Lecture Notes to Accompany

Scientific Computing

An Introductory Survey

Second Edition

by Michael T. Heath

Chapter 11

Partial Differential Equations

Partial Differential Equations

Partial differential equations (PDEs) involve partial derivatives with respect to more than one independent variable

Independent variables typically include one or more space dimensions and possibly time dimension as well

More dimensions complicate problem formulation: can have pure initial value problem, pure boundary value problem, or mixture

Equation and boundary data may be defined over irregular domain in space

Partial Differential Equations, cont.

For simplicity, we will deal only with single PDEs (as opposed to systems of several PDEs) with only two independent variables, either

- two space variables, denoted by x and y, or
- one space variable and one time variable, denoted by x and t, respectively

Partial derivatives with respect to independent variables denoted by subscripts:

- $u_t = \partial u/\partial t$
- $u_{xy} = \partial^2 u/\partial x \partial y$, etc.

Example: Advection Equation

Advection equation:

$$u_t = -c\,u_x,$$

where c is nonzero constant

Unique solution determined by initial condition

$$u(0, x) = u_0(x), \qquad -\infty < x < \infty,$$

where u_0 is given function defined on $\mathbb{R}$

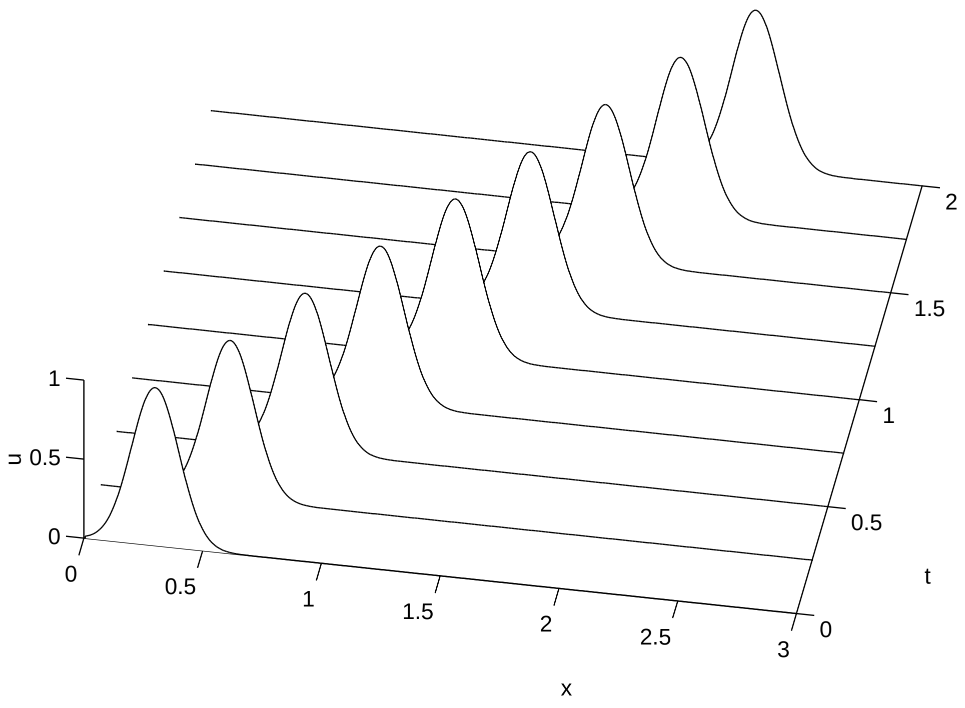
We seek solution $u(t, x)$ for $t \geq 0$ and all $x \in \mathbb{R}$

From chain rule, solution given by

$$u(t, x) = u_0(x - c\,t),$$

i.e., solution is initial function u_0 shifted by $c\,t$ to right if $c > 0$, or to left if $c < 0$

Example Continued



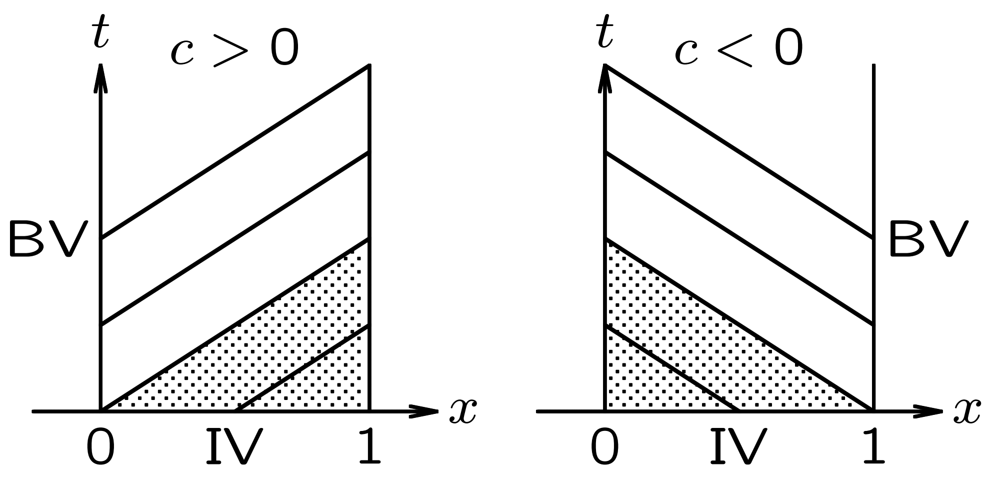
Typical solution of advection equation

Characteristics

Level curves of solution to PDE are called *characteristics*

Characteristics for advection equation, for example, are straight lines of slope c

Characteristics determine where boundary conditions can or must be imposed for problem to be well-posed



Classification of PDEs

Order of PDE is order of highest-order partial derivative appearing in equation

Advection equation is first order, for example

Important second-order PDEs include

- *Heat* equation: $u_t = u_{xx}$
- *Wave* equation: $u_{tt} = u_{xx}$
- *Laplace* equation: $u_{xx} + u_{yy} = 0$

Classification of PDEs, cont.

Second-order linear PDEs of form

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0$$

are classified by value of *discriminant*, $b^2 - 4ac$,

$b^2 - 4ac > 0$: *hyperbolic* (e.g., wave eqn)

$b^2 - 4ac = 0$: *parabolic* (e.g., heat eqn)

$b^2 - 4ac < 0$: *elliptic* (e.g., Laplace eqn)

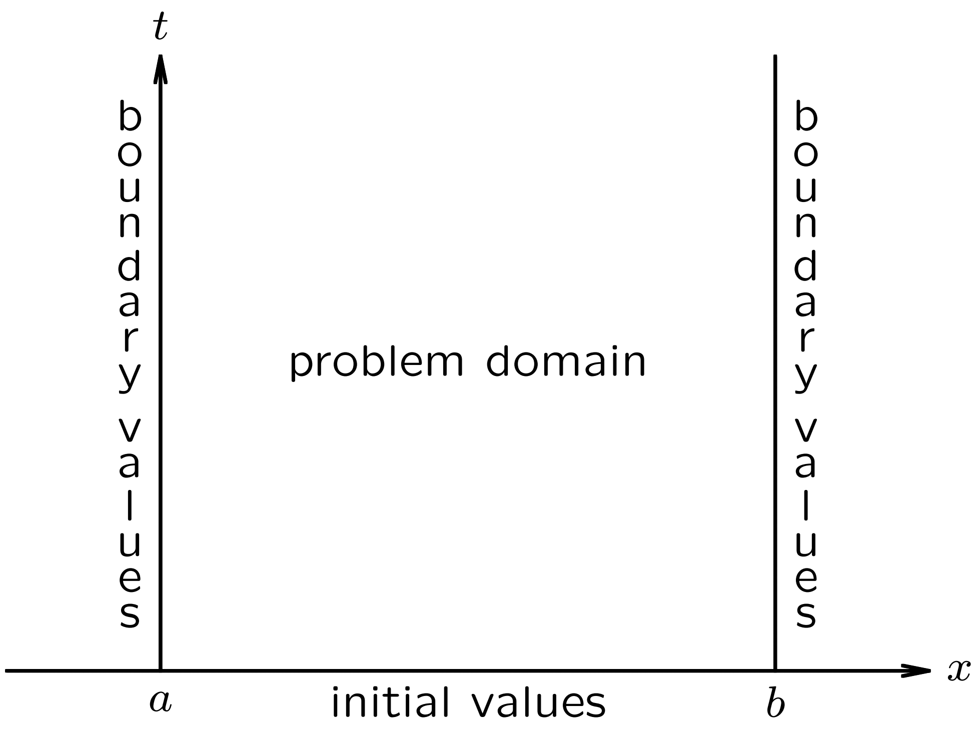
Classification of PDEs, cont.

Classification of more general PDEs not so clean and simple, but roughly speaking:

- *Hyperbolic* PDEs describe time-dependent, conservative physical processes, such as convection, that *are not* evolving toward steady state
- *Parabolic* PDEs describe time-dependent, dissipative physical processes, such as diffusion, that *are* evolving toward steady state
- *Elliptic* PDEs describe processes that have already reached steady state, and hence are time-independent

Time-Dependent Problems

Time-dependent PDEs usually involve both initial values and boundary values



Semidiscrete Methods

One way to solve time-dependent PDE numerically is to discretize in space, but leave time variable continuous

Result is system of ODEs that can then be solved by methods previously discussed

For example, consider heat equation

$$u_t = c\,u_{xx}, \qquad 0 \le x \le 1, \qquad t \ge 0,$$

with initial condition

$$u(0,x) = f(x), \qquad 0 \le x \le 1,$$

and boundary conditions

$$u(t,0) = 0, \qquad u(t,1) = 0, \qquad t \ge 0$$

Semidiscrete Finite Difference Method

If we introduce spatial mesh points $x_i = i\Delta x$, $i = 0, \ldots, n+1$, where $\Delta x = 1/(n+1)$ and replace derivative u_{xx} by finite difference approximation

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

then we get system of ODEs

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

$i = 1, \ldots, n$, where $y_i(t) \approx u(t, x_i)$

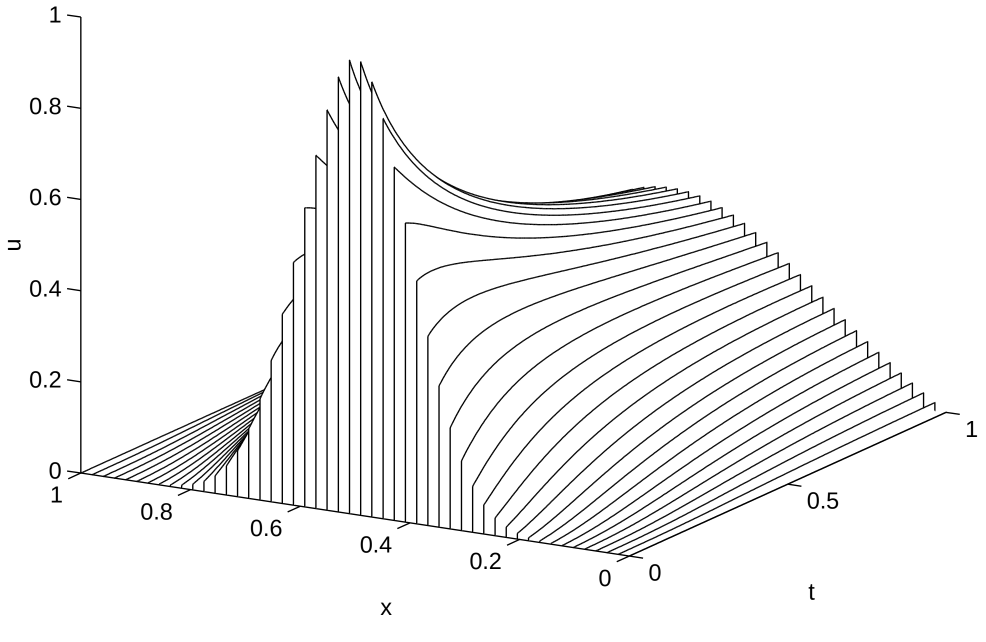
From boundary conditions, $y_0(t)$ and $y_{n+1}(t)$ are identically zero, and from initial conditions, $y_i(0) = f(x_i)$, $i = 1, \ldots, n$

Can therefore use ODE method to solve initial value problem for this system

Method of Lines

Approach just described is called *method of lines*

MOL computes cross-sections of solution surface over space-time plane along series of lines, each parallel to time axis and corresponding to one of discrete spatial mesh points



Stiffness

Semidiscrete system of ODEs just derived can be written in matrix form

$$\boldsymbol{y}' = \frac{c}{(\Delta x)^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \boldsymbol{y} = \boldsymbol{A}\boldsymbol{y}$$

Jacobian matrix $\boldsymbol{A}$ of this system has eigenvalues between $-4c/(\Delta x)^2$ and 0, which makes ODE very stiff as spatial mesh size Δx becomes small

This stiffness, which is typical of ODEs derived from PDEs in this manner, must be taken into account in choosing ODE method for solving semidiscrete system

Semidiscrete Collocation Method

Spatial discretization to convert PDE into system of ODEs can also be done by spectral or finite element approach

Approximate solution is linear combination of basis functions, but now coefficients are time dependent

Thus, we seek solution of form

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

where $\phi_j(x)$ are suitably chosen basis functions

If we use collocation, then we substitute this approximation into PDE and require that equation be satisfied exactly at discrete set of points x_i

Semidiscrete Collocation, continued

For heat equation, this yields system of ODEs

$$\sum_{j=1}^{n} \alpha'_j(t)\phi_j(x_i) = c \sum_{j=1}^{n} \alpha_j(t)\phi''_j(x_i),$$

whose solution is set of coefficient functions $\alpha_i(t)$ that determine approximate solution to PDE

Implicit form of this system is not explicit form required by standard ODE methods, so we define $n \times n$ matrices $\boldsymbol{M}$ and $\boldsymbol{N}$ by

$$m_{ij} = \phi_j(x_i), \qquad n_{ij} = \phi''_j(x_i)$$

Semidiscrete Collocation, continued

Assuming $\boldsymbol{M}$ is nonsingular, we then obtain system of ODEs

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

which is in form suitable for solution with standard ODE software (as usual, $\boldsymbol{M}$ need not be inverted explicitly, but merely used to solve linear systems)

Initial condition for ODE can be obtained by requiring that solution satisfy given initial condition for PDE at points x_i

Matrices involved in this method will be sparse if basis functions are "local," such as B-splines

Semidiscrete Collocation, continued

Unlike finite difference method, spectral or finite element method does not produce approximate values of solution u directly, but rather it generates representation of approximate solution as linear combination of basis functions

Basis functions depend only on spatial variable, but coefficients of linear combination (given by solution to system of ODEs) are time dependent

Thus, for any given time t, corresponding linear combination of basis functions generates cross section of solution surface parallel to spatial axis

As with finite difference methods, systems of ODEs arising from semidiscretization of PDE by spectral or finite element methods tend to be stiff

Fully Discrete Methods

Fully discrete methods for PDEs discretize in both time and space dimensions

In fully discrete finite difference method, we

- Replace continuous domain of equation by discrete mesh of points
- Replace derivatives in PDE by finite difference approximations
- Seek numerical solution that is table of approximate values at selected points in space and time

Fully Discrete Methods, continued

In two dimensions (one space and one time), resulting approximate solution values represent points on solution surface over problem domain in space-time plane

Accuracy of approximate solution depends on stepsizes in both space and time

Replacement of all partial derivatives by finite differences results in system of algebraic equations for unknown solution at discrete set of sample points

System may be linear or nonlinear, depending on underlying PDE

Fully Discrete Methods, continued

With initial-value problem, solution is obtained by starting with initial values along boundary of problem domain and marching forward in time step by step, generating successive rows in solution table

Time-stepping procedure may be explicit or implicit, depending on whether formula for solution values at next time step involves only past information

Example: Heat Equation

Consider heat equation

$$u_t = c\,u_{xx}, \qquad 0 \le x \le 1, \qquad t \ge 0,$$

with initial and boundary conditions

$$u(0,x) = f(x), \qquad u(t,0) = \alpha, \qquad u(t,1) = \beta$$

Define spatial mesh points $x_i = i\Delta x$, $i = 0, 1, \ldots, n+1$, where $\Delta x = 1/(n+1)$, and temporal mesh points $t_k = k\Delta t$, for suitably chosen Δt

Let u_i^k denote approximate solution at (t_k, x_i)

If we replace u_t by forward difference in time and u_{xx} by centered difference in space, we get

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

or

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

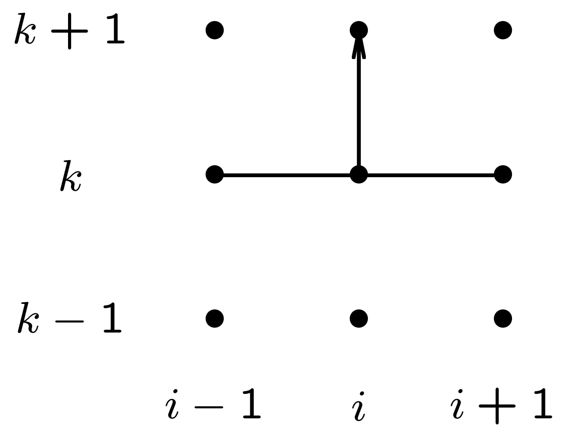
$i = 1, \ldots, n$

Heat Equation, continued

Boundary conditions give us $u_0^k = \alpha$ and $u_{n+1}^k = \beta$ for all k, and initial conditions provide starting values $u_i^0 = f(x_i)$, $i = 1, \ldots, n$

So we can march numerical solution forward in time using this *explicit* difference scheme

Pattern of mesh points, or *stencil*, involved at each level is shown below



Local truncation error is $\mathcal{O}(\Delta t) + \mathcal{O}((\Delta x)^2)$, so scheme is first-order accurate in time and second-order accurate in space

Example: Wave Equation

Consider wave equation

$$u_{tt} = c\, u_{xx}, \qquad 0 \le x \le 1, \qquad t \ge 0,$$

with initial and boundary conditions

$$u(0, x) = f(x), \qquad u_t(0, x) = g(x),$$

$$u(t, 0) = \alpha, \qquad u(t, 1) = \beta$$

With mesh points defined as before, using centered difference formulas for both u_{tt} and u_{xx} gives finite difference scheme

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

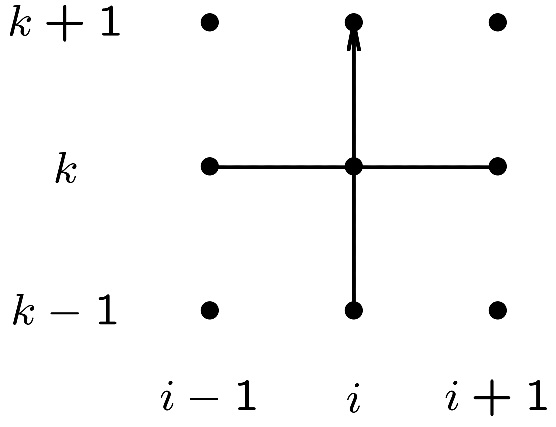
or

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

$i = 1, \ldots, n$

Wave Equation, continued

Stencil for this scheme is shown below



Using data at two levels in time requires additional storage

Also need u_i^0 and u_i^1 to get started, which can be obtained from initial conditions

$$u_i^0 = f(x_i), \qquad u_i^1 = f(x_i) + (\Delta t) g(x_i),$$

where latter uses forward difference approximation to initial condition $u_t(0,x) = g(x)$

Stability

Unlike Method of Lines, where time step is chosen automatically by ODE solver, user must choose time step Δt in fully discrete method, taking into account both accuracy and stability requirements

For example, fully discrete scheme for heat equation is simply Euler's method applied to semidiscrete system of ODEs for heat equation given previously

We saw that Jacobian matrix of semidiscrete system has eigenvalues between $-4c/(\Delta x)^2$ and 0, so stability region for Euler's method requires time step to satisfy

$$\Delta t \le \frac{(\Delta x)^2}{2\,c}$$

This severe restriction on time step makes this explicit method relatively inefficient compared to implicit methods we will see next

Implicit Finite Difference Methods

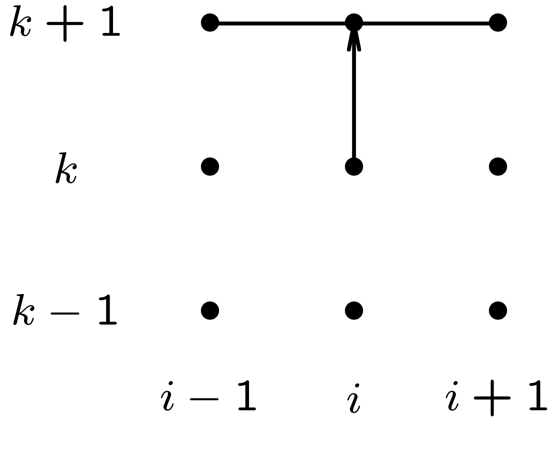
For ODEs we saw that implicit methods are stable for much greater range of stepsizes, and same is true of implicit methods for PDEs

Applying backward Euler method to semidiscrete system for heat equation gives *implicit* finite difference scheme

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

$i = 1, \ldots, n$

Stencil for this scheme is shown below



Implicit Finite Difference Methods, cont.

This scheme inherits unconditional stability of backward Euler method, which means there is no stability restriction on relative sizes of Δt and Δx

However, first-order accuracy in time still limits time step severely

Crank-Nicolson Method

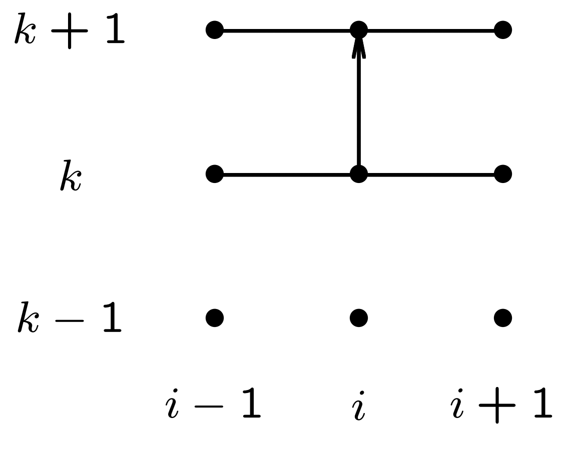
Applying trapezoid method to semidiscrete system of ODEs for heat equation yields implicit *Crank-Nicolson* method

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

$$+\, u_{i+1}^k - 2u_i^k + u_{i-1}^k\Big), \quad i = 1, \ldots, n,$$

which is unconditionally stable and second-order accurate in time

Stencil for this scheme is shown below



Implicit Finite Difference Methods, cont.

Much greater stability of implicit finite difference methods enables them to take much larger time steps than explicit methods, but they require more work per step, since system of equations must be solved at each step

For both backward Euler and Crank-Nicolson methods for heat equation in one space dimension, this linear system is tridiagonal, and thus both work and storage required are modest

In higher dimensions, matrix of linear system does not have such simple form, but it is still very sparse, with nonzeros in regular pattern

Convergence

In order for approximate solution to converge to true solution of PDE as stepsizes in time and space jointly go to zero, two conditions must hold:

- *Consistency*: local truncation error goes to zero
- *Stability*: approximate solution at any fixed time t remains bounded

Lax Equivalence Theorem says that for well-posed linear PDE, consistency and stability are together necessary and sufficient for convergence

Stability

Consistency is usually fairly easy verified using Taylor series expansion

Stability is more challenging, and several methods are available:

- *Matrix method*, based on location of eigenvalues of matrix representation of difference scheme, as we saw with Euler's method
- *Fourier method*, in which complex exponential representation of solution error is substituted into difference equation and analyzed for growth or decay
- *Domains of dependence*, in which domains of dependence of PDE and difference scheme are compared

CFL Condition

Domain of dependence of PDE is portion of problem domain that influences solution at given point, which depends on characteristics of PDE

Domain of dependence of difference scheme is set of all other mesh points that affect approximate solution at given mesh point

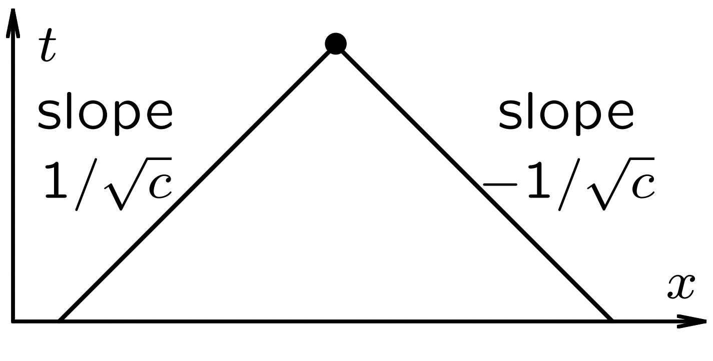
CFL Condition: necessary condition for explicit finite difference scheme for hyperbolic PDE to be stable is that for each mesh point domain of dependence of PDE must lie *within* domain of dependence of finite difference scheme

Example: Wave Equation

Consider explicit finite difference scheme for wave equation given previously

Characteristics of wave equation are straight lines in (t, x) plane along which either $x + \sqrt{c}\,t$ or $x - \sqrt{c}\,t$ is constant

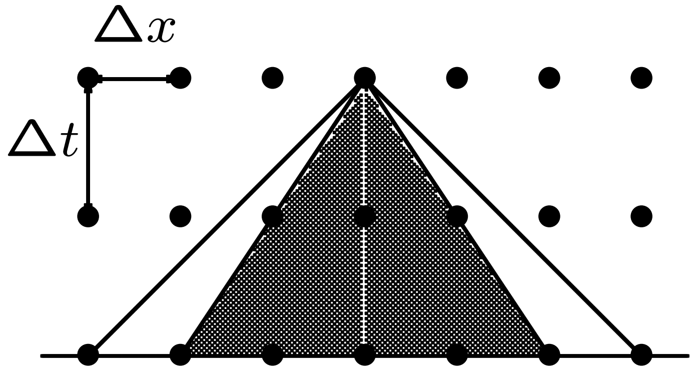
Domain of dependence for wave equation for given point is triangle with apex at given point and with sides of slope $1/\sqrt{c}$ and $-1/\sqrt{c}$



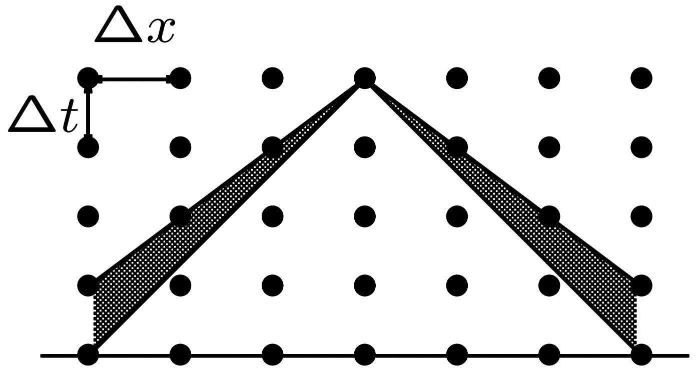
Example: Wave Equation

CFL condition implies step sizes must satisfy

$$\Delta t \leq \frac{\Delta x}{\sqrt{c}}$$



Unstable finite difference scheme



Stable finite difference scheme

Time-Independent Problems

We next consider time-independent, elliptic PDEs in two space dimensions, such as *Helmholtz* equation

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

Important special cases:

Poisson equation: $\lambda = 0$

Laplace equation: $\lambda = 0$ and $f = 0$

For simplicity, we will consider this equation on unit square

Numerous possibilities for boundary conditions specified along each side of square:

Dirichlet: u specified

Neumann: u_x or u_y specified

Mixed: combinations of these specified

Finite Difference Methods

Finite difference methods for such problems proceed as before:

- Define discrete mesh of points within domain of equation
- Replace derivatives in PDE by finite differences
- Seek numerical solution at mesh points

Unlike time-dependent problems, solution not produced by marching forward step by step in time

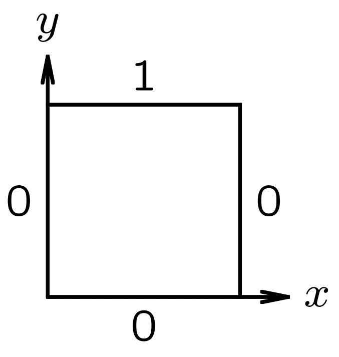
Approximate solution determined at all mesh points simultaneously by solving single system of algebraic equations

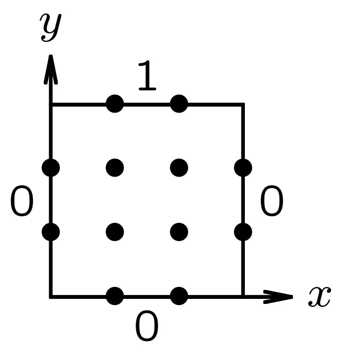
Example: Laplace Equation

Consider Laplace equation

$$u_{xx} + u_{yy} = 0$$

on unit square with boundary conditions shown on left below





Define discrete mesh in domain, including boundaries, as shown on right above

Interior grid points where we will compute approximate solution are given by

$$(x_i, y_j) = (ih, jh), \qquad i, j = 1, \ldots, n,$$

where in example $n = 2$ and $h = 1/(n+1) = 1/3$

Laplace Equation, continued

Next we replace derivatives by centered difference approximation at each interior mesh point to obtain finite difference equation

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} = 0,$$

where $u_{i,j}$ is approximation to true solution $u(x_i, y_j)$ for $i, j = 1, \ldots, n$, and represents one of given boundary values if i or j is 0 or $n+1$

Simplifying and writing out resulting four equations explicitly gives

$$4u_{1,1} - u_{0,1} - u_{2,1} - u_{1,0} - u_{1,2} = 0$$

$$4u_{2,1} - u_{1,1} - u_{3,1} - u_{2,0} - u_{2,2} = 0$$

$$4u_{1,2} - u_{0,2} - u_{2,2} - u_{1,1} - u_{1,3} = 0$$

$$4u_{2,2} - u_{1,2} - u_{3,2} - u_{2,1} - u_{2,3} = 0$$

Laplace Equation, continued

Writing previous equations in matrix form gives

$$\boldsymbol{Ax} = \begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{1,2} \\ u_{2,2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \boldsymbol{b}$$

System of equations can be solved for unknowns $u_{i,j}$ either by direct method based on factorization or by iterative method, yielding solution

$$\boldsymbol{x} = \begin{bmatrix} u_{1,1} \\ u_{2,1} \\ u_{1,2} \\ u_{2,2} \end{bmatrix} = \begin{bmatrix} 0.125 \\ 0.125 \\ 0.375 \\ 0.375 \end{bmatrix}$$

Laplace Equation, continued

In a practical problem, mesh size h would be much smaller, and resulting linear system would be much larger

Matrix would be very sparse, however, since each equation would still involve only five variables, thereby saving substantially on work and storage

Finite Element Methods

Finite element methods are also applicable to boundary value problems for PDEs as well as for ODEs

Conceptually, there is no change in going from one dimension to two or three dimensions:

- Solution is represented as linear combination of basis functions
- Some criterion (e.g., Galerkin) is applied to derive system of equations that determines coefficients of linear combination

Main practical difference is that instead of subintervals in one dimension, elements usually become triangles or rectangles in two dimensions, or tetrahedra or hexahedra in three dimensions

Finite Element Methods, continued

Basis functions typically used are bilinear or bicubic functions in two dimensions or trilinear or tricubic functions in three dimensions, analogous to “hat” functions or piecewise cubics in one dimension

Increase in dimensionality means that linear system to be solved is much larger, but it is still sparse due to local support of basis functions

Finite element methods for PDEs are extremely flexible and powerful, but detailed treatment of them is beyond scope of this course

Sparse Linear Systems

Boundary value problems and implicit methods for time-dependent PDEs yield systems of linear algebraic equations to solve

Finite difference schemes involving only a few variables each, or localized basis functions in finite element approach, cause linear system to be *sparse*, with relatively few nonzero entries

Sparsity can be exploited to use much less than $\mathcal{O}(n^2)$ storage and $\mathcal{O}(n^3)$ work required in naive approach to solving system

Sparse Factorization Methods

Gaussian elimination and Cholesky factorization are applicable to large sparse systems, but care is required to achieve reasonable efficiency in solution time and storage requirements

Key to efficiency is to store and operate on only nonzero entries of matrix

Special data structures are required instead of simple 2-D arrays for storing dense matrices

Band Systems

For 1-D problems, equations and unknowns can usually be ordered so that nonzeros are concentrated in narrow band, which can be stored efficiently in rectangular 2-D array by diagonals

Bandwidth can often be reduced by reordering rows and columns of matrix

For problems in two or more dimensions, even narrowest possible band often contains mostly zeros, so 2-D array storage is wasteful

General Sparse Data Structures

In general, sparse systems require data structures that store only nonzero entries, along with indices to identify their locations in matrix

Explicitly storing indices incurs additional storage overhead and makes arithmetic operations on nonzeros less efficient due to indirect addressing to access operands

Data structure is worthwhile only if matrix is sufficiently sparse, which is often true for very large problems arising from PDEs and many other applications

Fill

When applying LU or Cholesky factorization to sparse matrix, taking linear combinations of rows or columns to annihilate unwanted nonzero entries can introduce new nonzeros into matrix locations that were initially zero

Such new nonzeros, called *fill*, must be stored and may eventually be annihilated themselves in order to obtain triangular factors

Resulting triangular factors can be expected to contain at least as many nonzeros as original matrix and usually a significant amount of fill as well

Reordering to Limit Fill

Amount of fill is sensitive to order in which rows and columns of matrix are processed, so basic problem in sparse factorization is reordering matrix to limit fill during factorization

Exact minimization of fill is hard combinatorial problem (NP-complete), but heuristic algorithms such as minimum degree and nested dissection do well at limiting fill for many types of problems

Example: Laplace Equation

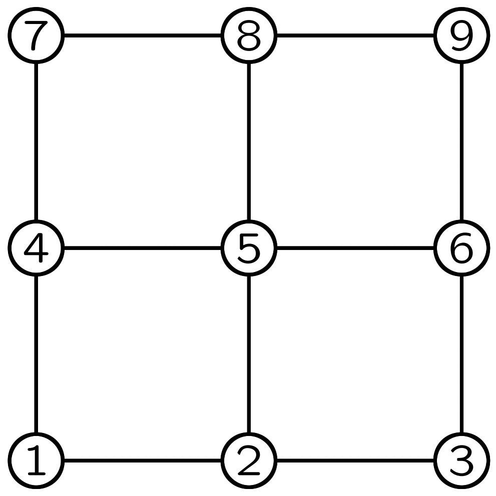
Discretization of Laplace equation on square by second-order finite difference approximation to second derivatives yields system of linear equations whose unknowns correspond to mesh points (nodes) in square grid

A pair of nodes is connected by an *edge* if both appear in the same equation in this system

Two nodes are *neighbors* if they are connected by an edge

Diagonal entries of matrix correspond to nodes in mesh, and nonzero off-diagonal entries correspond to edges in mesh ($a_{ij} \neq 0 \Leftrightarrow$ nodes i and j are neighbors)

Grid and Matrix



$$\begin{bmatrix} \times & \times & & \times & & & & & \\ \times & \times & \times & & \times & & & & \\ & \times & \times & & & \times & & & \\ \times & & & \times & \times & & \times & & \\ & \times & & \times & \times & \times & & \times & \\ & & \times & & \times & \times & & & \times \\ & & & \times & & & \times & \times & \\ & & & & \times & & \times & \times & \times \\ & & & & & \times & & \times & \times \end{bmatrix}$$

With nodes numbered row-wise (or column-wise), matrix is block tridiagonal, with each nonzero block either tridiagonal or diagonal

Matrix is banded but has many zero entries inside band

Matrix and Factor

$$\underset{\boldsymbol{A}}{\begin{bmatrix}
\times & \times & & \times & & & & & \\
\times & \times & \times & & \times & & & & \\
& \times & \times & & & \times & & & \\
\times & & & \times & \times & & \times & & \\
& \times & & \times & \times & \times & & \times & \\
& & \times & & \times & \times & & & \times \\
& & & \times & & & \times & \times & \\
& & & & \times & & \times & \times & \times \\
& & & & & \times & & \times & \times
\end{bmatrix}}
\qquad
\underset{\boldsymbol{L}}{\begin{bmatrix}
\times & & & & & & & & \\
\times & \times & & & & & & & \\
& \times & \times & & & & & & \\
\times & + & + & \times & & & & & \\
& \times & + & \times & \times & & & & \\
& & \times & + & \times & \times & & & \\
& & & \times & + & + & \times & & \\
& & & & \times & + & \times & \times & \\
& & & & & \times & + & \times & \times
\end{bmatrix}}$$

Cholesky factorization fills in band almost completely

Graph Model of Elimination

Each step of factorization process corresponds to elimination of one node from mesh

Eliminating a node causes its neighboring nodes to become connected to each other

If any such neighbors were not already connected, then *fill* results (new edges in mesh and new nonzeros in matrix)

Minimum Degree Ordering

Good heuristic for limiting fill is to eliminate first those nodes having fewest neighbors

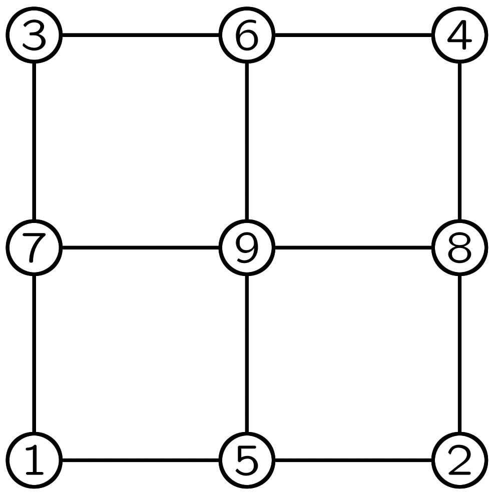
Number of neighbors is called *degree* of node, so heuristic is known as *minimum degree*

At each step, minimum degree selects for elimination a node of smallest degree, breaking ties arbitrarily

After node has been eliminated, its neighbors become connected to each other, so degrees of some nodes may change

Process is then repeated, with a new node of minimum degree eliminated next, and so on until all nodes have been eliminated

Grid and Matrix with Minimum Degree Ordering



$$\begin{bmatrix}
\times & & & & \times & & \times & & \\
 & \times & & & \times & & & \times & \\
 & & \times & & & \times & \times & & \\
 & & & \times & & \times & & \times & \\
\times & \times & & & \times & & & & \times \\
 & & \times & \times & & \times & & & \times \\
\times & & \times & & & & \times & & \times \\
 & \times & & \times & & & & \times & \times \\
 & & & & \times & \times & \times & \times & \times
\end{bmatrix}$$

Matrix and Factor with Minimum Degree Ordering

$$\underset{\boldsymbol{A}}{\begin{bmatrix} \times & & & & \times & & \times & & \\ & \times & & & \times & & & \times & \\ & & \times & & & \times & \times & & \\ & & & \times & & \times & & \times & \\ \times & \times & & & \times & & & & \times \\ & & \times & \times & & \times & & & \times \\ \times & & \times & & & & \times & & \times \\ & \times & & \times & & & & \times & \times \\ & & & & \times & \times & \times & \times & \times \end{bmatrix}} \qquad \underset{\boldsymbol{L}}{\begin{bmatrix} \times & & & & & & & & \\ & \times & & & & & & & \\ & & \times & & & & & & \\ & & & \times & & & & & \\ \times & \times & & & \times & & & & \\ & & \times & \times & & \times & & & \\ \times & & \times & & + & + & \times & & \\ & \times & & \times & + & + & + & \times & \\ & & & & \times & \times & \times & \times & \times \end{bmatrix}}$$

Cholesky factor suffers much less fill than with original ordering, and advantage grows with problem size

Sophisticated versions of minimum degree are among most effective general-purpose orderings known

Nested Dissection Ordering

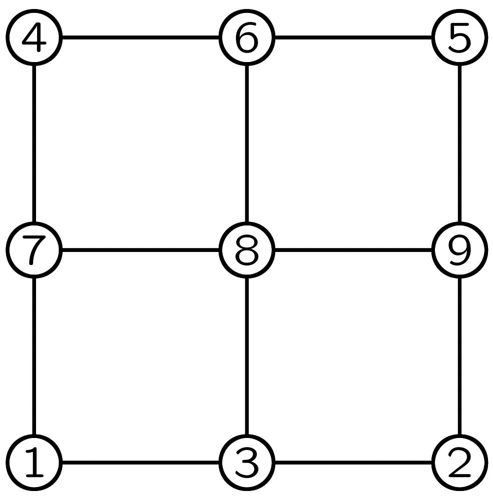
Nested dissection is based on a divide-and-conquer strategy

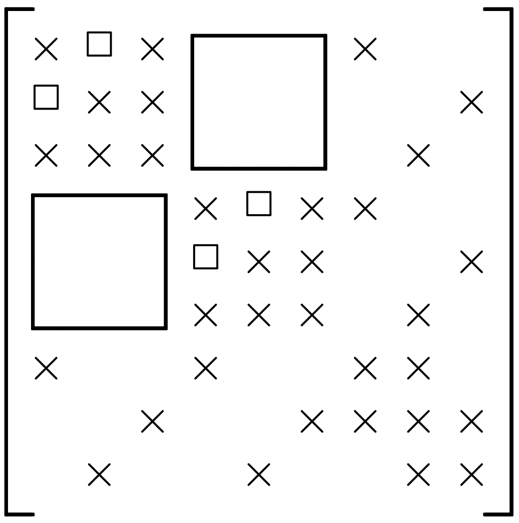
First, a small set of nodes is selected whose removal splits mesh into two pieces of roughly equal size

No node in either piece is connected to any node in other, so no fill occurs in either piece due to elimination of any node in the other

Separator nodes are numbered *last*, then process is repeated recursively on each remaining piece of mesh until all nodes have been numbered

Grid and Matrix with Nested Dissection Ordering

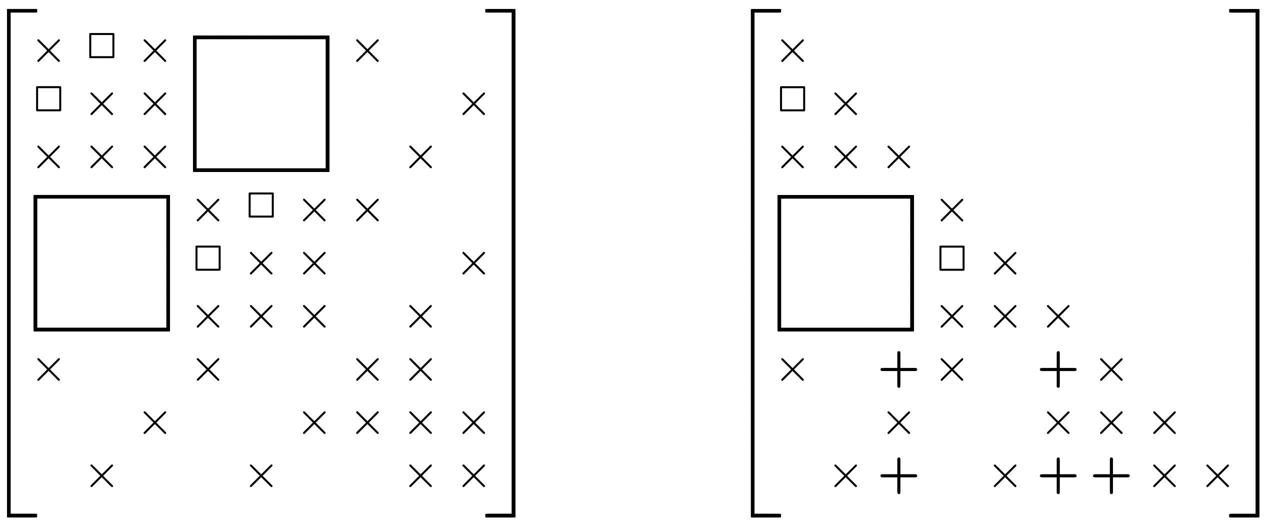




Dissection induces blocks of zeros in matrix (indicated by squares in diagram) that are automatically preserved during factorization

Recursive nature of algorithm can be seen in hierarchical block structure of matrix, which would involve many more levels in larger problems

Matrix and Factor with Nested Dissection Ordering



$\boldsymbol{A}$ $\boldsymbol{L}$

Again, Cholesky factor suffers much less fill than with original ordering, and advantage grows with problem size

Sparse Factorization Methods

Sparse factorization methods are accurate, reliable, and robust

They are methods of choice for 1-D problems and are usually competitive for 2-D problems, but they can be prohibitively expensive in both work and storage for very large 3-D problems

We will see that iterative methods provide a viable alternative in these cases

Fast Direct Methods

For certain elliptic boundary value problems, such as Poisson equation on rectangular domain, fast Fourier transform can be used to compute solution to discrete system very efficiently

For problem with n mesh points, solution can be computed in $\mathcal{O}(n \log n)$ operations

This technique is basis for several "fast Poisson solver" software packages

Cyclic reduction can achieve similar efficiency, and is somewhat more general

FACR method combines Fourier analysis and cyclic reduction to produce even faster method with $\mathcal{O}(n \log \log n)$ complexity

Iterative Methods for Linear Systems

Iterative methods for solving linear systems begin with initial guess for solution and successively improve it until solution is as accurate as desired

In theory, infinite number of iterations might be required to converge to exact solution

In practice, iteration terminates when residual $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|$, or some other measure of error, is as small as desired

Stationary Iterative Methods

Simplest type of iterative method for solving $\boldsymbol{Ax} = \boldsymbol{b}$ has form

$$\boldsymbol{x}_{k+1} = \boldsymbol{G}\boldsymbol{x}_k + \boldsymbol{c},$$

where matrix $\boldsymbol{G}$ and vector $\boldsymbol{c}$ are chosen so that fixed point of function $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{Gx} + \boldsymbol{c}$ is solution to $\boldsymbol{Ax} = \boldsymbol{b}$

Method is *stationary* if $\boldsymbol{G}$ and $\boldsymbol{c}$ are constant over all iterations

$\boldsymbol{G}$ is Jacobian matrix of fixed-point function $\boldsymbol{g}$, so stationary iterative method converges if

$$\rho(\boldsymbol{G}) < 1,$$

and smaller spectral radius yields faster convergence

Example: Iterative Refinement

Iterative refinement is example of stationary iterative method

Forward and back substitution using LU factorization in effect provides approximation, call it $\boldsymbol{B}^{-1}$, to inverse of $\boldsymbol{A}$

Iterative refinement has form

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{B}^{-1}(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k) \\ &= (\boldsymbol{I} - \boldsymbol{B}^{-1}\boldsymbol{A})\boldsymbol{x}_k + \boldsymbol{B}^{-1}\boldsymbol{b},\end{aligned}$$

so it is stationary iterative method with

$$\boldsymbol{G} = \boldsymbol{I} - \boldsymbol{B}^{-1}\boldsymbol{A}, \qquad \boldsymbol{c} = \boldsymbol{B}^{-1}\boldsymbol{b},$$

and it converges if

$$\rho(\boldsymbol{I} - \boldsymbol{B}^{-1}\boldsymbol{A}) < 1$$

Splitting

One way to obtain matrix $\boldsymbol{G}$ is by *splitting*,

$$\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N},$$

with $\boldsymbol{M}$ nonsingular

Then take $\boldsymbol{G} = \boldsymbol{M}^{-1}\boldsymbol{N}$, so iteration scheme is

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

which is implemented as

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

(i.e., we solve linear system with matrix $\boldsymbol{M}$ at each iteration)

Convergence

Stationary iteration using splitting converges if

$$\rho(\boldsymbol{G}) = \rho(\boldsymbol{M}^{-1}\boldsymbol{N}) < 1,$$

and smaller spectral radius yields faster convergence

For fewest iterations, should choose $\boldsymbol{M}$ and $\boldsymbol{N}$ so $\rho(\boldsymbol{M}^{-1}\boldsymbol{N})$ is as small as possible, but cost per iteration is determined by cost of solving linear system with matrix $\boldsymbol{M}$, so there is tradeoff

In practice, $\boldsymbol{M}$ is chosen to approximate $\boldsymbol{A}$ in some sense, but is constrained to have simple form, such as diagonal or triangular, so linear system at each iteration is easy to solve

Jacobi Method

In matrix splitting $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$, simplest choice for $\boldsymbol{M}$ is diagonal, specifically diagonal of $\boldsymbol{A}$

Let $\boldsymbol{D}$ be diagonal matrix with same diagonal entries as $\boldsymbol{A}$, and let $\boldsymbol{L}$ and $\boldsymbol{U}$ be strict lower and upper triangular portions of $\boldsymbol{A}$

Then $\boldsymbol{M} = \boldsymbol{D}$ and $\boldsymbol{N} = -(\boldsymbol{L} + \boldsymbol{U})$ gives splitting of $\boldsymbol{A}$

Assuming $\boldsymbol{A}$ has no zero diagonal entries, so $\boldsymbol{D}$ is nonsingular, this gives *Jacobi method*:

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

Jacobi Method, continued

Rewriting this scheme componentwise, we see that, beginning with initial guess $\boldsymbol{x}^{(0)}$, Jacobi method computes next iterate by solving for each component of $\boldsymbol{x}$ in terms of others:

$$x_i^{(k+1)} = \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right) / a_{ii}, \quad i = 1, \ldots, n$$

Jacobi method requires double storage for $\boldsymbol{x}$, since old component values are needed throughout sweep, so new component values cannot overwrite them until sweep has been completed

Example: Jacobi Method

If we apply Jacobi method to system of finite difference equations for Laplace equation, we get

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left(u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}\right),$$

which means new approximate solution at each grid point is average of previous solution at four surrounding grid points

Jacobi method does not always converge, but it is guaranteed to converge under conditions that are often satisfied in practice (e.g., if matrix is strictly diagonally dominant)

Unfortunately, convergence rate of Jacobi is usually unacceptably slow

Gauss-Seidel Method

One reason for slow convergence of Jacobi method is that it does not make use of latest information available

Gauss-Seidel method remedies this by using each new component of solution as soon as it has been computed:

$$x_i^{(k+1)} = \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) / a_{ii}$$

Using same notation as before, Gauss-Seidel method corresponds to splitting $\boldsymbol{M} = \boldsymbol{D} + \boldsymbol{L}$ and $\boldsymbol{N} = -\boldsymbol{U}$, and can be written in matrix terms as

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \boldsymbol{D}^{-1} \left(\boldsymbol{b} - \boldsymbol{L}\boldsymbol{x}^{(k+1)} - \boldsymbol{U}\boldsymbol{x}^{(k)} \right) \\ &= (\boldsymbol{D} + \boldsymbol{L})^{-1} \left(\boldsymbol{b} - \boldsymbol{U}\boldsymbol{x}^{(k)} \right) \end{aligned}$$

Gauss-Seidel Method, continued

In addition to faster convergence, another benefit of Gauss-Seidel method is that duplicate storage is not needed for $\boldsymbol{x}$, since new component values can overwrite old ones immediately

On other hand, updating of unknowns must be done successively, in contrast to Jacobi method, in which unknowns can be updated in any order, or even simultaneously

If we apply Gauss-Seidel method to solve system of finite difference equations for Laplace equation, we get

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left(u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}\right)$$

Gauss-Seidel Method, continued

Thus, we again average solution values at four surrounding grid points, but always use new component values as soon as they become available, rather than waiting until current iteration has been completed

Gauss-Seidel method does not always converge, but it is guaranteed to converge under conditions that are often satisfied in practice, and are somewhat weaker than those for Jacobi method (e.g., if matrix is symmetric and positive definite)

Although Gauss-Seidel converges more rapidly than Jacobi, it is often still too slow to be practical

Successive Over-Relaxation

Convergence rate of Gauss-Seidel can be accelerated by *successive over-relaxation* (SOR), which in effect uses step to next Gauss-Seidel iterate as search direction, but with fixed search parameter denoted by ω

Starting with $\boldsymbol{x}^{(k)}$, first compute next iterate that would be given by Gauss-Seidel, $\boldsymbol{x}_{GS}^{(k+1)}$, then instead take next iterate to be

$$\begin{aligned}\boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \omega(\boldsymbol{x}_{GS}^{(k+1)} - \boldsymbol{x}^{(k)}) \\ &= (1-\omega)\boldsymbol{x}^{(k)} + \omega\boldsymbol{x}_{GS}^{(k+1)},\end{aligned}$$

which is weighted average of current iterate and next Gauss-Seidel iterate

Successive Over-Relaxation, cont.

ω is fixed *relaxation* parameter chosen to accelerate convergence

$\omega > 1$ gives *over*-relaxation, $\omega < 1$ gives *under*-relaxation, and $\omega = 1$ gives Gauss-Seidel method

Method diverges unless $0 < \omega < 2$, but choosing optimal ω is difficult in general and is subject of elaborate theory for special classes of matrices

Successive Over-Relaxation, cont.

Using same notation as before, SOR method corresponds to splitting

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

and can be written in matrix terms as

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \omega\left(\boldsymbol{D}^{-1}(\boldsymbol{b} - \boldsymbol{L}\boldsymbol{x}^{(k+1)} - \boldsymbol{U}\boldsymbol{x}^{(k)})\right. \\ &\quad \left. - \boldsymbol{x}^{(k)}\right) \\ &= (\boldsymbol{D} + \omega\boldsymbol{L})^{-1}\left((1-\omega)\boldsymbol{D} - \omega\boldsymbol{U}\right)\boldsymbol{x}^{(k)} \\ &\quad + \omega\,(\boldsymbol{D} + \omega\boldsymbol{L})^{-1}\boldsymbol{b} \end{aligned}$$

Conjugate Gradient Method

If $\boldsymbol{A}$ is $n \times n$ symmetric positive definite matrix, then quadratic function

$$\phi(\boldsymbol{x}) = \tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b}$$

attains minimum precisely when $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

Optimization methods have form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha\boldsymbol{s}_k,$$

where α is search parameter chosen to minimize objective function $\phi(\boldsymbol{x}_k + \alpha\boldsymbol{s}_k)$ along $\boldsymbol{s}_k$

For quadratic problem, negative gradient is residual vector,

$$-\nabla\phi(\boldsymbol{x}) = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} = \boldsymbol{r}$$

Line search parameter can be determined analytically,

$$\alpha = \boldsymbol{r}_k^T\boldsymbol{s}_k/\boldsymbol{s}_k^T\boldsymbol{A}\boldsymbol{s}_k$$

Conjugate Gradient Method, cont.

Using these properties, we obtain *conjugate gradient* (CG) method for linear systems

```
$\boldsymbol{x}_0 =$ initial guess
$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$
$\boldsymbol{s}_0 = \boldsymbol{r}_0$
for $k = 0, 1, 2, \ldots$
    $\alpha_k = \boldsymbol{r}_k^T \boldsymbol{r}_k / \boldsymbol{s}_k^T \boldsymbol{A}\boldsymbol{s}_k$
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$
    $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{s}_k$
    $\beta_{k+1} = \boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1} / \boldsymbol{r}_k^T \boldsymbol{r}_k$
    $\boldsymbol{s}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{s}_k$
end
```

Conjugate Gradient Method, cont.

Key features that make CG method effective:

- Short recurrence determines search directions that are $\boldsymbol{A}$-orthogonal (conjugate)
- Error is minimal over space spanned by search directions generated so far

Minimum error implies exact solution reached in at most n steps, since n linearly independent search directions must span whole space

In practice, loss of orthogonality due to rounding error spoils finite termination property, so method is used iteratively

Preconditioning

Convergence rate of CG can often be substantially accelerated by *preconditioning*

Apply CG algorithm to $\boldsymbol{M}^{-1}\boldsymbol{A}$, where $\boldsymbol{M}$ is chosen so that $\boldsymbol{M}^{-1}\boldsymbol{A}$ is better conditioned and systems of form $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{y}$ are easily solved

Some choices of preconditioners include:

- Diagonal or block-diagonal
- SSOR
- Incomplete factorization
- Polynomial
- Approximate inverse

Generalizations of CG Method

CG is not directly applicable to nonsymmetric or indefinite systems

CG cannot be generalized to nonsymmetric systems without sacrificing one of its two key properties (short recurrence and minimum error)

Nevertheless, several generalizations have been developed for solving nonsymmetric systems, including GMRES, QMR, CGS, BiCG, and Bi-CGSTAB

These tend to be less robust and require more storage than CG, but they can still be very useful for solving large nonsymmetric systems

Example: Iterative Methods

We illustrate various iterative methods by using them to solve 4×4 linear system for Laplace equation example

In each case we take $\boldsymbol{x}_0 = 0$ as starting guess

Jacobi method gives following iterates:

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.250	0.250
2	0.062	0.062	0.312	0.312
3	0.094	0.094	0.344	0.344
4	0.109	0.109	0.359	0.359
5	0.117	0.117	0.367	0.367
6	0.121	0.121	0.371	0.371
7	0.123	0.123	0.373	0.373
8	0.124	0.124	0.374	0.374
9	0.125	0.125	0.375	0.375

Example Continued

Gauss-Seidel method gives following iterates:

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.250	0.312
2	0.062	0.094	0.344	0.359
3	0.109	0.117	0.367	0.371
4	0.121	0.123	0.373	0.374
5	0.124	0.125	0.375	0.375
6	0.125	0.125	0.375	0.375

SOR method (with optimal $\omega = 1.072$ for this problem) gives following iterates:

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.268	0.335
2	0.072	0.108	0.356	0.365
3	0.119	0.121	0.371	0.373
4	0.123	0.124	0.374	0.375
5	0.125	0.125	0.375	0.375

Example Continued

CG method converges in only two iterations for this problem:

k	x_1	x_2	x_3	x_4
0	0.000	0.000	0.000	0.000
1	0.000	0.000	0.333	0.333
2	0.125	0.125	0.375	0.375

Rate of Convergence

For more systematic comparison of methods, we compare them on $k \times k$ model grid problem for Laplace equation on unit square

For this simple problem, spectral radius of iteration matrix for each method can be determined analytically, as well as optimal ω for SOR

Gauss-Seidel is asymptotically twice as fast as Jacobi for this model problem, but for both methods, number of iterations per digit of accuracy gained is proportional to number of mesh points

Optimal SOR is order of magnitude faster than either of them, and number of iterations per digit gained is proportional to number of mesh points along one side of grid

Rate of Convergence, continued

For some specific values of k, values of spectral radius are shown below

k	Jacobi	Gauss-Seidel	Optimal SOR
10	0.9595	0.9206	0.5604
50	0.9981	0.9962	0.8840
100	0.9995	0.9990	0.9397
500	0.99998	0.99996	0.98754

Spectral radius is extremely close to 1 for large values of k, so all three methods converge very slowly

For $k = 10$ (linear system of order 100), to gain single decimal digit of accuracy Jacobi method requires more than 50 iterations, Gauss-Seidel more than 25, and optimal SOR about 4

Rate of Convergence, continued

For $k = 100$ (linear system of order 10,000), to gain single decimal digit of accuracy Jacobi method requires about 5000 iterations, Gauss-Seidel about 2500, and optimal SOR about 37

Thus, Jacobi and Gauss-Seidel methods are impractical for problem of this size, and optimal SOR, though perhaps reasonable for this problem, also becomes prohibitively slow for still larger problems

Moreover, performance of SOR depends on knowledge of optimal value for relaxation parameter ω, which is known analytically for this simple model problem but is much harder to determine in general

Rate of Convergence, continued

Convergence behavior of CG is more complicated, but error is reduced at each iteration by factor of roughly

$$\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$$

on average, where

$$\kappa = \text{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\| \cdot \|\boldsymbol{A}^{-1}\| = \frac{\lambda_{\text{max}}(\boldsymbol{A})}{\lambda_{\text{min}}(\boldsymbol{A})}$$

When matrix $\boldsymbol{A}$ is well-conditioned, convergence is rapid, but if $\boldsymbol{A}$ is ill-conditioned, convergence can be arbitrarily slow

This is why preconditioner is usually used with CG method, so preconditioned matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ has much smaller condition number than $\boldsymbol{A}$

Rate of Convergence, continued

This estimate is conservative, and CG method may do much better

If matrix $\boldsymbol{A}$ has only m distinct eigenvalues, then theoretically CG converges in at most m iterations

Detailed convergence behavior depends on entire spectrum of $\boldsymbol{A}$, not just its extreme eigenvalues, and in practice convergence is often superlinear

Smoothers

Disappointing convergence rates observed for stationary iterative methods are *asymptotic*

Much better progress may be made initially before eventually settling into slow asymptotic phase

Many stationary iterative methods tend to reduce high-frequency (i.e., oscillatory) components of error rapidly, but reduce low-frequency (i.e., smooth) components of error much more slowly, which produces poor asymptotic rate of convergence

For this reason, such methods are sometimes called *smoothers*

This observation provides motivation for *multigrid* methods

Multigrid Methods

Smooth or oscillatory components of error are relative to mesh on which solution is defined

Component that appears smooth on fine grid may appear oscillatory when sampled on coarser grid

If we apply smoother on coarser grid, then we may make rapid progress in reducing this (now oscillatory) component of error

After a few iterations of smoother, results can then be interpolated back to fine grid to produce solution that has both higher-frequency and lower-frequency components of error reduced

Multigrid Methods, continued

This idea can be extended to multiple levels of grids, so that error components of various frequencies can be reduced rapidly, each at appropriate level

Transition from finer grid to coarser grid involves *restriction* or *injection*

Transition from coarser grid to finer grid involves *interpolation* or *prolongation*

Residual Equation

If $\hat{\boldsymbol{x}}$ is approximate solution to $\boldsymbol{Ax} = \boldsymbol{b}$, with residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}$, then error $\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ satisfies equation $\boldsymbol{Ae} = \boldsymbol{r}$

Thus, in improving approximate solution we can work with just this *residual equation* involving error and residual, rather than solution and original right-hand side

One advantage of residual equation is that zero is reasonable starting guess for its solution

Two-Grid Algorithm

1. On fine grid, use few iterations of smoother to compute approximate solution $\hat{\boldsymbol{x}}$ for system $\boldsymbol{Ax} = \boldsymbol{b}$

2. Compute residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{x}}$

3. Restrict residual to coarse grid

4. On coarse grid, use few iterations of smoother on residual equation to obtain coarse-grid approximation to error

5. Interpolate coarse-grid correction to fine grid to obtain improved approximate solution on fine grid

6. Apply few iterations of smoother to corrected solution on fine grid

Multigrid Methods, continued

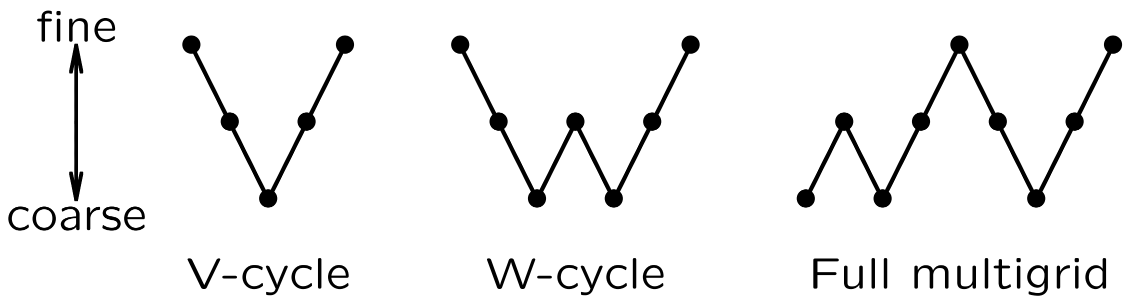
Multigrid method results from recursion in Step 4: coarse grid correction is itself improved by using still coarser grid, and so on down to some bottom level

Computations become progressively cheaper on coarser and coarser grids because systems become successively smaller

In particular, direct method may be feasible on coarsest grid if system is small enough

Cycling Strategies

There are many possible strategies for cycling through various grid levels, most common of which are depicted schematically below



V-cycle starts with finest grid and goes down through successive levels to coarsest grid and then back up again to finest grid

In order to get more benefit from coarser grids, where computations are cheaper, *W-cycle* zig-zags among lower level grids before moving back up to finest grid

Cycling Strategies, continued

Full multigrid starts at coarsest level, where good initial solution is easier to come by (perhaps by direct solution), then bootstraps this solution up through grid levels, ultimately reaching finest grid

Multigrid Methods, continued

By exploiting strengths of underlying iterative smoothers and avoiding their weaknesses, multigrid methods are capable of extraordinarily good performance

At each level, smoother reduces oscillatory component of error rapidly, at rate independent of mesh size h, since few iterations of smoother, often only one, are performed at each level

Since all components of error appear oscillatory at some level, convergence rate of entire multigrid scheme should be rapid and independent of mesh size, in contrast to other iterative methods

Multigrid Methods, continued

Moreover, cost of entire cycle of multigrid is only modest multiple of cost of single sweep on finest grid

As result, multigrid methods are among most powerful methods available for solving sparse linear systems arising from PDEs

They are capable of converging to within truncation error of discretization at cost comparable with fast direct methods, although latter are much less broadly applicable

Direct vs. Iterative Methods

- Direct methods require no initial estimate for solution, but take no advantage if good estimate happens to be available
- Direct methods are good at producing high accuracy, but take no advantage if only low accuracy is needed
- Iterative methods are often dependent on special properties, such as matrix being symmetric positive definite, and are subject to very slow convergence for badly conditioned systems. Direct methods are more robust in both senses

Direct vs. Iterative Methods

- Iterative methods usually require less work if convergence is rapid, but often require computation or estimation of various parameters or preconditioners to accelerate convergence
- Iterative methods do not require explicit storage of matrix entries, and hence are good when matrix can be produced easily on demand or is most easily implemented as linear operator
- Iterative methods are less readily embodied in standard software packages, since best representation of matrix is often problem dependent and “hard-coded” in application program, whereas direct methods employ more standard storage schemes

Computational Cost for $k \times k\,(\times k)$ Grid

method	2-D	3-D
Dense Cholesky	k^6	k^9
Jacobi	$k^4 \log k$	$k^5 \log k$
Gauss-Seidel	$k^4 \log k$	$k^5 \log k$
Band Cholesky	k^4	k^7
Optimal SOR	$k^3 \log k$	$k^4 \log k$
Sparse Cholesky	k^3	k^6
Conjugate Gradient	k^3	k^4
Optimal SSOR	$k^{2.5} \log k$	$k^{3.5} \log k$
Preconditioned CG	$k^{2.5}$	$k^{3.5}$
Optimal ADI	$k^2 \log^2 k$	$k^3 \log^2 k$
Cyclic Reduction	$k^2 \log k$	$k^3 \log k$
FFT	$k^2 \log k$	$k^3 \log k$
Multigrid V-cycle	$k^2 \log k$	$k^3 \log k$
FACR	$k^2 \log \log k$	$k^3 \log \log k$
Full Multigrid	k^2	k^3

Comparison of Methods

For those methods that remain viable choices for finite element discretizations with less regular meshes, computational cost of solving elliptic boundary value problems is given below in terms of exponent of n (order of matrix) in dominant term of cost estimate

method	2-D	3-D
Dense Cholesky	3	3
Band Cholesky	2	2.33
Sparse Cholesky	1.5	2
Conjugate Gradient	1.5	1.33
Preconditioned CG	1.25	1.17
Multigrid	1	1

Comparison of Methods, continued

Multigrid methods can be optimal, in sense that cost of computing solution is of same order as cost of reading input or writing output

FACR method is also optimal for all practical purposes, since $\log\log k$ is effectively constant for any reasonable value of k

Other fast direct methods are almost as effective in practice unless k is very large

Despite their higher cost, factorization methods are still useful in some cases due to their greater robustness, especially for nonsymmetric matrices

Methods akin to factorization are often used to compute effective preconditioners

Software for PDEs

Methods for numerical solution of PDEs tend to be very problem dependent, so PDEs are usually solved by custom written software to take maximum advantage of particular features of given problem

Nevertheless, some software does exist for a few general classes of problems that occur often in practice

In addition, several good software packages are available for solving sparse linear systems arising from PDEs as well as other sources