Torsion Waves

Completed and Analyzed in class, March 21, 2025

This is the thirteenth notebook for you to complete. It bears strong similarity to our ninth notebook (Many Harmonic Oscillators).

Formulas for the Angular Accelerations

This angular acceleration formula

$$\alpha_j = -\omega_0^2(\theta_j - \theta_{j-1}) + \omega_0^2(\theta_{j+1} - \theta_j)$$

is valid for but the ends, but we have to handle those separately.

Fixed Ends

In the fixed-end case, the left-most rod's angular acceleration is

$$\alpha_1 = -\omega_0^2(\theta_1 - 0) + \omega_0^2(\theta_2 - \theta_1)$$

and the right-most rod's angular acceleration is

$$\alpha_n = -\omega_0^2(\theta_n - \theta_{n-1}) + \omega_0^2(0 - \theta_n)$$

Free Ends

In the free-end case, the left-most rod's angular acceleration is

$$\alpha_1 = 0 + \omega_0^2(\theta_2 - \theta_1)$$

and the right-most rod's angular acceleration is

$$\alpha_n = -\omega_0^2(\theta_n - \theta_{n-1}) + 0$$

Implementing the Angular Accelerations

In[20]:=
```
n = 72;
omega0 = 2 Pi;
(* The following is on the right track, but it doesn't work for the ends *)
wrongEquationsForα[j_, allθs_] :=
 -omega0^2 (allθs[[j]] - allθs[[j - 1]]) + omega0^2 (allθs[[j + 1]] - allθs[[j]])
(* A fancy person could probably handle the ends and whether or not they
 are free in one big equation, but let's build it up by cases *)

free = True;

freeα[j_, allPositions_] :=
 -omega0^2 If[j == 1, 0, allPositions[[j]] - allPositions[[j - 1]]] +
  omega0^2 If[j == n, 0, allPositions[[j + 1]] - allPositions[[j]]]

fixedα[j_, allPositions_] :=
 -omega0^2 (allPositions[[j]] - If[j == 1, 0, allPositions[[j - 1]]]) +
  omega0^2 (If[j == n, 0, allPositions[[j + 1]]] - allPositions[[j]])

α[j_, allPositions_] := If[free, freeα[j, allPositions], fixedα[j, allPositions]]
```

Initial Conditions

First set up the duration. Let's also define **steps** and **deltaT** while we are at it::

In[27]:=
```
tInitial = 0.0;
tFinal = 10.0;
steps = 2000;
deltaT = (tFinal - tInitial) / steps;
```

For now, just implement some simple-minded initial condition:

In[31]:=
```
pluck = -0.5;
(* Make pluck the second mass's displacement and all the others zero. *)
initialPositions = Table[0.0, n];
initialPositions[[2]] = pluck;
(* Make all the initial velocities zero *)
initialVelocities = Table[0.0, n];
initialConditions = {tInitial, initialPositions, initialVelocities};
```

Second-Order Runge-Kutta — Implementation

Your turn to put it all together into the real thing:

In[36]:=
```
rungeKutta2[cc_] := (
  curTime = cc[[1]];
  curPositions = cc[[2]];
  curVelocities = cc[[3]];
  newTime = curTime + deltaT;
  positionsStar = curPositions + curVelocities deltaT / 2;
  αs = α[#, positionsStar] & /@ Range[n];
  newVelocities = curVelocities + αs deltaT;
  newPositions = curPositions + (curVelocities + newVelocities) deltaT / 2;
  {newTime, newPositions, newVelocities}
 )
(* Test your implementation on the initial conditions *)
N[rungeKutta2[initialConditions]]
```

Out[37]=
```
{0.005, {-0.00024674, -0.499507, -0.00024674, 0., 0., 0., 0., 0., 0., 0., 0., 0.,
  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.},
 {-0.098696, 0.197392, -0.098696, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
  0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.}}
```

Using NestList[] to Repeatedly Apply rungeKutta2[]

In[38]:=
```
rk2Results = NestList[rungeKutta2, initialConditions, steps];
```

Transposing to Get Points We Can Put in ListLinePlot[]

In[39]:=
```
rk2ResultsTransposed = Transpose[rk2Results];
positionLists = rk2ResultsTransposed[[2]];
```