Second-Order Runge-Kutta — Mass on a Spring

Done in class, January 29, 2025

This is a third notebook for you to finish in-class. First let's recap and improve the last notebook which had a sinusoidal acceleration hard-coded into it.

Sinusoidal Acceleration Notebook — Recap

Problem Description

Our final version of the sinusoidal acceleration problem had $F = -10 \sin 2\, t$ and $m = 5$. With $t_{\text{initial}} = 0.0$ and $t_{\text{final}} = 3\,\pi$, we had three full periods of the sine function. We had 72 steps:

```
force[t_] := -10 Sin[2 t]
m = 5;
a[t_] := force[t] / m;
tInitial = 0.0;
tFinal = 3 Pi;
steps = 72;
deltaT = (tFinal - tInitial) / steps;
```

Initial Conditions

We chose $x_{\text{initial}} = 0$ and $v_{\text{initial}} = 1.0$.

```
xInitial = 0.0;
vInitial = 1.0;
aInitial = 0.0;
initialConditions = {tInitial, xInitial, vInitial, aInitial};
```

Update Procedure — Theory

For the sinusoidal acceleration notebook, the theory we implemented updated all three of time, position, and velocity as follows:

$$t_{i+1} = t_i + \Delta t$$

$$v(t_{i+1}) = v(t_i) + a\!\left(t_i + \tfrac{\Delta t}{2}\right) \cdot \Delta t$$

$$x(t_{i+1}) = x(t_i) + (v(t_i) + v(t_{i+1})) \cdot \tfrac{\Delta t}{2}$$

Update Procedure — Implementation

Below is an implementation of that is a bit better than the one we did in class. I added some parenthesis and within those parenthesis I am able to define variables before returning a result. This makes the implementation more readable.

In[120]:=

```
procedure[cc_] := (
  (* Extract time, position, and velocity from the list. *)
  curTime = cc〚1〛;
  curPosition = cc〚2〛;
  curVelocity = cc〚3〛;
  (* Implement the three equations in the theory above. *)
  newTime = curTime + deltaT;
  newVelocity = curVelocity + a[curTime + deltaT / 2] deltaT;
  newPosition = curPosition + (curVelocity + newVelocity) deltaT / 2;
  (* We don't need newAcceleration for the procedure,
  but it is nice to show acceleration as the first plot,
  so tabulate it too. *)
  newAcceleration = a[newTime];
  {newTime, newPosition, newVelocity, newAcceleration}
 )
```

By defining variables, it has become much clearer what each line of procedure does, and also made it less likely that we will have typos in getting the time, the position, and the velocity from a list.

Plotting the Sinusoidal Acceleration Results

We nested the procedure 72 times:

In[121]:=

```
results = NestList[procedure, initialConditions, steps];
```

We transposed the results to make it easier to take parts of them:

```
resultsTransposed = Transpose[results];
```

Then we plotted them.

```
accelerationPlot = ListPlot[Transpose[resultsTransposed〚{1, 4}〛]];
velocityPlot = ListPlot[Transpose[resultsTransposed〚{1, 3}〛]];
positionPlot = ListPlot[Transpose[resultsTransposed〚{1, 2}〛]];
```

You could display those as a column.

Mass on a Spring

Everything up to this point in this notebook was review (with slight improvements) of stuff we did in the last class. Finally we get to the new stuff.

Problem Description

In[126]:=

```
springForce[x_] := -20 x
mass = 5;
a[x_] := springForce[x] / mass;
tInitial = 0.0;
tFinal = 3 Pi;
steps = 7200;
deltaT = (tFinal - tInitial) / steps;
```

Initial Conditions

Let's stretch this spring to $x_{initial} = 25$ and let it go with no initial velocity, so $v_{initial} = 0.0$.

In[133]:=

```
xInitial = 25.0;
vInitial = 0.0;
aInitial = a[xInitial];
initialConditions = {tInitial, xInitial, vInitial, aInitial};
```

Second-Order Runge-Kutta — Theory — Summary

Because the force now depends on the position instead of the time, we had to revisit our procedure, and the procedure we ended up with is called second-order Runge-Kutta:

$$t_{i+1} = t_i + \Delta t$$

$$x^*(t_{i+1}) = x(t_i) + v(t_i) \, \Delta t$$

$$v(t_{i+1}) = v(t_i) + (a(x(t_i)) + a(x^*(t_{i+1}))) \cdot \frac{\Delta t}{2}$$

$$x(t_{i+1}) = x(t_i) + (v(t_i) + v(t_{i+1})) \cdot \frac{\Delta t}{2}$$

Second-Order Runge-Kutta — Implementation

```
rungeKutta2[cc_] := (
  (* Extract time, position, and velocity from the list. *)
  curTime = cc〚1〛;
  (* Implement the four equations in the theory above. *)
  (* So we can display it easily, tabulate acceleration too. *)
  newAcceleration = a[newPosition];
  {newTime, newPosition, newVelocity, newAcceleration}
 )
```

Displaying the Mass on a Spring Results

Nest the procedure 72 times and then transpose the results and produce three plots:

```
rk2Results = NestList[rungeKutta2, initialConditions, steps];
rk2ResultsTransposed = Transpose[rk2Results];
accelerationPlot = ListPlot[Transpose[rk2ResultsTransposed〚{1, 4}〛]];
velocityPlot = ListPlot[Transpose[rk2ResultsTransposed〚{1, 3}〛]];
positionPlot = ListPlot[Transpose[rk2ResultsTransposed〚{1, 2}〛]];
Column[{accelerationPlot, velocityPlot, positionPlot}]
```

Conclusion / Commentary

The plots above look mighty similar to the plots we got last time, but now we are getting them completely differently! Our idealized spring just obeys $F(x) = -10\,x$. Nowhere did we put sines or cosines into the problem. Oscillation emerged naturally.

It happens that the position and acceleration are following a cosine function, and the velocity is following a sine function. With different initial conditions you can switch these or make any mixture. Here is a video showing the bizarre fact that a mass on a spring has some resemblance to a particle moving in a circle: https://youtu.be/ZlleypTKfGY. I happen to know, using calculus, that the exact solution for the mass on a spring with the initial conditions that we chose is:

$a(t) = -100 \cos 2\,t$
$v(t) = -50 \sin 2\,t$
$x(t) = 25 \cos 2\,t$

You can see by studying the graphs that Mathematica has done a darned good job of using approximate methods to get close to the exact solution, and if you changed the number of steps to 720, or 7200, Mathematica would get even closer to the exact solution.