General Second-Order Runge-Kutta — Forced Oscillation

Done in class, February 4, 2025

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

Forced Oscillation

Problem Description

```
omega1 = 1;
externalForce[t_] := 100 Sin[omega1 t]
springConstant = 20;
dampingConstant = 1;
force[t_, x_, v_] := -springConstant x - dampingConstant v + externalForce[t]
mass = 5;
a[t_, x_, v_] := force[t, x, v] / mass;
tInitial = 0;
tFinal = 100;
steps = 25 000;
deltaT = (tFinal - tInitial) / steps;
```

Initial Conditions

Let’s let the spring be initially unstretched with no velocity and see what the external force does to it:

```
In[ ]:= xInitial = 0.0;
vInitial = 0.0;
(* We also want to be able to visualize the external force,
so let's tack it on to the quantities that will be tabulated by NestList *)
initialConditions =
  {tInitial, xInitial, vInitial, externalForce[xInitial] / springConstant};
```

General Second-Order Runge-Kutta — Implementation

The implementation will be almost the same as in the damped oscillation notebook you completed on Friday.

There are two small things that have to be changed. Figure out what they are and then if you still have that code handy, you can almost completely re-use what you did in that notebook. Or just write it out again. The more times you write it out, the better you will remember it.

```
α = 1; (* I am going to switch to nicer typography for the Greek lettter alpha,
here and you should too. You will see my reason in the next class;
we are running out of common letters and I will want to re-use some. *)
rungeKutta2[cc_] := (
  (* Extract time, position, and velocity from the list *)
  curTime = cc⟦1⟧;
  (* Compute tStar, xStar, vStar *)
  tStar = curTime + alpha deltaT;
  (* General Second-Order Runge-Kutta *)
  newTime = curTime + deltaT;
  (* We are keeping track of the contribution
   to the acceleration due to the external force *)
  {newTime, newPosition, newVelocity, externalForce[newTime] / springConstant}
 )
```

Displaying Forced Oscillation

Nest the procedure and then transpose the results to produce position and velocity plots:

```
In[ ]:= rk2Results = NestList[rungeKutta2, initialConditions, steps];
        rk2ResultsTransposed = Transpose[rk2Results];
        positionPlot = ListPlot[Transpose[rk2ResultsTransposed⟦{1, 2}⟧]]
```

```
In[ ]:= positions = rk2ResultsTransposed⟦2⟧;
        forces = rk2ResultsTransposed⟦4⟧;
```

```
In[ ]:= Animate[NumberLinePlot[{positions⟦step⟧, forces⟦step⟧}, PlotRange → {-50, 50}],
         {step, 0, steps, 1}, DefaultDuration → 40]
```

Conclusion / Commentary

Our oscillator now has the force law $F(x) = -20\,x - v$ *and in addition a sinusoidal external force, often called a driving force.*

You will remember that in the conclusion of the previous notebook I defined ω_0 and γ. We also have the driving frequency **omega1**. Now you see why I put the subscript “0” on ω in the previous notebook. We now have three relevant frequencies in the damped, driven harmonic oscillator, and we need to keep them all straight:

ω_1	(the external or driving frequency)
ω_0	(the natural frequency of the oscillator in the absence of damping)
γ	(the frequency that controls the rate of decay of the exponential)

There is enough complexity here that this system is pretty complex to play around with. Once you all have it working, we’ll adjust the input parameters to get other values for these three frequencies.

Epilog — Resonance

Let’s display all three of the important frequencies for easy comparison:

```
omega0 = Sqrt[springConstant / mass];
gamma = dampingConstant / (2 mass);
{omega1, omega0, gamma}
```

$\left\{2, 2, \frac{1}{10}\right\}$

At minimum, one of the things we should do as part of playing around with this system is just to jack up ω_1 so that it is almost as large as ω_0. For example $\omega_1 = 1.8$ is good to try. That is “just below resonance.”

Then let’s go right on resonance, $\omega_1 = 2$. When the system is driven at resonance, it oscillates widely. It will help to make a smoother and easier-to-contemplate animation to jack up the number of steps to 50 000 and also to slow the animation duration down by doubling the animation duration: `DefaultDuration → 40`.

Seeing real systems being driven near resonance this was the traditional way of doing this before video and computing reduced the popularity of real lecture demonstrations, but I still find seeing videos of real systems helpful: https://youtu.be/aZNnwQ8HJHU.