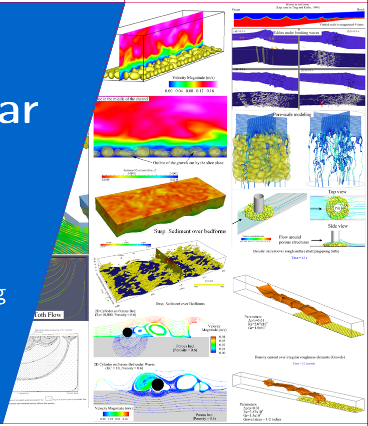
CE597D Special Topics, Fall, 2014

Chapter 4, Part 5: Linear System Solvers



Xiaofeng Liu, Ph.D., P.E.
Assistant Professor
Department of Civil and Environmental Engineering
Pennsylvania State University
xliu@engr.psu.edu

Outline

Linear system solvers

Linear System of Equations

This chapter is not intended to be a complete lecture on linear system solvers. For these, you need to take a course or read some good books.

- Yousef Saad, Iterative Methods for Sparse Linear Systems, second edition http://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf

The content of this lecture is mostly from online forum, open reports, and other people's lecture notes.

Linear System of Equations

In OpenFOAM® , the major part related to the linear system of equations is in

`src/OpenFOAM/matrices/lduMatrix/`

It has the following sub-directories:

- `lduAddressing`: addressing for lower triangle, diagonal, and upper triangle coefficients of the matrix
- `lduMatrix`: the `ldu` matrix itself
- `preconditioners`: preconditioners for linear solvers
- `smoothers`: smoothers for linear solvers
- `solvers`: linear solvers

Linear System of Equations

Nomenclature

- For each computational cell, we will create an equation

$$a_P \phi_P + \sum_N a_N \phi_N = b$$

- Equations form a **linear system** or a matrix

$$A\phi = b$$

  where A contain matrix coefficients, ϕ is the value of ϕ_P in all cells and b is the right-hand-side
- A is potentially very big: M $\times$ M, M being the number cells
- A is a **square matrix**: the number of equations equals the number of unknowns
- A is sparse, i.e., very few coefficients are non-zero. The reason is descretizaiton stencil is local.
- A needs good format to reduce storage size and computational efficiency

Matrix Storage Formats

Since most of CFD codes have to deal with sparse matrix, we will only look at sparse cases.

- Sparse matrix format: Only non-zero coefficients will be stored.
  - Considerable savings in memory
  - Need a mechanism to indicate the position of non-zero coefficients
  - The format is static, which imposes limitations on the operations: if a coefficient is originally zero and not stored, it is very expensive to set its value. This is usually termed a zero fill-in condition.
  - There are several sparse matrix storage formats.
    - coordinate format
    - compressed row storage (CRS) format
    - compressed column storage (CCS) format
    - ...

More information:

A survey of sparse matrix storage formats:

http://netlib.org/linalg/html_templates/node90.html

Section 3.4 of Saad (2003) (Free online book).

Matrix Storage Formats

Design of sparse matrix format needs to consider:

- Storage overhead
- Access of matrix elements
- Operations on the matrix (e.g., LU decomposition)
- Operations with the matrix (e.g., matrix-vector multiplication)

Matrix Storage Formats

Example: Compressed Row Storage (CRS) Format

- Three arrays: `value` for the matrix coefficients, `column` and `row` for addressing
- Coefficients ordered row-by-row
- The column array records the column index for each coefficients. Size of column array equal to the number of off-diagonal coefficients
- The row array records the start and end of each row in the column array. Thus, row i has got coefficients from $row[i]$ to $row[i+1]$ (not included). Size of row arrays equal to number of rows + 1

$$A = \begin{bmatrix} 1 & & & & \\ & 2 & 5 & & \\ & 3 & 6 & & 9 \\ & 4 & & 8 & \\ & & 7 & & 10 \end{bmatrix} \tag{1}$$

$$\begin{aligned} val &= [1, 2, 5, 3, 6, 9, 4, 8, 7, 10] \\ column &= [1, 2, 3, 2, 3, 5, 2, 4, 3, 5] \\ row &= [1, 2, 4, 7, 9, 11] \end{aligned}$$

Matrix Storage Formats

Example: Arrow Format (used in OpenFOAM®)

- Actual data stored in the `lduMatrix` class and addressing implemented in the `lduAddressing` class
- Arbitrary sparse format. Diagonal coefficients typically stored separately
- Coefficients in 2-3 arrays: diagonal, upper and lower triangle in `lduMatrix` class, which has member functions:

```
const scalarField & lower () const
const scalarField & diag () const
const scalarField & upper () const
```

- Diagonal addressing implied (cells)
- Off-diagonal addressing in 2 arrays: "owner" (row index) "neighbour" (column index) array.
- Size of lower and upper addressing = the number of coefficients = number of internal faces
- If the matrix coefficients are symmetric, only the upper triangle is stored – a symmetric matrix is easily recognised and stored only half of coefficients

Matrix Storage Formats

For example, given a simple 3×3 orthogonal mesh (in 2-dimensions):

-1-	-2-	-3-
-6-	-5-	-4-
-7-	-8-	-9-

The coefficient matrix for diffusion equation might look like:

i	-1-	-2-	-3-	-4-	-5-	-6-	-7-	-8-	-9-
-1-	X	N				N			
-2-	O	X	N		N				
-3-		O	X	N					
-4-			O	X	N				N
-5-		O		O	X	N		N	
-6-	O				O	X	N		
-7-						O	X	N	
-8-					O		O	X	N
-9-				O				O	X

Where X indicates diagonal cells, O and N indicate non-zero off-diagonal cells.

Matrix Storage Formats

- Since the matrix is sparse, only the non-zero entries need to be stored.
  - All diagonal coefficients a_P are non-zero.
  - Sparsity only refers to the off-diagonal.
  - lower and upper triangles have $N(N-1)$ entries; most of them are zero.
  - The non-zero coefficients correspond to "internal faces" in the mesh: one appears in the lower triangle and one in upper triangle
  - The two coefficients corresponding to one internal face could be the same (e.g., diffusion term) or different (e.g., advection term)
  - So Laplace (Poisson) equation results a symmetric matrix A, while others (advection-diffusion) results in asymmetry.

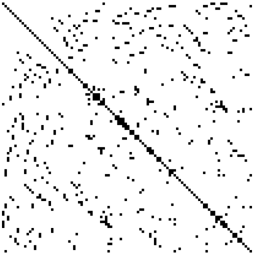


Figure: An example of sparse matrix (source:wikipedia)

Matrix Storage Formats

- The storage of matrix is implemented in *lduAddressing* class in OpenFOAM® .
- It works closely with the mesh data structure
  - points: (x, y, z) coordinates
  - faces: list of points whose order makes sure the face normal pointing from owner to neighbor (using right hand rule). Boundary faces always point outward.
  - face owner and neighbor: cells on both sides of the face. The owner always has lower number. Boundary faces only have owner and no neighbor.
  - The `owner` and `neighbor` files inside `constant/polyMesh` directory: `owner` has all the faces (internal + boundary); `neighbor` only have internal faces and thus is shorter and `owner` by the number of boundary faces.
  - Thus `neighbor`, i.e., internal faces, determines the off-diagonal coefficients.
  - the
  - Boundary patches are defined in `boundary` file: name, type, number of faces `nFaces`, start index `startFace` in the `face` list.
  - Remember the effect of boundary faces: only affect the diagonal and the right hand side of the linear equation. The class `fvMatrix` (inherited from `lduMatrix` has two functions:

```
addBoundaryDiag();
addBoundarySource();
```

PENNSTATE

Matrix Storage Formats

- *lduAddressing* works by recognizing that:
  - Every cell has a diagonal coefficient; and
  - Every cell has off-diagonal coefficients for each of their neighbors.
- Therefore the diagonal coefficients are stored in an "N"-long vector array, where "N" is the number of cells.
- The number of off-diagonal coefficients is equal to the number of cell-pairs (internal faces) that directly influence one another in the linearized equations.
- At the matrix level in OpenFOAM® , this only includes "adjacent" cells. Therefore the number of off-diagonal coefficients is equal to the number of shared "faces" (internal faces) in the mesh.

Matrix Storage Formats

`lduAddressing`:

- The diagonal coefficients are indexed by "cell index". So `Diag[cellI]` is the diagonal coefficient for cellI.
- The off-diagonal coefficients (lower and upper triangles) are indexed by "face index".
- For example, to recover the full 2D matrix:

```
//A defined as a 2D matrix
//Loop over all the cells to assemble the diagonal coefficents
for (label cellI=0; cellI<Diag.size(); cellI++)
{
   A[cellI, cellI] = Diag[cellI];
}

//Loop over all the internal faces to assemble the
//off-diagonal coefficients.
for (label face=0; face<l.size(); face++)
{
   A[l[face],u[face]] = lower[face];
   A[u[face],l[face]] = upper[face];
}
```

Matrix Storage Formats

`lduAddressing`:

- `l` and `u` are lower and upper triangle addressing
- The lower triangle is called “owner” coefficients which comes from the face’s owner cell
- The upper triangle is called “neighbor” coefficients which comes from the face’s neighbor cell

Matrix Format and Discretisation

FVM and Matrix Structure: `fvMatrix`

- A class inherited from `lduMatrix`
- It adds:
  - Reference to the solution variable ϕ
  - Dimensions: M, L, T, etc.
  - Stores the right hand side b.
  - Handles boundary condition effects in the discretization:

    `addBoundaryDiag()`

    `addBoundarySource()`
  - ...

Linear Solvers

The Role of a Linear Solver

- Numerical simulation software will spend a big portion of CPU time solving $Ax = b$:
- Performance of linear solvers is absolutely critical for the performance of the simulation
- The performance depends on several factors:
  - the linear solver itself
  - the property of matrix A due to discretization schemes

Linear Solvers

Category of linear solvers:

- **Direct solver**
  - Example: LU decomposition
  - Not attractive to solver large linear system
  - Computational time grows very quickly as $O(N^3)$ or $O(N^2)$
  - Exact solution is not necessary since the linear system is an approximation
  - Breaks sparsity
- **Iterative solver**:
  - The algorithm will start from an initial solution and perform a number of operations which will result in an improved solution.

$$\phi^{(0)} \rightarrow \phi^{(1)} \rightarrow \phi^{(2)} ... \rightarrow \phi^{(i)} \rightarrow ... \rightarrow \phi$$

  - Usually takes advantage and preserve the sparsity pattern
  - Types:
    - Relaxation methods: Jacobi-, Gauss-Seidel-Relaxation, ...
    - Krylov subspace solvers: PCG, PBiCG, etc.
    - Multigrid methods

Properties of Iterative Solvers

- Performance of iterative solvers depends on the matrix characteristics.

$$Ax = b$$

- The solver operates by incrementally improving the solution, i.e., reducing the error
- If the error is amplified in the iterative process, the solver diverges
- Preconditioners: make sure the convergence of the preconditioned system is much faster than the original one.

$$M^{-1}Ax = M^{-1}b$$

- Smoothers: smoothing algorithms for multi-grid method guarantee that the approximate solution after each solver iteration will be closer to the exact solution than all previous approximation. An example of a smoother is the Gauss-Seidel algorithm

Matrix Properties

Matrix Characterisation

- A matrix is **sparse** if it contains only a few non-zero elements
- A sparse matrix is **banded** if its non-zero coefficients are grouped in a stripe around the diagonal
- A sparse matrix has a **multi-diagonal structure**if its non-zero off-diagonal coefficients form a regular diagonal pattern
- A **symmetric** matrix is equal to its transpose

$$[A] = [A]^T$$

- A matrix is **positive definite** if for every $[\phi] \neq [0]$

$$[\phi]^T [A][\phi] > 0$$

Matrix Properties

Matrix Characterisation

- A matrix is **diagonally dominant** if in each row the sum of off-diagonal coefficient magnitudes is equal or smaller than the diagonal coefficient

$$a_{ii} \geq \sum_{j=1}^{N} |a_{ij}| \quad ; \quad j \neq i$$

and for at least one i

$$a_{ii} > \sum_{j=1}^{N} |a_{ij}| \quad ; \quad j \neq i$$

Definition of a Residual

- Matrix form of the system we are trying to solve is

$$[A][\phi] = [b]$$

- The exact solution can be obtained by inverting the matrix $[A]$:

$$[\phi] = [A]^{-1}\,[b]$$

- Iterative solvers start from an approximate solution ϕ^0 and generates a sequence of solution estimates ϕ^k, where k is the iteration counter
- convergence is measured through residual:

$$[r]^k = [b] - [A][\phi]^k$$

Residual is a vector showing how far is the current estimate $[\phi]^k$ from the exact solution $[\phi]$.

Definition of a Residual

- How to measure residual? Norm of $||r||$

$$||r|| = \sum_{j=1}^{N} |r_j|$$

which i the absolute error norm.

- We usually need to normalize it for easy quantification (compare with `tolerance` in `fvSolution`).

$$R_s = \frac{||r||}{normFactor} < tolerance\,?$$

Definition of a Residual

- How to define the `normFactor`? OpenFOAM® does it in a slightly unusual way.

Residual is defined as

$$\mathrm{r} = \mathrm{b} - \mathrm{A}\phi$$

OpenFOAM® defines a reference solution value ϕ_{ref} = volume average of ϕ over the whole domain, then

$$w_A = A\phi^k$$

$$p_A = A\phi^k_{ref}$$

The scaling factor is then calculated as

$$scaleFactor = \sum |w_A - p_A| + |b - p_A| + 10^{-20}$$

The scaled absolute error norm:

$$r_s = \frac{\sum |b - w_A|}{scaleFactor}$$

Definition of a Residual

- This definition of `normFactor` breaks down when the solution $\phi \equiv 0$ and `normFactor` is a very small number.
- The code implementation is in class `solverPerformance` and the linear system solver such as `PCG`

  `src/OpenFOAM/matrices/lduMatrix/solvers/PCG`

- You can print out the value of `normFactor` by setting the debug level to ≤ 2 because all solvers have:

```
if (lduMatrix::debug >= 2)
{
    Info<< "   Normalisation factor = " << normFactor << en
}
```

- The actual calculation of `normFactor` is in the `lduMatrix` class:

```
src/OpenFOAM/matrices/lduMatrix/lduMatrix
...
Foam::scalar Foam::lduMatrix::solver::normFactor(...)
```

Definition of a Residual

- To check the convergence, one can also use relative convergence measure (compare with `relTol` in `fvSolution`)

$$\frac{||r_k||}{||r_0||} < relTol\,?$$

- If both `tolerance` and `relTol` are specified, the stopping criterion is "OR" relationship, . This is coded in `/src/OpenFOAM/matrices/LduMatrix/LduMatrix`

```
if
(
    finalResidual_ < Tolerance
 || (
        RelTolerance
      > small_*pTraits<Type>::one
     && finalResidual_ < cmptMultiply(RelTolerance, initial
    )
)
{ converged_ = true;}
```

PENNSTATE

Examples of Iterative Solvers

In OpenFOAM® , the linear solvers are in

`src/OpenFOAM/matrices/lduMatrix/solvers`

- BICCG: Diagonal incomplete LU preconditioned BiCG solver
- diagonalSolver: diagonal solver for both symmetric and asymmetric problems
- GAMG: Geometric agglomerated algebraic multigrid solver (also named Generalised geometric-algebraic multi-grid in the manual)
- ICCG: Incomplete Cholesky preconditioned Conjugate Gradients solver
- PBiCG: Preconditioned bi-conjugate gradient solver for asymmetric lduMatrices using a run-time selectable preconditioner
- PCG: Preconditioned conjugate gradient solver for symmetric lduMatrices using a run-time selectable preconditiioner
- smoothSolver: Iterative solver using smoother for symmetric and asymmetric matrices which uses a run-time selected smoother

Examples of Iterative Solvers

In OpenFOAM® , the preconditioners are in

`src/OpenFOAM/matrices/lduMatrix/preconditioners`

- diagonalPreconditioner: Diagonal preconditioner for both symmetric and asymmetric matrices.
- DICPreconditioner: Simplified diagonal-based incomplete Cholesky preconditioner for symmetric matrices (symmetric equivalent of DILU).
- DILUPreconditioner: Simplified diagonal-based incomplete LU preconditioner for asymmetric matrices.
- FDICPreconditioner: Faster version of the DICPreconditioner diagonal-based incomplete Cholesky preconditioner for symmetric matrices (symmetric equivalent of DILU)
- GAMGPreconditioner: Geometric agglomerated algebraic multigrid preconditioner
- noPreconditioner: Null preconditioner for both symmetric and asymmetric matrices.

Examples of Iterative Solvers

In OpenFOAM® , the smoothers are in

`src/OpenFOAM/matrices/lduMatrix/smoothers`

- DIC: diagonal-based incomplete Cholesky smoother for symmetric matrices.
- DICGaussSeidel: Combined DIC/GaussSeidel smoother for symmetric matrices
- DILU: diagonal-based incomplete LU smoother for asymmetric matrices.
- DILUGaussSeidel: Combined DILU/GaussSeidel smoother for asymmetric matrices
- GaussSeidel

Examples of Iterative Solvers

In OpenFOAM® , the specification of linear solver is in file `fvSolution`:

```
solvers
{
    p PCG
    {
        preconditioner DIC;
        tolerance 1e-06;
        relTol 0;
    };

    U PBiCG
    {
        preconditioner DILU;
        tolerance 1e-05;
        relTol 0;
     };
}
```

Examples of Iterative Solvers

Demonstration of the effects of different choices of linear solvers.

- Pressure linear solver PCG with different preconditioners
- Different pressure linear solvers