A project to develop analysis of Radio JOVE data

Stuart Mumford
stm7@aber.ac.uk
Planetary and Space Physics Group
Aberystwyth University

Keiron Pizzey
kpp7@aber.ac.uk
Condensed Matter Group
Aberystwyth University

May 2010

Abstract

The aim of this project is to develop a working signal processing program for use by third year students at Aberystwyth University. In producing the PriMaRy Jove Analysis program (written in python 2.6 and licensed under the GNU General Public License) the theory of signal processing is discussed and it is shown that an accurate Io storm event can be retrieved from the data.

Contents

1 Introduction

The aim of the project is to create a re-usable data analysis program that will accurately process Radio Jove data so Io storm events can be detected in the not inconsiderable background noise. Radio Jove is a project controlled by NASA for use in schools and universities designed to help pupils/students to measure radio frequency radiation from the Sun, Jupiter and Jupiter's interaction with its moons. The final design is the PriMaRy JOVE Analysis v1.0 (Pizzey and Mumford Radio JOVE Analysis). It is formed into a full installer package which is compatible with Windows XP (or later).

The detection of JOVE radio

2 Data Analysis

Signal processing of the data primarily revolves around reducing the magnitude of noise whilst still maintaining the underlying signal pattern; this is a difficult balance to maintain as if the signal processing is taken to extremes it can reduce the underlying signal to nothing.

2.1 Binning the data array

Binning the signal array involves averaging equal sections of the array and saving the result to a new smaller array; This step will reduce the signal resolution however is a necessary step in the analysis process. Equation (1) shows how an array A may be binned into the array B where N is the number of elements binned into a single element of B.

$$B_i = \frac{1}{N} \sum_{i}^{N} A_i \tag{1}$$

The number N is calculated in such a way that it changes from bin to bin to keep the time-length of the bin the constant chosen by the user.
Binning is important because it reduces the run-time of the analysis code, less elements lead to quicker code, and it also enables the end-user to have a stronger grip on the numbers (the bin time is changeable but is set to default of one second per bin)

2.2 FFT frequency filtering

After the signal has been binned into an appropriate array (discussed in Section 2.1) it is necessary to process the signal further as it is likely still not usable to gain acceptable results. A strong data analysis tool is filtering frequencies using FFT. The Fast-Fourier-Transform is an algorithm designed to compute discrete fourier transforming, using this technique it is possible to remove almost all noise from a signal however it must be used carefully as severe changes can be exacted upon a signal [3].

Fourier transforming a signal involves transforming from one domain into its reciprocal, in the case of signal processing this is transforming between the time domain and the frequency domain and vice versa. Figure 1 is an example of a fourier transform of a single sine wave into the frequency domain [3].

Noise is typically a much higher frequency than any underlying signal we are searching for; by fourier transforming into the frequency domain, reducing high frequencies to zero and then inverse fourier transforming back into the time domain it is possible to reduce all high frequencies to zero and so substantially reduce the noise in the signal. Reducing high frequencies to zero is referred to as a low-pass filter whilst the opposite (low frequencies to zero) is a high-pass filter.

2.3 Apodization Functions (Window Smoothing)

Apodization is another noise reduction technique included with the signal processing suite as an alternative to FFT filtering, however it is not as powerful in its noise reduction (and yet less dangerous when misused). In apodization a

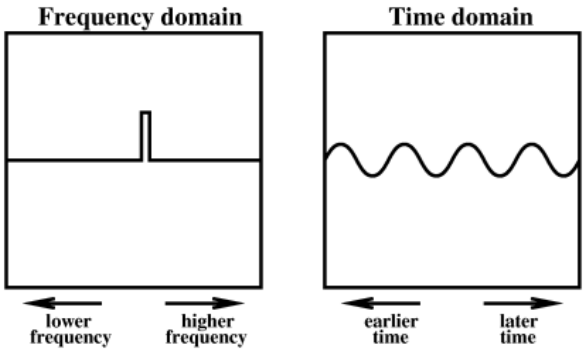

Figure 1: Example of fourier transforming [1]

window type and length is chosen to create a window function which is then convolved with the signal. The result is a signal of similar shape but reduced noise. [3] The window length must be an integer greater than or equal to three while the window types available are:

- Bartlett;
- Blackman;
- Flat;
- Hanning;
- Hamming.

The window type chosen will strongly affect the smoothing of the data, for example the flat type provides excellent resolution for signals of comparable strength but is a poor choice if the amplitudes differ too much (referred to as low-dynamic range); whilst the other extreme is a high-dynamic range type (such as Bartlett) which are poor in terms of sensitivity (as such weak signals are easily lost in the noise) and in the middle are types such as Hamming and Hanning. It is left to the end-user to experiment with the window types and window lengths to see what fits their data best [3].

2.4 Peak fitting

Included within the signal processing suite are functions which are not designed to reduce noise but to aid the end-user in analysis of the processed data. One of these functions is *Gaussfit*, a gaussian peak fitting function. *Gaussfit* makes use of the Levenberg-Marquardt algorithm (LMA) to compute a non-linear regression in parameter space to accurately reflect measure data.

$$y = ae^{\frac{(x-b)^2}{2c^2}} + d \begin{cases} a = \text{Height of peak;} \\ b = \text{Centre of peak;} \\ c = \text{Width parameter;} \\ d = \text{Vertical offset.} \end{cases} \tag{2}$$

The function chosen for peak fitting is the gaussian function with an additional vertical offset, given by Equation (2). The LMA iterates through parameter space until a set of parameters (a,b,c,d) are found which minimise the residuals (fit the measured data). A good approximation must be made for the starting parameters otherwise the LMA will not find the global minimum [3].

Gaussfit itself is not available to the end-user as a function but is used in a function which is available, the function *Peakfinder*. *Peakfinder* is a function designed to search through the full signal and search for peaks matching certain

criteria, after finding peaks it then attempts to fit a gaussian curve to them using *Gaussfit*. The criteria needed for a peak to be accepted as a true peak are:

- It must be a point of inflexion, in that the gradient $\frac{\partial y}{\partial x}$ changes sign from positive to negative;
- The top of the peak must be greater than G times the standard deviation of the average signal value where G is a parameter chosen by the end-user.

The criteria are chosen in such a way to prevent background noise from being classed as peaks.

3 Implementation

After considering the signal analysis the next major consideration is to decide a reliable and user friendly method of providing the end user with a method of applying the analysis to their data. The decision was made to develop a GUI(graphical user interface) primarily to allow use of the program without a working knowledge of python or bash commands. The GUI was developed using the Qt4 package and PyQt4, this allowed for a graphical design of the form and a logical implementation into python code.

The first feature that was crucial to making the program useful to the user was to integrate support for the Sky-Pipe data format (.spd files); this was a considerable hurdle to overcome, as such the decision was taken to use the free, pre-written SPD2FITS converter(http://www.jupiterradio.com/jove-spd2fits.php). This converter can be called from the command line within python and therefore allows the program to seamlessly read the spd data file by first converting it into a python readable FITS file.

At Aberystwyth the Sky-Pipe data recorder is set up to record 5 data channels simultaneously, this creates a recorded spd file with the five data channels and a time channel. SPD2FITS only supports one data channel per spd file, therefore only one data channel must be saved to the spd file at a time. Python support for FITS files is provided by a module (PyFITS), the files generated by SPD2FITS required iterative processing on loading to spilt time and data into two separate arrays which takes a substantial amount of time for large files.

To implement this complex opening procedure and include the ability to open comma separated, and basic ASCII data files, the first routine to be created was a class to open and process all these types of data files into a universal format for the rest of the routines to draw on. The class (*opendata*) is called with a file name, from this it determines the file type and processes the data as required. The resulting instance of the class has the following useful attributes:

- Time
- Data
- Header

The time and data arrays are the same format for any compatible opened data type and the header attribute will be available if a spd or FITS file is opened containing header data. In the program three data/time arrays are maintained. The first is the attributes of the opendata class, this is kept to maintain a copy of the original, raw data to fall back on. Next are a current data array and a saved data array; these provide the ability to make changes to the data and then save or discard that change, rather than making every change permanent. When the user runs a process on the data, it is run on the saved data array and saved to the current data array. If the user is satisfied with the result and decides to save the data, the change is saved over the saved data array, if not any following analysis will be performed on the saved data array, ignoring the last process.

The next stage of the GUI development was to develop the calling interfaces for each of the available data processing routines. This was a case of creating boxes and buttons for the users to specify each parameter, all of the buttons for each process use the variables set in the boxes and run the process on the saved data array, then plot in the internal window the current data arrays.

The peak fitting routines required separate consideration, with regards to plotting, because they generate an extra data array rather than replacing one. As such when the peak plotting routine is run an extra data array is created, which is then plotted on top of the initial data using a different call to the plotting functions. The automated analysis button is a one click button that runs, the binning function, pass filter smoothing and peak finding. These functions are run on the raw imported data with the parameter's specified in all the boxes in the GUI.

The internal plot window in the GUI is a widget derived from the matplotlib library for python plotting. It is however not as flexible as calling the matplotlib.pyplot functions directly. For this reason it was decided to include the ability for the user to plot the same data as is automatically plotted in the internal widget, in a external plot window by calling the library directly. The additional functionality of customising the external plot was coded into a separate window allowing the setting of graph titles.

The final feature of the GUI to be implemented was the ability to save data. Due to the method the spd files were imported it was impossible to save back to this format; also the ability to save to FITS files was not included due to the limited use of such a format for this type of data, and the technical challenges this presented. The ability to save out raw and processed data is available in both csv(comma separated variable) and non-delimited ASCII format, the saving routines are provided by the Python NumPy library, the program is compatible with all data files it can save.

4 Results

Shown in this section are graphs which document the progress of a signal through the program and explain how processed are applied to retrieve usable data. Figure 2 shows the original signal, measured by Pope and Watts [2], plotted against time. The measurements were taken for three hours beginning on November 1st 2009 at 16:30 and were taken at the Institute of Mathematical and Physical Sciences, Aberystwyth.

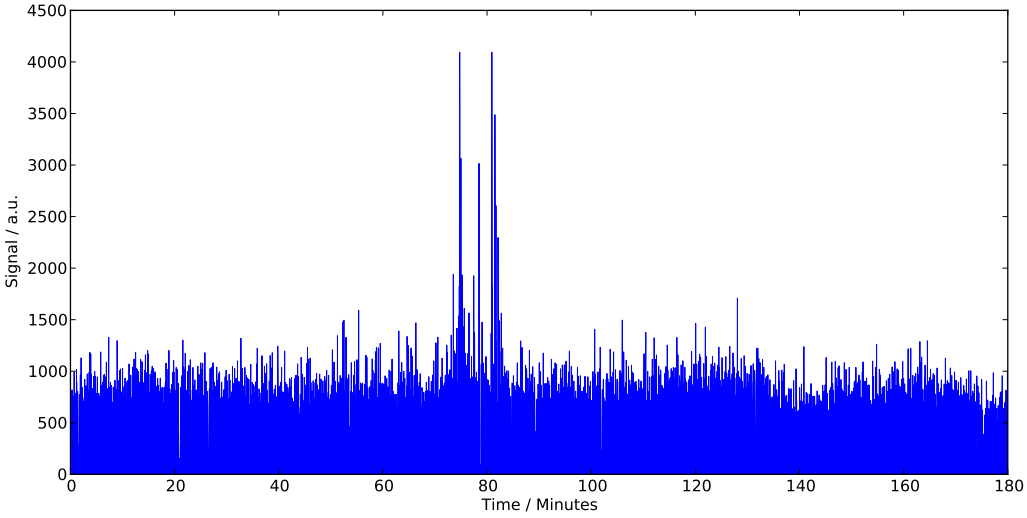

Figure 2: Graph of raw results

Figure 3 shows the signal binned into one second intervals and plotted on the same time scale; this step involved taking an average of 42 signals for each bin. In comparing the signal between Figure 2 and Figure 3 it is possible to see the reduction in noise however the noise levels are still too high to make an accurate assessment of Io events.

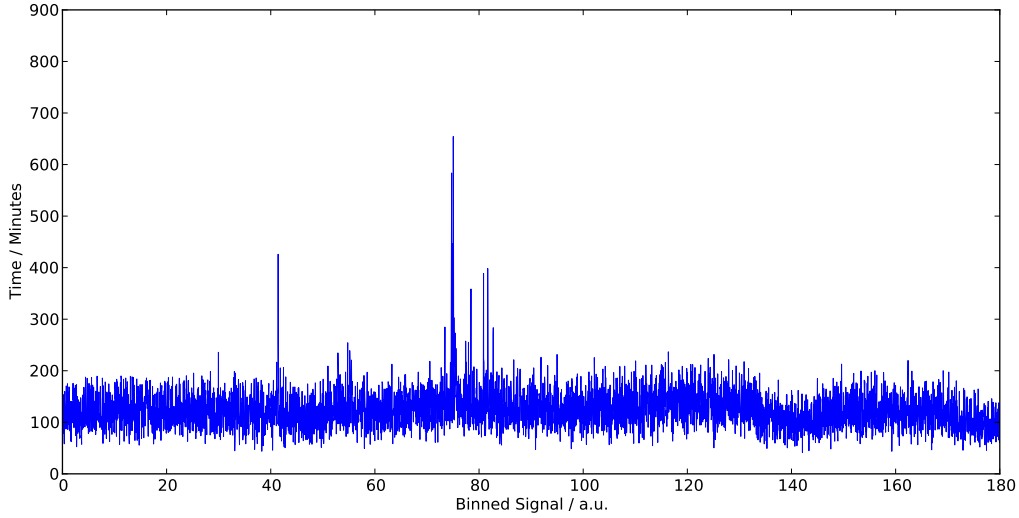

Figure 3: Graph of results binned into 1sec intervals

Figure 4 shows the signal after it has been passed through an FFT Filter (type = “low”, lim = 120) to reduce the noise.

Figure 4 clearly shows two Io events at 75:10 and 81:10 minutes and also one suspected Io event at 73:14.

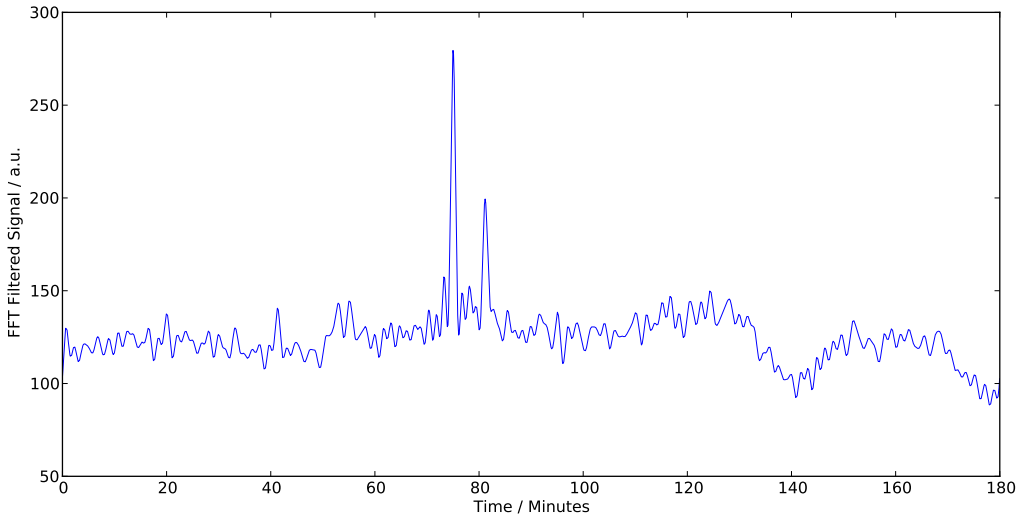

Figure 4: Graph of FFT Filtered results

Figure 5 is a graph of the filtered Io events, the red lines show fitted gaussian peaks found using the *Peakfinder* algorithm. Starting parameters were estimated using the *Peakfinder* algorithm and the final parameters are given in Table 1.

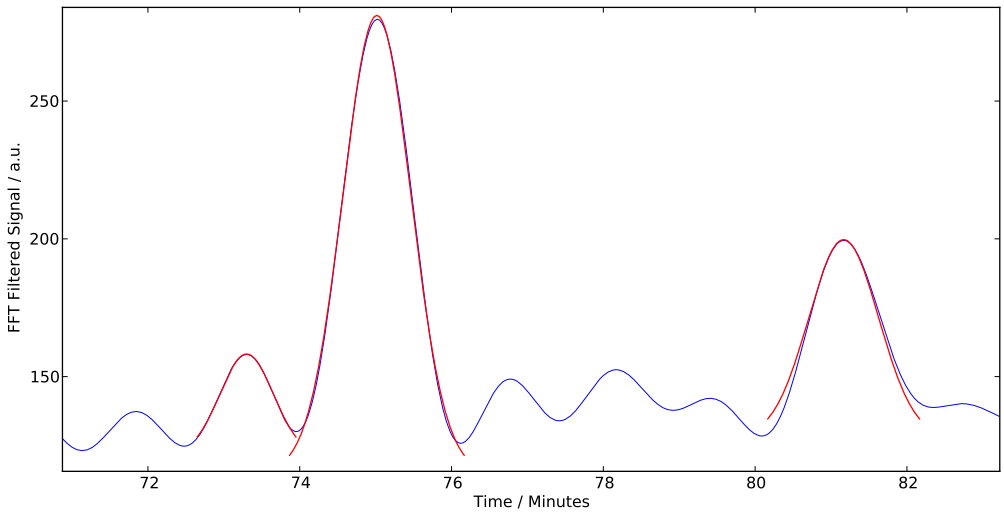

Figure 5: Graph showing three fitted gaussian peaks

Event No.	Height of peak / a.u.	Central position / mm:ss	Width Parameter / a.u.	Vertical Offset / a.u.
1	35.50	73:14	21.80	121.46
2	161.17	75:10	28.04	115.64
3	70.50	81:10	28.23	128.16

Table 1: Results from fitting Io events using *Peakfinder*

5 Discussion

The aim of the project was to design a user-friendly signal processing program which can be used by third year physics students at Aberystwyth University; a program was designed in python 2.6 and successfully compiled into an executable file. As shown in Section 4 the signal processing is successful in reducing noise and accurately measuring Io events.

Initially the binning algorithm used a constant number of elements for each bin, this constant derived from the bin-time chosen by the end user. This method was chosen because it was assumed the sampling rate was constant enough to make it feasible. Through debugging, however, it was found that this was an incorrect assumption and so a new binning technique was developed; this new technique involves using the time array itself for choosing binning ranges, as such the number of elements per bin is not a constant.

The main signal filtering technique used in the experiment is FFT filter processing (Section 2.2) with an additional method available through window smoothing. FFT filtering is a powerful tool which can systematically remove the majority of noise from a signal, leaving the underlying signal easily available for analysis; there are, however, problems in using FFT filtering which can render it useless. If care is not taken in removing frequencies it is possible to destroy intricacies in a signal which may be overlooked, for the vast majority of cases a simple low-pass filter (the default setting in the signal processor) is sufficient to remove noise from the signal however the end-user should be careful in their application of FFT filtering. Window smoothing, on the contrary, is much safer in its usage (if window types are chosen correctly) but is not as effective at removing noise.

The *Peakfinder* algorithm is powerful in its potential to ease the workload of the end-user but can also fail if the end-user does not use it properly. There is a limit which is chosen by the end-user which controls what is defined as a "peak" by the algorithm; if this limit is chosen to be too small then background noise will be fitted as gaussian peaks and may interfere with finding the underlying signal. Another issue with the use of *Peakfinder* is the use of the vertical offset component of the gaussian function. The vertical offset is necessary because of peaks being in higher levels of noise than other peaks, by adding a vertical offset it is possible to fit each peak equally without having to fit then to the same baseline. The (necessary) evil added by the vertical offset is the fact that the FWHM may not be at the half-maximum of the peak height because the baseline may be significantly below the background noise. The end-user, as is a general rule for data analysis programs, must keep their wits about them in looking at the outputs.

The implementation of the code into a GUI was successful, however there are areas that with further developement could improve the user experience and flexibility of the program. The main issue, is a bug where in the external matplotlib plotting window which causes the icons to not display correctly. There are other improvements that could be made to the program, this includes translating the SPD2FITS code from Java into python to make this feature native and faster, also translating this feature would allow saving to the spd file format. Another area where improvement could be made is in the graphic design of the GUI, while the current version is functional it is not asthetically pleasing or particularly intuitively laid out.

As well as improvements to features and the user interface, there are many improvements to the source code that could be made to enhance the expansibility of the code and its readability and efficiency. The code written is functional but may not subscribe to all desirable object orientated programing principles as this was the authors first venture into GUI development and OOP in python.

The data analysis for the program could be improved by giving the option of different smoothing filters; one of the functions not added into the final program is the Savitzky-Golay algorithm. The Savitzky-Golay algorithm performs

a local polynomial regression to determine a smoothed value for each point. Depending on the degree of polynomial will depend on the smoothing applied, this parameter can be easily chosen by the end user.

Further improvement which would greatly help the computation is improving the memory allocation. By using pointers to address memory the computation speed can be increased drastically especially over file iteration. Further speed increases could be gained from using the F2py interface, an interface designed to compile FORTRAN code usable in python programs. By coding in FORTRAN computational speed gains are possible due to FORTRAN's low-level environment and its general enhanced speed, especially in areas of iteration (which the project uses at length).

6 Conclusion

A project was undertaken to develop an easy to use signal processor to reduce noise and improve the detection of Jupiter radio events. A GUI program was developed that facilitated three data processing methods, binning, FFT pass filtering and window smoothing. There was also a gaussian peak finding and fitting routine to provide parameter information about peaks in the data. It is possible to load data into the program from the Sky-Pipe recorder as well as other sources if the data is formatted in a compatible way.

The analysis methods provide a wide variety of flexible routines to remove noise and minimise signal loss. However as with all noise reduction techniques it is possible to remove the signal from the data accidentally; the program includes a trial and error approach to fine tuning the noise reduction parameter to minimise this effect.

The program was byte-compiled into a standalone windows executable program, and tested with Windows XP and above. Finally the program and the source code was packaged into a installer and released under the GPL v3 licence.

7 Acknowledgments

The authors would like to thank Dr Balazs Pinter for supervision and the opportunity to do something different and challenging in this project, Mr Les Dean for help with the radio antenna and their colleagues Matt, Frances, Paddy, Hannah, Drew, Sam, Alex and Karl for providing data to test the program with.

8 Python Code

As the coding was so heavily involved in the project, the python code used to devise the program is included.

8.1 Program Code

```
 """
      PriMaRy JOVE Analysis is free software:  you can redistribute it and/or modify
      it under the terms of the GNU General Public License as published by
      the Free Software Foundation, either version 3 of the License, or any later version.

      PriMaRy JOVE Analysis is distributed in the hope that it will be useful,
      but WITHOUT ANY WARRANTY; without even the implied warranty of
      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
      GNU General Public License for more details.

      You should have received a copy of the GNU General Public License
      along with PriMaRy JOVE Analysis.  If not, see <http://www.gnu.org/licenses/>.
"""
import sys
from PyQt4 import QtCore, QtGui
#Import Form from file
from MainWindow1 import Ui_MainWindow
from about import Ui_AboutDialog
from plotcontrol import Ui_PlotDialog
import subprocess as sub
import os
import shutil
import time
import math
os.chdir('../../')
sys.path.append(os.getcwd())
import epic

class About(QtGui.QDialog):
      def __init__(self,parent=None):
            QtGui.QDialog.__init__(self,parent)
            self.about = Ui_AboutDialog()
            self.about.setupUi(self)

class PlotControl(QtGui.QDialog):
      def __init__(self,x,y,parent=None):
            #Deafult Init Stuffs:
            QtGui.QDialog.__init__(self,parent)
            self.plotcontrol = Ui_PlotDialog()
            self.plotcontrol.setupUi(self)
            #Rename plot to make life easier
            self.plt = epic.plt
            #Define Varibles:
            self.x = x
            self.y = y
            self.xlabel = ''
            self.ylabel = ''
            self.title = ''
            self.firstrun = True
            #Connect Some stuff:
            conn = QtCore.QObject.connect
            ui = self.plotcontrol
            #Plot Button
```

```
            conn(ui.PlotButton, QtCore.SIGNAL("clicked()"), self.ploty)
            #Plot Title
            conn(ui.Title, QtCore.SIGNAL('textChanged(QString)'), self.titley)
            #Plot xlabel
            conn(ui.xLabel, QtCore.SIGNAL('textChanged(QString)'), self.xlabely)
            #Plot ylabel
            conn(ui.yLabel, QtCore.SIGNAL('textChanged(QString)'), self.ylabely)

      #Define some functions
      def titley(self,text):
            self.title = str(text)

      def xlabely(self,text):
            self.xlabel = str(text)

      def ylabely(self,text):
            self.ylabel = str(text)

      def ploty(self):
            if self.firstrun == True:
                  fig = self.plt.figure()
                  ax = fig.add_subplot(111)
                  ax.plot(self.x,self.y,'g',linewidth = 0.25)
                  ax.xaxis.set_major_formatter(epic.tick.FuncFormatter(epic.axesformatter))
                  self.firstrun = False
            self.plt.title(self.title)
            self.plt.xlabel(self.xlabel)
            self.plt.ylabel(self.ylabel)
            self.plt.show()

class Main(QtGui.QMainWindow):

      def __init__(self,parent=None):

            #Deafult Initalistion and display form
            QtGui.QMainWindow.__init__(self, parent)
            self.ui = Ui_MainWindow()
            self.ui.setupUi(self)
            self.ui.statusbar.showMessage('Ready')
            self.file = ''
            #This is the current data in use by the user, not the
            #original file data that is self.file.data
            self.data = ''
            self.currentdata = ''
            self.time = ''
            self.currenttime = ''
            self.peakplot = ''
            self.starttime = () #Set deafult start and end times,
                                          #so if left unmodified it will export
            self.endtime = ()   #From begging or end of array.
            #Other instance varibles that need defining:
            #These are the deafult values in the form.
            self.binval = 1
            self.FFTlim = 10
            self.FFTtype = 'low'
            self.winlength = 11
            self.wintype = 'hanning'
            self.peakcutoff = 2.0
      #Shortcuts
```

```
        conn = QtCore.QObject.connect
        self.mpl   = self.ui.mplwidget
#Create Plot Stuffs
        self.fig = self.mpl.figure
        self.ax1 = self.fig.add_subplot(111)
#Connect About box
        conn(self.ui.pushButtonAbout, QtCore.SIGNAL("clicked()"), self.about)
#Connect Custom Plot box
        conn(self.ui.CustomPlot, QtCore.SIGNAL("clicked()"), self.customplot)
#Connect Open box
        conn(self.ui.pushButtonOpen, QtCore.SIGNAL("clicked()"), self.open)
#Connect Plot Button
        conn(self.ui.pushButtonPlotRaw, QtCore.SIGNAL("clicked()"),self.plotdata)
#Connect Save Graph Button
        conn(self.ui.SaveGraphButton, QtCore.SIGNAL("clicked()"),self.savedata)
#Connect Clear Button
        conn(self.ui.ClearButton, QtCore.SIGNAL("clicked()"),self.clearbutton)
#Connect External Plot Button
        conn(self.ui.ExternalPlot, QtCore.SIGNAL("clicked()"), self.externalplot)
#Connect Binning Button and Spin Box
        conn(self.ui.BinButton, QtCore.SIGNAL("clicked()"),self.binning)
        conn(self.ui.BinNumber, QtCore.SIGNAL("valueChanged(int)"),self.binvalue)
#Connect FFT Button and spin box and combo box
        conn(self.ui.FFTbutton, QtCore.SIGNAL("clicked()"),self.fft)
        conn(self.ui.FFTlim, QtCore.SIGNAL("valueChanged(int)"),self.fftlimit)
        conn(self.ui.FFTtype, QtCore.SIGNAL("currentIndexChanged(const QString)"),self.ffttype)
#Connect Window function
        conn(self.ui.WindowLength, QtCore.SIGNAL("valueChanged(int)"),self.windowlength)
        conn(self.ui.WindowType, QtCore.SIGNAL("currentIndexChanged(const QString)"),self.window
        conn(self.ui.WindowButton, QtCore.SIGNAL("clicked()"), self.window)
#Connect Peak Fit / Find
        conn(self.ui.PeakFitCutoff, QtCore.SIGNAL("valueChanged(int)"),self.peakfitcutoff)
        conn(self.ui.FindPeaks, QtCore.SIGNAL("clicked()"), self.findpeaks)
        conn(self.ui.PlotPeaks, QtCore.SIGNAL("clicked()"), self.plotpeaks)
        conn(self.ui.AutomatedAnalysis, QtCore.SIGNAL("clicked()"), self.automatedanalysis)
#Connect the save whole file button
        conn(self.ui.SavetoFile, QtCore.SIGNAL("clicked()"), self.savetofile)
#Connect the time start and end and the part save button
        conn(self.ui.StartTime, QtCore.SIGNAL("textChanged(QString)"), self.starttimef)
        conn(self.ui.EndTime, QtCore.SIGNAL("textChanged(QString)"), self.endtimef)
        conn(self.ui.PartSave, QtCore.SIGNAL("clicked()"), self.partsave)
    def about(self):
        """ Opens the About window """
        ab = About()
        ab.exec_()
    def customplot(self):
        """Opens the custom plot window, and feeds it current time and data """
        cp = PlotControl(self.currenttime,self.currentdata)
        cp.exec_()
    def plot(self,x,y):
        """ Simplyfies a simple x,y plot"""
        self.mpl.axes.cla()
        self.mpl.axes.plot(x,y)
        self.mpl.axes.xaxis.set_major_formatter(epic.tick.FuncFormatter(epic.axesformatter))
        self.mpl.draw()
    def open(self):
        """ This opens a file using the opendata class in epic """
        #Set status Bar
        self.ui.statusbar.showMessage('Opening File ....  Please Wait')
```

```
        #Create a open file box, looking only for spd fits and csv files.
        filename = QtGui.QFileDialog.getOpenFileName(self, 'Open data file','./',"Data Files (*.
        #Set filename to box in UI
        self.ui.lineEditCurrentFile.setText(filename)
        #Run open data class
        self.file = epic.opendata(str(filename))
        #Set current in use data to the file data on Open
        self.data = self.file.data
        self.time = self.file.time
        #Get start and end time, format them to be pretty.
        self.starttime = self.file.time[0]
        self.endtime = self.file.time[-1]
        print self.starttime, self.endtime
        self.prettystart = time.strftime("%d %b %Y %H:%M:%S",time.gmtime(self.starttime))
        self.prettyend = time.strftime("%d %b %Y %H:%M:%S",time.gmtime(self.endtime))
        self.lentime = self.endtime - self.starttime
        #Set them into the boxes
        self.ui.lineEditFileStart.setText(self.prettystart)
        self.ui.lineEditFileEnd.setText(self.prettyend)
        #Set lengths and Number of samples
        self.ui.LengthofFile.setText(str(math.ceil(self.lentime / 60.0)))
        self.ui.NumberofSamples.setText(str(len(self.file.data)))
        self.data = self.file.data
        #Set status bar to finished.
        self.ui.statusbar.showMessage('Ready')
    def plotdata(self):
        """ This is the raw data file plotting, it sets the
        current data files to the original data and plots """
        self.currentdata = self.file.data
        self.currenttime = self.file.time
        self.plot(self.currenttime,self.currentdata)
    def externalplot(self):
        """ This plots the current data object in an external window,
        it also checks for peak fitting to be run """
        if self.peakplot != '': # If peak finding has been run then self.peakplot
                                                #will be populated and therefore
                                                #both things need plotting
                fig = epic.plt.figure()
                ax = fig.add_subplot(111)
                ax.plot(self.currenttime,self.currentdata,'b', linewidth = 0.25)
                ax.plot(self.currenttime,self.peakplot)
                ax.xaxis.set_major_formatter(epic.tick.FuncFormatter(epic.axesformatter))
        else:
                fig = epic.plt.figure()
                ax = fig.add_subplot(111)
                ax.plot(self.currenttime,self.currentdata,'g',linewidth = 0.25)
                ax.xaxis.set_major_formatter(epic.tick.FuncFormatter(epic.axesformatter))
                epic.plt.show()

    def savedata(self):
        """ This saves the current data and time """
        self.data = self.currentdata
        self.time = self.currenttime
    def clearbutton(self):
        """ This plots the saved data, replacing the current
        data and clears peakplot for good measure """
        self.plot(self.time, self.data)
        self.peakplot = ''
    #Binning
```

```
def binvalue(self,i):
        """ Set Bin value """
        self.binval = i
def binning(self):
        """ Binning routine, binns and plots """
        self.ui.statusbar.showMessage('Binning Data')
        self.currenttime,self.currentdata = epic.binner(self.time,self.data,self.binval)
        self.plot(self.currenttime,self.currentdata)
        self.ui.statusbar.showMessage('Ready')
#FFT Smoothing
def fft(self):
        """ Run the band pass filter on the saved data set """
        self.ui.statusbar.showMessage('Pass Filter Smoothing')
        self.currentdata = epic.passfilter(self.data,self.FFTtype,self.FFTlim)
        self.plot(self.currenttime,self.currentdata)
        self.ui.statusbar.showMessage('Ready')
def fftlimit(self,i):
        """ Set frequency limit """
        self.FFTlim = i
def ffttype(self,FFTtype):
        """ Set pass filter type to eith high or low """
        self.FFTtype = str(FFTtype)
#Window Smoothing
def windowlength(self,i):
        """ Set window smoothing window length"""
        self.winlength = i
def windowtype(self,wintype):
        """ Set Window Smooothing Type """
        self.wintype = str(wintype)
def window(self):
        """ Window smoothing function, again with plot """
        self.ui.statusbar.showMessage('Window Smoothing Data')
        self.currentdata = epic.smooth(self.data,self.winlength,self.wintype)
        self.plot(self.currenttime,self.currentdata)
        self.ui.statusbar.showMessage('Ready')
#Peak Finder
def peakfitcutoff(self,i):
        """ Set the peak cut off varible from the input """
        self.peakcutoff = i
def findpeaks(self):
        """ This runs the peak finding routine, then prints
        out the time of each peak in a message box """
        self.ui.statusbar.showMessage('Finding Peaks')
        peaks = epic.peakfinder(self.data, self.peakcutoff)
        peaktime = []
        prettypeak = []
        for i,each in enumerate(peaks):
                peaktime.append(str(self.time[each[1]]))
                t = self.time[each[1]]
                prettypeak.append(time.strftime("%H:%M:%S",time.gmtime(t)))
        foundpeaks = "Peaks have been found at the following times:  \n %s" %(prettypeak)
        reply = QtGui.QMessageBox.question(self, 'Message', foundpeaks, QtGui.QMessageBox.Ok)
        self.ui.statusbar.showMessage('Ready')
def plotpeaks(self):
        """ This Finds the peaks, then plots both
        the data and the compressed peaks on the graph"""
        self.ui.statusbar.showMessage('Finding Peaks')
        peaks = epic.peakfinder(self.data,self.peakcutoff)
        self.peakplot = epic.peakplotter(self.data,peaks)
```

```
            self.mpl.axes.cla()
            self.mpl.axes.plot(self.currenttime,self.peakplot,self.currenttime,self.currentdata,'-')
            self.mpl.axes.xaxis.set_major_formatter(epic.tick.FuncFormatter(epic.axesformatter))
            self.mpl.draw()
            self.ui.statusbar.showMessage('Ready')
      def automatedanalysis(self):
            """ This is quite simple, it runs the three best
            processes, saves the data after each one."""
            #Reset to Original file data
            self.currentdata = self.file.data
            self.currenttime = self.file.time
            self.binning()
            self.savedata()
            self.fft()
            self.savedata()
            self.plotpeaks()
            self.savedata()
      def starttimef(self,time):
            h = str(time[:2])
            m = str(time[3:5])
            s = str(time[-2:])
            self.starttime = (int(h),int(m),int(s))
            print self.starttime
      def endtimef(self,time):
            h = str(time[:2])
            m = str(time[3:5])
            s = str(time[-2:])
            self.endtime = (int(h),int(m),int(s))
            print self.endtime
      def partsave(self):
            print self.starttime, self.endtime
            self.ui.statusbar.showMessage('Saving File')
            filename = QtGui.QFileDialog.getSaveFileName(self, 'Save Partial Data Array','./',"Data
            epic.datasaver(self.time,self.data,str(filename),self.starttime,self.endtime)
            self.ui.statusbar.showMessage('Ready')
      def savetofile(self):
            self.ui.statusbar.showMessage('Saving File')
            filename = QtGui.QFileDialog.getSaveFileName(self, 'Save Data Array','./',"Data Files (*
            epic.datasaver(self.time,self.data,str(filename))
            self.ui.statusbar.showMessage('Ready')
if __name__ == "__main__":
      app = QtGui.QApplication(sys.argv)
      window = Main()
      window.show()
      sys.exit(app.exec_())
```

8.2 Data Analysis Code

```
"""
      PriMaRy JOVE Analysis is free software:  you can redistribute it and/or
      modify it under the terms of the GNU General Public License as published by
      the Free Software Foundation, either version 3 of the License, or any later
      version.

      PriMaRy JOVE Analysis is distributed in the hope that it will be useful,
      but WITHOUT ANY WARRANTY; without even the implied warranty of
      MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
      GNU General Public License for more details.
```

```
      You should have received a copy of the GNU General Public License
      along with PriMaRy JOVE Analysis <http://www.gnu.org/licenses/>.
"""


import time
time.clock()

print 'Importing modules...'

from numpy import *
print 'NumPy imported.'

import matplotlib.pyplot as plt
print 'Matplotlib imported as plt.'

import matplotlib.ticker as tick
print 'Matplotlib.ticker imported as tick'

from scipy.fftpack import *
print 'SciPy FFT algorithms imported.'

import scipy.optimize as opt
print "SciPy Optimization algorithms imported as opt"

import pyfits as fits
print 'PyFITS imported as fits.'

import os
print 'OS imported.'

import subprocess as sub
print 'subprocess imported as sub'

import shutil
print 'importing shell utils'

import datetime
print 'Importing datetime'

print 'Importing took %s seconds' % time.clock()

class Error(Exception):
      def __init__(self,error):
            print error

#############################################################################

def nt_time(nttime):
      """Converts a NT time stamp to unix time.

            NT time is a floating point number used to represent time from
            some point in the past (the point being currently unknown.)

            Python time is a tuple of numbers which represent the data
            time in several numbers.

            Unix time is seconds since the epoch.
```

```
            Input:
                  nttime = Input time float.

            Output:
                  unix = Output float in unix time
      """
      unix = (nttime - 25569.0) * 86400.0
      return unix

##############################################################################

class opendata():
      def __init__(self,filename):
            self.filename = filename
            self.Fdata = ''
            self.header = ''
            self.data = ''
            self.time = ''
            #Firstly Check filename exists
            if not os.path.exists(self.filename):
                  raise Error("File Does not Exist")
            #Next Determine Type of file
            self.filetype = self.filename[self.filename.rfind('.'):]
            print self.filetype
            if self.filetype == '.spd':
                  self.spd()
                  self.timeconvert()
            elif self.filetype == '.csv':
                  self.csv()
                  self.timeconvert()
            elif self.filetype == '.fits' or '.fit':
                  self.import_fits()
                  self.timeconvert()
            elif self.filetype == '.txt' or '':
                  self.txtload()
                  self.timeconvert()
            else:
                  try:
                        self.txtload()
                        self.timeconvert()
                  except:
                        raise Error("Cannot open this file - Incompatible Type")

      def spd(self):
            #Convert to FITS
            self.convert()
            #Re-Set filename to FITS
            self.filename = self.filename+'.fits'
            #Import FITS
            self.import_fits()
            #Zip
            self.zipdata()

      def csv(self):
            #Read csv
            self.nttime,self.data = loadtxt(self.filename, delimiter=',')
            #self.zipdata()
```

```
	def convert(self):
		if not os.path.exists('./SPD2FITS'):
			raise Error('SPD2FITS Converter Not Found')
		else:
			#Copy Input file from input location to SPD2FITS folder
			shutil.copy(self.filename,'./SPD2FITS')
			os.chdir('./SPD2FITS')
			sub.call(['spd2fits.bat',self.filename])
			shutil.move(self.filename+'.fits',self.filename+'.fits')

	def import_fits(self):

		#Open fits file
		FF = fits.open(self.filename)
		#Get Header and check channels = 1
		header = FF[1].header
		#Get number of channels from header
		channels = header['CHANNELS']

		if not channels == 1:
			raise Error("Please supply a file with only one channel")

		self.Fdata = FF[1].data
		self.zipdata()

	def txtload(self):
		self.nttime,self.data = loadtxt(self.filename)

	def zipdata(self):
		zippy = zip(*self.Fdata)
		self.data = zippy[1]
		self.nttime = zippy[0]

	def timeconvert(self):
		print self.nttime[0]
		self.time = []
		if self.nttime[0] < 1000000:
			for i,t in enumerate(self.nttime):
				self.time.append(nt_time(t))
		else:
			self.time = self.nttime

################################################################################


def passfilter(y, filtertype = 'low', lim = 100):
	"""Filter function to remove unwanted frequencies.

	Function uses discrete FFT theory to transform from
	time domain to frequency domain.

	The unwanted frequencies are then set to zero as per the parameters input.

	Inputs:
		Array = Input array to be filtered;
		lim = Limit of the band allowed to pass;
		filtertype = Type of filter, either "high" or "low"
			"low" = All frequencies above lim will be set to zero
			"high" = All frequencies below lim will be set to zero
```

```
        Output:
                newy = The array y filtered of unwanted frequencies
        """

        if lim > len(y)/2.0:
                raise Error('\nError:  lim is too high for this size array!')

        yfft = fft(y)                                           # FFT the input data.

        if filtertype.capitalize() == 'Low':
        # Will filter using low pass frequency filter.

                yfft[lim:-lim-1] = 0                    # Filters the data as per lim.
                newy = ifft(yfft)                         # Inverse FFT the filtered data.

        elif filtertype.capitalize() == 'High':
        # Will filter using a high pass frequency filter.

                yfft[:lim],yfft[-lim-1:] = 0, 0 # Filters the data using lim.
                newy = ifft(yfft)                           # Inverse FFT the filtered data.
        else:
                raise Error('\nNot a correct pass filter type!')

        return newy

##############################################################################

def smooth(x, window_len=11 ,window='hanning'):
        """Window smoothing function.

        This method is based on the convolution of a
        scaled window with the input signal.

        The signal is prepared by introducing reflected copies of the signal
        (with the window size) in both ends so that transient parts are minimized
        in the begining and end part of the output signal.

        input:
                x = Input signal;
                window_len = The dimension of the smoothing window,
                                        should be an odd integer greater than 2;
                window = The type of window from 'flat', 'hanning',
                                'hamming', 'bartlett', 'blackman'
                        flat window will produce a moving average smoothing.

        output:
                y = Smoothed signal
        """

        if x.ndim != 1:
                raise Error("smooth only accepts 1 dimension arrays.")

        if x.size < window_len:
                raise Error("Input vector needs to be bigger than window size.")

        if (not isodd(window_len)) or (window_len < 3):
                raise Error("Window length is either below 3 or not even.")
```

```
        if not window in ['flat', 'hanning', 'hamming', 'bartlett', 'blackman']:
                raise Error("Window is out of 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'")


        s = r_[2*x[0]-x[window_len:1:-1],x,2*x[-1]-x[-1:-window_len:-1]]

        if window == 'flat': # moving average
                w = ones(window_len,'d')
        else:
                w = eval(window+'(window_len)')

        y = convolve(w/w.sum(),s,mode='same')

        return y[window_len-1:-window_len+1]

##############################################################################

def spd2fits(filename):
        """Function which converts a Skypipe data set
        into a FITS file using bash commands.

             Uses the python subprocess module to run windows prompt commands"""

        cwd = os.getcwd()

        spd2fitsdir = './SPD2FITS/'

        sub.call(spd2fitsdir + 'spd2fits ' + filename)

        print 'FITS file created.\n'

##############################################################################

def isodd(num):
        """Function which checks whether a number is even or odd.

        Input:
                num = Integer

        Output:
                Boolean output, if number is off then output is True.
        """

        if num != int(num):
                raise Error("Number is not integer!")

        return num & 1 and True or False

##############################################################################

def binner(time, data, bintime = 1.0):
        """Bins an input array and also creates
        a time-array to reflect this binning.

        Inputs:
                time = Input time-array;
                data = Input data-array;
                bintime = Time length of bins.
```

```
	Output:
		fx = Binned time array;
		fy = Binned data array.

	Data binning is done on a smart basis, the array indexes are calculated
	from the time array in such a way that the time of bins is always constant
	even if the number of elements per bin may change.  The last data bin,
	unless it factors in completely, is simply the rest of the array averaged.

	The time-array is binned in such a way that the first time of the binned
	data-array is chosen to be the time for that bin.

	"""

	if len(time) != len(data):
		raise Error("Arrays are not same size")

	N = len(time)                                                          # Number

	time_bin_list, data_bin_list = [], []                          # Empty lists

	j = 0                                                                  # Initial

	timeunix = time

	#for i,val in enumerate(time):
	#	timeunix[i] = nt_time(val)

	timex = timeunix - timeunix[0]

	for i,val in enumerate(timex):
		if timex[i] > (timex[j] + bintime) or i == N:
			time_bin_list.append(timeunix[j])
			data_bin_list.append(mean(data[j:i]))
			j = i

	time_bin = array(time_bin_list)
	data_bin = array(data_bin_list)

	return time_bin, data_bin

##############################################################################

def normal(p, x):
	"""Calculates gaussian function from a set of parameters, p, and data, x.

	Uses the standard gaussian function and an additional vertical offset
	to return an array when given parameters and x data.

	!!!!REQUIRED BY RESIDUALS AND GAUSSFIT AMONGST OTHERS!!!!

	Input:
		p = List of parameters
			p[0] = a = Height of peak from base;
			p[1] = b = Centre of peak;
			p[2] = c = Width parameter;
			p[3] = d = Vertical offset
		x = Input array of floats
```

```
        Output:
                y = Gaussian array.
        """

        x = real(x)

        a, b, c, d = p
        y = (a * exp(-(x - b)**2 / (2.0 * c**2))) + d

        return y

################################################################################

def residuals(p, y, x):
        """Calculates the residuals between fit gaussian and data.

        Calculates the difference between gaussian fit data and measured data.

        !!!!REQUIRED BY GAUSSFIT!!!!

        Input:
                p = List of parameters
                        p[0] = a = Height of peak from base;
                        p[1] = b = Centre of peak;
                        p[2] = c = Width parameter;
                        p[3] = d = Vertical offset
                y = Measured data;
                x = Array with which y is measured against;

        Output:
                err = Difference between arrays.
        """

        err = y - normal(p, x)
        return err

################################################################################

def gaussfit(p0, y):
        """Function which fits a gaussian function to y and returns the parameters
        needed to recreate the fit.

        Function uses the Levenberg-Marquardt algorithm to fit a gaussian function
        to input data y.  LMA is an algorithm which uses non-linear regression
        to vary the parameters to fit the data.

        Input:
                p0 = Starting parameters
                y = Measured data

        Output:
                pnew = Fitted parameters which can be plugged into normal(p,x)

        """
        z = range(len(y))

        output = opt.leastsq(residuals, p0, args = (y, z))
```

```
        pnew = output[0]

        return pnew

##############################################################################

def peakfinder(arr, limit):
        """Function designed to automatically find peaks in a data set and
        fit a gaussian curve to them.

        Function will search through measured data looking for peaks, after
        finding these peaks it will then proceed to use gaussfit to attempt
        to fit peaks to them and then save all the parameters to an array
        to be output to the user.

        Input:
                arr = Measured data array;
                limit = Float used to choose what is deemed as a peak.
                                A peak is registered if it's tip is greater than
                                limit*standard_deviation of the data set.

        Output:
                parameters = List of lists containing parameter
                data for all peaks found.

        """

        arr = real(arr)       # Removes are imaginary components of the measured data.

        arrmean = mean(arr)# Calculates average of measured data.
        arrstd = std(arr)   # Calculates standard deviation of measured data.

        peaks = []
        parameters = []

        # for loop will iterate through the data array and search for any peaks.
        # A peak is definsed as a point of inflexion limit*arrstd above arrmean.
        for (i, val) in enumerate(arr):
                if i == len(arr) - 2:     # Stops the iteration if end of array is reached.
                        break
                elif arr[i] < arr[i+1] and arr[i+1] > arr[i+2] and arr[i+1] > arrmean + (limit * arrstd
                        peaks.append(i+1)     # Appends the point of inflexion to a list

        print 'Peaks found at %s' % peaks

        # for loop iterates over the points of inflexion and finds
        # the start and end of the peak.

        # The start and end are defined as points of inflexion.
        for peakmid in peaks:

                # Finds end of the peak.
                for (i, val) in enumerate(arr[peakmid:]):
                        diff = arr[peakmid+i+1] - arr[peakmid+i]
                        if diff > 0:
                                peakend = i
                                break

                # Finds the start of the peak.
```

```
            for (i, val) in enumerate(reversed(arr[peakmid:])):
                  diff = arr[peakmid+i+1] - arr[peakmid+i]
                  if diff > 0:
                        peakstart = i
                        break

            # Creates a new array so only fitting
            # gaussian curve over peak data range.
            ypeak = arr[peakmid - peakstart : peakmid + peakend]

            # Creates good guesses for starting parameters of the peak.
            pguess = [max(ypeak) - min(ypeak), len(ypeak)/2, len(ypeak)*0.1, min(ypeak)]

            pfit = gaussfit(pguess, ypeak)

            # Defines central position as middle of peak (Instead of index of ypeak)
            pfit[1] = peakmid

            # Appends the parameters to a list to be output to user.
            parameters.append(pfit)

      return parameters

##############################################################################

def peakrammer(array1,array2):
      if len(array1) != len(array2):
            raise
      arr = zeros(len(array1))
      for i in range(len(array1)):
                  a = array1[i]
                  b = array2[i]
                  if math.isnan(a) and math.isnan(b):
                        arr[i] = nan
                  elif math.isnan(a) and not math.isnan(b):
                        arr[i] = b
                  elif not math.isnan(a) and math.isnan(b):
                        arr[i] = a
                  elif not math.isnan(a) and not math.isnan(b):
                        if a > b:
                              arr[i]=a
                        elif a < b:
                              arr[i]=b

      return arr

##############################################################################

def peakplotter(y, para):
      """Plots data and any peaks which have been found using peakfinder.

      Will plot the data and then iterate through para to
      plot any peaks which have been found.

      Inputs:
            x = Time-array;
            y = Smoothed data;
            para = Parameters of fitted peaks
```

```
        Output:
                No output, just a graphing function.
        """

        z = range(len(y))

        for i,p in enumerate(para):
                exec('array%s = normal(p, z)' % str(i))
                for j,val in enumerate(eval('array%s' % str(i))):
                        if val < 1.01*p[3] or val < 0.0:
                                exec('array%s[%s] = nan' % (str(i), str(j)))

        N = len(para)                                                   # Number of peaks

        for i in range(0,N-1):
                if i == N-1:
                        break
                exec('array%s = peakrammer(array%s, array%s)' % (str(i+1), str(i+1), str(i)))

        exec('result = array%s' % str(N-1))

        return result



##############################################################################

def difference(y0, y1):
        """Simple function which calculates the difference between two arrays.

        Input:
                y0 = First array;
                y1 = Second array.

        Output:
                diff = Difference.
        """

        size0, size1 = size(y0), size(y1)                       # Calculates array sizes.

        # Checks for irregularities in array sizes.
        if N0 != N1:
                raise "Arrays not same size!"

        diff = y0 - y1                                                  # Calculate difference

        return diff

##############################################################################

def timegrinder(t):
        """Function to edit a time array to reflect
        change in seconds since beginning.

        Input:
                t = Time array

        Output:
                tnew = New time array
```

```
        """

        start = t[0]

        tnew = t - start

        return tnew

#############################################################################

def datasaver(arrt, arry, filename, start = 0, end = -1):
        """Function which saves time/data to a text file

        Input:
                t = Time array;
                y = Data array;
                filename = the full save path, or relative save path with filename;
                start = A time to slice the array to, or 0 to start at the begging;
                end = A time to slice the array to, or -1 to stop at the end;

        """

        # Make sure filename is not blank
        if filename ==   '':
                raise Error('Please speicfy a filename')

        # Get filetype from extension on filename
        filetype = filename[filename.rfind('.'):]
        print start, end

        #Convert times to index
        if start == 0:
                startindex = 0
        if end == -1:
                endindex = -1
        else:
                startindex,endindex = timetoindex(arrt,start,end)

        # Depending on filetype save to file
        if filetype == '.csv':
                savetxt(filename,[arrt[startindex:endindex],arry[startindex:endindex]],delimiter=',')
        elif filetype == '.txt' or filetype == '':
                savetxt(filename,[arrt[startindex:endindex],arry[startindex:endindex]],delimiter=' ')
        elif filetype == '.fits':
                raise Error('FITS support not avible at the moment!')
        else:
                raise Error('Incorrect File Type')

def timetoindex(arrt,start,end):
        """ This function will convert a time
        to a index for the time and data arrays """
        timestep = arrt - arrt[0]

        databegin = time.gmtime(arrt[0])
        beginsec = (databegin[3] * 3600.0) + (databegin[4] * 60.0) + (databegin[5])

        # Number of seconds since midnight for start and ending of saving range.

        startsec = (start[0] * 3600.0) + (start[1] * 60.0) + (start[2])
```

```
        endsec = (end[0] * 3600.0) + (end[1] * 60.0) + (end[2])

        # Takes into account when the saving range crosses over the midnight mark.
        startstep = startsec - beginsec
        if startstep < 0.0:
                startstep = startstep + 86400.0

        endstep = endsec - beginsec
        if endstep < 0.0:
                endstep = endstep + 86400.0

        # Iterates through the time array and sets start/end index.

        for i,val in enumerate(timestep):
                if val < startstep:
                        startindex = i
                if val < endstep:
                        endindex = i

        return startindex, endindex
#################################################################################

def axesformatter(x, pos=None):
        """ This if called as matplotlib.ticker.FuncFormatter()
        will format the x axis with string time.  """
        struct_time = time.gmtime(x)
        stringtime = time.strftime("%H:%M:%S",struct_time)
        return stringtime

#################################################################################
```

9 References

[1] http://upload.wikimedia.org/wikipedia/commons/3/3d/reed-solomon2.png.

[2] Matthew Watts Frances Pope. Investigation into io storms using radio jove. *Advanced Techniques PH34040*, 2009.

[3] William H Press. *Numerical Recipes*. Cambridge University Press, 2007.