```
from tkinter import *
from tkinter import filedialog
from numpy import *
import numpy.matlib
import matplotlib.pyplot as plt
import os, sys
import csv
import time as systime
from tkinter import ttk



# Visual theme controls
fonts = 12
titlefonts = ("Helvetica", 11)
fonts1 = ("helvetica", 11)
fonts2 = ("ariel", 12, "bold")
monofont1 = ("monospace",12)
monofont2 = ("monospace",12,"underline")

brs = ("Hevetica", 10)
runcmnds = ("Helvetica", 9)
background1 = '#c5ddeb'
bkgr2 = "white"



#### Funcion Block For GUI Interface (beg) ####

# This class and subsequent function allows the simplified creation of explanation
text on widgest aa
# copied from http://www.voidspace.org.uk/python/weblog/arch_d7_2006_07_01.shtml#e387
class ToolTip(object):
    def __init__(self, widget):
        self.widget = widget
        self.tipwindow = None
        self.id = None
        self.x = self.y = 0
    def showtip(self, text):
        "Display text ub tooltip window"
        self.text = text
        if self.tipwindow or not self.text:
            return
        x, y, cx, cy = self.widget.bbox("insert")
        x = x + self.widget.winfo_rootx() + 27
        y = y + cy + self.widget.winfo_rooty() + 27
        self.tipwindow = tw = Toplevel(self.widget)
        tw.wm_overrideredirect(1)
        tw.wm_geometry("+%d+%d" % (x, y))
        try:
            # For Mac OS
            tw.tk.call("::tk::unsupported::MacWindowStyle",
                       "style", tw._w,
                       "help", "noActivates")
        except TclError:
            pass
        label = Label(tw, text=self.text, justify=LEFT,
                      background="#ffffe0", relief=SOLID, borderwidth=1,
                      font=("tahoma", "8", "normal"))
        label.pack(ipadx=1)
```

```
    def hidetip(self):
        tw = self.tipwindow
        self.tipwindow = None
        if tw:
            tw.destroy()

def createToolTip(widget, text):
    toolTip = ToolTip(widget)
    def enter(event):
        toolTip.showtip(text)
    def leave(event):
        toolTip.hidetip()
    widget.bind('<Enter>', enter)
    widget.bind('<Leave>', leave)


# function reloads gui (used when user picks new number of minerals)
def relo(event):
    FileHand.grid()
    ModelChar.grid()
    RockChar.grid()
    GraphChar.grid()
    nono.grid_remove()

# used when saving model params (incase a field is empty is saves it as a 0)
def getval(widget):
    try:
        a=widget.get()
    except TclError:
        a=0
    return a

# used to make sure model params its saving are non zero
def checkvalid():
    test=1
    test=test*getval(duration)
    test=test*getval(timestep)
    test=test*getval(modelend)
    test=test*getval(modelstart)

    if test==1:
        print('good')
    else:
        print('bad')

# removes unnecessary entry fields (when number of minerals is changed)
def removel(x):
    x=int(x)
    rocksl[x].grid_remove()
    for i in range(1,12):
        Rocks[x][i].grid_remove()


# adds entry fields (when number of minerals is changed)
def addl(x):
    x=int(x)
    rocksl[x].grid()
    for i in range(1,12):
        Rocks[x][i].grid()

def importf():
    a = filedialog.askopenfilename(filetypes=(("Plain Text (.txt)",".txt"),("all
```

```
files","*.*")),defaultextension=".txt")
    importb.set(a)

def exportf():
    a = filedialog.asksaveasfilename(filetypes=(("Plain Text (.txt)",".txt"),("all
files","*.*")),defaultextension=".txt")
    exportb.set(a)

def poss_cool(event):
    print(1)
    if cooling.get()=="Custom":
         print(2)
         global cool_file
         cool_file = filedialog.askopenfilename(filetypes=(("Plain Text
(.txt)",".txt"),("all files","*.*")),defaultextension=".txt")


def readjustn():
    a=nummin.get()
    for i in range(2,a+1):
        addl(i)
        lookupb[i].grid()
        lookupbb[i].grid()
        GraphCheck[i].grid()
        Graph[i].grid()
    lookupbb[1].grid()

    for i in range(a+1,9):
        removel(i)
        lookupb[i].grid_remove()
        lookupbb[i].grid_remove()
        GraphCheck[i].grid_remove()
        Graph[i].grid_remove()

def readjust(event):
    readjustn()

def writedata():
    nm = exportb.get()
    fileobj = open(nm, "w")

    dat = ""
    dat = dat + " " +str(nummin.get())
    dat = dat + " " +str(duration.get())
    dat = dat + " " +str(timestep.get())
    dat = dat + " " +str(cooling.get())
    dat = dat + " " +str(modelstart.get())
    dat = dat + " " +str(modelend.get())
    dat = dat + " " +str(wrd180t.get())
    fileobj.write(dat)

    dat = " "
    for i in range(1,nummin.get()+1):
        for j in range(1,12):
            dat = dat+" "+str(rocks[i][j].get())


    fileobj.write(dat)
    fileobj.close()

def readdata():
    nm = importb.get()
```

```
    fileobj = open(nm, "r")
    dat = fileobj.read()
    dat = dat.split()

    nummin.set(int(dat[0]))
    duration.set(float(dat[1]))
    timestep.set(float(dat[2]))
    cooling.set(dat[3])
    modelstart.set(float(dat[4]))
    modelend.set(float(dat[5]))
    wrd180t.set(float(dat[6]))

    k=7
    for i in range(1,int(dat[0])+1):
        for j in range(1,12):
            rocks[i][j].set(dat[k])
            k=k+1
    readjustn()

def cbrowse():
    a=filedialog.asksaveasfilename(filetypes=(("Comma Sperated File (.csv)",".csv"),
("all files","*.*")),
                                   defaultextension=".csv")
    csvfile.set(a)

def csave():
    if len(csvfile.get())>0:
        MatSave=yresult
        MatSave=moveaxis(MatSave,2,0)
        MatSave=reshape(MatSave,(MatSave.shape[0],-1),order='F')
        MatSave=concatenate((repeat(timeresult,nummin.get()).reshape
(1,-1),MatSave),axis=0)
        MatSave=concatenate((concatenate((zeros
((1,4)),xresult),axis=0),MatSave),axis=1)
        savetxt(csvfile.get(),MatSave,delimiter=",")
    else:
        warningwin = Toplevel()
        warningwin.config(bg=background1)
        warninglab = Label(warningwin, text='Please choose a valid file path
first.',font=("Helvetica", 13, "bold"))
        warninglab.config(bg=background1)
        warninglab.grid(row=1, column=1, padx=(12,12), pady=(12,12))


def cformat():
    format_win = Toplevel()
    format_win.wm_title('Format used for saved CSV file')
    format_win.config(bg=background1)
    formwind=Frame(format_win, bg='black')
    my_mess="The table below shows the layout for how the simulation data is saved.
This is an example for a model run with 4 minerals. X_i is the vector of depths (cm)
for mineral i, and yi(t_i) is the corresponding delta-18 values at time t_i."
    T=Message(format_win,text=my_mess,font=("helvetica",13), bg=background1,
width=800)
    T.grid(row=0,column=0, sticky=W+E)
    formwind.grid(row=1, column=0, padx=(5,5), pady=(5,5))

    cback='white'
    cfont=("helvetica",13)
    cxpad=(1,1)
    cypad=(1,1)
    Label(formwind,text='0', font=cfont, bg=cback).grid(row=2, column=1, padx=cxpad,
```

```
pady=cypad, stick=N+W+S+E)
    Label(formwind,text='0', font=cfont, bg=cback).grid(row=2, column=2, padx=cxpad,
pady=cypad, stick=N+W+S+E)
    Label(formwind,text='0', font=cfont, bg=cback).grid(row=2, column=3, padx=cxpad,
pady=cypad, stick=N+W+S+E)
    Label(formwind,text='0', font=cfont, bg=cback).grid(row=2, column=4, padx=cxpad,
pady=cypad, stick=N+W+S+E)
    Label(formwind,text='x_1', font=cfont, bg=cback).grid(row=3, column=1,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='x_2', font=cfont, bg=cback).grid(row=3, column=2,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='x_3', font=cfont, bg=cback).grid(row=3, column=3,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='x_4', font=cfont, bg=cback).grid(row=3, column=4,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_0', font=cfont, bg=cback).grid(row=2, column=5,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_0', font=cfont, bg=cback).grid(row=2, column=6,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_0', font=cfont, bg=cback).grid(row=2, column=7,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_0', font=cfont, bg=cback).grid(row=2, column=8,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_1', font=cfont, bg=cback).grid(row=2, column=9,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_1', font=cfont, bg=cback).grid(row=2, column=10,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_1', font=cfont, bg=cback).grid(row=2, column=11,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_1', font=cfont, bg=cback).grid(row=2, column=12,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_2', font=cfont, bg=cback).grid(row=2, column=13,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_2', font=cfont, bg=cback).grid(row=2, column=14,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_2', font=cfont, bg=cback).grid(row=2, column=15,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='t_2', font=cfont, bg=cback).grid(row=2, column=16,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y1(t_0)', font=cfont, bg=cback).grid(row=3, column=5,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y2(t_0)', font=cfont, bg=cback).grid(row=3, column=6,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y3(t_0)', font=cfont, bg=cback).grid(row=3, column=7,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y4(t_0)', font=cfont, bg=cback).grid(row=3, column=8,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y1(t_1)', font=cfont, bg=cback).grid(row=3, column=9,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y2(t_1)', font=cfont, bg=cback).grid(row=3, column=10,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y3(t_1)', font=cfont, bg=cback).grid(row=3, column=11,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y4(t_1)', font=cfont, bg=cback).grid(row=3, column=12,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y1(t_2)', font=cfont, bg=cback).grid(row=3, column=13,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y2(t_2)', font=cfont, bg=cback).grid(row=3, column=14,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y3(t_2)', font=cfont, bg=cback).grid(row=3, column=15,
padx=cxpad, pady=cypad, stick=N+W+S+E)
    Label(formwind,text='y4(t_2)', font=cfont, bg=cback).grid(row=3, column=16,
```

```
padx=cxpad, pady=cypad, stick=N+W+S+E)
    for i in range(1,2):
       for j in range(1,17):
           Label(formwind, text='|', font=cfont, bg=cback).grid(row=i+3, column=j,
stick=N+W+S+E, padx=cxpad)
    for j in range(1,17):
           Label(formwind, text='|', font=cfont, bg=cback).grid(row=i+5, column=j,
stick=N+W+S+E, padx=cxpad, pady=(0,1))



#### Function Block for GUI Interface (end) ####



### Creation Block for fractionation popup GUI (beg)###

def frac_search(but_num):

    def find_path(list1,list2,node1,node2):
        #find unique connection points
        n=len(list1)
        edges=list(set((a,b) if a<b else (b,a) for a,b in zip(list1,list2)))

        #initialize sets
        checked=set([])
        work_chains=[[node1]]
        final_path=[]

        #perform loops for Breadth-First vector style search algorithm
        while(len(work_chains)>0):
             cchain=work_chains[0]
             cnode=cchain[-1]
             checked=checked.union([cnode])
             nextsteps=list(set([x for y,x in edges if y==cnode and x!=y]+[x for x,y
in edges if y==cnode and x!=y])-checked)
             del work_chains[0]
             for k in range(0,len(nextsteps)):
                 if nextsteps[k]==node2:
                      final_path=cchain+[node2]
                      checked=checked.union([nextsteps[k]])
                 else:
                      checked=checked.union([nextsteps[k]])
                      work_chains.append(cchain+[nextsteps[k]])

        #start complete search (find all versions of edges)
        all_steps=[]
        for k in range(0,len(final_path)-1):
             min1=final_path[k]
             min2=final_path[k+1]
             pairs=[(x,y) for x,y in zip(list1,list2)]
             poss_steps=[i for i in range(0,n) if (min1,min2)==pairs[i]]
             poss_steps=poss_steps+[-i for i in range(0,n) if (min2,min1)==pairs[i]]
             all_steps.append(poss_steps)

        return all_steps



    def create_menus(fracdata,node1,node2):
```

```
        #use path finder to find current best path
        list1=[x[1] for x in fracdata]
        list2=[x[2] for x in fracdata]
        path=find_path(list1,list2,node1,node2)

        #clear all previous elements so new window can be created
        for child in mainwin.winfo_children():
             child.destroy()

        #used when table entries can't be found and allows manual entry
        def enter_manual(num):
            for j in [6,7,8]:
                Rocks[num][j].grid_remove()
                Rocks[num][j]=Entry(RockChar, textvariable=rocks[num][j], width=5,
font=fonts1)
                Rocks[num][j].grid(row=num, column=j+1, sticky=N+S+W+E)
            mainwin.destroy()

        #place manual button
        Man=Button(mainwin, text="Enter Manually", relief="groove", bg=background1)
        Man.bind('<Button-1>', lambda event: enter_manual(but_num))
        Man.grid(row=1, column=2, columnspan=2, pady=(0,10))

        #check that there is a valid path
        if len(path)<1:
            direc=Label(mainwin, text='There does not exist any remaining path of
studies that can link this mineral to your monitor.')
            direc.config(font=fonts2, bg=background1)
            direc.grid(row=0, column=2, padx=(10,10), pady=(10,10))
            return
        else:
            direc=Label(mainwin, text='I have the following conversions available for
mineral --> monitor')
            direc.config(font=fonts2, bg=background1)
            direc.grid(row=0, column=0, pady=(0,0), columnspan=2)
            sep_top = Label(mainwin,
text="-----------------------------------------------------------------------------------------------
            sep_top.config(font=fonts1, bg=background1)
            sep_top.grid(row=1, column=0, columnspan=2, pady=(0,10))

        #create step locations and then flatten path list
        allpaths=path
        pathssize=[len(x) for x in allpaths]
        path=[x for y in path for x in y]
        n=len(path)
        m=len(allpaths)

        #swap any names and change sign of A,b,c values for negative path numbers
        for k in path:
            if k<0:
                k=abs(k)
                fracdata[k][1:3]=fracdata[k][2:0:-1]
                fracdata[k][7:10]=[str(-1*float(x)) for x in fracdata[k][7:10]]
                fracdata[k][3]=fracdata[k][3].split('-')[1]+'-'+fracdata[k][3].split
('-')[0]
        path=[abs(x) for x in path] #make all pos now that table has been adjusted


        #create nice-looking equal length strings for display
        path=[0]+path
        opt=['' for x in path]
        optlist=[3,7,8,9,10,4]
```

```
        for k in optlist:
                #place approp white space inbetween
                optadd=[fracdata[abs(x)][k] for x in path]
                lengths=[len(x) for x in opt]
                maxl=max(lengths)+1+2*(j>1)
                opt = [opt[x] + ' ' * (maxl - len(opt[x])) + optadd[x] for x in range
(0, len(opt))]

                #add space after all placed
                if k==optlist[-1]:
                        lengths=[len(x) for x in opt]
                        maxl=max(lengths)+1
                        opt=[opt[x]+' '*(maxl-len(opt[x])) for x in range(0,len(opt))]

        #create vector to hold choices for each mineral
        choices=zeros(len(allpaths))-1
        rem_list=[]

        #functions to handle choosing and colver of hover
        def lock_choice(jump,jumpchoice):
                if choices[jump]>-1:
                     unlock_choice(jump,int(choices[jump]))
                path_opt[jump][jumpchoice].unbind('<Leave>')
                path_opt[jump][jumpchoice].config(bg='#A0A0A0')
                path_opt[jump][jumpchoice].bind('<Button-1>', lambda event, i=jump,
j=jumpchoice: unlock_choice(i,j))
                choices[jump]=jumpchoice
                if (choices>-1).all():
                     fin_but[1].config(state='normal')

        def unlock_choice(jump,jumpchoice):
                choices[jump]=-1
                fin_but[1].config(state='disabled')
                path_opt[jump][jumpchoice].bind('<Leave>', lambda event, i=jump,
j=jumpchoice: change_back(i,j))
                path_opt[jump][jumpchoice].bind('<Button-1>', lambda event, i=jump,
j=jumpchoice: lock_choice(i,j))
                path_opt[jump][jumpchoice].config(bg=background1)

        def change_to(jump,jumpchoice):
                path_opt[jump][jumpchoice].config(bg='#A0A0A0')

        def change_back(jump,jumpchoice):
                path_opt[jump][jumpchoice].config(bg=background1)

        def notes_look(jump,jumpchoice):
             notes_but[jump][jumpchoice].config(relief="sunken")
             notes_win=Toplevel()
             notes_win.config(bg=background1)
             curr_notes=fracdata[allpaths[jump][jumpchoice]][11]
             T=Message(notes_win, text=curr_notes.replace('^',','), width=600)
             T.config(bg=background1, font=monofont1)
             #S=Scrollbar(notes_win)
             #T.insert(END, curr_notes.replace('^',','))
             T.grid(row=0, column=0, padx=(8,8), pady=(8,8))
             #S.grid(row=0, column=1)
             #S.config(command=T.yview)
             #T.config(yscrollcommand=S.set)

        def unclick_note(jump,jumpchoice):
             notes_but[jump][jumpchoice].config(relief="groove")
```

```
        def change_note_to(jump,jumpchoice):
             notes_but[jump][jumpchoice].config(bg='#ececec')

        def change_note_back(jump,jumpchoice):
             notes_but[jump][jumpchoice].config(bg=background1)

        #make label defined at top
        Fracheader=Label(mainwin, text=opt[0])
        Fracheader.config(font=monofont2, bg=background1)
        Fracheader.grid(row=2, column=1, columnspan=2, sticky=W)
        Label(mainwin, text='Notes', font=monofont2, bg=background1).grid(row=2,
column=2, sticky=W, padx=(8,55))

        #create all buttons (as Labels so binds can be manually controlled and
updated without .destroy())
        path_opt=dict()
        step_lab=dict()
        rem_checks=dict()
        Rem_checks=dict()
        notes_but=dict()


        rowtrack=3
        for jump in range(0,m):
                #create label for current step
                step_lab[jump]=Label(mainwin,text=fracdata[abs(allpaths[jump][0])][1]
+' --> '+fracdata[abs(allpaths[jump][0])][2])
                step_lab[jump].grid(row=rowtrack, column=1, sticky=W)
                step_lab[jump].config(font=("helvetica",11,"bold"), bg=background1)
                rowtrack=rowtrack+1

                #create buttons for each option for current step
                path_opt[jump]=dict()
                rem_checks[jump]=dict()
                Rem_checks[jump]=dict()
                notes_but[jump]=dict()
                for jumpchoice in range(0,pathssize[jump]):
                        # create checkmarks for delete buttons
                        rem_checks[jump][jumpchoice]=IntVar(mainwin)
                        rem_checks[jump][jumpchoice].set(0)
                        Rem_checks[jump][jumpchoice]=Checkbutton(mainwin, text="",
bg=background1, variable=rem_checks[jump][jumpchoice])
                        Rem_checks[jump][jumpchoice].grid(row=rowtrack, column=0)

                        #create main button and place
                        path_opt[jump][jumpchoice]=Label(mainwin, text=opt[int
(jumpchoice+sum(pathssize[0:jump])+1)], relief="groove")
                        path_opt[jump][jumpchoice].config(font=monofont1,
bg=background1)
                        path_opt[jump][jumpchoice].grid(row=rowtrack, column=1,
stick=W, padx=(0,10))

                        #note button and place
                        notes_but[jump][jumpchoice]=Label(mainwin, text="---",
font=monofont1, relief="groove", bg=background1)
                        notes_but[jump][jumpchoice].grid(row=rowtrack, column=2, padx=
(10,0), sticky=N+S+W)

                        #make all appropriate path binds
                        path_opt[jump][jumpchoice].bind('<Enter>', lambda event,
i=jump, j=jumpchoice: change_to(i,j))
                        path_opt[jump][jumpchoice].bind('<Leave>', lambda event,
```

```
i=jump, j=jumpchoice: change_back(i,j))
                        path_opt[jump][jumpchoice].bind('<Button-1>', lambda event,
i=jump, j=jumpchoice: lock_choice(i,j))

                        #make all appropriate notes binds
                        notes_but[jump][jumpchoice].bind('<Button-1>', lambda event,
i=jump, j=jumpchoice: notes_look(i,j))
                        notes_but[jump][jumpchoice].bind('<Enter>', lambda event,
i=jump, j=jumpchoice: change_note_to(i,j))
                        notes_but[jump][jumpchoice].bind('<Leave>', lambda event,
i=jump, j=jumpchoice: change_note_back(i,j))
                        notes_but[jump][jumpchoice].bind('<ButtonRelease-1>', lambda
event, i=jump, j=jumpchoice: unclick_note(i,j))

                        #update row tracker
                        rowtrack=rowtrack+1

        #create final run with current path function
        def use_curr():
            need_sub=[abs(allpaths[x][int(choices[x])]) for x in range(0,m)][::-1]
            final_factors=[float(x) for x in fracdata[need_sub[0]][7:10]]
            for i in range(1,m):
                final_factors=[float(fracdata[need_sub[i]][x+7])-final_factors[x] for
x in range(0,3)]
            #set fractionation factor values
            for i in range(0,3):
                rocks[but_num][6+i].set(final_factors[i])
            #destory window now that values are set
            mainwin.destroy()

        #create final removal function
        def rem_curr():
            need_remove=[]
            for jump in range(0,m):
                for jumpc in range(0,pathssize[jump]):
                    if(rem_checks[jump][jumpc].get()==1):
                        table_index=abs(allpaths[jump][jumpc])
                        need_remove.append(table_index)
            need_remove=sorted(need_remove)[::-1] #put list in reverse order to
handle deletes
            for i in need_remove:
                del fracdata[i]
            #now rerun whole script without these elements
            create_menus(fracdata,node1,node2)


        #create two final buttons
        fin_but=dict()
        fin_but[1]=Button(mainwin, text="Use Current Mineral Studies Chosen",
bg=background1, command=use_curr)
        fin_but[1].config(state='disabled')
        fin_but[2]=Button(mainwin, text="Remove", bg=background1, command=rem_curr)


        #place buttons
        fin_but[1].grid(row=rowtrack+1, column=1, stick=W, pady=(0,10))
        fin_but[2].grid(row=rowtrack+1, column=0, padx=(8,8), pady=(0,10))
        Man.grid(row=rowtrack+1, column=0)

        #place division line
        sep_bot = Label(mainwin,
text="----------------------------------------------------------------")
```

```
        sep_bot.config(font=fonts1, bg=background1)
        sep_bot.grid(row=rowtrack, column=0, columnspan=2, padx=(14,0), pady=(0,0),
sticky=W)




    a='FractionationFactorsR.csv'
    file = open(a, 'r', encoding='ISO-8859-1')
    raw=file.read()
    rawlines=raw.split('\n')
    rawlines=[x for x in rawlines]
    n=len(rawlines)-1
    fractiondata=[x.split(',') for x in rawlines[0:n]]

    #divide factors a and b by 10^6 and 10^3
    for x in fractiondata[1:n]:
        x[7]=str(float(x[7])/10**6)
        x[8]=str(float(x[8])/10**3)



    #print([x for y in find_path(nameA,nameB,'biotite','wolframite') for x in y])



    #make main window and run function to create all buttons
    mainwin = Toplevel()
    mainwin.wm_title('Find Fractionation Values')
    mainwin.config(bg=background1)
    #create_menus(fractiondata,'biotite','wolframite')
    create_menus(fractiondata,rocks[but_num][1].get(),rocks[1][1].get())


### Creation Block for popup Fraction GUI (end)###









### Creation Block for popup Diffusion GUI (beg)###

def diff_search(but_num):

    #used when table entries can't be found and allows manual entry
    def enter_manual(num):
        for j in [9,10]:
            Rocks[num][j].grid_remove()
            Rocks[num][j]=Entry(RockChar, textvariable=rocks[num][j], width=5,
font=fonts1)
```

```
            Rocks[num][j].grid(row=num, column=j+2, sticky=N+S+W+E)
        mainwin.destroy()

    #change color of hover over
    def change_to(num):
        Possible_options[num].config(bg='#A0A0A0')

    def change_back(num):
        Possible_options[num].config(bg=background1)

    #used for last selection to pull values from table
    def set_diff_params(num):
        Ea=diffdata[num][5]
        if len(Ea)>0:
            rocks[but_num][10].set(float(Ea))
        else:
            rocks[but_num][10].set(0)
        rocks[but_num][9].set(1000*float(diffdata[num][6]))
        mainwin.destroy()


    #used after conversions and table search has been done.
    def final_options(num):
        Diffheader.grid_remove()
        if num>0:
            readjust_diff(1)
            direc.config(text="Would you like to use your current selection or enter
manually?")
            possible_options[1].set('Use Current Selection')
            Possible_options[1].bind('<Button-1>', lambda event: set_diff_params(num))

        else:
            direc.config(text="I could not find any table entries for this mineral.")
            readjust_diff(0)

    # set what options are available based on conversions
    def set_options(curr_min):
        if curr_min in conversions:
            options_list = conversions[curr_min]
            numopt = size(options_list)
            for i in range(1, numopt + 1):
                possible_options[i].set(options_list[i-1])
            readjust_diff(numopt)
        else:
            readjust_diff(0)
            mat = find_in_table(curr_min)
            if len(mat)>1:
                direc.config(text="I have the following table entries for this
mineral:")

                #create same length option lines
                opt = ['' for x in mat]

                optlist=[1,2,3,4,5,6,8,10]
                for j in optlist:
                    optadd = [diffdata[x][j] for x in mat]
                    lengths = [len(x) for x in opt]
                    maxl = max(lengths)+1+2*(j>1)

                    opt = [opt[x] + ' ' * (maxl - len(opt[x])) + optadd[x] for x in
range(0, len(opt))]
                    if j == optlist[-1]:
```

```
                        lengths=[len(x) for x in opt]
                        maxl=max(lengths)+1
                        opt = [opt[x]+' '*(maxl-len(opt[x])) for x in range(0,len
(opt))]

                #make header lable bar
                Diffheader.config(text=opt[0])
                Diffheader.grid()


                for i in range(0, min(25,size(mat)-1)):
                    possible_options[i+1].set(opt[i+1])
                    Possible_options[i+1].bind('<Button-1>', lambda event, i=i:
final_options(mat[i+1]))

                #after buttons have been made final options, adjust number of buttons
                readjust_diff(min(25, size(mat)))
                readjust_diff(min(25, size(mat)-1))
            else:
                final_options(0)


    # add or removed unused options
    def readjust_diff(num):
        for i in range(1, num + 1):
            Possible_options[i].grid(row=i+3, column=0, pady=(5*(i<2),0-0*(i<num)),
padx=(8,8))
        for i in range(num + 1, 26):
            Possible_options[i].grid_remove()
        #if num>0:
        #    Possible_options[num].grid(row=num+3, column=0, pady=(20*(num<2),10))

    # choose current selection and readjust options
    def choose_opt(num):
        name_selec = possible_options[num].get()
        curnam = selection.get()
        selection.set(curnam+' > '+name_selec)

        set_options(name_selec)

    #search for mineral name in table
    def find_in_table(min_name):
        matches = [0]
        for i in range(0, len(diffdata)-1):
            if str(min_name.upper()) in diffdata[i][1].upper():
                matches.append(i)
        return matches


    mainwin = Toplevel()
    mainwin.wm_title('Find Diffusion Parameters for Mineral')
    mainwin.config(bg=background1)

    curr_min = rocks[but_num][1].get()
    selection = StringVar(mainwin)
    selection.set('Current mineral: '+curr_min)
    Selection = Label(mainwin, textvariable=selection)
    Selection.config(font=titlefonts, bg=background1)
    direc = Label(mainwin, text="For your class of minerals I have the following
options for subclass':")
    direc.config(font=fonts2, bg=background1)
```

```
    possible_options = dict()
    Possible_options = dict()
    for i in range(1,27):
        possible_options[i] = StringVar(mainwin)
        possible_options[i].set('Option'+str(i))
        Possible_options[i] = Label(mainwin, textvariable=possible_options[i],
relief="groove")
        Possible_options[i].config(font=monofont1, bg=background1)

    for i in range(1,26):
        Possible_options[i].bind('<Button-1>', lambda event, i=i: choose_opt(i))

    for i in range(1,27):
        Possible_options[i].bind('<Enter>', lambda event, i=i: change_to(i))

    for i in range(1,27):
        Possible_options[i].bind('<Leave>', lambda event, i=i: change_back(i))

    possible_options[26].set(' Enter Manually ')
    Possible_options[26].bind('<Button-1>', lambda event: enter_manual(but_num))
    sep_line = Label(mainwin,
text="-------------------------------------------------------------------------------------------")
    sep_line.config(font=fonts1, bg=background1)


    sep_line.grid(row=28, column=0)
    Possible_options[26].grid(row=29, column=0, pady=(2,10))
    Selection.grid(row=1, column=0, sticky=W)
    direc.grid(row=2, column=0, pady=(0,20))

    #create header for table entries
    Diffheader = Label(mainwin, text='', font=monofont2, bg=background1)
    Diffheader.grid(row=3, pady=(0, 0))
    Diffheader.grid_remove()

    #start process of finding parameters
    if (len(curr_min)>0):
        set_options(curr_min)
    else:
        direc.config(text='Please give mineral name to search for diffusion
parameters.')


#load diffusion CSV, add b for binary read
a='ODiffusionR.csv'
#file = open(a, 'r')
file = open(a, 'r', encoding='ISO-8859-1')
raw=file.read()
rawlines=raw.split('\n')
diffdata=[x.split(',') for x in rawlines]
diffdata=[[x.replace(':',',') for x in y] for y in diffdata]
diffdata=(diffdata[0:1]+[(y[0:2]+[x.replace(' ','') for x in y[2:]]) for y in diffdata
[1:]])


### Creation Block for popup Diffusion GUI (end)###
```

```
#### Function Block for Diffusion Solver (beg) ####
def fluxbal(z, e, f, h, j, k, l):
    X = zeros([z])
    A = zeros([z, z])
    for m in range(0, z - 1):
        A[m, 0] = 1
        A[m, m + 1] = -1
    A[z - 1, :] = j * k * l * f
    B = zeros([z])
    B[0:z - 1] = h[1:z]
    B[z - 1] = sum(j * k * l * f * e)
    X = linalg.solve(A, B)
    return X


def octrave(b):
    file = open(b, 'r')
    str = file.read()
    rows = str.split('\n')
    rowb = len(rows)
    colb = len(rows[0].split('\t'))
    dat = zeros([rowb, colb])
    for i in range(0, rowb):
        dat[i, :] = rows[i].split('\t')
    if colb == 3:
        distance = dat[:, colb - 1]
    else:
        print('et')
        X0 = dat[0, colb - 2]
        Y0 = dat[0, colb - 1]
        ##select traverse origin
        origin = zeros([colb])
        origin[colb - 2] = X0
        origin[colb - 1] = Y0
        OR = numpy.matlib.repmat(origin, rowb, 1)
        # referance all coordinates to newly selected origin
        C = dat - OR
        N = C[:, colb - 2:colb]
        travend = C[rowb - 1, colb - 2:colb]
        distance = zeros([rowb])
        for i in range(0, rowb):
            distance[i] = dot(N[i, :], travend) / sqrt(pow(travend[0], 2) + pow
(travend[1], 2))
    return [distance, dat[:, 0], dat[:, 1]]
#### Function Block for Diffusion Solver (end) ####


### Create conversion table for mineral searchs
conversions = dict()
conversions['feldspar'] = ['plagioclase','kfeldspar','KAISi3O8']
conversions['plagioclase'] = ['albite','anorthite']
conversions['kfeldspar'] = ['microcline','orthoclase','sanidine']

conversions['clay'] = ['kaolinite','smectite','inite']




# This is the main diffusion solver function
```

```
def runmain():
    start_time=systime.time()
    loading.grid(row=3, column=1, columnspan=1, sticky=W)
    root.update()

    # get user defined info
    ttot = float(duration.get())
    dt = float(timestep.get())
    WRd180 = float(wrd180t.get())
    Tstart = float(modelstart.get())
    Tend = float(modelend.get())
    nmin = int(nummin.get())
    de = 100

    if cooling.get() == "Custom":
        #read data in as matrix without using pandas
        file=open(cool_file,'r',encoding='ISO-8859-1')
        raw=file.read()
        raw_lines=raw.split('\n')
        raw_data = [x.split(',') for x in raw_lines[0:-1]]
        segs = array([[float(x) for x in y] for y in raw_data])

        #compute cooling steps
        [rw, cl] = segs.shape;
        segtimes = divide(segs[:,0],dt)
        segtimes = [round(x) for x in segtimes]
        SegDTdt=[]
        for p in range(0,rw):
            thisseg=ones(segtimes[p])*segs[p][1]
            SegDTdt=concatenate((SegDTdt,thisseg))
        tend = sum(segtimes);
        ttot = tend*dt;

    # unit definitions and converions
    deltat = dt*3.1536e+13
    tend = math.ceil(ttot/dt)
    Tstart = Tstart + 273
    Tend = Tend + 273
    T0 = Tstart
    T = Tstart

    # initialize storage matrices
    mode = zeros([nmin])
    shape = zeros([nmin])
    L = zeros([nmin])
    w = zeros([nmin])
    r = zeros([nmin])
    SA = zeros([nmin])
    dx = zeros([nmin])
    gb = zeros([nmin])
    d180 = zeros([nmin])
    Afac = zeros([nmin])
    Bfac = zeros([nmin])
    Cfac = zeros([nmin])
    D0 = zeros([nmin])
    Q = zeros([nmin])
    D = zeros([nmin])
    fracfax = zeros([nmin])
    oxcon = zeros([nmin])
    R = 8.3144621  # J/K*ml

    ## get all rock properties
```

```
    # mineral 1 - monitor, quartz
    mode[0] = 0.20
    shape[0] = 2
    r[0] = 20
    L[0] = 2 * r[0]
    w[0] = 20
    dx[0] = r[0] / de
    gb[0] = math.ceil(L[0] / dx[0])
    Afac[0] = 0
    Bfac[0] = 0
    Cfac[0] = 0
    d180[0] = 99
    D0[0] = 3.4e-9
    Q[0] = 98000
    oxcon[0] = 0.0882

    # mineral 2 - alkali feldspar
    mode[1] = 0.76
    shape[1] = 1
    r[1] = 30
    L[1] = 2 * r[1]
    w[1] = 30
    dx[1] = r[1] / de
    gb[1] = math.ceil(L[1] / dx[1])
    Afac[1] = 0
    Bfac[1] = 0
    Cfac[1] = 1.0
    d180[1] = 99
    D0[1] = 7.6e-6
    Q[1] = 129500
    oxcon[1] = 0.0734

    # mineral 3 - titanite
    mode[2] = 0.01
    shape[2] = 1
    r[2] = 450
    L[2] = 2 * r[2]
    w[2] = 700
    dx[2] = r[2] / de
    gb[2] = math.ceil(L[2] / dx[2])
    Afac[2] = 0
    Bfac[2] = 0
    Cfac[2] = 3.66
    d180[2] = 99
    D0[2] = 2.05e-8
    Q[2] = 180000
    oxcon[2] = 0.0874

    # mineral 4 - augite
    mode[3] = 0.03
    shape[3] = 1
    r[3] = 30
    L[3] = 2 * r[3]
    w[3] = 30
    dx[3] = r[3] / de
    gb[3] = math.ceil(L[3] / dx[3])
    Afac[3] = 0
    Bfac[3] = 0
    Cfac[3] = 2.75
    d180[3] = 99
    D0[3] = 1.5e-6
```

```
    Q[3] = 226000
    oxcon[3] = 0.0892

    # convert input to micron
    L = L * 1e-4
    w = w * 1e-4
    r = r * 1e-4
    dx = dx * 1e-4
    maxdim = max(gb)
    # normalize mineral modes
    mode = mode.copy() / sum(mode)

    # caclculate mineral surface area
    for m in range(0, nmin):
        if shape[m] == 1:
            SA[m] = (4 * pi * pow(r[m], 2))
        else:
            if shape[m] == 2:
                SA[m] = 2 * L[m] * w[m]
            else:
                SA[m] = 2 * upi * r[m] * h[m]

    #initial conditions (starting concentration profiles)
        for m in range(0, nmin):
            fracfax[m] = Afac[m] + ((Bfac[m] * 1e3) / T0) + ((Cfac[m] * 1e6) / pow
(T0, 2))

    #recalculate estimated whole rock based on disequilibrium phase (only works
    #for one diseq phase in this formulation; preferrably a low-volume/accessory
phase)
    for m in range(0, nmin):
        if d180[m] < 99:
            WRd180 = WRd180 * (1 - mode[m]) + d180[m] * mode

    d180mon = WRd180 + dot(mode, fracfax)
    gbvalinit = zeros([nmin])
    for m in range(0, nmin):
        gbvalinit[m] = d180mon - fracfax[m]
    Told = zeros([nmin, int(max(gb))])
    for m in range(0, nmin):
        if d180[m] == 99:
            Told[m, 0:int(gb[m])] = gbvalinit[m]
        ####################elseif d18O == 100 %load precursor profile from text file
        ##################### Told(m) = reshape(load(uigetfile('*.txt')),1,1:gb(m))
        else:
            Told[m, 0:int(gb[m])] = d180[m]

    #### Solve fully implicit
    #####define data storage matrices
    time = zeros([tend])
    Temphx = zeros([tend])
    Tnew = zeros([nmin, int(max(gb))])
    pregbval = zeros([nmin])
    result = zeros([nmin, tend, int(maxdim)])  # array for storing results for all
    # minerals, indices (m,t,i)
    DTdt = zeros([tend])

    loading.config(maximum=int(tend/10))
    for t in range(0, int(tend)):

        if(t%10==0):
          loading.step()
```

```
          root.update()
        if cooling.get() == "Linear":
            DTdt = (Tstart - Tend) / ttot  # linear in t
            T = T0 - (DTdt * (t + 1) * dt)
        elif cooling.get() == "Inverse":
            k = ttot / ((1 / Tend) - (1 / Tstart))
            T = 1 / ((((t + 1) * dt) / k) + (1 / Tstart))
        else:
            DTdt = SegDTdt[t];
            T = T - (DTdt*dt);

        D = D0 * exp(-Q / (R * T))
        fracfax = Afac + Bfac * (1e3 / T) + Cfac * (1e6 / pow(T, 2))
        coeff = D / dx
        ###checked to here good
################################################################################
        for m in range(0, nmin):
            if shape[m] == 1:  # spherical/isotropic diffusion geometry
                gb[m] = math.ceil(r[m] / dx[m]) + 1
                a = ones([int(gb[m])])
                a[int(gb[m]) - 1] = 2
                b = (-2 - ((dx[m] * dx[m])) / (D[m] * deltat)) * ones([int(gb[m])])
                c = ones([int(gb[m])])
                c[0] = 2
                d = -((dx[m] * dx[m]) / (D[m] * deltat)) * Told[m, 0:int(gb[m])]
                for i in range(1, int(gb[m]) - 1):
                    a[i] = (i - 1) / i
                    c[i] = (i + 1) / i
            else:  # slab/infinite plane diffusion geometry
                a = ones([int(gb[m])])
                a[int(gb[m]) - 1] = 2
                b = (-2 - ((dx[m] * dx[m])) / (D[m] * deltat)) * ones([int(gb[m])])
                c = ones([int(gb[m])])
                c[0] = 2
                d = -((dx[m] * dx[m]) / (D[m] * deltat)) * Told[m, 0:int(gb[m])]
            TD = diag(b) + diag(a[1:], -1) + diag(c[0:-1], 1)
            Tnew[m, 0:int(gb[m])] = linalg.solve(TD, d)
            pregbval[m] = Tnew[m, int(gb[m]) - 1]

        gbval = fluxbal(nmin, pregbval, coeff, fracfax, mode, SA, oxcon)
        for m in range(0, nmin):
            Tnew[m, int(gb[m]) - 1] = gbval[m]
            if shape[m] == 2:
                Tnew[m, 1] = gbval[m]
            Told[m, :] = Tnew[m, :]
            time[t] = t * dt
            Temphx[t] = T
            result[m, t, 0:int(gb[m])] = Told[m, 0:int(gb[m])]

    #create global variables to store results in
    global yresult, timeresult, xresult
    yresult=result
    xsteps=yresult.shape[2]
    xresult=zeros((xsteps,nummin.get()))
    timeresult=linspace(0,1,yresult.shape[1])*duration.get()
    for i in range(0,nummin.get()):
        if shape[i]==1:
            xresult[:,i]=1e4*linspace(dx[i],2*r[i]-dx[i],xsteps)
            onesidey=result[i,:,0:int(xsteps/2)]
            yresult[i,:,0:2*int(xsteps/2)]=concatenate((onesidey
[:,::-1],onesidey),axis=1)
        else:
```

```
            xresult[:,i]=1e4*linspace(dx[i],L[i]-dx[i],xsteps)
    print(systime.time()-start_time)



    #remove loading bar
    loading.grid_remove()
    graphbutton.config(state='normal')
    csvsave.config(state='normal')
    root.update()


    if 0 == 1:
        # loc1 = os.path.dirname(sys.argv[0])
        # loc1 = loc1 + '/HA03_S1_T1.txt'
        loc1 = 'C:/Users/gabe_/Documents/PyCharm/HA03_S1_T1.txt'
        y = result[2, tend - 1, 0:int(gb[2])].reshape(1, int(gb[2]))
        a = octrave(loc1)
        plt.figure(1)
        plt.errorbar(a[0], a[1], a[2], fmt='o')
        u = 471  # distance in m
        u = u * 1e-4
        a = y[0:int(gb[2]) - 1]
        uvec = 1e4 * linspace(u, r[2] + u - dx[2], int(gb[2]))
        plt.plot(uvec, a[0, :], 'k-')
        v = 52
        v = v * 1e-4
        vvec = 1e4 * linspace(v, r[2] + v - dx[2], int(gb[2]))
        plt.plot(vvec, a[0, ::-1], 'k-')



#produce graphs on button call
def make_graphs():
    currentfig=0
    tend=len(timeresult)
    nmin=nummin.get()

    for i in range(1,9):
        checkif=graphcheck[i].get()
        if checkif == 1:
            currentfig=currentfig+1
            plt.figure(currentfig)
            plottimes = graph[i].get()
            plottimes = [float(x) for x in plottimes.split(',')]
            legendtimes = [str(x)+'  million years' for x in plottimes]
            n=len(plottimes)
            for j in range(0,n):
                ygr=yresult[i-1,max(0,min(int((plottimes[j]/duration.get
())*tend),tend-1)),:]
                xgr=xresult[:,i-1]
                plt.plot(xgr,ygr)
            plt.legend(legendtimes)
            plt.xlabel('x (cm)')
            plt.ylabel('Delta 18-O')
            plt.title('Plot of oxygen isotope ratios for '+rocks[i][1].get())

    if graphcheck[9].get() == 1:
        currentfig=currentfig+1
        plt.figure(currentfig)
        plottimes = graph[9].get()
        plottimes = [float(x) for x in plottimes.split(',')]
```

```
        legendtimes = [str(x)+' million years' for x in plottimes]
        plotindices = [max(0,min(tend-1,int((x/duration.get())*tend))) for x in
plottimes]
        for t in plotindices:
            yworking = yresult[:, int(t), :]
            for m in range(0, nmin):
                plt.subplot(nmin, 1, m + 1)
                xgr=xresult[:,m]
                ygr=yworking[m,:]
                plt.plot(xgr,ygr)
        plt.legend(legendtimes)
    plt.show()




### Create Main Window
root = Tk()
root.title("Diffusion Solver - Fast Grain Boundary Conditions")

## Create subsections of Main Window
# File handler window (import and export)
FileHand = LabelFrame(root, text="Import & Export Model Parameters", bg=background1)
FileHand.config(font=titlefonts)

# Model characteristics window (model characteristics ie runtime cooling rate)
ModelChar = LabelFrame(root, text="Global Model Characteristics", bg=background1)
ModelChar.config(font=titlefonts)

# Rock characteristics window (properties of each mineral ie diffusion and
fractionation)
RockChar = LabelFrame(root, text="Properties of Minerals", bg=background1)
RockChar.config(font=titlefonts)

# Graphing options window
GraphChar = LabelFrame(root, text="Graphing Options", bg=background1)
GraphChar.config(font=titlefonts)

# Export data window
CSVexp = LabelFrame(root, text='Export Simulation Data', bg=background1)
CSVexp.config(font=titlefonts)

#loading bar creation
#mystyle = ttk.Style()
#mystyle.theme_use('clam')
#mystyle.configure("red.Horizontal.TProgressbar", foreground='blue', background='red')
loading = ttk.Progressbar(root, mode='determinate', length='2i', maximum=8000)

# now we create the elements in each of these four subsections of the main window

## File handler elements
importl = Label(FileHand, text="Import")
importl.config(font=fonts2, bg=background1)
importb = StringVar(root)
importb.set("")
Importb = Entry(FileHand, textvariable=importb, width=40, font=fonts2)
Importb.config(font=fonts1)
ImpBut = Button(FileHand, text="Browse", command=importf)
ImpBut.config(font=brs)
exportl = Label(FileHand, text="Export", bg=background1)
```

```
exportl.config(font=fonts2)
exportb = StringVar(root)
exportb.set("")
Exportb = Entry(FileHand, textvariable=exportb, width=40, font=fonts2)
Exportb.config(font=fonts1)
ExpBut = Button(FileHand, text="Browse", command=exportf)
ExpBut.config(font=brs)
comm = Frame(FileHand)
runb = Button(comm, text="Run", command=runmain)
runb.config(font=runcmnds)
openb = Button(comm, text="Open", command=readdata)
openb.config(font=runcmnds)
saveb = Button(comm, text="Save", command=writedata)
saveb.config(font=runcmnds)


## Model characteristics elements
coollabel = Label(ModelChar,text="Cooling Type", bg=background1)
coollabel.config(font=fonts2)
cooling = StringVar(root)
cooling.set("Linear")
Cooling = OptionMenu(ModelChar, cooling, "Linear", "Inverse", "Custom",
command=poss_cool)
Cooling.config(font=fonts1, bg="white", relief="sunken", activebackground=bkgr2, bd=1,
               highlightthickness=0)
Cooling["menu"].config(bg="white")

numlabel = Label(ModelChar, text="Minerals", bg=background1)
numlabel.config(font=fonts2)
nummin = IntVar(root)
nummin.set("2")
Nummin = OptionMenu(ModelChar, nummin, "2", "3", "4", "5", "6", "7", "8",
command=readjust)
Nummin.config(font=fonts1, bg="white", relief="sunken", activebackground=bkgr2, bd=1,
              highlightthickness=0)
Nummin["menu"].config(bg="white")

durlabel = Label(ModelChar, text="Model Duration", bg=background1)
durlabel.config(font=fonts2)
duration = DoubleVar(root)
duration.set("")
Duration = Entry(ModelChar, textvariable=duration, width=10)

timlabel = Label(ModelChar, text="Time Step", bg=background1)
timlabel.config(font=fonts2)
timestep = DoubleVar(root)
timestep.set("")
Timestep = Entry(ModelChar, textvariable=timestep, width=10, font=fonts1)

wrdlabel = Label(ModelChar, text="Whole Rock", bg=background1)
wrdlabel.config(font=fonts2)
wrd180t = DoubleVar(root)
wrd180t.set("")
Wrd180t = Entry(ModelChar, textvariable=wrd180t, width=10, font=fonts1)

startlabel = Label(ModelChar, text="Starting Temp", bg=background1)
startlabel.config(font=fonts2)
modelstart = DoubleVar(root)
modelstart.set("")
Modelstart = Entry(ModelChar, textvariable=modelstart, width=10, font=fonts1)

endlabel = Label(ModelChar, text="End Temp", bg=background1)
```

```
endlabel.config(font=fonts2)
modelend = DoubleVar(root)
modelend.set("")
Modelend = Entry(ModelChar, textvariable=modelend, width=10, font=fonts1)


## Rock characteristics elements
rocks = dict()
Rocks = dict()
rocksl = dict()
for i in range(1,9):
    rocks[i] = dict()
    Rocks[i] = dict()

    rocksl[i] = Label(RockChar, text='Sample '+str(i))
    rocksl[i].config(font=fonts2, bg=background1)
rocksl[1].config(text='Monitor')

for i in range(1,9):
    for j in range(1,12):
        if j in [1,3]:
            rocks[i][j] = StringVar()
        else:
            rocks[i][j] = DoubleVar(root)
            rocks[i][j].set("")
        if j not in [3,6,7,8,9,10]:
         Rocks[i][j] = Entry(RockChar, textvariable=rocks[i][j], width=5, font=fonts1)
        else:
            if j==3:
                rocks[i][j].set('Slab')
                Rocks[i][j] = OptionMenu(RockChar, rocks[i][j], "Slab",
"Cylindrical", "Spherical")
                Rocks[i][j].config(font=fonts1, bg="white", relief="sunken",
activebackground=bkgr2, bd=1,
                                   highlightthicknes=0)
                Rocks[i][j]["menu"].config(bg="white")

            else:
                Rocks[i][j] = Label(RockChar, textvariable=rocks[i][j], bg='#dedede',
relief="sunken", bd=1, font=fonts1)
                #Rocks[i][j] = Entry(RockChar, textvariable=rocks[i][j], width=5,
font=fonts1, disabledbackground='#dedede')
                #Rocks[i][j].config(state='disabled')
rocks[1][6].set('0.00')
rocks[1][7].set('0.00')
rocks[1][8].set('0.00')
#################
#################BIG NO NO
#################

rockpropnam=[' Name',' Mode','Shape','  R  ','  W  ','Afrac','Bfrac','Cfrac',' D0
','  Q  ','Oxcon']
rockproplab = dict()
for i in range(1,12):
    rockproplab[i] = Label(RockChar, text=rockpropnam[i-1])
    rockproplab[i].config(font=fonts2, bg=background1)

#make look up buttons
lookupb = dict()
lookupbb =  dict()
# temporary fix to allow passable arguements
#for i in range(1,9):
```

```
#   lookupb[i] = Button(RockChar, text='Frac'+str(i), command = popup_search)
#   lookupb[i].bind("<Button-1>",popup_search[i])
#   lookupbb[i] = Button(RockChar, text='Diff'+str(i))

#search_photo=PhotoImage(file='search3.png')

#for i in range(2,9):
#  lookupb[i]=Button(RockChar, image=search_photo, bg=background1, relief='flat')
#  lookupb[i].bind('<Button-1>', lambda event, i=i: frac_search(i))
#  lookupbb[i]=Button(RockChar, image=search_photo, bg=background1, relief='flat')
#  lookupbb[i].bind('<Button-1>', lambda event, i=i: diff_search(i))
#lookupbb[1]=Button(RockChar, image=search_photo, bg=background1, relief='flat')
#lookupbb[1].bind('<Button-1>', lambda event, i=i: diff_search(1))


#Fracheader.image=asss
lookupbb[1] = Button(RockChar, text='Diff1', command= lambda: diff_search(1))
lookupb[2] = Button(RockChar, text='Frac2', command= lambda: frac_search(2))
lookupbb[2] = Button(RockChar, text='Diff2', command= lambda: diff_search(2))
lookupb[3] = Button(RockChar, text='Frac3', command= lambda: frac_search(3))
lookupbb[3] = Button(RockChar, text='Diff3', command= lambda: diff_search(3))
lookupb[4] = Button(RockChar, text='Frac4', command= lambda: frac_search(4))
lookupbb[4] = Button(RockChar, text='Diff4', command= lambda: diff_search(4))
lookupb[5] = Button(RockChar, text='Frac5', command= lambda: frac_search(5))
lookupbb[5] = Button(RockChar, text='Diff5', command= lambda: diff_search(5))
lookupb[6] = Button(RockChar, text='Frac6', command= lambda: frac_search(6))
lookupbb[6] = Button(RockChar, text='Diff6', command= lambda: diff_search(6))
lookupb[7] = Button(RockChar, text='Frac7', command= lambda: frac_search(7))
lookupbb[7] = Button(RockChar, text='Diff7', command= lambda: diff_search(7))
lookupb[8] = Button(RockChar, text='Frac8', command= lambda: frac_search(8))
lookupbb[8] = Button(RockChar, text='Diff8', command= lambda: diff_search(8))



## Graphing options elements
GraphChar = LabelFrame(root, text="Graphing Options", bg=background1)
GraphChar.config(font=titlefonts)

graph=dict()
Graph=dict()
graphcheck=dict()
GraphCheck=dict()

graphbutton = Button(GraphChar, text='Produce Plots', command=make_graphs)
graphbutton.config(state='disabled')
for i in range(1,10):
    graph[i]=StringVar(GraphChar)
    graph[i].set("")
    Graph[i] = Entry(GraphChar, textvariable=graph[i], width=10, font=fonts1)
    graphcheck[i]=IntVar(GraphChar)
    graphcheck[i].set(0)
    GraphCheck[i]=Checkbutton(GraphChar, text="Plot Mineral "+str(i), font=fonts2,
bg=background1, variable=graphcheck[i])

GraphCheck[9] = Checkbutton(GraphChar, text="Plot All ", font=fonts2, bg=background1,
                            variable=graphcheck[9])


# Make export csv elements
csvlabel = Label(CSVexp, text="File:", bg=background1)
csvlabel.config(font=fonts2)
csvfile = StringVar(CSVexp)
```

```
csvfile.set("")
Csvfile = Entry(CSVexp, textvariable=csvfile, font=fonts2, width=30)
Csvfile.config(font=fonts1)

# make buttons for csv
CSVbutt=Frame(CSVexp)
csvbrowse = Button(CSVbutt, text="Browse", command=cbrowse)
csvbrowse.config(font=runcmnds)
csvsave = Button(CSVbutt, text="Export", command=csave)
csvsave.config(font=runcmnds, state="disabled")
csvform = Button(CSVbutt, text="Format", command=cformat)
csvform.config(font=runcmnds)



# Now we make the tooltips to explain all buttons and entry fields
createToolTip(numlabel,"Number of minerals in sample")
createToolTip(durlabel, "Duration of model in millions of years")
createToolTip(timlabel, "incremental time step in diffusion solver in milions of
years, ie 5e-6")
createToolTip(coollabel, "This creates temperature profile")
createToolTip(startlabel, "Beginning temperature in C")
createToolTip(endlabel, "Ending temperature in C")
createToolTip(wrdlabel, "Estimate of whole rock")

#for i in range(1,12):
#    createToolTip(rockproplab[i], "Explanation of "+rockpropnam[i-1]+" parameter")

for i in range(1,9):
    createToolTip(rocksl[i], "Properties of mineral sample "+str(i))

createToolTip(runb, "Run model with current parameters")
createToolTip(openb, "Import parameters from previously saved file in import")
createToolTip(saveb, "Save current parameters to file in export")

#Now we place everything on the main window

#place main frames
FileHand.grid(row=0, column=1, padx=5, pady=5, ipadx=5, ipady=5, stick=W+E)
ModelChar.grid(row=1, column=1, padx=5, pady=5, ipadx=5, ipady=5, stick=W+E)
RockChar.grid(row=2, column=1, padx=5, pady=5, ipadx=5, ipady=5, stick=W+E)
GraphChar.grid(row=0, column=2, rowspan=2, padx=5, pady=5, ipadx=5, ipady=5, stick=N
+S)
CSVexp.grid(row=2, column=2, padx=5, pady=5, ipadx=5, ipady=5, sticky=N+S+W+E)
#give weights for scaling
#root.grid_columnconfigure(0,weight=1)
#root.grid_columnconfigure(1,weight=1)
#root.grid_columnconfigure(2,weight=1)
#root.grid_rowconfigure(0,weight=1)
#root.grid_rowconfigure(1,weight=1)
#root.grid_rowconfigure(2,weight=1)



#place everything file frame
importl.grid(row=0, column=1)
Importb.grid(row=0, column=2)
ImpBut.grid(row=0,column=3)

exportl.grid(row=1, column=1)
```

```
Exportb.grid(row=1, column=2)
ExpBut.grid(row=1,column=3)

comm.grid(row=2, column=1, columnspan=1, pady=(10,0), padx=(10,0))
runb.grid(row=0, column=3)
openb.grid(row=0, column=1)
saveb.grid(row=0, column=2)

wt=10
t1=10
t2=20
t3=20

#place everything in model frame
numlabel.grid(row=0, column=1, stick=E, padx=(t1,0))
Nummin.grid(row=0, column=2, stick=W+E, padx=(wt,0))

durlabel.grid(row=0, column=3, stick=E, padx=(t2,0))
Duration.grid(row=0, column=4, stick=W+E, padx=(wt,0))

timlabel.grid(row=0, column=5, stick=E, padx=(t3,0))
Timestep.grid(row=0, column=6, stick=W+E, padx=(wt,0))

coollabel.grid(row=1, column=1, stick=E, padx=(t1,0))
Cooling.grid(row=1, column=2, stick=W+E, padx=(wt,0))

startlabel.grid(row=1, column=3, stick=E, padx=(t2,0))
Modelstart.grid(row=1, column=4, stick=W+E, padx=(wt,0))

endlabel.grid(row=1, column=5, stick=E, padx=(t3,0))
Modelend.grid(row=1, column=6, stick=W+E, padx=(wt,0))

wrdlabel.grid(row=2, column=1, stick=E, padx=(t1,0))
Wrd180t.grid(row=2, column=2, stick=W+E, padx=(wt,0))



#place everything rock frame

for i in range(1,9):
  rocksl[i].grid(row=i, column=1)
for i in range(1,12):
  rockproplab[i].grid(row=0, column=i+1+(i>8)+(i==11))

for i in range(1,9):
    for j in range(1,12):
        Rocks[i][j].grid(row=i, column=j+1+(j>8)+(j==11), stick=N+S+W+E)

for i in range(2,9):
    lookupb[i].grid(row=i, column=10, stick=N+S+E+W)
    lookupbb[i].grid(row=i, column=13, sticky=N+S+W+E)

lookupbb[1].grid(row=1, column=13, sticky=N+S+E+W)


#place everything in graphing options pane
graphpad1=5
graphpad2=40
graphpad3=33
graphpad4=0
```

```
for i in range(1,5):
    GraphCheck[2*i].grid(row=2*i-1, column=1, sticky=W)
    GraphCheck[2*i-1].grid(row=2*i-1, column=0, sticky=W, padx=(graphpad1, graphpad2))
    Graph[2*i].grid(row=2*i, column=1, sticky=W, padx=(graphpad3,graphpad4))
    Graph[2*i-1].grid(row=2*i, column=0, sticky=W, padx=(graphpad3,graphpad4))

GraphCheck[9].grid(row=9,column=0, sticky=W, padx=(graphpad1,0))
Graph[9].grid(row=10,column=0, sticky=W, padx=(graphpad3,graphpad4))
graphbutton.grid(row=10, column=1, sticky=W, padx=(graphpad3,0), pady=(0,5))


# place everything in csv options pane
csvlabel.grid(row=1, column=1, padx=(10,0), pady=(10,0))
Csvfile.grid(row=1, column=2, columnspan=3, stick=W+E, padx=(0,5), pady=(10,0))
CSVbutt.grid(row=2, column=1, columnspan=2, padx=(10,0), pady=(5,0))
csvbrowse.grid(row=1, column=1)
csvsave.grid(row=1, column=2, sticky=W)
csvform.grid(row=1, column=3, sticky=W)


readjustn()


# start main loop
root.config(bg=background1)
root.mainloop()
```