EDA017 – Beräkningsprogrammering

Föreläsning 1

Christian.Soderberg@cs.lth.se

20 mars 2013

Vad vi skall lära oss under detta pass

- Grundläggande beräkningar i OCTAVE/MATLAB.
- Vektorer och matriser.
- Standardfunktionerna, och vad de egentligen gör.
- Att definiera egna funktioner.
- Att plotta grafer.
- Scripts.

OCTAVE/MATLAB

- OCTAVE/MATLAB är gjort för att vi enkelt skall kunna utföra beräkningar.
- Vi har variabler och funktioner som i Java, men det är enklare att hantera dem.
- Vi kan använda OCTAVE/MATLAB *interaktivt*.

Värden och grundläggande funktioner i OCTAVE/MATLAB

- I princip bara ett slags värde – allt är komplexa matriser.
- De mest grundläggande funktionerna är +, -, *, / och ^ – dessa fungerar alla som matris-operationer.
- Det finns några enkla funktioner för att generera särskilda matriser: `zeros`, `ones`, `eye`.

Vektorer som ’förvaringsutrymme’

- Man använder ofta vektorer för att spara följder av värden, ungefär som vi använder listor (eller vektorer) i Java.
- Vi kan göra det på flera sätt:
  - `:`-notation: låter oss räkna upp med ett givet steg
  - `linspace`: ger ett antal värden i ett givet intervall
- *Anledningen att de flesta standardoperationerna är elementvisa är att vi vill kunna anropa dem på följder av värden.*
- *När vi gör beräkningar på en följd av värden är det antagligen* `.*`, `./` *och* `.^` *som vi vill använda (*`*`*,* `/` *och* `^` *skulle ge dimensionsfel).*

:-operatorn

`:`-operatorn låter oss räkna upp med ett givet steg:

- `1:2:9` ger $\begin{pmatrix}1 & 3 & 5 & 7 & 9\end{pmatrix}$
- `1:5` ger $\begin{pmatrix}1 & 2 & 3 & 4 & 5\end{pmatrix}$

Matris-operationer är undantaget!

- De operationer vi har sett hittills är ’riktiga’ matrisoperationer, som vi känner dem från Linjär algebra.
- Det finns även mängder av inbyggda funktioner för allt möjligt, som `sqrt`, `exp`, `sin`, … – dessa är dock i allmänhet inte ’riktiga’ matrisoperationer, utan opererar *elementvis* på våra vektorer och matriser.

Elementvisa operationer

- Det är aldrig några problem att addera och subtrahera lika stora vektorer med varandra, operationerna sker elementvis.
- För att få elementvis multiplikation, division och exponentiering måste vi använda `.*`, `./` och `.^`
- Alla standardoperationer, som `exp`, `sin` och `abs`, opererar elementvis (så `*`, `/` och `^` är egentligen undantag).
- När vi använder någon av standardfunktionerna är alltså `.*`, `./` och `.^` de naturliga operatorerna att använda.

`linspace`-funktionen

`linspace` ger ett antal värden i ett givet intervall:

- `linspace(0, 10, 42)` ger 42 värden mellan 0 och 10, jämnt fördelade.
- `linspace(0, 10)` ger 100 värden mellan 0 och 10, jämnt fördelade.

Att definiera egna funktioner

- Det finns mängder av standardfunktioner.
- Vi kan enkelt definiera egna funktioner, ungefär som i Java (fast enklare):
  - Enradsfunktioner: skrivs på en rad, och kan användas för enklare funktioner.
  - Funktionsfiler: skrivs i separata filer, och kan göra mycket mer än enradsfunktionerna (exempelvis loopa).

'Enradsfunktioner'

- `@(x) ...` definierar en funktion i variabeln x.
- Uttrycket

```
@(x) sin(2*pi*x)/x;
```

  ger funktionen som beräknar $\frac{\sin(2\pi x)}{x}$ för ett givet värde x.
- Vi kan lagra en sådan funktion i en variabel, exempelvis:

```
f = @(x) sin(2*pi*x)/x;
```

- Vi kan nu anropa vår funktion som om den vore en standardfunktion:

```
y = f(1.25);
```

- Vi kan ha flera parametrar om vi vill.

`plot`-funktionen

- `plot(y)`: plottar y-värden i x-koordinaterna 1 . . .
- `plot(x,y)`: plottar alla punkter (x_k, y_k), och drar linjer mellan på varandra följande punkter.
- `plot(x,y,ch)`: markerar alla punkter (x_k, y_k) med en given markering.

Scripts

- Vi kan lägga följd kommandon i en ’script-fil’.
- Vi kan köra alla kommandon i scriptet genom att ange namnet på script-filen.
- I våra script kan vi använda `if`-satser, `for`-satser och `while`-satser.
- Vi kan även använda `input`- och `print`-satser, ungefär som i Java.
- Ofta bättre att skriva korta scripts än att använda OCTAVE/MATLAB interaktivt:
  - Vi kan lätt göra om beräkningarna.
  - Vi kan lätt korrigera gamla tilldelningar utan att behöva göra om satser ’för hand’.

`if`-satsen

```
if ...uttryck...
    ...
elseif ...uttryck...
    ...
else
    ...
end
```

- Ingen typ `boolean`, värdet 0 betyder `false`, i princip allt annat är `true`.
- Uttrycken `0 < 1` och `1 == 1` har värdet 1.
- Uttrycken `0 >= 1` och `1 != 1` har värdet 0.
- Även en tom vektor betraktas som `false`.

`while`-loopen

```
while ...uttryck...
    satser
end
```

- Villkoret är falskt om uttryckets värde är 0 (som i `if`-satsen).
- Vi kan avbryta med `break`, och hoppa direkt till nästa varv med `continue`.

`for`-loopen

```
for index = ...vektor...
    satser
end
```

- Räknarvariabeln går igenom samtliga värden i en vektor.
- Vi kan avbryta med `break`, och hoppa direkt till nästa varv med `continue`.