TMV156/TMV155E Inledande matematik E, 2009

MATLAB — FÖRELÄSNINGSANTECKNINGAR OCH ÖVNINGAR

1. Inledning och syfte

Matlab-delen i den här kursen syftar till elementär förståelse av Matlab som prgrammeringsspråk och som "funktionskatalog". Efter genomlidna föreläsningar och genomförda laborationer bör du kunna den syntax som används för att anropa Matlabs färdiga funktioner samt skriva egna enklare sådana. Du bör kunna definera anonyma funktioner och använda dessa för att rita grafer till matematiska funktioner. Viktigt är också att du förstår begränsningarna i grafritningsverktygen och behovet av en matematiskt kunnig användare för att inte dra fel slutsatser då man studerar en med hjälp av Matlab genererad figur. Kursinslaget syftar också till att ge dig kusnkaper om grundläggande programmeringsmetoder. Du bör efter avslutad kurs också kunna leta i Matlabs kataloger efter användbara funktioner samt kunna läsa och förstå hjälpfilerna som finns till dessa. Vi hoppas att du efter denna kurs ska kunna ta till dig information om Matlab från olika källor och kunna utforska miljön på egen hand.

2. Litteratur

Vi kräver inte att du köper någon bok om Matlab för den här kursen men tre böcker som är användbara är

Matlab for Eingineers av Holly More
Matlab - beräkningar inom teknik och naturvetenskap av Per Jönsson samt
Användarhandledning för Matlab av Eva Pärt-Enander och Anders Sjöberg.

Däremot är häftet 'Matematik med Matlab - En handledning' av Jörgen Löfström att betrakta som kurslitteratur. Det hittas som pdf-dokument på kurshemsidan. Vid föreläsningen den 23/9 delades dessutom ut några stenciler ur Jönssons bok. Betrakta även dem liksom desssa sidor samt laborationsbeskrivningarna som kurslitteratur. Det är möjligt att fler stenciler kommer att delas ut under kursens gång. Dessa kommer framför allt innehålla listor med viktiga funktioner som man bör känna till. Vi kommer också att referera till Bill Karlströms kompendium 'Något om Matlab'.

3. Föreläsning 1, 23/9

Under föreläsningen den 23:e september gicks följande saker igenom:

Räkneoperatorer. I synnerhet

```
A.*B
A ./B
A .^3 .
```

Anrop av Matlabs inbyggda funktioner. Invariabler/argument och utvariabler.
Deklaration av variabler, tilldelning, tildelningsoperatorn =.
Att skapa och manipulera matriser/listor.

```
x=x_0:Delta x :x_1;
y=linspace(x0,x1,N);
x(1,4);
x(4); %(!).
```

Grafritning med plot-kommandot.

```
figure(1)
plot(x,z,'-or')
figure(2)
...
```

Date: 2008–09–29, Fredrik Lindgren, Matematiska vetenskaper, Chalmers tekniska högskola.

Olika typer. Stängar, dubblar (doubles), funktionshandtag.

Inför föreläsningen den 1:a oktober är det bra om du har hunnit repetera detta till exempel genom att göra repetitionsövningarna nedan. Ni får ca tio minuter i början på föreläsning två till att arbeta med och fråga om dessa uppgifter.

3.1. **Repetitionsövningar.**

3.1.1. *Enkel plotövning.*

Generera hundra punkter jämt fördelade mellan $-\pi$ och π.
Beräkna cosinus och sinus för dessa funktioner och plotta graferna för funktionerna i samma figurfönster men med olika färger.
Deklarera en anonym funktion f som beräknar produkten av cosinus och sinus och plotta även dess graf.

3.1.2. *Brownsk rörelse.*

Generera en matris A med 10 rader och 1000 kolumner med normalfördelade slumptal. Sådana genereras med kommandot randn. Skriv "help randn" i kommandofönstret.
Sätt alla element i A:s första rad till 1.
Sätt de hundra första elemnten i den sista raden till 2.
Deklarera en variabel x och tilldela den värdet av den andra raden i A.
Plotta x mot vektorindexet genom att skriva 'plot(x)' i kommandofönstret.
Deklarera en variabel y och tilldela den värdet i rad tre i A.
Skapa en ny variabel z och omdefinera y som z =cumsum(x) respektive y =cumsum(y) (help cumsum).
Plotta z i samma figur som x och plotta z mot y i ett nytt fönster. Är det senare grafen av en funktion?

4. Föreläsning 2, 1/10:Programmering, m-filer och funktionsfiler

Den här föreläsningen börjar vi med att titta på hur plot-verktyget gör när det ritar. Vi tittar också på hur matlabs help-kommando fungerar samt *lookfor* och *doc*. Sedan skriver vi ett litet program i en *skriptfil* (m-fil). Vi nämner några ord om *sekventiella* programmeringsspråk och försöker förstå vad det innebär. Därefter gör vi vår egen *funktionsfil* och försöker förstå syntaxen för dessa och hur man får en funktion att ta emot olika antal *inargument* och lämna ifrån sig olika antal *utargument*. Slutligen så diskuterar vi skillnaden mellan funktions- och skriptfiler. Här blir variablers *räckvidd* ett viktigt begrepp. Vad är en *lokal* respektive *global* variabel?

4.1. **Övningsuppgifter.**

4.1.1. *Hjälp!* Skriv

```
help plot
```

och studera texten som skrivs ut i kommandofönstret. Ha framför allt klart för dig vad skillnaden mellan

```
plot(x),
plot(x,y)
```

respektive

```
plot(x,y,s)
```

är. Titta igenom listorna med olika typer av färgenr, prickar och linjer och se till att du förstår hur dessa ska användas. Tycker du att du har tid över kan du senare studera första stycket i hjälptexten till funktionen fzero.

4.1.2. *Hitta funktioner som innehåller sinus-relaterat material.* Se vad som finns genom att skriva

```
lookfor sine.
```

Titta på hjälpfilerna för någon av funktionerna som dyker upp. Hitta på något eget som du vill leta reda på på samma sätt.

4.1.3. *Utförligare hjälp.* Skriv

```
doc plot
```

och se vad som händer. Titta på dokumentationen för någon mer Matlab-funktion.

4.1.4. *Rätt filkatalog.* Skapa en katalog för den här kursen. Skapa i den ytterligare en katalog med namnet Matlab. Gör i denna katalog ytterligare en katalog som du kallar Intro. Du kan skapa kataloger direkt i Matlabs kommandofönster med kommandot 'mkdir katalognamn'.

4.1.5. *Brownsk rörelse revisited.* Gör om uppgiften för Brownsk rörelse men i en m-fil. Kalla filen minBrown.m och se till att den sparas i katalogen Intro som du gjorde ovan.

4.1.6. *Trigonometriska funktioner. Igen.* Gör funktionen cos(x).*sin(x) som en funktionsfil.

4.1.7. *Elemetvis operation.* Skriv en funktionsfil element.m som utför elementvis addition, subtraktion, multiplikation och division mellan två vektorer och returnerar alla dessa svar i olika utvariabler. Anropa dessa med vektorerna $x = [1\,2\,3\,4\,5\,6\,7\,8\,9\,10]$ samt $y = [2\,3\,4\,5\,6\,7\,8\,9\,10\,11]$ och plotta alla resultaten i samma figur. Deklarera x, y i en skriptfil från vilken anropet av element.m och plot görs.

5. Föreläsning 3, 1/10: Programmering - snurror och villkor

Väldigt mycket programmering bygger på att upprepa samma procedur ett på förhand givet eller okänt antal gånger. I det första fallet skulle man ju kunna skriva satsen som ska upprepas så många gånger som den ska upprepas i sitt program. Det kan dock bli jobbigt och minneskrävande, i synnerhet om det är något som ska upprepas sisådär en miljon gånger. För att hantera detta problem så har man utvecklat for-satsen. I Matlab följer den syntaxen

```
for k=[vektor]
   (Programkod)
end
```

och i synnerhet

```
for k=1:N
   (Programkod)
end
```

där N är något heltal. Vi ska förklara detta under föreläsningen. Ofta vill man upprepa en procedur tills det att något villkor är uppfyllt, ett villkor som det inte går att räkna ut i förväg när det är uppfyllt. Då använder man så kallade while-satser. De följer i Matlab syntaxen

```
while (logisk sats)
  (Programkod)
end
```

där (logisk sats) i allmänhet innehåller någon variabel som ändras i (Programkod). Om så inte är fallet håller while-loopen på i all oändlighet (åja...).
Vad är då en logisk sats? Det är ett påstående som är sant eller falskt. Givet två variabler x och y kan vi ställa oss frågan om de är doubles genom att påstå att så är fallet med kommandot

```
bool=isa(x,'double')
```

Om x är en double så får variabeln *bool* värdet 1. Det här talet är en ny typ som kallas för logisk variabel. Logiska variabler har antingen värdet 1 som betyder sant eller värdet 0 som betyder falskt. Om x är en double så kommer alltså variabeln *bool* vara av typen logisk variabel och ha värdet 1. Skriver vi nu

```
isa(bool,'double')
```

så kommer Matlab returnera en nolla. Variabeln *bool* är ju ingen double utan en logisk variabel. Påståendet att *bool* skulle vara en double är alltså falskt. Förutom i while-loopar så är logiska satser viktiga i så kallade if-satser. Dessa kan i Matalab se ut som följer.

```
if (logisk sats 1)
   (Programkod 1)
elseif (logisk sats 2)
   (Programkod 2)
else
   (Programkod 3)
end
```

Om (logisk sats 1) är sann så utförs (Programkod 1) och resten hoppas över. Om den är falsk så testas (logisk sats 2). Om den däremot är sann så utförs (Programkod 2) medan (Programkod 3) hoppas över. Om både (logisk sats 1) och (logisk sats 2) är falska så utförs (Programkod 3). Man kan ta bort både

```
elseif (logisk sats 2)
   (Programkod 2)
```

och/eller

```
else
   (Programkod 3)
```

i koden ovan. Vad tror du händer då?
Antag nu att x och y ovan är doubles. Då kanske vi vill jämföra dem. Är de lika stora? Vilken är störst? Det gör man med jämförelseoperatorer. Uttrycket $x == y$ innebär att man påstår att x och y är samma tal. Matlab returnerar en logisk etta om det är fallet och en nolla annars. Det torde inte vara så svårt att räkna ut vad

```
x<y,
x>y,
x<=y
```

respektive

```
x>=y
```

innebär. Uttrycket

```
x~=y
```

betyder ”x är inte lika med y” och är alltså sant om $x == y$ är falskt och vise versa. Notera skillnaden mellan tilldelningsoperatorn = och jämförelseoperatorn ==. Vad gör följande rad?

```
z=x==y;
```

Ibland vill man att flera påståenden ska vara sanna eller falska samtidigt. Alternativt att något av flera ska vara det. Påståendet

```
isa(x,'double') & isa(y,'double')
```

är sant om både x och y är doubles. Med två påståenden A och B som vardera kan vara sant eller falskt så gäller att $A|B$ är sant om något av påståendena A eller B är sanna $AxorB$ är sant om någon av de är sanna men inte om båda är det och $\tilde{}A$ är sant om A är falskt. Matlab använder sig alltså av de *logiska operatorerna* & (och), | (eller), *xor* (exklusivt eller) och ~ (icke). Studera följande exempel.

```
if isa(x,'double') & isa(y,'double')
   if x==y
      svar='x och y är lika stora';
   elseif x<y
      svar='y är större än x';
   else
      svar='x är större än y'
   end
else
   warning('x och y måste vara dubblar')
end
```

Vad gör programsnutten?

5.1. **Övningar.**

5.1.1. *Enkel for-loop.* Skriv en for-loop som räknar ut

$$\Sigma_{i=1}^{100} i\,.$$

5.1.2. *Enkel if-sats.* Skriv en skriptfil som testar om variabeln $fraga$ innehåller strängen hej. Låt den då sätta variabeln *svar* till strängen Hej!. Om den inte innehåller strängen hej så bör den testa om den innehåller strängen *hej då*. I detta fall bör *svar* tilldelas strängen *Hej då!*. Om $fraga$ inte har något av dessa värden bör *svar* tilldelas värdet *Jag förstår inte vad du säger!*

5.1.3. *Vad gör följande while-loop?*

```
s=2;
j=1;
while s<100
   s=s^2;
   j=j+1;
end
j=j
```

5.1.4. *Skytt med skakig hand.* En äldre skytt som är bra på att sikta men har blivit lite darrhänt kanske kan antas träffa en skyttetavlas origo med normalfördelade x- och y-koordinater. Generera i en for-loop hundra styckna skott med hjälp av *randn* och testa hur många som träffar centrum (radie 1), utanför centrum men innanför cirkeln av radie 2 etc upp till 5. Plotta alla träffpunkter. Fundera över hur du grafiskt kan visa hur många som träffar var.

5.1.5. *Skakig skytt igen.* Skriv ett program som ovan men där en while-loop avbryts om skytten träffar centrum. Låt programmet hålla reda på hur många skott som behövs innan det händer.