Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 1

Løsningsforslag

Oppgave 2 – Litt aritmetikk

a) Her har vi skrevet ut det som kommer opp i kommandovinduet når vi utfører operasjonene.

```
> 2+2
ans =  4
> 3-2
ans =  1
> 3*2
ans =  6
> 6^2
ans =  36
> 6/2
ans =  3
```

Vi kan gjerne utføre kompliserte og "styggere" utregninger også:

```
> 3^2.12*(2+4*(-3.1))/(2^2-1)
ans = -35.597
```

Vi får da ofte bruk for paranteser.

b) I kommandovinduet får vi:

```
> sin(0.7)
ans =  0.64422
> log(5)
ans =  1.6094
> sqrt(9)
ans =  3
> exp(1)
ans =  2.7183
> atan(2)
ans =  1.1071
```

'`sin`' er sinus-funksjonen, '`log`' er den naturlige logaritmen, $\ln x$, '`sqrt`' er kvadratrotfuksjonen, '`exp`' er eksponentialfunksjonen, e^x, og 'atan', er den inverse tangensfunksjonen, $\arctan x$. Legg merke til at argumentet skal stå i en parantes, og at desimaltall blir angitt med punktum – ikke komma.

c) Vi forsøker å regne ut disse uttrykkene i MATLAB:

```
> 1/0
warning: division by zero
ans = Inf
```

Her blir vi advart mot å gjøre en ting vi vet vi ikke har lov til å gjøre i matematikk: Å dele på 0. Vi ser også at MATLAB velger å gi 'uendelig' som svar.

```
> 0^0
ans =  1
```

Her har også MATLAB gjort en 'tolkning' som kan diskuteres. I matematikk kan man både argumentere for at 0^0 er 1 siden x^0 er èn for alle andre verdier av x. Men man kan også argumentere for at det er 0, siden $0^x = 0$ for alle andre verdier av x enn 0. Vi ser at MATLAB har valgt den første tolkningen.

```
> 5/Inf
ans = 0
```

Dette virker vel rimelig, gjør det ikke? Det gjør i alle fall det om vi tolker `5/Inf` som grenseverdien $\lim_{x\to\infty} 5/x$.

```
> 0*Inf
ans = NaN
> Inf/Inf
ans = NaN
```

Det virker også rimelig at hverken "$0\cdot\infty$" eller "∞/∞" blir gitt noen verdi; disse uttrykken gir ikke mening matematisk.

```
> 10^999
ans = Inf
```

Her blir en av de fundamentale forskjellene mellom numerisk og analytisk matematikk tydelig; selv om 10^{999} er et veldig stort tall, er det slett ikke uendelig, slik MATLAB 'påstår'. Men der er grenser for hvor store tall MATLAB er i stand til å håndtere; dette tallet var tydeligvis for stort. På tilsvarende måte er MATLAB også ute av stand til å forholde seg til uendelig små tall. Der finst et minste tall, ϵ, som er slik at tall mindre enn dette, blir regnet for 0 av MATLAB.

```
> 1.2e4
ans =  12000
```

Dette er måten vi skriver tall på normalform i MATLAB; '`1.2e4`' betyr altså $1.2 \cdot 10^4$.

```
> NaN+1
ans = NaN
```

At et udefinert tall pluss én forblir et udefinert tall er vel i grunn ganske opplagt.

d)
```
> pi
ans =  3.1416
> i
ans =  0 + 1i
> eps
ans = 2.2204e-016
```

Det bør her legges til at MATLAB opererer med en langt mer nøyaktige verdi av π enn det antall desimaler tilsier her. Bare et visst antall desimaler blir skrevet til skjermen. Vi kan velge å endre dette formatet. Hvis du vil ha flere desimaler, for eksempel, kan du skrive '`> format long`':

```
> format long
> pi
ans =  3.14159265358979
```

Man kan synes det er litt pussig at ikke Eulers tall, e, er tilordna til `e` fra før. Uansett er det ikke verre enn at man kan skrive '`>e=exp(1)`' for å ordne dette.

Vi kan gå tilbake til det kortere formatet igjen ved å skrive '`format short`'.

Oppgave 3 – Tilordning

a) Slik vil det se ut når vi sier at variabelen x skal ha verdien 7:

```
> x=7
x =  7
```

Merk at vi nå i tillegg til selve tilordninga også har sagt at *der skal være* ein variabel som heter `x`. Vi sier at vi har *initiert* variabelen.

Resultatene av reknestykkene blir:

```
> x=7
x =  7
> x*2
ans =  14
> x^2
ans =  49
> y=2
y =  2
> x/y
ans =  3.5000
```

Dersom det skulle bli litt mange variable å holde styr på, kan man fjærne variable med kommandoen `clear`; for eksempel kan man slette `y` ved å skrive '`>clear y`'. Om man vil "renske tavla" helt, gjør man det ved å skrive '`>clear all`'.

b) Symbolet `'='` betyr ikke helt det samme i MATLAB-sammenheng som det gjør i matematisk notasjon; her er likhetstegnet brukt som tilordning; altså for å gi en variabel en verdi. Derfor vil følgende gi mening i MATLAB:

```
> x=2
x =  2
> x=x+1
x =  3
```

Vi ser at den siste kommandoen la én til variabelen x og tilordna svaret tilbake til x; den *gamle* x var 2 og den *nye* x ble 3. Matematisk er den siste linja meningsløs siden den gir at "$x = x + 1 \Leftrightarrow x - x = 1 \Leftrightarrow 0 = 1$". Om vi ønsker å uttrykke likhet, ikke tilordning, i MATLAB, skriver vi '`==`' – altså et dobbelt likhetstegn. Dette skal vi komme tilbake til senere.

c) Vi tilordner i kommandovinduet:

```
> pi
ans =  3.1416
> pi=10
pi =  10
```

Vi ser at `pi`, som var π, nå har fått verdien 10. Tilsvarende skjer også med `i`. Vi ser at våre tilordninger overstyrer de som MATLAB måtte ha fra før. Dette gjelder også om vi gir variable navn på funksjoner som MATLAB har innebygd. Dette skal vi komme tilbake til.

Oppgave 4 – Vektorer

a) Variablene i MATLAB kan være tall, vektorer eller matriser. Vi kan for eksempel gi vektoren $\mathbf{x} = [1, 0, -3]$ på denne måten:

```
> x=[1, 0, -3, 7]
x =

   1   0  -3   7

> x(2)
ans = 0
```

`x(2)` gir altså den andre komponenten i vektoren $\mathbf{x}$. Tilsvarende gir 'textttx(end)' den siste komponenten i $\mathbf{x}$:

```
> x(end)
ans = 7
```

Vi kan også referere til flere komponenter samtidig:

```
> x(1:3)
ans =

   1   0  -3

> x(3:end)
ans =

  -3   7
```

Om vi avslutter ei kommandolinje med semikolon, vil ikke noe bli skrevet til skjerm. Det betyr ikke at kommandoen ikke har blitt utført. Det har blitt gjort, vi bare slipper å *se* resultatet. Om vi for eksempel skriver

```
>y=2+3;
```

så har faktisk variabelen `y` blitt 5 – selv om det ikke har blitt skrevet til skjerm.

b) Vi får:

```
> x^2
Error using ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

Vi ser at MATLAB "nekter" å regne ut x^2. Og det er ikke så rart; operasjonen er ikke veldefinert matematisk. Retnok har vi definert *skalarprodukter* for vektorer, men det blir noe annet. MATLAB tolker multiplikasjonstenget '`*`' det som en matrise-multiplikajson, og gitt at `x` er en vektor, er ikke multiplikasjonen vi ba MATLAB utføre, en meningsfull operasjon. Om vi tar med punktum foran potens-sybolet, derimot, blir det tolket som at man skal kvadrere elementvis:

```
> x.^2
ans =

   1   0   9
```

Vi får altså vektoren der hvert element blir kvadrert, $[x(1)^2\ x(2)^2\ x(3)^2]$, til svar.

c) `1:5` gir som svar `1 2 3 4 5`. Tilsvarende kan heltallene fra og med -3 til og med 3 skrives slik: `-3:3`. En vektor bestående av tallene 0, 0.25, 0.5, ..., 1.75, 2 kan uttrykkes slik: `0:.25:2` ('.25' betyr det samme som 0.25).

d) Vektoren `y` kan tilordnes slik:

```
> y=2:2:10
y =

    2    4    6    8   10
```

Kommandoen `length` brukes for å finne ut hvor "lang" en vektor er – altså hvor mange elementer den har:

```
> length(y)
ans =  5
```

`y` har altså 5 elementer. Kommandoen `fliplr` vil snu rekkefølgen elementene kommer i:

```
> fliplr(y)
ans =

   10    8    6    4    2
```

Om vi søker opp dokumentasjon for `fliplr`-funksjonen, får vi opp dette:

```
fliplr

Flip matrix left to right
expand all in page
Syntax
```

```
B = fliplr(A)
Description

B = fliplr(A) returns A with columns flipped in the left-right direction,
that is, about a vertical axis.

If A is a row vector, then fliplr(A) returns a vector of the same length
with the order of its elements reversed. If A is a column vector, then
fliplr(A) simply returns A.
Examples

If A is the 3-by-2 matrix,

A =
    1    4
    2    5
    3    6

then fliplr(A) produces

    4    1
    5    2
    6    3

If A is a row vector,

A =
    1    3    5    7    9

then fliplr(A) produces

    9    7    5    3    1

Limitations

The array being operated on cannot have more than two dimensions.
This limitation exists because the axis upon which to flip a
multidimensional array would be undefined.
See Also

flipdim | flipud | rot90
```

-Grundig dokumentasjon, altså – med både eksempler og henvisninger til andre lignende funksjoner.

e) Tilordning:

```
> Vektor=[9 pi -2 100 8]
Vektor =

     9.0000     3.1416    -2.0000   100.0000     8.0000
```

Sortering heter *sort* på engelsk. Om vi søker på dette, finner vi fort at der er en rutine som heter sort. Vi kan også lese om hvordan den fungerer. Vi finner nok fort ut at den gjør det vi ønsker:

```
> sort(Vektor)
ans =

    -2.0000     3.1416     8.0000     9.0000   100.0000
```

Jo, så menn, tallene kom i stigende rekkefølge!

Denne skrivemåten, navnet på funksjonen etterfulgt av det som vi skal utføre funksjonen på i parantes, er ganske gjennomgående i MATLAB.

Vi kan gå fram på same måte for å finne en summeringsfunksjon. Vi finner fort ut at fuksjonen '`sum`' gjør jobben:

```
> sum(Vektor)
ans =  118.14
```

Det er ikke meninga at vi skal pugge navnet på en masse funksjoner i MATLAB. Men det er en klar fordel å lære seg å lete etter de funksjonene vi kunne ha bruk for eller nytte av. Men generelt er det ganske få funksjoner vi vil ha direkte bruk for i dette kurset.