Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 4

Intervallhalveringsmetoden – med mer

I dette settet skal vi jobbe litt videre med `for`-løkker. Til slutt skal vi implementere[1] en teknikk som er i stand til å løse ligninger som er uløselige med papir og blyant. I denne sammenhengen introduserer vi et nytt begrep mot slutten av leksjonen: `while`-løkker.

Vi minner, som vanlig, om at det forutsettes at de tidligere oppgavesettene er gjort.

Oppgave 1 – Fakultetfunksjonen

I matematikk er fakultetfunksjonen definert slik:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n \quad .$$

For eksempel er $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$. Som vi ser, er denne funksjonen i utgangspunktet bare definert for naturlige tall, altså positive heltall. Men man har også valgt å definere 0! til å være 1.

a) Lag ei funksjonfil som implementerer fakultetsfunksjonen. Kontrollér at du får at $4! = 24$. Hva blir 10! og 20!?

   (Dette kan gjøres på flere måter; prøv å gjøre det ved å bruke ei `for`-løkke.)

b) I MATLAB ligger faktisk fakultets-funksjonen inne fra før. Den heter `factorial` og kan ta vektor-argument – ikke bare skalarer. I tillegg finnes det en funksjon som "generaliserer" fakultetsfunksjonen slik at den kan ta reelle tall, ikke bare naturlige tall, som argumenter. Den heter $\gamma(x)$, i MATLAB: `gamma`, og har egenskapen $\gamma(n+1) = n!$ når n er et naturlig tall[2].

[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] Navnet på bokstaven γ uttales "gamma". Det er en gresk g.

For $x \in [0, 5]$ og $n \in \{1, 2, 3, 4, 5\}$, plott $\gamma(x+1)$ og $n!$ sammen og kontrollér at de blir like når x er et heltall. Plott $n!$ uten linjer mellom punktene, for eksempel som røde sirkler.

c) Vi ser at fakultets-funksjonen (og γ-funksjonen) vokser veldig fort. Man kan jo lure på hva som vokser fortest av denne og eksponentialfunksjonen. For å få en viss idé: Plott brøken $e^x/\gamma(x+1)$ med en maksimalverdi for x som er passe høy. Hvilken funksjon ser ut til å vokse raskest? For hvilken x-verdi ser brøken ut til å være størst?

Oppgave 2 – Intevallhalveringsmetoden

I denne oppgava skal vi løse likninga

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

a) Hvorfor trenger vi MATLAB, kalkulator eller noe lignende for å kunne løse denne likninga?

b) Plott grafene til $\sqrt{x}$ og $\cos x$ sammen. Hvor mange løsninger ser likninga ut til å ha? Finner du en tilnærma verdi for en løsning?

c) Forklar hvorfor likninga $\sqrt{x} = \cos x$ kan skrives på forma $f(x) = 0$; hva blir $f(x)$?

Forklar også hvorfor $f(x)$ må ha minst ett nullpunkt intervallet $[0, \pi/2]$.

d) Finn en tilnærma verdi for nullpunktet (der er bare ett) ved å følge denne algoritmen i kommandovinduet i MATLAB:
1) Bestem to tall a og b slik at $a < b$ og $f(a)$ og $f(b)$ har ulike fortegn.
2) La c være midpunktet mellom a og b og regn ut $f(c)$.
3a) Dersom $f(a) \cdot f(c)$ er negativ ($f(a)$ og $f(c)$ har motsatt fortegn), la c bli din nye b (høgre grense).
3b) Dersom $f(a) \cdot f(c)$ er positiv ($f(a)$ og $f(c)$ har samme fortegn), la c bli din nye a (venstre grense).
4) Gå tilbake til punkt 2 og gjenta flere ganger helt til $b - a$ er liten nok (du bestemmer selv hva som kvalifiserer til "liten nok").
5) La løsninga være midtpunktet mellom a og b

e) Ett av poengene med deloppgave e) er å illustrere hva en *algoritme* er. Et annet poeng er å gjøre det tydelig hvor kjedelig det er å gjøre slike gjentagelser for hånd. Vi håper dette illustrerer hvorfor det kan være nyttig å skive små programmer, eller skript, av og til. Prøv å skrive et skript som impelenterer denne algoritmen, intervallhalveringsmetoden, for dette problemet. Bruk ei `for`-løkke og bestem deg på forhånd hvor mange ganger denne skal kjøre.

Hint: `for`-løkka kan for eksempel starte slik: "`for i=1:10`". Her har vi bestemt oss for å gjenta prosessen 10 ganger. Det som er litt spesielt her,

er at vi ikke får bruk for selve indeksen, "`i`", i dette tilfellet; den er der bare for å telle gjentagelsene (eller *iterasjonene*). Inne i selve `for`-løkka vil du få bruk for en `if`/`else`-sats.

Kjør skriptet og finn en verdi for nullpunktet. Lag et plott av $f(x)$ der du markerer denne nullpunkts-kandidaten. Ser svaret ut til å stemme?

Vi vil nå innføre en ny kommando – en ny type løkker: `while`. Dette er en av de aller siste 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[3].

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

g) 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.

h) 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.

Intervallhalveringsmetoden er den mest kompliserte algorimten vi skal implementere i dette kurset.

[3] 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`.