Advection: Getting Carried Away

MATH1091: ODE methods for a reaction diffusion equation

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

---

Wind and water will carry away any items they catch.

The Advection Equation

The wind carries along heat, humidity, dust; rivers transport quantities of silt and debris. The advection equation models how such quantities move when they are carried along in the flowstream.

1 A model of passive transport

Consider a one-dimensional finite region $0 \leq x \leq xmax$, such as a hose, along which some fluid is flowing. If the fluid is water, and it enters the region on the left with a velocity c, then we expect the flow to have that same velocity c at all times and locations in the hose. This is because water is essentially incompressible. We will stick with this constant velocity version of the problem; the compressible flow problem, while it can handle important problems in atmospheric dynamics, is more than we can cover in one lecture.

Now imagine that we add some secondary item to the flow, such as a small amount of dye. We can designate by $u(x,t)$ the local concentration of dye at location x at time t. Moreover, we can find a formula $f(t)$ for the concentration of dye that is being added to the water at the left endpoint for every time $0 < t < tmax$. Presumably, knowing the initial condition, that is $u(x,0)$ for all x, and the injection function $f(t)$, we have enough information to completely describe what is going to happen in this simple physical model.

But how do we mathematically describe this behavior, and how do we computationally approximate it?

We can get a handle on what is going on by thinking of a small interval, $[x, x+\Delta x]$. In a time interval of Δt, our substance flows *into* the interval in the amount $c * \Delta t * u(x)$, and flows *out* of the interval in the amount $c * \Delta t * u(x + \Delta x)$. Since $u(x,t)$ represents the concentration of the substance, the *amount* of substance in

the interval is roughly $u * dx$. We are interested in the change in this amount:

$$\begin{aligned} \Delta u * \Delta x &= c * \Delta t * u(x) - c * \Delta t * u(x + \Delta x) \\ \frac{\Delta u}{\Delta t} &= c \frac{(u(x) - u(x + \Delta x))}{\Delta x} \quad \text{(rearranging)} \\ \frac{\partial u}{\partial t} &= -c \frac{\partial u}{\partial x} \quad \text{(taking the limit)} \end{aligned}$$

This is known as the **advection equation for constant velocity**.

2 A simple solution for a constant velocity field

If the advection equation is posed with a constant positive velocity c, and posed over the entire real line, and the initial condition is $u_0(x)$, then the solution for any time $0 \leq t$ is

$$u(x,t) = u_0(x - ct)$$

This solution simply moves the initial condition, unchanged, to the right with velocity c.

Moreover, if the advection equation with constant positive velocity c is posed on a *finite* interval $xmin \leq x \leq xmax$, with initial condition $u_0(x)$, *and* we impose a periodic boundary condition, requiring $u(xmin) = u(xmax)$, then the solution for all time is

$$u(x,t) = u_0(xmin + \mathrm{mod}(x - ct, xmax - xmin))$$

which means that the initial condition again simply moves to the right, but "wraps around" and re-enters the interval from the left.

3 The constant velocity, finite interval, periodic case

Just to get a feeling for how to deal with an advection problem, let's begin by consider the finite interval $0 \leq x \leq 100$ meters, through which we have a flow with a constant velocity $c = 0.8 \frac{\text{meters}}{\text{second}}$, and for which we have a formula for the exact solution, `exact(x,t,c)`.

We are interested in watching the solution over the time interval $0 \leq t \leq 126$ seconds.

An outline of our code might look like this:

```
function exercise1 ( exact )
  set velocity c
  set spatial quantities nx, xmin, xmax, x, dx
  set time quantities nt, tmin, tmax, t, dt

  for k = 1 : nt

    u = exact ( x, t(k), c );

    plot ( x, u, 'b-' );

  end

end
```

It seems we don't need the variables `dx` and `dt`, but we set them now because they will be useful very soon!

In this first example, we don't have to do anything to enforce the periodicity of the solution. We assume that the `exact()` function will take care of this, by returning `exact(xmin,t(k),c) == exact(xmax,t(k),c)`.

The input parameter `exact` is the name of a MATLAB function that evaluates the exact solution. We will consider two examples:

- `bumpy()`, a solution with sharp corners;
- `smoothie()`, a smooth solution;

We choose the solution function we are interested in by specifying it as the input parameter when we run the code, with an @-sign: `exercise1(@bumpy)` or `exercise1(@smoothie)`.

4 Exercise #1: Display the smooth and bumpy solutions

Create a MATLAB file *exercise1.m* which can display the smooth and bumpy solutions.

Use discretization parameters $nx = 101$ and $nt = 253$.

Although you can plot with a single statement, I would suggest adding some extra lines to improve the display. The `axis()` function will be useful later, when we want to make sure that each image shows up in a graph of the same scale. The `pause()` function will ensure that the succession of images shows up at a steady, but not too rapid, pace.

```
plot ( x, un, 'b-', 'linewidth', 3 );
grid ( 'on' );
xlabel ( '<--- x --->' );
ylabel ( '<--- u(x,t) --->' );
label = sprintf ( 'Exact: k = %3d, t = %4.1f', k, t(k) );
title ( label );
axis ( [ xmin, xmax, 0.0, 1.1 ] );
pause ( 0.1 );
```

Listing 1: Suggested enhanced plotting commands.

Run your code with both exact solutions. In each case, you should see a wave that never changes shape, but moves to the right at a constant speed.

5 The FTCS method

A mathematician derives equations by starting with small quantities and letting them go to zero ina limit process. A computational scientist looks at mathematical equations involving infinitesimals, and tries to "inflate" them to tiny, computable finite differences.

We begin by discretizing the geometry, so the interval $[0, xmax]$ is replaced by nx nodes x equally spaced by dx. (We really should call this Δx but we know what we mean!) Similarly, the time interval $[0, tmax]$ is split into nt times spaced by dt. Now we look at the advection equation. Writing $u(i, j)$ to mean our estimated solution at node $x(i)$ and time $t(j)$, the left hand side of the equation can be approximated by a simple difference:

$$\frac{\partial u}{\partial t}(x_i, t_j) \approx \frac{u(i, j+1) - u(i, j)}{dt}$$

We could approximate the right hand side at the old or the new time. For simplicity, we will start by choosing to approximate at the old time, working with known data. But because the right hand side is a

first spatial derivative, not a second derivative, we still have several choices to consider.

$$\begin{aligned}\frac{\partial u}{\partial x}(x_i,t_j) &\approx \frac{u(i,j)-u(i-1,j)}{dx} \quad \text{backward difference} \\ &\approx \frac{u(i+1,j)-u(i-1,j)}{2dx} \quad \text{centered difference} \\ &\approx \frac{u(i+1,j)-u(i,j)}{dx} \quad \text{forward difference}\end{aligned}$$

For the Forward Time Centered Space method, we combine the forward time difference by the centered space difference, so that, to estimate $u(i,j+1)$, we will use the formula

$$\frac{u(i,j+1)-u(i,j)}{dt} = -c\,\frac{u(i+1,j)-u(i-1,j)}{2dx} \quad \text{the FTCS method}$$

We simplify the relationship by writing `uo()` for the solutions at time j and `un()` at the new time $j+1$. Then we can rewrite so that we have an explicit formula for `un()` in terms of `uo`:

$$un(i) = uo(i) - 0.5\,(c\,dt/dx)\,(uo(i+1) - uo(i-1))$$

6 Outline of an FTCS advection solver

An outline of our code might look like this:

```
function exercise1 ( exact )
  set velocity c
  set spatial quantities nx, xmin, xmax, x, dx
  set time quantities nt, tmin, tmax, t, dt

  set up special index arrays

  for k = 1 : nt

    if k is 1
      un = exact solution at time t(k)
    else
      uo = un
      un = ftcs update for nodes 1 through nx-1
      u(nx) = u(1)
    end

    v = exact solution at time t(k)
    plot ( x, u, 'b-', x, v, 'r--' );

  end

end
```

Since we have periodic boundary conditions, the node at `x(nx)` is really a phantom node, and the value of u there should simply be a copy of the value at `x(1)`. Therefore, we want to use FTCS to set the values of `un()` only at nodes `i = 1, ..., nx-1`, using values of `uo()` at indices `i-1, i,` and `i+1`.

Of course, the equation for un(1) will reference uo(i-1), which will be 0 (illegal) and the equation for un(nx-1) will reference uo(nx), the phantom node, when we really should reference uo(1).

If we do the calculation in a `for` loop, we will have to handle these two problems as special cases. But an alternative is to create three index arrays as follows:

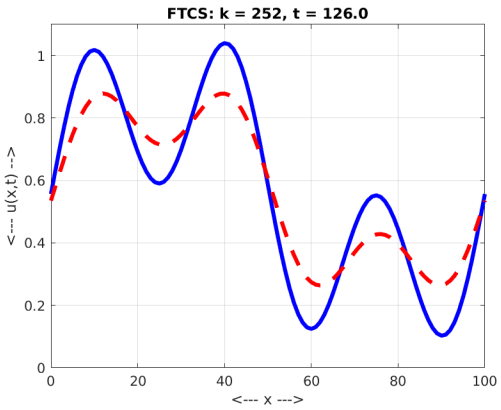
```
IM1 = [ nx-1, 1:nx-2 ];
I   = [ 1:nx-1 ];
IP1 = [ 2:nx-1, 1 ];
```

Listing 2: Useful index arrays for advection.

because now, instead of a loop, we can write a single statement:

$$un(I) = uo(I) - 0.5\ (c\ dt/dx)\ (uo(IP1) - uo(IM1))$$

which does all the updating, and uses the correct indexes for the first and last assignments.



The solution at the final time step, using the FTCS method.

7 Exercise #2: Write an FTCS advection solver

Create *exercise2.m* by starting with a copy of *exercise1.m*.

Modify the time loop so that, on step `k=1` it sets `un` to the initial condition, but thereafter, it uses the index vectors to set `un(I)` using the FTCS method, and then updates the phantom node value `un(nx)`.

Run your code with the smooth initial condition. Now run with the bumpy initial condition. Watch carefully when something bad begins to happen. Can you suggest a reason that could explain why the bumpy problem is not handled well?

8 The upwind method

In the FTCS method, we approximated the derivative using a central difference:

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

This seems to make sense because of symmetry. However, consider that

- the time derivative is looking "into the future";
- the velocity c is positive;
- the flow at `x(i)` is carrying information from the `x(i-1)` node;
- hence, for physical reasons, the backward spatial derivative might be a better approximation than the forward derivative, and possibly even better than the central derivative;

So, assuming a constant positive velocity c, the upwind method for the advection equation is

$$\frac{u(i,j+1)-u(i,j)}{dt} = -c\,\frac{u(i,j)-u(i-1,j)}{dx} \quad \text{the upwind method}$$

9 Exercise #3: Write an upwind advection solver

Make *exercise3.m* by copying *exercise2.m*.

Replace the FTCS method by the upwind method.

Run your code with the bumpy initial condition. How has the result changed?

10 The Lax-Friedrichs method

The Lax-Friedrichs method starts with the FTCS method, and tries to avoid the problems we saw with the bumpy initial condition. It does this by modifying the way the time derivative is handled:

$$\frac{u(i,j+1)-0.5(u(i-1,j)+u(i+1,j))}{dt} = -c\,\frac{u(i+1,j)-u(i-1,j)}{2\,dx} \quad \text{Lax-Friedrichs method}$$

As before, we can rewrite this as an explicit formula for `u(i,j+1)`:

```
un(i) = 0.5 * (uo(i-1)+uo(i+1)) - (c dt)/(2 dx) * (uo(i+1)-uo(i-1))
```

Listing 3: Lax Friedrichs formula.

The effect of the revised treatment of the time derivative is to smooth out the approximation. This combats the jagged oscillatory solutions that we saw when FTCS was applied to the bumpy problem.

11 Exercise #4: Write a Lax-Friedrichs advection solver

Make *exercise4.m* by copying *exercise2.m*.

Replace the FTCS method by the Lax-Friedrichs method.

Run your code with the bumpy initial condition. Did you get a jagged solution that blows up?

Run your code with the smooth initial condition. Does it behave well here also?

12 The Lax-Wendroff method

So far, our solution methods have used a first order approximation for the time derivative, while the spatial derivative has been approximated to first order (upwind method) or second order (FTCS and Lax-Friedrichs). The Lax-Wendroff method uses a complicated formula that produces an approximation that is second order in time and space. So far, we have seen a solution method that blows up, and solution methods that try to simulate the exact wave but seem to spread out and flatten its shape. But because of the higher accuracy of the Lax-Wendroff method, there is hope that we can produce more reliable approximations for our simple constant velocity problems.

The Lax-Wendroff formula looks like this:

$$\frac{u(i,j+1)-u(i,j)}{dt} = -c\,\frac{u(i+1,j)-u(i-1,j)}{2\,dx} + \frac{c^2\,dt}{dx}\,\frac{u(i+1,j)-2\,u(i,j)+u(i-1,j)}{2\,dx} \quad \text{Lax-Wendroff method}$$

As we have done in previous examples, to make a computational version of this method, we write `un()` and `uo()` for the new and old values, and move the old values to the right hand side. This gives us an explicit formula for the solution values at the next time step. For this homework exercise, I will not write out the equation for you, but rather expect you to put it together yourself.

13 Homework: Write a Lax-Wendroff advection solver

Make *hw.m* by copying *exercise2.m*.

Replace the FTCS method by the Lax-Wendroff method.

Run your code with the smooth initial condition. What improvement do you see, compared to previous calculations?

Run your code with the bumpy initial condition. Does the solution blow up, or show spikes, or does it get smoothed out?

Send me a snapshot, called *hw6.png*, of the solution on the last time step, for the bumpy initial condition at **jvb25@pitt.edu**. I would like to see your work by Friday, 12 June 2020.