PHYS 361: Lab 9, Solving 2nd Order ODEs

Background

This week's online lectures focused on how to solve coupled first-order equations using Runge-Kutta and MATLAB's built-in solvers.

Using the Built-in ODE Solvers to Solve a Single 1st-order ODE

In the last lab, we worked on solving 1st-order ODEs using solvers (Euler, Runge-Kutta) that we wrote ourselves. However, using built-in solvers in real life can be more efficient.

In the lecture, I showed you how to solve the radioactive problem,

$$\frac{dN}{dt} = -\gamma N \tag{1}$$

using an anonymous function decayode=@(t,N) -gamma*N. Note, that the anonymous function I defined depends on both t and N, even though the ODE only depends on N. That is because MATLAB's built-in function only work on functions that have two inputs. The first input must be time (or whatever the derivative is "with respect to"), and the second input must be the thing you are trying to determine the rate of change of. In this case, we are trying to understand the rate of change of the number of nuclei, N, with respect to time, t, so the input goes t then N.

You can also define a single 1st-order ODE as a user-defined function and pass it into a built-in solver like ode45. For the radioactive problem, the function would look like

```
function dudt = decayode(t,u)

    gamma=0.1;
    dudt=-gamma*u;

end
```

There isn't much of an advantage to using a user-defined function when you are only solving one 1st-order ODE, but it is worth practicing doing so before using it to define coupled 1st-order ODEs. Note that the same rules apply for the user-defined function's number of inputs and their order.

Solving coupled ODEs with a modified user-defined Runge-Kutta function

One way to solve coupled 1st order ODEs is to pass them into a Runge-Kutta function, like the ones we wrote last week, and find solutions at each time step by calculating mid-point slopes for each ODE (i.e., the k values). Note in the code below that the ODEs are solved simultaneously. The mid-point values depend on one another. In the lecture example, the number of lions *depended* on the number of gazelles. So it is not accurate to solve the ODE that describes the number of lions over a time period and *then* solve the ODE that describes the number of gazelles. They *must* be solved together. Not solving them together is a common mistake!

```matlab

function [t,y1,y2] = rk4(ode1,ode2,tmax,tmin,h,y10,y20)

    %Number of iterations
    N=round((tmax-tmin)/h);

    %Initial Values
     t(1)=tmin;
     y1(1)=y10;
     y2(1)=y20;

    %Loop to solve ODE
     for i=1:N

         %Find midpoint values
          k1=ode1(y1(i),y2(i));
          j1=ode2(y1(i),y2(i));

          k2=ode1(y1(i)+.5*k1,y2(i)+.5*j1);
          j2=ode2(y1(i)+.5*k1,y2(i)+.5*j1);

          k3=ode1(y1(i)+.5*k2,y2(i)+.5*j2);
          j3=ode2(y1(i)+.5*k2,y2(i)+.5*j2);

          k4=ode1(y1(i)+k3,y2(i)+j3);
          j4=ode2(y1(i)+k3,y2(i)+j3);

         %Find the average of midpoint values and update the solution
          y1(i+1)=y1(i)+1/6*h*(k1+2*k2+2*k3+k4);
          y2(i+1)=y2(i)+1/6*h*(j1+2*j2+2*j3+j4);
          t(i+1)=t(i)+h;
     end
end
```

Solving coupled ODEs in matrix format

The lecture also demonstrated how to use matrices to solve the same problem more efficiently. In this case, the coupled ODEs become the rows of a dudt array. In the predator-prey problem, we defined this array using a function.

```
function dudt = predatorprey(t,u)
    %u(1)=nl;
    %u(2)=ng;
    %Define constants
    bg= 1.1;%birthrate of gazelles, 1/years
    bl= 0.00025; %birthrate of lions, 1/years
    dg= 0.0005; %deathrate of gazelles, 1/years
    dl= 0.7; %deathrate of lions, 1/years

    dudt=zeros(2,1);

    %Rate of change of lions
    dudt(1)=bl*u(1)*u(2)-dl*u(1);
    %Rate of change of gazelles
    dudt(2)=bg*u(2)-dg*u(2)*u(1);
end
```

Note, the dudt is initialized with a specific size (2 rows, 1 column) with this line dudt=zeros(2,1). That is a helpful step that prevents things from getting funky later (and by funky, I mean unintended issues with matrix multiplication).

This whole function can be passed as input into another function that uses the 4th-order Runge-Kutta algorithm to solve for the number of lions and gazelles at each time step.

```

function [t,u] = rk4mat(dudt,tmax,tmin,h,u0)

    %Number of iterations
    N=round((tmax-tmin)/h);

    %Initial Values
    t(1)=tmin;

    %Sets the rows in the first column equal to the initial values
    u(:,1)=u0;

    %Loop to solve ODE
    for i=1:N

        %Find midpoint values
        %u(:,i) is both rows at time "i"
        %k1,k2,k3,k4 will be 2x1 arrays
        k1=dudt(t(i),u(:,i));
        k2=dudt(t(i)+.5*h,u(:,i)+0.5*k1);
        k3=dudt(t(i)+.5*h,u(:,i)+0.5*k2);
        k4=dudt(t(i)+h,u(:,i)+k3);

        %Find the average of midpoint values and update solution
        %u(:,i+1) sets the next column in my 2-row u array
        u(:,i+1)=u(:,i)+1/6*h*(k1+2*k2+2*k3+k4);
        t(i+1)=t(i)+h;
    end
end
```

Note this function does the same thing as the earlier one. It is just doing it in matrix form. The output, u, will be a 2xN array where the first row stores the solution to the first ODE (in this case, the number of lions), and the second row stores the solution to the second ODE (gazelles). Each column gives these values at a different time. The number of time steps determines the number of columns, N.

Solving coupled ODEs with `MATLAB`'s built-in functions

You need to know how to solve ODEs in matrix format because that is how MATLAB's built-in solvers require you to input your ODEs in matrix form. These built-in solvers are more advanced than the basic RK algorithms we have learned. They run faster and can be more accurate. So you should use them!

In the program below, I define the ODE in a function like the previous example. Note the function is at the very end of the file (as it must be). However, now I am passing the function that defines the ODEs (`@`predatorprey), the time span (`[`tmin tmax`]`), and the initial values (`[`nl0; ng0`]`) into the built-in solver `ode45`. The output of the built-in solver is two matrices: t is a column matrix with the times for each solution, and u stores the solution to the ODEs. Here, the first column of values is the number of lions, and the second is the number of gazelles. **You should follow the examples from this introduction when solving this week's lab problems.**

```matlab

%Program: predator_prey
%Predator-prey model for lions and gazelles
%Author: Darci Snowden
%Date: October 22, 2020

clear all;
close all;

%Define initial values
nl0=500;  %number of lions at t=0
ng0=3000; %number of gazelles at t=0

%Define h, tmin, tmax
tmin=0;
tmax=25;      %max time, years
h=0.1;        %time step, years

%Call the built-in function
[t,u]=ode45(@predatorprey,[tmin tmax],[nl0; ng0]);

%Extract number of lions, 1st column of u array
nl=u(:,1);

%Extract number of gazallels, 2nd column of u array
ng=u(:,2);

plot(t,nl,'r',t,ng,'b','LineWidth',2);
legend('Lions','Gazelles');
set(gca,'FontSize',14.);
xlabel('Time (years)','FontSize',14.)
ylabel('Number of Animals','FontSize',14.)

%Define the coupled ODEs in matrix form
function dudt = predatorprey(t,u)
    %u(1)=nl;
    %u(2)=ng;
    %Define constants
    bg= 1.1;%birthrate of gazelles, 1/years
    bl= 0.00025; %birthrate of lions, 1/years
    dg= 0.0005; %deathrate of gazelles, 1/years
    dl= 0.7; %deathrate of lions, 1/years

    dudt=zeros(2,1);
    %Rate of change of lions
    dudt(1)=bl*u(1)*u(2)-dl*u(1);
    %Rate of change of gazelles
    dudt(2)=bg*u(2)-dg*u(2)*u(1);
end
```

Separating a 2nd-order ODE into two 1st-order ODEs

Lots of physics problems are described by second-order ODEs (blame Newton). However, this is not challenging because higher-order ODEs can always be rewritten as a set of coupled 1st-order ODEs.

In the space below, write how you would separate the 2nd-order ODE that describes the motion of a simple harmonic oscillator.

$$\frac{d^2x}{dt^2} = -\omega^2 x \tag{2}$$

Now separate the 2nd-order ODE you will solve in Problem 2.

$$\frac{w}{g}\frac{d^2y}{dt^2} = T - w - D, \tag{3}$$

where $w(t) = 3000 - 80t$, $D = 0.005g(\frac{dy}{dt})^2$, and T is a constant.

(You don't have to turn this page in.)

Directions: You should write and upload a single script (.m file) for each problem below. Put the number of the problem in the name of your script. Include all user-built functions at the bottom of your script. Make sure to comment your code and follow the format of a script given out in class (an example can be found on Canvas). You do not need to upload the plots. I will run the code and generate the plots from it.

1. For the first problem, we will practice using the built-in ODE solver `ode45` to solve a single 1st-order ODE.

$$\frac{dy}{dx} = x - \frac{xy}{2} \tag{4}$$

   (a) Write the ODE as an anonymous function and use `ode45` to solve it from x= 1 to 3 with an initial value of y=1.

   (b) Write the same ODE as a user-defined function similar to the example given for the radioactive problem in the lectures (also described in the introduction) and use `ode45` to solve it for the same conditions as part (a).

   (c) Compare the solutions from parts (a) and (b) to the exact solution $y = 2 - e^{\frac{1-x^2}{4}}$ by making a plot.

2. If you haven't already, finish the problem we started in class on the Simple Harmonic Oscillator. In particular, modify your 2nd-order and 4th-order Runge-Kutta functions to accept a system of equations (in matrix form, similar to the example program on Canvas). Then use both functions to solve the equation of motion for the simple harmonic oscillator:

$$\frac{d^2x}{dt^2} = -\omega^2 x \tag{5}$$

   (a) The exact solution of the position of the simple harmonic oscillator is

$$x = Acos(\omega t) + Bsin(\omega t). \tag{6}$$

   Solve for the coefficients A and B by hand, assuming v(t=0 s) = 1.0 m/s and x(t=0 s)=0 m.

   (b) Make a single figure (with four sub-panels) showing the exact and numerical solutions for the following conditions using both your 2nd-order and 4th-order Runge-Kutta functions. (Each plot should have three lines.)

      i. $\omega = 0.1, v_0 = 1.0, x_0 = 0$,h=1
      ii. $\omega = 0.1, v_0 = 1.0, x_0 = 0$,h=5
      iii. $\omega = 0.1, v_0 = 1.0, x_0 = 0$,h=10
      iv. $\omega = 0.1, v_0 = 1.0, x_0 = 0$,h=50

v. Extra fun (extra credit). Use tic and toc to time each call of your user-defined functions and compare. Write a few sentences about your findings in the comments of your program.

(c) The equation of motion for a damped harmonic oscillator is,

$$\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + \omega^2 x = 0. \tag{7}$$

Copy and paste the user-defined function that defined the coupled first-order differential equations for the non-damped simple harmonic oscillator. Title your new function shodamped and modify the equations to include the damping term. Define γ at the beginning of your program and pass it into your function as a global variable (same as ω). Simulate the system for $\omega = 0.1$ s^{-1},$v_0 = 1.0$ m/s ,$x_0 = 0$ m, h=1 s, and $\gamma = 0.02$ s^{-1} .

3. A small rocket with an initial weight of 3000 lb (including 2400 lb of fuel), initially at rest, is launched vertically upward. The rocket burns fuel at a constant rate of 80 lb/s, which provides a continuous thrust, T, of 8000 lb. The instantaneous weight of the rocket is given by $w(t) = 3000 - 80t$ lb. The drag force, D, experienced by the rocket is given by $D = 0.005g(\frac{dy}{dt})^2$ lb, where y is the distance in ft, and $g = 32.2$ ft/s^2. The equation of motion for the rocket is given by:

$$\frac{w}{g}\frac{d^2y}{dt^2} = T - w - D. \tag{8}$$

Determine and plot the rocket's position, velocity, and acceleration (three separate panels of one figure) as a function of time from $t = 0$ to $t = 3$ s. To do this, you must reduce the second-order ODE to a system of two first-order ODEs.

(a) Solve the problem using your own 4th-order Runge-Kutta algorithm (that accepts systems of linear equations) using a step size of 0.05 s. You should be able to do this by modifying the function we defined in class to specify the system of linear equations for the simple harmonic oscillator.

(b) Use one of MATLAB's built-in functions (like ode45) to solve the problem. You should put this code in the same script as part (a). You do not need to make a new plot since the results should be the same as what you found in part (a).

(c) Make sure to label the plots.