APPM 2460
Systems of ODEs

1 Introduction

Last time we solved ODEs numerically, we focused on solving first order ODEs using Euler's method. However, Euler's method can be rather inaccurate since it uses a "linear approximation" to find the solution to the ODE. Furthermore, we did not explore what to do once we are tasked with solving higher order problems.

To solve higher order problems, we rewrite the problem as a *system* of first order differential equations and then solve the system of equations *simultaneously*. Unless the system has a very special structure, using a simple method like Euler's no longer works to find the numerical solution.

Today, we will be implementing a built-in ODE solver in Matlab called `ode45`. This function implements the Runge-Kutta method to more accurately solve ODEs (that is where the "45" part of the function's name comes from). Before we can implement this solver to find the solutions to higher order problems, we need to understand the relationship between second order ODEs and systems.

2 Solving a Second-Order ODE with `ode45`

2.1 Higher order problems as system

Consider the initial value problem for an undamped, unforced harmonic oscillator:

$$x'' + x = 0, \quad x(0) = 1, \quad x'(0) = 1. \tag{1}$$

Some calculations show that the solution to the IVP is

$$x(t) = \sin(t) + \cos(t)$$

In order to solve this system in Matlab, we must first *convert* (1) *to a system of first order equations*. To do this, we pick the variable transformation

$$y = x'$$

Plugging this variable transformation into the second order equation gives the system

$$\begin{aligned} x' &= y, \\ y' &= -x, \\ x(0) = 1, &\quad y(0) = 1 \end{aligned} \tag{2}$$

Notice that the second order equation (1) is equivalent to the first order system (2).

2.2 Using `ode45`

We would like to solve (2) forward in time. To do this,

- *we must first create a function m-file that holds the differential equation.* Open a new script and save it as `system_ex`.
  - We have created function m-files before, but we have to be more careful when we do this for (2) because we must treat our x and y variables as the components of a larger *vector* instead of individual scalars.
  - Similarly, the output `v_out` of this function must be a *vector*. We will define a vector of zeros (using the `zeros` function) with the same number of elements as there are variables in the system.
    - NOTE: the vector `v_out` MUST be a column vector.

  In this m-file, define the system of equations in (2) as follows:

```
function v_out = system_ex(t,v)
    v_out = zeros(2,1);
    v_out(1) = v(2);       % The dx/dt eq
    v_out(2) = -v(1);      % The dy/dt eq
end
```

  To emphasize:

  - In `system_ex`, the input `v` is a vector where `v(1)` is associated with x and `v(2)` is associated with y.
  - The same is true of `v_out`, where `v_out(1)` is associated with x and `v_out(2)` is associated with y.

- To solve ODE systems, we need to pick a time interval where we want to solve the problem: $t \in [0, 20]$.
  - Notice that we are not defining a time step here; `ode45` uses an "adaptive" (changing) time step when none is specified. (This helps ensure the accuracy of the solutions.)

- We also need to specify the initial conditions of the problem, $x(0) = 1$ and $y(0) = 1$. This should be written as a vector `[1,1]`, where the components are in the *same* order as the components of `v`.

- The `ode45` command has the syntax

```
[t_output, v_output] = ode45( @function_name, [t0,t_end],[x0,y0])
```

  For our specific problem, this becomes

```
[t,v] = ode45(@system_ex,[0,20],[1,1])
```

  The system has been numerically solved. Looking in the workspace, you now see the two variables t (holds all the time steps) and v (a matrix with 2 columns). The first column of the matrix is all the x values and the second column is all the y values.

  You can plot these against time to see the solution of each variable, or plot them against each other to generate solutions in the phase plane:

```
>> figure(1)
>> plot(t,v(:,1))
>> figure(2)
>> plot(t,v(:,2))
>> figure(3)
>> plot(v(:,1),v(:,2))
```

which will result in the plots shown in Figure 1 (more code will need to be written to include axis labels)

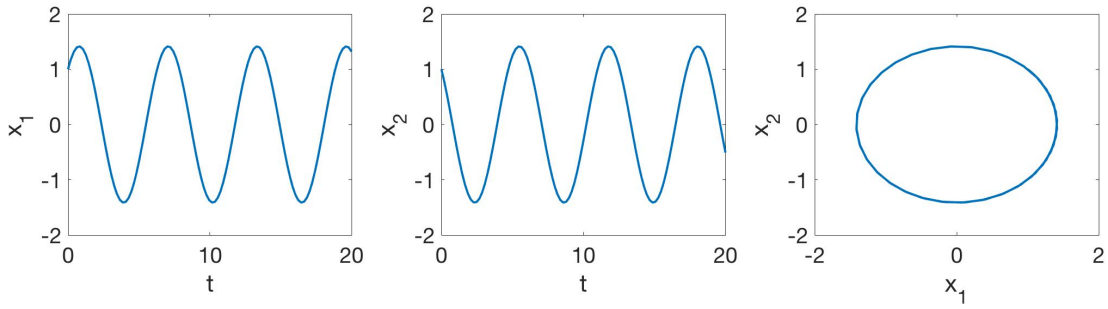

Figure 1: Plots for the harmonic oscillator example equation (1).

3 A nonlinear example

Consider the van der Pol equation,

$$x'' - x'(1 - x^2) + x = 0, \quad x(0) = 2, \quad x'(0) = 0. \tag{3}$$

This is a second-order ODE, but we will now recast it as a *system* of first-order ODEs.

- Let x_1 to denote x and x_2 to denote $x'(t)$. Then, we can write (3) as

$$\begin{aligned} x_1'(t) &= x_2(t) \\ x_2'(t) &= x_2(t)(1 - x_1^2(t)) - x_1(t) \end{aligned}$$

  where $x_1(0) = 2$ and $x_2(0) = 0$. We thus see that second-order ODEs (and indeed, any order ODE) can be rewritten as a system of first-order ODEs. This first-order form is what we need in order to use `ode45`. Note that we could treat the two elements of our system (x_1 and x_2) as two elements of a vector, that is to say our vector $\vec{v}$ would be

$$\vec{v}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \tag{4}$$

  So in this case, $x = x_1$ and $y = x_2$. Note that $\vec{v}\,'(t) = [x_1'(t), x_2'(t)]^T$. We could then write our ODE as a **vector-valued** function, i.e.

$$\vec{v}\,'(t) = f(t, \vec{v}) \quad \text{where} \quad f(t, \vec{v}) = \begin{bmatrix} x_2 \\ x_2(1 - x_1^2) - x_1 \end{bmatrix} \tag{5}$$

  With our ODE in this form, we can put it into MATLAB.

- As before, our first task is to write a function m-file that takes in a time `t` (i.e. a number) and a vector `v` and outputs another vector `[t_out, v_out]`. Well, we can do this by modifying the code for the previous example:

```
function v_out = odefun(t,x)
    v_out = zeros(2,1);
    v_out(1) = x(2);
    v_out(2) = x(2)*(1-x(1)^2) - x(1);
end
```

As before, the above function generates an output `v_out` that is a column vector, and we can call ode45 in the command window:

```
>> v0 = [2, 0];
>> tspan = [0, 20];
>> [t,v] = ode45(@odefun, tspan, x0);
>> plot(t,v(:,1),t,v(:,2)); % plot the two columns of the result
```

We have plotted both $x(t) = x_1(t)$ and $y(t) = x_2(t) = x'(t)$ against time.

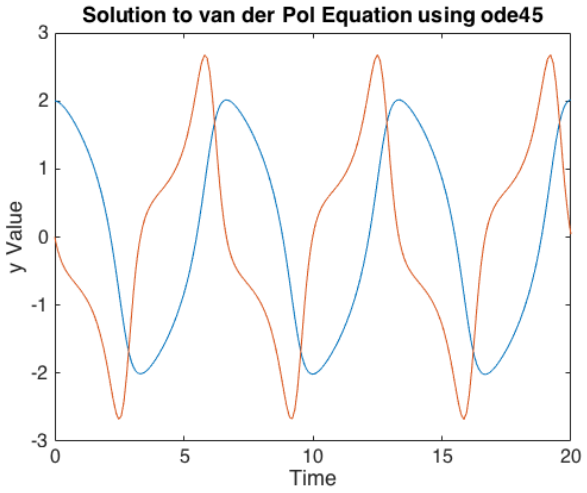

Figure 2: Solution to the van der Pol equation, using supplied code.

4 Phase Space

Are there other ways to visualize solutions to systems of ODEs? Yes. A final way to visualize these solutions is by examining the **phase space**.

Recall that we have a vector that depends on time, $v(t) = [x_1(t), x_2(t)]$. Plotting in phase space is simply plotting the parametric curve given by $(x(t), x_2(t))$. We can modify the above code to plot in phase space as follows:

```
>> v0 = [2, 0];
>> tspan = [0, 20];
>> [t,v] = ode45(@odefun, tspan, x0);
>> plot(v(:,1),v(:,2)); % plot the two columns of the result AGAINST ONE ANOTHER
```

Submit a published pdf of your script and any other supporting code needed to solve the following problem to Canvas by Monday, February 25 at 11:59 p.m. See the 2460 webpage for formatting guidelines.

Consider the second order ODE

$$x(t)'' - \mu(1 - x^2(t))x'(t) + x(t) - A\sin(\omega t) = 0$$

(a) Write a script that finds a numerical solution to the second-order IVP:

- Use parameters $\mu = 8.53$, $A = 1.2$, and $\omega = 2\pi/10$.
- Use the initial conditions $x(0) = 2$ and $x'(0) = 0$.

(Note that the above equation reduces to the van der Pol equation we solved above if we set $\mu = 1$ and $A = 0$. Adding in nonzero A adds **forcing** to the system, i.e. makes it non-homogeneous.)

(b) Solve the above equation over the time interval $t = [300, 600]$. In a single figure window, plot x versus time on one subplot and y versus time on a second subplot (label the axes). (To do this, first type `subplot(2,1,1)` to divide the figure window in half and choose the top half of the pane, then plot the $x(t)$ solution in the usual way. After you have finished plotting the $x(t)$ solution and labeling the axes, type `subplot(2,1,2)` and plot the $y(t)$ solution.)

(c) Solve the above equation over the time interval $t = [0, 20]$. Plot a phase space plot of the solutions with x on the horizontal axis and y on the vertical axis (label the axes and include a title)

If you solve the system correctly, you will see some very strange behavior! In particular, there will be a pattern that almost repeats itself, but not exactly. This is an example of **chaotic** dynamics.