```
#Oliver Gorton
#May 2016
#UC Berkeley

'''
Requires ffmpeg be installed and accessible
'''

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
from Tkinter import *
import tkMessageBox as msg

hbar=1
m=1
dt=0.05
dx=0.5
K = 1

def gaussX(x, a, x0, k0):
      """Gaussian wave packet where a=width, x0=center, k0=momentum, hbar=1
      the energy of the particle is given by 0.5k0^2"""
      return (np.exp(-0.5 * ((x - x0) * 1. / a) ** 2 + 1j * x * k0))
#(a * np.sqrt(np.pi)) ** (-0.5)

class Psi(object):
      def __init__(self,a,x0,k0):
            self.t=0
            self.a=a
            self.x0=x0
            self.k0=k0
            self.E=0.5*self.k0*self.k0
            self.xdomain=np.arange(-100,100,dx)
            self.dx=dx
            self.dk=2*np.pi/(200)

            self.psix1=gaussX(self.xdomain,a,x0,k0)
            self.psix2=gaussX(self.xdomain,a,x0+0.5*dt*k0,k0)

            self.R_prev=1*np.real(self.psix1)
            self.I_prev=[]

            self.R=[]
            self.I=1*np.imag(self.psix2)

            self.R_next=[]
            self.I_next=[]

            if self.R_next == []:
                  self.P=(np.absolute(self.psix1))**2
            else:
                  self.P=(self.I*self.I+self.R_next*self.R_prev)

            self.Psum0 = np.sum((np.absolute(self.psix1))**2)
            self.Pnorm = np.sum(self.P) / self.Psum0
```

```
    def T(self,dt):
        '''
        Keeps track of time and updates psi^2 from Re[psi] and Im[psi]
        which are evolved seperately but not independently.
        '''
        self.t+=dt
        if self.R_next == []:
                    self.P=(np.absolute(self.psix1))**2
        else:
            self.P=(self.I*self.I+self.R_next*self.R_prev)

    def norm_update(self):
        self.Pnorm = np.sum(self.P) / self.Psum0

    def H_R(self, potential):
        '''
        Taylor approximation of the Hamiltonian for Re[psi]
        '''
        H = np.zeros(len(self.xdomain))
        for i in range(1,len(self.R)-2):
                    H[i]=-0.5*((self.R[i+1]-2*self.R[i]+self.R[i-1]))/
(self.dx*self.dx) + (potential.potential[i]*self.R[i])
        return H

    def H_I(self, potential):
        '''
        Taylor approximation of the Hamiltonian for Im[psi]
        '''
        H=np.zeros(len(self.xdomain))
        for i in range(1,len(self.R)-2):
            H[i]=-0.5*((self.I[i+1]-2*self.I[i]+self.I[i-1]))/
(self.dx*self.dx) + (potential.potential[i]*self.I[i])
        return H

    def Rstep(self,dt,potential):
        '''
        Computes one step of the TDSE for Re[psi]
        '''
        self.R_next=self.R_prev+dt*self.H_I(potential)

    def Istep(self,dt,potential):
        '''
        Computes one step of the TDSE for Im[psi]
        '''
        self.I_next=self.I_prev-dt*self.H_R(potential)

        '''
        Since we don't want to store psi for all time and only
        care about the closest three time increments (for the
        Hamiltonian approximation) we only keep three time steps
        and discard the rest.

        We discretize Psi(t) and only keep Psi_t-1, Psi_t, and Psi_t+1
        '''
    def memoryShiftA(self):
        '''
        Shifts time indices of psi(x,t) as we progress through time.
        '''
        self.I_prev=1*self.I
```

```
            self.R=1*self.R_next

      def memoryShiftB(self):
            '''
            Shifts time indices
            '''
            self.I=1*self.I_next
            self.R_prev=1*self.R

      #Momentum space
      def calc_Kspace(self):
            '''
            To represent Psi in momentum space, we have to Fourier Transform
            Psi(x,t). We do this by understanding that the DFT of Psi can be
            deduced from FFT(PSI). This function does that.
            '''
            g = np.fft.fft(self.dx*np.exp(-
1j*self.k0*self.xdomain)*(self.R+self.I*1j)/(np.sqrt(2*np.pi)))
            w = np.fft.fftfreq((self.xdomain).size)*2*np.pi/self.dx
            g *= np.exp(-1j*self.dk*-100)

            self.Kdomain = w
            self.K = g

########################################################################
class Potential(object):
      '''
      Storing the potential as a class means that it could easily be made
      to be time dependent of have other fun properties.
      '''
      def __init__(self,left,right,height,M):
            '''
            Creates a potential that is zero everywhere
            except in a finite region where it has a value
            "height"
            '''
            self.domain=np.arange(-100,100,dx)
            self.left=left
            self.right=right
            self.height=height
            self.M = M

            if self.M == 0:
                  pot=np.zeros(len(self.domain))
                  pot[self.left<self.domain]=self.height
                  pot[self.domain>self.right]=0

            elif self.M == 1:
                  pot=np.zeros(len(self.domain))
                  pot[self.left<self.domain]=self.height
                  pot[self.domain>self.right]=0
                  pot *= np.sin(0.5*self.domain)

            elif self.M == 2:
                  pot=np.zeros(len(self.domain))
                  pot[self.left<self.domain]=self.height
                  pot[self.domain>self.right]=0
                  pot *= abs(0.5*np.sin(self.domain))
```

```
            elif self.M == 3:
                  pot=np.zeros(len(self.domain))
                  pot[self.left<self.domain]=self.height
                  pot[self.domain>self.right]=0
                  pot2 = np.zeros(len(self.domain))
                  pot2[(self.left+10*(self.right-
self.left))<self.domain]=self.height
                  pot2[self.domain>(self.right+10*(self.right-self.left))]=0
                  pot += pot2

            elif self.M == 4:
                  pot = np.ones(len(self.domain))
                  pot *= self.domain
                  pot[self.left>self.domain]=0
                  pot[self.domain>self.right]=0
                  pot *= self.height

            #pot[self.domain>95.0]=10
            #pot[self.domain<-95.0]=10
            self.potential=pot

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

def TimeStep(psi,potential,dt):
      '''
      Running this function computes one step,
      progressing the system by dt.
      '''
      psi.Rstep(dt,potential)
      psi.memoryShiftA()
      psi.Istep(dt,potential)
      psi.memoryShiftB()
      psi.calc_Kspace()
      psi.norm_update
      psi.T(dt)

######################################################################
#Plotting

fig = plt.figure()

ax=fig.add_subplot(211,xlim=(-100,100),ylim=(-.5,1.1))
ax.set_xlabel('$x$')
ax.set_ylabel(r'$|\psi(x)|^2$')
ax.set_title(r'$i\hbar\frac{\partial}{\partial t}\Psi(x,t)=-\frac{\hbar^2}
{2m}\nabla^2\Psi(x,t)+V(x)\Psi(x,t)$' '\n')

pot, =ax.plot([],[],c='k',label=(r'$V(x)$'))
psix, =ax.plot([],[],c='r',label=r'$|\Psi(x)|^2$')
vel =ax.axvline(0,c='k',ls=':', label=r'$x_0+k_0 t$')

ax.legend()

time=ax.text(-80,.7, '')
step=ax.text(-80,.8,'')
norm=ax.text(-80,-.2,'')

ax2=fig.add_subplot(212,xlim=(-2,2),ylim=(-.1,1.1))
ax2.set_xlabel(r'$k$')
```

```
ax2.set_ylabel(r'$|\Psi(k)|^2$')

pmk01 =ax2.axvline(0,c='k',ls=':',label=r'$\pm k_0$')
pmk02 =ax2.axvline( 0,c='k',ls=':')

psik, =ax2.plot([],[],c='b',label=r'$|\tilde{\Psi}(k)|^2$')

ax2.legend()

plt.tight_layout()

######################################################################
#Animation

def init():
    pot.set_data([],[])
    psix.set_data([],[])
    psik.set_data([],[])

    vel.set_data([],[])
    pmk01.set_data([],[])
    pmk02.set_data([],[])

    time.set_text('')
    step.set_text('')
    norm.set_text('')
    return psix,

go = True

speed = 1

def animate(j):
      while go == True:
            for i in range(0,speed):
                  TimeStep(psi1,pot1,dt)

            x = psi1.xdomain
            y = psi1.P

            u = (psi1.Kdomain+psi1.k0)
            v = np.absolute(psi1.K) / 10

            V = 1 * pot1.potential

            time.set_text(r'$t = $'+str(psi1.t))
            step.set_text(r'$dt = $'+str(dt))
            norm.set_text(r'$\int_{-100}^{100}|\Psi(x)|^2 dx$ = '+str(psi1.Pnorm))

            pot.set_data(x,V)
            psix.set_data(x,y)
            psik.set_data(u,v)

            vel.set_data([psi1.x0+psi1.t*psi1.k0],[0,1])
            pmk01.set_data([-psi1.k0],[0,1])
            pmk02.set_data([psi1.k0],[0,1])

            return psix, time, psik, pot, vel
#Tkinter
```

```
intro_page = Tk()
intro_page.title('Welcome!')
intro_page.minsize(width=500,height=250)

main = Tk()
main.title('Oliver\'s Particle in a Box')
main.minsize(width=1225,height=780)

about1 = 'In this simulation a Gaussian wave-packet is incident upon a potential
barrier. In the animation the wave packet is scaled such that the height of the
packet is proportional to its energy, which is given by E=0.5*k0^2 where k0 is the
momentum. (hbar and m are both set to 1).'

about2 = 'If the energy of the particle is less than the energy of the potential
barrier, then classically we would expect the entire particle to reflect from the
barrier. However, quantum mechanics tells us that some of the particle/wave will
tunnel through the barrier. This simulation is good for demonstrating this fact. '

about3 = 'This simulation works by numerically solving the time-dependent
schrodinger equation using an algorithm outlined in a paper called "A fast explicit
algorithm for the time-dependent Schrodiger equation" by P.B. Visscher. Also
included is a momentum-space representation of the wave-equation which is
calculated using the method described by Jake Vanderplas in his article "Quantum
Python: Animating the Schrodinger Equation". (In this, Vanderplas produces similar
overall results using a different integration algorithm.)'

About = Text(intro_page)
About.insert(INSERT, 'About this program:')
About.insert(END,'\n \n')
About.insert(END,about1)
About.insert(END,'\n \n \n')
About.insert(END,about2)
About.insert(END,'\n \n \n')
About.insert(END,about3)
About.insert(END,'\n \n \n')
About.insert(END,'Program written by Oliver Gorton, April 2016')
About.pack()

dismiss = Button(intro_page, text = 'Dismiss', command=intro_page.destroy)
dismiss.pack()

canvas = FigureCanvasTkAgg(fig, master=main)
canvas.get_tk_widget().config(width=750,height=750)
canvas.get_tk_widget().grid(row=0, column=0, rowspan=31)

def quit():
      sys.exit()

quit_button = Button(main, text = 'Quit', command = quit,width=20)
quit_button.grid(row=26,column=2)

label0 = Label(main, text = 'Custom settings')
label0.grid(row=2, column=2)

label1 = Label(main, text = 'Left edge of potential (-100 to 100)')
label1.grid(row=3, column=1, sticky=E)
e1 = Entry(main, bd = 5)
e1.grid(row=3, column=2)
left = e1
```

```
label2 = Label(main, text = 'Right edge of potential (-100 to 100)')
label2.grid(row=4,column=1,sticky=E)
e2 = Entry(main,  bd = 5)
e2.grid(row=4, column=2)
right = e2

label3 = Label(main, text = 'Energy (depth) of potential')
label3.grid(row=5, column=1,sticky=E)
e3 = Entry(main, bd = 5)
e3.grid(row=5, column=2)
depth = e3


label4 = Label(main, text = 'Momentum of particle')
label4.grid(row=6, column=1,sticky=E)
e4 = Entry(main, bd = 5)
e4.grid(row=6, column=2)
energy = e4


label5 = Label(main, text = 'Initial position of particle')
label5.grid(row=7, column=1, sticky = E)
e5 = Entry(main, bd = 5)
e5.grid(row=7, column=2)
pos = e5

mod = IntVar()

def select0():
      global mod, M, modulation
      modulation = 'Flat'
      M = 0

def select1():
      global mod, M, modulation
      modulation = 'Sinusoidal'
      M = 1

def select2():
      global mod, M, modulation
      modulation = 'Abs(Sinusoidal)'
      M = 2

def select3():
      global mod, M, modulation
      modulation = 'Double Square'
      M = 3

def select4():
      global mod, M, modulation
      modulation = 'Ramp'
      M = 4

label7 = Label(main, text = 'Modulation of the potential barrier:')
label7.grid(row=8, column=1)

option1 = Radiobutton(main, text = 'Rectangular', variable=mod, value=0,
```

```
command=select0)
option1.grid(row=9,column=1,sticky=W)

option2 = Radiobutton(main, text = 'Sin(x)', variable=mod, value=1,command=select1)
option2.grid(row=9,column=2,sticky=W)

option3 = Radiobutton(main, text = '|Sin(x)|', variable=mod, value=2,
command=select2)
option3.grid(row=10, column=2,sticky=W)

option4 = Radiobutton(main, text = 'Double Rectangular', variable=mod, value=3,
command=select3)
option4.grid(row=10,column=1,sticky=W)

option5 = Radiobutton(main, text = 'Ramp', variable=mod, value=4, command=select4)
option5.grid(row=11, column=1, sticky=W)


psi1 = []
pot1 = []

L, R, D, E, X, modulation = 0, 0, 0, 0, 0, ''

history = 0

def enter():
      global go, history
      global psi1, pot1
      global L, R, D, E, X, M, modulation

      go = True
      if history == 0:
            run_program.configure(state=NORMAL)
      history = 1


      L = float(left.get())
      R = float(right.get())
      D = float(depth.get())
      E = float(energy.get())
      X = float(pos.get())
      M = float(M)

      psi1 = Psi(5,X,E)

      pot1 = Potential(L,R,D,M)

      print L, R, D, E, X, M

      return psi1, pot1

def run():
      global go, history
      if history == 1:
            if msg.askokcancel('Keep these parameters?',str('Left edge: '+str(L)
+'\nRight edge: '+str(R)+'\nDepth: '+str(D)+'\nParicle Energy: '+str(E)
+'\nModulation: '+str(modulation))):
                  anim = animation.FuncAnimation(fig,animate,
init_func=init,blit=False,frames=5000,interval=1)
```

```
                  plt.draw()
                  run_program.configure(state = DISABLED)
                  return anim


def pause():
      global go
      go = False

def speed_value():
      global speed
      speed = int(speed_set.get())

speed_set_label = Label(main, text = 'Animation speed multiplier')
speed_set_label.grid(row=15,column=1,sticky='E')

speed_set = Spinbox(main, from_=1, to=100, width=5, command=speed_value)
speed_set.grid(row=15, column=2,sticky='W')

initialize = Button(main, text = 'Set initial conditions', command =
enter,width=20)
initialize.grid(row=13,column=2)

run_program = Button(main, text = 'Start Simulation', state = DISABLED, command =
run,width=20)
run_program.grid(row=14,column=2,sticky='E')

stop_program = Button(main, text = 'Stop Simulation', command = pause,width=20)
stop_program.grid(row=14, column=1)

labela = Label(main, text = 'Preset settings')
labela.grid(row=0, column=2)

current = StringVar()
current.set('Select one')

ea = Menubutton(main, text = current.get(), relief = RAISED, width = 20)
ea.grid(row=1,column=2)
ea.menu = Menu(ea, tearoff = 0)
ea["menu"] = ea.menu

def preset_none():
      e1.delete(0,END)
      e2.delete(0,END)
      e3.delete(0,END)
      e4.delete(0,END)
      e5.delete(0,END)
def preset0(): #ht
      e1.delete(0,END)
      e1.insert(0,'0') #left
      e2.delete(0,END)
      e2.insert(0,'1.5') #right
      e3.delete(0,END)
      e3.insert(0,'0.25') #depth
      e4.delete(0,END)
      e4.insert(0,K) #momentum
      e5.delete(0,END)
      e5.insert(0,'-40') #pos
def preset1(): #lt
```

```
      e1.delete(0,END)
      e1.insert(0,'0') #left
      e2.delete(0,END)
      e2.insert(0,'1') #right
      e3.delete(0,END)
      e3.insert(0,'2') #depth
      e4.delete(0,END)
      e4.insert(0,K) #momentum
      e5.delete(0,END)
      e5.insert(0,'-40') #pos
def preset2(): #et
      e1.delete(0,END)
      e1.insert(0,'0') #left
      e2.delete(0,END)
      e2.insert(0,'1') #right
      e3.delete(0,END)
      e3.insert(0,'0.9') #depth
      e4.delete(0,END)
      e4.insert(0,K) #momentum
      e5.delete(0,END)
      e5.insert(0,'-40') #pos
def preset3(): #nt
      e1.delete(0,END)
      e1.insert(0,'0') #left
      e2.delete(0,END)
      e2.insert(0,'5') #right
      e3.delete(0,END)
      e3.insert(0,'5') #depth
      e4.delete(0,END)
      e4.insert(0,K) #momentum
      e5.delete(0,END)
      e5.insert(0,'-40') #pos'

ea.menu.add_radiobutton(label = 'None', command = preset_none)
ea.menu.add_radiobutton(label = 'High Transmission', command = preset0)
ea.menu.add_radiobutton(label = 'Low Transmission', command = preset1)
ea.menu.add_radiobutton(label = 'Equal Transmission', command = preset2)
ea.menu.add_radiobutton(label = 'No Transmission', command = preset3)

label20 = Label(main, text = 'Video Generator')
label20.grid(row=16,column=2)


label21 = Label(main, text = 'Frames')
label21.grid(row=17,column=1)
e21 = Entry(main, bd = 5)
e21.grid(row=17, column=2)
frames_setting = e21

label22 = Label(main, text = 'Interval')
label22.grid(row=18,column=1)
e22 = Entry(main, bd = 5)
e22.grid(row=18, column=2)
interval_setting = e22

label23 = Label(main, text = 'FPS')
label23.grid(row=19, column=1)
e23 = Entry(main, bd = 5)
e23.grid(row=19, column=2)
```

```
fps_setting = e23

label24 = Label(main, text = 'File name (*.mp4)')
label24.grid(row=20, column=1)
e24 = Entry(main, bd = 5)
e24.grid(row=20, column=2)
file_name = e24


def video_maker():
      global frames_setting, interval_setting, fps_setting, file_name
      frames_setting = int(frames_setting.get())
      interval_setting = int(interval_setting.get())
      fps_setting = int(fps_setting.get())
      file_name = str(file_name.get())
      print frames_setting, interval_setting, fps_setting, file_name
      if msg.askokcancel('Save this animation? (This might take a 
while!)',str('Left edge: '+str(L)+'\nRight edge: '+str(R)+'\nDepth: '+str(D)
+'\nParicle Energy: '+str(E)+'\nModulation: '+str(modulation))):
                        save1=animation.FuncAnimation(fig,animate, 
init_func=init,blit=False,frames=int(frames_setting),interval=int(interval_setting)
).save(str(file_name),writer='ffmpeg',fps=str(fps_setting))
                        print 'done'
                        msg.showinfo('Finished','File was saved successfully.')
                        return save1

save_anim = Button(main, text='Save this animation',command=video_maker,width=20)
save_anim.grid(row=21,column=2)

main.mainloop()
```