Matematikk 1000

Øvingsoppgaver i numerikk – leksjon 4

m-filer

I denne øvinga skal vi lære oss å lage `m`-filer – små tekstfiler som vi bruker i MATLAB-sammenheng. Der finst to typer `m`-filer: Funksjonsfiler og skript.

Funksjonsfiler gjør oss i stand til å definere våre egne funksjoner og kalle dem på samme måte som de funsjonene som ligger inne fra før (`sin`, `log` etc.). Et skript kan vi se på som et lite program – altså en sekvens av kommandoer.

Den siste typen, skript, er den viktigste.

Det forutsettes at du har gjort de tidligere oppgavesettene.

Oppgave 1 – Funksjonsfiler

Vi skal nå se hvordan vi kan lage våre egne funksjoner i MATLAB. Disse kan brukes på samme måte som de funksjonene MATLAB har inne fra før. Vi skal ta utgangspunkt i funksjonen $f(x) = \sin(2x) - x^2$. Først åpner vi en *teksteditor*. Det mest naturlige er nok å bruke MATLAB sin egen. Den kan åpnes ved å trykke på "New Script"-knappen oppe til venstre. Du får da opp et vindu hvor du kan skrive inn tekst.

a) Skriv av følgende i teksteditoren:

```
function F=FunksjonenMin(x)

% Funksjonen f(x)=sin(2x) - x^2.
% Funksjonen tar bare skalarer som input.

F=sin(2*x)-x^2;
```

Lagre fila i en passende mappe og kall den "`FunksjonenMin.m`". (Teksteditoren til MATLAB vil automatisk gi den "etternavnet" '.m'.) Om du er i den samme mappa som denne fila i MATLAB, kan du regne ut funksjoneverdier for denne funksjonen på akkurat samme måte som for de

elementære funksjonene vi har sett på før; $f(5)$ kan for eksempel finnes ved å skrive '`» FunksjonenMin(5)`' i kommandovinduet.

Regn ut $f(x)$ for noen x-verdier du velger selv på denne måten.

Man kan lage funksjonsfiler på flere måter. Men to ting kommer man ikke utenom:
1) Den første linja *skal* ha denne strukturen:

```
function <Navn på ut-variabel>=<Navn på funksjon>(<Liste med inn-variabler>)
```

I eksempelet over, ser vi at ut-variabelen har fått navnet `F` og at funksjonen har fått navnet `FunksjonenMin`. Dette navnet bør være det samme som navnet på fila (utenom '`.m`'). Til sist, inni en parantes, lister man opp de variablene som skal inn. I dette tilfellet er det bare én, `x`, men det er ingenting i veien for at det kan være flere[1].
2) Til sist i funksjonsfila *skal* ut-variabelen, `F` i vårt tilfelle, ha blitt tilordna.

b) Hva skjer hvis du skriver '`» help FunksjonenMin`'?

c) Hvis vi skal *plotte* en funksjon gitt ved ei funksjonsfil, er det en stor fordel om funksjonsfila kan ta vektorer som inn-variabel på en slik måte at den gir som svar en vektor (ut-variabel) som består av funksjonsverdien av hvert element i vektoren vi gir inn. Vi har allerede sett at for eksempel `sin` – funksjonen har denne egenskapen; hvis `x=[1 2 3]`, vil '`» sin(x)`' gi vektoren $[\sin 1, \sin 2, \sin 3]$ som svar.

   Med ei ørlita endring kan funksjonen over bli i stand til dette. Gjør denne endringa og plott funksjonen. Velg selv hvilket intervall argumentet skal gå over og hvilken steglengde du vil ha.

d) Lag ei funksjonsfil for en eller annen elementær funksjon du velger selv og bruk denne til å plotte grafen til funksjonen.

Oppgave 2 – Delt forskrift – `if`-satser

Vi har sett av funksjoner kan bli gitt med *delt forskrift* – altså at funksjonen er gitt ved ett uttrykk for visse x-verdier og et annet uttrykk for andre x-verdier. Det kan også være snakk om flere enn to ulike uttrykk. I denne oppgava skal vi se hvordan vi kan lage funksjonsfiler for, eller *implementere*, slike funksjoner. Men først skal vi se litt på logiske utsagn i MATLAB.

a) Som tidligere nevnt, betyr '`=`' *tilordning* – ikke likhet – i MATLAB. Likhet skrives slik: '`==`'. Større enn og mindre enn, derimot skrives som normalt.

[1] Det kan gjerne være flere ut-variable også.

'Er ulik', og ikke-strenge ulikheter kan skrives slik: '~=', '<=' og '>='. Forsøk å skrive noen sanne og noen usanne logiske påstander i kommandovinduet, som for eksempel $3 = 2$, $-1 < 0$, og $-1 \neq 0$, og se hva du får til svar. Forsøk gjerne å kombinere med *eller* og *og* også. 'Eller' kan skrives som '|' og 'og' kan skrives som '&'. For eksempel kan påstanden $x \in [-2, 1]$ skrives slik: `x>=-2 & x<=1`.

I en slik sammenheng, hva betyr "0" og "1"?

b) Skriv av denne funksjonsfila i editoren din, lagre den og kall den '`DeltForskrift.m`':

```
function F=DeltForskrift(x)

% Her bør det stå noe om hvilken funksjon det er snakk om

if x<2
  F=cos(pi*x)+2;
else
  F=x^2-2;
end
```

Regn ut noen funksjonverdiene for noen x-verdier du velger selv. Hvilken funksjon er dette ei implementering av? Merk at denne fuksjonsfila bare tar skalarer (tall) som input (ikke vektorer)

c) Funksjonsfila over har en alvorlig mangel: Den tar bare skalarer som variable; vi kan ikke gi vektorer som input. Om vi lar `x` være en vektor og kaller funksjonen med denne, `» DeltForkrift(x)`, vil bare det siste elementet i vektoren bli brukt i `if`-satsen. Men dette problemet kan løses. En mulig løsning er å lage ei `for`-løkke som går over alle elementene i `x`-vektoren. (Dette har vi ikke gått gjennom i dette kurset enda.) En annen er å bruke *logiske variable*. Hva skjer når du skriver?

```
>> x=1:10;
>> x>5
```

Hvordan kan dette brukes til å lage ei funksjonfil for funksjonen fra b) som kan ta vektor-input?

Litt mer om `if`-satser:

De enkleste `if`-satsene har denne strukturen

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

Kommandoen eller kommandoene vil bare bli utført hvis den logiske påstanden er sann. Legg merke til at vi har gjort et lite "innhopp" i teksten i linja mellom `if` og `end`. Dette gjør funksjonsfila, eller hva det måtte være, mye mer oversiktlig. Så vi anbefaler at du også legger deg til denne vanen.

Denne strukturen er også mye brukt:

```
if <logisk påstand>
  <utfør kommandoer>
else
  <utfør andre kommandoer>
end
```

Vi kjenner denne strukturen igjen fra funksjonsfila over. Her blir altså "andre kommandoer" utført i stedet for "kommandoer" dersom den logiske påstanden er feil. I eksempelet over er "andre kommandoer" tilordninga '`F=x ^ 2-2;`', mens "kommandoer" er tilordninga '`F=cos(pi*x)+2;`'.

En tredje vanlig struktur er denne:

```
if <logisk påstand>
  <utfør kommandoer>
elseif <annen logisk påstand>
  <utfør andre kommandoer>
else
  <utfør et tredje sett av kommandoer>
end
```

Her kan man bygge på med så mange `elseif`-satser man bar vil etter hverandre.

Oppgave 3 – Hva gjør disse skriptene?

Vi så et eksempel på et skript i oppgave 5 i forrige leksjon. Nedenfor har vi gitt to andre eksempler på skript. I begge tilfeller har vi brukt funksjonen `input`, som vi kan bruke til å gi verdier fra kommandovinduet når vi kjører skriptet. Skriv av eller kopiér disse skriptene i teksteditoren og kjør skriptene. Forstår du hva de gjør?

a) Skriptet "`ABCformel.m`":

```
a=input('Gi verdien for a: ');
b=input('Gi verdien for b: ');
c=input('Gi verdien for c: ');

x1=(-b-sqrt(b^2-4*a*c))/(2*a)
x2=(-b+sqrt(b^2-4*a*c))/(2*a)
```

b) Skriptet "`TypeTall.m`":

```
if x<0
  'Tallet er negativt'
else
  'Tallet er positivt'
end

if round(x)~=x
  'Tallet er ikke et heltall'
else
  if round(x/2)==x/2
    'Tallet er et partall'
  else
    'Tallet er et oddetall'
  end
end
```

Tips: Om du lurer på hva funksjonen `round` gjør, skriv `» help round` i kommandovinduet.

c) Begge skriptene over har en alvorlig mangel: De er ikke **kommenterte**. Å *kommentere* et skript eller ei funksjonsfil vil si å legge til tekst som forklarer hva det gjør. Dette gjør vi både i starten og underveis i koden. For eksempel kan en kommentert versjon av skriptet fra oppgave a) se slik ut:

```
1 % Skript som løser likninga a x^2 + b x + c = 0
2
3 % Gir verdiene på a, b og c:
```

```
4  a=input('Gi verdien for a: ');
5  b=input('Gi verdien for b: ');
6  c=input('Gi verdien for c: ');
7
8  % Regner ut løsningene og skriver dem til skjerm:
9  x1=(-b-sqrt(b^2-4*a*c))/(2*a)
10 x2=(-b+sqrt(b^2-4*a*c))/(2*a)
```

Vi bruker altså prosent-tegnet, %, for å legge til kommentarer. Det som står bak %, blir ignorert av MATLAB, men ikke nødvendigvis av en person som leser programmet. Her har vi også lagt til linjenummer for å lettere kunne referere til ulike deler av koden. (Disse er selvsagt ikke med i selve skriptet.)

Gjør det samme med skriptet fra oppgave b) – altså legg til forklarende kommentarer.

d) **Ekstra:** For skriptet i a), klarer du å bruke `if`-satser til å lage en mer "idiotsikker" versjon av skriptet? Kanskje du kan sjekke at a er ulik null – eller at andregradslikninga har reelle løsninger? Om vi kan unngå å skrive to identiske løsninger til skjerm, er vel også det en fordel... Alle disse justeringene kan gjøres ved hjelp av `if`-satser av typen beskrevet mellom oppgave 2 og 3. I tillegg får du bruk for kommandoen `return`, som avslutter skriptet selv om man ikke har kommet til enden av det.

Litt om forskjellen på funksjonsfiler og skript: Ei funksjonsfil skal, som vi har sett, ha en viss struktur. Den skal for eksempel alltid starte med "`function`". I tillegg skal vi alltid gi (minst) et tall (eller en vektor) inn til ei funksjonsfil. Dette gjør vi, som vi har sett, ved å skrive dette tallet eller denne vektoren i parantes etter navnet på funksjonen i kommandoviunduet. (`» FunksjonenMin(5)`.)

Der er ikke noen spesiell struktur for skript; et skript er bare ei oppramsing av kommandoer. Og når vi kjører skriptet, skal det ikke stå noe i noen parantes etter navnet på skriptet.

I tillegg: Eventuelle variable som blir tilordna i ei funksjonfil, vil bare være å finne i funksjonfila; de vil ikke dukke opp i minnet til MATLAB. Tilordninger som blir gjort i et skript, derimot, vil også dukke opp i minnet til MATLAB når skriptet blir kjørt.