Damped Pendulum — Phase Space

Done in class, February 7, 2025

This is the sixth notebook for you to finish in-class.

Forced Oscillation

Force and Acceleration

```
gravity = 9.80665; (* the value of gravity in units of meters / second-squared *)
length = 0.24840;
(* A pendulum whose length is 9.7795 inches converted to meters  *)
(* The natural frequency and period of such a pendulum,
provided you do not swing it too wildly is: *)
omega0 = Sqrt[gravity / length];
gamma = 0.03;
(* A real pendulum swinging in air typically has a small gamma. *)
period = 2 Pi / omega0;
(* The length was chosen so that the period is 1 second. To be *)
(* precise, 2 Pi / omega0 = 0.999989,
and 2 Pi / Sqrt[omega0^2-gamma^2] = 1.000000. *)
alpha[t_, theta_, omega_] := - (gravity / length) Sin[theta] - 2 gamma omega;
```

Simulation Parameters

```
In[•]:= tInitial = 0.0;
tFinal = 50.0;
steps = 20000;
deltaT = (tFinal - tInitial) / steps;
```

Initial Angle and Angular Velocity

Let’s let the pendulum be initially held still at 10° and gently released:

```
In[541]:=
thetaInitial = 10 °;
omegaInitial = -gamma thetaInitial; (*
gamma is small, and this is only 0.3° / second. *)
(* Putting in the small initial velocity
 makes things work out more tidily later. *)
initialConditions = {tInitial, thetaInitial, omegaInitial};
```

General Second-Order Runge-Kutta — Implementation

The implementation of the damped pendulum is almost the same is as the damped oscillator that you just completed. Paste in the code and figure out what needs to be changed.

```
lambda = 1;
rungeKutta2[cc_] := (
  (* Extract time, angle, and angular velocity from the list *)
  curTime = cc[[1]];
  curAngle = cc[[2]];
  curAngularVelocity = cc[[3]];
  (* Compute tStar, xStar, vStar *)
  tStar = curTime + lambda deltaT;
  xStar = curPosition + curOmega lambda deltaT;
  vStar = curOmega + a[curTime, curPosition, curOmega] lambda deltaT;
  (* Implement General Second-Order Runge-Kutta *)
  newTime = curTime + deltaT;
  newVelocity = curOmega + ((1 - 1/(2 lambda)) a[curTime, curPosition, curOmega] +
        1/(2 lambda) a[tStar, xStar, vStar]) deltaT;
  newPosition = curPosition + (curOmega + newVelocity) deltaT / 2;
  (* We are keeping track of the contribution
   to the acceleration due to the external force *)
  {newTime, newPosition, newVelocity, externalForce[newTime] / springConstant}
 )
N[rungeKutta2[initialConditions]] (* Test your rungeKutta2 function. *)
(* The output just below should be {0.004, 3.19999*10^-7, 0.00016, 0.0199999}. *)
```

Displaying Oscillation

Nest the procedure, transpose the results, and produce a plot of the angle θ as a function of time:

```
In[ ]:= rk2Results = NestList[rungeKutta2, initialConditions, steps];
        rk2ResultsTransposed = Transpose[rk2Results];
        times = rk2ResultsTransposed⟦1⟧;
        thetas = rk2ResultsTransposed⟦2⟧;
        thetaPlot = ListPlot[Transpose[{times, thetas}]];
        (* the theoretical solution is approximately known,
        provided the angle remains small *)
        (* let's plot the envelope of the theoretical solution *)
        envelopeFunction[t_] := thetaInitial Exp[-beta t]
        approximateTheoreticalEnvelope =
          Plot[{envelopeFunction[t], -envelopeFunction[t]}, {t, tInitial, tFinal}];
        Show[{thetaPlot, approximateTheoreticalEnvelope}]
```

In the preceding plot, the theoretical solution is approximately known, provided the angle remains small, and so I added the envelope of the theoretical solution to the plot.

Displaying Theory

In the following plot, I have included the theoretical oscillation, not just the envelope (but the same approximation just mentioned still applies):

```
In[ ]:= Plot[envelopeFunction[t] Cos[omega3 t], {t, tInitial, tFinal}]
```

Animating Oscillation

It’s also nice to have an animation, arranged so that the default duration of the animation is the actual duration of the animation:

```
In[ ]:= Animate[NumberLinePlot[thetas〚step〛, PlotRange → {-20 Degree, 20 Degree}],
         {step, 0, steps, 1}, DefaultDuration → tFinal - tInitial]
```

Wouldn’t it be even nicer to have our animation look like an actual pendulum?

Epilog — Phase Space