Diffusion in One Dimension

Completed and Analyzed in class, April 18, 2025

This is our twentieth notebook. In the last two we re-did guitar strings and drumheads using Mathematica's ability to solve wave equations.

Now I want to introduce you to a very different kind of problem, but one which has a lot of the same mathematics as wave equations. The problem is diffusion. An example would be, how does heat distribute itself through a piece of metal?

Diffusion — Theory

The fundamental idea of diffusion is concentration differences, and random motion that tends (on average) to even out the differences.

Consider a one-dimensional rod and let's have what is being diffused be the heat energy. To model this, I might break the rod up into chunks of length a, and if the chunks are sufficiently small, I can assign a temperature to each of the chunks. (Smallness is important because if a chunk is too big, assigning a temperature to the whole chunk might not be a good approximation because the temperature varies over the size of the chunk.)

In a one-dimensional chunk, if we assign an integer j to the chunk, then the chunk to the right of it might have integer $j + 1$ and the chunk to the left $j - 1$. There will be a temperature in each chunk: T_{j-1}, T_j, and T_{j+1}.

The amount of heat flowing from into

Let's be clear about the dependent and independent variables. The time is t. The guitar string is strung

along the x-axis. Those are the independent variables. The dependent variable is the displacement, which we are putting in the z-direction, so the dependent variable is z, and we want to find the function of two variables, $z(t, x)$.

Partial Derivatives and Their Notation

Now that we have two independent variables, we have to clarify for Mathematica which variable we are taking a derivative with respect to. In other words, the obvious generalization of

```
In[•]:= Derivative[2][z][t] // TraditionalForm
```

Out[•]//TraditionalForm=

$$z''(t)$$

which would be

```
In[•]:= Derivative[2][z][t, x] // TraditionalForm
```

Out[•]//TraditionalForm=

$$z''(t, x)$$

is ambiguous. Are we taking the derivative with respect to t or x? Mathematica specifies it this way:

```
In[•]:= Derivative[2, 0][z][t, x] // TraditionalForm
```

Out[•]//TraditionalForm=

$$z^{(2,0)}(t, x)$$

That's two derivatives with respect to the first variable, t. And here is two derivatives with respect to the second variable, x:

```
In[•]:= Derivative[0, 2][z][t, x] // TraditionalForm
```

Out[•]//TraditionalForm=

$$z^{(0,2)}(t, x)$$

You can even do mixed partial derivatives, such as one derivative with respect to t and one derivative with respect to x, but we have no need for that.

The Guitar String Differential Equation

Now that we know how the notation for how Mathematica expects us to specify partial differential equations, we can give it this differential equation:

$$\frac{\partial^2 z}{\partial t^2} = {V_0}^2 \frac{\partial^2 z}{\partial x^2}$$

Review the previous notebook to see how that was done in for the harmonic oscillator.

```
(* All the places you have work to do have variations on the *)
(* following earliest etymology of rootin' tootin': *)
(* “Well... Afore hoo’d bin here three days hoo’d hauve *)
(* a dozen colliers whewtin’ an’ tootin’ after her every neet.” *)
```

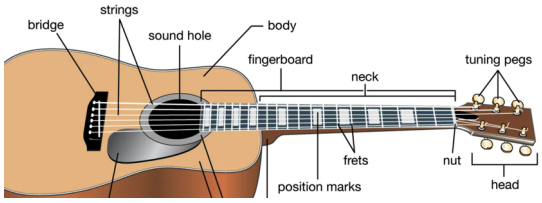
```
Module[{v0 = 1}, {rootin tootin}] // TraditionalForm
```

Out[•]//TraditionalForm=

$$\left\{z^{(2,0)}(t, x) = z^{(0,2)}(t, x)\right\}$$

Adding the Boundary Conditions

We are missing the boundary conditions on the string at the bridge and the nut of the guitar.



Put those in:

```
length = 1;
```

```
Module[{v0 = 1}, {Derivative[2, 0][z][t, x] == v0^2 Derivative[0, 2][z][t, x],
   colliers whewtin}] // TraditionalForm
```

Out[•]//TraditionalForm=

$$\left\{z^{(2,0)}(t, x) = z^{(0,2)}(t, x), z(t, 0) = 0, z(t, 1) = 0\right\}$$

Adding the Initial Conditions

We are also missing any specification of the initial motion of the string. It isn't just going to start vibrating by itself. Here is an initial displacement function:

In[•]:=
```
amplitude = 1;
mode = 1;
```

```
f[x_] := amplitude Sin[mode Pi x]
```

With mode=1, you have the "fundamental." With mode=2, you have the "first harmonic." With mode=3, you have the "second harmonic." Add f[x] as an initial displacement to the equations. Also add an initial velocity all along the string of 0. Give the resulting problem, which is finally completely specified, a name:

```
exactGuitarStringProblem = Module[{v0 = 1}, {every neet}];
```

Making Mathematica Exactly Solve the Problem

This problem has an exact solution. So we can use DSolve[] rather than NDSolve[]:

```
DSolve[exactGuitarStringProblem]
```

Out[•]=

```
{z[t, x] → Cos[π t] Sin[π x]}
```

We have the same problems as we had with the harmonic oscillator solution: (1) It is a list of lists of rules, and (2) we haven't given it a name so don't have a convenient way of using it elsewhere. Fix these problems:

```
exactGuitarStringSolutionRule = well afore hoo' d
```

Out[•]=

```
{z[t, x] → Cos[π t] Sin[π x]}
```

It is still a rule. Turn it into a function that we can plot:

In[•]:=

```
exactGuitarStringSolution[t_, x_] = z[t, x] /. exactGuitarStringSolutionRule
```

Out[•]=

```
Cos[π t] Sin[π x]
```

Plotting Exact Solution at $t = 1/4$

In[•]:=

```
Plot[exactGuitarStringSolution[1 / 4, x],
 {x, 0, 1}, PlotRange → {-1.1, 1.1}, AspectRatio → 0.1]
```

Animating the Exact Solution

In[•]:=

```
Animate[Plot[exactGuitarStringSolution[t, x],
  {x, 0, 1}, PlotRange → {Automatic, {-1.1, 1.1}}, AspectRatio → 0.1],
 {t, 0, 10}, DefaultDuration → 10, AnimationRunning → False]
```

Changing the Mode

Try changing the mode to some other number, like 3, and see what the second harmonic looks like.

Completely Different Initial Conditions

When a guitar string is plucked with something sharp, we can imagine that instead of an initial shape looking like,

$z(0, x) = \text{amplitude} \sin(m\pi x)$

instead we have, if $0 < x < b$,

$z(0, x) = \text{amplitude} \frac{x}{b}$

and if $b < x <$ length,

$z(0, x) = \text{amplitude} \frac{\text{length}-x}{\text{length}-b}$

Using Module[] with {b=1/4}, define a function g[x] that is the right-hand side, taking into account the differing behavior when $x < b$ vs. $x > b$:

```
g[x_] := Module[{b = 1 / 4}, amplitude If[hauve, a, dozen]]
```

Plot your function:

In[•]:=
```
Plot[g[x], {x, 0, 1}, AspectRatio → 0.1, PlotRange → {Automatic, {-1.1, 1.1}}]
```

Does that look like the way a pick might stretch the string (near the bridge or the sound hole) just before it lets it go?

It turns out Mathematica has a lot of trouble handling a solution with such a sharp kink. So now I will soften the kink a little:

In[•]:=
```
h[x_] :=
 Sum[(32 * Sin[(Pi * K[1]) / 4] * Sin[Pi * x * K[1]]) / (3 * Pi ^ 2 * K[1] ^ 2), {K[1], 1, 10}]

Plot[h[x], {x, 0, 1}, AspectRatio → 0.1, PlotRange → {Automatic, {-1.1, 1.1}}]
```

In[•]:=
```
numericalGuitarStringProblem =
  Module[{v0 = 1}, {Derivative[2, 0][z][t, x] == v0^2 Derivative[0, 2][z][t, x],
    z[t, 0] == 0, z[t, length] == 0, z[0, x] == h[x], Derivative[1, 0][z][0, x] == 0}];
```

Making Mathematica Numerically Solve the Problem

```
numericalGuitarStringSolutionRule =
 NDSolve[numericalGuitarStringProblem, z, {t, 0, 10}, {x, 0, 1}][[1]]
```

Convert the rule to a function:

```
numericalGuitarStringSolution[t_, x_] = hauve a dozen colliers
```

Plotting the Numerical Solution at $t = 1/8$

```
Plot[after her, PlotRange → {-1.1, 1.1}, AspectRatio → 0.1]
```

Animating the Numerical Solution

```
Animate[Plot[every neet, {t, 0, 10}, DefaultDuration → 10, AnimationRunning → False]
```

The fact that this isn't some simple sine wave is part of what gives the guitar its tone or timber. You can exaggerate this effect by picking the string very near the bridge.