ME-474 Numerical Flow Simulation

Comments on exercise: Nonlinearity and solution of the linear system

Fall 2021

Nonlinearity

In section 1 the solution of the 1D steady diffusion equation with nonlinear source term is required. The nonlinear term can be linearized using different schemes, e.g. Picard's or Newton's method. The resulting linear system is then solved iteratively until the convergence criterion is satisfied.

Newton's method, which is based on a Taylor expansion of the nonlinear term, can be applied to any nonlinear function. This is not the case for Picard's method. Indeed, the latter can be applied to quasi-linear functions only. In the present case, the nonlinear source term is $S(T) = 4 - 5T^3$, which can easily be linearized as $\tilde{S}(T^*)T = (4 - 5T^{*2})T$. However, this is not always possible. For example, $S(T) = \sin(T)$ can not be written in a quasi-linear form. In such a case, a possible alternative approach is to consider the nonlinear term as a fully explicit term (function of the solution obtained at the previous iteration), i.e. $S(T) \rightarrow \tilde{S}(T^*) = \sin(T^*)$. In general, Newton method is preferable, not only for its applicability to any nonlinear problem, but also for its faster convergence.

A crucial point in the solution of nonlinear problem via iterative methods is the choice of a reasonable initial guess for the iterative loop. Unfortunately, there is no criterion to determine the optimal initial guess for a given problem. Nevertheless, we can tentatively select some possible good choices. For example, the value of the unknown at the boundaries is defined by the boundary conditions of the problem. Therefore, a possible choice is a simple function, which satisfies the boundary conditions, e.g. linear $T_{guess}(x) = (T_b - T_a)x/L + T_a$, or quadratic, etc.

Another possibility, particularly useful when a simple analytical initial guess can not be easily identified, is to solve the original system neglecting nonlinearities; the solution of this simplified linear system will be then used as initial guess for the iterative algorithm in the original nonlinear problem.

If the nonlinearities are too strong, all these approaches could fail when a non adequate initial guess is chosen. In such a case, one could progressively introduce the nonlinearities in the system, for example writing $S(T) = 4 - c\ 5T^3$, with $0 < c < 1$. In this way, starting from the linear case ($c = 0$), we can solve the nonlinear system increasing the parameter c and using as initial guess the final solution obtained for the previous smaller value of c. We will finally get the desired solution when $c = 1$.

Solution of the linear system: direct methods

Note that `inv(A)*b` is slower and less accurate than `A\b`. The second command should be preferred whenever solving a linear system, and when the inverse of the matrix doesn't need to be computed explicitly. This is clearly explained in Matlab's help.

Although the details are different, the following simple analogy is interesting: when you want to compute 3/7, you don't compute 1/7 explicitly and then multiply by 3, you directly perform the division.

Solution of the linear system: iterative methods

We have seen how a linear system can be solved either directly or iteratively. The idea behind iterative methods is to replace the exact solution with a good approximation of the solution in order to reduce the overall problem complexity, which is normally $O(n^3)$ for direct methods. Only in special cases, e.g. when the system is represented by a tri-diagonal matrix, the complexity is $O(n)$ and we can reasonably use direct

algorithms as TDMA.
In section 3 the linear system is required to be solved via iterative techniques, e.g. Jacobi, Gauss-Seidel and SOR methods, whose complexity at each step is generally $O(n^2)$. We briefly recall here the formula for these three different methods:

$$\text{Jacobi}: \qquad \phi_i^{(k)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} \phi_j^{(k-1)} \right), \tag{1}$$

$$\text{GS}: \qquad \phi_i^{(k)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j<i} a_{i,j} \phi_j^{(k)} - \sum_{j>i} a_{i,j} \phi_j^{(k-1)} \right), \tag{2}$$

$$\text{SOR}: \qquad \phi_i^{(k)} = (1-\omega)\phi_i^{(k-1)} + \frac{\omega}{a_{i,i}} \left(b_i - \sum_{j<i} a_{i,j} \phi_j^{(k)} - \sum_{j>i} a_{i,j} \phi_j^{(k-1)} \right), \qquad 0 < \omega < 2. \tag{3}$$

This is what is implemented in the solution code.

An alternative and practical way to implement the three algorithms above is the following: considering the linear system $A\phi = b$, we decompose the matrix A into two parts, $\boxed{A = P - N}$. Then we construct the iterative process as

$$\boxed{\phi^{(k)} = P^{-1} \left(N\phi^{(k-1)} + b \right)}. \tag{4}$$

The definitions of matrices P and N for the three different cases are given below. Defining the following three sub-matrices, D, L and U,

$$\underbrace{\begin{bmatrix} a & a & a \\ a & a & a \\ a & a & a \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} d & & \\ & d & \\ & & d \end{bmatrix}}_{D=\text{diag}(A)} - \underbrace{\begin{bmatrix} & & \\ l & & \\ l & l & \end{bmatrix}}_{L=-\text{tril}(A,-1)} - \underbrace{\begin{bmatrix} & u & u \\ & & u \\ & & \end{bmatrix}}_{U=-\text{triu}(A,+1)} \tag{5}$$

we have,

- Jacobi

$$\boxed{P_{\text{Jacobi}} = D, \qquad N_{\text{Jacobi}} = L + U,} \tag{6}$$

- GS

$$\boxed{P_{\text{GS}} = D - L, \qquad N_{\text{GS}} = U,} \tag{7}$$

- SOR

$$\boxed{P_{\text{SOR}} = \frac{1}{\omega} \left(D - \omega L \right), \qquad N_{\text{SOR}} = \left[\left(\frac{1}{\omega} - 1 \right) D + U \right], \qquad 0 < \omega < 2.} \tag{8}$$

Once the matrices P and N are built, we simply solve equation (4). Note that the compact matrix formulations above are formally equivalent to the index formulation given in equations (1)-(2)-(3).