ΥΠΟΛΟΓΙΣΤΙΚΉ ΚΒΑΝΤΙΚΉ ΦΥΣΙΚΉ ΚΑΙ ΕΦΑΡΜΟΓΈΣ

Bantis Peter

MSc in Computational Physics

Aristotle University of Thessaloniki

19/6/2022

Aristotle University of Thessaloniki

LISTINGS

- 6.6 - Spontaneous Decay Simulation
- 6.19 - Double well transitions
- 6.24 – Feynman Path Integral Quantum Mechanics

SPONTANEOUS DECAY SIMULATION

- An excited particle decays into other particles without any external stimulation
- Probability of decay

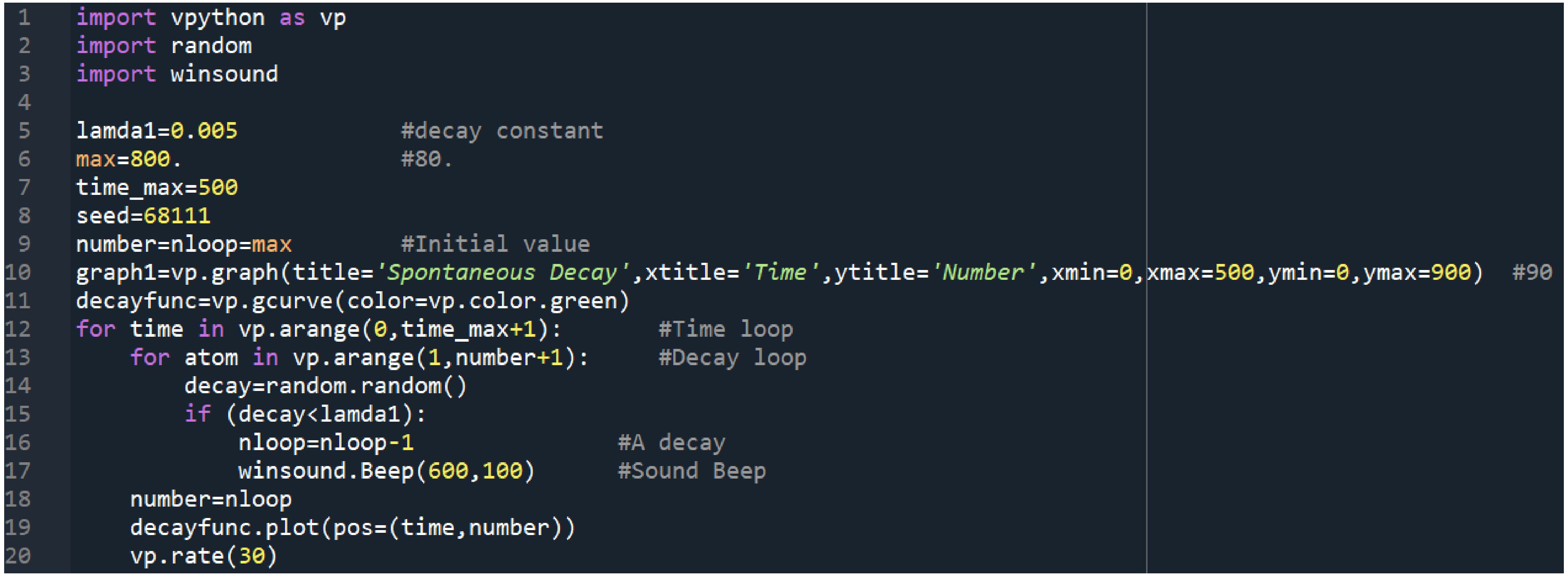
$$P = \frac{\Delta N(t)/N(t)}{\Delta \mathrm{t}} = -\lambda$$, where λ is the decay constant.

- Finite-difference equation:

$$\frac{dN(t)}{dt} \approx -\lambda N(t) \;\; \rightarrow \;\; N(t) \approx N(0)e^{-\lambda t}$$

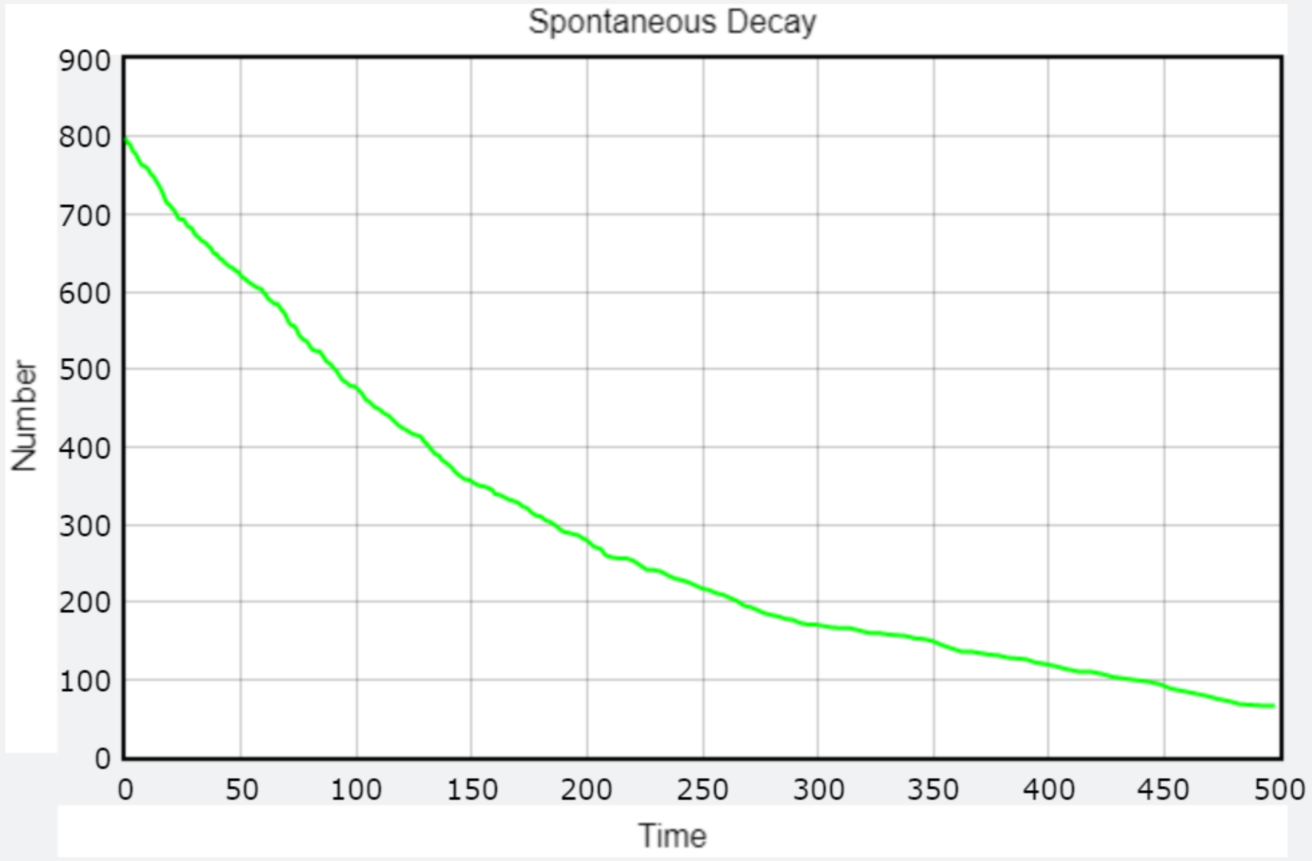
CODE

- Line 1-3: calling the libraries that will be used
- Line 5-9: setting up parameters
- Line 10-11: setting up the graph canvas and the curve-function that will be drawn
- Line 12: A time loop for each time unit that we will measure the decay
- Line 13: A loop that will check the decay for every nuclei (N) contained in the nucleus
- Line 14-17: Checking randomly if the decay occurs, if it happens a beep sound is made and the N decreases
- Line 19-20: a point is added to the plot and the loop goes on from the beginning



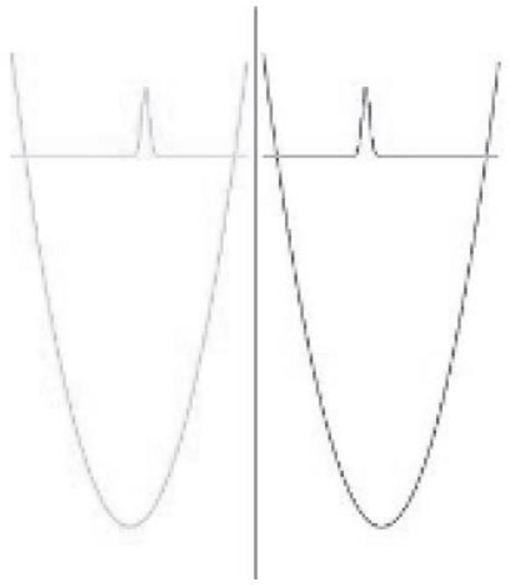
```
1   import vpython as vp
2   import random
3   import winsound
4
5   lamda1=0.005            #decay constant
6   max=800.                #80.
7   time_max=500
8   seed=68111
9   number=nloop=max        #Initial value
10  graph1=vp.graph(title='Spontaneous Decay',xtitle='Time',ytitle='Number',xmin=0,xmax=500,ymin=0,ymax=900)  #90
11  decayfunc=vp.gcurve(color=vp.color.green)
12  for time in vp.arange(0,time_max+1):       #Time loop
13      for atom in vp.arange(1,number+1):     #Decay loop
14          decay=random.random()
15          if (decay<lamda1):
16              nloop=nloop-1              #A decay
17              winsound.Beep(600,100)     #Sound Beep
18      number=nloop
19      decayfunc.plot(pos=(time,number))
20      vp.rate(30)
```

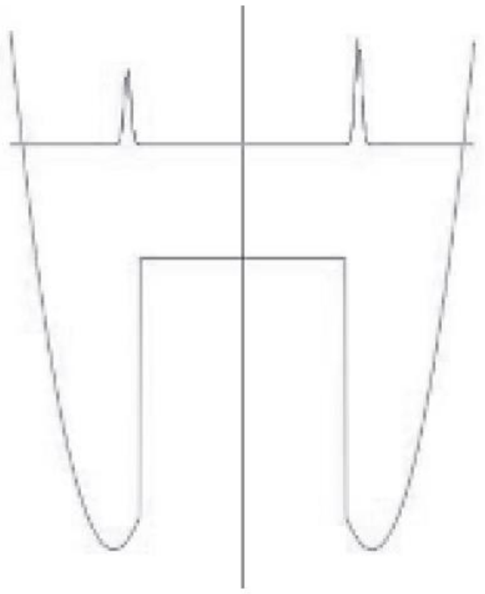
LISTING 6.6


Spontaneous Decay
Number
Time
0
100
200
300
400
500
600
700
800
900
0
50
100
150
200
250
300
350
400
450
500

DOUBLE WELL TRANSITIONS

- Given 2 identical potential wells with a barrier of infinite height and finite width between them, a particle placed on one well will remain there for ever (left side).
- If there is a perturbation ΔE that lowers the barrier between them, then one can expect that the bound particle will make a transition from left to right side with a certain probability (right side).
- Changing the width and the height of the wells affect how the particles jump for one well to another.

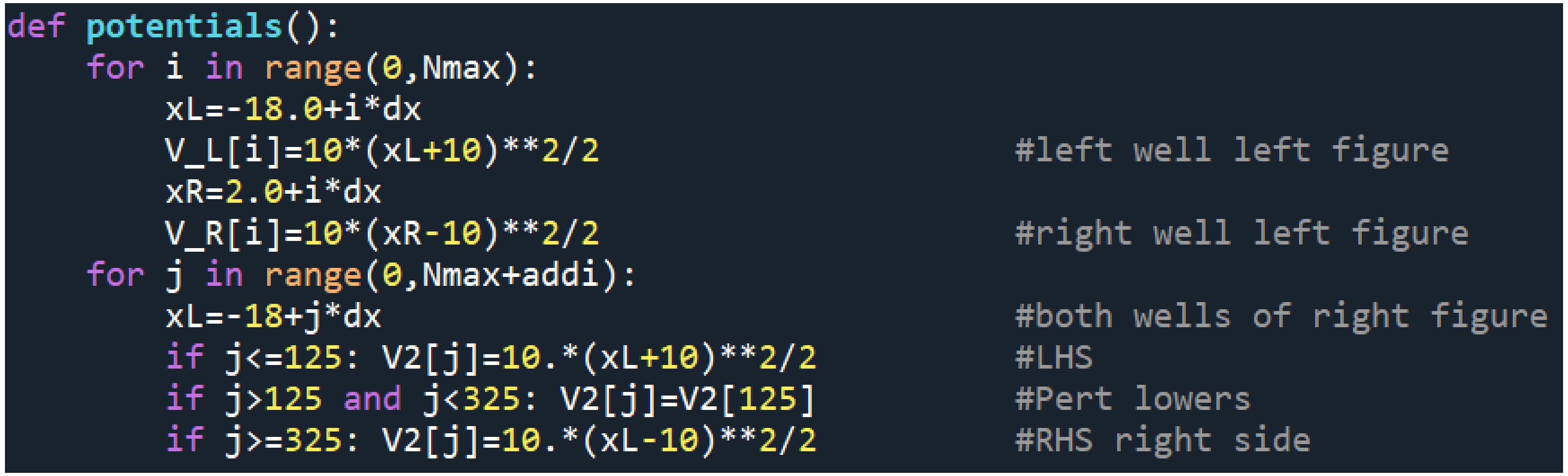




CODE

For **def potential():**

- V_L, V_R are potentials of the separate wells (without perturbation)
- V2 is the potential for the perturbation connected wells

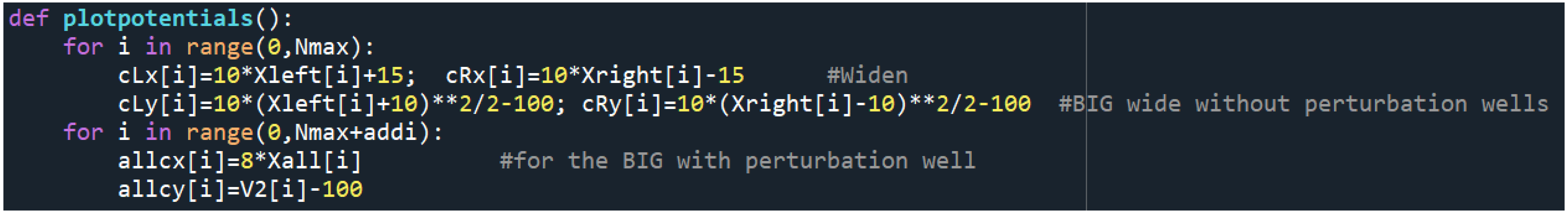


```
def potentials():
    for i in range(0,Nmax):
        xL=-18.0+i*dx
        V_L[i]=10*(xL+10)**2/2                         #left well left figure
        xR=2.0+i*dx
        V_R[i]=10*(xR-10)**2/2                         #right well left figure
    for j in range(0,Nmax+addi):
        xL=-18+j*dx                                    #both wells of right figure
        if j<=125: V2[j]=10.*(xL+10)**2/2              #LHS
        if j>125 and j<325: V2[j]=V2[125]              #Pert lowers
        if j>=325: V2[j]=10.*(xL-10)**2/2              #RHS right side
```

CODE

For **def plotpotentials():**

- cLx, cRx, cLy, cRy are the BIG (widen) <u>without</u> perturbation potential wells - (cLz, cRz → 0)
- allcx, allcy is the BIG <u>with</u> perturbation potential well
- Xleft, Xright and Xall are the x axis for the small well virtualization

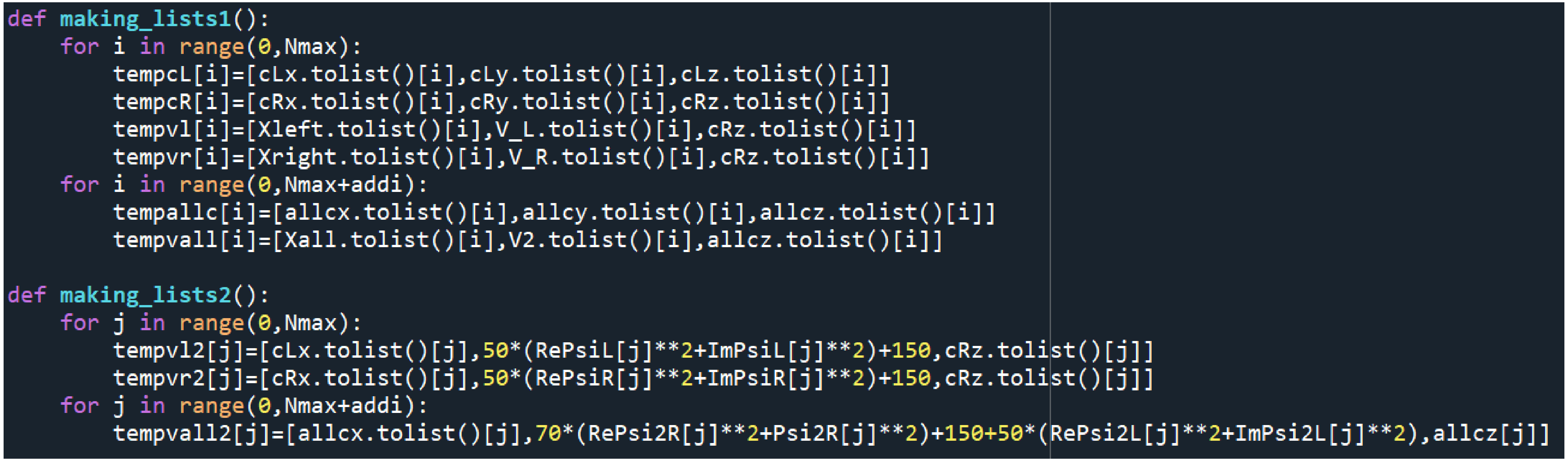


```
def plotpotentials():
    for i in range(0,Nmax):
        cLx[i]=10*Xleft[i]+15;  cRx[i]=10*Xright[i]-15      #Widen
        cLy[i]=10*(Xleft[i]+10)**2/2-100; cRy[i]=10*(Xright[i]-10)**2/2-100  #BIG wide without perturbation wells
    for i in range(0,Nmax+addi):
        allcx[i]=8*Xall[i]          #for the BIG with perturbation well
        allcy[i]=V2[i]-100
```

CODE

For **def making_list():**

- Is responsible for wrapping the variables together in order to be plotted



```python
def making_lists1():
    for i in range(0,Nmax):
        tempcL[i]=[cLx.tolist()[i],cLy.tolist()[i],cLz.tolist()[i]]
        tempcR[i]=[cRx.tolist()[i],cRy.tolist()[i],cRz.tolist()[i]]
        tempvl[i]=[Xleft.tolist()[i],V_L.tolist()[i],cRz.tolist()[i]]
        tempvr[i]=[Xright.tolist()[i],V_R.tolist()[i],cRz.tolist()[i]]
    for i in range(0,Nmax+addi):
        tempallc[i]=[allcx.tolist()[i],allcy.tolist()[i],allcz.tolist()[i]]
        tempvall[i]=[Xall.tolist()[i],V2.tolist()[i],allcz.tolist()[i]]

def making_lists2():
    for j in range(0,Nmax):
        tempvl2[j]=[cLx.tolist()[j],50*(RePsiL[j]**2+ImPsiL[j]**2)+150,cRz.tolist()[j]]
        tempvr2[j]=[cRx.tolist()[j],50*(RePsiR[j]**2+ImPsiR[j]**2)+150,cRz.tolist()[j]]
    for j in range(0,Nmax+addi):
        tempvall2[j]=[allcx.tolist()[j],70*(RePsi2R[j]**2+Psi2R[j]**2)+150+50*(RePsi2L[j]**2+ImPsi2L[j]**2),allcz[j]]
```

CODE

For **main body:**

- cL, cR plot the big (without perturbation) wells → red - yellow
- PlotObj,PlotObjR plot the small (without perturbation) wells → red – yellow
- allc plots the big (with perturbation) well → green
- PlotAllR plots the small (with perturbation) well → blue
- Using the small Xleft, Xright:

  I) we compute the initial left Ψ (RePsiL, ImPsiL, Rho=RePsiL^2+ ImPsiL^2)

  For 0-225 (-18 - 0): normal calculation of RePsi2L and ImPsi2L

  For 225-450 (0 - 18): RePsi2L=ImPsi2L=0

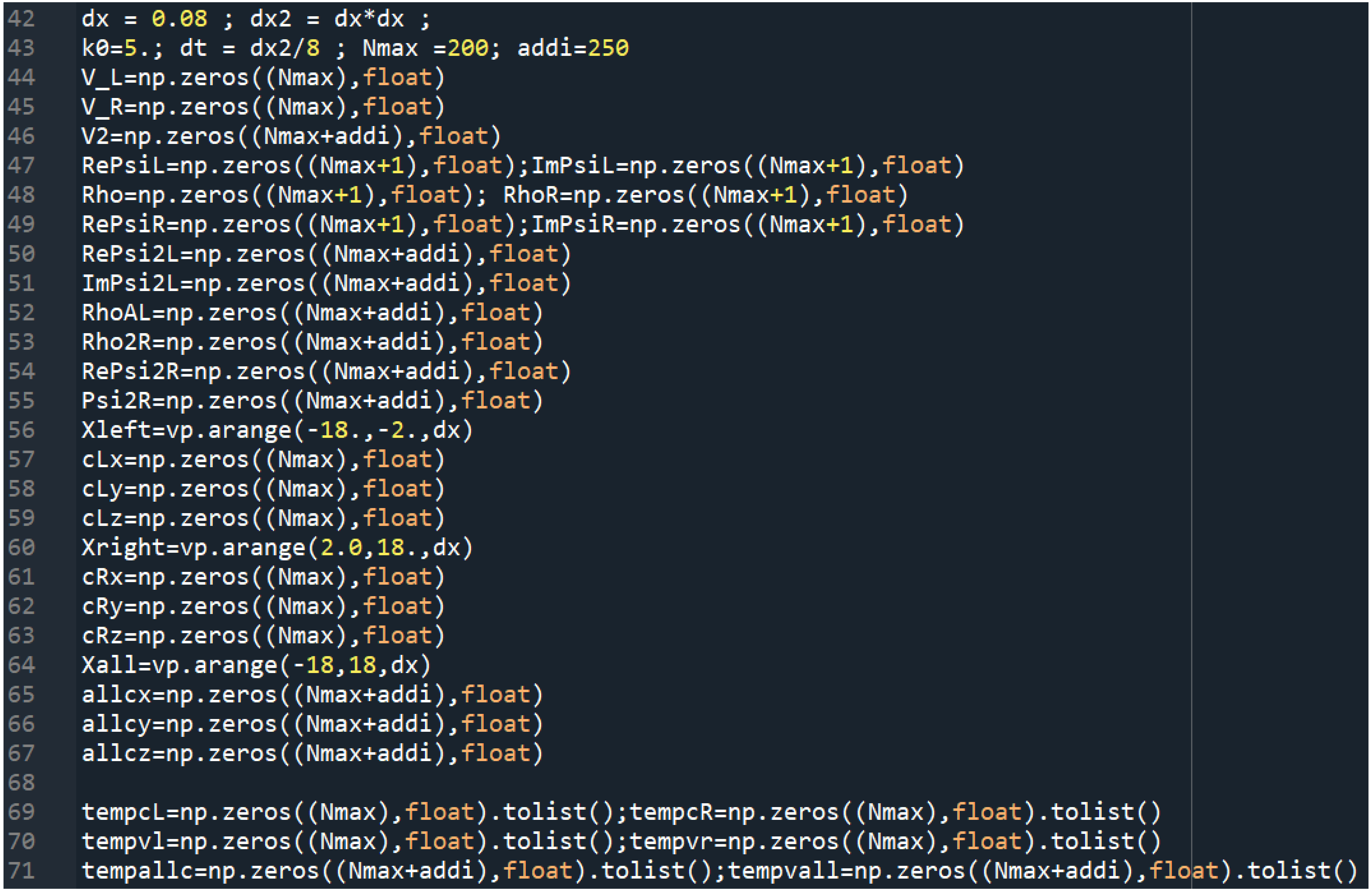
  RhoAL=50*(RePsi2L^2+ImPsi2L^2)

  II) we compute the initial right Ψ (RePsiR, ImPsiR, RhoR=RePsiR^2+ ImPsiR^2)

  For 0-225 (-18 - 0): RePsi2R=ImPsi2R=0

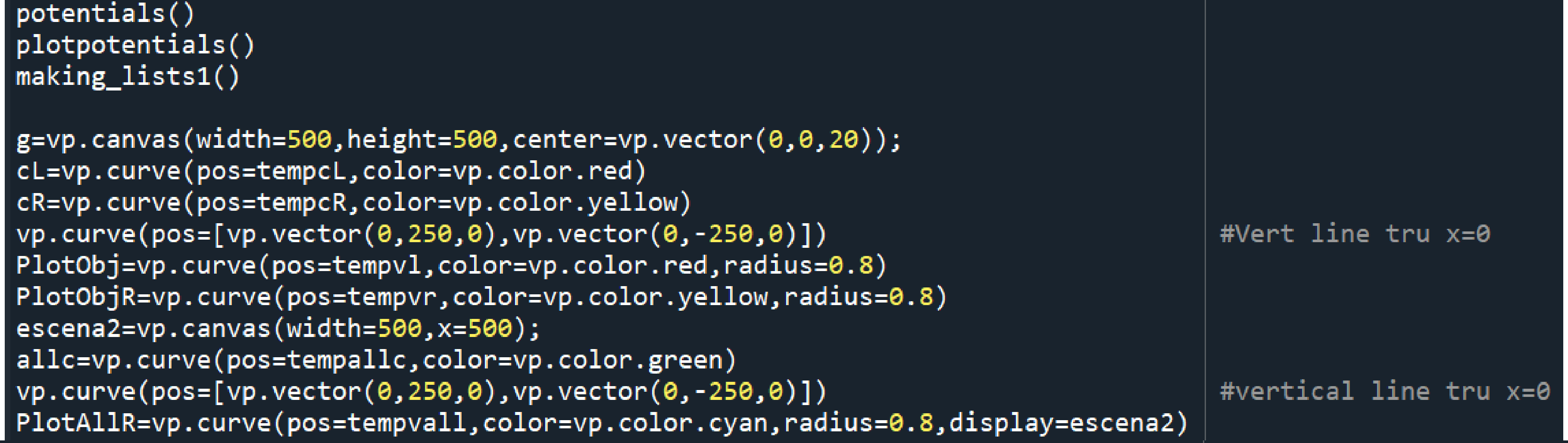
  For 225-450 (0 - 18): normal calculation of RePsi2R and ImPsi2R

  Rho2R=50*(RePsi2R^2+ImPsi2R^2)



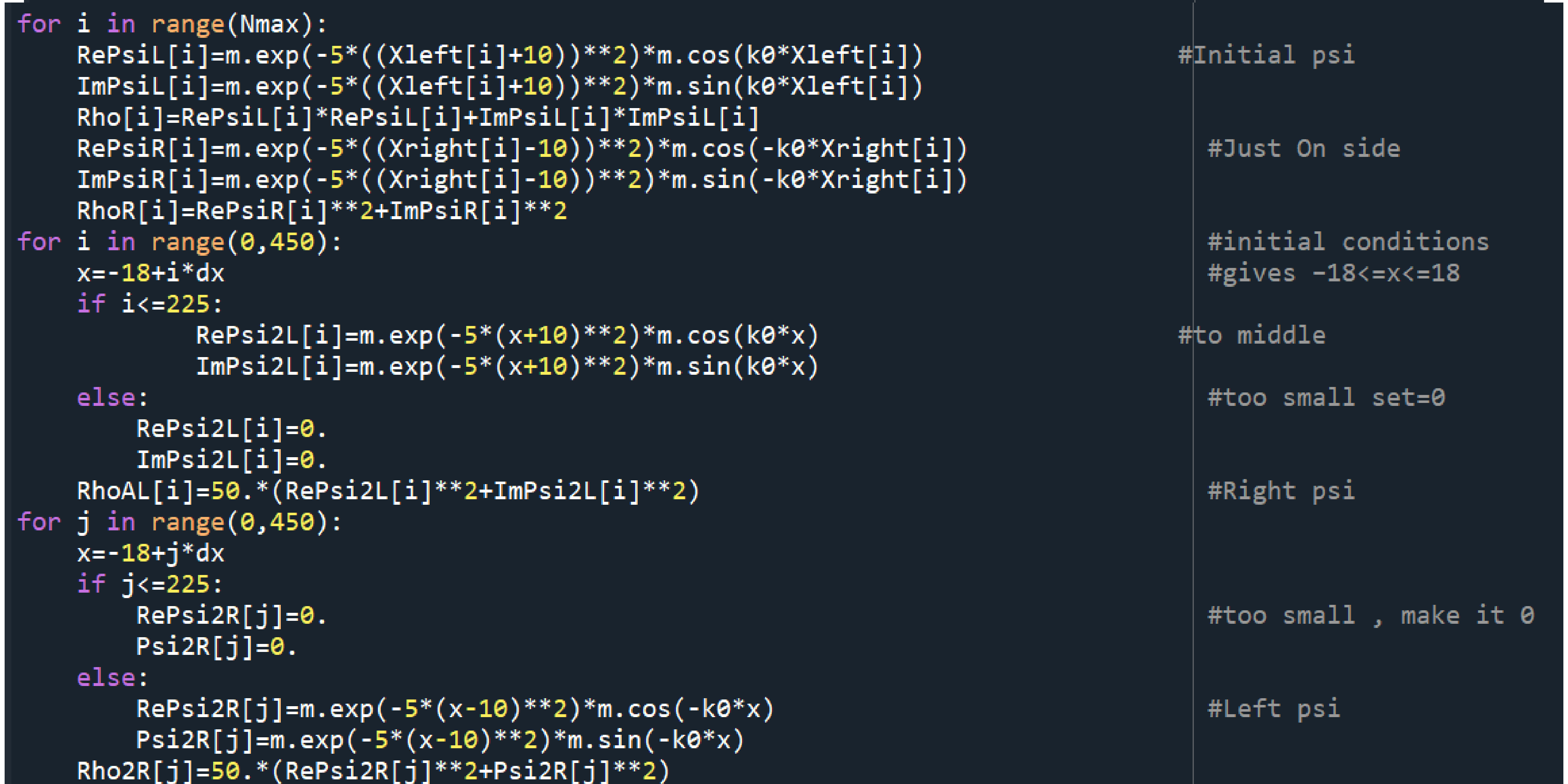
```python
dx = 0.08 ; dx2 = dx*dx ;
k0=5.; dt = dx2/8 ; Nmax =200; addi=250
V_L=np.zeros((Nmax),float)
V_R=np.zeros((Nmax),float)
V2=np.zeros((Nmax+addi),float)
RePsiL=np.zeros((Nmax+1),float);ImPsiL=np.zeros((Nmax+1),float)
Rho=np.zeros((Nmax+1),float); RhoR=np.zeros((Nmax+1),float)
RePsiR=np.zeros((Nmax+1),float);ImPsiR=np.zeros((Nmax+1),float)
RePsi2L=np.zeros((Nmax+addi),float)
ImPsi2L=np.zeros((Nmax+addi),float)
RhoAL=np.zeros((Nmax+addi),float)
Rho2R=np.zeros((Nmax+addi),float)
RePsi2R=np.zeros((Nmax+addi),float)
Psi2R=np.zeros((Nmax+addi),float)
Xleft=vp.arange(-18.,-2.,dx)
cLx=np.zeros((Nmax),float)
cLy=np.zeros((Nmax),float)
cLz=np.zeros((Nmax),float)
Xright=vp.arange(2.0,18.,dx)
cRx=np.zeros((Nmax),float)
cRy=np.zeros((Nmax),float)
cRz=np.zeros((Nmax),float)
Xall=vp.arange(-18,18,dx)
allcx=np.zeros((Nmax+addi),float)
allcy=np.zeros((Nmax+addi),float)
allcz=np.zeros((Nmax+addi),float)

tempcL=np.zeros((Nmax),float).tolist();tempcR=np.zeros((Nmax),float).tolist()
tempvl=np.zeros((Nmax),float).tolist();tempvr=np.zeros((Nmax),float).tolist()
tempallc=np.zeros((Nmax+addi),float).tolist();tempvall=np.zeros((Nmax+addi),float).tolist()
```



```
potentials()
plotpotentials()
making_lists1()

g=vp.canvas(width=500,height=500,center=vp.vector(0,0,20));
cL=vp.curve(pos=tempcL,color=vp.color.red)
cR=vp.curve(pos=tempcR,color=vp.color.yellow)
vp.curve(pos=[vp.vector(0,250,0),vp.vector(0,-250,0)])                         #Vert line tru x=0
PlotObj=vp.curve(pos=tempvl,color=vp.color.red,radius=0.8)
PlotObjR=vp.curve(pos=tempvr,color=vp.color.yellow,radius=0.8)
escena2=vp.canvas(width=500,x=500);
allc=vp.curve(pos=tempallc,color=vp.color.green)
vp.curve(pos=[vp.vector(0,250,0),vp.vector(0,-250,0)])                         #vertical line tru x=0
PlotAllR=vp.curve(pos=tempvall,color=vp.color.cyan,radius=0.8,display=escena2)
```



```
for i in range(Nmax):
    RePsiL[i]=m.exp(-5*((Xleft[i]+10))**2)*m.cos(k0*Xleft[i])                 #Initial psi
    ImPsiL[i]=m.exp(-5*((Xleft[i]+10))**2)*m.sin(k0*Xleft[i])
    Rho[i]=RePsiL[i]*RePsiL[i]+ImPsiL[i]*ImPsiL[i]
    RePsiR[i]=m.exp(-5*((Xright[i]-10))**2)*m.cos(-k0*Xright[i])                #Just On side
    ImPsiR[i]=m.exp(-5*((Xright[i]-10))**2)*m.sin(-k0*Xright[i])
    RhoR[i]=RePsiR[i]**2+ImPsiR[i]**2
for i in range(0,450):                                                          #initial conditions
    x=-18+i*dx                                                                  #gives -18<=x<=18
    if i<=225:
            RePsi2L[i]=m.exp(-5*(x+10)**2)*m.cos(k0*x)                        #to middle
            ImPsi2L[i]=m.exp(-5*(x+10)**2)*m.sin(k0*x)
    else:                                                                       #too small set=0
        RePsi2L[i]=0.
        ImPsi2L[i]=0.
    RhoAL[i]=50.*(RePsi2L[i]**2+ImPsi2L[i]**2)                                  #Right psi
for j in range(0,450):
    x=-18+j*dx
    if j<=225:
        RePsi2R[j]=0.                                                           #too small , make it 0
        Psi2R[j]=0.
    else:
        RePsi2R[j]=m.exp(-5*(x-10)**2)*m.cos(-k0*x)                             #Left psi
        Psi2R[j]=m.exp(-5*(x-10)**2)*m.sin(-k0*x)
    Rho2R[j]=50.*(RePsi2R[j]**2+Psi2R[j]**2)
```

CODE

RightPsi:

- $\Psi_{initial} = e^{-5(Xleft+10)^2} * e^{ik_0 Xleft}$
- $Rho = \Psi_{initial} * \Psi_{initial}$

LeftPsi:

- $\Psi_{side} = e^{-5(Xright-10)^2} * e^{-ik_0 Xright}$
- $RhoR = \Psi_{side} * \Psi_{side}$

OLD

- PlotObj→tempvl→[Xleft,V_L=10*(xL+10)^2/2,cRz]
- PlotObjR→tempvr→[Xright,V_R,cRz]
- PlotAllR→tempvall→[Xall,V2,allcz]

NEW

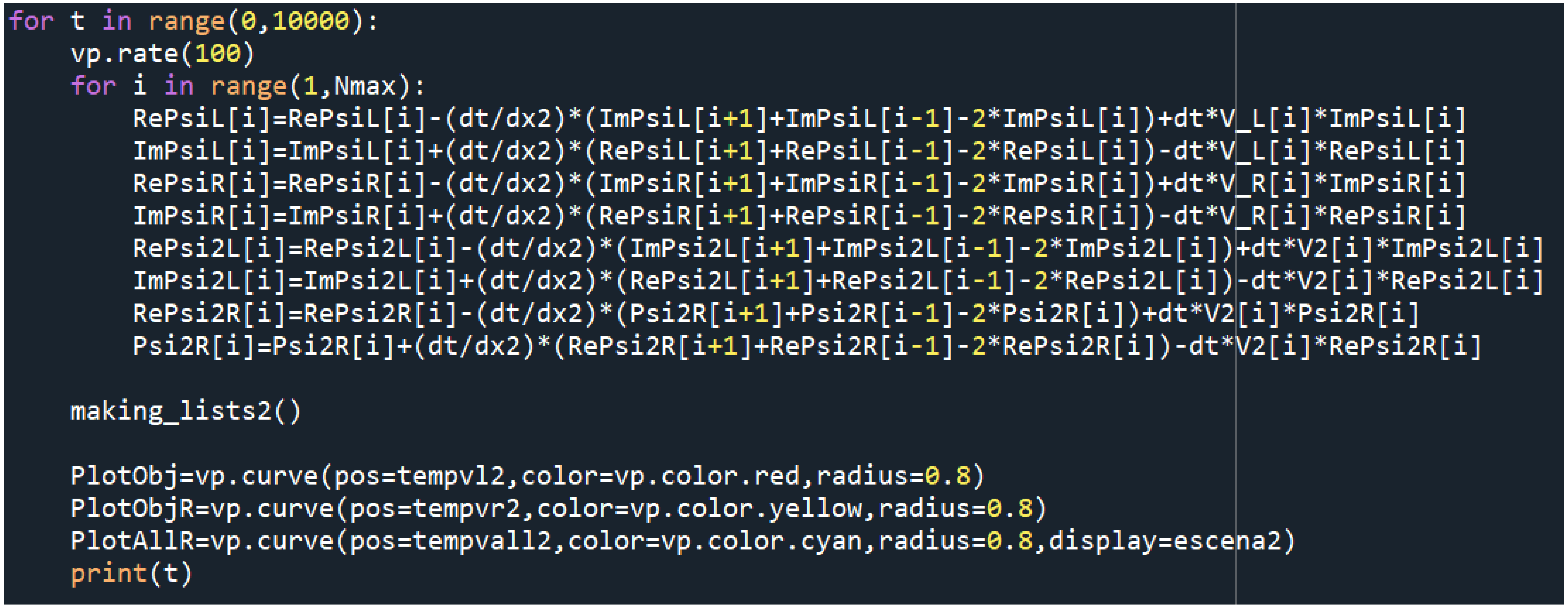
- PlotObj→tempv2→[cLx,50*(RePsiL^2+ImPsiL^2)+150,cRz]
- PlotObjR→tempvr2→[cRx=10*Xright-15, 50*(RePsiR^2+ImPsiR^2)+150, cRz]
- PlotAllR→tempvall2→[allcx=8*Xall,
  70*(RePsi2R^2+Psi2R^2)+150+50*(RePsi2L^2+ImPsiL^2), allcz]

0 - 225	225 - 450

RePsi2L=0
ImPsi2L=0
	RePsi2L=0
	ImPsi2L=0

RhoAL=50*(RePsi2L^2+ImPsi2L^2)=0	

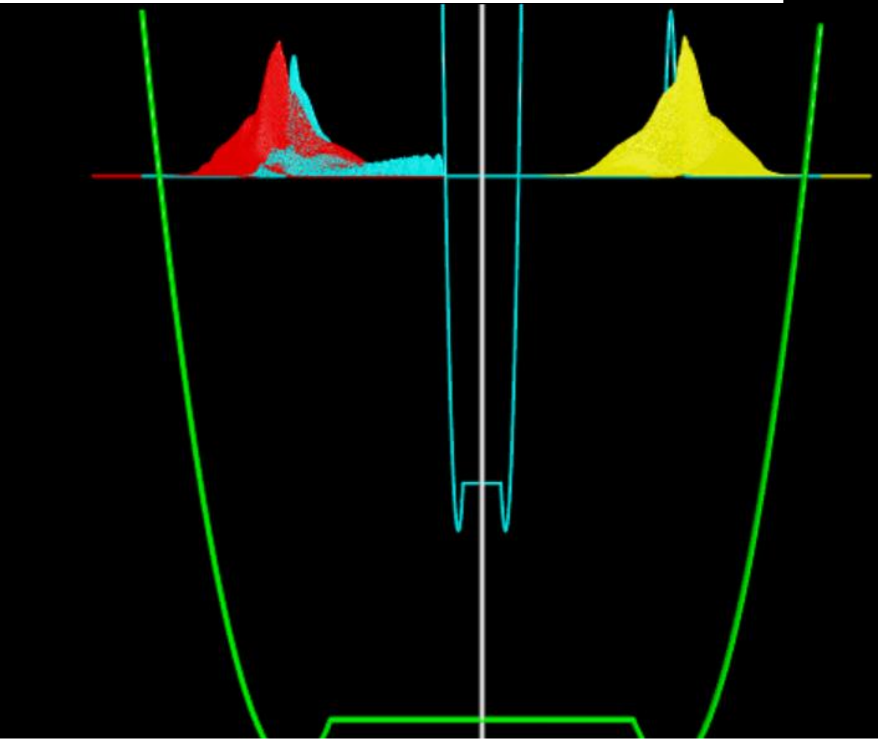
RePsi2R=0
ImPsi2R=0
	RePsi2R=$e^{-5(Xright-10)^{\wedge}2}$ cos(-k$_o$x)
	ImPsi2R=$e^{-5(Xright-10)^{\wedge}2}$ sin((-k$_o$x)

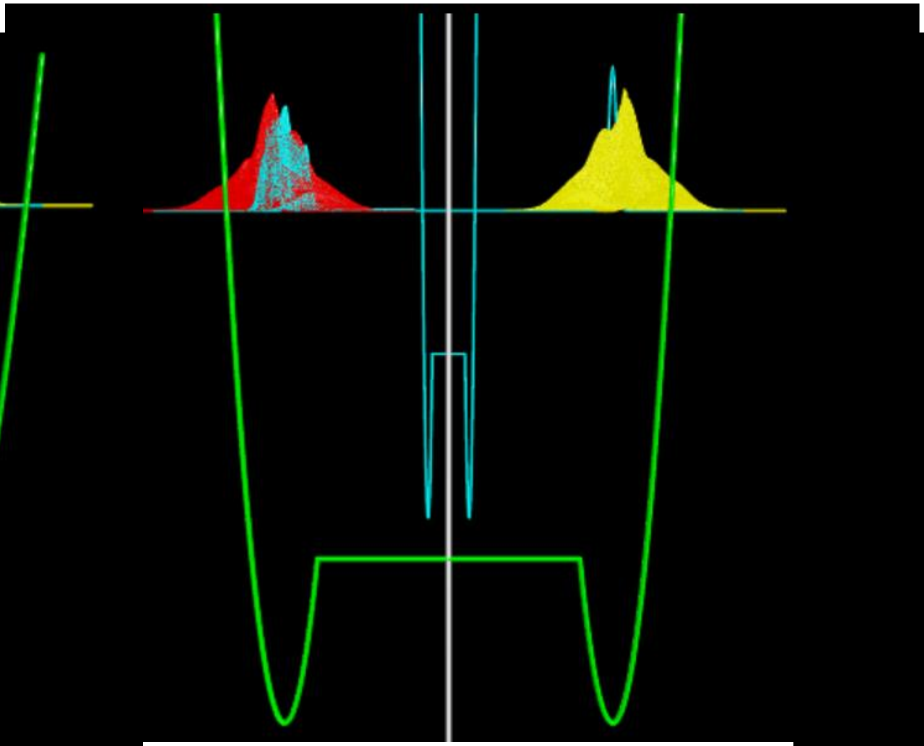


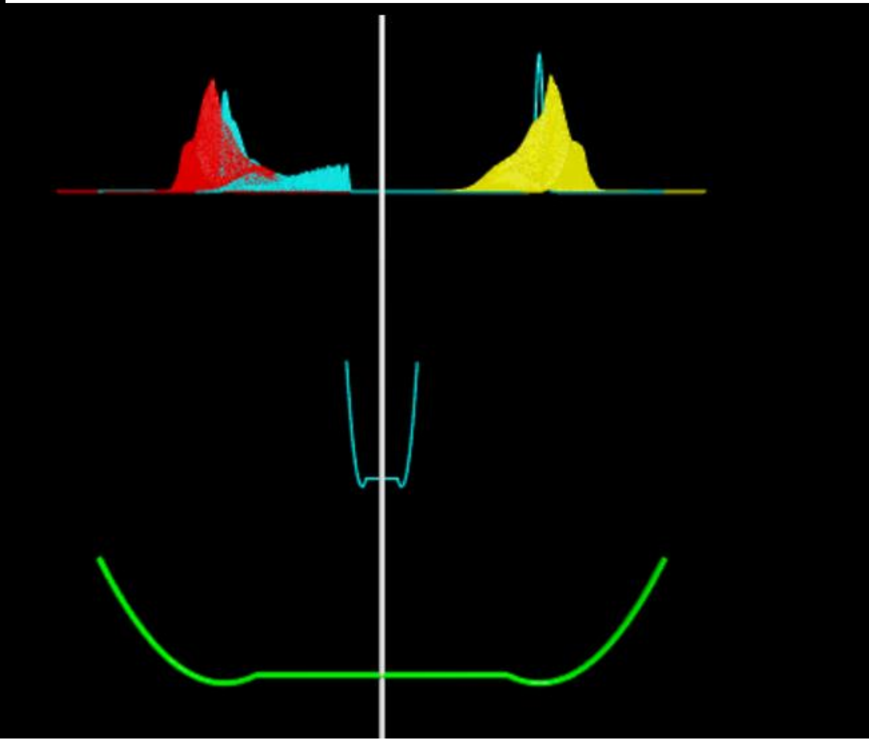
```
for t in range(0,10000):
    vp.rate(100)
    for i in range(1,Nmax):
        RePsiL[i]=RePsiL[i]-(dt/dx2)*(ImPsiL[i+1]+ImPsiL[i-1]-2*ImPsiL[i])+dt*V_L[i]*ImPsiL[i]
        ImPsiL[i]=ImPsiL[i]+(dt/dx2)*(RePsiL[i+1]+RePsiL[i-1]-2*RePsiL[i])-dt*V_L[i]*RePsiL[i]
        RePsiR[i]=RePsiR[i]-(dt/dx2)*(ImPsiR[i+1]+ImPsiR[i-1]-2*ImPsiR[i])+dt*V_R[i]*ImPsiR[i]
        ImPsiR[i]=ImPsiR[i]+(dt/dx2)*(RePsiR[i+1]+RePsiR[i-1]-2*RePsiR[i])-dt*V_R[i]*RePsiR[i]
        RePsi2L[i]=RePsi2L[i]-(dt/dx2)*(ImPsi2L[i+1]+ImPsi2L[i-1]-2*ImPsi2L[i])+dt*V2[i]*ImPsi2L[i]
        ImPsi2L[i]=ImPsi2L[i]+(dt/dx2)*(RePsi2L[i+1]+RePsi2L[i-1]-2*RePsi2L[i])-dt*V2[i]*RePsi2L[i]
        RePsi2R[i]=RePsi2R[i]-(dt/dx2)*(Psi2R[i+1]+Psi2R[i-1]-2*Psi2R[i])+dt*V2[i]*Psi2R[i]
        Psi2R[i]=Psi2R[i]+(dt/dx2)*(RePsi2R[i+1]+RePsi2R[i-1]-2*RePsi2R[i])-dt*V2[i]*RePsi2R[i]

    making_lists2()

    PlotObj=vp.curve(pos=tempvl2,color=vp.color.red,radius=0.8)
    PlotObjR=vp.curve(pos=tempvr2,color=vp.color.yellow,radius=0.8)
    PlotAllR=vp.curve(pos=tempvall2,color=vp.color.cyan,radius=0.8,display=escena2)
    print(t)
```


Normal perturbation height (5c)


Higher perturbation height (20c)


Lower perturbation height (c)

LISTING 6.19

FEYNMAN PATH INTEGRAL QUANTUM MECHANICS

- Feynman postulated that the quantum–mechanical wave function describing the propagation of a free particle from the space-time point a=(x_a,t_a) to the point b=(x_b,t_b) can be expressed as:

$$\psi(x_b, t_b) = \int G(x_b, t_b; x_a t_a)\psi(x_a, t_a)dx_a$$

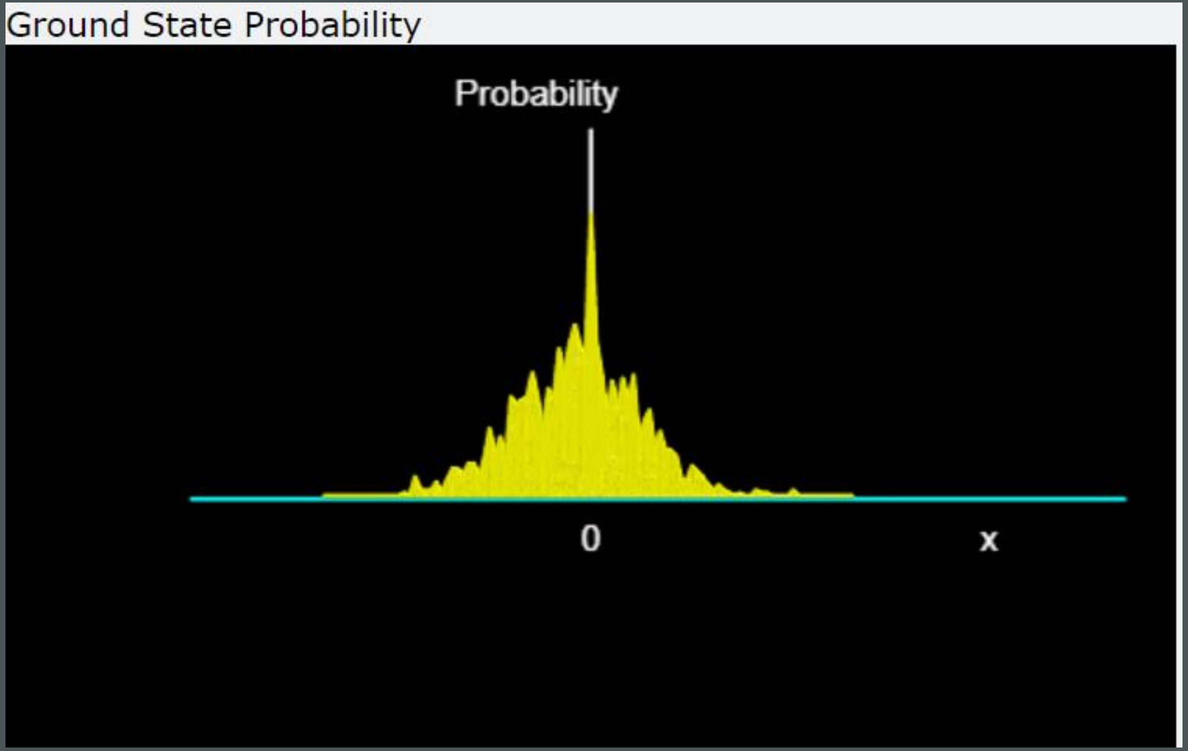
Where G is the Green 's function propagator

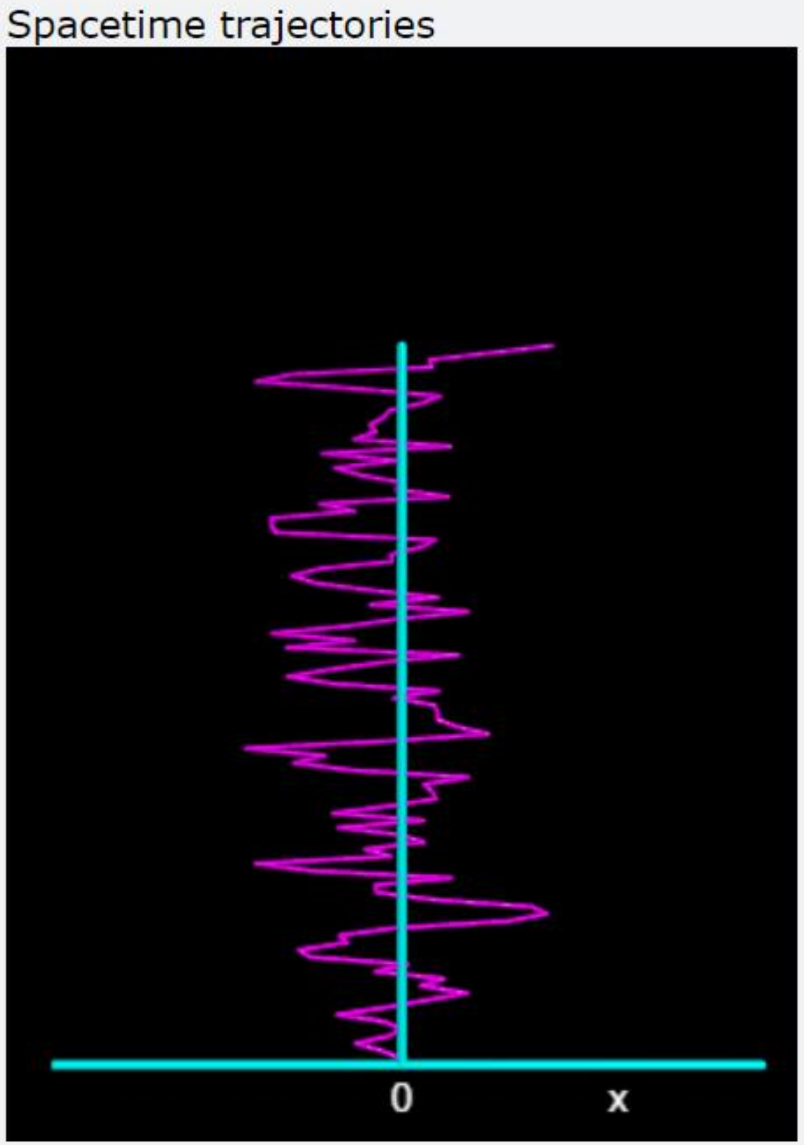
$$G(x_b, t_b; x_a t_a) \equiv G(b, a) = \sqrt{\frac{m}{2\pi i(t_b - t_a)}}\, e^{i\frac{m(x_b - x_a)^2}{2(t_b - t_a)}} = \sum_{paths} e^{iS[b,a]/\hbar} \text{ (path integral)}$$

- Action is: $S[b, a] = \frac{m}{2}\frac{(x_b - x_a)^2}{t_b - t_a}$
- The connection between the bound wave state wave function and the Green 's function is:

$$|\psi_0\ (x)|^2 = \lim_{\tau\to\infty} e^{E_0\tau} G(x, -i\tau; x, 0) = \frac{1}{Z}\lim_{\tau\to\infty}\int e^{-\varepsilon E} dx_1 \dots dx_{N-1}, \text{where } Z = \frac{1}{Z}\lim_{\tau\to\infty}\int e^{-\varepsilon E} dx dx_1 \dots dx_{N-1}.$$

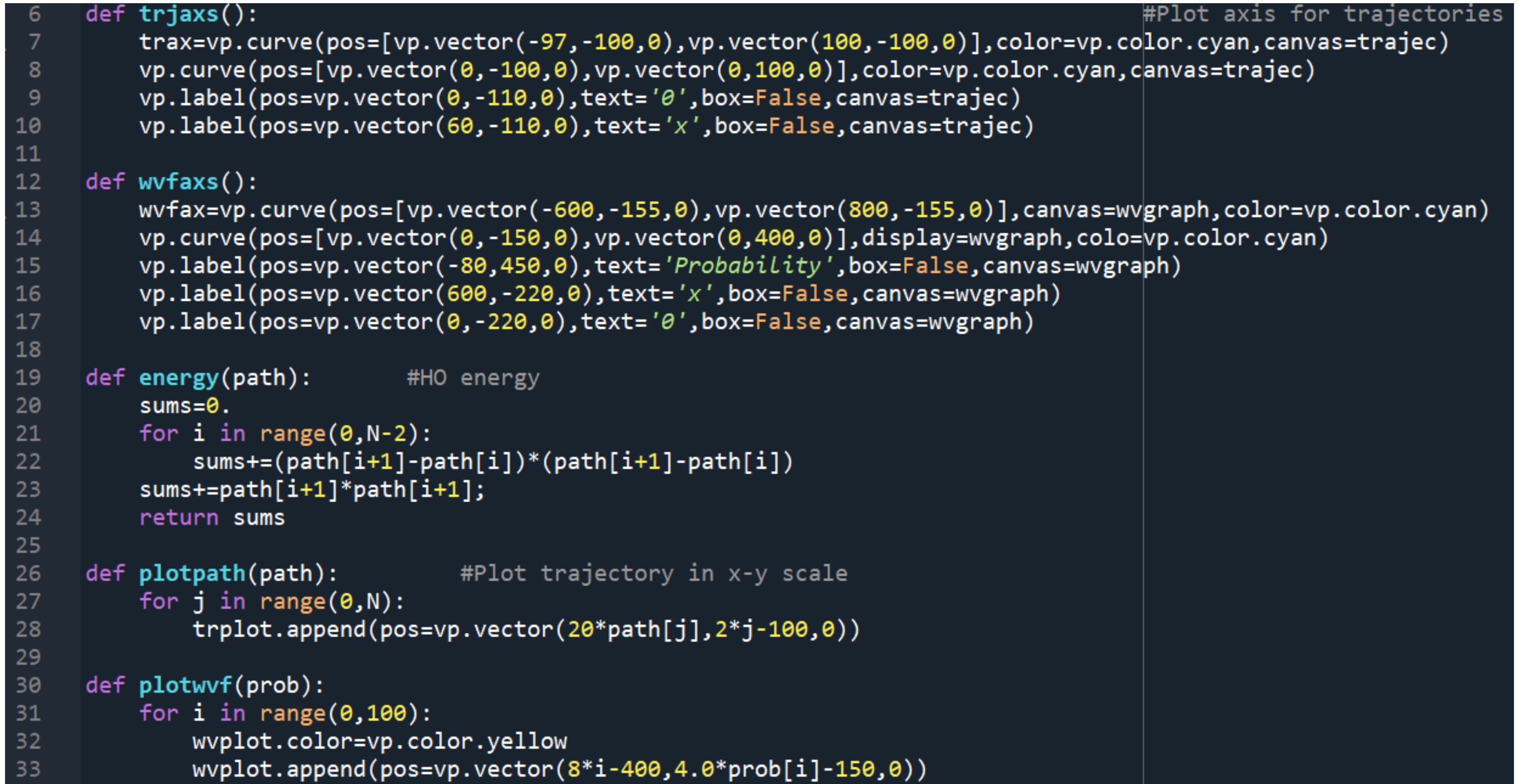
- Basically, the program sets up a lattice in space and time so that positions and times are discrete, with integrals evaluated as sums over values at the lattice points, and derivatives evaluated as the differences in values at successive lattice points.
- A Metropolis algorithm is used to vary the trajectory and search for the state with lowest energy. The classical path (least action) remains most likely, with nearby paths being less likely to occur.





CODE

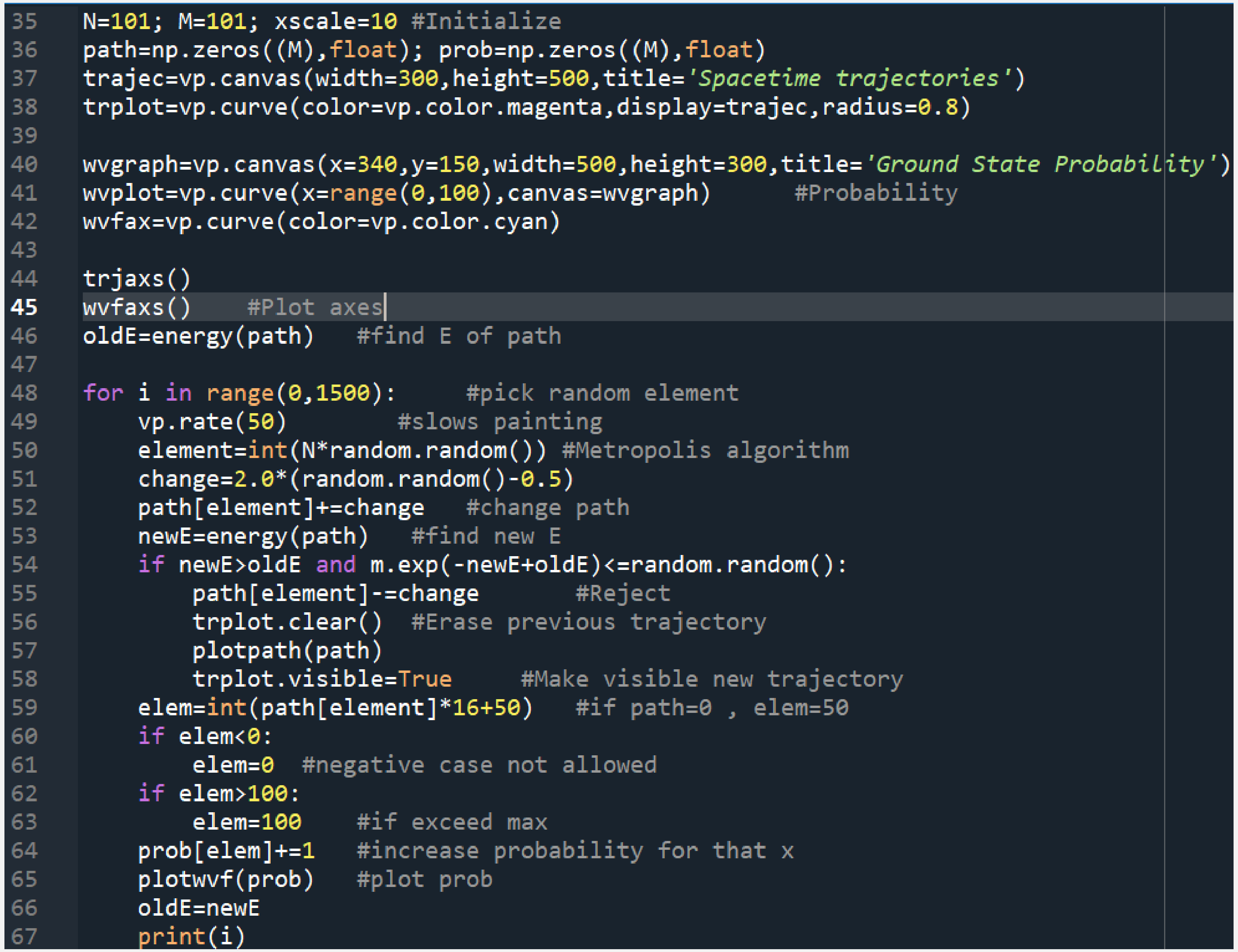
- Line 6-17: plots the axis for the trajectories and the probability to take a certain route.
- Line 19-33: plots the different paths a wave packet can take, their respective probabilities and energy values for each step for every single route



```
 6  def trjaxs():                                                                      #Plot axis for trajectories
 7      trax=vp.curve(pos=[vp.vector(-97,-100,0),vp.vector(100,-100,0)],color=vp.color.cyan,canvas=trajec)
 8      vp.curve(pos=[vp.vector(0,-100,0),vp.vector(0,100,0)],color=vp.color.cyan,canvas=trajec)
 9      vp.label(pos=vp.vector(0,-110,0),text='0',box=False,canvas=trajec)
10      vp.label(pos=vp.vector(60,-110,0),text='x',box=False,canvas=trajec)
11
12  def wvfaxs():
13      wvfax=vp.curve(pos=[vp.vector(-600,-155,0),vp.vector(800,-155,0)],canvas=wvgraph,color=vp.color.cyan)
14      vp.curve(pos=[vp.vector(0,-150,0),vp.vector(0,400,0)],display=wvgraph,colo=vp.color.cyan)
15      vp.label(pos=vp.vector(-80,450,0),text='Probability',box=False,canvas=wvgraph)
16      vp.label(pos=vp.vector(600,-220,0),text='x',box=False,canvas=wvgraph)
17      vp.label(pos=vp.vector(0,-220,0),text='0',box=False,canvas=wvgraph)
18
19  def energy(path):       #HO energy
20      sums=0.
21      for i in range(0,N-2):
22          sums+=(path[i+1]-path[i])*(path[i+1]-path[i])
23      sums+=path[i+1]*path[i+1];
24      return sums
25
26  def plotpath(path):         #Plot trajectory in x-y scale
27      for j in range(0,N):
28          trplot.append(pos=vp.vector(20*path[j],2*j-100,0))
29
30  def plotwvf(prob):
31      for i in range(0,100):
32          wvplot.color=vp.color.yellow
33          wvplot.append(pos=vp.vector(8*i-400,4.0*prob[i]-150,0))
```

CODE

- Line 35-45: the plots are initialized
- Line 46: Initial energy is calculated
- Line 48-67: random path elements are induced through Metropolis algorithm and new energies are calculated in order to make new trajectories and achieve higher probabilities



```
35  N=101; M=101; xscale=10 #Initialize
36  path=np.zeros((M),float); prob=np.zeros((M),float)
37  trajec=vp.canvas(width=300,height=500,title='Spacetime trajectories')
38  trplot=vp.curve(color=vp.color.magenta,display=trajec,radius=0.8)
39  
40  wvgraph=vp.canvas(x=340,y=150,width=500,height=300,title='Ground State Probability')
41  wvplot=vp.curve(x=range(0,100),canvas=wvgraph)      #Probability
42  wvfax=vp.curve(color=vp.color.cyan)
43  
44  trjaxs()
45  wvfaxs()     #Plot axes
46  oldE=energy(path)   #find E of path
47  
48  for i in range(0,1500):     #pick random element
49      vp.rate(50)        #slows painting
50      element=int(N*random.random()) #Metropolis algorithm
51      change=2.0*(random.random()-0.5)
52      path[element]+=change   #change path
53      newE=energy(path)   #find new E
54      if newE>oldE and m.exp(-newE+oldE)<=random.random():
55          path[element]-=change       #Reject
56          trplot.clear()  #Erase previous trajectory
57          plotpath(path)
58          trplot.visible=True     #Make visible new trajectory
59      elem=int(path[element]*16+50)   #if path=0 , elem=50
60      if elem<0:
61          elem=0  #negative case not allowed
62      if elem>100:
63          elem=100    #if exceed max
64      prob[elem]+=1   #increase probability for that x
65      plotwvf(prob)   #plot prob
66      oldE=newE
67      print(i)
```

THANK YOU FOR YOUR TIME!