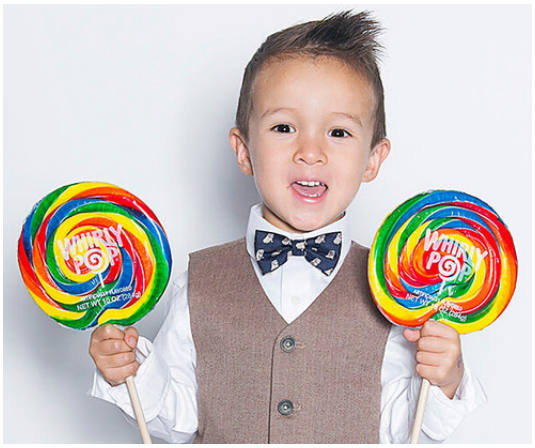
A spiral reaction-diffusion solution

MATH1091: ODE methods for a reaction diffusion equation

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



A reaction of confusion to a spiral solution.

A reaction diffusion equation

Reaction-diffusion equations are differential equations that can model processes that involve chemical reactions, biochemical processes, pattern formation, and the separation of a mixture of two liquids. We will see several ways of approximating the Laplacian, which models diffusion, and a variety of different equations with surprising behaviors.

1 Estimating the Laplacian on a periodic domain

The equation for the variation in heat was expressed in terms of the negative Laplacian operator applied to the solution:

$$\frac{\partial u}{\partial t} = -\nabla^2 u = -\frac{\partial^2 u}{\partial x^2} \qquad \text{1D equation}$$

$$\frac{\partial u}{\partial t} = -\nabla^2 u = -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} \qquad \text{2D equation}$$

Our heat equation typically came with boundary conditions, and so in our discrete computation, we never had to approximate the Laplacian at a boundary node. Because we worked in the interior of the region, we always had enough neighbors to apply our standard formula, involving three neighboring nodes in the 1D case, or 5 nodes in the 2D case.

We are about to look at some problems where it is much more convenient to avoid the issue of boundary conditions. One way to do this is to use the periodic boundary condition, to behave as though our grid was part of a quilt involving infinitely many identical copies of the grid. Another way is to use the "zero normal derivative" boundary condition, which mathematically says that the function $u(x)$ is "flat" at the boundary, that is, $\frac{du}{dx} = 0$. Computationally, we can enforce this by setting the boundary value as a copy of its immediate neighbor.

To start with, we will look at cases where we treat the region periodically. Thus, a 1D region can be thought of as a sort of circle, and a 2D region becomes what is known as a *torus*. Mathematically, this means that boundary points on opposite ends of the region are regarded as a single point. One way to handle this computationally, in the 1D case, is to start with a grid of nx points, and then remove the last point. Any time you would want to reference that point, you use the value at $x(1)$ instead.

2 Exercise #1: Approximate Laplacian on a 1D periodic domain

Copy the file *skeleton1.m*.

The main program defines a uniform grid of nx points on $[0,1]$. It then removes the last grid point, leaving a periodic grid of $nxp = nx - 1$ points. On that grid, it evaluates a function up and its Laplacian $uxxp$, which in 1D is simply the value of $\frac{d^2u}{dx^2}$.

Then, the code wants to call a function `lp = laplacian_circle(up,dx)` which should use the values `up` and `dx` to estimate the Laplacian. The code wants to compute the RMS norm $||uxxp - lp||$. This is to be done for several values of nx, which will presumably show that the RMS error decreases as nx increases.

Your task is to fill in the details of the function *laplacian_circle()* so that it correctly estimates the Laplacian. The only tricky matters to note are

- the number of points is nxp, not nx, because we dropped the value at $x = 1$;
- therefore the result vector lp should also have length nxp;
- the i-th value of lp wants to use the values of up at indices $i-1, i, i+1$. What must you do when $i = 1$ and when $i = nxp$?

When you run the program, you should see that the RMS error goes down in relationship to nx.

3 Exercise #2: Approximate Laplacian on a 2D periodic domain

Copy the file *skeleton2.m*.

This exercise repeats exercise #1, but now for the case of a 2D periodic domain.

In the previous examples of storing 2D data in an array, I tried a different convention, but that was just as confusing as the usual way. So let's assume that we have sample values of a quantity U at an $nx \times ny$ grid of points, stored in an an $nx \times ny$ array with typical value $U(i,j)$. We have already seen several times how to approximate the laplacian using 5 values, assuming that nodes are uniformly spaced by dx and dy in the x and y directions, and that i and j are away from the boundaries:

$$\nabla^2 U(i,j) \approx lp(i,j) = \frac{U(i+1,j) - 2U(i,j) + U(i-1,j)}{dx^2} + \frac{U(i,j+1) - 2U(i,j) + U(i,j-1)}{dy^2}$$

Similar to exercise #1, this exercise defines a uniform periodic grid of $nxp \times nyp$ points, that is, $nx-1 \times ny-1$ points, on $[0,2] \times [-1,+1]$. On that periodic grid, it evaluates a function up and its Laplacian $uxxp$, which is the value of $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$. Then, the code wants to call a function `lp = laplacian5_torus(up,dx,dy)` to estimate the Laplacian.

Your task is to fill in the necessary details for the function laplacian5_torus(). You want to estimate the laplacian at **every** node in the region. Since the region is periodic, you will have to be careful when you are working with points on the boundary. For example, to compute

- `lp(1,j)`, instead of `U(1-1,j)`, you want to use `U(nxp,j)`;
- `lp(i,nyp)`, instead of `U(i,nyp+1)`, you want to use `U(i,1)`, and so on;

Just as in exercise #1, the code will compute the RMS difference between the exact laplacian and your estimate for a number of different grid sizes. If things are working correctly, you should see this difference decrease in a natural way.

4 Reaction diffusion equations

A reaction-diffusion equation is a mathematical model of a system with a spatially varying concentration of one or more substances. At each point, the given substances may undergo a reaction that changes their concentration. Moreover, a diffusion process means that the concentration at each point is also tends to match the average concentration at nearby points.

Typically, the concentrations are given in relative terms, with values between 0 and 1. When there is a single substance, the corresponding differential equation will have the form

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u + f(u)$$

where the nonnegative parameter ν is the diffusion coefficient, and $f(u)$ is the reaction term.

Fisher's equation is an example of a reaction-diffusion equation involving one space dimension x and one concentration variable u:

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u + u(1-u)$$

Imagine that ν is 0 for the moment, and focus on the reaction term. First we notice that this term is 0 if $u = 0$ or $u = 1$. This means that a concentration of exactly 0 or 1 would have a zero time derivative; that is, it would never change. However, for $0 < u < 1$, the reaction term is strictly positive, driving the value of u to increase. This means that the reaction term essentially forces all nonzero u values to move to 1.

Now let's "turn on" the diffusion term. Its effect is to smooth out variations in u. In fact, the discretized version of this term pushes u towards the average of its two neighbors. How does this affect the behavior of the solution? Without diffusion, we can see that a zero value of u will stay zero forever. But with diffusion turned on, it the zero value has a positive neighbor, then this will push it, ever so slightly, towards also having a positive value. Once that happens, it will be forced to gradually ascend to 1.

This suggests that, as long as $0 < \nu$, and as long as we have any nonzero value of u in our initial condition, the eventual solution of the Fisher equation will be the constant solution $u(x) = 1$. In the following exercise, we will set up this equation and verify these claims.

5 Exercise #3: The Fisher 1D reaction-diffusion equation

Copy the file *skeleton3.m*, which sets up most of the Fisher equation.

Inside the time loop, the Euler step is written as follows:

```
uxxp = laplacian_circle ( up, dx );
dudt = ?
up = up + dt * dudt;
```

Listing 1: Euler step for Fisher equation.

Add your `laplacian_circle()` function so that you can compute the laplacian, and replace the question mark by the discretized Fisher equation.

Run your program with the values $\nu = 0, 0.001, 1.0$. Do you see the effect that the diffusion term has on the results?

6 The Allen-Cahn 1D reaction diffusion equation

A metallic ore may actually comprise a mixture of two distinct metals. When the ore is refined by heating, the liquid will at first consist of many small separate drops of either metal, but as time progresses, droplets of the same type will tend to merge, so that the fluid organizes itself into large regions of the same type, with thin layers of mixed type separating any two regions.

The Allen-Cahn equation can simulate this physical behavior. By tradition, the variable u is allowed to range $-1 \leq u \leq +1$, with -1 or +1 indicating a pure region, while intermediate values indicate a mixed state. Unlike the Fisher equation, we are told to expect that between regions of $u = +1$ and $u = -1$, there may be a thin boundary layer with intermediate values of u, and this boundary layer will not necessarily disappear.

The version of the equation that we are interested in is:

$$\frac{\partial u}{\partial t} = \nu \nabla^2 u - \frac{u(1-u^2)}{2\,\xi}$$

The equations include a parameter ξ, or `xi` which determines how thick the boundary layer will be. Small values of ξ will allow for a solution with more closely packed alternating regions; larger values will force such regions to disappear or to spread away from each other.

A larger value of the diffusion coefficient ν or `nu` will tend to quickly smear out the irregularities in the initial condition, and will also tend to discourage "thin" regions.

Our version of the equation will not use periodic boundary conditions. Instead, it will use the "zero Neumann" condition: $\frac{\partial u}{\partial n} = 0$. We will implement this by setting $u(1) \leftarrow u(2)$ and $u(nx) \leftarrow u(nx-1)$. This will affect how we deal with the computation of the Laplacian.

7 Exercise #4: The Allen-Cahn 1D reaction diffusion equation

Copy the file *skeleton4.m* which contains most of the code needed to solve the Allen-Cahn equation.

Inside the time loop, the Euler step is written as follows:

```
uxxp = laplacian_interval ( u, dx );
dudt = ?
u = u + dt * dudt;
```

Listing 2: Euler step for Allen-Cahn equation.

Create your `laplacian_interval()` function by copying your `laplacian_circle()` function. Simply set the first and last component of the laplacian to zero (it's someone else's problem now, that is, the boundary conditions will be used to set u(1) and u(nx).) Since we are skipping the first and last nodes, your loop to compute the laplacian can run from `i=2` to `nx-1`, and so there's no need to worry about the legality of the indices `i-1` and `i+1`.

Fill in the missing code for the value of `dudt` in the time loop.

You run the program by passing in the values of `xi, nu`. Leave `nu=1`, but check out the following values for `xi=0.01, 0.015, 0.0155, 0.02`. In each case, you should initially see three "humps" with $u = +1$, but what happens next will vary. so that you can compute the laplacian,

8 The Grey-Scott 2D reaction-diffusion equation

Often, a process involves a pair of chemical species, u and v, which are spatially distributed over a domain. These species may react with each other, changing their concentration. Normally, such a chemical process will simply proceed until an equilibrium is reached and all the reactions have been carried out. But in a biological process, there may be related processes which continually add some species and remove others. This is a reasonable model of many biochemical processes, such as the way in which stripes or spots are organized on the skin of an animal under development.

The Gray-Scott model simulates a biochemical process in which the chemical reaction of the two species u and v is symbolized by

$$u + 2v \rightarrow 3v$$
$$v \rightarrow p$$

where p represents an inert product of the reaction, and the differential equations for the concentrations u and v are:

$$\frac{\partial u}{\partial t} = d_u \nabla^2 u - uv^2 + f(1-u)$$
$$\frac{\partial v}{\partial t} = d_v \nabla^2 v - (f+k)v$$

where d_u and d_v are the diffusion coefficients, k represents the chemical reaction rate for $v \rightarrow p$, and f is the rate at which additional reactant species are supplied.

We are interested in simulating a solution of the Gray-Scott equation on a square, with periodic boundary conditions - that is, the region is logically a torus.

Moreover, we wish to approximate the Laplacian using a stencil of 9 points rather than the usual 5. For convenience, we will assume that the spacing is uniform, and the same in both x and y directions. In that case, we can write

$$\nabla^2 u \approx (u_{i-1,j-1} + 4u_{i-1,j} + u_{i-1,j+1} + 4u_{i,j-1} - 20u_{i,j} + 4u_{i,j+1} + u_{i+1,j-1} + 4u_{i+1,j} + u_{i+1,j+1})/(6\ dx^2)$$

9 Exercise #5: Simulate the Grey-Scott equation

Copy the file *skeleton5.m*, which contains most of a Gray-Scott equation solver.

Currently, the time step code looks like this:

```
uLaplace = laplacian9_torus ( u, dx, dy );
vLaplace = laplacian9_torus ( v, dx, dy );

dudt = ?
dvdt = ?

u = u + dt * dudt;
v = v + dt * dvdt;
```

You should be able to fill in the question marks with code that looks almost exactly the same as the mathematical differential equation.

If you are easily discouraged, you can use your *laplacian5_torus()* code here, but I encourage you, instead, to try to build the *laplacian9_torus()*. If you simply create variables `im1, ip1, jm1, jp1` and modify them when you are on the boundary, you should be able to get this working.

Inside the code, there are a number of parameters you can change. The only ones I would look at are the variables `f` and `k`, the "feed" and "kill" rates. The code lists suggested values called *rings, splits, coral, bubbles,* and *waves*. Definitely try the *coral* values. The code should be set up right now so that that is the default.

If your program runs to completion, you will also have an AVi movie that you can replay or show to friends.

10 A spiral reaction-diffusion equation

Professor Trenchea's goal for this class was to be able to set up the spiral reaction-diffusion equation. The original description of this problem is in the file *spiral_project.pdf*. There have been a few changes to the project since then, so I will restate the problem here.

Our homework problem involves the following pair of reaction-diffusion equations:

$$\frac{\partial u}{\partial t} = \nabla^2 u + \frac{1}{\epsilon} u(1-u)(u - \frac{v+\beta}{\alpha})$$
$$\frac{\partial v}{\partial t} = \delta \, \nabla^2 v + u - v$$

with parameters

$$\alpha = 0.25$$
$$\beta = 0.001$$
$$\delta = 0.00001$$
$$\epsilon = 0.002$$

The equations are set inside the rectangle $[0,80] \times [0,80]$, using a 401×401 grid of nodes. The boundary conditions are $\frac{\partial u}{\partial n} = 0$, $\frac{\partial v}{\partial n} = 0$, which we will enforce by computing u and v at all the interior nodes, then copying the second row into the first row, the next to the last row into the last row, and similarly copying the second column into the first column, and next to last column into the last column.

The time variable runs $0 \leq t \leq 5$, and we plan to compute 5001 equally spaced values in time.

As a convenience, the grid coordinates and the initial condition have been supplied as part of the program.

11 Homework #10: The spiral reaction-diffusion equation

Copy the file *skeleton6.m*, which contains an almost complete version of the spiral reaction-diffusion equation solver.

The time looks something like this:

```
      if ( true )
        del2U = laplacian5_torus ( U, dx, dy );
        del2V = laplacian5_torus ( V, dx, dy );
      else
        del2U = laplacian9_torus ( U, dx, dy );
        del2V = laplacian9_torus ( V, dx, dy );
      end

      dUdt = ?
      dVdt = ?

      U = U + dt * dUdt;
```

```
    V = V + dt * dVdt;
```

Listing 3: Euler step for spiral reaction diffusion equation.

You will need to replace the question marks by the correct statement of the equations.

You can reuse your previously written *laplacian5_torus()* function here. That code assumed we were using periodic boundary conditions, and so it was able to supply laplacian values at the boundary nodes. For our problem, we only need the laplacian at the interior nodes, so we just ignore the extra data.

When you run your program, some time after $t = 1.0$, you will start to see a spiral forming, and this will get more obvious as time progresses. However, you will notice that the spiral seems to have a somewhat rectangular shape.

If you have time, then, copy the *laplacian9_torus()* function that you used in exercise #5, or write it now, if you didn't previously. Adjust the time loop so that it calls that function for the laplacians, and observe how the spiral forms again, but this time with a somewhat rounded shape.

At the end of the run, the program will automatically save the last plot in a file.

Send the plot *hw10.png* to me at **jvb25@pitt.edu** AND to Professor Trenchea at **trenchea@gmail.com**. I would like to see your work by Friday, 10 July 2020.

This is the last assignment for our class. Professor Trenchea and I hope that these two classes have given you some interesting insights into mathematics, computing, and the modeling of physical processes using differential equations.

Professor Trenchea and I have enjoyed working with you, and we wish you good luck in your graduate studies!