Ditt og Datt i MATLAB

En introduksjon til Matlab og Simulink for ferske kybernetikk-studenter

Sist oppdatert 17. juli 2014

NTNU
Institutt for teknisk kybernetikk

Innhold

Introduksjon

Heftet er tenkt som et oppslagsverk når du står fast med et konkret problem. Det er i hovedsak rettet mot studenter i emnet TTK4100 - Kybernetikk Introduksjon, og vil bli oppdatert gjennom semesteret.

Gjør deg kjent med hva som står her, du vil få bruk for mye av dette til øvingene. Spesielt seksjonen om Figurer anbefales å gå gjennom.

Ved kommentarer, feil e.l, ta kontakt med vit.ass. i faget. Send også gjerne en mail om du har forslag til nye emner i heftet!

–

Opprinnelig skrevet av Kristian Klausen, januar 2012. Oppdatert av Michael R.P. Ragazzon, siste endring: 17. juli 2014.

1 Generelle tips Matlab

1.1 Kommandovinduet vs `.m`-skript

Nesten all kode i Matlab kan skrives enten i kommando-vinduet (*Command Window*) eller i Matlab-skript, dvs. i egne filer med filnavn `.m`. Resultatet når du kjører koden blir identisk, men det er mange fordeler med å ha all koden din i egne skript. Da kan du nemlig kjøre koden på nytt igjen uten å måtte skrive inn alt på nytt. Tenk på hva som skjer om du ønsker å gjøre en liten endring i starten av koden din. Hadde du bare brukt kommandovinduet måtte du ha startet helt på nytt igjen.

1.2 Få hjelp

Matlab er veldig god til å gi hjelp til konkrete funksjoner. Her bruker du kommandoen `help`, f.eks

```
% Get help on the plot-command
help plot;
```

Dette vil eksempelvis gi deg alle tilpasningsmulighetene til plot-funksjonen. Du kan også markere tekst og trykke på `F1` for å få opp mer info om en funksjon e.l.

1.3 Programflyt

Matlab lar deg bruke løkker, if-setninger o.l. For å lage en løkke som teller fra 1 til 10, går en frem slik:

```
% Creates the vector and starts the loop
for i = 1:10
    do_something(i)
end % Ends the loop
```

Altså ingen paranteser (som i C/C++), og `end` avslutter blokken. If-setninger lages likedan:

```
input = 3;

if input < 5
    disp('Input er et lavt tall!');
else
    disp('Input er litt høyt');
end
```

1.4 Vektorer

For å lage sekvenser av tall, bruker en kolon-operatoren `:`. Brukes slik:

```
% Syntax
v = first : spacing : last

% Creates a vector containing the numbers [first,
%   first + spacing, first + 2*spacing, ..., last]
% The spacing parameter is optional

% Create a vector from 1 to 10
v = 1:10;

% Create a vector containing 1 3 5 7 9
v = 1:2:10;
```

Alternativt kan du velge tall direkte i vektoren ved å bruke klammeparanteser slik:

```
u = [1, 5, 9, 13];
% Både med og uten komma fungerer, men vær konsekvent
v = [-1 0.5 80 4];
```

For å hente ut verdier fra vektoren `v` over kan vi gjøre som følger:

```
v(1)      % Henter ut verdien -1
v(3)      % Henter ut verdien 80

% Du kan også bruke en vektor av indekser
v(2:3)    % Henter ut [0.5, 80]
v([1,3])  % Henter ut [-1, 80]

% Stikkordet 'end' kan brukes slik
v(2:end)  % Henter ut [0.5, 80, 4]
```

1.5 Semikolon

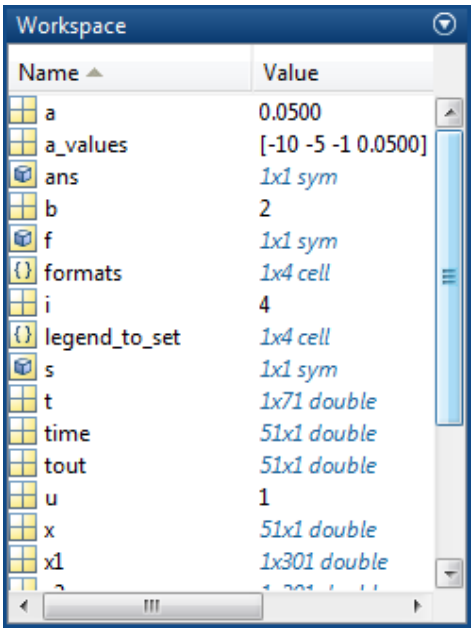
Når du skriver semikolon (`;`) på slutten av en linje, hindrer du at det blir skrevet noe ut til kommandovinduet. Dette er best illustrert ved et eksempel (prøv å kjøre denne koden selv):

```
% Skriver ut variabelen 'a'
a = 1:5

% Her blir verdien ikke skrevet ut
b = 1:3;
```

Ellers er det ingen forskjell med eller uten semikolon.

1.6 Workspace

Alle vektorer og andre variabler du lager blir lagt i Matlabs *Workspace*. Når det ligger en variabel her, betyr det at den er tilgjengelig for bruk der du måtte ønske – enten det er i et `.m`-skript eller som en parameter i et Simulink-diagram. Se Figur 1 for et eksempel på hvordan dette ser ut i Matlab. Du kan dobbeltklikke på de forskjellige variablene for å se på verdiene deres.



Figur 1: Workspace

1.7 Publish

Publish lar deg *publisere* koden og resultatene dine til HTML, PDF, LaTeX, m.m. Dette er meget nyttig i bl.a. øvingssammenhenger. (Ikke bruk dette til rapportskriving!)

I editoren, gå til *Publish*-fanen, og trykk så på pila under *Publish*-ikonet. Hvis du så trykker på *Edit ...* får du opp mange valgmuligheter. Her kan du sette diverse parametere, bl.a. om du vil ha med koden eller ikke. Du kan også endre format. Du kan publisere rett til PDF, eller til LaTeX.

Når du publiserer, vil Matlab kjøre hele `.m` fila di, og få med seg alle plots som blir laget. Den bruker celler (`%%`) for å lage overskrifter. Se under for et enkelt eksempel du selv kan prøve å publiserere:

```
%% Test document
% This text will appear as plain text

%% Task 1
% Here we play with figures

figure(1); clf(1);
plot(1:10);

%% Conclusion
% We conclude that Matlab can publish documents.
```

2 Figurer

Her går vi gjennom litt generelle tips om figurer.

2.1 Aktuelle funksjoner

Funksjoner 1: Relevante funksjoner for figurer

```
% Opprette en figur med nummeret a_number
figure(a_number);

% Tøm/clear en figur
clf(a_number);

% Plotter y mot x, med valgfrie instillinger i 'options'
plot(x, y, options);

% Deler opp figuren. Merk; den plotter ingenting!
subplot(n, m, i);

% Setter på rutenett
grid on;

% "Holder" figuren, slik at flere plots kan tegnes
hold on;

% Zoom
xlim([xmin xmax]); % f.eks xlim([0 5])
ylim([ymin ymax]);

% Setter navn aksene
xlabel('Time (s)');
ylabel('Position (m)');

% Setter navn på grafene i figuren
legend('Funksjon 1', 'Funksjon 2');
```

Data

Funksjonene definert ved x og y vil bli brukt som eksempler gjennom denne seksjonen.

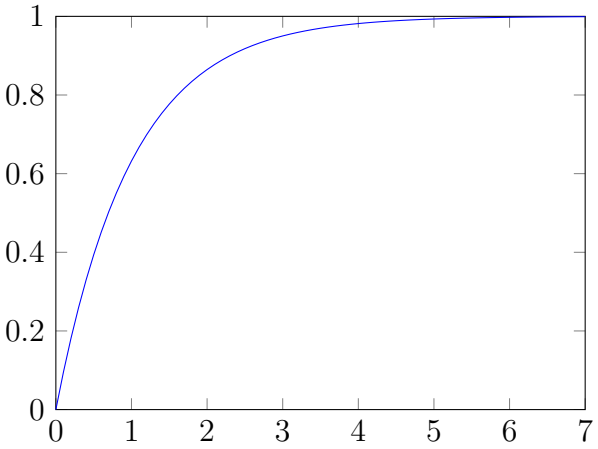
```
% Timevector
t = 0:0.1:7;

% Function to plot: x = 1 - e^{-t}
x = 1-exp(-t);

% Second function: y = e^{-t}
y = exp(-t);
```

2.2 Selve plottet

Alle plots i Matlab lages i en *figure*. Funksjonen `plot` lager bildet i en slik *figure*. Dersom du bare skriver `plot(t, x)`, lager Matlab figuren for deg. Men, det er god vane å gjøre dette selv:

```
% Create and Clear figure nr 1
figure(1); clf(1);

% Plot
plot(t, x)
```

Fordelen med å nummerere figurene selv er at du har full kontroll på dem. Nå som du har nummeret, kan du bruke funksjonen `hgexport(figure_number, ... filename)` for å eksportere figuren. Du kan også bruke menyen for å eksportere figuren til et bilde. Dette gir Figur 2.



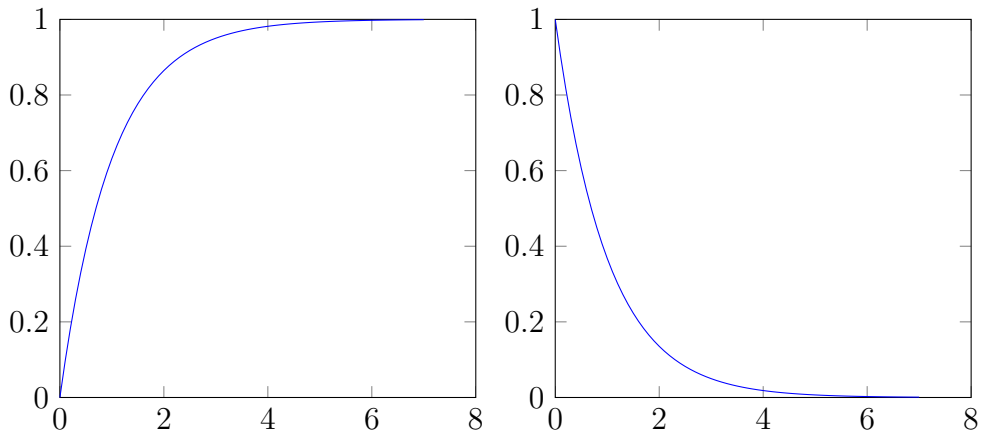
Figur 2: Enkel figur

2.3 Subplots

Bruk kommandoen subplot til å lage flere plots. `subplot(h, b, n)`, hvor `h` er antall plots i høyden, `b` er antall plots i bredden og `n` er nummeret den aktuelle figuren du vil tegne. Nummeret går fra venstre i hver linje. Eksempel:

```
% Lag figuren og clear
figure(2); clf(2);

% Subplot, to i bredden
subplot(1, 2, 1);
plot(t, x);

subplot(1, 2, 2);
plot(t, y);
```

Gir Figur 3.



Figur 3: Subplot

For mer avanserte subplot, se Figur 8 i A.1.

2.4 Farger, grid og zoom

Tredje argument til `plot` angir måten plottet lages på. Under er et par eksempler. TIPS: bruk `help plot` for å liste opp alle alternativene.

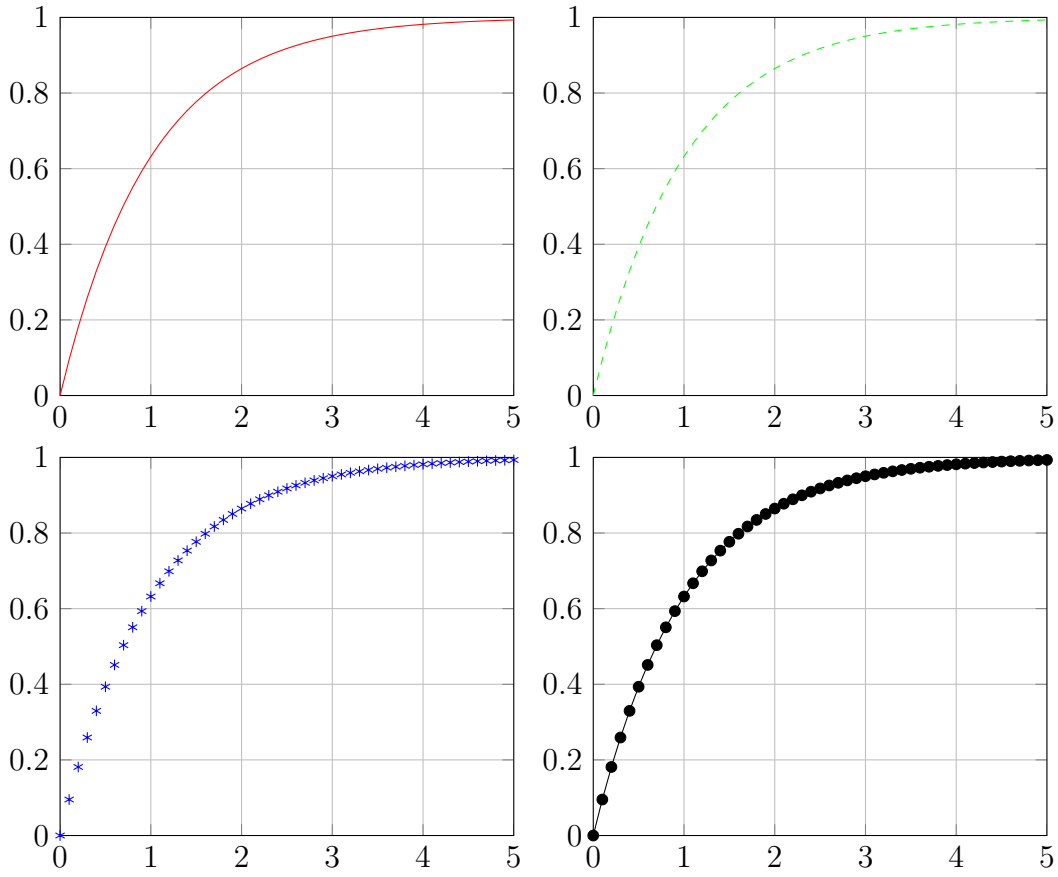
```
figure(3); clf(3);  % Lage figur

subplot(2,2,1)      % Forste subplot
plot(t, x, 'r')     % Red

subplot(2,2,2)      % Andre subplot.
plot(t, x, 'g--')   % Green lined

subplot(2,2,3)      % Tredje
plot(t, x, '*')     % Stjerne

subplot(2,2,4)      % Fjerde, nede til h0yre
plot(t, x, 'k.-')   % Svart, dashdot
```

Rutenett lages ved `grid on;`. Zoom gjøres enklest ved `xlim([xmin xmax])` og `ylim([ymin ymax])`.

Merk at det finnes en bedre måte å gjøre de fire subplottene på, ved å bruke *celler*. Mer om dette senere. Samme resultat kan lages ved disse linjene:

```
 figure(3); clf(3);

% Definerer formatene jeg onsker.
% Merk krollparantes!
formats = {'r', 'g--', '*', 'k.-'};

for i = 1:4
    subplot(2, 2, i);
    plot(t, x, formats{i});
    grid on;
    xlim([0 5]);
end
```

Gir Figur 4.

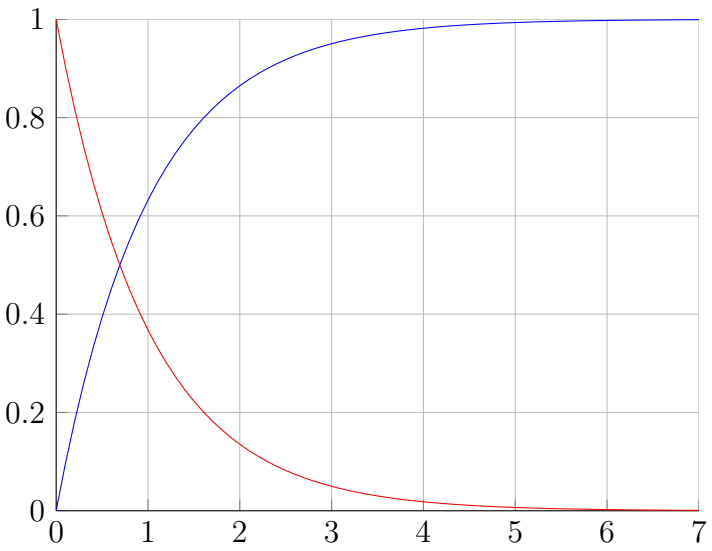


Figur 4: Fargelek

2.5 Flere grafer

Her bruker vi kommandoen `hold on;`. Denne gjør slik at du kan skrive plot flere ganger, uten at grafen slettes. Merk at nå er det viktig å bruke `clf(n)`, ellers vil alle de gamle grafene bli liggende hele tiden!

```
figure(4); clf(4);   % Lag figur
hold on;             % Sett pa hold
grid on;             % Sett pa grid

plot(t, x);          % To plot etter hverandre
plot(t, y, 'r');     % Denne vil ikke overskrive den andre!
```



Figur 5: Flere grafer

Se Figur 5.

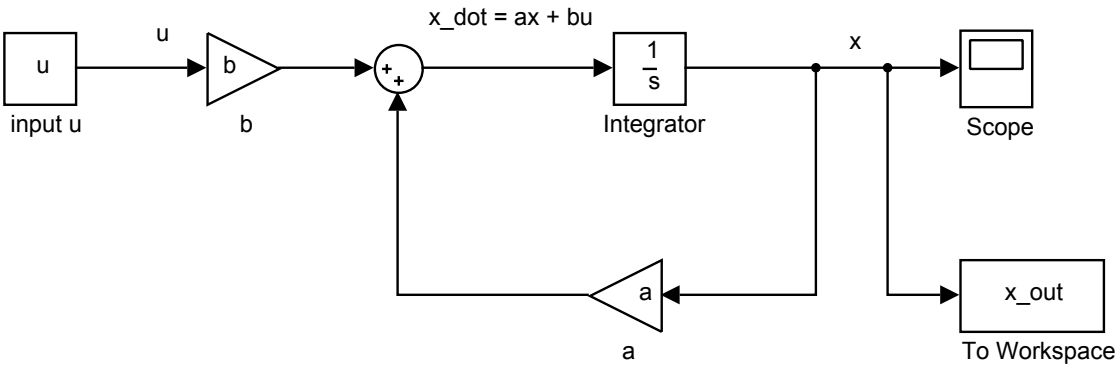
3 Simulink

3.1 Et komplett Simulinkeksempel

For å illustrere bruken av Simulink sammen med Matlab, skal vi bruke dette enkle førsteordens systemet,

$$\begin{aligned} \dot{x} &= ax + bu \\ x(0) &= x_0 \end{aligned} \tag{1}$$

Hvor a, b og x_0 er konstanter, og u input. Implementert i Simulink, vil det se ut som Figur 6.



Figur 6: Enkelt Simulinkdiagram

En liten beskrivelse av noen utvalgte blokker:

Scope Plotter signalene som scopet er koblet til.

To Workspace Eksporterer signalet til Matlabs *Workspace* slik at man kan bruke signalet (her x) i et `.m`-skript.

Constant (input u) Gir et signal med konstant verdi u som er definert i *Workspace*.

Gain (a & b) Multipliserer signalet med en konstant verdi.

For å forstå hvordan man kan sette opp et slikt diagram gitt et system som i (1), er det greieste å ta utgangspunkt i integrator-blokka i systemet. Løsningen av et initialverdi-problem som i (1) er jo x, så det er dette signalet vi er interessert i. På venstre side av integratoren ønsker vi $\dot{x}$, fordi når vi integrerer $\dot{x}$ får vi selvsagt x. Deretter er utfordringen å føre riktige signaler til $\dot{x}$, dette gjøres ved å følge formelen i (1). Husk at du nå kan bruke x fra utgangen av integratoren.

Legg merke til at konstantene er skrevet inn i blokkene. x_0 er satt som *Initial Condition* i integratorblokken. Matlabkoden (i en `.m` fil!) kan se slik ut:

```
% Define my constants
a   = -1;
b   = 2;

u   = 1;       % u is constant
x_0 = 5;       % Initial Condition

% Start Simulation
sim simple_simulink;
```

3.2 Hente grafer fra Simulink

Først, bruk blokka `To Workspace`.

Her har du et par valg, hvor *Timeseries* som regel er det greieste. Etter at simuleringen nå er kjørt, ligger resultatet i et `Timeseries` objekt. I dette eksempelet heter objektet `x_out`. Nå ligger tidsvektoren under `x_out.time`, og selve signalet ligger under navnet `x_out.data`. Da kan vi hente ut plottet slik[1]:

```
% The output is stored in x_out, from the To Workspace block.
time = x_out.time;
x    = x_out.data;

% Create the Figure
figure(1); clf(1);

plot(time, x); grid on;
```

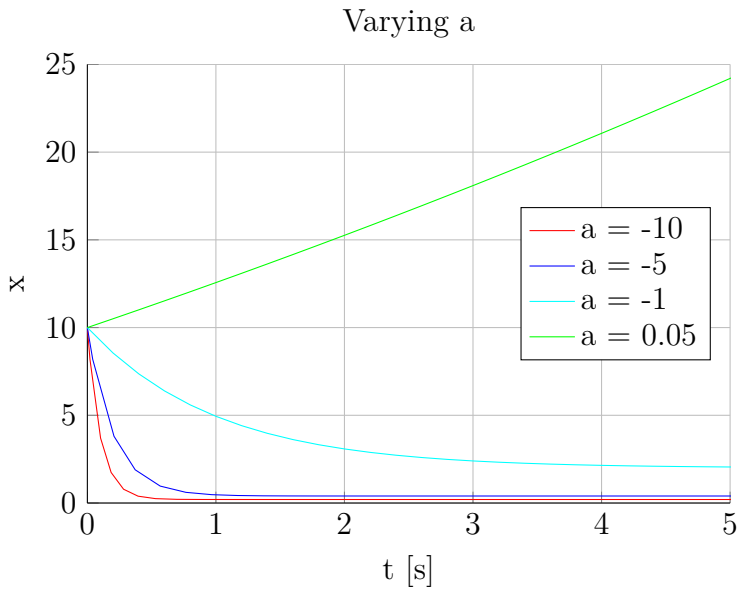
3.3 Flere simuleringer, mer avansert

Men hva om vi har lyst til å prøve flere verdier for parameteren `a`, og plotte dette i samme graf? Under følger et eksempel, med det resulterende plottet gitt i Figur 7.

```
% Formatting
formats = {'r', 'b', 'c', 'g'};
```

[1] Se kapittelet om figurer for info om plottingen

```
% The different values to try
a_values = [-10, -5, -1, 0.05];

% Create the figure
figure(2); clf(2);
hold on; grid on;

% Legend, used at the end
legend_to_set = {};

for i = 1:4

    a = a_values(i);          % Set value for a

    sim simple_simulink       % Run simulation

    plot(x_out.time, x_out.data, formats{i});

    legend_to_set{i} = ['a = ' num2str(a)];
    xlim([0 5]);

end
title('Varying a')
ylabel('x'); xlabel('t [s]');
legend(legend_to_set, 'location', 'east');
```



Figur 7: Plotting med varierende parameter

4 Avansert Matlab

Her vil jeg gå gjennom litt mer avanserte temaer, som celler og funksjoner. Dette er meget nyttige verktøy som kan gjøre hverdagen din enklere, men er ikke en nødvendighet for å gjøre de fleste oppgaver. Celler er blant annet meget nyttig når du skal tegne flere plots med forskjellig farge!

4.1 Sette sammen strenger

Du ønsker å printe en streng til kommandovinduet med funksjonen `disp`. Denne skal inneholde en tekst, en verdi på en variabel, etterfulgt av mer tekst. Ved å bruke et triks med strenger og vektorer, blir det så enkelt som:

```
to_print = ['The value is ' num2str(my_value) ' kroner'];
disp(to_print);
```

Funksjonen `num2str()` oversetter fra tall til tekst. Det finnes også en funksjon `strcat()`, men denne metoden er gjerne enklere.

4.2 Celler

Fra tidligere er vi vant til hvordan matriser og vektorer fungerer i Matlab. Man skriver `a = [5, 6, 7]`, og man kan slå opp ved å bruke `a(1)` osv. Dette fungerer fordi tallene `5, 6, 7` er *like store*. Hva om vi har lyst til å lagre strenger? Typisk eksempel er farger på en graf. Sett at du skal tegne fire plots, og vil bruke en `for` løkke. Du kan bli fristet til å skrive noe slikt:

```
formats = ['b', 'b.-', 'r', 'r.-'];
for i = 1:4
    % Do something to obtain new x

    plot(t, x, formats(i))
end
% This will NOT give wanted result
```

Dette vil ikke gi ønsket resultat. Grunnen er at `formats` nå er en streng som ser slik ut: `formats = 'bb.-rr.-'`. Ved å slå opp som `formats(i)` fra 1 til 4, vil vi få ut `fomats(1)= 'b', formats(2)= 'b'`, `formats(3)= '.'`, `formats(4)= ... '-'`.

Løsningen er å bruke `celler`. Disse lar oss lagre verdier med ulik lengde. Ved å erstatte `(), []` med {} (pass på å slå opp med krøllparanteser også), slik at det blir:

```
formats = {'b', 'b.-', 'r', 'r.-'};
for i = 1:4
    % Do something to obtain new x

    plot(t, x, formats{i})
end
% Correct!
```

Dette vil gi ønsket resultat. Lek deg litt med cellene til du forstår dem! Er også veldig nyttige i sammenheng med `legend()`.

4.3 Funksjoner

Matlab definerer to ulike typer funksjoner. Store funksjoner som plasseres i en egen `m`-fil, og små funksjoner skrevet inne i scriptet ditt, som kalles *anonymous functions*.

Du bruker dem helt likt i scriptene dine.

4.3.1 Småfunksjoner, anonymous functions

Dette er enkle funksjoner, eller rene matematiske funksjoner. Sett at du har definert f som

$$f(x) = \frac{x}{2+x} \tag{2}$$

Da kan det ofte være nyttig å kunne finne $f(3)$ f.eks, eller hva om du har lyst til å evaluere f med en vektor av x'er?

Den kan implementeres slik:

```
f = @(x) ( x. / (2 + x) );
```

(NB: Merk dotten i `x. /`. Denne gjør at hvert element i en vektor `x` blir ganget ut. Se seksjon 4.4.).

Det viktige er `@`. Den første parentesen sier at `x` er input. Den neste regner ut svaret.

Nå kan vi gjøre slik:

```
x = [2, 3, 4];  % Define my x
results = f(x)  % Store my result

% Or, for plotting
x = 0:0.1:10;       % Define my x

figure(1); clf(1);  % Create my figure
plot(x, f(x));      % Plot f(x) vs x
```

4.3.2 Funksjonsfiler

Alle innebygde funksjoner i Matlab er skrevet i slike funksjonsfiler. Hver m-fil har en funksjon. Under følger et eksempel.

```
function [ sum, product ] = my_function( x, y)
%MY_FUNCTION Denne funksjonen gjør noe med x og y
%   Returnerer sum = x + y, product = x*y

    sum     = x + y;
    product = x * y;
end
```

NB: Navnet på `m`-filen må være det samme som funksjonsnavnet! I dette tilfellet `my_function.m`.

I funksjonen over er `sum`, `product` output og `x`, `y` er input. Den kan kalles slik:

```
[res, res2] = my_function(2, 3)

% Result:
> res:  5
> res2: 6
```

4.4 Vektorer, * vs .*

Operatoren $\star$ er i Matlab reservert for matriseoperasjoner. Den utfører altså matrisemultiplikasjon. Dersom du ønsker at hvert element i en vektor skal multipliseres med hvert element i en annen (*inner product*), bruker man `.*`.

Typiske eksempler er operasjoner man ønsker utført på en vektor av inputverdier. Når man bare ønsker å multiplisere med en skalar-verdi spiller det ingen rolle hvilken operator du bruker.

4.5 Printing til kommandovinduet

Ved bruk av større scripts er det ofte fordelaktig å skrive ut ting til skjermen. Dette gjøres enklest ved kommandoen `disp()`, som lar deg skrive ut tekststrenger til kommandovinduet. Men, dersom du også ønsker å skrive ut resultater av dine beregninger, blir det fort mer jobb:

```
a = 2 + 2;

disp(['Research shows that 2 + 2 is still ' num2str(a)]);
```

Et alternativ er derfor å bruke `fprintf`, som lar en bruke *c-style* output, med formater i en streng og data i egne parametre. Da blir det:

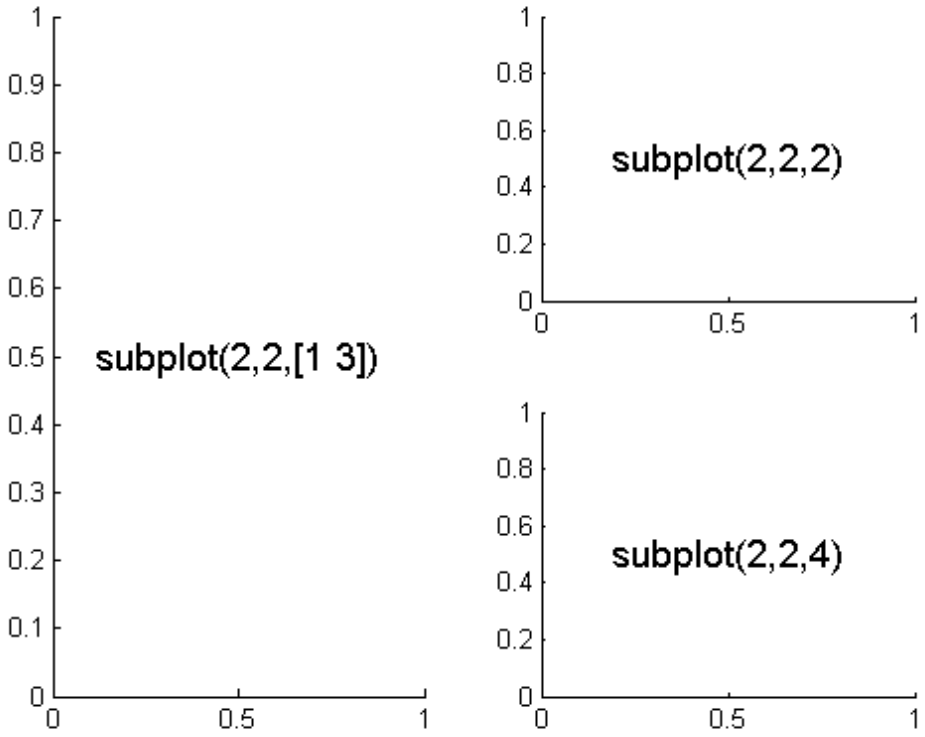
```
fprintf('Research shows that 2 + 2 is still %g \n', a);
```

(Merk; `'\n'` gir oss linjeskift)

Velg selv hva du liker best, men jeg mener sistnevnte er mer ryddig. Skriv `doc ... fprintf` for en liste over formatene. De mest brukte er `'%g'` for flyttall (mest brukt), `'%d'` for heltall og `'%s'` for tekststrenger.

A Appendix

A.1 Subplot avansert



Figur 8: Avansert subplot

Register