Introduktion i MATLAB

Hampus Malmberg
Jesper Karlsson
Sven Jacobsson

30 augusti 2011

Förord

Dokument behandlar grundläggande användning av MATLAB och skall ge viss förståelse i programmeringstänk och hur MATLAB kan tillämpas på att lösa matematiska problem, för att på så sätt förbereda för vidare användning av verktyget i fortsatta kurser.

Materialet kräver ingen förkunskap inom MATLAB eller annan programmering. Dock förutsätts viss kunskap i matematik.

Kontakt

Vi som har håller i introduktionen och skrivit undervisningmaterialet är:

Hampus Malmberg *<hammal@student.chalmers.se>*
Jesper Karlsson *<jeskarl@student.chalmers.se>*
Sven Jacobsson *<jsven@student.chalmers.se>*

Om ni har en fråga om kursens innehåll är ni alltid välkomna att höra av er till någon av oss. Om ni har en fråga som rör innehållet, eller upptäckt ett fel så hör av er till: *<jsven@student.chalmers.se>*

Övningsuppgifter

Utöver detta dokument finns också ett häfte med övningsuppgifter och lösningsförslag designade att passa detta kompendiums innehåll och svårhetsgrad.

Innehåll

1 Inledning

1.1 Kort om MATLAB

MATLAB är ett oerhört kraftigt verktyg för att utföra olika sorters beräkningar. Förutom de egenskaper som finns i traditionella programmeringsspråk som C++ och Java, har MATLAB också inbyggda funktioner för att lösa numeriska problem inom linjär algebra och matematisk analys.

Namnet MATLAB står för Matrix Laboratory vilket syftar på att variablerna i språket hanteras som matriser.
Kännedom om vektorer och matriser anses inte som ett förkunskapskrav för denna kurs och kommer således inte att behandlas mer än att nämnas. Mer kännedom om dessa begrepp kommer ges senare i kursen *Inledande matematik* och framförallt *Linjär algebra*

1.2 Användningsområden

Språket används flitigt av ingenjörer i industrin inom områden som: matematiska beräkningar, simulering, kommunikation, bildbehandling, finansiell modellering, etc. Listan skulle kunna göras hur långs som helst och ovanstående områden är bara ett kort utdrag ur MATLABs alla tillämpningsområden.

Dessutom kommer ni under stora delar av er utbildning stöta på verktyget och ni kommer ha enormt stor nytta av att kunna använda det.

1.3 Arbetsmiljö

I bilden ovan ser du utvecklingsmiljön (desktop) i MATLAB. Här hittar vi en mängd verktyg som förenklar vårt arbete:

- **Command window**
  Här skriver man in sina kommandon.
- **Workspace browser**
  Visa och ändra dina variabler
- **Command history**
  Se dina tidigare kommandon
- **Current folder**
  Se i vilken mapp du befinner dig. Viktigt att veta då man kör egna program.
- **Details**
  Ger kort information om valt objekt
- **Start**
  Härifrån kan du komma åt verktyg och dokumentation

En stor fördel med MATLAB är att du interaktivt kan använda programmet, det vill säga du skriver ett kommando och MATLAB svarar omedelbart, med till exempel en graf.

Exempel

Då vi startar MATLAB möts vi av utvecklingsmiljön där ett av verktygen är kommandofönstret (*command window*). Om vi där skriver:

```
>> 5*4+7
```

och trycker på *Enter* kommer programmet svara med:

```
ans =

    27
```

Vilket naturligtvis är den rätta lösningen till talet.

1.4 Installera MATLAB på egen dator

Alla studenter på Chalmers har tillgång till MATLAB som gratis går att ladda hem via studentportalen där man loggar in med sitt CID. För att komma åt installationsfilerna går man in på följande sida:

http://studentfile.portal.chalmers.se/library/Matlab/

och klickar sig vidare till Chalmers filserver där MATLAB finns att ladda hem till Windows, Mac och Linux. Dokumentation om hur installationen går till hämtas också på filservern.

2 Grundläggande hantering av värden

2.1 Deklarera variabler

En variabel i MATLAB är alltid en eller flera bokstäver eventuellt följd av siffror som tilldelats ett värde.
A, *b*, *uranium238*, *eV* och *Boltzman* är alla exempel på tillåtna variabelnamn. Lägg märke till att MATLAB skiljer på små och stora bokstäver.

Exempel

Om vi i kommandofönstret skriver:

```
>>  a = 5
```

svarar MATLAB med:

```
a =

     5
```

Vi har alltså sparat värdet 5 till variabeln a. Om vi senare vill hämta värdet skriver vi i kommandofönstret bara:

```
>>  a
```

Vilket resulterar i att MATLAB skriver ut värdet 5.

Exempel

Om vi enligt tidigare deklarerat värdet 5 till variabeln a och om variabeln b har värdet 3 innebär

```
>>  c = a - b
```

att värdet i a subtraherat med värdet i b deklareras till variabeln c, MATLAB svarar med:

```
c =

     2
```

Vilket är att vänta eftersom c = a - b = 5 - 3 = 2

2.2 Senast sparade värde

Om vi väljer att inte spara ett värde till en specifik variabel kommer MATLAB att deklarera värdet till variabeln *ans*. Precis som för övriga variabler kan vi komma åt det senaste värdet genom att skriva:

```
>>  ans
```

Exempel

Om vi i kommandofönstret skriver kommandot:

```
>>  7
```

Returnerar MATLAB:

```
ans =

     7
```

Vi har alltså deklarerat värdet 7 till variabeln *ans*.

2.3 Hantera variabler

Alla variabler vi deklarerar sparas i *Workspace* och finns kvar där tills vi tar bort dem, skriver över dem eller avslutar programmet. Genom att dubbelklicka på en variabel i *Workspace* öppnar vi *Variable Editor* där vi kan ändra värden.

Exempel

Om vi tidigare skapat variabeln *a* men ångrat oss, kan vi enkelt åtgärda detta genom att i kommandofönstret skriva:

```
>> clear a
```

Vill man radera samtliga variabler på en gång skriver man istället:

```
>> clear all
```

Exempel

Om man inte vill se en utskrift av sina värden i kommandofönstret när man deklarerar variabler skriver man ett semikolon efter kommandot. Skulle man till exempel skriva:

```
>>  C = 2;
```

skrivs värdet inte ut i kommandofönstret dock sparas variabeln C i *Workspace* precis som tidigare.

2.4 Skapa listor

Ibland kan man behöva deklarera en lista med värden, det vill säga en variabel som innehåller fler än ett värde.

Exempel

För att skapa en lista A med värdena 5, 3 och 2 skriver vi:

```
>> A = [5 3 2]
```

Alternativt:

```
>> A = [5,3,2]
```

Båda kommandon ger svaret:

```
A=

     5     3     2
```

Vi kan alltså dra slutsatsen att blanksteg och kommatecken båda åstadkommer samma sak nämligen separering av värden. MATLAB har nu skapat en lista med tre värden, det vill säga ett endimensionellt fält med tre värden, en rad och tre kolumner.

Exempel

Ett fält i två dimensioner skapas till exempel genom kommandot:

```
>> B = [4 2 1; 1 7 3]
```

MATLAB svarar med:

```
B =

     4     2     1
     1     7     3
```

Semikolon i det här sammanhanget innebär radbyte.

I själva verket är samtliga variabler i MATLAB listor. Till exempel är $a = 5$ en lista som endast innehåller värdet 5.

Två variabler som innehåller listor av värden kan deklareras till en ny variabel:

```
>>  C = [A;B]
```

MATLAB skapar då variabeln C genom att lägga till värdena i A i första raden och värdena i B på rad två och tre, vilket för A och B enligt ovan ger:

```
C =

     5     3     2
     4     2     1
     1     7     3
```

Viktigt att notera är att alla rader i C måste innehålla lika många kolumner, eller beskrivet på ett annat sätt, alla kolumner måste innehålla lika många rader. Om inte detta uppfylls kommer programmet att protestera.

I kursen *Linjär Algebra* kommer ni i detalj gå igenom vektorer och matriser och därför nöjer vi oss här med att tänka på fält som en lista med en samling av värden.

2.5 Skapa intervall av värden

Det finns ytterligare två väldigt användbara metoder att skapa fält i MATLAB. Vi börjar med kommandot *linspace* som skapar ett intervall av värden mellan en start- och stoppunkt. *linspace* är en inbyggd funktion i MATLAB och dessa behandlas mer noggrant i kapitlet *Tal och Matematiska funktioner*.

Exempel

För att skapa en lista med värden mellan 1 och 2 skriver man:

```
>> x = linspace(1,2)
```

Programmet skapar då variabeln x med värden från 1 till 2 i 100 jämt utspridda punkter. Att det blir just 100 värden är en standardinställning. Detta kan vi påverka genom att skriva:

```
>>  y = linspace(1,2,3)
```

Variabeln y skapas då med tre värden mellan 1 och 2. Ett annat sätt att utföra samma sak är genom att skriva:

```
>>  y = 1:0.5:2
```

Variabeln y skapas då med 1 som första värde och steglängden (hur mycket större det efterföljande värdet blir) 0.5, upp till värdet 2. Detta ger oss för ovanstående fall:

```
x =

    1.000    1.0101    1.0202    ...    1.9899    2.000

y =

    1.000    1.500    2.000
```

2.6 Indexering av värden

Varje värde i en lista i MATLAB är kopplat till ett index på två olika sätt. Dels får värdet på första raden i första kolumnen index 1, värdet på andra raden i första kolumnen index 2 och sedan följer index kolumnvis. Om en lista endast innehåller ett värde får detta värde index 1. För att få tag i ett värde i en lista skriver vi *variabel(index)*.

Exempel

```
>> A = [1 2 3; 4 5 6; 7 8 9];

>> A(3)

ans =

     7

>> A(8)

ans =

     6
```

Exempel

Värden i en lista har även ett rad och ett kolumnindex. Vi kan alltså använda värdet på en rad och i en kolumn genom att skriva *variabel(rad,kolumn)*.

```
>> A(2,3)

ans =

    6
```

Det går även att göra utdrag och endast använda delar av en lista. Låt säga att vi vill göra en ny lista med enbart en av raderna. Vi vill alltså komma åt en rad men alla kolumner. Vi skriver därför: *variabel(rad,:)* där kolontecknet betyder just *alla*.

Exempel

```
>> A = [1 2 3; 4 5 6; 7 8 9];

>> B = A(2,:)

B =

     4     5     6
```

Exempel

Slutligen kan vi anpassa vår utdrag genom att bestämma vilken första/sista rad samt första/sista kolumn vi vill hämta värden ur genom att skriva: *variabel(första rad:sista rad,första kolumn:sista kolumn)*. Kommandot:

```
>> A(2:3,1:2)
```

Ger för A enligt föregående uppgift:

```
ans =

     4     5
     7     8
```

3 Hjälpkommandon

MATLAB innehåller omfattande dokumentation och hjälpfunktioner som man måste lära sig att använda. Speciellt för att förstå hur de *inbyggda funktionerna* (nämns senare) fungerar. Det finns flera olika sätt att i MATLAB söka hjälp:

3.1 help

Att i kommandofönstret skriva:

help *funktion*

ger en kortare förklaring av den sökta funktionen (om en sådan funktion finns) och länkar dessutom till relaterade ämnen.

Exempel

Om vi i kommandofönstret söker information om den för oss bekanta funktionen *linspace* skriver vi in följande:

```
>>  help linspace
```

MATLAB svarar med följande:

```
 LINSPACE Linearly spaced vector.
    LINSPACE(X1, X2) generates a row vector of 100 linearly
    equally spaced points between X1 and X2.

    LINSPACE(X1, X2, N) generates N points between X1 and X2.
    For N < 2, LINSPACE returns X2.

    Class support for inputs X1,X2:
       float: double, single

    See also logspace, colon.

    Reference page in Help browser
       doc linspace
```

3.2 doc

Om man i hjälpen inte fann det man sökte efter, kan man istället skriva:

doc *funktion*

som leder till programmets dokumentation och en mer omfattande beskrivning av funktionen tillsammans med exempel på tillämpningar.
Dokumentationen kan också nås genom att klicka på frågetecknet i menyraden.

3.3 lookfor

Ytterligare ett sätt att söka information på är att skriva:

lookfor *ämne*

Vilket ger en lista över länkar som innehåller samma följd av bokstäver som ämnet. Klickar man på någon av länkarna innebär det samma sak som att skriva help *ämne* och en kort förklaring visas.
Denna funktion är mycket användbar då man vill komma åt en matematisk funktion där man inte känner till namnet i MATLAB.

Exempel

Om man behöver beräkna en integral men inte känner till någon funktion för detta i MATLAB skriver vi:

```
>>  lookfor integral
```

Vilket ger följande svar:

```
cosint     - Cosine integral function.
sinint     - Sine integral function.
ellipke    - Complete elliptic integral.
expint     - Exponential integral function.
dblquad    - Numerically evaluate double integral over a rectangle.
quad       - Numerically evaluate integral, adaptive Simpson ...
quad2d     - Numerically evaluate double integral over a planar ...
quadgk     - Numerically evaluate integral, adaptive Gauss-Kronrod ...
quadl      - Numerically evaluate integral, adaptive Lobatto ...
triplequad - Numerically evaluate triple integral.
assema     - Assembles area integral contributions in a PDE problem.
ellipk     - Complete elliptic integral of first kind.
```

Man kan ana att någon av ovantsående funktioner utför just det vi vill. I kursen *Inledande Matematik* kommer ni noggrant gå igenom hur MATLAB kan beräkna integraler

3.4 Sammanfattning

Help, *doc* och *lookfor* kan användas med parametrar för att till exempel söka hjälp med vissa inställningar eller begränsa antalet träffar i en sökning. Läs mer i hjälpen om hjälpfunktionerna om du vill fördjupa dig i detta genom att skriva något följande kommandon

```
>>  help help
>>  help doc;
>>  doc doc;
>>  doc help;
```

Skulle du mot förmodan inte hitta det du söker efter i MATLABs inbyggda dokumentation finns massor av information att nå via Google.

Vi rekommenderar att ni lägger tillräckligt med tid på att riktigt lära er söka efter hjälp i MATLAB då detta kommer hindra er från att köra fast. Ni behöver inte lära er alla funktioner i MATLAB utan och innan, satsa istället snarare på att lära er kunna hitta de verktyg ni behöver för att lösa den uppgift ni har framför er.

4 Tal och Matematiska funktioner

4.1 Aritmetiska operationer

Då MATLAB beräknar aritmetik utförs operationer i prioritetsordning. Om två operationer har samma prioritetsordning utförs beräkningar från vänster till höger. MATLAB har fem aritmetiska operationer. Dessa är i prioritetsordning:

Kommando	Funktion	Prioritet
.^	Potens	1
.*	Multiplikation	2
./	Division	2
+	Addition	3
-	Subtraktion	3

4.2 Skillnad mellan elementvis multiplikation och matrismultiplikation

MATLAB hanterar matriser och det finns två sätt att multiplicera dessa, nämligen matrismultiplikation och elementvis multiplikation. Detta måste man beakta när man använder programmet.
Elementvis multiplikation är den typ av multiplikation vi är vana med från grundskolan medan matrismultiplikation är lite klurigare.

I denna kurs beaktar vi endast elementvis multiplikation vilken utförs genom att skriva en punkt framför operatorerna multiplikation, division och potens. Undantaget är då man multiplicerar två stycken skalärer med varandra eller multiplicerar en skalär med en lista.

Exempel

Kodstycket nedan:

```
>>  A = [1 2 3];
>>  B = [1 2 3];
>>  C = A .* B
```

Ger följande svar:

```
C =

     1     4     9
```

Medan:

```
>>  C = A * B
```

Ger svaret:

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Vi kommer inte i detalj gå in på varför detta sker men i denna kurs skall ni ALLTID skriva multiplikation, division och potens med en punkt framför.

4.3 Elementära funktioner

MATLAB har ett stort antal förprogrammerade funktioner. En sådan funktion används enklast genom att anropa dess funktionsnamn och ange nödvändiga inargument.

Exempel

Hur beräknar man i MATLAB $\sqrt{9}$?

I MATLABs kommandofönster anropar man den inbyggda funktionen *sqrt*, där *sqrt* står för *square root* med inargumentet 9 enligt uppgift:

```
>>  sqrt(9)
```

Matlab ger då

```
ans =

     3
```

Talet 9 är i detta fall inargumentet i en funktion som kräver endast ett inargument.

Inbyggda funktioner i fallet med *linspace* kräver ibland fler än ett inargument. Andra exempel på inbyggda funktioner är:

```
sin(inargument)
cos(inargument)
tan(inargument)
log(inargument)
exp(inargument)
```

Alla ovan nämnda funktioner är vad man kallar elementära funktioner och en komplett lista med funktionsnamn samt tillhörande förklaringar ges genom att skriva följande kommando:

```
>>  help elfun;
```

MATLAB svarar då med en lista över de elementära funktionerna.

MATLAB innehåller en mängd andra funktioner utöver de elementära. Fler funktioner hittas enklast genom att söka i MATLABs inbyggda dokumentation (se kapitlet: *Hjälpkommandon*)

5 Scriptfiler

5.1 Skapa en scriptfil

När man använder MATLAB skriver man med fördel sina kommandon i något som kallas script- eller m-filer istället för direkt i *Command window* som hade varit alternativet.
En scriptfil är en fil där man skriver de kommandon man önskar utföra i den ordning de skall utföras.

När sedan alla kommandon är inskrivna i scriptfilen körs den och varje rad verkställs var för sig i den angivna ordningen.

Fördelen med att arbeta i scriptfiler är att de ger en god översyn över de olika sekvenserna samt att de kan sparas på ett enkelt sätt. Då man exempelvis vill kunna redovisa en lösningsgång eller arbeta i större kodmängder är således scriptfiler att rekommendera.

För att skapa en scriptfil skriver man i kommandofönstret:

```
>>  edit filensnamn
```

Varpå MATLAB frågar dig om man har för avsikt att skapa en ny m-fil med namnet *filensnamn*.
Efter att ha accepterat öppnas ett nytt fönster som du kan arbeta i.

Ett alternativt sätt är att klicka på *File/New/Script* i menyraden.

När man väl skapat en scriptfil är det bara att direkt skriva kommandon rad för rad precis som vi tidigare skrivit i *Command window*. När scriptfilen är redo att köras klickar man på den gröna knappen märkt *Run* och MATLAB utför dina kommandon.

Exempel

Beräkna polynomet: $y = 2 * x^2 + 3 * x + 4$ i en script-fil då $x = 4$

I *Command window* skriver vi:

```
>> edit exempel
```

En blank scriptfil öppnas med filnamnet *exempel.m*. Här skriver vi önskade kommandon:

```
a = 2;
b = 3;
c = 4;
x = 5;

y = a.*x.^2+b.*x+c
```

Därefter trycker vi på *Run* i verktygsfältet. MATLAB svarar i *Command window* med:

```
y =

    69
```

Dessutom sparas scriptfilen *exempel.m* till katalogen *Current folder*

Då man namnger m-filer bör man göra så med viss omtanke. Man bör exempelvis inte välja filnamn innehållande å,ä,ö, endast siffror eller som innehåller mellanslag. Man bör heller inte välja samma namn som någon av MATLABs inbyggda funktioner då det kommer uppstå problem.

5.2 Kommentera kod

Då man arbetar med en scriptfil lönar det sig ofta att kommentera sin kod för att utomstående enklare skall kunna sätta sig in i koden eller då man efter ett avbrott åter börjar arbeta med ett kodstycke. Detta görs enklast genom att skriva tecknet % innan eller efter en rad.

Att skriva % i en scriptfil innebär att allt som skrivs efter %-tecknet på samma rad kommer att ses som kommentarer och ej beräknas av MATLAB.

Exempel

En kommenterad scriptfil kan se ut som följer:

```
y = 10;
x = 15; %Variablerna deklareras

x = y+x
%y = x+y

% Operationen y = x+y kommer inte utföras således är y fortfarande
% lika med 10 till skillnad från x som ges värdet 25 i den fjärde
% raden
```

5.3 Publicera kod

För att redovisa sina resultat och beräkningsgångar kan man använda verktyget publish. Detta görs genom att trycka på knappen *Publish* i verktygsmenyn (samma meny som du hittade den gröna pilen). Vad publish gör är att den utför samtliga räkningar och sparar dem grafiskt i ett pdf, html eller latex-format.
Man kan också ändra inställningar för din publikation, till exempel om man inte vill skriva ut kommenterad kod, grafer, etc.

5.4 Rubriker, Underrubriker

Då man vill dela in sitt script i olika rubriker och underrubriker skriver man: %%.

Exempel

Följande skrivs i en scriptfil:

```
%% Huvudrubrik
%% Underrubrik 1
% Nedanstående exempel är skapat för att visa tillämpningar av
% tecknet % i MATLAB-kod

a=10;
b=20;

c=a+b

%% Underrubrik 2
% Lägg märke till att en innehållsförteckning har skapats automatiskt.
```

Ovan skrivna script kan publiceras. En bra övning vore att kopiera koden och publicera resultatet.

Notera att MATLAB också skapar en innehållsförteckning baserat på de rubriker som angivits. En publicerad scriptfil är ett utmärkt sätt att presentera beräkningar samt data för till exempel en labbrapport.

6 Grafisk visualisering

6.1 Grundläggande plottning

I många tillämpningar vill man analysera data genom att rita grafer. Detta är naturligtvis inga problem i MATLAB. I denna introduktionskurs kommer vi att plotta tvådimensionella figurer vilket oftast görs genom funktionen:

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

Funktionen ritar upp variabeln y mot variabeln x. En heldragen linje kommer att förbinda punkterna. En figur i vilken plottningen sker kommer också automatisk skapas.

Exempel

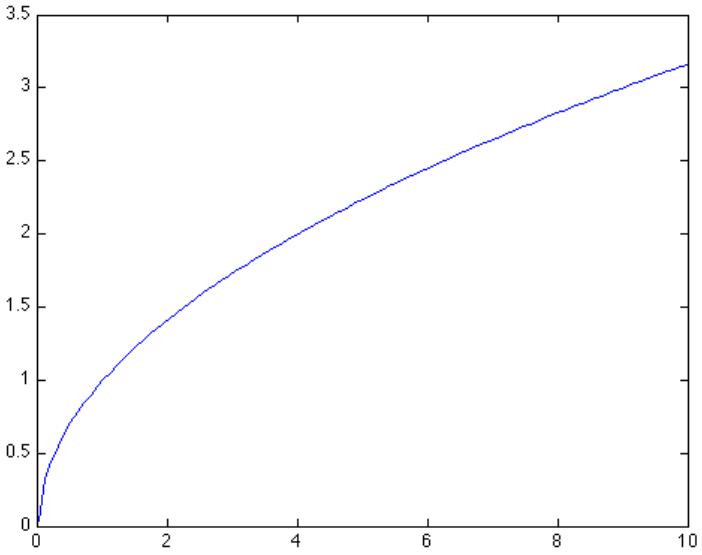
Vi definerar två listor:

```
x = linspace(0,10);
y = sqrt(x);
```

Vi vill sedan åskådliggöra hur y beror av x genom att illustrera sambandet i en graf. Detta görs genom att mata in kommandot för plottning:

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

vilket resulterar i följande figur:



6.2 Figurer

Då föregående exempel behandlas i MATLAB kommer användaren att upptäcka att en ny figur med titeln *figure 1* har skapats i vilken grafen har ritats. Detta sker automatiskt då man anropar funktionen plot*(inargument)*.
Vill man skapa en helt ny figur använder man kommandot:

```
figure(n);
```

där n = heltal

Då två grafer plottas efter varandra kommer man upptäcka att bara den senare åskådliggörs i figuren. Detta beror på att det endast finns en figur att plotta i. Alltså behöver man skapa en figur för varje plottning enligt nedan:

```
...
figure(1);
plot(x1,y1);
figure(2);
plot(x2,y2);
...
```

Vill man istället studera två grafer i samma figur måste man se till att den första inte skrivs över. Detta görs genom kommandot:

```
hold on;
```

Efter att kommandot körts kommer samtliga framtida plottar hamna i samma figur, tills dess att man skriver:

```
hold off;
```

Exempel

Teckna funktionerna $y(x) = \sin(x)$ och $z(x) = \cos(x)$ med 1000 punkter i intervallet $-10 < x < 10$.

```
x = linspace(-10,10,1000);
y = sin(x);
z = cos(x);
```

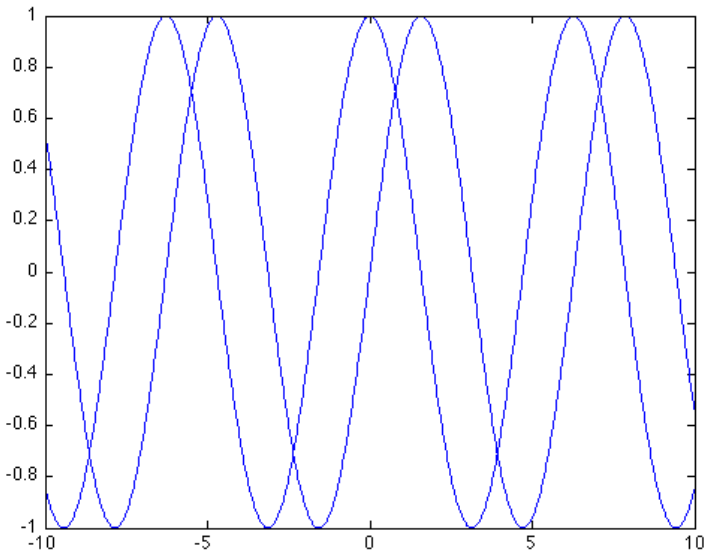
a) Plotta funktionerna $y(x)$ och $z(x)$ i två separata figurer

```
figure(1);
plot(x,y);
figure(2);
plot(x,z);
```

b) Plotta funktionerna i samma figur

```
figure(3);
plot(x,y);
hold on;
plot(x,z);
hold off;
```

Plotten i (b) resulterar i följande figur:



6.3 Plotegenskaper

Genom att ange vissa attribut för plotkommandot kan vi påverka hur grafen kommer att visas efter att koden exekverats. Attributet avskiljs med ett kommatecken och anges som en textsträng (Se exempel nedan). De olika påverkbara egenskaperna vi kommer att gå igenom är:

- Linjetyp
- Punkttyp
- Färg

Linjetyp

De olika linjetyperna som kan framställas är:

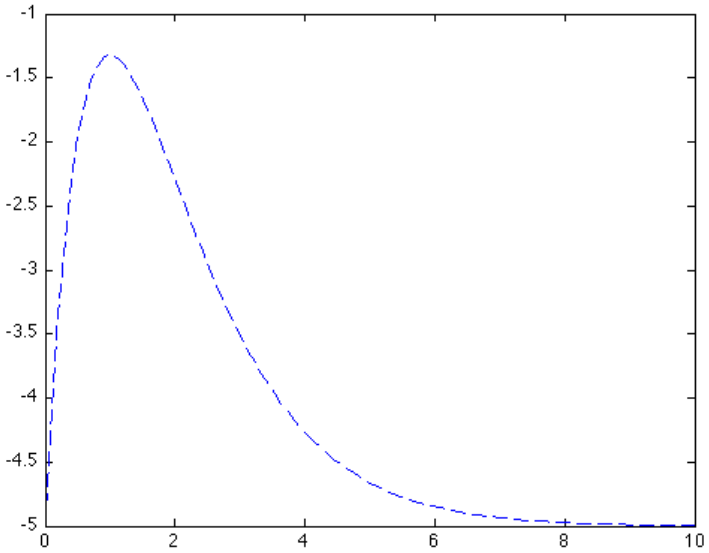
Attribut	**Resultat**
'-'	Heldragen linje
'- -'	Streckad linje
':'	Prickad linje
'-.'	Streck-prickad linje
'none'	Ingen linje

Exempel

Plotta grafen till funktionen $y(x) = 10xe^{-x} + 5$ i intervallet $0 < x < 10$ med en streckad linje:
Vi matar in följande i MATLAB:

```
x = linspace(0,10);
y = 10*x.*exp(-x)+5;
plot(x,y,'--');
```

Vilket resulterar i grafen på följande sida:



Punkttyp

Förutom att man kan plotta grafer med olika linjetyper kan man också göra illustrationer med olika punkttyper. De olika attributen som åstadkommer detta är:

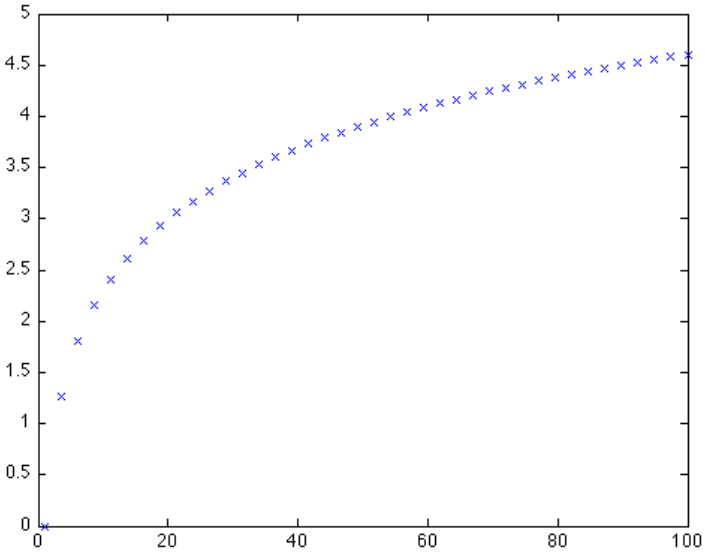
Attribut	Resultat
'.'	Punkter
'*'	Asterixer
'square'	Fyrkanter
'diamond'	Rutor
'o'	Ringar
'+'	Plustecken
'x'	Kryss

Exempel

Plotta $y(x) = \ln x$ i intervallet $1 < x < 100$ med kryss i 40 punkter:

```
x = linspace(1,100,40);
y = log(x);
plot(x,y,'x');
```

Plotten blir:



Färg

Vi kan också ändra färgsättningen på våra grafer genom att ändra attributen. De olika färgerna ges av följande attribut:

Attribut	Resultat
'b'	Blå
'g'	Grön
'r'	Röd
'y'	Gul
'c'	Cyan
'm'	Magenta
'k'	Svart

Kombinera attribut

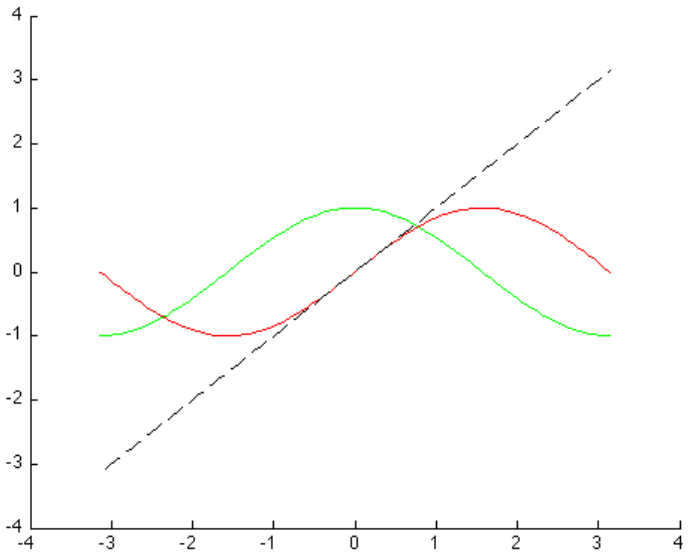
Det går naturligtvis att tillämpa flera attribut för varje plot. Detta åstadkommer man genom att skriva de olika symbolerna för attributen efter varandra i samma textsträng.

Exempel

Plotta funktionerna $y(x) = cos(x)$, $z(x) = sin(x)$ och $w(x) = x$ i intervallet $-\pi < x < \pi$ med några olika attribut.

```
x = linspace(-pi,pi);
y = cos(x);
z = sin(x);
w = x;
figure(1);
hold on;
plot(x,y,'g');
plot(x,z,'r');
plot(x,w,'k--');
hold off;
```

Koden ovan ger figuren:



6.4 Axlar och skalning

Efter att ha skapat en plot märker man att axlarna automatiskt blivit skalade för att passa inmatad data. Ibland önskar man dock själv anpassa skalningen för att till exempel märka ut ett visst område i plotten.
Det finns i MATLAB flera sätt att manipulera axlarna och skalningen, nedan följer några grundläggande.

Vill man plotta utan axlar skriver man innan plotkommandot:

```
axis off;
```

och för att få tillbaka dem, skriver man helt enkelt:

```
axis on;
```

Notera att MATLAB som standard är inställt på att rita ut axlarna.
Man kan också välja mellan att plotta med eller utan rutnät detta görs genom att skriva;

```
grid on
grid off
```

Standarinställningen är plottning utan rutnät
Man kan också påverka skalningen och manipulera i vilket område man vill rita ut plotten. Några grundläggande kommandon för detta är:

Kommando	**Resultat**
`axis equal`	Samma skalning på $x-$ och $y-$axeln
`xlim([x1 x2])`	$x-$axeln skalas mellan startpunkten $x1$ och slutpunkten $x2$.
`ylim([y1 y2])`	$y-$axeln skalas mellan startpunkten $y1$ och slutpunkten $y2$.
`axis([x1 x2 y1 y2])`	x skalas mellan $x1$ och $x2$. och y mellan $y1$ och $y2$.

Notera att då start- och stoppvärdena skall matas in, så matas de in som vektorer, vilket avslöjas av hakparenteserna.

Linjär/logaritmisk skala

MATLAB plottar per automatik i en linjär skala, dock finns det tillfällen då en logaritmisk skala vore att föredra. Ni kommer till exempel i kretsanalysen att rita upp grundläggande kretsars så kallade överföringsfunktioner i logaritmisk skala, när ni i senare kurser kommer stöta på mer avancerade sådana kan man med fördel låta MATLAB göra jobbet. Några kommandon för att plotta i logaritmisk skala med basen tio är:

Kommando	**Resultat**
`loglog(x,y)`	Plottar x och y i logaritmisk skala på båda axlarna.
`semilogx(x,y)`	Plottar $x-$axeln i logaritmisk skala medan $y-$axeln förblir linjär.
`semilogy(x,y)`	Plottar $y-$axeln i logaritmisk skala medan $x-$axeln förblir linjär.

6.5 Förklarande textsträngar

Då du vill publicera dina resultat eller i annat syfte behöver förklara vad du åstadkommit, kan beskrivande text till dina figurer komma väl till pass. MATLAB har inbyggda kommandon för att skapa titlar, textrutor och teckenförklaringar.

Vi börjar med att behandla kommandot för att skapa en titel till din plot. Det enda man gör är att efter plotkommandot skriva:

```
title(titel)
```

ovanstående kommando tilldelar plotten titeln: *titel*, men man kan naturligtvis döpa den till vad man vill.

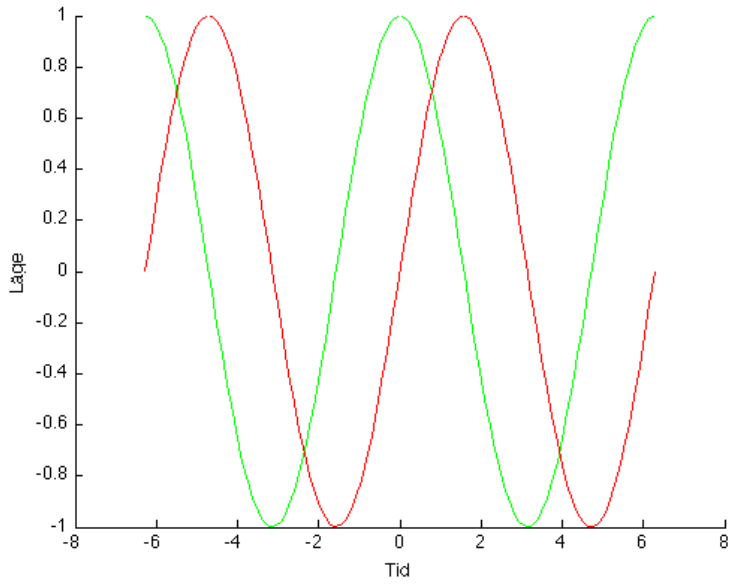
För att skriva förklarande textsträngar till axlarna skriver man:

```
xlabel(x-axeln);
ylabel(y-axeln);
```

Vilket resulterar i att x-axeln och y-axeln tilldelas textsträngarna: *x-axeln* och *y-axeln* respektive.

Exempel

Två kroppars rörelse beskrivs av grafen nedan:



Som genererats av följande kodstycke:

```
x = linspace(-2*pi,2*pi);
y = cos(x);
z = sin(x);
w = x;
figure(1);
hold on;
plot(x,y,'g');
plot(x,z,'r');
xlabel('Tid');
ylabel('Läge')
```

x-axeln och $y-$axeln har alltså erhållit titlarna *Tid* och *Läge*.

legend

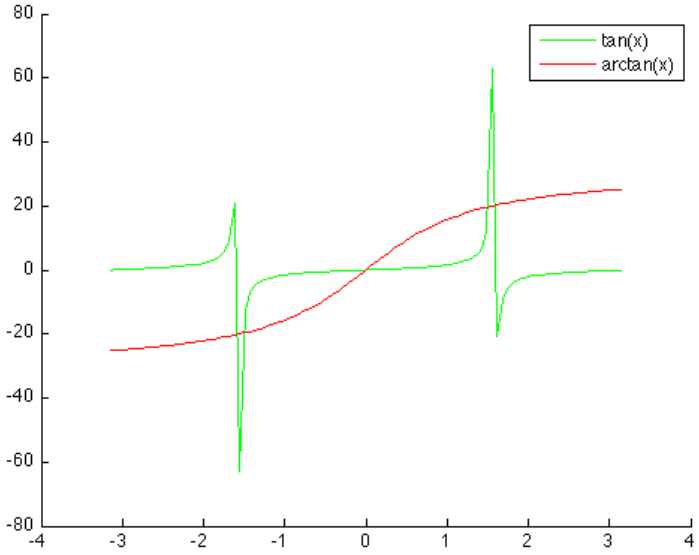
Om man till exempel ritat flera kurvor i samma plot och tydligt vill särskilja dem är funktionen *legend* mycket användbar. Dess funktion är att tilldela en förklarande textruta till kurvan. För användning, se exemplet nedan:

Exempel

Kodstycket:

```
x = linspace(-pi,pi);
y = tan(x);
z = 20*atan(x);
figure(1);
hold on;
plot(x,y,'g');
plot(x,z,'r');
legend('tan(x)','arctan(x)','Location','NorthEast')
```

Ger upphov till följande figur:



Kommandot fungerar genom att matcha första attributet med den första plotten. Genom att plotta sina figurer i olika färger och/eller linjetyper kan man i sin plottning särskilja kurvorna.
Genom att skriva *'Location'* följt av ett väderstreck bestämmer man var i plotten textrutan ritas. Skulle man istället vilja ha rutan längst ner till vänster i plotten hade man skrivit:

```
legend('tan(x)','arctan(x)','Location','SouthWest')
```

För mer information om *legend* och andra grafiska verktyg, se den inbyggda hjälpdokumentationen i MATLAB.

7 Funktionshantering

7.1 Anonyma funktioner

I tidigare kapitel studerade vi hur funktionsbegreppet används i MATLAB och då speciellt de elementära funktionerna. Vi skall nu använda oss av funktionsbegreppet på ett vidare plan och påvisa hur centralt ”funktionstänket” är i MATLABs programmeringsmiljö.

Tänk dig att du känner en viss funktion som beskriver ett analytiskt samband. För enkelhetens skull låt oss anta att sambandet ser ut som polynomet av grad tre nedan.

$y(x) = ax^3 + bx^2 + cx + d$

Då man känner respektive värde för a,b,c och d är det enkelt att räkna ut y för ett givet x. Även om proceduren ovan är enkel kan den vara tidskrävande. Särskilt då man räknar ut olika y för ett antal x-värden.

Ett sätt att hantera ekvationer är med hjälp av *Anonyma funktioner*.
Att skapa en *Anonym funktion* av tredjegradspolynomet ovan görs på följande sätt:

```
y = @(x) a*x^3+b*x^2+c*x+d
```

Där @(x) är ett funktionshandtag till y som är en funktion av x.

Vi har nu skapat en funktion *y(x)* som kräver ett inargument för att generera ett värde

Att skapa en anonym funktion påminner i mångt och mycket om att deklarera en variabel vilket vi gjort tidigare. Den stora skillnaden är att vår anonyma funktion, likt de elementära funktionerna, kräver ett inargument för att generera ett värde. Notera att även anonyma funktioner sparas i *Workspace*.

Exempel

Om vi för polynomet ovan ger värden till konstanterna a, b, c och d. och sedan deklarerar den anonyma funktionen *y(x)* enligt nedan:

```
a=1;
b=2;
c=3;
d=4;
y = @(x) a*x^3+b*x^2+c*x+d;
```

Om vi sedan vill veta vad funktionen *y(x)* ger för värde då *x=2* skriver vi:

```
y(2)
```

Vilket ger resultatet:

```
ans =

    26
```

Eftersom $y = 1 * 2^3 + 2 * 2^2 + 3 * 2 + 4 = 8 + 8 + 6 + 4 = 26$

Exempel

Skapa en anonym funktion av uttrycket

$y = 3x + 5$

beräkna även $y(x)$ då $x = 3$

Vi skriver uttrycket som en anonym funktion:

```
y=@(x) 3.*x+5;
```

Då man vill beräkna ett y-värde för x=*3* som i detta fall skriver man:

```
y(3)
```

Vilket ger svaret:

```
ans

      = 14
```

Exempel

Skapa en anonym funktion med avseende på x av det analytiska uttrycket:

$p = \sqrt{x^2 + \sin(2\pi x)}$

och beräkna $p(x)$ då $x = 0.5$.

I Matlab skriver vi

```
p=@(x) sqrt(x^2+sin(2*pi*x));
```

För att beräkna ekvationens lösning för ett x värde skriver vi till exempel:

```
>>  p(0.5)
```

som ger:

```
ans

      = 0.5
```

Exempel

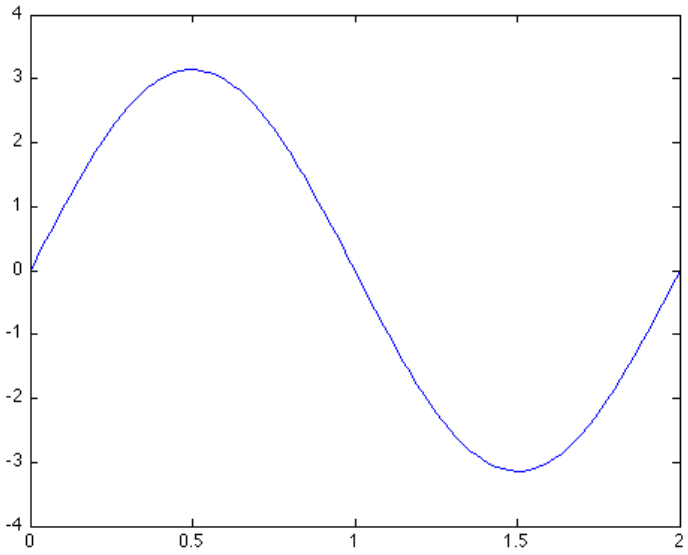
En anonym funktion kan byggas upp av andra anonyma funktioner som i fallet *x(t)* nedan:

```
t=linspace(0,2);

w=@(n) 5*pi*n;
s=@(n) (pi/n)*sin(w(n)*t/5);
x=@(t) s(1);

plot(t,x(t))
```

Grafen som generas ser ut som följer:



Alltså en sinusvåg med periodtiden 2 och amplituden *pi*.
För att förstå detta behandlar vi w, s och x algebraiskt:

$w(n) = 5\pi n$
$s(n) = (\pi/n)\sin(w(n)t)$
$x(t) = s(1)$

Insättning av $w(n)$ i $s(n)$ ger:

$s(n) = (\pi/n)\sin(5\pi nt)$

Insättning av $s(n)$ i $x(t)$ då $n = 1$ ger:

$x(t) = s(1) = \pi\sin(5\pi t)$

Vilket är funktionen till den graf vi ser ovan.

7.2 Funktionsfiler

Funktioner kan förutom att skrivas som anonyma funktioner, också skrivas som så kallade *funktionsfiler*. Dessa skrivs absolut lättast i scriptfiler. Fördelarna med funktionsfiler gentemot anonyma funktioner ökar i samband med att funktionerna blir mer och mer komplicerade.

En slutsats är således att anonyma funktioner är bra för enklare uttryck medan mer omfattande mängd kod skrivs i funktionsfiler.

Skapa en funktionsfil

För att skapa en funktionsfil börjar man med att öppna en ny scriptfil. I följande exempel väljer man att döpa funktionen till *myfun*. Scriptfilen skapas genom att i *Command Window* skriva följande:

```
>>   edit myfun
```

När vi väl skapat scriptfilen är nästa steg att definiera den som en funktionsfil. Detta görs till exempel genom att i scriptfilen skriva följande:

```
function [y] = myfun(x)
y=2*x
end
```

Där x är inargumentet till funktionen och y det värde som returneras.

Efter att scriptfilen sparats är den en fullt fungerande funktion som returnerar en fördubbling av inargumentet.
Observera uttrycket *end* i slutet av funktionen. Uttrycket är nödvändigt för att funktionsfilen skall fungera. Ty en funktion måste inte bara initieras utan också avslutas, djupare förklaring om varför ges i kursen *Objektorienterad programmering*.

Anropa en funktionsfil

Hur anropas funktionen ovan? På samma sätt som vi tidigare anropade MATLABs inbyggda funktioner och de anonyma funktionerna anropas en funktionen genom att skriva:

```
>>  y = myfun(5)
```

MATLAB svarar då:

```
y =

    10
```

Funktionen ovan kan tyckas vara tämligen onödig eftersom ett enklare sätt att lösa problemet vore att i *Command Window* skriva:

```
>>  x = 5;
>>  y = 2*x;
```

vilket skulle resulterat i samma sak men syftet med exemplet är att ge insikt i vad en funktionsfil är och hur den används.

Exempel

Ett annat exempel på hur man kan använda funktionsfiler är att konstruera exakt samma funktion som den vi använde i kapitlet om anonyma funktioner.

Skapa en funktion av det analytiska uttrycket:

$\sqrt{x^2 + \sin(2\pi x)}$

och beräkna $y(x)$ för $x = 0.5$.

Detta görs genom att skapa en funktionsfil som vi väljer att kalla *funktion*.

```
>>  edit funktion
```

I den genererade script filen skiver vi:

```
function [u] = funktion(x)
u = sqrt(x.^2+sin(2.*pi.*x))
end
```

Observera att namnet *funktion* är godtyckligt till skillnad från *function* som är nödvändigt för att MATLAB skall erkänna scriptfilen som en funktion.

För att beräkna $y(x)$ då $x = 0.5$ skriver vi:

```
>>  funktion(0.5)
```

som ger svaret:

```
ans =

    0.5
```

7.3 Sammanfattning

Funktionshantering är ett väldigt centralt begrepp inom många programmeringsspråk och väl värt att lägga ner tid på att förstå. Begreppet behandlas djupare i kursen *Objektorienterad programmering*

Det är faktiskt så att MATLABs inbyggda funktioner vi tidigare stött på, såsom *sin*, *rand*, etc. samtliga är uppbyggda som funktionsfiler.

8 Felsökning

8.1 Olika typer av fel

När ni börjat arbetar med MATLAB kommer ni troligtvis erfara diverse rödfärgade felmeddelande i samband med att ni kör era program. Bortsett från den uppenbara frustration som uppstår då detta sker finns det saker man kan läsa ut av dessa felmeddelanden.

Det värsta felet man kan tänkas göra med MATLAB är det man själv inte inser och som MATLAB inte varnar för och därmed går obemärkt. För att undvika sådana fel bör man förstå MATLABs begränsningar.

Samtliga av dagens elektroniska räknemaskiner såsom miniräknaren och persondatorn är fantastiskt snabba i att utföra beräkningar. Det är dock inga svårigheter att ge de en så omfattande kalkyl att de fastnar och inte kan slutföra beräkningarna. Ett exempel är att fråga vilket räkneinstrument som helst efter en högre ordningens fakultet.

Exempel

Testa att beräkna 1000! med MATLAB

Tips: i MATLAB kallas operationen ! för factorial(n)

Blev svaret vad du trodde det skulle bli? Vad blev det?

Generellt kan detta fenomenet sammanfattas med att räkneinstrument inte kan hantera för stora tal.

8.2 Att dela med noll

Det enklaste sättet att få samma resultat som ovan är när man delar ett tal med noll eller ett tal väldigt nära värdet noll. I ett rent matematiskt synsätt är detta en omöjlig operation eftersom man kan inte dela med noll, men med MATLAB så är detta fullt möjligt och man bör därför se upp så att det räkneoptimistiska MATLAB inte börjar utföra oändligt långa räkningar som resulterar i oändligt stora tal eftersom detta slösar både våran tid och datorns lagringsutrymme.

8.3 Tre specialfall

inf

inf, (infinite), är det största talet som MATLAB kan räkna med och alla tal större än detta avrundas därför ner till detta tal.

Exempel

```
>> 2.*inf

ans  =
     inf
```

eps

eps, (epsilon), är det minsta talet skiljt från noll som MATLAB kan hantera och är av storleksordningen 2^{-52}

NaN

NaN, (Not a Number), ges då man blandar olika typer av tal på ett felaktigt sätt. Exempelvis om man blandar logiska och numeriska tal. NaN är således MATLABs svar då programmet inte vet hur den skall representera ett svar numeriskt.

9 Logik

9.1 Logiska operationer

Ofta vill man utföra olika kodsekvenser eller beräkningar beroende på om ett villkor är sant eller falskt. I vissa fall vill man upprepa en beräkning tills dess att ett villkor blir uppfyllt alternativt slutar att uppfyllas. I alla programmeringsspråk finns det operationer för att testa detta. I MATLAB betyder ett villkor som är lika med 0 att det är falskt och ett villkor som är lika med 1 att det är sant. Alla värden skilda från 0 tolkas som sanna men vi håller oss här till ettor och nollor.

Relativa operatorer

De operatorer som testar likhet mellan tal kallar vi för relativa operatorer och de känns igen från matematiken även om vi här tecknar dem något annorlunda.

Relativa operatorer:

Operator	Funktion
<	mindre än
<=	mindre än eller lika med
>	större än
>=	större än eller lika med
==	lika med *eftersom '=' används för att deklarera variabler*
~=	inte lika med

Exempel

```
x = 5;
y = 2;
x < y

ans =

    0

x > y

ans =

    1

>> s = x ~= y

s =

    1
```

Och/eller

Ibland vill man testa flera villkor. En sekvens kan köras om två villkor är uppfyllda eller om något villkor är uppfyllt. Vi använder oss här av de logiska operatorerna *och/eller*.

Operator	Funktion
&	och
|	eller

Exempel

```
a = 3;
b = 5;
c = 9;
a < b & b < c

ans =

    1
```

Vi kontrollerar om $a < b$ *och* om $b < c$. Svaret blir 1 och villkoret är sant.

```
>> a == b | b ~= c


ans =

    1
```

Om a är lika med b eller b är skilt ifrån c. Svaret blir även här 1 och villkoret är sant eftersom b inte är lika med c.

```
a == b | b == c

ans =

    0
```

Detta villkor blir sant antingen om a är lika med b eller om b är lika med c. I exemplet är inget av detta sant och villkoret blir falskt.

9.2 if/else

Som nämnts ovan kan ett uttryck eller en beräkning köras om ett visst villkor är uppfyllt och detta är precis vad if-satsen gör. Om det första villkoret är sant körs bara det första uttrycket. Är det första villkoret däremot falskt kontrolleras nästa villkor i eventuella elseif-satser. Är det villkoret sant körs bara det uttrycket. Om inget villkor är sant körs uttrycket i else. Observera att vi avslutar med end.

Exempel

```
a = 3;
b = 5;

if a < b
     a = 2*a
else
     a = a/2
end
```

Om a är mindre än b multiplicerar vi a med 2 och sparar till variabeln a, vilket också sker och ger resultatet:

```
a =

    6
```

Exempel

```
a = 7;
b = 3;

if a < b
     a = 2*a;
elseif a == b
     a = a+b;
else
     a = 0;
end
```

I detta fall är a varken mindre eller lika med b, vilket ger resultatet:

```
a =

    0
```

9.3 for

For-loopen utför en sekvens ett förutbestämt antal gånger. Detta går till så att loopen körs för varje värde i en lista. Låt säga att vi deklarerar variabeln k = [3 5 6]. For-loopen körs då tre gånger. Första gången med k = 3, andra med k = 5 och tredje och sista gången med k = 6.

Exempel

```
a = 3;

for k = [3 5 6]

     a = a+k;

end
```

Lägg märke till till att vi väljer att inte skriva ut resultatet. Alternativet är att skriva ut varje addition med det aktuella värdet i k vilket kan vara användbart vid felsökning men kan för en for-loop ge väldigt långa utskrifter. Vill vi veta vilket värde a fick står detta i *Workspace* eller så skriver vi:

```
>> a

a =

    17
```

Exempel

```
b = 5;

for i = 1:5
     b = b*i;

end
```

Vi deklarerar listan:

i = 1:5 = [1 2 3 4 5].

Loopen körs för varje värde av variabeln i.

9.4 while

While-loopen upprepas ett på förhand obestämt antal gånger. Innan loopen körs kontrolleras om villkoret är sant. Detta kallas för ett test för loopen. Är villkoret sant körs loopen första gången. Nästa varv går till på precis samma sätt. Villkoret kontrolleras och loopen körs. Detta upprepas tills villkoret blir falskt.

Exempel

```
a = 5;
B = [2 3 1 4 7];

while a > 0

     B = B/a;
     a = a-1;

end
```

Exemplet ovan ger resultatet:

```
B =

    0.0167    0.0250    0.0083    0.0333    0.0583
```

10 Referenser

1. Moore, H. (2009) MATLAB for Engineers. Pearson Education.
2. Jönsson, P. (2006) (MATLAB-beräkningar inom teknik och naturveten- skap). Studentlitteratur AB.
3. Downey, A, B (2010) Physical Modeling in MATLAB. Green Tea Press.
4. MATLAB. (2011) Product Documentation. http://www.mathworks.com/help/techdoc/ (2011-05-02)
5. Karlström, B (2009) Något om MATLAB Chalmers tekniska högskola