Scientific Computing: Symmetric, Overdetermined and Sparse Linear Systems

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

[1]Course MATH-GA.2043 or CSCI-GA.2112, Spring 2012

February 16th, 2011

Outline

1. Symmetric Positive-Definite Matrices
2. Overdetermined Linear Systems
3. Sparse Matrices
4. Conclusions

Numerical Solution of Linear Systems

$$\mathbf{Ax} = \mathbf{b}$$

- The most appropriate algorithm really depends on the properties of the matrix **A**:
  - General **dense matrices**, where the entries in **A** are mostly non-zero and nothing special is known: Use LU factorization.
  - **Symmetric** ($a_{ij} = a_{ji}$) and also **positive-definite** matrices.
  - General **sparse matrices**, where only a small fraction of $a_{ij} \neq 0$.
  - Special **structured sparse matrices**, arising from specific physical properties of the underlying system.
- It is also important to consider **how many times** a linear system with the same or related matrix or right hand side needs to be solved.

Positive-Definite Matrices

- A real symmetric matrix $\mathbf{A}$ is positive definite iff (if and only if):
  1. All of its eigenvalues are real (follows from symmetry) and positive.
  2. $\forall x \neq \mathbf{0}$, $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$, i.e., the quadratic form defined by the matrix $\mathbf{A}$ is convex.
  3. There exists a *unique* lower triangular $\mathbf{L}$, $L_{ii} > 0$,

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T,$$

     termed the **Cholesky factorization** of $\mathbf{A}$ (symmetric LU factorization).

1. For Hermitian complex matrices just replace transposes with adjoints (conjugate transpose), e.g., $\mathbf{A}^T \rightarrow \mathbf{A}^\star$ (or $\mathbf{A}^H$ in the book).

Cholesky Factorization

- The MATLAB built in function

$$R = chol(A)$$

  gives the Cholesky factorization and is a good way to **test for positive-definiteness**.
- The cost of a Cholesky factorization is about half the cost of LU factorization, $n^3/3$ FLOPS.
- Solving linear systems is as for LU factorization, replacing $\mathbf{U}$ with $\mathbf{L}^T$.
- For Hermitian/symmetric matrices with positive diagonals MATLAB tries a Cholesky factorization first, *before* resorting to LU factorization with pivoting.

Non-Square Matrices

- In the case of **over-determined** (more equations than unknowns) or **under-determined** (more unknowns than equations), the solution to linear systems in general becomes **non-unique**.
- One must first define what is meant by a solution, and the common definition is to use a **least-squares formulation**:

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{A}\mathbf{x} - \mathbf{b}\| = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \Phi(\mathbf{x})$$

  where the choice of the L_2 norm leads to:

$$\Phi(\mathbf{x}) = (\mathbf{A}\mathbf{x} - \mathbf{b})^T (\mathbf{A}\mathbf{x} - \mathbf{b}) .$$

- Over-determined systems, $m > n$, can be thought of as **fitting a linear model (linear regression)**:
  The unknowns $\mathbf{x}$ are the coefficients in the fit, the input data is in $\mathbf{A}$ (one column per measurement), and the output data (observables) are in $\mathbf{b}$.

Normal Equations

- It can be shown that the least-squares solution satisfies:

$$\nabla \Phi(\mathbf{x}) = \mathbf{A}^T \left[2\left(\mathbf{A}\mathbf{x} - \mathbf{b}\right)\right] = \mathbf{0} \text{ (critical point)}$$

- This gives the square linear system of **normal equations**

$$\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{x}^\star = \mathbf{A}^T\mathbf{b}.$$

- If $\mathbf{A}$ is of full rank, $\text{rank}\left(\mathbf{A}\right) = n$, it can be shown that $\mathbf{A}^T\mathbf{A}$ is positive definite, and Cholesky factorization can be used to solve the normal equations.
- Multiplying $\mathbf{A}^T$ $(n \times m)$ and $\mathbf{A}$ $(m \times n)$ takes n^2 dot-products of length m, so $O(mn^2)$ operations

Problems with the normal equations

$$\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{x}^\star = \mathbf{A}^T\mathbf{b}.$$

- The conditioning number of the normal equations is

$$\kappa\left(\mathbf{A}^T\mathbf{A}\right) = \left[\kappa(\mathbf{A})\right]^2$$

- Furthermore, roundoff can cause $\mathbf{A}^T\mathbf{A}$ to no longer appear as positive-definite and the Cholesky factorization will fail.
- If the normal equations are ill-conditioned, another approach is needed.

The QR factorization

- For nonsquare or ill-conditioned matrices of **full-rank** $r = n \leq m$, the LU factorization can be replaced by the QR factorization:

$$\mathbf{A} = \mathbf{QR}$$
$$[m \times n] = [m \times n][n \times n]$$

  where $\mathbf{Q}$ has **orthogonal columns**, $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_n$, and $\mathbf{R}$ is a **non-singular upper triangular** matrix.
- Observe that orthogonal / unitary matrices are **well-conditioned** ($\kappa_2 = 1$), so the QR factorization is numerically better (but also more expensive!) than the LU factorization.
- For matrices **not of full rank** there are modified QR factorizations but **the SVD decomposition is better** (next class).
- In MATLAB, the QR factorization can be computed using *qr* (with column pivoting).

Solving Linear Systems via QR factorization

$$\left(\mathbf{A}^T\mathbf{A}\right)\mathbf{x}^\star = \mathbf{A}^T\mathbf{b} \text{ where } \mathbf{A} = \mathbf{QR}$$

- Observe that $\mathbf{R}$ is the Cholesky factor of the matrix in the normal equations:

$$\mathbf{A}^T\mathbf{A} = \mathbf{R}^T\left(\mathbf{Q}^T\mathbf{Q}\right)\mathbf{R} = \mathbf{R}^T\mathbf{R}$$

$$\left(\mathbf{R}^T\mathbf{R}\right)\mathbf{x}^\star = \left(\mathbf{R}^T\mathbf{Q}^T\right)\mathbf{b} \quad \Rightarrow \quad \mathbf{x}^\star = \mathbf{R}^{-1}\left(\mathbf{Q}^T\mathbf{b}\right)$$

  which amounts to solving a triangular system with matrix $\mathbf{R}$.
- This calculation turns out to be much **more numerically stable** against roundoff than forming the normal equations (and has similar cost).

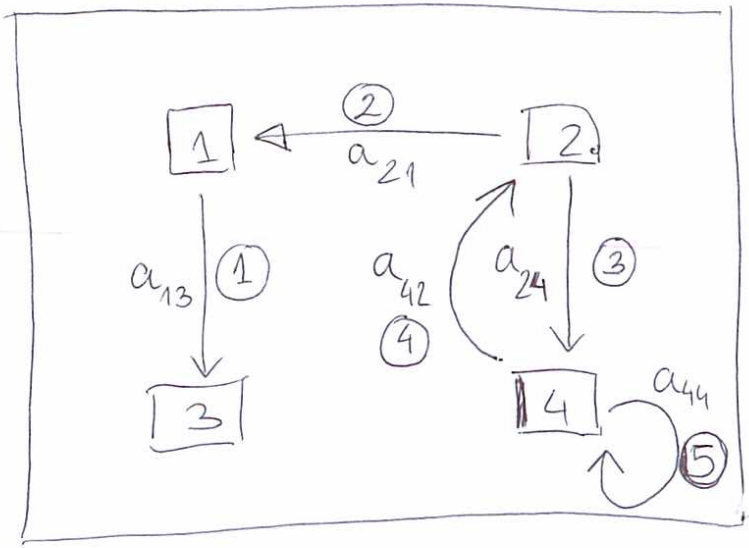
Sparse Matrices

- A matrix where a substantial fraction of the entries are zero is called a **sparse matrix**. The difference with **dense matrices** is that only the nonzero entries are stored in computer memory.
- Exploiting sparsity is important for **large matrices** (what is large depends on the computer).
- The structure of a sparse matrix refers to the set of indices i, j such that $a_{ij} > 0$, and is visualized in MATLAB using *spy*.
- The structure of sparse matrices comes from the nature of the problem, e.g., in an inter-city road transportation problem it corresponds to the pairs of cities connected by a road.
- In fact, just counting the number of nonzero elements is not enough: the **sparsity structure** is the most important property that determines the best method.

Sparse Matrices

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 5 \end{bmatrix}$$

Sparse matrix



DIRECTED Graph representation:

→ NODES (VERTICES) are variables or equations

→ ARCS (EDGES) are the non-zeros

UNDIRECTED GRAPH FOR SYMMETRIC MATRICES

Sparse matrices in MATLAB

```
>> A = sparse( [1 2 2 4 4], [3 1 4 2 3], 1:5 )
A =
   (2,1)        2
   (4,2)        4
   (1,3)        1
   (4,3)        5
   (2,4)        3
>> nnz(A) % Number of non-zeros
ans =     5
>> whos A
  A          4x4                   120  double     sparse

>> A = sparse([],[],[],4,4,5); % Pre-allocate memory
>> A(2,1)=2; A(4,2)=4; A(1,3)=1; A(4,3)=5; A(2,4)=3;
```

Fill-In

- There are general techniques for dealing with sparse matrices such as **sparse LU factorization**. How well they work depends on the structure of the matrix.
- When factorizing sparse matrices, the factors, e.g., **L** and **U**, can be much less sparse than **A**: **fill-in**.
- Pivoting (**reordering** of variables and equations) has a dual, sometimes conflicting goal:
  1. Reduce fill-in, i.e., **improve memory use**.
  2. Reduce roundoff error, i.e., **improve stability**. Typically some **threshold pivoting** is used only when needed.
- For many sparse matrices there is a large fill-in and **iterative methods** are required.

Why iterative methods?

- Direct solvers are great for dense matrices and are implemented very well on modern machines.
- **Fill-in** is a major problem for certain sparse matrices and leads to extreme memory requirements.
- Some matrices appearing in practice are **too large** to even be represented explicitly (e.g., the Google matrix).
- Often linear systems only need to be **solved approximately**, for example, the linear system itself may be a linear approximation to a nonlinear problem.
- Direct solvers are much harder to implement and use on (massively) **parallel computers**.

Stationary Linear Iterative Methods

- In iterative methods the core computation is **iterative matrix-vector multiplication** starting from an **initial guess** $\mathbf{x}^{(0)}$.
- Prototype is the **linear recursion**:

$$\mathbf{x}^{(k+1)} = \mathbf{B}\mathbf{x}^{(k)} + \mathbf{f},$$

  where $\mathbf{B}$ is an **iteration matrix** somehow related to $\mathbf{A}$ (many different choices/algorithms exist).
- For this method to be **consistent**, we must have that the actual solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ is a **stationary point** of the iteration:

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{f} \quad \Rightarrow \quad \mathbf{A}^{-1}\mathbf{b} = \mathbf{B}\mathbf{A}^{-1}\mathbf{b} + \mathbf{f}$$

$$\mathbf{f} = \mathbf{A}^{-1}\mathbf{b} - \mathbf{B}\mathbf{A}^{-1}\mathbf{b} = (\mathbf{I} - \mathbf{B})\,\mathbf{x}$$

- For example, this fixed-point iteration is an iterative method:

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \mathbf{A})\,\mathbf{x}^{(k)} + \mathbf{b}$$

Termination

- The iterations of an iterative method can be terminated when:
  1. The **residual** becomes small,

$$\left\|\mathbf{r}^{(k)}\right\| = \left\|\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}\right\| \leq \varepsilon \|\mathbf{b}\|$$

     This is good for well-conditioned systems.
  2. The solution $\mathbf{x}^{(k)}$ stops changing, i.e., the **increment** becomes small,

$$[1 - \rho(\mathbf{B})] \left\|\mathbf{e}^{(k)}\right\| \leq \left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\| \leq \varepsilon \|\mathbf{b}\|,$$

     which can be shown to be good if convergence is rapid.
- Usually a careful **combination** of the two strategies is employed along with some **safeguards**.

Special Matrices in MATLAB

- MATLAB recognizes (i.e., tests for) some special matrices automatically: banded, permuted lower/upper triangular, symmetric, Hessenberg, but **not** sparse.
- In MATLAB one may specify a matrix **B** instead of a single right-hand side vector **b**.
- The MATLAB function

$$X = linsolve(A, B, opts)$$

  allows one to specify certain properties that speed up the solution (triangular, upper Hessenberg, symmetric, positive definite, none), and also estimates the condition number along the way.
- Use *linsolve* instead of backslash if you know (for sure!) something about your matrix.

Conclusions/Summary

- For symmetric positive definite matrices the Cholesky factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ is preferred and does not require pivoting.
- The QR factorization is a numerically-stable method for solving **full-rank non-square systems**.
- **Sparse matrices** deserve special treatment but the details depend on the specific field of application.
- In particular, special sparse **matrix reordering** methods or iterative systems are often required.
- When **sparse direct methods** fail due to memory or other requirements, **iterative methods** are used instead.