Penning Trap:

The Penning trap is an electromagnetic trap to confine charged particles in a small space. We would be starting out with a differential equation, for which we could uncouple them into a series of first order differential equations and solve it.

We first start with the simple Lorentz Force equation:

We note that is the velocity of the particle, is the charge of the particle, is the external magnetic field applied to the atom, and is the potential that is caused by the external electric field on the particle.

The final three variables that we would want to introduce are , the displacement of the particle with respect to a defined origin, , the acceleration of the particle, and , the mass of the particle.

Suppose that we have , and we have an electric quadrupole potential using cylindrical coordinates (which is symmetric on the direction):

Decouple into its three components in the direction: By noting that , and the mass of the electron never changes, and assuming non-relativistic conditions:

So essentially, we can obtain three sets of coupled differential equations, all of second order and thankfully homogeneous. By noting that , we have:

We can decouple , but that is not the point of this report. For those who are interested in solving out these kind of equations, I would analytically solve them in Appendix A.

When we are dealing with these three sets of equation numerically, we will make use of a property of higher order differential homogeneous equations, and that is:

***Property 1:*** *All higher order forms of ordinary homogeneous differential equations could be decomposed into a linear system of first order ordinary equations.*

Now we note that to completely determine the motion of the particle in the Penning trap, we must have 6 initial condition that describe the three sets of couple equations. They are ) and , the initial displacement and velocity of the mentioned particle.

With all of the preparation in place, lets open up MATLAB and use the ODE45 function to help us solve the differential equation in a numerical way.  
  
ODE45 is a inbuilt MATLAB function that uses the Runge-Kutta method (A more robust form of Euler’s Method that you should have encountered in your Calculus for the Sciences course 1, or if not, Ordinary Differential Equations as taught by Fedor) to solve out systems of first order differential equations. In extension, ODE45 allows us to solve higher order ODE, and even coupled higher order ODE by the property 1 that I have highlighted in the previous page.

The Pseudocode goes like this:

Define a function The input will be the expression, in which is a single double precision number that reflects on the time elapsed, and is an array of values, in which , is our respectively. The expressions , which is the charge, electric potential constant, trap dimension, mass and the strength of the external magnetic field respectively are experimental parameters, which are inputs by the user in the main script.

. The output will be a 6 by array of values, for which we will decompose the 3 systems of second order differential equations into a 6 first order equations. Here is what the function does:

dydt = zeros(6,1);

-This defines zero matrix of dimension 6 by 1, or

dydt(1) = y(4);

dydt(2) = y(5);

dydt(3) = y(6);

dydt(4) = (q/m)\*((-(V0)/(2\*d^2))\*(-y(1))+B0\*y(5));

dydt(5) = (q/m)\*((-(V0)/(2\*d^2))\*(-y(2))-B0\*y(4));

dydt(6) = (q/m)\*((-(V0)/(2\*d^2))\*(2\*y(3)));

-This is the heart of the function, in which we will define to be , to be , to be .

Now notice that is , is , and is , all written in terms of , which is our respectively.

Now with the function being defined, we can proceed on to our main script.

clc

clear all

close all

-This is to clean our list of stored variables and graphs

q = 1.60E-19;

V0=5;

d = 1;

m = 9.11E-31;

B0=1e-5;

-We define the experimental constraints , which is the charge, electric potential constant, trap dimension, mass and the strength of the external magnetic field respectively.

tf=0.5e-4;

tspan=[0 tf];

-We then define the start and the end time in a 2 by 1 array of values in this fashion [. For the sake of convenience, we set . Alternatively, you can also specify the time step by using the linspace function , where is the number of nodes that the user wants to have and the time step is.

x0 = [0 0 4e-2 1e-6 0 0];

-Initialize a 6 by 1 array of initial values, arranged in this fashion: [], to be fed to the ODE45 function.

opts = odeset('RelTol',1e-15);

-This is to force our ODE45 differential equation solver to solve within an error of . “RelTol” means relative tolerance.

[t,y]=ode45(@(t,y) Ian\_Yap\_Chang\_Jie\_02\_odefcn\_pt2(t,y,q,m,d,V0,B0),tspan,x0,opts);

-This is the ODE45 function that is being inbuilt into the MATLAB software. Note that “@(t,y) odefcn\_pt2(t,y)” means that the MATLAB calls in the function “odefcn\_pt2” while knowing that that function takes in two data sets and for iterative Runge Kutta method, starting from the initial condition , which we have defined earlier. are the time domain and the initial condition, and has to be the same matrix dimension as (Theoretically and computationally and common sensically), which is a array. We also note that ODE45 will determine the next value of using an adaptive response, and hence the step time between each iteration will not be equal (That is a reason why Runge Kutta method is more robust than the Euler’s Method, which uses a fixed time step of )

-ODE45 will output t and y,in which will be a matrix in which there will be values of that ranges from to . y will be a matrix in which for every describes at the time (The left hand side is different from right hand side , in which on the right hand side is used to describe the matrix that is output by ODE45).

X=y(:,1);

Y=y(:,2);

Z=y(:,3);

-A fancy way of renaming , such that we would have columns of the individual displacements individually. Note that the dimension of , , are matrices.

plot3(X(1),Y(1),Z(1));

fig=gcf;

fig.Units='normalized';

fig.OuterPosition=[0 0 1 1];

curve = animatedline('LineWidth',3);

set(gca,'XLim', [min(X) max(X)]\*1.05,'YLim', [min(Y) max(Y)]\*1.05,'ZLim', [min(Z) max(Z)]\*1.05);

view(3);

hold on

rotate3d on

-This is the 3D plot of the displacement with respect to the distance. The animated line function creates an animated line without any points (i.e an empty curve). We further let the linewidth of the curve (that is to be added) to be of size 3.

-The three lines fig=gcf; fig.Units='normalized'; fig.OuterPosition=[0 0 1 1]; is to force the graph screen to take up the whole screen of your computer. (Thanks to  [sipsj11](https://www.mathworks.com/matlabcentral/profile/authors/4430212-sipsj11) for pointing out this solution in the MATLAB forum: <https://www.mathworks.com/matlabcentral/answers/280686-how-do-i-make-a-figure-full-screen>). Note: I disabled that portion because MATLAB seems to take too much memory for that process.

-The set function is to allow us to define the range of our 3 axes, in which we will take the min and max of our range to be and where and are the minimum and maximum of the displacement for each direction

- “view(3)” refers to the standard viewing point when we are looking at the graph, “hold on” allows us to superimpose a graph with another (Basically allowing us to modify the current graph) and “rotate3d on” allows us to look at the motion of the particle from various perceptive.

xlabel('x')

ylabel('y')

zlabel('z')

title(['Motion of the Penning trap of a particle, with ' num2str(q) ' C of charge, ' num2str(m) ' kg of mass, characteristic trap dimension d=' num2str(d) ' m, electric quadrupole potential coefficient of ' num2str(V0) ' V, and a coefficient of strength of magnetic field B=' num2str(B0) ' T'])

-Standard labelling of graphs and so on and so forth. Note that ‘num2str(x) converts the value of that particular x into a string, hence the word play “number (num) to (2) string (str)”.

sizey=size(y);

for i=1:sizey(1)

addpoints(curve,X(i),Y(i),Z(i));

head=scatter3(X(i),Y(i),Z(i),'filled','MarkerFaceColor','b','MarkerEdgeColor','b');

drawnow;

current\_frame=getframe(gcf);

if i == 1

[mov(:,:,1,i), map]=rgb2ind(current\_frame.cdata, 256, 'nodither');

else

mov(:,:,1,i) = rgb2ind(current\_frame.cdata, map, 'nodither');

end

delete(head);

end

-We will start to do a for loop for , while adding successive points to the curve.

-We note the addpoints function is to add points to the curve that was defined earlier in the previous previous green paragraph. This means that the curve will increase its dimension by on the horizontal side, i.e the curve at step is a array.

-The head is just a point in which we represent it as a point particle (We can also represent it as a space filling sphere, and it will be just a simple modification of a cluster of points.

-So far, we would want the head to move, so for , we force a “drawnow” for the graph to plot the head and the trajectories, and then delete the head as so to add in the new head at a new point at the iteration

-Last but not least, ”getframe(gcf)” allows us to have a capture of our graph as a frame. “rgb2ind” is a function in which you can convert the RGB image, which is our graph, into an indexed image, which will be stored into our “mov” file. So the “mov” file is just an sequence of index image, just like the frame of the movie.

head = scatter3(X(i),Y(i),Z(i),'filled','MarkerFaceColor','b','MarkerEdgeColor','b');

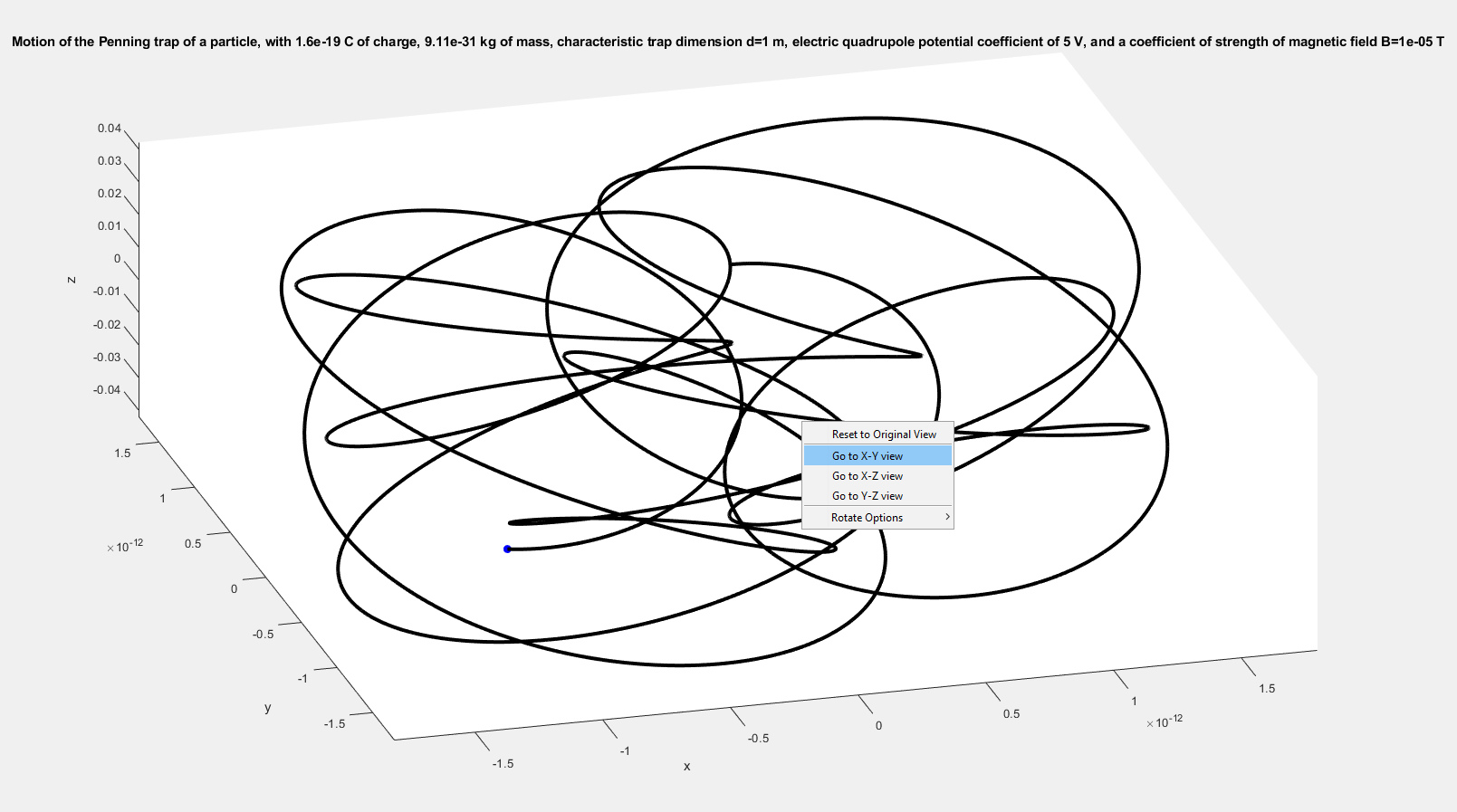
imwrite(mov ,map, 'Penning\_trap.gif', 'DelayTime', 0.03, 'Loopcount', inf);

- I am putting in the head again (the same as the previous yellow paragraph) before I send it to the current MATLAB folder path as a .gif file.

Well, this concludes my code that is attached on Github. Now we will look at the results:

How it should look like:



You can also pan it out into an x-y graph:   
  
  
  
  
  
And have look at the beautiful graph, which is the difference of angular frequency between the radial circular motion and the vertical oscillation of the graph:



That concludes my own project on the simulation of the Penning Trap. Quite a nice project to do! :D  
  
And a shoutout to the students who are reading this document. You should learn measure theory (Lebesgue measure) and the application of measure theory into partial differential equations. I am currently learning that now at this point of writing.

**Appendix A**

You might ask if there is an analytical way to solve this coupled differential equations. There is, and we can do that by a clever trick. Now let’s write down the 3 equations of motion again:

We note that the variable is decoupled. This is a second order homogeneous equation and we can see that the solution to is:

And note that and is either real or complex based on whether is negative or positive. We would want to confine our particle in a box, and hence we don’t want any terms in which is not bounded by , where is a positive number. In other words, . This constraint forces , and hence and is complex in nature (complex solution space). We hence note that and must be of opposite polarity.

By a change of basis in the solution space: , we can rewrite:

We note that by this change of basis, we can find real valued , that is a unique solution to the equation of motion in the z-direction. For more details, please consult this book: <https://www.amazon.com/Elementary-Differential-Equations-Boundary-Problems/dp/1118157389>

This is a very good book and it goes through the theorem in an acceptable deep detail. Highly recommended to buy for future references.

We have solved out , which will be confined between and , oscillating with an angular frequency of. Good. Now let’s focus on the coupled x-y equations.

We use a very clever trick. First, let’s notice that there is some symmetry between the two equations, i.e. the x and y variable are swapped when we compare both equations. However, the symmetry is broken in this manner of transformation from the x equation to the y-equation:

Now let’s write down the two equations in a more explicit form:

Calling as and as , we then have:

Differentiate both equations once to get

We differentiate again:

Substitute into to obtain:

Substituting into gives:

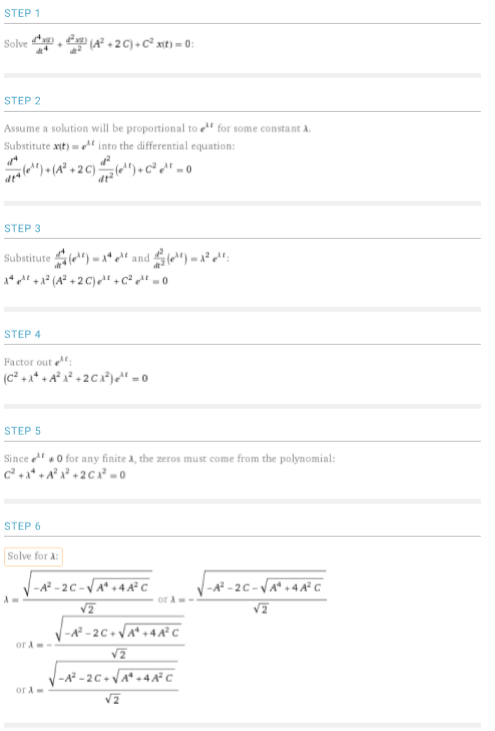
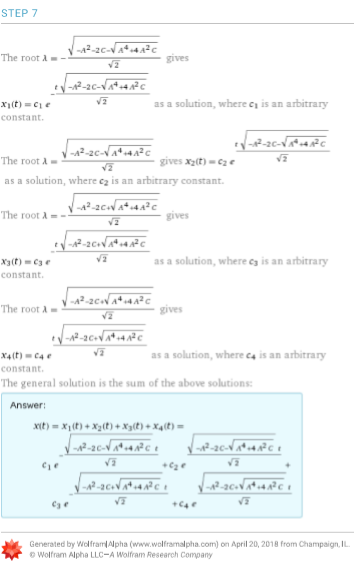
In a similar way, substituting into gives:

Then substituting into gives:

Summing up:

So, in all, in order to decouple the two equations, we have to sacrifice the simplicity of the second order differential equation to solve a fourth order differential equation. Note that and are perfectly symmetrical.

Since I am lazy to solve a fourth order differential equation, we then have, with the help of Wolfram Alpha Pro,

The same analysis could be done for as well. Note that we if we want to bound both and , we need and to be a complex number. Note that there are many possibilities of and to satisfy this equation, and while we can solve it analytically, I feel that this appendix is becoming a too long. So, I will stop here, and have a good weekend.   
  
Note that you can also use Laplace’s Transform to solve this equation as well. Read up on this book <https://www.amazon.com/Elementary-Differential-Equations-Boundary-Problems/dp/1118157389> for more details.

Acknowledgements:

I would like to thank Hasan Mehedi for his help in the technical parts of MATLAB, as well MATLAB for having such a comprehensive manual, such as the ODE45 manual and so. Last of all, I would like to thank Dr Koh for giving such a wonderful project for us to do.