NE 155, Topic 11, S21
Direct Solutions for Linear Systems
March 11, 2021

Direct Solutions of Linear Systems

Given $\mathbf{A}\vec{x} = \vec{b}$, how do you find $\vec{x}$?

- compute (explicitly or implicitly) $\mathbf{A}^{-1} \rightarrow \vec{x} = \mathbf{A}^{-1}\vec{b}$
- typically requires many operations (FLOPs), large storage (even if $\mathbf{A}$ is sparse, $\mathbf{A}^{-1}$ may not be)
- seldom parallelizes well
- *not* used in practice
- incomplete factorizations (which we'll cover) *are* used as pre-conditioners for iterative methods

We can also use iterative and semi-iterative methods, which we will talk about in a later class.

We can reduce a matrix to echelon form (discussed later) to determine uniqueness properties of the solution space.

1. A system has **a unique solution** when there is one unknown for each equation when expressed in echelon form.
2. A system **does not have a solution** when the determinant of its matrix is zero but reduction to echelon form does not yield any free variables.
3. A system has **infinitely many** solutions when the determinant of its matrix is zero and reduction to echelon form does yield a free variable.

Diagonal Systems

$$\begin{pmatrix} a_{11} & 0 & \cdots & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{ii} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \\ \vdots \\ b_n \end{pmatrix}$$

To solve this, simply

$$x_i = \frac{b_i}{a_{ii}}, i = 1, \ldots, n \, .$$

- this requires $O(n)$ operations.
- if $a_{ii} = 0$, then $\det(\mathbf{A}) = 0$ (the determinant of a diagonal matrix is the product of its diagonal entries) and the system does not have a solution.
- If, however, $b_i = 0$ for the a_{ii} is question, then the system has an infinite number of solutions.

Lower/Upper Triangular Systems

$$\begin{pmatrix} a_{11} & 0 & \cdots & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ii} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{ni} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_j \\ \vdots \\ b_n \end{pmatrix}$$

To solve this, solve the first equation for the one unknown, which becomes know. Then the second equation for the (now) one unknown, etc. This is called forward substitution.

$$x_1 = \frac{b_1}{a_{11}}$$

for $i = 2, n$

$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j)$$

Upper triangular systems work the same way, but you go in the reverse order $\rightarrow$ backward substi-

tution.

$$x_n = \frac{b_n}{a_{nn}}$$
$$\text{for } i = n-1, -1, 1$$
$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=i+1}^{n} a_{ij}x_j)$$

These both require $O(n^2)$ operations, and each have the same caveats about zeros as the diagonal system.

LU Decomposition

Theorem

Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be regular (recall: regular means $\mathbf{A}^{-1}$ exists). Then there are matrices $\mathbf{L}$ (lower triangular) with $l_{ii} = 1$ and $\mathbf{U}$ (upper triangular) such that $\mathbf{A} = \mathbf{LU}$ is unique.

We want to solve $\mathbf{A}\vec{x} = \vec{b}$:

$$\mathbf{A}\vec{x} = \mathbf{LU}\vec{x} = \vec{b}$$
$$\text{define } \mathbf{U}\vec{x} = \vec{y}$$
$$\text{then note } \mathbf{L}\vec{y} = \vec{b}$$

1. Solve $\mathbf{L}\vec{y} = \vec{b}$ for $\vec{y}$ using forward substitution
2. Solve $\mathbf{U}\vec{x} = \vec{y}$ for $\vec{x}$ using backward substitution

This requires $O(n^2)$ operations.

The **LU decomposition** is formed by Gaussian Elimination, requiring $O(n^3)$ operations. $\mathbf{U}$ is the upper triangular matrix from Gaussian Elimination; $\mathbf{L}$ is the lower triangular matrix formed by recording the operations you perform on $\mathbf{U}$ (explained below). To check, if you perform the

operations on $\mathbf{L}$ that you perform on $\mathbf{U}$, you should get $\mathbf{I}$.

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{n,n-1} & 1 \end{pmatrix} \qquad \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix}$$

For a given column, $j = 1, \ldots, n$, the m_{ij} (which as you'll see below we can think of as $m$ultipliers) are a^*_{ij}/a^*_{jj} for $i = j, \ldots, n$.

Here the a^* are the the entries in the version of $\mathbf{A}$ *that comes from the process of row-reducing* $\mathbf{A}$ to get $\mathbf{U}$. This will be made clearer through an example:

example

$$\mathbf{A} = \begin{pmatrix} 5 & 2 & 1 \\ 4 & 1 & -1 \\ -2 & 3 & -3 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 3 \\ -3 \\ 5 \end{pmatrix}$$

Create $\mathbf{U}$ by row-reducing $\mathbf{A}$ to echelon form (remember to operate on $\mathbf{b}$ as well).

$$\begin{aligned} \mathbf{U} &= \begin{pmatrix} 5 & 2 & 1 \\ 4 & 1 & -1 \\ -2 & 3 & -3 \end{pmatrix} & \begin{pmatrix} \\ R_2 - m_{21}R_1, & m_{21} = 4/5\,(a_{21}/a_{11}) \\ R_3 - m_{31}R_1, & m_{31} = -2/5\,(a_{31}/a_{11}) \end{pmatrix} \\ &= \begin{pmatrix} 5 & 2 & 1 \\ 0 & -3/5 & -9/5 \\ 0 & 19/5 & -13/5 \end{pmatrix} & \begin{pmatrix} \\ \\ R_3 - m_{32}R_2, & m_{32} = -19/3\,(a^*_{32}/a^*_{22}) \end{pmatrix} \\ &= \begin{pmatrix} 5 & 2 & 1 \\ 0 & -3/5 & -9/5 \\ 0 & 0 & -14 \end{pmatrix} \end{aligned} \tag{1}$$

Now, we fill the m_{ij}s we just discovered into the $\mathbf{L}$ matrix.

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 4/5 & 1 & 0 \\ -2/5 & -19/3 & 1 \end{pmatrix}$$

Using all of this, we can finally solve our equation. Begin with $\mathbf{L}\vec{y} = \vec{b}$ for $\vec{y}$ using forward

substitution:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 4/5 & 1 & 0 \\ -2/5 & -19/3 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 3 \\ -3 \\ 5 \end{pmatrix}$$

$$\boxed{y_1 = 3}$$
$$(\frac{4}{5})(3) + y_2 = -3 \rightarrow \boxed{y_2 = -\frac{27}{5}}$$
$$(-\frac{2}{5})(3) + (-\frac{19}{3})(-\frac{27}{5}) + y_3 = 5 \rightarrow \boxed{y_3 = -28}$$

Now, we solve $\mathbf{U}\vec{x} = \vec{y}$ for $\vec{x}$

$$\mathbf{U} = \begin{pmatrix} 5 & 2 & 1 \\ 0 & -3/5 & -9/5 \\ 0 & 0 & -14 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ -\frac{27}{5} \\ -28 \end{pmatrix}$$

$$-14x_3 = -28 \rightarrow \boxed{x_3 = 2}$$
$$(-\frac{3}{5})x_2 + (-\frac{9}{5})(2) = -\frac{27}{5} \rightarrow \boxed{x_2 = 3}$$
$$(5)x_1 + (2)(3) + (1)(2) = 3 \rightarrow \boxed{x_1 = -1}$$

- The LU decomposition and solution are exact
- given the LU decomposition, multiple solution vectors can be computed for a small cost
- the LU decomposition can overwrite the original matrix, if desired
- the LU decomposition requires $O(n^3)$ FLOPS
- the LU decomposition 'fills' a sparse matrix

Example of Real Use

"Improving Thermal-Hydraulic Calculation Modules of THERMIX Code Based on LU Decomposition" by Yang Tang, Yangping Zhou, Zhiwei Zhou; DOI: 10.1115/ICONE21-16575; Conference: 2013 21st International Conference on Nuclear Engineering

ABSTRACT: THERMIX is a software package for analyzing the thermal-hydraulic and safety behavior of pebble-bed high temperature gas-cooled reactor under both normal condition and accident conditions. The point-wise iterative solution method of THERMIX is time-consuming and difficult to be extended. For the solid-phase thermal conductivity calculation module (THERMIX), gas flow calculation module (KONVEK) and primary loop flow network system calculation module (KISMET) respectively, a global solution method based on LU decomposition with higher efficiency is developed and tested. With good calculation accuracy, the new method has a greater computational efficiency, compared with the original method.

Pivoting

When any of the diagonal coefficients are equal to zero (or close to zero), one way of avoiding it is row or column pivoting (exchange of rows or columns). We keep track of pivoting through a permutation matrix:

$$\mathbf{PA} = \mathbf{LU}$$

where $\mathbf{P}$ has one entry of unity per column and row.

For example, if we have a situation where u_{22} is very small but u_{32} isn't, we could flip rows 2 and 3. Note: this also requires flipping b_2 and b_3. In this case,

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

** Note: the example we did above would have been easier if we had pivoted the second and third rows. If you use the scipy LU decomposition, it will do this (`scipy.linalg.lu`) **

You can find another example both with and without pivoting here: `https://www.seas.ucla.edu/~vandenbe/103/lectures/lu.pdf`.

Direct Factorization Algorithm

Begin with $n \times n$ matrices $\mathbf{U}$ and $\mathbf{L}$ filled with zeros.

1. For $i = 1, \ldots, n$: $l_{ii} = 1$

2. $u_{11} = a_{11}$

3. For $j = 2, \ldots, n$
   $u_{1j} = a_{1j}$
   $l_{j1} = a_{j1}/u_{11}$

4. For $i = 2, \ldots, n-1$
   $u_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik}u_{ki}$
   For $j = i+1, \ldots, n$
      $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{ki}$
      $l_{ji} = 1/u_{ii}[a_{ij} - \sum_{k=1}^{i-1} l_{jk}u_{ki}$

5. $u_{nn} = a_{nn} - \sum_{k=1}^{n-1} l_{nk}u_{kn}$

Tridiagonal

If **A** is tridiagonal, LU decomposition can be streamlined by using the Thomas algorithm:

$$\mathbf{T} = \begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \cdots & 0 \\ 0 & a_{32} & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & a_{n-1n} \\ 0 & 0 & \cdots & a_{nn-1} & a_{nn} \end{pmatrix} = \begin{pmatrix} d_1 & u_1 & 0 & \cdots & 0 \\ l_2 & d_2 & u_2 & \cdots & 0 \\ 0 & l_3 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & u_{n-1} \\ 0 & 0 & \cdots & l_n & d_n \end{pmatrix}$$

Thomas Algorithm

1. For $i = 2, \ldots, n$, overwrite
   $d_i = d_i - (l_i/d_{i-1})u_{i-1} \qquad b_i = b_i - (l_i/d_{i-1})b_{i-1}$ (forward sub)

2. Get $\vec{x}$ through backward sub:
   $x_n = b_n/d_n$
   For $i = n-1, \ldots, 1$:
   $x_i = (b_i - u_i x_{i+1})/d_i$