Lecture 3: Systems of Linear Equations

CSC 338: Numerical Methods

Ray Wu

University of Toronto

January 25, 2023

Systems of Linear Equations

This lecture is on solving

$$Ax = b \tag{1}$$

Unless otherwise stated,

- A is an n-by-n matrix,
- x, b are vectors of size n.
- A, b given, x unknown.

Topics:

- Diagonal systems
- Triangular systems
- Gaussian Elimination and LU decomposition
- Cholesky Decomposition
- Sparse Matrices
- Error and condition number

Introduction and notation

Problem in matrix form:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{2}$$

Diagonal systems

The simplest of all matrix equations; n independent linear equations.

$$\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 \\ 0 & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3}$$

Solution:

$$\forall i, x_i = \frac{b_i}{a_{i,i}} \tag{4}$$

(Upper) Triangular systems

Triangular systems are only slightly more complicated than diagonal systems.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \cdots & 0 & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{5}$$

Solution: start from last equation

$$x_n = \frac{b_n}{a_{n,n}} \tag{6}$$

Iteratively update from $n-1$ to 1 in reverse order (backward solve):

$$x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{k,j} x_j}{a_{k,k}} \tag{7}$$

(Lower) Triangular systems

Lower Triangular Systems are similar to Upper triangular systems

$$\begin{bmatrix} a_{1,1} & 0 & \cdots & 0 & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & 0 \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{8}$$

Solution: start from first equation

$$x_1 = \frac{b_1}{a_{1,1}} \tag{9}$$

Iteratively update from 2 to n in forward order (forward solve):

$$x_k = \frac{b_k - \sum_{j=1}^{k-1} a_{k,j} x_j}{a_{k,k}} \tag{10}$$

Gaussian Elimination

Finally, we consider general matrices and have Gaussian Elimination

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{11}$$

Solution: transform to upper triangular system. First step, for $i > 1$:

$$a'_{i,j} = a_{i,j} - (a_{i,1}/a_{1,1})a_{1,j} \tag{12}$$

and

$$b'_i = b_i - (a_{i,1}/a_{1,1})b_i \tag{13}$$

if $j = 1$, $a_{i,1}$ becomes assigned to zero.

We get the system of equations on the following page:

Gaussian Elimination (II)

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a'_{2,2} & \cdots & a'_{2,n-1} & a'_{2,n} \\ 0 & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n-1,2} & \cdots & a'_{n-1,n-1} & a'_{n-1,n} \\ 0 & a'_{n,2} & \cdots & a'_{n,n-1} & a'_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ \vdots \\ b'_{n-1} \\ b'_n \end{bmatrix} \tag{14}$$

Repeat the process for the system of equations with $a'_{i,j}$ and b'_i (ignore the blue components)

- Review from Linear Algebra
- We focus on how to use computers to carry out this algorithm.
- We will discuss some *numerical* issues that arise from this algorithm.
- We will also generalize it to make it more stable.

LU decomposition

Recall Gaussian Elimination where we transform the following system to a triangular system:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{15}$$

- LU decomposition expresses the above procedure as a matrix.
- Hence, it factors the matrix A into an lower triangular matrix L and an upper triangular matrix U such that $A = LU$.

LU decomposition (II)

Recall Gaussian Elimination:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n-1} & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \tag{16}$$

Let $l_{i,1} = (a_{i,1}/a_{1,1})$. Then,

$$a'_{i,j} = a_{i,j} - l_{i,1} a_{1,j} \tag{17}$$

This is a linear combination of matrix entries, hence

LU decomposition (III)

Let M_1 be defined as

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ -l_{2,1} & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -l_{n-1,1} & 0 & \cdots & 1 & 0 \\ -l_{n,1} & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{18}$$

Then, $M_1 A$ produces

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a'_{2,2} & \cdots & a'_{2,n-1} & a'_{2,n} \\ 0 & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n-1,2} & \cdots & a'_{n-1,n-1} & a'_{n-1,n} \\ 0 & a'_{n,2} & \cdots & a'_{n,n-1} & a'_{n,n} \end{bmatrix} \tag{19}$$

LU decomposition (IV)

Repeat the procedure from the previous slide for M_2, M_3, ..., M_{n-1}. Then,

$$M_{n-1}M_{n-2}\cdots M_2M_1A = U \tag{20}$$

where U is upper triangular.

Inverting each M_i matrix gives us

$$A = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1}U \tag{21}$$

Last question/obstacle: What is M_i^{-1}?

LU decomposition (V)

What is M_i^{-1}? Recall that

$$M_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -l_{n,1} & 0 & \cdots & 1 \end{bmatrix} \tag{22}$$

To invert, find a matrix which cancels out the terms in the lower diagonal.

$$M_1^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & 0 & \cdots & 1 \end{bmatrix} \tag{23}$$

You can verify this fact for yourself.

LU decomposition (VI)

Product of $L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}$

$$M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n,1} & l_{n,2} & \cdots & 1 \end{bmatrix} \tag{24}$$

Hence, the Gaussian elimination process produces both L and U, where $A = LU$. Then, given $Ax = b$ our algorithm is as follows:

1. Compute $A = LU$.
2. Solve $Ly = b$, for y
3. Solve $Ux = y$.

What is the purpose of LU when we already have Gaussian Elimination?

- Multiple b's: Can compute $A = LU$ once, then use the more efficient forward/backwards solves on each b vector.

Partial Pivoting

So far, we have assumed that $a_{i,i}$ are nonzero. What if we have $a_{i,i} = 0$?

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{25}$$

Clearly, $x_1 = b_2$, and $x_2 = b_1$. But LU will break down in the first step.

Solution: interchange (permute) rows 1 and 2.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_2 \\ b_1 \end{bmatrix} \tag{26}$$

Interchange rows, but not columns – partial pivoting.

Partial Pivoting (II)

Key idea of partial pivoting: At each stage, interchange the rows to get the largest pivot:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n-1} & a_{1,n} \\ 0 & a'_{2,2} & \cdots & a'_{2,n-1} & a'_{2,n} \\ 0 & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n-1,2} & \cdots & a'_{n-1,n-1} & a'_{n-1,n} \\ 0 & a'_{n,2} & \cdots & a'_{n,n-1} & a'_{n,n} \end{bmatrix} \tag{27}$$

Here, we would pick the largest pivot from $a'_{i,2} \quad (2 \le i \le n)$ in absolute value, and interchange the rows.

- Recall that $L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}$ in GE without PP, in other words, $L^{-1} = M_{n-1} M_{n-2} \cdots M_1$
- In GE with PP, $B = M_{n-1} P_{n-1} \dots M_2 P_2 M_1 P_1$
- Next, we need to show that $B = L^{-1} P$.

Partial Pivoting (III)

- Recall that $B = M_{n-1}P_{n-1} \dots M_2P_2M_1P_1$.
- Define
  - $\tilde{M}_{n-1} = M_{n-1}$,
  - $\tilde{M}_{n-2} = P_{n-1}M_{n-2}P_{n-1}^T$,
  - $\tilde{M}_{n-3} = P_{n-1}P_{n-2}M_{n-1}P_{n-2}^TP_{n-1}^T$, etc.
- Then, we have $B = \tilde{M}_{n-1}\tilde{M}_{n-2} \dots \tilde{M}_1P_{n-1}P_{n-2} \dots P_1$.
- $\tilde{M}_i$ remain lower-triangular, because only diagonal entries and zero entries get permuted.
- Hence, we have shown that B can be written as $L^{-1}P$.

Partial Pivoting (IV)

- Since we have shown that $B = L^{-1}P$, then we have $BA = U$ or equivalently $PA = LU$.
- In Matlab, use `[P, L, U] = lu(A)`
  - There are other options; such as no pivoting, complete pivoting, etc.
- We are done with general linear equations, just a few more notes:
- We cannot demonstrate stability for partial pivoting or scaled partial pivoting, only with complete pivoting.
- Complete pivoting will find the largest entry among the entire submatrix rather than just the entries of the column in question, and will interchange a pair of both rows and columns.
- However, in practice, most matrices are stable with just partial pivoting.
- Additionally, some matrices that arise in practice require no pivoting.

Cholesky Decomposition

- So far, we have looked at general matrices, where no special structure exists.
- In practice, many matrices are symmetric positive definite (SPD). The Cholesky decomposition factorizes SPD matrices into

$$A = RR^T \tag{28}$$

  where R is triangular.
  - Note that if $n = 1$, this becomes a scalar square root.
- Symmetric positive definiteness is the matrix analogue of a positive real number.
- Cholesky decomposition finds the "square root".

Review: Symmetric positive definite matrices

A matrix A is symmetric positive definite (SPD) if

- $A = A^T$ (symmetry)
- for all nonzero vectors x, $x^T Ax > 0$. (positive definite)

Where do SPD matrices arise? In this course:

- least squares
- convex optimization

Beyond this course:

- numerical methods for partial differential equations (PDEs)
  - Finite Difference Methods
  - Finite Element Methods

  Many applications, such as structural engineering, computer graphics, finance, etc.

Cholesky Decomposition

- We can consider the Cholesky decomposition as a symmetric LU decomposition. Let
$$A = LU \tag{29}$$
where A is symmetric positive definite.
- We can factor out the diagonal to get
$$A = LDU \tag{30}$$
- Due to symmetry, $U = L^T$ and hence
$$A = LDL^T \tag{31}$$
- Let $R = D^{1/2}L^T$, then, $A = R^T R$.
- $D^{1/2}$ is elementwise square root.
- **NO** permutations required!

Cholesky Decomposition example

- Consider a 2x2 matrix

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} r_{11} & 0 \\ r_{21} & r_{22} \end{bmatrix} \begin{bmatrix} r_{11} & r_{21} \\ 0 & r_{22} \end{bmatrix} \tag{32}$$

  - From the a_{11} entry we get $a_{11} = r_{11}^2$, hence $r_{11} = \sqrt{a_{11}}$.
  - From the a_{12} entry we get $a_{12} = r_{11} r_{21}$, hence $r_{21} = a_{12}/r_{11}$, with r_{11} already known.
- Full algorithm on p. 116 of Ascher & Greif.
- in Matlab, use `R = chol(A)` to calculate the Cholesky factor.

Sparse matrices and fill-in

- Many matrices that arise from practical applications are sparse
  - You will see one in assignment 2.
- Large size, but comparatively few nonzero entries. For example, a tridiagonal matrix has n^2 entries but $3n - 2$ nonzero entries.
- Sparse matrices are useful because they can be solved more efficiently than non-sparse matrices (e.g. tridiagonal solvers, banded LU, etc).
- There are many reasons to not invert a matrix (conditioning, time, etc), but one of the main reasons is that inverting a sparse matrix can destroy the sparsity structure.

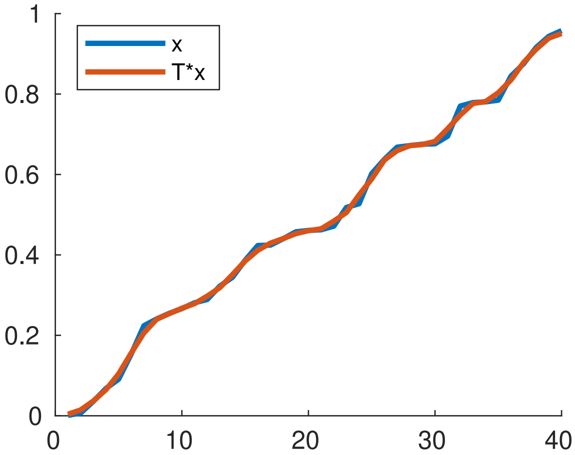
Sparse Matrix Example

- Given a vector x, compute a new vector with the entries being the average of itself and neighbouring entries.
- This can be represented as a linear transformation, hence, a matrix T.
- The computation can be represented by a sparse matrix:

$$T = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & \dots & 0 \\ 1/3 & 1/3 & 1/3 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 1/3 & 1/3 & 1/3 \\ 0 & \dots & 0 & 0 & 1/2 & 1/2 \end{bmatrix} \tag{33}$$

Sparse Matrix Example continued

- Plot of x and Tx:



- Matlab script on the website.

Condition number and error propagation

- Consider again solving
$$Ax = b \tag{34}$$
- Denote exact solution as x, computed solution as $\hat{x}$. We want to estimate
$$\frac{\|x - \hat{x}\|}{\|x\|} \tag{35}$$
- Condition number: error amplification.

Residual

- Let $\hat{x}$ denote the computed solution. Then, the **residual** is defined as

$$\hat{r} = b - A\hat{x}. \tag{36}$$

- Additionally, note that $b = Ax$, so

$$\hat{r} = Ax - A\hat{x} = A(x - \hat{x}) \tag{37}$$

- Rearrange to get

$$x - \hat{x} = A^{-1}\hat{r}. \tag{38}$$

Review: Vector and matrix norms

Norms: length of a vector. Three requirements:

- **Nonnegativity**:
$$\|x\| \geq 0, \|x\| = 0 \text{ iff } x = \vec{0} \tag{39}$$
- **Absolute Homogeniety**:
$$\|\alpha x\| = |\alpha|\|x\| \text{ for } \alpha \in \mathbb{R} \tag{40}$$
- **Triangle inequality**:
$$\|x + y\| \leq \|x\| + \|y\|. \tag{41}$$

p (or L_p) norm

Let $p \geq 1$ be a real number. Then the p-norm for vectors is defined as

$$\|x\|_p = \left(\sum |x_i|^p\right)^{1/p} \tag{42}$$

Common values of p:

- $p = 2$: Euclidian norm (conventional "distance" norm).
- $p = 1$: Manhatten norm (taxicab norm)
- $p = \infty$: max-norm.
- $p = 0$: "hamming distance" (number of differences between vector entries)
  - Not really a norm.
  - Used in applications of machine learning, statistics, etc.

Matrix norms

Two types of matrix norms you should know:

- Element norms: norms based on the entries of the individual elements. Example: Forbenius norm:

$$\|A\|_F = \sqrt{\sum_{i,j=1}^{n} a_{i,j}^2} \tag{43}$$

  treats the matrix as a vector stored in a different arrangement.
- Induced (operator) norms: norms based on viewing the matrix as an operation to a vector.
- Instead of viewing A as a matrix, view A as a function/operator, where Ax is the output and x is the input.

Operator norms

Since we are only considering the input x and the output Ax, we define the operator norm as the "maximum stretching factor" between x and Ax.

$$\|A\| = \max \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\| \tag{44}$$

- Which vector norm for Ax, which vector norm for x not defined.
- Can pick any true vector norm.

Submultiplicativity:

$$\|Ax\| \leq \|A\|\|x\| \tag{45}$$

Condition number (II)

- Recall that

$$x - \hat{x} = A^{-1}\hat{r}. \tag{46}$$

and

$$Ax = b \tag{47}$$

- Use matrix norm bounds to get

$$\|x - \hat{x}\| \leq \|A^{-1}\|\|\hat{r}\| \tag{48}$$

and

$$\|A\|\|x\| \geq \|b\| \tag{49}$$

- Multiply and rearrange to get

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\hat{r}\|}{\|b\|} \tag{50}$$

- $\|A\|\|A^{-1}\|$ is the condition number, denoted by $\kappa(A)$.

Condition number (III)

- Hence,

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \kappa(A) \frac{\|r\|}{\|b\|} \tag{51}$$

- The condition number is a measurement of the magnification of relative error in residual to the relative error in solution.
- The exact condition number can be difficult to obtain, and rarely matters. The order of magnitude is what's important.
- Condition number can be computed with `cond` for dense matrices or estimated with `condest` for sparse matrices in Matlab.

Summary of Matlab commands in this lecture

name	description	usage
`lu`	compute LU decomposition	`[P, L, U] = lu(A)`
`chol`	compute Cholesky factor	`R = chol(A)`
`\`	solve a system of linear equations	`x = A\b`
`spdiags`	create a sparse matrix	`A = spdiags([e, e, e],` `-1:1, n, n)`
`speye`	sparse identity matrix	`I = speye(n,n)`
`norm`	compute vector norm	`l = norm(x, 2)`
`condest`	estimate condition number	`kappa = condest(A)`