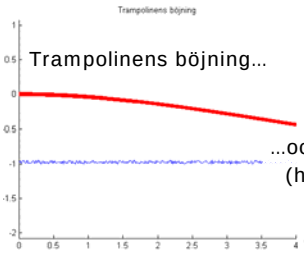
# Block 2: Lineära system

**Del 1**

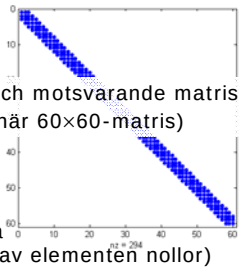
## Exempel

Från labben:

Trampolinens böjning…



…och motsvarande matris (här 60×60-matris)



Matrisen är ett exempel på
- *gles matris* (huvuddelen av elementen nollor)
- *bandmatris*

## I stora drag

- Ca 60-70% av exekveringstiden i beräkningsprogram
- Grundalgoritmer:
  - *Gausseliminering med radpivotering*
  - *Bakåtsubstitution*
- Exekveringstiden växer som $n^3$
- LU-faktorisering sparar exekveringstid
- Speciella algoritmer för stora, glesa ekvationssystem
- Noggrannheten: test genom insättning ej tillförlitlig

## Från labben

- Beräkningstid ökar kraftigt med ökande storlek. Hur mycket?
- Med LU-faktorisering kan man lösa många system på nästan samma tid som ett
- Hur kan man kontrollera noggrannhet? Insättning av lösning ej säker.
- Konditionstal

## Löpsedel

*Dagens föreläsning*

- Algoritmerna gausseliminering och bakåtsubstitution
- Gausseliminering är instabil
- Radpivotering för att stabilisera
- Exekveringstid
- LU-faktorisering

## Läroboken

- Kap 9.2-3 (s 218-227)
- Kap 10.1-2 (s 236-243)
- OBS 1
  Läroboken ger en ofullständig motivering till behovet av pivotering
- OBS 2
  Läroboken visar inte LU-faktorisering *med* pivotering, men det ingår i kursen

## Mål Linj algebra – Ber vet I

Målen här jämfört med matematikursen Linjär algebra

- Mål i matematikkursen
  - att kunna lösa *små ekvationssystem för hand*
  - att teoretiskt förstå egenskaper hos lineära ekvationssystem i allmänhet
- Mål i vår kurs
  - att kunna lösa *stora ekvationssystem med dator*
  - att förstå de datoranpassade *algoritmerna* och deras egenskaper



## Representation av lineära ekvationssystem i datorn

1. Skriv ekvationssystemet på matris-/vektor-form:
   $$Ax = b$$
   där $A$ är en $n \times n$-matris, $x$ och $b$ är $n \times 1$-vektorer (kolonnvektorer av längd $n$)
2. I datorns minne lagras ekvationssystemet genom att vi lagrar matrisen $A$ och vektorn $b$. Vi säger att ekvationssystemet *representeras* med $A$ och $b$ i datorn.



## Algoritmerna gausseliminering och bakåtsubstitution

- Matlabs "*backslash*"-operator (\) löser systemet $Ax = b$:

```
>> x = A\b
```

- När vi gör detta kommando i Matlab utförs två algoritmer:
  1. Gausseliminering
     Systemet $Ax = b$ överförs till formen $Ux = d$, där $U$ är en övertriangulär matris
  2. Bakåtsubstitution
     Systemet $Ux = d$ löses och lösningen lagras i vektorn $x$



## Algoritmen gausseliminering, "naiv" version

- Indata: $A$, $b$, $n$
  Elementet i $A$, rad $i$, kolonn $k$, betecknas $a_{ik}$

1. Bilda totalmatrisen $Aug = [A\ b]$
2. För $k = 1, 2, \ldots, n-1$
   Använd rad $k$ i $Aug$ för att nollställa $a_{ik}$ för $i=k+1, k+2, \ldots, n$



## Algoritmen gausseliminering, "naiv" version, pseudokod

- Indata: $A$, $b$, $n$
- Bilda totalmatrisen Aug = [A b]
- För k = 1, 2, ... , n-1:
  För i = k+1, k+2, ... , n:
  faktor = Aug(i,k)/Aug(k,k)
  Aug(i,k:n+1) =
  Aug(i,k:n+1) – faktor*Aug(k,k:n+1)

OBS minustecknet



## Algoritmen bakåtsubstitution, pseudokod

- Indata: $U$, $d$, $n$
  Efter föregående algoritm finns $U$ i $Aug(1:n,1:n)$ och $d$ i $Aug(1:n,n+1)$
- x(n) = Aug(n,n+1)/Aug(n,n)
  För i = n-1, n-2, ... , 1:
  x(i) = (Aug(i,n+1) –
  Aug(i,i+1:n)*x(i+1:n))/Aug(i,i)

## ”Naiv” gausseliminering instabil

Samma exempel, nu med 3 siffrors noggrannhet

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 0 & -1 \\ 4 & 2 & -3 \end{bmatrix} \begin{bmatrix} 8 \\ -1 \\ -4 \end{bmatrix} \quad l_{21} = fl(1/3) = 0.333 \quad l_{31} = fl(4/3) = 1.33$$

Beteckning: $l_{ik}$ är faktorn som används för att nollställa $a_{ik}$

$$\begin{bmatrix} 3 & -1 & 2 \\ 0 & 0.333 & -1.67 \\ 0 & 3.33 & -5.66 \end{bmatrix} \quad \begin{bmatrix} 8 \\ -3.67 \\ -14.6 \end{bmatrix}$$

$$l_{32} = fl(3.33/0.333) = 10$$



## ”Naiv” gausseliminering är instabil

$$\begin{bmatrix} 3 & -1 & 2 \\ 0 & 0.333 & -1.67 \\ 0 & 0 & 11.0 \end{bmatrix} \begin{bmatrix} 8 \\ -3.67 \\ 22.1 \end{bmatrix} \Rightarrow \quad x_3 = 2.01 \quad x_2 = -0.848 \quad x_1 = 1.61$$

*Instabil algoritm*
håller inte avrundningsfelen ”i schack” utan de ackumuleras till ett stort fel i den beräknade lösningen



## Stabilisera algoritmen genom att införa radpivotering

- Problemet med den ”naiva” algoritmen: Om $|l_{ik}| > 1$ så kommer multiplikation med $l_{ik}$ att förstora avrundningsfel

I algoritmen finns — Felen i A(k,k:n) förstoras

$A(i,k:n) = A(i,k:n) - l_{ik} * A(k,k:n)$

där elementen i $A$ innehåller olika fel, t ex avrundningsfel. Om $l_{ik}$ är stor till belopp förstoras dessa fel successivt i processen



## Stabilisera algoritmen genom att införa radpivotering

- Åtgärd: Radpivotering
  För varje nytt k-värde:
  Finn rad $m$ så att
  $|Aug(m,k)| \geq |Aug(i,k)|$, i=k, k+1, …, n
  Byt plats mellan rad m och rad k
- När $l_{ik}$ skapas divideras då med största elementet i kolonnen
- Resultat: $|l_{ik}| \leq 1$, algoritmen blir stabil

OBS! Läroboken ger en alltför förenklad motivering till behovet av pivotering



## Tillämpa gausseliminering med radpivotering tidigare exempel

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 0 & -1 \\ 4 & 2 & -3 \end{bmatrix} \quad \begin{bmatrix} 8 \\ -1 \\ -4 \end{bmatrix}$$

Radbyte:

$$\begin{bmatrix} 4 & 2 & -3 \\ 1 & 0 & -1 \\ 3 & -1 & 2 \end{bmatrix} \quad \begin{bmatrix} -4 \\ -1 \\ 8 \end{bmatrix}$$



$$\begin{bmatrix} 4 & 2 & -3 \\ 1 & 0 & -1 \\ 3 & -1 & 2 \end{bmatrix} \quad \begin{bmatrix} -4 \\ -1 \\ 8 \end{bmatrix} \quad l_{21} = fl(1/4) = 0.25 \quad l_{31} = fl(3/4) = 0.75$$

$$\begin{bmatrix} 4 & 2 & -3 \\ 0 & -0.5 & -0.25 \\ 0 & -2.5 & 4.25 \end{bmatrix} \quad \begin{bmatrix} -4 \\ 0 \\ 11 \end{bmatrix}$$

Informationsteknologi

Radbyte: $\begin{bmatrix} 4 & 2 & -3 \\ 0 & -2.5 & 4.25 \\ 0 & -0.5 & -0.25 \end{bmatrix}$ $\begin{bmatrix} -4 \\ 11 \\ 0 \end{bmatrix}$

$$l_{32} = fl(-0.5/-2.5) = 0.2$$

$$\begin{bmatrix} 4 & 2 & -3 \\ 0 & -2.5 & 4.25 \\ 0 & 0 & -1.1 \end{bmatrix} \quad \begin{bmatrix} -4 \\ 11 \\ -2.2 \end{bmatrix}$$

$$x_3 = 2$$
$$x_2 = -1$$
$$x_1 = 1$$



Informationsteknologi

## Exekveringstid

- Hur lång tid kommer det att ta för en dator att *exekvera* (utföra/köra) algoritmerna? (Hur kommer exekverings-tiden i Matlab att bli när vi gör kommandot **x = A\b**?)
- Lämpligt med ett datoroberoende mått på exekveringstiden. Man talar om en algoritms *komplexitet*. Olika mått för olika typer av algoritmer.



Informationsteknologi

## Komplexitet hos gauss-eliminering

- Ett lämpligt komplexitetsmått är *antal aritmetiska operationer*. Exekveringstiden kommer väsentligen att vara proportionell mot detta antal.
- Det intressanta är: hur beror exe-kveringstiden på antalet ekvationer, *n*?
- Vi vill alltså uttrycka komplexiteten som en funktion av *n*



Informationsteknologi

## Komplexitetsanalys av gauss-eliminering

1. Undersök antal operationer för ett godtyckligt k-värde:
   För k:
   *n-k* st i-värden
   För i = *k*+1, *k*+2, ... , *n*
   faktor = Aug(i,k)/Aug(k,k) — 1 op
   Aug(i,k:n) =
   Aug(i,k:n) – faktor*Aug(k,k:n)
   2(*n-k*) op

Slutsats: ca $2(n-k)^2$ aritmetiska operationer för steg k i algoritmen



Informationsteknologi

## Komplexitetsanalys (forts)

2. Summera antal operationer för samtliga k-värden, $k=1,\ldots,n-1$
   totalt antal operationer =
   $$\sum_{k=1}^{n-1} 2(n-k)^2 = \frac{2}{3}n^3 + O(n^2)$$
2. Slutsats: Gausseliminering av ett $n \times n$ -system kräver ca (⅔)$n^3$ aritmetiska operationer, eller $O(n^3)$ operationer
3. Kan även visa att *bakåtsubstitution* kräver ca $n^2$ aritmetiska operationer



Informationsteknologi

## Komplexitetsanalys (forts)

Vad blir det här i tid?

Antag $t_f$ = $10^{-9}$ s/flyttalsoperation (s/flop)

| | Gausseliminering | Bakåtsubstitution |
|---|---|---|
| $n$ | $(2/3)n^3 t_f$ | $n^2 t_f$ |
| $10^3$ | 0.67 s | $10^{-3}$ s |
| $10^6$ | 0.67 $10^9$ s ≈ 21 år | $10^3$ s ≈ 17 min |

## Komplexitetsanalys (forts)

Hur stort system kan lösas på en timme om datorn klarar 1 Gflop/s ?

0.67 $n^3$ $10^{-9}$ s = 1 tim = 3600 s,

ger $n \approx 18000$

Hur stort system kan lösas på en minut?

0.67 $n^3$ $10^{-9}$ s = 60 s,

ger $n \approx 4500$



## Behov av effektiva algoritmer

- Komplexiteten $O(n^3)$ begränsar användbarheten hos gausseliminering.
- Alternativ:
  - Utnyttja ”struktur” hos koefficientmatrisen om möjligt (exempelvis bandmatriser). Fortfarande gausseliminering men lägre komplexitet.
  - Iterativa metoder för mycket stora, glesa system
  - *LU-faktorisering* (se nästa bild)



## LU-faktorisering, idé

- Vanlig situation: Följd av ekvationssystem med samma koefficientmatris, olika högerled (jfr trampolinexemplet)

$$Ax^{(k)} = b^{(k)}, k = 1,..., m$$

Idé:

Gausseliminera $A$ en gång för alla
- Spara $U$.
- Spara faktorerna $l_{ik}$ i en matris $L$.
- Spara information om pivoteringen i en matris $P$

Detta kallas *LU-faktorisering av A.*
Man kan visa att: LU = PA



## LU-faktorisering, användning av

- Utför en gång
  $Ax=b \Rightarrow PAx = Pb \Rightarrow LUx = Pb$
  LU-faktorisering $O(n^3)$
- För varje högerled $b^{(k)}$:

  - $Ld = Pb$ (*framåtsubstitution*) $O(n^2)$
    Bestäm motsvarande vektor $d$

  - $Ux = d$ (*bakåtsubstitution*) $O(n^2)$
    Bestäm lösningen $x$



## LU-faktorisering, vinst

- Ineffektivt:
  Lös varje system med $x = A\backslash b$ (gausseliminering av A för varje nytt högerled)
  $m(\frac{2}{3}n^3 + n^2)$
  aritmetiska operationer
- Effektivt:
  LU-faktorisera $A$ (**`lu(A)`** i Matlab) och lös sedan varje system med
  - $d = L\backslash b$
  - $x = U\backslash d$

  $\frac{2}{3}n^3 + m(n^2 + n^2)$
  aritmetiska operationer



## LU-faktorisering i Matlab

```
>> A=[3 -1 2;1 0 -1;4 2 -3];
>> b= [8;-1;-4];
>> [L,U,P]=lu(A)
L = 1.0000         0         0
    0.7500    1.0000         0
    0.2500    0.2000    1.0000

U = 4.0000    2.0000   -3.0000
         0   -2.5000    4.2500
         0         0   -1.1000

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

## LU-faktorisering i Matlab

Stämmer PA=LU ?

```
>> P*A
ans = 4     2    -3
      3    -1     2
      1     0    -1

>> L*U
ans = 4     2    -3
      3    -1     2
      1     0    -1
```

Lösning

```
>> d = L\(P*b)
d = -4.0000
    11.0000
    -2.2000

>> x = U\d
x =  1
    -1
     2
```

Backslash använder algoritmerna för framåt- och bakåtsubstitution när matriserna är under respektive över triangulära



## LU-faktorisering i praktiken

**Ett exempel**

*Matematiskt objekt* — *Matris*

$$\begin{pmatrix} 2 & 2 & -2 \\ -4 & -2 & 2 \\ -2 & 3 & 9 \end{pmatrix}$$

*Datastruktur* — *Matris*

| | | |
|---|---|---|
| 2 | 2 | -2 |
| -4 | -2 | 2 |
| -2 | 3 | 9 |

*Vektor p*

| |
|---|
| 1 |
| 2 |
| 3 |

Radbyte:

$$\begin{pmatrix} -4 & -2 & 2 \\ -4 & -2 & -2 \\ -2 & 3 & 9 \end{pmatrix}$$

| | | |
|---|---|---|
| 2 | 2 | -2 |
| -4 | -2 | 2 |
| -2 | 3 | 9 |

| |
|---|
| 1 |
| 2 |
| 3 |



## LU-faktorisering i praktiken

Eliminering av $x_1$:

$l_{21}$=-1/2, $l_{31}$=1/2

$$\begin{pmatrix} -4 & -2 & 2 \\ 0 & 1 & -1 \\ 0 & 4 & 8 \end{pmatrix}$$

| | | |
|---|---|---|
| -4 | -2 | 2 |
| -1/2 | 1 | -1 |
| 1/2 | 4 | 8 |

| |
|---|
| 2 |
| 1 |
| 3 |

Radbyte:

$$\begin{pmatrix} -4 & -2 & -2 \\ 0 & 4 & 8 \\ 0 & 1 & -1 \end{pmatrix}$$

OBS! Hela rader byter plats

| | | |
|---|---|---|
| -4 | -2 | 2 |
| $\frac{1}{2}$ | 4 | 8 |
| $-\frac{1}{2}$ | 1 | -1 |

| |
|---|
| 2 |
| 3 |
| 1 |



## LU-faktorisering i praktiken

Eliminering av $x_2$:

$l_{32}$=1/4

$$\begin{pmatrix} -4 & -2 & 2 \\ 0 & 4 & 8 \\ 0 & 0 & -3 \end{pmatrix}$$

| | | |
|---|---|---|
| -4 | -2 | 2 |
| $\frac{1}{2}$ | 4 | 8 |
| $-\frac{1}{2}$ | $\frac{1}{4}$ | -3 |

| |
|---|
| 2 |
| 3 |
| 1 |

Klart! Tolkning av datastrukturernas innehåll:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ -\frac{1}{2} & \frac{1}{4} & 1 \end{pmatrix} \quad U = \begin{pmatrix} -4 & -2 & 2 \\ 0 & 4 & 8 \\ 0 & 0 & -3 \end{pmatrix} \quad P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$



## LU-faktorisering i praktiken

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ -1/2 & 1/4 & 1 \end{pmatrix} \begin{pmatrix} -4 & -2 & 2 \\ 0 & 4 & 8 \\ 0 & 0 & -3 \end{pmatrix} = \begin{pmatrix} -4 & -2 & 2 \\ -2 & 3 & 9 \\ 2 & 2 & -2 \end{pmatrix}$$

$$PA = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 2 & -2 \\ -4 & -2 & 2 \\ -2 & 3 & 9 \end{pmatrix} = \begin{pmatrix} -4 & -2 & 2 \\ -2 & 3 & 9 \\ 2 & 2 & -2 \end{pmatrix}$$

Slutsats: LU = PA

- L och U sparas i A:s minnesutrymme
- P lagras som en vektor (inga nollor lagras)