Solving Poisson's Equation via Finite Difference

In this worksheet you'll use MATLAB's matrix capabilities to solve the most common second-order elliptic PDE - Poisson's equation.

Constructing the Finite Difference Matrix

Let $\Omega = [0, 1] \times [0, 1]$ be the unit square domain, with boundary $\partial\Omega$. We will be using a Finite Difference method to obtain a numerial solution to Poisson's equation on Ω, subject to homogeneous Dirichlet boundary conditions:

$$\begin{aligned} -\nabla^2 u(\mathbf{x}) &= f(\mathbf{x}) \qquad \mathbf{x} \in \Omega \\ u|_{\partial\Omega} &= 0. \end{aligned}$$

The function u is the solution that we seek and f is a known source term.

To begin the Finite Difference implimentation, we first need to choose a *mesh* for Ω, on which we will approximate the solution u.

Taking $N \in \mathrm{N}$ we uniformly discretise in the x- and y- directions with N interior points, so we obtain $N + 2$ points in each direction:

$$x_i = \frac{i}{N+1}, \quad y_j = \frac{j}{N+1} \qquad i, j \in \{0, \ldots, N+1\}$$

and the set of all pairings (x_i, y_j) details the points on the mesh.

It's also useful to introduce the mesh *diameter* or *parameter*, $h = \frac{1}{N+1}$; as well as some convenient notation:

$$u_{i,j} := u\left(x_i, y_j\right) \text{ and } f_{i,j} := f\left(x_i, y_j\right)$$

Next we write down the difference equations to approximate the Laplacian in each direction:

$$\begin{aligned} \frac{\partial^2 u_{i,j}}{\partial x^2} &\approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \\ \frac{\partial^2 u_{i,j}}{\partial y^2} &\approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j-1}}{h^2} \end{aligned}$$

and so our approximate system of equations, which approximate Poisson's equation on our mesh, is

$$\frac{1}{h^2}\left(-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i+1,j+1} - u_{i,j+1}\right) = f\left(x_i, y_j\right), \qquad \forall i, j \in \{1, \ldots, N\}.$$

We only need to determine $u_{i,j}$ at interior mesh points; because of the homogeneous Dirichlet conditions we know that $u_{0,j} = u_{N+1,j} = u_{i,0} = u_{i,N+1} = 0$. Therefore the previous system of equations is a system of N^2 equations in N^2 unknowns, and we can represent the system in matrix form as

$$M\mathbf{U} = \mathbf{F}.$$

Here $\mathbf{U}$ is a vector of the $u_{i,j}$, $\mathbf{F}$ is a vector of the $f_{i,j}$, and M is the finite difference matrix

$$M = \frac{1}{h^2}\left(\text{tridiag}_N(-1, 2, -1) \otimes I_N\right) + \left(I_N \otimes \text{tridiag}_N(-1, 2, -1)\right)$$

where $\otimes$ denotes the Kronecker product and $\text{tridiag}_N(-1, 2, -1)$ denotes the $N \times N$ matrix with 2 on the leading diagonal and -1 on the first super- and sub- diagonals.

The elements of $\mathbf{U}$ and $\mathbf{F}$ are ordered in x then y - this ordering is automatically compatable with MATLAB's reshape command). That is, $\mathbf{U} = \left(u_{1,1}, u_{2,1}, \ldots, u_{N,1}, u_{1,2}, u_{2,2}, \ldots, u_{N-1,N}, u_{N,N}\right)^\top$, and the elements of $\mathbf{F}$ are similar.

Finding an approximation to the true solution u then amounts to solving the linear system $M\mathbf{U} = -\mathbf{F}$ for $\mathbf{U}$, which can be done with MATLAB's `\` operator.

Tasks

Unless otherwise stated, perform all tasks in a single MATLAB script, in the order the tasks are presented. In what follows, variables with names formated `like this` refer to MATLAB variables and/or functions, whereas typeset variables *like this* refer to the mathematical variables above.

📝 1: Constructing M

Read in the file Data.mat - a .mat file containing the number of gridpoints N as well as the arrays `x,y` containing the gridpoints x_i, y_j in each axis. Additionally, `x` and `y` have been turned into a meshgrid (variables `X,Y`) for the interior points using `[X,Y]=meshgrid(x(2:end-1),y(2:end-1))`.

Write a function `FDM(N)` which takes one input N, and returns the finite-difference matrix M defined above. Your function should construct the matrix M efficiently - IE should construct the matrix $\mathrm{tridiag}_{\mathrm{N}}(-1, 2, -1)$ first, using the `diag` function, and then use the `kron` function to build M.

Solution (./03_matlab-ws-soln.html#-1%3A-Constructing-M%0A)

📝 2: Testing Construction

Test that `FDM` works by:

- Extracting the lower and upper parts of the returned matrix, and checking that they are each other's transpose. This process should use `tril` and `triu`. If you want to perform a logic test, the MATLAB command `all` may be useful.
- Sum the rows of M, reshape the resulting column vector to a $N \times N$ matrix. View the result in the variable viewer or using `imagesc`. This should tell you how many edges of the domain contain each node (up to a normalising factor). Does the result you obtain agree with this interpretation?

  Solution (./03_matlab-ws-soln.html#-2%3A-Testing-Construction%0A)

Solving the Poisson Equation

We are interested in the Poisson equation with source term

$$f(x, y) = -4\pi \sin(2\pi x)(\pi(1 + (2y - 1)^2)\cos(2\pi(y - 0.5)^2) + \sin(2\pi(y - 0.5)^2))$$

which results in the analytic solution

$$u(x, y) = \sin(2\pi x)\cos(2\pi(y - 0.5)^2).$$

📝 3: Source and Analytic Functions

Write a function called `F(z)` which, for any $n \in \mathrm{N}$, takes in a $n \times 2$ vector `z`, where each row is a co-ordinate pair, and returns the source term f evaluated at each of the co-ordinate pairs as an $n \times 1$ column vector.

Write an analogous function `Analytic(z)` which evaluates the exact solution (defined above) and returns the result in the same way.

Solution (./03_matlab-ws-soln.html#-3%3A-Source-and-Analytic-Functions%0A)

📝 4: Solving the System

In your script, evaluate `F` and `Analytic` at each of the meshpoints $x_{i,j}$, placing the result into vectors called `source` (for the output of `F`) and `uExact` (`Analytic`).

You now have everything you need to solve the matrix problem - compute the solution to $M\mathbf{U} = -\mathbf{F}$ and store it in a vector `uApprox`. Then:

- Create a surface plot of the solution - to do this you will need to reshape the solution vector `uApprox` into an $N \times N$ matrix (use of the `reshape` command should be sufficient). For ease later, create a variable `uApproxReshape` to store the reshaped matrix, or simply call `reshape` inside the call to `surf`.
- Print the error $||$ `uApprox` - `uExact` $||_2$ to the screen, where $|| \cdot ||_2$ is the vector 2-norm, to 7 decimal places and in scientific format.

TIP: Note that `uApprox` is only finding the solution on the interior of Ω - so don't expect the surface plot to be 0 at the edges!

Solution (./03_matlab-ws-soln.html#-4%3A-Solving-the-System%0A)

5: Error Analysis

Edit your script so that rather than reading in a value of N from a file, the problem instead loops over each value of N in the range $\{5, 10, 25, 50, 100\}$, performing the steps outlined in task 4. Note that for each value of N you will need to construct the arrays `x`, `y`, `X`, and `Y` manually, using `linspace` and `meshgrid`.

Your script should display each approximate solution in a new figure window, rather than overwriting the previous plot. The script should also `pause` after each surface plot is generated, to allow the user to examine the plot. Additionally, your script should also store the error associated with each value of N in a (**preallocated**) column vector `errVec`.

Upon completion of the loop, your script should create semilog plot of the mesh diameter h (which is derived from N) against the error in the solution for that particular mesh. Does what you see coincide with what you expect?

WARNING: The $N = 100$ case may take a long while to run.

TIP: The MATLAB command `close all` can be used to close all currently open figure windows, if you are debugging and want to clear the screen.

Solution (./03_matlab-ws-soln.html#-5%3A-Error-Analysis%0A)