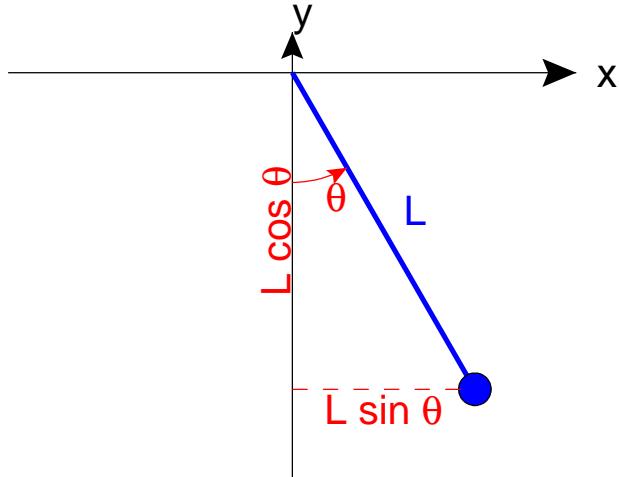


MECH 221 Computer Lab 6: The Simple Pendulum

OVERVIEW: Linear Approximation versus the Real Thing

In the ideal pendulum, a point mass m at the end of a light rod with length L swings under the influence of the gravitational acceleration g . We consider motion in a plane, with no friction.



Balancing torques around the pivot point leads to the equation of motion (EOM)

$$\ddot{\theta}(t) + \frac{g}{L} \sin \theta(t) = 0. \quad (*)$$

The sine function appearing here makes the differential equation $(*)$ nonlinear, and this puts it outside the scope of the methods we have developed so far. In fact, no known methods will produce an algebraic formula for solutions of $(*)$. But even without a formula, it is possible to compute highly accurate approximations to $\theta(t)$ for various initial conditions. Matlab has built-in functions for just this purpose, and this lab will introduce the most popular of them, `ode45`.

With the help of `ode45`, we will investigate a number of features of pendulum motion:

- (i) conservation of energy;
- (ii) discrepancy between the function $\theta(t)$ and a pure sinusoid; and
- (iii) relationship of period to swing-amplitude.

Our baseline for comparison in these categories is the related ODE built from the small-angle approximation:

$$\sin \theta \approx \theta \quad \text{for } |\theta| \ll 1.$$

This approximate EOM, namely,

$$\ddot{\theta}(t) + \frac{g}{L} \theta(t) = 0, \quad (\dagger)$$

is linear, so we can write down its general solution immediately:

$$\theta_{\text{lin}}(t) = C \cos \left(\sqrt{\frac{g}{L}} t + \phi \right), \quad C > 0, \phi \in \mathbb{R}.$$

Each of these solutions is a pure sinusoid whose period is unrelated to the amplitude factor C ; for any choices of C and ϕ , we do have

$$\frac{1}{2}mL^2\dot{\theta}(t)^2 + \frac{1}{2}mgL\theta(t)^2 = \text{const},$$

but the “conserved quantity” on the left is only an approximation to the energy of the true pendulum. We can anticipate seeing something interesting in each of the three categories mentioned above.

While we deepen our understanding of the physics of the true pendulum, we will also gain experience with Matlab’s built-in ODE solver, `ode45`. This function solves *systems of first order differential equations*. So before we can get started, we must define/invent the auxiliary function $\omega(t) = \dot{\theta}(t)$ to express the EOM in the first-order form:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ -(g/L)\sin\theta \end{bmatrix}. \quad (**)$$

For comparison, the small-angle approximation leads to a slightly different system:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ -(g/L)\theta \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -g/L & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \omega \end{bmatrix}, \quad (\ddagger)$$

Computer Lab 5 provides experience with linear systems like (\ddagger) : we know how to set up a phase plane diagram and relate the solution function $\theta_{\text{lin}}(t)$ to the motion of a point in the (θ, ω) -plane. Today’s project will extend this experience by filling in the phase plane trajectories for $(**)$ with computed approximations, and creating plots that give different views of the information encoded in the phase plane diagram. (For more discussion of the phase plane, see Appendix A.)

PENDULUM PHYSICS: Force and Energy

In the sketch provided, the angle θ determines the Cartesian coordinates of the pendulum bob:

$$x = L \sin \theta, \quad y = -L \cos \theta.$$

As the pendulum swings, all of x , y , and θ vary with time, but the equations above remain true. So it’s safe to differentiate both sides, and this explains why the bob’s linear speed v obeys

$$v^2 = |\mathbf{v}|^2 = \dot{x}^2 + \dot{y}^2 = (L \cos \theta \dot{\theta})^2 + (L \sin \theta \dot{\theta})^2 = L^2 (\cos^2 \theta + \sin^2 \theta) \dot{\theta}^2 = L^2 \dot{\theta}^2.$$

With the familiar definition $\omega = \dot{\theta}$, it follows that the pendulum’s kinetic energy is

$$T(\omega) = \frac{1}{2}mv^2 = \frac{1}{2}mL^2\omega^2.$$

Let’s choose the reference point for measuring potential energy at the bottom of the pendulum’s arc, where $y = y_0 \stackrel{\text{def}}{=} -L$. For any pendulum angle θ , the mass’s height above this point is $h = y - y_0$, so the pendulum’s potential energy is

$$V(\theta) = mgh = mg([-L \cos \theta] - [-L]) = mgL(1 - \cos \theta).$$

As the pendulum moves, both T and V will change. Their sum is the function

$$E_{\text{tot}}(t) = T(\omega(t)) + V(\theta(t)) = \frac{1}{2}mL^2\omega(t)^2 + mgL(1 - \cos \theta(t)).$$

The derivative of this function contains useful information. Thanks to the EOM $(*)$, we have

$$\frac{dE_{\text{tot}}}{dt} = \frac{1}{2}mL^2[2\omega\dot{\omega}] + mgL(0 + \sin\theta\dot{\theta}) = mL^2\dot{\theta}\left[\ddot{\theta} + \frac{g}{L}\sin\theta\right] = 0.$$

This is important: *for any unforced motion of the pendulum, the total energy must constant!*

Special Values. Let's focus on a particular setup where the physical quantities m , g , and L have convenient numerical values. Adjusting the units as needed, we take $m = 1$ and $g = L$. Then the key ingredients of our study simplify as follows:

- the true equation of motion $\ddot{\theta}(t) + \sin \theta(t) = 0,$
- the approximate equation of motion $\ddot{\theta}(t) + \theta(t) = 0, \text{ useful when } |\theta(t)| \ll 1,$
- the coordinate relationships $x = \sin \theta, y = -\cos \theta,$
- the angular velocity definition $\omega(t) = \dot{\theta}(t),$
- the scaled kinetic energy $K(t) = L^{-2}T(\omega(t)) = \frac{1}{2}\omega(t)^2,$
- the scaled potential energy $P(t) = L^{-2}V(\theta(t)) = 1 - \cos \theta(t),$
- the scaled total energy $E(t) = L^{-2}E_{\text{tot}} = K(t) + P(t).$

Energy-Based Predictions. When the pendulum is at its lowest point, $y = -L$, its potential energy is 0, thanks to our choice of reference level. The physical setup requires $-L \leq y \leq L$ always, so there is an absolute ceiling on the potential energy: in the notation above, the scaled potential

$$P(t) = L^{-2}V(\theta(t)) = 1 - \cos \theta(t)$$

must obey $0 \leq P(t) \leq 2$ at all times.

As the pendulum swings in obedience to the EOM, $\theta(t)$ changes and the potential energy function $P(t)$ changes too. But $E(t)$ stays constant, so the changes in $P(t)$ must be exactly compensated by changes in $K(t)$. At the instant when the pendulum swings through its lowest point, all of its energy must be in kinetic form. That is, $P(t) = 0$ and $E(t) = K(t)$ at that instant.

At any instant when the pendulum stops moving, its kinetic energy will be 0 and all the energy must be stored in potential. The point mass must be at the highest y -value it will ever reach; its angular velocity $\omega = \dot{\theta}$ is 0 because a sign-change is in progress, and the corresponding value of θ is either a maximum or a minimum.

Can the total energy function E (actually a constant) ever be larger than 2, the maximum of the potential energy function? Yes ... but then consistency requires that even when function $P(t)$ is at its maximum value—that is, even when the pendulum bob is at its highest point—some fraction of the total energy must still be in kinetic form. That is, the bob must pass through the straight-up position with instantaneous kinetic energy $K = E - 2$. The more E exceeds 2, the faster the bob moves at this point.

Force-Based Predictions. There is a perfect analogy between the approximate EOM (\dagger) and the equation of motion for a standard mass-spring system with $m = 1$:

$$\ddot{y} + ky = 0.$$

In this system, the exact solutions really are pure sinusoids. Using a weaker spring corresponds to using a smaller constant k , and the result is lower accelerations and a longer period of oscillation. To compare with this, let's rearrange our pendulum EOM (*) as

$$\ddot{\theta}(t) + \left(\frac{\sin \theta}{\theta} \right) \theta(t) = 0.$$

This puts the ratio $\sin \theta / \theta$ in the role of the spring constant k above. Although this ratio is not constant, the analogy can help us make some rough predictions. It's a basic math fact that $\sin \theta < \theta$

whenever $\theta > 0$, and the ratio $\sin \theta / \theta$ decreases as θ grows. Thus the force that acts to correct positive deviations from $\theta = 0$ is always weaker than the linear correction force one might expect from an ordinary spring. The weakening is slight near $\theta = 0$, but the restoring force gets smaller as the angles get larger. Thus it makes sense to predict that the period of our ideal pendulum will always be larger than 2π , and that the period might actually *increase* as the swing amplitude grows.

PENDULUM MATH: The Linear Case

For the approximate equation of motion $\ddot{\theta} + \theta = 0$, the general solution can be presented in the form

$$\theta_{\text{lin}}(t) = C \cos(t + \phi), \quad C > 0, \phi \in \mathbb{R}.$$

Using $\omega = \dot{\theta}$ gives the general solution for the first-order system (‡):

$$\begin{bmatrix} \theta_{\text{lin}}(t) \\ \omega_{\text{lin}}(t) \end{bmatrix} = C \begin{bmatrix} \cos(t + \phi) \\ -\sin(t + \phi) \end{bmatrix}, \quad C > 0, \phi \in \mathbb{R}.$$

Note that $\theta_{\text{lin}}(t)^2 + \omega_{\text{lin}}(t)^2 = C^2$, so the phase-plane path of the point (θ, ω) is a circle centred at the origin. The radius of that circle equals the amplitude of the oscillation in $\theta_{\text{lin}}(t)$. Further, the function $\theta_{\text{lin}}(t)$ is periodic with period 2π , and the value of C makes no difference to the period.

PENDULUM COMPUTATIONS: Matlab's `ode45`

Imagine confronting a general initial-value problem (IVP) in the math style:

$$\dot{\mathbf{y}}(t) = \mathbf{G}(t, \mathbf{y}(t)), \quad \mathbf{y}(t_0) = \mathbf{y}_0. \quad (3.1)$$

To get the computer to return a useful approximation to the solution $\mathbf{y}(t)$, we need to inform the machine about the time interval of interest, the initial vector, and the function \mathbf{G} . Here is a Matlab command that does all this:

$$[T\text{col}, Y\text{cols}] = \text{ode45}(\mathbf{G}, \text{tspan}, \text{startvector}); \quad (3.2)$$

- `ode45` is the name of Matlab's built-in solver. Matlab has other solvers, too, but they are for problems with particular features. Say `doc ode45` at the Matlab prompt for more information.
- `startvector` is the Matlab version of \mathbf{y}_0 , presented as a column vector.
- `tspan` is a row vector listing the times t for which we are requesting computed values of $\mathbf{y}(t)$. The first element, `tspan(1)`, should be the initial time t_0 . Matlab will solve (3.1) on the interval $[t_0, t_1]$, where t_1 is the user-provided number in the last element of vector `tspan`. (In this lab we must make sure `tspan` contains more than 2 elements, because `ode45` does something different in that case.)
- `Tcol` is a column vector of times from t_0 to t_1 returned by `ode45`. It declares the t -values for which computed values of $\mathbf{y}(t)$ are being provided. Typically, the elements in `Tcol` are identical to the requested times in `tspan`. (In the special case where `tspan` contains just 2 elements, t_0 and t_1 , `ode45` automatically chooses a sequence of points in the interval $[t_0, t_1]$ using some sophisticated error-control techniques, and returns these points in `Tcol`. We will

need equally-spaced times in `Tcol`, so we can't rely on Matlab's automated node generation scheme.)

- `Ycols` is a matrix holding the computed values of the solution vector $\mathbf{y}(t)$ at particular instants. Suppose t_5 is the time value in `Tcol(5)`. Then the entries in row 5 of `Ycols` will be the components of the vector $\mathbf{y}(t_5)$.

In another view, the desired solution $\mathbf{y} = (y_1, y_2, \dots, y_N)$ is a time-varying vector with N components. Each component is a time-varying scalar function. Selected values of function $y_1(t)$ are returned in column 1 of `Ycols`; corresponding values for $y_2(t)$ appear in column 2; etc. Each column lines up properly with the t -values in `Tcol`, so we can draw the graph of component function $y_2(t)$ by saying

```
plot(Tcol,Ycols(:,2));
```

(Recall that `Ycols(:,2)` denotes column 2 of `Ycols`, selected by specifying a column-index of 2 and using the colon to specify all rows.)

- `G` is a function handle. It refers to a function that receives two input arguments, the time t and a vector \mathbf{y} , in that order, and returns the corresponding *column* vector for use in ODE (3.1). For the linearized pendulum equation shown in line (‡), we could use a 2-element vector y and plan for $y(1)$ to represent θ and $y(2)$ to represent ω . Then we would say

```
G = @(t,y) [y(2); -(g/L)*y(1)]; % (In fact, we will have g/L=1 later.)
```

(This function actually makes no reference to the value of the input variable `t`, but `ode45` is designed always to apply `G` with the state vector in the second position, so the definition of `G` must allow for a t -value even if it is never used.)

PENDULUM GRAPHICS: Handle Graphics and `drawnow`; Today's Setup

Matlab's uses an elaborate object-oriented setup to draw its plots. Every element of a figure is an object; using the "handle" associated with any given element, we can modify its properties in our scripts and functions. To update the current on-screen view after making some changes, we give the command `drawnow`.

For example, suppose we say

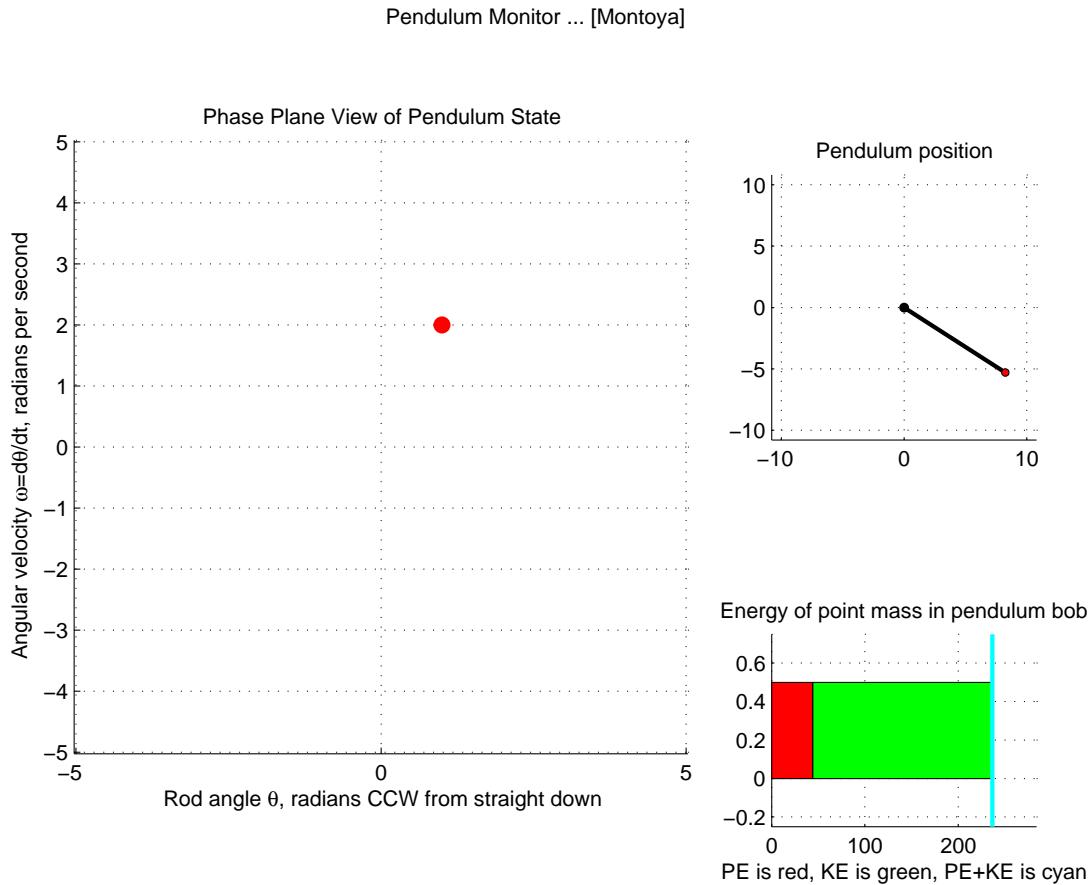
```
dothandle = plot(0,0,'r.');
```

This puts a red dot at the origin, and returns a handle to that object. Saying `get(dothandle)` will reveal a long list of property/value pairs, each of which we can change. Two interesting properties are named `XData` and `YData`: these tell the dot's current location. To move the same dot to (2,1), it suffices to say

```
set(dothandle,'XData',2,'YData',1);
drawnow;
```

To produce a figure where the dot seems to move, we just need a loop that alternately sets new coordinates and then refreshes the view by giving commands like these. To regulate the speed of the update process, we can insert a command like `pause(0.1)` in the loop. Generally, `pause(t)` will pause for approximately t seconds, so `pause(0.1)` will give about 10 updates per second.

In support of this lab, two functions are provided on Connect. The first one, `newpendulum`, creates a new figure window containing three subplots that show the phase plane, a sketch of the pendulum itself, and an energy monitor, like this:



To activate this function, say

```
hlist = newpendulum('Pendulum Monitor ... [Montoya]');
```

The input argument provides the text for the figure window: you will want to change Inigo Montoya's name to your own, and choose a more meaningful title. The output variable, here `hlist`, will be assigned a collection of graphics handles, one for each of several elements in the new figure. Any other script or function with access to these handles can make changes to the figure by using the `set` command as described above. Reading the code in `newpendulum.m` may be educational, but *it should not be necessary to make any changes to this file during today's lab.*

The second given function, named `newstate`, responds to commands like this:

```
newstate(hlist,theta,omega);
```

The first input is the cell array of graphics handles produced by `newpendulum`. Then the real scalar variables `theta` and `omega` provide the values for a pendulum angle θ and angular velocity ω to be shown in the monitoring window. Knowing just these two scalars is enough to update (i) the

position of the model pendulum, (ii) the values of the kinetic and potential energy, and (iii) the current point in the (θ, ω) -phase plane. The handles that function `newstate` takes in through the parameter `hlist` allow it to address the relevant elements of the pendulum monitoring window and make all the required changes. *It will be necessary to make some changes to this file during today's lab.*

Details of the cell array that `newpendulum` builds for the use of `newstate` can be found by reading the definitions of these two functions. This is encouraged, of course, but one entry will be particularly important during the lab: to get it, say

```
phaseplane = hlist{2};
```

Now `phaseplane` is the handle for the object of type “axes” in which the monitoring figure shows the (θ, ω) -phase plane. By giving the command `axes(phaseplane)`, we can activate that subplot in the pendulum figure to receive plotting commands. This will be the key to drawing a velocity field during the lab period.

Every time we call `newpendulum`, we get a new figure window and a new cell array for the objects in that window. So, by carefully giving distinctive names to each list of handles we receive, we can easily have two or more pendulum windows open and accessible at the same time.

Matlab has a whole system for preparing stand-alone animations that can be shown without recomputing each video frame in real time. This is nice to know, but not applicable in today's lab.

LAB ACTIVITIES AND DELIVERABLES

1. Get set up and play around. There is nothing to hand in here, but the experience will help later. Allow up to 30 minutes for this.
 - Download `newpendulum.m`, `newstate.m`, and `pendtemplate.m` from Connect. Open these files and look through them. Run the script `pendtemplate`.
 - Adjust `pendtemplate` to replace the figure title it produces with more descriptive text and your own name.
 - Edit `pendtemplate` to improve the animation by showing 90 frames for each complete cycle instead of just 12. If you can think of a way to produce many more frames and simultaneously remove many lines of code, please do! You may then wish to adjust the variable `fps` (frames per second) to improve your viewing experience.
 - As supplied, `pendtemplate` uses $C = 1$ in the functions

$$\theta(t) = C \cos(t), \quad \omega(t) = -C \sin(t).$$

These are the components of the approximate equation (‡), but the maximum swing of $C = 1$ rad is not a very “small angle”. Re-run the script with various choices for the amplitude C . Observe, experience, and remember:

- When C is small, everything is pretty believable.
- For C around 2, the animated picture of the swinging mass seems pretty credible, but the energy monitor shows that something is not quite right.
- For $C = \pi - 0.3$, periodic variations in the total energy are getting worse, and even the animation seems a little strange.
- For $C = \pi + 0.3$, the animation is clearly absurd.

Let’s be clear: The real motion of the pendulum must conserve total energy. We see the total energy meter vary because we are not animating a physically realistic motion. (But without the energy-meter, it’s a little hard to see what’s wrong with the moving pictures even when $C \approx 2$. Designers of video games and special effects in movies are actively researching ways to trade worse physics for faster rendering, subject to the requirement of no loss of believability for the viewer.)

2. Make a new script based on `pendtemplate.m` that enhances the pendulum-monitor window for the linear approximation case as follows:
 - Draw a velocity field into the box where the phase plane appears. To gain access to that box, give the command `axes(hlist{2})`, where `hlist` is whatever variable you have chosen to receive the output from `newpendulum`. Get the velocity vectors from line (‡), using the special parameter choices described above; make the velocity field the same way you did in Computer Lab 5.
 - Modify the function `newstate` so that instead of moving the dot marking the state to some new location, it leaves behind a short line segment connecting the old location to the new one. With this change in place, running the animation will appear to draw a picture of the phase-plane trajectory of the ODE system.
 - Retain or re-import whatever enhancement you made to the number of frames to show per cycle in Activity 1. The pendulum trajectories in phase space should look like the

nice smooth circles they really are, even though the shapes we are actually drawing are many-sided polygons.

- Adapt your replacement for `pendtemplate` so that it will trace several different phase-plane trajectories, corresponding to various choices for the parameter C . Suggested values, in Matlab notation: `C = 0.5:0.5:5`.

With these enhancements in place, your script should produce a figure that contains useful information even when frozen in time by printing it. Make sure the title mentions your name and print a copy to hand in.

3. Get the script `threeplots.m` from Connect. As provided, this makes three separate figures, each showing the graph of $\theta_{\text{lin}}(t)$ in the top pane and the graph of $\omega_{\text{lin}}(t)$ in the bottom pane. Except for the amplitudes, these figures are identical.

Modify the given script so that it uses `ode45` to solve the nonlinear pendulum equations (***) and determine the true values of $\theta(t)$ and $\omega(t)$ at the time instants t already being used to plot the linear graphs. Then, have the script overlay the true $\theta(t)$ and $\omega(t)$ plots on top of the approximate graphs as provided. Draw the true curves in fat lines with a contrasting colour.

Your work should produce three plots to hand in, showing the relationship between the true motion and its linear approximation for each set of initial conditions listed below:

- (i) $\theta(0) = 1 \text{ rad}$, $\omega(0) = 0 \text{ rad/s}$
- (ii) $\theta(0) = 2 \text{ rad}$, $\omega(0) = 0 \text{ rad/s}$
- (iii) $\theta(0) = 3 \text{ rad}$, $\omega(0) = 0 \text{ rad/s}$

4. Use your skills from Activity 3 to produce an accurate alternative to the phase plane sketch from Activity 2. In detail, make a fresh copy of your script from Activity 2, and improve it as follows.

- Change the velocity field from the approximate velocities in (‡) to the true velocities in (**).
- Use pre-computed values of $\theta(t)$ and $\omega(t)$ from `ode45` to control the animation, instead of simple trigonometric formulas.
- Make plenty of computed trajectories. Start some with $\omega(0) = 0$ and $\theta(0)$ from the Matlab-style vector `0.5:0.5:3.0`. Then start some others with $\theta(0) = -\pi$ and $\omega(0)$ chosen from `0.5:0.5:3.0`. Then think carefully about what changes to make, if any, to fill the whole window with representative trajectories.

Once you have a good phase portrait of the nonlinear pendulum, showing plenty of representative trajectories in every region, print it and hand it in.