Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 5

Skript

I denne øvinga skal vi lære oss mer om skript. Et skript kan vi se på som et lite *program* – altså en sekvens av kommandoer.

Til sist skal vi se litt på hvordan man kan lage sine egne funksjoner i MATLAB. Dette kan gjøres ved å lage egne funksjonsfiler. Men ofte kan det også gjøres på en litt mer kompakt måte – ved å bruke en spesiell skrivemåte med "@".

Som vanlig forutsettes det at du har gjort de tidligere leksjonene[1].

Oppgave 1 – Hva gjør disse skriptene?

a) Som i oppgave 5 b) i leksjon 3, kopiér eller skriv av dette skriptet, lagre det med et eller annet navn (husk at de må slutte på '`.m`') i ei passe mappe og kjør skriptet[2]. Hva gjør dette skriptet?

```
1  x=-5:1e-2:5;
2  y=cos(-pi*x/10).^2;
3  z=y.*sin(5*x);
4
5  plot(x,y,'k','linewidth',2)
6  hold on
7  plot(x,-y,'k--','linewidth',2)
8  plot(x,z,'r')
9  hold off
```

Her har vi lagt til linjenummer for å lettere kunne referere til ulike deler av koden. (Disse er ikke med i selve skriptet.)

Skriptet har en alvorlig mangel: Det er ikke *kommentert*. Å kommentere et skript vil si å legge til tekst som forklarer hva skriptet eller programmet gjør. Dette gjør vi både i starten og underveis i koden. For eksempel kan en kommentert versjon av skriptet fra oppgave a) se slik ut:

[1] ... med mulig unntak av leksjon 4; den kan du eventuelt ta igjen senere.

[2] Dette gjøres i *teksteditoren*, som du får opp ved å klikke på "new script" oppe i venstre hjørne av kommandovinduet – eller ved å skrive '`» edit`' i kommandovinduet

```
% Skript som plotter visse funksjoner

% Tilordner x-verdier
x=-5:1e-2:5;
% Tilordner funksjonsverdier
y=cos(-pi*x/10).^2;
z=y.*sin(5*x);

% Plotter funksjonene sammen
plot(x,y,'k','linewidth',2)
hold on
plot(x,-y,'k--','linewidth',2)
plot(x,z,'r')
hold off
```

Vi bruker altså prosent-tegnet, "%", for å legge til kommentarer. Det som står bak %, blir ignorert av MATLAB, men ikke nødvendigvis av en person som leser skriptet.

Kommentarene er ofte helt avgjørende for at skriptet skal kunne bli forstått av andre – eller av deg selv på et senere tidspunkt, for den saks skyld[3].

b) Endre skriptet over slik at det i tillegg plotter funksjonen $f(x) = \cos^2(\pi x/10)\cos(5x)$. Justér også `x`-vektoren slik at funksjonene blir plotta i intervallet $[-7, 7]$.

c) I det neste skriptet har vi brukt en funksjon som heter `input`. Når skriptet kjøres, vil denne kommandoen gjøre at det stopper opp og venter på at brukeren (du) gir en tallverdi fra kommandovinduet. Skriv av eller kopiér skriptet nedenfor i teksteditoren, lagre det i ei passe mappe med et passe navn (med "`.m`" til slutt) og kjør det. (Pass på at du er i samme mappe som fila ligger i.)

```
a=input('Gi verdien for a: ');
b=input('Gi verdien for b: ');
c=input('Gi verdien for c: ');

x1=(-b-sqrt(b^2-4*a*c))/(2*a)
x2=(-b+sqrt(b^2-4*a*c))/(2*a)
```

Prøv å kjøre skriptet for ulike verdier av `a`, `b` og `c`. Forstår du hva skriptet gjør? Kommentér skriptet[4].

d) La oss anta at én Euro er verd 9.00 norske kroner, ett britisk pund er verd 12.81 kroner, og 100 svenske kroner er verd 95.30 norske. Lag et lite skript som leser inn ett beløp i norske kroner og returnerer (til kommandovinduet) tilsvarende beløp i euro, britiske pund og svenske kroner.

[3] Her snakker jeg av bitter erfaring.

[4] Ikke muntlig; legg till kommentarer i selve fila.

Oppgave 2 – Logiske variable, `if`-satser

a) Som tidligere nevnt, betyr '`=`' *tilordning* – ikke likhet – i MATLAB. Likhet skrives slik: '`==`'. Større enn og mindre enn, derimot skrives som normalt. 'Er ulik', og ikke-strenge ulikheter kan skrives slik: '`~=`', '`<=`' og '`>=`' – tilsvarende '$\neq$', '$\leq$' og '$\geq$' i vanlig matematikk-notasjon. Forsøk å skrive noen sanne og noen usanne logiske påstander i kommandovinduet, som for eksempel $3 = 2$ (husk å skrive likhet som '`==`' i MATLAB), $-1 < 0$, og $-1 \neq 0$, og se hva du får til svar. Forsøk gjerne å kombinere med *eller* og *og* også. 'Eller' kan skrives som '`|`' (tasten nest øverst til venstre på tastaturet ditt) og 'og' kan skrives som '&'. For eksempel kan påstanden $x \in [-2, 1]$ skrives slik i MATLAB: `x>=-2 & x<=1`.

I denne sammenhengen, hva betyr '0' og '1'?

b) I det neste skriptet, som vi kaller '`TypeTall.m`', introduserer vi `if`-satser. Skriv av og kjør skriptet:

```
x=input('Gi verdien for x: ')

if x<0
  disp('Tallet er negativt')
else
  disp('Tallet er positivt')
end

if round(x)~=x
  disp('Tallet er ikke et heltall')
else
  if round(x/2)==x/2
    disp('Tallet er et partall')
  else
    disp('Tallet er et oddetall')
  end
end
```

Om du lurer på hva funksjonen `round` gjør, skriv "`» help round`" i kommandovinduet. `disp`-funksjonen gjør at det som står inni parantesen blir skrevet til skjerm uten variabelnavn. Apostroffene bruker vi fordi det som skal skrivest til skjerm, er tekst – ikke tall.

Hva gjør skriptet? Kommentér det.

c) **For spesielt interesserte:** For skriptet i oppgave 1 b), klarer du å bruke `if`-satser til å lage en mer "idiotsikker" versjon av skriptet? Kanskje du kan sjekke at a er ulik null – eller at andregradslikninga har reelle løsninger? Om vi kan unngå å skrive to identiske løsninger til skjerm (d.v.s. i kommandovinduet), er vel også det en fordel... Alle disse justeringene kan gjøres ved hjelp av `if`-satser av typen beskrevet nedenfor.

Litt mer om `if`-satser:

De enkleste `if`-satsene har denne strukturen

```
if <logisk påstand>
  <utfør kommandoer>
end
```

Kommandoen eller kommandoene vil bare bli utført hvis den logiske påstanden er sann. Legg merke til at vi har gjort et lite "innhopp" i teksten i linja mellom `if` og `end`. Dette gjør koden mye mer oversiktlig. Så vi anbefaler at du også legger deg til denne vanen[5].

Denne strukturen er også mye brukt:

```
if <logisk påstand>
  <utfør kommandoer>
else
  <utfør andre kommandoer>
end
```

Vi kjenner denne strukturen igjen fra funksjonsfila over. Her blir altså "andre kommandoer" utført i stedet for "kommandoer" dersom den logiske påstanden er feil. I `if`-satsen fra og med linje 3 til og med linje 7 i skriptet fra oppgave 2b) over, for eksempel, er "kommandoer" '`disp('Tallet er negativt')`', altså å skrive "Tallet er negativt" til skjerm, mens "andre kommandoer" her er ' `disp('Tallet er positivt')`'.

En tredje vanlig struktur er denne:

```
if <logisk påstand>
  <utfør kommandoer>
elseif <annen logisk påstand>
  <utfør andre kommandoer>
else
  <utfør et tredje sett av kommandoer>
end
```

Her kan man bygge på med så mange `elseif`-satser man bar vil etter hverandre. Som vi så i oppgave 2 b), kan man godt ha flere `if`-satser inni hverandre. I slike sammenhenger er det ekstra viktig å holde styr på innhoppene.

[5] I Python er det faktisk påkrevd.

Oppgave 3 – Funksjoner i MATLAB

Som vi har sett, har MATLAB de grunnleggende elementære funksjonene innebygd – altså funksjoner som $\sin x$, e^x, $\arctan x$ etc. Men vi kan også definere våre egne funksjoner i MATLAB. Dette er en måte å gjøre det på[6]:

```
Funk=@(x) x^2*cos(x);
```

Vi har nå tilordna ein funksjon som heiter "`Funk`" og som kan kalles med et argument – akkurat som `sin(x)` og `log(x)` etc. I kommandovinduet kan vi for eksempel nå skrive

```
>> Funk(0)
ans =
     0
>> Funk(pi)
ans =
   -9.8696
>> Funk(-3)
ans =
   -8.9099
```

Funksjonen vil nå "leve" i minnet til MATLAB til den blir fjerna (`clear`) eller til MATLAB blir slått av. Man kan definere funksjoner på denne måten både direkte i kommandovinduet og i et skript.

a) I kommandolinja i MATLAB: Lag deg en funksjon på denne måten og regn ut funksjonsverdier for noen argumentverdier (x-verdier) du velger selv.

b) Man kan også bruke egendefinerte funksjoner til å plotte:

```
>> x=-6:1e-2:6;
>> plot(x,Funk(x))
```

Med `Funk` definert som over fungerer dette dårlig. Prøv å justere funksjonsuttrykket slik at dette går bra; det er ei veldig lita justering som skal til...

[6] I MATLAB-sammenheng kalles funksjoner lagt på denne måten *anonyme funksjoner*.

Man kan også lage funksjoner ved hjelp av egne filer. Funksjonen over kan for eksempel implementeres slik:

```
1 function F=Funk(x)
2
3 % Funksjonen f(x)=x^2 cos x.
4 % Funksjonen tar bare skalarer som input.
5
6 F=x^2*cos(x);
```

Man lagrer denne fila og bør gi den det samme navnet som navnet på høgre side i linje 1. Videre skal navnet, som med skript, slutte på "`.m`". Denne funksjonsfila skal altså hete "`Funk.m`". Dersom man i kommandovinduet er i den samme mappa som denne fila ligger i, kan man kalle den på samme måte som over.

Ekstraoppgave – Funksjonsfiler

a) Implementér funksjonen $f(x) = x^2 \cos x$ på måten beskrevet over – altså som ei funksjonfil. Som i oppgave 3: Regn ut funksjonsverdien for visse argumentverdier.

b) Hva skjer hvis du skriver '`» help Funk`' i kommandovinduet?

I eksempelet over er selve funksjonen gitt ved èi linje. Men slik trenger det ikke være. Dette er ein funksjon gitt med *delt forskrift*:

$$f(x) = \begin{cases} \cos(\pi x) + 2, & x < 2 \\ x^2 - 2, & x \geq 2 \end{cases} .$$

c) Lag ei funksjonsfil som implementerer denne. Navnet velger du selv.

d) Dersom man løser deloppgave c) ved hjelp av en `if`-sats, har man ett problem: Funksjonen tar ikke vektor-argument; `if`-satsen leser ikke vektorer. Klarer du å lage ei implementering av fuksjonen som tar vektor-argument? Hint: Hva får du i kommandovinduet om du skriver

```
>> x=0:10
x =
     0     1     2     3     4     5     6     7     8     9    10
>> x>4
```

Litt om forskjellen på funksjonsfiler og skript:
Ei funksjonsfil skal, som vi har sett, ha en viss struktur. Den skal for eksempel alltid starte med "`function`". I tillegg skal vi alltid gi (minst) et tall (eller en vektor) inn til ei funksjonsfil. Dette gjør vi, som vi har sett, ved å skrive dette tallet eller denne vektoren i parantes etter navnet på funksjonen i kommandoviunduet. (`» Funk(5)`.)

Et skript, derimot, har ikke noen spesiell struktur; det er bare ei oppramsing av kommandoer. Og når vi kjører skriptet, skal det ikke stå noe i noe argument i noen parantes etter navnet på skriptet.

I tillegg: Eventuelle variable som blir tilordna i ei funksjonfil, vil bare være å finne i funksjonfila; de vil ikke dukke opp i minnet til MATLAB – til forskjell fra tilordninger som blir gjort i et skript.