Lesson 4: Numerical 1-D Conduction with Matrices

Show Blackboard system & HW#1

We initially presented content in this class through the Engineering Equation Solver (EES) software program. This was done because EES is the easiest solver there is. It is an implicit equation solver so EES implicitly solves a system of equations, or whether linear or nonlinear.

Notice that the order of variables & equations is irrelevant— I can program the equations exactly as derived with minimal algebra. EES does this via implementation of Jacobi's method.

Jacobi's method builds a system of matrices $A\mathbf{x} = \mathbf{b}$

where $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$

$A$ is then decomposed into diagonal & non-diagonal elements

$A = D + R$ where $D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}$ and $R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$

The solution is then obtained by iteration for

$x^{(k+1)} = D^{-1}(b - Rx^{(k)})$ or $x^{(k+1)}_i = \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij}\cdot x^{(k)}_j)$
Because of the way EES parses and builds these matrices, the order is important, and a considerable amount of effort is required to iterate continuously on variables that may have minimal change. So you sacrifice speed for ease with EES. With numerical heat transfer analyses we’ll often push EES to the limits. This is where more formal programming languages like MATLAB and have an advantage.

Numerical Solutions in Matrix Format

MATLAB is an explicit solver. Everything on one side of an equation must be known in advance and must equal a single unknown on the other side of the equation. No iteration required. But it requires some additional work on our part. Let’s reconsider the Rayleigh-Bénard problem from last time:

\[ \frac{dT}{dt} = \frac{g}{R} \left( T - T_0 \right) \]

Where \( \Delta T = (T_i - T_f) \) and \( \Delta T' = \frac{R}{R} \left( T_i - T_f \right) \). Our energy balance was

\[ \sum n_h \frac{q_i}{\Delta T} + \sum n_h \frac{q_i}{\Delta T'} = 0 \]

Which are combined into:

\[ \sum \frac{2\pi \Delta T}{\Delta T} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) + \sum \frac{2\pi \Delta T}{\Delta T'} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) + \sum \frac{2\pi \Delta T}{\Delta T'} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) = 0 \]

**Node 1:**

\[ \frac{2\pi \Delta T}{\Delta T} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) + \frac{2\pi \Delta T}{\Delta T'} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) + \sum \frac{2\pi \Delta T}{\Delta T'} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) = 0 \]

**Node N:**

\[ \frac{2\pi \Delta T}{\Delta T} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) + \frac{2\pi \Delta T}{\Delta T'} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) + \sum \frac{2\pi \Delta T}{\Delta T'} \left( \frac{h}{2} + \frac{h}{2} \right) \left( T_i - T_f \right) = 0 \]
This linear system of equations with unknowns can be easily placed into matrix format: \[ A \mathbf{x} = \mathbf{b} \] where \( A \) is a matrix, \( \mathbf{x} \) are vectors, and \( \mathbf{b} \) are constants.

\[ A = \begin{bmatrix} \text{Row 1: Control volume equation 1} \\ \text{Row 2: Control volume equation 2} \\ \vdots \\ \text{Row N: Control volume equation N} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_N \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \]

to do this we need to manipulate our equations

For Node 1:
\[ \frac{T_1}{\Delta r} \left[ -2\frac{\partial T}{\partial r}(\frac{r_1}{2}) + \frac{T}{\Delta r} \right] + \frac{T}{2} \left[ 2\frac{\partial T}{\partial r}(\frac{r_1}{2}) \right] = -q'''' \Delta r \]

For Internal Nodes:
\[ \frac{T_i}{\Delta r} \left[ -2\frac{\partial T}{\partial r}(i - \frac{\Delta r}{2}) - \frac{T_i}{\Delta r} \right] + \frac{T}{2} \left[ 2\frac{\partial T}{\partial r}(i + \frac{\Delta r}{2}) \right] = -q'''' \Delta r \]

For Node N:
\[ \frac{T_N}{\Delta r} \left[ -2\frac{\partial T}{\partial r}(N - \frac{\Delta r}{2}) - \frac{1}{\text{Roop Thm}} \right] + \frac{T}{2} \left[ 2\frac{\partial T}{\partial r}(N - \frac{\Delta r}{2}) \right] = -\frac{T_N}{\text{Roop Thm}} \]

Where the solution is then
\[ \mathbf{x} = A^{-1} \mathbf{b} \]

Show example code.
While this code may be fast, it is not quite as flexible as EES. Temperature dependent properties are non-linear which is easy for EES but harder for explicit solvers like Matlab because the equations can no longer be solved without iteration. Here's the process:

**Step 1:** Assume a temperature distribution \( T_i \) for \( i = 1 \ldots N \)

**Step 2:** Solve \( \theta_i = T_i - T_{i-1} \) to solve for an unknown coefficient \( \theta_i \)

**Step 3:** Solve \( A \mathbf{x} = \mathbf{b} \) to order to predict \( T_i \) for \( i = 1 \ldots N \)

**Step 4:** Compute convergence error \( \varepsilon = \sqrt{\sum_{i=1}^{N} \left( \frac{T_i - T_{i-1}}{T_i} \right)^2} \)

**Step 5:** Iterate steps 2-4 until convergence tolerance is met

*In general, you'll have to select your modeling software considering*:

**Computational Flexibility**

- ANSYS
- StarSolve
- EES
- Octave
- Matlab
- Mathematica
- Maple

**Speed**

- Analytical