Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 5
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 0 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

$$\sum_{i=0}^{12} \frac{1}{i} \quad .$$

Vi kommer tilbake til summer av denne typen når vi senere i kurset skal beregne integraler.

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. 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 3.

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.

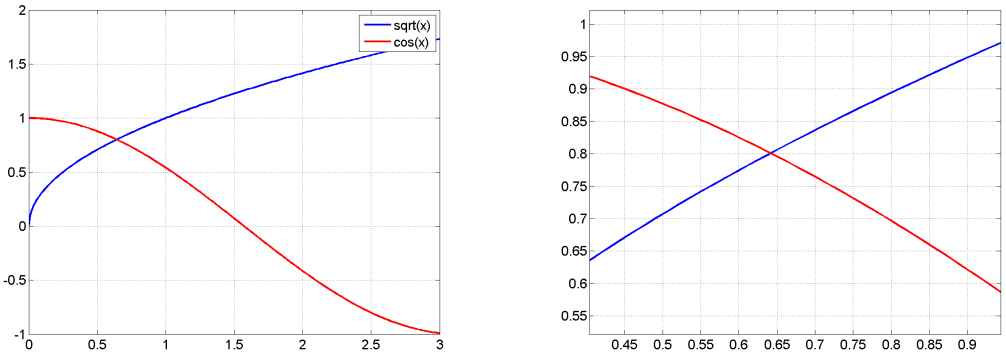
Vi plotter:

```
>> 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: Plot av grafen til $\sqrt{x}$ og til $\cos x$. I plottet til høgre har vi "zoomet" inn på skjæringspunktet.

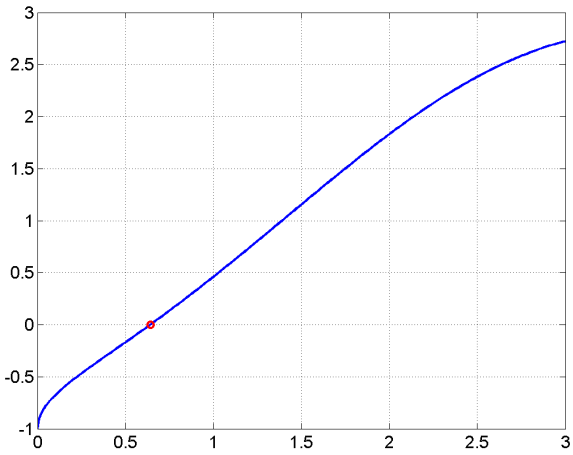
Altså har $f(x)$ ulike fortegn for $x = 0$ og $x = \pi/2$. Siden f er kontinuerlig, må den krysse x-aksen ($y = 0$) for (minst) en verdi mellom 0 og $\pi/2$[2].

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

```
% Implementering av halveringsmetoden for likninga
% sqrt(x)-cos(x)=0 med a=0 og b=pi/2 som start-grenser

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

% Funksjonsverdier
Fa=sqrt(a)-cos(a);
Fb=sqrt(b)-cos(b);

% Starter for-løkke som kjøres 10 ganger
for i=1:10
  c=(a+b)/2;          % Midtpunktet
  Fc=sqrt(c)-cos(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
```

Vi har kalt det "`Halvering.m`". Vi har valgt å gjøre 10 iterasjoner (linje

[2]Denne sammenhengen kalles *skjæringssetninga*.



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

13). I kommandovinduet gir det følgende svar:

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

Når du skal implemengere 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 på en systematisk måte, er helt essensielt når det gjelder numeriske metoder.

Oppgave 4 – Fakultetfunksjonen

a) Vi har nå sett hvordan vi kan bruke `for`-løkker til å utføre gjentatte addisjoner. Vi kan bruke den samme tankegangen til å utføre gjentatte multiplikasjoner. Funksjonen kan implementeres slik:

```
1  function F=Fakultet(n)
2
3  % Funksjon som regner ut n!, n-fakultet, for det naturlige tallet n
4  % Funksjonen tar bare skalarer som input.
5
6  % Begynner med å sette F til èn
7  F=1;
```

```
for i=1:n
  F=F*i;        % Mulitipliserer med i med i fra 1 til og med n
end
```

Vi lagrer den som "`Faktultet.m`" og tester den i kommandovinduet:

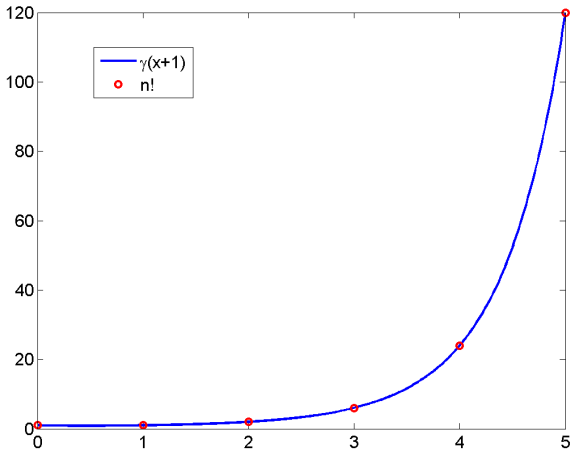
```
>> format compact
>> Fakultet(1)
ans =
     1
>> Fakultet(4)
ans =
    24
>> Fakultet(10)
ans =
     3628800
>> Fakultet(20)
ans =
   2.4329e+18
```

$n!$ blir tydeligvis ganske stor ganske fort. Slik vi har implementert funksjonen nå, gir den faktisk rett svar for $n = 0$ også. Men dette er ikke så opplagt. Vi kan bruke `if`-satser til å tvinge funksjonen til å gi rett svar for $n = 0$. Vi kan også bruke `if`-satser til å presisere at argumentet ikke kan være negativt:

```
function F=FakultetV2(n)

% Funksjon som regner ut n!, n-fakultet, for det naturlige tallet n
% Funksjonen tar bare skalarer som input.

if n<0
  disp('n kan ikke være negativ')
  return
elseif n==0
  F=1;
else
  F=1;              % Setter F til én
  for i=1:n
    F=F*i;          % Mulitipliserer med i med i fra 1 til og med n
  end
end
```

Vi tester "versjon 2" også:

```
>> FakultetV2(-5)
n kan ikke være negativ
>> FakultetV2(0)
```



Figur 3: Plott av $\gamma(x+1)$ og $n!$.

```
ans =
     1
>> FakultetV2(4)
ans =
    24
```

Vi kunne også brukt en `if`-sats for å gi ei feilmelding dersom man satte inn et ikke-heltallig argument.

b) Når vi nå lager et plott, er det mer praktisk å bruke MATLABs egen fakultets-funksjon siden denne tar vektor-argumenter:

```
>> x=0:1e-2:5;
>> y=gamma(x+1);
>> n=0:5;
>> y2=factorial(n);
>> plot(x,y,'linewidth',2)
>> hold on
>> plot(n,y2,'ro','linewidth',2)
>> hold off
>> set(gca,'fontsize',15)
>> legend('\gamma(x+1)','n!')
```

Resultatet er vist i figur 3. I den siste linja, den som lager tekstboksen som forklarer hva grafene er, har vi skrevet "`\gamma`" for å få fram den greske bokstaven. Slik koding, LaTeX-kode, gjør at det ser litt "penere" ut, men det ser selvsagt ikke noe vi forventer at dere skal kunne[3].

[3] Spør gjerne om LaTeXom du er interessert!

Ekstra-opgave 1 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:

```
>> format compact
>> x = -2:3
    -2    -1     0     1     2     3
>> MaksFunk(x)
ans =
     3
>> 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?)

Ekstra-oppgave 2: Halvering og plotting

Det aller første vi gjør er å plotte funksjonen og endepunktene. Vi bruker x-verdier som går fra litt lavere enn a til litt større enn b:

```
xVektor=a:(b-a)/500:b;
yVektor=sqrt(xVektor)-cos(xVektor);
plot(xVektor,yVektor,'linewidth',2)
hold on
plot([a b],[0 0],'rx','linewidth',2)
plot([a b],[0 0],'k-')
set(gca,'fontsize',15)
```

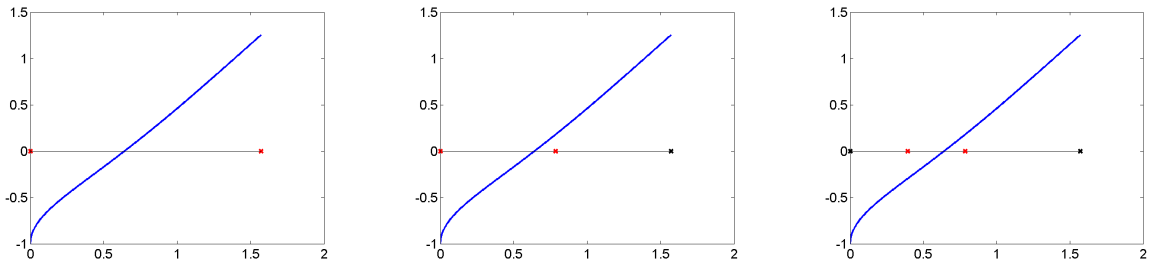
Her har vi også tatt med x-aksen som ei tynn, svart linje. Vi kunne også ha valgt å bruke `grid on`-funksjonen. Vi lar `hold` fortsatt være på. Dette plottet må vi oppdatere inni `for`-løkka. Vi kan selvsagt velge å plotte alt på nytt. Men det kan være greit å se de gamle grensene også – og hvordan det aktuelle intervallet blir smalere og smalere. Dette får vi til om vi plotter de gamle grensene i svart og de aktuelle grensene i rødt. Dette kan implementeres på denne måten:

```
% Implementering av halveringsmetoden for likninga
% sqrt(x)-cos(x)=0 med a=0 og b=pi/2 som start-grenser
% Funksjonen og de aktuelle grensene blir plotta fortløpende

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

% Iterasjoner
N=7;

```

```
12 % Plotter
13 xVektor=a:(b-a)/500:b;
14 yVektor=sqrt(xVektor)-cos(xVektor);
15 plot(xVektor,yVektor,'linewidth',2)
16 hold on
17 plot([a b],[0 0],'rx','linewidth',2)
18 plot([a b],[0 0],'k-')
19 set(gca,'fontsize',15)  % Justerer skriftstørrelsen på aksene
20 pause                  % Tar pause
21
22 % Funksjonsverdier
23 Fa=sqrt(a)-cos(a);
24 Fb=sqrt(b)-cos(b);
25
26 % Starter for-løkke som kjøres N ganger
27 for i=1:7
28   plot([a b],[0 0],'kx','linewidth',2)    % Gjør gamle grenser svarte
29   c=(a+b)/2;         % Midtpunktet
30   Fc=sqrt(c)-cos(c); % Funksjonsverdi i midtpunktet
31   if Fa*Fc<0
32     b=c;             % Setter ny b til c
33   else
34     a=c;             % Setter ny a til c
35   end
36   plot([a b],[0 0],'rx','linewidth',2)    % Plotter nye grenser i rødt
37   pause              % Pauser
38 end
39 hold off
40
41 % Regner ut nytt midtpunkt og skriver svaret til skjerm
42 % Vi regner også ut usikkerheten
43 x=(a+b)/2;
44 DeltaX=(b-a)/2;
45 disp(['Løsning: ',num2str(x),'.'])
46 disp(['Usikkerhet: ',num2str(DeltaX),'.'])
```

Vi kjenner igjen linje 13 - 19 fra over. Her har vi i tillegg "pynta" litt på skriptet fra oppgave 3. Vi har gjort antall iterasjoner til en variabel som blir tilordna i linje 10. Videre har vi regna ut usikkerheten i svaret, linje 44, og skrevet ut denne til skjerm sammen med løsninga. Vi har også gjort disse utskriftene litt mer elegante ved å bruke `disp`-kommandoen, som gjør at navnet på variabelen ikke blir skrevet til skjerm, og ved å bruke `num2str`-kommandoen, som gjør oss i stand til å kombinere tekst og tall.

En annen forbedring kunne være å lage ei funksjonsfil i MATLAB for den funksjonen vi skal finne nullpunktet til. Denne ville vi i så fall ha referert til i linje 14, 23, 24 og 30 og dermed sluppet å skrive mer eller mindre det samme fire



Figur 4: De tre første iterasjonene av halveringsmetoden.

ganger. I tillegg ville skriptet ha blitt mer fleksibelt; om vi skulle ha løst ei anna liking, ville vi bare ved ei justering av funksjonsfila ha fått oppdatert funksjonen på alle de aktuelle stedene i skriptet. Dette kunne vi også ha gjort ved å bruke @-skrivemåten i selve skriptet – (`funk=@(x) sqrt(x)-cos(x);`).

De tre første figurene vi får når vi kjører skriptet er vist i figur 4.

Vi har kalt `HalveringMedPlott.m`. I kommandovinduet får vi

```
>> HalveringMedPlott
Løsning: 0.64427.
Usikkerhet: 0.0061359.
```

når vi kjører skriptet med `N`=7.