Final Project CEE 505

Cyndi Lopez

Project Description

The purpose of this project was to create a database of ground motions and to create a GUI for selecting ground motions and plotting acceleration, velocity, displacement, response spectra; and for filtering acceleration data.

Introduction

There are a few databases that currently exist which are great for selecting ground motions and matching response spectra to a target spectrum. However, those databases are primarily for areas dominated by shallow, crustal earthquakes. Many parts of Washington State are affected by subduction zone earthquakes. The ground motions in this database are well suited for design in Washington State since subduction zone earthquakes are included. The ground motions are divided by those best-suited for a certain region in Washington. The region identification is shown in Figure 1.

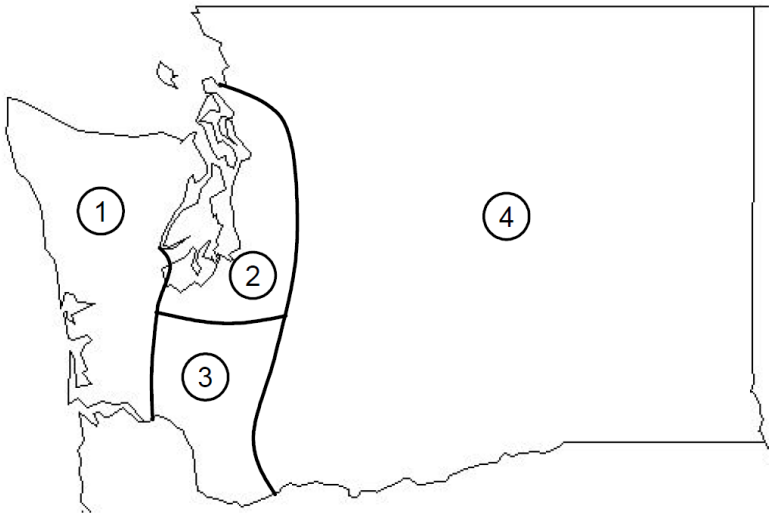

Figure 1 Regions used to develop ground motion databases

Database Organization

A database for 425 ground motions was created in SQLite. Tables were created for each of the ground motions with the ground motion filename as the table name, and columns of integer IDs, time, and acceleration. The following tables were also created: motions, regions, groups,

events, and stations. The tables were organized as follows:

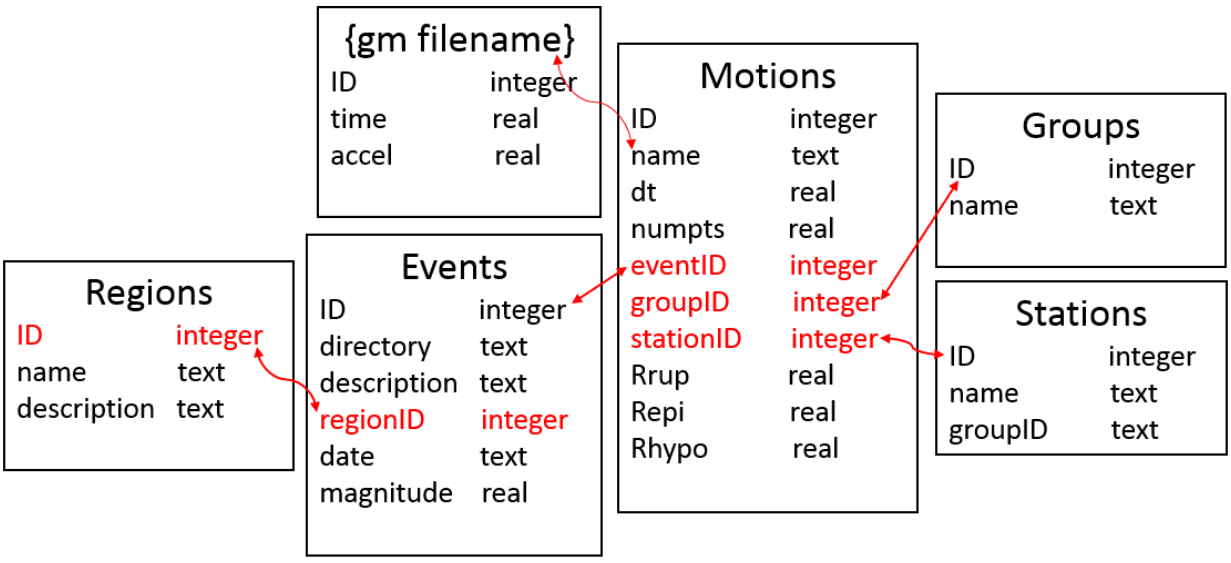

Figure 2: Database organization

Ground Motion Selection and Plotting

The window opens with a screen (Figure 3) that contains 3 different dialog group boxes: library of motions, filtering, and plots. The user selects the region, type, and magnitude range, and selects “Get GM” to get all the ground motions that satisfy the selections (Figure 4). Upon clicking “Plot”, the acceleration, displacement, and velocity time histories, and response spectrum is plotted (Figure 5).The acceleration data can be filtered with a Butterworth filter by checking the “Butterworth” box and selecting either a low pass or high pass filter. “Refresh” is clicked to generate the filtered time history plots (Figure 6). The plot can be saves as shown on Figures 7 and 8, by going to File, Save Plot, and following the prompts on the dialog box that pops up.

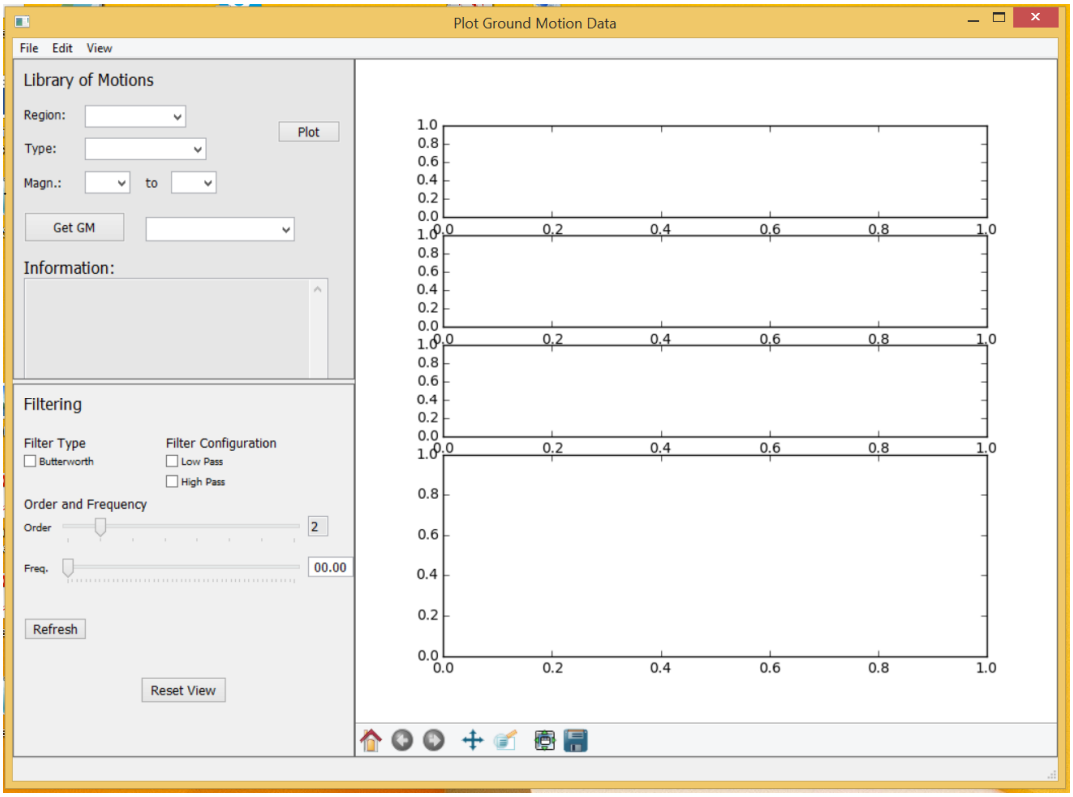

Figure 3: Initial Input Screen

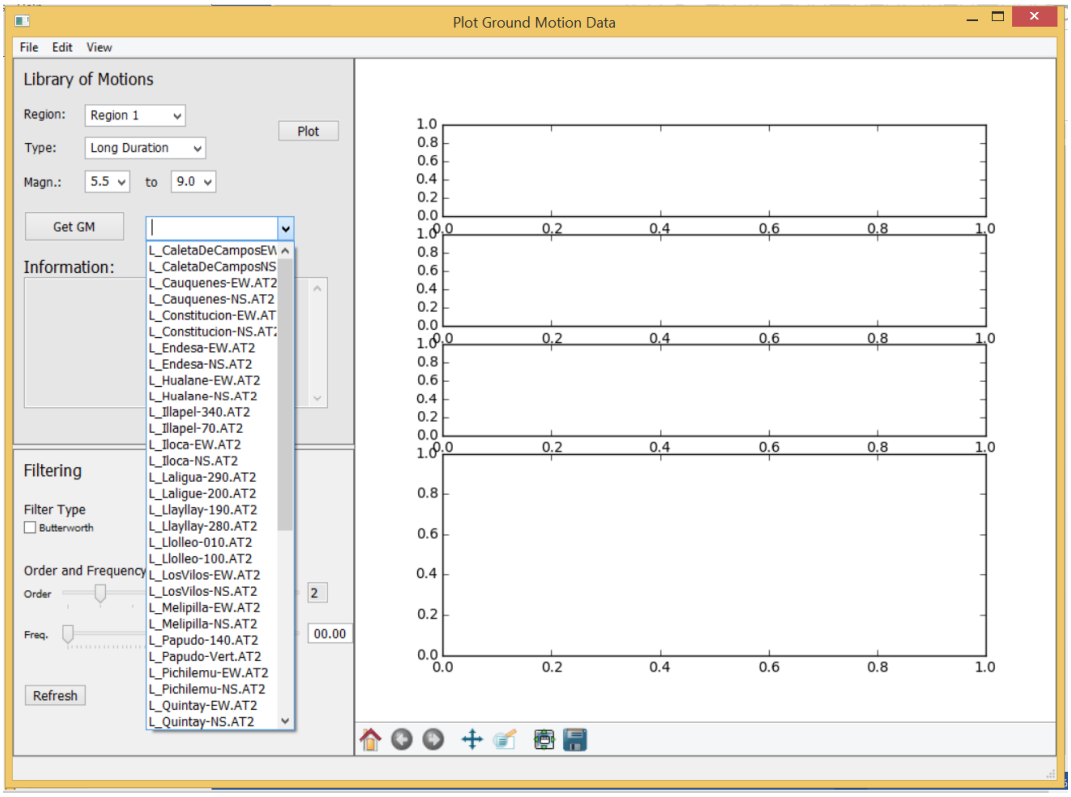

Figure 4: Selecting Ground Motion

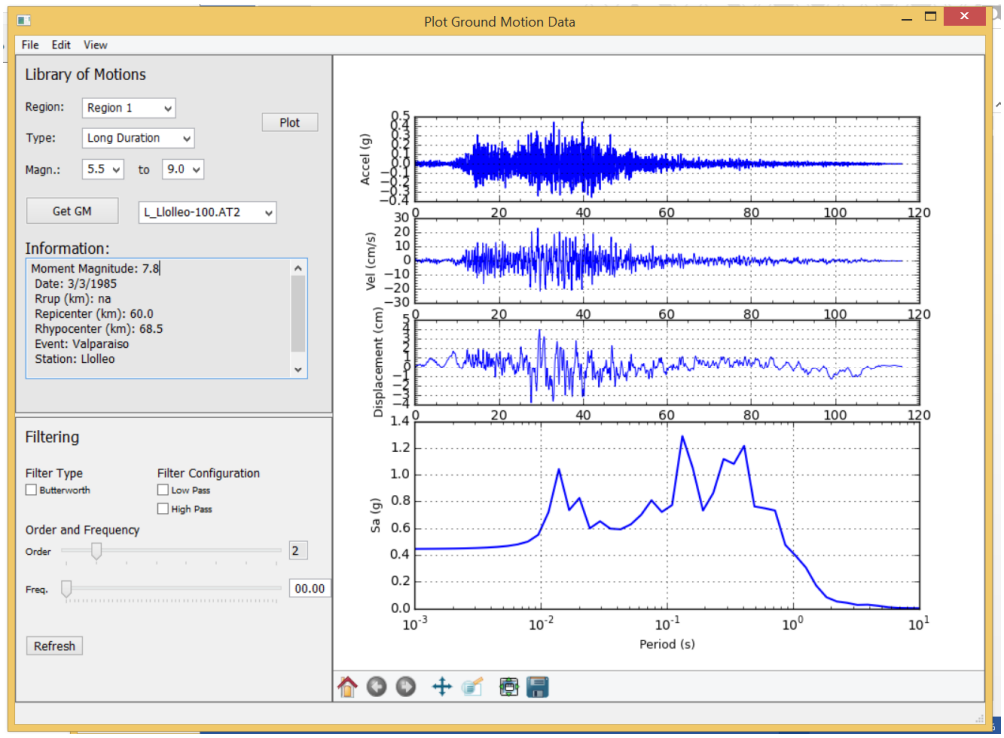

Figure 5: After Selecting Ground Motion and Plotting

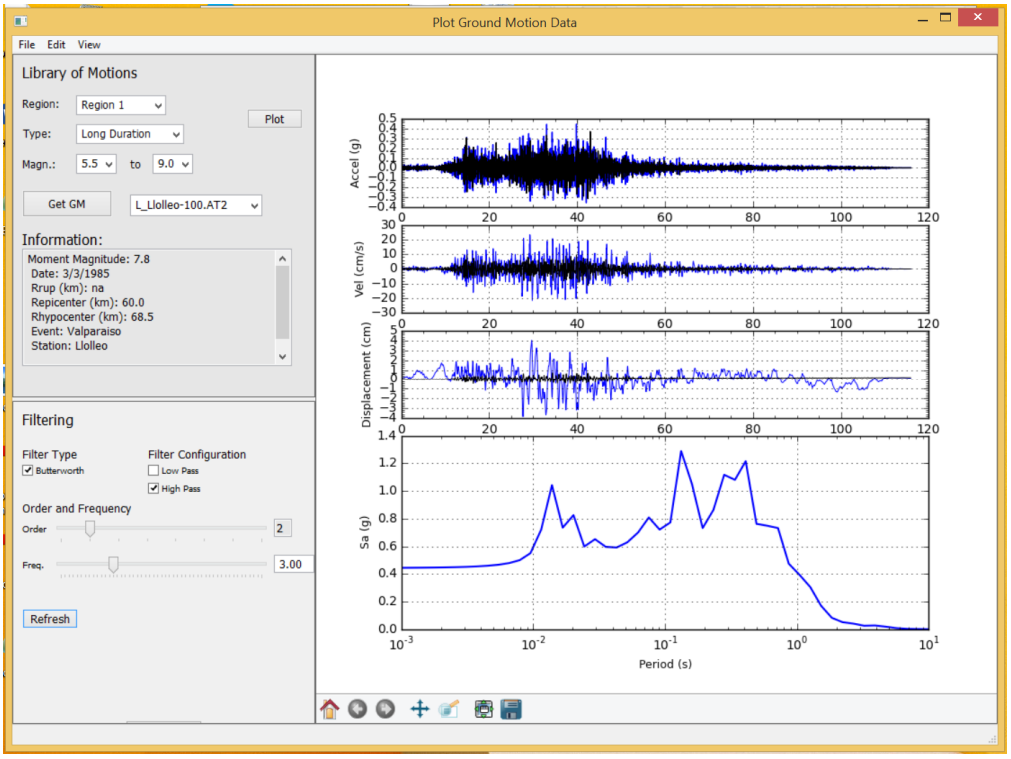

Figure 6: Filtering Ground Motion Data

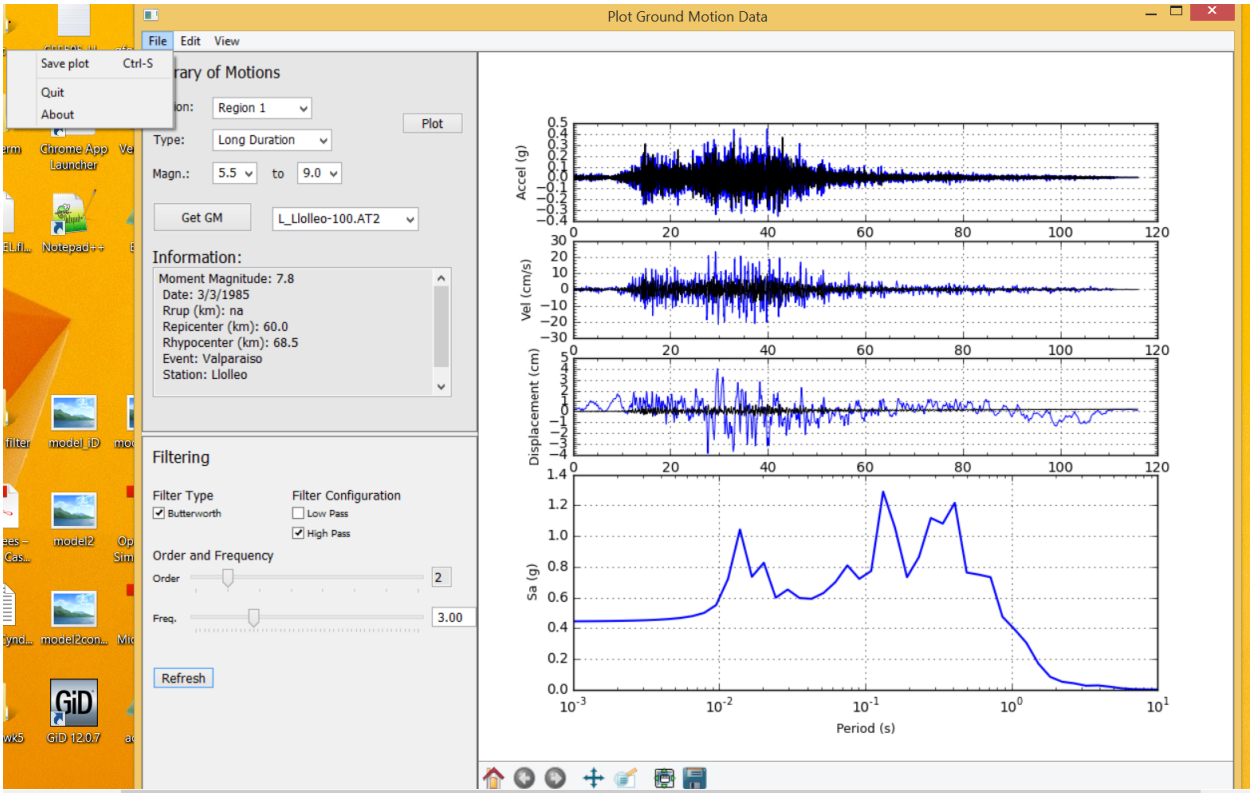

Figure 7: File Menu

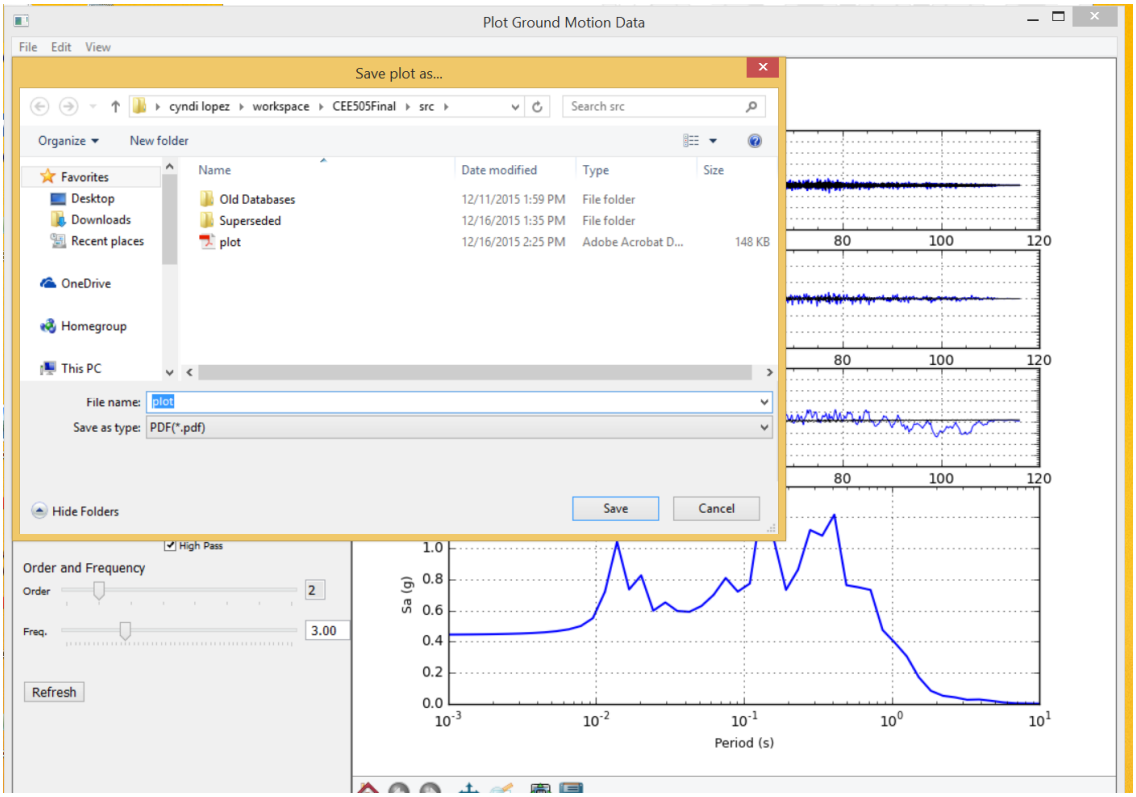

Figure 8: Saving Plot

Appendix A
GUI Algorithm

The python code is attached and sufficiently annotated. However, I will give a general overview of the code here. The wx package along with other packages such as matplotlib and various functions of scipy are imported. A python file, SQLqueries, is also imported. This is where all the select statements that are used to obtain information from the database are stored.

A class is created which will create the GUI window. It takes as input a frame. The frame is constructed, which creates the window.

```
class MyAppWindow(wx.Frame):
    '''
    classdocs
    '''

    def __init__(self, title='Plot Ground Motion Data'):
        '''
        Constructor
        '''

        wx.Frame.__init__(self, None, ID_SELF, title, size=(1020,960))
```

The database is defined and connected to and all the ground motion names are fetched from the database,

```
        self.database = "WSGMDB.db"
        self.db = dbi.connect(self.database)

        # get all gm from databse
        gm = getGMnames(self.db)
```

The menubar where File, Edit, and View are menus accessible at the top of the page.

```
filemenu = wx.Menu()
        filemenu.Append(ID_FILE3,"&Save plot\tCtrl-S", "Save plot to file")
        self.Bind(wx.EVT_MENU, self.OnSave)
        filemenu.AppendSeparator()
        filemenu.Append(ID_QUIT,"&Quit","Quit the Application")
        filemenu.Append(ID_FILE5,"&About","Find fastest Path")

        editmenu = wx.Menu()
        editmenu.Append(ID_EDIT1,"&Copy","Copy selection to buffer")
        editmenu.Append(ID_EDIT2,"C&ut","Cut selection")
        editmenu.Append(ID_EDIT3,"&Paste","Paste buffer")

        viewmenu = wx.Menu()
        viewmenu.Append(ID_VIEW1,"&Shrink","Reduce size to half")
        viewmenu.Append(ID_VIEW2,"&Maximize","Maximize window")
```

```
        viewmenu.Append(ID_VIEW3,"&Full screen","Switch to full screen mode")
        viewmenu.Append(ID_VIEW4,"&Reset","Reset View")

        menubar = wx.MenuBar()
        menubar.Append(filemenu,"&File")
        menubar.Append(editmenu,"&Edit")
        menubar.Append(viewmenu,"&View")

        self.SetMenuBar(menubar)
```

The menus have events associated with them, which are defined. When the event happens, which in this GUI are driven by a click, a function is called which tells the window what to do. For example, upon clicking “About” in the File menu, a dialog box appears with information about the GUI.

```
        wx.EVT_MENU(self, ID_QUIT, self.onQuit)
        wx.EVT_MENU(self, ID_VIEW1, self.onViewShrink)
        wx.EVT_MENU(self, ID_VIEW2, self.onViewMaximize)
        wx.EVT_MENU(self, ID_VIEW3, self.onViewFullScreen)
        wx.EVT_MENU(self, ID_VIEW4, self.onViewReset)
        wx.EVT_MENU(self, ID_FILE5, self.onAbout)

def onQuit(self, event):
        msg = "Good bye"

        dlg = wx.MessageDialog(self,
                               msg,
                               "Exit",
                               wx.OK | wx.ICON_INFORMATION)

        dlg.ShowModal()
        dlg.Destroy()
        self.Close()

  def onViewShrink(self,e):
        msize = []
        for i in self.GetSize():
            msize.append(int(0.5*i))
        self.SetSize(msize)


   def onAbout(self,event):
        info = wx.AboutDialogInfo()
        info.SetName("Simple GUI")
        info.SetVersion('1.0')
        info.SetDescription("Plots acc-vel-disp time histories and response
spectra.")
        info.AddDeveloper("Cyndi")
        wx.AboutBox(info)

    def onViewReset(self,e):
        self.ShowFullScreen(False)
        self.SetSize((768,430))
        self.Center()
```

```
    def onViewFullScreen(self,e):
        self.ShowFullScreen(True)

    def onViewMaximize(self,e):
        self.Maximize(True)

    def OnSave(self,e):
        file_choices = "PDF(*.pdf)|*.pdf"

        dlg = wx.FileDialog(
            self,
            message="Save plot as...",
            defaultDir=getcwd(),
            defaultFile="plot.pdf",
            wildcard=file_choices,
            style=wx.SAVE)

        if dlg.ShowModal() == wx.ID_OK:
            path = dlg.GetPath()
            self.canvas.print_figure(path, dpi=self.dpi)
            self.flash_status_message("Saved to %s" % path)
```

The panels are defined using wx.panel. The parent of the panels is the frame. Two panels are created, and the left panel is split horizontally using the SplitterWindow class. Sizers are set so that the right panel can expand as the window size changes but the left panels remain fixed even as the window size changes.

```
      # create panels; self.panel and panel3 are the left panel and panel 2 is the
right panel
        splitter = wx.SplitterWindow(self, wx.ID_ANY,style = wx.SP_BORDER)
        self.panel = wx.Panel(splitter,ID_PANEL1,style=wx.BORDER)
        self.panel.SetBackgroundColour([230,230,230])

        self.panel3 = wx.Panel(splitter, ID_PANEL3,style=wx.BORDER,size = (360,400))
        splitter.SplitHorizontally(self.panel, self.panel3)

        panel2 = wx.Panel(self, ID_PANEL2, size=(680,960),style=wx.BORDER)
        panel2.SetBackgroundColour([230,230,230])


        # sizers for panels
        hsizer1 = wx.BoxSizer(wx.HORIZONTAL)
        hsizer1.Add(splitter, 0, wx.EXPAND)
        hsizer1.Add(panel2, 2, wx.EXPAND)
        vbox1 = wx.BoxSizer(wx.VERTICAL)
        vbox1.Add(hsizer1, 1, wx.EXPAND)
        self.SetSizer(vbox1)
```

The figures were then created using matplotlib and Figure. Subplots were added to the figure for each time history and response spectrum. The figure was added using FigCanvas which connects matplotlib to wxpython and draws the figure on the parent specified, which in this case is panel2.

```
        #create figures
          gs = gridspec.GridSpec(5,1)
          self.dpi = 100
          self.fig = Figure(figsize=(9,8.0), dpi=self.dpi,facecolor='white')
          self.canvas=FigCanvas(panel2,-1,self.fig)
          self.axes = self.fig.add_subplot(gs[0])
          self.axes2 = self.fig.add_subplot(gs[1])
          self.axes3 = self.fig.add_subplot(gs[2])
          self.RP = self.fig.add_subplot(gs[3:])

          # add toolbar from matplot lib
          self.toolbar = NavigationToolbar(self.canvas)
```

The sizer is set up so that the plot takes up all of panel2 and changes in size as the window changes in size.

```
          #sizers for figures
          vsizeC = wx.BoxSizer(wx.VERTICAL)
          vsizeC.Add(self.canvas,1,wx.EXPAND|wx.ALL)
          vsizeC.Add(self.toolbar, 0, wx.EXPAND)
          panel2.SetSizerAndFit(vsizeC)
```

The static label texts are then defined using the StaticText class. The parent, label, and position are inputs used by this class.

```
          # create all Static Text labels
          self.lbl1 = wx.StaticText(self.panel,label = "Library of Motions", pos =
(10,10))
          self.lbl2 = wx.StaticText(self.panel,label="Region:",pos = (10,47))
          self.lbl3 = wx.StaticText(self.panel,label= "Type:", pos = (10,80))
          self.lbl4 = wx.StaticText(self.panel,label = "Magn.:", pos = (10,114))
          self.lbl5 = wx.StaticText(self.panel,label="to",pos = (130,114))
          self.lblinfo = wx.StaticText(self.panel,label="Information: ",pos = (10,195))
          self.lblf = wx.StaticText(self.panel3,label = "Filtering", pos=(10,10))
          self.lblf2 = wx.StaticText(self.panel3,label = "Filter Type",pos=(10,50))
          self.lblf3 = wx.StaticText(self.panel3,label = "Filter Configuration",
pos=(150,50))
          self.lblf4 = wx.StaticText(self.panel3,label = "Order and Frequency", pos =
(10,110))
          self.lblf5 = wx.StaticText(self.panel3,label = "Order",pos=(10,135))
          self.lblf6 = wx.StaticText(self.panel3,label = "Freq.",pos=(10,175))
```

These are also adjusted for font size using the Font Class. The pixelsize, family, style, and weight are used as inputs for this class.

```
          # define fonts
          self.font = wx.Font(10,wx.DEFAULT,wx.NORMAL,wx.NORMAL)
          self.fontLarge = wx.Font(12,wx.DEFAULT,wx.NORMAL,wx.NORMAL)
          self.fontSmall = wx.Font(9,wx.DEFAULT,wx.NORMAL,wx.NORMAL)

          self.lbl1.SetFont(self.fontLarge)
          self.lbl2.SetFont(self.fontSmall)
          self.lbl3.SetFont(self.fontSmall)
          self.lbl4.SetFont(self.fontSmall)
          self.lblinfo.SetFont(self.fontLarge)
```

```
        self.lblf.SetFont(self.fontLarge)
        self.lblf2.SetFont(self.font)
        self.lblf3.SetFont(self.font)
        self.lblf4.SetFont(self.font)
        self.lbl5.SetFont(self.fontSmall)
```

Lists for the comoboxes are initialized as empty lists. A list of regions and magnitudes is defined manually, although in the future, simple select statements can be implemented to call these choices from the tables.

```
        # initialize lists for dropdown boxes
        regionsList = []
        groupsList = []
        magn1List = []
        magn2List = []
        regionslist = ["Region 1", "Region 2", "Region 3", "Region 4", "ALL"]
        groupslist = ["Long Duration","Near Fault", "Other","ALL"]
        magn1list = [str(i) for i in linspace(5.5,8.5,7)]
        magn2list = [str(i) for i in linspace(6,9,7)]
```

Combo boxes are created using the class ComboBox, which takes as input the parent, size, list of choices, and position.

```
        # create comboboxes
        self.cbGM = wx.ComboBox(self.panel,size = wx.DefaultSize, choices = [],pos =
(130,155))
        self.cbGM.SetFont(self.font)
        self.cbR = wx.ComboBox(self.panel,size = (100,15),choices = regionsList, pos
= (70,45))
        self.cbTy = wx.ComboBox(self.panel,size = (120,15), choices = groupsList, pos
= (70,77))
        self.cbM1 = wx.ComboBox(self.panel,size = (45,15), choices = magn1List, pos =
(70,110))
        self.cbM2 = wx.ComboBox(self.panel,size = (45,15),choices = magn2List, pos =
(155,110))
```

A function called widgetMaker is called to bind the callback function to the event wx.EVT_COMBOBOX as shown below.

```
        # call function to bind chosen values from dropdown
        self.widgetMaker(self.cbR,regionslist,self.onChoiceReg)
        self.widgetMaker(self.cbTy,groupslist,self.onChoiceGrp)
        self.widgetMaker(self.cbM1,magn1list,self.onChoiceMagMin)
        self.widgetMaker(self.cbM2,magn2list,self.onChoiceMagMax)
        self.widgetMaker(self.cbGM,gm,self.onSelect)



    def widgetMaker(self, widget, objects,function):
        """ bind combobox selection to function"""
        for obj in objects:
```

```
            widget.Append(obj)
        widget.Bind(wx.EVT_COMBOBOX, function)
```

The function onChoiceReg clears the ground motion combo box and gets the region selected using the method GetStringSelection. It gets the integer associated with the string selected using getGroupID, which is located in the SQLqueries.py file. This function is not shown here for brevity.

The ground motion combo box is the one that returns the ground motions that are specific to the region, type, and magnitude range selected. Whenever a new region, magnitude, or type is selected, the ground motion combo box is cleared so that the user doesn't accidentally select a ground motion from the old list.

```
def onChoiceReg(self,event):
        self.cbGM.Clear()
        self.region = self.cbR.GetStringSelection()
        self.region = self.region.replace(" ","")
        self.region = getRegionID(self.db,self.region)
```

The function onChoiceGrp is similar to the onChoiceReg function. It gets the type selected by the user and clears the ground motion combo box whenever a new item from the dropdown list is selected. It calls on the function getGroupID, which is located in the SQLqueries.py file, to fetch the integers associated with the group "name" selected by the user. This makes selecting the motions from the motion table easier since the groupID column is stored as an integer in the "Motions" table.

```
    def onChoiceGrp(self,event):
        self.cbGM.Clear()
        self.grp = self.cbTy.GetStringSelection()
        self.grp = getGroupID(self.db,self.grp)
```

The following two functions are used to select the minimum magnitude value and the maximum magnitude value, forming a range. First, the ground motion combo box is cleared. The reason for this was talked about above. The GetStringSelection method obtains the strings selected by the user from the dropdown list. In this case, if the minimum and maximum values are the same, a dialog box pops up displaying an error message. These two values cannot be the same because then a range would not be formed and there would be an error when selecting ground motions from the database.

```
    def onChoiceMagMin(self,event):
        self.cbGM.Clear()
        self.mag1 = self.cbM1.GetStringSelection()
        self.mag1 = float(self.mag1)
        try:
            if self.mag1 == self.mag2:

                msg = "ERROR: Magnitude range cannot be the same."

                dlg = wx.MessageDialog(self,
                                       msg,
                                       "Error",
                                       wx.OK | wx.ICON_INFORMATION)
```

```
                dlg.ShowModal()
                dlg.Destroy()
            except:
                pass

    def onChoiceMagMax(self,event):
        self.mag2 = self.cbM2.GetStringSelection()
        self.mag2 = float(self.mag2)
        try:
            if self.mag1 == self.mag2:
                msg = "ERROR: Magnitude range cannot be the same."

                dlg = wx.MessageDialog(self,
                                       msg,
                                       "Error",
                                       wx.OK | wx.ICON_INFORMATION)

                dlg.ShowModal()
                dlg.Destroy()
            except:
                pass
```

The “Get GM” button is then defined, where GM stands for ground motion.

```
        #create "get groundmotion" button
        self.GMbutton = wx.Button(self.panel,ID_BUTTON6,label = "Get GM", pos =
(10,150),size = (100,30))

        #change font of ground motion button
        self.GMbutton.SetFont(self.font)
```

The event for the button is generated.

```
        wx.EVT_BUTTON(self,ID_BUTTON6,self.onSelectGM)
```

The callback function onSelectGM selects a ground motion through an SQL query. The getSelectGM is located in the SQLqueries.py file and filters the ground motions per the group, region, and magnitude range selected by the user.

```
 def onSelectGM(self,e):
        gm = getSelectGM(self.db,self.region, self.grp,self.mag1,self.mag2)
        for i in gm:
            self.cbGM.Append(i)
        self.cbGM.SetFont(self.font)
```

Because the getSelectGM function and the SELECT statements it uses are important, the function is shown below. SQL SELECT statements are used that are particular to the region, type, and magnitude selected. If “ALL” is selected, SELECT statements are used that ignore the region ID or type ID condition.

```
def getSelectGM(db,region,group,magn1,magn2):
    cu=db.cursor()
    if region == "ALL":
        if group == "ALL":
            call = """SELECT m.name
```

```
            FROM Motions as m,
                events as e
            WHERE m.eventID = e.ID
                and e.magnitude >= {}
                and e.magnitude <= {}
            ;
            """
            cu.execute (call.format(magn1,magn2))
        else:
            call = """SELECT m.name
            FROM Motions as m,
                events as e,
                groups as g
            WHERE m.eventID = e.ID
                and e.magnitude >= {}
                and e.magnitude <= {}
                and g.ID = {}
                and m.groupID = g.ID
            ;
            """
            cu.execute(call.format(magn1,magn2,group))

    elif group == "ALL":
        call = """SELECT m.name
        FROM Motions as m,
            events as e,
            regions as r
        WHERE m.eventID = e.ID
            and e.magnitude >= {}
            and e.magnitude <= {}
            and r.ID = {}
            and e.regionID = r.ID
        ;
        """
        cu.execute(call.format(magn1,magn2,region))

    else:
        call = """SELECT m.name
            FROM motions as m,
                regions as r,
                groups as g,
                events as e
            WHERE r.ID = {}
                and e.regionID = r.ID
                and g.ID = {}
                and m.groupID = g.ID
                and m.eventID = e.ID
                and e.magnitude >= {}
                and e.magnitude <= {}
                ;
                """
        cu.execute(call.format(region,group,magn1,magn2))
    gm = cu.fetchall()
    gm = [i[0] for i in gm]
    return gm
```

The returned ground motions are added to the ground motion combo box using the method Append.

The “Plot” button and the event associated with clicking on this button are then defined. In this case, self.Bind rather than self.button.Bind works because the Bind class is used directly after the button is defined.

```
        # create Plot button
        self.button = wx.Button(self.panel,ID_BUTTON3,label="Plot", pos =
(260,60),size = wx.DefaultSize)
        self.button.SetFont(self.font)

        # define what Plot button should do when clicked
        self.Bind(wx.EVT_BUTTON,self.PlotWhenClicked)
```

The callback function called, PlotWhenClicked, calls the draw function. Try and except is used here to prevent an error message from popping up when the user clicks plot without selecting a region, type, and/or magnitude.

```
    def PlotWhenClicked(self,event):
        try:
            self.draw(self.gm)
        except:
            pass
```

The draw function takes the ground motion selected, and gets the time, acceleration, time step, and number of points from the database using the gettime, getacc, getDT, and getnumpts functions. These are defined in the SQLqueries.py file and use SQL SELECT statements to obtain the correct information. The acceleration data fetched is then integrated to get velocity using the cumptrapz function found in scipy. Displacement is obtained from the velocity also using cumptrapz. The acceleration, velocity, and time histories are plotted using matplotlib. The plots are formatted to show minor ticks, major gridlines. The y-axes are also labeled, and linewidth adjusted to 0.7. The response spectra is then calculated using the calcResponseSpectra function. The values obtained from here are then plotted.

```
def draw(self,gm):
        """compute and plot accel, vel, and disp"""
        self.t = gettime(self.db,self.gm)
        self.s = getacc(self.db,self.gm)
        self.dt = getDT(self.db,self.gm)
        self.numpts = getnumpts(self.db,self.gm)
        self.numpts = self.numpts[0]
        self.vel = integrate.cumtrapz(self.s,x=None,dx=self.dt,axis=-1,initial = 0)
        self.vel = self.vel*980
        self.disp = integrate.cumtrapz(self.vel,x=None,dx=self.dt,axis=-1,initial =
0)
        self.axes.clear()
        self.axes2.clear()
        self.axes3.clear()
        self.RP.clear()
        self.axes.plot(self.t, self.s,linewidth=0.7)
        self.axes.set_ylabel('Accel (g)')
        self.axes.minorticks_on()
        self.axes.grid(True)
```

```
        self.axes2.plot(self.t,self.vel,linewidth = 0.7)
        self.axes2.set_ylabel('Vel (cm/s)')
        self.axes2.minorticks_on()
        self.axes2.grid(True)

        self.axes3.plot(self.t,self.disp,linewidth = 0.7)
        self.axes3.set_ylabel('Displacement (cm)')
        self.axes3.minorticks_on()
        self.axes3.grid(True)
        self.axes3.set_xlabel('Time (s)')

        self.calcResponseSpectra(copy.copy(self.s),self.dt,self.numpts)
        self.RP.semilogx(self.p,self.amax,linewidth = 1.5)
        self.RP.set_ylabel("Sa (g)")
        self.RP.set_xlabel("Period (s)")
        self.RP.grid(True)

        self.canvas.draw()
```

The calcResponseSpectra is shown below. However, this code is sufficiently annotated and commented. Note that the user can change the number of points (nPeriod) used to calculate the response spectra to refine the plot.

```
def calcResponseSpectra(self,a,dt,n):
        """calculate response spectra for plotting"""

        # add initial zero value to acceleration
        a=insert(a,0,0)
        # number of periods at which spectral values are to be computed
        nPeriod = 50
        # define range of considered periods by power of 10
        minPower = -3
        maxPower = 1
        #create vector of considered periods
        self.p = logspace(minPower,maxPower,nPeriod)
        # incremental circular frequency
        dw = 2*pi/(self.dt*n)
        #vector of circular frequency
        w = linspace(0,n*dw,n)

        # fast fourier transform of acceleration
        Afft = fft(a)

        #arbitrary stiffness value
        k = 1000
        #damping ratio
        damp = 0.05
        # initialize arrays
        umax = [0]*(nPeriod+1)
        vmax = [0]*(nPeriod+1)
        self.amax = [0]*(nPeriod)

        # loop to compute spectral values at each period
        for j in range(0,nPeriod):
```

```
            # compute mass and dashpot coeff to produce desired periods
            m = ((self.p[j]/(2*pi))**2)*k
            c = 2*damp*(k*m)**0.5
            #initialize transfer function
            H = [0]*(n+1)
            #compute transfer function
            for l in range(0,n/2+1):
                H[l] = 1/(-m*w[l]*w[l]+1j*c*w[l] + k)
                # mirror image of transfer function
                H[n-l] = conj(H[l])

            # compute displacement in frequency domain using transfer function
            Qfft = -m*Afft
            u = [0]*(n+1)
            for l in range(0,n+1):
                u[l] = H[l]*Qfft[l]

            # compute displacement in time domain
            utime = real(ifft(u))

            #spectral displacement, velocity, and acceleration
            umax[j] = max(abs(utime))
            vmax[j] = (2*pi/self.p[j])*umax[j]
            self.amax[j] = (2*pi/self.p[j])*vmax[j]
```

The textbox right under the StaticText “Information” label, displays information associated with the chosen motion. Above, after defining the combo box for the ground motion, the following event was defined:

```
self.widgetMaker(self.cbGM,gm,self.onSelect)
```

This calls widgetMaker which binds the event to the onSelect callback function. This callback function fetches information about that ground motion from the database. First it clears the textbox, which it populated with this information. This prevents a bunch of data from continuously being added onto, or appended, into this textbox. The ground motion selected is obtained through the GetStringSelection method. The getallinfo function found in SQLqueries.py is called which fetches the magnitude, date, rupture distances, and station associated with the motion. These are all added to the textbox using the Append method.

```
def onSelect(self, event):
        """ bind ground motion selected and get magnitude,
        date, rupture distance, event, and station from
        Motions table"""
        self.txtbox.Clear()
        self.gm = self.cbGM.GetStringSelection()
        info = getallinfo(self.db,self.gm)
        info = info[0]
        magn = info[0]
        date = info[1]
        Rrup = info[2]
        Repi = info[3]
        Rhypo = info[4]
        event = self.stri(info[5])
```

```
        station = info[6]
        station = self.stri(station)
        if Rrup == 0.0:
            Rrup = "na"
        if Repi == 0.0:
            Repi = "na"
        if Rhypo == 0.0:
            Rhypo = "na"
        label = "Moment Magnitude: {}\n Date: {}\n Rrup (km): {}\n Repicenter (km):
{}\n Rhypocenter (km): {}\n Event: {}\n Station:
{}".format(magn,date,Rrup,Repi,Rhypo,event,station)
        return self.txtbox.AppendText(label+"\n")
```

The textbox that was just referred to is defined as follows:

```
#create textbox and format for motion info
        self.txtbox = wx.TextCtrl(self.panel,1,"", pos=(10,215),size=(300,130),style
= wx.TE_MULTILINE | wx.SUNKEN_BORDER | wx.TE_READONLY)
        self.txtbox.SetFont(self.font)
        self.txtbox.SetBackgroundColour((230,230,230))
```

It is formatted to have multiple lines and be read-only. The sizer is set so that it expands to fit the panel width.

Next, checkboxes are created for the filters, which include “Butterworth”, “Low Pass”, and “High Pass” checkboxes. A checkbox was added for butterworth with the idea that other filters will be added later. The butterworth checkbox and either the low pass or high pass filter checkbox have to be selected in order to call the filterGM function which calculates the butterworth filter parameters and filters the data. The SetValue method either checks or unchecks the check boxes. Here it is set to False so the checkboxes are initially not checked when the GUI is run.

```
        # create check boxes
        self.cb = wx.CheckBox(self.panel3,-1,"Butterworth",(10,70))
        self.cb.SetValue(False)
        self.Bind(wx.EVT_CHECKBOX,self.UponFilterSelection)
        # create low pass checkbox
        self.cbFC = wx.CheckBox(self.panel3,-1,"Low Pass",(150,70))
        self.cbFC.SetValue(False)
        # create high pass checkbox
        self.cbFC2 = wx.CheckBox(self.panel3,-1,"High Pass",(150,90))
        self.cbFC2.SetValue(False)
```

Next, the sliders are created using wx.Slider. This takes as input the parent, ID, default value to start on, minimum slider bar value, maximum slider bar value, position, size, and style.

```
#create sliders
        self.sld = wx.Slider(self.panel3,ID_SLIDER1,2,1,8,pos=(40,130),size=(250,-1),
                             style=wx.SL_AUTOTICKS|wx.SL_HORIZONTAL)
        self.sld2 =
wx.Slider(self.panel3,ID_SLIDER2,0.0,0.0,50.0,pos=(40,170),size=(250,-1),
                              style=wx.SL_AUTOTICKS|wx.SL_HORIZONTAL)

        wx.EVT_SLIDER(self,ID_SLIDER1,self.onMoveSlider)
        wx.EVT_SLIDER(self,ID_SLIDER2,self.onMoveSlider)
```

The textboxes right next to the slider bars are also created. The top slider is used to select order and the bottom slider is used to select cutoff frequency. The textbox right next to the order slider bar displays the order selected with the slider bar. The default starting value is 2, and the textbox is read-only. The textbox right next to the frequency slider bar displays the cutoff frequency selected with the slider bar. However, it is editable by the user. For future use, this textbox entry should be validated, i.e. must be input with a certain format.

```
        # create textbox for displaying "order"
        self.txtbox1 = wx.TextCtrl(self.panel3,1,"2",pos = (290,130),size =
(20,20),style=wx.TE_READONLY)
        self.txtbox1.SetFont(self.font)
        self.txtbox1.SetBackgroundColour((230,230,230))

        #create textbox for displaying "frequency"
        self.txtbox2 = wx.TextCtrl(self.panel3,1,"00.00",pos = (290,170),size =
(45,20))
        self.txtbox2.SetFont(self.font)
```

The callback function associated with the sliders is onMoveSlider. This clears the textbox next to the slider every time it is moved. The text associated with the chosen order and cutoff frequency numbers on the sliders are appended to the textboxes directly next to the sliders.

```
    def onMoveSlider(self,event):
        self.txtbox1.Clear()
        self.txtbox2.Clear()
        self.order = self.sld.GetValue()
        self.freq = self.sld2.GetValue()
        self.freq= "{00:.2f}".format(self.freq)
        self.txtbox2.AppendText(str(self.freq))
        self.txtbox1.AppendText(str(self.order))
```

The “Refresh” button is created which filters the ground motion per the pass, order, and cutoff frequency selected. Its event is bound to the callback function filterGM, which computes the filtered acceleration, velocity, and displacement data in real time and plots these time histories against the unfiltered data. This function is sufficiently annotated for any user trying to modify this code.

```
        #create "Refresh button"
        self.button2 = wx.Button(self.panel3,ID_BUTTON4,label = "Refresh",pos =
(10,230))
        self.button2.SetFont(self.font)

        wx.EVT_BUTTON(self,ID_BUTTON4,self.filterGM)


    def filterGM(self,e):
        """perform high pass and low pass butterworth filters and plot"""
        # if "butter" is checked, i.e. GetValue() == TRUE
        if self.cb.GetValue():
            # if "Low Pass" is checked, i.e. GetValue() == TRUE
            if self.cbFC.GetValue():
                # this val is only used if I write files
```

```
                val = "LP"
                # define order and frequency selected. get values from textboxes
                self.freq_update = float(self.txtbox2.GetValue())
                self.order_update = int(self.txtbox1.GetValue())

                #calculate critical frequency norm
                freqnorm = self.freq_update*2*self.dt
                # try and get butterworth parameters. If cutoff frequency >1 or <0 an
error occurs because these are outsde the range of butterworth filters
                try:
                    b,a = signal.butter(self.order_update,freqnorm,btype="lowpass")
                # if critical frequency not within bounds, pop up dialog warning user
                except:
                    msg= "Frequency norm is {}. Frequency norm needs to be between 0
and 1.".format(freqnorm)

                    dlg = wx.MessageDialog(self,
                                           msg,
                                           "Error",
                                           wx.OK | wx.ICON_INFORMATION)

                    dlg.ShowModal()
                    dlg.Destroy()

            # if "High Pass" is checked, i.e. GetValue() == TRUE
            if self.cbFC2.GetValue():
                # val only used if I write files
                val = "HP"
                # define order and frequency selected. get values from textboxes
                self.freq_update = float(self.txtbox2.GetValue())
                self.order_update = int(self.txtbox1.GetValue())
                #calculate critical frequency norm
                freqnorm = self.freq_update*2*self.dt
                # try and get butterworth parameters. If cutoff frequency >1 or <0 an
error occurs because these are outsde the range of butterworth filters
                try:
                    b,a = signal.butter(self.order_update,freqnorm,btype="highpass")
                # if critical frequency not within bounds, pop up dialog warning user
                except:
                    msg= "Frequency norm is {}. Frequency norm needs to be between 0
and 1.".format(freqnorm)

                    dlg = wx.MessageDialog(self,
                                           msg,
                                           "Error",
                                           wx.OK | wx.ICON_INFORMATION)

                    dlg.ShowModal()
                    dlg.Destroy()
            # butterworth filters parameters may not have been computed because of
the critical frequency issue so use try-except
            try:
                # filter data
                self.s_filt = signal.lfilter(b,a,self.s)
                # calculate vel by integrating acc data
```

```
                vel_filt = integrate.cumtrapz(self.s_filt,x=None,dx=self.dt,axis=-
1,initial = 0)
                vel_filt = vel_filt*980
                # calc displacement by integrating disp data
                disp_filt = integrate.cumtrapz(vel_filt,x=None,dx=self.dt,axis=-
1,initial = 0)
                # clear plot after refresh button clicked
                self.axes.clear()
                self.axes2.clear()
                self.axes3.clear()
                self.RP.clear()
                # plot unfiltered time histories and response spectrum
                self.draw(self.gm)

                # plot filtered acc vs time
                self.axes.plot(self.t, self.s_filt,linewidth=1.0,color="black")
                self.axes.set_ylabel('Accel (g)')
                self.axes.minorticks_on()
                self.axes.grid(True)

                #plot filtered vel vs time
                self.axes2.plot(self.t,vel_filt,linewidth = 0.7,color="black")
                self.axes2.set_ylabel('Vel (cm/s)')
                self.axes2.minorticks_on()
                self.axes2.grid(True)

                #plot filtered acc vs time
                self.axes3.plot(self.t,disp_filt,linewidth = 0.7,color="black")
                self.axes3.set_ylabel('Displacement (cm)')
                self.axes3.minorticks_on()
                self.axes3.grid(True)
                self.axes3.set_xlabel('Time (s)')

                self.canvas.draw()
            except:
                pass#
```

A Reset View button is then created and positioned in the center and at the bottom of the window:

```
        self.button = wx.Button(self.panel3, ID_BUTTON1, label='Reset View',
pos=(10,10))
        self.button.SetFont(self.font)

        # set up sizers for Reset View button
        hsizeB = wx.BoxSizer(wx.HORIZONTAL)
        hsizeB.Add(self.button, 0, wx.EXPAND)
        hsizeB.AddStretchSpacer(1)
        vsizeB = wx.BoxSizer(wx.VERTICAL)
        vsizeB.SetMinSize(wx.Size(0,-1))
        vsizeB.AddStretchSpacer(1)
        vsizeB.Add(hsizeB, 0, wx.EXPAND)
        vsizeB.Add(hsizeB, 0, wx.ALIGN_CENTER)
```

```
        self.panel.SetSizer(vsizeB)
```

This button resets the size of the window per the callback function below.

```
        wx.EVT_BUTTON(self, ID_BUTTON1, self.onViewReset)


    def onViewReset(self,e):
        self.ShowFullScreen(False)
        self.SetSize((768,430))
        self.Center()
```

Appendix B
Create Database Algorithm

After all the required libraries and packages that are required to run the code are imported, the database is connected to and the tables are created.

```
# connect to databse
db= connect()
# create all tables
createTables()
```

These functions are found in SQLqueries.py and are as follows:

```
def connect():
    print 'accessing ground motion database ...' ,
    try:
        db = dbi.connect('WSGMDB.db')
        print 'success'
    except:
        print 'failed'
        sys.exit()
    return db

def createTables():
    db = connect()

    cu = db.cursor()

    cu.execute("""drop table if exists Motions;""")
    cu.execute("""drop table if exists Events;""")
    cu.execute("""drop table if exists Regions;""")
    cu.execute("""drop table if exists Networks;""")
    cu.execute("""drop table if exists Stations;""")
    cu.execute("""drop table if exists Groups;""")
    cu.execute("""drop table if exists LongDuration;""")

    cu.execute("""create table Motions (
        ID              integer not null primary key autoincrement,
        name            text,
        dt              real,
        numpoints       real,
        eventID         int,
        groupID         int,
        stationID       int,
        Rrup            real,
        Repi            real,
        Rhypo           real
        );""")

    cu.execute("""create table Events (
        ID              integer not null primary key autoincrement,
        directory       text,
        description     text,
        regionID        int,
        date            text,
```

```
        magnitude       real
        );""")

    cu.execute("""create table Regions (
        ID              integer not null primary key autoincrement,
        name            text,
        description     text
        );""")

    cu.execute("""create table Stations (
        ID              integer not null primary key autoincrement,
        name            text
        );""")

    cu.execute("""create table Groups (
        ID              integer not null primary key autoincrement,
        name            text,
        groupID         text
        );""")
```

As stated previously, the ground motions used to create this database are all stored as “event” folders, which are stored in one folder. Event folders are the earthquake name or location. For example, “Tabas” is an event folder that 3 ground motion files stored within it. The code walks through the path defined, and picks out the directory path, directory names, and filenames. The directories are the event folders, and I loop through the directories, accessing the files and creating a table for each filename. I then open the file, and read it line by line, adding acceleration and time columns to the table just created.

```
# create ground motion tables by walking through folders in mypath
mypath = "D:\WSGMDB\All Motions\By Earthquake CL"
numfiles = 0
for (dirpath, dirnames,filenames) in walk(mypath):
    for dire in dirnames:
        for filename in listdir("{}\{}".format(mypath,dire)):
            numfiles +=1
            print filename
            # creata table for each filename
            createGMtables(db,filename)
            time = 0
            # open and read file; read each header line
            with open("{}\{}\{}".format(mypath,dire,filename),'r') as fi:
                hdr = fi.readline().split()
                hdr = fi.readline().split()
                hdr = fi.readline().split()
                hdr = fi.readline().split()
                # define number of points, obtained from header line
                n = float(hdr[0])
                # define dt, obtained from header line
                dt = float(hdr[1])
                # loop through the file rows, parse lines, and add each incremental
time period and accel value to the gm table
                for line in fi:
```

```
                    accel = parse_line(line,"space")
                    accel = [float(i) for i in accel]
                    for i in accel:
                        createGMEntry(db,stri(filename),time,i)
                        time += dt
                fi.close()
```

As an example for how the tables are created and filled up, the functions for creating tables and inputting data is shown below. The table name is the ground motion filename accessed with the walk function. The cursor points to the database and the statement is execute using the execute() method. Entries for the table are created using the statement INSERT.

```
def createGMtables(db,GM):
    cu = db.cursor()
    call="""drop table if exists '{}';"""
    cu.execute(call.format(GM))
    call = """create table '{}' (
    ID            integer not null primary key,
    time          real,
    accel         real
    );"""
    cu.execute(call.format(GM))

def createGMEntry(db,filename,time,accel):
    cu = db.cursor()
    sql_command = """INSERT INTO '{}' (time,accel)
                     VALUES ({},{})"""
    cu.execute(sql_command.format(filename,time,accel))
```

Next, the regions table is created. The Regions text file contains the entries necessary to create the table. The text file is opened, lines read line by line, and the info picked out and inserted into the Regions table.

```
# define file path where all text files used are stored
fileroot = "C:\Users\cyndi\workspace\CEE505Final"
# define filename, which changes depending on the text file being used
filename = "Regions.txt"
# open and read filename
f = accessFile("{}\{}".format(fileroot,filename))
# read header line
hdr = f.readline().split('\t')

# now parse the file line by line
for line in f:
    region_info = parse_line(line,"tab")

    # define variables per their column location in the table
    name = region_info[1][:]
    description = region_info[2][:]

    #add region info to Regions Table
    createRegionsEntry(db,name,description)
f.close()
```

The same methodology is used to create the Groups table.

```
# define filename
filename = "Groups.txt"
# open and read file
f = accessFile("{}\{}".format(fileroot,filename))
# read header line
hdr = f.readline().split('\t')

# now parse the file line by line
for line in f:
    group_info = parse_line(line,"tab")

    # define variables per their column location in the table
    name = group_info[1][:]
    groupID = group_info[2][:]

    #add region info to Regions Table
    createGroupsEntry(db,name,groupID)
f.close()
```

The motions text file includes all the relevant information about each ground motion, including the date, event, rupture distance, station, and region. The file is accessed, and the information for the events table picked out. However, because some motions are associated with more than one event, an event input into the Events table is appended to a list, which is checked each time a line is read with an if statement to ensure that the event is not already in the table.

```
# define filename
filename = "Motions.txt"
# open and read file
f = accessFile("{}\{}".format(fileroot,filename))
hdr = f.readline().split('\t')
# now parse the file line by line
events = []
for line in f:
    events_info = parse_line(line,"tab")

    # define variables per their column location in the table
    dire = events_info[0][:]
    dire2 = events_info[2][:]
    description = events_info[4][:]
    date = stri(events_info[6][:])
    magn = stri(events_info[7][:])
    regionID = randint(1,4)
    station = stri(events_info[5][:])

    # if the event is not in the events list, create an entry in the Events table
    if dire not in events:
        createEventsEntry(db,dire,description,regionID,date,magn)
    if dire2 not in events:
        if dire2 != dire:
            createEventsEntry(db,dire2,description,regionID,date,magn)
    # add each event to the events list to avoid duplication
    events.append(dire)
    # once again, ensure events are unique
```

```
    if dire != dire2:
        events.append(dire2)
    # add columns to Stations table
    createStationsEntry(db,station)
```

The Motions table is then created. The relevant information is picked out from the motions text file, however, the integer IDs associated with these strings are fetched from the tables already created. For example, a group “LD”, standing for long duration, was obtained from the motions text file. A Groups table was already created, and “LD” is looked up in the groups table and its associated integer primary key obtained and stored in the Motions table. An example is shown below.

```
    # get motion info
    motions_info = parse_line(line,"tab")
    motion = motions_info[1][:]
    motion2 = motions_info[3][:]
    Rrup = myfloat(motions_info[8][:])
    Repi = myfloat(motions_info[9][:])
    Rhypo = myfloat(motions_info[10][:])
    groupID = motions_info[11][:]
    # get the IDs associated with the event, group, and station to add as columns to
Motions table
    eventID_int = get_eventID(db,dire)
    eventID2_int = get_eventID(db,dire2)
    groupID_int = get_groupID(db,groupID)
    stationID_int = get_stationID(db,station)
    # add motions info to motions table

createMotionsEntry(db,motion,eventID_int,groupID_int,stationID_int,Rrup,Repi,Rhypo)

createMotionsEntry(db,motion2,eventID2_int,groupID_int,stationID_int,Rrup,Repi,Rhypo)

# close open files
f.close()


def get_eventID(db,dire):
#     db.commit()
    cu = db.cursor()

    call = """
    SELECT e.ID
        FROM Events as e
        WHERE e.directory = "{}"
        ;
    """
#     print call.format(dire)
    cu.execute(call.format(dire))
    eventID = cu.fetchall()
    eventID = [int(i[0]) for i in eventID]
    eventID = eventID[0]
    return eventID
```

The database is saved and closed.

```
# save all data input to database
db.commit()
# close the database
db.close()
```

PYTHON CODE

```
'''
Created on Dec 9, 2015

@author: cyndi
'''

import sqlite3 as dbi
import sys      # needed for the emergency exit() call
from os import getcwd
from os import listdir
from os import walk
from os.path import isfile, join
from SQLqueries import *
from CreateDB import *
from random import randint

def stri(s):
    """Takes as input a string, s, and removes quotes
    at the beginning and end of string"""
    if s.startswith('"') and s.endswith('"'):
        string = s[1:-1]
    else:
        string = s
    return string

def myfloat(s):
    """Takes as input a string, s, and converts into floating
    point number"""
    # if string is empty, this error handling will convert empty string into 0.0
    try:
        v = float(s)
    except:
        v =0.0
    return v

def parse_line(line_in,delim):
    """takes as input a line from a file and a delim, either
    "space" or "tab", and parses the line"""
    if delim == "space":
        line_out = line_in.split()
    elif delim == "tab":
        line_out = line_in.split('\t')
    # removes white space at end of string
    line_out[-1] = line_out[-1].rstrip()
    #take values out of list
    for i in range(1,len(line_out)):
        try:
            val = line_out[i]
        except:
            val = 0
        line_out[i] = val
    return line_out

def accessFile(filename):
    """ read and open filename"""
    print 'accessing file ...' ,
```

```
    try:
        f = open(filename,'r')
        print 'success'
    except IOError:
        print 'failed'
        sys.exit()
    return f

# connect to databse
db= connect()
# create all tables
createTables()

# create ground motion tables by walking through folders in mypath
mypath = "D:\WSGMDB\All Motions\By Earthquake CL"
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath,f))]
numfiles = 0
for (dirpath, dirnames,filenames) in walk(mypath):
    for dire in dirnames:
        for filename in listdir("{}\{}".format(mypath,dire)):
            numfiles +=1
            print filename
            # creata table for each filename
            createGMtables(db,filename)
            time = 0
            # open and read file; read each header line
            with open("{}\{}\{}".format(mypath,dire,filename),'r') as fi:
                hdr = fi.readline().split()
                hdr = fi.readline().split()
                hdr = fi.readline().split()
                hdr = fi.readline().split()
                # define number of points, obtained from header line
                n = float(hdr[0])
                # define dt, obtained from header line
                dt = float(hdr[1])
                # loop through the file rows, parse lines, and add each incremental time
  period and accel value to the gm table
                for line in fi:
                    accel = parse_line(line,"space")
                    accel = [float(i) for i in accel]
                    for i in accel:
                        createGMEntry(db,stri(filename),time,i)
                        time += dt
                fi.close()

# define file path where all text files used are stored
fileroot = "C:\Users\cyndi\workspace\CEE505Final"
# define filename, which changes depending on the text file being used
filename = "Regions.txt"
# open and read filename
f = accessFile("{}\{}".format(fileroot,filename))
# read header line
hdr = f.readline().split('\t')

# now parse the file line by line
for line in f:
```

```
112     region_info = parse_line(line,"tab")
113 
114     # define variables per their column location in the table
115     name = region_info[1][:]
116     description = region_info[2][:]
117 
118     #add region info to Regions Table
119     createRegionsEntry(db,name,description)
120 f.close()
121 
122 # define filename
123 filename = "Groups.txt"
124 # open and read file
125 f = accessFile("{}\{}".format(fileroot,filename))
126 # read header line
127 hdr = f.readline().split('\t')
128 
129 # now parse the file line by line
130 for line in f:
131     group_info = parse_line(line,"tab")
132 
133     # define variables per their column location in the table
134     name = group_info[1][:]
135     groupID = group_info[2][:]
136 
137     #add region info to Regions Table
138     createGroupsEntry(db,name,groupID)
139 f.close()
140 
141 # define filename
142 filename = "Motions.txt"
143 # open and read file
144 f = accessFile("{}\{}".format(fileroot,filename))
145 hdr = f.readline().split('\t')
146 # now parse the file line by line
147 events = []
148 for line in f:
149     events_info = parse_line(line,"tab")
150 
151     # define variables per their column location in the table
152     dire = events_info[0][:]
153     dire2 = events_info[2][:]
154     description = events_info[4][:]
155     date = stri(events_info[6][:])
156     magn = stri(events_info[7][:])
157     regionID = randint(1,4)
158     station = stri(events_info[5][:])
159 
160     # if the event is not in the events list, create an entry in the Events table
161     if dire not in events:
162         createEventsEntry(db,dire,description,regionID,date,magn)
163     if dire2 not in events:
164         if dire2 != dire:
165             createEventsEntry(db,dire2,description,regionID,date,magn)
166     # add each event to the events list to avoid duplication
167     events.append(dire)
```

```
168     # once again, ensure events are unique
169     if dire != dire2:
170         events.append(dire2)
171     # add columns to Stations table
172     createStationsEntry(db,station)
173 
174     # get motion info
175     motions_info = parse_line(line,"tab")
176     motion = motions_info[1][:]
177     motion2 = motions_info[3][:]
178     Rrup = myfloat(motions_info[8][:])
179     Repi = myfloat(motions_info[9][:])
180     Rhypo = myfloat(motions_info[10][:])
181     groupID = motions_info[11][:]
182     # get the IDs associated with the event, group, and station to add as columns to Motions
    table
183     eventID_int = get_eventID(db,dire)
184     eventID2_int = get_eventID(db,dire2)
185     groupID_int = get_groupID(db,groupID)
186     stationID_int = get_stationID(db,station)
187     # add motions info to motions table
188     createMotionsEntry(db,motion,eventID_int,groupID_int,stationID_int,Rrup,Repi,Rhypo)
189     createMotionsEntry(db,motion2,eventID2_int,groupID_int,stationID_int,Rrup,Repi,Rhypo)
190 
191 # close open files
192 f.close()
193 
194 print numfiles
195 print "done"
196 # save all data input to databse
197 db.commit()
198 # close the database
199 db.close()
200 
201 
202 
```

```
'''
Created on Dec 11, 2015

@author: cyndi
'''
import wx
from IDs import *
import traceback
from SQLqueries import *
import matplotlib
from os import getcwd
from wx import BU_EXACTFIT
matplotlib.use('WXAgg')
from scipy import integrate
from scipy import signal
import matplotlib.gridspec as gridspec
from math import pi
from scipy.fftpack import fft
from numpy import logspace
from numpy import conj
from scipy import ifft
from numpy import real
from numpy import linspace
import copy
from numpy import insert
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigCanvas,\
    NavigationToolbar2WxAgg as NavigationToolbar
from matplotlib.figure import Figure
from matplotlib.backends.backend_pdf import PdfPages

# update formatting of plots in matplotlib
matplotlib.rcParams.update({'font.size':9})
matplotlib.rc('xtick',labelsize = 10)
matplotlib.rc('ytick',labelsize = 10)

class MyAppWindow(wx.Frame):
    '''
    classdocs
    '''

    def __init__(self, title='Plot Ground Motion Data'):
        '''
        Constructor
        '''

        wx.Frame.__init__(self, None, ID_SELF, title, size=(1020,960))

        # define and connect to database
        self.database = "WSGMDB.db"
        self.db = dbi.connect(self.database)
        self.dirname = "C:\Users\cyndi\workspace\CEE505Final"

        #create menubar
        filemenu = wx.Menu()
        filemenu.Append(ID_FILE3,"&Save plot\tCtrl-S", "Save plot to file")
        self.Bind(wx.EVT_MENU, self.OnSave)
```

```
57          filemenu.AppendSeparator()
58          filemenu.Append(ID_QUIT,"&Quit","Quit the Application")
59          filemenu.Append(ID_FILE5,"&About","Find fastest Path")
60
61          editmenu = wx.Menu()
62          editmenu.Append(ID_EDIT1,"&Copy","Copy selection to buffer")
63          editmenu.Append(ID_EDIT2,"C&ut","Cut selection")
64          editmenu.Append(ID_EDIT3,"&Paste","Paste buffer")
65
66          viewmenu = wx.Menu()
67          viewmenu.Append(ID_VIEW1,"&Shrink","Reduce size to half")
68          viewmenu.Append(ID_VIEW2,"&Maximize","Maximize window")
69          viewmenu.Append(ID_VIEW3,"&Full screen","Switch to full screen mode")
70          viewmenu.Append(ID_VIEW4,"&Reset","Reset View")
71
72          menubar = wx.MenuBar()
73          menubar.Append(filemenu,"&File")
74          menubar.Append(editmenu,"&Edit")
75          menubar.Append(viewmenu,"&View")
76
77          self.SetMenuBar(menubar)
78
79          #create status bar
80          self.statusbar = self.CreateStatusBar()
81
82          # create panels; self.panel and panel3 are the left panels and panel 2 is the right
      panel
83          splitter = wx.SplitterWindow(self, wx.ID_ANY,style = wx.SP_BORDER)
84          self.panel = wx.Panel(splitter,ID_PANEL1,style=wx.BORDER)
85          self.panel.SetBackgroundColour([230,230,230])
86
87          self.panel3 = wx.Panel(splitter, ID_PANEL3,style=wx.BORDER,size = (360,400))
88          splitter.SplitHorizontally(self.panel, self.panel3)
89
90          panel2 = wx.Panel(self, ID_PANEL2, size=(680,960),style=wx.BORDER)
91          panel2.SetBackgroundColour([230,230,230])
92
93          # sizers for panels
94          hsizer1 = wx.BoxSizer(wx.HORIZONTAL)
95          hsizer1.Add(splitter, 0, wx.EXPAND)
96          hsizer1.Add(panel2, 2, wx.EXPAND)
97          vbox1 = wx.BoxSizer(wx.VERTICAL)
98          vbox1.Add(hsizer1, 1, wx.EXPAND)
99          self.SetSizer(vbox1)
100
101         # get all gm from databse
102         gm = getGMnames(self.db)
103
104         # create all Static Text labels
105         self.lbl1 = wx.StaticText(self.panel,label = "Library of Motions", pos = (10,10))
106         self.lbl2 = wx.StaticText(self.panel,label="Region:",pos = (10,47))
107         self.lbl3 = wx.StaticText(self.panel,label= "Type:", pos = (10,80))
108         self.lbl4 = wx.StaticText(self.panel,label = "Magn.:", pos = (10,114))
109         self.lbl5 = wx.StaticText(self.panel,label="to",pos = (130,114))
110         self.lblinfo = wx.StaticText(self.panel,label="Information: ",pos = (10,195))
111         self.lblf = wx.StaticText(self.panel3,label = "Filtering", pos=(10,10))
```

```
112         self.lblf2 = wx.StaticText(self.panel3,label = "Filter Type",pos=(10,50))
113         self.lblf3 = wx.StaticText(self.panel3,label = "Filter Configuration", pos=(150,50))
114         self.lblf4 = wx.StaticText(self.panel3,label = "Order and Frequency", pos = (10,110))
115         self.lblf5 = wx.StaticText(self.panel3,label = "Order",pos=(10,135))
116         self.lblf6 = wx.StaticText(self.panel3,label = "Freq.",pos=(10,175))
117 
118         # define fonts
119         self.font = wx.Font(10,wx.DEFAULT,wx.NORMAL,wx.NORMAL)
120         self.fontLarge = wx.Font(12,wx.DEFAULT,wx.NORMAL,wx.NORMAL)
121         self.fontSmall = wx.Font(9,wx.DEFAULT,wx.NORMAL,wx.NORMAL)
122 
123         self.lbl1.SetFont(self.fontLarge)
124         self.lbl2.SetFont(self.fontSmall)
125         self.lbl3.SetFont(self.fontSmall)
126         self.lbl4.SetFont(self.fontSmall)
127         self.lblinfo.SetFont(self.fontLarge)
128         self.lblf.SetFont(self.fontLarge)
129         self.lblf2.SetFont(self.font)
130         self.lblf3.SetFont(self.font)
131         self.lblf4.SetFont(self.font)
132         self.lbl5.SetFont(self.fontSmall)
133 
134         # initialize lists for dropdown boxes
135         regionsList = []
136         groupsList = []
137         magn1List = []
138         magn2List = []
139         regionslist = ["Region 1", "Region 2", "Region 3", "Region 4", "ALL"]
140         groupslist = ["Long Duration","Near Fault", "Other","ALL"]
141         magn1list = [str(i) for i in linspace(5.5,8.5,7)]
142         magn2list = [str(i) for i in linspace(6,9,7)]
143 
144         # create comboboxes
145         self.cbGM = wx.ComboBox(self.panel,size = wx.DefaultSize, choices = [],pos =
    (130,155))
146         self.cbGM.SetFont(self.font)
147         self.cbR = wx.ComboBox(self.panel,size = (100,15),choices = regionsList, pos =
    (70,45))
148         self.cbTy = wx.ComboBox(self.panel,size = (120,15), choices = groupsList, pos =
    (70,77))
149         self.cbM1 = wx.ComboBox(self.panel,size = (45,15), choices = magn1List, pos =
    (70,110))
150         self.cbM2 = wx.ComboBox(self.panel,size = (45,15),choices = magn2List, pos =
    (155,110))
151 
152         # call function to bind chosen values from dropdown
153         self.widgetMaker(self.cbGM,gm,self.onSelect)
154         self.widgetMaker(self.cbR,regionslist,self.onChoiceReg)
155         self.widgetMaker(self.cbTy,groupslist,self.onChoiceGrp)
156         self.widgetMaker(self.cbM1,magn1list,self.onChoiceMagMin)
157         self.widgetMaker(self.cbM2,magn2list,self.onChoiceMagMax)
158 
159         # change font size of comboboxes
160         self.cbR.SetFont(self.fontSmall)
161         self.cbTy.SetFont(self.fontSmall)
162         self.cbM1.SetFont(self.fontSmall)
```

```
163         self.cbM2.SetFont(self.fontSmall)
164 
165         #create "get groundmotion" button
166         self.GMbutton = wx.Button(self.panel,ID_BUTTON6,label = "Get GM", pos = (10,150),size
    = (100,30))
167 
168         #change font of ground motion button
169         self.GMbutton.SetFont(self.font)
170 
171         # create Plot button
172         self.button = wx.Button(self.panel,ID_BUTTON3,label="Plot", pos = (260,60),size =
    wx.DefaultSize)
173         self.button.SetFont(self.font)
174 
175         # define what Plot button should do when clicked
176         self.Bind(wx.EVT_BUTTON,self.PlotWhenClicked)
177 
178         #create textbox and format for motion info
179         self.txtbox = wx.TextCtrl(self.panel,1,"", pos=(10,215),size=(300,130),style =
    wx.TE_MULTILINE | wx.SUNKEN_BORDER | wx.TE_READONLY)
180         self.txtbox.SetFont(self.font)
181         self.txtbox.SetBackgroundColour((230,230,230))
182 
183         # create check boxes
184         self.cb = wx.CheckBox(self.panel3,-1,"Butterworth",(10,70))
185         self.cb.SetValue(False)
186 #         self.Bind(wx.EVT_CHECKBOX,self.UponFilterSelection)
187         # create low pass checkbox
188         self.cbFC = wx.CheckBox(self.panel3,-1,"Low Pass",(150,70))
189 #         self.Bind(wx.EVT_CHECKBOX,self.UponPassSelection)
190         self.cbFC.SetValue(False)
191         # create high pass checkbox
192         self.cbFC2 = wx.CheckBox(self.panel3,-1,"High Pass",(150,90))
193 #         self.Bind(wx.EVT_CHECKBOX,self.UponPassSelection)
194         self.cbFC2.SetValue(False)
195 
196         # define Reset View button
197         self.button = wx.Button(self.panel3, ID_BUTTON1, label='Reset View', pos=(10,10))
198         self.button.SetFont(self.font)
199 
200         # set up sizers for Reset View button
201         hsizeB = wx.BoxSizer(wx.HORIZONTAL)
202         hsizeB.Add(self.button, 0, wx.EXPAND)
203         hsizeB.AddStretchSpacer(1)
204         vsizeB = wx.BoxSizer(wx.VERTICAL)
205         vsizeB.SetMinSize(wx.Size(0,-1))
206         vsizeB.AddStretchSpacer(1)
207         vsizeB.Add(hsizeB, 0, wx.EXPAND)
208         vsizeB.Add(hsizeB, 0, wx.ALIGN_CENTER)
209         self.panel.SetSizer(vsizeB)
210 
211         #create sliders
212         self.sld = wx.Slider(self.panel3,ID_SLIDER1,2,1,8,pos=(40,130),size=(250,-1),
213                              style=wx.SL_AUTOTICKS|wx.SL_HORIZONTAL)
214         self.sld2 = wx.Slider(self.panel3,ID_SLIDER2,0.0,0.0,50.0,pos=(40,170),size=(250,-1),
215                               style=wx.SL_AUTOTICKS|wx.SL_HORIZONTAL)
```

```
216
217
218         # create textbox for displaying "order"
219         self.txtbox1 = wx.TextCtrl(self.panel3,1,"2",pos = (290,130),size =
    (20,20),style=wx.TE_READONLY)
220         self.txtbox1.SetFont(self.font)
221         self.txtbox1.SetBackgroundColour((230,230,230))
222
223         #create textbox for displaying "frequency"
224         self.txtbox2 = wx.TextCtrl(self.panel3,1,"00.00",pos = (290,170),size = (45,20))
225         self.txtbox2.SetFont(self.font)
226
227         #create "Refresh button"
228         self.button2 = wx.Button(self.panel3,ID_BUTTON4,label = "Refresh",pos = (10,230))
229         self.button2.SetFont(self.font)
230
231         #create figures
232         gs = gridspec.GridSpec(5,1)
233         self.dpi = 100
234         self.fig = Figure(figsize=(9,8.0), dpi=self.dpi,facecolor='white')
235         self.canvas=FigCanvas(panel2,-1,self.fig)
236         self.axes = self.fig.add_subplot(gs[0])
237         self.axes2 = self.fig.add_subplot(gs[1])
238         self.axes3 = self.fig.add_subplot(gs[2])
239         self.RP = self.fig.add_subplot(gs[3:])
240
241         # add toolbar from matplot lib
242         self.toolbar = NavigationToolbar(self.canvas)
243
244         #sizers for figures
245         vsizeC = wx.BoxSizer(wx.VERTICAL)
246         vsizeC.Add(self.canvas,1,wx.EXPAND|wx.ALL)
247         vsizeC.Add(self.toolbar, 0, wx.EXPAND)
248         panel2.SetSizerAndFit(vsizeC)
249
250         # Set signal handling
251         wx.EVT_MENU(self, ID_QUIT, self.onQuit)
252         wx.EVT_MENU(self, ID_VIEW1, self.onViewShrink)
253         wx.EVT_MENU(self, ID_VIEW2, self.onViewMaximize)
254         wx.EVT_MENU(self, ID_VIEW3, self.onViewFullScreen)
255         wx.EVT_MENU(self, ID_VIEW4, self.onViewReset)
256         wx.EVT_MENU(self, ID_FILE5, self.onAbout)
257
258         wx.EVT_SLIDER(self,ID_SLIDER1,self.onMoveSlider)
259         wx.EVT_SLIDER(self,ID_SLIDER2,self.onMoveSlider)
260         wx.EVT_BUTTON(self, ID_BUTTON1, self.onViewReset)
261         wx.EVT_BUTTON(self,ID_BUTTON4,self.filterGM)
262         wx.EVT_BUTTON(self,ID_BUTTON6,self.onSelectGM)
263
264         # my window doesn't close without popping up an error message, thought this would help
265         self.Bind(wx.EVT_CLOSE,self.closeWindow)
266
267     def onChoiceReg(self,event):
268         self.cbGM.Clear()
269         self.region = self.cbR.GetStringSelection()
270         self.region = self.region.replace(" ","")
```

```
271         self.region = getRegionID(self.db,self.region)
272 
273     def onChoiceGrp(self,event):
274         self.cbGM.Clear()
275         self.grp = self.cbTy.GetStringSelection()
276         self.grp = getGroupID(self.db,self.grp)
277 
278     def onChoiceMagMin(self,event):
279         self.cbGM.Clear()
280         self.mag1 = self.cbM1.GetStringSelection()
281         self.mag1 = float(self.mag1)
282         try:
283             if self.mag1 == self.mag2:
284 
285                 msg = "ERROR: Magnitude range cannot be the same."
286 
287                 dlg = wx.MessageDialog(self,
288                                        msg,
289                                        "Error",
290                                        wx.OK | wx.ICON_INFORMATION)
291 
292                 dlg.ShowModal()
293                 dlg.Destroy()
294         except:
295             pass
296 
297     def onChoiceMagMax(self,event):
298         self.mag2 = self.cbM2.GetStringSelection()
299         self.mag2 = float(self.mag2)
300         try:
301             if self.mag1 == self.mag2:
302                 msg = "ERROR: Magnitude range cannot be the same."
303 
304                 dlg = wx.MessageDialog(self,
305                                        msg,
306                                        "Error",
307                                        wx.OK | wx.ICON_INFORMATION)
308 
309                 dlg.ShowModal()
310                 dlg.Destroy()
311         except:
312             pass
313 
314     def widgetMaker(self, widget, objects,function):
315         """ bind combobox selection to function"""
316         for obj in objects:
317             widget.Append(obj)
318         widget.Bind(wx.EVT_COMBOBOX, function)
319 
320     def PlotWhenClicked(self,event):
321         try:
322             self.draw(self.gm)
323         except:
324             pass
325 
326     def onSelectGM(self,e):
```

```
327         gm = getSelectGM(self.db,self.region, self.grp,self.mag1,self.mag2)
328         for i in gm:
329             self.cbGM.Append(i)
330         self.cbGM.SetFont(self.font)
331 
332     def onMoveSlider(self,event):
333         self.txtbox1.Clear()
334         self.txtbox2.Clear()
335         self.order = self.sld.GetValue()
336         self.freq = self.sld2.GetValue()
337         self.freq= "{00:.2f}".format(self.freq)
338         self.txtbox2.AppendText(str(self.freq))
339         self.txtbox1.AppendText(str(self.order))
340 
341     def closeWindow(self,event):
342         self.Destroy()
343 
344     def onAbout(self,event):
345         info = wx.AboutDialogInfo()
346         info.SetName("Simple GUI")
347         info.SetVersion('1.0')
348         info.SetDescription("Plots acc-vel-disp time histories and response spectra.")
349         info.AddDeveloper("Cyndi")
350         wx.AboutBox(info)
351 
352     def OnSave(self,e):
353         file_choices = "PDF(*.pdf)|*.pdf"
354 
355         dlg = wx.FileDialog(
356             self,
357             message="Save plot as...",
358             defaultDir=getcwd(),
359             defaultFile="plot.pdf",
360             wildcard=file_choices,
361             style=wx.SAVE)
362 
363         if dlg.ShowModal() == wx.ID_OK:
364             path = dlg.GetPath()
365             self.canvas.print_figure(path, dpi=self.dpi)
366             self.flash_status_message("Saved to %s" % path)
367 
368     def flash_status_message(self, msg, flash_len_ms=1500):
369         self.statusbar.SetStatusText(msg)
370         self.timeroff = wx.Timer(self)
371         self.Bind(
372             wx.EVT_TIMER,
373             self.on_flash_status_off,
374             self.timeroff)
375         self.timeroff.Start(flash_len_ms, oneShot=True)
376 
377     def on_flash_status_off(self, event):
378         self.statusbar.SetStatusText('')
379 
380     def onQuit(self, event):
381         msg = "Good bye"
382 
```

```
383         dlg = wx.MessageDialog(self,
384                                msg,
385                                "Exit",
386                                wx.OK | wx.ICON_INFORMATION)
387 
388         dlg.ShowModal()
389         dlg.Destroy()
390         self.Close()
391 
392     def onViewReset(self,e):
393         self.ShowFullScreen(False)
394         self.SetSize((768,430))
395         self.Center()
396 
397     def onViewFullScreen(self,e):
398         self.ShowFullScreen(True)
399 
400     def onViewMaximize(self,e):
401         self.Maximize(True)
402 
403     def onViewShrink(self,e):
404         msize = []
405         for i in self.GetSize():
406             msize.append(int(0.5*i))
407         self.SetSize(msize)
408 
409     def cause_error(self):
410 #         raise Exception, 'this is a test'
411         pass
412 
413     def draw(self,gm):
414         """compute and plot accel, vel, and disp"""
415         self.t = gettime(self.db,self.gm)
416         self.s = getacc(self.db,self.gm)
417         self.dt = getDT(self.db,self.gm)
418         self.numpts = getnumpts(self.db,self.gm)
419         self.numpts = self.numpts[0]
420         self.vel = integrate.cumtrapz(self.s,x=None,dx=self.dt,axis=-1,initial = 0)
421         self.vel = self.vel*980
422         self.disp = integrate.cumtrapz(self.vel,x=None,dx=self.dt,axis=-1,initial = 0)
423 
424 #         file2write = open("Vel{}".format(self.gm), "w")
425 #         for i in range(0,self.numpts):
426 #             file2write.write(str(self.t[i])+"\t"+str(self.vel[i])+"\n")
427 #         file2write.close()
428 #
429 #         file2write = open("Disp{}".format(self.gm), "w")
430 #         for i in range(0,self.numpts):
431 #             file2write.write(str(self.t[i])+"\t"+str(self.disp[i])+"\n")
432 #         file2write.close()
433 
434         self.axes.clear()
435         self.axes2.clear()
436         self.axes3.clear()
437         self.RP.clear()
438         self.axes.plot(self.t, self.s,linewidth=1.0)
```

```
439         self.axes.set_ylabel('Accel (g)')
440         self.axes.minorticks_on()
441         self.axes.grid(True)
442 
443         self.axes2.plot(self.t,self.vel,linewidth = 0.7)
444         self.axes2.set_ylabel('Vel (cm/s)')
445         self.axes2.minorticks_on()
446         self.axes2.grid(True)
447 
448         self.axes3.plot(self.t,self.disp,linewidth = 0.7)
449         self.axes3.set_ylabel('Displacement (cm)')
450         self.axes3.minorticks_on()
451         self.axes3.grid(True)
452         self.axes3.set_xlabel('Time (s)')
453 
454         self.calcResponseSpectra(copy.copy(self.s),self.dt,self.numpts)
455         self.RP.semilogx(self.p,self.amax,linewidth = 1.5)
456         self.RP.set_ylabel("Sa (g)")
457         self.RP.set_xlabel("Period (s)")
458         self.RP.grid(True)
459 
460         self.canvas.draw()
461 
462     def calcResponseSpectra(self,a,dt,n):
463         """calculate response spectra for plotting"""
464 
465         # add initial zero value to acceleration
466         a=insert(a,0,0)
467         # number of periods at which spectral values are to be computed
468         nPeriod = 50
469         # define range of considered periods by power of 10
470         minPower = -3
471         maxPower = 1
472         #create vector of considered periods
473         self.p = logspace(minPower,maxPower,nPeriod)
474         # incremental circular frequency
475         dw = 2*pi/(self.dt*n)
476         #vector of circular frequency
477         w = linspace(0,n*dw,n)
478 
479         # fast fourier transform of acceleration
480         Afft = fft(a)
481 
482         #arbitrary stiffness value
483         k = 1000
484         #damping ratio
485         damp = 0.05
486         # initialize arrays
487         umax = [0]*(nPeriod+1)
488         vmax = [0]*(nPeriod+1)
489         self.amax = [0]*(nPeriod)
490 
491         # loop to compute spectral values at each period
492         for j in range(0,nPeriod):
493             # compute mass and dashpot coeff to produce desired periods
494             m = ((self.p[j]/(2*pi))**2)*k
```

```
495             c = 2*damp*(k*m)**0.5
496             #initialize transfer function
497             H = [0]*(n+1)
498             #compute transfer function
499             for l in range(0,n/2+1):
500                 H[l] = 1/(-m*w[l]*w[l]+1j*c*w[l] + k)
501                 # mirror image of transfer function
502                 H[n-l] = conj(H[l])
503 
504             # compute displacement in frequency domain using transfer function
505             Qfft = -m*Afft
506             u = [0]*(n+1)
507             for l in range(0,n+1):
508                 u[l] = H[l]*Qfft[l]
509 
510             # compute displacement in time domain
511             utime = real(ifft(u))
512 
513             #spectral displacement, velocity, and acceleration
514             umax[j] = max(abs(utime))
515             vmax[j] = (2*pi/self.p[j])*umax[j]
516             self.amax[j] = (2*pi/self.p[j])*vmax[j]
517 
518     def filterGM(self,e):
519         """perform high pass and low pass butterworth filters and plot"""
520         # if "butter" is checked, i.e. GetValue() == TRUE
521         if self.cb.GetValue():
522             # if "Low Pass" is checked, i.e. GetValue() == TRUE
523             if self.cbFC.GetValue():
524                 # this val is only used if I write files
525                 val = "LP"
526                 # define order and frequency selected. get values from textboxes
527                 self.freq_update = float(self.txtbox2.GetValue())
528                 self.order_update = int(self.txtbox1.GetValue())
529 
530                 #calculate critical frequency norm
531                 freqnorm = self.freq_update*2*self.dt
532                 # try and get butterworth parameters. If cutoff frequency >1 or <0 an error
    occurs because these are outsde the range of butterworth filters
533                 try:
534                     b,a = signal.butter(self.order_update,freqnorm,btype="lowpass")
535                 # if critical frequency not within bounds, pop up dialog warning user
536                 except:
537                     msg= "Frequency norm is {}. Frequency norm needs to be between 0 and
    1.".format(freqnorm)
538 
539                     dlg = wx.MessageDialog(self,
540                                            msg,
541                                            "Error",
542                                            wx.OK | wx.ICON_INFORMATION)
543 
544                     dlg.ShowModal()
545                     dlg.Destroy()
546 
547             # if "High Pass" is checked, i.e. GetValue() == TRUE
548             if self.cbFC2.GetValue():
```

```
549                 # val only used if I write files
550                 val = "HP"
551                 # define order and frequency selected. get values from textboxes
552                 self.freq_update = float(self.txtbox2.GetValue())
553                 self.order_update = int(self.txtbox1.GetValue())
554                 #calculate critical frequency norm
555                 freqnorm = self.freq_update*2*self.dt
556                 # try and get butterworth parameters. If cutoff frequency >1 or <0 an error
    occurs because these are outsde the range of butterworth filters
557                 try:
558                     b,a = signal.butter(self.order_update,freqnorm,btype="highpass")
559                 # if critical frequency not within bounds, pop up dialog warning user
560                 except:
561                     msg= "Frequency norm is {}. Frequency norm needs to be between 0 and
    1.".format(freqnorm)
562 
563                     dlg = wx.MessageDialog(self,
564                                            msg,
565                                            "Error",
566                                            wx.OK | wx.ICON_INFORMATION)
567 
568                     dlg.ShowModal()
569                     dlg.Destroy()
570             # butterworth filters parameters may not have been computed because of the
    critical frequency issue so use try-except
571             try:
572                 # filter data
573                 self.s_filt = signal.lfilter(b,a,self.s)
574                 # calculate vel by integrating acc data
575                 vel_filt = integrate.cumtrapz(self.s_filt,x=None,dx=self.dt,axis=-1,initial =
    0)
576                 vel_filt = vel_filt*980
577                 # calc displacement by integrating disp data
578                 disp_filt = integrate.cumtrapz(vel_filt,x=None,dx=self.dt,axis=-1,initial = 0)
579 
580#                 file2write = open("AccFilt{}{}".format(val,self.gm), "w")
581#                 file2write.write(str(self.order_update)+"\t"+str(self.freq_update))
582#                 for i in range(0,self.numpts):
583#                     file2write.write(str(self.t[i])+"\t"+str(self.s_filt[i])+"\n")
584#                 file2write.close()
585#
586#                 file2write = open("VelFilt{}{}".format(val,self.gm), "w")
587#                 file2write.write(str(self.order_update)+"\t"+str(self.freq_update))
588#                 for i in range(0,self.numpts):
589#                     file2write.write(str(self.t[i])+"\t"+str(vel_filt[i])+"\n")
590#                 file2write.close()
591#
592#                 file2write = open("DispFilt{}{}".format(val,self.gm), "w")
593#                 file2write.write(str(self.order_update)+"\t"+str(self.freq_update))
594#                 for i in range(0,self.numpts):
595#                     file2write.write(str(self.t[i])+"\t"+str(disp_filt[i])+"\n")
596#                 file2write.close()
597 
598                 # clear plot after refresh button clicked
599                 self.axes.clear()
600                 self.axes2.clear()
```

```
601                 self.axes3.clear()
602                 self.RP.clear()
603                 # plot unfiltered time histories and response spectrum
604                 self.draw(self.gm)
605 
606                 # plot filtered acc vs time
607                 self.axes.plot(self.t, self.s_filt,linewidth=1.0,color="black")
608                 self.axes.set_ylabel('Accel (g)')
609                 self.axes.minorticks_on()
610                 self.axes.grid(True)
611 
612                 #plot filtered vel vs time
613                 self.axes2.plot(self.t,vel_filt,linewidth = 0.7,color="black")
614                 self.axes2.set_ylabel('Vel (cm/s)')
615                 self.axes2.minorticks_on()
616                 self.axes2.grid(True)
617 
618                 #plot filtered acc vs time
619                 self.axes3.plot(self.t,disp_filt,linewidth = 0.7,color="black")
620                 self.axes3.set_ylabel('Displacement (cm)')
621                 self.axes3.minorticks_on()
622                 self.axes3.grid(True)
623                 self.axes3.set_xlabel('Time (s)')
624 
625     #             self.calcResponseSpectra(copy.copy(self.s_filt),self.dt,self.numpts)
626     #             self.RP.semilogx(self.p,self.amax,linewidth=0.7,color="black")
627     #             self.RP.set_ylabel("Sa (g)")
628     #             self.RP.set_xlabel("Period (s)")
629     #             self.RP.grid(True)
630                 #draw
631                 self.canvas.draw()
632             except:
633                 pass
634 #
635 #     def UponFilterSelection(self,event):
636 #         if self.cb.GetValue():
637 #             self.filterGM = "butter"
638 #
639 #     def UponPassSelection(self,event):
640 #         if self.cbFC.GetValue():
641 #             self.passtype = "low"
642 #         if self.cbFC2.GetValue():
643 #             self.passtype = "high"
644 
645     def onSelect(self, event):
646         """ bind ground motion selected and get magnitude,
647         date, rupture distance, event, and station from
648         Motions table"""
649         self.txtbox.Clear()
650         self.gm = self.cbGM.GetStringSelection()
651         info = getallinfo(self.db,self.gm)
652         info = info[0]
653         magn = info[0]
654         date = info[1]
655         Rrup = info[2]
656         Repi = info[3]
```

```
657         Rhypo = info[4]
658         event = self.stri(info[5])
659         station = info[6]
660         station = self.stri(station)
661         if Rrup == 0.0:
662             Rrup = "na"
663         if Repi == 0.0:
664             Repi = "na"
665         if Rhypo == 0.0:
666             Rhypo = "na"
667         label = "Moment Magnitude: {}\n Date: {}\n Rrup (km): {}\n Repicenter (km): {}\n
    Rhypocenter (km): {}\n Event: {}\n Station:
    {}".format(magn,date,Rrup,Repi,Rhypo,event,station)
668         return self.txtbox.AppendText(label+"\n")
669 
670     def stri(self,s):
671         """ remove quotation marks before and after string"""
672         if s.startswith('"') and s.endswith('"'):
673             string = s[1:-1]
674         else:
675             string = s
676         return string
677 
678 def show_error():
679     message = ''.join(traceback.format_exception(*sys.exc_info()))
680     dialog = wx.MessageDialog(None, message, 'Error!', wx.OK|wx.ICON_ERROR)
681     dialog.ShowModal()
682 
683 def main():
684     app = wx.App()
685     try:
686         frame = MyAppWindow()
687         frame.Fit()
688         frame.Center()
689         frame.Show()
690         size = frame.GetSize()
691         frame.SetMinSize((size[0]/2,size[1]/2))
692 #         frame.cause_error()
693         app.MainLoop()
694 
695     except:
696         show_error()
697 
698 if __name__ == '__main__':
699     main()
```

```
'''
Created on Dec 9, 2015

@author: cyndi
'''
import sqlite3 as dbi
import sys

# connect to the GM database
def connect():
    print 'accessing ground motion database ...' ,
    try:
        db = dbi.connect('WSGMDB.db')
        print 'success'
    except:
        print 'failed'
        sys.exit()
    return db

def createTables():
    db = connect()

    cu = db.cursor()

    cu.execute("""drop table if exists Motions;""")
    cu.execute("""drop table if exists Events;""")
    cu.execute("""drop table if exists Regions;""")
    cu.execute("""drop table if exists Networks;""")
    cu.execute("""drop table if exists Stations;""")
    cu.execute("""drop table if exists Groups;""")
    cu.execute("""drop table if exists LongDuration;""")

    cu.execute("""create table Motions (
        ID              integer not null primary key autoincrement,
        name            text,
        dt              real,
        numpoints       real,
        eventID         int,
        groupID         int,
        stationID       int,
        Rrup            real,
        Repi            real,
        Rhypo           real
        );""")

    cu.execute("""create table Events (
        ID              integer not null primary key autoincrement,
        directory       text,
        description     text,
        regionID        int,
        date            text,
        magnitude       real
        );""")

    cu.execute("""create table Regions (
        ID              integer not null primary key autoincrement,
```

```
57          name            text,
58          description     text
59          );""")
60
61#      cu.execute("""create table Networks (
62#          ID              integer not null primary key autoincrement,
63#          name            text
64#          );""")
65
66
67      cu.execute("""create table Stations (
68          ID              integer not null primary key autoincrement,
69          name            text
70          );""")
71
72      cu.execute("""create table Groups (
73          ID              integer not null primary key autoincrement,
74          name            text,
75          groupID         text
76          );""")
77
78#      cu.execute("""create table LongDuration (
79#          ID              integer not null primary key autoincrement,
80#          name            text
81#          );""")
82
83  def createGMtables(db,GM):
84#      db = connect()
85      cu = db.cursor()
86      call="""drop table if exists '{}';"""
87      cu.execute(call.format(GM))
88      call = """create table '{}' (
89      ID              integer not null primary key,
90      time            real,
91      accel           real
92      );"""
93      cu.execute(call.format(GM))
94
95  def getGMnames(db):
96      cu = db.cursor()
97      call = """SELECT m.name
98                  FROM Motions as m
99                  ;
100                 """
101     cu.execute(call)
102     gm = cu.fetchall()
103     gm = [i[0] for i in gm]
104     return gm
105
106 def gettime(db,gm):
107     cu=db.cursor()
108     call = """SELECT m.time
109                 FROM '{}' as m
110
111                 ;
112                 """
```

```
113     cu.execute(call.format(gm))
114     time = cu.fetchall()
115     time = [i[0] for i in time]
116     return time
117 
118 def getacc(db,gm):
119     cu=db.cursor()
120     call = """SELECT m.accel
121                 FROM '{}' as m
122                 ;
123                 """
124     cu.execute(call.format(gm))
125     accel = cu.fetchall()
126     accel = [i[0] for i in accel]
127     return accel
128 
129 def getDT(db,gm):
130     cu=db.cursor()
131     call = """SELECT m.time
132                 FROM '{}' as m
133                 WHERE m.ID = 2
134 
135                 ;
136                 """
137     cu.execute(call.format(gm))
138     times = cu.fetchall()
139     times = [i[0] for i in times]
140     dt = times[0]
141     return dt
142 
143 def getnumpts(db,gm):
144     cu=db.cursor()
145     call = """SELECT count(m.accel)
146                 FROM '{}' as m
147                 ;
148                 """
149     cu.execute(call.format(gm))
150     numpts = cu.fetchall()
151     numpts = [i[0] for i in numpts]
152     return numpts
153 # Functions for inserting columns into tables
154 def createGroupsEntry(db,name,groupID):
155     cu = db.cursor()
156     sql_command = """INSERT INTO Groups (name,groupID)
157                     VALUES ('{}','{}')"""
158     cu.execute(sql_command.format(name,groupID))
159 
160 def createRegionsEntry(db,name,description):
161     cu = db.cursor()
162     sql_command = """INSERT INTO Regions (name, description)
163                     VALUES ('{}','{}')"""
164 #     print sql_command.format(name, description)
165     cu.execute(sql_command.format(name, description))
166 
167 def createNetworksEntry(db,name):
168     cu = db.cursor()
```

```
169     sql_command = """INSERT INTO Networks (name)
170                      VALUES ('{}')"""
171     cu.execute(sql_command.format(name))
172 
173 def createStationsEntry(db,name):
174     cu = db.cursor()
175     sql_command = """INSERT INTO Stations (name)
176                      VALUES ('{}')"""
177     cu.execute(sql_command.format(name))
178 
179 def createEventsEntry(db,dire,description,regionID,date,magnitude):
180     cu = db.cursor()
181     sql_command = """INSERT INTO Events (directory, description, regionID, date, magnitude)
182                      VALUES ('{}','{}',{},'{}',{})"""
183     cu.execute(sql_command.format(dire,description,regionID,date,magnitude))
184 
185 def createGMEntry(db,filename,time,accel):
186     cu = db.cursor()
187     sql_command = """INSERT INTO '{}' (time,accel)
188                      VALUES ({},{})"""
189     cu.execute(sql_command.format(filename,time,accel))
190 
191 def createMotionsEntry(db,name,eventID,groupID,stationID,Rrup,Repi,Rhypo):
192     cu = db.cursor()
193     sql_command = """INSERT INTO Motions (name,eventID,groupID,stationID,Rrup,Repi,Rhypo)
194                      VALUES ('{}',{},{},{},{},{},{})"""
195     cu.execute(sql_command.format(name,eventID,groupID,stationID,Rrup,Repi,Rhypo))
196 
197 def get_eventID(db,dire):
198 #     db.commit()
199     cu = db.cursor()
200 
201     call = """
202     SELECT e.ID
203         FROM Events as e
204         WHERE e.directory = "{}"
205         ;
206     """
207 #     print call.format(dire)
208     cu.execute(call.format(dire))
209     eventID = cu.fetchall()
210     eventID = [int(i[0]) for i in eventID]
211     eventID = eventID[0]
212     return eventID
213 
214 def get_groupID(db,groupID):
215     #     db.commit()
216     cu = db.cursor()
217 
218     call = """
219     SELECT g.ID
220         FROM Groups as g
221         WHERE g.groupID = "{}"
222         ;
223     """
224 #     print call.format(groupID)
```

```
225     cu.execute(call.format(groupID))
226     ID = cu.fetchall()
227     ID = [int(i[0]) for i in ID]
228     ID = ID[0]
229     return ID
230 
231 def get_stationID(db,stationID):
232     cu = db.cursor()
233     call = """SELECT s.ID
234                 FROM Stations as s
235                 WHERE s.name = "{}"
236                 ;
237                 """
238 #     print call.format(stationID)
239     cu.execute(call.format(stationID))
240     ID = cu.fetchall()
241     ID = [int(i[0]) for i in ID]
242     ID = ID[0]
243     return ID
244 
245 def getallinfo(db,gm):
246     db.text_factory = str
247     cu = db.cursor()
248     call = """SELECT e.magnitude, e.date, m.Rrup, m.Repi, m.Rhypo, e.description, s.name
249         FROM events as e,
250              motions as m,
251              stations as s
252         WHERE m.name = "{}"
253                 and m.eventID = e.ID
254                 and m.stationID = s.ID
255                 ;
256                 """
257     cu.execute(call.format(gm))
258     info = cu.fetchall()
259 #     info = [i[0] for i in info]
260 #     print info
261     return info
262 
263 def getRegionID(db,name):
264     cu = db.cursor()
265     if name == "ALL":
266         regionID = name
267     else:
268         call = """SELECT r.ID
269                 FROM regions as r
270                 WHERE r.name = "{}"
271                 ;
272                 """
273         cu.execute(call.format(name))
274         regionID = cu.fetchall()
275         regionID = [i[0] for i in regionID]
276         regionID = regionID[0]
277     return regionID
278 
279 def getGroupID(db,name):
280     cu = db.cursor()
```

```
    if name == "ALL":
        groupID = name
    else:
        call = """SELECT g.ID
                FROM groups as g
                WHERE g.name = "{}"
                ;
                """
        cu.execute(call.format(name))
        groupID = cu.fetchall()
        groupID = [i[0] for i in groupID]
        groupID = groupID[0]
    return groupID


def getSelectGM(db,region,group,magn1,magn2):
    cu=db.cursor()
    if region == "ALL":
        if group == "ALL":
            call = """SELECT m.name
            FROM Motions as m,
                events as e
            WHERE m.eventID = e.ID
                and e.magnitude >= {}
                and e.magnitude <= {}
            ;
            """
            cu.execute (call.format(magn1,magn2))
        else:
            call = """SELECT m.name
            FROM Motions as m,
                events as e,
                groups as g
            WHERE m.eventID = e.ID
                and e.magnitude >= {}
                and e.magnitude <= {}
                and g.ID = {}
                and m.groupID = g.ID
            ;
            """
            cu.execute(call.format(magn1,magn2,group))

    elif group == "ALL":
        call = """SELECT m.name
        FROM Motions as m,
            events as e,
            regions as r
        WHERE m.eventID = e.ID
            and e.magnitude >= {}
            and e.magnitude <= {}
            and r.ID = {}
            and e.regionID = r.ID
        ;
        """
        cu.execute(call.format(magn1,magn2,region))

```

```
337     else:
338         call = """SELECT m.name
339             FROM motions as m,
340                 regions as r,
341                 groups as g,
342                 events as e
343             WHERE r.ID = {}
344                 and e.regionID = r.ID
345                 and g.ID = {}
346                 and m.groupID = g.ID
347                 and m.eventID = e.ID
348                 and e.magnitude >= {}
349                 and e.magnitude <= {}
350                 ;
351                 """
352         cu.execute(call.format(region,group,magn1,magn2))
353     gm = cu.fetchall()
354     gm = [i[0] for i in gm]
355     return gm
356 
357 # database = "WSGMDB.db"
358 # db = dbi.connect(database)
359 # print gettime(db,'vquez090.AT2')
360 
361 
362 
363 
364 
365 
```