Programmering i realfagene

Faglig opplegg: programmering i matematikk

kodeskolen simula

kodeskolen@simula.no

De to siste ukene har vi gått igjennom en grunnleggende innføring i Pythonprogrammering. Denne uka skal vi se på mer konkrete realfagsprosjekter. Vi begynner med numerisk løsning av likninger med *halveringsmetoden*

Det er mange grunner til at det er nyttig å løse likninger numerisk. For det første er det ikke sikkert at det er mulig å løse likningen analytisk ($cos(x) - x = 0$ er et eksempel på en slik likning), til tross for det er det null problem for en numerisk metode å finne en veldig god tilnærming. I tillegg til dette er det sjeldent man har behov for det korrekte svaret, et svar som er 10^{-10} unna er godt nok.

Innhold

1 Prosjekt: Løse likninger med datamaskinen

Her skal vi komme frem til en algoritme for å løse vilkårlige likninger ved hjelp av datamaskinen. Men, før vi begynner på det, skal vi fortsette med gjetteleken vi lærte om forrige uke.

1.1 Omvendt gjettelek

Sist uke lagde vi et program hvor datamaskinen tenker på et tall mellom 0 og 1000, og vi skal finne ut hva det er. Hver gang vi gjetter får vi vite om tallet er for høyt eller for lavt. Programmet vi endte opp med så slik ut:

```
fasitsvar = 45
maksforsøk = 100
print("Jeg har tenkt på et tall mellom 1 og 1000.")
print("Prøv å gjett det!")
print()

antall_forsøk = 0

while gjett != fasitsvar and antall_forsøk < maksforsøk:
    gjett = int(input("Gjett: "))
    antall_forsøk += 1

    if gjett == fasitsvar:
        print("Helt riktig!")
        print(f"Du brukte {antall_forsøk} forsøk!")
    elif gjett > fasitsvar:
        print(f"Ditt gjett på {gjett} er for høyt.")
    elif gjett < fasitsvar:
        print(f"Ditt gjett på {gjett} er for lavt.")

if gjett != fasitsvar:
    print(f"Der har du prøvd {maksforsøk} ganger og gått
       tom for gjett! Riktig svar er {fasitsvar}")
```

I tillegg til dette, fant vi ut at den beste måten å gå frem på, var å gjette i midten av intervallet (`mintall`, `makstall`) og oppdatere det ut i fra hvilket svar vi får. Hvis

du ikke husker dette, kan det være en idé å lese i kompendiet fra forrige kursdag. Nå skal vi snu programmet på hodet – vi skal lage et program som gjetter seg frem til hvilket tall vi tenker på!

Det første vi trenger er en funksjon som returnerer et gjett. Og vi ønsker at gjettet skal være midtpunktet mellom det minste mulige tallet og det største mulige tallet. Vi trenger altså en funksjon som tar inn to tall og returner midtpunktet:

```
def midtpunkt(a, b):
    return (a + b)/2
```

Vi vil også ha en løkke hvor datamaskinen gjetter hvilket tall vi tenker på. Tilsvarende slik vi hadde det da vi hadde en løkke hvor vi gjetter hvilket tall datamaskinen tenker på.

```
maks_tall = 100
min_tall = 0

maks_forsøk = 100

print(f'Tenk på et tall mellom {min_tall} og {maks_tall}')

gjettet_riktig = False

while not gjettet_riktig:
    gjett = int(midtpunkt(min_tall, maks_tall))
    print(f'Tenker du på {gjett}?')
```

```
Tenk på et tall mellom 0 og 100
Tenker du på 50?
Tenker du på 50?
Tenker du på 50?
...
Tenker du på 50?
Tenker du på 50?
```

Det observante øyet ser det står `gjett =int(midtpunkt(min_tall, maks_tall)`). Grunnen til at vi bruker `int` er at vi tenker kun på heltall, så derfor vil vi

at maskinen bare skal gjette heltal. Hvis datamaskinen gjetter et desimaltall så rundes det ned til nermeste heltall.

Vi ser også at datamaskinen alltid gjetter 50 og at vi har fått en *uendelig løkke*. For å oppdatere gjettet må jo vi gi en tilbakemelding om datamaskinen gjettet for høyt eller for lavt. For å få tilbakemelding fra brukeren kan vi bruke `input` funksjonen vi lærte forrige uke og en `if`-test.

Det vi ønsker er at dersom brukeren svarer `'riktig'`, så skal vi bryte ut av løkka. Da har vi jo funnet svaret og vi trenger ikke gjette mer. For å bryte ut av løkka holder det å sette `gjettet_riktig =True`, Da er ikke lenger betingelsen `not gjettet_riktig` sann og `while` løkka stopper.

```
maks_tall = 100
min_tall = 0

print(f'Tenk på et tall mellom {min_tall} og {maks_tall}')
gjettet_riktig = False
while not gjettet_riktig:
    gjett = int(midtpunkt(min_tall, maks_tall))
    print(f'Tenker du på {gjett}?')
    svar = input('Var dette for høyt, for lavt eller
       riktig?')
    if svar == 'riktig':
        print(f'Jippi, jeg greide det!')
        gjettet_riktig = True
```

```
Tenk på et tall mellom 0 og 100
Tenker du på 50?

Var dette for høyt, for lavt eller riktig? @@riktig@@
Jippi, jeg greide det!
```

Det virker! Eller i hvertfall om datamaskinen gjetter riktig på første forsøk... La oss modifisere koden slik at den virker om vi sier for høyt eller for lavt og. Før vi gjør den modifiseringen må vi bestemme oss for hva datamaskinen skal gjøre om den gjetter for lavt eller for høyt.

Hvis 50 var for høyt så ønsker vi at datamskinen skal gjette mellom null og femti.

For å gjøre det, oppdaterer vi `maks_tall` og setter den lik `gjett` (som nå er 50). Tilsvarende, hvis 50 var for lavt oppdaterer vi `min_tall` og setter den lik `gjett`. For å gjøre dette bruker vi `elif`-setningene vi lærte forrige uke.

```
maks_tall = 100
min_tall = 0

print(f'Tenk på et tall mellom {min_tall} og {maks_tall}')

gjettet_riktig = False

while not gjettet_riktig:
    gjett = int(midtpunkt(min_tall, maks_tall))
    print(f'Tenker du på {gjett}?')
    svar = input('Var dette for høyt, for lavt eller
       riktig? ')
    if svar == 'riktig':
        print(f'Jippi, jeg greide det!')
        gjettet_riktig = True
    elif svar == 'for høyt':
        maks_tall = gjett
    elif svar == 'for lavt':
        min_tall = gjett
    else:
        print('Jeg forsto ikke hva du mente, du må svare "
           riktig", "for høyt" eller "for lavt"')
```

```
Tenk på et tall mellom 0 og 100
Tenker du på 50?

Var dette for høyt, for lavt eller riktig? for høyt
Tenker du på 25?

Var dette for høyt, for lavt eller riktig? for lavt
Tenker du på 12?

Var dette for høyt, for lavt eller riktig? riktig
Jippi, jeg greide det!
```

Her ser vi at datamaskinen krymper intervallet den gjetter i hver gang den gjetter feil helt til den til slutt gjetter riktig. Dette kalles *halveringsmetoden* eller *bisection method* på engelsk. Halveringsmetoden kan også brukes til å løse likninger numerisk.

1.2 Bruke halveringsmetoden til å finne nullpunkt

Nå har vi klart å lage et program som lar datamaskinen systematisk søke seg frem til riktig tall. Den samme strategien kan brukes for å systematisk søke seg frem til et nullpunkt for en funksjon $f(x)$.

La oss si at vi har en funksjon $f(x) = 5x - 125$ som vi vil løse. Vi kan koden denne funksjonen i Python slik:

```
def f(x):
    return 5*x - 125
```

Vi ønsker at maskinen skal finne nullpunktet til denne funksjonen med halveringsmetoden. Vi begynner helt likt som for gjetteleken med å definere en midtpunktfunksjon, et `min_tall` og et `maks_tall`.

```

def midtpunkt(a, b):
    return (a+b)/2

maks_tall = 100
min_tall = 0

maks_forsøk = 100

gjettet_riktig = False
while not gjettet_riktig:
    gjett = midtpunkt(min_tall, maks_tall)
    print(f'Jeg gjetter {gjett}!')

....
```

Så ønsker vi at maskinen skal sjekke om den gjettet riktig, for høyt eller for lavt.

For å gjøre det kan den sjekke om `f(gjett)` er 0, over null, eller under null. Hvis `f(gjett)` er 0, så har vi funnet riktig svar. Men hva med dersom `f(gjett)` er over 0?. Vel, vi vet jo at `f(gjett)` er en linær funksjon som stiger. Og vi vet at nullpunktet er mellom 0 og 100. Så hvis `f(gjett)` er positivt, da må nullpunktet ligge mellom 0 og `gjett`. Funksjonen må jo passere 0 for å bli positiv! Vi kan altså sette `maks_gjett =gjett` akkurat slik som vi gjorde det for gjetteleken.

På samme måte kan vi sette `min_gjett =gjett` dersom `f(gjett)< 0`. Det fulle programmet blir da:

```

def midtpunkt(a, b):
    return (a+b)/2

maks_tall = 100
min_tall = 0

gjettet_riktig = False
while not gjettet_riktig:
    gjett = midtpunkt(min_tall, maks_tall)
    print(f'Jeg gjetter {gjett}!')

    if f(gjett) == 0:
        print(f'Jippi, jeg greide det!, f({gjett})={f(
            gjett)}!')
        gjettet_riktig = True

    elif f(gjett) > 0:
        maks_tall = gjett
    else:
        min_tall = gjett
```

```
Jeg gjetter 50.0!
Jeg gjetter 25.0!
Jippi, jeg greide det!, f(25.0)=0.0!
```

Som vi ser fungerte dette veldig bra. Datamaskinen fant nullpunktet fort.

La oss prøve å endre funksjonen til noe som er litt vanskeligere, f.eks

$$f(x) = \frac{x}{100}^{5} - 0.1log\left(\frac{x+1}{100}\right) - 0.5 \tag{1}$$

I Python kan vi definere den slik:

```
from pylab import log

def f(x):
    return (x/100)**5 - 0.1 * log((x + 1)/100) - 0.5
```

Hvis vi bytter ut `f` funksjonen i programmet over med denne funksjonen, dukker det opp et problem. Programmet vi kjøre for alltid og etterhvert begynner det å gjette det samme om igjen og om igjen.

```
...
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
```

Hvorfor skjer dette? La oss begynne med å sette et maks antall forsøk for å unngå en uendelig løkke. Det modifiserte programmet blir da slik:

```
maks_tall = 100
min_tall = 0

maks_forsøk = 100

gjettet_riktig = False
forsøk = 1
while not gjettet_riktig and forsøk < maks_forsøk:
    gjett = midtpunkt(min_tall, maks_tall)
    print(f'Jeg gjetter {gjett}!')

    if f(gjett) == 0:
        print(f'Jippi, jeg greide det!, f({gjett})={f(
           gjett)}!')
        gjettet_riktig = True

    elif f(gjett) > 0:
        maks_tall = gjett
    else:
        min_tall = gjett

    forsøk += 1
```

Vi kan nå legge til en if test på slutten som printer ut siste gjett dersom vi ikke klarte det:

```
if not gjettet_riktig:
    print(f'Jeg brukte opp alle {maks_forsøk} forsøk uten
       å finne nullpunktet')
    print(f'Det er et sted mellom {min_tall} og {maks_tall
       }')
    print(f'Siste gjett var f({gjett}) = {f(gjett)}')
```

Utskriften vi får fra programmet vårt nå er:

```
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg gjetter 86.58862684798157!
Jeg brukte opp alle 100 forsøk uten å finne nullpunktet
```

```
Det er et sted mellom 86.58862684798157 og
   86.58862684798159
Siste gjett var f(86.58862684798157) = -5.551115123125783e
   -17
```

Men $-5.55111512312578310^{-17}$ er jo fryktelig nærme 0! Det som har skjedd her er at det korrekte svaret krever fryktelig mange desimalers nøyaktighet. Og datamaskinen har en begrensning for hvor mange desimaler nøyaktighet den kan lagre. Det gjør at den ikke kan finne nøyaktig nullpunkt. Hva gjør vi da?

Likninger er (som oftest) ikke noe vi løser for morroskyld. Vi løser dem som en del av et større problem, slik som "hvor høy spenning bør jeg ha i denne kretsen?", "hva er avstanden mellom disse hjernecellene?" eller "kommer kanonkula til å treffe borgen vår?". Da er vi ikke interessert i det "sanne" svaret med to streker under, for vi er en gang helt sikre på hvordan likningene vi løser skal se ut. Da er det nok å finne et omtrentlig svar.

Vi vil altså sjekke at tallet er nærme nok null. Det kan vi gjøre ved å sjekke at absoluttverdien til tallet er mindre enn en grense. Grensen setter vi etter hvor nøyaktig vi vil at svaret skal være. For å finne absoluttverdi kan vi bruke `abs` funksjonen som er innebygd i Python, eller lage vår egen absoluttverdifunksjon.

Det vi må gjøre er å definere en terskel for hvor nærme null vi ønsker å være. For eksempel `terskel =0.0001` og så bytter vi ut `f(gjett)== 0` med `abs(f(gjett)) < terskel` Den fullstendige koden blir da:

```python
def midtpunkt(a, b):
    return (a+b)/2

maks_tall = 100
min_tall = 0

maks_forsøk = 100
terskel = 0.001

gjettet_riktig = False
forsøk = 1
while not gjettet_riktig and forsøk < maks_forsøk:
    gjett = midtpunkt(min_tall, maks_tall)
    print(f'Jeg gjetter {gjett}!')

    if abs(f(gjett)) < terskel:
        print(f'Jippi, jeg greide det på {forsøk} forsøk!, f({gjett})={f(gjett)}!')
        gjettet_riktig = True

    elif f(gjett) > 0:
        maks_tall = gjett
    else:
        min_tall = gjett

    forsøk += 1

if not gjettet_riktig:
    print(f'Jeg brukte opp alle {maks_forsøk} forsøk uten å finne nullpunktet')
    print(f'Det er et sted mellom {min_tall} og {maks_tall}')
    print(f'Siste gjett var f({gjett}) = {f(gjett)}')
```

```
Jeg gjetter 50.0!
Jeg gjetter 75.0!
Jeg gjetter 87.5!
Jeg gjetter 81.25!
Jeg gjetter 84.375!
Jeg gjetter 85.9375!
```

```
Jeg gjetter 86.71875!
Jeg gjetter 86.328125!
Jeg gjetter 86.5234375!
Jeg gjetter 86.62109375!
Jippi, jeg greide det på 10 forsøk!, f(86.62109375)
   =0.0008761689140125428!
```

Som vi ser fungerte dette veldig bra. Datamaskinen fant nullpunktet på 10 forsøk med ganske god nøyaktighet. Og dersom vi ønsker bedre nøyaktighet kan vi gjøre terskelen mindre. Prøv selv og se hva slags utskrift du får for terkel lik 0.0001 eller 0.0000001

Men, det er et problem med denne koden. For å illustrere dette prøver vi å finne nullpunktet til funksjonen

$$g(x) = -f(x) = -\left(\frac{x}{100}^{5} - 0.1log\left(\frac{x+1}{100}\right) - 0.5\right) \tag{2}$$

I Python kan vi definere den slik:

```
1 def g(x):
2     return -f(x)
```

Hvis vi bytter ut f med g i koden vår, får vi følgende utskrift:

```
Jeg gjetter 50.0!
Jeg gjetter 25.0!
Jeg gjetter 12.5!
...
Jeg gjetter 6.310887241768094e-28!
Jeg gjetter 3.155443620884047e-28!
Jeg gjetter 1.5777218104420236e-28!
Jeg brukte opp alle 100 forsøk uten å finne nullpunktet
Det er et sted mellom 0 og 1.5777218104420236e-28
Siste gjett var f(1.5777218104420236e-28) =
   0.039482981401190886
```

Her har datamaskinen havner helt på villspor! Den prøver å finne nullpunktet

fryktelig nærme 0. Men vi vet jo at det er i $x \approx 86$!

La oss undersøke hva som skjer her. $g(50) = 0.4$, og derfor blir øvre grense satt til å være femti. Dette er feil, nullpunktet er jo i $x \approx 86$! Koden burde ha flyttet den nedre grensa, ikke den øvre. Grunnen til dette er at vi naivt tror at funksjonen er positiv i $x = 100$ og negativ i $x = 0$. Dette stemmer ikke for vår funksjon g, og dermed virker ikke koden.

Hvordan kan vi fikse dette? Vi trenger en annen måte å bestemme om vi skal flytte øvre eller nedre grense? Dette kan vi gjøre ved å sjekke *fortegnet* til gjettet vårt. Hvis fortegnet til `f(gjett)` er likt fortegnet til `f(maks_tall)` betyr det at `gjett` og `maks_tall` er på samme side av nullpunktet. Altså må nullpunktet ligge mellom `min_tall` og `gjett`. Vi flytter derfor den øvre grensen. Dersom fortegnet til `f(gjett)` istedet er likt fortegnet til `f(maks_tall)`, flytter vi den nedre grensen.

For finne fortegn for et tall importerer vi `sign` fra pylab. Sign er en funksjon som tar inn et tall og returnerer 1 dersom det er et negativt tall og -1 dersom det er et positivt tall.

Her er den ferdige koden:

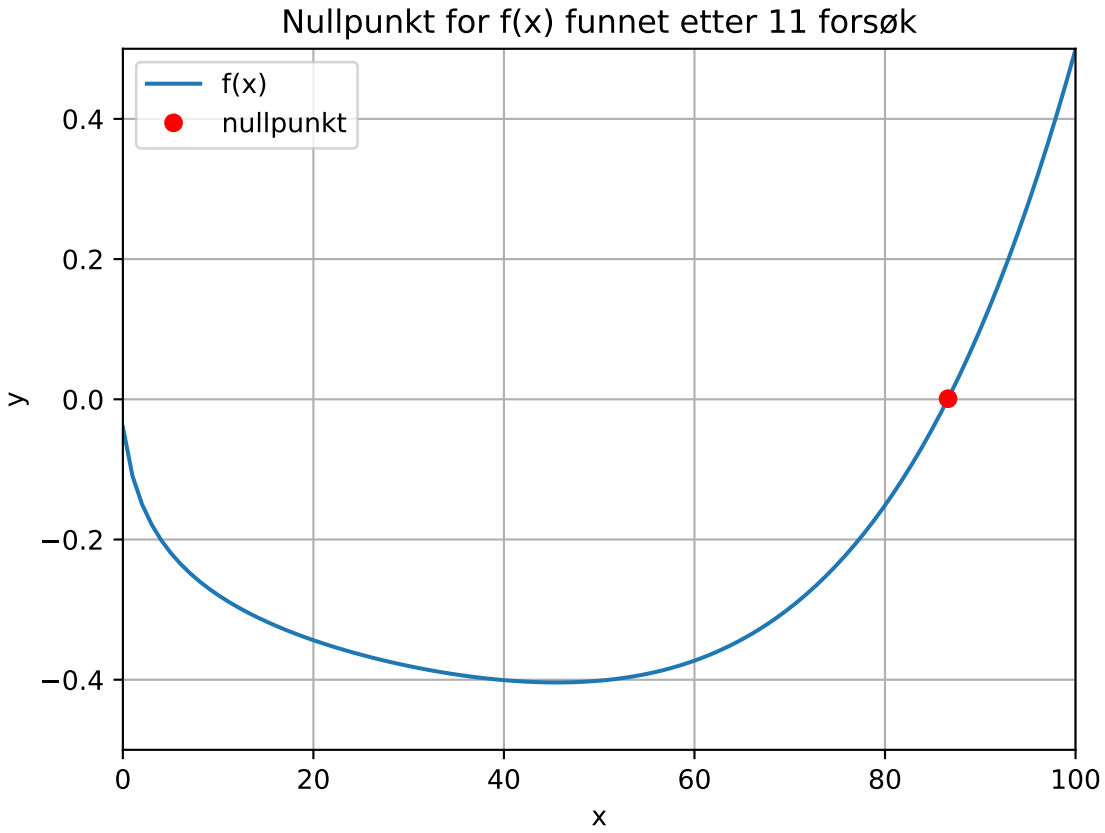
```python

from pylab import log, sign

def f(x):
    return (x/100)**5 - 0.1 * log((x + 1)/100) - 0.5

def g(x):
    return -f(x)

def midtpunkt(a, b):
    return (a+b)/2

maks_tall = 100
min_tall = 0

maks_forsøk = 100
terskel = 0.001

gjettet_riktig = False
forsøk = 1
while not gjettet_riktig and forsøk < maks_forsøk:
    gjett = midtpunkt(min_tall, maks_tall)
    print(f'Jeg gjetter {gjett}!')

    if abs(g(gjett)) < terskel:
        print(f'Jippi, jeg greide det på {forsøk} forsøk!,
            f({gjett})={f(gjett)}!')
        gjettet_riktig = True

    elif sign(g(gjett)) == sign(g(maks_tall)):
        maks_tall = gjett
    else:
        min_tall = gjett

    forsøk += 1

if not gjettet_riktig:
    print(f'Jeg brukte opp alle {maks_forsøk} forsøk uten
       å finne nullpunktet')
    print(f'Det er et sted mellom {min_tall} og {maks_tall
       }')
    print(f'Siste gjett var f({gjett}) = {f(gjett)}')
```

```
Jeg gjetter 50.0!
Jeg gjetter 75.0!
Jeg gjetter 87.5!
Jeg gjetter 81.25!
Jeg gjetter 84.375!
Jeg gjetter 85.9375!
Jeg gjetter 86.71875!
Jeg gjetter 86.328125!
Jeg gjetter 86.5234375!
Jeg gjetter 86.62109375!
Jippi, jeg greide det på 10 forsøk!, f(86.62109375)
   =0.00087616891401254 28!
```

1.3 Bonus: plotte funksjon og løsning

Hvis dere ønsker å visualisere løsningen med et plot er det også fullt mulig å gjøre. Det beste er om dere prøver ut selv, men under er et eksempel på hvordan det kan gjøres:

```python
from pylab import *

def f(x):
    return (x/100)**5 - 0.1 * log((x + 1)/100) - 0.5

def midtpunkt(a, b):
    return (a+b)/2

maks_tall = 100
min_tall = 0

# Plot funksjonen
x_verdier = linspace(min_tall, maks_tall, 100)
plot(x_verdier, f(x_verdier), label='f(x)')
axis([min_tall, maks_tall, -0.5, 0.5])

maks_forsøk = 100
terskel = 0.001

gjettet_riktig = False
forsøk = 1
while not gjettet_riktig and forsøk < maks_forsøk:
    gjett = midtpunkt(min_tall, maks_tall)
    print(f'Jeg gjetter {gjett}!')

    if abs(g(gjett)) < terskel:
        print(f'Jippi, jeg greide det på {forsøk} forsøk!,
            f({gjett})={f(gjett)}!')
        gjettet_riktig = True

    elif sign(g(gjett)) == sign(g(maks_tall)):
        maks_tall = gjett
    else:
        min_tall = gjett

    forsøk += 1

if not gjettet_riktig:
    print(f'Jeg brukte opp alle {maks_forsøk} forsøk uten
       å finne nullpunktet')
    print(f'Det er et sted mellom {min_tall} og {maks_tall
       }')
    print(f'Siste gjett var f({gjett}) = {f(gjett)}')
```




```
# Plot løsning
plot(gjett, f(gjett), 'ro', label='nullpunkt')
xlabel('x')
ylabel('y')
legend()
grid()
title(f'Nullpunkt for f(x) funnet etter {forsøk} forsøk')
show()
```