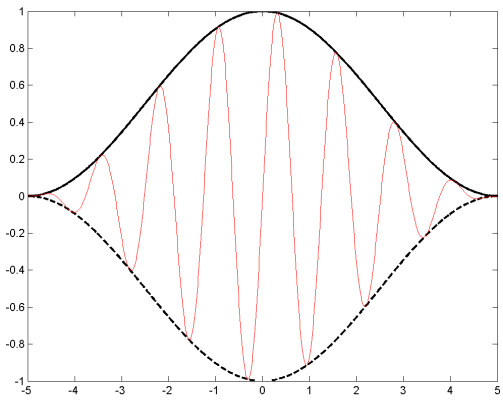
Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 5
Løsningsforslag

Oppgave 1 – Hva gjør disse skriptene?

a) Skriptet lager plottet vi ser i figur 1.



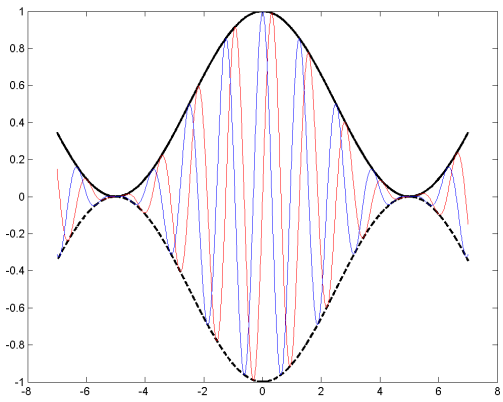
Figur 1: Plott fra oppgave 1 a).

b) Om vi endrer skriptet til, for eksempel, dette:

```
% Skript som plotter visse funksjoner

% Tilordner x-verdier
x=-7:1e-2:7;
% Tilordner funksjonsverdier
y=cos(-pi*x/10).^2;
z=y.*sin(5*x);
f=y.*cos(5*x);

% Plotter funksjonene sammen
plot(x,y,'k','linewidth',2)
```

```
12 hold on
13 plot(x,-y,'k--','linewidth',2)
14 plot(x,z,'r')
15 plot(x,f,'b')
16 hold off
```

får vi opp plottet vist i figur 2. Her har vi justert litt på `x`-vektoren i linje 4 samt lagt til linje 8 og 15.



Figur 2: Plott fra oppgave 1 b).

c) Her kjenner vi nok igjen *abc*-formelen for løsning av andregradsliknigner:

$$ax^2 + bx + c = 0 \Leftrightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad .$$

Skriptet regner altså ut løsningene av likinga $ax^2 + bx + c = 0$.

Vi kjører skriptet for noen verdier av a, b og c[1]:

```
>> format compact
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: -2
Gi verdien for c: -4
x1 =
   -1.2361
x2 =
    3.2361
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: -11
```

[1] '`Format compact`'-kommandoen gjør at MATLAB skriver ting med litt færre mellomrom.

```
Gi verdien for c: 30
x1 =
     5
x2 =
     6
```

For noen verdier går det ikke så bra:

```
>> ABCformel
Gi verdien for a: 0
Gi verdien for b: 1
Gi verdien for c: 2
x1 =
  -Inf
x2 =
   NaN
```

Vi minner om at "svarene" `Inf` og `NaN` står for uendelig (*infinity*) og udefinert (*not a number*).

```
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: 1
Gi verdien for c: 2
x1 =
  -0.5000 - 1.3229i
x2 =
  -0.5000 + 1.3229i
```

I dette eksemplet har vi fått komplekse løsninger. For andre verdier av a, b og c får vi bare én løsning:

```
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: -6
Gi verdien for c: 9
x1 =
     3
x2 =
     3
```

d) Dette er et eksempel på hvordan et slikt skript kan se ut:

```
1  % Skript som regner ut hva et beløp i NOK tilsvarer i pund, euro
2  % og svenske kroner. De aktuelle kursene er fiksert her i skriptet.
3
4  % Kurser
5  KursEuro=9;
6  KursPund=12.81;
7  KursSVK=95.3/100;
8
9  % Leser inn beølp i NOK:
10 NOK=input('Beløp i norske kroner: ');
11
12 % Resultat
13 BelopEuro=NOK/KursEuro
14 BelopPund=NOK/KursPund
15 BelopSVK=NOK/KursSVK
```

Vi tester skriptet, som vi har kalt `KursKonvertering.m`:

```
>> KursKonvertering
Beløp i norske kroner: 512
BelopEuro =
   56.8889
BelopPund =
   39.9688
BelopSVK =
  537.2508
```

Utskrifta av svaret hadde kanskje vært litt mer elegant om vi erstatter linje 13-15 med

```
disp('Beløp i Euro: ')
disp(NOK/KursEuro)
disp('Beløp i Pund: ')
disp(NOK/KursPund)
disp('Beløp i Svenske kroner: ')
disp(NOK/KursSVK)
```

Oppgave 2 – Hva gjør disse skriptene?

b) Dette skriptet undersøker om det aktuelle tallet, `x`, er positivt eller negativt. I tillegg undersøker det om `x` er et heltall. Dersom det *er* det, vil det også undersøke om det er et partall eller et oddetall. Vi har her tatt for gitt at tallet er reelt. Vi viser noen eksempler – $x = 5$, $x = -8$ og $x = \pi$:

```
>> TypeTall
Gi et tall 5
Tallet er positivt
Tallet er et oddetall
>> TypeTall
Gi et tall -8
Tallet er negativt
Tallet er et partall
>> TypeTall
Gi et tall pi
Tallet er positivt
Tallet er ikke et heltall
```

d) En kommentert versjon av `TypeTall`-skriptet, kan se slik ut:

```
% Skript som, for et gitt tall, avgjør om tallet er positivt
% eller negativt, om det er er et heltall eller ikke, og - dersom
% det er et heltall - om det er et partall eller et oddetall.

% Leser inn tallet fra kommandovinduet
x=input('Gi et tall ');

% Avgjør om tallet er positivt eller negativt
if x<0
  disp('Tallet er negativt')
else
  disp('Tallet er positivt')
end

% Avgjør om tallet er et heltall
if round(x)~=x
  disp('Tallet er ikke et heltall')
else
  if round(x/2)==x/2            % Avjør om heltallet er et partall
      disp('Tallet er et partall')
  else
      disp('Tallet er et oddetall')
  end
end
```

Det bør nevnes at dette ikke er et spesielt interessant skript i seg; vi skal lage nyttigere skript enn dette. Meininga med denne oppgava er først og fremst å introdusere `if`-satser.

c) En litt mer "idiotsikker" versjon av skriptet `ABCformel` kan se slik ut:

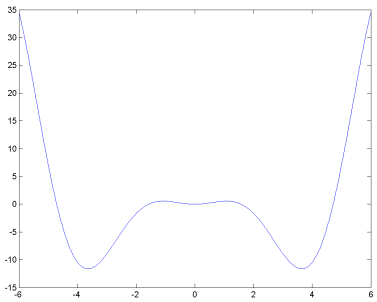
```
% Skript som løser likninga a x^2 + b x + c = 0
% Vi har tatt for gitt at ikke både a og b kan være null.

% Gir verdiene på a, b og c:
a=input('Gi verdien for a: ');
b=input('Gi verdien for b: ');
c=input('Gi verdien for c: ');

if a==0                          % Ser om det er en førstegradslikning
    x=-c/b
% Undersøker om vi har to, én eller ingen reelle løsninger
elseif b^2-4*a*c<0               % Ingen reell løsning
    disp('Ingen reell løsning')
elseif b^2-4*a*c==0              % En løsning
    x=-b/(2*a)
else                             % To løsninger
    x1=(-b-sqrt(b^2-4*a*c))/(2*a)
    x2=(-b+sqrt(b^2-4*a*c))/(2*a)
end
```

Om vi ser på de samme eksemplene som i a) en gang til, får vi:

```
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: -11
Gi verdien for c: 30
x1 =
     5
x2 =
     6
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: 1
Gi verdien for c: 2
Ingen reell løsning
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: -6
Gi verdien for c: 9
x =
     3
```

Her blir svaret 3 bare skrevet én gang. I tillegg er skriptet nå i stand til å håndtere førstegradslikninger også ($a = 0$):

```
>> ABCformel
```



Figur 3: Plott fra oppgave 3 b).

```
Gi verdien for a: 0
Gi verdien for b: -5
Gi verdien for c: 10
x =
     2
```

Her har vi valgt å ikke regne og skrive ut komplekse løsninger:

```
>> ABCformel
Gi verdien for a: 1
Gi verdien for b: 1
Gi verdien for c: 2
Ingen reell løsning
```

Selvsagt kunne vi ha valgt å ta med komplekse løsninger.

Oppgave 3 – Funksjoner i MATLAB

b) Problemet med `Funk`, slik som den er gitt i utgangspunktet, er at den bare tar skalare varibler; det fungerer ikke dersom `x` er en vektor. Problemet er ikkje værre enn at det løser seg om vi bruker punktum-skrivemåten:

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

Med dette, kan vi utføre kommandoene

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

og få plottet vist i figur 3.

Ekstraoppgave – Funksjonsfiler

b) når man skriver '`» help Funk`' i kommandovinduet får man opp kommentarene som står øverst:

```
% Funksjonen f(x)=x^2 cos x.
% Funksjonen tar bare skalarer som input.
```

c) Funksjonen

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

kan implementeres slik:

```
1  function F=DeltForskrift(x)
2
3  % Funsjonen f(x) der f=cos(pi*x) + 2 når x<0 og
4  % x^2-2 når x er større eller lik 2.
5  % Funksjonen tar bare skalare input.
6
7  if x<2
8    F=cos(pi*x)+2;
9  else
10   F=x^2-2;
11 end
```

Om man setter inn et punktum i linje 10 og forsøker å kalle funksjonen med et vektor-argument, ser det ut til å gå bra; man får et svar og det kommer ingen feilmelding. Men det som har skjedd er at barde det siste elementet i input-vektoren har blitt lest i linje 7. Eksempel:

```
>> x=0:2;
>> DeltForskrift(x)
ans =
    -2    -1     2
```

Her har uttrykket $x^2 - 2$ blitt brukt for alle tre x-verdiene, selv om dette bare var riktig for den siste.

d) Man kan slippe hele `if`-satsen ved å kombinere elementvis multiplikasjon, '`.*`', med logiske variable:

```
F=@(x) (x<2).*(cos(pi*x)+2) + (x>=2).*(x.^2+2);
```

Med en slik skrivemåte kan godt `x` være en vektor. For elementer i `x` som er mindre enn 2, vil "`x>=2`" gi 0 og "`x<0`" gi 1 – slik at det blir det første uttrykket, $\cos(\pi x) + 2$, gjelder. Motsatt, for elementer større eller lik 2, blir det første uttrykket ganget med 0 og det andre uttrykket, $x^2 + 2$ – skrevet med punktum-notasjon i MATAB, blir ganget med 1.