# Matlab-tips ved oblig3 i FYS-MEK/F 1110 våren 2006

**Utforsking av et kaotisk system.**

I dette skrivet gir vi noen tips som kan være nyttige når man skal skrive et Matlab-program for å gjøre beregninger av den kaotiske pendelen.

## a) Stykke opp Matlab-program i flere funksjoner innenfor samme .m-fil

Det kan være en fordel å stykke opp et lengre Matlab-program i mindre funksjoner for å få et mer oversiktlig program. Dette gjøres på lignende måte som i de fleste andre programmerings-språk, men det er likevel visse særtrekk som man bør kjenne til.

Nedenfor er det gitt et eksempel:

```
function TestOverfParam1

    theta = zeros(1,1000);          % Allokerer hukommelse til array
    t = linspace(0,10*pi,1000);     % Lager array med stigende tall
    theta = utregning(theta,t);     % En eller annen beregning basert på input
    %for i=1:10
    %    theta(i)    % Testutskrift for å se om theta er endret eller ikke
    %end
    plotKurve(theta,t);

% ************************************************************************

function nn = utregning(theta,t)    % Objektet nn må gis en verdi før retur
                                    % Input-parametre blir lokale!!!

    theta = 4.0*sin(t);     % Beregningen som skal gjøres
    nn = theta;             % Her fortelles det hva som skal returneres

% ************************************************************************

function plotKurve(theta,t)
    plot(t,theta,'-r');             % Plotter
    %axis([-3.5 3.5 -0.0 20.0]);      % Kan gi manuell skalering på plot
    xlabel('Et eller annet argument (enhet)');     % Tekst langs aksene
    ylabel('En eller annen størrelse (enhet)');

% ************************************************************************
```

Merk at “hovedprogrammet” må komme først. Man behøver ikke avslutte programmet med “end”, for så snart man skriver en ny “function” vil Maple skjønne at den funksjonen man inn-til da var i da skal være slutt.

Hvordan overfører man så argumenter fra en funksjon til en annen? Det kan gjøres på flere måter. I eksempelprogrammet ovenfor har vi valgt å overføre parametre ved kall til de ulike funksjonene. Når vi skriver “utregning(theta,t);” kaller vi funksjonen “utregning”, og objek-

tene theta og t (begge arrayer i vårt tilfelle) blir overført til funksjonen "utregning", og kan brukes innenfor denne funksjonen fullt ut.

Man må imidlertid merke seg at selv om det er en peker som overføres når funksjonen "utregning" kalles, arbeider vi IKKE med samme hukommelsesområdet innenfor funksjonen som utenfor. Arrayene theta og t blir rene lokale variable innenfor "utregning", og vi får IKKE automatisk endringene med oss tilbake til hovedprogrammet når man er ferdig i "utregning".

For å få med seg tilbake til hovedprogrammet det man har beregnet i funksjonen "utregning", må vi gi dataene som et retur-objekt. Det skjer i følgende to step:

```
function nn = utregning(theta,t)
....
nn = theta; .
```

I funksjons-headingen signalerer vi at vi skal returnere et eller annet objekt, her kalt nn. Før vi går ut av funksjonen må man da ha fortalt hva dette objektet skal være. I vårt tilfelle velger vi å returnere den nyberegnede theta-arrayen.

I funksjonen plotKurve har vi ikke noe retur-parameter, så da dropper man konstruksjonen vi hadde i "utregning".

Program-snutten

```
for i=1:10
    theta(i)    % Testutskrift for å se om theta er endret eller ikke
end
```

er et eksempel på ekstra testutskrifter som jeg selv velger å putte inn på ulike steder i programmene jeg lager. Med slike testutskrifter kan jeg lett teste at verdier underveis ser rimelige ut, slik at jeg lettere kan finne ut hvor eventuelle feil oppstår. Jeg kommenterer ut disse testutskriftene når de ikke brukes, slik som gjort i programeksemplet på forrige side. Dersom det blir mange slike ut-kommenterte testutskrifter, fjerner jeg de minst viktige etter hvert som jeg ser at programmet fungerer, for at ikke koden skal bli for uoversiktlig.

Det finnes imidlertid også andre måter å gjøre variable tilgjengelig innenfor flere ulike funksjoner. Et alternativ til programmet på forrige side kan da for eksempel se slik ut:

```
function TestOverfParam2
global T THETA;
   THETA = zeros(1,1000);          % Allokerer hukommelse til array
   T = linspace(0,10*pi,1000);     % Lager array med stigende tall
   utregning;      % En eller annen beregning basert på størreler overført
                   % som globale variable
   %for i=1:10
   %    THETA(i)    % Testutskrift for å se om theta er endret eller ikke
   %end
   plotKurve(THETA,T); % Her har vi valgt å overføre som parametre ved kall

% ********************************************************************
```

```
function utregning    % Ingen overføring av parametre ved kall, verken
global T THETA;       % når funksjonen kalles eller når den returnerer

   THETA = 4.0*sin(T);     % Beregningen som skal gjøres

% ********************************************************************

function plotKurve(theta,t)
   plot(t,theta,'-r');              % Plotter
   %axis([-3.5 3.5 -0.0 20.0]);      % Kan gi manuell skalering på plot
   xlabel('Et eller annet argument (enhet)');     % Tekst langs aksene
   ylabel('En eller annen størrelse (enhet)');

% ********************************************************************
```

Merk at vi her deklarerer de variablene vi ønsker som *globale variable*. Vi må gjenta deklarasjonen i alle funksjonene der vi ønsker å benytte oss av at størrelsene er globale. Det gjør at vi nå kan kalle funksjonen "utregning" uten å overføre parametre ved kall på funksjonen, og uten at vi må gjøre spesielle triks for å overføre objekter tilbake til hovedprogrammet igjen.

For å huske på at enkelte parametre er globale, er det vanlig å skrive disse med store bokstaver.

Merk at når vi kaller plotKurve, har vi også i den siste programvarianten valgt å overføre de to arrayene ved kall. Navnet på arrayene i hovedprogrammet er THETA og T. Men inne i funksjonen plotKurve har vi valgt å kalle arrayene som vi bruker der (de lokale arrayene) for theta og t, skrvet med små bokstaver. Dette svarer til det vi kjenner fra Java (og andre språk) at vi kan bruke andre navn på de lokale variablene enn navnet som er brukt i det kallende programmet.

Generelt sett *frarådes det å bruke globale variable* i tide og utide. Det har blant annet å gjøre med beskyttelse av data for uønskede endringer, slik dere allerede kjenner fra Java-programmeringen. Det er derfor bedre å bruke første programeksempel enn det siste i dette skrivet.

Et par små detaljer kan kommenteres før vi forlater denne koden:

Vi har valgt å plotte ut kurven vår som en rød strek. Det får vi til med koden plot(t,theta,'-r');. Hadde vi i stedet ønsket at man bare ga en liten sirkulær prikk for hvert datapunkt, kunne vi i stedet skrevet plot(t,theta,'.r'); (punktum i stedet for et minus/bindestrek). Og ville vi hatt en annen farge enn rødt, kunne vi valgt f.eks. b (**b**lå), g (**g**rønn), k (svart/blac**k**), m (**m**agenta), c (**c**yan), eller y (gul, **y**ellow).

Vi kan av og til ønske å selv velge hvilket utsnitt vi skal ha på aksene i et plot. Da kan vi bruke kommandoen axis([-3.5 3.5 -0.0 20.0]);. Inne i klammeparantesen skal det da stå xmin, xmax, ymin og ymax med blank som skilletegn. Dette kan være praktisk dersom man vil kunne sammenligne to ulike plot ved å legge dem over hverandre.

## b) Eulers enkle metode og Euler-Cromer's-metoden

I oppgaveteksten for oblig 3 står det at man skal utlede relasjonen:

$$\tau(\theta, t) \;=\; -mgL\sin\theta - qL^2\omega + \tau_{y0}\sin(\omega_y t) \;=\; mL^2\alpha \qquad (1)$$

hvor de ulike størrelsene er gitt i oppgavetekst og notat om kaotiske systemer. Denne likningen kan vi omforme slik:

$$\alpha \;=\; -\frac{g}{L}\sin\theta - \frac{q}{m}\omega + \frac{\tau_{y0}}{mL^2}\sin(\omega_y t)$$

Dette er en likning som gir vinkelakselerasjonen til pendelen for gitt vinkel og vinkelhastighet for pendelen (θ og ω), for ulike tider t og andre parametre. Men vi kjenner også relasjonene mellom vinkel, vinkelhastighet og vinkelakselerasjon:

$$\alpha \;=\; \frac{d\omega}{dt} \qquad \text{og} \qquad \omega \;=\; \frac{d\theta}{dt}$$

som kan omskrives til differens form:

$$\Delta\omega \;=\; \alpha \cdot \Delta t \qquad \text{og} \qquad \Delta\theta \;=\; \omega \cdot \Delta t$$

Vi ser da at vi kan lage enkle rekursjonsformler for å beregne utviklingen av bevegelsen til et system, forutsatt at vi har et sett initialverdier vi kan starte ut med. I vårt tilfelle ville vi kunne bruke f.eks. følgende sett med rekursjonsformler (som representerer såkalt eksplisitt metode):

$$\alpha(n) \;=\; -\frac{g}{L}\sin\theta(n) - \frac{q}{m}\omega(n) + \frac{\tau_{y0}}{mL^2}\sin(\omega_y t) \qquad (2a)$$

$$\omega(n+1) \;=\; \omega(n) + \alpha(n) \cdot \Delta t \qquad (2b)$$

$$\theta(n+1) \;=\; \theta(n) + \omega(n) \cdot \Delta t \qquad (2c)$$

Dette settet med rekursjonsformler tilsvarer det vi tidligere har omtalt som Eulers (enkle) metode, og den er beskrevet i kapittel 10.8 (side 518) i Tom Lindstrøms bok Kalkulus, bind 1.

Eulers metode for å løse differentialligninger er en av de groveste metodene som finnes, og du vil se at for pendelbevegelsen blir resultatet ganske dårlig, uansett hvor liten vi velger Δt.

Man kan i stedet bruke Eulers midtpunktsmetode, eller enda bedre: 4. ordens Runge-Kutta-metode, som begge er beskrevet i Lindstrøms bok. Men for pendelbevegelsen finnes det et enda enklere alternativ som fungerer bra, nemlig den såkalte Euler-Cromer's metode. Den eneste forskjellen fra algoritmen ovenfor er at siste ligning erstattes med:

$$\theta(n+1) \;=\; \theta(n) + \omega(n+1) \cdot \Delta t \qquad (2c')$$

Vi bruker altså den *nye* verdien for vinkelhastigheten i stedet for den forrige, slik som i Eulers enkle metode. Det er med andre ord ekstremt enkelt å skifte fra Eulers metode til Euler-Cromers metode.

## c) Hvordan man kan lage et plot for et enkelt datasett

Iblant har vi et sett data som vi gjerne skulle ha laget en graf av. Her ved UiO er Origin et utmerket plotteprogram som f.eks. mastergradsstudenter bruker. For enkle plot klarer du deg også med Matlab. Nedenfor er gitt et lite program som viser hvordan man kan plotte seks datapunkter ut fra x- og y-verdier som er skrevet inn for hånd i programmet:

```
function enkeltPlot
    x = [0.0 1.2 2.7 4.1 4.9 7.2];
    y = [4.2 3.9 4.4 5.0 4.7 4.3];
    plot(x,y,'-r');
    hold on
    plot(x,y,'.r');
    xlabel('Frekvens (kHz)');
    ylabel('Amplitude (V)');
```

Her har jeg valgt å plotte de samme dataene to ganger i samme plot, en gang som linjeplot, neste gang som punktplot. Det er gjort for å få bedre fram hvor selve målepunktene ligger, samtidig som man ønsker å få fram "grafen" også på en ok måte.

Man kan evt. bruke et lignende opplegg når man i oblig 3 skal lage en graf som viser periodetiden til pendelbevegelsen for ulike start-vinkelhastigheter i det tilfellet at vi ikke har luftmotstand (dempning) eller et påtrykt ytre kraftmoment.

## d) Andre små-detaljer

**1)** I Matlab behøver vi ikke å spesifisere så nøye hva slags størrelser de ulike variablene skal være. Det kan være praktisk i de fleste tilfeller, men iblant må man inn og styre dette selv for å være sikker på at alt går som det skal.

Dividerer man to heltall med hverandre, blir ikke resultatet automatisk et heltall i Matlab. For å presse det til å bli et heltall, kan man f.eks. bruke en av følgende funksjoner: floor( ), ceil( ), fix( ), round( ) eller int16( ).

Skriver vi f.eks. 5/3, får vi 1.66666...667, men dersom vi skriver int16(5/3), round(5/3) eller ceil(5/3), får vi resultatet 2. Det blir et heltall, men man runder av på vanlig måte i de to første tilfellene i stedet for å trunkere resultatet. For ceil( ) tar man neste heltall over det rasjonale tallet man har som argument. For å få et trunkert resultat (avrundet mot nærmeste heltall i retning minus uendelig), kan man i stedet bruke floor( ). Beregner man floor(5/3), får man 1.

Det kan lønne seg å sjekke bruken av ulike funksjoner i Matlab ved å gå inn i hjelpfunksjonen til programmet. Selv velger jeg ofte å bruke index i hjelp-winduet og så skrive inn det stikkordet jeg tipper på, men her må enhver finne en måte som passer for egne arbeidsvaner.

**2)** Iblant kan man ønske å plukke ut en del av en array for videre behandling (f.eks. plotting). Dersom man har en array theta som har elementer fra 1 til N, kan man plukke ut f.eks. bare elementene fra og med element nr 30 til og med 1000 ved å skrive

```
theta=theta(30:1000);
```

Siden vi satte resultatet inn i samme array som vi startet ut med, vil arrayen theta *etter* denne programsetningen bare være 971 elementer lang. Det er forutsatt at theta *før* dette statementet hadde *minst* 1000 elementer, ellers ville man fått feilmelding.

**3)** I oblig 3 beregner vi en vinkel som i prinsippet kan bli svært stor (enten positiv eller negativ) dersom pendelen dreier seg i samme retning runde etter runde. Men når vi observerer pendelen, ser vi at den til ethvert tidspunkt ligger innenfor vinkelintervallet $-\pi$ og $+\pi$. Vi må derfor presse dataene våre inn i dette intervallet.

Det gjør vi ved å fjerne et heltalls multippel av $2\pi$ fra den beregnede vinkelen. Dette kan enklest gjøres ved å bruke modulus-funksjonen. Vi kunne da f.eks. ha skrevet:

```
theta = mod(theta,2*pi);
```

men det ville ikke vært optimalt, for da fikk vi bare presset verdiene til å ligge i intervallet 0 til $2\pi$. For å få intervallet slik vi ønsker, må vi forskyve verdiene før vi tar modulus, og så forskyve tilbake etterpå, f.eks. slik:

```
theta = mod(theta+pi,2*pi)-pi;
```

Det er denne varianten som ble brukt da figurene i notatet om kaotiske systemer i fysikken ble generert.

**4)** Vanligvis er det greit å ha ett og samme dataprogram som vi ved små endringer i programmet kan få til å gjøre flere ulike regneoperasjoner. I oblig 3 er det selvfølgelig mulig å gjøre det på samme vis, men som du ser av oppgaveteksten foreslår vi at du heller lager to eller tre ulike varianter av programmet for å få litt mer ryddig struktur. Spesielt er det nødvendig å skille ut den programvarianten hvor vi skal generere et Poincaré-snitt, fordi vi i det tilfellet ikke kan ta vare på en fullstendig registrering av bevegelsen (vinkel og vinkelhastighet og tidspunkt for ethvert steg i bevegelsen). Vi kan da ikke bruke arrayer i rekursjonsformlene 2a, 2b og 2c', men heller bruke to varianter av hver variabel, f.eks. theta_n og theta_nplus1, der theta_n settes lik theta_nplus1 før løkken kjøres på ny.

I løkken må man så sette opp en test for når $\omega_y t$ passerer et multippel av $2\pi$, og ta vare på (i to arrayer) akkurat de $\theta$ og $\omega$-verdiene som svarer til den fasen i den ytre påvirkningen som vi har valgt som grunnlag for Poincaré-snittet vårt. Hvordan man kan lage en test for å plukke ut de riktige punktene får du finne ut selv, men en eller annen bruk av modulus eller floor-funksjonene kan være en vei å gå. Vi må forresten selv holde orden på indekseringen av arrayene i dette tilfellet.

Til slutt en liten tilståelse. I Poincaré-snitt figurene i notatet har har jeg feilaktig valgt ut punkter der $(\omega_y t + \pi)$ passerer et multippel av $2\pi$. Så dersom du forsøker å gjenskape de figurene som er angitt, vil du ikke få samme resultat som dersom du velger ut punktene på angitt måte i oppgaveteksten. Men de to settene har en del likheter likevel...