Computer Modeling and Simulation Project

Catherine Slaughter

August 5, 2020

1 Exploration Problem

The purpose of this problem is to plot the probability of a spin-flip for a spin-1/2 particle under the influence of the time-dependent Hamiltonian

$$H = \epsilon(t)S_z + \Delta S_x$$

Where $\epsilon(t) = \frac{E}{T}(2t - T)$ and E and Δ are constants such that $E \gg \Delta$. For my program, I somewhat arbitrarily chose Δ to be 1 and E to be 10^4. These values are acceptable because $E \gg \Delta$ holds. I also chose $0 \leq T \leq 20$ to be the available values of T, and have $\Delta t = T/1000$ update as T does, so Δt is always 1/1000 the length of the X-axis. This is sufficiently small for a smooth-looking plot. Later in the program, I choose my Y-axis scale to be whatever is needed to see the plot well for all available values of T.

One way to go about solving this problem is to split the Hamiltonian up into small sections of width Δt, and treat the Hamilton between t_0 and $t_0 + \Delta t$ as a constant equal to the Hamiltonian at $t = t_0$. An illustration of this process over one block of time can be seen below.

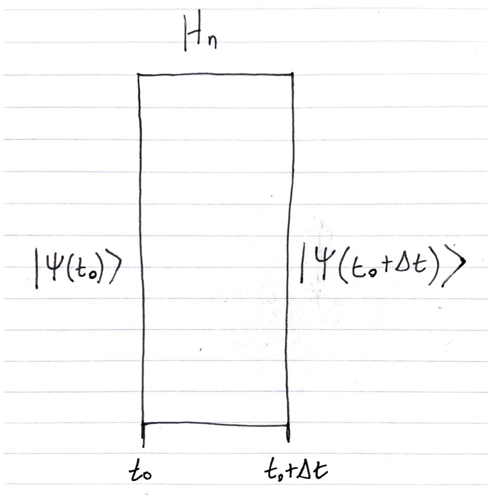


Figure 1: One section of H of width Δt

The output vector $|\Psi(t_0 + \Delta t)\rangle$ can be found using a complex exponential formula.

$$|\Psi(t_0 + \Delta t)\rangle = e^{-\frac{i}{\hbar}H_n\Delta t}|\Psi(t_0)\rangle$$

Breaking down H into small sections like this is very easy in MATLAB. By defining t as a vector with the first value 0 and the last value T with steps of Δt, most of the work is already done. From there $\epsilon(t)$ and H are easily calculated for values of t separated by Δt using MATLAB's element-by-element arithmetic functionality. For each value in the t vector, an $\epsilon(t)$ is calculated. From this, an H, $|\Psi(t_0)\rangle$ (plugging in the previous $|\Psi(t_0 - \Delta t)\rangle$), and a probability ($||\Psi(t_0)\rangle|^2$)are calculated. Finally, the probability is plotted vs. t, an example plot can be found below.

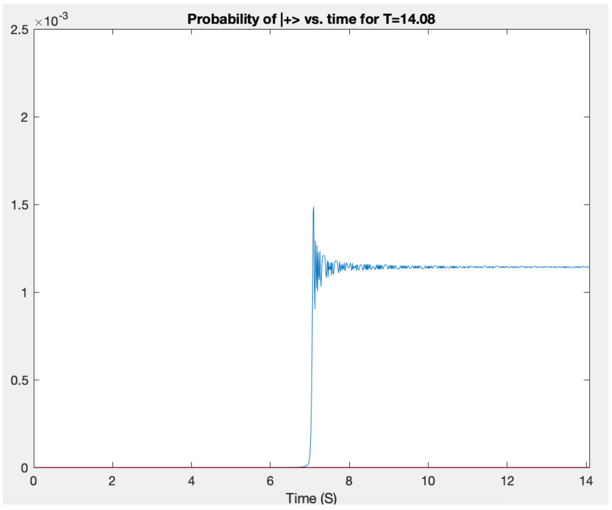


Figure 2: The probability of a spin-flip vs. t for T=14.08

A simple graphical user interface is also included, it has a slider to adjust T and a button to stop the program running. When T is adjusted, the bounds of the plot x-axis also switch so $0 \leq t \leq T$. The bounds of the y-axis remain constant so it is easy to see the relative values of the probability for a given T.

For a given value of T, the probability of a spin-flip hovers just above 0 for $0 < t < T/2$. Then, at

$t = T/2$, it quickly shoots up to a maximum value, before oscillating decay to a steady value. As T increases, both the maximum value of the probability and the probability at T increase. This makes sense, because t is bounded above by T, and it is expected that the longer the system is allowed to run, the higher the probability that a spin-flip occurs.

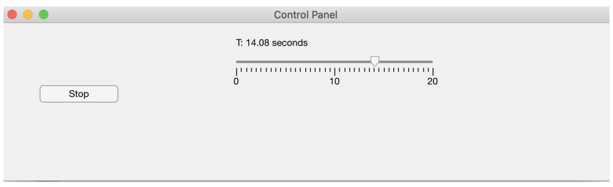


Figure 3: The GUI for Problem 1

2 McIntyre 5.32

For this problem, the goal is to recreate the plot in Fig. 5.24, which shows the values of three different energy level states over varying sizes for GaAs quantum wells.

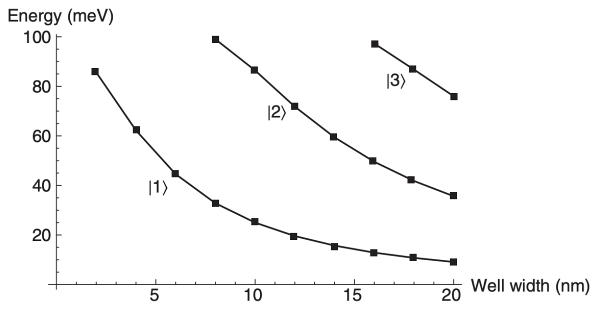


Figure 4: Fig. 5.24 from McIntyre

We can begin this problem by looking at the transcendental equations (5.88) from McIntyre:

$$ztan(z) = \sqrt{z_0^2 - z^2}$$

$$-zcot(z) = \sqrt{z_0^2 - z^2}$$

and the associated values z and z_0:

$$z = \sqrt{\frac{2mEa^2}{\hbar^2}}$$

$$z_0 = \sqrt{\frac{2mV_0a^2}{\hbar^2}}$$

The first step toward starting this problem is to find values of z_0 for varying well widths (2*a). This is done in the program mcintyre5_32.m by creating a list of well values ranging from 2 to 20, and calculating a list of z_0s from that. From there, a loop is used to find the value(s) of z that solve the transcendental equations for each z_0. The program stores the values in the column vectors of an n-by-3 matrix, where n is the number of z_0 values used. In doing so, the matrix lines up such that the 1st, 2nd, and 3rd rows of the z matrix correspond to the 1st, 2nd, and 3rd energy level lines in the plot.

Next, for each row vector, the program calculates a vector of associated E values. From the equations for z and z_0, we can find E by:

$$\frac{z^2}{z_0^2} = \frac{E}{V_0}$$

$$E = \frac{z^2}{z_0^2}V_0$$

Finally, each energy level vector is appropriately plotted as a line. The final plot can be found below.

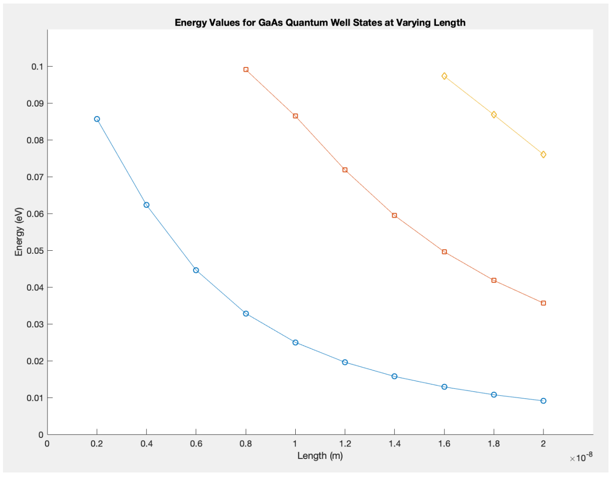


Figure 5: Final plot from MATLAB

3 McIntyre 6.11

The basic setup for this program is decently simple, eq. 6.52 (in blue) and the square root of eq. 6.53 (in orange) are plotted on the same axes for initial conditions $\beta = 1$,$t = 0$, and $p_0 = 1$.

eq. 6.52:

$$\psi = (\frac{1}{2\pi\alpha^2})^{1/4}\frac{1}{\sqrt{\gamma}}e^{ip_0(x-p_ot/2m)/\hbar}e^{-(x-p_ot/m)^2/4\alpha^2\gamma}$$

eq. 6.53: $P = \frac{1}{\sqrt{2\pi}\alpha\Gamma}e^{-(x-p_ot/m)^2/2\alpha^2\Gamma^2}$

Where $\gamma = 1 + \frac{it}{\tau}$, $\tau = \frac{m\hbar}{2\beta^2}$, $\alpha = \frac{\hbar}{2\beta}$, and $\Gamma = \sqrt{1 + \frac{t^2}{\tau^2}}$

6.52 is the sinusoidal wave function, and 6.53 is the probability density. The square root of 6.53 represents the motion of the wave packet envelope, as explained in chapter 6 of McIntyre.

The program is animated using a while loop, which increments t by a small amount before re-calculating and re-plotting both equations. After t reaches a pre-determined maximum value, it is reset to 0 so the animation appears to loop. There is also a simple graphical user interface which includes sliders for β and p_0. Every time the while loop runs again, these values are taken in to recalculate the equations. The GUI also displays the current time t in nanoseconds (because the animation is slowed down significantly) and the current values of Δx and Δp. Finally, there are buttons to pause the time change and to stop the program all together.

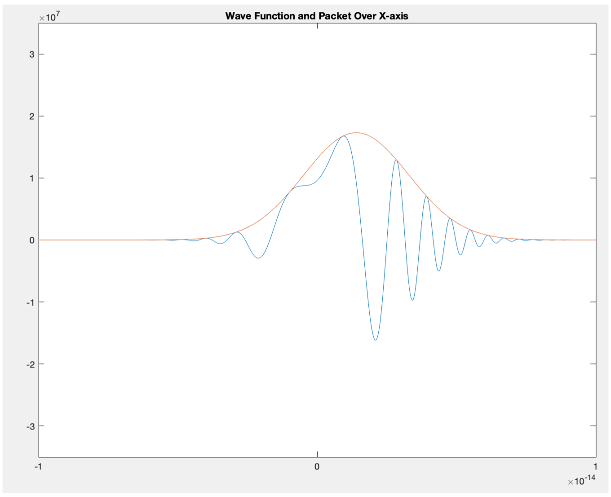
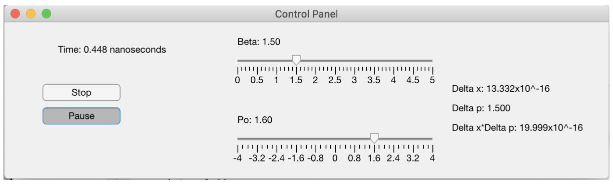


Figure 6: An example of a plot generated by the program. In this image, t=.448 ns, β=1.50 and p_o=1.60



Time: 0.448 nanoseconds

Beta: 1.50

Po: 1.60

Delta x: 13.332x10^-16

Delta p: 1.500

Delta x*Delta p: 19.999x10^-16

Figure 7: The GUI

After the Stop button is pressed for the first time, a plot of $\Delta x \Delta p$ vs. time comes up. Where

$\Delta x = \alpha \Gamma$

and

$\Delta p = \beta$

as is outlined in McIntyre section 6.2

For this graph, the user can again change the β and p_0 sliders to see how doing so may change the value of $\Delta x \Delta p$ over time. As one would expect, the initial value p_0 does not change $\Delta x \Delta p$. It should be noted that the x axis of the plot starts just before t=0, to give the user a better look at what is happening at t=0.

The most important trend to note is that $\Delta x \Delta p$ equals $\frac{\hbar}{2}$ (shown with the orange line) at t=0 and increases with t for all values of β. This makes sense because the Heisenberg uncertainty principle says that $\Delta x \Delta p \leq \frac{\hbar}{2}$. Additionally, it stands to reason that $\Delta x \Delta p$ should increase with time because the width of the distribution in momentum space doesn't change with time, and (as can be seen in the first part of the program) the width of the distribution in x clearly increases with time. The width can be represented as $2\Delta x$ for x-space and $2\Delta p$ for p-space. Analytically, the same result regarding $\Delta x \Delta p$ can be achieved, as $2\Delta x = 2\alpha\Gamma$ increases with t based on the t-dependence of Γ and $2\Delta p = 2\beta$ has no t-dependence. Hitting the Stop button one last time ends the program. An example plot can be seen below.

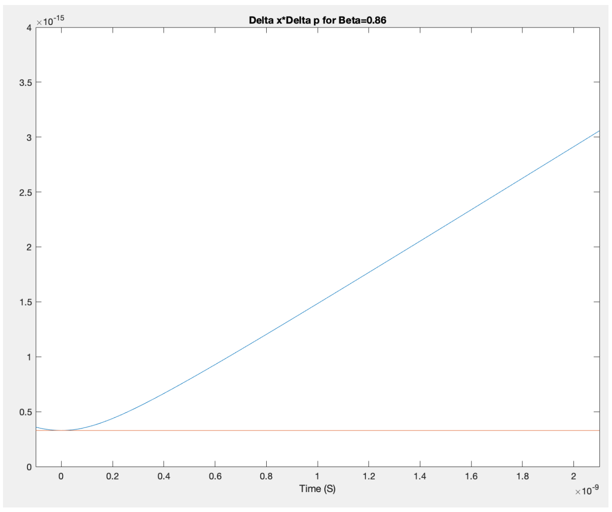


Figure 8: The $\Delta x \Delta p$ vs. time graph for $\beta = .86$. The blue line is $\Delta x \Delta p$, the orange line is $\frac{\hbar}{2}$

The code for all three of these problems can be found on the following pages, and the .m files will be included in a .zip with the assignment.

Exploration Problem Code:

```
%control panel setup
control=uifigure('Name','Control Panel','Position',[100,150,775,200]);

Tslid=uislider(control,'Value',10,'Position',[300,150,250,3],'Limits',[0 20], 'MajorTicks', [0,10,20]
Tlabel=uilabel(control,'Text',"T: 5 seconds",'Position',[300 160 200 32]);

stopbutton=uibutton(control, 'state','Position', [50 100 100 22],'Text', 'Stop'); %button to stop

%constants
E=1E4;
delta=1E0;
hbar=6.5821*10^-16;

%Spin matricies
Sz=(hbar/2).*[1,0;0,-1];
Sx=(hbar/2).*[0,1;1,0];

psi0=[0;1];
psiF=[1;0];

stopped=get(stopbutton,'Value');

while stopped==0

    stopped=get(stopbutton,'Value');

    T=get(Tslid, 'Value');
    Tlabel.Text = "T: "+sprintf('%0.2f',T)+" seconds"; %update label

    %time values
    deltat=T/1000;
    t=(0:deltat:T);

    %for use declaring future matricies
    siz=size(t);
    siz=siz(2);

    %epsilon vector
    epsilon=(E/T)*((2.*t)-T);

    prob = zeros(siz);

    psit=psi0;

    for n=1:siz %this loop is very time consuming...
        epn=epsilon(n);

        H = epn*Sz+delta*Sx;

        psit=expm((-1i/hbar)*deltat*H)*psit;

        prob(n)=abs(psiF'*psit)^2;
    end

    plot(t,prob)
    hold on
    xlim([0 T])
    ylim([0 2.5E-3])
    xlabel('Time (S)')
    title('Probability of |+> vs. time for T='+string(T))
    hold off

    drawnow

    pause(.001)

end
```

McIntyre 5.32 Code:

```
l=2:2:20;
l=l.*10^-9; %switches from nm to m

Vo=.1;
me=.511*10^6;
m=.067*me/(3E8)^2;
hbar=6.5821*10^-16;

zo=sqrt(2*m*Vo.*((l./2).^2)./(hbar^2)); %array of zo values for each value of l

syms z

zs = zeros(size(zo,1),3);
zoindex=0;

for x = zo
    digits(10) %changes the precision of the vpasolve function
    zoindex=zoindex+1;

    eveEq= z*tan(z) == sqrt(x^2-z^2);
    oddEq= -z*cot(z) == sqrt(x^2-z^2);

    zindex=0;

    if x == zo(9)
        min = .1;
    else
        min = 0;
    end
    %The vpasolve function screws up for zo(9) when the min is set to 0. It
    %jumps to the second-to-smallest solution for eveEq in the given range
    %instead of the smallest. Playing around some, I've concluded that this
    %is an impact of the precision of the vpasolve function. Even at the
    %default of 32-bit precision, the problem remains. In theory, I
    %could've increased the precision used by the solve function, and had
    %min be 0 for all values, but this increased the runtime of the program
    %significantly, so instead I chose the above solution.

    max = ceil(x);
    done = 0;


    while done == 0
        next = vpasolve(eveEq,z,[min max]); %starts with even solution

        if not(isempty(next))
            zindex=zindex+1;
            zs(zindex,zoindex)=next;
            min = next;
        else
            done = 1;
            min = x;
        end

        next = vpasolve(oddEq,z,[min max]);

        if not(isempty(next))
```

```
            zindex=zindex+1;
            zs(zindex,zoindex)=next;
            min = next;
        else
            done = 1;
            min = x;
        end

    end

end

z1 = zs(1,:);
z2 = zs(2,:);
z3 = zs(3,:);

E1 = ((z1.^2)./(zo.^2))*Vo;
E2 = ((z2.^2)./(zo.^2))*Vo;
E3 = ((z3.^2)./(zo.^2))*Vo;

hold on

plot(l,E1,'-o')
plot(l(4:10),E2(4:10),'-s')
plot(l(8:10),E3(8:10),'-d')
title('Energy Values for GaAs Quantum Well States at Varying Length')
xlabel('Length (m)')
ylabel('Energy (eV)')
xlim([0 2.2E-8])
ylim([0 .11])
hold off
```

McIntyre 6.11 Code:

```
%control panel setup
control=uifigure('Name','Control Panel','Position',[100,150,775,200]);

betaslid=uislider(control,'Value',1,'Position',[300,150,250,3],'Limits',[0,5]);%slider for beta
blabel=uilabel(control,'Text',"Beta: 1",'Position',[300 160 100 32]);

poslid=uislider(control,'Value',0,'Position',[300,50,250,3],'Limits',[-4,4]);%slider for po
polabel=uilabel(control,'Text',"Po: 0",'Position',[300 60 100 32]);

pausebutton=uibutton(control, 'state','Position', [50 70 100 22], 'Text', 'Pause'); %button to pause
stopbutton=uibutton(control, 'state','Position', [50 100 100 22],'Text', 'Stop'); %button to stop

tlabel=uilabel(control,'Text',"Time=0",'Position',[70 150 150 32]);
deltaxlabel=uilabel(control,'Text',"Delta x: ",'Position',[575 100 150 32]);%labels
deltaplabel=uilabel(control,'Text',"Delta p: ",'Position',[575 75 150 32]);
bothlabel=uilabel(control,'Text',"Delta x*Delta p: ",'Position',[575 50 200 32]);
%end control setup

%initial values
paused=get(pausebutton, 'Value');
stopped=get(stopbutton, 'Value');
t=0;

%constants
m=.511*10^6;
hbar=6.5821*10^-16;
x=(-10E-15:1E-18:10E-15);

while stopped==0

    paused=get(pausebutton, 'Value');%check if paused
    stopped=get(stopbutton, 'Value');%check if stopped

    if paused==0 %if not paused, loop the time from 0 to 10 microsec
        if t<2E-9
            t=t+7E-12;
        else
            t=0;
        end
        tlabel.Text = "Time: "+sprintf('%0.3f',t*10^9)+" nanoseconds"; %update label
    end

    beta=get(betaslid,'Value'); %get beta value
    blabel.Text = "Beta: "+sprintf('%0.2f',beta); %update label

    po=get(poslid,'Value');%get po value
    polabel.Text = "Po: "+sprintf('%0.2f',po); %update label

    %reset
    tau=m*hbar/(2*beta^2);
    Gamma=sqrt(1+(t^2/tau^2));%from book
    gamma=1+((1i*t)/tau);
    alpha=hbar/(2*beta);


    %x, p spreads
    deltax=alpha*Gamma;
    deltaxlabel.Text = "Delta x: "+sprintf('%0.3f',deltax*10^16)+ "x10^-16";
    deltap=beta;
    deltaplabel.Text = "Delta p: "+sprintf('%0.3f',deltap);
    bothlabel.Text = "Delta x*Delta p: "+sprintf('%0.3f',deltap*deltax*10^16)+"x10^-16";

    %calculate wave function and packet gaussian
    p=(1/(sqrt(2*pi)*alpha*Gamma))*exp((-(x-(po*t/m)).^2)./(2*alpha^2*Gamma^2));%probablility func
    psi=(1/(2*pi*alpha^2))^(1/4)*(1/sqrt(gamma)).*exp(1i*po.*(x-(po*t/(2*m)))./hbar).*exp((-(x-(po*t/

    %plot everythinggggggg
    plot(x,real(psi))
    hold on
    plot(x,sqrt(p))
    ylim([-3.5E7 3.5E7])
    xlim([-10E-15 10E-15])
    title('Wave Function and Packet Over X-axis')
    hold off

    drawnow

    pause(.001); %wait before incrementing again

end

stopbutton.Value=0;

pausebutton.Visible='off';
tlabel.Visible='off';
deltaxlabel.Visible='off';
deltaplabel.Visible='off';
bothlabel.Visible='off';

stopped=get(stopbutton, 'Value');

tarray=(-.1E-9:.002E-9:2.1E-9);

 while stopped==0

    stopped=get(stopbutton, 'Value');%check if stopped

    beta=get(betaslid,'Value'); %get beta value
    blabel.Text = "Beta: "+sprintf('%0.2f',beta); %update label

    po=get(poslid,'Value');%get po value
    polabel.Text = "Po: "+sprintf('%0.2f',po); %update label

    tau=m*hbar/(2*beta^2);

    Gammaarray=sqrt(1+(tarray.^2./tau^2));
    alpha=hbar/(2*beta);

    deltaxarray=alpha.*Gammaarray;
    deltaparray=beta.*ones(size(tarray));
    hbararray=hbar/2.*ones(size(tarray));

    plot(tarray, deltaxarray.*deltaparray)
    hold on
    plot(tarray,hbararray)
```

```
   xlim([-.1E-9 2.1E-9])
   ylim([0 4E-15])
   xlabel('Time (S)')
   title('Delta x*Delta p vs. time for Beta='+string(beta))
   hold off

   drawnow

   pause(.001);

end
```