Introduktion till MATLAB

- Aritmetiska uttryck
- Matematiska funktioner
- Datatyper och variabler
- Vektorer/listor

Aritmetiska uttryck

- De fyra räknesätten *addition*, *subtraktion*, *multiplikation* och *division* utförs i MATLAB med operatorerna

  `+`

  `-`

  `*`

  `/`

- *Exempel.* Om det aritmetiska uttrycket

  ```
  >> 1+2
  ```

  skrivs in vid MATLAB-prompten (inmatningen avslutas med att trycka på *retur*-tangenten) utför MATLAB operationen och svarar

  ```
  ans =
       3
  ```

  där `ans` är en förkortning för *answer*

Aritmetiska uttryck (forts.)

- *Exempel.*

```
>> 4-6.5
ans =
    -2.5000

>> 4*0.9
ans =
     3.6000

>> 1/3
ans =
     0.3333
```

- Notera att decimaltal i MATLAB skrivs med *punkt*
- MATLAB räknar med cirka 16 siffrors noggrannhet men som default visas endast fyra decimaler

Aritmetiska uttryck (forts.)

- Om ett aritmetiskt uttryck innehåller fler än en operator utförs multiplikationer och divisioner först (*från vänster till höger*)
- Vill man ändra på denna *prioritet*sordning kan man använda parenteser `( )`
- Uttryck innanför parenteser beräknas alltid först
- *Exempel.*

```
>> 2*3-4
ans =
     2

>> 2*(3-4)
ans =
    -2
```

Aritmetiska uttryck (forts.)

- *Exempel.*

```
>> 4/2+1
ans =
     3

>> 4/(2+1)
ans =
    1.3333

>> (1+2+3)*(4+5)
ans =
    54

>> 1/2*3
ans =
    1.5000
```

Aritmetiska uttryck (forts.)

- *OBS!*

```
>> 1/0
Warning: Divide by zero.
ans =
   Inf

>> 0/0
Warning: Divide by zero.
ans =
   NaN
```

- *Inf* = Infinity
- *NaN* = Not a Number

Potenser

- Potenser beräknas i MATLAB med operatorn ^
- *Exempel.* För att beräkna 2^3, 2^0 och $2^{-3} = \frac{1}{2^3}$ skriver man

```
>> 2^3
ans =
     8

>> 2^0
ans =
     1

>> 2^(-3)
ans =
    0.1250
```

Potenser (forts.)

- I uttryck där både potenser och de fyra räknesätten ingår, beräknas potenser *före* multiplikationer/divisioner
- Som alltid kan man med parenteser kringgå dessa prioritetsregler: Uttryck innanför parenteser beräknas *alltid först*
- *Exempel.*

```
>> 2^2*3
ans =
    12

>> 2^(2+3)
ans =
    32
```

Potenser (forts.)

- *Exempel.* 10-potensform

```
>> 5.1*10^3
ans =
    5100

>> 3.1*10^(-2)
ans =
    0.0310
```

- I MATLAB finns en speciell syntax för tal på 10-potensform
- Med denna syntax kan det sista exemplet skrivas:

```
>> 5.1e3      % OBS! Har inget att göra med talet e = 2.71828...
>> 3.1e-2     %      Här betyder e "exponent"
```

Prioritet

- Vi sammanfattar ordningen i vilken uttryck med parenteser, potenser och de fyra räknesätten utförs:
  1. Parenteser: `( )`
  2. Potenser: `^`
  3. Multiplikation och division: `* /`
  4. Addition och subtraktion: `+ -`
- Parenteser sägs ha *högst prioritet*, o.s.v.
- Operationer med samma prioritet utförs från vänster till höger

Matematiska funktioner

- Potenser med icke-heltalsexponenter:

```
>> 16^(1/2)                    % Alternativ: >> nthroot(16,2)
ans =
     4

>> 8^(1/3)                     % Alternativ: >> nthroot(8,3)
ans =
     2
```

- Att beräkna *kvadratroten* ur ett tal, $\sqrt{x} = x^{1/2}$, är en så vanlig operation att det finns ett speciellt kommando, `sqrt` (förkortning för *square root*), för detta:

```
>> sqrt(81)
ans =
     9
```

Matematiska funktioner (forts.)

- Den *naturliga exponentialfunktionen*, e^x, heter `exp` i MATLAB
- *Exempel.*

```
>> exp(1)
ans =
    2.7183

>> exp(2)
ans =
    7.3891
```

  vilket motsvarar $e^1 = e = 2.718281828...$ resp. $e^2 = 7.389056098...$

- OBS! Man skriver INTE

```
>> e^2    % FEL!!!
```

Matematiska funktioner (forts.)

- Den *naturliga logaritmfunktionen*, $\ln(x)$, heter `log` i MATLAB
- Logaritmfunktionen med 10 som bas, $\log_{10}(x)$, heter `log10` i MATLAB
- *Exempel.*

```
>> log(2)
ans =
    0.6931
>> log10(1000)
ans =
    3.0000
>> log(exp(2))
ans =
     2
```

vilket motsvarar $\ln(2) = 0.693147...$, $\log_{10}(1000) = 3$, $\ln(e^2) = 2$.

Matematiska funktioner (forts.)

- De *trigonometriska funktionerna*: $\sin(x)$, $\cos(x)$ och $\tan(x)$ heter `sin`, `cos` och `tan` i MATLAB
- *Exempel.* (Notera att π skrivs `pi` i MATLAB)

```
>> sin(pi/2)   % OBS! Vinklar skall anges i radianer!
ans =
     1

>> cos(pi)
ans =
    -1

>> tan(pi/3)
ans =
    1.7321
```

vilket motsvarar $\sin(\frac{\pi}{2}) = 1$, $\cos(\pi) = -1$, $\tan(\frac{\pi}{3}) = \sqrt{3} = 1.7320508...$

Matematiska funktioner (forts.)

- Vill man ange vinklar i grader kan man istället använda kommandona `sind`, `cosd` och `tand`. (d = degree)
- *Exempel.*

```
>> sind(90)
ans =
     1

>> cosd(180)
ans =
    -1

>> tand(60)
ans =
    1.7321
```

Matematiska funktioner (forts.)

- Fler funktioner:

`abs(x)`	beräknar absolutbeloppet $\|x\|$
`round(x)`	avrundar x till närmaste heltal
`floor(x)`	avrundar x nedåt till heltal
`ceil(x)`	avrundar x uppåt till heltal
`mod(x,y)`	beräknar resten vid heltalsdivisionen x/y

- Skriv

```
>> help elfun
```

för att få en lista på ännu fler funktioner!

Matematiska funktioner (forts.)

- *Exempel.*

```
>> abs(-4)
ans =
     4
>> round(-3.7)
ans =
    -4
>> floor(-3.7)    % OBS: Avrundning nedåt.
ans =
    -4
>> ceil(-3.7)     % OBS: Avrundning uppåt.
ans =
    -3
>> mod(20,3)      % 20 = 3*6 + 2
ans =
     2
```

Variabler

- Betrakta kommandona:

```
>> 4+9
ans =
    13

>> 8*7
ans =
    56
```

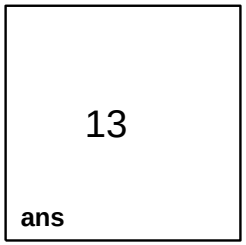
  där *värdet* på `ans` förstås beror på vilket uttryck som matats in.

- En sådan storhet, som `ans`, vars värde kan ändra sig (är variabelt) kallas för en *variabel*.
- Jämför med `pi` som har samma (konstant) värde hela tiden och därför kallas för en *konstant*.

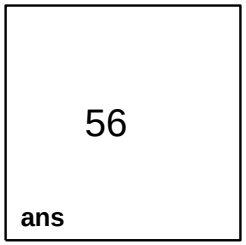
Variabler (forts.)

- Man kan se en variabel som en låda med ett innehåll som kan ändras.
- Skilj mellan *namnet* på lådan, `ans`, och lådans *innehåll*, t.ex. 13.
- *Exempel.*

```
>> 4+9
ans =
    13
```



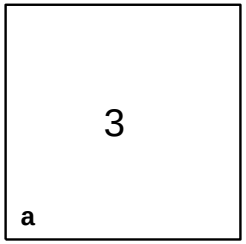

```
>> 8*7
ans =
    56
```



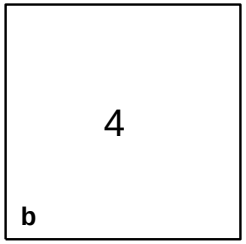
Variabler (forts.)

- Variabeln `ans` skapades automatiskt av MATLAB.
- Man kan också skapa *egna* variabler:

```
>> a = 3
a =
     3
```




```
>> b = 4
b =
     4
```



  skapar två variabler, `a` och `b`, med värden 3 och 4.

- Man säger att man *tilldelar* variablerna `a` och `b` värdena 3 och 4.

Variabler (forts.)

- Om man i ett uttryck skriver *namnet* på en variabel, kommer MATLAB att ersätta namnet med variabelns aktuella *värde*.
- *Exempel.* (Kom ihåg: Variablerna `a` och `b` tilldelades värdena 3 resp. 4.)

```
>> a+b
ans =
     7
>> c = a*b
c =
   12
```

- Generell MATLAB-syntax för tilldelning av värde till variabel:

  `variabelnamn` = *uttryck*

- Notera i exemplet ovan att om man *enbart* skriver ett uttryck så tilldelas variabeln `ans` uttryckets värde. Variabeln `ans` är alltså en slags "default-variabel".

Variabler (forts.)

- Notera att när man skriver:

  `variabelnamn` = *uttryck*

  så beräknas *först* värdet av *uttryck*, och *sedan* tilldelas variabeln `variabelnamn` detta värde.
- *Exempel.*

```
>> n = 1
n =
     1
>> n = n+1
n =
     2
>> n = 10*n
n =
    20
```

- OBS! Kommandona `n = n+1` och `n = 10*n` är *inte* matematiska *likheter* utan *tilldelningar*.

Variabler (forts.)

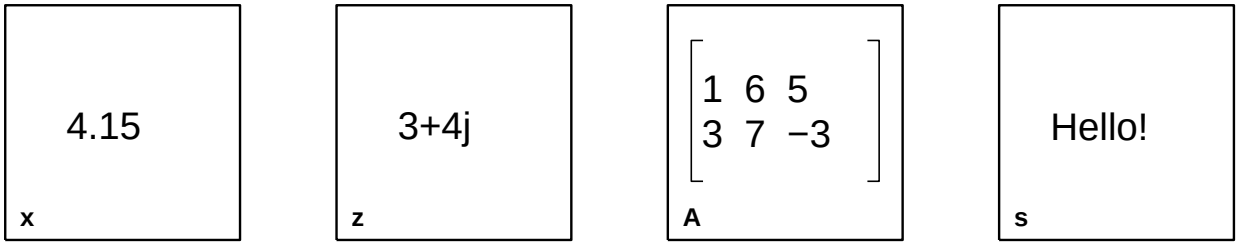
- Notera att MATLAB skiljer på stora och små bokstäver.
- *Exempel.*

```
>> h = 2
h =
   2

>> H = 3
H =
   3

>> H^h
ans =
     9
```

- `H` och `h` är alltså två *olika* variabler.

Datatyper

- Reella tal
- Komplexa tal
- Matriser/Vektorer
- Strängar



Reella tal

- MATLAB räknar som default med flyttal i *dubbel precision*
- Ett reellt tal tar upp 64 bitar (“1:or och 0:or”) i minnet (d.v.s. 8 bytes)
- Ger cirka 16 värdesiffror
- Alltså *inte* exakta beräkningar

Reella tal (forts.)

- Som default *visas* endast fyra decimaler. Vill man se fler kan man ändra *utskriftsformat* med kommandot `format`.
- *Exempel.*

```
>> pi
ans =
    3.1416
>> format long
>> pi
ans =
   3.14159265358979
>> format short   % Byter tillbaka till default
```

- För att se fler utskriftsformat skriv:

```
>> help format
```

Reella tal (forts.)

- Var försiktig vid addition/subtraktion av tal av olika storleksordningar
- *Exempel.*

```
>> format long
>> 1 + 1e-14
ans =
   1.00000000000001
>> 1 + 1e-20   % MATLAB räknar med ca 16 siffrors noggrannhet
ans =
     1
>> 1 + 1e-20 - 1    % Ger FEL svar!
ans =
     0
>> 1 - 1 + 1e-20    % Ger RÄTT svar!
ans =
    1.000000000000000e-020
```

Komplexa tal

Syntax

- Man kan använda både `i` och `j` för att beteckna den *imaginära enheten* $i = j = \sqrt{-1}$.
- *Exempel.* För att skapa två komplexa variabler `z1` och `z2` med värden $z_1 = 3 - 2i$ och $z_2 = 1 + 4i$ kan man skriva:

```
>> clear i j  % Raderar eventuella egna variabler med dessa namn!

>> z1 = 3 - 2i
z1 =
   3.0000 - 2.0000i

>> z2 = 1 + 4j
z2 =
   1.0000 + 4.0000i
```

Komplexa tal (forts.)

Komplex aritmetik

- De aritmetiska operatorerna:

  ```
  +
  -
  *
  /
  ```

  är också definierade för komplexa tal.

- *Exempel.* (Kom ihåg: $z_1 = 3 - 2i$ och $z_2 = 1 + 4i$)

  ```
  >> z1 + z2
  ans =
     4.0000 + 2.0000i

  >> z1*z2
  ans =
    11.0000 + 10.0000i
  ```

Komplexa tal (forts.)

Matematiska funktioner för komplexa tal

- Låt $z = x + iy = r(\cos\theta + i\sin\theta)$

`real(z)`	beräknar realdelen x
`imag(z)`	beräknar imaginärdelen y
`conj(z)`	beräknar konjugatet $\bar{z} = x - iy$
`abs(z)`	beräknar absolutbeloppet $r = \lvert z\rvert = \sqrt{x^2 + y^2}$
`angle(z)`	beräknar (principal)argumentet $\theta = \operatorname{Arg} z$

Komplexa tal (forts.)

- *Exempel.*

```
>> z = 1 + i
>> real(z)
ans =
     1
>> conj(z)
ans =
   1.0000 - 1.0000i
>> abs(z)
ans =
    1.4142
>> angle(z)    % 45 grader = pi/4 radianer = 0.785398... radianer
ans =
    0.7854
```

Matriser/Vektorer

Syntax

- Matrisen (ett rektangulärt schema) är den grundläggande datatypen i MATLAB (MATrix LABoratory!)
- Matriser skapas med hakparenteser `[` och `]`
- Kolonner skiljs åt med mellanrum *eller* komma `,`
- Rader skiljs åt med semi-kolon `;`
- Vi ger några exempel, men kommer därefter i dag enbart att studera vektorer/listor med tal (där matrisen endast har en rad eller en kolonn/kolumn)

Matriser/Vektorer (forts.)

Syntax

- *Exempel.* För att skapa matrisen $A = \begin{bmatrix} 1 & 2 & -3 \\ -2 & 0 & 4 \end{bmatrix}$ skriver man antingen:

```
>> A = [1 2 -3;-2 0 4]
  A =
       1     2    -3
      -2     0     4
```

eller:

```
>> A = [1,2,-3;-2,0,4]
  A =
       1     2    -3
      -2     0     4
```

Matriser/Vektorer (forts.)

Syntax

- *Exempel.* För att skapa radmatrisen/radvektorn $B = \begin{bmatrix} 7 & 1 & -2 & 4 \end{bmatrix}$ skriver man antingen:

```
>> B = [7 1 -2 4]
  B =
       7     1    -2     4
```

eller:

```
>> B = [7,1,-2,4]
  B =
       7     1    -2     4
```

Matriser/Vektorer (forts.)

- *Exempel.*

```
>> y = [2,5+i,7-4i]   % Elementen kan vara komplexa tal.
y =
   2.0000        5.0000 + 1.0000i   7.0000 - 4.0000i
```

- *Exempel.* För att skapa kolonnmatrisen/kolonnvektorn $C = \begin{bmatrix} -4 \\ 5 \\ 3 \end{bmatrix}$

  skriver man:

```
>> C = [-4;5;3]
  C =
      -4
       5
       3
```

Vektorer

- Man kan tilldela elementen i en vektor/lista värden ett och ett.
- *Exempel.*

```
>> v(1) = 3
v =
     3

>> v(2) = 8
v =
     3     8

>> v(3) = -4
v =
     3     8    -4
```

Kolonnotation och linspace

- Med *kolonnotation* kan man generera en vektor där elementens värden ligger på konstant avstånd.
- *Syntax.*

  `a:b` Ger en vektor med värden från a till b i steg om ett.

  `a:h:b` Ger en vektor med värden från a till b i steg om h. Steglängden h kan vara negativ.

- *Exempel.*

```
>> x = 2:6
x =
     2     3     4     5     6
```

Kolonnotation och linspace (forts.)

- *Exempel.*

```
>> x = 3:2:11
x =
     3     5     7     9    11
```

- *Exempel.*

```
>> x = -1:0.5:1
x =
   -1.0000   -0.5000         0    0.5000    1.0000
```

- *Exempel.*

```
>> x = 9:-1:1
x =
     9     8     7     6     5     4     3     2     1
```

Kolonnotation och linspace (forts.)

- Man kan alternativt använda kommandot `linspace` för att generera en vektor där elementens värden ligger på konstant avstånd.
- *Syntax.*

`linspace(a,b)`	Ger en vektor med hundra element jämnt fördelade mellan a och b.
`linspace(a,b,n)`	Ger en vektor med n element jämnt fördelade mellan a och b.

- *Exempel.*

```
>> x = linspace(0,1,5)
x =
         0    0.2500    0.5000    0.7500    1.0000
```

Delvektorer

- Det går att referera till delar av en vektor.
- *Syntax.*

`x(i)`	Ger elementet på plats i.
`x(i:j)`	Ger delvektorn bestående av elementen från i till j i steg om ett.
`x(i:k:j)`	Ger delvektorn bestående av elementen från i till j i steg om k.
`x([i1 ... ip])`	Ger delvektorn bestående av elementen i_1, ..., i_p.

- *Notera.* De tre översta är samtliga specialfall av den fjärde.

Delvektorer (forts.)

- *Exempel.*

```
>> x = [-6 -2 -1 0 3 9]
x =
    -6    -2    -1     0     3     9
>> y = x(4)                                  % Element 4.
y =
     0
>> y = x(2:5)                                % Element 2 t.o.m. 5.
y =
    -2    -1     0     3
>> y = x(2:2:6)                              % Element 2, 4 och 6.
y =
    -2     0     9
>> y = x([1 3 4 6])                          % Element 1, 3, 4 och 6.
y =
    -6    -1     0     9
```

Delvektorer (forts.)

- Med en *tom vektor* `[]` kan man ta bort element ur en vektor.
- *Exempel.*

```
>> x = [-6 -2 -1 0 3 9]
x =
    -6    -2    -1     0     3     9

>> x([2 5]) = []                          % Ta bort element 2 och 5.
x =
    -6    -1     0     9

>> x([3]) = []                            % Ta bort element 3.
x =
    -6    -1     9
```

Strängar

Syntax

- Strängar omges av apostrofer ’ och ’
- Man kan betrakta en sträng som en radmatris/radvektor där varje element är ett tecken
- *Exempel.*

```
>> namn = 'Cleve Moler'      % Skrev den första versionen
namn =                       % av MATLAB på 70-talet
Cleve Moler
```

Inläsning och utskrift av variabler

- En variabel kan läsas in från tangentbordet med kommandot `input`.
- *Syntax.*

  `x = input(`*teckensträng*`)`

  `x = input(`*teckensträng*`,'s')`
- *Exempel.* (Inmatning av tal.)

```
>> x = input('Mata in ett tal: ')
Mata in ett tal: 9
x =
     9
```

Inläsning och utskrift av variabler (forts.)

- *Exempel.* (Inmatning av matris.)

```
>> A = input('Mata in en matris: ')
Mata in en matris: [1 2;3 4]
A =
     1     2
     3     4
```

- *Exempel.* (Inmatning av *teckensträng*.)

```
>> frukt = input('Ge en frukt: ','s')
Ge en frukt: Mandarin
frukt =
Mandarin
```

Inläsning och utskrift av variabler (forts.)

- Man kan skriva ut en variabel på skärmen:
  1. genom att ge dess namn (i detta fall skrivs *både* variabelns namn och dess innehåll ut)
  2. med kommandot `disp` (i detta fall skrivs endast variabelns innehåll ut)
- *Exempel.*

```
>> x
x =
     9

>> disp(x)
     9
```

Inläsning och utskrift av variabler (forts.)

- *Exempel.*

```
>> frukt
frukt =
Mandarin

>> disp('Inmatad frukt:'), disp(frukt)
Inmatad frukt:
Mandarin
```

- Notera i det sista exemplet att man kan ge flera kommandon på samma rad om man sätter *komma* , emellan.

Inläsning och utskrift av variabler (forts.)

- För *formatterad* utskrift kan man använda kommandot `fprintf`
- Egentligen står det första *f*:et för *file*. Kommandot används också för formatterad utskrift till fil.
- *Syntax.* (Vid utskrift till skärm)

  `fprintf(`*format*`,x,y,z,...)`

- Data i matrisen `x` (och eventuellt efterföljande matriser `y`, `z`, ...) formatteras enligt teckensträngen *format* och skrivs ut på skärmen.

Inläsning och utskrift av variabler (forts.)

- Exempel på *formatkoder*:
  (Följer standard för programmeringsspråket C)

`%i`	utskrift av ett heltal
`%`a`.`b`f`	utskrift av tal i decimalform
`%`a`.`b`e`	utskrift av tal i exponentform (litet e)
`%`a`.`b`E`	utskrift av tal i exponentform (stort E)
`\n`	matar fram ny rad

- I formatkoderna anger a hur många positioner ett tal skall uppta medan b anger hur många av de a positionerna som skall vara decimaler.
- Tal skrivs alltid högerjusterade.

Inläsning och utskrift av variabler (forts.)

- *Exempel.*

```
>> r=2, omk=2*pi*r
r =
     2
omk =
   12.5664

>> fprintf('Radien är %i och omkretsen är %5.2f\n',r,omk)
Radien är 2 och omkretsen är 12.57
```

Inläsning och utskrift av variabler (forts.)

- *Exempel.*

```
>> x = 123.4567, y = pi
x =
  123.4567
y =
    3.1416

>> fprintf('x är %10.2e\n y är %9.7f\n',x,y)
x är  1.23e+002
 y är 3.1415927
```

Inläsning och utskrift av variabler (forts.)

- *Exempel.*

```
>> A = [1 2 3 4]
A =
     1     2     3     4

>> fprintf('a11 = %u\na21 = %u\na12 = %u\na22 = %u\n',A)
a11 = 1
a21 = 2
a12 = 3
a22 = 4
```

Variabelnamn, fördefinierade variabler

- Variabelnamn får bestå av:
  1. Små och stora bokstäver (a–z, A–Z) OBS! Ej å, ä, ö, Å, Ä, Ö
  2. Siffror (0–9)
  3. "Underscoretecknet" (_)
- Första tecknet i variabelnamnet måste vara en bokstav
- *Exempel.*

Tillåtna variabelnamn	Otillåtna variabelnamn
`massa`	`mössa`
`uppgift1`	`4tal`
`MATLAB_7p11`	`MATLAB 7.11` (två fel!)

Variabelnamn, fördefinierade variabler (forts.)

- Hur många tecken får ett variabelnamn innehålla? Svar fås med kommandot `namelengthmax`:

```
>> namelengthmax
ans =
    63

>> kanske_onodigt_langt_men_fullt_tillatet_namn = 1
kanske_onodigt_langt_men_fullt_tillatet_namn =
     1
```

Variabelnamn, fördefinierade variabler (forts.)

- *Kom ihåg:* MATLAB skiljer på små och stora bokstäver.
  T.ex. så är `bokstav` och `Bokstav` två *olika* variabler.
- Man bör *inte* använda namn på *matematiska funktioner* (t.ex. `sin`) eller *fördefinierade variabler* (t.ex. `pi`) som variabelnamn.
- Hur vet man om ett namn redan är taget? Använd kommandot `which`:

```
>> which pi
built-in (C:\Program Files\MATLAB\R2010b\toolbox\matlab\elmat\pi)

>> which sin
built-in (C:\Program Files\MATLAB\R2010b\toolbox\matlab\elfun\@double\sin)

>> which sinkadus
'sinkadus' not found.
```

Variabellistor, radering av variabler

- Man kan se vilka variabler man skapat genom att:
  1. Ge kommandot `who` eller `whos`. (Det senare ger en mer detaljerad lista.)
  2. Titta i MATLAB:s *Workspace Browser* uppe till höger i det stora MATLAB-fönstret.
- Man kan radera samtliga sina variabler med kommandot:

```
>> clear all
```

- Vill man bara radera t.ex. variablerna `x` och `A` så skriver man:

```
>> clear x A
```

Läsa och skriva variabler till fil

- När MATLAB avslutas försvinner samtliga variabler som skapats.
- Variabler kan *sparas* i en fil med kommandot `save`.
- Variabler kan *läsas* från en fil med kommandot `load`.

Läsa och skriva variabler till fil (forts.)

Exempel. Antag att du skapat följande variabler:

```
>> clear all              % Raderar samtliga variabler
>> x = 7                  % Tal
>> A = [1 2 3;4 5 6]      % 2 x 3 - matris
>> s = 'Hello world!'     % Teckensträng
```

och att du nu vill spara variablerna i en fil med namnet **trevariabler** i katalogen **Z:\Matlab**. Gör först **Z:\Matlab** till *aktuell katalog*:

```
>> cd Z:\Matlab      % Alt: ändra adress längst upp i MATLAB-fönstret
```

Därefter kan du spara variablerna med kommandot `save`:

```
>> save trevariabler
```

Läsa och skriva variabler till fil (forts.)

Kommandot:

```
>> save trevariabler
```

sparar *samtliga* variabler (`x`, `A` och `s`) i filen **trevariabler.mat** i aktuell katalog (**Z:\Matlab**).

Ändelsen **.mat**, som MATLAB automatiskt lägger till filnamnet om man inte anger någon annan ändelse, anger att variablerna sparats i en s.k. MAT-fil. Detta är ett speciellt format som MATLAB använder.

Allmän syntax:

```
>> save filnamn
```

sparar *samtliga* variabler på filen **filnamn.mat** i aktuell katalog.

Läsa och skriva variabler till fil (forts.)

Antag nu att du avslutat och startat om MATLAB, eller skrivit:

```
>> clear all
```

så att samtliga variabler är borta:

```
>> x
??? Undefined function or variable 'x'.

>> A
??? Undefined function or variable 'A'.

>> s
??? Undefined function or variable 's'.
```

Läsa och skriva variabler till fil (forts.)

Med kommandot `load` kan du läsa de sparade variablerna:

```
>> load trevariabler   % Man kan, men behöver ej, ange ändelsen .mat

>> who
Your variables are:
A  s  x

>> s
s =
Hello world!
```

Läsa och skriva variabler till fil (forts.)

Allmän syntax:

```
>> load filnamn
```

läser *samtliga* variabler från filen **filnamn.mat**

Man behöver inte läsa samtliga variabler:

```
>> clear all
>> load trevariabler x A
>> who
Your variables are:
A  x
```

läser *endast* variablerna `x` och `A` från filen **trevariabler.mat**

Läsa och skriva variabler till fil (forts.)

Man kan analogt välja att spara endast vissa av sina variabler:

```
>> clear all, load trevariabler, who
Your variables are:
A  s  x

>> save tvavariabler s x
```

sparar *endast* variablerna `s` och `x` i filen **tvavariabler.mat**:

```
>> clear all, load tvavariabler, who
Your variables are:
s  x
```

Läsa och skriva variabler till fil (forts.)

- MAT-filer är *binär-filer*, där data lagras i ett "MATLAB-format"
- Om man vill att andra program, t.ex. en texteditor, skall kunna läsa de filer med data man sparat kan man istället spara variabler i vanliga *text-filer* (*ASCII-filer*)

Läsa och skriva variabler till fil (forts.)

Exempel.

```
>> B = [1 2; 3 4; 5 6]
>> save B.txt B -ascii
```

sparar matrisen `B` i ASCII-filen **B.txt**. Denna fil ser ut på följande sätt:

```
1.0000000e+000  2.0000000e+000
3.0000000e+000  4.0000000e+000
5.0000000e+000  6.0000000e+000
```

- Matriselementen sparas på *tio-potensform* med 8 siffrors noggrannhet som default.
- Vi har här valt matrisens namn som filnamn, med ändelsen **.txt** för att indikera att det är en vanlig text-fil.

Läsa och skriva variabler till fil (forts.)

Vill man spara matriselementen med 16 siffrors noggrannhet (*dubbel precision*) får man ange det:

```
>> C = [pi; sqrt(2); exp(1)]
>> save C.txt C -ascii -double
```

sparar matrisen `C` i ASCII-filen **C.txt** med dubbel precision. Denna fil ser ut på följande sätt:

```
3.141592653589793e+000
1.414213562373095e+000
2.718281828459045e+000
```

Läsa och skriva variabler till fil (forts.)

Vi kan även hämta variabler sparade i ASCII-filer med kommandot `load`:

```
>> clear all, load B.txt, load C.txt

>> format long, B, C
B =
     1     2
     3     4
     5     6
C =
   3.14159265358979
   1.41421356237310
   2.71828182845905
```

Läsa och skriva variabler till fil (forts.)

Kommentar.

- Man kan också skriva:

```
>> load B.txt -ascii
>> load C.txt -ascii
```

  men det behövs inte. En fil vars namn har en annan ändelse än **.mat** antas vara en ASCII-fil.

- Notera att när vi skriver:

```
>> load filnamn.txt
```

  hämtas innehållet i filen **filnamn.txt** och placeras i en variabel med namn `filnamn`.

Läsa och skriva variabler till fil (forts.)

Man kan förstås hämta data från ASCII-filer som skapats av andra program än MATLAB. Betrakta t.ex. följande ASCII-fil **data.txt**:

```
0.90 0.76 1.67 1.44
1.80 1.36 2.92 2.61
```

Dessa data kan hämtas in i matrisen `data`:

```
>> clear all, load data.txt, who
Your variables are:
data

>> data
data =
    0.9000    0.7600    1.6700    1.4400
    1.8000    1.3600    2.9200    2.6100
```

Formatterad inläsning och utskrift till fil

- Med kommandona `save` och `load` kan man, som vi sett ovan, på ett relativt enkelt sätt *skriva* till respektive *läsa* från MAT-filer och text-filer.
- Detta ger dock inte så stor valfrihet.
- Ibland vill man mer *i detalj kunna styra formatet*:
  1. Du kanske vill spara tal, som t.ex. representerar priser i kronor och ören, med *precis 2 decimaler i en textfil.*
  2. Eller så kanske du vill lagra *heltal i en binärfil* (till skillnad från MAT-filer som är binärfiler där tal normalt lagras som flyttal)
- I fall som dessa måste man använda mer "komplicerade" kommandon:
  `fwrite` och `fread` (för binär-filer)
  `fprintf` och `fscanf` (för text-filer)

Filformat

Man kan importera och exportera ett stort antal olika filformat till och från MATLAB. Här är några exempel:

- Ljudfiler: **.au**, **.wav**
- Bildfiler: **.bmp**, **.jpeg**, **.tiff**, **.gif**
- Videofiler: **.avi**
- Kalkylblad: *Lotus:* **.wk1**
  Excel: **.xls** (kan importeras och under vissa förutsättningar exporteras)

Exempel.

- En ljudsignal som är sparad som en **.wav**-fil importeras till MATLAB. Därefter görs en frekvensanalys med hjälp av verktyg som finns i MATLAB. Resultatet, som presenteras grafiskt i en MATLAB-figur, exporteras därefter som en **jpeg-**bild.

Filformat (forts.)

Det finns ett grafiskt användargränssnitt, *Import Wizard*, som underlättar inläsning av många olika filformat, t.ex. kalkylblad, binära filer, text- och grafikfiler.

Du kan starta *Import Wizard* genom att välja **Import Data...** från MATLAB-menyn **File**, eller med kommandot:

```
>> uiimport
```