Richard Magnotti

Undergraduate Research 2015

Introduction:

Within a magnet, there exists 'n' atoms, each with a net magnetic moment. The magnetic moment is generated by the valence electrons of the atom. This is a quantum phenomenon. A classical description says that the magnetic moment is caused by the orbital motion of electrons. In this research, we used the fourth order Runge-Kutta algorithm to first solve a practice problem in which we found the numerical solution of the second order differential equation for an anharmonic oscillator. We then constructed a two-spin lattice in which both of them varied with dependence on one another. Finally, we constructed a three spin lattice of three magnetic spins in which nearest neighbor interactions were considered.

Methods:

The Anharmonic Oscillator Practice Problem

$$F_{Harmonic} = -kx \qquad (1)$$

$$F_{Anharmonic} = -kx - bx^3 \qquad (2)$$

$$V(t) = \lim_{\Delta t \to 0} \frac{x(t+\Delta t)}{\Delta t} \qquad (3)$$

In the practice problem, we used the fourth order Runge-Kutta[1] method of integration to numerically solve the differential equation representing an anharmonic oscillator. Being an anharmonic oscillator, we know that the system displaced from equilibrium responds with a

1: "Runge-Kutta Method." -- *from Wolfram MathWorld*. Wolfram Alpha, n.d. Web. 06 Nov. 2015

restoring force that is not proportional to its displacement (see equation (2)). Figure (1) is a graph of the behavior of the oscillator as a function of “time”, where one step corresponds to one update step of the set of differential equations.

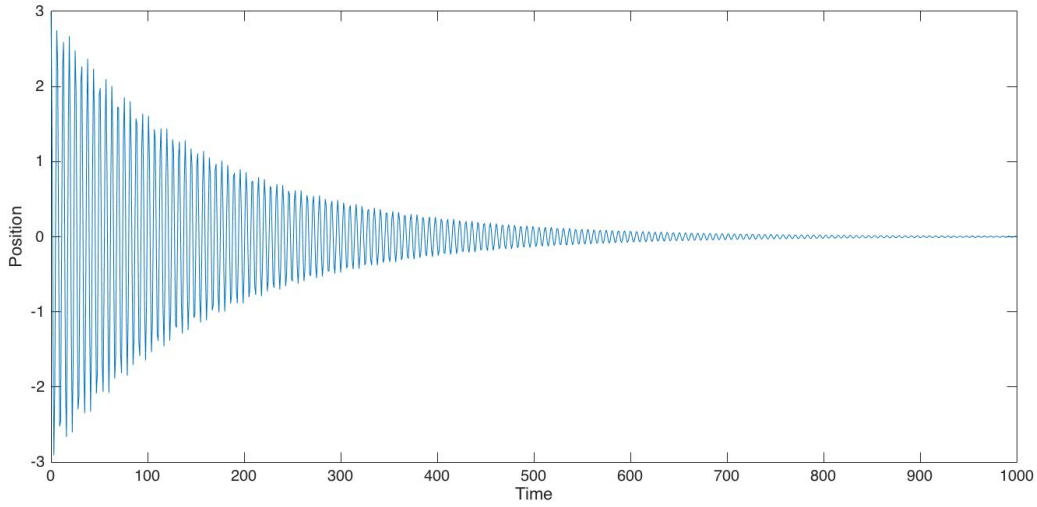

Figure (1)

Anharmonic oscillator with terminal time t=1000, number of steps m=1500, step size h=.6667, anharmonic constant c=.01

Figure (1) demonstrates that as time progresses, the displacement of the oscillator decreases. This is to be expected, as we know the restoring force should vary since it is not proportional to the amount of displacement. However, Figure (1) is a somewhat poor representation of the position as a function of time. This is simply because of numerical limitations. This limitation can be seen directly from the definition of velocity as a derivative of time (equation (3)). The time step Δt can only be numerically programmed to be finitely small. However, in Equation (3), by definition of the derivative, Δt approaches zero. The smaller the time step, the more

accurate the estimate of the new position and velocity of the oscillator. Therefore, the results in Figure (2) are when taking 2000 steps in time rather than 1000 (with the same terminal time of 1000 seconds as Figure (1)). Keeping the terminal time the same, but changing the amount of steps taken, decreases the number of time between each step, thus delivering a smaller Δt. What we see is much less decayed values for position at later t. In other words, the results are the same, but the clarity of the reuslts is better.

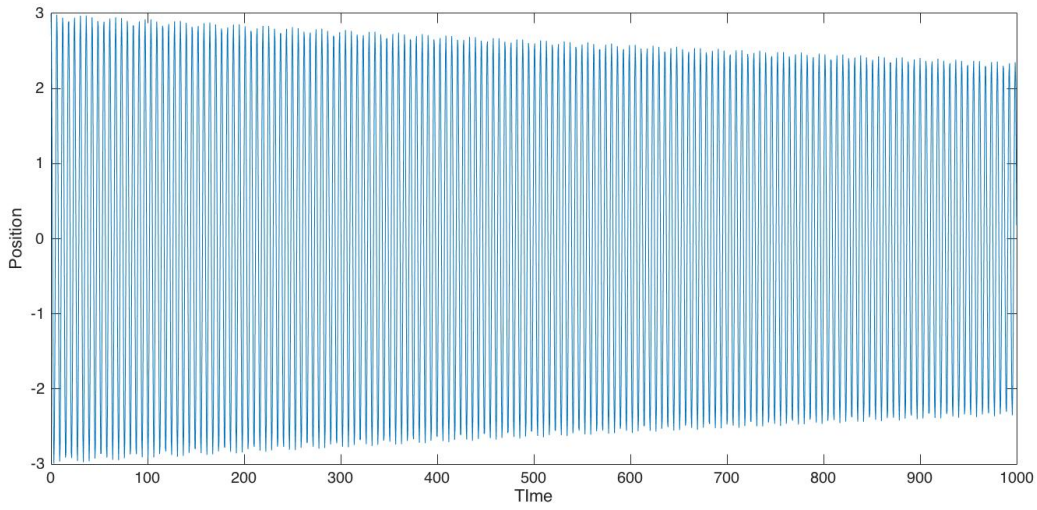

Figure (2)

Anharmonic oscillator with terminal time t=1000, number of steps m=2000, step size h=.5, anharmonic constant c=.01

In the ordinary differential equation (ODE) representing the anharmonic oscillator (equation (2)), there is a term that decides whether or not the oscillator will behave anharmonic or harmonic. Therefore, we set the 'anharmonic term', c=.4 (as opposed to the default c=.01 in the previous two Figures). This altered c value turns off the anharmonic property of the oscillator.

As is demonstrated by Figure (3), now the system follows Hooke's law. Thus, the restoring force of the oscillator is proportional to the displacement, i.e. no change in position as a function of time.

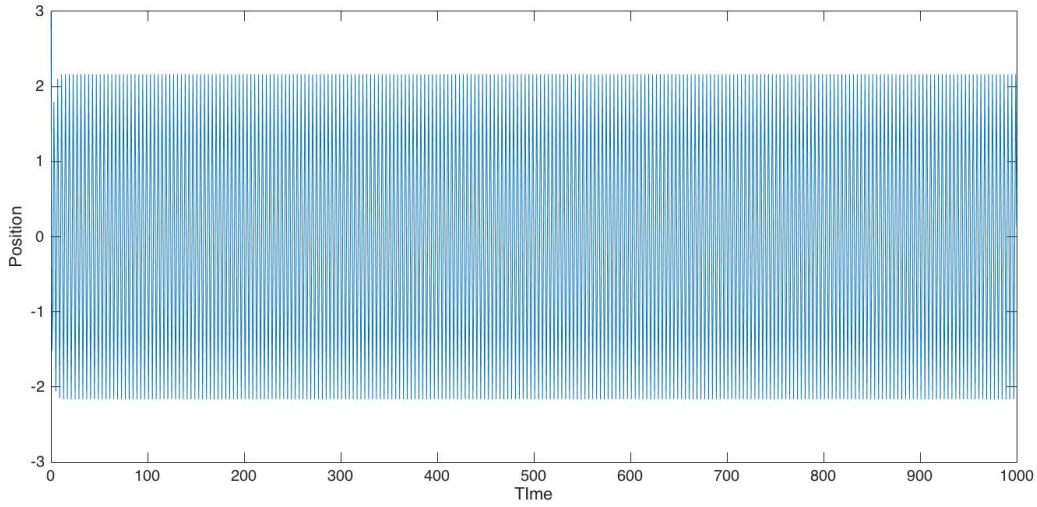

Figure (3)

Harmonic oscillator with terminal time t=1000, number of steps m=1500, step size h=.5, anharmonic constant c=.4

Spin Dynamics

$$H = -J\boldsymbol{S}_i \bullet \boldsymbol{S}_j \qquad (4)$$

$$d\boldsymbol{S}_i/dt = -\boldsymbol{S}_i \times \boldsymbol{h}_{eff,I} \qquad (5)$$

$$\boldsymbol{h}_{eff,i} = -J\boldsymbol{S}_j \qquad (6)$$

Two-Spin Lattice

The purpose of the anharmonic oscillator practice problem was to develop a sense of how to use the Runge-Kutta method and exactly what is happening when using it. Therefore, when we set up the two-spin lattice, we employed the same method of numerical analysis to solve the

differential equation (see equation (2)). However, we now have to solve two coupled ordinary differential equations that depend on each other as well as time. Figure (4) shows the energy of the system of spins as a function of time.

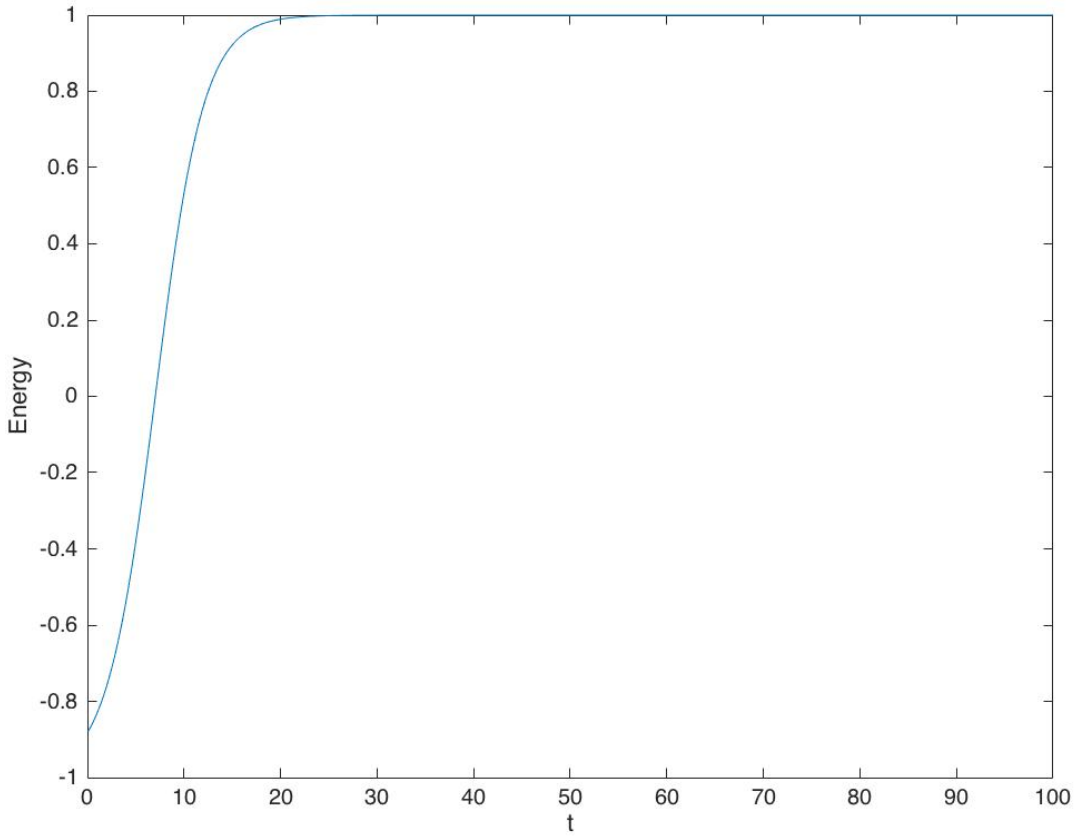

Figure (4)

Two-spin lattice with m=1000 steps, time b=100

One thing to consider about this problem is that we cannot accurately represent the system numerically because of limitations described previously. Therefore, the plots in Figure (4)-Figure (9) are showing us the values as close to the real system as possible. Therefore, what we see is

not necessarily what we would expect if the system was real. We would expect the total energy to be slightly decreasing until the spins aligned ferromagnetically. However, what we see is a very slight increase in the energy. This is because the Runge-Kutta method can only take steps in a certain step size. Therefore, it overestimates values of the spin, causing slight error. Numerically, this system is showing us that the Energy of the system will increase in very small increments until the spin processes to its highest energy state. Much like the anharmonic oscillator practice problem, the smaller the step size taken, the smaller the error. We find that as we take smaller steps, the energy as a function of time will increase less and less, which makes it a more accurate representation of a real system.

In Figure (4), we set the number of steps through which the Runge-Kutta will process equal to m=1000 steps, as well as setting the terminal time b=100. As we changed the number of steps to m=5000, and the terminal time b=100, we get a more accurate plot of what is actually happening. This is because we have increased the number of calculations exponentially, while simultaneously decreasing the time interval which the steps are taken. We can see this clearly in Figure (5) by the now slight-curve of the graph, as opposed to Figure (4), which is a much steeper curve. The time interval is greater, however, the energy doesn't change very much. This is because the plot is a much better approximation than Figure (4).

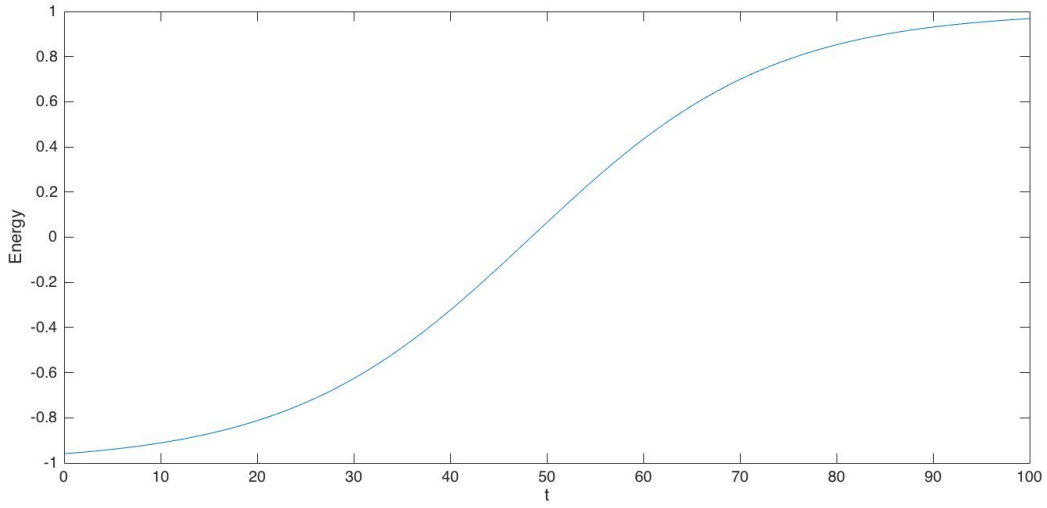

Figure (5)

Two-spin lattice with m=5000 steps, time b=100

Furthermore, we can see the improvements even more so when we increase the number of steps to m=10000 (and the terminal time to b=100) such as in Figure (6).

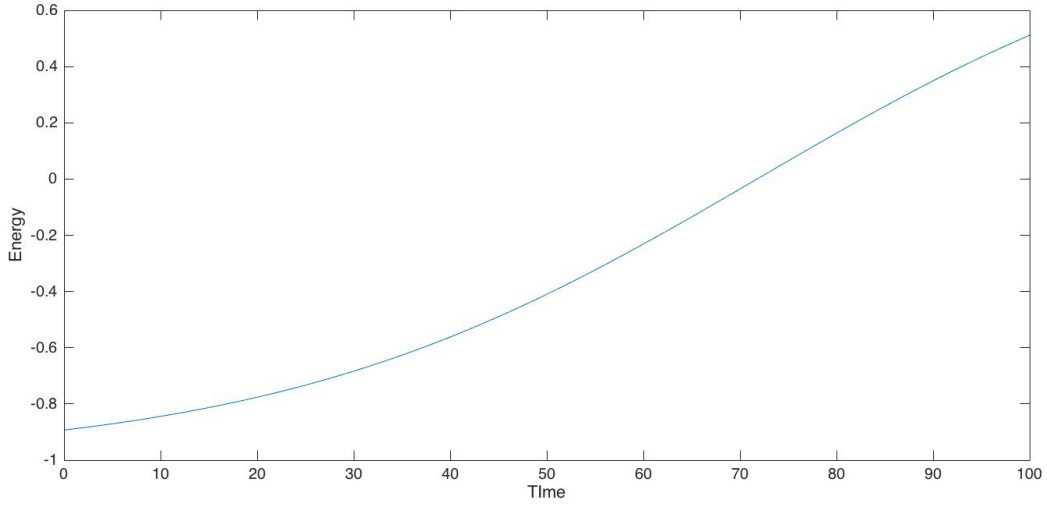

Figure (6)

Two-spin lattice with m=10000 steps, time b=100

Three-Spin Lattice

The data gathered by the two-spin lattice is still applicable. In fact, we would expect very little change. The only difference is now we have three spins represented by ODE's that are all coupled with each other and have time dependence.

Figure (7) shows the energy with respect to time of the three-spin lattice over time b=100, taking 5,000 steps via the Runge-Kutta. This step amount is clearly not as accurate as it could be. This is why with Figure (8) and Figure (9) we increase the number of steps to 50,000 and 500,000 steps respectively.

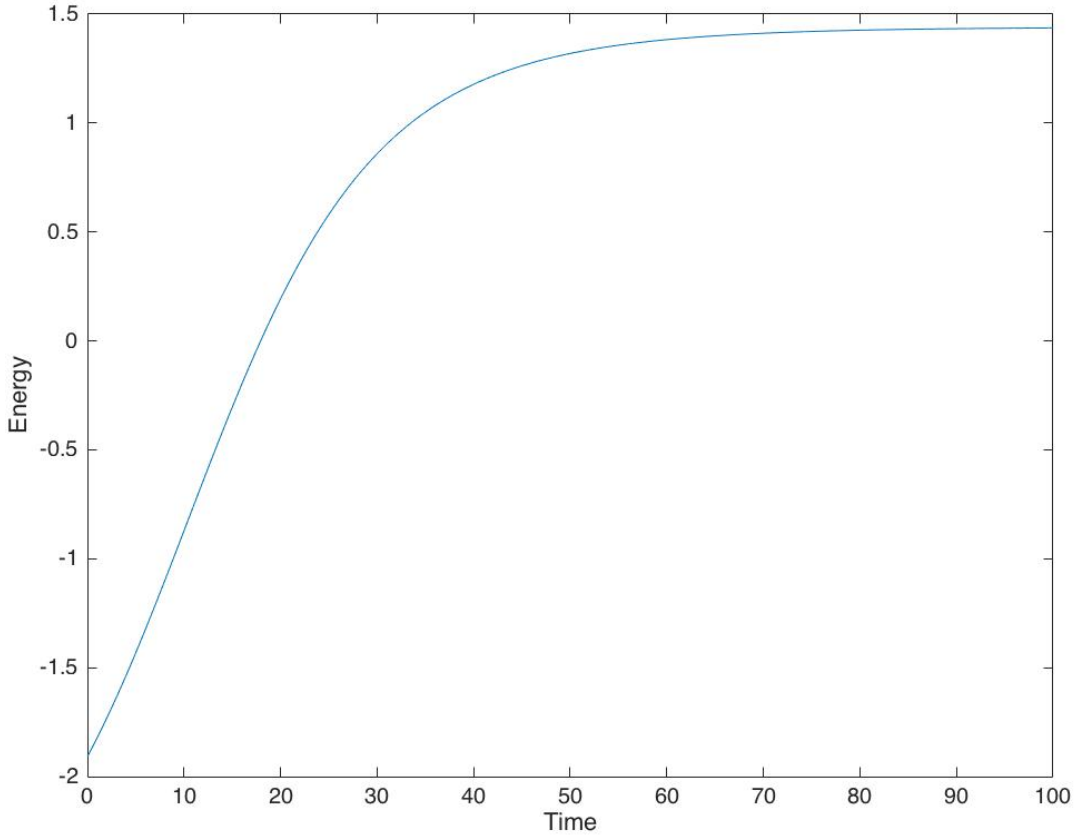

Figure (7)

Three-spin lattice with m=5000 steps, time b=100

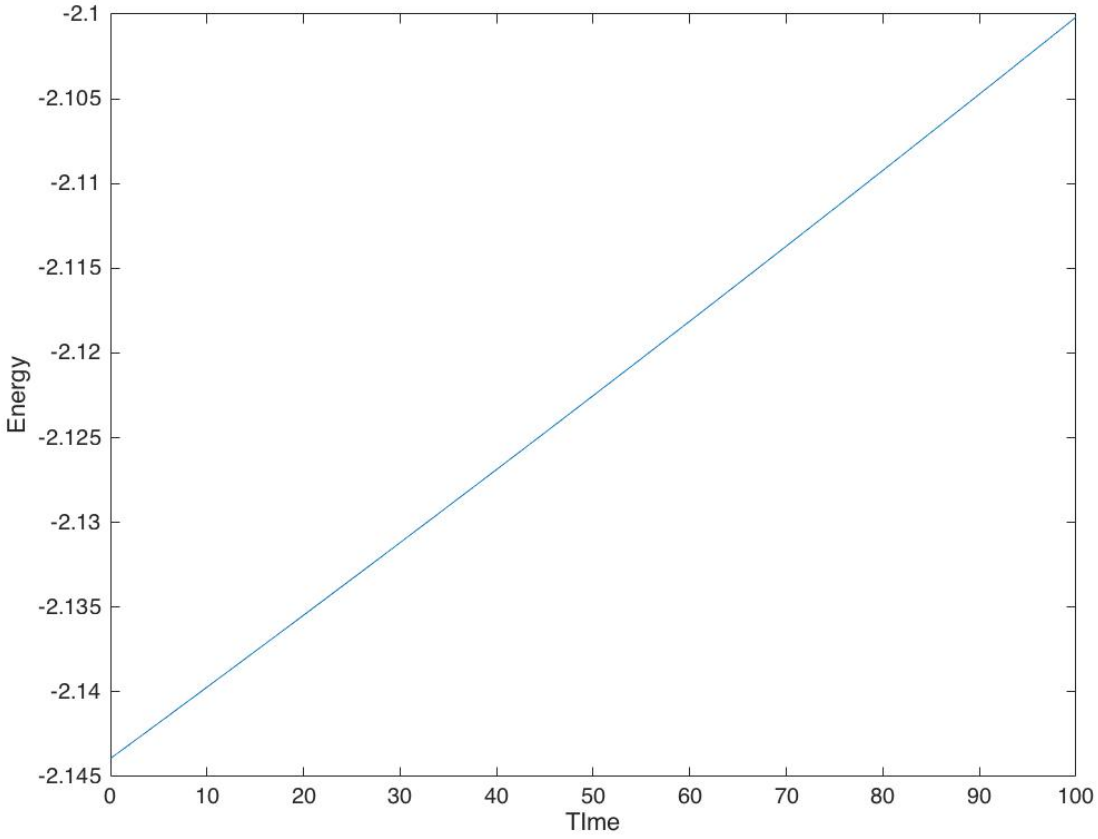

Figure (8)
Three-spin lattice with m=50000 steps, time b=100

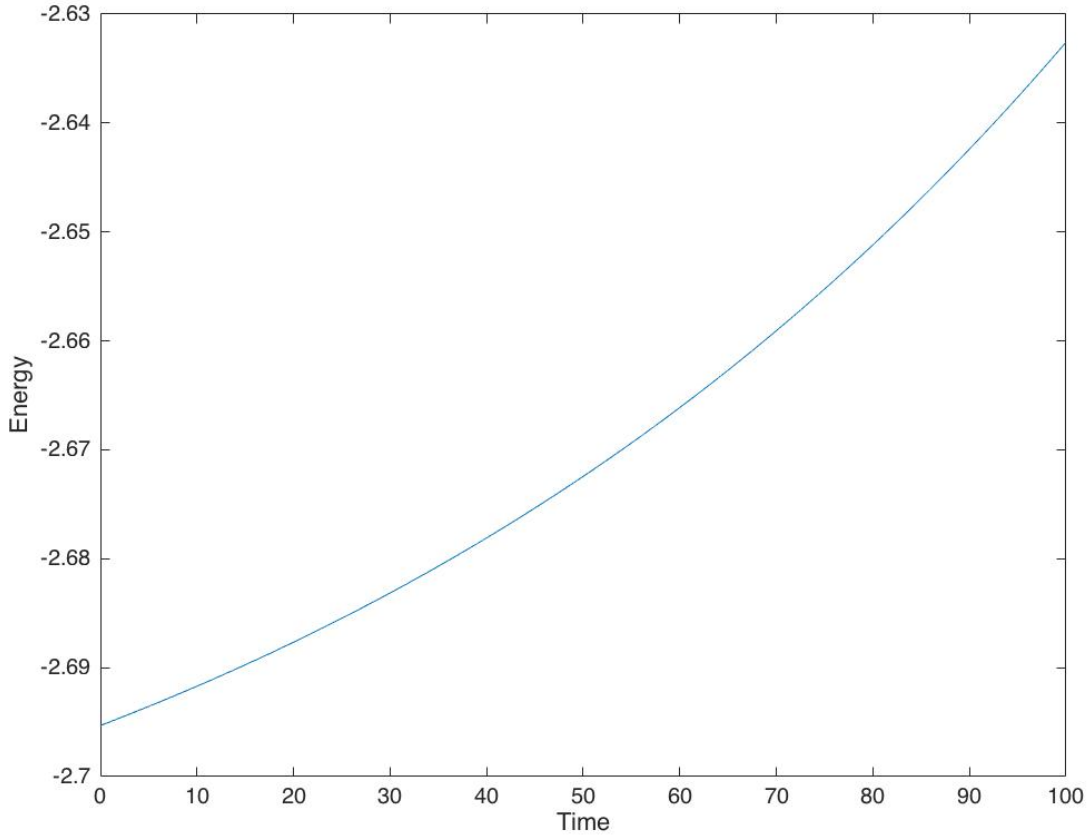

Figure (9)

Three-spin lattice with m=500000 steps, time b=100

Discussion:

The reason we used the Fourth Order Runge-Kutta (rk4) method to solve the discussed three examples, is because we cannot solve the differential equations in the tradition manner (pen and paper). The rk4 method is a computer algorithm to approximate the integral of differential equations by stepping through each calculation four times (hence, “Fourth Order”), and getting a more accurate approximation each time. Then we average the approximations. The rk4 method is essentially the backbone of each example problem. We found that in all three examples (anharmonic example practice problem, two-spin lattice, three-spin lattice), the

smaller the terminal time and the larger the amount of steps worked through, the smaller the step size. Recall that we want the step size as close to zero as possible for the most accurate approximation. The more accurate approximations are shown as in each Figure. E.g. Figure (2) shows a more accurate approximation of Figure (1), and Figure (6) shows a better approximation of figure (5) and Figure (4).

Appendix

```
%Anharmonic practice problem
%following eq.'s are how second order
%ODE breaks down into two, coupled first order
%ODE's

k = 1.0; %spring constant

c = 0; %some constant

f = @(t,y,z) z;

g = @(t,y,z) -k*y - c*y^3;


a=0; %intiial time

b=1000; %terminal time

ya=3; %initial position

za=0; %initial velocity

m=1500; %number of steps


[T,Y,Z] = rk4(f,g,a,b,ya,za,m); %returns T,Y,Z vectors
                                %with each respective t,y,z values
                                %at each time step


%plot(T,Y)
```

Published with MATLAB® R2015b

```
%Subroutine for rkTestAnharm
%This is the Runge-Kutta Stepper
function[T,Y,Z] = rk4(f,g,a,b,ya,za,m)


h = (b-a)/m; %step size


T = zeros(1,m+1); %time vector (stores all t values)

Y = zeros(1,m+1); %position vector (stores all position y values)

Z = zeros(1,m+1); %velcotiy vector (stores all velocity v values)


T(1) = a; %initial time

Y(1) = ya; %intial position

Z(1) = za; %initial velocity


for j=1:m, %iterates one time step (entails 4 approximations)

    tj = T(j); %increase j value per iteration of loop

    yj = Y(j);

    zj = Z(j);


    k1 = h*feval(f,tj,yj,zj); %h(run)*(slope of position (y) tangent
 line) = approx. rise from last iteration

    l1 = h*feval(g,tj,yj,zj); %h(run)*(slope of velocity (z) tangent
 line) = approx. rise from last iteration

    k2 = h*feval(f,tj+(h/2),yj+(k1/2), zj+(l1/2)); % "" but using
 updated values of y and z (half steps)

    l2 = h*feval(g,tj+(h/2),yj+(k1/2),zj+(l1/2)); % "" (half steps)

    k3 = h*feval(f,tj+(h/2),yj+(k2/2),zj+(l2/2)); % "" (half steps)

    l3 = h*feval(g,tj+(h/2),yj+(k2/2),zj+(l2/2)); % "" (half steps)
```

```
    k4 = h*feval(f,tj+h,yj+k3,zj+l3); % ""

    l4 = h*feval(g,tj+h,yj+k3,zj+l3); % ""


    Y(j+1) = yj + (k1 + 2*k2 + 2*k3 + k4)/6; %computes new y value by
 averaging all 4 approximations of y

    Z(j+1) = zj + (l1 + 2*l2 + 2*l3 + l4)/6; %"" z

    T(j+1) = a + h*j; %updates by taking another step

end
```

```
Not enough input arguments.

Error in rk4 (line 7)
h = (b-a)/m; %step size
```

Published with MATLAB® R2015b

```
%Code for two-spin lattice
%Makes call to subroutine TSLrk4

%defining variables
spin = zeros (3,2);
J = 1; %interaction parameter
%M; %magnetization

m = 10000; %num steps
a = 0; %intial time
b = 100.0; %terminal time (seconds)

%to fill the matrix elements (random num between 0 and 1)
for i = 1:3,
    for j = 1:2,
        spin(i,j)=rand;
    end
end

%function to normalize the matrix columns, so length of vector adds to
 1
spin=normc(spin);

s1a = spin(:,1); %setting initial particle1 spin = column 1 of spin
 matrix
s2a = spin(:,2);

heff1 = J*s2a;
heff2 = J*s1a;

f = @(t,heff1,s1a) cross(heff1,s1a); %spin1 function
g = @(t,heff2,s2a) cross(heff2,s2a);

%dSi/dt = -Si x h(eff)i,j
[T,S1,S2,E,HEFFs1,HEFFs2] = TSLrk4(f,g,a,b,s1a,s2a,m,J, heff1, heff2);

%plot(T,E)
```

Published with MATLAB® R2015b

```
%Subroutine for TwoSpinLattice Code
%This is the Runge-Kutta Stepper
function[T,S1,S2,E,HEFFs1,HEFFs2] = TSLrk4(f,g,a,b,s1a,s2a,m,J,
 heffs1, heffs2)


h = (b-a)/m; %step size


T = zeros(1,m+1); %time vector (stores all t values)...(1xm+1) matrix

S1 = zeros(3,m+1); %Spin1 vector (stores all Spin1 values)
S2 = zeros(3,m+1); %Spin2 vector

S1(:,1) = s1a; %intial spin1 (particle1) value
S2(:,1) = s2a;

HEFFs1 = zeros(3,m+1); %h effective vector
HEFFs2 = zeros(3,m+1);

HEFFs1(:,1) = heffs1; %initial effective h of particle 1
HEFFs2(:,1) = heffs2;


T(1) = a; %initial time

E = zeros(1,m+1); %to store all Hamiltonian values

sDot = dot(s1a,s2a);

E(:,1) = -J*sDot; %Initial energy

for j=1:m, %iterates one time step (entails 4 approximations)

           %like for loop in Java (for(j = 1; j = m; j++))



    tj = T(j); %increase j value per iteration of loop

    s1j = S1(:,j);

    s2j = S2(:,j);

    heff1 = HEFFs1(:,j);

    heff2 = HEFFs2(:,j);

    k1 = h*feval(f,tj,heff1,s1j);
    %Original: k1 = h*feval(f,tj,s1j,s2j); %h(run)*(slope of spin1
 (s1) tangent line) = approx. rise from last iteration
```

```
    l1 = h*feval(g,tj,heff2,s2j);
    %Original: l1 = h*feval(g,tj,s1j,s2j); %h(run)*(slope of spin2
 (s2) tangent line) = approx. rise from last iteration

    k2 = h*feval(f,tj+(h/2),heff1+(l1/2),s1j+(k1/2)); % "" but using
 updated values of s1 and 2 (half steps)

    l2 = h*feval(g,tj+(h/2),heff2+(k1/2),s2j+(l1/2)); % "" (half
 steps)

    k3 = h*feval(f,tj+(h/2),heff1+(l2/2),s1j+(k2/2)); % "" (half
 steps)

    l3 = h*feval(g,tj+(h/2),heff2+(k2/2),s2j+(l2/2)); % "" (half
 steps)

    k4 = h*feval(f,tj+h,heff1+l3,s1j+k3); % ""

    l4 = h*feval(g,tj+h,heff2+k3,s2j+l3); % ""


    %takes old value of S1 and adds new value to it to update it
    s1j = s1j + (k1 + 2*k2 + 2*k3 + k4)/6; %computes new y value by
 averaging all 4 approximations of y
    s1j = normc(s1j);
    S1(:,j+1) = s1j;

    s2j = s2j + (l1 + 2*l2 + 2*l3 + l4)/6; %"" z
    s2j = normc(s2j);
    S2(:,j+1) = s2j;

    T(j+1) = a + h*j; %updates by taking another step

    HEFFs1(:,j+1) = -J*s2j;
    HEFFs2(:,j+1) = -J*s1j;

    sDot = dot(s1j,s2j);

    E(:,j+1) = -J*sDot; %Energy Hamiltonian
end
```

```
Not enough input arguments.

Error in TSLrk4 (line 6)
h = (b-a)/m; %step size
```

Published with MATLAB® R2015b

```
%Code for three-spin lattice
%Makes call to subroutine ThSLrk4

%defining variables
spin = zeros(3,3);
J = 1; %interaction parameter
%M; %magnetization

m = 5000; %num steps
a = 0; %intial time
b = 100.0; %terminal time (seconds)

%to fill the matrix elements (random num between 0 and 1)
for i = 1:3,
    for j = 1:3,
        spin(i,j)=rand;
    end
end

%function to normalize the matrix columns, so length of vector adds to
 1
spin=normc(spin);

s1a = spin(:,1); %setting initial particle1 spin = column 1 of spin
 matrix
s2a = spin(:,2);
s3a = spin(:,3);

%inc. number of heff by one
heff1 = -J*(s2a + s3a);
heff2 = -J*(s1a + s3a);
heff3 = -J*(s1a + s2a);

f1 = @(t,heff1,s1a) cross(heff1,s1a); %spin1 function
f2 = @(t,heff2,s2a) cross(heff2,s2a);
f3 = @(t,heff3,s3a) cross(heff3,s3a);

%dSi/dt = -Si x h(eff)i,j
[T,S1,S2,S3,E,HEFFs1,HEFFs2,HEFFs3] =
 ThSLrk4(f1,f2,f3,a,b,s1a,s2a,s3a,m,J,heff1,heff2,heff3);

%plot(T,E)
```

Published with MATLAB® R2015b

```
%Subroutine for ThreeSpinLattice Code
%This is the Runge-Kutta Stepper
function[T,S1,S2,S3,E,HEFFs1,HEFFs2,HEFFs3] =
 ThSLrk4(f1,f2,f3,a,b,s1a,s2a,s3a,m,J,heff1,heff2,heff3)

h = (b-a)/m; %step size


T = zeros(1,m+1); %time vector (stores all t values)...(1xm+1) matrix

S1 = zeros(3,m+1); %Spin1 vector (stores all Spin1 values)
S2 = zeros(3,m+1); %Spin2 vector
S3 = zeros(3,m+1);

S1(:,1) = s1a; %intial spin1 (particle1) value
S2(:,1) = s2a;
S3(:,1) = s3a;

HEFFs1 = zeros(3,m+1); %h effective vector
HEFFs2 = zeros(3,m+1);
HEFFs3 = zeros(3,m+1);

HEFFs1(:,1) = heff1; %initial effective h of particle 1
HEFFs2(:,1) = heff2;
HEFFs3(:,1) = heff3;

T(1) = a; %initial time

E = zeros(1,m+1); %to store all Hamiltonian values

sDot1 = dot(s2a,s3a);
sDot2 = dot(s1a,s3a);
sDot3 = dot(s1a,s2a);

E(:,1) = -J*(sDot1+sDot2+sDot3); %Initial energy

for j=1:m, %iterates one time step (entails 4 approximations)

           %like for loop in Java (for(j = 1; j = m; j++))


    tj = T(j); %increase j value per iteration of loop

    s1j = S1(:,j);

    s2j = S2(:,j);

    s3j = S3(:,j);


    heff1 = HEFFs1(:,j);
```

```
   heff2 = HEFFs2(:,j);

   heff3 = HEFFs3(:,j);


   %start rk4 portion
   k1 = h*feval(f1,tj,heff1,s1j); %h(run)*(slope of spin1 (s1)
tangent line) = approx. rise from last iteration
   l1 = h*feval(f2,tj,heff2,s2j); %h(run)*(slope of spin2 (s2)
tangent line) = approx. rise from last iteration
   m1 = h*feval(f3,tj,heff3,s3j);

   k2 = h*feval(f1,tj+(h/2),heff1+(l1/2),s1j+(k1/2)); % "" but using
updated values of s1 and 2 (half steps)
   l2 = h*feval(f2,tj+(h/2),heff2+(k1/2),s2j+(l1/2)); % "" (half
steps)
   m2 = h*feval(f3,tj+(h/2),heff3+(m1/2),s3j+(m1/2));

   k3 = h*feval(f1,tj+(h/2),heff1+(l2/2),s1j+(k2/2)); % "" (half
steps)
   l3 = h*feval(f2,tj+(h/2),heff2+(k2/2),s2j+(l2/2)); % "" (half
steps)
   m3 = h*feval(f3,tj+(h/2),heff3+(m2/2),s3j+(m2/2));

   k4 = h*feval(f1,tj+h,heff1+l3,s1j+k3); % ""
   l4 = h*feval(f2,tj+h,heff2+k3,s2j+l3); % ""
   m4 = h*feval(f3,tj+h,heff3+m3,s3j+m3);


   %takes old value of S1 and adds new value to it to update it
   s1j = s1j + (k1 + 2*k2 + 2*k3 + k4)/6; %computes new y value by
averaging all 4 approximations of y
   s1j = normc(s1j);
   S1(:,j+1) = s1j;

   s2j = s2j + (l1 + 2*l2 + 2*l3 + l4)/6; %"" z
   s2j = normc(s2j);
   S2(:,j+1) = s2j;

   s3j = s3j + (m1 + 2*m2 + 2*m3 + m4)/6; %"" z
   s3j = normc(s3j);
   S3(:,j+1) = s3j;

   T(j+1) = a + h*j; %Updates by taking another step

   HEFFs1(:,j+1) = -J*(s2j+s3j);
   HEFFs2(:,j+1) = -J*(s1j+s3j);
   HEFFs3(:,j+1) = -J*(s1j+s2j);

   sDot1 = dot(s2j,s3j);
   sDot2 = dot(s1j,s3j);
   sDot3 = dot(s1j,s2j);
```

```
    E(:,j+1) = -J*(sDot1 + sDot2 + sDot3); %Energy Hamiltonian
end
```

```
Not enough input arguments.

Error in ThSLrk4 (line 5)
h = (b-a)/m; %step size
```

Published with MATLAB® R2015b

```
%Code for all-spin lattice
%Makes call to subroutine ASLrk4

%this code is designed to accept 'n' amount of spins (as opposed to 2
 or 3
%and numerically integrate via Runge-Kutta
%However, we were only able to start it int the time allotted

%defining variables
s = 50; %number of spins

spin = zeros (3,s);
J = 1; %interaction parameter
%M; %magnetization

spinMat = zeros(3,s); %matrix of spin initial values
heffMat = zeros(3,s); %matrix of feff initial values
fMat = zeros(1,s); %matrix of funcions

m = 5000; %num steps
a = 0; %intial time
b = 50.0; %terminal time (seconds)

%to fill the matrix elements (random num between 0 and 1)
for i = 1:3,
    for j = 1:s,
        spin(i,j)=rand;
    end
end

%function to normalize the matrix columns, so length of vector adds to
 1
spin=normc(spin);

%matrix of initial values
for j=1:s,

    spinMat(:,j) = spin(:,j); %setting initial particle1 spin = column
 1 of spin matrix

end

%matrix of intial heff values
for j=1:s,

    heffMat(:,j) = -J*(spinMat(:,j-1) + spinMat(:,j+1));

end

for j=1:s,
```

```
    fMat(1,j) = @(t,heffMat(:,j),spinMat(:,j))
 cross(heffMat(:,j),spinMat(:,j)); %spin1 function

end

%left off here -- replace initial values with initial values from
 matrices
%dSi/dt = -Si x h(eff)i,j
[T,S,E,HEFF] = ASLrk4(fMat,a,b,spinMat,m,J,heffMat,s);

%plot(T,E)
```

```
Error using dbstatus
Error: File: /Users/Masane 1/Library/Mobile Documents/
com~apple~CloudDocs/SCSU Fall 2015/Research/AllSpinLattice/
AllSpinLattice.m Line: 49 Column: 28
Unbalanced or unexpected parenthesis or bracket.
```

Published with MATLAB® R2015b

```
%Subroutine for AllSpinLattice Code
%This is the Runge-Kutta Stepper
function[T,S,E,HEFF] = TSLrk4(fMat,a,b,spinMat,m,J,heffMat,s)

% Basic Variables

h = (b-a)/m; %step size

T = zeros(1,m+1); %time vector (stores all t values)...(1xm+1) matrix

T(1) = a; %initial time

% 3D Spin matrix (spin xyz values -> rows, spin# -> columns, spin at t
 -> page)
% pages change w/ time steps

S = zeros(3,m+1,s);

for j = 1:s

    S(:,j,1) = spinMat(:,j); %storing intial spin values

end

% 3D heff matrix (heff xyz values -> rows, heff# -> columns, heff @ t
 -> page)
% pages change w/ time steps

HEFF = zeros(3,m+1,s);

for j = 1:s

    HEFF(:,j,1) = heffMat(:,j); %storing intial heff value per
 particle

end

% Vector with one row and m+1 columns to store the total energy of the
% system per time step

E = zeros(1,m+1); %to store all Hamiltonian values

% matrix to represent the dot products of each matrix with two nearest
% neighbors (spin dot prod. value -> row, spin nearest neighbor pair #
 ->
% column)

sDotMat = zeros(1,m+1);
sDotMat(1,1) = dot(s1a,s2a); %dot prod. of initial s values

E(:,1) = -J*sDot; %Initial energy
```

```
-------------------------------------------------------------------------
for j=1:m, %iterates one time step (entails 4 approximations)

           %like for loop in Java (for(j = 1; j = m; j++))



    tj = T(j); %increase j value per iteration of loop

    s1j = S1(:,j);
    s2j = S2(:,j);
    s3j = S3(:,j);

    f1 = fMat(1,j);
    f2

    heff1 = HEFFs1(:,j);
    heff2 = HEFFs2(:,j);
    heff3 = HEFFs3(:,j);


    %start rk4 portion
    k1 = h*feval(f1,tj,heff1,s1j); %h(run)*(slope of spin1 (s1)
 tangent line) = approx. rise from last iteration
    l1 = h*feval(f2,tj,heff2,s2j); %h(run)*(slope of spin2 (s2)
 tangent line) = approx. rise from last iteration
    m1 = h*feval(f3,tj,heff3,s3j);

    k2 = h*feval(f1,tj+(h/2),heff1+(l1/2),s1j+(k1/2)); % "" but using
 updated values of s1 and 2 (half steps)
    l2 = h*feval(f2,tj+(h/2),heff2+(k1/2),s2j+(l1/2)); % "" (half
 steps)
    m2 = h*feval(f3,tj+(h/2),heff3+(m1/2),s3j+(m1/2));

    k3 = h*feval(f1,tj+(h/2),heff1+(l2/2),s1j+(k2/2)); % "" (half
 steps)
    l3 = h*feval(f2,tj+(h/2),heff2+(k2/2),s2j+(l2/2)); % "" (half
 steps)
    m3 = h*feval(f3,tj+(h/2),heff3+(m2/2),s3j+(m2/2));

    k4 = h*feval(f1,tj+h,heff1+l3,s1j+k3); % ""
    l4 = h*feval(f2,tj+h,heff2+k3,s2j+l3); % ""
    m4 = h*feval(f3,tj+h,heff3+m3,s3j+m3);


    %takes old value of S1 and adds new value to it to update it
    s1j = s1j + (k1 + 2*k2 + 2*k3 + k4)/6; %computes new y value by
 averaging all 4 approximations of y
    s1j = normc(s1j);
    S1(:,j+1) = s1j;

    s2j = s2j + (l1 + 2*l2 + 2*l3 + l4)/6; %"" z
    s2j = normc(s2j);
    S2(:,j+1) = s2j;
```

```
    s3j = s3j + (m1 + 2*m2 + 2*m3 + m4)/6; %"" z
    s3j = normc(s3j);
    S3(:,j+1) = s3j;

    T(j+1) = a + h*j; %updates by taking another step

    HEFFs1(:,j+1) = -J*(s2j+s3j);
    HEFFs2(:,j+1) = -J*(s1j+s3j);
    HEFFs3(:,j+1) = -J*(s1j+s2j);

    sDot1 = dot(s2j,s3j);
    sDot2 = dot(s1j,s3j);
    sDot3 = dot(s1j,s2j);

    sDotTot

    E(:,j+1) = -J*(sDot1 + sDot2 + sDot3); %Hamiltonian
end
```

Error using dbstatus
Error: File: /Users/Masane 1/Library/Mobile Documents/com~apple~CloudDocs/SCSU Fall 2015/Research/AllSpinLattice/ASLrk4.m Line: 49 Column: 76
Expression or statement is incomplete or incorrect.

Published with MATLAB® R2015b