Some notes about PDEs

-Bill Green Nov. 2015

Partial differential equations (PDEs) are all BVPs, with the same issues about specifying boundary conditions etc. Because they are multi-dimensional, they can be very CPU intensive to solve, similar to multidimensional integrals.

For example, a 3-d pde (e.g. steady-state Navier Stokes) will typically require a mesh of at least $(100)^3 = 10^6$ points, so one is computing at least $10^6 * N_{variables}$ unknowns, where the unknowns might be (for a compressible reacting flow) [$\rho$, T, $v_x$, $v_y$, $v_z$, y1, y2, …] where $y_i$ is the mass fraction of species i. So $N_{variables} = N_{species}+5$. As the Reynolds number increases the mesh density has to increase, so this quickly becomes unmanageable.

Note that while we often can reduce problems to manageable 2-d PDEs taking advantage of symmetry or making approximations, many real-world PDEs have dimensionality>3 (e.g. the Schroedinger equation, time-dependent Navier-Stokes, etc.).

Like ODEs, certain PDEs are intrinsically unstable (e.g. because the physical situation is extremely sensitive to small fluctuations), and those are challenging-to-impossible to solve accurately using numerical methods, since numerical errors get amplified. One example is detonation, where a small initiation event gets amplified into a strong shock wave. Another is the growth of a tumor or a bacterial culture, where a mutation or other small change in one cell can lead to a complete change in morphology and composition of the system at later times.

Another class of PDEs, called “hyperbolic” (see Beers page 276 for the mathematical definition) have solutions that are propagating waves. Any small fluctuation at early times propagates to later times, often with no or very minor damping. Examples include acoustics and Maxwell’s equations. Special methods are needed to model these, to avoid having the undamped numerical errors accumulate into a lot of noise in the answer. If hyperbolic problems are solved using naïve numerical solution methods, one often obtains unphysical oscillations in the numerical solution.

In 10.34 we focus primarily on “elliptic” and “parabolic” PDEs, where the physics (e.g. diffusion, viscosity) dampens both the real fluctuations, and the numerical noise introduced during the solution.

The conceptual difference between hyperbolic, parabolic, and elliptic PDEs has to do with the flow of information. If I adjust the value of a state variable at one mesh point, does that adjustment affect the computed value at other points, i.e. does point X know what is happening at point Y? In the case of elliptic PDEs, each point is sensitive to all the others. In hyperbolic PDEs, there are regions that are not sensitive to each other at all (e.g. a supersonic shock wave is propagating so fast that it cannot feel pressure fluctuations happening behind it at all, since information about them only moves at the speed of sound). Parabolic PDEs have all points in the future sensitive to everything that happened in the past, but not vice-versa.

For problems with significant convection, what happens downwind is very much controlled by what already happened upwind. In that case, it is much better to use "upwind differencing" than centered-differencing. Only if there is real information flow from downwind should the downwind values affect upwind values in the numerical procedure. Contrast Fig. 6.7 and Fig. 6.8 in Beers' textbook, to see how important this is for avoiding numerical instabilities.

## Methods for solving PDEs

One can directly apply all the relaxation methods (collocation, Galerkin, finite differences) much the same as in ODE-BVPs, converting the PDE into a large system of nonlinear algebraic equations.

$$F(c)=0$$

This is how most elliptic PDEs are solved. The main challenge is the huge number of unknowns, and the size of the corresponding Jacobian matrix. Using a method that makes a very sparse Jacobian (e.g. using a local basis set) is key. Finding a sufficiently good initial guess can also be very challenging. There are many special methods developed to try to handle these huge systems.

Note that the key step at each iteration in solving a system of nonlinear equations is to solve the equation

$$J*\Delta c = - F$$

Normally we would solve this by Gaussian elimination. However, when J is huge this is not practical, e.g. we may not have enough memory to even store the intermediate matrices (remember the problem of Gaussian fill-in, so even if J is sparse the intermediate matrices will not be so sparse). So instead we would like to solve this equation using "direct" methods, which do not require storing any large intermediate objects. Several methods have been proposed to do this. One of the most successful is Conjugate Gradient, where one rewrites the problem as a minimization

$$\min q = (F+J*\Delta c)^T(F+J*\Delta c)$$

The Conjugate Gradient method for minimization only requires evaluating J*v not solving J*v=b, and evaluating J*v does not require storing any matrices (you can compute the non-zero elements of J, use them, and delete them). However, it is an iterative method, so it does not solve this quadratic minimization problem exactly in one step like Newton-Raphson would (so at each iteration in solving the PDE problem we are going to do several sub-iterations to solve $J*\Delta c=-F$). Also, if many sub-iterations are required numerical noise can creep in causing problems, so people currently use a fancier version called bicgstab instead of the simple Conjugate Gradient algorithm. The performance of Conjugate Gradient is dramatically improved if the matrix J is well-conditioned (i.e. cond(J) ~ 1), so people often use "preconditioners" A and B to modify the matrix:

$$(A*J*B)*v = -(A*F) \qquad B*v = \Delta c$$

There are many different ways to come up with preconditioners A and B that work well for a particular J; for some introduction see discussion in Beers' textbook.

**Finding a good initial guess at c can also be a big problem.** Sometimes the PDE system is for a steady-state solution, and the corresponding time-dependent PDE is also known. If by the physics any initial guess will eventually lead to the steady-state of interest, one can start from a poor initial guess and time-march toward the solution for a while (using the method of lines discussed below) to refine the initial guess. When the guess gets good enough that || F(c_guess) || < tol, then one switches from time-marching to just solving the system of non-linear equations.

***Method of Lines***

For parabolic PDEs, where the special direction is time, one popular approach is to discretize the space dimensions (e.g. replace all the spatial derivatives with finite-difference approximations). For example, the PDE equation

$\partial c/\partial t = D\, \partial^2 c/\partial x^2 - v\, \partial c/\partial x + r(c)$

might be replaced by the finite difference equation

$dc(m)/dt = D\, \{c(m-1) - 2\, c(m) + c(m+1)\}\, /(\Delta x)^2 - v\, \{ c(m-1)-c(m) \}\, /\Delta x + r(c(m))$

There will be similar equations for each mesh point xm. This can be compactly written

$dc/dt = F(c)$

where c is the vector [c(1); c(2);…; c(m); c(m+1);…c(M)]. Note every element of this long vector (except maybe some on the boundaries) are time dependent.

It one knows all the boundary conditions at $t_0$, this is an ODE-IVP, and can be solved that way. This is called "The Method of Lines". This is very practical for systems with one spatial dimension, where the number of spatial mesh points M might be as small as 100. It can even be done for 3-d problems, where the number of mesh points $M>10^6$, if an explicit ODE-IVP method can be used, in fact this is how the best existing Navier-Stokes reacting-flow solvers work (see papers by Jacqueline H. Chen).

If one is using Method-of-Lines to time-march to an initial guess, there is no reason that one needs to be "time-accurate": all you want to do is to get to the long-time solution (which will be close to the steady-state solution and so a good initial guess) as quickly as possible. One way to accelerate the time-marching is to use Implicit Euler with relatively large time steps:

$C_{new} = C_{old} + \Delta t * F(C_{new})$

However, note that this is still a (huge) system of nonlinear equations, and so it may need to be solved iteratively using Conjugate Gradient methods rather than Newton-Raphson with Gaussian elimination. And solving this systems of equations also needs an initial guess… which perhaps we could supply using an explicit ODE-IVP solver.

MIT OpenCourseWare
https://ocw.mit.edu

# 10.34 Numerical Methods Applied to Chemical Engineering

Fall 2015

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms.