Quantum Harmonic Oscillator Time Evolution

Timothy Holmes

May 18, 2018

The idea of this computational project is to simulate how a quantum harmonic oscillator evolves over time. In order to simulate this, there is no longer a need for just one ψ state. There are now multiple ψ state and in order for the user to easily test these state an input variable is defined. Depending of the users input the if statement will run the users input.

```
function [x, N, mass, E0] = compHomework2(x, N, mass, E0)

%% Choose state

stateType = input('Choose the state type; 1, 2, 3, or 4:\n'); %Asks user to select a state

load('C-2.mat')

if (stateType == 1) %Selects state from input above
    state = state1;
elseif (stateType == 2)
    state = state2;
elseif (stateType == 3)
    state = state3;
elseif (stateType == 4)
    state = state4;
elseif (stateType < 1 || stateType > 4)...
        , error('Invalid input');
end
```

Next, many of the inputs need to be defined. This includes the mass of a particle, Planck's constant in eV, ω at the ground state, and a value we define as ξ. Where ξ is given by,

$$\xi = \sqrt{\frac{m\omega}{\hbar}}x.$$

```
%% Input values

mass = mass/(3e8^2); %mass
hbar = 6.582*10^-16; %Planck's constant
omega = 2*E0/hbar; %E = hbarw(n +1/2), n = 0 -> w = 2*E/
    hbar
xi = sqrt(mass*omega/hbar).*x; %
```

The initial state selected by the user is not normalized. Therefore, to normalize the function the following equation is used to normalize analytically,

$$1 = \int_{-\infty}^{\infty} A^2 |\psi(x)|^2 dx.$$

Translating this into code we get the following below. The trapz command is a the trapezoidal numerical method that allow the computer to calculate the values we need.

```
%% Normalize

stateNorm = state/sqrt(trapz(x,conj(state).*state)); %
    normalizing wave function with integration
A = ((mass*omega/(pi*hbar))^(.25));
```

Preallocating in Matlab is not necessary but does allow for the code to run faster by presetting arrays.

```
%% preallocate

phi = zeros(length(x),N+1);
E = zeros(1,N+1);
c = zeros(1,N+1);
```

To calculate what we need for the quantum harmonic oscillator we need to define a few equations. The first equation defined is the energy eigenstate by,

$$\varphi_n(x) = \left(\frac{m\omega}{\pi\hbar}\right)^{\frac{1}{4}} \frac{1}{\sqrt{2^n n!}} H_n(\xi) e^{\frac{-\xi^2}{2}}.$$

Where $H_n(\xi)$ is the n^{th} hermite polynomial, this is defined later in the code and a recursive relation was used to calculate the values. The next equation defined is the energy eigenvalue,

$$E_n = \hbar\omega(n + 1/2).$$

Finally, for to find the time evolution we need to the c_n terms. This again can not be done analytically so a numerical approach is needed. To integrate the trpaz command is used and this maps to the numerical trapezoidal method. The equation used is,

$$c_n = \int_{-\infty}^{\infty} \varphi(x)^* \psi(x,0)dx.$$

```
%% Calculate

for n = 1:N+1

    phi(:,n) = A*(1./sqrt(2.^(n)*factorial(n))).*hermite(n,xi).*exp((-xi.^2)/2);
    E(:,n) = hbar*omega*(n-1+.5);
    c(n) = trapz(x,(conj(phi(:,n)).*stateNorm ));

end
```

Now that the values we need are calculated we can plot the initial state. Then in the for loop we calculate the time evolution,

$$\sum_{n=0}^{\infty} c_n e^{-iE_n t/\hbar} \varphi_n(x).$$

This equation will allow for our system to evolve through time. Every time the for loop is complete it updates the graph and we will see the system move though time.

```
%% Plot
fig = figure;
hold on
plotReal = plot(x,real(stateNorm), 'linewidth', 2); %real number psi part of the plot
plotImag = plot(x,imag(stateNorm), 'linewidth', 2); %imag number psi part of the plot
plotAbs = plot(x,abs(stateNorm), 'linewidth', 2); %abs value of psi part of the plot
legend('real','imaginary','Absolute Value') %labels
xlabel('x (nm)')
ylabel('\bf{\psi(t)}')
ylim([-1*10^5 1*10^5]);

video = VideoWriter('TimeEvolution.avi'); %starts writting frames
open(video);
```

```
%% Time Evolution, eigenstates, c terms, and updates plot

```

```
tic %start timing for loop
dt = 1; %time step
timeTotal = 1*10^3; %total time
count = 0;

for j = 1:dt:timeTotal

    time = j*10^-18;
    psi = zeros(size(state));

    for k = 1:N

    psi = psi + c(k).*phi(:,k)*exp(-1i*E(k)*time/hbar);

    end

    count = count + 1; %checking iteration number

    title(sprintf('Time Evolution Time = %g (Seconds)', toc))

    set(plotReal, 'YData', real(psi)) %update plot
    set(plotImag, 'YData', imag(psi))
    set(plotAbs, 'YData', abs(psi))

    currentFrame = getframe(gcf); %grabs frames from iteration
    writeVideo(video,currentFrame); %writes frames out to .avi file

    drawnow %draws updated plot
    pause(0.005) %waits for next iteration

end

%     stop = input('Do you want to continue, Y/N [Y]:','s');
%     if stop == 'N'
%           break
%     end

toc %ends for loop timing

fprintf('Count %f: \n',count) %prints number of for loop iterations

```

```
close(fig);
close(video);


end
```

The code for the recursive relation that computes the hermite polynomials.

```
function f = hermite(N, xi)

herm = zeros(length(xi), 1); %preallocate
herm(:,1) = ones(length(xi), 1); %H_0(xi)
herm(:,2) = 2*xi; %H_1(xi)

if (N < 0); error('Invalid input'); end;
if (N == 0); f = herm(:,1); return; end; %If N = 0 f = 1
if (N == 1); f = herm(:,2); return; end; %If N = 0 f = 2xi
if (N > 1) %If N > 1 f = H_n+1(xi) = 2(xi)H_n(xi) - 2nH_n-a(xi)

a = 1; b = 2; %setting first two Hermite polynomials

  for n = 2:N

    newHerm = 2*xi.*herm(:,b)-2*(n-1)*herm(:,a); %H_n+1(xi) = 2(xi)H_n(xi) - 2nH_n-a(xi)
    tem = a; a = b; b = tem; %Change variables
    herm(:,b) = newHerm;

  end

  f = newHerm;

end

end
```

The full matlab code.

```
function [x, N, mass, E0] = compHomework2(x, N, mass, E0)

%% Choose state

stateType = input('Choose the state type; 1, 2, 3, or 4: \n'); %Asks user to select a state

```

```matlab
load('data.mat')

%data = load('data.mat');
%csvwrite('data.csv',data);


if (stateType == 1) %Selects state from input above
    state = state1;
elseif (stateType == 2)
    state = state2;
elseif (stateType == 3)
    state = state3;
elseif (stateType == 4)
    state = state4;
elseif (stateType < 1 || stateType > 4)...
        , error('Invalid input');
end

%% Input values

mass =  mass/(3e8^2); %mass
hbar = 6.582*10^-16; %Planck's constant
omega = 2*E0/hbar; %E = hbarw(n +1/2), n = 0 -> w = 2*E/hbar
xi = sqrt(mass*omega/hbar).*x; %
size(xi)
%% Normalize

A = 1/sqrt(trapz(x,conj(state).*state)) %normalizing wave function with integration
stateNorm = A .* state;

A = ((mass*omega/(pi*hbar))^(.25));

size(stateNorm)
%fprintf('phi %.100f: \n',stateNorm)
%% preallocate
phi = zeros(length(x),N+1);
E = zeros(1,N+1);
c = zeros(1,N+1);

v = 1/2*mass*omega^2.*x.^2;
size(v)

%% Calculate

```

```
for n = 1:N+1
    a(:,n) = hermite(n,xi);
    phi(:,n) = A*(1./sqrt(2.^(n)*factorial(n))).*hermite(n,xi).*exp((-xi.^2)/2); %eigenstate
    E(:,n) = hbar*omega*(n-1+.5); %eigenvalue
    c(n) = trapz(x,(conj(phi(:,n)).*stateNorm )); %cn term for time evolution

end
fprintf('herm %.10f: \n',a(1,3))
%fprintf('a %.100f: \n',a)
%fprintf('phi %.100f: \n',phi(:,2))
%fprintf('En %.10f: \n',E)
fprintf('Cn %.10f: \n',c(:,5))
%size(phi)
%size(c)
%size(hermite(n,xi))


%% Plot
fig = figure;
hold on
plotReal = plot(x,real(stateNorm), 'linewidth', 2); %real number psi part of the plot
plotImag = plot(x,imag(stateNorm), 'linewidth', 2); %imag number psi part of the plot
plotAbs = plot(x,abs(stateNorm), 'linewidth', 2); %abs value of psi part of the plot
plotPar = plot(x, v-1*10^5);
legend('real','imaginary','Absolute Value') %labels
xlabel('x (nm)')
ylabel('\bf{\psi(t)}')
ylim([-1*10^5 1*10^5]);

video = VideoWriter('TimeEvolution.avi'); %starts writting frames
open(video);


%% Time Evolution, eigenstates, c terms, and updates plot

tic %start timing for loop
dt = 1; %time step
timeTotal = (1*10^3)/2; %total time
count = 0;

```

```
for j = 1:dt:timeTotal
```