Introduksjon til Marinteknikk

MATLAB notater MAS124

Gloria Stenfelt

Høgskulen på Vestlandet
gste@hvl.no

Vad er MATLAB?

- Beregningsverktøy som bruker et spesifikt programmeringsspråk, på samme måte som JAVA, C-kod, python
- Brukes over hele verden i industrien, forskning og utdanning
- Enklere for ingeniører å forstå og bruke, da det er bygget opp på hvordan vi bruker matematikken
- Står for MATrix LABoratory
- Alle verdier lagres som matriser – enkelt at gjøre lineær algebra
- Kan behandle store datamengder og analysere data raskt
- Simulering, Optimering, regulering og mange andre nyttige toolboxes finnes.

www.mathworks.com

Start av MATLAB

Installere ved hjelp av Matlab_Install_Guide.pdf som finnes på Canvas:
Kompendier/Notater -> Kompendier_Books -> Matlab_UserGuides

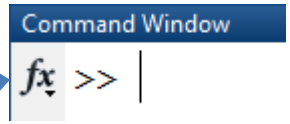
Begynn med:

Anvisninger i Øving 1 : *Ov1.pdf*

MatlabIntro.pdf på Canvas (Kompendier_Books/Matlab_info)

MatlabUserGuide.pdf også på Canvas

Hjelp til selvhjelp, skriv i Command Window:



`>>doc` og søk i dokumentasjonen

Søk under funksjonstegnet etter funksjoner, samt når du vet funksjonsnavnet

For eksempel sin for sinus:

`>> help sin` (kort beskrivning)

`>> doc sin` (flere eksempler på hvordan du bruker funksjonen)

Start av MATLAB

Vi lager skript i *Editor* vinduet for å kunne lagre våre beregninger og kunne gjenta og endre i disse ved seinere tilfelle. (*Se anvisninger Ov1.pdf hvordan du setter opp vinduer og lager navn – ikke lov med å æ ø eller mellomrom.)*

Command Window kan brukes som en helt vanlig kalkulator.

Et nytt skript (*Editor* vindu) skal alltid begynne med `clear all` for å være sikker at gamle variabler som er definert i tidligere beregninger ikke er aktive.

Kommentarer som ikke er med i beregningen må skrives etter %-tegn.
Kommenter skripten sånn at man husker vad som har gjorts samt enheter.

```
% Oppgave 1
T=10; % [m] Dypgang
```

Variabler

Variabler må defineres før man bruker de. Ellers får man feilmelding:

```
>> a*b
Undefined function or variable 'a'.
```

I et skript må man definere opp ifra og nedover sånn at alt er definert når formel skrives:

```
T=1
B=2
L=4
nabla=T*B*L
```

Semikolon etter en definisjon gjør at verdien ikke skrives ut i Command Window:

```
T=1;
```

Matrisemultiplikasjon (liten huskehjelp)

MATLAB «ser» alle verdier som matriser og beregninger med operatorene $*$ $/$ $\hat{}$ gjøres automatisk etter regler for matriseoperasjoner:

$$[1\ 2\ 3] * \begin{bmatrix}1\\2\\3\end{bmatrix} = [14]$$

1x3 3x1 1x1

må være lik for å kunne multiplisere

Størrelse på svar

$$\begin{bmatrix}1\\2\\3\end{bmatrix} * [1\ 2\ 3] = \begin{bmatrix}1 & 2 & 3\\2 & 4 & 6\\3 & 6 & 9\end{bmatrix}$$

3x1 1x3 3x3

eller $$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} * \begin{bmatrix}1 & 2 & 3 & 4\\5 & 6 & 7 & 8\\1 & 2 & 3 & 4\end{bmatrix} = \begin{bmatrix}14 & 20 & 26 & 32\\35 & 50 & 65 & 80\end{bmatrix}$$

2x3 3x4 2x4

Elementvis multiplikasjon

For å kunne multiplisere tall i to vektorer **elementvis**, hvilket er av større nytte i denne kurs må man bruke punkt fremfor operatorene:

```
.*   ./   .^
```

Då går det at multiplisere og dividere en vektor med et tall eller med en vektor av samme størrelse:

```
>> x=0.5 / [3   6   8   10   12]; % Uten punkt
Error using  /
Matrix dimensions must agree.
>> x=0.5 ./ [3   6   8   10   12]; % Med punkt
ans = 0.1667    0.0833    0.0625    0.0500    0.0417

>> x=[ 1 2 3]; y= [3 4 5]; f=x.*y
ans =  3     8     15
```

Vektorer

Matriser med størrelse 1xn eller nx1 kalles også for vektorer .

$[1\ 2\ 3]' = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ Transponat av en vektor (eller matrise)

Radvektor 1x3

```
>>[ 1  2  3];
```

Kolonnvektor 3x1;

```
>>[ 1  2  3]'
```

eller

```
>>[ 1; 2; 3]
```

Lengde og størrelse

Vi har en vektor for dypgang

T=[3 6 8 10 12]

Lengden på vektorn = antall verdier:

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

Index nummer 3 har verdi 8, index nummer 5 har verdi 12, plukke ut index:

```
>> T(3)
ans = 8
>> T(end)
ans = 12
```

Størrelse og index (matrise)

Vi har en matrise (Tabell) med dypgang i rad 1 og Oppdriftsenter i rad 2:

```
>> Tab=[3    6   8    10   12;
        1.5  3   4    5    6]
>> size(Tab)
ans = 2    5
```

Plukk ut vektoren for T:

```
>>T=Tab(1,:)   (rad 1, alle kolonner)
```

Plukk ut vektoren for KB:

```
>>KB=Tab(2,:)   (rad 2, alle kolonner)
```

Plukk ut den tredje kolonnen i tabellen :

```
>>KB3=Tab(:,3)   (alle rader, kolonn 3)
```

Grafer via `plot`

Vi plotter figurer med plot kommandoen. X og Y verdiene som plottes må selvfølgelig være av samme størrelse.

```
plot(T,KB)
```

Linjestiler legges til etter hvert x ,y argument, se andre plot «ingredienser» i Ov1.pdf, samt i Command Window skriv `>>help plot` og se for eksempel på linken LineSpec for flere linjemarkeringsalternativ.

```
plot(T,KB,'bx--',T,KG,'r-')
```

Vi hadde bare et konstant KG verdi lik 3 som vi vil plotte mot T.

Istedenfor å skrive 5 ganger,

```
KG=[3 3 3 3 3]
```

så kan vi bruke `ones` for å lage en vektor (matrise) med ett-tall som er 1x5 lang og multiplisere med 3:

```
KG=3*ones(1,5)
```

På samme småte kan man lage en vektor (matrise) med nuller hvis det trengs.

```
>>nuller=zeros(1,5)
ans =   0     0     0     0     0
```

Bruk av handle og plotargument

Typiske argument som brukes i samband med plot, xlabel, ylabel og legend: LineWidth, FontSize, MarkerSize, Location. Flere finns i MATLAB dokumentasjonen, alt kan gjøres!

```
plot(T,KM,'bx-', T,KB,'go--', T, KG,'r-',...
    'linewidth',1.3,'markersize',7)

handle
 ↓
hx=xlabel('T [m]');
hy=ylabel('Height over keel [m]');
hl=legend('KB','KM','KG')
set([hx hy],'fontsize',12)
set(hl,'fontsize',14)
set(gca,'fontsize',12)
print -dpdf Kcurve.pdf
```

Skifte rad: 3 blåe prikker, virker og for vanlig beregn-ingsrad

Bestemmer linjebredde og størrelse på markører

Bestemmer størrelse på tekst for labels og legend

Bestemmer størrelse på tekst for siffrer på aksler (gca-get current axis)

Printer figur i pdf format, den lagres i samme mappe som m-filen

Man kan editere figur i GUI og lagre som m-fil for å få se koden bak

Flere `print`-funksjoner

```
print -dpng Kcurve.png
print -djpeg Kcurve.jpg


print -dpdf ./figs/Kcurve.pdf


print -dpdf ../report/figs/Kcurve.pdf



print -dpdf C:\lab/report/figs/Kcurve.pdf
```

Printer figur i png eller jpeg format, den lagres i samme mappe som m-filen.
`>>doc print` for flere formater

Inn i en underliggende mappenivå

TIP: Zoom in i de lagrede figurene, og man ser at post script basert pdf format gir finest oppløsning!

Printer figur i en mappe *figs* som ligger inne i aktuell mappe for m-filen.

Ut ur aktuell mappenivå

Printer figur i en mappe en nivå ovenfor den aktuelle mappen med navn *report* og undermappe *figs*

Printer i en angitt mappe utifra C:

Plassering av legend

```
hl=legend('KB','KM','KG',1);
hl=legend('KB','KM','KG','location','east')
```

Eller plassere i set:

```
set(hl, 'fontsize',12,'location','southwest')
```

Rask måte, prøv 1,2,3,4 legend plasseres i hjørnene. Prøv!

Plasser etter himmel-retninger, eller prøv `'best'`

Kode tenk

I skriptet skapes det etter hvert noe som vi kaller for kode. En kode skal være fleksibel og robust.

Endrer man størrelse på en vektor i begynnelsen av koden, skal koden være tilpasset og fortsette å virke.

Derfor når vi plotter KG mot T i Øving 1, Oppgave 3:

```
T=1:5
```

`KG=3*ones(1,length(T))` istedenfor «hardkodet» `KG=3*ones(1,5)`

Då oppdateres lengden på KG automatisk når vi endrer til flere dypganger.

```
T=1:0.1:5;
```

Kode tenk forts:

Repetisjoner skal unnvikes:

Vi skrev

```
KG=3;
```

Og seinere bare for å kunne plotte:

```
KG=3*ones(1,length(T))
```

Vi burde ha skrivit:

```
KG=3;
```

Og seinere før plotten:

```
KG=KG*ones(1,length(T))
```

Vi trenger altså kun oppdatere verdien på KG et sted i begynnelsen av koden hvis vi vil endre det.

Kode tenk forts:

Unnvik å døp variabler etter innebygde funksjoner, da vil de ikke gå å bruke til alle variabler er slettet i din *Workspace* med `clear all .`

For eksempel med innebygd `pi` som er verdiet på π. Hvis man definerer en variabel

pi=6;

Så kommer pi at være 6 og ikke verdien på π frem til det du sletter variablene som er aktive i din *Workspace.*

Bruk av `axis`

MATLAB plotter alltid figurer sånn at den fyller opp en kvadratisk formet figur. Akslene kan derfor vise ulik størrelse på skritt i x og y. For å vise figurer i riktig skala forhold kan man etter plot skrive

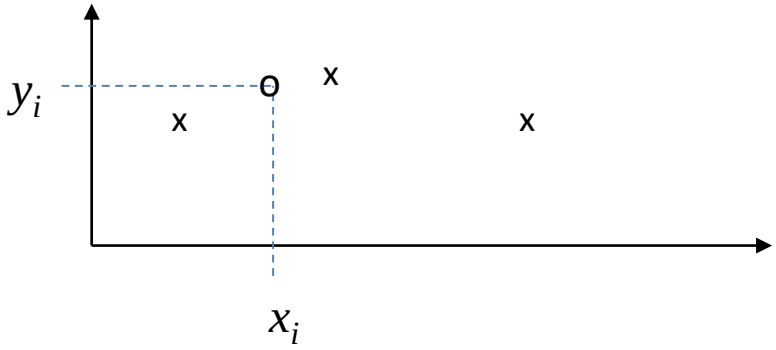
```
axis equal
```

Vil man lage litt luft rundt om de tegnede linjene kan man stille inn rekkevidden på akslene ved

```
axis([0 5 0 8]) % axis([xmin xmax ymin ymax])
```

Bruk av `interp1` og `spline`

Lineær og ulineær interpolasjon: Brukes for å finne ut av verdier mellom gitte x og y verdier i en graf. Gitte punkter tilnærmes med lineær eller ulineær kurve, og då kan verdier langs den nye tilnærmete kurven finnes.



x – gitte punkter
o - vil finne y_i for valgt x_i

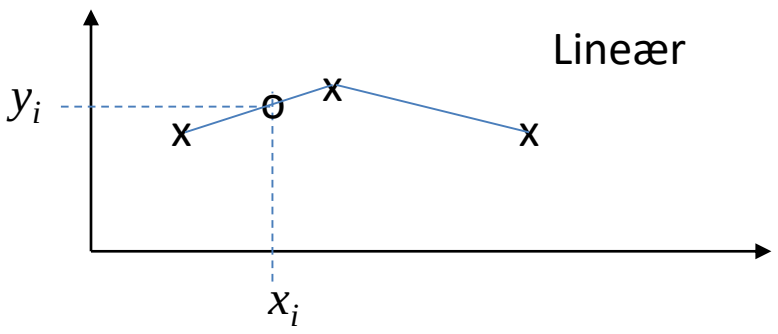
Lineær vs. Ulineær interpolasjon

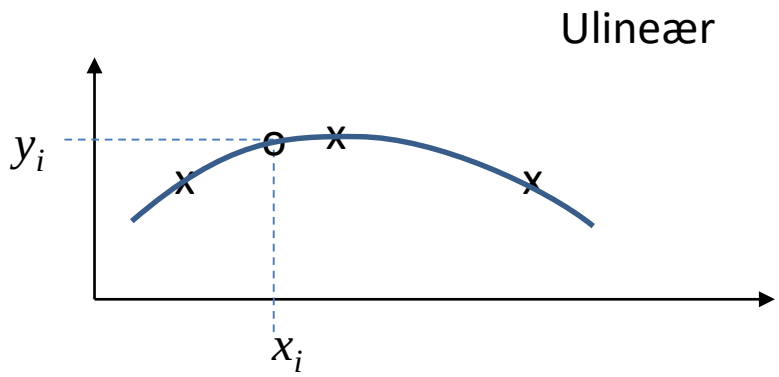
Gitte verdier x og y.
Vil finne y verdi ved en bestemt xi verdi 3.2.
xi kan og være en vektor, dvs man finner flere yi verdier på en gang.

```
x=[2  4  7];
y=[3  5  3];
xi=3.2;
```

Lineaær:
`yi=interp1(x,y,xi);` (Svar: yi=4.2)

Ulineaær:
`yi=spline(x,y,xi);` (Svar: yi=4.52)

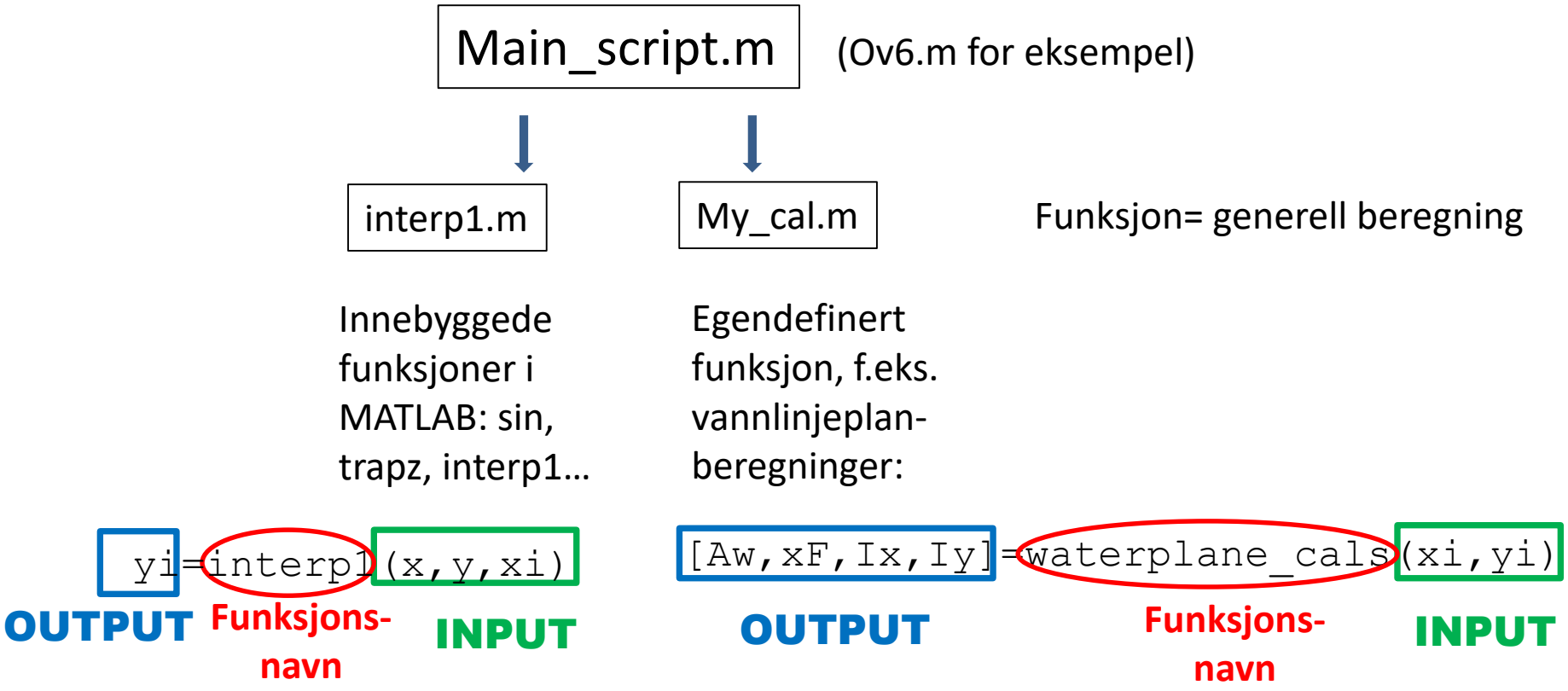




`spline` tilnærmer punktene med en ulineær funksjon etter minste kvadrat metoden. Dvs, den beste tilnærmingen blir funnet, men funksjons uttrykket blir ukjent. Det finnes derfor også muligheter at tilnærme punkter med en viss type funksjon, for eksempel et polynom, når man vet at punktene er fordelt etter denne type funksjon og man vil finne koeffisienter for denne.

Egendefinert `function`

Mange ganger bruker man de samme ligningene om og om igjen og då er det mulig å lagre en egendefinert generell beregning i en funksjon som kalles fra et hovedskript på samme måte som MATLAB's innebyggede funksjoner blir brukt fra hovedskriptet:



Egendefinert `function`

Alle funksjoner er bygget opp av posisjoner separert med komma, så kalte argument.

```
yi=interp1(x,y,xi)
```

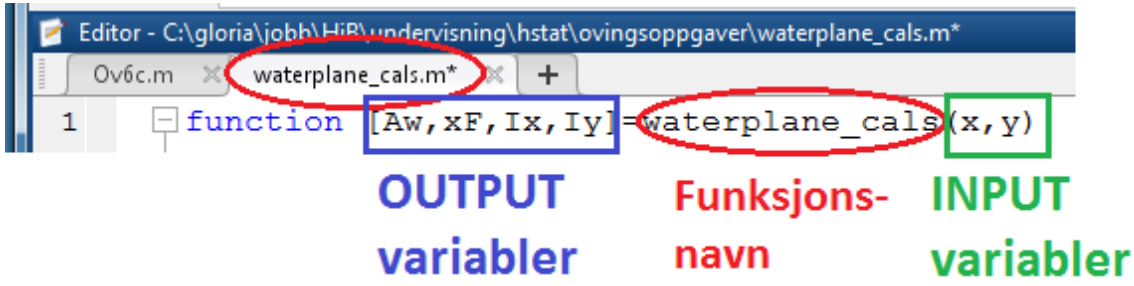
1 output argument
3 input argument

```
[Aw0,xF0,Ix0,Iy0]=waterplane_cals(xi,yi)
```

4 output argument
2 input argument

Det er posisjonen som inneholder verdien å ikke variabelnavnet, det er derfor man kan nevne input og output hva man vil i sitt Main-script, bare man holder kontroll på hvilke beregninger som utføres på den variabelen i den bestemte argument posisjonen.

Skriptet for funksjonen må hete samme sak som selve funksjonen og begynne med funksjonsanrop ved å bruke samme argument struktur.



`for` - loop (Sløyfe)

Brukes ved repetisjon av et bestemt antall beregninger eller programmerte kommandoer. De fleste repetitive beregningene kan gjøres med elementvis multiplikasjon, men `for` loop kan benyttes når elementvis multiplikasjon blir uoversiktlig eller når numerisk iterasjon* er nødvendig for å finne løsninger på problemer. Man kan også stille inn tiden som en for loop skal utføres på ved å bruke `pause`.

```
k=0;

for i=1:6
    f=k*2;
    g(i)=k*2
    k=k+5;
end
```

i : antall gånger (iterasjoner) som sløyfen skal gå
k: variabel som skal endres i hver iterasjon
f,g: funksjonen som verdien k er variabel innom.

Startverdi for variabel k.

f verdien blir oppdatert i hver iterasjon, men blir ikke lagret.

g(i) : i hver iterasjon øker g i lengde og funksjonsverdien lagres i den i:te indeks-posisjonen i vektor g. Dvs g blir direkte en vektor full med lagrede verdier.

Variabel k øker med 5 i hver iterasjon, tenk på når variabelen skal endre verdi, før eller etter funksjonsberegningen gjør forskjell.

*Numerisk iterasjon: For skip kan det typisk handle om å iterere seg frem til likevektstilstand (feks ved: Newton method)

Plot innenfor og utenfor en `for` loop, eks.

```
L=64;
B=10;
Tv=1:0.1:5;
Ix=L*B^3/12;
```

kasseform

T: Variabel som skal endres i hver iterasjon
i: antall iterasjoner=antall T verdier=antall index i T

Plukker ut den i:te verdien, dvs bestemmer aktuell variabelverdi T for denne iterasjon, (istedenfor k=k+5)

```
for i=1:length(Tv)
    T=Tv(i);
    KB=T/2;
    nabla=L*B*T;
    KM=KB+Ix/nabla;
    figure(1)
    plot(T,KB,'x',T,KM,'o'), hold on
end
```

Alternativ 1:
Gjør beregninger uten å lagre de som vektorer, plotter direkte som en punkt og holder kvar figuren, sånn att punkter fylles på i hver iterasjon.
Medfører at, linjestiler er vanskelige å endre.

```
for i=1:length(Tv)
    T=Tv(i);
    KB=T/2;
    KBv(i)=KB;
    nabla=L*B*T;
    KMv(i)=KB+Ix/nabla;
end

figure(2)
plot(Tv,KBv,'-x',Tv,KMv,'-or')
```

Alternativ 2, mest brukbar:
Lagrer beregninger i vektorer og plotter etterpå. Enklere å editere linjestil i figuren.

`while` - loop (kontinuerlig sløyfe)

Brukes ved repetisjon av beregninger/kommandoer som skal slutte når noe spesifikt inntreffer.
(Et PC program som kjører kontinuerlig, frem til man trykker på stopp for eksempel)
I skips verden: trimning av skip numerisk med hydrodynamiske krefter aktive, dvs finne trimvinkelen når ny likevektssituasjon er oppnådd ved en viss fart på skipet.

```
k=5;
i=0;
while k<10
    k=k+1
    i=i+1;
end
```

Bare for å regne antall iterasjoner, kan være viktig å vite etterpå.

k: verdi på denne bestemmer hvor lenge sløyfen skal være aktiv.

Startverdi for variabel k.

Mens verdien er lavere en 10, fortsetter sløyfen sine iterasjoner

Hvordan verdien endrer seg i hver iterasjon. (For en Stopp-knapp i et program, vore det at en STOPbutton aktiveres og endrer verdi fra 0 til 1, dvs:

```
while STOPbutton==0,
    %programmet fortsetter være åpent
    end
```

Kurvetilnærming – polynom med `polyfit`

`polyfit` tilnærmer data med et polynom og finner koeffisientene til polynomet.

```
x=[ 1 2.2 3.5 5.2]
y=[3.4 5.3 9.2 16.5]
p=polyfit(x,y,2)
x_fit=0:0.5:5.5;
y_fit=p(1).*x_fit.^2 + p(2).*x_fit + p(3)
```

$$y = ax^2 + bx + c$$

```
y_fit=polyval(p,x_fit)
```

Målte verdier, som ligner på et 2-ordens polynom

Grad på polynom som skal tilnærme målepunkter.
p: vektor som inneholder de 3 koeffisientene: a, b, c

Lag x og y verdier for tilnærmet kurve

Lag y-verdier ved å bruke ferdig MATLAB funksjon