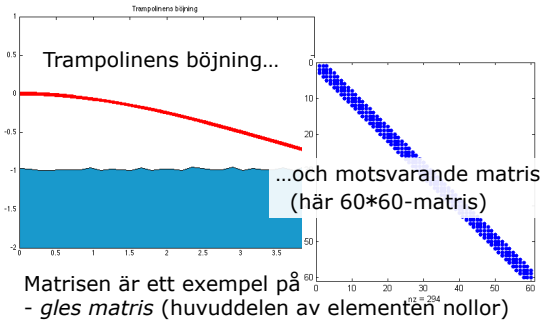
Block 2: Linjära system

Del 1

Exempel

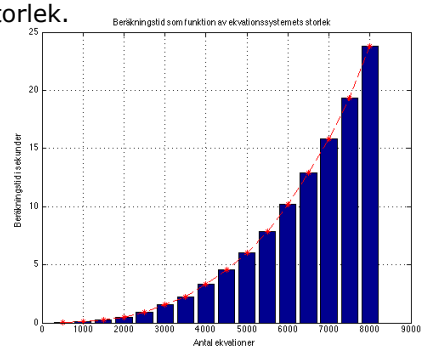
Från labben:



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

Från labben

Beräkningstid ökar kraftigt med ökande storlek.



Från labben

- Med LU-faktorisering kan man lösa många system på nästan samma tid som ett – sparar exekveringstid
- Beräkning med \ (Gausselimination) lite mer än dubbelt så snabbt som **`inv(A)`**
- En första programmeringsstruktur, for-loop, t ex

```
for i=1:n
    v(i) = …
end
```

"for i lika med 1 till n"

I stora drag

- Ca 60-70% av exekveringstiden i beräkningsprogram
- Grundalgoritmer:
  - *Gausseliminering med radpivotering*
  - *Framåtsubstitution, bakåtsubstitution*
- Speciella algoritmer för exempelvis stora, glesa ekvations-system

Löpsedel

Dagens föreläsning

- Grundalgoritmerna: LU-faktorisering, bakåt- och framåtsubstitution
- Gausseliminering är instabil - radpivotering för att stabilisera
- Exekveringstid

- Kursboken:
  Kap (8.1, 8.2), 9.1.1, 9.2, 9.3, 10.1, 10.2, 10.4

Mål Linj algebra – BerVet

Målen här jämfört med matematikursen Linjär algebra

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

Representation i datorn

1. Ekvationssystemet skrivs som matris/vektor:
   $$Ax = b$$
   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 => ekvationssystemet *representeras* med **A** och **b** i datorn.

Algoritmerna gausseliminering och bakåtsubstitution

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

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

- \ använder Gausselimination baserad på s k LU-faktorisering som standard
- ”Intelligent” operator – väljer automatiskt olika metoder beroende på problemet som ska lösas (hur matrisen ser ut)

Gausseliminering – grundalgoritmen

- Grundalgoritmen består av 3 algoritmer:
  1. LU-faktorisering, dvs Gausseliminering av enbart matrisen A => matris L, U
  2. Framåtsubstitution
     Systemet $Ld = b$ löses => lösningen d
  3. Bakåtsubstitution
     Systemet $Ux = d$ löses =>lösningen x

Punkt 1 “dyr” (många operationer), punkt 2 och 3 “billiga” i jämförelse.

Gausseliminering i Matlab

Backslash: \
t ex **x=A\b**

- LU-faktorisering:
  [L,U,P]=lu(A);
- Framåtsubstitution
  d=L\(P*b);
- Bakåtsubstitution
  x=U\d;

- Backslash innehåller de tre stegen
- Om matrisen ser ut som L och U, utför backslash istället framåt- respektive bakåtsubstitutionen

Algoritmen framåtsubstitution, pseudokod

- Indata: *L*, *b*, *n*
  Efter LU-faktorisering finns *L*. *n* är problemstorlek

```
d = zeros(n,1)
d(1) = b(1)
for i = 2:n
    d(i) = ( b(i) –
       L(i,1:i–1)*d(1:i–1) )
end
```

Algoritmen bakåtsubstitution, pseudokod

- Indata: *U*, *d*, *n*

```
x = zeros(n,1);
x(n) = d(n)/U(n,n)
for i = n-1, n-2,…, 1:
    x(i) = ( d(i) –
       U(i,i+1:n)*x(i+1:n) )/U(i,i)
end
```


Exekveringstid

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

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 exekverings-tiden på antalet ekvationer, n?
- Vi vill alltså uttrycka komplexiteten som en funktion av n

Komplexitetsanalys av framåt-substitution

```
d(1) = b(1)
for i = 2:n
    d(i) = ( b(i) –
       L(i,1:i-1)*d(1:i-1) )
end
```

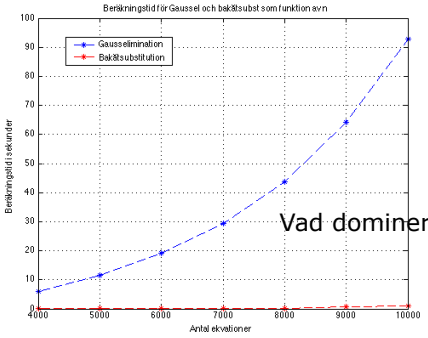
I varje varv i loopen, dvs för varje i utförs $2i\text{-}2 \approx 2i$ operationer. I snitt blir det $2n/2$, dvs n operationer varje varv. Detta upprepas n gånger i loopen, dvs totalt n^2 aritmetiska operationer.

Komplexitetsanalys (forts) $n \times n$

- Framåtsubstitution kostar n^2 operationer, eller $O(n^2)$
- På samma sätt kan man visa att *bakåtsubstitution* kräver ca n^2 aritmetiska operationer, eller $O(n^2)$
- LU-faktorisering (Gausseliminering av matrisen) kräver ca $\frac{2}{3}n^3$ aritmetiska operationer, eller $O(n^3)$ operationer (krångligare att visa)
- Totalt för lösning av ett $n \times n$ linjärt ekvationsystem: $\frac{2}{3}n^3 + 2n^2$ eller $O(n^3)$

Komplexitetsanalys (forts)

Gausselimination jfr. med bakåtsubst



Vad dominerar?

Komplexitetsanalys (forts)

Vad blir det här i tid?

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

	Gausseliminering	Bakåtsubstitution
n	$(2/3)n^3 \cdot t_f$	$n^2 \cdot t_f$
10^3	0.67 s	$5 * 10^{-4}$ s
10^6	$0.67*10^9$ s ≈ 21 år	500 s ≈ 8.3 min

Komplexitetsanalys (forts)

Hur stort system kan lösas på **en timme** om datorn klarar 1 Gflop/s ?
(Gflop = 1 miljard flyttalsoperationer)

$(2/3) \cdot n^3 \cdot 10^{-9}$ s = 1 tim = 3600 s
=> $n \approx 18000$

Hur stort system kan lösas på **en minut**?

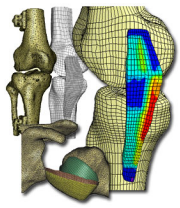
$0.67 \cdot n^3 \cdot 10^{-9}$ s = 60 s => $n \approx 4500$

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 (ingår ej i denna kurs)
  - *LU-faktorisering* (se längre fram)
  - Dessutom använda speciella högpresterande datorer

Behov av effektiva algoritmer

- Man löser system med ända upp 10^9 obekanta
- Använder speciella datorer, t ex grids, och speciella beräkningsmetoder
- Vanligt att lösning av partiella diffar leder till stora linjära ekvationssystem



Problem: ”Naiv” gausseliminering ej stabil

- Om multiplikator $| l_{ik} | > 1$ så kommer multiplikation med l_{ik} att förstora avrundningsfel

I algoritmen finns — Felen förstoras här

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

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 ny kolumn som ska nollställas:
  - Hitta det element i kolumnen, från diagonelementet och nedåt, med störst belopp
  - Byt plats på rader så att detta element hamnar i pivotposition
- När multiplikator l_{ik} skapas divideras det då med det till belopp största elementet i kolonnen
- Resultat: $|l_{ik}| \leq 1$, algoritmen blir stabil

Gausseliminering med radpivotering

Samma exempel (vi bildar inte *Aug* här):

k=1:

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

Radbyte:

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

$$\begin{bmatrix} 4 & 2 & -3 \\ 1 & 0 & -1 \\ 3 & -1 & 2 \end{bmatrix} \quad \begin{bmatrix} -4 \\ -1 \\ 8 \end{bmatrix} \qquad l_{21} = 1/4 \quad l_{31} = 3/4$$

k=2:

$$\begin{bmatrix} 4 & 2 & -3 \\ 0 & -\frac{1}{2} & -\frac{1}{4} \\ 0 & -\frac{5}{2} & \frac{17}{4} \end{bmatrix} \quad \begin{bmatrix} -4 \\ 0 \\ 11 \end{bmatrix}$$

Radbyte:

$$\begin{bmatrix} 4 & 2 & -3 \\ 0 & -\frac{5}{2} & \frac{17}{4} \\ 0 & -\frac{1}{2} & -\frac{1}{4} \end{bmatrix} \quad \begin{bmatrix} -4 \\ 11 \\ 0 \end{bmatrix} \qquad l_{32} = -\frac{1}{2} / -\frac{5}{2} = \frac{1}{5}$$

$$\begin{bmatrix} 4 & 2 & -3 \\ 0 & -\frac{5}{2} & \frac{17}{4} \\ 0 & 0 & -\frac{22}{20} \end{bmatrix} \quad \begin{bmatrix} -4 \\ 11 \\ -\frac{11}{5} \end{bmatrix}$$

Kan nu lösa ut *x* "baklänges" – bakåtsubstitution:

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

Slutligen, den fullständiga algoritmen

- Lagra radbyten i en matris/vektor
- Gausseliminera enbart matrisen först med LU-faktorisering, sedan applicera på högerledet

LU-faktorisering

Matematiskt objekt
Matris

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

Radbyte:

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

Datastruktur
Matris

3	-1	2
1	0	-1
4	2	-3

4	2	-3
1	0	-1
3	-1	2

Vektor p

1
2
3

3
2
1

LU-faktorisering

Eliminering av x_1:

L_{21}=1/4, l_{31}=3/4

$$\begin{pmatrix} 4 & 2 & -3 \\ 0 & -\frac{1}{2} & -\frac{1}{4} \\ 0 & -\frac{5}{2} & \frac{17}{4} \end{pmatrix}$$

4	2	-3
1/4	-1/2	-1/4
3/4	-5/2	17/4

3
2
1

Radbyte:

$$\begin{pmatrix} 4 & 2 & -3 \\ 0 & -\frac{5}{2} & \frac{17}{4} \\ 0 & -\frac{1}{2} & -\frac{1}{4} \end{pmatrix}$$

OBS! Hela raden byter plats

4	2	-3
3/4	-5/2	17/4
1/4	-1/2	-1/4

3
1
2

LU-faktorisering

Eliminering av x_2:

l_{32}=1/5

$$\begin{pmatrix} 4 & 2 & -3 \\ 0 & -\frac{5}{2} & \frac{17}{4} \\ 0 & 0 & -\frac{22}{20} \end{pmatrix}$$

4	2	-3
3/4	-5/2	17/4
1/4	1/5	-22/20

3
1
2

Klart! Tolkning av datastrukturernas innehåll:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{4} & 1 & 0 \\ \frac{1}{4} & \frac{1}{5} & 1 \end{pmatrix} \quad U = \begin{pmatrix} 4 & 2 & -3 \\ 0 & -\frac{5}{2} & \frac{17}{4} \\ 0 & 0 & -\frac{22}{20} \end{pmatrix} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

LU-faktorisering

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{4} & 1 & 0 \\ \frac{1}{4} & \frac{1}{5} & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 & -3 \\ 0 & -\frac{5}{2} & \frac{17}{4} \\ 0 & 0 & -\frac{22}{20} \end{pmatrix} = \begin{pmatrix} 4 & 2 & -3 \\ 3 & -1 & 2 \\ 1 & 0 & -1 \end{pmatrix}$$

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

Slutsats: $LU = PA$

- Detta kallas LU-faktorisering
- *L* och *U* sparas i *A*:s minnesutrymme
- *P* lagras som en vektor, inga nollor lagras
- Har nu utfört $Ax = b \Rightarrow PAx = Pb \Rightarrow LUx = Pb$

LU-faktorisering

- Nu återstår framåt- och bakåtsubstitution
- Sätt $d = Ux \Rightarrow Ld = Pb$:

$$\begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{4} & 1 & 0 \\ \frac{1}{4} & \frac{1}{5} & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} -4 \\ 8 \\ -1 \end{pmatrix} \Rightarrow \begin{array}{l} d_1 = -4 \\ d_2 = (8 - \frac{3}{4} \cdot (-4)) = 11 \\ d_3 = (-1 - \frac{1}{4} \cdot (-4) - \frac{1}{5} \cdot 11) = -\frac{11}{5} \end{array}$$

- Lös $Ux = d$:

$$\begin{pmatrix} 4 & 3 & -3 \\ 0 & -\frac{5}{2} & \frac{17}{4} \\ 0 & 0 & -\frac{22}{20} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -4 \\ 11 \\ -\frac{11}{5} \end{pmatrix} \Rightarrow \begin{array}{l} x_3 = 2 \\ x_2 = -1 \\ x_1 = 1 \end{array} \Rightarrow x = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$

LU-faktorisering

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

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

Idé:

- Gausseliminera A en gång för alla genom LU-faktorisering
- Utför sedan framåt- och bakåtsubstitution på högerleden

LU-faktorisering, användning

- Utför en gång: LU-faktorisering
  $Ax = b \Rightarrow PAx = Pb \Rightarrow LUx = Pb$ — $\frac{2}{3}n^3$
- För varje högerled b :
  - $Ld = Pb$ (*framåtsubstitution*) — n^2
    Bestäm d (vektor)
  - $Ux = d$ (*bakåtsubstitution*) — n^2
    Bestäm lösningen x
- Innebär att man skiljer Gausseliminationen från hantering av högerledet – först elimineras enbart matrisen, sedan applicera på högerledet

LU-faktorisering, vinst

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

  $\frac{2}{3}n^3 + 2mn^2$ 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 övertriangulära

LU-faktorisering i Matlab

Använder backslash LU-faktorisering?
Litet test:

```
>> n = 2000;
>> A = rand(n,n);
>> b40 = rand(n,40); b1 = rand(n,1);
>> tic; x = A\b40; toc
Elapsed time is 5.553007 seconds.
>> tic; x = A\b1; toc
Elapsed time is 5.174191 seconds.
```

40 system med samma matris löses nästan lika fort som 1 system => LU-faktorisering

OBS 40 högerled lagrade $[b_1\ b_2 \ldots b_{40}]$