MATH 5620
SPRING 2011

Chap 6-7 Solving linear systems of equations

Preliminaries: I will assume some familiarity with linear algebra, and in particular the following notions:

- matrix, vector
- matrix-matrix, matrix-vector product
- linear system
- singular, non-singular matrices
- determinants
- solving linear systems by hand (with row by row operations)

Goal of this section: solve a linear system of the form

$$Ax = b,$$

where $A \in \mathbb{R}^{n \times n}$ matrix (non singular) and $x \in \mathbb{R}^n$, $b \in \mathbb{R}^n$.

There are two main kinds of algorithms for solving linear systems:

Direct methods

- LU factorization (Gaussian Elimination)
- Cholesky factorization

→ Requires full knowledge of A

→ Solves for x up to maximal accuracy that can be expected.

→ Can be made very efficient for SPARSE matrices

Iterative methods

- Conjugate Gradient (CG)
- Generalized Minimum Residual (GMRES)
- Requires only knowledge of A (or sometimes A^T) acting on a vector
- iterations $x_{n+1} = f(x_n)$ can be stopped when desired accuracy is reached

Direct Methods (Chap 6 in class book)

First here are some examples of systems $Ax = b$ that are easy to solve:

Diagonal systems:

$$A = \operatorname{diag}(a_1, a_2, \dots, a_n)$$

$$= \begin{bmatrix} a_1 & & & \\ & a_2 & & \\ & & \ddots & \\ & & & a_n \end{bmatrix}$$

then (if the $a_i \neq 0$):

$$x = A^{-1} b = \begin{bmatrix} b_1/a_1 \\ b_2/a_2 \\ b_3/a_3 \\ \vdots \\ b_n/a_n \end{bmatrix} = b./a \text{ in Matlab notation}$$

Lower triangular systems

$$\underset{A}{(\Delta)}\underset{x}{[\ ]} = \underset{b}{[\ ]}, \quad \text{here } A = \begin{bmatrix} a_{11} & & & & \\ a_{21} & a_{22} & & & 0 \\ a_{31} & a_{32} & a_{33} & & \\ \vdots & & & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

How to solve such a system:

$$x_1 = b_1/a_{11}$$

$$x_2 = (b_2 - a_{21}x_1)/a_{22}$$

$$x_3 = (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33}$$

$$\vdots$$

$$x_i = \Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j\Big)/a_{ii} \qquad (i \leq n)$$

this is called forward substitution as we compute x_i "forward" (in direction of increasing i)

We can write forward substitution as a for loop.

$$\boxed{\Theta(n^2) \text{ flops.}}$$

```
for $i = 1 .. n$
    $x_i = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j) / a_{ii}$
```

or using Matlab notation:

```
for $i = 1 .. n$
    $x_i = (b_i - A_{i,1:i-1} \; x_{1:i-1}) / A_{ii}$
```

Upper triangular systems: (∇)

Algorithm for solving them is similar, it's backward substitution, because we progress backwards from x_n to x_1:

Backward substitution

```
for $i = n : -1 : 1$
    $x_i = b_i - \sum_{j=i+1}^{n} a_{ij} x_j$
```

Triangular systems with permuted rows

For example consider permutation $p_1 \, p_2 .. p_n$ of $\{1, 2, \dots, n\}$.

If the matrix $[a_{p_i j}]_{i,j=1..n}$ is lower (or upper) triangular

then we can apply forward (or backward) substitution with a slight modification

```
for $i=1,\dots,n$
    $x_i = \left(b_i - \sum_{j=1}^{i-1} a_{p_i j}\, x_j\right)/a_{p_i i}$    ($n^2$ flops)
```

or in Matlab notation:

```
for i=1:n,
    x(i) = (b(p(i)) - A(p(i),1:i-1)*x(1:i-1))/A(p(i),i)
```

Similarly Backward substitution becomes:

```
for i= n:-1:1
    $x_i = \left(b_i - \sum_{j=i+1}^{n} a_{p_i j} x_j\right)/a_{p_i i}$
```

Gaussian Elimination (or LU decomposition)

Recall GE transforms $n\overset{n}{\boxed{A}} \to$ ◹ by applying simple transf on the left.

this process is equivalent to :

$$\underbrace{L_{n-1} \cdots L_2 L_1}_{L^{-1}} A = U$$

where $U = \overset{n}{◹}\, n$ upper triangular

$L = n\,\underset{n}{◺}$ lower triangular with ones on diag (unit-lower triangular).

$= L_1^{-1} \cdots L_{n-1}^{-1}$

$$\boxed{A = LU}$$ LU-factorization.

Example:

$$\underset{A}{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}} \xrightarrow{L_1} \underset{L_1 A}{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}} \xrightarrow{L_2} \underset{L_2 L_1 A}{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & 0 & \times & \times \\ & 0 & \times & \times \end{bmatrix}} \xrightarrow{L_3} \underset{L_3 L_2 L_1 A}{\begin{bmatrix} \times & \times & \times & \times \\ & \times & \times & \times \\ & & \times & \times \\ & & 0 & \times \end{bmatrix}}$$

gray = not updated at this step

k-th transf. introduces zeros below diagonal in column k, by subtracting multiples of row k from rows k+1, .., m

Another example:

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$$

$$L_1 A = \begin{bmatrix} 1 & & & \\ -2 & 1 & & \\ -4 & & 1 & \\ -3 & & & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ & 3 & 5 & 5 \\ & 4 & 6 & 8 \end{bmatrix}$$

$$L_2 L_1 A = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & -3 & 1 & \\ & -4 & & 1 \end{bmatrix} L_1 A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 2 & 4 \end{bmatrix}.$$

$$L_3 L_2 L_1 A = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1 & 1 \end{bmatrix} L_2 L_1 A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 0 & 2 \end{bmatrix} = U$$

Now we need $L = L_1^{-1} L_2^{-1} L_3^{-1}$

$$L_1^{-1} = \begin{bmatrix} 1 & & & \\ -2 & 1 & & \\ -4 & & 1 & \\ -3 & & & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & & 1 & \\ 3 & & & 1 \end{bmatrix}$$

same for L_2, L_3
inverse obtained by negating
entries below diag.

Also $L_1^{-1} L_2^{-1} L_3^{-1}$ can be obtained by simply putting entries in right pla

$$A = \underbrace{\begin{bmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & 3 & 1 & \\ 3 & 4 & 1 & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 0 & 2 \end{bmatrix}}_{U}$$

In general:

Let x_k be k-th column of matrix at beginning of step k.

then transf L_k is s.t.

$$x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ x_{k+1\,k} \\ \vdots \\ x_{nk} \end{bmatrix} \xrightarrow{L_k} L_k x_k = \begin{bmatrix} x_{1k} \\ \vdots \\ x_{kk} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{n,k} & & & 1 \end{bmatrix}$$

where: $l_{jk} = \dfrac{x_{jk}}{x_{kk}}$, $j = k+1, \dots, n$

(L_k = subtract from j-th row $\dfrac{x_{jk}}{x_{kk}} \times$ k-th row)

We can now explain the recipes we used:

$$L_k = I - l_k e_k^T \quad \text{where: } l_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{n,k} \end{bmatrix}, \quad e_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} \\ \\ \\ \leftarrow k \\ \\ \\ \\ \end{array}$$

Because first k entries of l_k are zero:

$$e_k^T l_k = 0$$

$$\Rightarrow (I - l_k e_k^T)(I + l_k e_k^T) = I$$

For the reason that we need to transcribe entries to perform multiplication $L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}$:

Look at $L_k^{-1} L_{k+1}^{-1}$:

$$(I + l_k e_k^T)(I + l_{k+1} e_{k+1}^T)$$

$$= I + l_k e_k^T + l_{k+1} e_{k+1}^T + l_k \underbrace{e_k^T l_{k+1}}_{=0} e_{k+1} .$$

$$\Rightarrow \quad L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & & \ddots & \\ l_{n1} & l_{n2} & & l_{n,n-1} & 1 \end{bmatrix}$$

Note: in practise matrices L_k are never computed, we just keep track of multipliers by storing them directly in L.

Gaussian Elimination (no pivoting)

```
$U = A$, $L = I$
for $k = 1 \dots n-1$
    for $j = k+1, \dots, n$
        $l_{jk} = u_{jk} / u_{kk}$
        $u_{j,k:m} = u_{j,k:m} - l_{jk} u_{k,k:m}$
```

compute multiplier for row j

compute updated row j

$$\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \to \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}$$

note: A, L, U are not needed, everything can be stored in same memory as $A = [\, L \backslash U \,]$

Operation count: $O(n^3)$ (three nested loops of length n)

$\frac{2}{3}n^3$ flops. (if we do a more careful count of flops)

When can Gaussian elimination break down?

Here are some 2x2 examples, where the row op. involved in GE break down or give inaccurate answers.

(E1) $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ ⇝ can't add a multiple of (1st eq) to introduce zeros in (2nd eq) (but we know that it admits solution $x_1 = 1, x_2 = 1$)

(E2) $\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\downarrow$ GE

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & 1-\varepsilon^{-1} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2-\varepsilon^{-1} \end{bmatrix} \Rightarrow \begin{cases} x_2 = \dfrac{2-\varepsilon^{-1}}{1-\varepsilon^{-1}} \approx 1 \\ x_1 = (1-x_2)\varepsilon^{-1} \approx 0 \text{ (WRONG!)} \end{cases}$$

Since in floating point arithmetic x_2 could be so close to 1 that $fl(1-x_2) = 0$, where $fl(x) =$ floating point repr of x.

The problem is not only smallness of a_{11}, but how small it is compared to other elements in its row.

(E3) $\begin{bmatrix} 1 & \varepsilon^{-1} \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \varepsilon^{-1} \\ 2 \end{bmatrix}$ (same as (E2) but rescaling first row by ε)

$$\xrightarrow{GE} \begin{bmatrix} 1 & \varepsilon^{-1} \\ 0 & 1-\varepsilon^{-1} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \varepsilon^{-1} \\ 2-\varepsilon^{-1} \end{bmatrix} \rightsquigarrow \begin{cases} x_2 = \dfrac{2-\varepsilon^{-1}}{1-\varepsilon^{-1}} \approx 1 \\ x_1 = \varepsilon^{-1} - \varepsilon^{-1}x_2 \approx 0 \text{ (WRONG!)} \end{cases}$$

Solution to this problem is to permute rows:

$$\text{(E4)} \quad \begin{pmatrix}1 & 1\\ \varepsilon & 1\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}=\begin{pmatrix}2\\ 1\end{pmatrix} \xrightarrow{GE} \begin{pmatrix}1 & 1\\ 0 & 1-\varepsilon\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix}=\begin{pmatrix}2\\ 1-2\varepsilon\end{pmatrix}$$

$$\Rightarrow \quad x_2 = \frac{1-2\varepsilon}{1-\varepsilon} \approx 1 \qquad \text{(correct solution)}.$$

$$x_1 = 2 - x_1 \approx 1$$

<u>Morale of these examples</u>: We need to permute rows in general.

This is called <u>pivoting</u> because we choose $\boxed{\text{pivot}}$ (the akk by which we divide in LU algorithm) by something that is not zero (or some more elaborate rule as we will see).

In general LU factorization routines give a factorization of the form:

$$\boxed{PA = LU} \qquad (*)$$

where A, L, U are as in the LU factorization and P is a <u>permutation matrix</u>. This is the identity with permuted rows.

Here is an example:

$$P = \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}$$

which permute a vector as follows:

$$P\begin{bmatrix}1\\ 2\\ 3\\ 4\end{bmatrix} = \begin{bmatrix}3\\ 4\\ 2\\ 1\end{bmatrix}$$

Most of the time P is not represented as a matrix but simply as a vector p s.t.

$$e_{p(i)} = P e_i , \quad \text{here } e_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow i\text{th pos.}$$

= canonical basis vector

To solve a system $Ax = b$, assuming we have factorization (*) all we need to do is:

1) Solve $Lz = Pb$
2) Solve $Ux = z$

How to choose the pivot?

Our little examples (E1) - (E4) suggest two rules:

- select largest pivot (in $|\cdot|$) i.e. p s.t.

$$|u_{pk}| \geqslant |u_{ik}| \quad \text{for } i \geqslant k ,$$

- select largest pivot (in $|\cdot|$) relative to all other elements in a row (this is the recommended pivot selection rule because of (E3), and the one given in your book) i.e. p s.t.:

$$\frac{|u_{pk}|}{\max\limits_{k \leq j \leq n} |u_{pj}|} \geqslant \frac{|u_{ik}|}{\max\limits_{k < j \leq n} |u_{ij}|} \quad \text{for } i \geqslant k$$

Here is how Gaussian Elimination with (partial) pivoting looks like:

```
$U = A$, $L = I$, $P = I$
for $k = 1, .., m-1$
    1. Choose index $i \geq k$ of pivot according to one of previous rules.
    2. $u_{k,k:m} \leftrightarrow u_{i,k:m}$        (switch k-th row with pivot row $i$)
    3. $l_{k,1:k-1} \leftrightarrow l_{i,1:k-1}$    (Switch rows in $L$, this keeps $L = (\Delta)$ structure, and is needed to be consistent w/ permutations in $U$).
    4. $p_{k,:} \leftrightarrow p_{i,:}$            (this how we keep track of permutation, but could have done with a vector...)
    for $j = k+1, .. m$
        $l_{jk} = u_{jk} / u_{kk}$
        $u_{j,k:n} = u_{j,k:n} - l_{jk} u_{k,k:m}$
```

$O(n^3)$ flops

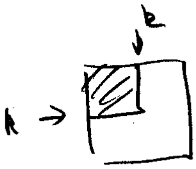
Note: There is also a *complete pivoting* where row and column permutations are allowed. This is much more expensive because we need to look for the pivot among a greater number of matrix elements, and the benefits in stability are not that significant compared to partial pivoting. The factorization is then:

$$PAQ = LU, \text{ where } P, Q \text{ are permutation matrices.}$$

Note that without pivoting the LU factorization of a matrix A may not even exist. Here is a sufficient condition that ensures existence:

Theorem: If all n leading (principal) minors of $A \in \mathbb{R}^{n \times n}$ are non-singular then A has an LU decomposition.

Here we call "k-th leading minor":



$= A(1:k, 1:k)$

One can dispense from pivoting when A is diagonally dominant (which sometimes appear in applications):

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$$

= diag element larger than sum of everything else in that row.

Cholesky Factorization

In many applications, A is symmetric positive definite, meaning it has the properties:

i) $A = A^T$ (symmetry)

ii) For any $x \in \mathbb{R}^n, x \neq 0$ we have $x^T A x > 0$.
(pos. definiteness)

Theorem:

A s.p.d $\Rightarrow$ A invertible

Proof: $Ax = 0 \Rightarrow x^T A x = 0 \Rightarrow x = 0$

Theorem: A s.p.d. always has an LU facto.

proof: All principal minors of A are s.p.d (and thus invertible) (why?):

$$(x_1 \cdots x_k\, 0 \cdots 0)\, A \begin{pmatrix} x_1 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{pmatrix} = (x_1 \cdots x_k)\, \underset{\substack{\text{k-th principal minor of } A. \\ = A(1:k,\,1:k)}}{A_k} \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix} > 0 \text{ if } (x_1 \cdots x_k) \neq 0.$$

Now let us see if the LU facto of A is any special.

$$\left.\begin{aligned} A &= LU \\ \| \\ A^T &= U^T L^T \end{aligned}\right\} \Rightarrow \underbrace{UL^{-T}}_{\substack{(\text{◹})(\text{◹}) \\ =(\text{◹})}} = \underbrace{L^{-1}U^T}_{\substack{(\text{◺})(\text{◺}) \\ =(\text{◺})}} = \underset{\text{diagonal}}{(\diagdown)} = D$$

here we used that:

$$(\text{◹})^{-1} = (\text{◹}) \qquad (\text{◹})(\text{◹}) = (\text{◹})$$

$$(\text{◺})^{-1} = (\text{◺}) \qquad (\text{◺})(\text{◺}) = (\text{◺})$$

Now write:

$$A = L\,\underbrace{U\,\mathrlap{\overbrace{\phantom{L^{-T}L^{T}}}^{=I}}L^{-T}}_{=D}L^{T} = LDL^T$$

Notice that D must have positive entries since:

$$D = L^{-1} A L^{-T}$$

$$x^T D x = x^T L^{-1} A L^{-T} x$$

$$= (L^{-T}x)^T A (L^{-T}x) > 0 \text{ if } x \neq 0.$$

Thus:

$$A = L D^{\frac{1}{2}} D^{\frac{1}{2}} L^T, \text{ where } D^{\frac{1}{2}} = \operatorname{diag}(d_1^{\frac{1}{2}}, \ldots, d_n^{\frac{1}{2}})$$

$$= \tilde{L} \tilde{L}^T, \quad \text{where } \tilde{L} = L D^{\frac{1}{2}}$$

Theorem If A s.p.d. it admits a unique Cholesky factorization

$$A = L L^T$$

Note: some routines give $L^T = (\nabla)$ instead of L.

To derive Cholesky factorization algorithm, partition

$$A = \begin{bmatrix} a_{11} & w^T \\ w & K \end{bmatrix}$$

and consider the following identity

$$A = \begin{bmatrix} a_{11} & w^T \\ w & K \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ w/\alpha & I \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & K - ww^*/a_{11} \end{bmatrix} \begin{bmatrix} \alpha & w^T/\alpha \\ & I \end{bmatrix}$$

(check)

Here we apply to A on both sides an elementary transformation so as to introduce zeros in first column *and* in first row simultaneously (which is what we expect since $A = A^T$)

The algorithm is: (here we compute $R = L^T = (\nabla)$ for convenience)

Cholesky

$\frac{1}{3} n^3$ flops

works only w/ (∇) part

```
$R = A$
for $k = 1, \dots, n$
    for $j = k+1, \dots, n$
        $R_{j,j:n} = R_{j,j:n} - R_{k,j:n} R_{kj} / R_{kk}$
    $R_{k,k:n} = R_{k,k:n} / \sqrt{R_{kk}}$
```

- Cholesky factorisation is very robust and does not have any of the stability problems that may affect LU facto
- No need for pivoting
- breaks down when $R_{kk} < 0$, but incomplete Cholesky can be used to approx an indefinite matrix by a pos semi def matrix (i.e. LL^T)