Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 6
Løsningsforslag

Oppgave 1 – Summer og `for`-løkker

a)

$$\sum_{i=1}^{10} i^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 + 8^2 + 9^2 + 10^2 =$$
$$1 + 4 + 9 + 16 + 25 + 36 + 49 + 64 + 81 + 100 = 385 \quad .$$

c) I kommandovinduet får vi opp følgende:

```
>> FinnSum
S =  385
```

Dette stemte jo bra. Om vi dropper semikolonet i linje 3, får vi

```
>> format compact
>>FinnSum
S =  1
S =  5
S =  14
S =  30
S =  55
S =  91
S =  140
S =  204
S =  285
S =  385
S =  385
```

d) Med kommentarer kan skriptet se slik ut:

```
1  % Skript som regner ut en sum
2
3  % Setter summen til 0
4  S=0;
5
6  for i=0:10          % Bestemmer summasjonsgrensene
7    S=S+i^2;          % Legger til et ledd
8  end
9
10 % Skriver summen til skjerm:
11 S
```

d) Skriptet kan lett tilpasses ved å endre 10 til 100 i linje 6. Linje 6 blir altså endra til

```
for i=1:100
```

Det er lurt allerede nå å venne seg til å lagre fila med hurtigtast, `Ctrl+S` på en PC, i stedet for å klikke i menyen; ellers kan det kan fort bli i overkant mye klikking. Om man bruker `Run`-knappen i menyen i teksteditoren, vil også fila bli lagra automatisk.

Om vi nå kjører skriptet, får vi svaret 338350.

e) Nedre grense skal nå være 5, og øvre grense skal være 20. Hvor hver gang legger vi til $\sqrt{i}$, som vi skriver som '`sqrt(i)`' i MATLAB:

```
1  % Skript som regner ut en sum
2
3  % Setter summen til 0
4  S=0;
5
6  for i=5:20              % Bestemmer summasjonsgrensene
7    S=S+sqrt(i);          % Legger til et ledd
8  end
9
10 % Skriver summen til skjerm:
11 S
```

Vi kjører skriptet og får (det tilnærma) svaret 55.5197.

f) I linje 2 ser vi at `i` går, i steg på 1, fra og med 1 til og med 12. Dette er altså summasjonsgrensene. I linje 3 ser vi at for hver omgang, hver *iterasjon*, blir det lagt til $1/i$. Summen er altså

$$\sum_{i=1}^{12} \frac{1}{i} \quad .$$

Oppgave 2 – Konvergens?

Vi tar utgangspunt i det samme skriptet igjen. Siden vi skal jusetere den øvre grensa flere ganger, kan det være en fordel å la denne grensa være en variabel som vi bestemmer i starten (se linje 4 og 9)[1]:

```
1  % Skript som regner ut en sum
2
3  % Bestemmer den øvre grensa
4  N=10;
5
6  % Setter summen til 0
7  S=0;
8
9  for i=-5:N
10   S=S+1/sqrt(i^2+1);          % Legger til et ledd
11 end
12
13 % Skriver summen til skjerm:
14 S
```

Vi får svaret 5.4582 når vi kjører det. Det at vi kaller indeksen `i` i skriptet, ikke `k` slik som i oppgaveteksten, spiller ingen rolle. Det spiller heller ingen rolle at `i` i utgangspunktet er den imaginære enhenten, $\sqrt{-1}$, i MATLAB; når vi tilordner noe annet til `i`, vil det som var der fra før, bli overskrevet. Når vi så skal la n ta andre verdier, endrer vi bare linje 4 i skriptet, lagrer og kjører det på nytt. For $n = 100$, $n = 1000$ og $n = 10000$ får vi svarene 7.7144, 10.0124 og 12.3146. Om vi tillater oss å fortsette med 100000 og 1000000, får vi svarene 14.6171 og 16.9197. Selv om summen øker veldig sakte når n øker, er det lite som tyder på at summen går mot noen bestemt verdi.

Det er forresten interessant å legge merke til hvor lite tid maskina bruker på å regne ut summen – selv med en million ledd. Min kontor-pc brukte 3 hundredels sekund på jobben.

[1] Til dette kunne vi godt ha brukt `input`-funksjonen vi så i leksjon 5.

Når vi så skal se på den andre summen, beholder vi `N` som variabel og justerer `for`-løkka:

```
for i=0:N
  S=S+i^2*2^(-i/2);          % Legger til et ledd
end
```

Med de samme n-verdiene som over, får vi svarene 33.8443, 48.0416, 48.0416 og 48.0416. Med fire desimaler ser vi ingen forskjell på de tre siste svarene. Om vi tar med litt flere desimaler (`» format long`), får vi svarene

```
33.844261793466430,
48.041630560319675,
48.041630560342625 og
48.041630560342625.
```

Selv ikke med 15 desimaler ser vi forskjell på de to siste summene. Mye tyder på at denne summen går mot noe endelig når den øvre grensa n går mot uendelig.

Oppgave 3 – Halveringsmetoden

a) Selv om likninga ser ganske enkel ut, har vi ingen teknikker for å løse denne med papir og blyant.

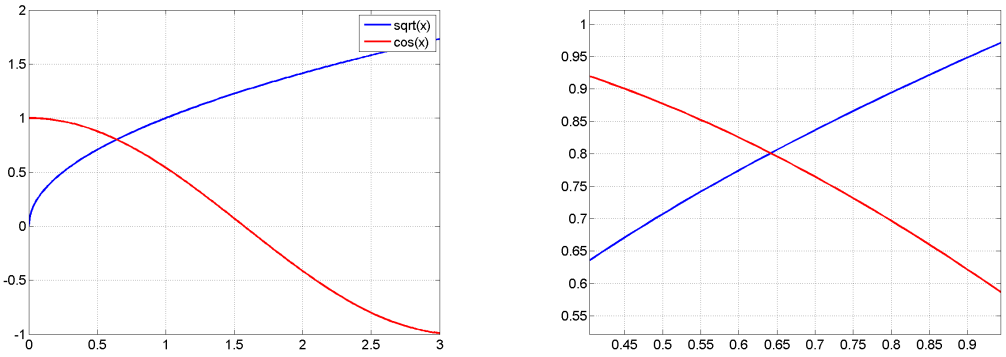
Vi plotter:

```
>> x=0:1e-2:3;
>> y1=sqrt(x);
>> y2=cos(x);
>> plot(x,y1,'linewidth',2)
>> hold on
>> plot(x,y2,'r','linewidth',2)
>> grid on
>> hold off
>> set(gca,'fontsize',15)
>> legend('sqrt(x)','cos(x)')
```

Resultatet ser vi i figur 1. Det ser ut til at vi har bare ett skjæringspunkt. Denne ene løsninga ser ut til å ligge i nærheten av $x = 0.65$. Her har vi satt på et rutenett med kommandoen '`grid on`'.

Vi kan skrive likninga som $\sqrt{x} - \cos x = 0$, eller $f(x) = 0$ der $f(x) = \sqrt{x} - \cos x$. Vi ser at

$$\begin{aligned} f(0) &= \sqrt{0} - \cos 0 = -1 < 0 \quad \text{og} \\ f\left(\frac{\pi}{2}\right) &= \sqrt{\frac{\pi}{2}} - \cos\frac{\pi}{2} = \sqrt{\frac{\pi}{2}} > 0 \quad . \end{aligned}$$



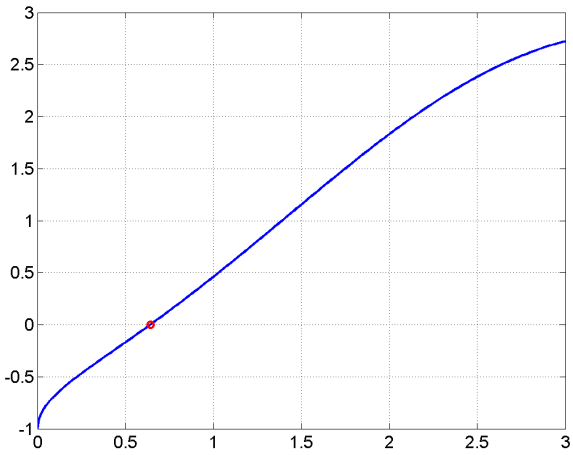
Figur 1: Plott av grafen til $\sqrt{x}$ og til $\cos x$. I plottet til høgre har vi "zoomet" inn på skjæringspunktet.

Altså har $f(x)$ ulike fortegn for $x = 0$ og $x = \pi/2$. Siden f er kontinuerlig, må den krysse x-aksen ($y = 0$) for (minst) en verdi mellom 0 og $\pi/2$. Denne sammenhengen kalles *skjæringssetninga*.

b) En kommentert versjon av skriptet gitt i oppgava kan se slik ut:

```
1  % Implementering av halveringsmetoden for likninga
2  % sqrt(x)-cos(x)=0 med a=0 og b=pi/2 som start-grenser
3
4  % Grenser
5  a=0;
6  b=pi/2;
7
8  % Funksjonsverdier
9  Fa=sqrt(a)-cos(a);
10 Fb=sqrt(b)-cos(b);
11
12 % Starter for-løkke som kjøres 10 ganger
13 for i=1:10
14   c=(a+b)/2;          % Midtpunktet
15   Fc=sqrt(c)-cos(c); % Funksjonsverdi i midtpunktet
16   if Fa*Fc<0
17     b=c;              % Setter ny b til c
18   else
19     a=c;              % Setter ny a til c
20   end
21 end
22
23 % Regner ut nytt midtpunkt og skriver svaret til skjerm
24 x=(a+b)/2
```

Vi har kalt det "`Halvering.m`". Vi har valgt å gjøre 10 iterasjoner (linje 13). I kommandovinduet gir det følgende svar:



Figur 2: Plott som viser $f(x) = \sqrt{x} - \cos x$ sammen med nullpunktet vi fant i oppgave 2b).

```
>> Halvering
x =
    0.6420
```

Når du skal implementere dette selv, må du ikke bli *for* frustrert om dette ikke går på første forsøk. Det er veldig vanlig å gjøre små feil, enten det er feil i logikken eller trykkfeil, som gjør at ting ikke fungerer med en gang. Når dette skjer, les feilmeldinga du får i MATLAB og se om du kan rette den opp. Ofte hjelper det også med friske øyne – å få noen andre til å ta ein kikk på skriptet ditt.

Vi plotter $f(x)$ sammen med nullpunkts-kandidaten vår:

```
>> xVektor=0:1e-2:3;
>> fVektor=sqrt(xVektor)-cos(xVektor);
>> plot(xVektor,fVektor,'linewidth',2)
>> grid on
>> hold on
>> plot(x,0,'ro','linewidth',2)
>> hold off
>> set(gca,'fontsize',15)
```

Som vi ser av figur 2, ser dette ut til å stemme ganske bra med den nøyaktigheten vi har i plottet.

Antall iterasjoner, 10, var her satt nokså tilfeldig. Vi kan få et intrykk av hvor nøyaktig svaret er om vi skriver midtpunktet `c` til skjerm for hver iterasjon (fjerner semikolonet i linje 14):

```
>> format compact
>> Halvering
>> Halvering
c =
    0.7854
c =
    0.3927
c =
    0.5890
c =
    0.6872
c =
    0.6381
c =
    0.6627
c =
    0.6504
c =
    0.6443
c =
    0.6412
c =
    0.6427
x =
    0.6420
```

Men siden vi vet at et intervall med lengde $\pi/2$ skal halveres 10 ganger, vil intervallet til slutt ha lengda $\frac{\pi/2}{2^{10}}$. Siden vi til sist velger midtpunktet i dette intervallet, vet vi med sikkerhet at feilen er mindre enn $\frac{\pi}{2^{12}} \approx 0.000767$. Dette med å ha kontroll på hvor stor feilen er – og å kunne gjøre den mindre og mindre på en systematisk måte, er helt essensielt når det gjelder numeriske metoder.

Før vi gjør oppgave c): Skriptet over gjør jobben helt fint. Men der er en veldig unødvendig svakhet ved det. Vi har skrevet inn det samme funksjonsuttrykket tre ganger – i linje 9, 10 og 15.. Dette er uheldig fordi det gjør skrptet sårbart for skrivefeil; om vi skriver en liten feil i ett av de tre uttrykkene, blir alt feil. Og det er uheldig fordi det gjør skriptet lite fleksibelt. Om vi skal løse ei anna likning, må vi endre skriptet tre steder. Det hadde vært enklere om vi bare må endre funksjonsuttrykket i éi linje.

Dette kan vi få til ved å definere en funksjon for det uttrykket som skal være null. Til dette kan vi for eksempel bruke @-notasjonen vi så i leksjon 5. En litt mer fleksibel versjon av skriptet kan se slik ut:

```
% Implementering av halveringsmetoden for likninga f(x)=0 der
% f er kontinuerlig på [a,b] og f(a) og f(b) har ulike fortegn.

% Grenser
a=0;
b=pi/2;

% Gir funksjonen som skal være null
NullFunk=@(x) sqrt(x)-cos(x);

% Funksjonsverdier
Fa=NullFunk(a);
Fb=NullFunk(b);

% Starter for-løkke som kjøres 10 ganger
while (b-a)>1e-5
  c=(a+b)/2;          % Midtpunktet
  Fc=NullFunk(c); % Funksjonsverdi i midtpunktet
  if Fa*Fc<0
    b=c;             % Setter ny b til c
  else
    a=c;             % Setter ny a til c
  end
end

% Regner ut nytt midtpunkt og skriver svaret til skjerm
x=(a+b)/2
```

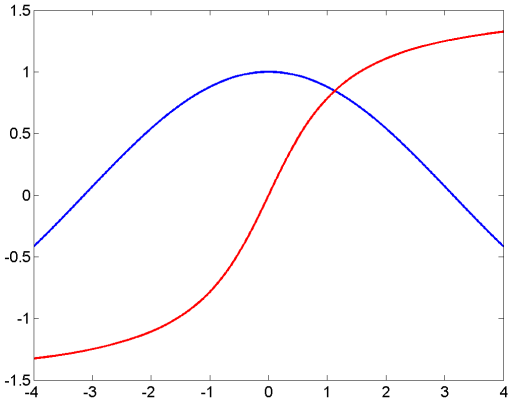
Merk at kommentarene øverst i skriptet også er oppdatert. Vi bruker dette skriptet videre.

c) i) $\cos\frac{x}{2} = \arctan x$
For å få en idé om hvor mange løsninger likninga har – og for å kunne velge passende verdier for a og b, plotter vi funksjonen først:

```
>> x=-4:1e-2:4;
>> y=cos(x/2); z=atan(x);
>> plot(x,y,'linewidth',2)
>> hold on
>> plot(x,z,'r','linewidth',2)
>> hold off
```

Plottet er vist i figur 3. Ut fra figuren ser likninga ut til å ha bare éi løsning. Videre ser det ut til at løsninga ligger mellom 0 og 2. Vi sjekker at den kontinuerlige funksjonen $f(x) = \cos(x/2) - \arctan x$ faktisk skifter fortegn på dette intervallet:

```
>> x=0;
>> f=cos(x/2)-atan(x)
```



Figur 3: Plott av funksjonene $\cos(x/2)$ (blått) og $\arctan x$ (rødt).

```
f =
     1
>> x=2;
>> f=cos(x/2)-atan(x)
f =
   -0.5668
```

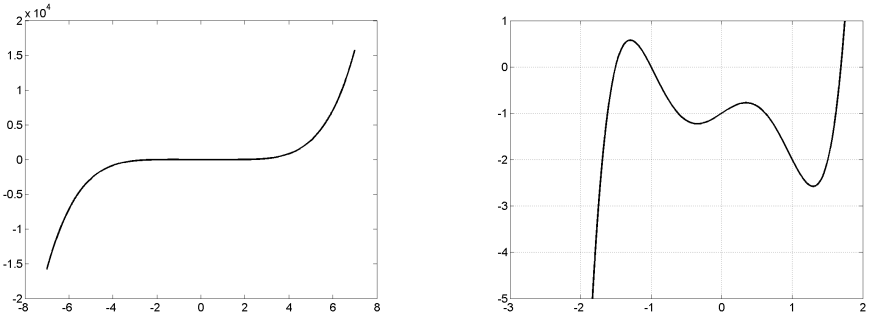
Vi løser likninga ved å bruke det samme skriptet som over. Vi oppaterer b i linje 6 (a var satt til null fra før) og funksjonen i linje 9:

```
1  % Implementering av halveringsmetoden for likninga f(x)=0 der
2  % f er kontinuerlig på [a,b] og f(a) og f(b) har ulike fortegn.
3
4  % Grenser
5  a=0;
6  b=2;
7
8  % Gir funksjonen som skal være null
9  NullFunk=@(x) cos(x/2)-atan(x);
10 ...
```

Vi kjører skriptet:

```
>> Halvering
x =
    1.1279
```

Siden vi har kjørt `for`-løkka 10 ganger og så valgt midtpunktet mellom a og b til svar, må feilen være mindre enn $2/2^{11} \approx 9.77 \cdot 10^{-4}$. Om vi hadde erstatta 10 i linje 16 med et høyere tall, ville vi fått et mer nøyaktig svar.



Figur 4: Plott av femtegradspolynomet vi skal finne nullpunktene til i deloppgave 3c) iii)

ii) $e^x = -\ln x$

Også denne likninga har bare éi løsning. At det ikkje kunne være flere enn éi kunne vi sagt med en gang siden både eksponential- og logaritmefunksjonen er én-entydige (*injektive*). Det er mange måter å velge a og b på. En måte kan være $a = 0.1$ og $b = 0.5$ siden den kontinuerlige funksjonen $f(x) = e^x + \ln x$ har ulike fortegn for disse argumentverdiene:

```
>> x=.1;
>> f=exp(x)+log(x)
f =
   -1.1974
>> x=.5;
>> f=exp(x)+log(x)
f =
    0.9556
```

Vi endrer linje 4 – 9:

```
% Grenser
a=.1;
b=.5;

% Gir funksjonen som skal være null
NullFunk=@(x) exp(x)+log(x);
```

Når vi kjører skriptet, frå vi løsninga $x \approx 0.2697$.

iii) $x^5 - 3x^3 + x - 1 = 0$

Dette er ei 5.-gradslikning, som i utgangspunktet kan ha 5 løsninger. Vi plotter polynomet for å undersøke nærmere. I figur 4 er polynomet plotta med to ulike intervaller på aksene. Det kan se ut til at polynomet har 3 nullpunkter – ett mellom 1 og 2, ett mellom -2 og -1 og ett nære -1. Vi ser ganske fort at $x = -1$ faktisk er ei eksakt løsning:

$$(-1)^5 - 3 \cdot (-1)^3 + (-1) - 1 = 0 \quad .$$

For å finne de to andre, bruker vi skriptet vårt. Vi finner at vi kan velge $a = -2$ og $b = -1.2$ for det første nullpunktet:

```
% Grenser
a=-2;
b=-1.2;

% Gir funksjonen som skal være null
NullFunk=@(x) x.^5-3*x.^3+x-1;
```

Når vi kjører skriptet, får vi svaret $x \approx -1.5051$.
For det siste nullpunktet, velger vi $a = 1$ og $b = 2$. Dette gir svaret $x \approx 1.6899$.

Ekstra-opgave: Maksimering

Siden det skal være ei funksjonsfil med vektoren `x` som argument, må første linje være slik:

```
function M=MaksFunk(x)
```

Selvsagt kunne funksjonen hatt et annet navn enn '`MaksFunk`', og ut-variabelen må ikke nødvendigvis hete '`M`'.

Vi skal finne det største elementet i `x`-vektoren. Det kan vi gjøre ved å gå gjennom vektoren element for element. For hvert nytt element sjekker vi om det er større enn det som har vist seg å vere størst så langt. Dersom det nye elementet ikke er større, gjør vi ingenting. Dersom elementet *er* større, oppdaterer vi hva som er det foreløpig største elementet. Vi kan starte med å si at det foreløpig største elementet er det første elementet i `x` – `M=x(1)`.

Dette er ei funksjonfil som implementerer nettopp dette:

```
1  function M=MaksFunk(x)
2
3  % Funksjon som finner det største elementet i vektoren x
4
5  M=x(1);             % tilordner M - foreløpig største element
6
7  for xElement=x      % for-løkke som løper gjennom x
8    if xElement>M
9      M=xElement;     % om elementet er større enn M, oppdaterer vi M
10   end
11 end
```

Vi tester funksjonsfila, som vi har kalt `MaksFunk.m`, på litt ulike vektorer. I kommandovinduet kan det se slik ut:

```
>> x = -2:3
    -2    -1     0     1     2     3
>> MaksFunk(x)
ans =
     3
>> x=[7 pi -4 14 3 9];
>> MaksFunk(x)
ans =
    14
>> y=3-(x-sqrt(2)).^2;
>> MaksFunk(y)
ans =
    2.8284
>> x=-2:.5:3;
>> y=3-(x-sqrt(2)).^2;
>> MaksFunk(y)
ans =
    2.9926
>> x=-2:0.1:3;
>> y=3-(x-sqrt(2)).^2;
>> MaksFunk(y)
ans =
    2.9998
```

Vi ser at når vi bruker finere og finere oppdeling i `x`-vektoren, ser maksimalverdien til `y` ut til å nærme seg 3, som er maksimalverdien til funksjonen $f(x) = 3 - (x - \sqrt{2})^2$. (Hvorfor er det mer eller mindre opplagt?)