

thicknesses = np.array([10.0, 15.0,30.0,50.0,70.0,100.0,150.0,160.0,190.0]);

startingEarth = earth.MTEarth(resistivities,thicknesses);

##### LOAD IN DATA ###########################

filename = *'example.data'*;

fieldData = dat.load(filename);

##### FORWARD MODEL TO GET STARTING RESULT####

syntheticData = mt.mt1dfwdalt(startingEarth,fieldData.frequencies);

inversion = inv.mtinversion(startingEarth,fieldData,mtinvparam);

inversion.invert()

#####################PLOT DATA########################

dpires = 150

inversion.exportAllResults(mtglobal.dir,datestamp,dpires)

if(mtglobal.verbosity >= 1) : print(str(mt.ntimes) + *' 1D forward models completed'*)

#Prints the number of forward models completed

print(str(mtglobal.dir) + str(datestamp) + *'\_inv\_dat.svg'*);

#Prints the name of the file that contains the result of the final forward model.

-----------------------------------------------------------------

## 8.3 MT Inversion Parameters.py

#Class that contains all the parameters required to run inversion of the given earth model

-----------------------------------------------------------------

maxIterations = 10; #Defines the maximum number of iterations

tolerance = 1.00; #Defines the amount of changes the program will allow

beta=0.01;#Damping factor

#beta controls the bias that is added to the model before inversion is done

pf = 3/100.0; #peturbation factor

#pf controls as a percentage the amount of changes that will be done to the starting model before inversion

armin = 0.0001; #Initializes the minimum apparent resistivity

armax = 10000; #Initializes the maximum apparent resistivity

#Values above are used to initialize their respective parameters#

-----------------------------------------------------------------

## 8.4 MT Iteration.py

#Class that represents all the data needed to define a single iteration of a 1D MT inversion. This class is the only input for running the next iteration.

-----------------------------------------------------------------

import mtdat as dat

import mtearth as earth

class **Iteration**:

earthModel = None; #This is the final earth model

syntheticData = None; #This is the final synthetic data post-inversion

iterationNumber = 1; #What iteration cycle does this represent (<maxIterations)

fieldData = dat.MTData(); #A link to the field data

-----------------------------------------------------------------

def **\_\_init\_\_**(*self*, iterationNumber, earthModel, syntheticData, fieldData):

*self*.earthModel = earthModel;

*self*.syntheticData = syntheticData;

*self*.iterationNumber = iterationNumber;

*self*.fieldData = fieldData;

#This is an object that takes into account iteration number, earth model, synthetic data and field data. It keeps track of respective parameters based on the iteration number#

-----------------------------------------------------------------

## 8.5 MT Invert.py

#Class that contains all the information required to do inversion of data.

-----------------------------------------------------------------

import math

import cmath

import time

import mt1dfwd as mt

import mtdat as dat

import mtearth as earth

import numpy as np

import copy

import mtiteration as mtit

import mtglobal

import matplotlib.pyplot as plt

import mtinvparameters as mtinvparams

#Lines that begin with ‘import’ contain either a class or library that is needed in the running of the program.

-----------------------------------------------------------------

def **invert**(*self*): #defining a class ‘invert’ which uses parameters from the class ‘self’.

if(mtglobal.verbosity >= 1) :

print(*'Starting inversion with the following parameters'*);

print(*'.Max Iterations= '* + str(*self*.maxIterations));

print(*'...Tolerance = '* + str(*self*.tolerance));

print(*'...Beta Smoothing = '* + str(*self*.bet));

print(*'...Peturbation Factor = '* + str(*self*.pf));

print(*'...Minimum Resistivity = '* + str(*self*.armin));

print(*'...Maximum Resistivity = '* + str(*self*.armax));

#program will print info according to the level of ‘verbosity’

*self*.iterations = [];

iteration = 1;

syntheticData = mt.mt1dfwd(*self*.startingEarth,*self*.startingData.frequencies);

startingIteration = mtit.Iteration(iteration, *self*.startingEarth, syntheticData,*self*.startingData)

*self*.iterations.append(startingIteration);

currentIteration = startingIteration;

while(not *self*.isFinished(*self*.iterations, *self*.maxIterations, *self*.tolerance)) :

*self*.iterations.append(*self*.computeNextIteration(*self*.iterations[len(*self*.iterations)-1]));

#program will do iterations until max iterations equals the number of iterations completed

def **isFinished**(*self*,iterations, maxIterations, tolerance) :

finalIteration = iterations[len(iterations)-1];

aresrel = finalIteration.fieldData.getApparentResistivityRelativePercentageError();

phaserel = finalIteration.fieldData.getPhaseRelativePercentageError();

if finalIteration.iterationNumber >= maxIterations :

return True;

elif finalIteration.fieldData.computeQError(finalIteration.syntheticData, aresrel,phaserel) < tolerance :

return True;

return False;

#once an iteration is completed the program will then compute error

def **computeNextIteration**(*self*,currentIteration):

#Program then calculates the next iteration keeping track of sensitivities via the jacobian matrix and the errors computed from the previous iteration.

if(mtglobal.verbosity >= 2) : print(*'Computing Iteration #%d'* % currentIteration.iterationNumber);

A = *self*.computeJacobian(currentIteration);

bet = *self*.bet

pf = *self*.pf

armax = *self*.armax

armin = *self*.armin

D=np.diag(np.power(np.diag(np.dot(A.T,A)),-0.5));

DA1 = np.dot(D,A.T);

DA2 = np.dot(A,D);

DA = np.dot(DA1,DA2);

#Check for infinite or NaN sensitivities

nanIndex = np.isnan(DA);

DA[nanIndex] = 0.0001;

infIndex = np.isinf(DA);

DA[infIndex] = 1000000;

M = 2\*currentIteration.earthModel.getNumberOfLayers() - 1; #Number of unknowns

beta = np.multiply(bet,np.eye(M)); #damping matrix

aerr = currentIteration.fieldData.getApparentResistivityRelativePercentageError();#calculates apparent resistivity error

perr = currentIteration.fieldData.getPhaseRelativePercentageError();

#calculates phase error

err = currentIteration.fieldData.computeErrorArray(currentIteration.syntheticData,aerr,perr); #computes an error array based on the current iteration, synthetic data, apparent resistivity error and phase error

pin = np.linalg.pinv(np.add(DA,beta));

dmg1 = np.dot(D,pin);

dmg2 = np.dot(np.dot(D,A.T), err);

dmg = np.dot(dmg1,dmg2);

isValid = True;

while(isValid) :

SCM = np.sqrt(np.divide(np.sum(np.power(dmg,2)),M));

maxx = np.max(np.abs(dmg));

if SCM > 1 or maxx > 3 :

dmg = dmg \* 0.9;

else :

isValid = False;

dmg = np.power(10,np.dot(pf,dmg));

####UPDATE EARTH MODEL

dResistivities = dmg[0:currentIteration.earthModel.getNumberOfLayers()];

dThicknesses = dmg[currentIteration.earthModel.getNumberOfLayers() : currentIteration.earthModel.getNumberOfLayers()\*2-1];

dResistivities = (dResistivities-1)\*20.00+1;

dThicknesses = (dThicknesses-1)\*20.00+1;

newResistivities = np.multiply(currentIteration.earthModel.resistivities,dResistivities);

newThicknesses = np.multiply(currentIteration.earthModel.thicknesses,dThicknesses);

#APPLY EARTH MODEL CONSTRAINTS

cr = currentIteration.earthModel.resistivities;

newResistivities = ((armax\*(cr - armin)\*dResistivities+armin\*(armax-cr))/((cr-armin)\*dResistivities+(armax-armin)))

newEarth = earth.MTEarth(newResistivities,newThicknesses);

iterationNumber = currentIteration.iterationNumber + 1;

syntheticData = mt.mt1dfwd(newEarth,currentIteration.fieldData.frequencies);

solvedIt = mtit.Iteration(iterationNumber, newEarth, syntheticData,currentIteration.fieldData)

return solvedIt

def **computeJacobian**(*self*,iteration):

start = mtglobal.current\_micro\_time();

nLayers = iteration.earthModel.getNumberOfLayers();

nFreqs = iteration.syntheticData.getNumberOfFrequencies();

b1 = np.zeros((nFreqs,nLayers));

b2 = np.zeros((nFreqs,nLayers));

b3 = np.zeros((nFreqs,nLayers-1));

b4 = np.zeros((nFreqs,nLayers-1));

tmpResArr = np.array(iteration.earthModel.resistivities);

tmpThickArr = np.array(iteration.earthModel.thicknesses);

tempEarth = earth.MTEarth(tmpResArr,tmpThickArr);

for li in range(0,nLayers) :

pf = *self*.pf;

oldRes = tempEarth.resistivities[li];

newRes = copy.copy(oldRes \* math.pow(10,pf));

tempEarth.resistivities[li] = newRes;

peturbed = mt.mt1dfwd(tempEarth,iteration.syntheticData.frequencies);

arf = peturbed.apparentResistivities;

ar = iteration.syntheticData.apparentResistivities;

phf = peturbed.phases;

ph = iteration.syntheticData.phases;

arerr = iteration.fieldData.getApparentResistivityRelativePercentageError();

pherr = iteration.fieldData.getPhaseRelativePercentageError();

b1[:,li] = np.divide(np.log10(np.divide(arf,ar)),arerr)/pf;

b2[:,li] = np.divide(np.subtract(phf,ph),pherr)/pf;

tempEarth.resistivities[li] = oldRes;

for li in range(0, nLayers-1) :

oldThick = iteration.earthModel.thicknesses[li];

newThick = oldThick\*math.pow(10,pf)

tempEarth.resistivities[0] = tempEarth.resistivities[0] + 0.01;

tempEarth.thicknesses[li] = newThick;

peturbed = mt.mt1dfwd(tempEarth,iteration.syntheticData.frequencies);

arf = peturbed.apparentResistivities;

ar = iteration.syntheticData.apparentResistivities;

phf = peturbed.phases;

ph = iteration.syntheticData.phases;

arerr = iteration.fieldData.getApparentResistivityRelativePercentageError();

pherr = iteration.fieldData.getPhaseRelativePercentageError();

b3[:,li] = np.divide(np.log10(np.divide(arf,ar)),arerr)/pf;

b4[:,li] = np.divide(np.subtract(phf,ph),pherr)/pf;

tempEarth.thicknesses[li] = oldThick;

tempEarth.resistivities[0] = tempEarth.resistivities[0] - 0.01;

a1 = np.hstack(([b1.T],[b3.T]))[0].T

a2 = np.hstack(([b2.T],[b4.T]))[0].T

A = np.vstack(([a1, a2]));

timeJA = mtglobal.current\_micro\_time() - start;

if(mtglobal.verbosity >= 3) : print(*"......Jacobian Time taken = "* + str(timeJA/1000000) + *"us"*)

return A;

def **exportAllResultsASCII**(*self*, dir, datestamp):

filenameSVG = dir + datestamp + *"\_all.txt"*;

#Results are exported as an ascii file

def **exportAllResults**(*self*, dir,datestamp, dpires):

filenameSVG = dir + datestamp + *"\_inv\_dat.svg"*;

filenamePNG = dir + datestamp + *"\_inv\_dat.png"*;

filenameEarthSVG = dir + datestamp + *"\_inv\_earth.svg"*;

filenameEarthPNG = dir + datestamp + *"\_inv\_earth.png"*;

#all other results are exported to the working directory

f, axarr = plt.subplots(2, sharex=True);

fearth, axarrearth = plt.subplots(1, sharex=True);

nth = 0.0;

iterations = *self*.iterations;

it = iterations[0]

for it in iterations:

#plot all of our data

nth = nth+1;

ap = nth/float(len(iterations));

axarr[0].set\_title(*'Apparent Resistivity versus Frequency'*);

axarr[0].set\_ylabel(*'Apparent Resistivity (Ohm.m)'*);

axarr[1].set\_title(*'Phase (Deg) versus Frequency (Hz)'*);

axarr[1].set\_ylabel(*'Phase (Deg)'*);

axarr[1].set\_xlabel(*'Frequency (Hz)'*);

#Apparent Resistivities

axarr[0].loglog(it.syntheticData.frequencies, it.syntheticData.apparentResistivities, color=*'b'*, alpha=ap);

axarr[0].scatter(it.syntheticData.frequencies, it.syntheticData.apparentResistivities,s=3, color=*'b'*, alpha=ap);

#Phases

axarr[1].semilogx(it.syntheticData.frequencies, it.syntheticData.phases, color=*'b'*, alpha=ap);

axarr[1].scatter(it.syntheticData.frequencies, it.syntheticData.phases,s=3, color=*'b'*, alpha=ap);

#thicknesses

axarr[0].grid(True,which=*"both"*);

axarr[1].grid(True,which=*"both"*);

axarrearth.grid(True,which=*"both"*);

#Plot our earth

axarrearth.set\_title(*'Apparent Resistivity versus Depth'*);

axarrearth.set\_xlabel(*'Apparent Resistivity (Ohm.m)'*);

axarrearth.set\_ylabel(*'Depth (M)'*);

#axarrearth.set\_autoscaley\_on(True);

axarrearth.set\_ylim([-2000,0]);

#START PLOTTING EARTH

layerResistivites = it.earthModel.resistivities

layerDepths = it.earthModel.getDepthsAndHalfspace(np.mean(it.earthModel.thicknesses));

finalResistivites = np.array([]);

finalDepths = np.array([]);

#start at the top

#each layer requires two points

depth = 0;

for i in range(0,len(layerResistivites)-1) :

finalResistivites = np.append(finalResistivites, layerResistivites[i]);

finalResistivites = np.append(finalResistivites, layerResistivites[i]);

finalDepths = np.append(finalDepths,depth);

depth = layerDepths[i];

finalDepths = np.append(finalDepths,depth);

axarrearth.semilogx(finalResistivites,finalDepths, color=*'b'*, alpha=ap/2);

#FINISH PLOTTING EARTH

axarr[0].set\_title(*'Apparent Resistivity versus Frequency'*);

axarr[1].set\_title(*'Phase versus Frequency'*);

axarrearth.set\_title(*'Apparent Resistivity versus Depth'*);

#Apparent Resistivities

axarr[0].loglog(it.fieldData.frequencies, it.fieldData.apparentResistivities, color=*'g'*);

axarr[0].errorbar(it.fieldData.frequencies, it.fieldData.apparentResistivities,it.fieldData.apparentResistivitiesError, color=*'g'*);

axarr[0].scatter(it.fieldData.frequencies, it.fieldData.apparentResistivities,s=3, color=*'g'*);

#Phases

axarr[1].semilogx(it.fieldData.frequencies, it.fieldData.phases, color=*'g'*);

axarr[1].errorbar(it.fieldData.frequencies, it.fieldData.phases,it.fieldData.phasesError, color=*'g'*);

axarr[1].scatter(it.fieldData.frequencies, it.fieldData.phases,s=3, color=*'g'*);

#START PLOTTING EARTH

layerResistivites = it.earthModel.resistivities

layerDepths = it.earthModel.getDepthsAndHalfspace(np.mean(it.earthModel.thicknesses));

finalResistivites = np.array([]);

finalDepths = np.array([]);

#start at the top

#each layer requires two points

depth = 0;

for i in range(0,len(layerResistivites)-1) :

finalResistivites = np.append(finalResistivites, layerResistivites[i]);

finalResistivites = np.append(finalResistivites, layerResistivites[i]);

finalDepths = np.append(finalDepths,depth);

depth = layerDepths[i];

finalDepths = np.append(finalDepths,depth);

axarrearth.semilogx(finalResistivites,finalDepths, color=*'g'*, alpha=ap, linewidth = 2.0);

#FINISH PLOTTING EARTH

axarr[0].grid(True,which=*"both"*);

axarr[1].grid(True,which=*"both"*);

axarrearth.grid(True,which=*"both"*);

f.savefig(filenameSVG, dpi=dpires);

f.savefig(filenamePNG, dpi=dpires);

fearth.savefig(filenameEarthSVG, dpi=dpires);

fearth.savefig(filenameEarthPNG, dpi=dpires);

if(mtglobal.verbosity >= 2) : print(*"Exported SVG to "* + filenameSVG);

if(mtglobal.verbosity >= 2) : print(*"Exported PNG to "* + filenamePNG);

if(mtglobal.verbosity >= 2) : print(*"Exported SVG to "* + filenameEarthSVG);

if(mtglobal.verbosity >= 2) : print(*"Exported PNG to "* + filenameEarthPNG);

#Results are then plotted and exported so that users can access them when needed.

-----------------------------------------------------------------

## 8.6 MT Earth.py

#Contains all the information needed to calculate and plot an earth model

-----------------------------------------------------------------

#Variables

resistivities = np.array([]);

thicknesses = np.array([]);

nlayers = -1;

def **\_\_init\_\_**(*self*, resistivities, thicknesses): #Passes resistivities and thicknesses from the array to be used in calculation#

*self*.resistivities = resistivities;

*self*.thicknesses = thicknesses;

def **getDepthsAndHalfspace**(*self*, halfspaceDistance):

depthsNoHalfspace = *self*.getDepths();

return np.append(depthsNoHalfspace,depthsNoHalfspace[len(depthsNoHalfspace)-1] - halfspaceDistance); #returns depth to final layer without halfspace distance#

def **getDepths**(*self*) : #gets individual depths of each layer#

depths = np.array([]);

currentDepth = 0

for i in range(0, *self*.getNumberOfLayers()-1) :

currentDepth = currentDepth - *self*.thicknesses[i];

depths = np.append(depths,currentDepth);

return depths;

def **getNumberOfLayers**(*self*) : #gets the number of layers present in the earth model#

if *self*.nlayers == -1 :

*self*.nlayers = len(*self*.resistivities);

return *self*.nlayers;

def **printEarth**(*self*) : #prints the earth using current depths, number of layers and the halfspace distance#

for i in range(0,*self*.getNumberOfLayers()) :

currentDepth = 0;

if(i != (*self*.getNumberOfLayers() - 1)) :

print(*"%f : %f Ohm.m (%f m)"* % (currentDepth,*self*.resistivities[i],*self*.thicknesses[i]));

currentDepth = currentDepth + *self*.thicknesses[i];

else :

print(*"%f : %f Ohm.m (halfspace)"* % (currentDepth,*self*.resistivities[i]));

def **plotEarth**(*self*) : #Plots the earth model so that we can view on screen#

f, axarr = plt.subplots(3, sharex=True);

axarr[0].set\_title(*'Apparent Resistivity (Ohm.m) versus Frequency (Hz)'*);

axarr[1].set\_title(*'Phase (Deg) versus Frequency (Hz)'*);

axarr[2].set\_title(*'Apparent Resistivity (Ohm.m) versus Depth (M)'*);

#Apparent Resistivities

axarr[0].loglog(it.fieldData.frequencies, it.fieldData.apparentResistivities, color=*'g'*);

axarr[0].errorbar(it.fieldData.frequencies, it.fieldData.apparentResistivities,it.fieldData.apparentResistivitiesError, color=*'g'*);

axarr[0].scatter(it.fieldData.frequencies, it.fieldData.apparentResistivities,s=3, color=*'g'*);

#Phases

axarr[1].semilogx(it.fieldData.frequencies, it.fieldData.phases, color=*'g'*);

axarr[1].errorbar(it.fieldData.frequencies, it.fieldData.phases,it.fieldData.phasesError, color=*'g'*);

axarr[1].scatter(it.fieldData.frequencies, it.fieldData.phases,s=3, color=*'g'*);

#Thicknesses

axarr[2].plot(it.fieldData.apparentResistivities, it.fieldData.currentDepth, color=*'g'*);

axarr[0].grid(True,which=*"both"*);

axarr[1].grid(True,which=*"both"*);

axarr[2].grid(True,which=*"both"*);

def **exportSVG**(*self*, filenameEarthSVG,dpires) :

#data exported as a SVG file#

print(*"Exported "* + filenameEarthSVG);

def **exportPNG**(*self*, filenameEarthPNG,dpires) :

#data exported as a PNG file#

print(*"Exported "* + filenameEarthPNG);

def **load**(filename): #reads in the file of the earth model - contains resistivities and thicknesses and returns the information to earth#

f = open(filename, *'r'*);

earth = MTEarth();

for line in f.readlines():

lineSplit = line.strip().split();

res = float(lineSplit[0]);

earth.resistivities.append(res);

if len(lineSplit) > 1 :

thick = float(lineSplit[1]);

earth.thicknesses.append(thick);

f.close()

return earth;

-----------------------------------------------------------------

## 8.7 MT Data.py

#This class contains all the information that is used/passed to other objects to be used in calculations#

-----------------------------------------------------------------

class **MTData**:

frequencies = np.array([]);

apparentResistivities = np.array([]);

phases = np.array([]);

apparentResistivitiesError = np.array([]);

phasesError = np.array([]);

#initialise on first compute

apparentResistivitiesErrorNorm = np.array([]);

phasesErrorNorm = np.array([]);

nfreq = -1;

#Creates the array necessary for the different parameters that users will have to enter for the program to do calculations

-----------------------------------------------------------------

def **\_\_init\_\_**(*self*): #defines what "self" does, which is creates an array of respective information to be used in that instance#

*self*.frequencies = np.array([]);

*self*.apparentResistivities = np.array([]);

*self*.apparentResistivitiesError = np.array([]);

*self*.phasesError = phasesError = np.array([]);

*self*.phases = np.array([]);

*self*.nfreq = len(*self*.frequencies);

*self*.currentDepth = np.array([]);

def **getNumberOfFrequencies**(*self*): #gets the frequencies and then calculates the number of frequencies #

return len(*self*.frequencies);

def **getApparentResistivityRelativePercentageError**(*self*): #returns the normalized error of the apparent resistivity#

if len(*self*.apparentResistivitiesErrorNorm) != 0 :

return *self*.apparentResistivitiesErrorNorm;

else :

*self*.apparentResistivitiesErrorNorm = *self*.apparentResistivitiesErrorNorm = np.log10(np.divide(np.add(*self*.apparentResistivities,*self*.apparentResistivitiesError),*self*.apparentResistivities));

return *self*.apparentResistivitiesErrorNorm;

def **getPhaseRelativePercentageError**(*self*): #returns the normalized error of phase#

if len(*self*.phasesErrorNorm) != 0 :

return *self*.phasesErrorNorm;

else :

*self*.phasesErrorNorm = *self*.phasesError;

return *self*.phasesErrorNorm;

def **computeRelativeApparentResistivityError**(*self*, data,fieldAResErr): #returns relative error of apparent resistivity#

aResRatio = np.log10(np.divide(*self*.apparentResistivities,data.apparentResistivities));

aResRel = np.divide(aResRatio,fieldAResErr);

return aResRel;

def **computeRelativePhaseError**(*self*,data, fieldPhaseErr): #return relative error of phase#

phaseDiff = np.subtract(*self*.phases, data.phases);

phaseRel = np.divide(phaseDiff,fieldPhaseErr);

return phaseRel;

def **computeErrorArray**(*self*, data,fieldAResErr,fieldPhaseErr): #returns an array of all the errors calculated#

aResRel = *self*.computeRelativeApparentResistivityError(data,fieldAResErr);

phaseRel = *self*.computeRelativePhaseError(data,fieldPhaseErr)

errorArray = np.concatenate([aResRel,phaseRel]); #append array

return errorArray;

def **computeQError**(*self*, data,fieldAResErr,fieldPhaseErr): #returns Q error

errorArray = *self*.computeErrorArray(data,fieldAResErr,fieldPhaseErr);

#print(errorArray)

q = math.sqrt(np.inner(errorArray,np.divide(errorArray,*self*.getNumberOfFrequencies()\*2)));

return q;

def **printData**(*self*):

#returns an output of the data contained in self#

print(len(*self*.frequencies));

for i in range(0,len(*self*.frequencies)) :

f = *self*.frequencies[i];

ares = *self*.apparentResistivities[i];

ares\_err = *self*.apparentResistivitiesError[i];

ph = *self*.phases[i];

ph\_err = *self*.phasesError[i];

output = *"%d : %f Hz rho=%f Ohm.m (%f) phase=%f Deg (%f)"* % (i, f, ares, ares\_err, ph, ph\_err);

print(output);

def **plotData**(*self*, filename): #Plots data#

f, axarr = plt.subplots(2, sharex=True);

axarr[0].set\_title(*'Apparent Resistivity (Ohm.m) versus Frequency (Hz)'*);

axarr[1].set\_title(*'Phase (Deg) versus Frequency (Hz)'*);

axarr[2].set\_title(*'Apparent Resistivity (Ohm.m) versus Depth (M)'*);

#Apparent Resistivities

axarr[0].loglog(*self*.frequencies, *self*.apparentResistivities, color=*'b'*);

axarr[0].errorbar(*self*.frequencies, *self*.apparentResistivities,*self*.apparentResistivitiesError, color=*'b'*);

axarr[0].scatter(*self*.frequencies, *self*.apparentResistivities,s=3, color=*'b'*);

#Phases

axarr[1].semilogx(*self*.frequencies, *self*.phases, color=*'b'*);

axarr[1].errorbar(*self*.frequencies, *self*.phases,*self*.phasesError, color=*'b'*);

axarr[1].scatter(*self*.frequencies, *self*.phases,s=3, color=*'b'*);

#Thicknesses

axarr[2].plot(it.fieldData.apparentResistivities, it.fieldData.currentDepth, color=*'g'*);

axarr[0].grid(True,which=*"both"*);

axarr[1].grid(True,which=*"both"*);

axarr[2].grid(True,which=*"both"*);

f.savefig(filename);

def **load**(filename): #reads in the file that has all the required information and returns it to dat#

f = open(filename, *'r'*);

dat = MTData();

for line in f.readlines():

line = line.strip().split();

freq = float(line[0]);

ares = float(line[1]);

ares\_err = float(line[2]);

phase = float(line[3]);

phase\_err = float(line[4]);

dat.frequencies = np.append(dat.frequencies, freq);

dat.apparentResistivities = np.append(dat.apparentResistivities, ares);

dat.apparentResistivitiesError = np.append(dat.apparentResistivitiesError, ares\_err);

dat.phases = np.append(dat.phases,phase);

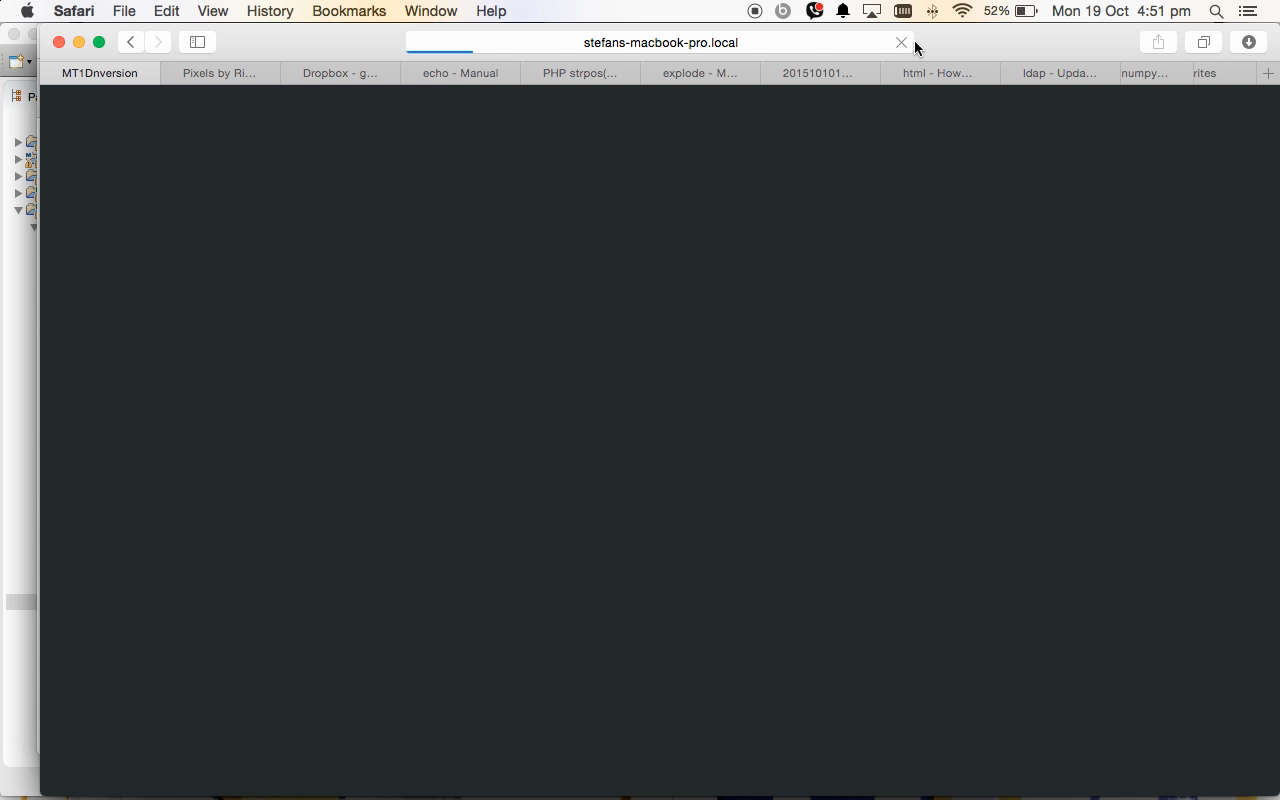
dat.phasesError = np.append(dat.phasesError,phase\_err);

f.close()

return dat;

-----------------------------------------------------------------

## 8.8 Video of the MT 1Dnversion Web Application in use



The video above shows a user accessing and utilizing the functional 1D inversion web application. The video shows correct data entry and subsequent results of inversion.