Cubical Grid of Masses — Transverse Waves

Completed and Analyzed in class, April 1, 2025

This is the sixteenth notebook for you to complete. It is our first notebook that has a three-dimensional network of masses. Since we actually live in a universe that has three spatial dimensions, it is of course important to actually study three-dimensional networks. However, most of the intuition about three-dimensional systems can actually be developed using one-dimensional and two-dimensional systems with many masses, of even one dimensional systems that only had a few masses, which is why we started with those.

Also, we are getting to the end of notebooks where we have to do the hard coding work of modeling a continuous system by creating a system with a large but finite number of masses. After this, we are going to let Mathematica do that hard work!

How can Mathematica do the hard work for us?! Well, from a mathematics perspective, the types of problems we are studying are called differential equations or partial differential equations. They have been studied numerically for decades (actually a full century, and people used mechanical calculators because they didn't yet have electronic ones), and painstakingly by hand for a century before that. Extensive libraries of software routines have been developed to do what we have been doing, and Mathematica has those software routines built into it. So we can just choose a physics problem, find out what differential equations are applicable to it, specify those differential equations to Mathematica, and Mathematica will (under the hood) do the work that we have been doing of chopping up the continuous system into little chunks and little time steps.

Periodic Boundary Conditions

We have considered fixed boundary conditions and free boundary conditions. Now I want to introduce you to one more common kind of boundary condition: periodic boundary conditions.

The universe does not seem to have an edge. Lots of media, like the ocean (if you are far from shore) do not have an edge. But because our computers can only do so much computation, we have to have a finite number of masses, and that means we have to come to an edge. One way to deal with the edge is to pretend that the right-most masses are connected to the left-most ones, the rear-most masses are connected to the front-most ones, and the top-most masses are connected to the bottom-most ones. Then as the wave leaves the right side (or the the rear, or the top), it can just travel into the left side.

What I have just described is the sneaky method of making an edgeless computer simulation, and ***this method or choice is known as "periodic boundary conditions."*** It is unlikely to ever be encountered in a real situation. Imagine walking out of the door of the Main Building closest to the Museum, and

finding yourself walking into the door at the end of the Main Building at the library. Or imagine looking towards the door closest to the Museum and in the distance seeing the back of your own head. That is what the experience of periodic boundary conditions would be like if you could eve encounter them in the real world.

Initial Conditions

Set up the duration, **steps,** and **deltaT**:

In[641]:=
```
tInitial = 0.0;
tFinal = 15.0;
steps = 3600;
deltaT = (tFinal - tInitial) / steps;
```

Set up the size of the grid (enough masses so that the grid is recognizably starting to form a continuum, but not so many that we tax our computers). During debugging of the code, we will keep the number of masses very small:

```
nx = 40;
ny = 4;
nz = 4;

(* You might want to dial 40, 4,
and 4 down while getting your code working. The values *)
(* I have chosen will take a computer 20 or so seconds which isn't that much *)
(* of an issue, but if you have bugs,
and symbolic nonsense is being passed from time *)
(* step to time step—instead of ordinary floating point numbers—then millions *)
(* of nonsensical symbolic computations can bring the house of cards down. So *)
(* consider something like nx=10, ny=2,
depth=2 while debugging, and then change *)
(* it back later. *)
```

Just to keep the code manageable, we are going to choose quite special initial conditions:

- ***The displacement of the masses will only be in the +z and -z directions.*** For atoms oscillating in a real cubical crystal, obviously the general case is that they can oscillate in any direction, although depending on the crystal structure, some oscillation directions might be preferred over others.
- The initial conditions will be chosen so that ***the wave will travel in the +x direction.***
- A wave that has displacement perpendicular to its direction of travel is called a ***transverse*** wave.
- There are also waves called ***longitudinal*** waves. An example would be a wave where the direction of displacement was in the $+x$ and $-x$ directions, and the direction of travel is also in the x direction. It will be interesting enough to get just transverse waves visualized. We don't also need to do longitudinal. Some continuous media, those which do not resist shearing force, only have longitudinal waves. Some continuous media have both longitudinal and transverse waves, but because of their differing

response to compression and shear, the two types of waves move at different speeds through them. I think it is too much for an introductory course to pile on any more of these interesting possibilities than I am already piling on, but it would only take us another notebook or two to do it.

In[648]:=
```
v0 = Pi;
maxz = 1.0;
thumpCenter = 12.0;
thumpWidth = 4.0;
initialzs =
  Table[maxz Exp[-(j - thumpCenter - 1)^2 / thumpWidth^2 ] , {j, nx}, {k, ny}, {l, nz}];
initialvs =
  v0/2 (Table[If[j == 1, initialzs[[nx, k, l]], initialzs[[j - 1, k, l]]], {j, nx}, {k, ny},
      {l, nz}] - Table[If[j == nx, initialzs[[1, k, l]], initialzs[[j + 1, k, l]]],
      {j, nx}, {k, ny}, {l, nz}]);
initialConditions = {tInitial, initialzs, initialvs};
```

Formulas for the Accelerations — Theory

The acceleration formula is now a three-dimensional second derivative:

$$a_{j,k,l} = v_0^2\left(z_{j+1,k,l} + z_{j-1,k,l} + z_{j,k+1,l} + z_{j,k-1,l} + z_{j,k,l+1} + z_{j,k,l-1} - 6\, z_{j,k,l}\right)$$

As mentioned in prior discussions, the addition of the second derivatives in each of the three directions is known as the Laplacian. You will learn a load more about the Laplacian and other special combinations of derivatives in three dimensions if you take an advanced calculus class (*e.g.,* the gradient, the curl, and the divergence).

Formulas for Periodic Boundary Conditions — Reminder

Don't forget, that as you code up $a_{j,k,l}$ that you need to capture what was described above under the section titled "Periodic Boundary Conditions."

Implementing the Accelerations

```
v0 = Pi;

a[j_, k_, l_, allzs_] := v0^2 (cómo se llama)
```

Second-Order Runge-Kutta — Implementation

In[657]:=
```
rungeKutta2[cc_] := (
  curTime = cc[[1]];
  curzs = cc[[2]];
  curvs = cc[[3]];
  newTime = curTime + deltaT;
  zsStar = curzs + curvs deltaT / 2;
  as = Table[a[j, k, l, zsStar], {j, 1, nx}, {k, 1, ny}, {l, 1, nz}];
  newvs = curvs + as deltaT;
  newzs = curzs + (curvs + newvs) deltaT / 2;
  {newTime, newzs, newvs}
 )

rk2Results = NestList[rungeKutta2, initialConditions, steps];
zResults = Transpose[rk2Results][[2]];
```

3D Graphics

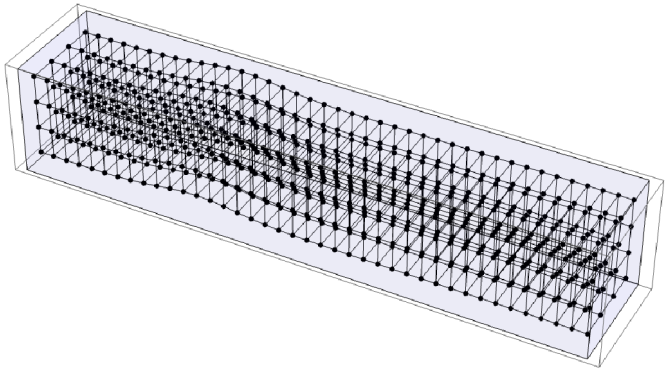
We need a graphics implementation with $n_x\, n_y\, n_z$ masses and the lines connecting them:

```
width = 60;
height = 12;
depth = 12;

xspacing = width / nx;
yspacing = depth / ny;
zspacing = height / nz;
cuboid =
 {FaceForm[{Blue, Opacity[0.04]}], Cuboid[{0, 0, 0}, {width, depth, height}]};
(* Notice how useful this next supporting function is: *)
pointPosition[j_, k_, l_, zs_] :=
  {(j - 1 / 2) xspacing, (k - 1 / 2) yspacing, (l - 1 / 2) zspacing + zs[[j, k, l]]};
latticeGraphic[zs_] := Graphics3D[Flatten[{
     {cuboid},
     Table[
      Point[pointPosition[j, k, l, zs]],
      {j, nx}, {k, ny}, {l, nz}
     ],
     cómo estás
    }, 1]];
```

We need a graphics implementation with $n_x\, n_y\, n_z$ masses and the lines connecting them. When you are done, you will can test your implementation of your graphics, and it should look do like this screenshot:

```
latticeGraphic[initialConditions[[2]]]
```



Animating the 3D Graphics

```
(* Let's leave this commented out until you get everything else going: *)

(* Animate[latticeGraphic[zResults[[step]]],
 {step, 0, steps, 1}, DefaultDuration→tFinal-tInitial] *)
```

Out[670]=