Forelesningsnotater, FYS1100 uke 8

Joakim Bergli

1 Numerisk derivasjon

1.1 Tilnærmet uttrykk for den deriverte

Den deriverte av en funksjon $f(x)$ i punktet a er gitt ved

$$f'(a) = \lim_{h\to 0} \frac{f(a+h) - f(a)}{h}.$$

Numerisk vil vi velge en liten verdi for h og regner ut den deriverte tilnærmet

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

Eksempel

$$f(x) = \sin x, \qquad a = 0,5$$

Vi vet at $f'(x) = \cos x$ slik at vi kan sammenlikne med det eksakte svaret. Vi lager et lite program som beregner $f'_h(a)$ for mindre og mindre verdier av h.

```
import numpy as np
import matplotlib.pyplot as plt

a = 0.5

print("h                df_h                df_h-cos(a)")

for h in 10.**(-np.arange(1,8)):
    d = (np.sin(a+h)-np.sin(a))/h
    print("%e     %e      %e"%(h,d,d-np.cos(a)))
```

Kjører vi det får vi dette

```
h                df_h                df_h-cos(a)
1.000000e-01     8.521693e-01      -2.541321e-02
1.000000e-02     8.751708e-01      -2.411734e-03
1.000000e-03     8.773427e-01      -2.398590e-04
1.000000e-04     8.775586e-01      -2.397274e-05
1.000000e-05     8.775802e-01      -2.397147e-06
1.000000e-06     8.775823e-01      -2.396897e-07
1.000000e-07     8.775825e-01      -2.469499e-08
```

Vi ser at verdien for den tilnærmede deriverte $f'_h(a)$ (andre kolonne) konvergerer mot en bestemt verdi når h avtar. Vi ser også at feilen $f'_h(a) - \cos a$ (tredje kolonne) avtar omtrent lineært med h. Vi skal nå prøve å forstå hvorfor.

1.2 Trunkeringsfeil

Vi bruker Taylors formel med restledd, og har at

$$f(a+h) = f(a) + f'(a)h + \frac{h^2}{2}f''(u), \qquad u \in [a, a+h].$$

Dermed er feilen

$$f'_h(a) - f'(a) = \frac{h}{2}f''(u).$$

Ettersom h avtar vil også u forandre seg, men siden intervallet $[a, a+h]$ blir ganske lite vil $u \approx a$ for nesten alle h vi har prøvd. Dermed er $\frac{1}{2}f''(u) \approx \frac{1}{2}f''(a) = -\frac{1}{2}\sin(a) = -0,23971276$. Vi ser at vi både forklarer at feilen avtar lineært med h og vi får til og med nesten riktig verdi for feilen.

1.3 Avrundingsfeil

Hva er den beste verdien å bruke for h? Er det alltid bedre med mindre h? Og kan datamaskina jobbe med vilkårlig små h? Hvis vi prøver med enda mindre verdier for h får vi

```
h               df_h              df_h-cos(a)
1.000000e-01    8.521693e-01     -2.541321e-02
1.000000e-02    8.751708e-01     -2.411734e-03
1.000000e-03    8.773427e-01     -2.398590e-04
1.000000e-04    8.775586e-01     -2.397274e-05
1.000000e-05    8.775802e-01     -2.397147e-06
1.000000e-06    8.775823e-01     -2.396897e-07
1.000000e-07    8.775825e-01     -2.469499e-08
1.000000e-08    8.775826e-01     2.850324e-10
1.000000e-09    8.775825e-01     -5.522612e-08
1.000000e-10    8.775825e-01     -1.107373e-07
1.000000e-11    8.775813e-01     -1.220960e-06
1.000000e-12    8.775758e-01     -6.772075e-06
1.000000e-13    8.776313e-01     4.873908e-05
1.000000e-14    8.770762e-01     -5.063724e-04
1.000000e-15    8.881784e-01     1.059586e-02
1.000000e-16    1.110223e+00     2.326405e-01
```

Vi ser at feilen har et minimum for omtrent $h = 10^{-8}$ og så øker igjen for mindre h. For de minste verdiene blir svaret ganske feil. For å se tydligere hva som skjer er det ofte lurt å plotte verdiene istedenfor å liste tallene. Vi legger derfor inn plotting i koden vår

```
import numpy as np
import matplotlib.pyplot as plt

a = 0.5

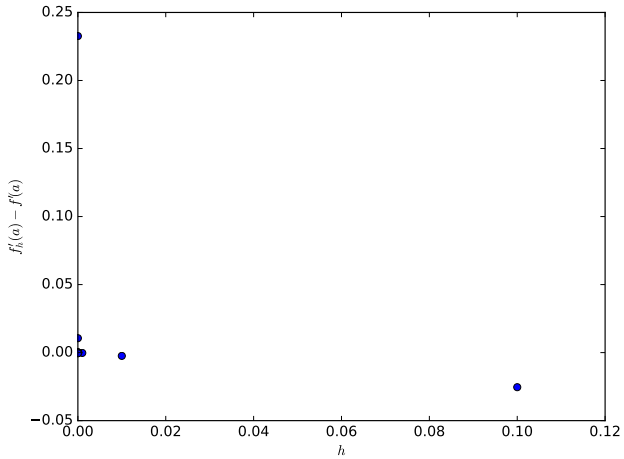
hL = []
eL = []
print("h               df_h              df_h-cos(a)")

for h in 10.**(-np.arange(1,17)):
    d = (np.sin(a+h)-np.sin(a))/h
    print("%e     %e      %e"%(h,d,d-np.cos(a)))
    hL.append(h)
    eL.append( d-np.cos(a))

plt.plot(hL,eL,"o")
plt.xlabel(r"$h$")
plt.ylabel(r"$  f_h'(a) - f'(a)$")
plt.savefig("d3.pdf")
```

```
plt.show()
```

og får dette plottet



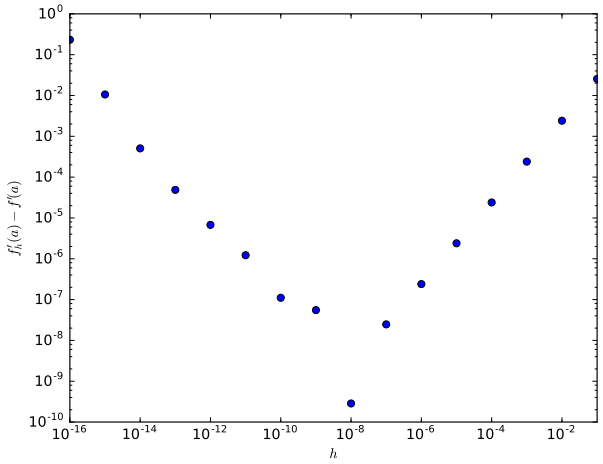
Som vi kan se er det ikke særlig informativt. Det er fordi alle de små verdiene av h kommer omtrent i origo siden det også er noen relativt store verdier av h. Et standard triks når man har veldig forskjellige verdier og har lyst til å vise dem i et plot er å plotte logaritmen til verdien istedenfor tallet selv (vi sier at vi bruker logaritmiske akser). Det får vi til ved å bytte ut kodelinja

```
plt.plot(hL,eL)
```

med

```
plt.loglog(hL,np.abs(eL))
```

Vi må ta absoluttverdien fordi vi ikke kan ta logarimen til negative tall. Da får vi isteden dette plottet



Nå ser vi tydelig at feilen avtar med h til $h \approx 10^{-8}$ og så øker igjen for mindre h. Hvorfor skjer dette? Datamaskina representerer desimaltall med et bestemt antall siffer. For flyttall i python er dette vanligvis 16 siffer. I tillegg har den en plass til en gitt tierpotens, slik at den fint kan representere tall

som $1,234567 \cdot 10^{45}$. Når h er veldig liten er $f(a+h)$ og $f(a)$ nesten like og dermed differansen nesten null. For eksempel hvis $h = 10^{-13}$ har vi for $a = 0,5$

$$\begin{aligned} f(a+h) &= 0.4794255386042908 \\ f(a) &= 0.4794255386042030 \\ f(a+h) - f(a) &= 0.0000000000000878 \end{aligned}$$

Differansen kan uttrykkes som $8.78???????????? \cdot 10^{-14}$, men alle desimalene etter de to første finnes egentlig ikke i maskina, og den fyller da på med noen tilfeldige(?), jeg fikk $8.7763130096618625 \cdot 10^{-14}$. Dermed blir den deriverte

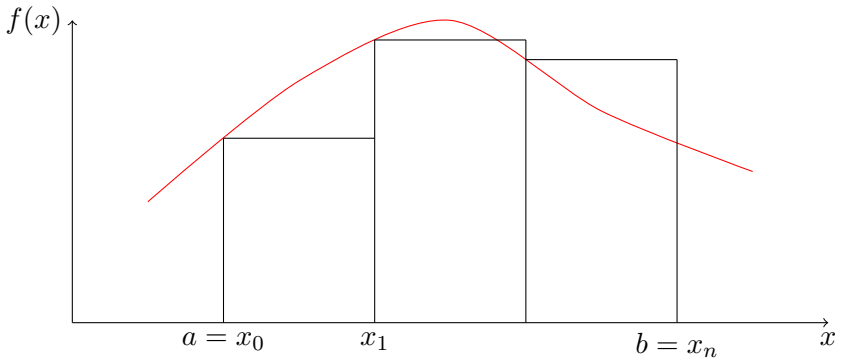
$$f'_{10^{-13}}(a) = 0.8776313009661862$$

men bare de første 3-4 sifrene har noen reell betydning, og resten er "funnet på" av maskina vår.

2 Numerisk integrasjon

Et integral kan vi litt forenklet tenke på som en sum av uendelig mange uendelig smale rektangler som fyller opp arealet under en graf. Når vi beregner integraler numerisk bruker vi et endelig antall smale rektangler (eller som vi skal se andre figurer) som tilnærmer det eksakte integralet. Når vi tar grensen der bredden av rektanglene går mot null trenger vi ikke å tenke så nøye på akkurat hvor vi velger toppen av rektangelet, men for endelige bredder har det noe å si, som gir oss forskjellige mulige integrasjonsmetoder. Vi skal se på et par enkle.

2.1 Rektangelmetoden (venstre trappesum)



Her velger vi høyden på rektangelet til funksjonsverdien i venstre endepunkt, slik at vi tilnærmer integralet som

$$I = \int_a^b f(x)dx \approx \sum_{i=0}^{n-1} f(x_i)\Delta x$$

der $\Delta x = x_{i+1} - x_i$ (vi velger lik bredde på alle rektanglene, men det finnes tilfeller der det er lurt å bruke ulike lengder også) og vi setter $x_0 = a$. Dette er den metoden vi har brukt tidligere f eks da vi gjorde eksperimentet med å måle akselerasjon i heisen, og så regnet ut fart og posisjon.

Eksempel

Vi vil regne ut

$$I = \int_0^1 \sin x dx = \big[-\cos x \big]_0^1 = 1 - \cos 1 = 0,459$$

Vi lager et lite program som beregner $\sum_{i=0}^{n-1} f(x_i)\Delta x$ for avtagende verdier av Δx.

```
import numpy as np
import matplotlib.pyplot as plt

a,b = 0.,1.

analytisk = 1-np.cos(1.0)


print("dx                integral_dx                feil")

for dx in 10.**(-np.arange(1,10)):
    x = np.arange(a,b,dx)
    integral = sum(np.sin(x))*dx
    print("%e     %.10e     %e"%(dx,integral,integral-analytisk))
```

Kjører vi det får vi dette

```
dx                 integral_dx              feil
1.000000e-01     4.1724099962e-01     -4.245669e-02
1.000000e-02     4.5548650839e-01     -4.211186e-03
1.000000e-03     4.5927692033e-01     -4.207738e-04
1.000000e-04     4.5965562020e-01     -4.207393e-05
1.000000e-05     4.5969348677e-01     -4.207359e-06
1.000000e-06     4.5969727340e-01     -4.207355e-07
1.000000e-07     4.5969765206e-01     -4.207361e-08
1.000000e-08     4.5969768992e-01     -4.207372e-09
1.000000e-09     4.5969769371e-01     -4.207485e-10
```

Under kjøringen observerte vi at tida det tar øker med Δx, og at siste linja tar en stund å beregne. Vi vil derfor ofte ønske å bruke så store Δx som mulig for å få tilstrekkelig nøyaktig resultat. Vi ser at feilen avtar omtrent lineært med Δx.

2.2 Trunkeringsfeil

Vi rekkeutvikler

$$f(x_i + y) = f(x_i) + f'(x_i)y + \cdots$$

og får

$$\int_{x_i}^{x_{i+1}} f(x)dx = \int_0^{\Delta x} f(x_i + y)dy \approx f(x_i)\Delta x + f'(x_i)\int_0^{\Delta x} ydy = f(x_i)\Delta x + \frac{1}{2}f'(x_i)\Delta x^2.$$

Hvis

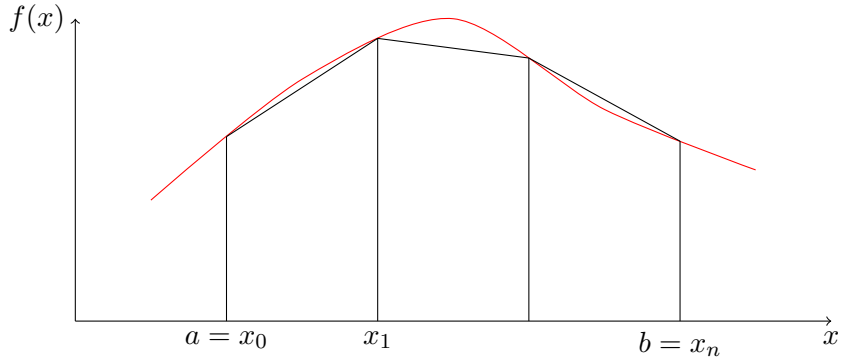
$$I = \int_a^b f(x)dx \qquad I_{\Delta x} = \sum_{i=0}^{n-1} f(x_i)\Delta x$$

er feilen

$$E = I - I_{\Delta x} = \sum_i \frac{1}{2} f'(x_i)\Delta x^2 = \frac{1}{2}\Delta x \sum_i f'(x_i)\Delta x \approx \int_a^b f'(x)dx = \frac{1}{2}\Delta x[f(b) - f(a)].$$

2.3 Trapesmetoden

Istedenfor å bruke rektangler kan vi bruke trapeser som stemmer med funksjonen både i høyre og venstre endepunkt. Intuitivt er dette bedre, og vi skal se at det stemmer, og analysere hvor mye bedre.



Dette gir approksimasjonen

$$I = \int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \frac{1}{2}[f(x_i) + f(x_{i+1}]\Delta x = \sum_{i=0}^{n-1} f(x_i)\Delta x + \frac{1}{2}\Delta x[f(b) - f(a)].$$

Legg merke til at dette er det samme uttrykket som ved venstre trappesum korrigert med feiluttrykket vi fant for den.

Vi kan gjøre et tilsvarende feilestimat for trapesmetoden som vi gjorde for venstre trappesum over. Vi må da rekkeutvikle til 2. orden

$$f(x_i + y) = f(x_i) + f'(x_i)y + \frac{1}{2}f''(x_i)y^2 + \cdots$$

og får

$$\begin{aligned}\int_{x_i}^{x_{i+1}} f(x)dx &= \int_0^{\Delta x} f(x_i + y)dy \approx f(x_i)\Delta x + f'(x_i)\int_0^{\Delta x} ydy + \frac{1}{2}f''(x_i)\int_0^{\Delta x} y^2 dy \\ &= f(x_i)\Delta x + \frac{1}{2}f'(x_i)\Delta x^2 + \frac{1}{6}f''(x_i)\Delta x^3.\end{aligned}$$

For å finne igjen trapesmetoden i dette må vi bruke Taylorrekka en gang til

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2.$$

Dette kan vi omforme til

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{\Delta x} - \frac{1}{2}f''(x_i)\Delta x^2.$$

Hvis vi setter inn dette i uttrykket over og summerer over alle intervallene får vi

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \frac{1}{2}[f(x_i) + f(x_{i+1}]\Delta x - \frac{1}{12}\Delta x^2[f'(b) - f'(a)].$$

Feilen i trapesmetoden er altså proposjonal med Δx^2 og avtar dermed mye fortere enn feilen i venstre trappesum som er proporsjonal med Δx.

Vi kan teste dette på eksempelet vårt og koder både venstre trappesum og trapesmetoden, sammenlikner med det analytiske svaret og plotter feilene sammen med våre estimat for feil (legg merke til logaritmiske akser).

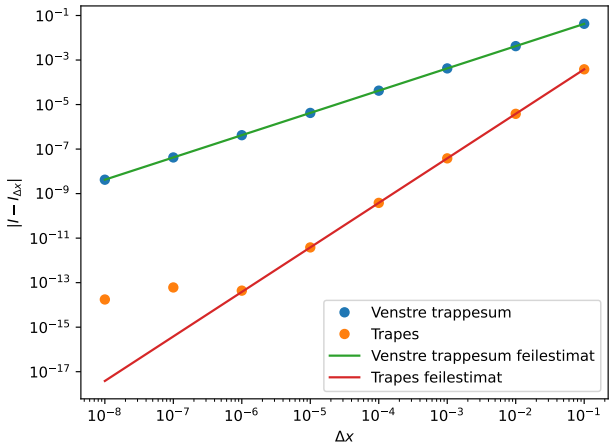
```
import numpy as np
import matplotlib.pyplot as plt

a,b = 0.,1.

eL = []
tL = []
dxL = []
analytisk = 1-np.cos(1.0)
a2 = (np.sin(b)-np.sin(a))/2
a3 = np.abs((np.cos(b)-np.cos(a))*1/12)


for dx in 10.**(-np.arange(1,9)):
    x = np.arange(a,b,dx)
    n = len(x)
    integral = sum(np.sin(x))*dx
    trap = integral +(np.sin(b)-np.sin(a))*dx/2.
    print("%e     %e     %e     %e     %e"%(dx,integral,integral-analytisk,trap,trap-analytisk))
    eL.append(integral)
    tL.append(trap)
    dxL.append(dx)


plt.loglog(dxL,np.abs(eL-analytisk),"o",label="Venstre trappesum")
plt.loglog(dxL,np.abs(tL-analytisk),"o",label="Trapes")
plt.loglog(dxL,[a2*dx for dx in dxL],label="Venstre trappesum feilestimat")
plt.loglog(dxL,[a3*dx**2 for dx in dxL],label="Trapes feilestimat")
plt.xlabel(r"$\Delta x$")
plt.ylabel(r"$|I-I_{\Delta x}|$")
plt.legend(loc="lower right")
plt.savefig("i2.pdf")
plt.show()
```



Vi ser at feilen i trapesmetoden avtar fortere, helt til den blir så liten at avrundingsfeil gjør at det ikke blir mer nøyaktig mer. For en gitt nøyaktighet kan vi altså bruke større Δx i trapesmetoden som gir raskere beregning.