PIC 16, Winter 2018 – Assignment 9M

Assigned 3/5/2018. Code (a single `.py` file) due by the end of class 3/9/2018 on CCLE. Hand in a printout of this document with the self-assessment portion completed by the end of class on 3/9/2018.

In this assignment, you will solve the equation of motion for a simple pendulum.

Task

Similar to Newton's familiar second law of motion:

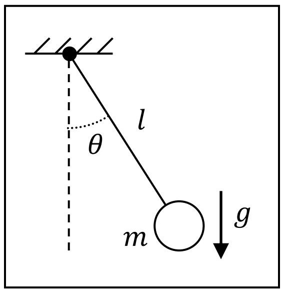
$$F = ma,$$

where F is the sum of all forces on a body, m is the inertia (mass) of the body, and a is the acceleration of the body, is a simplification of Euler's second law of motion

$$T = I\alpha,$$

where T is the sum of torques about a body, I is the rotational inertia ("moment of inertia") of the body, and α is the rotational acceleration.

Consider the simple pendulum of mass m suspended by a (massless) rod of length l from a frictionless pivot.



Gravity produces a force on the mass of magnitude mg. As the pendulum swings, its orientation at any instant t can be described by the angle $\theta(t)$ between the rod and the direction of gravity (measured positive counter-clockwise by convention). The distance between the mass and the pivot in the direction perpendicular to the force is therefore $l \sin\theta(t)$. The torque about a point is defined (in two dimensions) as the product of a force and the distance perpendicular to the force between the point and the location where the force acts. Therefore, the (only) torque about the pivot is $T = -mgl\sin\theta(t)$. The sign is negative because the torque acts clockwise, opposite our definition of positive angle.
The moment of inertia about a point is defined (in two dimensions) as the product of mass and the distance between the mass and the given point squared; therefore the moment of inertia of the pendulum about the pivot is $I = ml^2$.
Finally, the angular acceleration of the pendulum is simply $\alpha = \ddot{\theta}(t) = \frac{d^2\theta}{dt^2}$, the second derivative of the angle with respect to time.
Substituting these results into Euler's second law yields the equation of motion of the pendulum:

$$-mgl\sin\theta(t) = ml^2\ddot{\theta}(t)$$

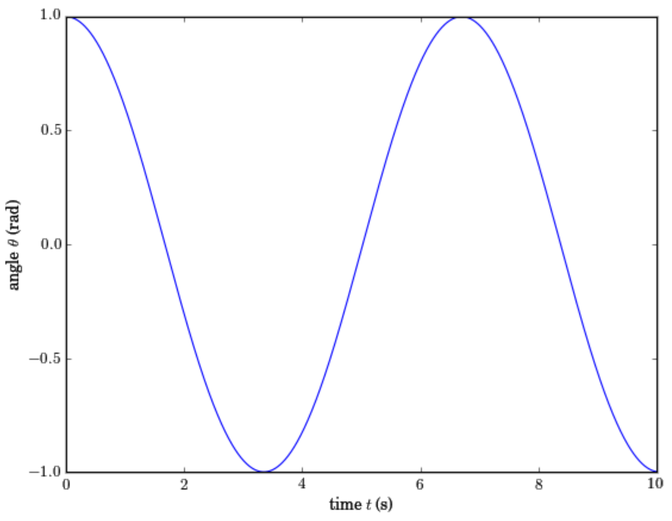
Your task is to solve this second-order differential equation numerically for the angle of the rod from time $t = 0$ to t = 10 from initial conditions $\theta(t = 0) = 1$ (radian) and $\dot{\theta}(t = 0) = 0$ (radians per second) given $m = $ 1kg, $l = $ 1m, and $g = $ 1m/s^2. Plot the response $\theta(t)$.

Self-Assessment

Write the second-order equation of motion as a system of two first-order equations. That is, using y_1 and y_2 to refer to θ and $\dot{\theta}$ (respectively) and using p_1, p_2, and p_3 to refer to m, g, and l (respectively), write equations for $\dot{y}_1$and $\dot{y}_2$ in terms of y_1, y_2, p_1, p_2, and p_3.

Did you write a Python function representing this first-order system of equations?

Were you able to use `odeint` to solve it correctly? If so, a plot of θ vs t would look like:



Lots of (Potentially Redundant) Hints

Unfortunately, SciPy does not have a function for solving second-order differential equations directly; it only has a solver for systems of first-order equations. This is not unusual in numerical computing because *n*th order differential equations can easily be reduced to a system of *n* first-order differential equations. Therefore, to solve our problem, we must transform the equation of motion into a system of two first-order differential equations and apply SciPy's first-order equation solver, `scipy.integrate.odeint`. This was discussed in the preparation. If you need help, ask!

When we call `odeint` , we typically need to provide four arguments:

- `f` – a *function* that encodes the system of first-order differential equations to be solved (as discussed below),
- `y0` – a list of initial conditions, that is, the values of the "*state variables*" (in our case, the angle and angular velocity of the pendulum) at the start of our simulation,
- `t` – a list (or NumPy array) of times, beginning with start time of our simulation, at which we want to know the values of the state variables, and
- args – additional parameters needed by our function `f` (discussed below).

`odeint` returns `y`, a NumPy array of the values of the state variables for all the times we specified in `t`. This is the (numerical) solution of our differential equation since it contains the angle of the pendulum through time, that is $\theta(t)$ for a range of values of t.

Where do the arguments we provide come from? The list `y0` is part of the problem we're trying to solve. If we want to know the angle of the pendulum through time, we must have in mind some angle and angular velocity it's starting from. (If we ask for directions to a destination, we had better specify the origin….) Likewise, the duration of time for which we want the angle is part of the problem statement, and we can choose the number of time points within this time interval for which we want the angle in order to produce, say, a smooth plot of the angle through time. These determine the array `t`.

The function `f` that "encodes" the system of differential equations is only a little less apparent. `odeint` is expecting the function to be of the form:

$$\dot{y} = f(y, t, p)$$

That is, we must write a Python function that accepts three arguments (provided by `odeint`):

- y – a list (or tuple, NumPy array, etc…) of *state* (position and speed, in our case) variable values. For us, this means instantaneous values of the pendulum angle θ and angular velocity $\dot{\theta}$
- t – a time value (the time at which the system is in the given state)
- p – a list of any additional parameters your function might need to do its job (below)

How does `odeint` choose the values of the arguments to `f`? We actually provided `odeint` with a list of parameters, `args`; and it passes them directly into our function as p. We chose the initial time `t[0]` and state `y0`, so `odeint`'s first call to our function will be with these values. Our function returns $\dot{y}$, the time derivative of the state of the system, from which `odeint` can approximate y a short time later like $y(t+\Delta t) = y(t) + \dot{y}\Delta t$ (or using a more accurate formula). `odeint` repeats this process until it can produce an approximation of y at all the times we requested with `t`.

Let's return to the function `f` itself. `odeint`, a function that is part of SciPy, is going to call `f`, a function provided by us. `odeint` will use it to calculate the numerical solution to the differential equation $y(t)$. In order for `odeint` to do its job, our function must return $\dot{y}$, a list containing the time derivative of each state variable, calculated according to the equations of motion given the provided current state variables y, time t, and parameters p. In this case, our function must return a list containing the angular velocity $\dot{\theta}$ and angular acceleration $\ddot{\theta}$ given the current state θ and $\dot{\theta}$, time t, and parameters m, g, and l.

Note that if we choose to order the list of state variables y as $[\theta, \dot{\theta}]$, our function needs to return the list of derivatives $\dot{y}$ in the corresponding order $[\dot{\theta}, \ddot{\theta}]$. We can choose any order to pass in the values of the state variables; `odeint` doesn't care. All that matters is that we return the list of corresponding time derivatives in the corresponding order.

How do we calculate the values to return? We passed in the angular velocity $\dot{\theta}$ (and θ) as part of the list y; we just need our function to spit that back out – albeit in different position of the output list. (Does this seem silly? It's not, really. This is how we transform our second-order differential equation into a system of two first-order differential equations.) And to calculate $\ddot{\theta}$, we just need to solve the equation of motion for $\ddot{\theta}$ in terms of the other variables.

What do we do with the time variable t passed into our function? Nothing, in this case, since time does not appear explicitly in the equation of motion. (This may be confusing as $\theta(t)$ and $\ddot{\theta}(t)$ appear in the equation, but these just means that equation of motion relates instantaneous values of θ and $\ddot{\theta}$ at *any* given time. The calculation of $\ddot{\theta}$ given a value for θ doesn't actually depending on the particular time. If our equation were $\ddot{\theta}(t) = \frac{\theta(t)}{t^2}$, that would be a different story, and that's why the general solver `odeint` expects it – for more general problems.)

And what about the parameters p? We could have hard-code the values for m, g, and l in our function, but we make it more general by getting these values out of the list p. That way we don't have to change the function even if we want to simulate a somewhat different pendulum. We tell `odeint` what `args` it should pass into `f` when we call `odeint`, and that's where p comes from.

That's about all there is to say about using `odeint` without actually writing the code for you. I hope this helps, but if there's a point you're confused about, please ask!