Notes on simulating a vibrating string

1 Modeling continuous things

A vibrating string, as on a guitar or violin, operates on the same principles of elasticity as a Hookes law spring. The statement that the restoring force is proportional to the amount of stretch or compression is very general, after all. However, Hooke's law only describes springs that do not bend or flex – straight line segments with no structure or inertia connecting one end to the other. However, to model a violin string we have to do better than this, since the movement of the elastic body is the whole point!

How can we model the motion of a flexible object, like a string, that can take on uncountably many shapes? In principle, our approach will be the same as it has been for many other problems in this class: divide the string into small regions and understand how to model each one so that, in the limit of a large number of regions, the behavior approx- imates the real string. This is superficially similar to the way we did integration: divide a perfectly smooth thing into a bunch of blocky things, understand how each of the blocky things works, and then reason that if there are enough blocky things it looks sufficiently like the smooth thing.

2 A digression on "lattice spacing"

Before I discuss how to do this, Id like to digress for a bit. This is a very general idea: in my own research on quark physics we do the same thing. This is done in climate modeling, in lattice QCD ("lattice" just means "grid"), and even in the simulations of supernovae and nuclear bombs (which are actually the same computer code!) You do this every time you see a digital image the preceding paragraph is just a description of pixels, after all, and the text will seem more continuous if the pixels are made smaller while the overall size is kept the same.

However, in any physics problem like this, there are now two kinds of "small quantities".

There is the integration timestep, which you are familiar with. If the integration timestep is not small enough, then your results will not be very accurate, as we have discussed.

Now, however, theres also a "space step" – in our case, the length of our small chunks of string that we are going to assume are locally straight. In climate models, the "space step" is the width of small chunks of atmosphere that you assume are locally uniform. As you might expect, you make an error if your "space step" – the size of your pixels (or whatever) is too big. So now you have two small quantities that you have to make small. Only in the limit of small pixel size (called "lattice spacing") do the differential equations youre solving approach the real physics describing the problem. Then, only in the limit of small timestep will your code solve those differential equations accurately.

You are likely familiar with this from movies. A moving picture is continuous in both space and time: we break this up into pixels in the space directions, and frames in the time direction. Both the pixel count and the framerate must be high enough to have a high-quality accurate reproduction.

A movie recorded in low resolution (say, 320 pixels by 240 pixels) at 60 frames per second won't contain much fine detail, but does have a fast framerate.

A movie recorded in high resolution (say, 1920x1080 pixels, the standard for "1080p" high definition) at only

10 frames per second will look like a very detailed slideshow – good spatial resolution, horrid time resolution.

If you want the best quality, you need lots of both: *The Hobbit* used 5120x2880 pixels and 48 frames per second (Hollywood standard is 24 frames per second). This is because Peter Jackson wanted to ensure that fast-moving, small elements in action sequences that would be perceptible with the very high spatial resolution didn't collapse into a blurry mush.

This last is a general idea that applies to our simulations too: *if you have a high spatial resolution, you need a small timestep* to get those small dynamics right. The smaller your lattice spacing is, the smaller the timestep required to solve the problem accurately. This makes intuitive sense: small things change more rapidly than big things, so if youre trying to get the behavior of small things right, you need a smaller timestep. Consider climate modelling for a minute. If you are trying to understand the behavior of a hurricane, a timestep of one minute might be good enough. However, if youre trying to understand a dust devil superficially similar, in that its air going in a circle you need a timestep of a a tenth of a second or so, simply because the smaller dust devil changes faster than the larger hurricane. Later we will make this idea quantitative.

3 How to model a string

As alluded to above, were going to approximate the flexible string as a sequence of connected "inflexible" things. As it is elasticity were trying to model, a good guess is to use Hookes law to describe each individual piece. This will give us the overall behavior were going for, and the large number of pieces will give us the flexibility we want.

3.1 Deriving the microscopic properties from the macroscopic ones

What properties does a guitar string have? Here we arent looking for quantities like "total mass", since that is different depending on the length of the string; we want to describe the properties that the string has in a way independent of its total length.

Going back to freshman physics and materials science, a string has the following properties at first order:

- Length
- Cross-sectional area (thickness)
- Density
- Young's modulus (Young's modulus is to spring constant as resistivity is to resistance)

These last two are dependent only on the material of which it is made. Recall that $k = \frac{EA}{L}$, where E is the Youngs modulus of the material.

Now, were not attempting to model the area of the string. We only care about the quantities $A\rho$ (mass per length) and EA (spring constant times length). So now we have the following list:

- Length

- Linear density μ
- Young's modulus times cross-sectional area, which I'll call α (the "stiffness")

However, we are dividing our string into N point masses connected by $N-1$ ideal Hookes law springs. So what are their parameters? We need to know their length, mass, and spring constant, along with their unstretched length.

- If the total unstretched length is L, the unstretched length r_0 of each microscopic spring has length $L/(N)$.
- If the density is μ, then the mass of each of our $N+1$ point masses is $L\mu/(N+1)$

The spring constant is more difficult. If we want our full string to have a spring constant k, what is the spring constant of each element? The key is that the force exerted by each subelement must equal the force exerted by the whole string, since the tension in a little piece of the string is the same as the tension in the whole thing. (Tension is an intensive quantity, if you know what that means).

If the whole string is stretched by an amount Δx, then each element is stretched by an amount $\Delta x/N$. Hooke's law for the whole string gives us $T = k\Delta x$, and for each element gives $T = k'(\Delta x/N)$, where k' is the spring constant of each element. Equating these gives us $k' = k/N$. (A shorter way to say this is that spring constants "add like capacitors rather than "adding like resistors, if youve taken circuits.)

Recall that the spring constant for the whole string is $k = \alpha/L$. However, it's k' that will appear in our simulations, so we can substitute this into the previous equation to get

- Spring constant for each element: $k' = N\alpha/L$

If you are going to model a stretched string, you have to know how far to stretch it to get the correct amount of tension. This is easy; you just do it with Hookes law. If the string has an unstretched length L and you want to apply a tension T, this will stretch it to a larger length L'. Hooke's law tells us

$$T = k(L' - L) \tag{1}$$

where $k = \alpha/L$.

4 The force law

Each mass feels forces from the two springs which connect it to its two neighbors. (The exceptions here are the ones on the ends, which we hold stationary.) These forces are given by Hooke's law

$$\vec{F} = k(r - r_0)\hat{r} \tag{2}$$

where here $\hat{r}$ is a unit vector pointing from the object feeling the force in the direction of the spring.

Here, we can write the force that mass i feels from its two neighbors (with indices $i-1$ and $i+1$) as

$$\vec{F}_i = k(r_{i,i-1} - r_0)(\hat{r}_{i,i-1}) + k(r_{i,i+1} - r_0)(\hat{r}_{i,i+1}) \tag{3}$$

where r_{ij} is a vector pointing from node i to node j (found by subtracting their coordinates, as in the gravity simulation). As before you can use the "hat trick" to write the components of $\hat{r}_{ij}$ in terms of x_i, y_i, x_j, and x_i. r_0 is the unstretched length of the spring.

5 Coding strategy

To code this you will need to do the following:

- Create arrays to hold all of your dynamical variables ($N+1$ x's, y's, v_x's, and v_y's, although the ones at the end don't move)
- Think very carefully (draw pictures with pen and paper first!) about how the force law works, to avoid minus sign errors
- Figure out what initial conditions the dynamical variables should have, and write a `for` loop to set them all
- Write your standard `for` loop over time
- At each timestep:
  - Do the first leapfrog position update for each of the points that move (advancing them an amount $dt/2$)
  - Do the leapfrog velocity update, calculating all the forces and updating the velocities for all of the points
  - Do the second leapfrog position update, advancing the positions a time $dt/2$ again
- After some number of timesteps:
  - Use another `for` loop to go down the length of the string again, printing `anim` commands that animate your string
  - Compute the total energy to see if it is well-conserved