Chapter 7. Direct Methods for Solving Linear Systems & Least Squares Problems.

Background: $Ax=b$

How do we solve it accurately & efficiently?

$Ax=b$ comes from optimization, DEQ's, PDE's

Past: $\underbrace{40\times 40}_{\text{limit}}$ or greater systems can't be solved!

1943: Hotelling concluded that errors in Gaussian elimination grow exponentially w/ the size of A.

1947: Goldstine, von Neumann showed that $Ax=b$ can be solved by $A^TAx=A^Tb$ (not a good thing to do though.)

1950s: Use of computers to solve 100×100 & bigger systems.

1961: Wilkinson applied the idea of backward error analysis to $Ax=b$

§7.1 : Review of matrix multiplication.

$$\underset{m\times n}{A} = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

$$A = (a_{ij}), \quad \underbrace{A^T = (a_{ij})}_{\text{transpose of } A} \quad \text{or} \quad \underbrace{A^* = (\overline{a_{ij}})}_{\text{Hermitian transpose of } A}$$

A is symmetric if $A^T = A$ ($A^* = A$)
(MATLAB : use A')

$$Ax = \begin{pmatrix} a_{11}x_1 + \dots + a_{1n}x_n \\ \cdots\cdots \\ a_{m1}x_1 + \dots + a_{mn}x_n \end{pmatrix} \overset{\text{or}}{=} x_1 \underbrace{\begin{pmatrix} a_{11} \\ \vdots \\ a_{m1} \end{pmatrix}}_{A_1} + \dots + x_n \underbrace{\begin{pmatrix} a_{1n} \\ \vdots \\ a_{mn} \end{pmatrix}}_{A_m}$$

lin. comb. of columns

§7.2 : Gaussian elimination.

$Ax = b$, $A_{n\times n}$, supposedly, nonsingular
($\Rightarrow \exists!$ solution)

Algorithm appeared in 200 BC in China.
Gauss developed the method and applied it to data collected between 1803 & 1809 (from study of the orbit of the asteroid Pallas). He had a system of 6 linear equations in 6 unknowns.

- Consider 3×3 system:

$Ax=b$ $$\underbrace{\begin{pmatrix}1&2&3\\4&5&6\\7&8&0\end{pmatrix}}_{A}\underbrace{\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}}_{x}=\underbrace{\begin{pmatrix}1\\0\\2\end{pmatrix}}_{b}$$ or $$\begin{aligned}x_1+2x_2+3x_3&=1\\4x_1+5x_2+6x_3&=0\\7x_1+8x_2&=2\end{aligned}$$

Solve using Gaussian elimination (GE):

$$\left(\begin{array}{ccc|c}1&2&3&1\\4&5&6&0\\7&8&0&2\end{array}\right)\xrightarrow[\text{row 3 - row1}\times 7]{\text{row2 - row1}\times 4}\underbrace{\left(\begin{array}{ccc|c}1&2&3&1\\0&-3&-6&-4\\0&-6&-21&-5\end{array}\right)}_{\text{Step 1}}\xrightarrow{\text{row3 - row2}\times 2}$$

$$\underbrace{\left(\begin{array}{ccc|c}1&2&3&1\\0&-3&-6&-4\\0&0&-9&3\end{array}\right)}_{\text{Step 2}}\Rightarrow \text{upper triangular system!}$$

$$\begin{aligned}x_1+2x_2+3x_3&=1\\-3x_2-6x_3&=-4\\-9x_3&=3\end{aligned}$$

It can be solved by back substitution:

$x_3=-1/3 \Rightarrow -3x_2-6(-1/3)=-4$ gives $x_2=2$

$\Rightarrow x_1+2\cdot 2+3(-1/3)=1$ gives $x_1=-2$

- Another way to think about GE:

transform the matrix A into an upper triangular matrix U by introducing zeros below the diagonal of A, column after column:

Step 1 $$L_1A=\underbrace{\begin{pmatrix}1&0&0\\-4&1&0\\-7&0&1\end{pmatrix}}_{\text{low triangular } L_1}\underbrace{\begin{pmatrix}1&2&3\\4&5&6\\7&8&0\end{pmatrix}}_{A}=\underbrace{\begin{pmatrix}1&2&3\\0&-3&-6\\0&-6&-21\end{pmatrix}}_{L_1A}$$

Note L_1 is invertible: $L_1^{-1}=\begin{pmatrix}1&0&0\\4&1&0\\7&0&1\end{pmatrix}$

(negate the off-diagonal entries)

Step 2 $L_2(L_1A) = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}}_{\text{low triangular } L_2} \underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -21 \end{pmatrix}}_{L_1A} = \underbrace{\begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -9 \end{pmatrix}}_{U = L_2L_1A,\ \text{upper triangular}}$

$$L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix}$$

So, $U = L_2L_1A$ or $A = \underbrace{L_1^{-1}L_2^{-1}}_{L}U = \underbrace{(L_2L_1)^{-1}}_{L}U$

$$L = L_1^{-1}L_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix}}_{\text{low triangular}}$$

$Ax = b \Rightarrow (LU)x = b \Rightarrow L(\underbrace{Ux}_{y}) = b \Rightarrow$ solve for y

then solve for x: $Ux = y$.

- Consider $A_{4\times4}$:

$$\underbrace{\begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{pmatrix}}_{A} \xrightarrow{L_1} \underbrace{\begin{pmatrix} \times & \times & \times & \times \\ 0 & \Delta & \Delta & \Delta \\ 0 & \Delta & \Delta & \Delta \\ 0 & \Delta & \Delta & \Delta \end{pmatrix}}_{L_1A} \xrightarrow{L_2} \underbrace{\begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \Delta & \Delta \\ 0 & 0 & \Delta & \Delta \end{pmatrix}}_{L_2L_1A} \xrightarrow{L_3}$$

$$\underbrace{\begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \Delta \end{pmatrix}}_{L_3L_2L_1A = U} \Rightarrow A = LU \quad \left(L = (L_3L_2L_1)^{-1}\right)$$

Generally, $A_{n\times n} = \underbrace{(L_{n-1}\cdots L_1)^{-1}}_{L}U$

MATLAB to implement.

Show

MATLAB notes: $Ax = b$

$$\underbrace{x = A\backslash b \quad \text{or} \quad x = \text{mldivide}(A, b)}$$

use different solvers depending upon the structure of A

type "mldivide" to see the diagram.

$(A = PLU)$

$A = QR$ or $A = LU$

or $A = LL^T$, ...

(Cholesky)

A\b returns a result in many cases,
it can give you a warning & a solution.
→ num. sol. s.t. $\|Ax - b\| \leq$ tolerance

Also: $x = A^{-1}b$ $\quad$ ($x = \text{inv}(A)*b$) bad!

or $x = \text{pinv}(A)*b$

$A^+ = (A^*A)^{-1}A^*$ or

$A^+ = (A^TA)^{-1}A^T$

§ 7.2.1 Operation Counts.

Timing of a computer code depends on FLOPs : $+, -, \times, /$

Q: How many FLOPs are required for GE? (without pivoting)
(Just direct computing, no parallelization.)

Total # FLOPs on row i at stage j is

$$\underbrace{1}_{/} + \underbrace{2n+2}_{\times, -} = 2n+3 \Rightarrow$$

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} (2n+3) = \sum_{j=1}^{n-1} (n-j)(2n+3)$$

$$= (2n+3) \sum_{j=1}^{n-1} (n-j) = (2n+3)\left[\underbrace{n(n-1)}_{\sum_{j=1}^{n} n} - \underbrace{\frac{n(n-1)}{2}}_{\sum_{j=1}^{n} j}\right]$$

$$= (2n+3)\frac{n(n-1)}{2} = n^3 + \underbrace{\frac{n^2}{2} - \frac{3n}{2}}_{O(n^2)} = n^3 + \underbrace{O(n^2)}_{\text{Big O notation, describes asymptotic behavior}}$$

If n is large, say $n \geq 1000 \Rightarrow$
we can say # FLOPs $= O(n^3)$

For modified code (p.138, see (7.1) & (7.2)),
we get $\frac{2}{3}n^3 + O(n^2)$

5.1

```
% Gaussian elimination (LU factorization) without pivoting
% for solving linear systems Ax=b
%
A = [1 2 3; 4 5 6; 7 8 0]; b = [1 0 2]';
n = size(A,1);
% ---------------------------------------------%
% This is Step 1 of Gaussian Elimination
%---------------------------------------------%
for i=2:n                  % Loop over rows below row 1
   mult = A(i,1)/A(1,1);        % Subtract this multiple of row 1 from
                         % row i to make A(i,1)=0.
   A(i,:) = A(i,:)-mult*A(1,:); % (this line is equivalent to the "for loop:
                         % for k=1:n, A(i,k) = A(i,k)-mult*A(1,k); end; ")
   b(i) = b(i) - mult*b(1);
end
A % display A
%
%---------------------------------------------%
% All steps of Gaussian elimination
%---------------------------------------------%
A = [1 2 3; 4 5 6; 7 8 0]; b = [1 0 2]';
n = size(A,1);
for j=1:n-1                    % Loop over columns.
   for i=j+1:n                 % Loop over rows below j.
1 -   mult = A(i,j)/A(j,j);          % Subtract this multiple of row j from
                              % row i to make A(i,j)=0.
2n -  A(i,:) = A(i,:) - mult*A(j,:); % This does more work than necessary! WHY?
2 -   b(i) = b(i) - mult*b(j);
   end
end % the resulting A is an upper triangular matrix
A % display A
%---------------------------------------------%
% Summary: LU factorization (without pivoting)
%---------------------------------------------%
A = [1 2 3; 4 5 6; 7 8 0]; b = [1 0 2]';
%A = [2 1 1 0; 4 3 3 1; 8 7 9 5; 6 7 9 8]; b = [2 3 4 5]';
n = size(A,1);
for j=1:n-1                 % Loop over columns.
   for i=j+1:n              % Loop over rows below j.
      mult = A(i,j)/A(j,j);   % Subtract this multiple of row j from
                        % row i to make A(i,j)=0.
      A(i,j+1:n) = A(i,j+1:n) - mult*A(j,j+1:n);  % This works on columns j+1 to n
      A(i,j) = mult; % A(i,j) = 0 => we use this space to store L(i,j)
   end
end
% As a result, A stores both matrices L and U:
A
% find U as
U = triu(A)
% and L as
L = A-U + eye(n)
% Find solution of Ax = b:
% y = L\b; x = U\y;
```

total Flops → see p.5

at row i, stage j

$= 2n+3$ flops

if we replace $A(i,:) = A(i,:) - mult \times A(j,:)$

with

$A(i,j:n) = A(i,j:n) - mult \times A(j,j:n)$

$\Rightarrow 2n \to 2(n-j+1)$

$\Rightarrow \#\text{Flop's} = \frac{2}{3}n^3 + O(n^2)$

```
function y=lsolve(L,b);
%
% Given Ax=b and A = LU, solve Ly = b and return y
n = length(b); % length of the vector b
for i = 1:n
   y(i) = b(i);
   for j = 1:i-1 % solve for each y(i) using  previously computed y(j), j = 1,...,i-1
      y(i) = y(i) - L(i,j)*y(j);
   end
end

y = y(:);
```

for each ith step, we have

$\sum_{j=1}^{i-1} (\underbrace{1 \text{ subtraction} + 1 \text{ mult.}}_{2})$

$\Rightarrow$ total FLOPs ; $\sum_{i=1}^{n}\sum_{j=1}^{i-1} 2 = 2\sum_{i=1}^{n}(i-1)$

$= 2\sum_{i=1}^{n-1} i = 2\frac{(n-1)n}{2} = n^2-n = \boxed{n^2 + O(n^2)}$

[Routine $Ux=y$ is left as an exercise.]