AMS526: Numerical Analysis I  
(Numerical Linear Algebra)  
Lecture 21: Sparse Linear Systems; Direct Methods vs. Iterative Methods

Xiangmin Jiao

SUNY Stony Brook
Outline

1. Sparse Linear Systems

2. Direct Methods for Sparse Linear Systems

3. Overview of Iterative Methods for Sparse Linear Systems
Sparse Linear System

- Boundary value problems and implicit methods for time-dependent PDEs yield systems of linear algebraic equations to solve.
- A matrix is *sparse* if it has relatively few nonzeros in its entries.
- Sparsity can be exploited to use much less than $O(n^2)$ storage and $O(n^3)$ work required in standard approach to solving system with dense matrix.
Storage Format of Sparse Matrices

- Sparse-matrices are typically stored in special formats that store only nonzero entries, along with indices to identify their locations in matrix, such as
  - compressed-row storage (CRS)
  - compressed-column storage (CCS)
  - block compressed row storage (BCRS)

- See survey at http://netlib.org/linalg/html_templates/node90.html
- Explicitly storing indices incurs additional storage overhead and makes arithmetic operations on nonzeros less efficient due to indirect addressing to access operands, so they are beneficial only for very sparse matrices
- Storage format can have big impact the effectiveness of different versions of same algorithm (with different ordering of loops)
- Besides direct methods, these storage formats are also important in implementing iterative and multigrid solvers
Example of Compressed-Row Storage (CRS)

\[ A = \begin{pmatrix}
10 & 0 & 0 & 0 & -2 & 0 \\
3 & 0 & 0 & 0 & 0 & 3 \\
0 & 7 & 8 & 7 & 0 & 0 \\
3 & 0 & 8 & 7 & 5 & 0 \\
0 & 8 & 0 & 0 & 0 & 13 \\
0 & 4 & 0 & 0 & 2 & -1
\end{pmatrix} \]

<table>
<thead>
<tr>
<th>val</th>
<th>10</th>
<th>-2</th>
<th>3</th>
<th>9</th>
<th>3</th>
<th>7</th>
<th>8</th>
<th>7</th>
<th>3</th>
<th>...</th>
<th>9</th>
<th>13</th>
<th>4</th>
<th>2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>col_ind</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>...</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>row_ptr</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. Sparse Linear Systems

2. Direct Methods for Sparse Linear Systems

3. Overview of Iterative Methods for Sparse Linear Systems
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* or *fill-in,* must be stored and may themselves eventually need to be annihilated in order to obtain triangular factors.

Resulting triangular factors can be expected to contain at least as many nonzeros as original matrix and usually significant fill as well.
Cost of factorizing banded linear system depends on bandwidth

- For SPD $n \times n$ matrix with semi-bandwidth $s$, total flop count of Cholesky factorization is about $ns^2$
- For $n \times n$ matrix with lower bandwidth $p$ and upper bandwidth $q$,
  - In $A = LU$ (LU without pivoting), total flop count is about $2npq$
  - In $PA = LU$ (LU with column pivoting), total flop count is about $2np(p + q)$

Banded matrices have their own special storage formats (such as Compressed Diagonal Storage (CDS))
In general, some heuristic algorithms are employed to reorder the matrix to reduce fills.

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 limit fill well for many types of problems.

For Cholesky factorization, both rows and columns are reordered.
Reordering to Reduce Bandwidth

- The Cuthill-McKee algorithm and reverse Cuthill-McKee algorithm
- The Cuthill-McKee algorithm is a variant of the breadth-first search algorithm on graphs.
  - Starts with a peripheral node
  - Generates levels $R_i$ for $i = 1, 2, \ldots$ until all nodes are exhausted
  - The set $R_{i+1}$ is created from set $R_i$ by listing all vertices adjacent to all nodes in $R_i$
  - Within each level, nodes are listed in increasing degree
- The reverse Cuthill–McKee algorithm (RCM) reserves the resulting index numbers
Graph Model of Elimination

- Each step of factorization process corresponds to elimination of one node from graph
- Eliminating 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 graph and new nonzeros in matrix)
- Commonly used reordering methods includes approximate minimum degree ordering (AMD) and nested dissection
Approximate 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, select node of smallest degree for elimination, 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 new node of minimum degree eliminated next, and so on until all nodes have been eliminated.
Minimum Degree Ordering, continued

- 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
Comparison of Different Orderings of Example Matrix

Nested Dissection Ordering

- Nested dissection is based on divide-and-conquer
- First, small set of nodes is selected whose removal splits graph 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 graph until all nodes have been numbered
Dissection induces blocks of zeros in matrix 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.

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

- For Gaussian elimination, only columns are reordered
- Pivoting introduces additional fills in sparse Gaussian elimination
- Reordering may be done dynamically or statically
- The reverse Cuthill-McKee algorithm applied to $A + A^T$ may be used to reduce bandwidth
- Column approximate minimum-degree, may be employed to reorder matrix to reduce fills
Comparison of Different Orderings of Example Matrix

Nonzero pattern of \( A \) and \( L + U \) with random ordering.

Nonzero pattern of \( A \) and \( L + U \) with column AMD ordering.
Comparison of Direct Methods

- Computational cost for Laplace equation on $k \times k(\times k)$ grid with $n$ unknowns

<table>
<thead>
<tr>
<th>method</th>
<th>2-D</th>
<th>3-D</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense Cholesky</td>
<td>$k^6$</td>
<td>$k^9$</td>
</tr>
<tr>
<td>banded Cholesky</td>
<td>$k^4$</td>
<td>$k^7$</td>
</tr>
<tr>
<td>sparse Cholesky</td>
<td>$k^3$</td>
<td>$k^6$</td>
</tr>
</tbody>
</table>

Software of Sparse Solvers

- Additional implementation complexities include cache performance and parallelism
- It is advisable to use software packages
- MATLAB has its own sparse solvers if matrix is stored in sparse format
  - Sparse matrix is created by using the “sparse” function
  - Reordering is implemented as “symrcm”, “symamd”, and “colamd”
- For symmetric matrices, a good software is Taucs
- For non-symmetric matrices, a good software is SuperLU
Outline

1. Sparse Linear Systems

2. Direct Methods for Sparse Linear Systems

3. Overview of Iterative Methods for Sparse Linear Systems
Direct vs. Iterative Methods

- **Direct methods**, or **noniterative methods**, compute the exact solution after a finite number of steps (in exact arithmetic)
  - Example: Gaussian elimination, QR factorization

- **Iterative methods** produce a sequence of approximations \( x^{(1)}, x^{(2)}, \ldots \) that hopefully converge to the true solution
  - Example: Jacobi, Conjugate Gradient (CG), GMRES, BiCG, etc.

Caution: The boundary between direct and iterative methods is vague sometimes

Why use iterative methods (instead of direct methods)?

- may be faster than direct methods
- produce useful intermediate results
- handle sparse matrices more easily (needs only matrix-vector product)
- often are easier to implement on parallel computers

Question: When not to use iterative methods?
Two Classes of Iterative Methods

- Stationary iterative methods is a fixed point iteration obtained by matrix splitting
  - Examples: Jacobi (for linear systems, not Jacobi iterations for eigenvalues), Gauss-Seidel, Successive Over-Relaxation (SOR) etc.

- Krylov subspace methods find optimal solution in Krylov subspace \( \{b, Ab, A^2b, \ldots A^k b\} \)
  - Build subspace successively
  - Example: Conjugate Gradient (CG), Generalized Minimum Residual (GMRES), BiCG, etc.
  - We will focus on Krylov subspace methods
Stationary Iterative Methods

- **Stationary iterative methods** find a splitting $A = M - N$ and iterates
  
  $$x^{(k+1)} = M^{-1}(N x^{(k)} + b)$$

- Suppose $r^{(k)} = b - A x^{(k)}$, we have $x^* = x^{(k)} + A^{-1} r^{(k)}$.

- Stationary iterative method approximates it by
  
  $$x^{(k+1)} = x^{(k)} + M^{-1} r^{(k)}$$

  because

  $$x^{(k+1)} = M^{-1} N x^{(k)} + M^{-1} b$$
  
  $$= M^{-1} N x^{(k)} + M^{-1} (r^{(k)} + A x^{(k)})$$
  
  $$= M^{-1} (N + A) x^{(k)} + M^{-1} r^{(k)}$$
  
  $$= x^{(k)} + M^{-1} r^{(k)}$$

- A stationary iterative method is good if
  
  - $\rho(M^{-1} N) < 1$, and
  - $M^{-1}$ is a good approximation to $A^{-1}$
Stationary Iterative Methods

- Different choices of splitting will lead to various schemes
- Let $A = L + D + U$, where $D$ is diagonal, $L$ is strictly lower triangular, and $U$ is strictly upper triangular.
  - Jacobi iteration: $M = D$, works well if $A$ is diagonally dominant
  - Gauss-Seidel: $M = L + D$, works well if $A$ is SPD
  - Successive Over-Relaxation (SOR): $M = \frac{1}{\omega}D + L$, where $1 \leq \omega < 2$, converges quickly proper choice of $\omega$
  - Symmetric SOR: symmetric version of SOR
- These methods work for some problems, but they may converge slowly
- Nevertheless, stationary methods are important as preconditioners for Krylov-subspace methods and smoothers in multigrid methods (later)
Stationary Iterative Methods

Example

For 2D Poisson equation, spectral radius of Jacobi iteration matrix is 
\[
\cos \left( \frac{\pi}{n} \right) \approx 1 - O \left( \frac{1}{n^2} \right).
\]

Number of iterations required to achieve \( \epsilon \) is 
\[
O(n^2 \ln \epsilon^{-1}).
\]

After 5 Jacobi iterations on a Poisson equation, error decreases very slowly.