ME-474 Numerical Flow Simulation

Exercise: Nonlinearity - Solution of the linear system

Fall 2021

1 Nonlinearity

Implement a FVM code in Matlab to solve the 1D steady diffusion eq. with a nonlinear source term,

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + S(T) = 0, \qquad S(T) = 4 - 5T^3.$$

The domain is $x \in [0, L]$, $L = 1$ m. Dirichlet boundary conditions $T(0) = T_a = 20$ K, $T(L) = T_b = 40$ K are imposed. The thermal conductivity $k = 400$ W/(K.m) is constant.

1. Implement both Picard's method and Newton's method to linearize the source term as $S(T) \approx S_c(T^*) + S_l(T^*)T$, with T^* the current guess value (solution at the current iteration). You can reuse the code from week 2 with a linear source term $S(T) = S_c + S_l T$. To evaluate convergence, you can use some measure of the solution variation from one iteration to the next.

2. Compare your converged solution to that obtained with Matlab's function `bvp4c`. Look at Matlab's help about `bvp4c` to understand how to define the equation and boundary conditions.

2 Solution of the linear system: direct methods

Implement a direct method to solve the linear system: reuse the code from week 2 to solve the 1D steady diffusion eq. with the linear source term $S(T) = 5000 - 100T$ W/m^3 and Dirichlet boundary conditions $T(0) = T_a = 300$ K, $T(L) = T_b = 320$ K, but now inverting the system with the TDMA method. Try different meshes, e.g. $n = 10$, 20, 50, 100, 500, 1000, 5000 and 10000.

Compare your solution with the theoretical one. Make a log-log plot of the mean error

$$||T - T_{theo}|| = \frac{1}{n}\sum_{i=1}^{n} |T_i - T_{theo,i}|$$

as a function of n. Do the same thing with Matlab's built-in solvers `T=inv(A)*b` (bad) and `T=A\b` (good), and compare with TDMA.

As an alternative measure of accuracy, compute the normalized residuals $||\mathbf{r}||/||\mathrm{diag}(\mathbf{A})\mathbf{T}||$ and make a log-log plot as a function of n. Compare again TDMA and Matlab's built-in solvers.

3 Solution of the linear system: iterative methods

Solve the same problem as question 2 above, but now inverting the linear system with iterative methods.

3.1 Point-iterative methods

Implement (1) the Jacobi method, (2) the Gauss-Seidel method, and (3) the SOR method.

Try different meshes, e.g. $n = 10$, 20, 50, 100 and 200. Do the same thing with Matlab's built-in solvers. Again, compare the solution with the theoretical one (plot the mean error vs. n), and plot the normalized residuals vs. n. Plot as well the number of iterations needed for convergence, as a function of n.

For the SOR method, change the value of the relaxation factor ω and observe its effect on the number of iterations needed for convergence.

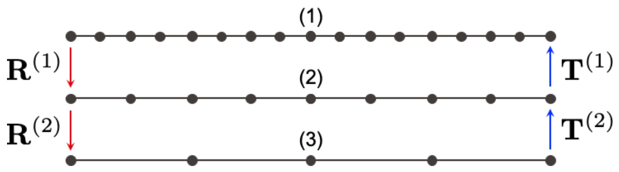
3.2 Multigrid method (optional)

Implement a multigrid method with V-cycles on 3 uniform meshes: (1) original = finest, $n = 17$; (2) intermediate, $n = 9$; (3) coarsest, $n = 5$. To save some time, you can use and fill in the incomplete code provided on Moodle.

In that code, the restriction and prolongation operators are already defined. The original operator $\mathbf{A} = \mathbf{A}^{(1)}$ of the linear system is interpolated onto meshes (2) and (3) as

$$\mathbf{A}^{(2)} = \mathbf{R}^{(1)}\mathbf{A}^{(1)}\mathbf{T}^{(1)}, \qquad \mathbf{A}^{(3)} = \mathbf{R}^{(2)}\mathbf{A}^{(2)}\mathbf{T}^{(2)},$$

with $\mathbf{R}^{(i)}$ the restriction operator from mesh (i) to mesh $(i+1)$, and $\mathbf{T}^{(i)}$ the prolongation operator from mesh $(i+1)$ to mesh (i).



In each V-cycle, use only a few Gauss-Seidel iterations (say two or three) on the finer meshes (1) and (2), and use as many iterations as needed for convergence on the coarsest mesh (3). It's convenient to define two sub-functions `GS_ONE(...)` (for one GS iteration) and `GS(...)` (for full GS solution) that you will call at the different steps of the V-cycle. (Reuse part of your code from question 3.1.) Repeat the whole V-cycle until convergence.

Remember that on the finest mesh (1) you solve the actual system $\mathbf{A}\boldsymbol{\phi} = \mathbf{b}$, whereas on each coarser mesh (i) you solve the equation $\mathbf{Ae} = \mathbf{r}$, with $\mathbf{e}$ and $\mathbf{r}$ the error vector and residual vector of the equation solved on mesh $(i-1)$. So here, one V-cycle looks like this:

$$\begin{aligned}
\text{mesh (1)}: \quad & \text{solve for the solution } \boldsymbol{\phi}: \ \mathbf{A}^{(1)}\boldsymbol{\phi}^{(1)} = \mathbf{b}, \\
& \ \to \text{residual } \mathbf{r}^{(1)} = \mathbf{b} - \mathbf{A}^{(1)}\boldsymbol{\phi}^{(1)}, \\
\text{mesh (2)}: \quad & \text{interpolate the residual: } \mathbf{r}^{(2)} = \mathbf{R}^{(1)}\mathbf{r}^{(1)}, \\
& \text{solve for the error of the upper-level equation: } \mathbf{A}^{(2)}\mathbf{e}^{(2)} = \mathbf{r}^{(2)}, \\
& \ \to \text{residual } \mathbf{s}^{(2)} = \mathbf{r}^{(2)} - \mathbf{A}^{(2)}\mathbf{e}^{(2)}, \\
\text{mesh (3)}: \quad & \text{interpolate the residual: } \mathbf{s}^{(3)} = \mathbf{R}^{(2)}\mathbf{s}^{(2)}, \\
& \text{solve for the error of the upper-level equation: } \mathbf{A}^{(3)}\mathbf{f}^{(3)} = \mathbf{s}^{(3)}, \\
\text{mesh (2)}: \quad & \text{interpolate back the lower-level error: } \mathbf{f}^{(2)} = \mathbf{T}^{(2)}\mathbf{f}^{(3)}, \\
& \text{update the error: } \mathbf{e}^{(2)}_{new} = \mathbf{e}^{(2)} + \mathbf{f}^{(2)}, \\
& \text{solve for the error } \mathbf{A}^{(2)}\mathbf{e}^{(2)} = \mathbf{r}^{(2)}, \text{ from the updated guess } \mathbf{e}^{(2)}_{new}, \\
\text{mesh (1)}: \quad & \text{interpolate the lower-level error: } \mathbf{e}^{(1)} = \mathbf{T}^{(1)}\mathbf{e}^{(2)}, \\
& \text{update the solution: } \boldsymbol{\phi}^{(1)}_{new} = \boldsymbol{\phi}^{(1)} + \mathbf{e}^{(1)}, \\
& \text{solve for } \boldsymbol{\phi}: \ \mathbf{A}^{(1)}\boldsymbol{\phi}^{(1)} = \mathbf{b}, \text{ from the updated guess } \boldsymbol{\phi}^{(1)}_{new}.
\end{aligned}$$

It's actually possible, more compact and more flexible to code all those steps recursively, i.e. with a function that performs the steps at level (i), and then calls itself to perform the steps of level $(i+1)$. For simplicity, in the code provided, those steps are coded explicitly (sequentially).