



## 3 SYSTEMS OF LINEAR EQUATIONS

### 3.1 INTRODUCTION

#### 3.1.1 Remember: what we saw during the last lessons

**Error analysis** where  $\hat{x} \approx x$  an approximation

$$e_x = \hat{x} - x, \quad r_x = \frac{\hat{x} - x}{x}, \quad \kappa = \frac{|\text{rel error out}|}{|\text{rel error in}|} \approx \left| \frac{x f'(x)}{f(x)} \right|$$

**Finite difference formulas** are obtained from Taylor expansions

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h^2), \quad f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^3)$$

**Matlab**

Think of vector operations. Perhaps the most important command: `help`.

#### 3.1.2 Overview: what you will learn today

**Solve linear systems with Matlab**

LU-factorization, special- and tridiagonal linear systems.

**Norms**

Vector and matrix norms, condition number.

### 3.2 LU factorisation / Gaussian elimination (NAM 1.4, H 2.4.4)

**Example** Solve a simple  $3 \times 3$  system with LU-factorization

$$\begin{cases} 1x_1 + 2x_2 + 2x_3 = 3 \\ 4x_1 + 4x_2 + 2x_3 = 6 \\ 4x_1 + 6x_2 + 4x_3 = 10 \end{cases} \Rightarrow \mathbf{Ax} = \begin{bmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 10 \end{bmatrix} = \mathbf{b}$$

Annihilate subdiagonal entries of 1<sup>st</sup> column – subtract 4 times 1<sup>st</sup> row:

$$\mathbf{L}_1 \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 4 & 4 & 2 \\ 4 & 6 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & -2 & -4 \end{bmatrix}$$

$$\mathbf{L}_1 \mathbf{b} = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \\ 10 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ -2 \end{bmatrix}.$$

Annihilate subdiagonal entries of 2<sup>nd</sup> column – subtract  $\frac{1}{2}$  times 2<sup>nd</sup> row:

$$\mathbf{L}_2 \mathbf{L}_1 \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & -2 & -4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{L}_2 \mathbf{L}_1 \mathbf{b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 3 \\ -6 \\ -2 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix}.$$

Solve the upper triangular system with back-substitution from the bottom:

$$\mathbf{Ux} = \mathbf{L}_2 \mathbf{L}_1 \mathbf{x} = \begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix} = \mathbf{L}_2 \mathbf{L}_1 \mathbf{b}$$

$$\begin{cases} -1x_3 = +1 \\ -4x_2 = -6 + 6x_3 \\ 1x_1 = +3 - 2x_2 - 2x_3 \end{cases} \Rightarrow \mathbf{x} = \begin{bmatrix} -1 \\ 3 \\ -1 \end{bmatrix}$$

The Lower/Upper triangular matrices factorize the system matrix  $\mathbf{A} = \mathbf{LU}$

$$\mathbf{L} = \mathbf{L}_2 \mathbf{L}_1 = \begin{bmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{2} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 4 & \frac{1}{2} & 1 \end{bmatrix}$$

and can be reused to solve problems with multiple right-hand side vectors with only a forward substitution followed by a backward substitution:

$$\mathbf{Ax} = \mathbf{L} \underbrace{\mathbf{Ux}}_y = \mathbf{c} \Leftrightarrow \mathbf{Ly} = \mathbf{b}, \quad \mathbf{Ux} = y$$

The rows can be interchanged to keep the multipliers smaller than unity in a procedure called *pivoting* with a permutation matrix  $\mathbf{PA} = \mathbf{LU}$ .

## Matlab commands to solve a linear system with a full matrix

```
>> A=[ 1 2 2 ; 4 4 2 ; 4 6 4]; % Matrix
>> b=[3 6 10]'; % Right-hand side
>> x=A\b % Solution using LU factorization
```

```
x = -1
      3
     -1
```

```
>> [L,U,P]=lu(A) % LU factorization with pivoting
```

```
L = 1.00    0    0
     1.00    1.00    0
     0.25    0.50    1.00
```

```
U = 4.00    4.00    2.00
     0     2.00    2.00
     0     0     0.50
```

```
P =  0     1     0
     0     0     1
     1     0     0
```

## Computational cost $\mathcal{O}(n^2)$ numbers in memory and $\mathcal{O}(n^3)$ multiplications ( $2 \times$ number of unknowns) $\Leftrightarrow$ ( $4 \times$ memory) $\Leftrightarrow$ ( $8 \times$ CPU time)

```
>> pmax=10; T=zeros(pmax,3); % 2^pmax is the max rank
>> for p=5:pmax % 32 to 1024 unknowns
    n=2^p; A=eye(n)+rand(n); b=(1:n)'; % Setup a random system
    T(p,1)=p; tic; x=A\b; T(p,2)=toc; % Time for solving it
    T(p,3)=T(p,2)/8^p; % Verify scaling  $\mathcal{O}(p^3)$ 
end
```

```
>>
```

```
>> T(5:p,:) % p          time (sec)          time/8^p
```

```
T =  5.000000000000000  0.000770000000000  0.00000002349854
     6.000000000000000  0.003128000000000  0.00000001193237
     7.000000000000000  0.017816000000000  0.00000000849533
     8.000000000000000  0.295849000000000  0.00000001763397
     9.000000000000000  2.248615000000000  0.00000001675349
    10.000000000000000 17.294158000000000  0.00000001610644
```

## Peer Teaching ( $2 \times 1$ minutes to think, explain to your neighbour and vote)

Cost of solving multiple right-hand sides. Having calculated an LU factorization, what is the work required for one forward-backward substitution?

$\leftarrow \mathcal{O}(n)$

$\uparrow \mathcal{O}(n^2)$

$\rightarrow \mathcal{O}(n^3)$

### 3.3 Special types of linear systems (NAM 1.5, H 2.5.0)

The shape of the matrix plays an essential role and can be exploited to reduce the memory and the work required by the solver.

**Symmetric**  $a_{ij} = a_{ji}$  or  $\mathbf{A}^T = \mathbf{A}$

Divide memory and work by 2. The inverse is also symmetric.

**Hermitian**  $a_{ij} = \bar{a}_{ji}$  or  $\mathbf{A}^H = \mathbf{A}$

Analogous to symmetric for complex numbers.

**Triangular**  $a_{ij} = 0, \forall i < j$  (lower) or  $i > j$  (upper)

Solve with  $\mathcal{O}(n^2)$  operations using forward or backward substitution.

The inverse has exactly the same shape.

**Banded**  $a_{ij} = 0, \forall |i - j| > \beta$

Store  $\mathcal{O}(\beta n)$  numbers and factorize with  $\mathcal{O}(\beta^2 n)$  operations.

**Tridiagonal**  $a_{ij} = 0, \forall |i - j| > 1$

Store  $\mathcal{O}(n)$  numbers and factorize with  $\mathcal{O}(n)$  operations – see below!

**Sparse** Most of the entries  $a_{ij}$  are zeros

Try to use iterative methods – check later in this course.

**Diagonally dominant**  $|a_{ii}| \geq \sum_{i \neq j} |a_{ij}|$

No need to check for pivoting in the LU factorization.

### 3.4 Tridiagonal systems (NAM 1.6, H 2.5.3)

Algorithm to solve tridiagonal systems with  $\mathcal{O}(n)$  operations

$$\mathbf{Ax} = \begin{bmatrix} d_1 & p_1 & 0 & \dots & 0 \\ q_1 & d_2 & p_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & q_{n-2} & d_{n-1} & p_{n-1} \\ 0 & \dots & 0 & q_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ \vdots \\ \vdots \\ b_n \end{bmatrix} = \mathbf{b}$$

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 \\ m_1 & 1 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & m_{n-2} & 1 & 0 \\ 0 & \dots & 0 & m_{n-1} & d_n \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} r_1 & p_1 & 0 & \dots & 0 \\ 0 & r_2 & p_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & r_{n-1} & p_{n-1} \\ 0 & \dots & \dots & 0 & r_n \end{bmatrix}$$

Download tridia from the course webpages:

```
function x = tridia(d,p,q,b)
% Computes solution x to a tridiagonal system with diagonal d,
% superdiagonal p, subdiagonal q and right hand side vector b
n=length(b); r=d; g=b; x=b;
for i=2:n
    i1=i-1; m=q(i1)/r(i1); % Gauss elimination
    r(i)=d(i)-m*p(i1);
    g(i)=b(i)-m*g(i1); % Forward substitution
end
x(n)=g(n)/r(n); % Backward substitution
for i=n-1:-1:1, x(i)=(g(i)-p(i)*x(i+1))/r(i); end
```

Store only three diagonals  $q, d, p$ , i.e.  $3n - 2 \sim \mathcal{O}(n)$  numbers

Eliminate only non zero subdiagonal  $q$ , i.e.  $n - 1 \sim \mathcal{O}(n)$  divisions

Forward and backward substitution each require  $\mathcal{O}(n)$  multiplications

Total  $3\mathcal{O}(n) \sim \mathcal{O}(n)$  operations (not counting  $+$ ,  $-$ , addressing, etc)

Note that tridia is not a standard Matlab function; later we will use the command sparse instead.

## Solve a tridiagonal system with tridia

$$\mathbf{Ax} = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 2 & 5 & 2 & 0 & 0 \\ 0 & 2 & 5 & 2 & 0 \\ 0 & 0 & 2 & 5 & 2 \\ 0 & 0 & 0 & 2 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 5.7 \\ 11.5 \\ 1.6 \\ 6.8 \\ 9.4 \end{bmatrix} = \mathbf{b}$$

Having copied tridia into your working directory, in Matlab type:

```
>> dia=[1 5 5 5 5]';  
>> sup=[2 2 2 2]; sub=sup;  
>> b=[5.7 11.5 1.6 6.8 9.4]';  
>> x=tridia(dia,sup,sub,b)
```

```
x = 1.5000    2.1000   -1.0000    1.2000    1.4000
```

## Inverse of a symmetric tridiagonal system by Edgar Asplund (KTH, ca 1960)

In general, the columns of any inverse matrix  $\mathbf{C} = \mathbf{A}^{-1}$  can be calculated by solving  $n$  linear systems taking the unit vectors  $\mathbf{e}_i$  as right hand side

```
>> n=size(a,1); c=zeros(size(a));  
>> for j=1:n  
    e=zeros(1,n); e(j)=1; c(:,j)=tridia(dia,sup,sub,e)';  
end  
>> c  
c = 341   -170    84   -40    16  
   -170    85   -42    20    -8  
     84   -42    21   -10     4  
   -40    20   -10     5    -2  
     16    -8     4    -2     1
```

The inverse is now a full matrix and requires  $n\mathcal{O}(n) = \mathcal{O}(n^2)$  operations. Asplund showed that for symmetric tridiagonal systems, the inverse can be calculated more efficiently as follows:

1. calculate the first and last column of the inverse by solving two systems  $\mathbf{Ac}_1 = \mathbf{e}_1$  and  $\mathbf{Ac}_n = \mathbf{e}_n$ ;
2. the first and last rows are obtained by symmetry
3. use the first row and last column to calculate the upper triangular matrix from the property  $c_{ik}c_{1n} = c_{in}c_{1k}$
4. complete the lower triangular part by symmetry.

$$\mathbf{C} = \begin{bmatrix} 341 & -170 & 84 & -40 & \boxed{16} \\ -170 & \times & \times & \times & -8 \\ 84 & \times & \times & \times & 4 \\ -40 & \times & \times & \times & -2 \\ 16 & -8 & 4 & -2 & 1 \end{bmatrix}, \quad c_{15} = 16, \quad \begin{cases} c_{22}c_{15} = -8 \cdot (-170) \\ c_{23}c_{15} = -8 \cdot (84) \\ c_{24}c_{15} = -8 \cdot (-40) \\ c_{33}c_{15} = 4 \cdot (84) \\ c_{34}c_{15} = 4 \cdot (-40) \\ c_{44}c_{15} = -2 \cdot (-40) \end{cases}$$

$$\mathbf{C} = \begin{bmatrix} 341 & -170 & 84 & -40 & 16 \\ -170 & 85 & -42 & 20 & -8 \\ 84 & \times & 21 & -10 & 4 \\ -40 & \times & \times & 5 & -2 \\ 16 & -8 & 4 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 341 & -170 & 84 & -40 & 16 \\ -170 & 85 & -42 & 20 & -8 \\ 84 & -42 & 21 & -10 & 4 \\ -40 & 20 & -10 & 5 & -2 \\ 16 & -8 & 4 & -2 & 1 \end{bmatrix}$$

Peer Teaching (2 × 1 minutes to think, explain to your neighbour and vote)

Number of operation for Asplund's inverse. How does Asplund's method scale with the rank of the tridiagonal matrix?

←  $\mathcal{O}(n)$                       ↑  $\mathcal{O}(n \log n)$                       →  $\mathcal{O}(n^2)$                       ↓ none of these

### 3.5 Vector and matrix norms, condition number (NAM 1.7, H 2.3)

#### Norm of a vector

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

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

$$\|\mathbf{x}\|_\infty = \lim_{p \rightarrow \infty} \|\mathbf{x}\|_p = \max_{1 \leq i \leq n} |x_i|$$

Properties:  $\|\mathbf{x}\| > 0$  if  $x \neq 0$   
 $\|\alpha \mathbf{x}\| = |\alpha| \|\mathbf{x}\|$   $\alpha$  scalar  
 $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$

#### Norms of a matrix

$$\|\mathbf{A}\| = \max_{\mathbf{x} \neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}$$

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|$$

Properties:  $\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\|$   
 $\|\mathbf{A}\mathbf{B}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$

#### Condition number of a matrix

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$$

### Illustration with the example above

$$\|\mathbf{b}\|_2 \simeq 17.38, \|\mathbf{b}\|_\infty = 11.5$$

$$\|\mathbf{A}\|_\infty = 9, \|\mathbf{A}^{-1}\|_\infty = 651$$

$\text{cond}(\mathbf{A}) = 9 \cdot 651 = 5859 \approx 6 \cdot 10^3$ , showing that the linear system is quite ill conditioned!

### Experimental perturbation of a linear system

Input parameters with one decimal, i.e. an uncertainty of  $\pm 0.05$ .

Idea: perturb the right hand side vector by that amount:

```
>> dia=[1 5 5 5 5]; sup=[2 2 2 2];sub=sup;
>> b=[5.7 11.5 1.6 6.8 9.4]; x=tridia(dia,sup,sub,b)
x = 1.5000    2.1000   -1.0000    1.2000    1.4000

>> b1=[5.75 11.45 1.65 6.75 9.45]; x1=tridia(dia,sup,sub,b1)
x1 = 34.0500  -14.1500    7.0500   -2.6500    2.9500

>> b2=[5.65 11.55 1.55 6.75 9.45]; x2=tridia(dia,sup,sub,b2)
x2 = -25.4500  15.5500   -7.6500    4.3500    0.1500
```

Not a single digit is right! This is the result of the large condition number which can also be calculated from the perturbations above:

$$\kappa = \frac{\|\text{rel error out}\|}{\|\text{rel error in}\|} \approx \frac{\|\mathbf{x}_1 - \mathbf{x}\|/\|\mathbf{x}\|}{\|\mathbf{b}_1 - \mathbf{b}\|/\|\mathbf{b}\|} \simeq \frac{32.55/2.1}{0.05/11.5} \simeq 3600 \approx 4 \cdot 10^3$$

This is consistent with the theoretical value  $6 \cdot 10^3$  calculated previously.

Software for linear systems in FORTRAN, C, Java  $\rightarrow$  <http://www.netlib.org/>

LAPACK: Linear Algebra PACKage contains hundreds of routines for special types of matrices, e.g. sgetrf (factorize), sgetrs (solve)

LINPACK: has been replaced by LAPACK, but is still often being referred to for benchmarks, e.g. sgefa (factorize), sgesl (solve)

BLAS 1-3: Basic Linear ALgebra Subprograms  $\mathcal{O}(n) - \mathcal{O}(n^3)$  operations

PETSc: is specialized more for iterative methods, but has also a few direct solvers for parallel computing  $\rightarrow$  <http://www-fp.mcs.anl.gov/petsc>