System of Linear Equations

Part-1: Introduction

Lecturer:

T. Basaruddin & Heru S

What is it?

- Definition: *Ax=b*

$$
\begin{array}{ccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + \cdots + & a_{1n}x_n & = & b_1 \\
a_{21}x_1 & + & a_{22}x_2 & + \cdots + & a_{2n}x_n & = & b_2 \\
\vdots & & \vdots & \ddots & \vdots & \vdots & \vdots \\
a_{m1}x_1 & + & a_{m2}x_2 & + \cdots + & a_{mn}x_n & = & b_m
\end{array}
$$

- Classification of Linear System
  - Over-determined: $m > n$
  - Square system: $m = n$
  - Under-determined: $m < n$
- This chapter will deal with square system

Why is it important?

- Ubiquity of linear equations system
  - Mathematical models of numerous real world problems
  - Linearization of more complex/non-linear model
- Accuracy is in some cases very critical
  - Very limited margin for error
- Sizes or scale can be so large
  - Order of 10^9 is not uncommon

Review Linear Algebra

- Existence of solution
  - *A* is invertible (non-singular)
  - *b* inside the space spanned by cols of *A*
- The solution is unique, i.e. $x = A^{-1} b$
- However
  - the inverse is not easily available, if it is, it is not economical to compute
  - analytically a square matrix is either singular or non-singular, but numerically a matrix can be nearly singular

Norms

Vector norms
$$\| x \|_p = \left(\sum_{i=1}^{n} | x_i |^p \right)^{1/p}$$

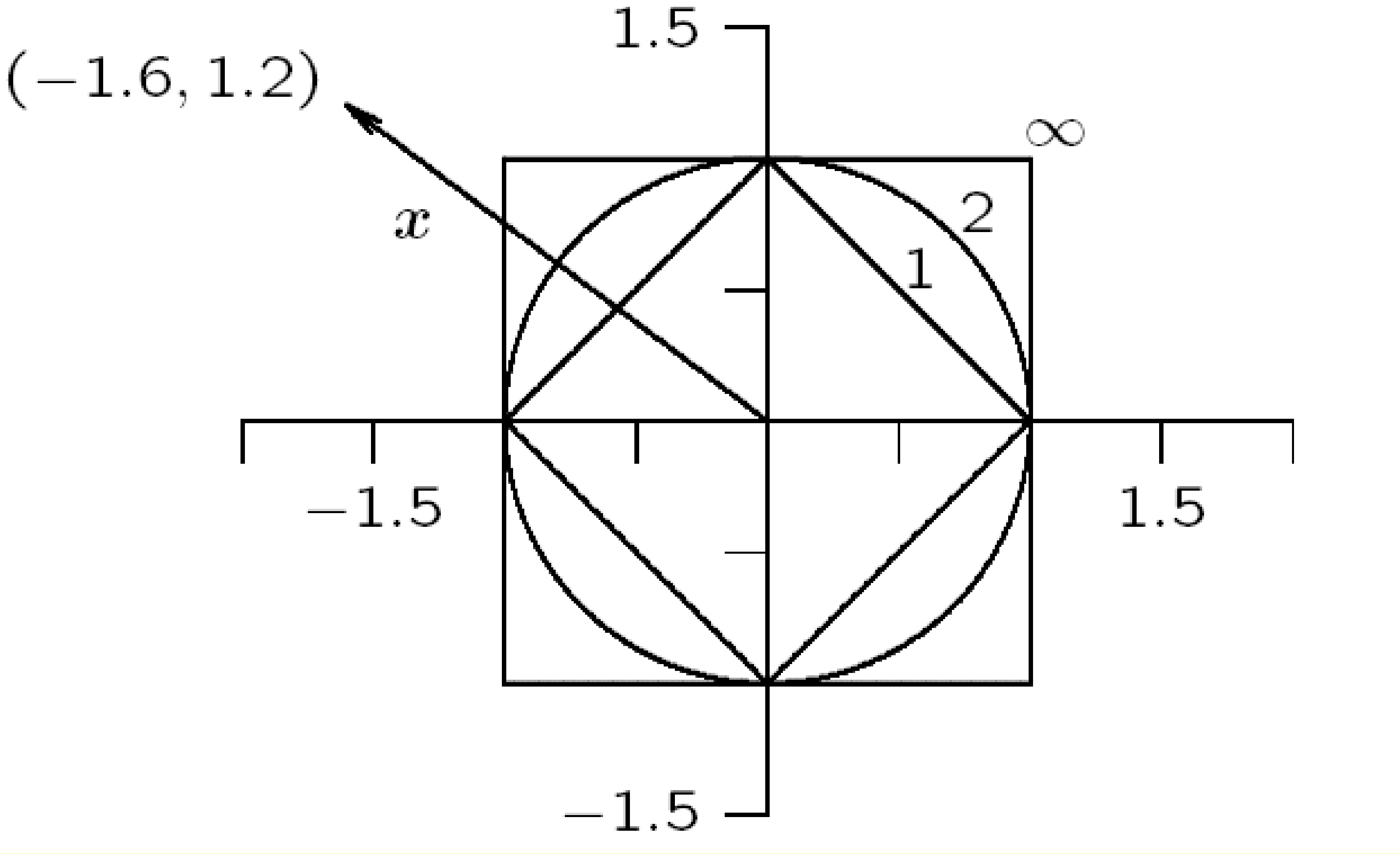
particularly

$$\| x \|_1 = \sum_{i=1}^{n} | x_i |$$

$$\| x \|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

$$\| x \|_\infty = \max_{1 \le i \le n} | x_i |$$

Unit circle in p - norms



Matrix Norms

$$\| A \|_p = \sup_{x \neq 0} \frac{\| Ax \|_p}{\| x \|_p}$$

$$= \sup_{\|x\|_p = 1} \| Ax \|_p$$

particularly

$$\| A \|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} | a_{ij} |;$$

$$\| A \|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} | a_{ij} |;$$

$$\| A \|_2 = \sqrt{\rho(A^T A)}$$

$\rho(A^TA)$ = max eigenvalue of A^TA

Condition Number and Sensitivity

$$K(A) = \| A \| \| A^{-1} \|$$ for some norms

- It measures the degree of departure from non-singularity
  - Matlab: **`cond(A) or rcond(A)`**
- The bigger the value of $K(A)$ the closer A to singularity
  - If A is singular then $K(A) = \infty$
- If $K(A)$ is large, the linear system $Ax=b$ is ill-conditioned

  In this case

$$\frac{\| \Delta x \|}{\| x \|} \leq K(A) \frac{\| \Delta b \|}{\| b \|}$$ and $$\frac{\| \Delta x \|}{\| x \|} \leq K(A) \in_{mach}$$

Where Δ represent the deviation to the actual value

Caveat ! Ill-conditioned system → more difficult to get accurate solution

Triangular System

Lower triangular system: *Lx=b*

$$\begin{pmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Upper triangular system: *Ux=b*

$$\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

Forward Elimination

Given a linear system *Lx=b* where *L* is a non-singular lower triangular matrix, then *x* can be computed through:

$$x_1 = b_1 / l_{11};$$

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

$$\vdots$$

$$x_n = (b_n - \sum_{j=1}^{n-1} l_{nj} x_j) / l_{nn}$$

Complexity

$$\sum_{i=2}^{n} (i-1) = \sum_{i=1}^{n-1} i = n(n-1)/2$$ **flops**

Algorithm: Matlab-Implementation

```
function [x] = forward(L,b)
%Input          : matriks L dan vector b
%Output         : vektor solusi x
        n = length(b);
        x = zeros(n,1);
        x(1) = b(1)/L(1,1);
        for i=2:n
            x(i) = (b(i)-L(i,1:i-1)*x(1:i-1))/L(i,i);
        end
```

Potential sources of errors:

- small value of L(i,i)
- LSD in case of b(i) is very close to **L(i,1:i-1)*x(1:i-1)**

How to avoid them ?

Backward Substitution

Given a linear system $Ux=b$ where U is a non-singular upper triangular matrix, then x can be computed through:

$$\begin{aligned}
x_n &= b_n / u_{nn}; \\
x_{n-1} &= (b_{n-1} - u_{n-1,n} x_n) / u_{n-1,n-1}; \\
&\vdots \\
x_1 &= (b_1 - \sum_{j=n}^{i+1} u_{1j} x_j) / u_{11}
\end{aligned}$$

Complexity is the same as the forward elimination

Back-sub: Matlab Implementation

```
function [x] = Backsub(U,b)
%Input : matriks U dan vector b
%Output      : vektor solusi x
    n = length(b);
    x = zeros(n,1);
    x(n) = b(n)/U(n,n);
    for i=n-1:1
        x(i) = (b(i)-U(i,i+1:n)*x(i+1:n))/U(i,i);
    end
```

Closing Remarks

- Non-singularity is the necessary and sufficient conditions for the existence and uniqueness of solution → numerically there are more to it
- The linear system $Ax=b$ can easily be solved if the matrix coefficient is triangular
  - Would it be possible to transform a general matrix into a triangular one ?

System of Linear Equations

Triangular Factorization

Motivation

- As previously discussed, some form of linear equation systems are easier to solve (e.g. triangular system)
- There are ways for transforming a matrix into some desirable form
- For the system $Ax=b$ and linear transformation T
  - T preserves the integrity of the solution i.e. $T*Ax=T*b$
  - T has to be invertible

Gaussian Matrix

Given a non-zero vector $a = (a_1, a_2, \cdots, a_k, \cdots a_n)^T$ where $a_k \neq 0$

There exists matrix M_k so that

$$M_k a = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

In this case

$$m_i = a_i / a_k, i = k+1, \cdots, n$$

The multiplication of *M* with a matrix *A* = elementary row operation

Gaussian Elimination

Given M_k is the Gaussian matrix and let $A^k = M_k A^{k-1}$ where $A^0 = A$ then $A^{(n-1)}$ is an upper triangular system provided that $A_{kk}^{k-1} \neq 0$

$$M_{n-1} M_{n-2} \cdots M_1 A = U$$

Example

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix}$$

➔ $M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}$ and $M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}$

Thus

$$MAx = Mb$$

$$Ux = \tilde{b}$$

Algorithm – GE - Matlab

```
Function [U,bt]=ge(A,b)
%Input: square matrix A and right hand side vektor b
%Output: upper triangular matrix U and associated rhs bt
[n,n]=size(A);
A=[A b] % Augmenting b in the last column of A
for k=1:n-1
  for i=k+1:n
          m=A(i,k)/A(k,k); %assume A(k,k)≠0
          A(i,:) = A(i,:) –m*A(k,:);
  end
end
U=triu(A(1:n,1:n));
bt=A(:,n+1)
```

Complexity: $\sum_{k=1}^{n-1}(n-k)^2 \approx n^3/3$

Note: No matrix *M* is needed

Summing up

- The Gaussian matrix is a lower triangular matrix
- Multiplication of two lower triangular matrix = lower triangular matrix
- The inverse of a non-singular lower triangular matrix is a lower triangular matrix
- The Gaussian matrix is non-singular → invertible

$$M\,A = U \quad \rightarrow \quad A = M^{-1}\,U = L\,U$$

Inverse of *M*

Let $a_k = (0, \cdots, 0, m_{k+1}, \cdots, m_n)$ and e_k be the *k*-th unit vector, then the Gaussian matrix M can be written as

$$M_k = (I - a_k e_k^T)$$

Further more it can easily be shown that $M_k^{-1} = (I + a_k e_k^T)$

since

$$\begin{aligned}(I - a_k e_k^T)(I + a_k e_k^T) &= I - (a_k e_k^T)(a_k e_k^T) \\ &= I - (e_k^T a_k) a_k e_k^T = I\end{aligned}$$

Thus, the inverse of *M* is also a Gaussian matrix → lower triangular

LU Factorization

Recall that $(M_{n-1}M_{n-2}\cdots M_1)A = U$

$$A = (M_{n-1}M_{n-2}\cdots M_1)^{-1}U$$

$$= (M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1})U$$

But it is already shown that the inverse of *M* is also a lower triangular matrix, thus $(M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1})$ must be a lower triangular matrix, say *L*

$$A = L\,U$$

Conclusion:
Given that the Gaussian elimination process is succeeded, there exists a lower triangular matrix *L* and an upper triangular *U* so that *A* = *LU*

L is a unit lower triangular matrix (diagonal element is 1)

Algorithm

```
Function [L,U]=lufactor(A)
%Input matrix A
%output matrices L & U
[n,n]=size(A);
L=eye(n) % an identity matrix of order nxn
for k=1:n-1
   L(k+1:n,k)=A(k+1:n,k)/A(k,k); %assumed A(k,k)≠0
   for i=k+1:n
            A(i,k:n)=A(i,k:n)-L(i,k)*A(k,k:n);
   end
end
U=triu(A);
```

Cost is exactly the same as the GE

Solving the $Ax=b$ through LU

Let A can be factorized as $A = LU$, then the solution x of $Ax=b$ can be found as follows:

1. Solve $Ly = b$ using the forward elimination
2. Solve $Ux = y$ using the backward substitution

Note that, the factorization of A is independent of the right hand side b
It is more attractive than the GE in case of multiple right hand sides

Pivoting

The Gaussian elimination process breaks down if
the pivot element $\boldsymbol{A^k(k,k)=0}$.
This however does not necessarily mean $\boldsymbol{A}$ is singular, e.g.

$$A = \begin{pmatrix} 0 & 4 & 3 \\ 1 & 3 & 1 \\ 3 & 4 & 3 \end{pmatrix}$$

The problem can easily be resolved by interchanging the
Rows or columns. This strategy is called “pivoting”.
Row/column interchanges do not change the system.

Problem will also occur if the pivot element is small.
In this case the pivoting is also needed for accuracy.

Permutation Matrix

Pivoting can mathematically be presented using the permutation matrix. A permutation matrix is an identity matrix with some row/col interchanges. E.g.

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

Effect of permutation

$$PA = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 4 & 3 \\ 1 & 3 & 1 \\ 3 & 4 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 1 \\ 0 & 4 & 3 \\ 3 & 4 & 3 \end{pmatrix}$$

$$AP = \begin{pmatrix} 0 & 4 & 3 \\ 1 & 3 & 1 \\ 3 & 4 & 3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 3 \\ 3 & 1 & 1 \\ 4 & 3 & 3 \end{pmatrix}$$

Pivoting strategy

- Partial pivoting
  - Searching the max element in the pivot column $\max_{k \le i \le n}\{abs(A_{ik}^{k-1})\}$
- Diagonal pivoting
  - Searching the max element in the diagonal; preserve symmetry
- Total pivoting
  - Searching the max in the entire sub-matrix; this is actually unnecessary.

Algorithm: LU Factorization with partial pivoting

```
%Input matriks A
%output matriks L & U dan vektor pivot p
[n,n]=size(A);
L=eye(n);
p=1:n; % vector pivot
for k=1:n-1
  % choose pivot
  [c,m]=max(abs(A(k:n,k))); % max in k-th col
  if c==0
     quit; % A is singular
  end
  tmpA=A(k,:); tmpp=p(k); tmpl=L(k,1:k-1);tempA=A;
  tempA(k,:)=tempA(m,:); p(k)=p(m);L(k,1:k-1)=L(m,1:k-1);
  A(m,:)=tmpA;p(m)=tmpp; L(m,1:k-1)=tmpl; L(p(k),k)=1.0;
for i=k+1:n
    L(p(i),k)=tempA(i,k)/tempA(k,k);
    for j=k:n
        tempA(i,j)=tempA(i,j)-L(p(i),k)*tempA(k,j);
    end
  end
end
U=triu(A);
```

No additional flops

Retrieving the Solution x

- Partial pivoting with the pivot P
  - $PAx=Pb$ → with $PA = LU$
  - Solve $Ly=Pb$ (b is to be reordered)
  - Solve $Ux=y$
- Diagonal pivoting with the pivot P
  - $PAP\ Px = Pb$ → $PAP = LU$
  - Solve $Ly=Pb$
  - Solve $Uw=y$
  - $x = Pw$

Error Analysis of LU

Let

$$\hat{L}\hat{U} = A + H$$

Then

$$|H| \le 3nu\{|A| + |\hat{L}||\hat{U}|\} + O(u^2)$$

Error analysis of solution thru LU

Let $\widehat{L}\widehat{U}$ be the computed LU-factorization of A and $\widehat{x}$ be the computed solution to the linear equation $Ax=b$ thru the LU factorization.

Then $\widehat{x}$ satisfies $(A+E)\widehat{x}=b$ where

$$|E| \leq n\mu\{3|A|+5|\widehat{L}||\widehat{U}|\}$$

Define now the growth factor

$$\rho = \max_{i,j,k} \frac{|a_{ij}^k|}{\|A\|_\infty}$$

Then

$$\|E\|_\infty \leq 8n^3\rho\|A\|_\infty \mu$$

Thus, pivoting improves accuracy !

System of Linear Equations

Special Linear System

Motivation

- GE or LU factorization can be used to solve a general linear system (with non-singular matrix coefficient)
- The cost is reasonably expensive of $O(n^3)$
- Need to exploit matrix structure for efficiency
  - Symmetric, Definite Positive, Banded, etc.

Symmetric Matrix

- Consider the linear system *Ax=b*
- *A* is a symmetric matrix, i.e. $A^T = A$
- Need to consider a symmetric factorization

$$A = LU \rightarrow A^T = (LU)^T = U^T L^T$$

This is in general not a symmetric factor. Consider now

$$A = LDL^T$$

Where *D* is a diagonal matrix. This is definitely a symmetric factorization.

$$A^T = (LDL^T)^T = (L^T)^T D^T L^T = LDL^T = A$$

We need to compute *L* and *D* only.

LDL^T - Factorization

Example

$$A = \begin{pmatrix} 2 & 4 & 6 \\ 4 & 9 & 14 \\ 6 & 14 & 19 \end{pmatrix}$$

1. D(1,1)=2; L(:,1)=$(1,2,3)^T$;
2. Note that elimination of *i*-th row = update *i*-th col
3. D(2,2)=9 – 4*2=1; L(:,2)=$(0,1,2)^T$;
4. D(3,3)=1 – 2*2=– 3

So,

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix}; \quad D = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -3 \end{pmatrix}$$

Cost ≈ 0.5 LU; but pivoting may be needed

Symmetric & Positive Definite

- *A* is defined as positive definite iff for any non-zero vector *x*, $x^TAx>0$
- Properties of symmetric & positive definite matrix
  - The eigen values are all positive
  - Non-singular
  - Strictly diagonally dominant

Cholesky Factorization

$$GG^T = A$$

$$\begin{pmatrix} g_{11} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ g_{i1} & \cdots & g_{ii} & \cdots & 0 \\ \vdots & \cdots & \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{ni} & \cdots & g_{nn} \end{pmatrix} \begin{pmatrix} g_{11} & \cdots & g_{j1} & \cdots & g_{n1} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \cdots & g_{jj} & \cdots & g_{nj} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & g_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ a_{i1} & \cdots & a_{ii} & \cdots & a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{ni} & \cdots & a_{nn} \end{pmatrix}$$

So

$$g_{i1}^2 + g_{i2}^2 + \cdots + g_{i,i-1}^2 + g_{ii}^2 = a_{ii} \quad \Rightarrow \quad g_{ii} = \sqrt{a_{ii} - \sum_{j=1}^{i-1} g_{ij}^2}$$

And

$$g_{i1}g_{j1} + g_{i2}g_{j2} + \cdots + g_{ij}g_{jj} = a_{ij} \quad \Rightarrow \quad g_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} g_{ik}g_{jk}}{g_{jj}}$$

An Example

Let

$$A = \begin{pmatrix} 4 & 4 & 6 \\ 4 & 5 & 8 \\ 6 & 8 & 22 \end{pmatrix}$$

Then

$$G(1,1) = \sqrt{4} = 2$$

$$G(2,1) = (4-0)/2 = 2$$

$$G(3,1) = (6-0)/2 = 3$$

$$G(2,2) = \sqrt{5-2*2} = 1$$

$$G(3,2) = (8-3*2)/1 = 2$$

$$G(3,3) = \sqrt{22-(3*3+2*2)} = 3$$

Cholesky - Algorithm

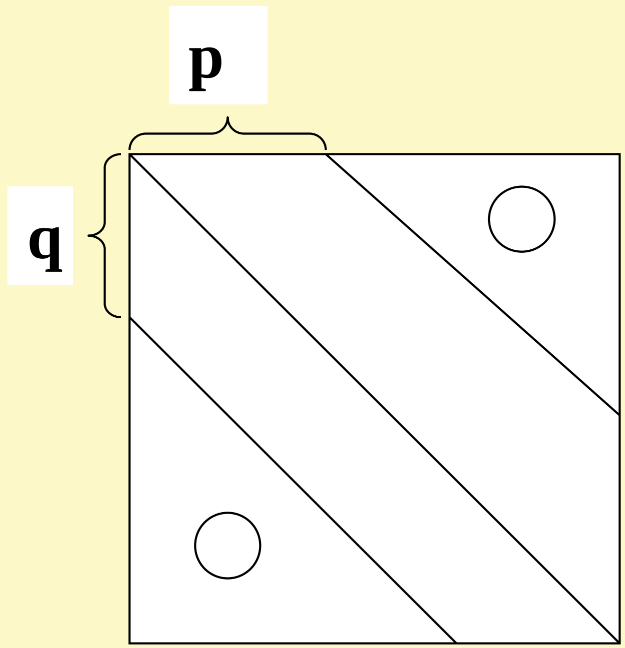
Algorithm 2.6: Cholesky Factorization

```
%Input: A symmetric and positive definite matrix  A
%Output: A lower triangular matrix G
[n,n]=size(A);
G(1,1)=sqrt(A(1,1));
for j=1:n-1
    for i=j+1:n
            G(i,j)=(A(i,j)-G(i,1:j-1)'*G(j,1:j-1))/G(j,j);
    end
    G(j+1,j+1)=sqrt(A(j+1,j+1)-G(j+1,1:j)'*G(j+1,1:j));
end
```

Overall Cost ≈ 0.5 LU; no pivoting is needed

Caveat: the function *sqrt* is rather costly

Banded System



p= upper-bandwidth; q= lower-bandwidth

Total-bandwidth = $p+q+1$

$$A(i,j) = 0 \; if \quad |i-j| > p+q+1$$

Algorithm: LU Fact for banded

```
%Input: A and bandwidth p,q
%Output: L and U
[n,n]=size(A); %p
L=I % matrik identitas
for k=1:n-1
    for i=k+1:min{k+q,n}
        L(i,k)=A(i,k)/A(k,k);
        for j=k+1:min{k+p,n}
            A(i,j)=A(i,j)-L(i,k)*A(k,j);
        end
    end
end
U=A;
```

Assume no pivoting

Cost

$$C(p,q)=\begin{cases} npq-\frac{1}{2}pq^2-\frac{1}{6}p^3+pn; & p\le q \\ npq-\frac{1}{2}qp^2-\frac{1}{6}q^3+qn; & p>q \end{cases}$$

Iterative refinement

Let $\hat{x}$ be the computed solution of Ax=b. Let also

$$e = x - \hat{x} \quad \text{and} \quad r = b - A\hat{x}$$

then

$$Ae = A(x - \hat{x}) = Ax - A\hat{x} = b - A\hat{x} = r$$

Thus

1. Compute $\boldsymbol{r}$
2. Solve $\boldsymbol{Ae}=\boldsymbol{r}$
3. Update $\boldsymbol{x}=\boldsymbol{x}+\boldsymbol{e}$
4. Re-iterate

will increase accuracy