Innlevering i	BYFE/EMFE 1000
	Oppgavesett 1
Innleveringsfrist:	14. september klokka 14:00
Antall oppgaver:	3

Løsningsforslag

Oppgave 1

a) $\ln\sqrt{a} - \ln\sqrt[3]{a} + \ln\sqrt[4]{a} = \ln a^{1/2} - \ln a^{1/3} + \ln a^{1/4} = \frac{1}{2}\ln a - \frac{1}{3}\ln a + \frac{1}{4}\ln a = \frac{6-4+3}{12}\ln a = \underline{\underline{\frac{5}{12}\ln a}}$

b) $\frac{2^3\cdot 4^2}{e^{\ln 32}} = \frac{2^3\cdot(2^2)^2}{32} = \frac{2^3\cdot 2^4}{2^5} = 2^{3+4-5} = 2^2 = \underline{\underline{4}}$

c)

$$\begin{aligned}
\sin^2 x - \cos^2 x &= 0 \\
-(\cos^2 x - \sin^2 x) &= 0 \\
-\cos(2x) &= 0 \\
\arccos 0 &= \frac{\pi}{2} \\
2x &= \frac{\pi}{2} + n\cdot 2\pi \quad \vee \quad 2x = -\frac{\pi}{2} + n\cdot 2\pi, \quad n\in\mathbb{Z} \\
x &= \frac{\pi}{4} + n\pi \quad \vee \quad x = -\frac{\pi}{4} + n\pi
\end{aligned}$$

Vi har her brukt at $\cos(2x) = \cos^2 x - \sin^2 x$.
Siden vi skal ha at $x \in [0, 2\pi)$, får vi svaret

$$\underline{\underline{x = \frac{\pi}{4} \quad \vee \quad x = \frac{3\pi}{4} \quad \vee \quad x = \frac{5\pi}{4} \quad \vee \quad x = \frac{7\pi}{4}}}$$

d)

$$\begin{aligned}
\frac{e^x}{1+4e^{-x}} &= 2 \\
e^x &= 2(1+4e^{-x}) = 2 + 8e^{-x} \\
e^x - 2 - 8e^{-x} &= 0 \\
e^x\left(e^x - 2 - 8e^{-x}\right) &= e^x\cdot 0 = 0 \\
(e^x)^2 - 2e^x - 8 &= 0
\end{aligned}$$

Dette er altså en 2.-gradslikning i e^x:

$$\begin{aligned} e^x &= \frac{-(-2) \pm \sqrt{(-2)^2 - 4 \cdot 1 \cdot (-8)}}{2 \cdot 1} = \frac{2 \pm 6}{2} = 1 \pm 3 \\ e^x &= 1 - 3 = -2 \quad \vee \quad x = 1 + 4 = 4 \end{aligned}$$

e^x kan aldri bli negativ for $x \in \mathbb{R}$.

$$\begin{aligned} e^x &= 4 \\ x &= \underline{\underline{\ln 4}} \end{aligned}$$

Oppgave 2

$$g(t) = \begin{cases} \cos(3t^2), & t < 0 \\ 1 - t^2, & t \geq 0 \end{cases}$$

a) Krav til kontinuitet i $t = 0$:

$$\lim_{t \to\ 0} g(t) = g(0)$$

For at grenseverdien på høyre side skal være definert, må de ensidige grenseverdiene vere like:

$$\begin{aligned} \lim_{t \to\ 0^-} g(t) &= \lim_{t \to\ 0^-} \cos(3t^2) = \cos 0 = 1 \\ \lim_{t \to\ 0^+} g(t) &= \lim_{t \to\ 0^+} (1 - t^2) = 1 - 0^2 = 1 = \lim_{t \to\ 0^-} g(t) \end{aligned}$$

Grenseverdien er veldefinert og lik 1. Funksjonsverdien $g(0) = 1 - 0^2 = 1$. Altså er kriteriet over oppfylt og $\underline{\underline{g \text{ er kontinuerlig for } t = 0}}$.

b) Funksjonsfil for $g(t)$ kan lages slik:

```
function G=FunksjonOblig1(t)

% Denne fila gir funksjonen g(t), som er lik cos(3t^2) naar t<0 og
% 1-t^2 naar t>=0. Input maa vaere en skalar.

if t<0
 G=cos(3*t^2);
else
 G=1-t^2;
end
```

Den har fått navnet `FunksjonOblig1.m`, og kan kalles fra Octave slik:

```
octave-3.2.4.exe:28> FunksjonOblig1(5)
ans = -24
octave-3.2.4.exe:29> FunksjonOblig1(-2)
ans =  0.84385
```

Merk at denne måten å lage funksjonsfila på bare tillater skalare input-variable. Det går også an å konstruere funksjonsfila slik at den *kan* ta vektorer som input. Det kan gjøres ved å bruke ei `for`-løkke inni funksjonsfila eller slik:

```
function G=FunksjonOblig1V2(t)

% Denne fila gir funksjonen g(t), som er lik cos(3t^2) naar t<0 og
% 1-t^2 naar t>=0. Input kan vaere en skalar eller en vektor.

G=(t<0).*cos(3*t.^2) + (t>=0).*(1-t.^2);
```

Denne funksjonsfila har jeg gitt navnet `FunksjonOblig1V2.m` (navnet trenger ikkje være det sammme som i første linje i fila, men det er nok ryddigst om det er slik). Når vi skriver '`t<0`', gir det en vektor av samme størrelse som `t` hvor hvert element er enten 1 eller 0 – avhengig av om det tilsvarende elementet i `t` er mindre enn 0 eller ikke. Men en slik funksjonfil kan man regne ut mange g-verdier samtidig:

```
octave-3.2.4.exe:7> FunksjonOblig1V2(-2:.5:2)
ans =

 Columns 1 through 7:

   0.84385   0.89301  -0.98999   0.73169   1.00000   0.75000   0.00000

 Columns 8 and 9:

  -1.25000  -3.00000
```

Merk at dersom vi hadde brukt punktum-notasjon i funksjonsuttrykkene i den første versjonen av funksjonsfila, hadde vil blitt lurt. Vektor-argument hadde blitt lest, men bare det siste elementet i input-vektoren hadde blitt brukt i `if`-satsene, slik at svaret ville ha blitt feil for alle negative t-verdier.

c) Når vi skal plotte grafen, er det en klar fordel å ha ei funksjonsfil som kan ta vektor-argument. Da kan det gjøres slik:

```
t=-4:.1:4;
g=FunksjonOblig1V2(t);
plot(t,g)
```

Eller enda mer kompakt:

```
t=-4:.1:4;
plot(t,FunksjonOblig1V2(t))
```

Vi kan godt sette navn på aksene også:

```
xlabel('t')
ylabel('g')
```

Dersom en funksjonfila bare tar skalare argument, må vi bruke ei `for`-løkke for å lage plottet. Det er i så fall best å lage et skript som man lagrer som ei `m`-fil. Skriptet kan for eksempel se slik ut:

```
tVektor=-4:.01:4;       % Lager en vektor med argument-verdier

ind=1;
for t=tVektor           % Løkke som løper gjennom alle argumentene
  gVektor(ind)=FunksjonOblig1(t); % Tilordner funksjonsverdier
  ind=ind+1;            % Øker indeksen med én
end
plot(tVektor,gVektor)   % Plotter grafen
xlabel('t')             % Skriver 't' på x-aksen
ylabel('g')             % Skriver 'g' på y-aksen
```

Dette skriptet heter `PlotteSkript.m` og kan kalles fra Octave ved å skrive `PlotteSkript`. Husk at du må være i samme katalog som skriptet ligger i. Figuren blir skrevet til fil ved hjelp av kommandoen print. Der er flere mulige formater. Om vi skriver

```
print -dpng FigurTilOppg2.png
```

får vi en png-fil med navnet `FigurTilOppg2.png`. `jpg` og `pdf` er også anvendelige formater. Figur 1 viser plottet. Vi ser at grafen ser ut til å "henge sammen" – også omkring $t = 0$.

Oppgave 3

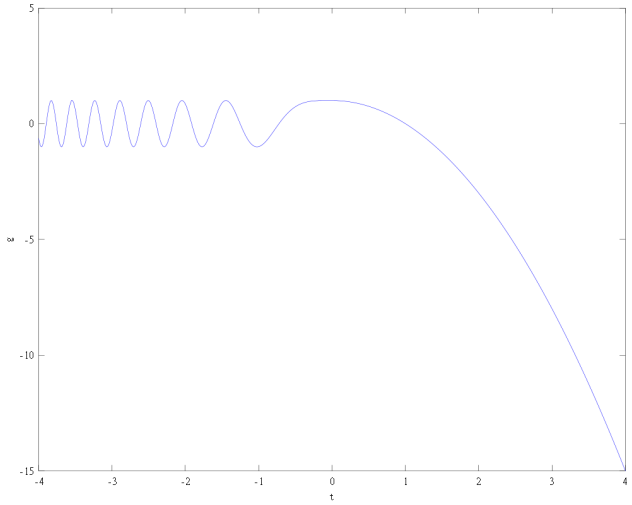
$$f(x) = x - 4\sin^2(x) + 1 \quad .$$

a) Funksjonen en sum av produkt av elementære funksjoner som alle har hele $\mathbb{R}$ som naturlig definisjonsmengde. Den er derfor kontinuerlig på hele $\mathbb{R}$.

$$\begin{aligned} f(0) &= 1 > 0 \\ f(\pi/2) &= \pi/2 - 4\sin^2\left(\frac{\pi}{2}\right) + 1 = -(3 - \pi/2) < 0 \end{aligned}$$

Siden $f(0)$ og $f(\pi/2)$ har ulike fortegn og funksjonen er kontinuerlig på hele det lukkede intervallet, må funsjonen ha minst ett nullpunkt ved skjæringssetninga. (Se setning 2.5.11 side 97 i Kalkulus-boka.)

b) Vi kan lage ei ny funksjonsfil, som i oppgave 2, og plotte denne. Jeg har kalt funksjonfila `FunkOblig1Oppg3.m`:



Figur 1: Plot av funksjonen $g(t)$ i oppgave 2.

```
funktion F=FunkObl1Oppg3

% Denne funksjonsfila returnerar verdien av funksjonen
% f(x) = x-4 sin^2(x) +1.
% Funksjonen godtar vektor-argument.

F=x-4*sin(x).^2+1;
```

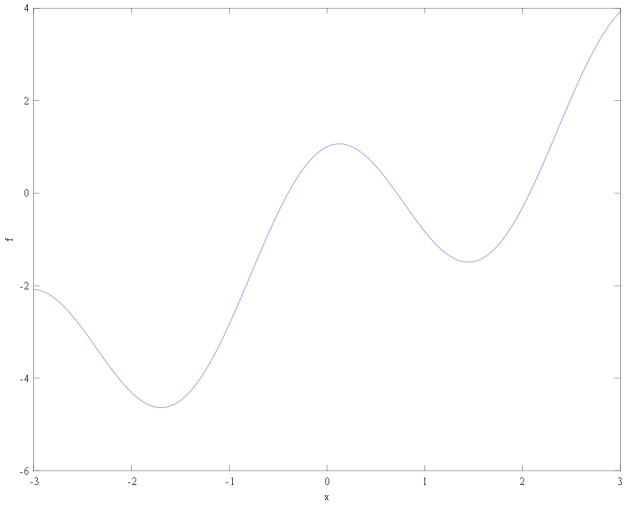
Plottinga kan vi gjøre som i oppgave 2. Vi velger at variabelen går fra -3 til 3 med steg på 0.01 (dette er nok rikelig fint nok – eksperimentér gjerne med større steg).

```
octave-3.2.4.exe:10> x=-3:.01:3;
octave-3.2.4.exe:11> plot(x,FunkObl1Oppg3(x))
octave-3.2.4.exe:12> xlabel('x'); ylabel('f')
octave-3.2.4.exe:13> print -dpng FigurTil0ppg3.png
```

Det går også bra å lage plottet uten å lage noen funksjonfil:

```
octave-3.2.4.exe:10> x=-3:.01:3;
octave-3.2.4.exe:11> plot(x,x-4*sin(x).^2+1)
```

Ut fra plottet, figur 2, ser det ut til at grafen skjærer x-aksen én gang i intervallet. Det er vanskelig å se nøyaktig hvor grafen krysser x-aksen.



Figur 2: Plot av funksjonen $f(x)$ i oppgave 3.

For det første er det ikke så lett å se hvor x-aksen går, og for det andre er området vi har plottet grafen på ganske stort. Vi plotter x-aksen og "zoomer" inn:

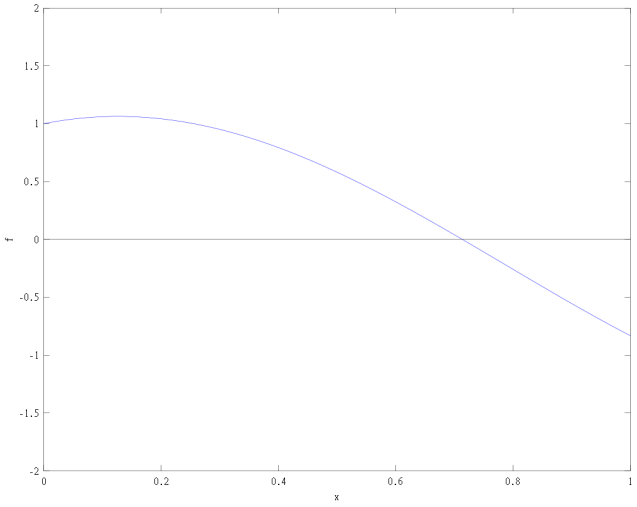
```
hold on
plot([-3 3],[0 0],'k')
axis([0 1 -2 2])
```

I figur 3 ser vi at $f(x) \approx 0$ for $\underline{\underline{x \approx 0.7}}$. Dette ser også ut til å være det eneste nullpunktet for $x \in [0, \pi/2]$. Om vi "zoomer" enda mer, får vi et svar som er litt mer nøyaktig. Og slik kan vi fortsette å "zoome" oss inn til et mer og mer nøyaktig svar (om steglengda i input-vektoren er liten nok). Men i neste deloppgave skal vi gjøre dette på en litt mer sofistikert måte.

c) Halveringsmetoden kan implementers slik:

```
% Skript som implementerer halveringsmetoden.

% Angir endepunkt og presisjon
a=0;
b=pi/2;
Tol=1e-10;
% Funksjonsverdier
```



Figur 3: Samme plot som i figur 3, men med x-aksen agrensa til $[0, 1]$.

```
Fa=FunkObl1Oppg3(a);
Fb=FunkObl1Oppg3(b);

% While-løkke som gjentas så lenge avstanden mellom a og b
% er større enn ønsket nøyaktighet
while abs(a-b)>Tol
  x=(a+b)/2;
  Fx=FunkObl1Oppg3(x);   % Tilsvarende funksjonsverdi
  if Fa*Fx<0              % Dersom om f(a) og f(x) har ulike fortegn
    b=x;                  % Velger ny høgre grense
  else                    % Dersom f(a) og f(x) har like fortegn
    a=x;                  % Velger ny venstre grense
  end
end

% Skriver løsninga til skjerm:
Nullpunkt=(a+b)/2
```

Men dette er ingen optimal impelementering. Ofte vil det være en fordel om man kan skrive skriptet på en slik måte at man må forandre minst mulig om man skal løse et lignende problem; man vil gjerne gjøre det *generisk*. Derfor vil dette være et bedre skript:

```
% Skript som implementerer halveringsmetoden.
% Metoden gir ei tilnærma løsning av likninga f(x)=0. Det forutsettes
% at funksjonen f(x) er kontiunerlig på det aktuelle intevallet.
% Skriptet vil be om endepunkter, og det kontrollerer at funksjonen
% faktisk har ulike fortegn for disse endepunktene. Navnet på funksjonen
% er 'hardkodet' inn i skriptet

% Leser inn venstre grense
a=input('Venstre grense: ');
% Leser inn høgre grense
b=input('Hoeyre grense: ');
% Angir ønsket presisjon
Tol=input('Noeyaktighet: ');

% Gir navnet på funksjonen
Funk=@FunkObl1Oppg3;

% Tilordne funksjonsverdiene i grensepunktene
Fa=Funk(a);
Fb=Funk(b);
% Sjekk at de faktisk har ulikt fortegn
if Fa*Fb>=0
 'f(a) og f(b) har samme fortegn'
 return
end

% Midtpunkt
x=(a+b)/2;

% Its er antall iterasjoner
Its=0;
% While-løkke som gjentas så lenge |pres| er større enn ønsket nøyaktighet
while abs(a-b)>Tol
  x=(a+b)/2;    % Nytt midpunkt
  Fx=Funk(x);   % Tilsvarende funksjonsverdi
  if Fa*Fx<0    % Sjekker om f(a) og f(x) har ulike fortegn
    b=x;        % Velger ny høgre grense
  else
    a=x;        % Velger ny venstre grense
  end
  Its=Its+1;    % Antall iterasjoner økes med én
end

Nullpunkt= x    % Skriver løsninga til skjerm
Iterasjoner=Its % Skriver antall iterasjoner til skjerm
```

I tillegg til å vere litt mer generisk og bedre dokumentert, har også dette skriptet den fordelen at det teller antall iterasjoner (`Its`) og det har en `if`-sats som kontrollerer at funksjonen faktisk har ulike fortegn i endepunktene (hvis ikke, vil ikke resten av skriptet bli utført). Legg merke til bruken av `@`; den tillater at vi gir funksjonsnavnet bare én gang – til forskjell fra tre ganger i den første versjonen.

Vi har lagt `while`-løkka slik at den kjøre så lenge forskjellen mellom det nye midpunktet og forige midpunkt er mindre enn en grense vi bestemmer selv (`while abs(x-xGammel)>Tol`). Vi kunne også valgt å lage `while`-løkka slik at absoluttveridien av funksjonen, $|f(x_n)|$, der x_n er midpunkt nummer n, skulle være mindre enn ei øvre grense.

Dette siste skriptet har jeg gitt navnet `HalveringsMetoden.m`. Vi kjører det i Octave og krever at løsningen skal ha en nøyaktighet på 10^{-5}:

```
octave-3.2.4.exe:18> HalveringsMetoden
Venstre grense: 0
Hoeyre grense: pi/2
Noeyaktighet: 1e-5
Nullpunkt =  0.71353
Iterasjoner =  17
```

Vi ser at svaret ble funnet etter 17 iterasjoner. (Vi har krevd ganske høy nøyaktighet – 10^{-5}.) Vi kan godt sjekke at $f(x)$ faktisk er nær 0 for denne løsninga:

```
octave-3.2.4.exe:19> FunkObl1Oppg3(Nullpunkt)
ans = -2.7482e-007
```

Altså er $f(x) \approx -2.75 \cdot 10^{-7} = -0.000000275$ for den løsninga vi har kommet fram til. Det må vel kunne sies å være godkjent.

Om vi vil, kan vi også se hvordan `x` nærmer seg løsninga – hvordan `x` *konvergerer* – ved å skrive ut `x` for hver iterasjon. Det gjør vi ved å fjerne semikolonet i linje 34. Vi får da:

```
octave-3.2.4.exe:13> HalveringsMetoden
Venstre grense: 0
Hoeyre grense: pi/2
Noeyaktighet: 1e-4
x =  0.78540
x =  0.39270
x =  0.58905
x =  0.68722
x =  0.73631
x =  0.71177
x =  0.72404
```

```
x =  0.71790
x =  0.71484
x =  0.71330
x =  0.71407
x =  0.71368
x =  0.71349
x =  0.71359
Nullpunkt =  0.71359
Iterasjoner =  14
```

d) Det kan synes noe unødvendig å løse likninga numerisk når vi allerede har eksakt løsning. Men det er en god vane å kontrollere skript og programmer ved å sjekke at de faktisk gir lønsninger vi vet er riktige. Likninga i 1 d) kan skrives som

$$\frac{e^x}{1+4e^{-x}} - 2 = 0$$

Vi lager en funksjonsfil for funksjonen på venstre side og gir den navnet `FunkOppg1d.m`:

```
function F=FunkOppg1d(x)

% Funksjonen f(x)=e^x/(1+4e^(-x)) - 2.
% Funksjonen tar skalare og vektorielle argumenter

F=exp(x)./(1+4*exp(-x))-2;
```

Vi trenger å forandre rad 16 i `HalveringsMetoden.m` til `Funk=@FunkOppg1d`. Vi kjører skriptet `Halveringsmetoden` i Octave:

```
octave-3.2.4.exe:7> HalveringsMetoden
octave-3.2.4.exe:41> HalveringsMetoden
Venstre grense: 1
Hoeyre grense: 2
Noeyaktighet: 1e-7
Nullpunkt =  1.3863
Iterasjoner =  24
octave-3.2.4.exe:42> Nullpunkt-log(4)
ans = -5.5795e-008
```

Her har vi også regnet ut differansen mellom den numeriske løsningen (`Nullpunkt`) og den eksakte løsningen (`log(4)`). Den er altså $-5.58 \cdot 10^{-8}$, noe som tross alt er ganske lite – og mindre enn den nøyaktigheten vi krevde (10^{-7}). Om vi vil ha et enda mer nøyaktig svar, kan vi velge en lavere '`Noeyaktiget`' enn det som vi har valgt her.