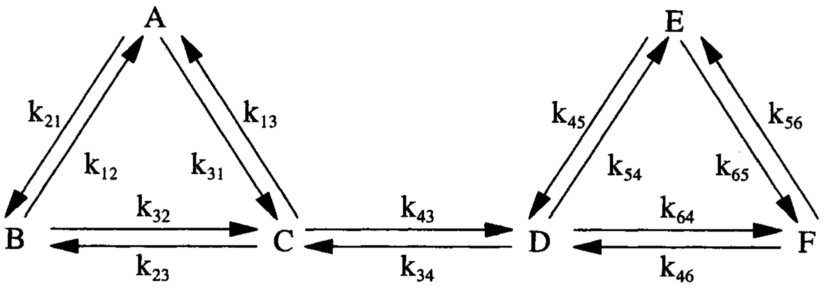
Linear systems of equations

Motivating example: Chemical reaction network

- We know:
  - initial concentrations of all components $A_0, B_0, C_0, D_0, E_0, F_0$
  - kinetic rate constants of reactions $k_{12}, \ldots, k_{65}$
- We want to compute
  - steady-state concentrations of all components *A, B, C, D, E, F*
- We model all reaction rates as 1st order kinetics



Constantinides & Mostoufi, Numerical methods for Chemical Engineers with MATLAB applications, p 139

Chemical reaction network: equations

- Mass balances:

$$dA/dt = \boxed{k_{12}B + k_{13}C} - \boxed{k_{21}A - k_{31}A}$$

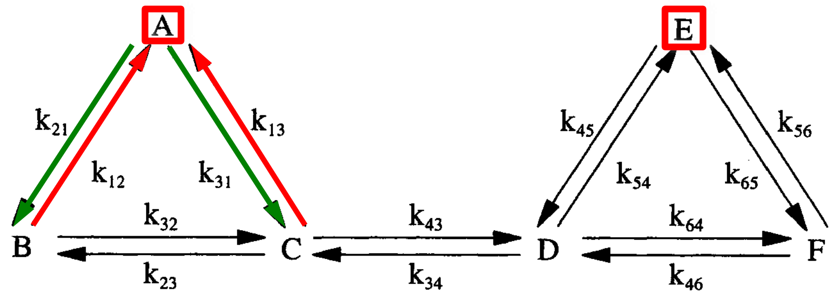
$$dB/dt = k_{21}A + k_{23}C - k_{12}B - k_{32}B$$

$$dC/dt = k_{31}A + k_{32}B + k_{34}D - k_{13}C - k_{23}C - k_{43}C$$

$$dD/dt = k_{43}C + k_{45}E + k_{46}F - k_{34}D - k_{54}D - k_{64}D$$

$$dE/dt = k_{54}D + k_{56}F - k_{45}E - k_{65}E$$

$$dF/dt = k_{65}E + k_{64}D - k_{46}F - k_{56}F$$



Constantinides & Mostoufi, Numerical methods for Chemical Engineers with MATLAB applications, p 139

- Conservation of the species:

$$dA/dt+dB/dt+dC/dt+dD/dt+dE/dt+dF/dt = 0$$

$$A + B + C + D + E + F = const$$

- *The data*

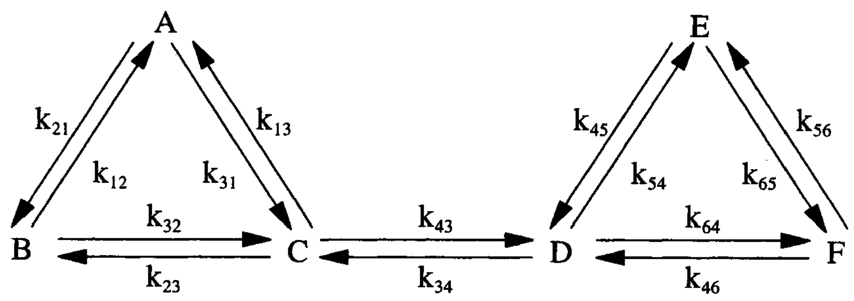
$A_0=1$; $B_0=0$; $C_0=0$; $D_0=0$; $E_0=1$; $F_0=0$;

$k_{21}=0.2$; $k_{31}=0.1$; $k_{32}=0.1$; $k_{34}=0.1$; $k_{54}=0.05$;

$k_{64}=0.2$; $k_{65}=0.1$; $k_{12}=0.1$; $k_{13}=0.05$; $k_{23}=0.05$;

$k_{43}=0.2$; $k_{45}=0.1$; $k_{46}=0.2$; $k_{56}=0.1$; ;

Chemical reaction network: steady state



Constantinides & Mostoufi, Numerical methods for Chemical Engineers with MATLAB applications, p 139

- At the steady state, the studied system of ODE equations reduces to :

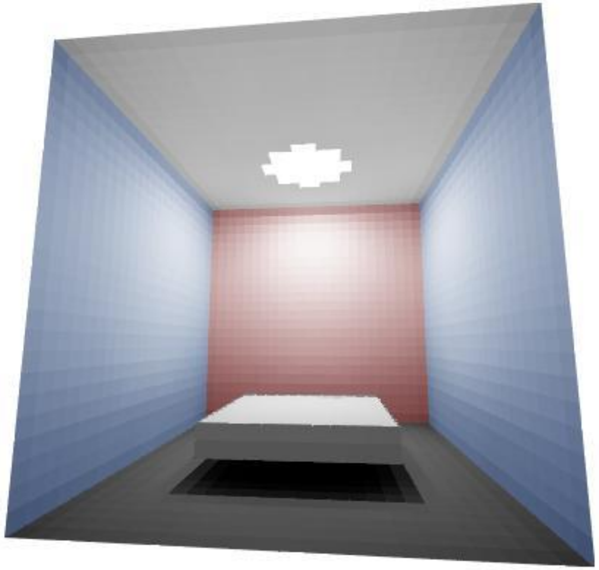
$$
\begin{aligned}
A + B + C + D + E + F &= 2 \\
0.2A - 0.2B + 0.05C &= 0 \\
0.1A + 0.1B - 0.3C + 0.1D &= 0 \\
0.2C - 0.35D + 0.1E + 0.2F &= 0 \\
0.05D - 0.2E + 0.1F &= 0 \\
0.2D + 0.1E - 0.3F &= 0
\end{aligned}
$$

Another motivating example: Radiosity methods

- Solve the diffusion of light for a room
- The radiosity of a pixel i, B_i, can be computed as a solution of the following linear system of n equations for a given 3D model with n pixels:

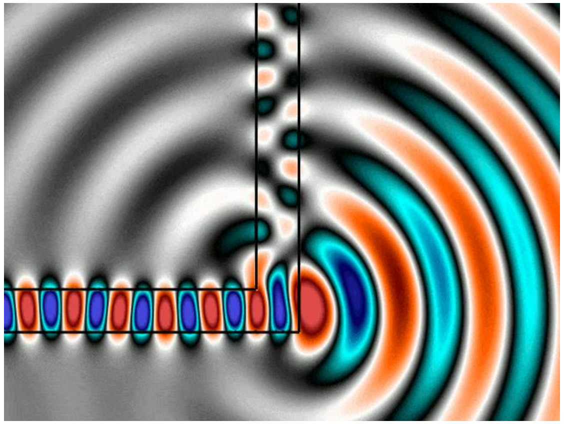
$$B_i = E_i + \rho_i \sum_j B_j F_{i,j}$$

where: B_i is radiosity of pixel i, E_i is the emission of pixel i, ρ_i is the reflectivity of pixel i, and $F_{i,j}$ is the fraction of energy leaving pixel i arriving directly at j



Linear systems of equations (LSE)

- Linear systems of equations are ubiquitous in other numerical problems:
  - Interpolation (e.g., construction of the cubic spline interpolant)
  - Boundary value problems (BVP) of ordinary differential equations (ODE)
  - Partial differential equations (PDE)
  - …



https://vimeo.com/160322285

Linear systems of equations (LSE)

- Find:
  - vector $x \in R^n$ such that it satisfies $Ax = b$

  where:
  - System matrix $A \in R^{n,n}$ with coefficients a_{ij}, $i,j=1..n$
  - Right hand side vector (RHS) $b \in R^n$ with coefficients b_j, $j=1..n$

- System matrix A can be:
  - Full matrix
  - Sparse matrix, that can have a sparsity pattern
    - Diagonal, tridiagonal, band-matrix, block-diagonal

Linear systems of equations (LSE)

- Solution methods:
  - Direct methods
    - provide the exact solution in a finite number of operations (Gauss elimination, etc.)
  - Iterative methods
    - start with an approximate (guess) solution and iterate to approach to the exact solution
- In all methods, we assume that matrix A is :
  - regular/invertible <-> full column rank <-> full row rank <-> non-zero determinant

Gauss elimination method

- LSE of *n* equations and *n* unknowns

$$Ax = b \quad \Leftrightarrow \quad \begin{bmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ii} & \cdots & a_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{ni} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}$$

- **Phase 1**: transform to a upper triangular LSE

$$\begin{bmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & a'_{ii} & \cdots & a'_{in} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b'_i \\ \vdots \\ b'_n \end{bmatrix}$$

- **Phase 2**: solve using the back substitution method

Phase 1: forward elimination

pivotal element

- LSE

$$
\begin{aligned}
2x_1 &- x_2 &+ x_3 &+ 2x_4 &= 1 \qquad & \times\ -0.5,\ \times\ -1.5,\ \times\ -0.5 \\
x_1 &+ 4x_2 &+ 2x_3 &- 4x_4 &= -2 \qquad & + \\
3x_1 &+ x_2 &- x_3 &- 10x_4 &= 5 \qquad & + \\
x_1 &+ x_2 &- x_3 &- 6x_4 &= 3 \qquad & +
\end{aligned}
$$

Phase 1: forward elimination

- LSE

$$
\begin{aligned}
2x_1 & - x_2 & + x_3 & + 2x_4 & = & \; 1 \\
0 & + 4.5x_2 & + 1.5x_3 & - 5x_4 & = & \; -2.5 \quad \times -1/3 \quad \times -5/9 \\
0 & + 2.5x_2 & - 2.5x_3 & - 13x_4 & = & \; 3.5 \quad + \\
0 & + 1.5x_2 & - 1.5x_3 & - 7x_4 & = & \; 2.5 \quad +
\end{aligned}
$$

pivotal element

Eliminated elements

Phase 1: forward elimination

- LSE

$2x_1$	$- x_2$	$+ x_3$	$+ 2x_4$	=	1	
0	$+ 4.5x_2$	$+ 1.5x_3$	$- 5x_4$	=	-2.5	
0	0	$- 3\frac{1}{3} x_3$	$- 10\frac{2}{9} x_4$	=	$4\frac{8}{9}$	$\times_{-3/5}$
0	0	$- 2x_3$	$- 5\frac{1}{3} x_4$	=	$3\frac{1}{3}$	+

pivotal element

Eliminated elements

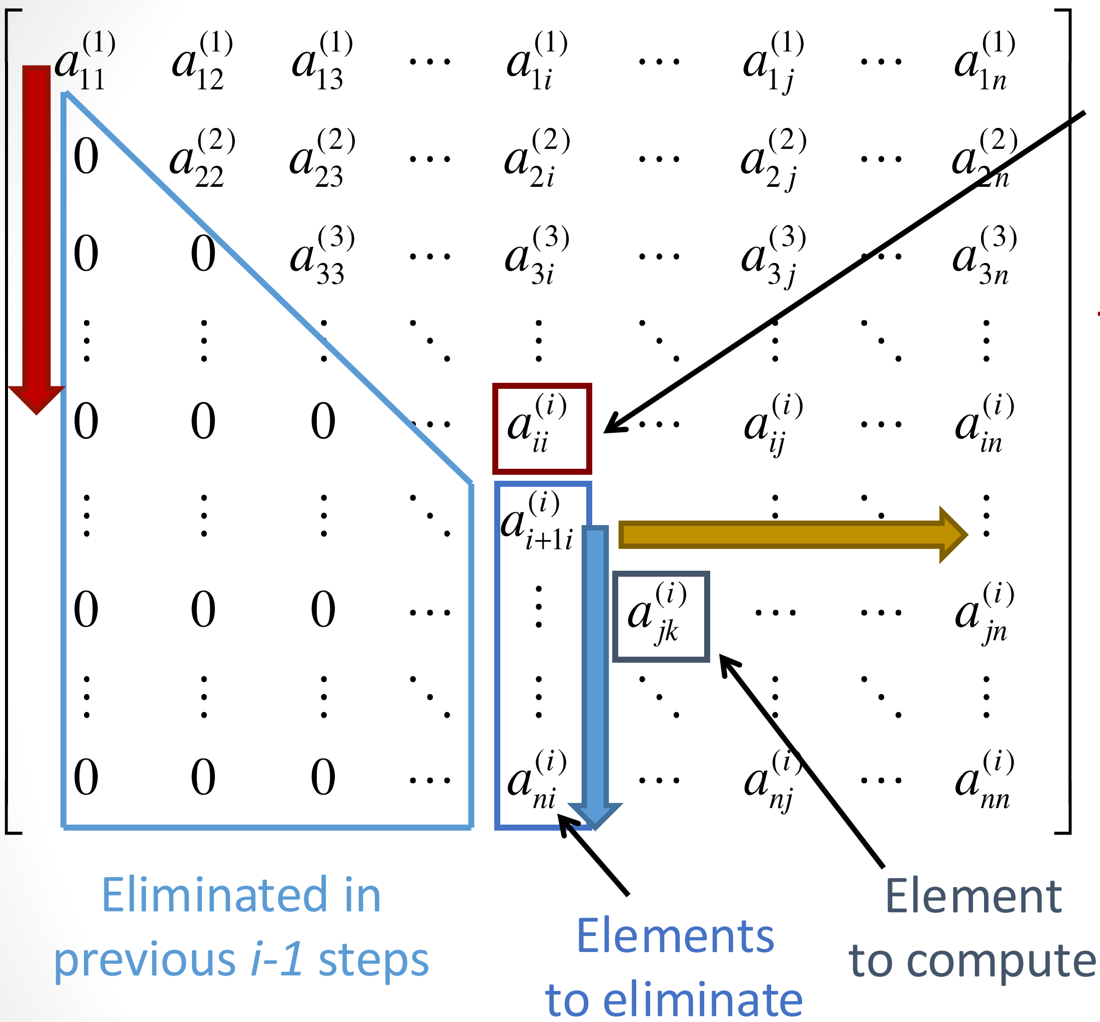
Phase 1: forward elimination

- LSE

$2x_1$	$- x_2$	$+ x_3$	$+ 2x_4$	$=$	1
0	$+ 4.5x_2$	$+ 1.5x_3$	$- 5x_4$	$=$	-2.5
0	0	$- 3\frac{1}{3} x_3$	$- 10\frac{2}{9} x_4$	$=$	$4\frac{8}{9}$
0	0	0	$0.8 x_4$	$=$	0.4

Eliminated elements

Forward elimination: algorithm




```
for i in range(n - 1): # Pivoting rows
    for j in range(i + 1, n):
        elFact = -a[j, i] / a[i, i]
        for k in range(i, n):
            a[j, k] += elFact * a[i, k]
        b[j] += elFact * b[i]
return a, b
```

Phase 2: back substitution

- LSE

$2x_1$	$- x_2$	$+ x_3$	$+ 2x_4$	=	1
0	$+ 4.5x_2$	$+ 1.5x_3$	$- 5x_4$	=	-2.5
0	0	$- 3\frac{1}{3} x_3$	$- 10\frac{2}{9} x_4$	=	$4 \frac{8}{9}$
0	0	0	$0.8 x_4$	=	*0.4*

$$x_4 = 0.5$$

Phase 2: back substitution

- LSE

$$
\begin{array}{ccccccc}
2x_1 & -\ x_2 & +\ x_3 & & = & 0 \\
& +\ 4.5x_2 & +\ 1.5x_3 & & = & 0 \\
& & -\ 3\tfrac{1}{3}\ x_3 & & = & 10 \\
& & & 0.8\ x_4 & = & \mathit{0.4}
\end{array}
$$

$$
\boxed{x_3 \quad = \quad -3}
$$

Phase 2: back substitution

- LSE

$$2x_1 \quad - x_2 \quad = \quad 3$$

$$+ 4.5x_2 \quad = \quad 4.5$$

$$- 3⅓\, x_3 \quad = \quad 10$$

$$0.8\, x_4 \quad = \quad 0.4$$

$$x_2 \quad = \quad 1$$

Phase 2: back substitution

- LSE

$$2x_1 = 4$$

$$+ 4.5x_2 = 4.5$$

$$- 3\tfrac{1}{3}\, x_3 = 10$$

$$0.8\, x_4 = 0.4$$

$$x_1 = 2$$

Back substitution: algorithm

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & a_{n-1n-1}^{(n-1)} & a_{n-1n}^{(n-1)} \\ 0 & 0 & 0 & 0 & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_{n-1}^{(n-1)} \\ b_n^{(n)} \end{bmatrix}$$

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}}$$

$$x_{n-1} = \frac{1}{a_{n-1n-1}^{(n-1)}} \left[b_{n-1}^{(n-1)} - a_{n-1n}^{(n-1)} x_n \right]$$

$$x_i = \frac{1}{a_{ii}^{(i)}} \left[b_i^{(i)} - \sum_{k=i+1}^{n} a_{ik}^{(i)} x_k \right] \quad i = n-1, n-2, \ldots, 1$$

Gauss elimination algorithm: cost

- Nº of arithmetic operations required to solve the LSE
  - Additions + subtractions: $\frac{n(n-1)(2n+5)}{6}$
  - Multiplications + divisions $\frac{n(n^2+3n-1)}{3}$
  - The total cost is around $\frac{2}{3}n^3$
  - Compare with Cramer's rule (*n!*):
    - on a gigaFLOPS machine, and n=50 we need to run for 10^{46} years
    - in comparison, Gauss elimination requires $8.3*10^{-5}$ secs

Pivoting

- A simple example:

$$\begin{bmatrix} 0.001 & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \times\ \text{-1000} \quad +$$

$$\begin{bmatrix} 0.001 & 1 \\ 0 & -999 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -998 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1000(1 - 998/999) \\ 998/999 \end{bmatrix} \xrightarrow[\text{intervention}]{\text{human}} \begin{bmatrix} 1000/999 \\ 998/999 \end{bmatrix}$$

- Assume now that we are using 4-digit arithmetic

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1000(1 - 0.998) \\ 0.998 \end{bmatrix} = \begin{bmatrix} 2 \\ 0.998 \end{bmatrix}$$

Pivoting

- Pivoting (swapping the rows)

$$\begin{bmatrix} 1 & 1 \\ 0.001 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

× -0.001
+

$$\begin{bmatrix} 1 & 1 \\ 0 & 0.999 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0.998 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 - 998/999 \\ 998/999 \end{bmatrix} \xrightarrow{\text{human intervention}} \begin{bmatrix} 1000/999 \\ 998/999 \end{bmatrix}$$

- Assume now that we are using 4-digit arithmetic

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 - 0.998 \\ 0.998 \end{bmatrix} = \begin{bmatrix} 1.002 \\ 0.998 \end{bmatrix}$$

Partial (row) pivoting

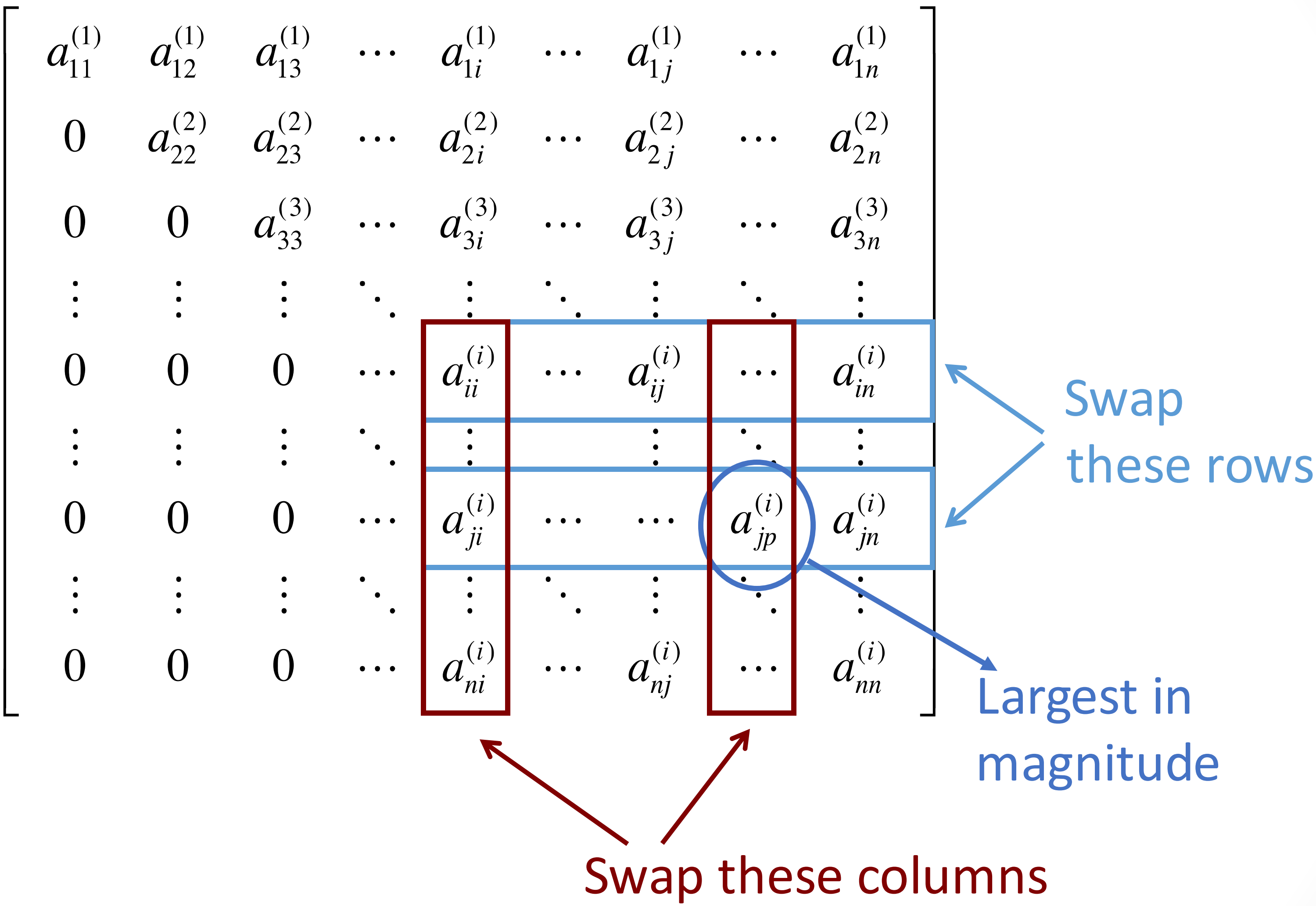
$$
\begin{bmatrix}
a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1i}^{(1)} & \cdots & a_{1j}^{(1)} & \cdots & a_{1n}^{(1)} \\
0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2i}^{(2)} & \cdots & a_{2j}^{(2)} & \cdots & a_{2n}^{(2)} \\
0 & 0 & a_{33}^{(3)} & \cdots & a_{3i}^{(3)} & \cdots & a_{3j}^{(3)} & \cdots & a_{3n}^{(3)} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & a_{ii}^{(i)} & \cdots & a_{ij}^{(i)} & \cdots & a_{in}^{(i)} \\
\vdots & \vdots & \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & a_{ji}^{(i)} & a_{jk}^{(i)} & \cdots & \cdots & a_{jn}^{(i)} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & a_{ni}^{(i)} & \cdots & a_{nj}^{(i)} & \cdots & a_{nn}^{(i)}
\end{bmatrix}
$$

Swap these rows

Largest in magnitude

- Swapping rows only changes the order of equations

Full pivoting



- When swapping columns, remember to swap the solution vector, i.e., x_p becomes x_i and vice versa!

Pivoting recapitulation

- Errors due to finite-precision arithmetic are introduced in each arithmetic operation
- Introduced errors propagate
- When the pivotal element, *a(i,i)*, is very small, the multiplying factor in the process of elimination, *elFact=a(j,i)/a(i,i)*, will be very large
- In a limiting case, when *a(i,i)* is zero, one has to perform division by zero!
- Solution: swap rows/columns or both rows and columns to use a pivotal element with the largest magnitude

LU decomposition

$$\underbrace{\begin{bmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ii} & \cdots & a_{in} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{ni} & \cdots & a_{nn} \end{bmatrix}}_{\textbf{A}} = \underbrace{\begin{bmatrix} l_{11} & & & & \\ \vdots & \ddots & & 0 & \\ l_{i1} & \cdots & l_{ii} & & \\ \vdots & & \vdots & \ddots & \\ l_{n1} & \cdots & l_{ni} & \cdots & l_{nn} \end{bmatrix}}_{\textbf{L}} \underbrace{\begin{bmatrix} 1 & \cdots & u_{1i} & \cdots & u_{1n} \\ & \ddots & \vdots & & \vdots \\ & & 1 & \cdots & u_{in} \\ & 0 & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}}_{\textbf{U}}$$

$$\left.\begin{aligned} Ax &= b \\ L\underbrace{Ux}_{y} &= b \end{aligned}\right\}$$

thus, we are solving instead:

$Ly = b$ - forward substitution

$Ux = y$ - back substitution

LU decomposition: example

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

$$l_{11} = a_{11} \quad l_{21} = a_{21} \quad l_{31} = a_{31}$$

$$l_{11}u_{12} = a_{12} \Rightarrow u_{12} = \frac{a_{12}}{l_{11}}$$

$$l_{11}u_{13} = a_{13} \Rightarrow u_{13} = \frac{a_{13}}{l_{11}}$$

$$l_{31}u_{13} + l_{32}u_{23} + l_{33} = a_{33}$$

$$\Leftarrow$$

$$l_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23}$$

$$l_{21}u_{12} + l_{22} = a_{22} \Rightarrow l_{22} = a_{22} - l_{21}u_{12}$$

$$l_{31}u_{12} + l_{32} = a_{32} \Rightarrow l_{32} = a_{32} - l_{31}u_{12}$$

$$l_{21}u_{13} + l_{22}u_{23} = a_{23} \Rightarrow u_{23} = \frac{a_{23} - l_{21}u_{13}}{l_{22}}$$

LU decomposition: algorithm

$$\begin{bmatrix} a_{11} & \cdots & a_{1i} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots & & \vdots \\ a_{i1} & \cdots & a_{ii} & \cdots & a_{in} \\ \vdots & & \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{ni} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & & & & 0 \\ \vdots & \ddots & & & \\ l_{i1} & \cdots & l_{ii} & & \\ \vdots & & \vdots & \ddots & \\ l_{n1} & \cdots & l_{ni} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & \cdots & u_{1i} & \cdots & u_{1n} \\ & \ddots & \vdots & & \vdots \\ & & 1 & \cdots & u_{in} \\ & 0 & & \ddots & \vdots \\ & & & & 1 \end{bmatrix}$$

1. Compute 1st column of L as: $l_{i1} = a_{i1}$

2. Compute 1st row of U as: $u_{1j} = \dfrac{a_{1j}}{l_{11}}$

3. Compute sequentially columns of L and rows of U as:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \qquad j \le i, \quad i = 1,2,\ldots,n$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}}{l_{ii}} \qquad i \le j, \quad j = 2,3,\ldots,n$$

LU decomposition: remarks

- Solving a LSE using LU decomposition requires
  - factorization of A as LU
  - forward substitution
  - backward substitution

  $\frac{2}{3}n^3 + n^2 + n^2$ total operations

- For any number of RHS vectors (vectors *b)* we need to perform LU decomposition only once!

LU decomposition: remarks

- Solving a LSE using LU decomposition requires
  - factorization of A as LU
  - forward substitution
  - backward substitution

$$\frac{2}{3}n^3 + n^2 + n^2 \quad \text{total operations}$$

- For any number of RHS vectors (vectors *b)* we need to perform LU decomposition only once!
- Each element of A matrix is used only once to compute the corresponding element of L or U matrix –> so L and U can be stored in A
- Partial pivoting is sufficient, and widely implemented

Cholesky decomposition

- For symmetric matrices, we can use the following decomposition:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

- Since $A=LL^T$, i.e. $U=L^T$, we need to compute only L
- Half as many operations as LU
- A must be symmetric, positive definite, i.e., $x^TAx > 0$, for all $x \neq 0$
- Recommended method for solving symmetric positive definite systems

Cholesky decomposition: example

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

$$l_{11}^2 = a_{11} \quad \Rightarrow \quad l_{11} = \sqrt{a_{11}}$$

$$l_{11} l_{21} = a_{12} \quad \Rightarrow \quad l_{21} = \frac{a_{12}}{l_{11}}$$

$$l_{11} l_{31} = a_{13} \quad \Rightarrow \quad l_{31} = \frac{a_{13}}{l_{11}}$$

$$l_{21}^2 + l_{22}^2 = a_{22} \quad \Rightarrow \quad l_{22} = \sqrt{a_{22} - l_{21}^2}$$

$$l_{21} l_{31} + l_{22} l_{32} = a_{23} \quad \Rightarrow \quad l_{32} = \frac{a_{23} - l_{21} l_{31}}{l_{22}}$$

$$l_{31}^2 + l_{32}^2 + l_{33}^2 = a_{33} \quad \Rightarrow \quad l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2}$$

Cholesky decomposition: algorithm

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}$$

$$\left.\begin{aligned} l_{11} &= \sqrt{a_{11}} \\ l_{22} &= \sqrt{a_{22} - l_{21}^{\ 2}} \\ l_{33} &= \sqrt{a_{33} - l_{31}^{\ 2} - l_{32}^{\ 2}} \end{aligned}\right\} \Rightarrow \quad l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2} \quad i = 1 \ldots n$$

$$\left.\begin{aligned} l_{21} &= a_{12}/l_{11} \\ l_{31} &= a_{13}/l_{11} \\ l_{32} &= (a_{23} - l_{21}l_{31})/l_{22} \end{aligned}\right\} \Rightarrow \quad l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik}l_{jk}}{l_{ii}} \quad j = i+1 \ldots n$$

- The algorithm would fail if it would be required to take the square root of a negative number
- Therefore, another condition on *A* -> positive definiteness

Recapitulation on direct methods

- Idea: transform the original LSE into a simpler, equivalent LSE that is ‘easier’ to solve
- Easy to solve LSE: diagonal, upper or lower triangular
- Finitely many elementary operations – number depending on *n*
- Gauss elimination methods, LU decomposition, Cholesky decomposition
- Some well-known methods such as the Cramer’s rule or the explicit computation of the inverse of *A* are computationally prohibitive.

Sparse systems

- For sparse systems

$$\begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} & 0 \\ 0 & a_{22} & 0 & a_{24} & 0 \\ a_{31} & 0 & a_{33} & 0 & a_{35} \\ a_{41} & a_{42} & 0 & a_{44} & 0 \\ 0 & 0 & a_{53} & 0 & a_{55} \end{bmatrix}$$

- During the process of forward elimination there is a fill-in effect

$$\begin{bmatrix} a_{11} & 0 & a_{13} & a_{14} & 0 \\ 0 & a_{22} & a'_{23} & a_{24} & 0 \\ 0 & 0 & a'_{33} & a'_{34} & a'_{35} \\ 0 & 0 & 0 & a'_{44} & a'_{45} \\ 0 & 0 & 0 & 0 & a'_{55} \end{bmatrix}$$

non-zero, fill-in, terms

- Fill-in terms can considerably increase storage requirements
- For large and sparse systems - use iterative methods instead, or direct methods adapted for sparse systems

Iterative methods - Main idea

- The original system

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

- Convert to

$$\left.\begin{aligned} x_1 &= \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 \\ x_2 &= \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1 - \frac{a_{23}}{a_{22}}x_3 \\ x_3 &= \frac{b_3}{a_{33}} - \frac{a_{31}}{a_{33}}x_1 - \frac{a_{32}}{a_{33}}x_2 \end{aligned}\right\} \quad Cx + d$$

- Solve iteratively

$$x^{(k)} = Cx^{(k-1)} + d$$

Iterative methods for solving LSE

- If LSE are of a very large size, the computational efforts required for applying direct methods are prohibitively expensive
- Iterative techniques for LSE (Relaxation methods)
  - Jacobi method
  - Gauss-Seidel method
  - Successive Over Relaxation (SOR) method
- One iteration step typically costs $\sim n$ arithmetic operations in case of a sparse matrix
- The cost depends on how many iteration steps are required to obtain a certain accuracy.

Jacobi method

$$x_1^{(k)} = -\frac{a_{12}}{a_{11}} x_2^{(k-1)} - \frac{a_{13}}{a_{11}} x_3^{(k-1)} + \frac{b_1}{a_{11}}$$

$$x_2^{(k)} = -\frac{a_{21}}{a_{22}} x_1^{(k-1)} - \frac{a_{23}}{a_{22}} x_3^{(k-1)} + \frac{b_2}{a_{22}}$$

$$x_3^{(k)} = -\frac{a_{31}}{a_{33}} x_1^{(k-1)} - \frac{a_{32}}{a_{33}} x_2^{(k-1)} + \frac{b_3}{a_{33}}$$

- Needs an initial solution vector
- Compute new values of solution, $x^{(k)}$, using the values from the previous iteration, $x^{(k-1)}$

Jacobi method: example

- Apply the Jacobi method to solve the system:

$$
\begin{aligned}
5x_1 - x_2 &= -7 \\
x_1 + 4x_2 &= 7
\end{aligned}
$$

- we got

$$x_1^{(k)} = -\frac{a_{12}}{a_{11}} x_2^{(k-1)} + \frac{b_1}{a_{11}} = \frac{1}{5} x_2^{(k-1)} - \frac{7}{5}$$

$$x_2^{(k)} = -\frac{a_{21}}{a_{22}} x_1^{(k-1)} + \frac{b_2}{a_{22}} = -\frac{1}{4} x_1^{(k-1)} + \frac{7}{4}$$

- if we take the starting point (x1, x2)=(0, 0) and perform iterations

$$x_1^{(1)} = \frac{1}{5} \times 0 - \frac{7}{5} = -\frac{7}{5} \qquad x_2^{(1)} = -\frac{1}{4} \times 0 + \frac{7}{4} = \frac{7}{4}$$

$$\Rightarrow \quad x_1^{(2)} = \frac{1}{5} \times \frac{7}{4} - \frac{7}{5} = -\frac{21}{20} \qquad x_2^{(2)} = -\frac{1}{4} \times \frac{7}{5} + \frac{7}{4} = \frac{42}{20}$$

$$\Rightarrow \quad \ldots \quad \Rightarrow \quad x_1^{(6)} = -1.0001 \qquad x_2^{(6)} = 2.0002$$

Gauss-Seidel method

- As soon as an element from *x* is updated, it is used subsequently

$$x_1^{(k)} = -\frac{a_{12}}{a_{11}} x_2^{(k-1)} - \frac{a_{13}}{a_{11}} x_3^{(k-1)} + \frac{b_1}{a_{11}}$$

$$x_2^{(k)} = -\frac{a_{21}}{a_{22}} x_1^{(k)} - \frac{a_{23}}{a_{22}} x_3^{(k-1)} + \frac{b_2}{a_{22}}$$

$$x_3^{(k)} = -\frac{a_{31}}{a_{33}} x_1^{(k)} - \frac{a_{32}}{a_{33}} x_2^{(k)} + \frac{b_3}{a_{33}}$$

- Typically converges more rapidly than the Jacobi method
- More difficult to parallelize

Gauss-Seidel method: example

- System:

$$5x_1 - x_2 = -7$$
$$x_1 + 4x_2 = 7$$

- Gauss-Seidel iteration:

$$x_1^{(k)} = -\frac{a_{12}}{a_{11}} x_2^{(k-1)} + \frac{b_1}{a_{11}} = \frac{1}{5} x_2^{(k-1)} - \frac{7}{5}$$
$$x_2^{(k)} = -\frac{a_{21}}{a_{22}} x_1^{(k)} + \frac{b_2}{a_{22}} = -\frac{1}{4} x_1^{(k)} + \frac{7}{4}$$

- if we take the starting point (x1, x2)=(0, 0) and perform iterations:

$$x_1^{(1)} = \frac{1}{5} \times 0 - \frac{7}{5} = -\frac{7}{5}$$
$$x_2^{(1)} = -\frac{1}{4} \times -\frac{7}{5} + \frac{7}{4} = \frac{21}{10}$$

$$x_1^{(2)} = \frac{1}{5} \times \frac{21}{10} - \frac{7}{5} = -\frac{49}{50}$$
$$x_2^{(2)} = -\frac{1}{4} \times -\frac{49}{50} + \frac{7}{4} = \frac{399}{200}$$

$$x_1^{(3)} = \frac{1}{5} \times \frac{399}{200} - \frac{7}{5} = -1.001$$
$$x_2^{(3)} = -\frac{1}{4} \times -1.001 + \frac{7}{4} = 2.0001$$

$$x_1^{(4)} = \frac{1}{5} \times \frac{8001}{4000} - \frac{7}{5} = 0.99995$$
$$x_2^{(4)} = -\frac{1}{4} \times -0.99995 + \frac{7}{4} = 1.99999$$

Compared to Jacobi:

$$x_1^{(6)} = -1.0001$$
$$x_2^{(6)} = 2.0002$$

Successive Over Relaxation (SOR)

$$x_1^{(k)} = (1 - w)x_1^{(k-1)} + \frac{w}{a_{11}}(b_1 - a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)})$$

$$x_2^{(k)} = (1 - w)x_2^{(k-1)} + \frac{w}{a_{22}}(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k-1)})$$

$$x_3^{(k)} = (1 - w)x_3^{(k-1)} + \frac{w}{a_{33}}(b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)})$$

- Can be derived by multiplying the decomposed system obtained from the Gauss-Seidel method by the relaxation parameter *ω*
- The iterative parameter *ω* should always be chosen such that *0 < ω < 2* (*1< ω < 2* – over-relaxation, *0 < ω < 1* – dampening)
- Number of iterations to reach desired accuracy depends on *ω*

On convergence of iterative methods

- A necessary and sufficient condition for convergence of these methods: the magnitude of the largest eigenvalue of C should be smaller than 1
- If A is diagonally dominant matrix (i.e., the size of the diagonal element is larger than the sum of the moduli of the other elements in the row - $|a_{ii}|>|a_{i1}|+|a_{i2}|+\ldots+|a_{ii-1}|+|a_{ii+1}|+\ldots+|a_{in}|$) then Jacobi and Gauss-Seidel converge (sufficient condition)
- If A is symmetric and positive definite, then Gauss-Seidel converges (Jacobi not necessarily) (sufficient condition)
- A necessary condition for convergence of SOR is $0 < \omega < 2$. If, in addition, A is symmetric and positive definite, then this condition is also sufficient

Recapitulation on iterative solution methods

- Starting from an initial solution x_0, the solution vector x is iteratively computed: $x_{k+1} = x_k + w(b - Ax_k), \quad w \neq 0$
- Number of iterations (and therefore elementary operations) is apriori unknown
- May not converge
- Jacobi method, Gauss-Seidel method, Succesive Over Relaxation (SOR)
- There exist other methods that solve linear systems by minimizing the residual of the equation: $r_k = (b - Ax_k)$ – see Krylov subspace methods

Sensitivity of linear systems

- For a given system $Ax=b$, the exact solution is given by $x^*=A^{-1}b$
- In many real world applications A and b are known only approximately -> which might prevent us to find x^*
- Assume that we know the exact A but not b (we know $\hat{b}$)
  - Q: how sensitive is our solution with respect to uncertainty in b?

Motivational example: deblurring images

- Images blurred: the lens out of focus, defects in lens or optical system, turbulence…
- Frequent problem in astronomy (e.g., Hubble)
- Linear system *Ax=b*:
  - A – blurring matrix
  - x – sharp image
  - b – blurred image

taken from L. Vandenberghe

Deblurring images

Blurred image (b)

Blurred image + noise
(jitter in b)

taken from L. Vandenberghe

Deblurring images

- Solve the system for the two blurred images

What happened here???

$A^{-1}b$

$A^{-1}\hat{b}$

taken from L. Vandenberghe

Sensitivity of linear systems

- For a given system *Ax=b,* the exact solution is given by $x^*=A^{-1}b$
- In many real world applications *A* and *b* are known only approximately -> which might prevent us to find *x**
- Assume that we know the exact *A* but not *b* (we know $\hat{b}$)
  - Q: how sensitive is our solution with respect to uncertainty in *b*?
  - If we denote $\hat{x} = A^{-1}\hat{b}$, we can define
    - Deviation: $\mathrm{e} = x^* - \hat{x}$
    - Residual: $\mathrm{r} = b - A\hat{x} = Ae$
- We need residual to be small in some sense -> we need a distance measure in the vector space

Vector norms

- It is a mapping *R^n* -> *R* satisfying
  - $\|x\| > 0$, $\|x\| = 0$ iff $x = 0$ – positivity
  - $\|ax\| = \|a\|\|x\|$ - homogeneity
  - $\|x + y\| \leq \|x\| + \|y\|$ - triangle inequality

- P-norm
  - $\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}$
    - Manhattan norm (1 norm) : $p = 1$
    - Euclidian norm (2 norm): $p = 2$
    - Maximum norm (∞ norm): $\|x\|_\infty = max_{1 \leq \mathrm{i} \leq \mathrm{n}} |x_i|$

Matrix norms

- It is a mapping R^{mxn} -> R satisfying the three properties presented for the vector norms. A matrix p-norm can additionally be
  - $\|Ax\| \leq \|A\|\|x\|$ - consistent
  - $\|AB\| \leq \|A\|\|B\|$ - submultiplicative
- 2-norm
  - $\|A\|_2 = \sqrt{\lambda_{max}(A^T A)}$

Condition number

- Relates relative errors in the inputs (*b*) to relative errors in solutions:

$$\delta x = \frac{\|x^* - \hat{x}\|}{\|x^*\|} \quad \propto \quad \delta b = \frac{\left\|b^* - \hat{b}\right\|}{\|b^*\|}$$

$$\delta x = \frac{\|x^* - \hat{x}\|}{\|x^*\|} = \frac{\left\|A^{-1}(b^* - \hat{b})\right\|}{\|x^*\|} \leq \frac{\|A^{-1}\|\left\|(b^* - \hat{b})\right\|}{\|x^*\|} \frac{\|b^*\|}{\|b^*\|}$$

$$\delta x \leq \frac{\|A^{-1}\|}{\|x^*\|} \|b^*\| \, \delta b = \frac{\|A^{-1}\|}{\|x^*\|} \|Ax^*\| \, \delta b \leq \|A^{-1}\| \|A\| \, \delta b$$

- Condition number: $\|A^{-1}\|\|A\|$
  - For 2- norm ($\|A\|_2 = \sqrt{\lambda_{max}(A^T A)}$), the condition number is equal to the ratio of the largest and smallest eigenvalue

Condition number

- Some properties
  - Scale-invariant *cond(aA)=cond(A)*, for all *a*
  - It is norm dependent
  - In general, *cond(A) >> 1* system is highly sensitive:
    - a small jitter in RHS (δb) results in big errors in solutions (δx)
    - ill-conditioned systems (otherwise well-conditioned systems)

- When there is uncertainty in *A*

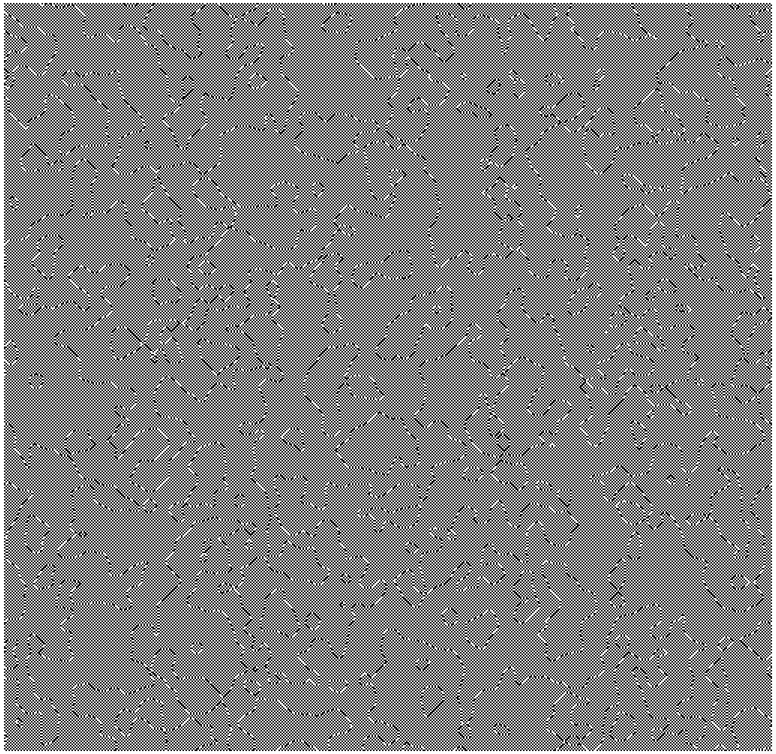
$$\delta x \leq cond(A)(\delta A + \delta b)$$

Deblurring images

- A is nonsingular with condition number ~10^9

$$\delta x \leq \|A^{-1}\| \|A\| \, \delta b$$

$A^{-1}b$



$A^{-1}\hat{b}$

taken from L. Vandenberghe