The 1D heat equation

MATH1091: ODE methods for a reaction diffusion equation

http://people.sc.fsu.edu/~jburkardt/classes/math1091_2020/heat_1d/heat_1d.pdf

---

How does heat "rearrange itself" over time?

The Heat Equation

If a metal rod is heated unevenly, and then left alone, the rod quickly assumes a uniform temperature, and then gradually cools down. How can we accurately simulate this process, which involves changes over time and space?

1 The Behavior of a Heated Metal Rod

Our (imaginary) experimental data will involve a metal rod. The endpoints of the rod can be temperature controlled, but we will assume that the rest of the rod is insulated from any outside influences. Using a blow-torch (the most fun we will have in this class) we can selectively apply heat to various portions of the rod, to get an initial configuration. After that, we will simply watch what happens over time.

In our first experiment, we simply want to know how our application of heat affects the rod immediately. This will involve variation in space only and is simply another example of a boundary value problem (BVP).

In our later experiments, we will try to model the process by which the heat differences in the rod gradually even out. Now we will be studying variation in both time and space. This problem is a combination of an initial value problem (IVP) and a boundary value problem (BVP). We will have to learn some new tricks to come up with an approximate solution.

2 Applying Heat to a Rod

When a metal rod has been heated by an external source $f(x)$, the distribution $u(x)$ of temperature might be modeled by the steady heat equation with Dirichlet boundary conditions:

$$-u'' = f(x)$$
$$u(a) = ua$$
$$u(b) = ub$$

where ua and ub are given temperature values. We can solve this problem in exactly the same way that we have handled previous examples of the 1D Poisson problem.

A few things will have changed, though. We will start preparing for the time dependent problem, so instead of using "h" for the spatial mesh size, we will use "dx". We will be looking at some problems for which an exact solution is not known, so we will often not assume we have a formula $g(x)$ for the exact solution. To specify the boundary conditions, we might simply give specific values or rules for the left and right endpoints. If we no longer are restricted to problems for which we have a simple formula for the exact solution, we can now use some trickier right hand side functions $f(x)$. However, this means it's not so easy to tell whether our approximate solution is accurate, and so we will want to get a little intuition about the solution of heat problems.

3 Exercise #1: Snapshot of a Heated Rod

For our first heat equation program, we can start with the *exercise2.m* file from our first lecture, *poisson_1d_steady*, or just work from scratch. Here is an outline of what you need to do:

1. Define the region to be $0 \leq x \leq 1.0$, and create a mesh x of `nx=21` nodes;
2. Assume the boundary conditions are `u(0.0)=0` and `u(1.0)=80`;
3. Assume the right hand side function is $f(x) = 0$;
4. Define the system matrix `A` in the usual way;
5. Solve for u;
6. Plot the solution;

From the plot, you should see that the solution has a very simple behavior, Now let's repeat the computation, but this time, we are going to superheat the rod in the region $0.5 \leq x \leq 0.7$. We do this by creating a function which is 1000 in that region, and 0 elsewhere. In MATLAB, you can do this with an `if()` statement, but here's another way:

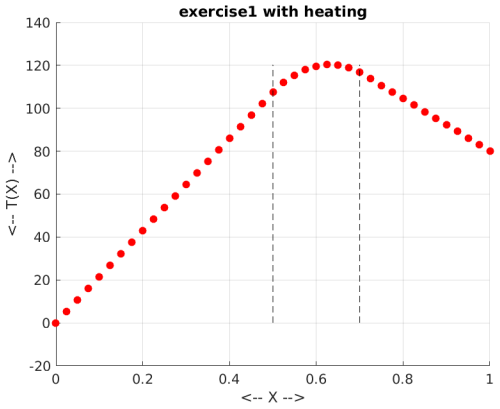
```
value = 1000.0 * ( 0.5 <= x && x <= 0.7 );
```

Listing 1: Define a step function.

After making this one change, run your code again. Now the solution looks more interesting! It is "simple" in the unheated region, but it does something new and interesting in the heated region: it **curves**.

This gives us a little insight into heat problems. If we hold the temperature of the rod fixed at both ends, then the temperature wants to behave linearly inbetween. But if we add heating over some region, this makes the local temperature profile nonlinear.

To emphasize that point, here is a plot of my solution, using 41 points, and highlighting the heated region:



The effect of local heating.

4 The Time Dependent Heat Equation

Once we have heated a rod, we might like to predict what is going to happen to the temperature distribution. We expect that hot and cold spots will even out, but we want to produce a model that approximates the situation for as long into the future as we are interested.

We can imagine such a model producing a sequence of snapshots, at regular time intervals. Since our spatial model involves a discrete set of nx points, we might think of our time model as a correspondence sequence of nt points. Our computed data might be thought of as a table where $U(i,j)$ is the predicted temperature at location $x(i)$ and time $t(j)$. But how can we fill in these numbers?

To keep things simple, we will assume that no more heating will be applied to the metal bar, so we won't have a source term $f()$. In that case, an appropriate model of the behavior of heat over time in a 1-dimensional region is:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

We need to start from this equation in order to create our computational model. The equation matches a change in time and a change in space. Our discretized data is now a grid of points in space and time. Suppose that at time t_j we have the temperature values at all the spatial locations. Our strategy will be to fill in the data at the next time value t_{j+1}.

To keep things simple, we will assume for now that the first and last nodes maintain their initial values forever, so it's easy to write

$$U(1, j+1) \leftarrow U(1, j)$$
$$U(nx, j+1) \leftarrow U(nx, j)$$

Now how do we approximate the temperature value $U(i, j+1)$ at a typical node in the interior of row $j+1$? Here is a natural discretization of the heat equation using our approximate solution values:

$$\frac{U(i, j+1) - U(i, j)}{dt} = \frac{U(i-1, j) - 2U(i, j) + U(i+1, j)}{dx^2}$$

We can rewrite this so that known quantities are on the right hand side, so that this now becomes an explicit formula for $U(i, j+1)$:

$$U(i, j+1) \leftarrow U(i, j) + \frac{dt}{dx^2} * (U(i-1, j) - 2U(i, j) + U(i+1, j))$$

5 Design of an explicit heat equation algorithm

Now we have a possible algorithm to solve the time dependent heat equation over a given space and time interval:

1. Let $h(x)$ be a formula for the initial condition;
2. Use `linspace()` to create nx points x in the spatial interval $[a, b]$;
3. Use `linspace()` to create nt points t in the time interval $[c, d]$;
4. Create an array U(1:nx,1:nt);
5. Set U(1:nx,1) to h(x(1:nx));
6. Set U(1,1) to h(x(1));
7. Set U(nx,1) to h(x(nx));
8. For j from 1 to nt-1:
   - set U(2:nx-1,j+1) using the explicit formula;
   - set U(1,j+1) to h(x(1));
   - set U(nx,j+1) to h(x(nx));
9. Use `surf()` to display U;

We will try this approach in the following exercise.

6 Exercise #2: Solve the heat equation with an explicit method

Create a file *exercise2.m* that uses the explicit method to solve a time dependent heat equation.

Assume that the problem is to be solved over the interval $0 \leq x \leq 3$, with time interval $0 \leq t \leq 1$. Create x and t arrays using $nx = 21$ nodes and $nt = 101$ time steps.

Let the initial condition be defined by the formula

$$h(x) = x(3 - x)(x^2 - 2x + 1)$$

Set the first row of U using the initial condition, and then fill in the rest of the array one row (time step) at a time.

To plot the solution with `surf()`, you will first want to build tables X and T from the vectors x and t using the `ndgrid()` command:

```
[ X, T ] = ndgrid ( x, t );
surf ( X, T, U );
```

You should be able to "grab" the surface plot and view it from different angles. The plot should show you that the solution starts out looking like a big hill, and then gradually settles down to an almost zero solution. This corresponds to our physical experience: local hot and cold spots tend to fade away. As long as we don't add any external influences, initial temperature differences should even out. Keep this in mind as we look at our next exercise!

7 Exercise #3: Varying grid and time stepsizes

In the previous exercise, we uses a grid of $nx = 21$ spatial points and $nt = 101$ time steps, which means our discretization parameters were $dx = \frac{3}{20}$ and $dt = \frac{1}{100}$. We took a lot of time steps in that calculation, but our mesh spacing was not as fine. It is natural to wonder whether we took *too many* time steps. Would it be more efficient to use fewer time steps? Would we get about the same answer?

For the explicit heat algorithm, the answer to these questions is contained in the **CFL parameter**. This is defined by $cfl = \frac{dt}{dx^2}$. The value of this parameter will tell us whether we have chosen safe values of dx and dt. Let's run some experiments first to see what can go wrong.

Let's solve our problem again, but with different discretizations. We will try

1. original: $nx = 21$, $nt = 101$
2. fewer steps: keep $nx = 21$, change $nt = 51$
3. more nodes: change $nx = 26$, keep $nt = 101$
4. more of both: $nx = 26$, $nt = 151$

Create a file *exercise3.m* by copying *exercise2.m*. Make it easy to set the values of nx and nt. I usually create my files as functions, so this means my *exercise3.m* file would begin with

```
function exercise3 ( nx, nt )
```

and so I can run the first experiment by typing

```
exercise3 ( 21, 101 )
```

If this style is not to your liking, you are free to work out another way to make the four experiments.

Inside your program, compute and print out the value of `cfl`.

Because something bad may happen, it might be better not to wait til the end to do the plotting. Perhaps you should delete the `surf()` command. Instead, inside of your time loop, plot the solution at the current time. This might be done something like:

```
  for j = 1 : nt - 1

%  Plot U(1:nx,j)

    plot ( x, U(1:nx,j) )
    axis ( [ 0.0, 3.0, 0.0, 3.0 ] )
    label = sprintf ( 'Time step %d', j );
    title ( label );
    pause ( )

%  Statements to compute U(1:nx,j+1) here


  end
```

Listing 2: Plot each solution by itself.

- The `axis()` command guarantees that all the plots will be shown on the same scale. This is because otherwise MATLAB will adjust the plot size to just fit the current data. We expect the data to decrease over time, but we don't want the plot size to change as this happens.
- We want the title to display the current time step. To do this, we have to use the slightly peculiar `sprintf()` function.
- The pause() function pauses the computation until you hit return; If you hold down the return key, you will get a sort of animation.

Now you should be ready to run the four experiments. You should expect to see two good results and two bad ones. The bad stuff should happen within about 20 steps. Once a computation goes bad (the plot will become very choppy) you can stop and move on to the next one. But make a note in each case of what happened, and what the value of the `cfl` parameter was.

Your experiments should provide evidence that supports the following result:

Theorem 1 *If the CFL parameter is greater than $\frac{1}{2}$, the explicit scheme for solving the heat equation is unstable.*

Roughly speaking, when an approximation scheme is unstable, small errors become large and spread throughout the solution. Solutions produced by an unstable scheme often include wild oscillations in value. We saw a similar example of a solution produced by an unstable scheme in the ODE class, when we used the explicit (forward) Euler method for a stiff equation.

8 Modeling insulation with a Neumann boundary condition

You may have noticed that in our heated rod simulation, the rod was clearly cooling down to a uniform temperature of 0. This is because we specified a permanent fixed zero temperature at both endpoints. This corresponds to our physical expectation that, if the ends of a heated rod are kept at zero degrees, and no extra heat is applied internally, then eventually all the heat energy will flow out of the rod and it will assume a temperature of zero. In general, if we fix the endpoints to any two temperatures, the rod will adjust to a linear profile between these two values. In the absence of a right hand side (external heating source), the Poisson equation tells the temperature to assume a linear profile.

We know that heat flows from hot to cold areas. That means that if we simply look at the slope of our temperature distribution, we can tell that the high spots are going to decrease, either to a flat profile, or a strictly linear one, depending on whether the boundary conditions are equal or not.

Heat represents energy, and we might be interested in trying to conserve it. Physically, we could try to retain heat in the rod by insulating it. In our mathematical model, heat can only leave through the endpoints. To stop that from happening, we could try to ensure that in the neighborhood of each endpoint, the temperature distribution was flat. We can't do this with a Dirichlet boundary condition, since we don't know in advance what the interior temperatures will be near the boundary. Instead, we can use the simplest kind of Neumann condition, the *zero derivative* condition. At the left end point, $x = a$, we would mathematically want the outward-pointing derivative to be zero:

$$-\frac{\partial u}{\partial x}(a) = 0$$

Computationally, this would correspond to a discrete zero Neumann boundary condition:

$$-\frac{u(2,j) - u(1,j)}{dx} = 0 \text{ for each time } j$$

which for a zero condition simplifies to

$$u(1,j) - u(2,j) = 0$$

A zero Neumann condition at the right endpoint would be written:

$$u(nx,j) - u(nx-1,j) = 0$$

9 Exercise #4: An insulated rod

Copy your file *exercise3.m* into a new file *exercise4.m*. Make the following changes:

1. Remove the statements that apply the Dirichlet conditions;
2. In the time loop, **after** you have updated `U(2:nx-1,j+1)`, apply a zero Neumann condition at both ends:

```
U(1,j+1) = U(2,j+1);
U(nx,j+1) = U(nx-1,j+1);
```

3. Follow by a heat energy estimation:

```
E = dx * ( sum ( U(1:nx,j+1) ) - 0.5 * U(1,j+1) - 0.5 * U(nx,j+1) );
```

4. Modify the computation of the title to report the energy:

```
label = sprintf ( 'Time step %d, E = %g', j, E );
title ( label );
```

5. Run your program with $nx = 21$ and $nt = 101$. Observe that the solution does not go to zero, and that the energy stays roughly constant.

The Dirichlet boundary condition is useful for modeling cases where we can specify the value of the temperature. The Neumann condition allows us to simulate the rate of temperature flux (heat loss) at an endpoint. Here, we are specifying insulation (no heat loss). By specifying a nonzero value, we could pump in or drain out a specified amount of energy from an end.

10 A periodic boundary condition

Suppose that we join the ends of the metal rod, so that it forms a ring, with no endpoints. What happens to our boundary conditions? Well, it turns out that we don't need them anymore. We still want to describe positions on the rod using $0 \leq x \leq 3$, but now, the locations $x = 0$ and $x = 3$ are identical. The node we create at $x = 3$ is really a sort of "phantom node". We could work without it, but it turns out to be convenient to keep it.

We will still set up nx equally spaced points, using `linspace()`, but the last point, at $x = 3$, will be treated specially. It's really the same as $x = 0$, so we will solve for the temperature at $x = 0$ and then just make a copy to assign to the "phantom node" at $x = 3$.

Because the rod is now a ring, the first node is not a boundary node, so we have to solve for the temperature there. And the first node has a left hand neighbor. It's node $nx - 1$ (not node nx!). Similarly, the last node (which is $nx - 1$, has a right neighbor at node 1. We can actually define three lists, the nodes to the left, center, and right, that are used in each calculation:

```
l = [ nx-1, 1:nx-2];
c = [ 1:nx-1];
r = [ 2:nx-1, 1 ];
```

For example, if $nx = 6$, we have

```
l = [ 5, 1, 2, 3, 4 ];
c = [ 1, 2, 3, 4, 5 ];
r = [ 2, 3, 4, 5, 1 ];
```

This means that we can update the temperature at time $j + 1$ using the formulas:

```
U(c,j+1) = U(c,j) + dt/dx^2 * ( U(l,j) - 2*U(c,j) + U(r,j) );
U(nx,j+1) = U(1,j+1);      % the phantom node
```

11 Exercise 5: Periodic boundary conditions on a rod

Copy your *exercise4.m* file into *exercise5.m*. Make the following modifications:

- Define the node index lists `l, c, r`.
- In the time loop, use the node index lists to compute `U(c,j+1)`;
- Replace boundary condition lines by a command that sets `U(nx,j+1) = U(1,j+1)`;

- The energy computation simplifies to `E = dx * ( sum ( U(c,j+1) );`

If you run this problem with, for example $nx = 21$ and $nt = 101$, you should see that the first node is immediately "warmed up" by the last node. The energy should stay roughly constant (the heat has nowhere to escape).

12 Including a diffusivity coefficient

A more realistic model of heat flow includes a diffusivity coefficient:

$$\frac{\partial u}{\partial t} = k * \frac{\partial^2 u}{\partial x^2}$$

where k models the speed at which temperature differences are propagated.

When k is included in the model, it must be included in the CFL parameter,

$$cfl = \frac{k\,dt}{dx^2}$$

and we must use this formula when choosing `dt` so that $cfl < \frac{1}{2}$

The coefficient also shows up in the update formula, which now becomes:

```
U(c,j+1) = U(c,j) + k * dt/dx^2 * ( U(l,j) - 2*U(c,j) + U(r,j) );
```

When k is a constant, this is all we have to do. However, if our rod consisted of an initial silver part, joined to an aluminum part, we would need to use a step-function k, and for more complicated materials, we might have a function $k(x)$, for which we would need to make more serious adjustments to the model.

13 Homework: A simple diffusivity problem

For previous problems, we have not had an exact solution available. But for this problem, consider the following function which describes a temperature distribution:

$$g(x,t) = 3\,\sin(3x)\,e^{-t}$$

The domain is the interval $0 \le x \le 3$, and the time interval is $0 \le t \le 1.0$.

Consider the following version of the heat equation:

$$\frac{\partial u}{\partial t} = k * \frac{\partial^2 u}{\partial x^2}$$

with $k = \frac{1}{9}$. The boundary conditions are:

$$u(0) = 0$$
$$u(3) = 0$$

and the initial condition is

$$u(x,0) = 3\,\sin(3x)$$

Write a MATLAB problem *hw3.m* to solve this problem. You might start from your file *exercise3.m*.

Here are some guidelines:

1. Include the value of k in your program;
2. Use $nx = 21$. Find a value of `nt` so that your algorithm satisfies the CFL condition;
3. Set the initial condition as given;
4. Change the solution update to include `k`;
5. At each time step, plot your computed solution and the exact $g(x, t)$ together;
6. Save one of your plots as *hw3.png*;

Send your plot *hw3.png* to me at **jvb25@pitt.edu**. I would like to see your work by Friday, May 22.