CS 450: Numerical Anlaysis[1]

Partial Differential Equations

University of Illinois at Urbana-Champaign

[1] *These slides have been drafted by Edgar Solomonik as lecture templates and supplementary material for the book "Scientific Computing: An Introductory Survey" by Michael T. Heath (slides).*

Partial Differential Equations

- *Partial differential equations (PDEs)* describe physical laws and other continuous phenomena:

  fluid dynamics
  quantum physics
  astrophysics

- The *advection PDE* describes basic phenomena in fluid flow,

  first order

$$u_t = -a(t, x)u_x$$

first order
problem-defined

where $u_t = \partial u / \partial t$ and $u_x = \partial u / \partial x$.

$$\frac{\partial u}{\partial t}(t,x) = -a(t,x)\frac{\partial u}{\partial x}(t,x)$$

multiple partial devs

second order PDES

e.g. u_{xy}
u_{tt}
u_{xx}

$$u_t = -c u_x$$

Types of PDEs

- Some of the most important PDEs are *second order*:

heat equation $u_t = u_{xx}$

wave equation $u_{tt} = u_{xx}$ → time-dependent

Laplace equation $u_{xx} + u_{yy} = 0$ ← static

- The *discriminant* determines the canonical form of second-order PDEs:

classify general second-order PDEs

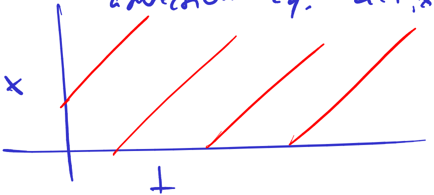
$$a u_{xx} + b u_{xy} + c u_{yy} + d u_x + e u_y + f = 0$$

discriminant $r = b^2 - 4ac \Rightarrow \begin{cases} r = 0 & : \text{heat equation} \\ r > 0 & : \text{wave equation} \\ r < 0 & : \text{Laplace} \end{cases}$

Characteristic Curves

- A *characteristic* of a PDE is a level curve in the solution:

advection eq. $a(t,x)=c$



$u(t, x(t)) = \text{const.}$

generally $x(t)$ gives curve along which PDE reduces to an ODE

- More generally, characteristic curves describe curves in the solution field $u(t,x)$ that correspond to solutions of ODEs, e.g. for $u_t = -a(t,x)u_x$ with $u(0,x) = u_0(x)$

advection

$$\frac{d\hat{x}(t)}{dt} = -a(t,\hat{x}(t)), \quad \hat{x}(0) = x_0 \qquad u(t,x) = u_0(\hat{x}(t))$$

$$u_t = \frac{\partial u_0}{\partial t}(\hat{x}(t)) = \underbrace{\frac{d\hat{x}}{dt}(t)}_{-a(t,\hat{x}(t))}\underbrace{u_0'(\hat{x}(t))}_{u_x} = -a(t,x)u_x$$

Method of Lines

- *Semidiscrete methods* obtain an approximation to the PDE by solving a system of ODEs, e.g. consider the heat equation,

$$u_t = cu_{xx} \text{ on } 0 \leq x \leq 1, \quad u(0, x) = f(x), u(t, 0) = u(t, 1) = 0.$$

- This *method of lines* often yields a stiff ODE:

Semidiscrete Collocation

- Instead of finite-differences, we can express $u(t, x)$ in a spatial basis:

- For the heat equation $u_t = cu_{xx}$, we obtain a linear constant-coefficient vector ODE:

Fully Discrete Methods

- Generally, both time and space dimensions are discretized, for example using finite differences:

Implicit Fully Discrete Methods

- Using Euler's method for the heat equation, stability requirement is

$$\Delta t = O\Big((\Delta x)^2\Big)$$

Convergence and Stability

- *Lax Equivalence Theorem*: consistency + stability = convergence

- Stability can be ascertained by spectral or Fourier analysis:

CFL Condition

- The domain of dependence of a PDE for a given point (t, x) is the portion of the problem domain influencing this point through the PDE:

- The Courant, Friedrichs, and Lewy (CFL) condition states that a *necessary* condition for an explicit finite-differencing scheme to be stable for a hyperbolic PDE is that the domain of the dependence of the PDE be contained in the domain of dependence of the scheme:

Time-Independent PDEs

- We now turn our focus to time-independent PDEs as exemplified by the *Helmholtz equation*:

$$u_{xx} + u_{yy} + \lambda u = f(x, y)$$

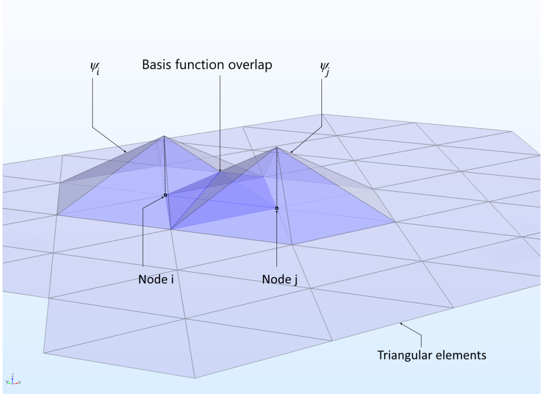
- We discretize as before, but no longer perform time stepping:

Finite-Differencing for Poisson

- Consider the Poisson equation with equispaced mesh-points on $[0, 1]$:

Multidimensional Finite Elements

- There are many ways to define localized basis functions, for example in the 2D FEM method[2]:



Sparse Linear Systems

- Finite-difference and finite-element methods for time-independent PDEs give rise to sparse linear systems:

- *Direct methods* apply LU or other factorization to $\boldsymbol{A}$, while *iterative methods* refine $\boldsymbol{x}$ by minimizing $\boldsymbol{r} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$, e.g. via Krylov subspace methods.

Direct Methods for Sparse Linear Systems

- It helps to think of $\boldsymbol{A}$ as the adjacency matrix of graph $G = (V, E)$ where $V = \{1, \dots n\}$ and $a_{ij} \neq 0$ if and only if $(i, j) \in E$:

- Factorizing the lth row/column in Gaussian elimination corresponds to removing node i, with nonzeros (new edges) introduces for each k, l such that (i, k) and (i, l) are in the graph.

Vertex Orderings for Sparse Direct Methods

- Select the node of minimum degree at each step of factorization:

- Graph partitioning also serves to bound fill, remove vertex separator $S \subset V$ so that $V \setminus S = V_1 \cup \cdots \cup V_k$ become disconnected, then order $V_1, \ldots, V_k, S$:

- *Nested dissection* ordering partitions graph into halves recursively, ordering each separator last.

Sparse Iterative Methods

- Direct sparse factorization is ineffective in memory usage and/or cost for many typical sparsity matrices, motivating iterative methods:

Sparse Iterative Methods

- The *Jacobi method* is the simplest iterative solver:

- The Jacobi method converges if A is strictly row-diagonally-dominant:

Gauss-Seidel Method

- The Jacobi method takes weighted sums of $\boldsymbol{x}^{(k)}$ to produce each entry of $\boldsymbol{x}^{(k+1)}$, while Gauss-Seidel uses the latest available values, i.e. to compute $x_i^{(k+1)}$ it uses a weighted sum of

$$x_1^{(k+1)}, \ldots x_{i-1}^{(k+1)}, x_i^{(k)}, \ldots, x_n^{(k)}.$$

- Gauss-Seidel provides somewhat better convergence than Jacobi:

Successive Over-Relaxation

- The *successive over-relaxation* (SOR) method seeks to improve the spectral radius achieved by Gauss-Seidel, by choosing

$$\boldsymbol{M} = \frac{1}{\omega}\boldsymbol{D} + \boldsymbol{L}, \quad \boldsymbol{N} = \left(\frac{1}{\omega} - 1\right)\boldsymbol{D} - \boldsymbol{U}$$

- The parameter ω in SOR controls the 'step-size' of the iterative method:

Conjugate Gradient

- The solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is a minima of the quadratic optimization problem,

$$\min_{\boldsymbol{x}} ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$$

- Conjugate gradient works by picking $\boldsymbol{A}$-orthogonal descent directions

- The convergence rate of CG is linear with coefficient $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ where $\kappa = \text{cond}(\boldsymbol{A})$:

Preconditioning

- Preconditioning techniques choose matrix $\boldsymbol{M} \approx \boldsymbol{A}$ and solve the linear system
$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}$$

- $\boldsymbol{M}$ is a usually chosen to be an effective approximation to $\boldsymbol{A}$ with a simple structure: