Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 8

Løsningsforslag

Oppgave 1 – Mange rektangler (og noen trapeser)

$$V_n = \sum_{i=0}^{n-1} hf(x_i) \quad \text{med} \quad h = (b-a)/n \quad \text{og} \quad x_i = a + ih \quad .$$

a) Det grønne området i figuren til venstre er summen av arealet av fire rektangler – alle med bredde 1/2:

$$A = hf(1) + hf(1.5) + hf(2) + hf(2.5) \quad .$$

Vi ser at høgda av hvert rektangel konsekvent er gitt ved funksjonsverdien i *venstre* kant. Dette kjenner vi igjen som summen V_4 med $a = 1$, $b = 3$ og $h = (3-1)/4 = 1/2$. $x_0 = a + 0 \cdot h = a = 1$, $x_1 = 1 + 1/2 = 1.5$, $x_2 = 2$ 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.

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

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

% Grenser
a=1;
b=3;

% Bestemmer h og intierer summen V
h=(b-a)/n;
V=0;

for i=0:(n-1)
    xi=a+i*h;
    fi=xi^3;
```

```
17      V=V+h*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 $\lim_{n\to\infty} V_n = 20$:

```
>> 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)

$$H_n = h \sum_{i=1}^{n} f(x_i) \quad .$$

Denne summen skiller seg fra V_n ved at summasjonsgrensene er justert; vi starter nå i $i = 1$ og ender med $i = n$. 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.

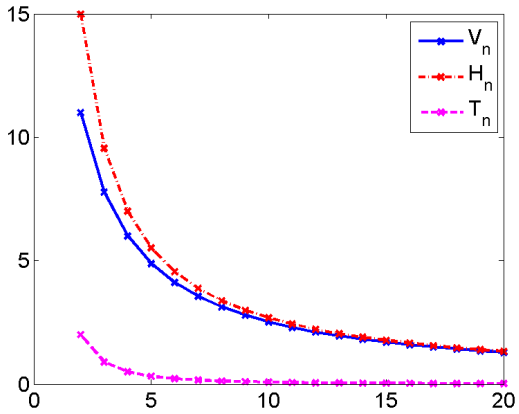
c) Vi justerer skriptet slik at det nå regner ut både V_n, H_n og T_n og skriver alle tre til skjerm:

```
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 summene V og H
```

```
11 h=(b-a)/n;
12 V=0;
13 H=0;
14
15 % for-løkke for V
16 for i=0:(n-1)
17     xi=a+i*h;
18     fi=xi^3;
19     V=V+h*fi;
20 end
21
22 % for-løkke for H
23 for i=1:n
24     xi=a+i*DeltaX;
25     fi=xi^3;
26     H=H+h*fi;
27 end
28
29 % Skriver summene V, H og T til skjerm
30 V
31 H
32 T=(V+H)/2
```

Det hadde også gått fint å implementere dette uten å bruke to separate `for`-løkker. Vi tester skriptet:

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



Figur 1: Plott av $|V_n - L|$, $|H_n - L|$ og $|T_n - L|$, der $L = \int_1^3 x^3\,dx = 20$, som funksjoner av n.

```
   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 1, som viser avstanden mellom de respektive summene og integralet for ulike verdier av n.

d)

$$
\begin{aligned}
T_n = \frac{1}{2}\left(V_n + H_n\right) = \frac{h}{2}\sum_{i=0}^{n-1} f(x_i) + \frac{h}{2}\sum_{i=1}^{n} f(x_i) = \\
\frac{h}{2}\left(f(x_0) + f(x_1) + f(x_2) + ... + f(x_{n-1}) + \right. \\
\left. f(x_1) + f(x_2) + ... + f(x_{n-1}) + f(x_n)\right) = \\
\frac{h}{2}\left(f(x_0) + f(x_1) + f(x_1) + f(x_2) + f(x_2) + ... \right. \\
\left. + f(x_{n-1}) + f(x_{n-1}) + f(x_n)\right) = \\
\frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + ... + 2f(x_{n-1}) + f(x_n)\right) = \\
h\left(\frac{1}{2}f(x_0) + f(x_1) + f(x_2) + ... + f(x_{n-1}) + \frac{1}{2}f(x_n)\right) \quad .
\end{aligned}
$$

I læreboka er det forklart hvordan dette tilsvarer en sum av arealene av ei rekke trapeser – stedet for rektangler, som tilfellet var med Riemann-summer. Om vi husker at formelen for et trapes med grunnlinjene a og b og høgda h er $h(a+b)/2$, kan vi også se det ved at T_n kan skrives slik:

$$
T_n = h\frac{f(x_0) + f(x_1)}{2} + h\frac{f(x_1) + f(x_2)}{2} + ... + h\frac{f(x_{n-1}) + f(x_n)}{2} \quad .
$$

Oppgave 2 – Med fasit

a)

$$\int_{-1}^{1} x^6\,dx = \frac{1}{7}\left[x^7\right]_{-1}^{1} = \underline{\frac{2}{7}}$$

Det andre integralet kan løses ved variabelbyttet $u = x^2$;

$$\int_{-1}^{3} x\sin x^2\,dx = \int_{u(-1)}^{u(3)} x\sin u\,\frac{dx}{2x} = \frac{1}{2}\int_{1}^{9}\sin u\,du = \frac{1}{2}\left[-\cos u\right]_1^9 = \underline{\frac{\cos 1 - \cos 9}{2}}$$

$$\int_2^4 \frac{x}{x-1}\,dx = \int_2^4 \frac{x-1+1}{x-1}\,dx = \int_2^4 \left(\frac{x-1}{x-1} + \frac{1}{x-1}\right)\,dx =$$

$$\int_2^4 \left(1 + \frac{1}{x-1}\right)\,dx = \left[x + \ln|x-1|\right]_2^4 = 4 + \ln 3 - (2 + \ln 1) = \underline{2 + \ln 3}$$

b) Med utgangspunkt i formelen fra oppgave 1 d) har vi laget et nytt skript som vi har kalt `Trapes.m`:

```
% Implementering av trapesmetoden for numerisk integrasjon.
% Integrasjonsgrensene a og b, oppdelinga n og integranden funk
% blir gitt heilt i toppen av skriptet.
% For å gi n, bruker vi input-funksjonen

% Integrasjonsgrenser
a=-1;
b=1;

% Integranden
funk=@(x) x^6;

% Oppdeling
n=input('Gi oppdelinga n: ');
h=(b-a)/n;                  % Skrittlengda

% Bidrag fra endene
T=h/2*(funk(a)+funk(b));

% Resten av bidragene
for i=1:(n-1)
    xi=a+i*h;
    T=T+h*funk(xi);
end

% Skriver svaret til skjerm
T
```

Vi kunne ha laga ei `for`-løkke som inkluderte `i=0` og `i=n` også, men da måtte vi hatt en liten `if`-sats inni løkka som spesifiserte at disse to bidra-

gene skulle ganges med 1/2. Det blir ryddigere å ta det siste og det første leddet for seg selv (linje 18).

Vi kjører skriptet for økende `n`-verdier og beregner feilen:

```
>> Int=2/7
Int =
    0.2857
>> Trapes
Gi oppdelinga n: 10
T =
    0.3252
>> Feil=abs(T-Int)
Feil =
    0.0395
>> Trapes
Gi oppdelinga n: 100
T =
    0.2861
>> Feil=abs(T-Int)
Feil =
   3.9995e-04
>> Trapes
Gi oppdelinga n: 1000
T =
    0.2857
>> Feil=abs(T-Int)
Feil =
   4.0000e-06
>> Trapes
Gi oppdelinga n: 500
T =
    0.2857
>> Feil=abs(T-Int)
Feil =
   1.6000e-05
>> Trapes
Gi oppdelinga n: 700
T =
    0.2857
>> Feil=abs(T-Int)
Feil =
   8.1632e-06
```

Det ser ut til at `n=700` er tilstrekkelig for å få en feil som er mindre enn 10^{-5}.

For å beregne det neste integralet oppdaterer vi øvre grense i linje 8 og integranden i linje 11:

```
funk=@(x) x*sin(x^2);
```

Vi tester med ulike `n`-verdier:

```
>> Int=(cos(1)-cos(9))/2
Int =
    0.7257
>> Trapes
Gi oppdelinga n: 100
T =
    0.7233
>> Feil=abs(T-Int)
Feil =
    0.0024
>> Trapes
Gi oppdelinga n: 1000
T =
    0.7257
>> Feil=abs(T-Int)
Feil =
   2.3881e-05
>> Trapes
Gi oppdelinga n: 1500
T =
    0.7257
>> Feil=abs(T-Int)
Feil =
   1.0614e-05
```

Vi ser at $n = 1500$ er nesten nok.

For det siste integralet endrer vi linje 6 til 11 til

```
% Integrasjonsgrenser
a=2;
b=4;

% Integranden
funk=@(x) x/(x-1);
```

Etter litt prøving og feiling finner vi at `n=175` er gir tilstrekelig nøyaktighet for dette integralet:

```
>> Int=2+log(3)
Int =
    3.0986

>> Trapes
```

```
Gi oppdelinga n: 175
T =
    3.0986
>> Feil=abs(T-Int)
Feil =
   9.6748e-06
```

Merk at Int og `T` ikke nødvendigvis er like selv om de fire første desimalene er like; MATLAB opererer med større nøyaktighet enn det som blir skrevet til skjerm.

c) Simpsons metode kan implementeres slik:

```
% Implementering av Simpsons metode.
% Integrasjonsgrensene a og b, oppdelinga n og integranden funk
% blir gitt heilt i toppen av skriptet.
% For å gi n, bruker vi input-funksjonen

% Integrasjonsgrenser
a=-1;
b=1;

% Integranden
funk=@(x) x^6;

% Oppdeling (kontrollerer at n er et partall)
n=input('Gi oppdelinga n: ');
if round(n/2) ~= n/2
    disp('n må være et partall')
    return
end
h=(b-a)/n;                  % Skrittlengda

% Bidrag fra endene
S=h/3*(funk(a)+funk(b));

% Oddetallsbidrag:
for i=1:2:(n-1);
    xi=a+i*h;
    S=S+h/3*4*funk(xi);
end

% Partallsbidrag
for i=2:2:(n-2)
    xi=a+i*h;
    S=S+h/3*2*funk(xi);
end

```

```
36 % Skriver svaret til skjerm
37 S
```

I dette skriptet, som vi har kalt `Simpson.m`, har vi laget separate `for`-løkker for de bidragene som skal ganges med 4 og de som skal ganges med 2. Vi har også lagt inn en liten `if`-sats som sjekker om `n` er et partal og avslutter skriptet dersom det ikke er det (linje 15-18).

Vi gjentar det vi gjorde i deloppgave b) med dette skriptet:

```
>> Int=2/7
Int =
    0.2857
>> Simpson
Gi oppdelinga n: 10
S =
    0.2878
>> Feil=abs(S-Int)
Feil =
    0.0021
>> Simpson
Gi oppdelinga n: 30
S =
    0.2857
>> Feil=abs(S-Int)
Feil =
   2.6254e-05
>> Simpson
Gi oppdelinga n: 40
S =
    0.2857
>> Feil=abs(S-Int)
Feil =
   8.3185e-06
```

Det var altså nok med `n=40` – mot 700 for trapesmetoden. For de to andre integralene gjør vi tilsvarende endringer i skriptet som i deloppgave b) og finner at vi trenger `n=100` for det andre integralet og `n=16` for det tredje – mot henholdsvis 1500 og 175 for trapesmetoden.

Oppgave 3 – Uten fasit

$$\int_{-1}^{1} \sin x^2 \, dx \, , \quad \int_{0}^{2} \sqrt{x} e^{-x} \, dx \, , \quad \int_{0}^{3} (\arctan x)^2 \, dx$$

a) Vi oppdaterer linje 6-11 i begge skriptene:

```
% Integrasjonsgrenser
a=-1;
b=1;

% Integranden
funk=@(x) sin(x^2);
```

Når vi skal undersøke nøyaktigheten, kan det være greit å se litt flere enn fire desimaler. Vi får opp alle om vi skriver `» format long`.

```
>> format long
>> Trapes
Gi oppdelinga n: 200
T =
   0.620554613924188
>> Trapes
Gi oppdelinga n: 300
T =
   0.620544608004491
>> Trapes
Gi oppdelinga n: 400
T =
   0.620541105991013
```

Med fire desimaler rett får vi at $\int_{-1}^{1} \sin x^2 \, dx = \underline{0.6205}$. Som ventelig får vi rett svar litt raskere med Simpsons metode:

```
>> Simpson
Gi oppdelinga n: 10
S =
   0.620273694885053
>> Simpson
Gi oppdelinga n: 20
S =
   0.620520468866442
>> Simpson
Gi oppdelinga n: 30
S =
   0.620533428546312
>> Simpson
Gi oppdelinga n: 40
S =
   0.620535600264149
```

Når vi gjentar samme prosedyre for de to andre integralene, finner vi at

$$\int_0^2 \sqrt{x}e^{-x}\,dx = \underline{0.6545} \tag{1}$$

$$\int_0^3 (\arctan x)^2\,dx = \underline{2.6050} \tag{2}$$

Den viktigste moralen i denne oppgava er at man kan alltids bestemme et integral så nøyaktig man vil selv om vi ikke klarer å anti-derivere integranden. En annen moral er at Simpsons metode er bedre enn trapesmetoden. Den siste moralen er at det kan være lurt å løse et problem på to ulike måter. Om man får samme svar, er det et godt tegn.

b) Vi velger oss like godt integralet fra oppgave 1: $\int_1^3 x^3\,dx = 20$. Så estimerer vi integralet med Simpsons metode med $n = 2$; vi setter line 6-11 til

```
% Integrasjonsgrenser
a=1;
b=3;

% Integranden
funk=@(x) x^3;
```

I kommandovinduet:

```
>> Simpson
Gi oppdelinga n: 2
S =
    20
>> S-20
ans =
     0
```

I følge MATLAB har Simpsons metode gitt eksakt rett svar. Faktisk ville dette ha skjedd for et hvilket som helst integral der integranden var et polynom av grad tre eller lavere. Lurer du på hvorfor? Ta en kikk på side 276 i læreboka.

Trapesmetoden gir eksakte svar for førstegradspolynom (jmf. svaret på oppgave 1 d)).

Ekstraoppgave 1: Plott feilen

Skriptet kan for eksempel se slik ut:

```
1 % Skript som plotter feilen for tre ulike metoder å estimere
2 % et integral på:
```

```
3  % Venstre Riemann-sum, trapesmeotden og Simpsons metode.
4
5  % Integrasjonsgrenser
6  a=-1;
7  b=3;
8
9  % Integranden
10 funk=@(x) x.*sin(x.^2);
11
12 % Kjent analytisk svar:
13 Fasit=(cos(1)-cos(9))/2;
14
15 % Bestemmer minimal og maksimal oppdeling n
16 nMin=2;        % Må være partall
17 nSteg=2;       % Må også være partall
18 nMax=40;
19
20 % Vektorer med n-verdier og h-verdier
21 nVektor=nMin:2:nMax;
22 hVektor=(b-a)./nVektor;
23
24 % Initierer indeksering
25 indeks=1;
26
27 % Regner ut de tre estimatene for alle verdier av h
28 for h=hVektor
29     xVektor=a:h:b;
30     yVektor=funk(xVektor);
31     V(indeks)=RiemannFunk(xVektor,yVektor);
32     T(indeks)=TrapesFunk(xVektor,yVektor);
33     S(indeks)=SimpsonFunk(xVektor,yVektor);
34     indeks=indeks+1;
35 end
36
37 % Plotter feil som funksjon av n:
38 plot(nVektor,abs(V-Fasit),'b-','linewidth',2)
39 hold on
40 plot(nVektor,abs(T-Fasit),'r--','linewidth',2)
41 plot(nVektor,abs(S-Fasit),'k-.','linewidth',2)
42 hold off
43 legend('Riemann','Trapes','Simpson')
44 set(gca,'fontsize',15)
```

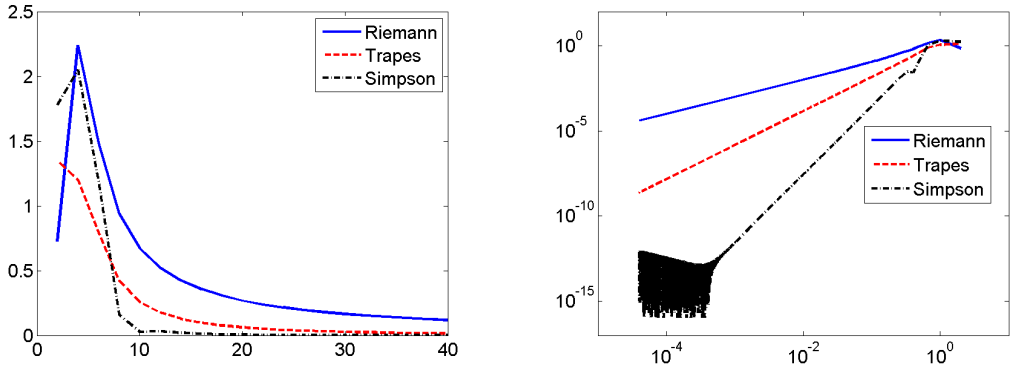
I linje 5-10 ser vi at vi har valgt å bruke det andre integralet fra oppgave 2. I linje 15-18 bestemmer vi hvilke n verdier vi skal se på – i dette tilfellet fra og med 2 til og med 40 i steg på 2. I `for`-løkka fra og med linje 28 regner vi ut de tre integral-estimatene. Dette gjør vi ved å kalle funksjoner som regner ut hver

og en av disse estimatene. Disse funksjonene kunne vi ha laga ved å gjøre om skriptene fra oppgave 2 og 3 til funksjonsfiler. Men her har vi valgt å gjøre det litt annerledes. For hver n har vi valgt å lage en vektor med de aktuelle x- og y-verdiene. Disse to vektorene alene er nok for å bestemme de tre estimatene. De tre ulike funksjonsfilene har nokså like strukturer. For Riemann-summen har vi valgt å regne ut en venstre-sum:

```
function V=RiemannFunk(x,y)

% Funksjon som beregner venstre Rimann sum for gitte x- og y-vektorer
% Partisjonen, gitt ved x-vektoren, må være regulær.

% Bestemmer steglengda og antall steg
h=x(2)-x(1);
n=length(x);

% Initierer summen
V=0;

for i=1:(n-1)
    V=V+h*y(i);
end
```

Trapesmetoden kan implementeres slik:

```
function T=TrapesFunk(x,y)

% Funksjon som beregner integral ved trapesmetoden
% for gitte x- og y-vektorer
% Partisjonen, gitt ved x-vektoren, må være regulær.

% Bestemmer steglengda og antall steg
h=x(2)-x(1);
n=length(x);

% Ende-bidrag:
T=h/2*(y(1)+y(n));

% Resten av bidragene
for i=2:(n-1)
    T=T+h*y(i);
end
```



Figur 2: Figurene viser feilen for dei tre estimatene: Venstre Riemann-sum, trapesmetoden og Simpsons metode. I plottet til venstre er feilen vist som funksjon av antall intervall n i oppdelinga. I figuren til høgre har vi plotta feilen som funksjon av steglengda h med logaritmiske akser.

Og – til sist – Simpsons metode som funksjonsfil:

```
function S=SimpsonFunk(x,y)

% Funksjon som beregner integral ved Simpsons metode
% for gitte x- og y-vektorer
% Partisjonen, gitt ved x-vektoren, må være regulær.

% Bestemmer steglengda og antall steg
h=x(2)-x(1);
n=length(x);

% Ende-bidrag:
S=h/3*(y(1)+y(n));

% Partalls-bidrag
for i=2:2:(n-1)
    S=S+h/3*4*y(i);
end

% Oddetalls-bidrag
for i=3:2:(n-2)
    S=S+h/3*2*y(i);
end
```

Her har vi ikke kontrollert at n er et partall, så denne funksjonsfila er ikke heilt idiotsikker.

Vi kjører skriptet og får opp plottet til venstre i figur 2. I figuren til høgre har vi plotta feilen mot h i stedet for n, og vi har brukt logaritmiske akser. På et slikt

plott ser feilen ut som rette linjer – med ulike stigningstall for de ulike metodene. Dette stemmer overens med at feilen går som h for Riemann-summen, som h^2 for trapesmetoden og h^4 for Simpsons metode[1]. Som i figur 1 ser vi at feilen ved trapesmetoden avtar mye raskere enn feilen ved en Riemann-sum når h avtar. Vi ser også at Simpsons-metode nærmest "knuser" begge de to andre.

Når feilen i metoden begynner å nærme seg maskinpresisjonen, får vi bare støy. Dermed vil det heller ikke ha noe for seg å redusere h ytterligere; dette ser faktisk ut til å gi et *dårligere* estimat. For Simpsons metode skjer dette allerede når h er omtrent 10^{-3}. Dette fenomenet er en vesentlig forskjell mellom numerikk og analytisk matematikk; analytisk *skal* estimatet bli bedre når h blir mindre – uansett. Den eksakte verdien av integralet får vi bare i grensa $h \to 0$ (eller $n \to \infty$). En datamaskin, derimot, er ikke i stand til å håndtere hvor små størrelser som helst; der er ei grense for hvor godt estimatet kan bli – og det optimale estimatet får vi for en eller annen endelig h-verdi.

Ekstra-oppgave 2: Monte Carlo-integrering

a) Siden begge uttrykkene skal gi $\overline{y}$, kan vi helt enkelt sette dem (tilnærma) lik hverandre:

$$\begin{aligned} \frac{1}{b-a}\int_a^b f(x)\,dx &\approx \frac{1}{n}\sum_{i=1}^{n} f(x_i) \Leftrightarrow \\ \int_a^b f(x)\,dx &\approx \frac{b-a}{n}\sum_{i=1}^{n} f(x_i) \end{aligned}$$

b) Hvis '`» rand`' gir et tilfeldig tall mellom 0 og 1, må '`a+(b-a)*rand`' gi et tilfeldig tall mellom `a` og `b`. Vi tester:

```
>> a=-1; b=3;
>> a+(b-a)*rand
ans =
   -0.6098
>> a+(b-a)*rand
ans =
    0.1140
>> a+(b-a)*rand
ans =
    1.1875
>> a+(b-a)*rand
ans =
    2.8300
```

Dette ser ut til å fungere. (Husk å bruke pil-tastene når du repeterer kommandoer på denne måten.)

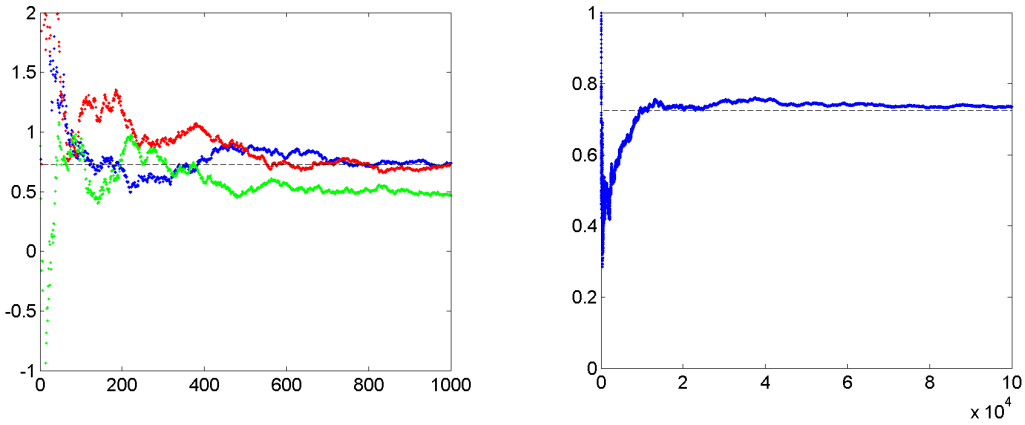
[1] Se side 276 i læreboka.

c) Vi velger oss det samme integralet som i ekstraoppgave 1:

$$\int_{-1}^{3} x \sin x^2 \, dx = \frac{\cos 1 - \cos 9}{2} \quad .$$

Når vi her har implementert denne integrasjonsmetoden, har vi valgt å gjøre det på en slik måte at for hver gang man trekker en ny x-verdi, plotter vi også det tilsvarende estimatet. På den måten vil vi kunne se at estimatet faktisk nærmer seg integralet når n øker opp mot en eller annen maksimalverdi `Nmax`, som vi bestemmer selv. Skriptet vårt ser slik ut:

```
% Skript som estimerer et integral ved Monte Carlo-metoden
% Selve integralet, og den analytiske løsninga av det, er
% spesifisert i starten av skriptet - i tillegg til
% maksimalt antall trekk.
% I tillegg til å estimere integralet, plotter skriptet
% estimatet for økende antall trekk. På den måten får vi
% sett hvordan estimatet nærmer seg det faktiske integralet.

% Integrasjonsgrenser
a=-1;
b=3;

% Integrand
funk=@(x) x*sin(x^2);

% Kjent analytisk svar:
Fasit=(cos(1)-cos(9))/2;

% Maksimalt antall trekk:
Nmax=1000;

% Initierer summen
MCsum=0;

% Lager klart plott for estimatene - sammen med eksakt svar
plot([1 Nmax],Fasit*[1 1],'k--')
hold on

% Vi utfører trekkene og legger til summen for hver gang
for n=1:Nmax
  x=a+(b-a)*rand;
  f=funk(x);
  MCsum=MCsum+f;
  Estimat=(b-a)*MCsum/n;          % Integral-estimat
  plot(n,Estimat,'b.')            % Plotter estimat
end
hold off
```



Figur 3: *Venstre*: Dette plottet viser hvordan tre ulike sekvenser av Monte Carlo-integrasjon varierer når vi øker antall trekk opp mot 1000. Som vi ser, er 1000 alt for lite til å få et presist estimat. *Høgre*: Dette plottet viser hvordan en Monte Carlo sekvens nærmer seg integralet når vi bruker hundre tusen trekk.

```
38 axis([0 Nmax -1 2])
39
40 % Skriver svaret til skjerm
41 Estimat
```

Her har vi fiksert antall trekk i linje 20. Det som har med plottet å gjøre, er gjort i linjene 25-27, 35, 37 og 38. Disse linjene er selvsagt ikke nødvendige for å finne selve estimatet.

Siden vi trekker ulike tilfeldige tall hver gang, blir ikke estimatet det samme hver gang selv om vi bruker samme `Nmax`. I plottet til venstre i figur 3 er det vist hvordan estimatet varierer når vi øker n for tre ulike eksempler. Selve integralet er også vist som ei stipla svart linje. Vi ser tydelig at 1000 trekk ikke er nok til å få et ok estimat for integralet $\int_{-1}^{3} x \sin x^2 \, dx$. I plottet til høgre, derimot, ser estimatet ut til å være bedre (legg merke til at y-aksen er forskjellig i de to plottene). Her har vi brukt 100 000 trekk.

I forhold til de andre metodene vi har sett på er det tydeligvis lite å hente her; vi får nøyaktige estimater mye raskere med både trapesmetoden og Simpsons metode enn det vi får med Monte Carlo-integrasjon i dette eksemplet. Men Monte Carlo-integrasjon kommer virkelig til sin rett når vi skal beregne fler-dimensjonale integraler, noe vi for øvrig ikke skal gjøre i dette kurset.