IS Physics Problem Set 1

Hrishi Olickel

27-08-2015

1 Part a

Let us consider the system at the point where we diverge using the small angle approximation:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l} sin\,\theta = 0 \tag{1}$$

It is at this point that we would assume $sin\theta \approx \theta$. But we can still make the assumption that the function for theta would resemble a sine curve, and that it relates to its amplitude, θ_0. Therefore,

$$\theta = A\,sin(\theta_0 t) \tag{2}$$

$$\frac{d\theta}{dt} = A\,cos(\theta_0 t) \tag{3}$$

$$\frac{d^2\theta}{dt^2} = -A\,sin(\theta_0 t) \tag{4}$$

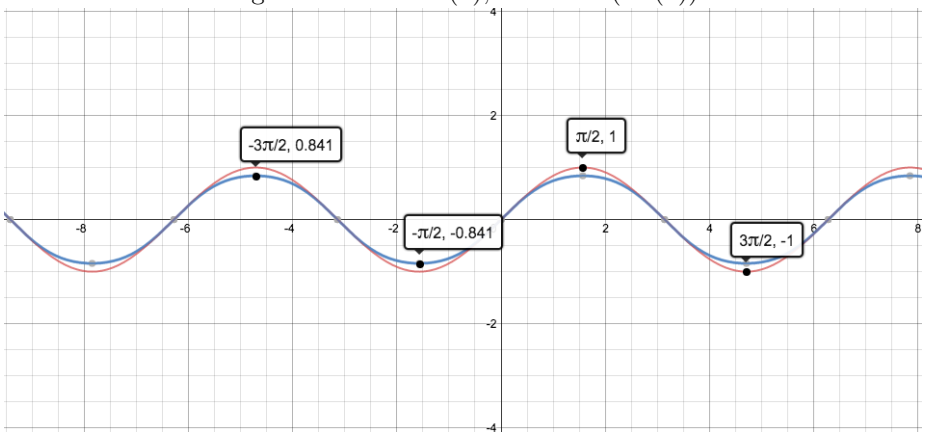
Substituting (4) and (2) into (1), we get

$$A\,sin(\theta_0\,t) + \frac{g}{l} sin(A\,sin(\theta_0\,t)) = 0 \tag{5}$$

$$\therefore A\,sin(\theta_0\,t) = -\frac{g}{l} sin(A\,sin(\theta_0\,t)) \tag{6}$$

This is a considerably difficult equation to solve, so let us see what the relationship between the two variable terms here are. Let us graph sin(x) and sin(sin(x)) on the same axes:

Figure 1: Red - sin(x), Blue - sin(sin(x))



Solving this equation means arriving at a solution for the constants θ_0 and A that would make these two graphs equal, in a sense. But looking at the graph we

can see that a simple solution wouldn't be possible, and that an approximation is the best option. Here we rest, hoping for the advent of computers.

2 Part b

Without modifying the original code for oscillator.m, we can easily modify the calculation functions to solve for a simple pendulum:

Firstly, calcForce is modified to use the equation of restoring force for a simple pendulum:

```
function F = calcForce(m, x)
    % Take current x coordinate and return force acting
        on mass
    F = -m*9.81*sin(x);
end
```

Next, calcAccel is modified to include the length parameter for the pendulum, for conversion from angular velocity to velocity:

```
function aNow = calcAccel(F, mass, length)
    % Take force and mass and return acceleration based
        Newtons second law
    aNow = F/(mass*length);
end
```

The other elements are left the same as they are, with the understanding that variables of x now hold values of theta. A damping constant *dampC* is added to adjust damping, and damping is introduced in the for loop show below (only relevant code is shown for the sake of brevity[1]:

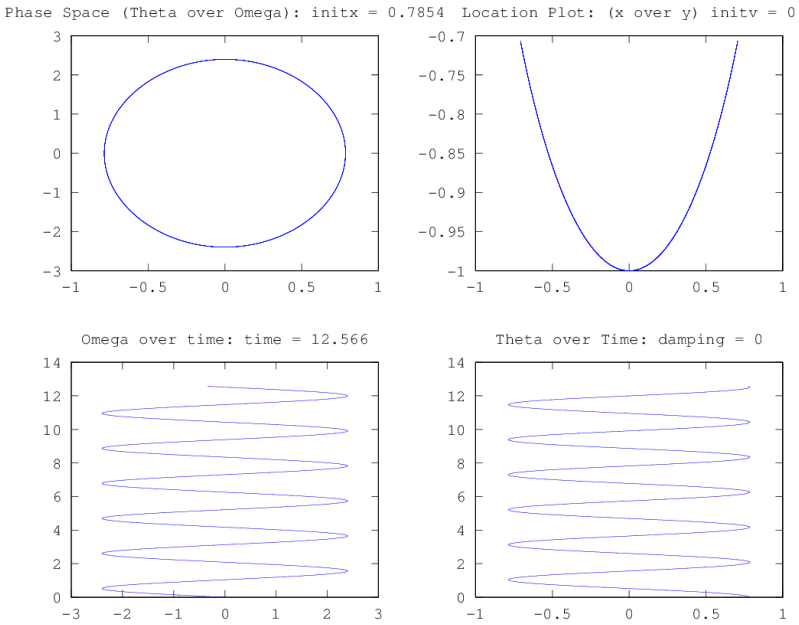
```
    %%% Initial conditions %%%
    initX = pi/4;
    initV = 5;
    mass = 1;
    length = 1;
    dampC = 0;

    vNow = initV;
    xNow = initX;
    timeNow = 0;

    %%% Time handling %%%
    totalTime = 10*2*pi; %Total time to simulate
    N = 10000; % Number of simulation steps (more = more
        accuracy)
```

[1]The full code for oscillator.m can be found at `http://pastebin.com/zumTEJWj`.

```
15      delT = totalTime/N;  %Time for each time step
16
17      %%% Arrays to store time points and associated x
            values %%%
18      tSteps = [];
19      xSteps = [];
20      vSteps = [];
21      dispSteps = [];
22      pxSteps = [];
23      pySteps = [];
24
25      %%% main "for" loop - do N steps if simulation %%%
26      for step = 1:1:N,
27          fNow = calcForce(mass, xNow);   % calculate force
                at point x
28
29          damping = dampC*vNow; %calculate damping
30          fNow  = fNow - damping; %adjust force accordingly
31          aNow = calcAccel(fNow,mass,length);   % calculate
                acceleration at point x
32
33          xNow = xNow + calcDeltaX(vNow,aNow,delT);   %
                update x (location)
34          vNow = vNow + calcDeltaV(aNow,delT);   % update
                velocity
35          timeNow = timeNow+delT;     % update time
36
37          tSteps = [tSteps, timeNow];  % store current time
                for plotting
38          xSteps = [xSteps, xNow];  % store current
                location for plotting
39          vSteps = [vSteps, vNow];
40          dispNow = length*xNow;
41          dispSteps = [dispSteps, dispNow];
42          pxSteps = [pxSteps, length * sin(xNow)];
43          pySteps = [pySteps, -(length * cos(xNow))];
44
45      end
```

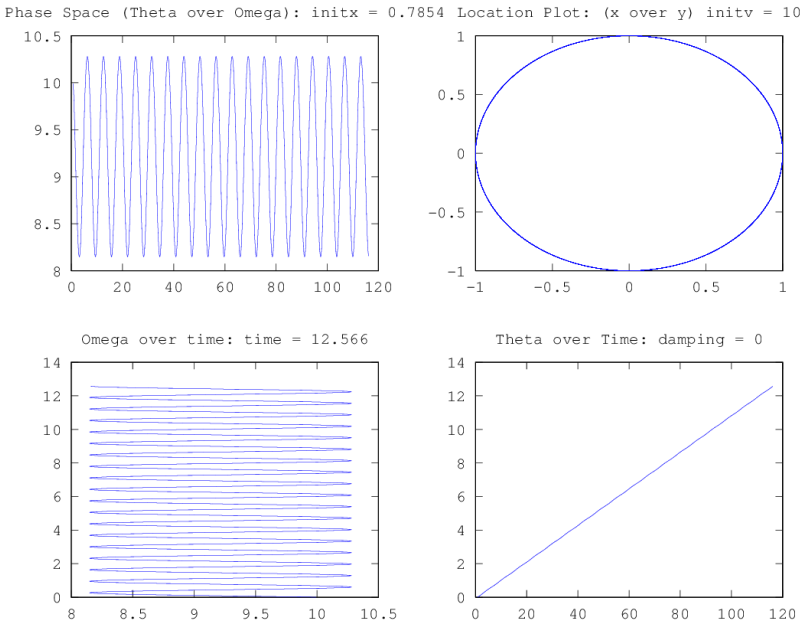
Without damping, the following plot is produced:

Figure 2: Time Simulated: $2 \times \pi$, mass=1, length=1



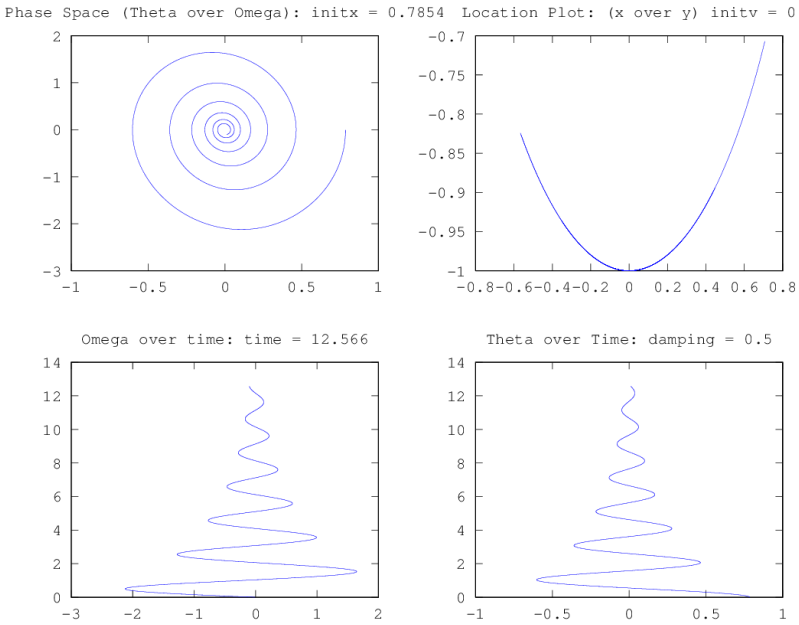
For very high initial velocity, the pendulum goes over the top (the location plot is now a complete circle):

Figure 3: Time Simulated: 2, mass=1, length=1



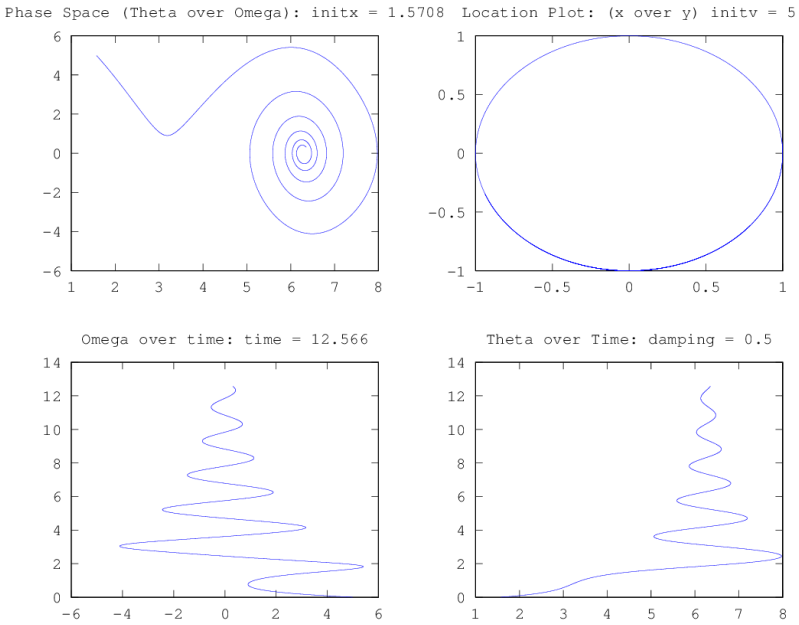
We can also see that the movement does not settle into a single phase space. Now, with damping:

Figure 4: Time Simulated: $2 \times \pi$, mass=1, length=1



We can see the phase space spiral into the center. What if we were to increase the initial velocity and position so it goes over the top?

Figure 5: Time Simulated: $2 \times \pi$, mass=1, length=1

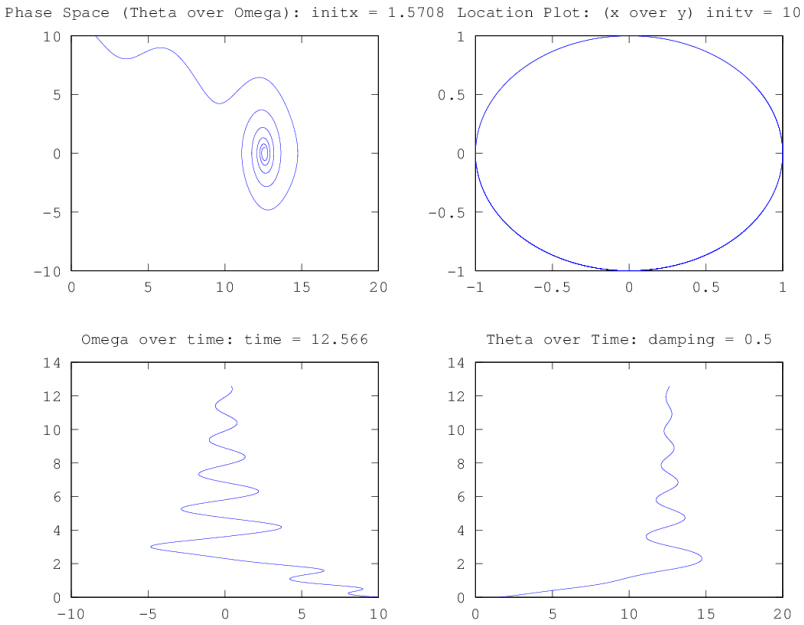


We can see that the pendulum goes over the top once, but settles into the next stable state where damping eventually leads it to rest.

An even higher velocity shows the pendulum moving to different possible loops in phase space before eventually not having enough energy and begin oscillating.

This is a good example of where the small angle approximation would not work very well. The error term is compounded with each rotation instead of cancelling out when θ is zero. After a few rotations the estimates are very far removed from observational or numerical results.

Figure 6: Time Simulated: $2 \times \pi$, mass=1, length=1



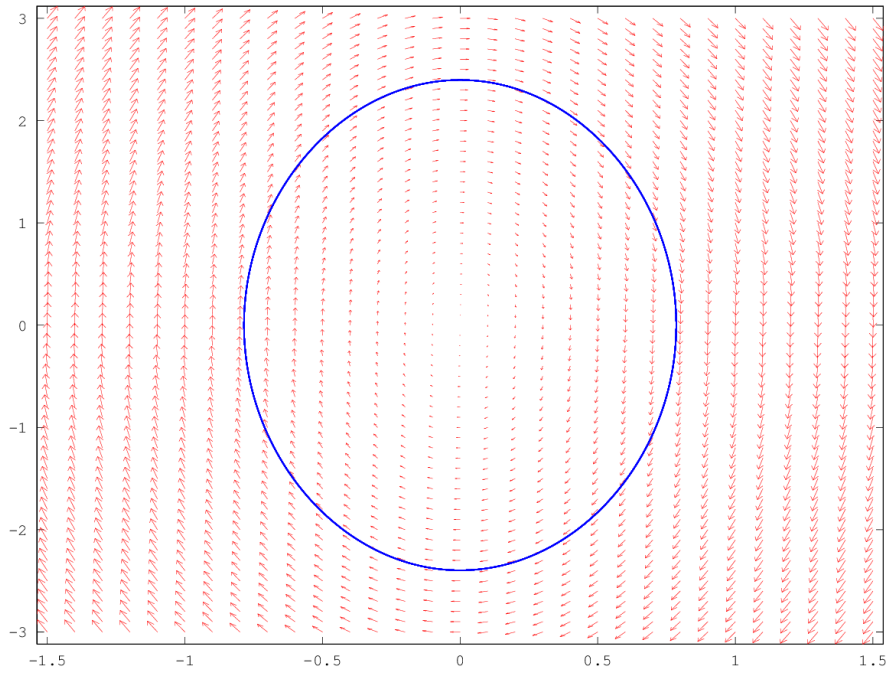
3 Part C

The input function xdot[2] was created for plotting the vector field:

```
function xdot = f(x,t)
      dampC = 0.5;
      % x(1) is theta, x(2) is omega
      length = 1;
      mass = 1;
      f = -mass*9.81*sin(x(1))-(dampC*x(2));
      xdot(2) = f/(mass*length);
      xdot(1) = x(2);
    endfunction
```

When solved using lose and plotted using the vector field, a pendulum without damping can be seen:

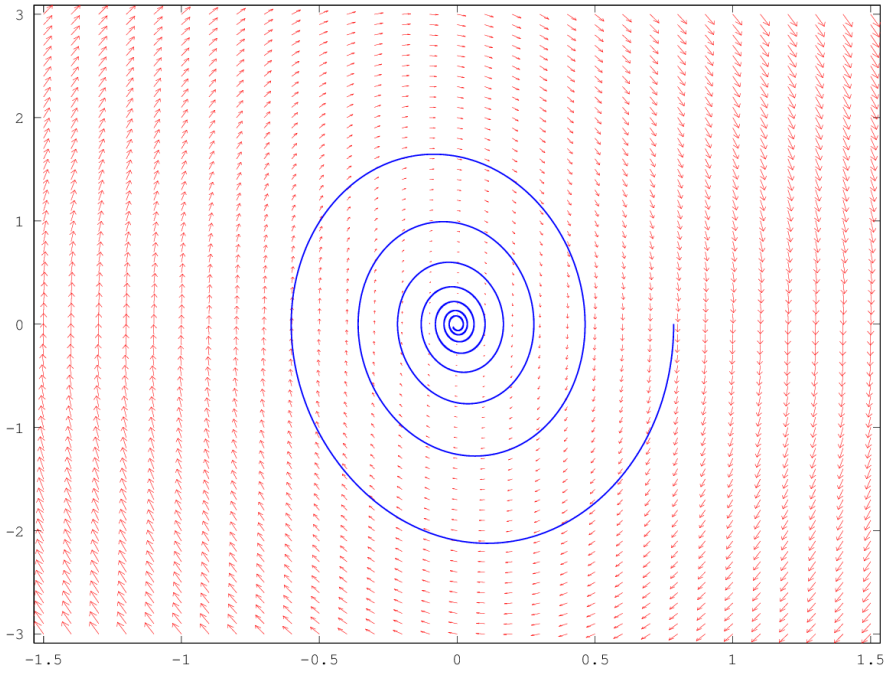
Figure 7: Time Simulated: $20 \times \pi$, mass=1, length=1, initX=$\pi/4$, initV=0, damping=0



With damping:

[2]The complete code for the vector field plotter (plotGraphs.m) can be found at `http://pastebin.com/f01GbgMw`

Figure 8: Time Simulated: $20 \times \pi$, mass=1, length=1, initX=$\pi/4$, initV=0, damping=0.5



Careful observation points out how damping affects the vector field. An increase in damping reduces the the length of vectors as we move away from the origin. This decrease in strength causes the trajectory to spiral inward.

4 Part D

The ode solver was used to plot the trajectory on top of the vector field in part C. The code is using the same function - xdot - used in part C, and the ODE solution can be obtained using a single line of code:

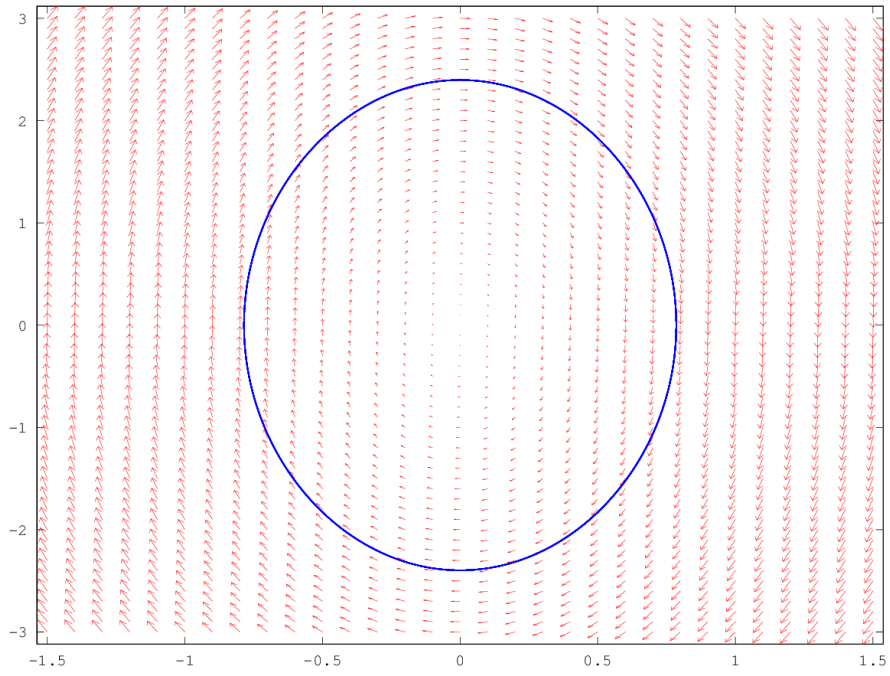
```
x = lsode("f", [pi/4;0], (t = linspace(0,15,20000)'));
```

It is notable that the ODE solution is obtained much faster than Part B and especially Part C due to the adaptive time stepping of the solver. Part C is extremely intensive in terms of computing power required, as the nested loops scale in complexity by $O(n^2)$. In our case, the solver returns the value of 2, indicating an absence of roots, but that the entire timespace was solvable.

5 Part E

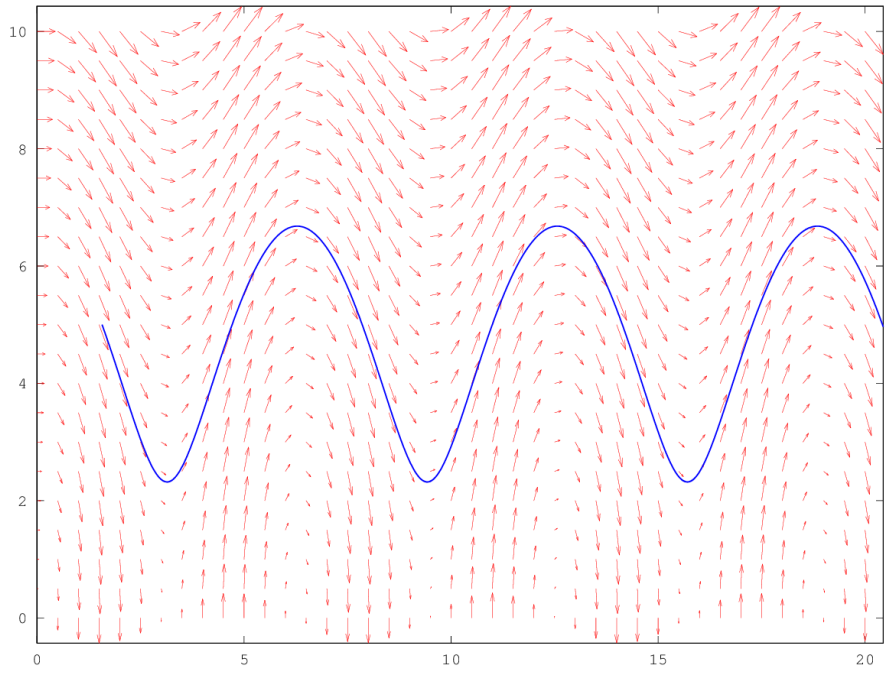
We have already looked at open and close trajectories for the pendulum, but let us plot a few trajectories along with the vector fields. The following is a closed trajectory, where the pendulum oscillates back and forth without damping:

Figure 9: Time Simulated: $20 \times \pi$, mass=1, length=1, initX=$\pi/4$, initV=0, damping=0



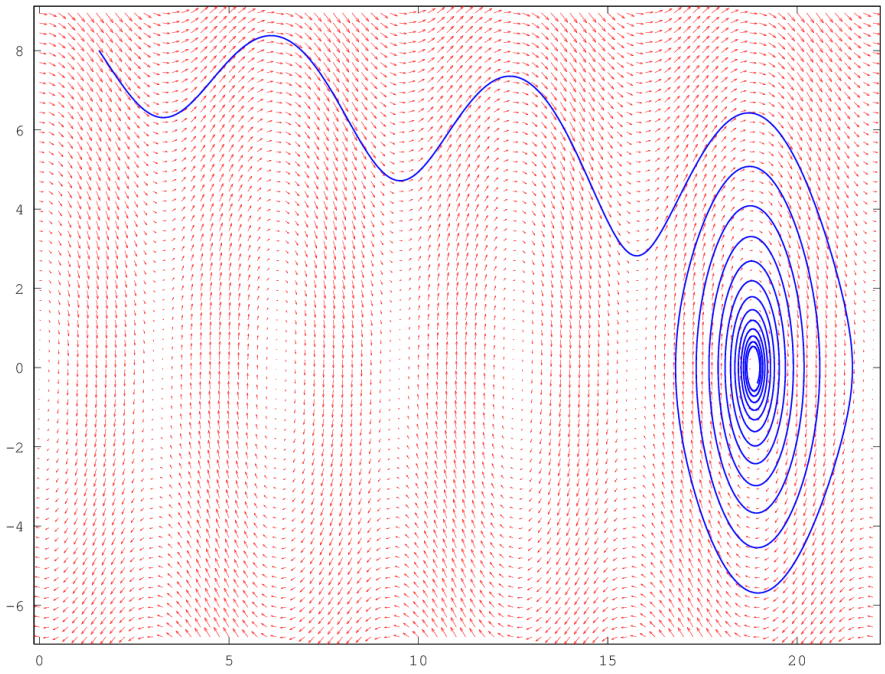
Without any damping, if we increase initial angle and velocity so that the pendulum were to go over the top, it would simply keep rotating forever:

Figure 10: Time Simulated: $20 \times \pi$, mass=1, length=1, initX=$\pi/2$, initV=5, damping=0



If we add damping, we can see similar behaviour as before, where the pendulum rotates until it doesn't have energy to keep rotating. It then returns to oscillation and eventually rest.

Figure 11: Time Simulated: $20 \times \pi$, mass=1, length=1, initX=$\pi/4$, initV=0



We can see that damping is removing energy from the pendulum with each rotation. With each rotation, it moves lower and lower in phase space until it crosses the separatrix, at which point it enters a new configuration, one that spirals toward the center. If damping were removed at this point, it would oscillate forever.

In this case, the separatrix is the exact angular momentum above which the pendulum rotates, and below which the pendulum oscillates. There is also a third condition, where the pendulum has just enough angular momentum to reach the top, and enters an unstable equilibrium, resting at the very top of it's motion. This, unfortunately for me, has been impossible to simulate (although I suspect an analytical solution can be found).