Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 7

Løsningsforslag

Oppgave 1 – Numerisk derivasjon

a) Vi kan for eksempel velge denne funksjonen:

$$f(x) = \sin x^2 \quad .$$

Vi bruker kjerneregelen når vi deriverer:

$$f'(x) = \cos x^2 \cdot (x^2)' = 2x \cos x^2 \quad .$$

Jeg velger $a = 1$ og får at $f'(a) = f'(1) = 2\cos 1$.

b) Det vil være nyttig å lage ei funksjonsfil for funksjonen vår:

```
function F=FunkTilOppg1(x)

% Funksjonen f(x) = sin x^2. Funksjonen tar vektor-argumenter.

F=sin(x.^2);
```

Jeg har valgt å kalle denne `FunkTilOppg1.m`. Vi regner ut gjennomsnittlig vekstrate for noen verdier av h:

```
>> format compact
>> h=1;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate = -1.5983
>> h=.5;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate = -0.12680
>> h=.1;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  0.94145
>> h=.05;
```

```
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  1.0174
>> h=1e-2;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  1.0689
>> h=1e-3;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  1.0795
>> h=1e-4;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  1.0805
```

Her kan det være verd å minne om at piltastene er svært nyttige når man skal skrive den samme (lange) kommandoen flere ganger – slik som her. I følge MATLAB er $f'(1) = 2 \cdot \cos 1 \approx 1.0806$, så vi begynner å nærme oss.

c) Vi gjentar det vi nettop gjore med "bakover-formelen":

```
>> h=1;
>> Rate=(FunkTilOppg1(1)-FunkTilOppg1(1-h))/h
Rate =  0.84147
>> h=.5;
>> Rate=(FunkTilOppg1(1)-FunkTilOppg1(1-h))/h
Rate =  1.1881
>> h=.1;
>> Rate=(FunkTilOppg1(1)-FunkTilOppg1(1-h))/h
Rate =  1.1718
>> h=.05;
>> Rate=(FunkTilOppg1(1)-FunkTilOppg1(1-h))/h
Rate =  1.1318
>> h=1e-2;
>> Rate=(FunkTilOppg1(1)-FunkTilOppg1(1-h))/h
Rate =  1.0918
>> h=1e-3;
>> Rate=(FunkTilOppg1(1)-FunkTilOppg1(1-h))/h
Rate =  1.0817
>> h=1e-4;
>> Rate=(FunkTilOppg1(1)-FunkTilOppg1(1-h))/h
Rate =  1.0807
```

Nok en gang ser vi at `Rate` nærmer seg $f'(1)$ når h blir liten.

d) Gjennomsnittet blir

$$\frac{1}{2}\left(\frac{f(a+h)-f(a)}{h}+\frac{f(a)-f(a-h)}{h}\right)=$$
$$\frac{f(a+h)-f(a)+f(a)-f(a-h)}{2h}=\frac{f(a+h)-f(a-h)}{2h}$$

Altså:

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}$$

når h er passe liten[1]. Legg merke til at dette $f'(a)$-estimatet er helt uavhengig av $f(a)$.

e) Vi gjentar prosedyren fra c) og d) med denne nye formelen:

```
>> h=1;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1-h))/2/h
Rate = -0.37840
>> h=.5;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1-h))/2/h
Rate = f 0.53067
>> h=.1;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1-h))/2/h
Rate =  1.0566
>> h=.05;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1-h))/2/h
Rate =  1.0746
>> h=1e-2;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1-h))/2/h
Rate =  1.0804
>> h=1e-3;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1-h))/2/h
Rate =  1.0806
```

Det ser ut til at vi nærmer oss riktig verdi for $f'(a)$ raskere på denne måten; for $h = 10^{-3}$ ser vi faktisk ikke forskjell med det antallet desimaler som blir skrevet til skjerm[2].

f) Maskinpresisjonen finner vi slik:

```
>> eps
ans = 2.2204e-016
```

–altså $2.22 \cdot 10^{-16}$. Dette er jo et veldig lite tall, men det er ikke null. Vi velger noen veldig små verdier for h og beregner den deriverte med "framover-formelen":

```
>> DerivFasit=2*cos(1)
DerivFasit =  1.0806
>> h=1e-13;
```

[1] "Passe liten" høres unektelig noe upresist ut. Det er mulig å sette opp rimelig presise kriterier for hvor liten h må være for at estimatet vårt skal ha for å ha en bestemt nøyaktighet. Taylors formel med restledd er svært nyttig i denne sammenhengen.

[2] MATLAB opererer som tideligere nevnt med langt flere desimaler egentlig; om du vil ha flere desimaler skrevet til skjerm, kan du skrive '`format long`'

```
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  1.0802
>> h=1e-14;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  1.0769
>> h=1e-15;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  1.2212
>> h=5e-16;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate =  0.88818
>> h=1e-16
h = 1.0000e-016
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate = 0
>> h=5e-17;
>> Rate=(FunkTilOppg1(1+h)-FunkTilOppg1(1))/h
Rate = 0
```

Nå ser vi faktisk at jo mindre h blir, jo *dårligere* estimat får vi for $f'(a)$. Her ser vi en vesensforskjell mellom numeriske metoder og analytisk matematikk; matematisk skal $(f(x+h)-f(x))/h$ vilkårsløst nærme seg $f'(a)$ når h nærmer seg 0, mens *numerisk* sett blir den deriverte best estimert med en liten – men *endelig* – h. Dette skyldes at datamaskiner har en nedre grense for hvor små tall den klarer å håndtere.

g) Vi kan lage en h-vektor som blir mindre og mindre ved å starte på 1 og så dele på to – igjen og igjen. Dette kan vi gjøre slik i MATLAB:

```
>> hVekt=[1 .5 .5^2 .5^3 .5^4 .5^5]
hVekt =
   1.000000   0.500000   0.250000   0.125000   0.062500   0.031250
```

Dette kunne også blitt gjort litt mer effektivt:

```
>> n=0:5;
>> hVekt=.5.^n;
```

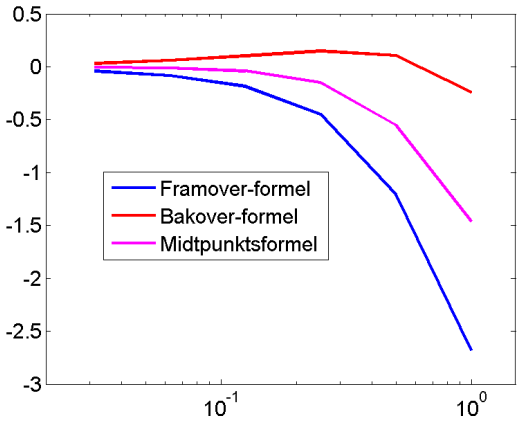
Vi lager vektorer som beregner $f'(a)$ ut fra de tre metodene over (husk punktum når vi deler):

```
>> DerivFram=(FunkTilOppg1(1+hVekt)-FunkTilOppg1(1))./hVekt;
>> DerivBak=(FunkTilOppg1(1)-FunkTilOppg1(1-hVekt))./hVekt;
>> DerivMidt=(FunkTilOppg1(1+hVekt)-FunkTilOppg1(1-hVekt))./(2*hVekt);
```

Vi plotter feilen i estimatet ved hjelp av `semilogx`-kommandoen:

```
>> DerivFasit=2*cos(1)
DerivFasit =  1.0806
>> semilogx(hVekt,DerivFram-DerivFasit,'b','linewidth',2)
>> hold on
>> semilogx(hVekt,DerivBak-DerivFasit,'r','linewidth',2)
>> semilogx(hVekt,DerivMidt-DerivFasit,'m','linewidth',2)
>> hold off
>> legend('Framover-formel','Bakover-formel','Midtpunktsformel')
>> axis([.02 1.5 -3 .5])
```

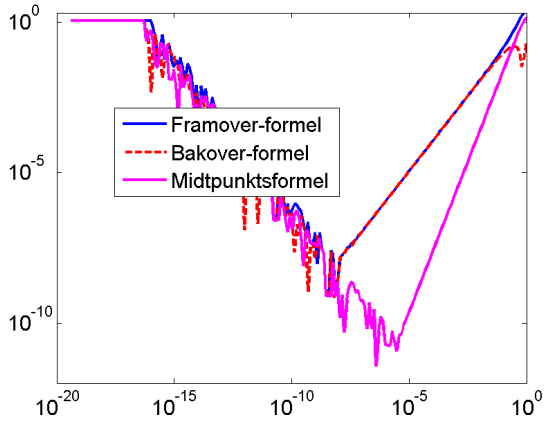
Langs y-aksen har vi her differansen mellom derivasjons-estimatet og den riktige verdien for den deriverte. Vi ser i figur 1 at alle tre formlene gir et estimat med en feil som ser ut til å gå mot 0 når h blir mindre og mindre. Men plottet for midtpunktsformelen går klart mot null raskest. (Her er det nok enklest å lese plottet fra høgre mot venstre.)



Figur 1: Plott som viser feilen i ulike estimat for $f'(a)$ som funksjon av h. Plottet er *semi-logaritmisk*. Den blå kurva tilsvarer "framover-formelen", den røde tilsvarer "bakover-formelen" og den lilla er "midtpunktsformelen".

I figur 2 har vi plotta absoluttverdien av feilen i de tre ulike måtene å estimere $f'(1)$ på for et veldig stort område av h-verdier. Her er både x- og y-aksen logaritmiske (I MATLAB: '`» loglog(x,y)`'). Vi ser feilen først avtar når h avtar – og at den begynner å vokse igjen når h blir ekstremt liten. Vi ser også at feilen avtar raskere for midpunktsformelen enn for framover- og bakoverformelen – og at feilen når en lavere verdi med midtpunktsformelen enn med de andre formlene. Legg også merke til at grafen begynner å bli "hakkete" når den numeriske feilen til datamaskina begynner å spille en rolle.

h) Vi kan for eksempel velge $g(x) = 2x^2 + x - 1$. Vi lager ei funksjonfil for denne:



Figur 2: Feilen i de ulike estimatene for $f'(a)$ for h-verdier fra 1 ned mot 10^{-20}.

```
function G=AndreGradPol(x)

% AndreGradPol(x) gir funksjonen
% g(x) = 2x^2+x-1.
% Funksjonen tar både skalar- og
% vektorargumenter.

G=2*x.^2+x-1;
```

Vi regner noen deriverte:

$$\begin{aligned} g'(x) &= 4x+1 \\ g'(-1) &= -4+1=-3 \\ g'(0) &= 1 \\ g'(2) &= 4\cdot 2+1=9 \quad . \end{aligned}$$

Vi estimerer disse ulike deriverte i MATLAB:

```
>> a=-1;
>> h=.5;
>> gDeriv=(AndreGradPol(a+h)-AndreGradPol(a-h))/(2*h)
gDeriv = -3
>> h=.1;
>> gDeriv=(AndreGradPol(a+h)-AndreGradPol(a-h))/(2*h)
gDeriv = -3.0000
>> a=0;
>> gDeriv=(AndreGradPol(a+h)-AndreGradPol(a-h))/(2*h)
gDeriv =  1.0000
>> h=2;
>> gDeriv=(AndreGradPol(a+h)-AndreGradPol(a-h))/(2*h)
```

```
gDeriv =  1
>> a=2;
>> h=.5;
>> gDeriv=(AndreGradPol(a+h)-AndreGradPol(a-h))/(2*h)
gDeriv =  9
>> h=500;
>> gDeriv=(AndreGradPol(a+h)-AndreGradPol(a-h))/(2*h)
gDeriv =  9
```

Vi ser ut til å få helt nøyaktige svar hver gang – uavhengig av vilken verdi vi velger for h. Om vi bruker midtpunktsformelen for å finne den deriverte for en vilkårlig x, ser vi at vi faktisk får $g'(x)$ eksakt:

$$\frac{g(x+h)-g(x-h)}{2h} =$$
$$\frac{2(x+h)^2+(x+h)-1-\big(2(x-h)^2+(x-h)-1\big)}{2h} =$$
$$\frac{2(x^2+2hx+h^2)+x+h-1-2(x^2-2hx+h^2-x+h-1)}{2h} =$$
$$\frac{2x^2-2x^2+4hx+4hx+x-x+h+h-1+1}{2h} = \frac{8hx+2h}{2h} =$$
$$4x+1 = g'(x) \quad .$$

Det er ikke så vanskelig å vise at det samme gjelder for et generelt andregradspolynom. Altså: Midtpunktsformelen er faktisk *eksakt* for andregradspolynom.

For hvilken type funksjoner er framover- eller bakoverformelen eksakt?

Oppgave 2 – Den deriverte fra en tabell

a) Plottet kan lages slik:

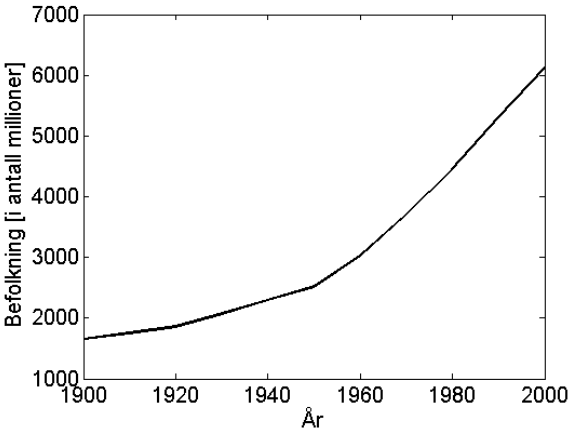
```
>> Aar=1900:10:2000;
>> Bef=[1650 1750 1860 2070 2300 2520 3020 3700 4450 5300 6123];
>> plot(Aar,Bef,'k','linewidth',2)
>> set(gca,'fontsize',15)
>> xlabel('År')
>> ylabel('Befolkning [i antall millioner]')
```

og resultatet kan sees i figur 3.

b) Vi lar $B(t)$ være antall millioner mennesker i verden i år t. Om vi bruker formelen fra 2 c) kan vi for eksempel estimere B(1910) slik

$$B'(1910) \approx \frac{B(1920)-B(1900)}{1920-10910} = \frac{B(1910+10)-B(1910-10)}{2\cdot 10} \quad .$$

Vi regner dette ut i kommandovinduet:



Figur 3: Verdens befolkning i antall millioner som funksjon av årstall.

```
>> (Bef(3)-Bef(1))/(Aar(3)-Aar(1))
ans =  10.500
```

eller, for å gjøre sammenghengen med midtpunktsformelen litt tydeligere:

```
>> h=Aar(2)-Aar(1);
>> (Bef(3)-Bef(1))/(2*h)
ans =  10.500
```

Altså, ved inngangen til år 1910 vokste jordas befolkning med ca 10.5 millioner mennesker per år. For de andre årene får vi $B'(1920) \approx 16$, $B'(1930) \approx 22, B'(1940) \approx 22.5, B'(1950) \approx 36, B'(1960) \approx 59, B'(1970) \approx 71.5$, $B'(1980) \approx 80$ og $B'(1990) \approx 83.65$.

c) Siden vi ikke har $B(1890)$, kan vi ikke bruke midpunktsformelen for å estimere $B'(1900)$. På samme måte er vi ute at stand til bruke midpunktsformelen for å finne $B'(2000)$ siden vi ikke har fått oppgitt $B(2010)$. Derimot kan vi bruke "framover"- og "bakover"-formlene selv om de ikke er like nøyaktige:

$$\begin{aligned} B'(1900) &\approx \frac{B(1910) - B(1900)}{1910 - 1900} = 10 \\ B'(2000) &\approx \frac{B(2000) - B(1990)}{2000 - 1990} = 82.3 \end{aligned}$$

d) Alle de estimatene vi har regna ut her, kan gjøres i "ett jafs" med dette skriptet:

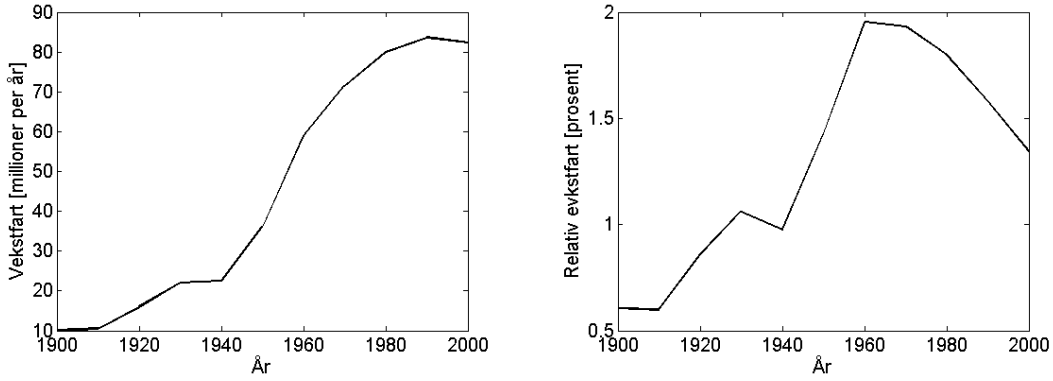
```
1 % Vektor med årstall:
2 Aar=1900:10:2000;
```

```
% Vektor med befolkning:
Bef=[1650 1750 1860 2070 2300 2520 3020 3700 4445 5300 6123];
% Bestemmer lengda av Aar og tilordner dette tallet til N:
N=length(Aar);

for n=1:N
  if n==1                                              % For det første punktet
    Bderiv(n)=(Bef(n+1)-Bef(n))/(Aar(n+1)-Aar(n));  % Framover-formel
  elseif n==N                                          % For det siste punktet
    Bderiv(n)=(Bef(n)-Bef(n-1))/(Aar(n)-Aar(n-1));  % Bakover-formel
  else                                                 % For alle andre punkt
    Bderiv(n)=(Bef(n+1)-Bef(n-1))/(Aar(n+1)-Aar(n-1));  % Midtpunktsformel
  end
end
% Skriver estimatene for de deriverte til skjerm:
Bderiv

% Plotter vekstfarten
figure(1)
plot(Aar,Bderiv,'k-','linewidth',2)
set(gca,'fontsize',15)
xlabel('År')
ylabel('Vekstfart [millioner per år]')

% Plotter den relative vekstfarten
figure(2)
plot(Aar,Bderiv./Bef*100,'k-','linewidth',2)
set(gca,'fontsize',15)
xlabel('År')
ylabel('Relativ evkstfart [prosent]')
```

Vi har kalt skriptet `BefDeriv.m`. Vi kjører det i MATLAB:

```
>> BefDeriv
Bderiv =

 Columns 1 through 8:

   10.000   10.500   16.000   22.000   22.500   36.000   59.000   71.500

 Columns 9 through 11:

   80.000   83.650   82.300
```

Vi får også opp plottene vist i figur 4. Som sagt, kunne dette også vært gjort uten å bruke noen `if`-sats. Linjene fra og med nummer 8 til og med 16 kan erstattes med følgende:



Figur 4: Tilnærma årlig tilvekst i verdens befolkning som funksjon av årstall. Figure til venstre gir den relative tilveksten, altså tilvekst i forhold til befolkning, i prosent.

```
% For-løkke for alle indre punkt
for n=2:(N-1)
  Bderiv(n)=(Bef(n+1)-Bef(n-1))/(Aar(n+1)-Aar(n-1));  % Midtpunktsformel
end
% Endepunktene:
Bderiv(1)=(Bef(2)-Bef(1))/(Aar(2)-Aar(1);
Bderiv(N)=(Bef(N)-Bef(N-1))/(Aar(N)-Aar(N-1));
```

Denne varianten er langt ryddigere siden vi ikke trenger bruke noen `if`-satser.

Oppgave 3 – Mange rektangler (og noen trapeser)

$$V_n = \sum_{k=0}^{n-1} \Delta x_n f(x_k), \quad \Delta x_n = (b-a)/n, \quad x_k = a + (k-1)\Delta x_n \quad .$$

a) Det grønne området i figuren til venstre er summen av arealet av fire rektangler – alle med bredde Δx_4:

$$A = \Delta x_4 f(1) + x_4 f(1.5) + x_4 f(2) + x_4 f(2.5) \quad .$$

Vi ser at høgda av hvert rektangel konsekvent er gitt ved funksjonsverdien i *venstre* kant. Dette kjenner vi igjen som summen V_4 med $a = 1$ og $b = 3$; $\Delta x_4 = (3-1)/2 = 1/2$, $x_0 = a + 0\Delta x_4 = a = 1$, $x_1 = 1 + 1/2 = 1.5$, $x_2 = 2$ og $x_3 = 2.5$. V_4 er altså arealet av det grønne området i figuren. Vi lager et skript som regner ut summen V_n og lar n være input.

```
1  % Skript som regner ut en sum av areal av rektangler
2
3  % Antall rektangler
4  n=input('Hvor mange rektangler? ');
5
6  % Grenser
7  a=1;
8  b=3;
9
10 % Bestemmer Delta x og intierer summen V
11 DeltaX=(b-a)/n;
12 V=0;
13
14 for i=0:(n-1)
15     xi=a+i*DeltaX;
16     fi=xi^3;
17     V=V+DeltaX*fi;
18 end
19 % Skriver summen V til skjerm
20 V
```

Vi kjører skriptet:

```
>> RektangelSum
Hvor mange rektangler? 4
V =
    14
```

Dette kunne vi selvsagt også godt ha gjort ved å lage ei funksjonsfil som tar n som argument. Vi kunne selvsagt også ha "hardkoda" `n`; ("`n=4;`" i linje 4).

Om vi øker n, vil vi få flere og smalere rektangler. Jo flere, jo nærmere ligger toppen av rektanglene grafen til f. I grensa $n \to \infty$ vil dette arealet bli identisk med arealet mellom grafen til $f(x)$ og x-aksen – altså integralet $\int_a^b f(x)\,dx$. I vårt tilfelle er dette arealet

$$\int_1^3 x^3\,dx = \frac{1}{4}\left[x^4\right]_1^3 = \frac{3^4-1}{4} = 20 \quad .$$

Vi sjekker om skriptet vårt ser ut til å gi at $\lim\limits_{n\to\infty} V_n = 20$:

```
>> RektangelSum
Hvor mange rektangler? 10
V =
   17.4800
>> RektangelSum
Hvor mange rektangler? 50
```

```
V =
   19.4832
>> RektangelSum
Hvor mange rektangler? 100
V =
   19.7408
>> RektangelSum
Hvor mange rektangler? 200
V =
   19.8702
>> RektangelSum
Hvor mange rektangler? 1000
V =
   19.9740
```

Mye tyder på at vi nærmer oss 20...

b)

$$H_n = \sum_{k=1}^{n} \Delta x_n f(x_k) \quad .$$

Denne summen skiller seg fra V_n ved at summasjonsgrensene er justert; vi starter nå i $k = 1$ og ender med $k = n$. Dette tilsvarer, som indikert i figuren, at rektanglene har høgder gitt ved funksjonsverdien i *høgre* kant i stedet for venstre kant. Dette kan vi ta høgde for ved å justere linje 14 i skriptet til "`for i=1:n`". Samtidig er det naturlig å endre navn på summen fra "`V`" til "`H`". Som før bør summen gå mot arealet under grafen når n blir stor. Vi sjekker:

```
>> RektangelSum
Hvor mange rektangler? 4
H =
    27
>> RektangelSum
Hvor mange rektangler? 10
H =
   22.6800
>> RektangelSum
Hvor mange rektangler? 50
H =
   20.5232
>> RektangelSum
Hvor mange rektangler? 100
H =
   20.2608
>> RektangelSum
Hvor mange rektangler? 1000
H =
```

```
20.0260
```

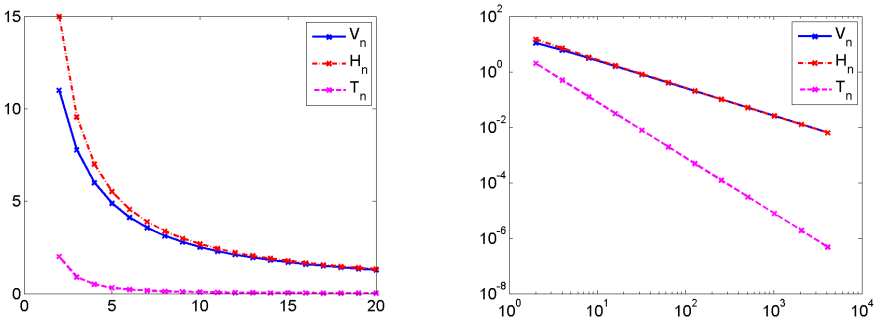
Vi ser ut til å nærme oss den samme grensa – fra oversida denne gangen.

c) Vi justerer skriptet slik at det nå regner ut både V_n, H_n og T_n og skriver alle tre til skjerm:

```
% Skript som regner ut en sum av areal av rektangler

% Antall rektangler
n=input('Hvor mange rektangler? ');

% Grenser
a=1;
b=3;

% Bestemmer Delta x og intierer summene V og H
DeltaX=(b-a)/n;
V=0;
H=0;

% for-løkke for V
for i=0:(n-1)
    xi=a+i*DeltaX;
    fi=xi^3;
    V=V+DeltaX*fi;
end

% for-løkke for H
for i=1:n
    xi=a+i*DeltaX;
    fi=xi^3;
    H=H+DeltaX*fi;
end

% Skriver summene V, H og T til skjerm
V
H
T=(V+H)/2
```

Det hadde ikke vært nødvendig å bruke to separate `for`-løkker her. Vi tester:

```
>> RektangelSum
Hvor mange rektangler? 4
V =
    14
H =
```



Figur 5: Plott av $|V_n - L|$, $|H_n - L|$ og $|T_n - L|$ som funksjoner av n. I plottet til høgre har vi brukt logaritmiske akser.

```
    27
T =
   20.5000
>> RektangelSum
Hvor mange rektangler? 10
V =
   17.4800
H =
   22.6800
T =
   20.0800
>> RektangelSum
Hvor mange rektangler? 100
V =
   19.7408
H =
   20.2608
T =
   20.0008
```

Vi ser at `T` konsekvent ligger mye nærmere integralet (20) enn det `V` og `H` gjør. Dette inntrykket blir ytterligere forsterket av det vi ser i figur 5, som viser avstanden mellom de respektive summene og integralet for ulike verdier av n.

La oss sette litt "navn på dyra": Både V_n og H_n er eksempler på *Riemann-summer*. Vi har delt intervallet $[a, b]$ opp i n like store deler (en *regulær partisjon*) og konsekvent valgt venstre ende i hvert delintervall for V_n og høgre ende for H_n. Når vi tar gjennomsnittet av V_n og H_n får vi, som vi har sett, en ganske god måte å estimere integraler på. Denne metoden kalles *trapesmetoden*.

Alt dette vil dere høre mer om senere.