13 Poisson's equation

Suppose that we want to describe the distribution of heat throughout a region Ω. Let $h(x)$ represent the temperature on the boundary of Ω ($\partial\Omega$), and let $g(x)$ represent the initial heat distribution at time $t = 0$. If we let $f(x,t)$ represent any heat sources/sinks in Ω, then the flow of heat can be described by the boundary value problem (BVP)

$$\begin{aligned} u_t &= \triangle u + f(x,t), \quad x \in \Omega, \quad t > 0, \\ u(x,t) &= h(x), \quad x \in \partial\Omega, \\ u(x,0) &= g(x). \end{aligned} \tag{13.1}$$

When the source term f does not depend on time, there is often a steady-state heat distribution u_∞ that is approached as $t \to \infty$. This steady state u_∞ is a solution of the BVP

$$\begin{aligned} \triangle u + f(x) &= 0, \quad x \in \Omega, \\ u(x,t) &= h(x), \quad x \in \partial\Omega. \end{aligned} \tag{13.2}$$

This last partial differential equation, $\triangle u = -f$, is called Poisson's equation. This equation is satisfied by the steady-state solutions of many other evolutionary processes. Poisson's equation is often used in electrostatics, image processing, surface reconstruction, computational fluid dynamics, and other areas.

Poisson's equation in two dimensions

Consider Poisson's equation together with Dirichlet boundary conditions on a rectangular domain $R = [a,b] \times [c,d]$:

$$\begin{aligned} u_{xx} + u_{yy} &= f, \quad x \text{ in } R \subset \mathbb{R}^2, \\ u &= g, \quad x \text{ on } \partial R. \end{aligned} \tag{13.3}$$

Let $a = x_0, x_1, \ldots, x_n = b$ and $c = y_0, y_1, \ldots, y_n = d$ be evenly spaced grids. Furthermore, suppose that $b - a = d - c$, so the rectangular domain is also square. Thus we have a single stepsize h, where $h = x_{i+1} - x_i = y_{i+1} - y_i$

We look for an approximation $U_{i,j}$ on the grid $\{(x_i, y_j)\}_{i,j=0}^{n}$.

Recall that

$$\begin{aligned}\triangle u &= u_{xx}(x,y) + u_{yy}(x,y) \\ &= \frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2} \\ &\quad + \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2} + \mathcal{O}(h^2).\end{aligned}$$

We replace $\triangle$ with the finite difference operator $\triangle_h$, defined by

$$\begin{aligned}\triangle_h U_{ij} &= \frac{U_{i+1,\,j} - 2U_{i,\,j} + U_{i-1,\,j}}{h^2} + \frac{U_{i,\,j+1} - 2U_{i,\,j} + U_{i,\,j-1}}{h^2}, \\ &= \frac{1}{h^2}(U_{i-1,\,j} + U_{i+1,\,j} + U_{i,\,j-1} + U_{i,\,j+1} - 4U_{i,\,j}).\end{aligned}$$

These equations are linear, so we can expect to write them in matrix form. However, since our unknown variables are doubly-indexed (for x_i and y_j), we first need to rewrite them as a 1-dimensional array. We can do this by "stacking" the columns of the 2-dimensional array. Let the vector of unknowns U be:

$$U = \begin{bmatrix} U^1 \\ U^2 \\ \\ U^{n-1} \end{bmatrix} \text{ where } U^j = \begin{bmatrix} U_{1,\,j} \\ U_{2,\,j} \\ \\ U_{n-1,\,j} \end{bmatrix} \text{ for each } j,\ 1 \le j \le n-1.$$

Then the set of equations

$$\triangle_h U_{ij} = f_{ij}, \quad i,j = 1,\ldots,n-1,$$

can be written in matrix form as

$$AU + p + q = f. \tag{13.4}$$

A is a block tridiagonal matrix, given by

$$\frac{1}{h^2}\begin{bmatrix} T & I & & & \\ I & T & I & & \\ & \ddots & \ddots & \ddots & \\ & & I & T & I \\ & & & I & T \end{bmatrix} \tag{13.5}$$

where I is the $n-1 \times n-1$ identity matrix, and T is the $n-1 \times n-1$ tridiagonal matrix

$$\begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 1 \\ & & & 1 & -4 \end{bmatrix}.$$

The vectors p and q come from the boundary conditions of (13.3), and are given by

$$p = \begin{bmatrix} p^1 \\ \cdots \\ \\ p^{n-1} \end{bmatrix}, \quad q = \begin{bmatrix} q^1 \\ \cdots \\ \\ q^{n-1} \end{bmatrix},$$

where

$$p^j = \frac{1}{h^2}\begin{bmatrix} g(x_0, y_j) \\ 0 \\ \vdots \\ 0 \\ g(x_n, y_j) \end{bmatrix}, \quad 1 \le j \le n-1,$$

and

$$q^1 = \frac{1}{h^2}\begin{bmatrix} g(x_1, y_0) \\ g(x_2, y_0) \\ \vdots \\ g(x_{n-2}, y_0) \\ g(x_{n-1}, y_0) \end{bmatrix}, \quad q^{n-1} = \frac{1}{h^2}\begin{bmatrix} g(x_1, y_n) \\ g(x_2, y_n) \\ \vdots \\ g(x_{n-2}, y_n) \\ g(x_{n-1}, y_n) \end{bmatrix}, \quad q^j = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad 2 \le j \le n-2.$$

The vector f comes from the source term of (13.3), and is given by

$$f = \begin{bmatrix} f^1 \\ \cdots \\ f^{n-1} \end{bmatrix}, \quad \text{where} \quad f^j = \begin{bmatrix} f(x_1, y_j) \\ f(x_2, y_j) \\ \cdots \\ f(x_{n-1}, y_j) \end{bmatrix}$$

Note that this linear system is very large (A has $(n-1)^4$ entries) and very sparse (most of the entries in A are zero). Thus we will should make use of sparse matrix routines (such as those in `scipy.sparse` and `scipy.sparse.linalg`) in order to reduce the time and memory used in setting up and solving the linear system.

Problem 1. Complete the function `poisson_square` by implementing the finite difference method 13.4. Use `scipy.sparse.linalg.spsolve` to solve the linear system. Use your function to solve the boundary value problem:

$$\begin{aligned} \Delta u &= 0, \quad x \in [0,1] \times [0,1], \\ u(x,y) &= x^3, \quad (x,y) \in \partial([0,1] \times [0,1]). \end{aligned} \tag{13.6}$$

Use $n = 100$ subintervals for both x and y. Plot the solution as a 3D surface.

Poisson's equation and conservative forces

In physics Poisson's equation is used to describe the scalar potential of a conservative force. In general

$$\Delta V = -f$$

where V is the scalar potential of the force, or the potential energy a particle would have at that point, and f is a source term. Examples of conservative forces include Newton's Law of Gravity (where matter become the source term) and Coulomb's Law, which gives the force between two charge particles (where charge is the source term).

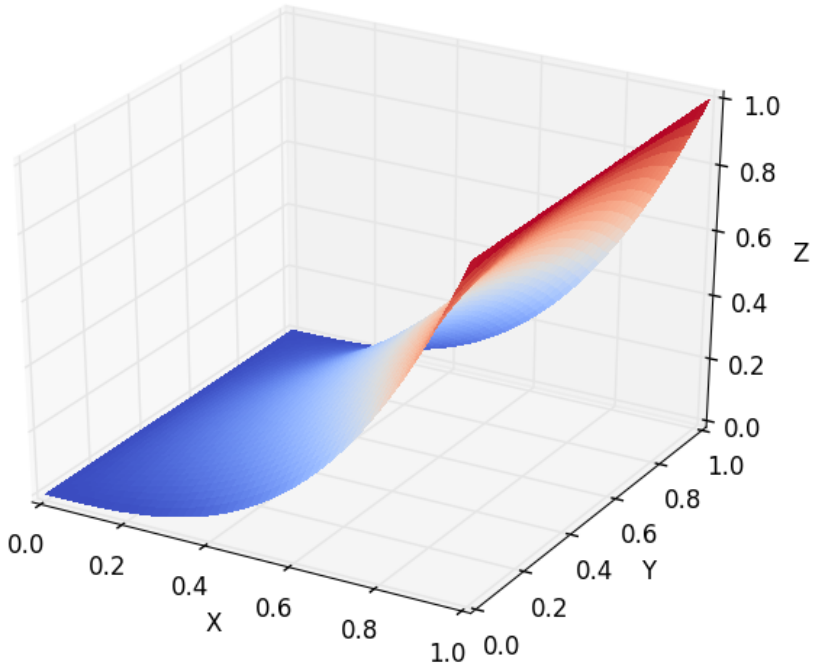

Figure 13.1: The solution of (13.6).

In electrostatics the electric potential is also known as the voltage, and is denoted by V. From Maxwell's equations it can be shown that that the voltage obeys Poisson's equation with the electric charge density (like a continuous cloud of electrons) being the source term:

$$\Delta V = -\frac{\rho}{\varepsilon_0},$$

where ρ is the charge density and ε_0 is the permissivity of free space, which is a constant that we'll leave as 1.

Usually a non zero V at a point will cause a charged particle to move to a lower potential, changing ρ and the solution to V. However, in this analysis we'll assume that the charges are fixed in place.

Suppose we have 3 nested pipes. The outer pipe is attached to "ground," which usually we define to be $V = 0$, and the inner two have opposite relative charges. Physically the two inner pipes would function like a capacitor.

The following code will plot the charge distribution of this setup.

```
import matplotlib.colors as mcolors

def source(X,Y):
    """
```

```
    Takes arbitrary arrays of coordinates X and Y and returns an array of the ↩
        same shape
    representing the charge density of nested charged squares
    """
    src = np.zeros(X.shape)
    src[ np.logical_or(
        np.logical_and( np.logical_or(abs(X-1.5) < .1,abs(X+1.5) < .1) ,abs(Y) ↩
            < 1.6),
        np.logical_and( np.logical_or(abs(Y-1.5) < .1,abs(Y+1.5) < .1) ,abs(X) ↩
            < 1.6))] = 1
    src[ np.logical_or(
        np.logical_and( np.logical_or(abs(X-0.9) < .1,abs(X+0.9) < .1) ,abs(Y) ↩
            < 1.0),
        np.logical_and( np.logical_or(abs(Y-0.9) < .1,abs(Y+0.9) < .1) ,abs(X) ↩
            < 1.0))] = -1
    return src

#Generate a color dictionary for use with LinearSegmentedColormap
#that places red and blue at the min and max values of data
#and white when data is zero

def genDict(data):
    zero = 1/(1 - np.max(data)/np.min(data))
    cdict = {'red':   [(0.0,  1.0, 1.0),
                       (zero,  1.0, 1.0),
                       (1.0,  0.0, 0.0)],
             'green': [(0.0,  0.0, 0.0),
                       (zero,  1.0, 1.0),
                       (1.0,  0.0, 0.0)],
             'blue':  [(0.0,  0.0, 0.0),
                       (zero,  1.0, 1.0),
                       (1.0,  1.0, 1.0)]}
    return cdict


a1 = -2.
b1 = 2.
c1 = -2.
d1 = 2.
n =100
X = np.linspace(a1,b1,n)
Y = np.linspace(c1,d1,n)
X,Y = np.meshgrid(X,Y)

plt.imshow(source(X,Y),cmap =  mcolors.LinearSegmentedColormap('cmap', genDict(↩
    source(X,Y))))
plt.colorbar(label="Relative Charge")
plt.show()
```

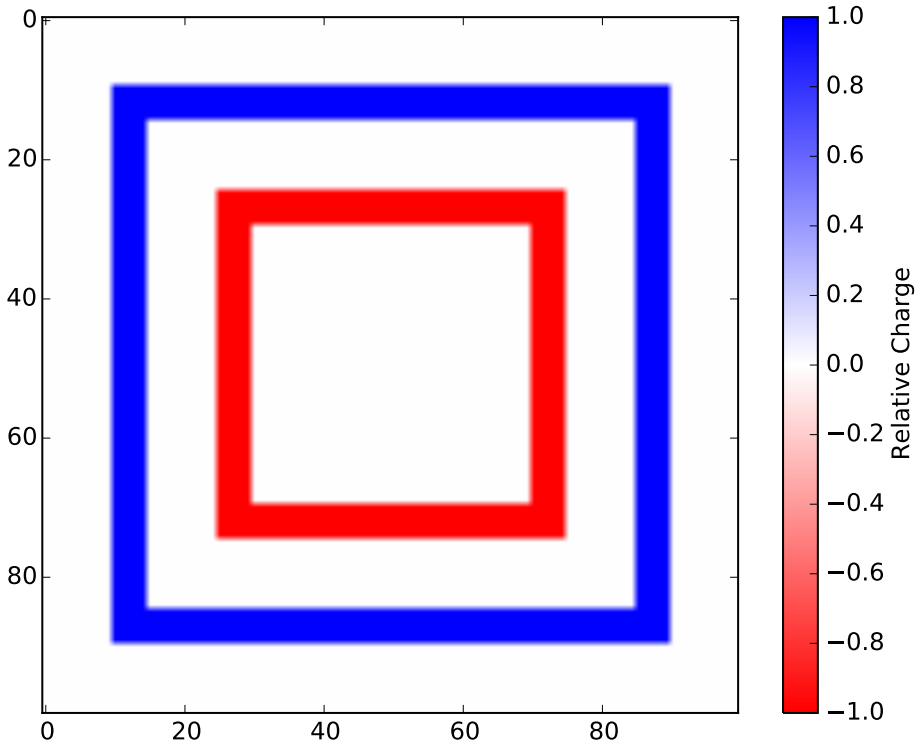

Figure 13.2: The charge density of the 3 nested pipes.

The function `genDict` scales the color values to be white when the charge density is zero. This is mostly to help visualize where there are neutrally charged zones by forcing them to be white. You may find it useful to also apply it when you solve for the electric potential.

With this definition of the charge density, we can solve Poisson's equation for the potential field.

Problem 2. Using the `poisson_square` function, solve

$$\begin{aligned} \Delta V &= -\rho(x,y), \quad x \in [-2,2] \times [-2,2], \\ u(x,y) &= 0, \quad (x,y) \in \partial([-2,2] \times [-2,2]). \end{aligned} \tag{13.7}$$

for the electric potential V. Use the source function defined above, such that $\rho(x,y) = \text{source}(x,y)$. Use $n = 100$ subintervals for x and y. Use the provided code to plot your solution.

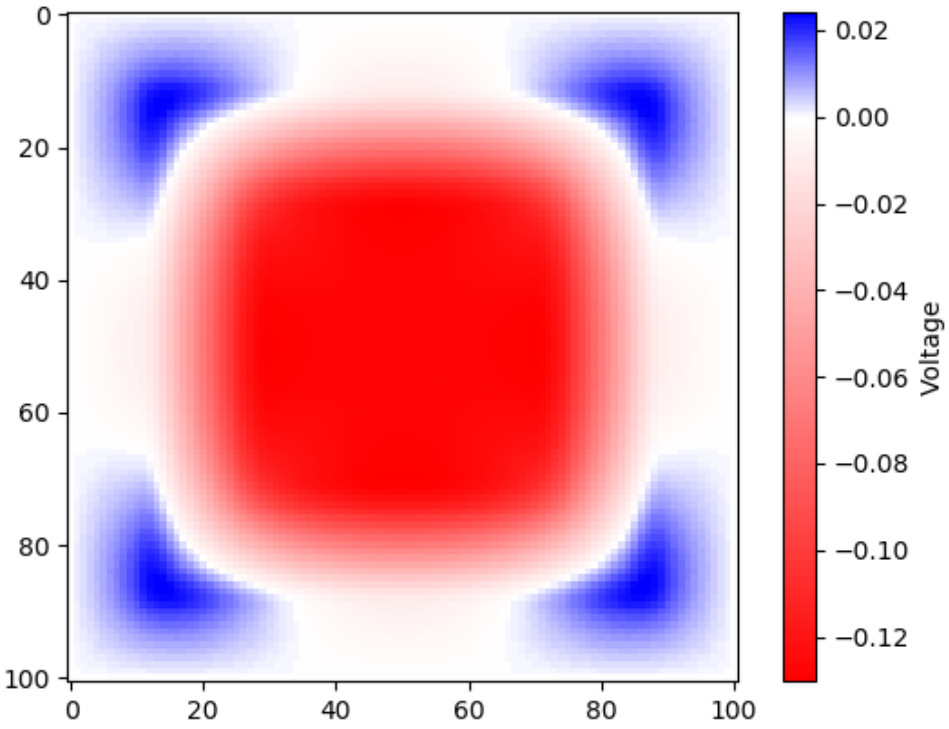

Figure 13.3: The electric potential of the 3 nested pipes.