The Burgers Equation

MATH1091: ODE methods for a reaction diffusion equation

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

A follower who's faster will cause a disaster!

The Burgers Equation

In advection, a velocity field carries along other items. The Burgers equation studies how the velocity field transports velocity itself.

1 Momentum transport

When we looked at the advection equation, we assumed the velocity was constant, and that the flow carried along some secondary quantity, such as a dye, heat energy, or pollutant. Especially if we used periodic boundary conditions, it was easy to see that over time, the initial condition simply slid to the right (assuming the velocity c was positive), wrapping around to return through the left endpoint.

It was surprisingly hard to come up with a solution strategy that could well approximate this simple behavior. However, it was important to practice on this simple case, because there are more interesting problems to study, for which we don't know the exact solution. In particular, we might consider a velocity field that varied with time or space.

However, another twist is to think about the fact that one of the quantities that the flow is transporting is velocity itself. This is a little tricky to think about, but let us assume that $u(x,t)$ represents *both* the velocity and the quantity being transported. Then we have a generalization of the advection equation as follows:

$$\frac{\partial u}{\partial t} = -u \frac{\partial u}{\partial x} \quad \text{BNI: Burgers nonconservative inviscid equation}$$

Note that we can rewrite this equation as

$$\frac{\partial u}{\partial t} = -\frac{1}{2} \frac{\partial u^2}{\partial x} \quad \text{BCI: Burgers conservative inviscid equation}$$

which is equivalent mathematically. However, we will see that the two equations are not equivalent computationally: each version of the equation suggests a different discretization, and the resulting approximations have different properties.

Both of these equations are "inviscid", which means that physically they do not try to model the way that fluid motion tends to get smoothed out. We will see that leaving out this feature makes both forms of the inviscid Burgers equation hard to handle, producing solutions that are very likely to break down in a nonphysical way. It will be a relief when we add viscosity and try things a second time!

2 Upwind for BNI

To begin with, we can simply treat the Burgers nonconservative inviscid equation as an advection equation where the velocity c is replaced by the unknown u. This means, for instance, that if we want to use the upwind scheme, and we know that the velocity will always be positive, we can consider the formula:

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

To specify the full problem, we need to state the range of x and t, the initial condition, and the kind of boundary conditions we want to use. We will find it convenient to use periodic boundary conditions, so that our waves can exit on the right and return from the left. (This is assuming that the velocity u is always and everywhere positive!).

So once again, a reasonable solution scheme is to treat the node at $x(nx)$ as a "phantom" node whose u value is always equal to that at `x(1)`. So we need to solve for values at nodes `I=1,2,...,nx-1`, and set the last value by copying the first.

For programming convenience, we will set the vectors `IM1, I, IP1` to make it easy to index the correct nodes for all our schemes. Instead of a single `u` array, we will have `un()` for the "new" velocity, and `uo` for the "old" velocity, so that we have to remember to update `uo` at the beginning of each upwind step.

3 Exercise #1: Upwind solver for BNI

Create the MATLAB file *exercise1.m* by copying *exercise3.m* from the advection directory.

Reformulate the upwind scheme for the BNI equation by replacing `c` in the upwind scheme by the value `uo(I)`.

Here, `I` is the vector of indices $1, 2, ..., nx-1$ that I set up in the advection exercises, along with `IM1` and `IP1`. By using these indices, we can skip using a `for` loop, and we can also take care of the periodic boundary conditions automatically. Refer to one of the exercises that I posted, so that you can see how this idea makes things simpler.

Instead of accepting an `exact()` function, make your code accept an `initial()` function. (Exact solutions for the BNI equation are hard to come up with!) The `initial()` function is simpler than before, and should have the form `un=initial(x)`. On time step 1, call `initial()` instead of `exact()`.

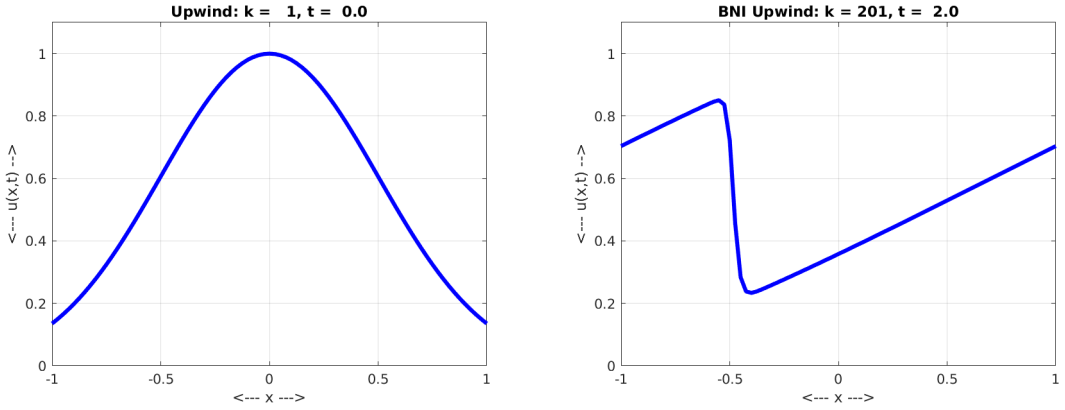
Since we don't know the exact solution, remove the code that computes the error `err`, that computes the exact solution `vn`, and that plots `vn` along with the computed solution `un`.

Your code should use `nx = 81` for $-1 \leq x \leq +1$ and `nt = 201` for $0 \leq t \leq 2$. When your code is ready, try running it with the command

```
exercise1 ( @gaussian_wave )
```

Listing 1: Upwind approximation for Gaussian wave.

You should expect to see a wave that starts out like a Gaussian curve, moves to the right, becoming steeper in the front (the fast portion) and shallower in the back, where it is moving more slowly and stretching out.



The initial Gaussian wave, and the predicted shape after 200 time steps.

4 Monitoring velocity conservation

Physics includes a law of conservation of momentum, the product of mass and velocity. Mathematically, solutions of the Burgers equation will also satisfy this law. Since our physical domain uses periodic boundary conditions, and there are no source terms to add or subtract energy, the conservation law then simply says that for any time $tmin \leq t \leq tmax$, we have

$$\int_{xmin}^{xmax} u(x,t)\, dx = \text{ Constant}$$

Our discrete solution produces values at `nx` equally spaced points. Because of periodicity, we can approximate the integral at time $t(j)$ essentially by averaging the first `nx-1` points:

$$\int_{xmin}^{xmax} u(x,t(j))\, dx = \text{ Constant} \approx \sum_{i=1}^{nx-1} un(i,j)\Delta x$$

but computationally, this is simply:

```
c(j) = sum ( un(I) ) * dx;
```

where, as before, `I=[1,2,...,nx-1]`.

5 Exercise #2: Plot velocity conservation using Upwind for BNI

Create *exercise2.m* by copying *exercise1.m*.

At the end of each iteration `j` of your time loop, evaluate the quantity `c(j)`, the integral of your velocity `un()`.

Once the time loop is finished, create a plot, which you might call *exercise2_conservation.png*, which plots `t` versus `c`. In order to give a base line for comparison, you can also plot the value of `c(1)` as a constant curve. You might do this with one plot command, like so:

```
plot ( t, c, 'b-', [t(1),t(nt)], [c(1),c(1)], 'r--' );
```

Run your program using the same parameters as used by `exercise1`. What does your conservation plot suggest to you about the upwind method?

6 Exercise #3: Lax-Friedrichs for BNI

We used the Lax-Friedrichs method previously, for the advection equation. However, in the advection equation, u was some quantity being carried by the flow, and the flow had a constant velocity c. Now, for the Burgers nonconservative inviscid equation, the velocity that was c in the advection equation is identical with $u()$, and so the Lax-Friedrichs scheme will be written as:

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

Create a MATLAB file *exercise3.m* by starting from your file *exercise1.m*. Implement the Lax-Friedrichs method for the BNI equation. You can refer back to *exercise4.m* from the advection section, to recall how the Lax-Freidrichs method was implemented there. Of course, back then c was a constant velocity, so you need to make that change.

Run your program with the same parameters and the Gaussian initial condition.

Your conservation plot for the Lax-Friedrichs solution should look better than what we saw for the upwind method. While this doesn't mean the solution is correct, it does mean that the Lax-Friedrichs method is getting a more physically correct result.

Compare the plot of your final Lax-Friedrichs solution with the plot you got using the upwind method. Both methods are trying to solve the same problem, so we'd hope to get similar solutions. But when I did this, it seemd my two curves were different in noticeable ways - just looking at them, in other words. Since we don't have a formula for an exact solution, we are in the same situation as someone with two watches that differ. Can we reassure ourselves that both methods are trying to solve the same problem, and will both get there eventually?

7 Exercise #4: Compare Upwind and Lax-Friedrichs for BNI

Of course, both our upwind and Lax-Friedrichs methods are approximations to a solution, and they are approximations with a relatively low spatial discretization of `nx=81`. So perhaps at least some of the difference in the two results may be because, while they are both heading towards the same solution, they haven't gotten there yet. Let's try to see whether this is true. We will rerun both computations, using `nx=161`, and we will show both solutions in the same plot.

Create the file *exercise4.m* by copying your upwind code *exercise1.m*. Your code calculates the upwind solution, using the variable names `uo()` and `un`. Now, carefully, insert your Lax-Friedrichs computation into the same code, but name these variables `vo()` and `vn()`, so that in a single loop, you are computing both solutions.

Modify your plot command to show both solutions:

```
plot ( x, un, 'r-', x, vn, 'b-' );
legend ( 'Upwind', 'Lax-Friedrichs' );
```

If, in previous computations, you were computing the conservation property and plotting that at the end, we don't need that information for this case. Instead, after the iteration is completed, compute and print the integral of the difference between the two solutions, which we can write as:

```
e = sum ( abs ( un(I) - vn(I) ) ) * dx;
```

If we had time, we could run `exercise4` repeatedly, with finer spatial resolutions, to see numerically whether the solutions provided by the two methods get closer or not. (However, as we increase `nx`, we might eventually also have to increase `nt` to avoid stability issues. There's always one more complication than you expect.)

Increase your spatial discretization to `nx=161` and run your code. Look carefully at the plot of the last result. Do the two results look almost identical? Close? Or still very different?

From my results, I concluded that the Lax-Friedrichs solution for `nx=81` was too smooth, and that by increasing `nx` I got a solution whose shape was much more similar to the upwind result. However, there was still one noticeable difference between the two curves.

8 The Upwind method for the BCI Equation

Let's consider the Burgers conservative inviscid equation, now, which is

$$\frac{\partial u}{\partial t} = -\frac{1}{2}\frac{\partial u^2}{\partial x} \quad \text{BCI: Burgers conservative inviscid equation}$$

To approximate this equation, we will again assume that the velocity is positive, so that what happens at node i is affected by changes at node $i-1$. Therefore, we choose to approximate the right hand side quantity by a backward difference. This gives us the difference equation:

$$\frac{u(i,j+1) - u(i,j)}{dt} = -\frac{1}{2}\frac{u^2(i,j) - u^2(i-1,j)}{dx} \quad \text{BCI upwind}$$

and this is the formula we rewrite in terms of `uo()` and `un()` for our computation.

9 Exercise #5: Upwind solver for BCI Equation

Create *exercise5.m* from your file *exercise1.m*. Rewrite the BCI upwind formula so that it is an explicit formula for `un(I)` in terms of `uo(IM1)` and `uo(I)`, and replace your old upwind formula with this new one.

Run the code with the same parameters you used for *exercise1.m*. If you are keeping track of conservation, you should see that, using BCI, the upwind method is able to do a much better job of conservation. In exercise1, the upwind method will produce a solution that slowly dies down to nothing, whereas using the BCI form of the equation, the upwind solution can go on and on without loss. This is what we would expect from an ideal wave.

Of course, in reality, waves *do* die down eventually; this is not because nature is using the BNI; it's because of something physical, called viscosity. We just have time to take a quick look at this factor in the homework exercise.

10 The Burgers Viscous Equations

Viscosity is a material property of a fluid that resists motion. It is the reason that your can run your finger quickly through a bowl of water, but not so quickly through a bowl of honey. If a particle in a viscous fluid begins to move, it feels a drag from its nieghbors, which results in the particle slowing down, and its

neighbors speeding up. In effect, the moving particle ends up dragging its neighbors along. The effect is similar to what happens in the heat equation, where a hot spot of high temperature gradually cools down by "sharing" its heat with its neighbors.

The Burgers equation is a model for fluid flow, and all real fluids have a viscosity. Therefore, it is important to be able to simulate this physical property. Viscosity is usually denoted by the Greek symbol ν, or spelled out as `nu`. It is a positive quantity, and for our problems, it will be a relatively small value, such as $\nu = 0.001$. The viscosity will multiply a second derivative of the velocity, creating what is known as the *diffusion term*, since this represents the way viscosity averages out highs and lows in velocity. Although we will prefer the conservative version of the Burgers equation, we can write viscous versions in either case:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + \nu\frac{\partial^2 u}{\partial x^2} \quad \text{BNV: Burgers nonconservative viscous equation}$$

and

$$\frac{\partial u}{\partial t} = -\frac{1}{2}\frac{\partial u^2}{\partial x} + \nu\frac{\partial^2 u}{\partial x^2} \quad \text{BCV: Burgers conservative viscous equation}$$

With the addition of viscosity, the BNV and BCV have the form of **parabolic** partial differential equations. The standard methods for approximating solutions are known as parabolic methods. In our parabolic approach, we will use a forward Euler difference in time, and centered differences for both spatial derivatives.

11 Homework: Parabolic solver for BCV Equation

Create a MATLAB file *hw7.m* by copying *exercise5.m*.

Insert a statement that sets the viscosity:

```
nu = 0.01;
```

Approximate the advection term by a first central difference:

$$-\frac{1}{2}\frac{\partial u^2}{\partial x} \approx -\frac{1}{2}\,\frac{uo^2(IP1) - uo^2(IM1)}{2\,dx}$$

Approximate the diffusion term by a second central difference:

$$+\nu\frac{\partial^2 u}{\partial x^2} \approx +nu\,\frac{uo(IP1) - 2\,uo(I) + uo(IM1)}{dx^2}$$

Now, using the usual forward Euler approximation for the time term, rewrite the BCV equation using the discrete approximations, isolate `un(I)` on the left hand side so that we have a formula. This is the parabolic scheme for solving the BCV equation.

Run your code with the same values of `nx` and `nt` that we have used so far, with the Gaussian wave as the initial condition. Save a plot of the solution at the last timestep as *hw7.png*.

Notice the effect of viscosity which tends to average out the wave. You may also notice that the conservation property does pretty well with this scheme.

Send me the plot *hw7.png* at **jvb25@pitt.edu**. I would like to see your work by Friday, 19 June 2020.