Oppgavesett til uke 8 (19/10 - 23/10)

Har du noen gang lurt på hvordan datamaskiner og kalkulatorer regner ut logaritmer? Og hva er egentlig poenget med et Taylor-polynom hvis vi allerede kjenner funksjonen?

Dette oppgavesettet handler om disse spørsmålene. En datamaskin er god på de fire regneartene (addisjon, subtraksjon, multiplikasjon og divisjon). Dette gjør det enkelt for den å jobbe med polynomer (inkludert Taylor-polynomer), og kan utnyttes til å jobbe med mer avanserte funksjoner.

Hovedpoenget med disse oppgavene er at alle i gruppa skal forstå

- hvordan datamaskinen regner ut logaritmer
- hvordan måten desimaltall representeres på i datamaskinen kan hjelpe oss her
- hvordan Taylor-polynomer kan være nyttige selv om vi kjenner funksjonen

Det er viktig å stille spørsmål og forklare til hverandre så alle forstår. Spør gjerne en gruppelærer etter behov.

Oppgave 1

I hele dette oppgavesettet skal vi jobbe med den naturlige logaritmen $\ln(x)$ eller `log(x)` som den heter i Python.

Se først på filen `taylorlog.py`. Denne inneholder følgende funksjoner:

Funksjonens navn	Hva funksjonen gjør	Ferdig laget?
`taylorterm`	Regner ut ett ledd av Taylor-polynomet	Ja
`taylor`	Regner ut hele Taylor-polynomet	Ja
`errorterm`	Regner ut absoluttverdien av restleddet til Taylor-polynomet	Nei (oppgave 1)
`taylorlog`	Vår egen logaritmefunksjon, som er smartere enn å bare bruke Taylor-polynomet direkte	Nei (oppgave 3)

Vi har valgt variabelnavn som samsvarer med de i kompendiet:

Variabel	Forklaring
`n`	Nummer på siste ledd i Taylor-polynomet
`a`	Punktet på x-aksen vi brukte for å lage Taylor-polynomet (større enn 0)
`lna`	Logaritmen til `a`. Av praktiske grunner[1] bruker vi denne som parameter til funksjonene i stedet for `a` direkte.
`x`	Verdien vi regner ut Taylor-polynomet/logaritmen for (større enn 0)
`xi`	Et ukjent punkt som ligger et sted på aksen mellom `a` og `x`. Vi bruker dette punktet til å regne ut restleddet.

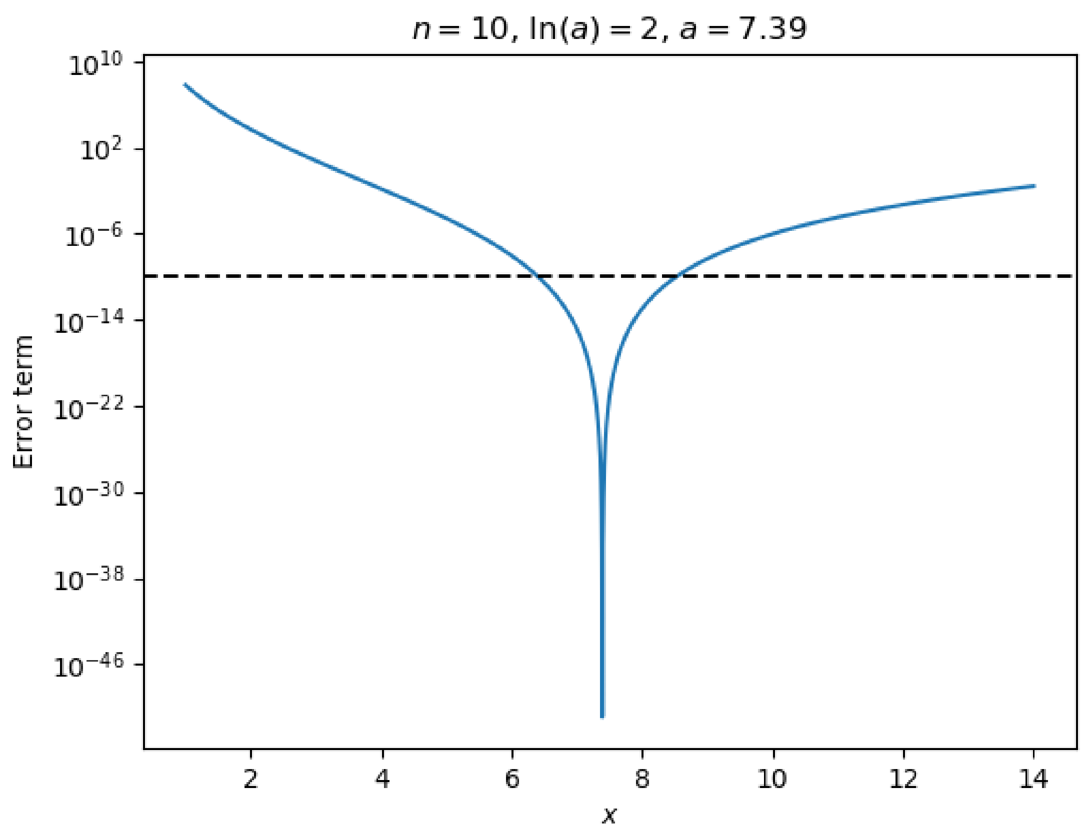
Det første dere skal gjøre er å fullføre funksjonen `errorterm`, slik at denne returnerer riktig verdi. En utfordring her er at dere ikke kjenner verdien av `xi`, alt dere vet er at den ligger mellom `a` og `x` et sted (uansett hvem av de to som er størst). Jobb rundt dette problemet ved å velge verdien av `xi` som gjør restleddet så stort som mulig og dermed gir oss en øvre grense for feilen.

[1] For å slippe å jukse og bruke `log(a)` i det 0-te leddet av Taylor-polynomet.

Oppgave 2

Test koden fra oppgave en ved å kjøre programmet `taylor_test.py`.

Med startverdiene `lna = 2, n = 10, xmin = 1` og `xmax = 14` skal programmet gi følgende plott av restleddet. Den horisontale linjen i plottet er en grense på 10^{-10} som vi sier er nøyaktig nok for dette oppgavesettet.



OBS: Hvis dere bare får «halve» plottet, husk at `errorterm` skal plotte *absoluttverdien* av restleddet (ellers mister vi alle de negative verdiene pga. den logaritmiske y-aksen).

Eksperimenter litt med å forandre verdiene av `lna` og `n` i `taylor_test.py`. Forklar hva som skjer med restleddet når dere endrer disse.

Oppgave 3

Det er kanskje ingen overraskelse at Taylor-polynomet er mest nøyaktig nært punktet `a` og mindre nøyaktig langt unna dette punktet. Siden vi bare kan velge ett punkt, er det vanskelig å lage et Taylor-polynom som gir oss en nøyaktig logaritme for alle positive tall[2].

For å jobbe rundt denne begrensningen, kan vi heller utnytte at på datamaskinen representeres desimaltall som flyttall på formen

```
x = mantissa * 2**exponent
```

Her er `mantissa` et tall mellom 0.5 and 1 og `exponent` er et entydig heltall. Hva skjer (på høyre side av likhetstegnet) når dere tar logaritmen dette (på begge sider)?

Bruk resultatet til å gjøre ferdig funksjonen `taylorlog` i `taylorlog.py.` Som dere ser, har Python allerede en funksjon som gir oss `mantissa` og `exponent` fra et gitt tall `x`, men husk at dere skal bruke funksjonen `taylor` i stedet for `log` når dere trenger en logaritme. Å bruke funksjonen vi prøver å lage fra bunnen av ville vært juks (ett unntak[3]: dere kan bruke den kjente verdien av logaritmen til 2, som med maskin-nøyaktighet er 0.6931471805599453).

For å teste om funksjonen fungerer, kjør programmet `log_test.py` med de oppgitte startverdiene `lna = 2` og `n = 10`. Hvis det fungerte, skal dere se dette resultatet (eller noe ganske likt):

[2] Faktisk er det umulig for akkurat denne funksjonen, uansett hvor mange ledd vi bruker, men grunnene til dette skal vi ikke gå nærmere innpå her. Det ville uansett krevd et altfor høyt antall ledd.

[3] Eller, hvis dere insisterer, bruk eventuelt `taylorlog` til dette også. Siden dette gjør oppgave 4 litt mer utfordrende krever vi ikke at dere gjør det, men det selvsagt ikke forbudt.

```
x = 0.001
---
log:             -6.907755278982137
taylorlog:       -6.570232074265512
relative error: 0.04886148844091062

x = 0.1
---
log:             -2.3025850929940455
taylorlog:       -2.1452253783194606
relative error: 0.06834045575704238

x = 10
---
log:             2.302585092994046
taylorlog:       2.550213826546409
relative error: 0.10754379254248189

x = 1000
---
log:             6.907755278982137
taylorlog:       7.009306421013527
relative error: 0.014701033538402011

ln(a) = 2
a     = 7.38905609893065
n     = 10
```

Som dere ser blir den relative feilen med `ln(a) = 2` og `n = 10` ganske store (vi får mange gale siffer i svaret). Ingen grunn til bekymring; dette skal vi fikse i neste oppgave. 😊

Oppgave 4

Det vi trenger for å få dette nøyaktig er at Taylor-polynomet til logaritmen er nøyaktig nok i hele intervallet mellom 0.5 og 1 (fordi `mantissa` i funksjonen `taylorlog` er et tall mellom 0.5 og 1)[4]:

Bruk først `taylor_test.py` til å finne verdier av `lna` og `n` som gir dere et lite nok restledd i *hele* intervallet mellom `xmin = 0.5` og `xmax = 1` (husk at 10^{-10} er godt nok i praksis her).

- Hint: Hvor bør `a` plasseres for å få det så nøyaktig som mulig?
- Prøv å bruke så få ledd som mulig (lav `n`) slik at programmet kjører raskere[5].
- Hvis dere vil sammenligne forskjellige plott, så lagrer programmet dem automatisk som bilder i den samme mappen som dere kjører programmene fra.

Når dere har funnet verdier som gir god nøyaktighet, bruk så `log_test.py` til å teste logaritmefunksjonen dere lagde i oppgave 3 (men husk for all del å kopiere over verdiene av `lna` og `n` som fungerte godt i plottene).

- Hvis feilen er større enn 10^{-10} for noen av eksemplene likevel, justér `lna` og `n` litt til, slik at feilen blir akseptabel for alle sammen.

Oppgave 5

Oppsummér for hverandre, med deres egne ord, hva vi har gjort:

- Hvordan datamaskinen regner ut logaritmer[6]
- Hvordan vi sjekket at Taylor-polynomet ble nøyaktig (nok)
- Hvordan vi utnyttet representasjonen av flyttall til å gå fra en tilnærming som måtte være nøyaktig for alle positive tall til en som bare måtte være nøyaktig i et lite intervall på x-aksen.

Målet her er at *alle* på gruppa skal forstå hva dere gjorde og hvorfor. Spør gjerne en gruppelærer om noe er uklart eller vanskelig å forklare, eller bare hvis dere vil sjekke om dere har fått med dere det viktigste.

---

[4] Hvis dere brukte `taylorlog` til å finne logaritmen til 2 i forrige oppgave må det være nøyaktig for 2 også, så vi ikke får en ekstra unøyaktighet inn der.

[5] Dette vil faktisk ha stor betydning hvis noen skal gjøre millioner av utregninger eller mer. Generelt er det et fornuftig prinsipp å skrive kode som er så effektiv som mulig.

[6] I praksis brukes tilnærminger med færre ledd enn et Taylor-polynom, så det skal gå enda raskere, men det grunnleggende prinsippet er det samme.