

Introduksjon til Matlab

Kristine Hein

Jan 16, 2020

En kort innføring i installasjon og programmering i bildebehandling.

Contents

1	Installasjon	1
1.1	Nedlastning til egen PC	1
2	Programmering	2
2.1	Grunnleggende	2
2.2	Bildebehandling	4

Installasjon

Matlab krever en lisens for å kjøres (som koster svimlende 18500kr!). Som student ved UiO har man tilgang til en lisens som gjør det mulig å laste ned Matlab på egen PC, eller bruke Matlab gjennom UiOs [programkiosk](#).

Nedlastning til egen PC

- 1) Gå inn på [mathworks](#) og registrer en bruker med din UiO-mail som brukernavn.
- 2) Trykk **Associate to license** og skriv inn *13796-65616-24271-45993-03655*.
- 3) Nå kan du laste ned Matlab ved å trykke på **Download Your Products now** og følge de gitte instruksjonene for den platformen du laster ned på.

Tips til Linux-brukere: Det kan hende Matlab har en keybinding som minner om Emacs. Det vil si, at typiske kommandoer som f.eks Ctrl+S for lagring, Ctrl+C og Ctrl+V for hhv. kopiering og lagring gjør noe annet enn det man kanskje er vant til. For å endre denne keybindingen, går man på **Preferences -> Keyboard -> Shortcuts**. Under **Active settings** kan du finne en drop-down meny med mulighetene for *Emacs Default Set* eller *Windows Default Set*. Ved å bruke *Windows Default Set*, vil det være mulig å bruke de typiske kommandoene som Ctrl+C, Ctrl+V osv.

Programmering

Her kan du finne demonstrasjoner av syntaksen som brukes i Matlab, samt et eksempel på hvordan et program kan utføre operasjoner på et bilde.

Grunnleggende

Matlab er et høynivåspråk, som betyr at Matlab er ment til å være lett å lese.

Mye av semantikken i Matlab bygger også på at man jobber med matriser. Bilder er i bunn og grunn matriser som gjør at Matlab er godt egnet til å jobbe med bilder.

Et program i Matlab har filendingen .m, og kjøres i kommandovinduet ("Command Window") ved å skrive

```
<navn på programmet ditt>
```

I alle Matlab-program er det viktig å huske et semikolon (;) i slutten av hver linje som du ikke ønsker å se verdien til. Det kan fort bli mye utskrift, spesielt ved store matriser. Er man uheldig og glemmer et semikolon, kan man avslutte utskriften ved å trykke Ctrl+C.

Matematikk. Vi kommer til å bruke en del aritmetiske operasjoner (dvs addering, subtrahering, divisjon og multiplikasjon) etterhvert, og da kan det være fint å kjenne til dem. Syntaksen for aritmetiske operasjoner er nokså lik som i Java/C/C++/Matlab.

Programmet under demonstrerer aritmetiske operasjoner, samt potens siden den skiller seg mye ut fra de nevnte språkene.

```
a = -2;  
b = 3;  
  
% Addisjon:  
a+b
```

```
% Subtraksjon
a-b

% Multiplikasjon
a*b

% Divisjon
a/b

% Opphøying
a^b
```

Programmet kan lastes ned [her](#) (høyreklikk og trykk "lagre lenke som"/"save link as")

Cell arrays. Cell arrays er en samling av elementer. Det som gjør cell arrays spesielt i Matlab sammenlignet med Java eller C/C++, er at array-ene kan inneholde elementer av ulik type. Det vil si at én og samme cell-array i Matlab kan inneholde datatyper som strings, ints og floats og andre ting som for eksempel funksjonspekere og andre objekter.

For å lage en tom cell, brukes krøllparantes:

```
cell1 = {}
```

Man kan utvide *cell1* ved å spesifisere på hvilken plass et nytt element skal settes inn. Husk at Matlab begynner indekseringen fra 1. Når vi jobber med cell arrays bruker vi krøllparantes til å indeksere med.

```
cell1{1} = 10; % Nå har liste kun ett element - nemlig 10
cell1{2} = 12; % Elementet 12 er nå plassert bak 10
```

For å lage en cell array som inneholder gitte elementer, brukes krøllparantes rundt elementene separert med komma:

```
a = 1
b = @sum % alpha-tegnet sørger for at b er en funksjonspeker til den innebygde funksjonen sum
c = 'hei'

cell2 = {a,b,c}
```

Når man skal hente ut elementer fra en cell array, er det viktig å huske at første element ligger på 1-te plass (i motsetning til Python der første element ligger på 0-te plass). For å hente ut et element fra en cell array, skriver vi først navnet til variabelen knyttet til array-et, og deretter krøllparantes med et tall inni som forteller fra hvilken plass programmet skal hente elementet fra.

```
cell1{1} % viser 10 i kommandovinduet
cell2{3} % viser 'hei' i kommandovinduet
```

Løkker. Løkker har noenlunde lik syntaks som de typiske høynivåspråkene.

For-løkker: Hensikten bak en for-løkke er å utføre en kodesnutt et bestemt antall ganger.

For-løkker er kanskje én av de viktigste redskapene som kommer til å bli brukt i emnet. Derfor er det viktig å ha god kjennskap til hvordan den kan brukes og *spesielt* indekseringen.

Konstruksjonen av en for-løkke gjøres slik:

```
for <variabel> = <startverdi>:<steg>:<sluttverdi>
% Kode som skal utføres
end
```

Det som er viktig å huske på å legge til i slutten av løkken, er `end`. Den brukes til å markere slutten av if-tester, samt funksjonsdefinisjoner. Det er mulig la være å spesifisere *steg* hvis man ønsker at det skal være lik 1. Syntaksten blir da

```
for <variabel> = <startverdi>:<sluttverdi>
% Kode som skal utføres
end
```

While-løkker: Det hender også at man ikke vet på forhånd hvor mange ganger en kodesnutt skal utføres av programmet. Dette kan f.eks være ved fjerning av piksler som vi ikke ønsker i bildet og søk etter noe i bildet.

En while-løkke er konstruert slik:

```
while <en betingelse er evaluert til True>
% kodesnutt som skal utføres
end
```

En betingelse kan være en sjekk på om ett tall er større enn en gitt verdi, om en variabel har endret seg osv.

Matriser. En matrise konstrueres radvis, og inni klammeparenteser. En rad er definert som en rekke av verdier separert av mellomrom. For å indikere slutt av en rad, brukes semikolon. Et eksempel på konstruksjon av en 3 x 2 - matrise:

```
matrise1 = [1 2 3; 4 5 6];
```

som ser ut slik i kommandovinduet:

```
matrise1 =
     1     2     3
     4     5     6
```

Merk at for at matrisen skal være veldefinert, må radene være like lange (altså bestå av like mange verdier).

Programmet under viser grunnleggende operasjoner man kan gjøre med en matrise:

```

A = [1 2 3; 4 5 6];
a = 3;

% Legge a til hvert element i A
A + a

% Trekke a fra hvert element i A
A - a

% Multipliserer hvert element i A med a
A.*a

% Del hvert element i A på a
A./a

% Opphøy hvert element i A med a
A.^a

% Transponer A (dvs, gjør om radene i A til kolonner)
A.'

B = [10 11;12 13;14 15];

% Matrisemultiplikasjon
A*B

```

Programmet kan lastes ned [her](#) (høyreklikk og trykk "lagre lenke som"/"save link as")

Bildebehandling

Det er mange ulike teknikker en kan bruke for å arbeide med bilder. Her vil det presenteres én metode å arbeide med bilder på, men du står selvfølgelig fritt til å bruke teknikkene som passer deg best.

Innlesing og lagring av bilde. Innlesing av bilde, f.eks [houses.png](#), i Matlab gjøres ved å bruke funksjonen *imread*:

```
img = imread('houses.png')
```

I emnet kommer vi til å jobbe hovedsakelig med gråtonebilder. Det hender at de bildene som gis, er fargebilder eller leses av som om de har tre fargekanaler. For å være på den sikre siden, kan man inkludere dette i programmet før man jobber videre med det:

```

img = imread(<filnavn til ditt bilde>)
if ndims(img) == 3 % det innleste bildet er et fargebilde
    img = rgb2gray(img) % konvertér til gråtonebilde
end

```

If-testen sjekker om antall dimensjoner i det innleste bildet er 3. Fargebilder er 3-dimensjonale, mens gråtonebilder er 2-dimensjonale. Vet programmet at bildet har 3 dimensjoner, så betyr det altså at bildet er et fargebilde.

Det er også *veldig* viktig å passe på at bildet programmet jobber med, er av riktig datatype. De innleste bildene er av typen *uint8*, som betyr at verdiene er heltall. Dette kan gi store numeriske feilberegninger hvis man utfører transformasjoner på bildet siden verdiene blir rundet til heltall mellom 0 og 255. Derfor er det en god vane i å konvertere det til f.eks *double* før man utfører matematiske operasjoner på bildet,

```
img = double(img)
```

Fremvising av bilde. Nå som et bilde har blitt innlest, er det fint å få se det også. Dette kommer til å være *svært* nyttig for å se hvordan diverse operasjoner påvirker et bilde.

Det finnes ulike funksjoner i Matlab for fremvising av bilde. En funksjon som kan brukes er *imshow*:

```
imshow(img, [])
```

De tomme klammeparantesene forteller at Matlabs fremvising av verdier er begrenset av innbildets minste og største verdi. Etter noen operasjoner på bildet (og gitt at bildet er konvertert til *double*), kan det hende den har verdier større enn 255 (som et typisk bilde har). For at resultatet av operasjonen skal vises tydelig frem, kan en bruke

```
imshow(img, [0 255])
```

for at Matlab framviser bildet som om det har verdier mellom 0 og 255.

Noen ganger kan det hende at det framviste bildet er lite i forhold til vinduet. Da kan man bruke et ekstra argument, *InitialMagnification* slik:

```
imshow(img, [0 255], 'InitialMagnification', 'fit')
```

Da vil bildet skaleres opp i forhold til vinduets størrelse.

Programmet under demonstrer det som har blitt beskrevet til nå:

```
img = imread('houses.png');
if ndims(img) == 3
    img = rgb2gray(img);
end
img = double(img);

img_contrast = img.*1.5;

imshow(img_contrast, [], 'InitialMagnification', 'fit')
title('Et gråtonebilde med økt kontrast?')

figure() % For å lage et nytt vindu
imshow(img_contrast, [0 255], 'InitialMagnification', 'fit')
title('Et gråtonebilde med økt kontrast')
```

Programmet kan lastes ned [her](#) (høyreklikk og trykk "lagre lenke som"/"save link as")

som gir disse bildene:

Et gråtonebilde med økt kontrast?



Figure 1: Bilde av det første vinduet.

Et gråtonebilde med økt kontrast



Figure 2: Bilde av det andre vinduet.

Dette er også viktig å bruke **figure()** hvis man ikke ønsker at det nye bildet som fremvises overlapper bildet som tidligere har blitt sendt inn til *imshow*.

En liten demonstrasjon. Denne demonstrasjonen oppsummerer det som har vært beskrevet i denne innledningen. Dette er kanskje en av de funksjonskallene som ofte vil bli brukt i programmene, og som er ganske nyttige for å se hvordan et bilde forandrer seg i løpet av et program.

Bildebehandling er et fag der eksperimentering er viktig, og da er det nyttig å vise fram og presentere resultatene programmet har fått.

```
img = imread('houses.png');
if ndims(img) == 3
    img = rgb2gray(img);
end
img = double(img);

% Øke kontrast
img_increased_contrast = img*1.5;

% Gjøre bildet lysere
img_brighter = img + 100;

figure()
subplot(2,1,1) % subplot deler et vindu i flere delplott.
imshow(img_increased_contrast,[0 255],'InitialMagnification','fit')
```



```
title('Økt kontrast')  
  
subplot(2,1,2)  
imshow(img_brighter,[0 255],'InitialMagnification','fit')  
title('Økt lyshet')
```

Programmet kan lastes ned [her](#) (høyreklikk og trykk "lagre lenke som"/"save link as")

God programmering!