Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 6

Å løse likninger

Dette settet handler primært om å løse likninger numerisk. Vi har alt sett på to teknikker for å gjøre dette: Halveringsmetoden og Newtons metode. I tillegg skal vi såvidt se på en tredje metode: Fikspunkt-iterasjoner. I forbindelse med disse metodene introduserer vi ei ny type løkker: `while`-løkker.

Men vi starter med å implementere[1] løsninga av tredjegradslikninger.

Oppgave 1 – Tredjegradslikninga

Vi husker at den generelle andregradslikninga $ax^2 + bx + c = 0$ har løsningene $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. I det komplekse planet vil denne likninga alltid ha to løsninger[2] Også den generelle tredje- og fjerdegradslikninga kan løses med "papir og blyant"[3]. For tredjegradslikninga,

$$ax^3 + bx^2 + cx + d = 0 \quad ,$$

er de tre løsningene gitt ved:

$$x_k = -\frac{1}{3a}\left(b + u_k C + \frac{\Delta_0}{u_k C}\right), \quad k \in \{1, 2, 3\}$$

$$u_1 = 1, \quad u_2 = \frac{-1 + \sqrt{3}i}{2}, \quad u_3 = \frac{-1 - \sqrt{3}i}{2}$$

$$C = \sqrt[3]{\frac{\Delta_1 + \sqrt{\Delta_1^2 - 4\Delta_0^3}}{2}}$$

$$\Delta_0 = b^2 - 3ac$$

$$\Delta_1 = 2b^3 - 9abc + 27a^2 d$$

De tre tallene u_1, u_2 og u_3 er de tre løsningene av likninga $u^3 = 1$.

[1] I følge ordboka til Språkrådet, kan dette ordet definieres slik: implementere v2 (fra eng, av lat implere 'gjøre ferdig') innføre, iverksette; i edb: installere og få maskinvare el. program til å virke.

[2] Det kan skje at formelen gir den samme løsninga to ganger.

[3] Litt mer presist: Man kan skrive opp den generelle løsninga ved hjelp av addisjon, divisjon og rot.

a) Lag ei `m`-fil, et skript eller ei funksjonsfil, som regner ut denne løsninga. Om du velger å gjøre det som ei funksjonsfil, skal de fire koeffisientene, a, b, c og d, være argumenter ("input") for funksjonen, og alle de tre løsningene, x_1, x_2 og x_3, skal være funksjonsverdier ("output"). Dersom du velger å implementere løsninga som et skript, kan du enten bruke `input`-funksjonen for å gi koeffisientene fra kommandolinja – slik som i oppgave 3 a) i leksjon 4 – eller du kan helt enkelt tilordne dem i selve skriptet (såkalt "hardkoding").

Bruk `m`-fila for noen ulike sett med koeffisienter som du velger selv, og kontrollér løsningene – for eksempel ved å plotte løsningene sammen med tredjegradspolynomet.

Ekstra: b) Formelen over er ikke uten videre i stand til å håndtere alle mulige tilfeller. Vi ser at den bryter sammen dersom $C = 0$, noe som skjer dersom $\Delta_0 = 0$ og $\Delta_1 < 0$. I dette tilfellet kan formelen "repareres" ved å sette

$$C = \sqrt[3]{\frac{\Delta_1 - \sqrt{\Delta_1^2 - 4 \cdot 0^3}}{2}} = \sqrt[3]{\Delta_1} \quad .$$

Som et særtilfelle har vi at $x_1 = x_2 = x_3 = -\frac{b}{3a}$ når $\Delta_0 = \Delta_1 = 0$. Modifiser `m`-fila slik at den blir i stand til å takle disse tilfellene også.

Oppgave 2 – Halveringsmetoden – igjen

I leksjon 5 løste vi likninga

$$\sqrt{x} = \cos x$$

ved hjelp av halveringsmetoden. Vi skal nå gjøre det samme – men på en litt mer elegant måte. Vi vil innføre en ny ny type løkker: `while`-løkker. Dette er den aller siste av de grunnleggende kommandoene vi skal lære å bruke i dette kurset. Ei `while`-løkke har denne strukturen:

```
while <logisk påstand>
  <utfør kommandoer>
end
```

Ei slik løkke kombinerer egenskapene til både `for` og `if`. Som med `for`, blir sekvensen av kommandoer mellom `while` og `end` utført flere ganger. Men med `for` blir dette antallet bestemt på forhånd. Med ei `while`-løkke vil det være avhengig av den logiske påstanden; kommandoene inne i løkka blir utført så lenge den logiske påstanden er sann. På den måten ligner satsen på `if`. Men når man i en `if`-sats bare går videre når man kommer til `end`, vil man gå tilbake til starten av løkka i ei `while`-løkke[4].

[4] Denne konstruksjonen er alt annet enn "idiotsikker". Om vi skulle komme i skade for å skrive en påstand som alltid er sann, vil løkka "aldri" stanse. Om dette skjer, kan skriptet stoppes fra kommandovinduet ved å trykke `Ctrl+C`.

a) Med utgangspunkt i skriptet du lagde i leksjon 5: Erstatt linja med `for` i skriptet ditt med denne: `while abs(b-a)>1e-3` og kjør skriptet igjen. Funksjonen `abs(x)` gir absoluttverdien, $|x|$, og '`1e-3`' betyr $1 \cdot 10^{-3} = 0.001$. Hvordan kan du med ei lita justering av skriptet få et mer nøyaktig svar?

b) Med utgangspunkt i det samme skriptet: Forsøk å løse denne likninga:

$$2x - \sqrt{x} = 4 \ .$$

Sørg for at feilen ikke blir større enn 10^{-5}.

Denne likninga kan faktisk løses med papir og blyant; løsninga er $x = (1 + \sqrt{33})^2/16$. Forsøk å komme fram til denne løsninga selv, og undersøk om den numeriske løsninga du fant faktisk ligger så nær den eksakte som den skulle.

c) I skriptet ditt har du antageligvis regna ut $f(x)$ flere steder, og det er jo litt upraktisk. Dette kan vi unngå om vi lager oss ei funksjonsfil for $f(x)$ og refererer til denne i skriptet de stedene $f(x)$ skal regnes ut. Prøv å gjøre dette (om du ikke har gjort det alt).

For spesielt interesserte: Man kan også legge inn funksjonen i skriptet uten å lage ei separat funksjonfil. For eksempel kan funksjonen $f(x) = \sqrt{x} - \cos x$ lages slik inni skriptet:

```
Funk=@(x) sqrt(x)-cos(x);
```

Denne funksjonen kan da kalles på samme måte som om det var ei funksjonsfil med navnet `Funk.m`.

d) Forsøk å finne ut hvor mange ganger `while`-løkka måtte gå, hvor mange *iterasjoner* vi måtte gjøre, for å få svaret med nøyaktighet 10^{-5}. Dette kan du gjøre ved å innføre en ny variabel som starter med å være 0 og som øker med én for hver iterasjon.

For ordens skyld: Når det gjelder halveringsmetoden, er sammenhengen mellom nøyaktigheten og antall halveringer såpass enkel at man lett kan regne ut hvor mange iterasjoner man trenger for en gitt nøyaktighet. Om man gjør det, kan man like godt bruke ei `for`-løkke som ei `while`-løkke. Med Newtons metode, derimot, er der ingen enkel sammenheng mellom antall iterasjoner og nøyaktighet.

Oppgave 3 – Newtons metode (og litt til)

Newtons metode går ut på å løse likninga $f(x) = 0$ ved å starte med å gjette på en x-verdi som tror ligger i nærheten av ei løsning. Denne kaller man x_0. Så

itererer man på dette uttrykket:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad .$$

Dette gjør man til x_{n+1} og x_n er tilnærma like.

a) Lag et skript som implementerer denne metoden for likninga $\sqrt{x} = \cos x$. Gjør dette både som ei `for`-løkke der du selv bestemmer antall iterasjoner og som ei `while`-løkke der du bestemmer nøyaktigheten.

   Hva er fordelen med å gjøre dette med ei `while`-løkke?

b) Hvor mange iterasjoner trenger du for få et svar med en feil som er mindre enn 10^{-5} ? Har halveringsmetoden noen fordeler framfor Newtons metode?

c) Bruk Newtons metode til å løse likninga fra oppgave 2 b) – med samme nøyaktighet. Stemmer løsninga?

Om vi kan skrive likninga vår på forma

$$x = g(x) \quad ,$$

ligger ting til rette for å kunne bruke en tredje iterativ metode for å løse likninga. Som med Newtons metode starter vi med å "tippe" på en x_0. Så itererer vi på denne måten:

$$x_{n+1} = g(x_n)$$

d) Hvorfor har vi funnet ei tilnærma løsning hvis $x_{n+1} \approx x_n$?

e) Implementér denne metoden og forsøk å bruke den til å løse de samme to likningene som vi løste med halveringsmetoden og Newtons metode. Merk: Ofte kan ei likning skrives på forma $x = g(x)$ på flere måter. I så fall kan det godt tenkes at metoden fungerer for én variant men ikke for andre.

Ekstra-oppgave: Bursdag samme dag

Vi antar at alle fødselsdagsdatoer er like sannsynlige. Hvor mange elever må det minst være i en gjennomsnittlig skoleklasse for at der sannsynligvis er noen med bursdag samme dag?

Litt starthjelp: Elev nummer 1 har bursdag en eller annen dato. Sannsynligheten for at elev nummer 2 har bursdag en annen dag, er 364/365. Elev nummer 3 har 2 dager mindre å velge i; sannsynligheten for at denne eleven har bursdag på en "ledig" dato er 363/365. Sannsynligheten for at alle tre har bursdag på ulike datoer, blir da produktet

$$\frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \, .$$