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) are equations describe physical laws and other continuous phenomena:
  - *They contain partial derivatives in multiple variables.*
  - *Examples include: electromagnetism, fluid flow, quantum mechanics, and general relativity.*
- A simple PDE is the advection equation, which describes basic phenomena in fluid flow:

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

  where $u_t = \partial u/\partial t$ *and* $u_x = \partial u/\partial x$*. Generally, we impose an initial condition with respect to* t*, i.e.,* $u(0, x) = u_0(x)$*. When* $a(t, x) = c$ *this is the Cauchy problem with solution*

  $$u(t, x) = u_0(x - ct).$$

Properties of PDEs

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

  For the Cauchy form of the advection equation, a characteristic $x(t)$ satisfies $u(t, x(t)) =$ const. One of their uses is identifying where boundary conditions must be satisfied, e.g. for the Cauchy equation, it tells us whether boundary conditions are needed on the left or the right (in terms of x) of the domain. More generally, characteristic curves describe curves in the solution field $u(t, x)$ that correspond to ODEs e.g. for the advection equation,

  $$\frac{\partial x(t)}{\partial t} = a(t, x(t)) \text{ with initial condition } x(0) = x_0.$$

- The order of a PDE is the highest-order of any partial derivative appearing in the PDE:

  The advection equation is a first order ODE. ODEs containing e.g. u_{xy} or u_{tt} are second order.

Types of PDEs

- Some of the most important PDEs are second order:
  - *Heat equation (diffusion)*, $u_t = u_{xx}$
  - *Wave equation (oscillation)*, $u_{tt} = u_{xx}$
  - *Laplace equation (steady-state)*, $u_{xx} + u_{yy} = 0$

  Any PDE of the form

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

  behaves (has the character) of one of the above equations.
- The *discriminant* determines the canonical form of second-order PDEs:

  The discriminant is $r = b^2 - 4ac$ *and linear PDEs are classified as*

$$\begin{cases} r > 0 & : \textit{hyperbolic, wave-equation-like} \\ r = 0 & : \textit{parabolic, heat-equation-like} \\ r < 0 & : \textit{elliptic, Laplace-equation-like} \end{cases}$$

  When coefficients are varying, a PDE may exhibit different behavior in different parts of the domain.

Method of Lines

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

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

We discretize over x and use finite differences to approximate

$$u_{xx} \approx \frac{u(t,x_{i+1}) - 2u(t,x_i) + u(t,x_{i-1})}{(\Delta x)^2}.$$

This approximation yields a system of ODEs

$$y_i'(t) = \frac{c}{(\Delta x)^2}\Big(y_{i+1}(t) - 2y_i(t) + y_{i-1}(t)\Big)$$

which we can write as $\boldsymbol{y}'(t) = \boldsymbol{A}\boldsymbol{y}(t)$, where $\boldsymbol{A}$ is tridiagonal.

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

  The eigenvalues of $\boldsymbol{A}$ range from near 0 to $-4c/(\Delta x)^2$, which means the ODE is stiff for small Δx. Intuitively, strong concentrations of heat dissipate very rapidly, while overall heat dissipates slowly.

Semidiscrete Collocation

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

$$u(t, x) \approx v(t, x, \boldsymbol{\alpha}(t)) = \sum_{j=1}^{n} \alpha_j \phi_j(x)$$

Collocation methods then ensure the approximation is exact on $x_1, \dots, x_n$, *yielding matrix equations.*

- For the heat equation $u_t = c u_{xx}$, we obtain an ODE:

$$\sum_{j=1}^{n} \frac{\partial \alpha_j}{\partial t}(t) \underbrace{\phi_j(x_i)}_{m_{ij}} = c \sum_{j=1}^{n} \alpha_j(t) \underbrace{\frac{\partial^2 \phi_j}{\partial x^2}(x_i)}_{n_{ij}}$$

written in matrix form,

$$\boldsymbol{\alpha}'(t) = c\boldsymbol{M}^{-1}\boldsymbol{N}\boldsymbol{\alpha}(t)$$

Fully Discrete Methods

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

  Lets again consider the heat equation $u_t = cu_{xx}$ *and discretize so that* $u_i^{(k)} \approx u(t_k, x_i)$*,*

  $$\frac{u_i^{(k+1)} - u_i^{(k)}}{\Delta t} = c\frac{u_{i+1}^{(k)} - 2u_i^{(k)} + u_{i-1}^{(k)}}{(\Delta x)^2}$$

  which yields an iterative scheme defined by 3-pt stencil

  $$u_i^{(k+1)} = u_i^{(k)} + c\Delta t\frac{u_{i+1}^{(k)} - 2u_i^{(k)} + u_{i-1}^{(k)}}{(\Delta x)^2}.$$

  The same scheme can be derived by applying Euler's method to the ODE given by the method of lines.

Implicit Fully Discrete Methods

- When using Euler's method for the heat equation, to stay in stability region, require

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

Step-size restriction on stability can be circumvented by use of implicit time-stepper, such as backward Euler,

$$u_i^{(k+1)} = u_i^{(k)} + c\Delta t \frac{u_{i+1}^{(k+1)} - 2u_i^{(k+1)} + u_{i-1}^{(k+1)}}{(\Delta x)^2}.$$

This scheme requires for a tridiagonal matrix system to be solved at each time-step, but obtains unconditional stability, albeit only first-order accuracy. Using the trapezoid method to solve the ODE we obtain the second-order Crank-Nicolson method,

$$u_i^{(k+1)} = u_i^{(k)} + c\Delta t \frac{u_{i+1}^{(k+1)} - 2u_i^{(k+1)} + u_{i-1}^{(k+1)} + u_{i+1}^{(k)} - 2u_i^{(k)} + u_{i-1}^{(k)}}{2(\Delta x)^2}.$$

Convergence and Stability

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

  Consistency means that the local truncation error goes to zero, and is easy to verify by Taylor expansions. Stability implies that the approximate solution at any time t must remain bounded. Together these conditions are necessary and sufficient for convergence.

- Stability can be ascertained by spectral or Fourier analysis:

  In the method of lines, we saw that the eigenvalues of the resulting ODE define the stability region. Fourier analysis decomposes the solution into a sum of harmonic functions and considers the behavior of their amplitude.

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:

  Generally determined by characteristics of PDE. For a stencil method, depends on the set of mesh-points influencing the mesh point at (t, x).

- 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:

  Intuitively, we can then achieve stability by choosing a sufficiently large grid spacing h, or including more mesh points in our stencil.

Time-Independent PDEs

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

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

  - $\lambda = 0$ *yields the Poisson equation*
  - $\lambda = 0$ *and* $f = 0$ *yields the Laplace equation*
  - *Boundary conditions (e.g. Dirichlet or Neumann or mixed) on surface of domain*

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

  For example given a domain $[0, 1]^2$, *we can tile it using* $n \times n$ *mesh points, and setup finite-difference equations or boundary-condition equations at each point.*

Finite-Differencing for Poisson

- Consider the Poisson equation with equispaced mesh-points on $[0, 1]$:
  If $\boldsymbol{u}$ is a vector containing the mesh-points, we have that

  $$\mathcal{D}_x \boldsymbol{u} + \mathcal{D}_y \boldsymbol{u} = \boldsymbol{b} \text{ where } b_i = f(u_i)$$

  where $\mathcal{D}_x$ and $\mathcal{D}_y$ are finite-difference operators along x and y dimensions, respectively. Given a differencing matrix $\boldsymbol{D}$ (e.g. tridiagonal with $1, -2, 1$), we obtain the matrix equation,

  $$(\boldsymbol{I} \otimes \boldsymbol{D} + \boldsymbol{D} \otimes \boldsymbol{I})\boldsymbol{u} = \boldsymbol{b}$$

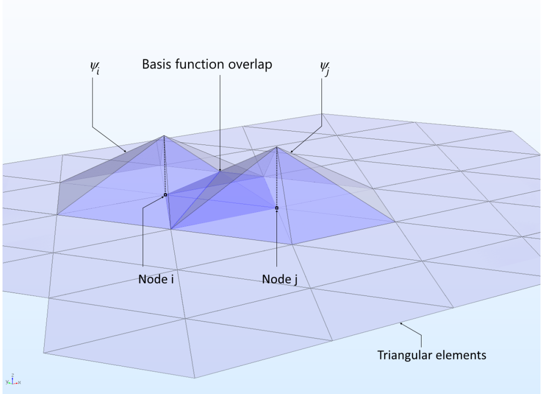
  where the Kronecker product is defined as

  $$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \cdots \\ a_{21}\boldsymbol{B} & \ddots & \\ \vdots & & \end{bmatrix},$$

  and the elements of $\boldsymbol{b}$ contain the mesh elements in column-major (or row-major in this example) order.

Multidimensional Finite Elements

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



We partition the domain into triangles (elements) and define linear basis functions that are 1 at the intersection of three or more elements (nodes).

[2] Source: Comsol Multiphysics Cyclopedia `https://www.comsol.com/multiphysics/finite-element-method`

Sparse Linear Systems

- Finite-difference and finite-element methods for time-independent PDEs give rise to sparse linear systems:
  - *typified by the 2D Laplace equation, where for both finite differences and FEM,*

$$\underbrace{(\boldsymbol{I} \otimes \boldsymbol{T} + \boldsymbol{T} \otimes \boldsymbol{I})}_{\boldsymbol{A}} \boldsymbol{x} = \boldsymbol{b} \quad \textit{where} \quad \boldsymbol{T} = \frac{1}{h^2} \begin{bmatrix} -2 & 1 & \\ 1 & \ddots & \ddots \\ & \ddots & \end{bmatrix}$$

  - *often have $O(1)$ nonzeros per row/column of the matrix,*
  - *for simple/regular problems matrices are near-Toeplitz (same entry along each (sub/super)diagonal), permitting fast solvers via e.g. FFT.*
- *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 provide a high-accuracy solution, but may not be effective at leveraging sparsity to reduce cost.*
  - *Iterative methods effectively leverage sparsity by computing matrix-vector products with $\boldsymbol{A}$, but may require many iterations to achieve high-accuracy.*

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 $(i, j) \in E$:

  The graph is invariant under permutation of vertices, i.e. for any permutation matrix $\boldsymbol{P}$, reordering V accordingly transeforms the adjacency matrix of the graph into $\boldsymbol{P}^T\boldsymbol{AP}$. Note that such reorderings of variables essentially do not change the linear system,

$$\boldsymbol{P}^T\boldsymbol{AP}\underbrace{\boldsymbol{P}^T\boldsymbol{x}}_{\hat{\boldsymbol{x}}} = \boldsymbol{b}$$

- 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.
  - *creates clique (fully connected subgraph) among neighbors of vertex i,*
  - *different orderings of vertices can result in radically different amounts of fill,*
  - *finding optimal ordering to reduce fill is NP complete.*

Vertex Orderings for Sparse Direct Methods

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

  Each step minimizes work, but not necessarily amount of fill at that step (depends on whether neighbors are already connected).

- 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 $V_1, \ldots, V_k, S$:

  Matrix takes on the form

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & & & \boldsymbol{A}_{1S} \\ & \ddots & & \vdots \\ & & \boldsymbol{A}_{kk} & \boldsymbol{A}_{kS} \\ \boldsymbol{A}_{S1} & \cdots & \boldsymbol{A}_{Sk} & \boldsymbol{A}_{SS} \end{bmatrix}$$

  where each $\boldsymbol{A}_{ii}$ for $i \in \{1, \ldots, k\}$ can be factored independently.

- *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:

  Many such basic iterative methods corresponds to linear fixed point iterations defined by linear system:

  $$\boldsymbol{M}\boldsymbol{x}_{k+1} = \boldsymbol{N}\boldsymbol{x}_k + \boldsymbol{b}$$

  which can be written in the form $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{M}^{-1}\boldsymbol{N}\boldsymbol{x} + \boldsymbol{M}^{-1}\boldsymbol{b}$*. We desire to have a fixed point whenever* $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$*, which implies*

  $$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{N}\boldsymbol{x} + \boldsymbol{A}\boldsymbol{x},$$

  so generally $\boldsymbol{M}$ *and* $\boldsymbol{N}$ *are chosen so that* $\boldsymbol{M} - \boldsymbol{N} = \boldsymbol{A}$*. Moreover, to achieve convergence we need* $\rho(\boldsymbol{g}) = \rho(\boldsymbol{M}^{-1}\boldsymbol{N}) < 1$*.*

Sparse Iterative Methods

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

  We split up $\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U}$ where $\boldsymbol{D}$ is diagonal while $\boldsymbol{L}$ and $\boldsymbol{U}^T$ are strictly lower triangular. Jacobi iteration uses a fixed point scheme with $\boldsymbol{M} = \boldsymbol{D}$ and $\boldsymbol{N} = -(\boldsymbol{L} + \boldsymbol{U})$, yielding a diagonal system of equations,

  $$\boldsymbol{D}\boldsymbol{x}^{(k+1)} = (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(k)} + \boldsymbol{b}.$$

  The cost of each iteration of Jacobi is proportional to SpMV with $\boldsymbol{A}$.

- The Jacobi method converges if $\boldsymbol{A}$ is strictly row-diagonally-dominant:

  A strictly row-diagonally-dominant $\boldsymbol{A}$ satisfies $|a_{ii}| < \sum_{j \neq i} |a_{ij}|$, which implies that for $\boldsymbol{B} = \boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U})$,

  $$\left|\sum_j b_{ij}\right| = \left|\sum_{j \neq i} a_{ij}/a_{ii}\right| < 1,$$

  so that $\rho(\boldsymbol{M}^{-1}\boldsymbol{N}) = \rho(\boldsymbol{B}) < 1$. For the Laplace problem, this condition holds, and the method corresponds to averaging neighbors. However, the coefficient in linear convergence is $\cos(\pi h)$ (goes to 1 with decreasing h).

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

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

  We can define the method by the splitting $\boldsymbol{M} = \boldsymbol{D} + \boldsymbol{L}$ *so that we have*

$$(\boldsymbol{D} + \boldsymbol{L})\boldsymbol{x}^{(k+1)} = \boldsymbol{U}\boldsymbol{x}^{(k)}.$$

  The Gauss-Seidel method performs an in-order traversal of the directed acyclic adjacency graph induced by the vertex ordering, and updates each vertex by taking newly computed values from incoming edges and values from the previous iteration from outgoing edges.

- Gauss-Seidel provides somewhat better convergence than Jacobi:
  Convergence and efficiency depend on vertex ordering and connectivity:
  - *for 2-D Poisson, spectral radius is* $\cos^2(\pi h)$*,*
  - *computational cost is same as Jacobi, but less parallelism available.*

Successive Over-Relaxation

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

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

  In the resulting iterative scheme, we have

  $$\Big(\frac{1}{\omega}\boldsymbol{D} + \boldsymbol{L}\Big)\boldsymbol{x}^{(k+1)} = ((1-\omega)\boldsymbol{D} + \boldsymbol{U})\boldsymbol{x}^{(k)}$$

  so that if $\boldsymbol{x}_{GS}^{(k+1)}$ *is the iterate produced by Gauss-Seidel, SOR instead produces*

  $$\boldsymbol{x}^{(k+1)} = (1-\omega)\boldsymbol{x}^{(k)} + \omega\boldsymbol{x}_{GS}^{(k+1)}.$$

- The parameter ω in SOR controls the 'step-size' of the iterative method:
  - *over-relaxation corresponds to* $\omega > 1$,
  - *under-relaxation corresponds to* $\omega < 1$,
  - *generally best choice of* $\omega \in (0, 2)$ *is hard to determine.*

Conjugate Gradient

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

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

  We can leverage this and employ optimization methods such as conjugate gradient (CG) in the case when $\boldsymbol{A}$ is SPD.

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

  Ensures search directions make progress and converge in at most n iterations.

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

  This convergence rate motivates techniques to improve the conditioning of $\boldsymbol{A}$ to accelerate convergence.

Preconditioning

- Preconditioning techniques choose matrix $\boldsymbol{M} \approx \boldsymbol{A}$ and solve the linear system
$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{M}^{-1}\boldsymbol{b}$$
For example at each step of CG, we replace multiplication with $\boldsymbol{A}$ by multiplication by $\boldsymbol{M}^{-1}\boldsymbol{A}$:
  - *cost of iteration depends on difficulty of applying $\boldsymbol{M}^{-1}$,*
  - *convergence rate depends on how close $\boldsymbol{M}$ is to $\boldsymbol{A}$.*
- $\boldsymbol{M}$ is a usually chosen to be an effective approximation to $\boldsymbol{A}$ with a simple structure:
  - *Jacobi preconditioning takes $\boldsymbol{M} = \boldsymbol{D}$ where $\boldsymbol{D}$ is the diagonal of $\boldsymbol{A}$,*
  - *incomplete factorization (ILU) uses $\boldsymbol{A} \approx \boldsymbol{L}\boldsymbol{U}$ where the sparsity pattern of $\boldsymbol{L}$ and $\boldsymbol{U}$ is restricted to that of $\boldsymbol{A}$ (factorization is then generally inexact), and employs $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{U}$.*