Case study: solving a Dirichlet problem in 2D

Goals

- manipulate 2D arrays
- reshape arrays
- set up a nontrivial system of linear equations
- solve a system of linear equations ... and finally:
- *be able to study the equilibrium state of the heat equation / electric networks / random walkers, and much more*

Physics

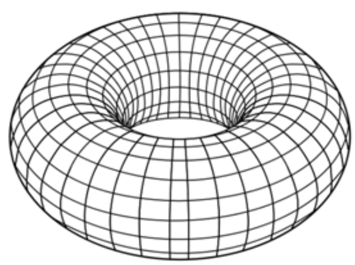
The heat equation, for homogeneous thermal diffusivity:

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0$$

In the steady state, $\partial u/\partial t$ becomes zero. Then, the sum of the curvatures $\nabla^2 u = \partial^2 u/\partial x_1^2 + \partial^2 u/\partial x_2^2 + \ldots + \partial^2 u/\partial x_p^2$ along the p dimensions of the domain must be zero everywhere except at the boundaries of the domain (including those locations where u is pinned so some constant value, e.g. by coupling to an infinite heat bath).

This is also called a Dirichlet problem.

Assume that our domain is a torus, parametrized in terms of a 2D Cartesian lattice.



On a 2D Cartesian lattice, the Laplace operator ∇^2 can be approximated using the stencil

```
0  1  0
1 -4  1
0  1  0
```

Note that this choice is arbitrary, and other approximations exist.

To solve for u with boundary values, we need to set up a linear system of equations

$$-Lu = r$$

If there are n nodes in our domain, then L is an $n \times n$ matrix constructed as follows:

- each row pertaining to one of the "clamped" nodes has a one on the diagonal and zero elsewhere
- each row pertaining to a "free" node has a +1 in the four columns corresponding to its north / south / east / west neighbors on the Cartesian lattice, and a -4 on the diagonal. All other entries are zero.

As to the right-hand side r, it has the fixed boundary value in a row that corresponds to a clamped node, and a zero in all other rows.

Task

Load these boundary conditions () using `numpy.load`, or generate some more to your liking (I used this (http://www.piskelapp.com)). Solve the Dirichlet problem assuming toroidal boundaries. Plot the boundary conditions, the Laplacian matrix L, and the result u.

Hints

The easy part, in terms of programming, is the actual solution of the linear system of equations. You can use `np.linalg.solve`

The hard part is the correct construction of the Laplacian matrix! I proceed as follows:

- iterate over all nodes / pixels
  - if the current node is "free", then
    - create the stencil in the spatial domain. I find it easiest to do this by creating an image with a single white pixel, and convolving it with the stencil using in-built routines. This takes care of all corner cases (literally).
    - reshape the resulting stencil image to yield a single row of the matrix L.
  - if the current node is a "clamped" node, then set the diagonal element to one.
  - initialize the corresponding row of r appropriately.