Linjära system

- Lösning av linjära ekvationssystem: huvuddelen av exekveringstiden i många beräkningsprogram!
- Denna föreläsning:
  - Om vektorer och matriser i Matlab
  - Algoritmer: grundalgoritm för **gausselimination**, **bakåtsubstitution**
  - Grundalgoritmen instabil!
  - Stabilisering via radpivotering
  - Exekveringstid
  - LU-faktorisering
- Dessa anteckningar bygger på material av Stefan Pålsson, Avdelningen för teknisk databehandling, Uppsala Universitet

Mål

Jämför målen med matematiks linjäralgebrakurs:

- Matematikkursen:
  - att förstå egenskaper hos linjära ekvationssystem i allmänhet
  - att kunna lösa små ekvationssystem för hand
- Här:
  - att kunna lösa stora ekvationssystem med dator
  - att förstå de datoranpassade algoritmerna och deras egenskaper

Vektorer i Matlab

Semikolon ger radbyte!

```
>> vk = [1; 2; 3; 4; 5];
>> vr = [5 6 7 8];
>> vk

vk =

     1
     2
     3
     4
     5

>> vr

vr =

     5     6     7     8
```


Vektorer i Matlab

- `vr(1)` till vänster om = betyder tilldelning av ett tal i 1:a komponenten av vektorn `vr`
- Tecknet ’ betyder transponat

```
>> vr(1) = -1.1; vr(2) = 3.5;
>> vr

vr =

   -1.1000    3.5000    7.0000    8.0000

>> vk = vr';
>> vk

vk =

   -1.1000
    3.5000
    7.0000
    8.0000
```

Vektorer, matriser i Matlab

Bestämma storlek och längd

```
>>  A = [1 7; 5 3];
>>  vk = [1; 2; 3; 4; 5];
>> size(A), size(vk), length(vk)
ans =
    2     2

ans =
     5     1

ans =
     5
```

$$A = \begin{pmatrix} 1 & 7 \\ 5 & 3 \end{pmatrix}$$

```
>> A(1,1) = 1; A(1,2) = 7;
>> A(2,1) = 5; A(2,2) = 3;
>> A

A =
     1     7
     5     3
```


Bygga matriser

Givet $A = \begin{pmatrix} 1 & 7 \\ 5 & 3 \end{pmatrix}$, skapa

$$\tilde{A} = \begin{pmatrix} 1 & 7 \\ 5 & 3 \\ 2 & 1 \end{pmatrix}$$

```
>> A = [A; 2 1]
```

alternativt

```
>> temp = [2 1];
>> A = [A; temp];
>> A

A =
     1     7
     5     3
     2     1
```

Skilj på rad- och kolonnvektorer!

```
>> tempk = [2; 1];
>> A = [A; tempk]
??? Error using ==> vertcat
CAT arguments dimensions are not consistent.
```


Bygga matriser

- Matlab kan behandla hela eller delar av rader, kolonner och submatriser
- Kolon (:) betecknar hel rad resp kolonn

```
>> A(2,:)

ans =
   5   3

>> A(2,:) = [0 0];

A =
  1   7
  0   0
  2   1
```


Bygga matriser

Kolonnotationen gör det enkelt att hantera delmatriser

- `A(:,j)` kolonn j i A
- `A(i,:)` rad i i A
- `A(i:k,j:m)` delmatris
  rader $i, i+1, \ldots, k$;
  kolonner $j, j+1, \ldots, m$

```
A =
   1   7   6
   0   0   5
   2   1   3

>> B = A(2:3, 1:2)

B =
   0   0
   2   1
```

Algoritmer

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

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

- Standardalgoritm: **gausselimination baserad på LU-faktorisering** (dagens ämne!)
- "Intelligent" operator: väljer olika metoder beroende på matrisens struktur

Gausselimination—grundalgoritmen

Gausselimination genomförs i två steg:

- **Egentlig gausselimination**: Systemet $Ax = b$ transformeras via *elementära radoperationer* till formen $Ux = d$ där U är en **övertriangulär** matris
- **Bakåtsubstitution**: Systemet $Ux = d$ löses

"Naiv" version (som när man räknar för hand):

- Indata: A, b, n (matrisordning)

1. Bilda totalmatrisen $\hat{A} = [A\ b]$
2. För kolonn $k = 1, 2, \ldots, n-1$

   Nollställ elementen i kolonn k för alla rader i nedanför kolonn k genom att addera lämplig multipel av rad k till rad $i = k+1, k+2, \ldots, n$

Gausselimination—naiv version i pseudokod

- Indata: `A` , `b` , `n` (matrisordning)
- Bilda totalmatrisen `Aug = [A  b]`
-
```
för k = 1: n-1
   för i = k+1:n
      Lik = Aug(i,k)/Aug(k,k)
      Aug(i,k:n+1) = Aug(i,k:n+1) - Lik*Aug(k,k:n+1)
```

- Obs att U skrivs in i övertriangeln av `Aug(1:n,1:n)` och d in i `Aug(1:n,n+1)`!
- Denna överskrivning gör att man sparar minne. Viktigt då matrisen är stor!

Bakåtsubstitution i pseudokod

- Indata: `U`, `d`, `n`
- `x(n) = d(n)/U(n,n)`
- för `i = n-1:-1:1`

```
x(i)= ( d(i) - U(i,i+1:n)*x(i+1:n) )/(U(i,i)
```

- Obs att i praktiken används `Aug(1:n,1:n)` istället för `U`
- Vanligen lagras `x` i `Aug(1:n,n+1)` (d v s man skriver över vektorn `d`) i stället för i en separat variabel `x`!

Naiv gausselimination är instabil

Exempel:

$$(A\,|\,b) = \left(\begin{array}{ccc|c} 3 & -1 & 2 & 8 \\ 1 & 0 & -1 & -1 \\ 4 & 2 & -3 & -4 \end{array}\right) \text{ med exakta lösningen } x = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}$$

Låt L_{ik} vara faktorn som används för att nollställa a_{ik}. Anta 3 siffrors noggrannhet

$$\text{fl}(L_{21}) = \text{fl}(1/3) = 0.333$$
$$\text{fl}(L_{31}) = \text{fl}(4/3) = 1.33$$

$$\Rightarrow \left(\begin{array}{ccc|c} 3 & -1 & 2 & 8 \\ 0 & 0.333 & -1.67 & -3.67 \\ 0 & 3.33 & -5.67 & -14.6 \end{array}\right)$$

$$\text{fl}(L_{32}) = \text{fl}(3.33/0.333) = 10$$

Naiv gausselimination är instabil

$$\Rightarrow \left(\begin{array}{ccc|c} 3 & -1 & 2 & 8 \\ 0 & 0.333 & -1.67 & -3.67 \\ 0 & 0 & 11.0 & 22.1 \end{array}\right) \Rightarrow \text{fl}(x) = \begin{pmatrix} 2.01 \\ -0.848 \\ 1.61 \end{pmatrix}$$

vilket är långt från den riktiga lösningen $x = (1, -1, 2)^T$

Instabil algoritm: algoritmen förstärker successivt avrundningsfelen. Orsakar ett stort fel i lösningen

Kom ihåg:

```
Aug(i,k:n+1) = Aug(i,k:n+1) - Lik*Aug(k,k:n+1)
```

Problemet: om $|L_{ik}| > 1$ så kommer multiplikationen att förstora avrundningsfelen!

Avrundningsfelen kommer successivt att bli värre och värre

Stabilisering via radpivotering

- Åtgärd: **radpivotering**
- För varje nytt k, hitta på vilken rad m det gäller att
  `Aug(m,k)` $\geq$ `Aug(i,k), i = k, k+1, ..., n`
- I ord: hitta det till beloppet största elementet i kolonn k i nedanför och inkluderande diagonalen
- Byt plats på rad m och k
- När L_{ik} skapas divideras då med det största elementet i kolonnen
- Resultat: $|L_{ik}| \leq 1$, vilken stabiliserar algoritmen

Radpivotering

Tidigare exempel:

$$(A\,|\,b) = \left(\begin{array}{ccc|c} 3 & -1 & 2 & 8 \\ 1 & 0 & -1 & -1 \\ 4 & 2 & -3 & -4 \end{array}\right) \begin{matrix} \leftarrow \\ \\ \leftarrow \end{matrix}$$ Byt plats på rad 1 och 3

$$\left(\begin{array}{ccc|c} 4 & 2 & -3 & -4 \\ 1 & 0 & -1 & -1 \\ 3 & -1 & 2 & 8 \end{array}\right)$$

$$\text{fl}(L_{21}) = \text{fl}(1/4) = 0.25$$
$$\text{fl}(L_{31}) = \text{fl}(3/4) = 0.75$$

$$\Rightarrow \left(\begin{array}{ccc|c} 4 & 2 & -3 & -4 \\ 0 & -0.5 & -0.25 & 0 \\ 0 & -2.5 & 4.25 & 11 \end{array}\right) \begin{matrix} \\ \leftarrow \\ \leftarrow \end{matrix}$$ Byt plats på rad 2 och 3

Radpivotering

$$\Rightarrow \left(\begin{array}{ccc|c} 4 & 2 & -3 & -4 \\ 0 & -2.5 & 4.25 & 11 \\ 0 & -0.5 & -0.25 & 0 \end{array}\right)$$

$$\mathrm{fl}(L_{32}) = \mathrm{fl}(-0.5/-2.5) = 0.2$$

$$\Rightarrow \left(\begin{array}{ccc|c} 4 & 2 & -3 & -4 \\ 0 & -2.5 & 4.25 & 11 \\ 0 & 0 & -1.1 & -2.2 \end{array}\right) \quad \Rightarrow \quad \begin{array}{l} x_1 = 1 \\ x_2 = -1 \\ x_3 = 2 \end{array}$$

Exekveringstid

- För stora matriser kan det ta mycket lång tid att **exekvera** (utföra) gausselimination
- Lämpligt med ett datoroberoende mått på hur beräkningskrävande en algoritm är
- Centralt begrepp: en algoritms **komplexitet**
- Komplexitet mäts på olika sätt för olika typer av algoritmer

Komplexitet hos gausselimination

- Lämpligt komplexitetsmått: **antal aritmetiska operationer**
- Exekveringstiden kommer väsentligen att vara proportionell mot detta (minneshanteringen påverkar också exekveringstiden, särskilt för mycket stora problem)
- Centralt problem: hur beror antalet aritmetiska operationer på matrisen ordning?

Komplexitetsanalys

- Undersök gausseliminationen för ett godtyckligt $k = 1, \ldots, n-1$

```
för i = k+1:n                                  utförs n − k gånger
   Lik = Aug(i,k)/Aug(k,k)                     1 op
   Aug(i,k:n+1) =                              n − k + 2 element
      Aug(i,k:n+1) - Lik*Aug(k,k:n+1)          2 op per element
```

  Antal op: $(n-k)\big[1 + 2(n-k+2)\big] \approx 2(n-k)^2$ (räknar enbart högsta potens)
- Summera över alla k:

$$\sum_{k=1}^{n-1} 2(n-k)^2 = \frac{2}{3}n^3 + O(n^2) \qquad \text{(Lemma 8.3.1 i bok)}$$

- Slutsats: **gausselimination** av ett $n \times n$-system kräver $\frac{2}{3}n^3 + O(n^2)$ aritmetiska operationer
- Analog analys: **bakåtsubstitution** kräver $\frac{n^2}{2} + O(n)$ aritmetiska operationer

Komplexitetsanalys

Vad betyder detta i exekveringstid?

Antag $t_f = 10^{-9}$ s/flyttalsop (s/flop); ett realistiskt tal

	faktorisering	bakåtsubstitution
n	$\frac{2}{3}n^3 t_f$	$n^2 t_f$
10^3	0.67 s	10^{-3} s
10^6	0.67×10^9 s $\approx$ 21 år	10^3 s $\approx$ 17 min

Komplexitetsanalys

Hur stort system kan lösas på en timme om datorn klarar 1 Gflop/s? (Gflop = 10^9 flyttalsoperationer)

Svar: $\frac{2}{3}n^3 \cdot 10^{-9} = 3\,600 \Rightarrow n \approx 18\,000$

Hur stort system kan lösas på en minut?

Svar: $\frac{2}{3}n^3 \cdot 10^{-9} = 60 \Rightarrow n \approx 4\,500$

Behov av effektiva algoritmer

- Komplexiteten $O(n^3)$ begränsar användbarheten hos gausselimination
- Alternativ:
  - Utnyttja **struktur** i matrisen, om möjligt. Finns versioner av gausselimination t ex för bandade eller mycket glesa matriser
  - En helt annan typ av algoritmer, **iterativa** algoritmer, blir nödvändiga för mycket stora, glesa matriser
    - Sådana matriser kommer ofta från lösning av **partiella differentialekvationer**
    - För sådana problem förekommer matrisordningar upp till $n = 10^8$!
    - Kräver stora paralleldatorer (t ex den nya Akka här i Umeå) och speciellt anpassade algoritmer

LU-faktorisering

- Vanlig situation: följd av ekvationssystem med samma matris men med olika högerled:
$$Ax^{(k)} = b^{(k)}, \quad k = 1, \ldots, m$$
- Idé: faktorisera A en gång för alla:
  - Spara U
  - Spara faktorerna L_{ik} i en matris L
  - Spara informationen om pivoteringen i en matris P
- Detta kallas **LU-faktorisering** av A
- Kan visa att $LU = PA$

LU-faktorisering

- Givet A, beräkna L, U, P, så att $LU = PA$

$$Ax = b \Rightarrow PAx = Pb \Rightarrow LU = Pb \qquad \text{(faktorisering, } O(n^3)\text{)}$$

- För varje högerled $b^{(k)}$, utför
  - Lös problemet

$$Ld = Pb \qquad \text{(framåtsubstitution, } O(n^2)\text{)}$$

  för att bestämma d
  - Lös problemet

$$Ux = d \qquad \text{(bakåtsubstitution, } O(n^2)\text{)}$$

  för att bestämma lösningen $x^{(k)}$

LU-faktorisering

Vad vinner man på LU-faktorisering jämfört med "vanlig" gausselimination?

- Ineffektiv strategi: Lös varje system med `x=A\b`
  - A kommer att faktoriseras på nytt för varje högerled!
  - Komplexitet: $m(\frac{2}{3}n^3 + n^2)$ (m system med faktorisering + bakåtsubstitution)
- Effektiv strategi: LU-faktorisera A (`lu(A)` i Matlab) och lös sedan varje system med
  - `d = L\b`
  - `x = U\d`
  - Komplexitet: $\frac{2}{3}n^3 + 2mn^2$
    (1 faktorisering + m framåt- och bakåtsubstitution)

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

Kolla:

```
>> 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 med LU-faktorisering

```
>> d = L\(P*b)
d =
   -4.0000
   11.0000
   -2.2000
>> x = U\d
x =
     1
    -1
     2
```

Obs: När matriserna är under- eller övertriangulära väljer backslashoperatorn algoritmerna för framåt- respektive bakåtsubstitution

LU-faktorisering i Matlab

Använder backslash LU-faktorisering? Testa!

```
>> n = 2000;
>> A = rand(n,n);
>> B40 = rand(n, 40); b1 = rand(n,1);
>> tic; X = A\B40; toc
Elapsed time is 1.883686 seconds.
>> tic; x = A\b1; toc
Elapsed time is 1.481570 seconds.
```

- 40 system med samma matris läses nästan lika fort som 1 system!
- Indikerar användning av LU-faktorisering
- Observera att 40 högerled lagras i matrisen $B_{40} = [b_1 b_2 \dots b_{40}]$

LU-faktorisering: exempel

Matematiskt objekt

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

Datastrukturer

```
 2   2  -2        1
-4  -2   2        2
-2   2   9        3
```

matris permutationsvektor

Radbyte:

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

```
-4  -2   2        2
 2   2  -2        1
-2   2   9        3
```

matris permutationsvektor

LU-faktorisering: exempel

Elimination, kolonn 1, med faktorer $L_{21} = -1/2$, $L_{31} = 1/2$:

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

```
-4   -2   2        2
-1/2  1  -1        1
 1/2  4   8        3
```

matris permutationsvektor

Radbyte:

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

```
-4   -2   2        2
 1/2  4   8        3
-1/2  1  -1        1
```

matris permutationsvektor

LU-faktorisering: exempel

Elimination, kolonn 2, med faktorer $L_{32} = 1/4$:

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

```
-4   -2    2        2
 1/2  4    8        3
-1/2  1/4 -3        1
```

matris permutationsvektor

Klart! Matristolkning av datastrukturerna:

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

LU-faktorisering: exempel

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

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

- Slutsats: $LU = PA$
- L och U lagras i A:s minnesutrymme
- Permutationsinformationen lagras i en vektor (en full matris P med mest nollor vore slöseri med minnesutrymme)