Lab 4: Exploring second-order differential equations numerically

Introduction: In this lab, we will explore second-order differential equations using MATLAB. For numerical explorations of such equations, **pplane8** is very useful. You will also get to look "under the hood" and build your own custom version of **pplane8**.

Questions 1-2: Consider the initial value problem given below.

$$\textbf{DE: } 16y'' + 24y' + 73y = 0 \qquad \textbf{IC: } y(0) = 4 \quad y'(0) = -3$$

Here it is again in monic form (with the leading coefficient normalized to one).

$$\textbf{DE: } y'' + \frac{3}{2}y' + \frac{73}{16}y = 0 \qquad \textbf{IC: } y(0) = 4 \quad y'(0) = -3$$

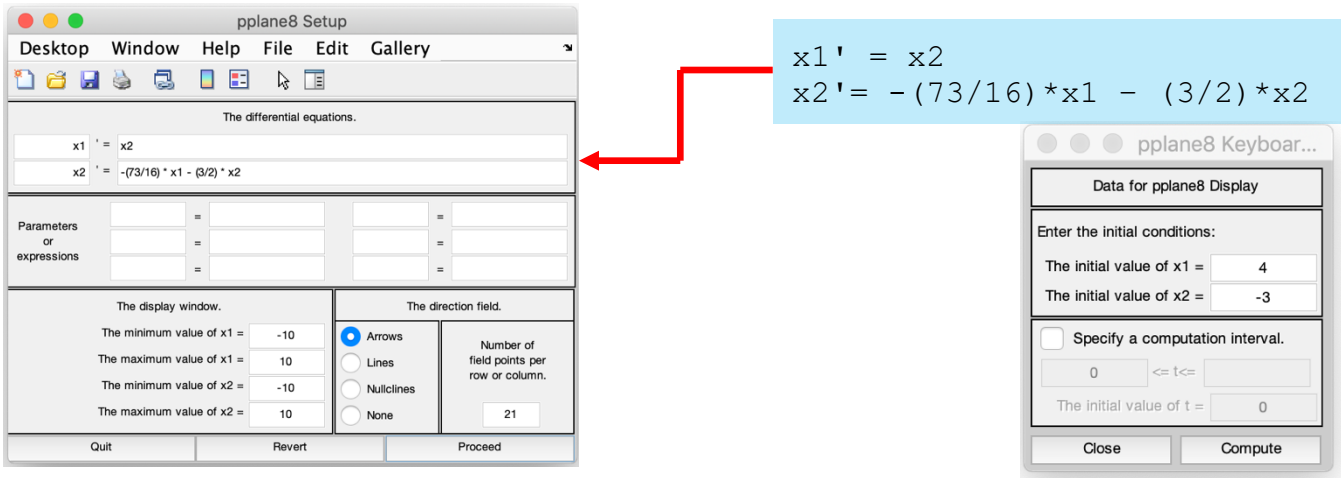
Note this 2nd-order equation is linear and homogeneous.

i. Download the pplane8.m file available in the **Software** folder. Look in the subfolder: MATLAB Versions of pplane8 and dfield8. Place in MATLAB's present working directory or pwd, then run it by entering: pplane8 in your MATLAB command window. Do not type the suffix .m

OR Run it by double clicking on the pplane8 file and select **Run**.

Create a **phase diagram**, for the DE above by entering the x'_1 and x'_2 equations and min and max values as below and then select **Proceed**.

You can enter the DE into pplane8 using the equations: $x'_1 = x_2 \qquad x'_2 = -\left(\frac{73}{16}\right)x_1 - \left(\frac{3}{2}\right)x_2$



ii. In the plot window, under the **Options** menu, make sure the Solver Direction is **Both**. Under the **Solutions** menu, using **Keyboard input**, find the solution (plot) satisfying $y(0) = 4, \quad y'(0) = -3$. Select this plot/solution curve and then under the **View** menu, use the **Property Inspector** to set the 'LineWidth' for the solution to 3. Then set the 'color' for this curve to **red**. Note, right clicking on the plot gives access to similar properties.

iii. Now use **Keyboard input**, find the solution satisfying $y(0) = 6, \quad y'(0) = -1$.
Change the 'LineWidth' to 3 and set the 'color' for this curve to **deep_purple**, a new custom color.

```
deep_purple = [0.5, 0, 0.5]
```

iv. Under the **Solutions** menu, select **Show nullclines**. Notice all solution curves that cross the nullclines do so at a local max or min. Give both nullclines a 'LineWidth' of 4.
Use the **Insert** menu choice, to place a yellow circle (ellipse) at the points (4,–3) and (8,–6) and place a red circle at the equilibrium point (0,0). (Red, for stopping point.)

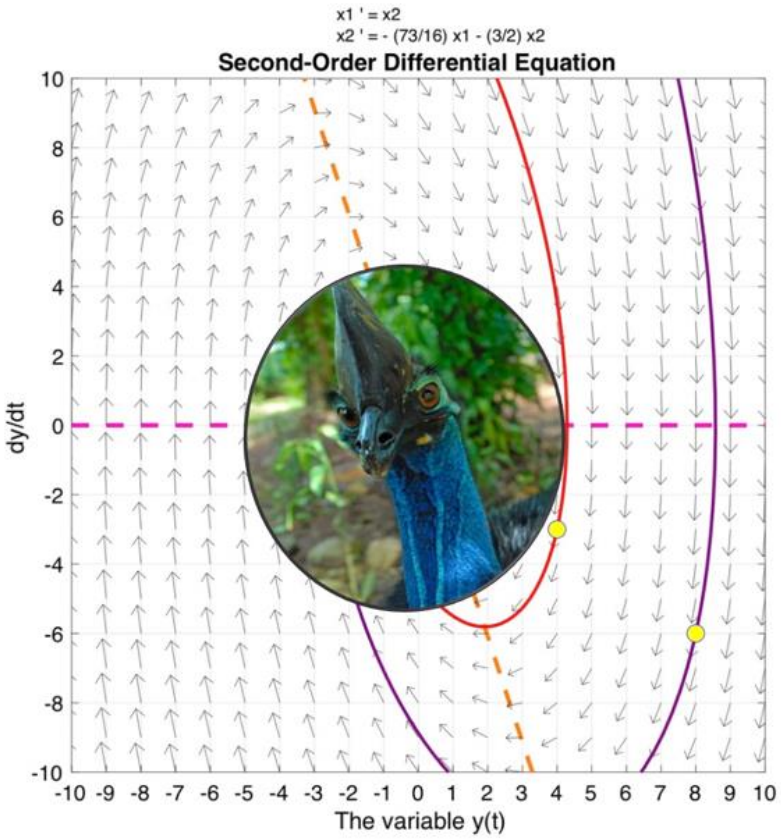
Questions 1-2: Paste your completed phase plot in the answer template.
Be sure the two solutions are shown in **red** and **deep_purple**. Be sure the two null lines are shown.

Replace this sample plot which is partially hidden by a graphic. Your answer must not show the cassowary!

SAMPLE

Grader:
i. One point if the curves are correct.

ii. One point if the colors are correct.



ENGR 232 – Lab 4: Second-Order Differential Equations

Questions 3-4-5: Build your very own custom pplane8 in three simple steps!

Now that you have explored the solutions **qualitatively** using **pplane8.m**, it's time to "look under the hood" so to speak. Let's try to make our own "**pplane8**". We'll need to plot the direction field and solution curves given a number of initial conditions. For the solution curves, we'll use **ode45**. In the next portion of this lab, we'll try to duplicate the graph you just obtained using **pplane8** from scratch – and even add a few bells and whistles!

The code below shows how you can create a direction field for any second-order differential equation. In practice, you will need to tweak the settings to obtain a polished look. I'll give the code in three blocks. Each block should remind you of settings that are given when you first open **pplane8**.

Block 1: Start with a differential equation of the form: $ay'' + by' + cy = 0$
(It doesn't matter what you name the variables, as long as you are consistent.)
For now, we'll treat *a*, *b* and *c* as constants but that can be relaxed later. We'll need to recast the DE in state vector form:

$$\vec{\mathbf{x}} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix}$$

Using the original second-order DE we find:

$$x_1' = x_2$$

$$x_2' = -\frac{c}{a}x_1 - \frac{b}{a}x_2$$

Let's create a block to enter our differential equation.

DE: $16y'' + 24y' + 73y = 0$ **IC:** $y(0) = 4 \quad y'(0) = -3$

```
%% Custom Direction Field for second-order differential equation:
%  ay'' + by' + cy = 0
clear, clc, close all
% BLOCK 1: Convert 2nd-order DE to a system of first order DEs.
% The new system will have the form dx/dt = F(t, x) where x is the state
% vector. So, x = [x1; x2]

% Define the constants here.
a=16; b=24; c=73;

% Define F(t, x) as an anonymous function.
% Below, x is a column vector with two components
F = @(t, x) [x(2); -(c/a) .* x(1) - (b/a) .* x(2)]
```

Notice we have defined the slope field $F(t, \vec{\mathbf{x}})$ using an **anonymous function**. Be sure to include the time variable *t*, even though this particular DE is time-independent, as we will need that to invoke **ode45** to plot the solutions.

Block 2: Setting up your plotting window.
Much of the code below is just to polish the appearance of the resulting graph.
Adjust the parameters as desired.

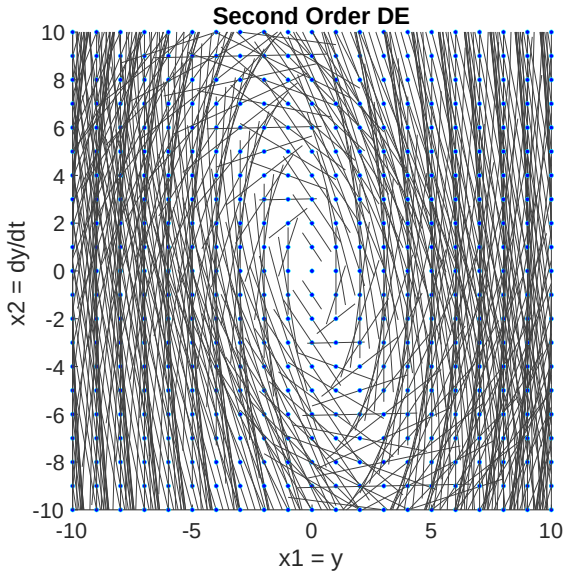
```
% BLOCK 2: Set the plot window dimensions here.
% Customize the plot settings.
L = 10 % max window dimension
x1min = -L; x1max= L;  x2min = -L;  x2max = L;
fig = figure(1); % Pop up a new figure
axis([x1min x1max x2min x2max ])
axis square
grid on; hold on
set(gca, 'FontSize', 20)
title('Second Order DE')
xlabel('x1 = y'); ylabel('x2 = dy/dt')
```

Block 3: Graph your Direction Field as a grid of tick marks with slope defined by the RHS of $\frac{d\vec{\mathbf{x}}}{dt} = F(t, \vec{\mathbf{x}})$
Focus on the **nested for loops** which is the heart of where the tick marks for the direction field are drawn.
We will soon encounter a problem which you will be asked to fix.

```
% BLOCK 3 - The Direction Field
radius = 0.4;                % Control the size of the tick marks
spacing_horizontal = 1;      % Control their horizontal spacing.
spacing_vertical = 1;        % Control their vertical spacing.
my_color = [0.25, 0.25, 0.25];  % Control their color.
optional_dots = 1;            % Control whether a dot is placed. Use 0 or 1


time = 0;
for x1p = x1min: spacing_horizontal: x1max
    for x2p = x2min: spacing_vertical: x2max
        x=[x1p;x2p]; % Update the state vector x.
        dx = F(time,x) ; % Find dx/dt
        % Add a line here to replace dx by a unit vector. Use norm().


        dx1 = dx(1) *radius;  dx2 = dx(2)*radius; % Scale tick marks.
        % Plot the tick marks!
       plot([x1p - dx1 , x1p + dx1 ], [x2p - dx2 , x2p + dx2] , 'Color', my_color);

        if optional_dots
            plot( x1p, x2p, 'cyano', 'MarkerSize', 4,  'MarkerFaceColor', 'blue')
        end
    end
    pause(0.1)
end
```

ENGR 232 – Lab 4: Second-Order Differential Equations

Question 3: Once you have all three blocks entered, run them to produce this direction field for our 2nd-order DE. Fix this graph so all tick marks have the same length.



Ack! Not so good.
The problem is some of the tick marks for **dx** are very long, resulting in a confusing picture. Where you see the **red** arrow in the starter code, add a line of code to convert the vector dx, into a **unit vector** by dividing it by its length.
Use the **norm()** command.
Then adjust the **radius** until you are satisfied with your tick marks.

Next, we need to add some solution curves.
Here is some code, which will plot **one solution** moving forward in time from the initial point (4,–3).

```
% Plot a single solution moving forward in time from the given initial point.
tStart = 0;  tEnd = 15;
both_directions = 1;      % Change to 0 to draw in the forward direction only.
show_initial_point = 1;   % Change to 0 to suppress drawing the initial points.

% Plot the solution curve with the given initial point.
x0 = [4;-3]; % Initial point #1

[~, x_out] = ode45(F, [tStart, tEnd], x0);
plot(x_out(:,1), x_out(:,2), red, 'LineWidth', 3)
% Show initial point using a yellow dot.
if show_initial_point
    plot(x0(1), x0(2), 'bo', 'MarkerSize', 12, 'MarkerFaceColor', 'yellow')
end
```

Question 4: A second solution curve: After drawing the first solution curve (using the code above) in red, add another solution for the second initial point (8, –6). Use our custom color **deep_purple**.

```
deep_purple = [1/2, 0, 1/2] % custom color
```

Hint: No need to repeat the first three lines above. Clone the remaining code, and simply adjust the initial point and color. You will need to add the 'color' keyword before deep_purple in your plot command.

ENGR 232 – Lab 4: Second-Order Differential Equations

Question 5: Solutions moving backwards in time: Now continue the solutions backwards in time, from the same points in the previous part. Draw each backwards solution using the same red/deep_purple color scheme used previously. Use an if statement so this is only done if the variable `both_directions` is `1`.

Hint: Clone your solution from the last part but adjust the call to **ode45** as follows.

```
[~, x_out] = ode45(F, tStart: -0.05: -tEnd, x0); % Backwards!!!
```

Solving backwards in time.

Question 6 : Add the nullclines

```
x1' = x2
x2'= -(73/16)*x1 - (3/2)*x2
```

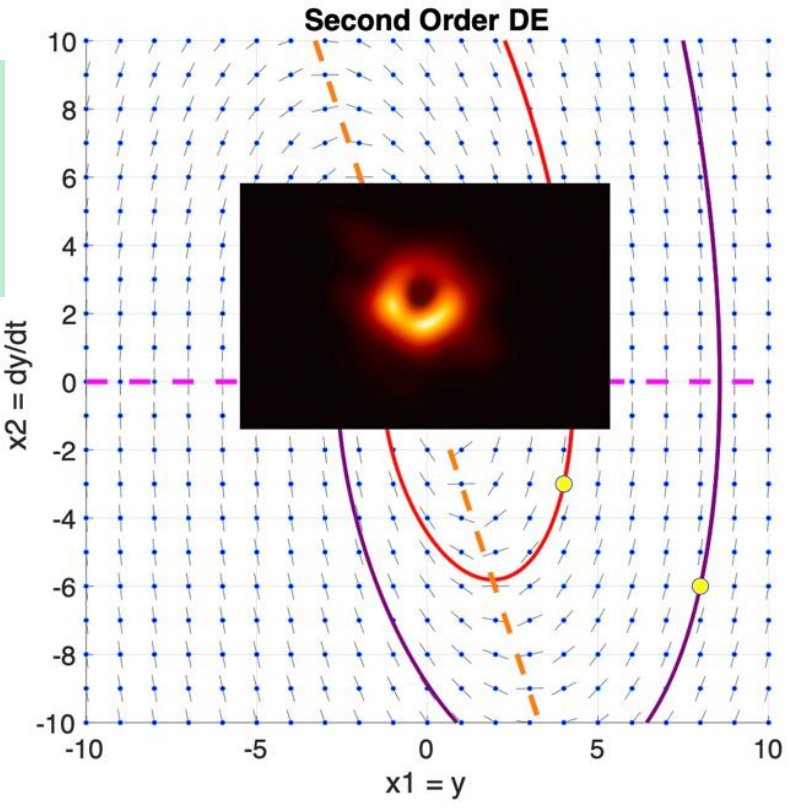
a. The nullclines are where each derivative is zero.
The first nullcline is simply $x_2 = 0$. Add the line through the points $(x_1, x_2) = (-10,0)$ and $(+10,0)$. Show this line in **magenta** using a **dash** and a line width of 4. Don't forget when you plot to bundle the two x-values together as in [–10,+10] as well as the y-values [0,0].

b. The second nullcline is given by the equation $x_2 = -\frac{73}{24} \cdot x_1$ and so passes through the two points $(x_1, x_2) = (-10, 730/24)$ and $(+10, -730/24)$. Show this line **in a custom orange** using a **dash** and a line width of 4. Define this color using:

```
orange = [1, 0.5, 0]
```

Questions 3-4-5-6: Replace this sample image with your completed graph.
Part of the image has been hidden by the recent image of the black hole in the M87 galaxy. Your submission must not include the black hole.

SAMPLE



Grader will award **four** points as follows.
i. The tickmarks all have the same length.
ii. Solution for (4,–3) is in **red**.
Solution for (8,–6) is in **deep_purple**.
iii. Solutions backwards in time are included.
iv. Nullclines are correct.

Question 7: Find the exact solution to your DE with the given initial point.

$$\textbf{DE: } 16y'' + 24y' + 73y = 0 \qquad \textbf{IC: } y(0) = 4 \quad y'(0) = -3$$

Here's some starter code:

```
%% Question 7:  Exact solution:
clear, clc
syms y(t) n
D1 = diff(y, t);  D2 = diff(y,t,t);
```

Now build your DE and find the exact solution using **dsolve**.

Question 7: The exact solution for **DE: $16y'' + 24y' + 73y = 0$** satisfying $\vec{\mathbf{x}}(0) = \begin{bmatrix} 4 \\ -3 \end{bmatrix}$ is:

$y(t) =$

Questions 8-10: Van der Pol Oscillator

Let's start over with a new DE, but now one that is non-linear. It can be considered a mass-dashpot-spring system with non-linear damping. You can read more about this famous oscillator here:

https://en.wikipedia.org/wiki/Van_der_Pol_oscillator

Van der Pol Oscillator:

$$y'' - \left(1 - \frac{y^2}{4}\right) \cdot y' + y = 0$$

The middle term is non-linear!

So now $a = 1, \;\; b = -\left(1 - \frac{y^2}{4}\right), \;\; c = 1$

Question 8: Start over in Block 1 and redefine $F(t, x)$ so it corresponds to this new non-linear system. Don't destroy your old answer. Just comment it out and add the new function. Or you can clone all your code by pasting it at the bottom of your script.

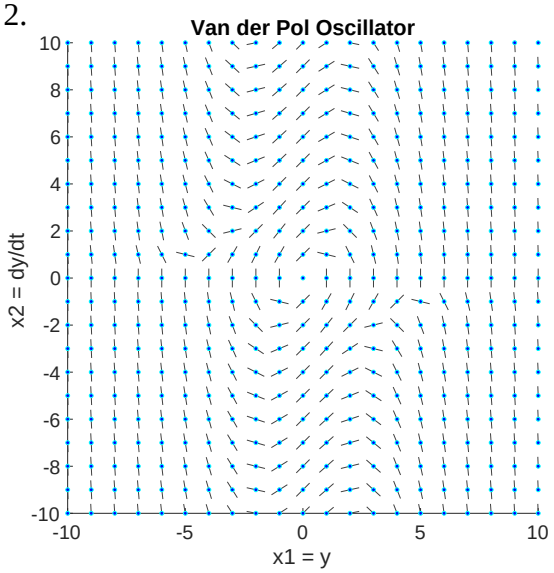
In terms of the state vector representation, you can see that if:

$$\vec{\mathbf{x}} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} y(t) \\ y'(t) \end{bmatrix} \qquad \text{then} \qquad \frac{d}{dt}\vec{\mathbf{x}} = \frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y' \\ y'' \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_1 + \left(1 - \frac{x_1^2}{4}\right) \cdot x_2 \end{bmatrix}$$

Tip: You will need to make careful use of pointwise operations by placing a dot before any ^ or *.

Alert: You are just changing the code for the anonymous function F for question 8. Might be safest to clone your code and paste it all into a new section at the bottom of your script file.

Next, change the title to '**Van der Pol Oscillator**' in Block 2.



Once you have created your phase plot for the Van der Pol oscillator, it should look like this. Now we are ready to plot solution curves using just a few mouse clicks.

Question 9: Use the **ginput** function so you can click on your figure and compute new orbits with a simple mouse click. The syntax for **ginput** is:

```
[X,Y,BUTTON] = ginput(N)
```

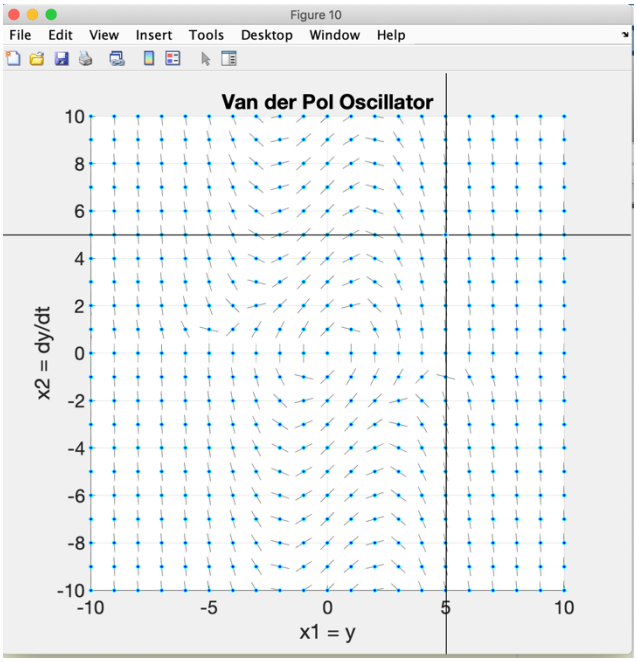
In addition to the x and y coordinates of each mouse click, this returns a third result, BUTTON, that contains a vector of integers specifying which mouse button was used (1,2,3 from left) or ASCII numbers if a key on the keyboard was used. The argument N is how many mouse clicks you wish to collect.

Rather than specify the number of mouse clicks, we will use a **while** loop and signal the end of our input by using a right click, after which the button value is 3. Input will also cease if you press **Enter** on the keyboard.

Enter this free code below in your MATLAB script file.

```
%% Interactive Graph Session using Mouse clicks
% Use mouse clicks to determine the initial points
% Use ordinary clicks (left clicks)
% Use a right click to end the interactive session or press Enter
clc
fprintf('Starting Interactive Graph Session\n')
tStart = 0;  tEnd = 15;
show_initial_point = 1;

mouse = 1;
while mouse==1
    [x, y, mouse] = ginput(1)
    x0 = [x;y]; % Initial point clicked on by user
    [~, x_out] = ode45(F, [tStart :0.01: tEnd], x0); % solution curve using ode45

    my_color = rand(3,1) % a random color
    plot(x_out(:,1), x_out(:,2), 'Color', my_color, 'LineWidth', 2)

    % Show initial point using a dot.
    if show_initial_point
        plot(x0(1), x0(2), 'bo',  'MarkerSize', 10, 'MarkerFaceColor', my_color)
    end
end

fprintf('End of Interactive Graph Session\n')
```

Run the interactive code. If you move the mouse over your interactive graph, you should now see crosshairs like this. Here we are about to click on a point near (5,5). Left click to see one solution. Keep on clicking to generate more solution curves. To stop the interactive session, right click – or press enter.



Snap-to-Grid: Inside the while loop, adjust the line below, so that the coordinates snap to the nearest integer values. Just use the round() command twice.

```
x0 = [x;y]; % Initial point clicked on by user
```

Question 10: Tons of solution curves all converging to the same limit cycle

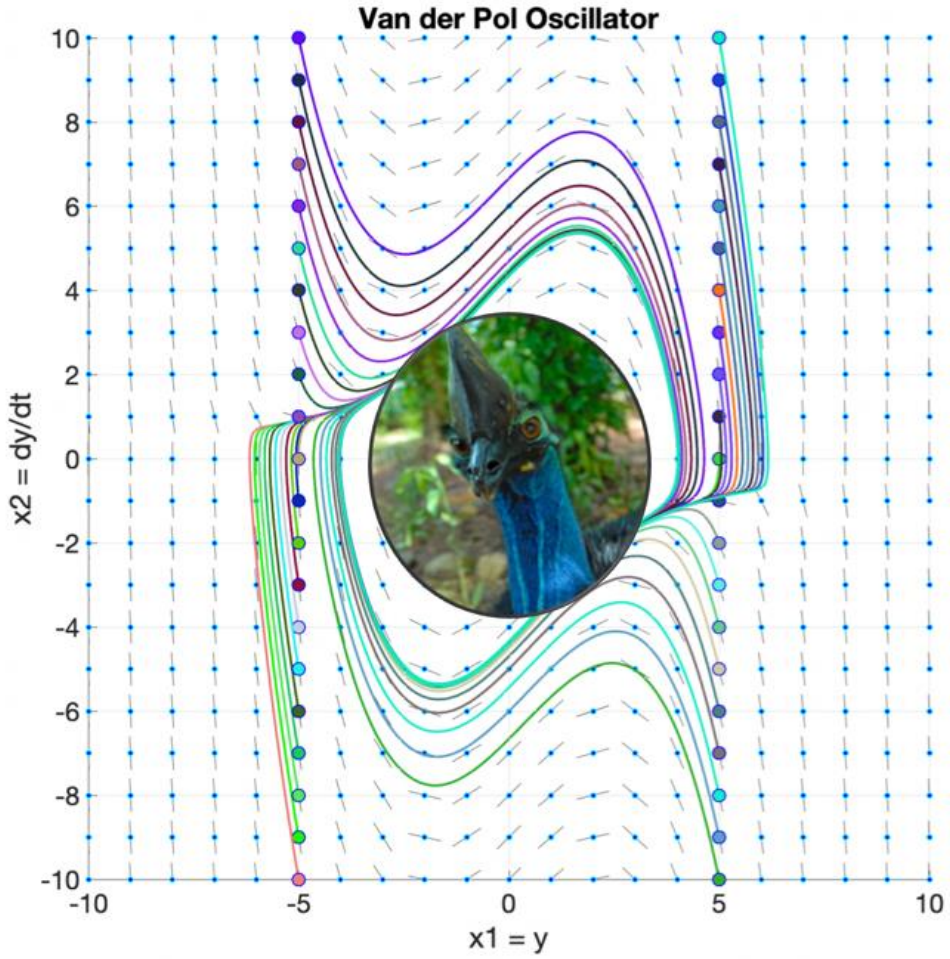
Start over with a fresh direction field and run the interactive code and click on all initial points for which the horizontal coordinate is ±5 and the vertical coordinate is an integer. That is a total of 21 points along the vertical line at –5 and another 21 points along the vertical line at +5.

You should notice how all 42 solutions curves converge to the same limit cycle!
It is important for this last point that all your initial points are equally spaced by one unit vertically as this allows the grader to see your snap-to-grid is working.

Questions 8-9-10: Replace the sample plot with your completed solution curves for the Van der Pol Oscillator. Sample shows the middle portion hidden behind an occluding image.

Grader will award one point for each of the three steps 8-9-10.

Sample with cassowary inside the limit cycle



Grader:
i. One point for the direction field and title.

ii. One point if there are 42 curves, 21 at –5 and 21 at +5.

iii. One point if snap-to-grid is working and all the initial points are perfectly, equally spaced vertically.

Ready to Submit?
Be sure all ten questions are answered. When your lab is complete, be sure to submit three files:

1. Your **completed Answer Template** as a PDF file
2. A copy of your **MATLAB Live Script**
3. A **PDF** copy of your **MATLAB Live Script** (Save-Export to PDF…)

The due date is the day after your lab section by **11:59pm** to receive full credit. You have one more day, to submit the lab (but with a small penalty), and then the window closes for good and your grade will be zero.