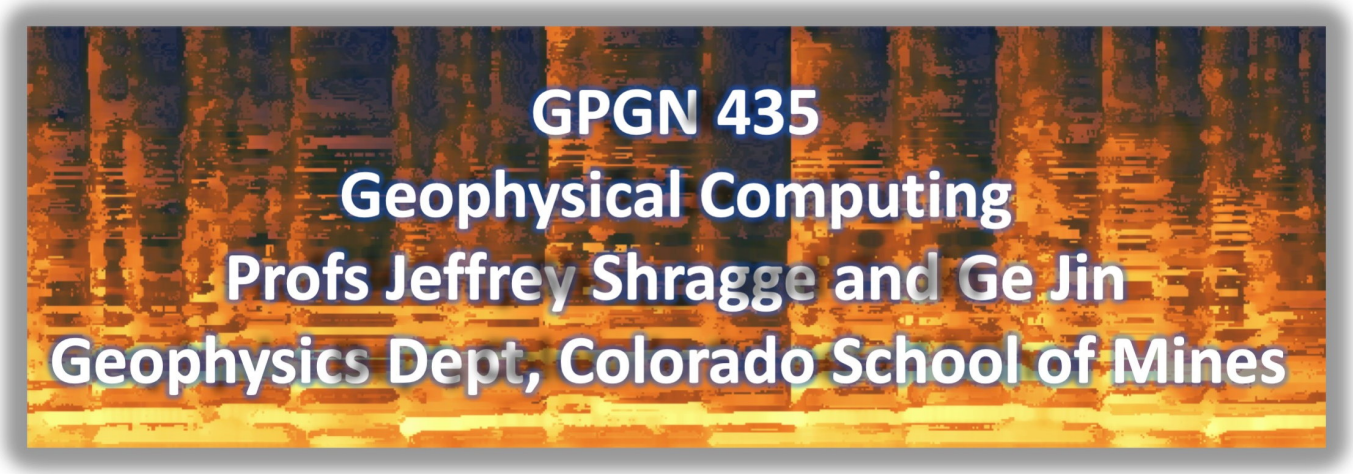
Out[1]: The raw code for this Jupyter notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.



Module 8: Numerical Solutions to Elliptical PDEs

This module is largely focused generating numerical solution of elliptical partial differtial equations (PDEs). Recall that these types of PDEs

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu + G = 0 \tag{1}$$

defined by the condition $B^2 - 2AC < 0$. The problems that arise most commonly in geophysics involve the Cartesian Laplacian operator

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}. \tag{2}$$

Two important examples are:

- **Laplace's Equation**: Used to model steady-state (i.e., time invariant) phenomena where the solution $\phi = \phi(x, y, z)$ is known on the boundary of the solution domain D (e.g., $\phi|_{\partial D} = F(x, y, z)$) and there are no "sources" of the phenomena inside of the domain

$$\nabla^2 \phi = 0. \tag{2b}$$

- **Poisson's Equation**: Used to model steady-state (i.e., time invariant) phenomena where the solution $\phi = \phi(x, y, z)$ is known on the boundary of the solution domain D (e.g., $\phi|_{\partial D} = F(x, y, z)$) where "sources" of the phenomena inside of the domain:

$$\nabla^2 \phi = G(x, y, z). \tag{2c}$$

Examples of the types of problems that can be addressed with these types of equations are:

- Steady-state temperature fields
- Steady-state diffusion processes
- Electrostatics
- Magnetostatics
- Field of constant current density
- Potential flow of an incompressible liquid

Discretization of the 2D Laplace's Equation

Let's now begin to appoach a numerical solution of these PDEs by specifying a discretization of the Laplacian operator operator. By following the Taylor series method developed in the previous notes, we can approximate the second-order partial derivative in the x-direction by a $O(\Delta x^2)$ accurate approximation:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \tag{3a}$$

Similarly, the second-order partial derivative in the y-direction is given by

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} \tag{3b}$$

Assuming that our discretization interval is the same in both directions (i.e., $\Delta x = \Delta y \equiv h$), then this allows us to combine equations 3a and 3b into the following $O(h^2)$ appoximation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = \frac{u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1}}{h^2} = 0 \tag{4}$$

This leads to the following finite-difference stencil:

```
plt.figure(figsize=(6,6)) nx=ny=5 x = np.arange(nx) y=np.zeros(ny) for iy in range(nx): plt.plot(x,y,'ko') y+=1 plt.plot(1,2,'ro',ms=20) plt.plot(3,2,'ro',ms=20) plt.plot([1,3],[2,2],'r-',linewidth=5) plt.plot(2,1,'ro',ms=20) plt.plot(2,3,'ro',ms=20) plt.plot([2,2],[1,3],'r-',linewidth=5) plt.plot(2,2,'bo',ms=20) plt.ylabel('Time Step: n',fontsize=16) plt.yticks(range(5),fontsize=16) plt.xlabel('Space Step: i',fontsize=16) plt.xticks(range(5),fontsize=16) plt.show()
```

Figure 8-1. Second-order Laplacian stencil where red points have a weighting of 1 and the blue point has weighting -4.

Given the finite-difference stencil in equation 4, it is thus trival to state what the **difference equation** (i.e., discrete numerical approximation) of the Laplace equation is:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i,j+1} + u_{i,j-1} = 0. \tag{5}$$

However, unlike for the 1D advection equation, we no longer have a time-evolution problem! Rather, we have a boundary value problem (BVP) given by the following boundary conditions:

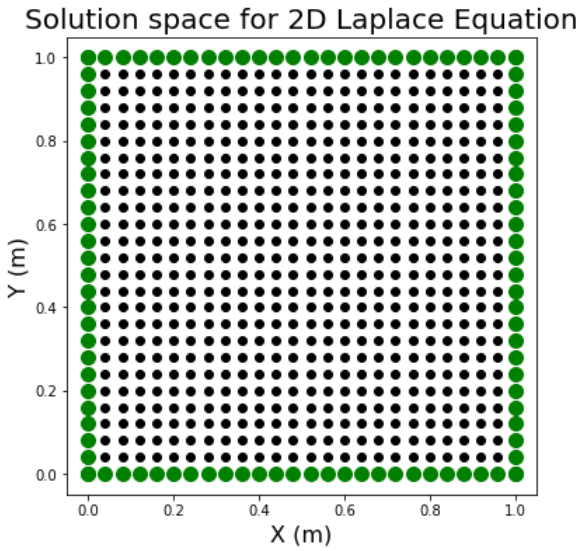


Figure 8-2. Example of a solution space for the 2D Laplace equation. Black points represent locations of unknown solution $u_{i,j}$ while those in green are fixed by the boundary conditions.

Solution by ADI methods

Let's now look at a numerical approach that uses an alternating direction implicit (ADI) method that iteratively updates solutions in one direction (e.g., x) and then in the second direction (y). This is similar to the split-step method discussed in the Lax-Wendroff methods in the previous section of the notes.

Step 1 - Solution in the x direction

Let's first write the Laplacing stencil from above in the following manner:

$$u_{i+1,j} - 4u_{i,j} + u_{i-1,j} = -u_{i,j+1} - u_{i,j-1}. \tag{6a}$$

and now hypothesize that we are introducing a **time-like** step from time level (m) to $(m+1)$, where the brackets are used to indicate that this is time-like ... but not truely time. This gives us the following approximation:

$$u^{(m+1)}_{i+1,j} - 4u^{(m+1)}_{i,j} + u^{(m+1)}_{i-1,j} = -u^{(m)}_{i,j+1} - u^{(m)}_{i,j-1}. \tag{6b}$$

If now we fix row j, we see that this represent a matrix system of equations of the form $\mathbf{Ax} = \mathbf{b}$:

$$\begin{bmatrix} -4 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} u^{(m+1)}_{0,j} \\ u^{(m+1)}_{1,j} \\ u^{(m+1)}_{2,j} \\ \vdots \\ u^{(m+1)}_{I-2,j} \\ u^{(m+1)}_{I-1,j} \\ u^{(m+1)}_{I,j} \end{bmatrix} = \begin{bmatrix} -U^{(m)}_{0,j+1} - U^{(m)}_{0,j-1} \\ -U^{(m)}_{1,j+1} - U^{(m)}_{1,j-1} \\ -U^{(m)}_{2,j+1} - U^{(m)}_{2,j-1} \\ \vdots \\ -U^{(m)}_{I-2,j+1} - U^{(m)}_{I-2,j-1} \\ -U^{(m)}_{I-1,j+1} - U^{(m)}_{I-1,j-1} \\ -U^{(m)}_{I,j+1} - U^{(m)}_{I,j-1} \end{bmatrix}. \tag{7}$$

where matrices $\mathbf{A}$ and $\mathbf{b}$ are known. Thus, we can use some of the methods discussed in numerical linear algebra section (e.g., Gaussian Elimination) to calculate the numerical solution. One then iterates over each j row to generate the solution at pseudo time-step $(m+1)$.

Step 2 - Solution in the y direction

Let's first write the Laplacing stencil from above in the following manner:

$$u_{i,j+1} - 4u_{i,j} + u_{i,j-1} = -u_{i+1,j} - u_{i-1,j}. \tag{8a}$$

and now hypothesize that we are introducing a **time-like** step from time level $(m+1)$ from Step 1 above to $(m+2)$, where the brackets again indicate that this is staggered in time. This gives us the following approximation:

$$u^{(m+2)}_{i,j+2} - 4u^{(m+2)}_{i,j} + u^{(m+1)}_{i,j-1} = -u^{(m+1)}_{i+1,j} - u^{(m+1)}_{i-1,j}. \tag{8b}$$

If now we fix row j, we see that this represent a matrix system of equations of the form $\mathbf{Ax} = \mathbf{b}$:

$$\begin{bmatrix} -4 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} u^{(m+2)}_{i,0} \\ u^{(m+2)}_{i,1} \\ u^{(m+2)}_{i,2} \\ \vdots \\ u^{(m+1)}_{i,J-2} \\ u^{(m+1)}_{i,J-1} \\ u^{(m+1)}_{i,J} \end{bmatrix} = \begin{bmatrix} -U^{(m+1)}_{i+1,0} - U^{(m+1)}_{i-1,0} \\ -U^{(m+1)}_{i+1,1} - U^{(m+1)}_{i-1,1} \\ -U^{(m+1)}_{i+1,2} - U^{(m+1)}_{i-1,2} \\ \vdots \\ -U^{(m+1)}_{i+1,J-2} - U^{(m+1)}_{i-1,J-2} \\ -U^{(m+1)}_{i+1,J-1} - U^{(m+1)}_{i-1,J-1} \\ -U^{(m+1)}_{i+1,J} - U^{(m+1)}_{i-1,J} \end{bmatrix}. \tag{9}$$

where matrices $\mathbf{A}$ and $\mathbf{b}$ again are known. One then iterates over each i column to generate the solution at pseudo time-step $(m+2)$.

Example 8-1 - Laplace's Equation with Variable Boundary Conditions

QUESTION: Let's look at an example where we want to generate a solution u to the 2D Laplace's equation

$$\nabla^2 u = 0 \tag{10}$$

on a rectilinear solution domain defined by $x, y \in [0, 3]$ where we have the following boundary conditions:

$$u(x, y = 0) = x^4 \tag{11a}$$
$$u(x, y = 3) = x^4 - 54x^2 + 81 \tag{11b}$$
$$u(x = 0, y) = y^4 \tag{11c}$$
$$u(x = 3, y) = y^4 - 54y^2 + 81 \tag{11d}$$

where the known analytic solution throughout the domain is given by:

$$u(x, y) = x^4 - 6x^2y^2 + y^4 \tag{12}$$

ANSWER: Let's first make it easy on ourselves by define a few helpful functions. The subroutine below defines a tridiagonal matrix where the value of the main diagonal is b, and the sub- and superdiagonals are a and c, respectively, and the dimensional of the $n \times n$ matrix is n.

Let's make a quick plot to visualize the function:

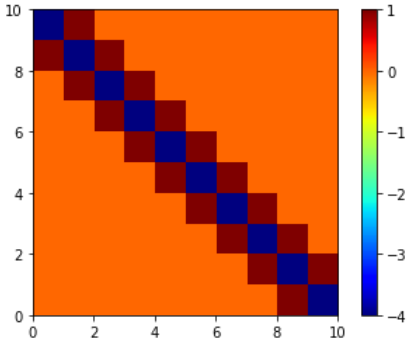


Figure 8-3. Looking at the tridiagonal matrix required for the 2D ADI solution.

Let's now develop the subroutine that computes the ADI solution. Below I have witten a function that takes in current solution U_0 and outputs the solution after doing one iteration of the ADI method in both directions.

Let's now setup the solution space, state the boundary conditions, and call the solver.

We can now plot the movie. Here, I've plotted the analytic solution (left), the numerical solution (center) and the difference between the two (right).

Out[7]:

0:00 / 0:09

Figure 8-4. Movie showing the time evolution of the ADI solution. Left: Analytic solution $u(x,y) = x^4 - 6x^2y^2 + y^4$. Center: Numerical solution animated at each solution step. Right: difference between the analytic and numerical solutions.

Now, if the analytic and numerical solutions are given by $a_{i,j}$ and $u_{i,j}^{(2m)}$, respectively, we can examine at how the overall error

$$err(m) = \sqrt{\sum_{i,j} \left(a_{i,j} - u_{i,j}^{(2m)}\right)^2} \tag{10}$$

changes as a function of iteration number. Here, I have plotted the error value as a function of step m normalized by the erro at the first iteration $err(0)$.

```
<matplotlib.figure.Figure at 0x1128a4240>
```

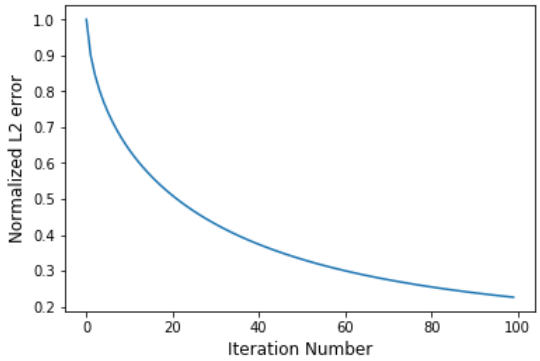


Figure 8-5. Least-squares error between the analytic and numerical solution. Note that it hasn't converged.

Example 8-2 - Poisson's Equation (Steady-state Heat with Sources)

QUESTION: Let's look at an example where we want to generate a solution to the 2D Poisson's equation for steady-state heat flow where there are heat sinks and sources within the solution domain. Here, we want to estimate the temperatue (i.e., where u = T) due to the distribution of heat sources and sinks. The PDE governing this is the following:

$$\nabla^2 T = F(x,y) \tag{10}$$

where we ae going to assume a rectilinear solution domain defined by $x, y \in [0,1]$ where we have the following boundary conditions:

$$T(x, y=0) = 0 \tag{11a}$$
$$T(x, y=1) = 0 \tag{11b}$$
$$T(x=0, y) = 0 \tag{11c}$$
$$T(x=1, y) = 0 \tag{11d}$$

and a forcing term $F(x,y)$ given by four delta-function-like heaters and coolerrs

$$F(x,y) = \left[\delta\left(x-\frac{1}{4}\right)\delta\left(y-\frac{1}{4}\right) - \delta\left(x-\frac{3}{4}\right)\delta\left(y-\frac{1}{4}\right) - \delta\left(x-\frac{1}{4}\right)\delta\left(y-\frac{3}{4}\right) + \delta\left(x-\frac{3}{4}\right)\delta\left(y-\frac{3}{4}\right)\right]. \tag{12}$$

ANSWER: We can reuse much of the machinary that we developed above in our solution to Laplace's equation. However, this time we need to be careful about satisifying the forcing term $F(x,y) = F_{i,j}$. Thus, we write

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \approx \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = \frac{T_{i+1,j} + T_{i-1,j} - TV_{i,j} + T_{i,j+1} + V_{i,j-1}}{h^2} = F_{i,j} \tag{13}$$

We can then develop the following two finite-difference equations that can be used in the two-step ADI method above

$$T_{i+1,j}^{(m+1)} - 4T_{i,j}^{(m+1)} + T_{i-1,j}^{(m+1)} = h^2 F_{i,j} - T_{i,j+1}^{(m)} - T_{i,j-1}^{(m)} \tag{14a}$$
$$T_{i,j+1}^{(m+2)} - 4T_{i,j}^{(m+2)} + T_{i,j-1}^{(m+2)} = h^2 F_{i,j} - T_{i+1,j}^{(m+1)} - T_{i-1,j}^{(m+1)} \tag{14b}$$

Let's first adapt what we had above to include the forcing term $F(x,y)$.

Let's now define our solution domain, discretization, and solve below:

```
0.0002956714624787456 -0.00029224288458615537
```

Out[11]:

0:00 / 0:19

Figure 8-6. Movie showing the convergence of the numerical solution to Poisson's equation for electric potential V assuming four point charges and $V = 0$ on the boundaries.

We can also look at the rate at which the solution is change, which is a measure of the **rate of convergence**. In the figure below, we see that it is change rapidly in the first say 5 iterations, but then is not changing much after this. As we will see in the lab, a good way to end the simulation is the include both the maximum number of iterations as well as a **stopping criterion** when the overall solution is no longer changing more than some small number ϵ.

```
<matplotlib.figure.Figure at 0x1129349e8>
```

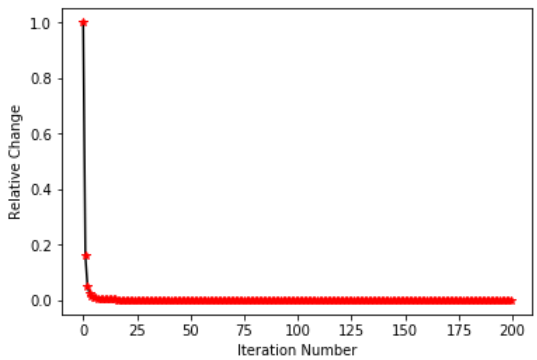


Figure 8-7. Convergence plot showing the relative change between the $(m + 2)$ and (m) solutions of the ADI solution method. Note that the majority of the change occurs within the first five iterations. However, by watching the movie we see that much information is actually filled during the later stages!

Example 8-3 - Including Heterogenity

So far, we have not really looked at situations where there might be heterogeneity in the medium. A common example would be what happens in steady-state processes (e.g., heat flow, diffusion processes) when one has material that is **spatially varying**? Let's again consider the scenario where we are examining steady-state heat flow.

In these cases, we must look at a slightly different version of the Laplace's (or Poisson's) equation from that presented above:

$$\nabla \cdot (K \nabla T) = 0 \tag{15a}$$

or explictly in 2D

$$\frac{\partial}{\partial x}\left(K\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(K\frac{\partial u}{\partial y}\right) = 0, \tag{15b}$$

where $K = K(x, y)$ is a spatially heterogeneous thermal conductivity field. In these cases we can do something a bit different that involves taking a **second-order PDF** and breaking it down into **two first-order PDEs**. If we let **auxilliary variables** v and w be defined in the following way:

$$v = K\frac{\partial u}{\partial x}, \tag{16a}$$

$$w = K\frac{\partial u}{\partial y}, \tag{16b}$$

then we can use them to calculate the following:

$$\frac{\partial v}{\partial x} + \frac{\partial w}{\partial y} = 0, \tag{17}$$

which will satisfy Laplace's equation for heterogenous fields.

Numerical solution

Let's write equation 16a and 16b in the following way that uses forward differences for the gradient operator:

$$v_{i,j} = K_{i,j}\left(\frac{u_{i+1,j} - u_{ij}}{h}\right) \tag{18a}$$

$$w_{i,j} = K_{i,j}\left(\frac{u_{i,j+1} - u_{ij}}{h}\right) \tag{18b}$$

we can then apply a backward difference when we apply the divergence operator:

$$\frac{v_{ij} - v_{i-1,j}}{h} + \frac{w_{ij} - w_{i,j-1}}{h} = 0 \tag{19}$$

However, we know the value of these four numerical approximations in terms of $u_{i,j}$. Thus, we can rewrite equation 19 as

$$\frac{K_{ij}\left(\frac{u_{i+1,j}-u_{ij}}{h}\right) - K_{i-1,j}\left(\frac{u_{ij}-u_{i-1,j}}{h}\right)}{h} + \frac{K_{ij}\left(\frac{u_{i,j+1}-u_{ij}}{h}\right) - K_{ij-1}\left(\frac{u_{ij}-u_{ij-1}}{h}\right)}{h} = 0 \tag{20}$$

Noting that all of h terms can be eliminated simplifies equation 20 to:

$$K_{ij}\left(u_{i+1,j} - u_{ij}\right) - K_{i-1,j}\left(u_{ij} - u_{i-1,j}\right) + K_{ij}\left(u_{i,j+1} - u_{ij}\right) - K_{ij-1}\left(u_{ij} - u_{ij-1}\right) = 0 \tag{21}$$

Regrouping terms leads to the following:

$$K_{ij}u_{i+1,j} - K_{ij}u_{ij} - K_{i-1,j}u_{ij} + K_{i-1,j}u_{i-1,j} + K_{ij}u_{i,j+1} - K_{ij}u_{ij} - K_{ij-1}u_{ij} + K_{ij-1}u_{ij-1} = 0 \tag{22}$$

Collecting terms for $u_{i,j}$ gives:

$$K_{ij}u_{i+1,j} + K_{i-1,j}u_{i-1,j} - \left(2K_{ij} + K_{i-1,j} + K_{ij-1}\right)u_{ij} + K_{ij}u_{i,j+1} + K_{ij-1}u_{ij-1} = 0. \tag{23}$$

Note that if K is constant then we recover exactly what we had in equation 4 above!

ADI Solution

Let's follow our solution approach above by alternating the direction of the solution. Thus, in the x-direction at pseudo-timestep $(m+1)$ we have

$$K_{ij}u_{i+1,j}^{(m+1)} + K_{i-1,j}u_{i-1,j}^{(m+1)} - \left(2K_{ij} + K_{i-1,j} + K_{ij-1}\right)u_{ij}^{(m+1)} = -K_{ij}u_{i,j+1}^{(m)} - K_{ij-1}u_{ij-1}^{(m)}. \tag{24}$$

Similaly, in the y-direction at pseudo-timestep $(m+2)$ we will have

$$K_{ij}u_{i,j+1}^{(m+2)} + K_{ij-1}u_{i,j-1}^{(m+2)} - \left(2K_{ij} + K_{i-1,j} + K_{ij-1}\right)u_{ij}^{(m+2)} = -K_{ij}u_{i+1,j}^{(m+1)} - K_{i-1,j}u_{i-1,j}^{(m+1)}. \tag{25}$$

Evidently, we now no longer have spatially constant coefficients. However, because we still have the same structure, we can still solve using a similar approach! Let's first create a subroutine that allows us to set up a tridiagonal system, but with heterogeneous coefficients.

We can then modify our solution mechanism above to incorporate the spatially varying coefficients.

Out[16]: