BTech 451 Final Report

PyjAmaseis

PyjAmaseis

An Application for Educational Seismology

Name: Saketh Vishnubhotla
UPI: svis267
ID: 2655131

31st October 2014

Acknowledgements

I would like to take this opportunity to express my deepest gratitude to my Industry Supervisor Dr. Kasper van Wijk for supporting me and giving me constant encouragement throughout the course of this project.

I also take this opportunity to express my sincere gratitude to my Academic Supervisor Dr. Patrice Delmas, who provided me with valuable advice on presentations and report writing throughout the year.

Lastly, I thank Matiu Carr from Science IT for offering his time to work with me to setup the remote functionality for this project.

Table of Contents

Abstract

Seismology is a topic hardly dealt with or taught in schools. This is due to several reasons ranging from not having a set curriculum to not having the tools in the classroom to demonstrate and simulate earthquakes. In an attempt to promote the teaching of Seismology in schools and to make it the best educational experience for the students, the Incorporated Research Institutes of Seismology (IRIS) and The University of Auckland's (UoA) Physics Department (RU) have invested into developing an education program called Seismographs in Schools in a bid to provide all the required components to teach Seismology in Schools. IRIS has created a curriculum involving classroom activities, quizzes, and learning content. These resources are freely available for teachers to access, allowing them to structure it into a subject. However there is more to what is being offered. The government has funded thousands of schools in USA to acquire a TC1 educational seismometer that can be used in conjunction with the curriculum provided by IRIS to make it a highly interactive course for students. In the same manner The UoA Physics Department has also provided many schools around NZ with computers and TC1 seismometers. However the software that is currently available is far too complicated for teachers and students to use. This project report will present a new software application developed in order to meet the needs of teachers and students for teaching and learning about seismology. This application features all the required core functionality developed within a simple and user-friendly user interface. The design, implementation of the application along with evaluation and future work are discussed in this report.

1. Introduction

Seismology is the study of earthquakes and seismic waves that move through an around the Earth. With the help of seismometers seismologists study the internal structure of the Earth. Although seismology is a very important modern day topic, schools have been unable to create a structured curriculum to teach seismology as a subject because there are several components required such as the necessary hardware and software to demonstrate what is taught in the curriculum. Therefore due to a lack of one or all the mentioned components (curriculum, hardware and software) many schools are unable to teach Seismology. To address this issue the Incorporated Research Institutions for Seismology (IRIS) have invested into creating a curriculum involving classroom activities, quizzes, and learning content. Along with this they have provided thousands of schools in the United States with educational seismometers to enhance the learning of the students. The University of Auckland's (UoA) Physics Department is also engaged in providing computers and TC1 educational seismometers to schools throughout New Zealand. This program is known as Seismographs in Schools. The aim is to provide the curriculum, hardware and software to schools so that they can formulate a structured course out of it and teach seismology as a subject.

The difficulty that both IRIS and UoA are facing is that although the curriculum is well developed and they are able to provide schools with educational seismometers, the currently available software that the teachers and students have to use is far too complicated for a classroom environment. The software is quite intricate and most of the complex features of the software are not used. There is a limited choice when it comes to Seismology data processing and sharing suites and the ones currently available aren't very suitable for teaching seismology. Hence the aim of this project is to develop an application that incorporates the core functionality present in the existing Seismology data processing and sharing suites along with laying heavy emphasis on a simple, intuitive and user-friendly design.

The goal of this project is to develop a robust cross platform application for educational seismology that provides the core features such as plotting live data from the seismometer, saving seismic data and sharing this data with other schools online. At the same time, whilst keeping the end users in mind, I will be looking into developing an interface that is suited for a learning environment.

This project report will discuss the design decisions, implementation, and a results section that highlights the significance of the solutions and features developed into the software.

2. Related Work

There are a number of libraries and modules available that let us work with seismic data but not too many applications. Among these most are limited to one platform and are all highly sophisticated. A few of the currently available Seismology Data processing and sharing suites are presented below.

2.1 Paradigm Seismic Processing and Imaging Solution

A professional seismic data processing and imaging solutions suite is offered by Paradigm. This software is highly professional and was built for professional seismologists. The Paradigm seismic processing and imaging solutions reduce uncertainty and improve reliability through better seismic signal quality, positioning, and content. Their proprietary algorithms translate billions of bits of seismic data into highly accurate, high-definition images of the subsurface, enabling geoscientists to visualize the earth's formations.

Although Paradigm seismic processing and imaging solutions provide high definition imaging tools and accurate seismic data plotting, it is not suited for a classroom scenario for teaching Seismology as it has been built for Professional Seismologists.

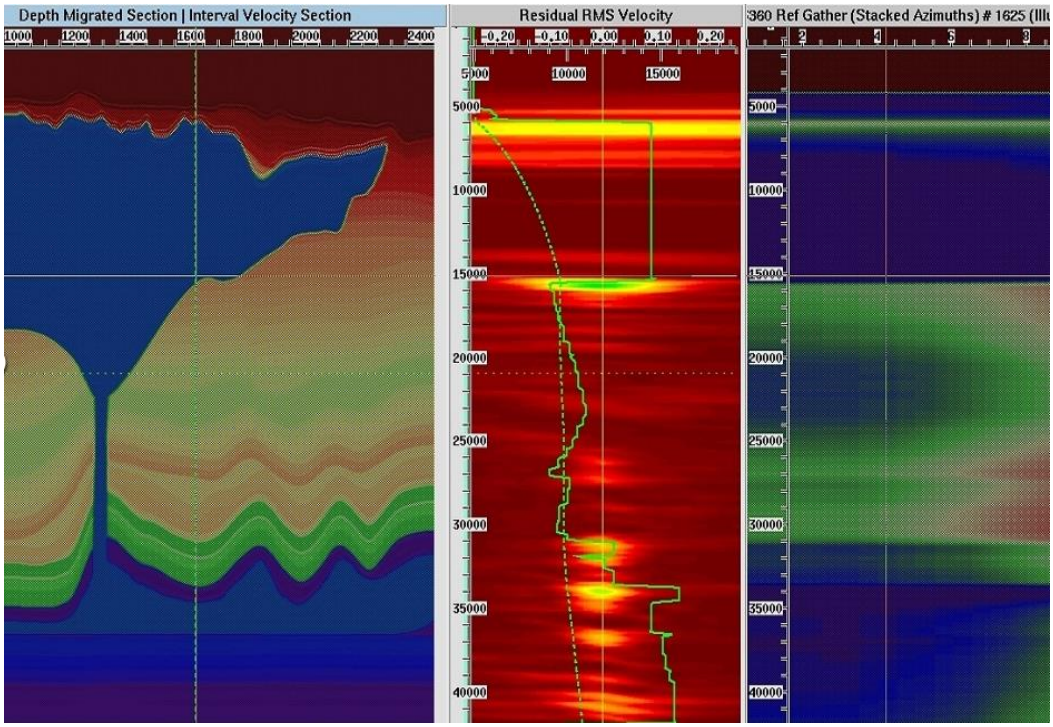

Figure 1 Paradigm software showing GeoDepth velocity determination, modeling and imaging

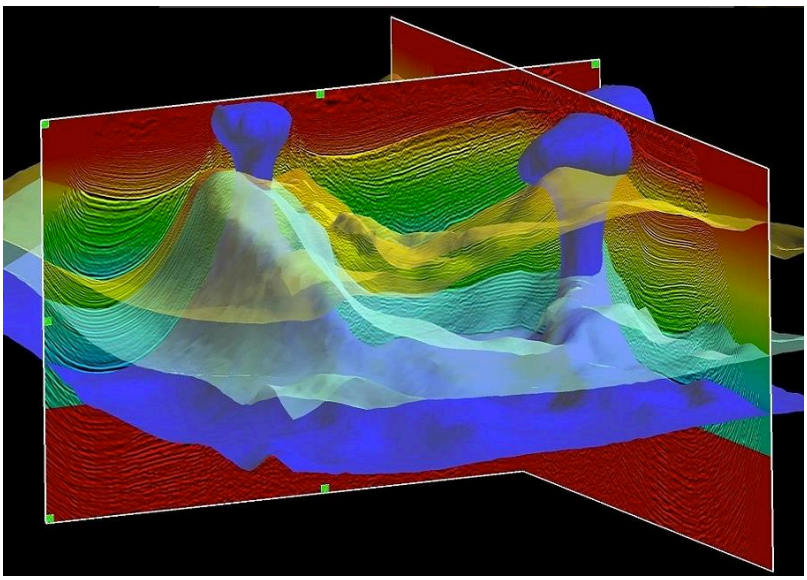

Figure 2 Paradigm software showing seismic velocity model with salt

2.2 The Omega Seismic Processing System

The Omega Seismic Processing system is a scalable system that allows for Seismic processing and imaging on a single workstation or clusters of computers. The Omega provides over 400+ geophysical algorithms for data manipulation. The Omega provides geophysicists support for Project management along with a workflow building application. The Omega SeisView geoscience and engineering software is a 2D canvas that gives geophysicists the tools to analyse and compare seismic data. The SeisView can store, display and plot trace attributes. While The Omega SeisView contains advance seismic data processing features and algorithms, it is not very suitable for teaching seismology in schools. The features are quite complicated and thus will be a challenge for teachers and students to use.

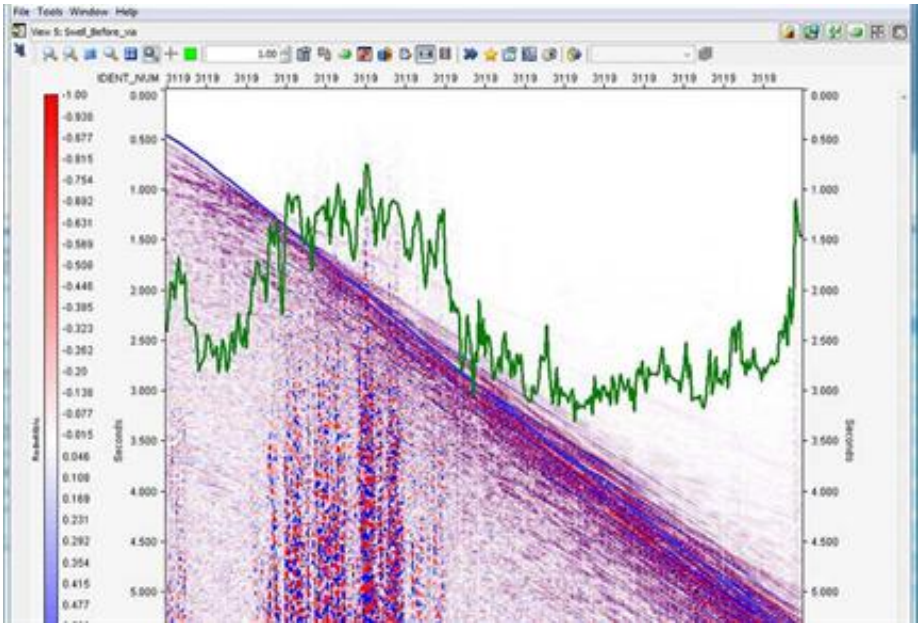

Figure 3 Omega SeisView 2D Canvas

2.3 jAmaseis

The currently used software in the Seismographs for schools program is known as jAmaseis. jAmaSeis facilitates the study of seismological concepts and allows users to obtain data in real-time from either a local instrument or from remote stations. As a result, users without an instrument can utilize the software. Users can view a graphical representation of seismic data in real time and can process the data to determine the characteristics of seismograms such as time of occurrence, distance from the epicentre to the station, magnitude, and location.

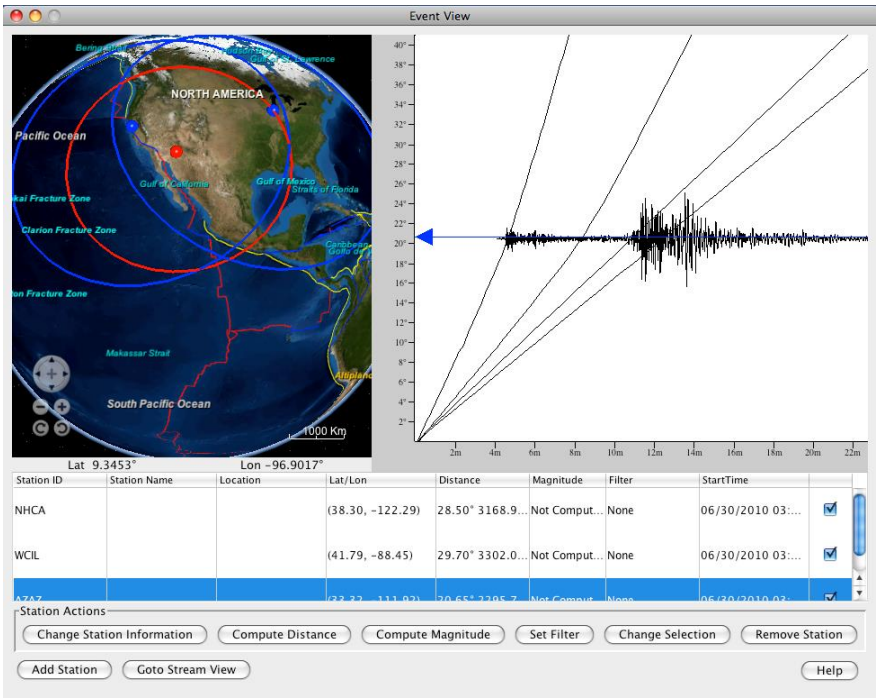

Figure 4 jAmaseis Event Model analysis

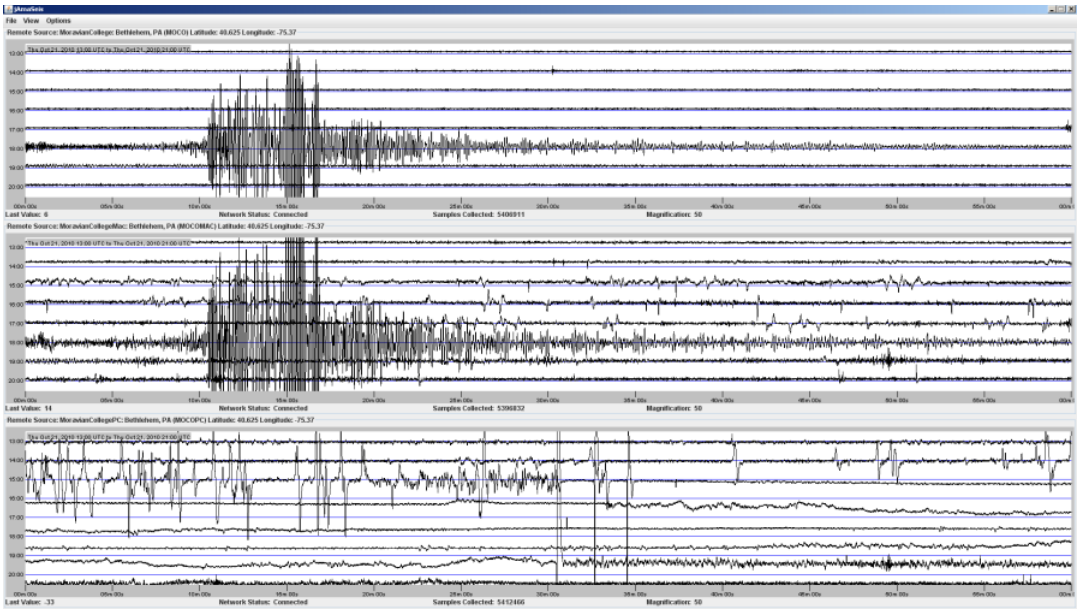

Figure 5 jAmaseis Stream View

jAmaseis contains all the basic functionality required in a seismology data processing suite to fit the Seismographs in Schools Program curriculum and goals, however it has been reported to be a struggle for students to start getting comfortable with using and learning from this software as the user interface is not easy to understand and not many learning cues are provided. The software provides the functionality but is not focused on being a learning tool and most of the intricate functionality also is not used.

2.4 Amaseis

Prior to the development of jAmaseis a developer named Alan Jones developed the first Seismology data processing and sharing suite known as Amaseis. This application recorded and monitored data from a seismometer known as the AS-1 (Amateur Seismometer). This software had the functionality to analyse seismic data such as mseed or SAC files downloaded from the Internet. The two major issues with this software was that it ran on Windows operating system only and after a decade of updating the software Alan stopped maintaining Amaseis. Although Amaseis is no longer being maintained, the functionality that was available has been developed into jAmaseis.

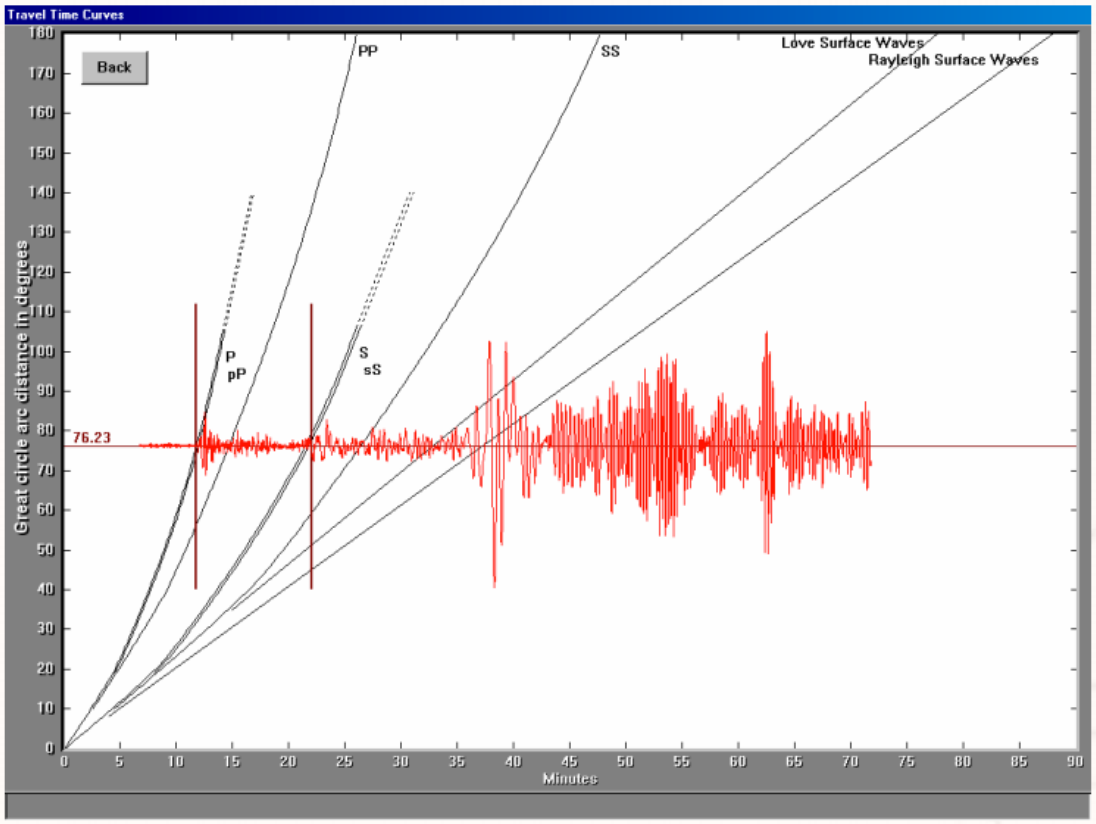

Figure 6 Amaseis Travel Time Canvas

3. Design

3.1 Project Requirements

The requirements for this project were given to me in the first meeting I had with my Industry Supervisor, Dr. Kasper van Wijk. The goal of this project was to create an application that was simple and intuitive, and develop all the core functionality present in the existing Seismology data processing and sharing suites such as Amaseis and jAmaseis. This core functionality consisted of data collection from the seismometer, displaying the collected data live, saving the data into seismology formatted files and finally being able to share this saved data online with other schools connected to the RU network. The application needed to be cross-platform so it may run on Linux, Mac OS, and Windows operating system.

Although there are several languages such as C++, C, Java, and Python that can be used to create a cross-platform application, Dr. Kasper insisted I developed the application in Python. The main reason behind this is because there is an open source Python framework available for Seismology called ObsPy. This framework provides parsers for common seismology file formats, clients to access data centres and seismological signal processing routines, which allow the manipulation of seismological time series. The goal of the ObsPy framework is to facilitate rapid application development for seismology.

Along with recommending the use of Python and ObsPy, Dr. Kasper also informed me in the first meeting that out of all the available Python Plotting libraries, Matplotlib is one of the most comprehensive 2D Plotting libraries available and requested that I use this library for the live plotting of the data collected from the TC1 seismometer.

Therefore the goals for this project can be summarized in the following points:

1. Develop a Python application that uses the ObsPy Framework and the Matplotlib Plotting Library
2. The application must provide the core features required to be a Seismology data processing and sharing suite. These consist of:
    a. Data Collection from TC1 seismometer
    b. Live plotting of data collected from TC1 seismometer
    c. Saving collected data into seismology specific file formats
    d. Sharing saved files on the NZSeis network
3. Application must be developed keeping the end users in mind, who are teachers and students. This requires the application to be:
    a. Simple and Intuitive
    b. Accurate
    c. Robust

d. User friendly

The aim of this project is to develop an application that incorporates all the requirements defined above.

3.2 Software Architecture

Prior to diving into the development of the application, I spent some time trying to understand how the overall structure of the application would look like, and plan how I would carry out the development of the core features. It is important to understand the software architecture in the initial phases of the project because the architecture once created will become harder to change in the future. Although modifying individual processes within the architecture can be done, the modifying of the overall structure will become difficult once development begins.

I began by visualizing the application to consist of the 5 core features - Data Collecting, Live Plotting, Saving, Sharing and the UI window. These individual components would need to talk to each other via some form of inter-process communication.

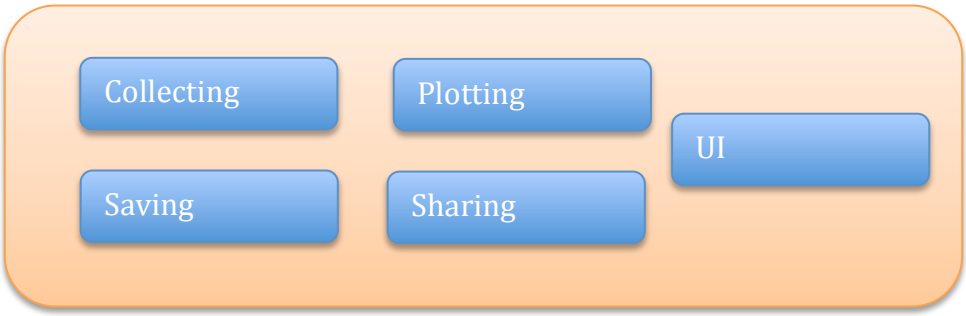

Figure 7 Component and connecters software architecture

In order to achieve this, I looked into a way to divide the application to run in five separate parts. This type of software architecture can be achieved by using the Multi-Threading or Multi-Processing modules to divide the functionality so that they ran separately on different threads or processes. Using inter-process/thread communication channels such as Pipes and Queues allows for communication between the various components. This sort of architecture is essential for this application, as we want the Plotting process/thread to run concurrently with the Collecting process/thread to allow real-time plotting of the collected data. A non-concurrent architecture would hinder the possibility of real-time concurrent collecting and plotting of the data, which is a key requirement for this application. With a concurrent component based architecture we can spawn another thread to manage the UI. This would be the ideal way of developing PyjAmaseis by giving each thread its own set of tasks to manage. This way the developers in the future would find it easier to debug or update specific parts of PyjAmaseis. This type of architecture is known as

component and connectors architecture where there is emphasis on the separation of concerns (SoC) based on the wide-ranging functionality provided by the application. Separation of concerns is a design principle for separating a computer program into distinct parts in such a way that each component carries out its own tasks but is a part of the whole system.

4. Implementation

Before I could begin I first setup the environment. I installed Python 2.7.8 along with the PySerial module. I had experience with using the Eclipse IDE, hence looked into a way to write, compile and run Python scripts on Eclipse. To achieve this I downloaded PyDev and followed a YouTube video (link) to setup the environment. PyDev is a Python IDE for Eclipse. Dr. Kasper offered me a TC-1 seismometer so that I could develop the project at home.

4.1 Collecting

I began the implementation of this project by focusing on one component of the architecture at a time starting with the Collecting component. However, because the Collecting component and Plotting component are interrelated I worked on them simultaneously but for the purpose of this report I would like to explain each component individually and will draw from other components where required.

4.1.1 Sampling Data

I started off by using a small Python script that was obtained from – http://eliaselectronics.com/plotting-serial-data-using-gnuplot-and-python/ that saved and printed the data coming in from the TC1 seismometer via the serial port. I used the *stream.readline()* method provided by the PySerial module to read each value from the seismometer. After collecting a fixed amount of samples the script then saves the recorded values into a .dat file (data.dat), which would then be plotted using an application called GNUPlot.

```python
#!/usr/bin/env python

# import the serial module so we can access the serial port
import serial

# set up serial port
serialPort0 = serial.Serial('COM3', 9600)

# open file object in write mode
dataFile = open('data.dat', 'w')

# get number of samples to take
# don't prompt user --> piped in from bash script
numberSamples = int(raw_input(""))

# get specified number of samples
for i in range(numberSamples):
  print i # output sample number to screen

  reading0 = serialPort0.readline()

  dataFile.write(str(i) + ' ' + str(reading0)) # write sample number and reading to the file

# close file object, good practice
dataFile.close()

# close serial port to free it for other applications
serialPort0.close()
```

Figure 8 Python script that reads data from TC1 and saves into .dat-formatted file

Figures 9 and 10 shows the data saved in a data.dat file, and an illustration of that data after being plotted using GNUPlot. Here we can see most values lie between 32000 and 33000.

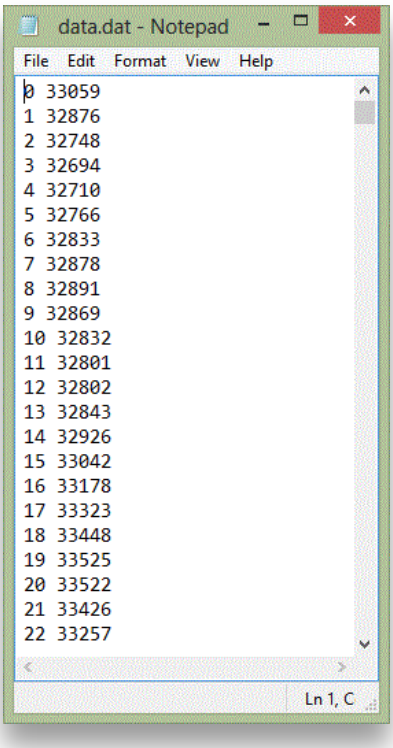

Figure 9 Data from TC1 saved in data.dat

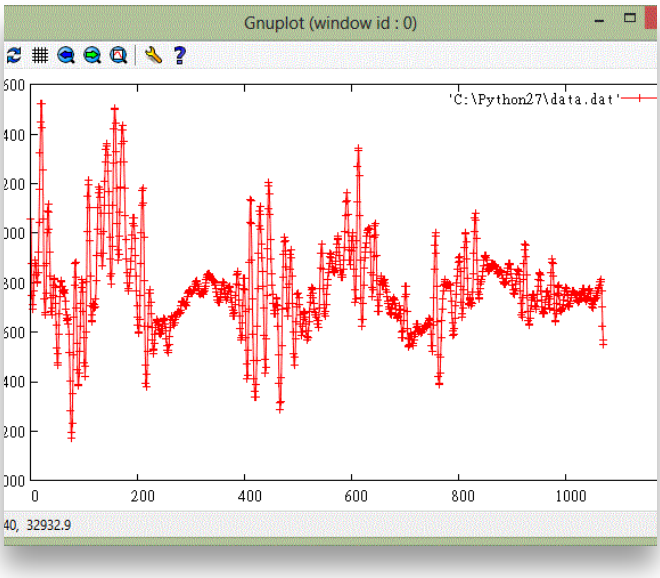

Figure 10 GNUPlot plotting data from data.dat

4.1.2 Continuous Collection

After sampling the values I shifted to a continuous form of reading the data from the TC1 seismometer by using an infinite while loop in Python and based on the time and other factors, carry out different tasks or send signals to other components of the application to change to the appropriate settings. For example when the hour changes the Collecting component sends a signal to the Plotting component to start plotting on a new line. The code sample below shows this behaviour, the *while True:* loop that continuously reads the values sent by the TC1 Seismometer over the serial port along with saving mseed files every hour.

```
try:
    while True:
        seismicData = np.array([ ])
        timeout_start = time.time()

        #run while loop for 1 hour
        while time.time() < timeout_start + timeout:
            reading0 = serialPort0.readline()
            linestr = reading0.decode('utf8')
            seismicData = np.insert(seismicData,0,linestr)

        #upon exiting while loop create a trace object and save the trace as an .mseed file
        trace = Trace(seismicData, None)
        trace.write('Mini-SEED-'+str(fileCounter)+'.mseed', format='MSEED')
        fileCounter = fileCounter + 1

        #creates a static plot of the trace
        trace.plot()
```

Figure 11 PyjAmaseis using an infinitely loop to continuously collect data from TC1 seismometer

4.2 Plotting

The plotting component of the application focuses on providing the user with a visual representation of the data that is collected by the TC1 seismometer. The aim of this component is to provide a live plot of the data that is to sub-second precision. In the field of seismology plotting data without any latency is of high importance. This will be further explained in the results section of this report.

4.2.1 Static Plotting

As covered in the Collecting component, I used a third party software named GNUPlot to plot a .dat file that contained samples of values collected from the TC1 seismometer. Instead of plotting the data using an external application, I looked into a way to plot the data as it was being collected. The first step I took was to create a static plot after every hour. These static plots plotted the values that were stored in an array which were used to save mseed or SAC files. These are seismology data file formats.

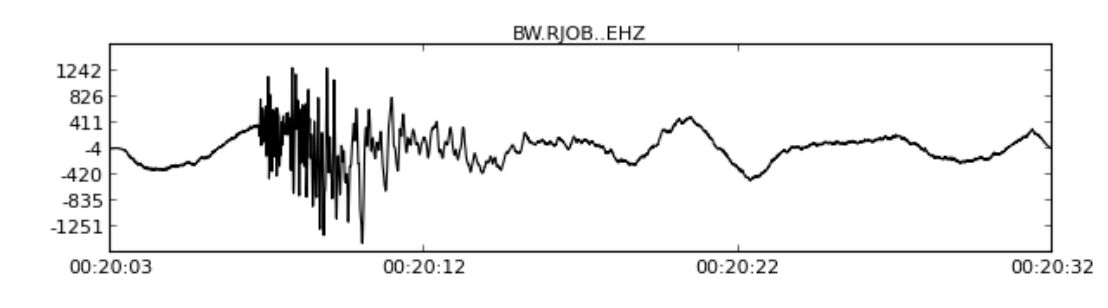

Figure 12 Plot of data displayed when trace.plot() method is called after saving an mseed file

4.2.2 Version Control Problem

I chose to implement a static plot as I was developing each component in versions, each version built upon the previous version. I faced a problem where I started having many individual Python scripts each with incremental changes and it became difficult to keep track of the latest version of the application. Therefore I started using Git version control system to manage my project. This effectively solved the problem of versioning and made it easier to create branches when I started working on a new functionality so that I always have a working copy of PyjAmaseis before I began any new implementation. Dr. Kasper had offered me to work on the project in his office where he had setup a station with a TC1 seismometer. Therefore any development that I did at home or at his office, I could push and pull accordingly with GitHub, which made it easier to manage the development of the application.

4.2.3 Dynamic Plotting

After developing a static plot feature in the application, I focused my attention on creating a dynamic plot that plotted data concurrently as it was being read from the TC1 seismometer. As part of the requirements I received from Dr. Kasper, Matplotlib was the plotting library for Python he requested me to use for the plotting functionality of this application. The reason behind this is not only because Matplotlib is a very comprehensive 2D plotting library but also Dr. Kasper has used it in previous projects, and there is a lot of support available for it on the web. Although Matplotlib provides support for plotting various types and styles of static graphs and plots when it comes to live plotting, the plotting functions within the library are highly inefficient.

The end goal of this Dynamic Plotting component is contained in the following points. Dr. Kasper provided specifications once I started working on the Plotting component of PyjAmaseis. These requirements were:

1. Display a 24 Hour plot
2. Show 0-60 minutes on the x axis
3. Show Current UTC Hour on the Top of the Y axis and increment hour as it comes down towards 0
4. Clear Plot at the end of 24 hours and restart plotting again from the top

I began the live plotting functionality by simply plotting an array that contained all the values that were read from the TC1 seismometer. All the values that were read were constantly being appended to the array. The existing plot would be cleared every time and array containing old and new values were redrawn over and over again.

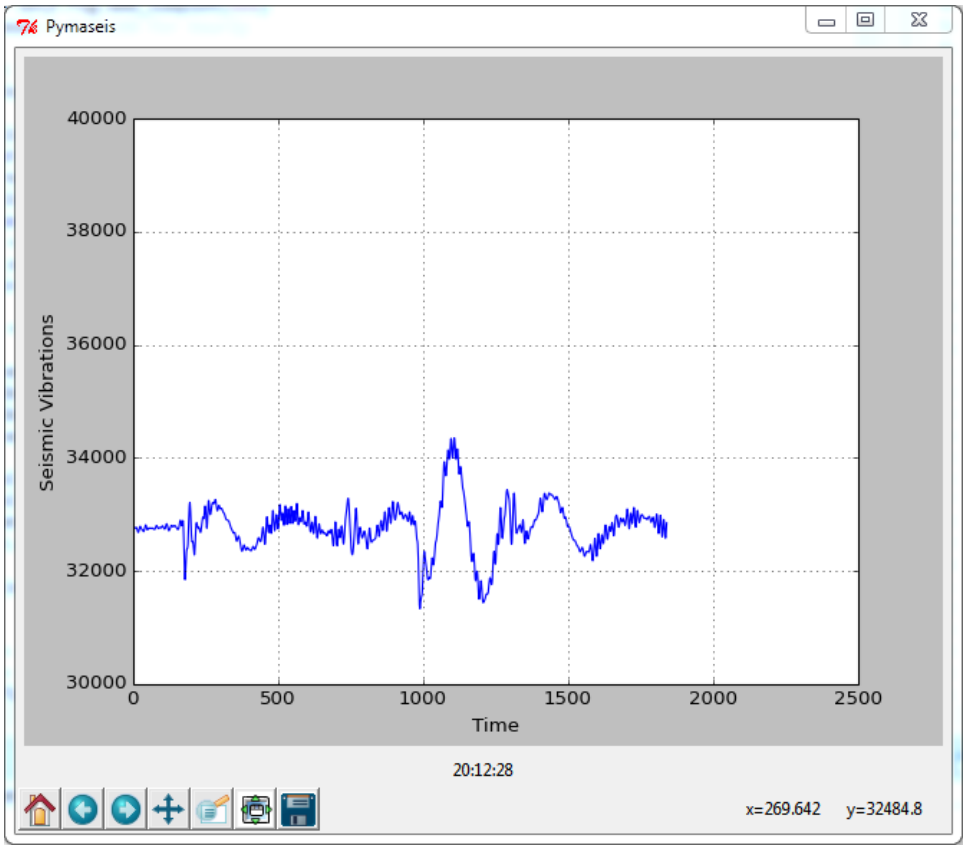

Figure 13 PyjAmaseis first attempt at live plotting

This form of plotting kept re-plotting an array that contained new values every 10 milliseconds. This proved to be highly inefficient however I did not notice the implications of this inefficiency until I ran the application for more than 5-7 minutes. Because of the current axis, the plot would only show the first 2 or 3 minutes of data hence the inefficiency was hardly noticeable but after I worked on setting the right x and y axis this became a major issue. I started noticing a delay between the time I physically shook the seismometer and when the application plotted this incident. Therefore instead of sending the complete array with all the values, I sent 10 values at a time and instead of clearing the

plot I tried to plot the 10 new values over the currently existing plot. However this did not solve the latency issues. I will discuss this in further detail after explaining how I set the correct x and y-axis for the plot.

4.2.4 Calibrating the axis

Setting the x-axis of the figure was relatively simple. Using the set xticks method that was provided in the Matplotlib module, I set the x-axis to go from 0 – 60 with an interval of 1. This can be seen in the figure 14. When plotting I send the plot function 2 arrays – x and y. The y values are provided by the seismometer at a rate of 18 values per second. Initially I was sending the y array with values received from the TC1 seismometer and the x array, which had incremental values starting from 0.1 onwards. This was an erroneous way of plotting, as the y values did not have a representative corresponding minute x value to link with. Therefore when a value is read from the input buffer of the serial port (i.e. sent by TC1 seismometer), the exact minute, second and millisecond it was read at would be recorded in the x array. This allowed each y value to have an accurate x value and thus accuracy of plotting was achieved.

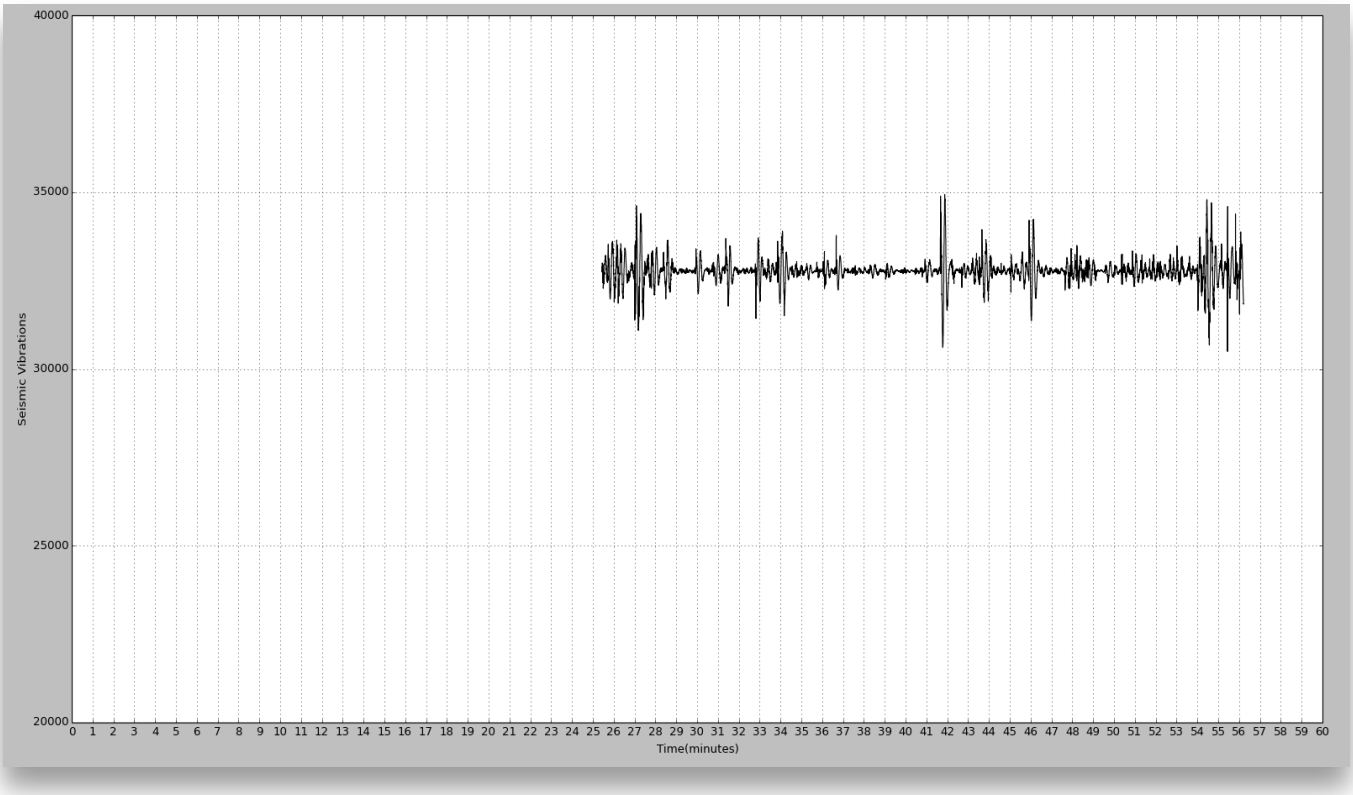

Figure 14 PyjAmaseis x-axis labels (0-60)

Setting the y-axis labels was a little trickier. The aim here was to have 24 plots in 1 plot. Matplotlib allows us to create sub-plots within 1 figure. That means you can create multiple separate plots within one window. Because I needed to

display 24 plots, plot of a whole day, I tried to create multiple subplots. When I created 24 sub plots the outcome was not pleasing. The data plotted couldn't be clearly seen, as the plots were too small and too close to each other, and even the axis labels were hard to understand. Hence I decided to move away from making multiple subplots, and used only 1 subplot effectively by translating the values according to the correct hour. This proved to be more effective and helped create a user interface that wasn't cluttered and allowed the users to see the plotting much more clearly.

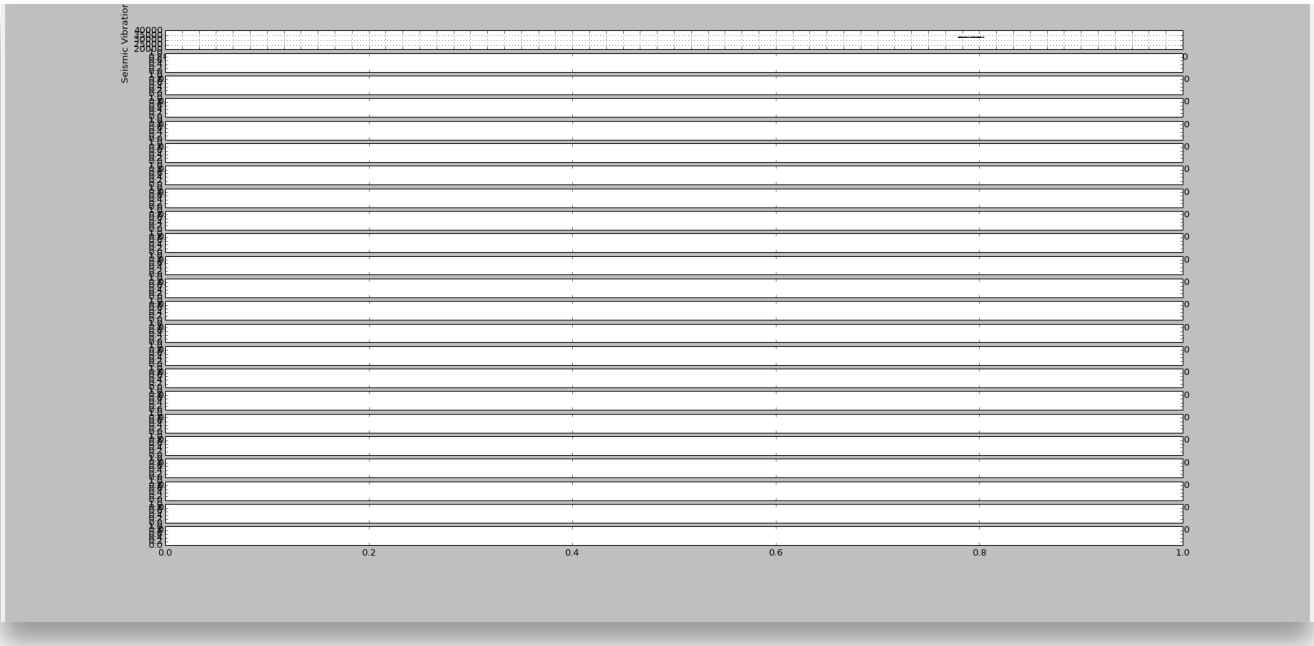

Figure 9 Using add_sublots() to create 24 subplots in one figure

For setting the y-axis labels to show 24 hours starting from the current hour, I first wrote a method that calculates the number that occurs the most in an array, and used the method to calculate the value TC1 provides when it is at rest. That means, when the seismometer is not disturbed by any seismic activity, it rests on or near a certain value and my aim was to find out what that was. After several tests I found that the mode number was approximately 32750. With this information I understood that the values provided by the TC1 seismometer ranged between the values of 30750 and 34750 with 32750 being at the center of oscillation. Thus I multiplied the range, which is 4000 (34750-30750) by 24 to create an axis that could fit 24 separate plots. Therefore the y-axis then ranged from 30750 to 130750. Depending on the hour the values from the TC1 seismometer are translated by a constant to plot according to the appropriate hour. Once I managed to get 24 plots to show on 1 subplot, I looked into getting the ytick labels. This was required because until now the y-axis labels were not informative as it was showing a range of values from 30750 to 130750. They have to display hours so using the datetime module I calculated the current hour and wrote a method that used this information to generate an array containing the next 24 hours along with providing their appropriate am/pm information.

The calculated times for the labels are in UTC time as per requirement. Thus the end result of calibrating the x and y-axis can be seen in figure 16 where the x-axis ranges from 0 to 60 to represent minutes and the y axis shows the hour information starting at 8 AM at the top and goes through 24 hours as it reaches the bottom. Note when the plot reaches the end of the hour it is shifted down by an hour and starts plotting again and after completing 24 hours of plotting the plot is cleared and the plotting begins again at the top.

I would like to briefly mention here how the graph plots without breaks. Basically every array that is sent to the plotting component of the application contains a new set of values that were read from the TC1 seismometer and in order to link the previously drawn plot to the current set of values, I store the last value of the previous array and insert it to the beginning of the new array of values. This way the new set of values always begin where the previous plot completed and this ensures continuity. However, when a change in hour is detected, the new array with the set of values from the new hour will not connect to the last value from the previous hour. Initially I faced a problem where the line would cut across the graph from the right edge where the plotting finished for the previous hour to the left edge where the plotting of the new hour began, but by placing a simple hour check I prevent the two plots from being connected. Therefore as long as the hour is the same, the last value of the previous array will be inserted to the new array of values, and when an hour change is detected this will not take place.

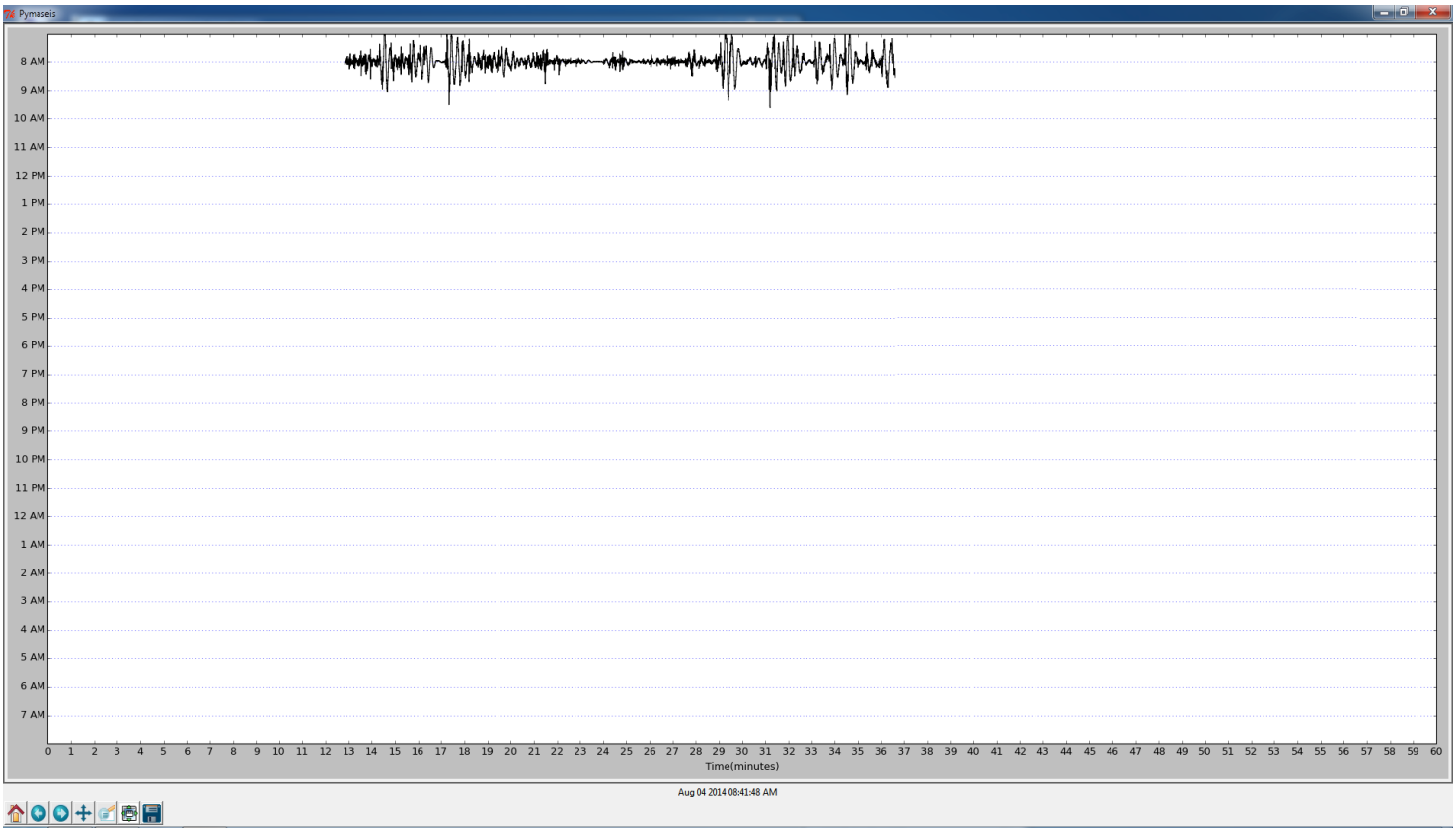

Figure 16 PyjAmaseis y-axis labels show hours in 12-hour format UTC time

4.2.5 Plotting Problems

I briefly mentioned earlier that the Matplotlib plotting function is quite inefficient. The problems I faced due to this inefficiency, the approaches I took and the solutions to the problems will be explained here in detail.

After setting the x and y-axis labels, I set the application to a test to see if it can plot 24 hours of continuous data and refresh and start again. However within 15 minutes the graph starts distorting and doesn't return back to normal. This can be seen in figure 17.

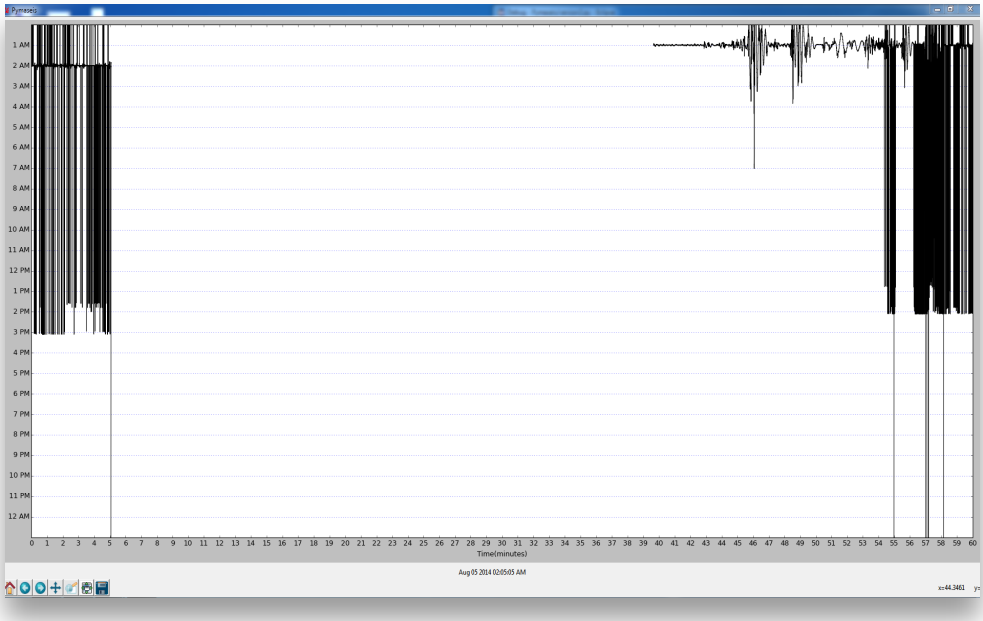

Figure 10 PyjAmaseis distorted plot

To understand this behavior I printed the y array before plotting to see if it contained any odd or erroneous values. The problem here was I was receiving values from the TC1 seismometer that when plotted resulted in plots as shown in figure 17. For example, figure 18 shows that in an array of values ranging from 87000 to 88000, an unexpected 919 value gets saved in the array. This causes the plot to draw a line from the previous value to the 919 value on the y-axis. This results in distorted plots. Initially I thought this could have been a hardware problem, however after thorough inspection of the errors printed in the output console I realized that I have no need to decode the values that are read from the TC1 seismometer. I was receiving errors such as "ValueError: invalid literal for int() with base 10: '"" and "TypeError: Can't convert 'str' object to int implicitly". When a value is placed in the serial port input buffer it contains the value and a new line character for example "32700\r\n" when I decode it "*decode = readValuefromTC1.decode ('utf8')"* the resulting decoded string is "32700rn" therefore it cannot be converted in to an integer and hence a lot of errors were

thrown due to this small mistake. I realized after that there was no need to decode the values and just by casting the read string into an int I would receive the value I need for plotting (*plottingValue = int(serialPort.readline()))*.

Although the distortion of the plots stopped, there was a delay that was noticed from when a simulated seismic activity occurred and when the application actually plotted it live. This delay would become noticeable after about 5-7 minutes and it kept increasing. A noticeable delay of 3-7 seconds increased into a delay of several minutes as the plotting went on.

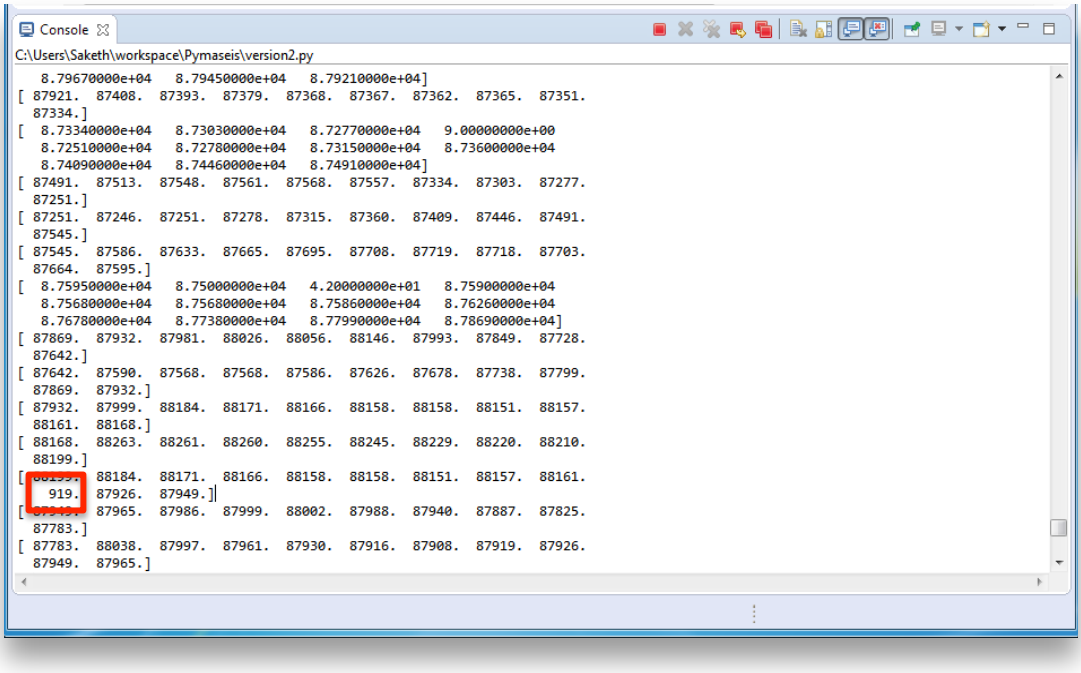

Figure 18 Cause of distorted plots found to be incorrect value in plotting array

I was under the impression that the delay was caused due to the fact that I was sending the whole array to be replotted each time but even after, when I sent only the 10 recent values collected from the TC1 to the plotting component – the delay still persisted.

Up till this point I was using the threading module for Python to create a multi-threaded application, one thread for collecting and one thread for plotting and was wondering if it was due to the Global interpreter lock (GIL) in Python which prevents multiple threads from running concurrently that delays were being caused. I also realized that because the two threads can not run concurrently we can never get true parallelism and one thread would always lag behind the other as their instructions are executed one after the other. At this point I decided to invest my time to learn about multiprocessing in Python and re-engineer the architecture if required.

The threading module uses threads whereas the multiprocessing module uses processes. The difference is that threads run in the same memory space, while processes have separate memory. This makes it a bit harder to share objects between processes with multiprocessing. Since threads use the same memory, precautions have to be taken or two threads will write to the same memory at the same time. This is what the Global Interpreter Lock is for.

Therefore upon realizing that threads will not allow for true parallelism I looked into turning PyjAmaseis into a multiprocessing architecture in hopes that the application would become more efficient and the delay problems would be addressed. I rewrote PyjAmaseis into a multi-processing application and used Pipes to send data from one process to another. The process of sending data through a pipe is known as Pickling (at the sending end) and Unpickling (at the receiving end). Although at first the re-engineering of the architecture seemed like it solved the delay problem, but to my dismay, it still persisted and went unnoticed for a longer period of time. The application was running more efficiently now for sure, however, the problem still persisted. The delay couldn't be noticed till the application ran for 3 hours but soon after; the delay was noticeable and kept on increasing.

Upon discussions with Dr. Robert Sheehan at the Computer Science Dept. regarding this bizarre behaviour and delay problems, he recommended I conduct some timing efficiency tests to see what component, or methods inside a component was causing this problem. I found that the easiest way to do this without having to spend a lot of time creating tests cases was to use the time module to create two timestamps before and after certain method calls and operations within the collecting and plotting components. After the block of code I'm testing for is run, the time recorded after its completion is subtracted from the time that was recorded before it started. This is demonstrated in the code sample in figure 19.

```
startTime = time.time()
plot. draw(x, y, color='k')
endTime = time.time()

print(endTime - startTime)
```

Figure 19 Computing the time taken to execute the plot.draw method

This provided me with a microsecond value of how long each section of the code ran for. To my surprise I noticed that the *plot.draw()* method in the plotting component of the application took 0.39 seconds to begin with and kept increasing to much larger time intervals as time progressed. What I discovered was that although I was only sending the most recent 10 values that were read from the TC1 seismometer, when these values are sent for plotting, all the points that were plotted previously were stored and re-plotted along with the new

points. Due to the draw method being so inefficient this caused the plotting component to slow down as the number of points it had to redraw kept increasing.

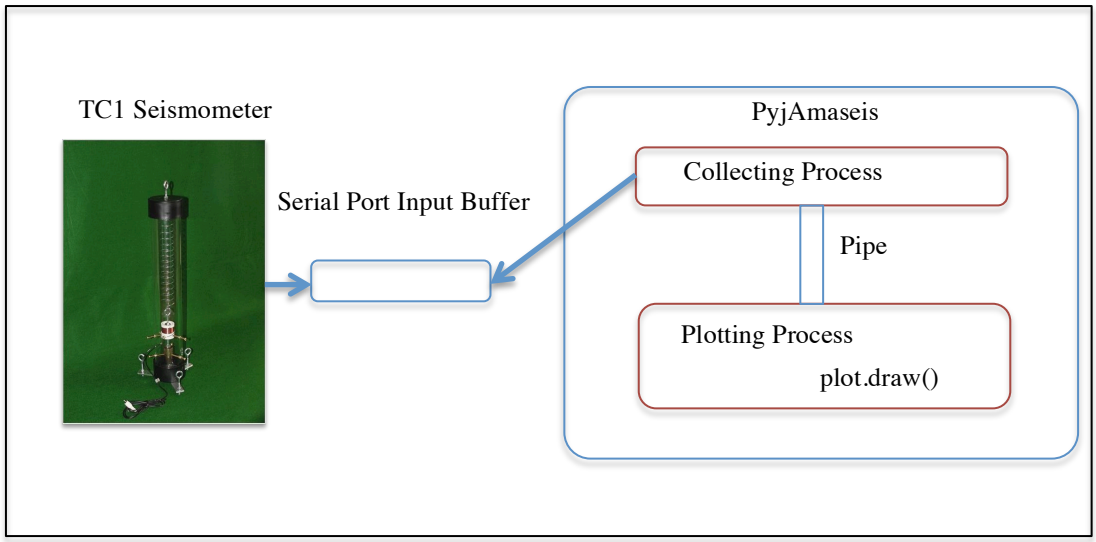

Figure 20 shows the how data is sent from TC1, collected by Collecting Process and sent Plotting Process via a Pipe

I will now provide specific information about the nature of the problem, and explain how I came to address this issue with the use of a very fast rendering technique known as Blitting. In my timing efficiency tests I also recorded several other key information that led me to understand how the delay caused by this one method affected the whole program. Below is a diagram depicting the structure of PyjAmaseis and how it is connected to TC1.

The following statistics were calculated and were crosschecked with Dr. Kasper to confirm that they aligned to the specifications. The TC1 seismometer pushes values into the serial port input buffer at the rate of 18.76 values per second. I could determine the size of the input buffer by using the *serialPort.inWaiting()* method which returns the number of characters currently in the input buffer. The input buffer can hold no more than 12290 characters, which means no more than 1755 values sent from the TC1 at any given time. After the input buffer is full the TC1 seismometer is unable to push any further values into the buffer at which stage any excess values are dropped until space is created in the buffer.

The collecting process is quite efficient, as it doesn't have too many instructions to process, unlike the plotting process that currently handles both the plotting, and managing the live plot display window. However as we've come to see, the only method that is causing all the problem is the *plot.draw()* method in the plotting component and not any others.

The pipe that connects the two processes can hold a maximum of 682 values at once. So once the nature of this system was understood, it made it easier to unravel how the whole application was effected due to the inefficient draw method. The following bullet points try to convey this effect on the application in a concise manner:

- Due to the increasing delay caused by the draw method that sat within the plotting process, the whole process became slower and slower as the time went on.
- Due to the plotting process taking more time, the rate at which it reads values from the Pipe decreased causing values in the Pipe to accumulate.
- This in turn caused the pipe to become full and remain full for longer periods of time until the slow plotting process read from the pipe to create space for more values to be sent from the collecting process.
- I learnt that the pipe method calls are blocking method calls meaning if you try to push a value on to a pipe that is full, the process would wait there without executing any other code until there is space on the pipe to push the value, and likewise if a process reads from an empty pipe it will get stuck in that method call until it receives a value from the pipe. This causes the processes to block further execution whenever these two conditions take place.

Action	Pipe Condition	Result
Process tries to read from pipe	Pipe empty	Process waits till it receives a value
Process tries to push value onto pipe	Pipe full	Process wait till there is space on pipe to push value

- Due to this nature of a Pipe, when the collecting process fills up the pipe faster than the plotting process reads them, the collecting process waits and blocks till it can push more values on to the pipe.
- This wait that the collecting process does, causes the serial port input buffer to fill up as more and more values are being placed in it while they are not being read fast enough by the collecting process causing a bottleneck situation in the application.
- Thus the delay starts of small but gets worse and worse and the rate at which the plotting component reads the pipe decreases.

Thus after understanding the whole situation, I looked into a way of replacing Matplotlib's draw method with something that was more efficient. Upon doing a bit of research I came across a rendering technique generally used in gaming known as Blitting. To "blit" is to copy bits from one part of a computer's graphical memory to another part. This technique deals directly with the pixels of an image, and draws them directly to the screen, which makes it a very fast rendering technique that's often used in fast-paced 2D action games (gaming). Therefore I tried and successfully implemented the use of Blitting in PyjAmaseis where after plotting an array of values I blit the figure and draw the new values on top of the currently blitted figure and repeat the process. Thus the plot is only drawing the values sent to it and the old values do not exist in any array, but they do as a blitted image of the figure. To my astonishment, with the help of blitting the rate at which the figure draws the new values and blits the figure together come to about 0.032 seconds which is roughly 12.2 times faster than what the plotting process took earlier to achieve the same result. This value was again calculated using the timestamp method explained earlier. Thus I successfully completed the plotting component of the application. The sub-second precision with which the application now plots has seismological significance and will be further explained in the results section. Upon receiving a request from Dr. Kasper, I reverted back to a multi-threaded architecture using queues, as opposed to a multi-processing architecture using pipes. I did not notice any delays or lags in the multi-threaded architecture after the use of blitting.

4.3 Saving

The saving component of PyjAmaseis is responsible for saving hour-long mseed files. MSEED is a seismology file format where SEED stands for Standard for Exchanging Earthquake Data and MSEED stands for mini SEED. The process for saving mseed files is quite simple with the help of ObsPy, which is one of the main reasons for choosing to develop this application in Python. The write module in the ObsPy framework allows us to export the data collected into seismology file formats very easily and efficiently. The process of saving these files is to create a trace object that contains all the data collected from the TC1 seismometer and then writing this trace along with header information into an mseed file.

In the process of creating header information, I came to notice the headers that are required to store the station location, station ID and station geo coordinates, were missing in the list of available headers in the MSEED file format. Upon raising this issue with Dr. Kasper, he recommended me to save the files into a SAC format. SAC stands for Seismic Analysis Code and is equivalent to SEED. With the required headers available in SAC I saved the header information and the hour-long seismic data into trace object and placed that within a Stream object before writing the stream into a SAC file. A trace contains hour-long seismic data while a stream object can contain several trace objects. Figure 21

presents the code that saves the header information and array of seismic data into a trace, which is then placed in a Stream object before being written into a SAC file.

```
st = Stream([Trace(data=hourSeismicData, header=stats)])
st.write(sacyear+sacmonth+sacday+sachour+sacminute+stats['station']+".sac", format='SAC')
```

Figure 21 Creating a trace object, placing it in a Stream and saving the stream as a .SAC file

This component of PyjAmaseis is also responsible for saving Screenshots of the plot at regular intervals. According to the initial specification, the SAC and Screenshots needed to be saved at least once every hour. There were several options available for capturing the window screenshot in Python. I chose to use the PIL module that contained the Image grab class that could take and save screenshots in PNG file format. PIL stands for Python Imaging Library. These files were saved in the same directory the script was running from. The following line of code carries out the screen capture.

```
ImageGrab.grab().save(now2[0]+'-'+str(now3)+".png", "PNG")
```

Figure 22 Line of code takes screen shot of the plotting window

4.4 Sharing

This component of the PyjAmaseis application focuses on providing the capability of uploading the SAC and screenshots saved by the application to the NZSeis central server. The aim behind this functionality is to create a network of schools that can share the seismic data that is collected at their school with other schools around the country.

I worked alongside Matiu (Mat) Carr from the ScienceIT department at University of Auckland in order to successfully achieve the implementation of this component.

Mat recommended I look into using PycURL to create HTTPS requests to upload the data to a server hosted by the ScienceIT called NZSeis. PycURL is a Python interface to Libcurl. Libcurl is a client-side URL transfer library supporting a large number of protocols such as HTTP, HTTPS, IMAP and SMTPS. Libcurl also supports SSL certificates, HTTP form based uploading, username and password authentication along with many other things.

It took me some time, but writing the code to upload the saved sac file or image was relatively easy with PycURL. The code snippet in figure 23 demonstrates this process.

```
##Upload
contentType = "application/octet-stream"
c = pycurl.Curl()
c.setopt(c.URL, 'https://nzseis.phy.auckland.ac.nz/pyjamaseis/upload/')
c.setopt(c.HTTPHEADER, ['Authorization:'+'Basic %s' % base64.b64encode("kofi:pyjamaseis")])
c.setopt(c.HTTPPOST, [("payload",(c.FORM_FILE, fileNaame, c.FORM_CONTENTTYPE, contentType)), ("mode","sac")])
```

Figure 23 This code creates a multipart form encoded HTTP Post request to upload a SAC file

In order to upload a SAC file I generated a HTTP Post request with PycURL and supplied the required URL location of the NZSeis server, authentication header information, and the SAC file itself. Here in the last line I created a multipart form encoding in order to pass the sac file to the right component of the php script running on the server side. The content type for a sac file is *"application/octet-stream"* where as for the screenshots; the content type needs to be changed to *"image/png".* The "mode" in the multipart form encoding also needs to be change to "image" instead of "sac" when uploading image files. ScienceIT supplies the credentials required in the authorization header to each individual school. These are custom generated to prevent any random user posting data to that address. In this example, "kofi" stands for "Kasper's Office".

The Apache Server has been configured to authenticate the request sent to that location prior to the handling of the payload. Mat has written a php script on the server side that checks the content type and payload to confirm that it is a SAC file or an image file. Upon confirmation, the php script archives this file in a file structure that is kept for back up and saves another copy on a public domain that can be seen on the internet by other schools. Currently there are a lot more security concerns that need to be addressed before we can make the data publicly available on the internet, but the aim is to allow these files to be seen on the ru.auckland.ac.nz website which is the Seismometer in Schools Programme website running in New Zealand headed by Dr. Kasper.

4.5 Additional Features

Along with completing the core functionality I looked into implementing additional functionality that supports the aim of PyjAmaseis and the end user.

4.5.1 TC1 Plug & Play

The first of these is the Plug & Play feature. For those who have used software such as jAmaseis or Amaseis, they are aware of the difficulty in having to look up the exact port name the TC1 seismometer is connected to in device manager and select it from a list of COM ports before they can begin engaging with the software. With teachers and students in mind, I knew that this would be a difficult task for them, and so I looked into a way of removing this requirement of knowing or searching for the COM port. Therefore an end user can directly connect the TC1 to any port and run PyjAmaseis without having to know or do any prior configuration to begin the plotting session. This feature works on all platforms.

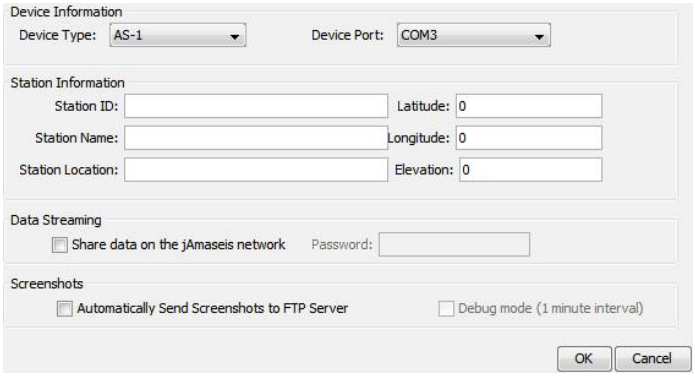

Figure 24 jAmaseis window showing the requirement of Device port selection

The way I carried this out was to inspect the properties of the port the TC1 seismometer is connected to, and with the help of PySerial, I had placed several checks in place that check for pre-determined port properties such as baudrate, parity, timeout, and xonxoff and then select that port as the port that the TC1 seismometer was connected to. Before this check takes place though, there are OS calls that are made which return the list of all active ports then the above-mentioned check takes place before finalizing the port. This functionality has been tested and has proven to successfully work on all operating systems and even when other USB devices are connected.

```
#### Method Returns all active usb ports
def serial_ports():
    """Lists serial ports

    :raises EnvironmentError:
        On unsupported or unknown platforms
    :returns:
        A list of available serial ports
    """
    if sys.platform.startswith('win'):
        ports = ['COM' + str(i + 1) for i in range(256)]

    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this is to exclude your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')

    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')

    else:
        raise EnvironmentError('Unsupported platform')

    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass
    return result
```

Figure 25 The serial_ports() responsible for returning all the currently active USB ports

```
def getSerialPort():
    try:

        activePorts = serial_ports()
        for port in activePorts:
            serialPort = serial.Serial(port)
            if (serialPort.baudrate == 9600):
                if (serialPort.parity == 'N'):
                    if (serialPort.timeout == None):
                        if (serialPort.xonxoff == False):
                            serialPort.close()
                            return port
    except:
        print("Device not found")
```

Figure 26 This method checks each active USB port to see if it's the TC1 connected to it

4.5.2 User Interface

In order to collect the station information I needed to create a UI that allows the user to enter their station name and address. I carried this out by creating an interface using wxPython, which is a wrapper for the cross-platform GUI wxWidgets (which is written in C++) for the Python programming language. Due to the short amount of time I had left near the end of the project, I looked for a quicker way to develop this interface. My search led me to WxGlade, which is a WYSIWYG (What you see is what you get) GUI designer that helps create wxWidgets/wxPython user interfaces. The use of WxGlade was a valuable decision as it allowed me to quickly develop the interface I needed without having to write any wxPython code. The information entered in the text fields is saved locally in a txt file and is accessed every time the application is launched to retrieve the saved information. This retrieved information will be used when creating header information for the SAC files.

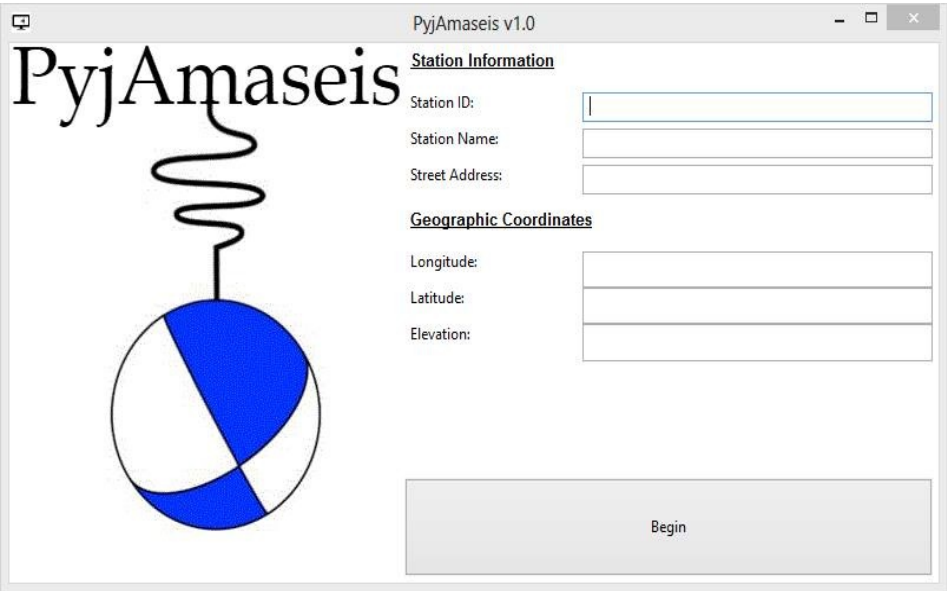

Figure 27 PyjAmaseis initial UI window

4.5.3 Real-time Geo-location Querying

A suggestion that came through in my mid semester presentation from my BTech IT coordinator Dr. Manoharan, was the possibility of looking into a way to auto generate the longitude and latitude fields required for header information. This requirement is understandable as many users especially teachers and students would not know these values and would have to manually search the Internet to know what the geo coordinates of their location are. To save them the trouble I implemented a feature with the help of a module named Pygeocoder, which allows any address written in the "Street Address" text field of the UI will automatically make a call to get the exact geo-coordinates. This call is made after each character is typed so by the end of it, the user will have the exact geo coordinates for the street address they have entered. These retrieved geocoordinates will populate their respective text fields providing real-time feedback to the user.

```
results = Geocoder.geocode(self.streetAddress.GetValue())
longitude, latitude, elevation = results[0].coordinates
self.text_ctrl_6.SetValue(str(longitude))
self.text_ctrl_7.SetValue(str(latitude))
self.text_ctrl_8.SetValue(str(elevation))
```

Figure 28 GeoCoder returns the Longitude, Latitude and Elevation of the Street address entered

4.5.4 Y Plot Shift

The necessity for this functionality arose when one of the TC1 seismometers I was using was not calibrated correctly and so its mode or resting value didn't

match the default value of 32750 that I was using for plotting. It was slightly higher so the whole plot was offset by an arbitrary value. Inorder to correct this, I provided the user with a simple Graph Shift option in the secondary options window that they can use to make incremental changes in shifting the plot on the Y axis. I understood that if such caliberation was required then everytime the aplication was loaded the user would have to manually configure this. Hence I developed it such that everytime the caliberation is made this setting would be saved locally in the text file and when the application is loaded the right configured plot would be displayed.

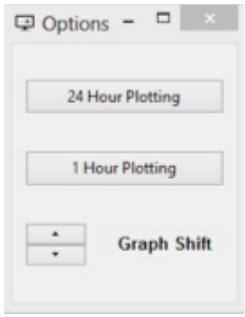

Figure 29 PyjAmaseis Options window

4.5.5 1 Hour Plot

The last functionality I implemented was the 1-hour plot as opposed to the default 24-hour plot. This was an important functionality to implement because in the 24-hour plot we can hardly see the nature of the seismic waves that were being recorded by the TC1 seismometer. However with the 1-hour plot, which plots 5-minute sections, we can clearly see the nature of this seismic activity. I followed the same plotting procedure as I did for plotting a 24-hour plot except it was developed for one hour. The user has the option to shift between the 24-hour plot and 1-hour plot at will without any problems. The one-hour plot is a very important feature because when a teacher is teaching seismology in a classroom a 24 hour plot will not be a very good representative plot to show the slight modulations created when walking or jumping near the TC1 seismometer. However with a 1-hour plot this can be clearly seen and so it serves as a better visual representation for demonstrating seismic activity.

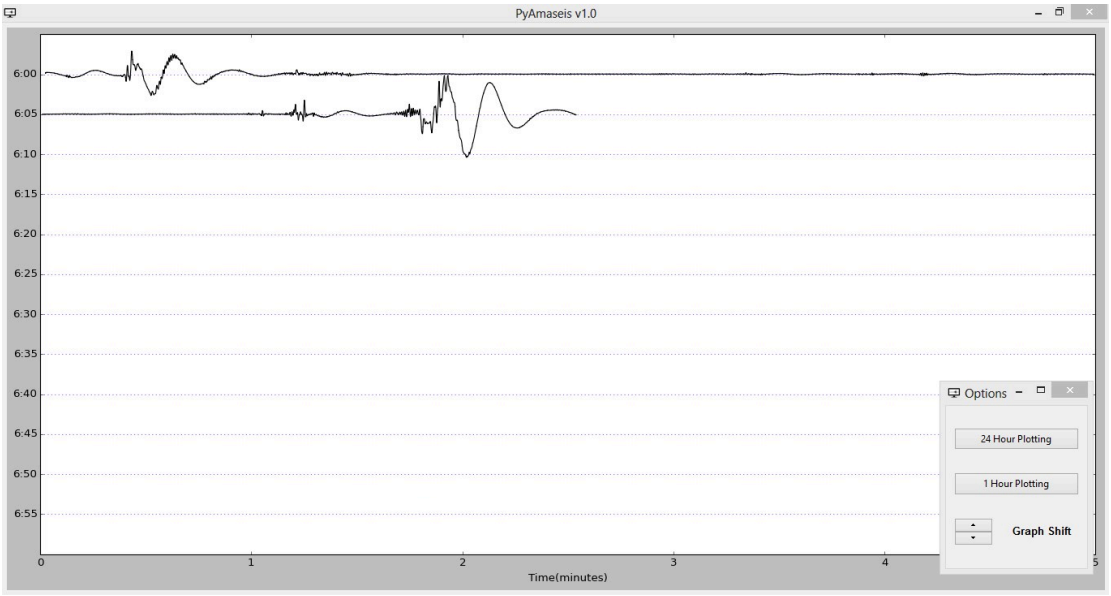

Figure 30 PyjAmaseis 1 hour plot

5. Results

Through out the course of the development of PyjAmaseis, I encountered numerous problems and challenges, which I have already covered in the implementation section of this report. In this section I would like to emphasize the outcome of the implementation and problems I addressed, and the significance the solutions bring to the end users such as teachers, students and seismologists.

The outcome of successfully addressing issues such as the distortion of the plot, delay and lag issues in the plotting component and the use of a single subplot vs 24 individual subplots all contribute to creating a robust and accurate live plotting component. Plotting is the most important functionality out of all (collecting, plotting, saving and sharing) because when teaching students about seismology they need to be able to engage with the seismometer and see the immediate responses to their interactions. The plotting component needs to be accurate and responsive to engage the students and provide them with another interactive way of learning about seismology. Learning to use Blitting was a crucial turning point in the development of the application as having to look into another plotting library would have cost me a lot of time. This could have hindered me from completing all the functionality that was laid out in the beginning of the project. Therefore learning about Blitting turned out to be a timely discovery, which I also happened to implement in a short amount of time. The significance of being able to use the blitting technique in the plotting component is a tremendous plus point because the Matplotlib draw() method was very inefficient and also was not fit for being used in a live plotting situation. The time difference is astounding when comparing the draw method vs using the

blitting technique in a live plotting scenario, with blitting being 12.2 times faster than the draw method, executing in only 0.032 seconds every time.

This technique comes as a significant break through in the field of live plotting applications for seismology because even the currently used applications such as jAmaseis show signs of noticeable latency issues. Thus PyjAmaseis would serve potentially as the first application that does live plotting to sub-second precision. When discussing this with Dr. Kasper he explained how Seismologists understand the internal structure of the earth and why sub-second precision is so critical. Professional seismologists measure the time taken for an earthquake to travel from one location to another. This time is measured using live plotting mechanisms such as seismometers. While analog seismometers can accurately indicate exactly when it experiences an earthquake, digital plotting has seen latency issues thus affecting the accuracy of the measurement. The time taken serves as an indicator as to what lies underneath the crust of the earth. Suppose an earthquake takes a few seconds longer than it was expected, then the seismologists would conclude there is some extra material in between that is responsible for causing this delay. Thus by taking thousands of these sample measurements over decades, seismologists attempt to understand the internal structure of the earth. Hence sub-second plotting is so crucial as it affects the way professional seismologists understand the earth's internal structure. Therefore I can proudly say that PyjAmaseis successfully addresses the latency issues of previous seismology live plotting applications and can successfully be used by professional seismologists around the world.

By implementing the additional features my aim was to remove some of the known problems with seismology applications such as jAmaseis, and present students with more engaging features allowing them to learn more about seismology. Some of the cumbersome functionality in jAmaseis such as - knowing which USB port the seismometer is connected to, and knowing the exact geo location coordinates of their address - are simplified and automated in PyjAmaseis so that teachers and students do not need to worry about these things and can focus on what is most important which is engaging in the live plotting, saving and sharing of seismic data. The 1-hour plot allows the user to experience the high resolution with which the TC1 seismometer captures seismic activity. Viewing the 1-hour plot is a more engaging experience as it shows clearly any subtle seismic activity that cannot be seen in a 24-hour plot.

The current version of PyjAmaseis allows schools to incorporate it into their curriculum and automatically have their SAC files and screenshots uploaded to the NZSeis server. It is only a matter of time before the network is live and every school gets access to every other school's data to compare and learn from.

6. Evaluation

Although PyjAmaseis was not tested in a classroom environment, over the course of development, all of the issues have been addressed and checks are in place to ensure that the application is not only robust but also simple and user friendly. This was part of the specification and has been successfully implemented along with the core functionality to create a new - more engaging software for educational seismology. PyjAmaseis meets all the requirements stated at the beginning of the project.

My contribution to this project is not just the purposeful application that I have built but the solutions to the challenges faced, bring to light some very useful techniques that enhance the live plotting component of seismology applications. I have also introduced several features that simplify and automate some of the technical features present in the previous seismology software such as jAmaseis, where users can now work with a simple user interface and can easily access the different options available without any difficulty or complexity.

Over the course of the project there were a number of things that I had learnt. Understanding how threads and processes worked in Python, inter-process communication and a deeper understanding of Seismology. This project served as a great opportunity for me to enhance my Python development skills along with maintaining and managing a large project using the Git version control system. I had the opportunity to build the TC1 seismometer and understand how it worked. The project contained elements of desktop application development along with adding remote functionality, which allowed me to refresh a lot of the concepts I had learnt in the previous years of my BTech IT degree.

7. Conclusion

With the completion of PyjAmaseis I have not only managed to achieve all my goals for this project but have managed to develop an application that carries more functionality than what was initially laid out. PyjAmaseis not only contains the core functionality such as live plotting, saving and uploading of data, but more importantly simplifies the whole process to make it easier for students and teachers to use. The additional functionality developed into PyjAmaseis enriches the application with fixes from currently existing seismology applications along with providing enhanced features such as a Graph Shift option for calibrating the plot on the y axis and displaying of a 1 hour plot to get clearer visual display of the seismic nature of the plot. PyjAmaseis as it is can now be deployed on computers in schools that have been provided a TC1 seismometer and therefore allow students to have a more informative and educative learning experience when learning about seismology.

The component and connectors architecture of PyjAmaseis allows developers around the world to easily implement and add their own features to the list of features already existing in PyjAmaseis. They can also modify existing components without affecting other components. Thus this makes PyjAmaseis highly adaptable and accessible to keen developers who want to enhance the capabilities of this application.

The sub-second plotting precision of the plotting component would definitely make Professional Seismologists around the world interested in using PyjAmaseis for real seismological calculations.

Thus PyjAmaseis is an all round application that can serve multiple purposes depending on the environment of use. PyjAmaseis is a great application for educational seismology and is equally as highly functional as other seismology applications for professional seismologists to use in their day-to-day work.

8. Future Work

There is always a way to make something better and PyjAmaseis is no different. There are several features I feel that must be developed into PyjAmaseis to make it a more comprehensive solution. Several of these features are described below along with potential ways of implementing them.

It would be a valuable feature to look into being able to stream data from one station to another and plotting it there live. This way PyjAmaseis could plot the live data from another station somewhere else in the country or even around the world. In discussions with Mat, he told me that this feature could be easily implemented using a free IRC (Internet Relay Chat) client that takes care of all the communication requirements between all the clients (PyjAmaseis instances) connected to it. It works by creating a room consisting of all clients and a client (A) can choose to subscribe to another client (B) in which case any data sent from client B to the IRC is forwarded to client A. Clients can also subscribe to multiple clients at once. There are several freely available IRC chat clients available on the Internet such as mIRC and kiwiirc to name a few.

One of the very important features that should be implemented is to provide visual cues such as popups and labels that will display when a pre-recorded seismic behaviour is noticed. Labels are a feature provided by Matplotlib and can be incorporated effectively to align with the overall learning goal of the application.

Finally the ability to drag and save a desired section of the plot into a SAC file should be incorporated into PyjAmaseis as this allows seismologists to quickly select a part of the plot that represents an earthquake and study or share it with others. This is a valuable feature as it gives the user the flexibility to save custom sized SAC files instead of the default 1 hour long SAC files. Matplotlib keeps track of the exact x and y coordinates of the cursor when it is within the plotting figure. By using mouse events, such as clicked and released, along with the cursor's starting and ending coordinates, we can determine which plot and what range of values the user is trying to select. Although the application does not hold the data that is read from the Seismometer after it's plotted, we can access the respective SAC file and retrieve the portion that represents the users selection.

9. Bibliography

1. Cochrane, L. (2009). Winquake Version 2.8 documentation. Retrieved on August 4th 2014 from http://psn.quake.net/software/wq28doc.pdf.
2. Coleman, B., & Gerencher, J. (2008) A software system for real-time sharing of seismic data in educational environments.
3. Garlan, G., & Shaw, M. (1994). An Introduction into Software Architecture. Carnegie Mellon University Pittsburgh, PA.
4. Gerencher, J., & Jackson, R. (1991). Classroom utilization of a multi-axis lehman seismograph system. Journal of Geological Education.
5. Gerencher, J., & Sands, M. (2004). Online near-real-time seismic system for the classroom. Journal of Geological Education.
6. IRIS. (2014). Iris - education and outreach. Retrieved August 3rd 2014 from http://www.iris.edu/hq/programs/education_and_outreach.
7. IRIS. (2014). Iris - incorporated research institutions for seismology. Retrieved August 3rd 2014 from http://www.iris.edu/hq/.
8. jAmaseis. (2009) Seismology Software Meeting the Needs of Educators. Retrieved October 18th 2014 from http://www.iris.edu/gallery3/research/2010proposal/E_and_O/system.
9. Matplotlib – Python Plotting Library. Retrieved October 25th 2014 from http://matplotlib.org/.
10. ObsPy. (2012). A Python Framework for Seismology. Retrieved October 25th 2014 from https://github.com/obspy/obspy/wiki.
11. Paradigm. (2014). Paradigm Advanced Science for everyone. Retrieved on August 5th 2014 from http://www.pdgm.com/solutions/seismic-processing-and-imaging/.
12. PyDev. (2011). PyDev – Python IDE. Retrieved August 24th 2014 from http://pydev.org/.
13. Ramirez, J. (2012). Learning from Manifold – Valued Data: An Application to Seismic Signal Processing. University of Colorado.
14. RU. (2014). New Zealand Seismographs for Schools. Retrieved October 15th 2014 from http://ru.auckland.ac.nz/.
15. TUTS+. (2010). Game Development Glossary – Blitting. Retrieved October 22nd 2014 from http://gamedevelopment.tutsplus.com/articles/gamedev-glossary-what-is-blitting--gamedev-2247.
16. WesternGeco. (2012). The Omega Seismic Processing System – Seismic analysis at your fingertips. Retrieved October 25th 2014.