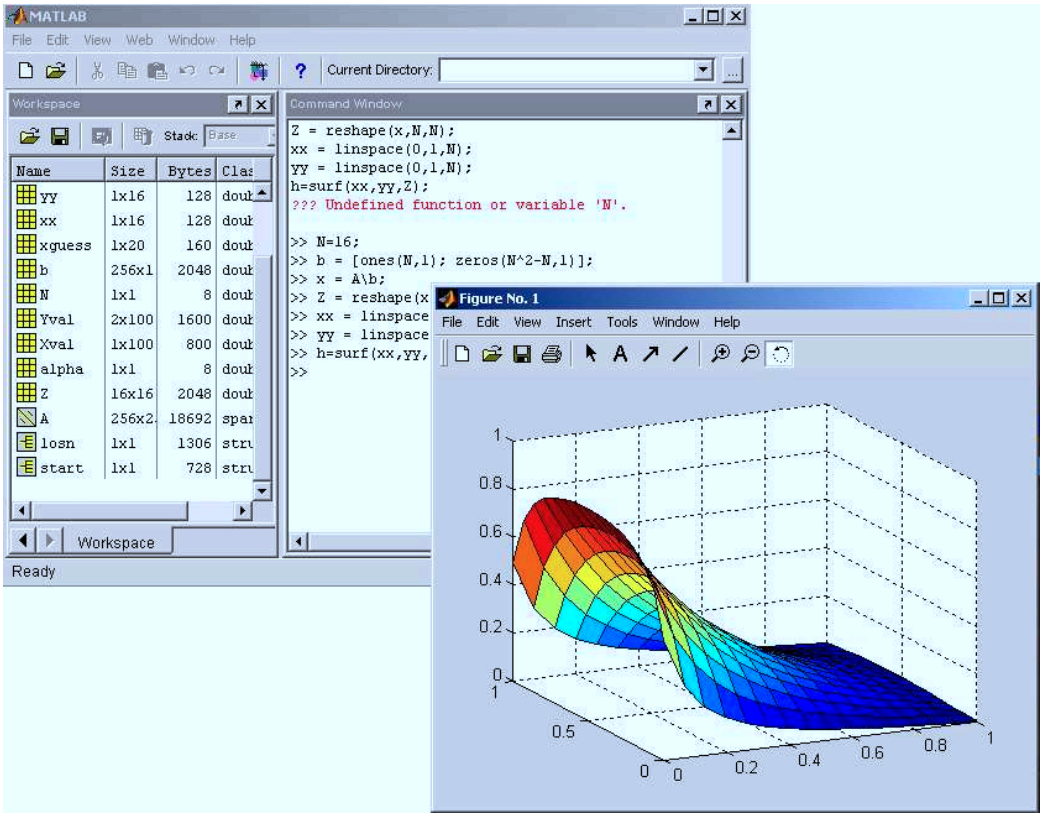
Lilla MATLAB

Lilla MATLAB –

en introduktion till MATLAB

Jan 2005



Inst för informationsteknologi, avd för teknisk databehandling

Introduktion till MATLAB

MATLAB är ett interaktivt programpaket för tekniska beräkningar och visualisering av data, utvecklat av The MathWorks, Inc. MATLAB är ett utmärkt redskap för utbildning och används inom t ex beräkningsvetenskap, tillämpad matematik, fysik, teknik, kemi, biologi, ekonomi och alla andra sammanhang där numeriska beräkningar förekommer.

Detta kompendium ger en relativt grundläggande introduktion till MATLAB. Det kräver inga djupare kunskaper i beräkningsvetenskap eller programmering. Kompendiet innehåller exempel och övningar i MATLAB, där exemplen är intriktade mot naturvetenskap. För djupare kunskaper i MATLAB hänvisas till *Användarhandledning för MATLAB 6.5* av Eva Pärt-Enander och Anders Sjöberg. Se boken webbplats,
http://www.it.uu.se/edu/bookshelf/matlab/
för mer information.

I kompendiet introduceras först MATLAB-kommandon som sedan tillämpas i efterföljande övningsuppgifter. Lösningsförslag till övningsuppgifterna finns på kompendiets hemsida,
http://www.it.uu.se/edu/bookshelf/LillaMatlab/

Materialet sammanställdes hösten 1999 av Anna Rydberg, Stina Svensson och Ken Mattsson (1999) och bygger till stora delar på boken *Användarhandledning för MATLAB 5* av Eva Pärt-Enander och Anders Sjöberg samt matematikboken *Naturlig matematik,* av Staffan Rodhe och Inger Sigstam.

Under hösten år 2000 omarbetades kompendiet av Andreas Pihl. Kompendiet fick då också sitt nuvarande namn (versionen från 1999 hette *MATLAB för naturvetare*). December 2003 uppdaterades kompendiet av Stefan Pålsson, för att stämma med nyare versioner av MATLAB. Viss omskrivning och omflyttning av olika delar gjordes samtidigt. Ytterligare uppdateringar och tillägg av kapitlet *Symbolisk hantering med MATLAB* gjordes under 2004 av Björn Schröder och Stefan Pålsson.

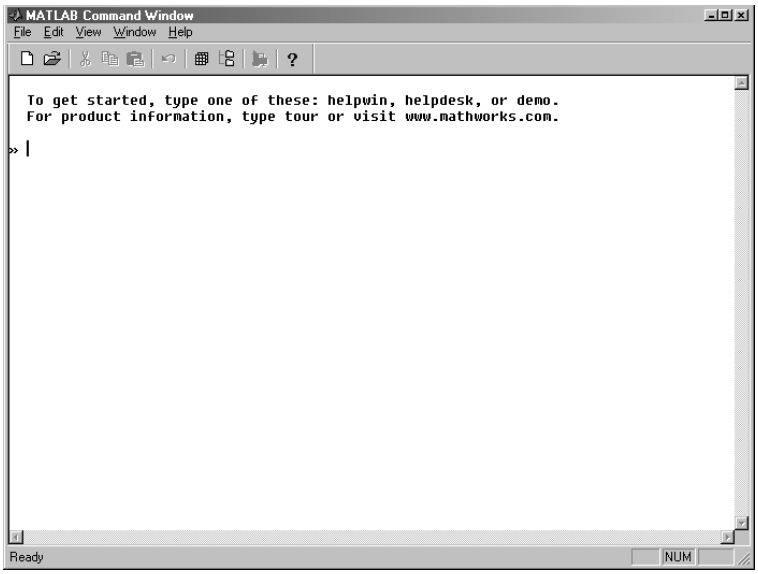
Hjälp med bra övningsuppgifter har vi fått från Christer Elvingson, institutionen för fysikalisk kemi, Ulf Lindh, institutionen för onkologi, radiologi och klinisk immunologi, och Johan Paulsson, institutionen för biologisk grundutbildning.

Bilden på framsidan visar en beräkning av värmeledning på en kvadratisk platta.

Innehåll

1 Att börja använda MATLAB

MATLAB startas genom att klicka på ikonen för MATLAB (Windows och Macintosh) eller att skriva *matlab* (UNIX) i UNIX-systemets kommandofönster. Vilket sätt som gäller beror alltså på vilket system som används. Ett tomt fönster öppnas, MATLAB:s *kommandofönster*, se Figur 1. Det exakta utseendet kan variera beroende på inställningar.

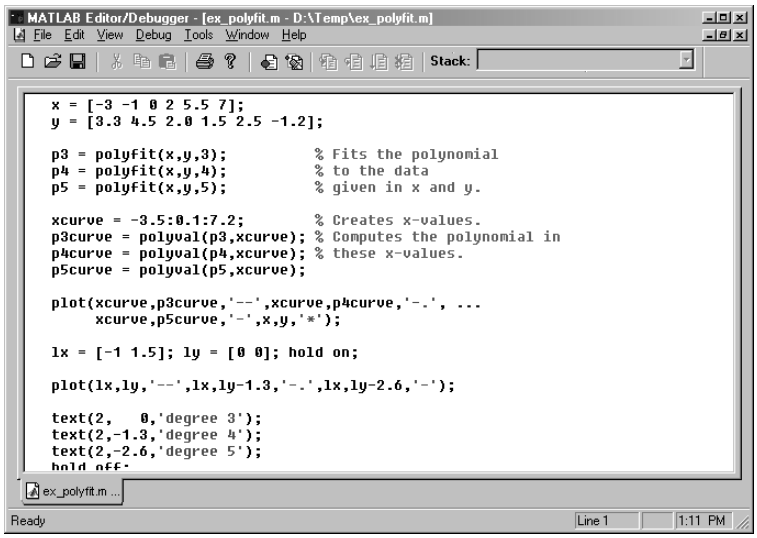


Figur 1: MATLAB:s kommandofönster.

Tecknet `>>` som syns i kommandofönstret kallas MATLAB-promptern och det visar att MATLAB är redo att ta emot ett kommando. Med *kommando* menas det som skrivs efter promptertecknen och som talar om för MATLAB vad som ska göras härnäst. Inmatningen av ett kommando avslutas genom att trycka på `return`-tangenten, då MATLAB utför kommandot. Eventuella svar, t ex resultatet av en beräkning, skrivs ut direkt under kommandot.

Istället för att utföra varje kommando interaktivt i MATLAB finns det också möjlighet att skriva längre sekvenser av kommandon i s k *m-filer*. En m-fil skapar man i ett annat fönster som kallas *editeringsfönster* och som är en enkel textediterare, se Figur 2. Därefter sparar man m-filen och låter MATLAB "köra" filen, dvs utföra alla kommandon i den. Man kan givetvis även använda andra textediterare om man önskar, t ex *emacs* i UNIX/Linux. Mer information om m-filer finns på sidan 29 i kapitel 2.

Efterhand när exemplen blir längre underlättas arbetet kraftigt om man använder m-filer. Dels blir överblicken bättre och dels kan man vid ett senare arbetspass återvända till exemplet utan att behöva skriva in alla kommandon igen.



Figur 2: MATLAB:s textediteringsfönster.

Kommandofönster, editeringsfönster och m-filer	
kommandofönster	det fönster som öppnas när MATLAB startas. Här kan beräkningar göras direkt som i en vanlig miniräknare genom att skriva kommandon och trycka på return. Här sker också exekveringen, dvs körningen, av m-filer.
grafikfönster	ett fönster som kommer fram när en graf ritas. Om inget särskilt anges kommer endast ett grafikfönster att användas: en ny graf ersätter en gammal.
editeringsfönster	ett fönster i MATLAB:s textediterare. Här skapas, editeras och sparas m-filer.
m-fil	textfil med MATLAB-kommandon. Se kapitel 7.

MATLAB avslutas genom att välja `Exit MATLAB` i menyn `File`. Man kan också ge kommandot `exit` eller `quit` i MATLAB:s kommandofönster.

1.1 Grundläggande beräkningar

När man skriver ett kommando i MATLAB kan det se ut så här:

```
>> 3546 + 4
ans =
      3550
```

Här har MATLAB använts ungefär som en vanlig miniräknare. Om inget annat anges kommer resultatet att läggas i variabeln `ans` som är en förkortning av det engelska ordet *answer*. Vill man istället spara resultatet i någon annan variabel anger man det på följande sätt:

```
>> x = 3546 + 4
x =
    3550
```

Resultatet av additionen $3546 + 4$, dvs 3550, är nu lagrat i variabeln x.

MATLAB är huvudsakligen *inte* ett program som kan räkna med symboler (jämför t ex med Maple). Det innebär bl a att man inte kan räkna med variablerna x, y, eller någon annan variabel utan att först tala om vilka tal dessa variabler representerar. I exemplet ovan är variabeln x satt till att representera talet 3550 och kan därför användas i senare beräkningar. Men däremot kan man inte göra beräkningen

```
>> x + y
??? Undefined function or variable 'y'.
```

eftersom variabeln y saknar värde. När y tilldelats ett värde kan man förstås utföra additionen. Det finns dock tillägg till MATLAB, s k *toolboxar*, och bl a finns sådana toolboxar för *symboliska beräkningar*. Detta beskrivs i avsnitt 8.

I likhet med miniräknare finns de vanliga matematiska funktionerna definierade i MATLAB, man kan t ex skriva

```
>> x = sin(pi/4)
x =
    0.7071
```

Observera att MATLAB använder decimal*punkt* istället för decimal*komma*. Detta gäller även tal som du som användare matar in.

Några vanliga matematiska funktioner i MATLAB	
`+, -, *, /`	addition, subtraktion, multiplikation & division.
`sin(x), cos(x), tan(x)`	sinus, cosinus, tangens. Vinkeln `x` anges i radianer.
`asin(x), acos(x), atan(x)`	arcsin, arccos, arctan. Resultatet ges i radianer.
`x^y`	upphöjt till, x^y.
`exp(x)`	exponentialfunktionen, e^x.
`log10(x)`	tiologaritmen av x, $log_{10}(x)$.
`log(x)`	naturliga logaritmen av x, $ln(x)$.
`sqrt(x)`	roten ur, $\sqrt{x}$.
`pi`	talet π.
`abs(x)`	absolutbeloppet av x, $\|x\|$.

MATLAB "kommer ihåg" alla variabler som har definierats under ett arbetspass. Om man har gett en variabel x ett visst värde i MATLAB så kommer x behålla det värdet ända tills man sätter x till ett annat värde eller avslutar arbetspasset. Om man skriver

```
>> y = 5
y =
    5
>> x = y
x =
    5
>> y = 6
y =
    6
```

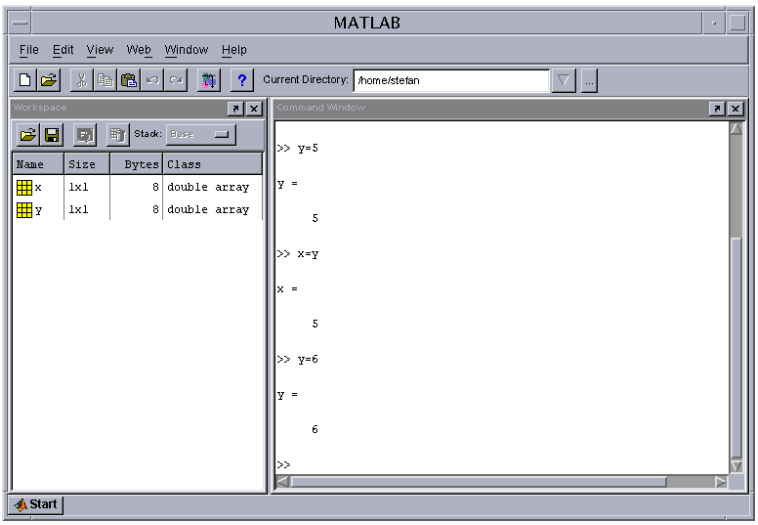
så kommer x fortfarande att ha värdet 5 eftersom man inte angivit att x ska ha något annat värde. Skulle man göra tilldelningen `x = y` ytterligare en gång så skulle givetvis värdet på x ändras till 6.

Med kommandot `who` (i kommandofönstret) får man en lista på alla variabler som definierats. Kommandot `whos` ger en mer detaljerad lista:

```
>> who
Your variables are:
x         y
```

```
>> whos
  Name      Size          Bytes  Class
  x         1x1               8  double array
  y         1x1               8  double array
Grand total is 2 elements using 16 bytes
```

Det finns också möjlighet att se detta i MATLAB:s s k *workspace*. MATLAB-fönstret kan nämligen delas upp i två delar, och i den vänstra delen kan t ex MATLAB:s *workspace* öppnas, se Figur 3. För att öppna *workspace*, markera `Workspace` i menyn `View`.



Figur 3: MATLAB:s workspace i den vänstra halvan av fönstret.

Variabler kan rensas bort med kommandot `clear`. Om man t ex vill radera variablerna x och y skriver man `clear x y`. Om man bara skriver `clear` rensas alla variabler bort.

Det finns också möjlighet att spara variabler mellan arbetspass, se sidan 131 i appendix B.

Matematiska uttryck skrivs enligt vanliga matematiska regler, t ex parenteser används för att ändra på i vilken ordning beräkningar utförs. Vid multiplikation måste man dock alltid skriva ut multiplikationstecknet, som följande exempel visar

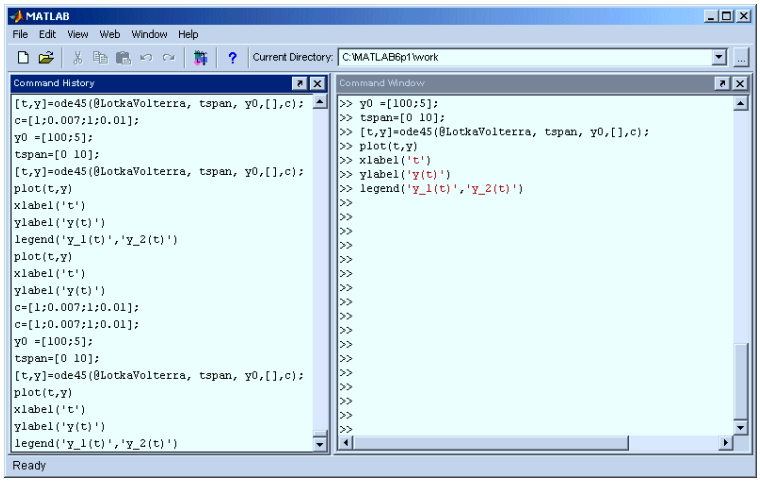
```
>> 2*x*(y + 1)
```

```
ans =
      70
```

Man kan om man vill skriva flera kommandon efter varandra på samma rad. Då avslutar man varje kommando med ett kommatecken eller ett semikolon. Semikolon gör att MATLAB inte skriver ut något på skärmen, men beräkningen utförs ändå.

Att avsluta ett kommando med semikolon, dvs så att ingen utskrift görs, är också användbart när man skriver endast ett kommando per rad. Ofta utförs beräkningarna på stora datamängder och man vill inte att alla tal ska skrivas ut.

Tidigare givna kommandon kan upprepas med hjälp av piltangenterna. Genom att trycka på uppåtpil bläddrar man bakåt till det kommando man är intresserad av. Detta är också smidigt att använda när man skrivit fel och vill slippa skriva om hela kommandot igen. Man kan också använda *command history* som precis som *workspace* kan öppnas ett delfönster. Detta görs genom att markera `Command History` i menyn `View`. Alla dina kommandon en tid bakåt finns samlade där. Genom att dubbelklicka på ett kommando upprepas det i kommandofönstret. De går går också att dra kommandon från `Command History` till kommandofönstret med musen.



Figur 4: MATLAB:s Command history i den vänstra halvan av fönstret.

Om en beräkning tar väldigt lång tid kan det verka som om MATLAB ”hängt sig”. Man kan då avbryta beräkningen genom att trycka `ctrl-c` och promp-

tern visas åter.

MATLAB använder normalt ca 16 decimalers noggrannhet vid beräkningar, men vanligen ser man bara fyra decimaler. Man kan ändra på detta genom att använda kommandot `format`. Kommandot `format long` gör att 14 decimaler visas på skärmen. För att återställa till fyra decimaler ger man kommandot `format short`.

Om man är osäker på vad ett kommando heter eller vad det gör kan man använda kommandot `help`. Vill man t ex veta vilka olika alternativ kommandot `format` har skriver man `help format`. Genom att bara ge kommandot `help` kan man leta sig fram till rätt kommando. Ett alternativ är att klicka på `MATLAB Help` i menyn `Help` som öppnar ett särskilt hjälp-fönster med all upptänklig information.

Små tips

.	punkt används för att beskriva decimaltal istället för kommatecken `(,)` .
;	semikolon skrivs efter ett kommando då man ej vill skriva ut resultatet på skärmen. Beräkningen utförs men inget skrivs ut.
↑	piltangent uppåt skriver ut tidigare kommandon.
`ctrl-c`	avbryter pågående beräkning.
`help`	hjälpkommando, skriv `help` åtföljt av kommando.
`lookfor`	hjälpkommando där man anger ett sökord. MATLAB ger en lista med alla kommandon där sökordet förekommer i deras beskrivning.
`clear`	rensar minnet på variabler. Bra att ha i början av m-filer.
`clc`	tömmer kommandofönstret.
`who`	ger en lista över de variabler som skapats.
`whos`	ger en utförlig lista över de variabler som skapats.
`format long`	visar 14 decimaler mot de 4 som MATLAB normalt skriver ut.
`format short`	återställer till att visa fyra decimaler.
`format short e`	visar fyra decimaler men med s k flytande decimalpunkt.
`format compact`	minskar mellanrummet mellan svarsraderna i kommandofönstret.

1.2 Vektorer/Talföljder

Hittills har endast *skalärer* använts, dvs vanliga enstaka tal som 0, 2.4 och -0.342. I MATLAB arbetar man vanligen med *talföljder*, eller *vektorer*, dvs följder av tal. I hjälptexter och felmeddelanden i MATLAB används det engelska ordet *vector*. Vi kommer därför fortsättningsvis använda begreppet *vektor* för en talföljd.

Att skapa vektorer kan man göra på flera sätt i MATLAB. Ett sätt är att använda kolon (:) för att definiera en följd av värden, s k *kolon-notation*.

```
>> x = 0:7
x =
    0   1   2   3   4   5   6   7

>> y = 0:0.5:4
y =
    0 0.5000 1.0000 1.5000 2.0000 2.5000 3.0000 3.5000 4.0000
```

Här har man skapat en vektor x mellan 0 och 7 som, eftersom inget annat anges, automatiskt får steglängden 1, och en talföljd y mellan 0 och 4 med steglängden 0.5. Kolon-notation är med andra ord bra när man vill skapa vektorer med en viss steglängd utan att antalet steg från start till slut spelar någon roll.

Kolon-notation	
`i:k`	ger en följd av värden från i till k i steg om ett, dvs $i, i+1, i+2, ..., k$. Om $i > k$ får man ingen värdeföljd utan tomma mängden. Talen i och k behöver ej vara heltal utan sista talet i följden är mindre eller lika med k.
`i:j:k`	ger en följd av värden från i till k i steg om j, dvs $i, i+j, i+2j, ..., k$. För $j = 0$ returneras tomma mängden. Talen i, j och k behöver inte vara heltal utan sista talet i följden är mindre eller lika med k.

Ett annat sätt att skapa vektorer är att använda kommandot `linspace`. Det är särskilt lämpligt när man vill skapa vektorer med ett specifikt antal steg utan att bry sig om steglängden.

```
>> a = linspace(0,1,5)
a =
    0  0.2500  0.5000  0.7500  1.0000
```

Här skapas en vektor från 0 till 1 i 5 steg. Lägg märke till att MATLAB använder kommatecken för att skilja de olika talen åt i kommandot.

Linspace	
`linspace(a,b)`	returnerar en vektor med 100 element. Första elementet är a, sista elementet är b och mellanliggande element är jämnt fördelade mellan a och b.
`linspace(a,b,n)`	returnerar en vektor med n element i intervallet a till b.

Man kan också skriva in talen i en vektor "manuellt" genom att använda hakparanteser, `[` och `]`

```
>> t = [0.5 1.2 3.75]
t =
    0.5000  1.2000  3.7500
```

När man utför matematiska beräkningar i MATLAB kan man på samma sätt som en skalär ha en vektor som inparameter till funktioner, dvs som argument. Om t ex a är definierad som i exemplet ovan, kan man skriva

```
>> sin(a)
ans =
      0    0.2474    0.4794    0.6816    0.8415
```

Resultatet blir att att sinusvärdet för varje enskilt tal i vektorn beräknas. Det är dock några saker man måste ta hänsyn till. Man måste tänka sig för när man har en vektor men *inte* vill utföra vektor- eller matrisoperationer utan vill göra beräkningar på ett element i taget i vektorn. I så fall måste man lägga till en punkt framför tecknen för multiplikation `*`, division `/` och upphöjt till `^`. Om man har en vektor x och vill utföra beräkningen $y = x^2$ måste man alltså skriva

```
>> x = [1 2 4 8 16 32 64 128 256];
>> y = x.^2
y =
    1 4 16 64 256 1024 4096 16384 65536
```

Detta eftersom multiplikation mellan två radvektorer inte är definierad. Punkten i `.^` betyder alltså elementvis upphöjt till. Om vi utelämnat punkten i detta exempel får man följande resultat

```
>> y = x^2
??? Error using ==> ^
Matrix must be square.
```

vilket talar om att storleken på matrisen (som är en vektor i detta fall) inte stämmer.
Vid addition och subtraktion klarar man sig däremot utan punkt framför $+$ och $-$. Det behövs inte heller någon punkt när man multiplicerar och dividerar talen i en vektor med en skalär. Att man behöver punkt ibland och ibland inte beror helt enkelt på vad man vill beräkna. Man bör vara medveten om detta och tänka efter när man arbetar med vektorer. Mer om detta kommer att förklaras i kapitel 4.

1.3 Komplexa tal

MATLAB kan också hantera komplexa tal. Komplexa tal uppstår ibland som resultat i beräkningar, men det går också att skapa sådana själv. I MATLAB är variabeln `i` reserverad för den imaginära enheten:

```
>> i
ans =
    0 + 1.0000i
>> sqrt(-1)
ans =
    0 + 1.0000i
```

Om man på egen hand vill skapa komplexa tal, t ex $z = 3.5 - 3.1i$ och $u = -1.5 + 2.5i$ använder man det inbyggda kommandot `complex`

```
>> z = complex(3.5, -3.1)
z =
    3.5000 - 3.1000i
>> u = complex(-1.5, 2.5)
u =
    -1.5000 + 2.5000i
```

Man kan sedan räkna med dessa tal "som vanligt", dvs vanliga räkneregler för komplexa tal gäller

```
>> z + u
ans =
   2.0000 - 0.6000i
>> z*u
ans =
    2.5000 + 13.4000i
>> abs(u)
ans =
   2.9155
```

Det finns flera kommandon för att räkna med komplexa tal, här representerat av $z = u + v \cdot i$.

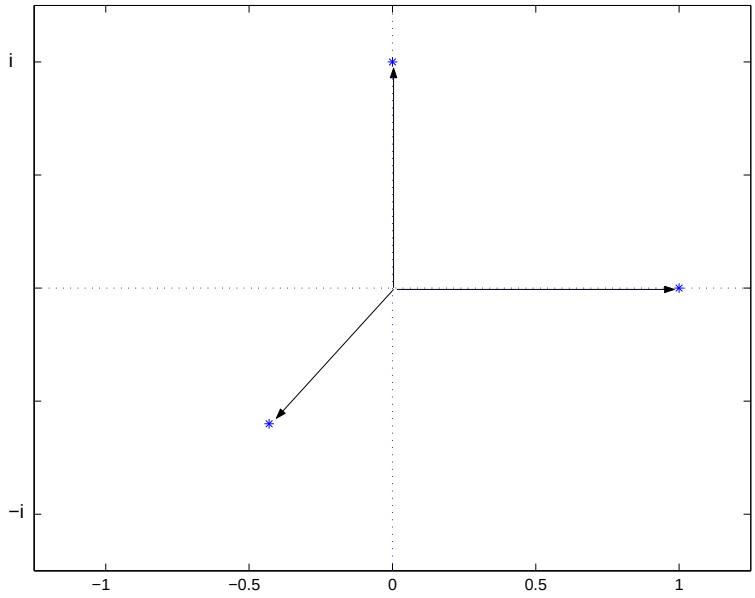
Funktioner för komplexa tal	
`complex(u,v)`	returnerar det komplexa talet $u + v \cdot i$.
`real(z)`	returnerar den realla delen av det komplexa talet z.
`imag(z)`	returnerar den imaginära delen av z.
`conj(z)`	returnerar komplexkonjugatet till z.
`abs(z)`	returnerar beloppet av det komplexa talet z, dvs $\sqrt{u^2 + v^2}$.
`angle(z)`	returnerar fasvinkeln i radiander för z, dvs vinkeln mellan den positiva reella axeln och vektorn som representerar z.

En vektor (talföljd) av komplexa tal kan skapas enligt

```
>> v = complex([0 1 -0.43],[1 0 -0.6])
v =
    0 + 1.0000i     1.0000                -0.4300 - 0.6000i
```

I Figur 5 visas talen i `v` grafiskt i det komplexa talplanet. Man kan arbeta med vektorer av komplexa tal precis som med andra komplexa tal i MATLAB. Fasvinkeln till de tre talen i `v` fås t ex genom

```
>> angle(v)
ans =
    1.5708    0    -2.1926
```



Figur 5: De komplexa talen lagrade i `v`.

Observera att den första vinkeln är detsamma som $\pi/2$. I Figur 5 kan du jämföra fasvinklarna med figuren. Komplexa tal i den tredje och fjärde kvadranten får negativa vinklar. Fasvinkeln ges i radianer, önskar man vinklar i grader istället, multiplicerar man med $\frac{180}{\pi}$

```
>> angle(v)*180/pi
ans =
    90.0000         0  -125.6279
```

Om man istället för cartesiska koordinater önskar polära finns en konverteringsfunktion, `cart2pol`, för den operationen

```
>> [theta,radie] = cart2pol(real(v),imag(v))
theta =
    1.5708         0  -2.1926
radie =
    1.0000    1.0000    0.7382
```

där vi här lagrat fasvinkeln i variabeln `theta` och radien i variabeln `radie`.

1.4 Noll är inte alltid noll

När tal lagras i en dator har man problemet att det enbart finns ett ändligt minnesutrymme för varje tal. Det fungerar bra för heltal upp till en viss

storlek, men hur gör man när man har oändliga decimalutvecklingar? Hur ska man t ex kunna lagra tal som π eller $\frac{1}{3}$? På datorer har man löst detta genom att alla decimaltal avrundas till ett visst antal decimaler. Detta leder till att decimaltal i MATLAB inte lagras exakt, och man säger att man utför beräkningar i en viss *precision*. Exakt samma problematik finns även i miniräknare.

I beräkningsprocesser med upprepade operationer kommer alla resultat i processen hela tiden att avrundas och det innebär att lösningen i princip alltid innehåller ett litet fel. Beräkningen $x = 0.42 - 0.5 + 0.08$ ger i exakt räkning resultatet $x = 0$. Samma beräkning i MATLAB ger

```
>> x = 0.42 - 0.5 + 0.08
x =
   -1.3878e-017
```

dvs $x = -1.3878 \cdot 10^{-17}$. Talet x ska alltså här betraktas som 0 trots att det inte är noll. Det fel som uppstått beror på avrundningar av alla successiva resultat i beräkningen.

Det som styr storleken på vad hur mycket som avrundas kallas *maskinepsilon* eller *avrundningsenheten* och kan i MATLAB hittas med kommandot `eps`

```
>> eps
ans =
   2.2204e-016
```

dvs maskinespilon är $2.2204 \cdot 10^{-16}$. I beräkningar i MATLAB får man alltid ett relativt fel i den storleksordningen beroende på avrunding. Talet `x` i exemplet ovan har ett fel som är aningen mindre än detta. Ett annat exempel på detta är beräkning av $sin(\pi)$ som bör ge resultatet 0. I MATLAB fås

```
>> sin(pi)
ans =
   1.2246e-016
```

Återigen blir felet i samma storleksordning som maskinepsilon.

I de flesta fall är felen så små så att det saknar betydelse. Men en följd av detta är att man aldrig kan testa om tal efter beräkningar är lika med noll eller om två tal är exakt lika. Även om de kan betraktas som lika eller noll finns det alltid avrundningsfel som gör att de inte är det.

1.5 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

1.1 Sätt x=30 och y=15. Prova vad som händer (och försök förstå varför) när du skriver följande kommandon

a) `format long, cos(x)`

b) `format short,  cos(x)`

c) `format short e,  cos(x)`

d) `2y-4`

e) `z=y^2`

f) `who`

g) `clear; 2*y-4`

h) `help whos`

1.2 a) Prova vad som händer när du använder piltangenterna upp, ner och vänster, samt prova att kopiera och klistra in gamla kommandosekvenser vid promptern.

b) Öppna delfönstret *Command history* och pröva att dubbelklicka på ett gammalt kommando. Vad händer? Pröva också att med musen dra ett kommando från *Command History* till kommandofönstret.

1.3 Sätt x=30 och y=15 och beräkna följande tal

a) 4!

b) e^2 (e^x skrivs `exp(x)` i MATLAB)

c) $\sqrt{36}/4$

d) lg(1000)

e) ln(12)

f) $e^{\frac{2-x}{y-2}}$ (OBS! Använd parenteser)

1.4 Hur skapar du bäst en vektor w

a) som är mellan 0 och 1 med steglängden 0.3. Finns talet 1 med i talföjden?

b) som består av 11 tal, startar vid 0 och som slutar vid 1?

b) Vad händer om du skriver `w^2`? Varför?

1.5 Skapa en vektor y mellan 0 och 10 och beräkna

a) $z = \sqrt{y}$

b) $z = e^{(y/10)^2}$

c) $z = \sin(\pi y/8)$

d) $z = 1/y$

Kontrollera att beräkningarna stämmer.

1.6 Skapa två komplexa tal $z = 2 + 5.5i$ och $u = 2 - 5.5i$.

a) Båda talen bör ha samma längd. Undersök detta i MATLAB genom att använda `abs`.

b) Beräkna summan av `z` och `u`.

c) Beräkna produkten av `z` och `u`.

d) Skriv ut realdelen till `z` genom att använda `real`.

e) Skriv på samma sätt ut imaginärdelen till `z`.

f) Beräkna fasvinkeln för `z` och `u`

1.7 Fasvinkeln för talen $1, i, -1, -i$ bör bli $0, \frac{\pi}{2}, \pi, -\frac{\pi}{2}$. Lagra de fyra talen $1, i, -1, -i$ i en vektor v och undersök detta i MATLAB (använd `angle` på vektorn)

2 Enkel grafik

I MATLAB ingår en stor mängd kommandon för avancerad grafik i två och tre dimensioner. I det här kapitlet visas främst hur man skapar enklare grafik. Den som vill se lite mer avancerade exempel kan köra de demonstrationsprogram som finns i MATLAB, där många av de olika grafikkommandona visas. Detta program startas genom att klicka på `demos` i menyn `Help`.

2.1 Att rita en graf

För att rita en graf till en funktion skapar man först två lika långa vektorer, t ex x och y, där x är den horisontella axeln och y motsvarande funktionsvärden.

För att skapa vektorn x används lämpligen kolon-notation eller kommandot `linspace` (se avsnitt 1.2). Därefter skapar man vektorn y genom att skicka x som parameter till den funktion man vill rita upp. Sedan är det bara att skriva `plot(x,y)` så ritas grafen upp. Nedan ges ett enkelt exempel där sinus ska ritas (plottas) för värden från 0 till 8:

```
x = 0:8; y = sin(x); plot(x,y)
```

Ett grafikfönster öppnas och kurvan ritas ut. Skalningen av axlarna utförs automatiskt. I exemplet användes nio x-värden $(0, 1, \ldots, 8)$. Som du kan se i Figur6 ger det en lite hackig, ojämn kurva. För att få en jämnare kurva måste man använda fler x-värden, t ex 100. Då är det lämpligare att använda kommandot `linspace` istället:

```
x = linspace(0,8); y = sin(x); plot(x,y)
```

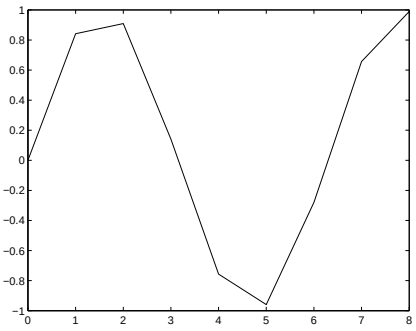
Resultatet, som visas visas i Figur 7 blir betydligt bättre.

Det är viktigt att de vektorer man vill rita mot varandra är av samma längd, dvs för varje x-värde måste det finnas ett motsvarande y-värde. Är de inte det ger MATLAB felmeddelandet

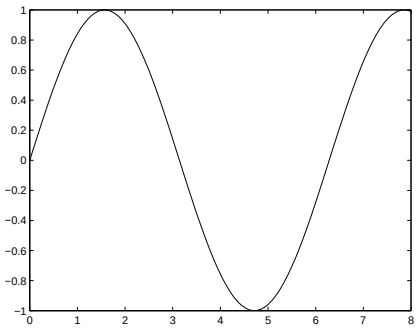
```
Vectors must be the same lengths.
```

Det är lätt att ta reda på en vektors längd genom att använda kommandot `whos` eller använda MATLAB:s workspace, som nämnts tidigare på sidan 7.

Det går att specificera olika linjetyper och färger på kurvan. Detta anger man i så fall i slutet av kommandot, se rutan *Plot-kommandon*. Den linjetyp



Figur 6: Funktionen $y = sin(x)$ med 9 värden.



Figur 7: Funktionen $y = sin(x)$ med 100 värden.

och färg man vill ha anges mellan två apostrofer, t ex `'r--'`, vilket talar om att man vill visa grafen som en röd streckad linje. Exemplet ovan skrivs alltså istället som `plot(x,y,'r--')` eller `fplot('sin',[0 8],'r--')`. Vilka färger och linjetyper man kan välja mellan finns specificerat i rutan *Punkt-, linje- och färgtyper*. Man kan också använda kommandot `help plot` i MATLAB.

Man kan också ändra i grafen interaktivt i efterhand, t ex byta färg och linjetyp. Detta beskrivs i avsnitt 2.7.

Punkt-, linje- och färgtyper

	Punkttyper:		**Linjetyper:**
`.`	punkt	`-`	heldragen linje
`*`	stjärna	`--`	streckad linje
`square`	fyrkant	`-.`	punkt-streckad linje
`diamond`	ruta	`:`	prickad linje
`pentagram`	femuddig stjärna		**Färgtyper:**
`hexagram`	sexuddig stjärna	`g`	grön
`o`	gement o, ringar	`m`	magenta
`+`	plustecken	`b`	blå
`x`	kryss	`c`	cyan
`<`	vänsterpekande triangel	`w`	vit
`>`	högerpekande triangel	`r`	röd
`^`	uppåtpekande triangel	`k`	svart
`v`	nedåtpekande triangel	`y`	gul

Plot-kommandon	
`plot(x,y)`	ritar vektorn y mot x. De ordnade talparen (x_j, y_j) ritas ut. Den horisontella axeln är x-axeln och den lodräta är y-axeln.
`plot(y)`	ritar de ordnade talparen (j, y_j). Den horisontella axeln är j-axeln och den lodräta är y-axeln.
`plot(x,y,'g*')`	ritar vektorn y mot x som ovan men 'g*' anger att färgen ska vara grön och att det ska vara stjärnor istället för heldragen linje. Andra färger och utseenden som linjen kan anta anges i rutan *Punkt-, linje- och färgtyper*.
`semilogx(x,y)`	som `plot` men med logaritmisk skala (10-logaritm) på x-axeln.
`semilogy(x,y)`	som `plot` men med logaritmisk skala (10-logaritm) på y-axeln.
`loglog(x,y)`	som `plot` men med logaritmisk skala (10-logaritm) på båda axlarna.

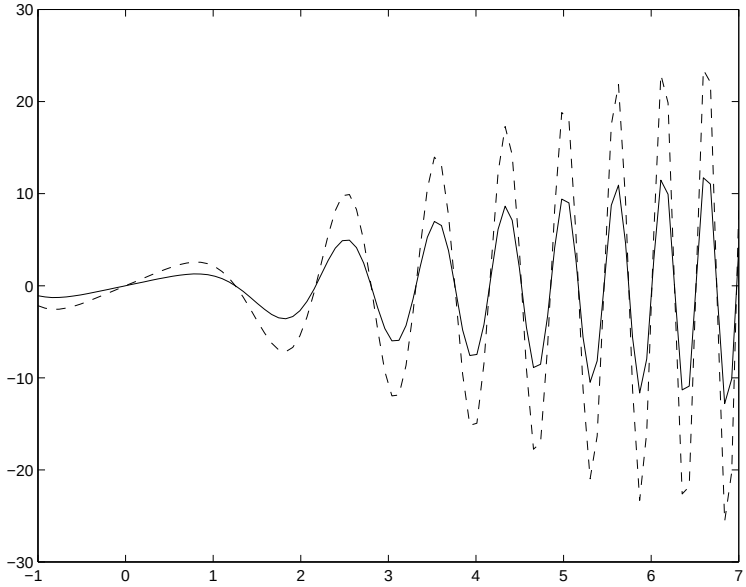
2.2 Flera grafer i samma fönster

När man ritar en graf med `plot` kommer MATLAB att stänga ett eventuellt redan öppet grafikfönster. Om man vill rita fler grafer i samma fönster måste man därför instruera MATLAB att låta fönstret förbli öppet, och det gör man med kommandot `hold on` efter att den första grafen är ritad. Med kommandot `hold off` "släpper" man sedan fönstret så att nästa graf hamnar i ett nytt fönster.

Antag som exempel att graferna till funktionerna $y = 2xcos(x^2)$ och $z = 4xcos(x^2)$ ska studeras i intervallet $-1 < x < 7$ och att kurvorna ska ritas i samma koordinatsystem. Koden för detta blir

```
x = linspace(-1,7); y = 2*x.*cos(x.^2); z = 4*x.*cos(x.^2);
plot(x,y); hold on; plot(x,z,'--'); hold off
```

Graferna visas i Figur 8. Observera här att semikolon används när x, y och z beräknas, dvs vi får ingen utskrift av dessa värden. Eftersom resultatet är hundra värden vardera och vi inte är intresserade av siffervärdena, utan snarare av den plot som ska ritas, är det lämpligt att stänga av utskriften.



Figur 8: Två grafer i samma koordinatsystem.

2.3 Att modifiera grafik

När man väl ritat sina kurvor kan man göra grafiken snyggare på olika sätt, t ex lägga till axelbeteckningar, titel och förklaringsruta. Det går även att lägga till ett rutnät och ändra skalan på axlarna.

Som nämnts tidigare kan man modifiera grafiken interaktiv, genom att klicka i menyer och dialogrutor i grafikfönstret (se avsnitt 2.7). Men om man automatiskt i en längre MATLAB-kod ska generera grafik med axelbeteckningar etc kan man inte använda menyer utan måste använda kommandon. Det är därför viktigt att känna till åtminstone några av de enklare kommandona.

Exemplet med de två graferna i samma grafikfönster i det föregående avsnittet kan förbättras med följande kommandon:

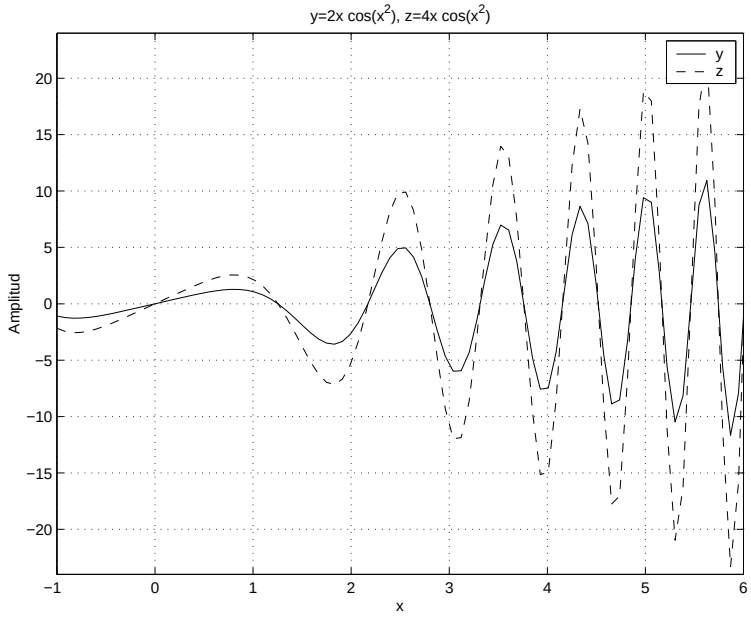
```
xlabel('x'); ylabel('Amplitud')
grid on
title('y=2x cos(x^2), z=4x cos(x^2)')
axis([-1 6 -24 24])
legend('y','z')
```

Resultatet visas i Figur 9. Man har på detta sätt fått axelbeteckningar (`xlabel`, `ylabel`) rutnät (`grid on`) och titel (`title`). Dessutom skalas ko-

ordinatsystemet om (`axis`) så att x-axeln visar intervallet $-1 \leq x \leq 6$ och y-axeln $-24 \leq y \leq 24$. Slutligen får man en förklaringsruta längst upp till höger i figuren (`legend`), där man talar om vilken av kurvorna som motsvarar vilken ekvation.

Kommandona ovan finns sammanfattade i rutan *Diverse grafikkommandon*. För utförligare beskrivning av något grafikkommando använd hjälpkommandot `help` för respektive kommando. Bra översikt över tillgängliga grafikkommanon får man också med `help graph2d` och `help graphics`.

Man kan även ändra teckensnitt och skriva matematiska tecken. Information om detta finns i Appendix B.



Figur 9: Två grafer i samma koordinatsystem, efter modifiering.

Det kan här tilläggas att det finns ett kommando som tar bort själva graferna men som behåller titel, axelbeteckningar, osv. Genom att ge kommandot `cla` suddas ritområdet, dvs koordinatsystemet, och nya grafer kan ritas. Grafikfönstret måste dock hållas kvar med `hold on`, annars kommer ett nytt grafikfönster öppnas.

Diverse grafikkommandon	
`axis([xmin xmax ymin ymax])`	skalar om grafen inom de gränser som angea av $xmin$, $xmax$, $ymin$ och $ymax$.
`xlabel(' ')`	sätter ut texten innanför apostroferna på grafens x-axel (den horisontella axeln), centrerat.
`ylabel(' ')`	sätter ut texten innanför apostroferna på grafens y-axel (den vertikala axeln), centrerat.
`title(' ')`	sätter texten innanför apostroferna som titel på grafen, centrerat.
`legend('a','b', ...osv)`	skriver en ruta med förklaringar till kurvorna i den aktuella grafen.
`grid on`	ritar ut ett rutmönster i grafikfönstret
`grid off`	tar bort rutnätet från grafikfönstret.
`cla`	Betyder "clear current axis" och suddar aktuellt koordinatsystem i en figur.
`clf`	Betyder "clear figure" och suddar hela figuren.

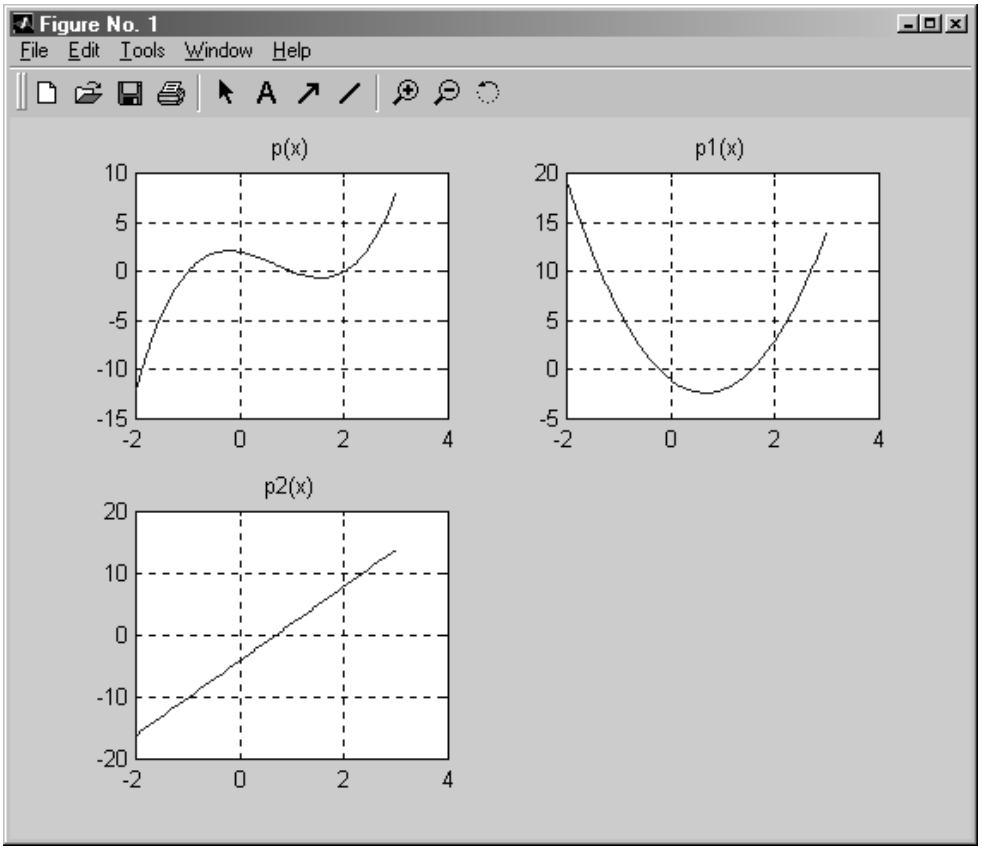
2.4 Rita i delfönster

I avsnitt 2.2 visades att man kan rita flera grafer i samma fönster. Där användes ett koordinatsystem per fönster, men man kan också lägga flera separata koordinatsystem i ett och samma grafikfönster.

Kommandot `subplot(m,n,p)` delar in grafikfönstret i $m \times n$ delfönster, s k *subplots*, där varje delfönster innehåller var sitt koordinatsystem. Variabeln p anger vilket delfönster som ska användas, dvs var nästa graf hamnar.

För att visa användningen av delfönster ska graferna till polynomet $p(x) = x^3 - 2x^2 - x + 2$ och dess två första derivator $p'(x)$ och $p''(x)$ ritas i intervallet $-2 \leq x \leq 3$. Varje graf ska ritas i ett eget delfönster.

```
x = linspace(-2,3);
p = x.^3-2*x.^2-x+2;
p1 = 3*x.^2-4*x-1;
p2 = 6*x-4;
subplot(2,2,1); plot(x,p); title('p(x)'); grid on
subplot(2,2,2); plot(x,p1); title('p1(x)'); grid on
subplot(2,2,3); plot(x,p2); title('p2(x)'); grid on
```



Figur 10: Exempel på grafikfönster med fler koordinatsystem.

I kommandosekvensen ovan skapas en vektor x för intervallet $-2 \leq x \leq 3$ (x-axeln i graferna) och de tre polynomen lagras i variablerna p, p_1 och p_2 (själva derivieringarna görs alltså för hand).

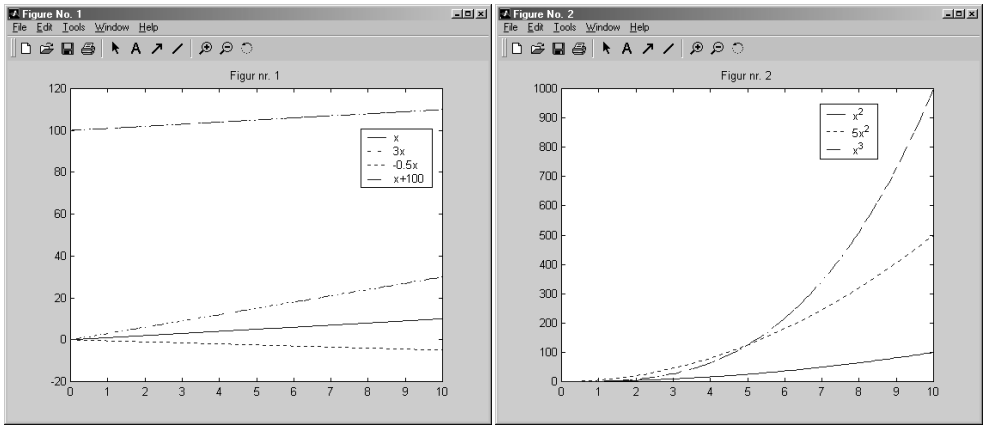
I kommandot `subplot(2,2,1)` talar första 2:an om att man vill skapa plats för två rader av grafer (dvs i vertikalled) medan den andra 2:an talar om att man vill skapa plats för två kolumner av grafer (dvs i horisontalled), alltså totalt 4 stycken grafer. Siffran 1 på slutet talar om vilket av delfönstren man vill skriva i nu, dvs det 1:a i det här exemplet. Numren på delfönstren beräknas från vänster till höger och uppifrån och ner, med övre vänstra hörnet som delfönster nr 1. I det första delfönstret ska polynomet $p(x)$ ritas, därför följs kommandot `subplot(2,2,1)` av `plot(x,p)`.

2.5 Flera grafikfönster öppna samtidigt

I MATLAB är det också möjligt att använda flera grafikfönster samtidigt. Till varje grafikfönster som öppnas i MATLAB knyts ett heltal. Med kommandot `figure(n)` öppnar man ett nytt grafikfönster med nummer n, eller, om det redan finns ett fönster med det numret, visar fönstret längst fram på skärmen. I båda fallen blir grafikfönster n det *aktuella* fönstret, nästa grafikkommando kommer att påverka just det fönstret.

Nedan visas ett exempel på hur man skapar två grafikfönster och ritar några enkla grafer i dem. Resultatet kan beskådas i Figuren 11.

```
x = linspace(0,10);
y1 = x; y2 = 3*x; y3 = 0.5*(-x); y4 = x + 100;
y5 = x.^2; y6 = 5*x.^2; y7 = x.^3;

figure(1), clf;
plot(x,y1,'-'); hold on
plot(x,y2,'-.'); plot(x,y3,':'); plot(x,y4,'--')
legend('x','3x','-0.5x','x+100')
title('Figur nr. 1')

figure(2), clf;
plot(x,y5,'-'); hold on
plot(x,y6,':'); plot(x,y7,'--')
legend('x^2','5x^2','x^3')
title('Figur nr. 2')
```



Figur 11: Två grafikfönster samtidigt.

Två kommandon som hanterar fönster som är värda att känna till är `clf` som suddar aktuellt grafikfönster och `gcf` som returnerar numret på aktuellt grafikfönster. Se också rutan *Fönsterhantering*.

Fönsterhantering	
`hold on`	håller kvar aktuellt grafikfönster. Detta gör att man kan rita in flera figurer i samma fönster som skalas om efter den "största" figuren.
`hold off`	släpper aktuellt grafikfönster.
`subplot(m,n,p)`	delar upp grafikfönstret i ett rutnät med m rader och n kolumner av små delfönster och väljer ut fönster nr p som aktuellt grafik-fönster. Här är m, n och p heltal. Delfönstren numreras från vänster till höger, uppfrån och ned. Kan även skrivas `subplot(mnp)`,men då måste m, n och p vara ensiffriga heltal.
`subplot`	återställer det normala fallet, dvs ett odelat fönster. Samma som `subplot(111)`.
`figure(n)`	visar/skapar grafikfönster nummer n. Kommandot kan användas både för att växla mellan olika grafikfönster och till att skapa nya.
`gcf`	betyder "get handle to current figure" och returnerar det aktuella grafikfönstrets nummer.
`shg`	betyder "show graph" och lägger aktuellt grafikfönster längst fram.

2.6 Att spara grafik

Man kan spara grafik för att sedan läsa in den igen vid ett senare tillfälle. Enklast är att under menyn *File* i grafikfönstret välja alternativen *Save* respektive *Open*. När man använder med med `save` sparas grafiken i ett speciellt format och kan bara öppnas i MATLAB igen.

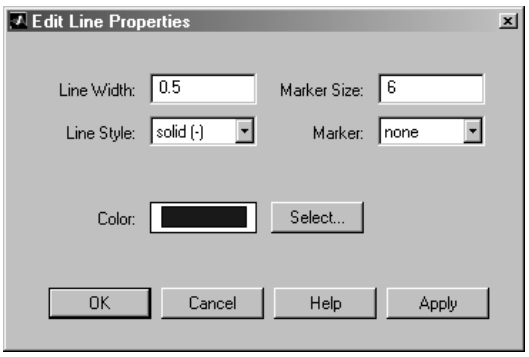
Om man vill lägga in en MATLAB-bild i något annat program, t ex i ett Word-dokument, kan man exportera grafiken till en rad olika grafikformat. Det gör man också i grafikfönstrets *File*-meny genom att välja alternativet *Export*. Det finns ett stort antal format att välja på när man på detta sätt exporterar figurer, t ex jpeg, tiff etc.

2.7 Att modifiera grafik interaktivt

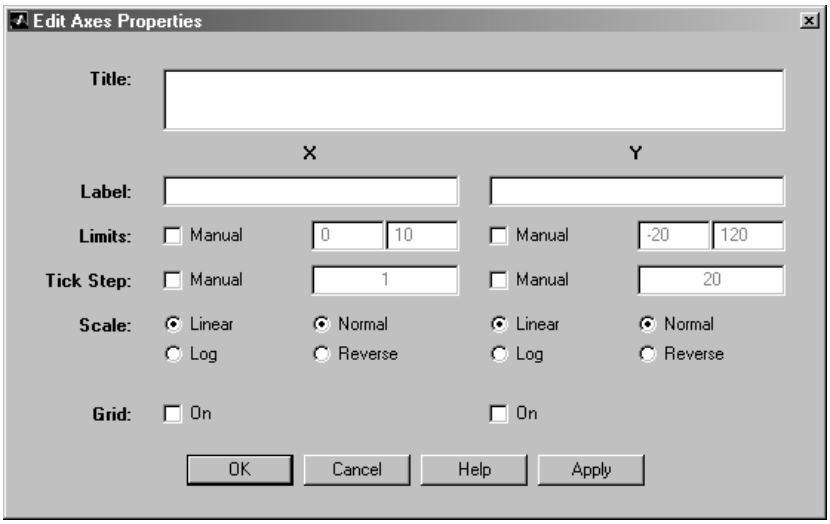
Man kan modifiera grafiken interaktivt i efterhand. Detta gäller inte äldre versioner av MATLAB, utan version 5.3 och framåt.

När man har ritat sin graf och ska modifiera den måste man först markera *Edit Plot* som finns i menyn *Tools* i grafikfönstrets överkant. Därefter är det fritt fram att klicka i grafikfönstret.

För att ändra färg och utseende på en kurva dubbelklickar man på kurvan och man får upp ett fönster liknande det som visas i Figur 12 (det exaka utseendet på fönstret kan variera beroende på MATLAB-version). På samma sätt dubbelklickar man någonstans i koordinatsystemet för att ändra skalningen på axlarna eller lägga till titel, axelbeteckningar, rutnät etc, se Figur 13.



Figur 12: Fönster för att ändra egenskaper hos linjer.



Figur 13: Fönster för att ändra hela ritområdets egenskaper.

I grafikfönstret kan man även lägga till förklarande text, linjer och pilar till sin graf. Likaså kan man förstora, förminska och rotera grafiken. Alla dessa funktioner väljs enkelt via knapparna som är placerade direkt under menyerna i grafikfönstret.

2.8 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

2.1 Rita grafen till funktionen $y = \arctan|x|$ för intervallet $-2\pi \leq x \leq 2\pi$. Lägg till titel och axelbeteckningar.

2.2 Rita upp graferna till nedanstående funktioner i intervallet $-2 \leq x \leq 2$ i separata delfönster.

a) $y_1 = x^2 + 3x - 4$

b) $y_2 = 3x - 1 - 2x^2$

c) $y_3 = 3x - 2x^2 - 2$

d) $y_4 = \frac{3}{2}x^2 - \frac{1}{2}x - 1$

2.3 Skapa vektorn $0 \leq x \leq 2\pi$ (π skrivs `pi` i MATLAB). Beräkna funktionerna nedan och rita upp dem i samma koordinatsystem med olika linjetyper. Rita också ett rutnät och prova att zooma i bilden.

a) $y_1 = \sin(2x)$;

b) $y_2 = 2\sin(x)$;

c) $y_3 = \sin^2(2x)$;

d) $y_4 = \sin(x + \pi/2)$;

2.4 Skapa vektorn $-2 \leq x \leq 2$. Beräkna funktionerna $y = x^k$ för $k = 1, 2, 3, 4$. Rita funktionerna i 4 olika delfönster med hjälp av `subplot`. Vad är det för skillnad mellan udda och jämna exponenter och lägg till en liten text ovanför varje kurva?

2.5 Rita graferna $y = \sin(x)$ och $y = \frac{1}{3}\sin(3x)$ i samma koordinatsystem. Genom att addera termer kan du också rita $y = \frac{1}{3}\sin(x) + \sin(x)$. Fortsätt att addera termer $\frac{1}{5}\sin(5x)$, $\frac{1}{7}\sin(7x)$, osv Vilken form får kurvan?

2.6 Skapa en vektor z med de två komplexa talen $z_1 = 3+4i$ och $z_2 = 3-4i$. Rita sedan ut dessa i det komplexa talplanet med punkttyp stjärna.
Tips: I plotkommandot `plot(x,y)` måste x- och y-axel anges. Här blir x-axel realdelen av talen och y-axel imaginärdelen eftersom du ska rita i komplexa talplanet.

3 M-filer

Antalet kommandon har, som du sett i exemplen i föregående kapitel, nu börjat bli några fler än tidigare. Vad gör man om man i en figur vill ändra lite, t ex ändra i Figur 11 från $0 \leq x \leq 10$ till $0 \leq x \leq 2$? Eller om man måste sluta för dagen och börja igen imorgon? Måste man skriva om alla kommandon igen? Nej, lösningen är att lagra kommandona som genererar figuren i en s k *m-filer*.

3.1 Vad är en m-fil?

En m-fil är en vanlig textfil där man skriver de satser/kommandon som MATLAB ska utföra. Detta gör man istället för att skriva dem i MATLAB:s kommandofönster. Ändelsen `.m` är till för att MATLAB ska kunna skilja filen från andra sorters filer och krävs för att MATLAB ska "hitta" filen. Resten av filnamnet, det som kommer före `.m` namnger man själv. Namnet får inte innehålla mellanslag, punkt (förutom i `.m` förstås) eller några svenska tecken, dvs å, ä, ö, Å, Ä, Ö.

När man skrivit in och sparat sina kommandon i en m-fil kan man köra det från MATLAB:s kommandofönster genom att skriva namnet på m-filen *utan* `.m` vid MATLAB-promtern. Kommandona kommer då att utföras uppifrån och ner precis som de står i m-filen.

Hur skriver man in kommandona i m-filen? Man kan använda vilken texteditor som helst, men enklast är det kanske att använda den editor som finns inbyggd i MATLAB. Den startas genom att markera *New*, sedan *M-file* i menyn *File* i MATLAB:s kommandofönster. Det går också att skriva kommandot `edit` vid MATLAB-promtern. Det öppnas då ett särskilt editeringsfönster, se Figur 2 på sidan 2. Kom ihåg att när man sparar filen ska namnet sluta på `.m`.

Som exempel kan man med MATLAB:s texteditor skriva in följande tre satser

```
x = linspace(0,2*pi);
y = sin(x);
plot(x,y)
```

och spara filen med ett lämpligt namn, t ex `sinuskurva.m`. I MATLAB:s kommandofönster skriver man `sinuskurva` varpå satserna utförs och det ritas upp en sinus-kurva för $0 \leq x \leq 2\pi$. Om man nu exempelvis vill ändra x-axeln

från $0 \leq x \leq 2\pi$ till något annat ändrar man i första raden i `sinuskurva.m` (med MATLAB:s texteditor) och kör programmet igen. Man slipper alltså skriva om alla kommandon igen.

3.2 Kommentarer i m-filer

Det hör till god programmeringssed att kommentera sin kod, så att man själv eller någon annan vid ett senare tillfälle förstår vad programmet gör och hur det fungerar.

I m-filer fungerar procenttecknet som en markering för en kommentar. En kommentar börjar efter `%` och fortsätter raden ut. MATLAB hoppar över det som står efter procenttecknet, dvs det tolkas inte som MATLAB-kommandon.

Om man har kommentarer i början av sitt program `filnamn.m` så kommer de att visas när man skriver `help filnamn` vid MATLAB-prompten. Syftet är att kommentarerna i början av m-filen ska förklara vad programmet gör och hur man använder det.

Nedan visas ett exempel på hur det kan se ut när man kommenterar ett program som beräknar och ritar $f(x) = x^2$, där $-2 \leq x \leq 2$. Programmet antas vara sparat som `komEx.m`.

```
% Exempel på hur man kommenterar MATLAB-kod.
%
% Programmet beräknar f(x)=x^2 och ritar ut kurvan på
% ett intervall.

a = -2; b = 2;                  % Definera gränser
x = linspace(a,b);              % Skapa x-axel

f = x.^2;                       % Beräkna funktionsvärdet i
                                % alla punkter

plot(x,f)                       % Rita kurvan
title('En andragradskurva')     % Namnge plot o axlar
xlabel('x')
ylabel('f(x)')
```

Genom att skriva `help komEx` vid MATLAB-promptern skrivs en hjälptext om m-filen ut på skärmen.

```
>> help komEx

  Exempel på hur man kommenterar MATLAB-kod.

  Programmet beräknar f(x)=x^2 och ritar ut kurvan på
  ett intervall.
```

Notera att det bara är de första kommentarerna som finns på rader utan MATLAB-kommandon på som skrivs ut.

3.3 Spara m-filer på rätt plats

Om man försöker köra sin m-fil och får felmeddelandet

```
??? Undefined function or variable 'filnamn'.
```

så betyder det att MATLAB inte hittar m-filen. Anledningen är att MATLAB bara söker igenom vissa kataloger efter m-filer och om man har sparat programmet i någon annan katalog, t ex på diskett eller en katalog man själv skapat, så tror MATLAB att m-filen inte finns.

För att veta vilken katalog man befinner sig i kan man markera *Current Directory* i menyn *View*. MATLAB-fönstret delas då i två och i den vänstra delen, som heter *Current Directory*, ser man vilken katalog man befinner sig i och vilka filer som finns där. Om allt är korrekt ska de m-filer du sparat synas där. Om man befinner sig i fel katalog kan man byta aktuell arbetskatalog. Detta gör man enklast genom att använda knapparna i *Current Directory*.

För den som är van vid UNIX eller Linux finns det alternativa sätt baserade på UNIX-kommandon. Genom att skriva kommandot `pwd` (*Print Working Directory*) i MATLAB:s kommandofönster skrivs aktuell katalog ut på skärmen. På samma sätt ger kommandot `ls` (*List*) en lista över alla filer som finns i katalogen. Kommandot `cd katalognamn` medför att `katalognamn` blir aktuell katalog.

3.4 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

Nedanstående är några uppgifter är från avsnitt 2. Gör dessa igen men nu genom att skriva in koden i m-filer. Kör sedan m-filerna för att generera grafiken.

3.1 a) Rita grafen till funktionen $y = \arctan|x|$ för intervallet $-2\pi \leq x \leq 2\pi$. Lägg till titel och axelbeteckningar.

b) Ändra intervallet till $-4\pi \leq x \leq 4\pi$ genom att ändra i m-filen. Spara och rita igen.

3.2 a) Skapa vektorn $-2 \leq x \leq 2$. Beräkna funktionerna $y = x^k$ för $k = 1, 2, 3, 4$. Rita funktionerna i 4 olika delfönster med hjälp av `subplot`.

b) Placera in intervallgränserna först i m-filen och ändra `linspace`-kommandot genom att skriva

```
a = -2;  % undre intervallgräns
b = 2;   % Övre intervallgräns
x = linspace(a,b);
```

Ändra sedan intervallet till $-1 \leq x \leq 1$ genom att ändra i `a` och `b` m-filen och rita igen. Pröva några olika intervallgränser.

4 Linjära ekvationssystem och matriser

Matriser är väldigt användbara i många olika sammanhang. MATLAB, som står för MATtrix LABoratory, är från början avsett att främst hantera matriser och är därför ett mycket användbart redskap. Matriser och vektorer är själva grundstrukturen i MATLAB och det är därför mycket viktigt att detta avsnitt behärskas. Även om en del begrepp förklaras är grundläggande kunskaper från algebra är en fördel.

4.1 Skapa matriser

En matris är en rektangulär struktur av tal av följande utseende

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Matrisen har m stycken rader och n stycken kolonner, dvs en $m \times n$-matris.

Antag att följande 3×4-matris ska skapas i MATLAB:

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}.$$

Man skriver då

```
A = [1 2 3 4; 5 6 7 8; 9 10 11 12]
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

Rader skiljs åt med hjälp av semikolon (;) och matrisen börjar och avslutas med hakparanteser ([respektive]).

En *radmatris*, dvs en matris som bara består av en enda rad, är detsamma som det som i föregående kapitel kallades för vektor eller talföljd och kallas vanligen *radvektor*. En radvektor $r = (\ 1 \quad 2 \quad 3 \quad 4\ )$ kan t ex skapas genom

```
r = [1 2 4 4]
r =
     1     2     3     4
```

Talen skiljs åt med ett mellanslag eller kommatecken.

Kolonnmatriser, eller *kolonnvektorer*, kan skapas genom

```
k = [5; 6; 7]
k =
     5
     6
     7
```

eftersom semikolon här betyder radbyte.

Ett element i en matris kommer man åt genom `A(i,j)`, där i är radnumret och j är kolonnumret på elementet. Om man för exempelmatrisen A ovan skriver

```
A(2,2)
```

skriver MATLAB ut

```
ans =
     6
```

Med denna notation kan man också ändra värdet på ett element i matrisen.

```
>> A(2,2) = 0
A =
     1     2     3     4
     5     0     7     8
     9    10    11    12
```

I MATLAB används kolon för att definiera en följd av värden. Det används t ex i kolonnotationen som visades i avsnitt 1.2. Man kan också skapa nya matriser och delmatriser med hjälp av den notationen.

```
>> B = A(1:3,1:2)
B =
     1     2
     5     0
     9    10
```

Matrisen B består nu av rad 1 till 3 och kolonn 1 till 2 från matrisen A. Notera att vanliga parenteser används för att plocka ut delar av en matris och hakparenteser för tilldelning av matriser.

Sammanfattning av delmatriser	
`A(i,j)`	ger elementet på plats (i, j) i matrisen A.
`A(:,j)`	ger den j:te kolonnen i A, dvs kolonn j och alla rader
`A(i,:)`	ger den i:te raden i A, dvs rad i och alla kolonner.
`A(:,j:k)`	ger en delmatris av A bestående av kolonnerna j till k.
`A(i:r,:)`	ger en delmatris av A bestående av raderna i till r.
`A(i:r,j:k)`	ger en delmatris av A bestående av elementen i raderna i till r och kolonnerna j till k.
`A(:)`	ger A i en enda lång kolonn genom att koppla ihop kolonnerna i A.

I MATLAB finns en rad olika matriser fördefinierade. Enhetsmatrisen kan t ex skapas genom `eye`-kommandot, t ex fås en 4×4 enhetsmatris med

```
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

För att skapa en matris med bara nollor kan `zeros`-kommandot användas och för en matris med bara ettor används `ones`-kommandot.

4.2 Matrisalgebra

I MATLAB kan alla matematiskt tillåtna räkneoperationer på matriser utföras. Det är viktigt att tänka efter vilka dimensioner, antal rader och kolonner, matriserna måste ha för att operationen ska kunna utföras.

Med hjälp av kommandot `size(A)` får man antalet rader och kolonner för matrisen A. Är man intresserad av antalet element i en vektor används kommandot `length`. Se rutan *Matrisstorlek* nedan. Man kan förstås också använda Workspace i MATLAB eller kommandot `whos` som ger en lista över de variabler som skapats tillsammans med information om deras storlek (se sidan 7).

Matrisstorlek	
`size(A)`	ger en radmatris med storleken av matrisen A. Det första elementet anger antal rader och det andra antal kolonner.
`length(x)`	ger längden av radmatrisen x.
`length(A)`	ger det största av antalet rader och antalet kolonner.

Addition

Två matriser, A och B, kan enkelt adderas på samma sätt som man adderar två tal, dvs genom `A+B`. Man får inte glömma bort att de båda matriserna måste ha samma antal rader och samma antal kolonner för att kunna adderas. Om man försöker addera två matriser som inte har samma antal rader och kolonner får man felmeddelandet

```
??? Error using ==> +
Matrix dimensions must agree.
```

Multiplikation

Två matriser, A och B, multipliceras på samma sätt som två tal multipliceras, dvs genom `A*B`. För att det ska fungera måste A ha samma antal kolonner som B har rader. Om detta inte gäller får man samma felmeddelande som för addition ovan.
I kapitel 2, sidan 9 användes punktnotation vid multiplikation mellan två vektorer. Skillnaden är att vid punktnotation utförs operationen elementvis. Om man har två matriser

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

så får man

$$A * B = \begin{pmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{11} * b_{12} + a_{12} * b_{22} \\ a_{21} * b_{11} + a_{22} * b_{21} & a_{21} * b_{12} + a_{22} * b_{22} \end{pmatrix},$$

$$A. * B = \begin{pmatrix} a_{11} * b_{11} & a_{12} * b_{12} \\ a_{21} * b_{21} & a_{22} * b_{22} \end{pmatrix}.$$

Om man ska använda punktnotation eller ej beror på vad som ska beräknas.

Transponering

Transponatet av en matris A, som brukar betecknas A^T eller A^*, får man genom `A'`. Transponatet av A är detsamma som att låta raderna och kolonnerna i matrisen A byta plats, dvs om

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix},$$

så är

$$A' = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}.$$

Matrisinvers

Inversen av en matris A får man genom `inv(A)`. För att inversen av en matris ska kunna beräknas måste A vara kvadratisk, dvs ha lika många rader som kolonner, och den måste givetvis vara inverterbar.

4.3 Lösning av ekvationssystem

Det linjära ekvationssystemet

$$\begin{cases} x_1 - 2x_2 &= 0 \\ x_1 - x_2 &= 1 \end{cases}$$

kan skrivas på matrisformen $Ax = b$, där A kallas koefficientmatrisen och innehåller koefficienterna framför x_1 och x_2, dvs

$$A = \begin{pmatrix} 1 & -2 \\ 1 & -1 \end{pmatrix},$$

och b är högerledet,

$$b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Man får alltså

$$Ax = b \Leftrightarrow \begin{pmatrix} 1 & -2 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Om A är inverterbar ser man att lösningen till ekvationssystemet är $x = A^{-1} \cdot b$. I MATLAB kan man alltså skriva `inv(A)*b`. Nu finns det ett effektivare sätt att lösa ekvationssystem med ungefär hälften så många operationer som med invers, nämligen med Gauss-elimination. I MATLAB skriver man `A\b` och detta kallas vänsterdivison och `\` kallas *backslash-operatorn*. Vilket metod som används har inte så stor betydelse för små system, men när man har lite större problemstorlekar får det effekt. Ta därför för vana att använda `\` när du beräknar lösningen till linjära ekvationssystem i MATLAB.

Ett ekvationssystem bestående av två ekvationer kan också representeras grafiskt. Antag att man har

$$\begin{cases} x_1 - 2x_2 &= 0 \\ x_1 - x_2 &= 1 \end{cases}$$

Det består av de två ekvationerna

$$x_1 - 2x_2 = 0 \text{ och } x_1 - x_2 = 1.$$

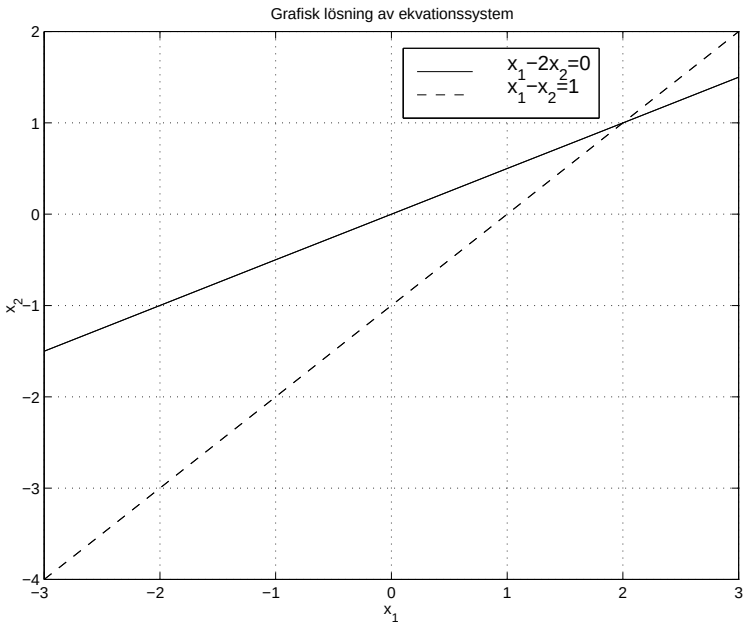
Dessa ekvationer representeras av två räta linjer i x_1, x_2-planet, nämligen $x_2 = x_1/2$ och $x_2 = x_1 - 1$. Man kan rita linjerna genom

```
x1 = linspace(-3,3,10);
x2 = 1/2*x1;
plot(x1,x2,'-')
hold on
x2 = x1 - 1;
plot(x1,x2,'--')

grid on
xlabel('x_1')
ylabel('x_2')
title('Grafisk lösning av ekvationssystem')
legend('x_1-2x_2=0','x_1-x_2=1')
hold off;
```

Grafen finns i Figur 14. Det finns en skärningspunkt mellan linjerna och den är lösningen till ekvationssystemet, nämligen $(x_1, x_2) = (2, 1)$.

Allmänt gäller för ett linjärt ekvationssystem ett av följande alternativ:

1. Systemet har en unik lösning, en unik skärningspunkt.



Figur 14: Grafisk lösning av ekvationssystem $x_1 - 2x_2 = 0,\ x_1 - x_2 = 1$

2. Systemet saknar lösning, linjerna parallella men går ej i varandra.

3. Systemet har oändligt många lösningar, linjerna går i varandra.

När ekvationssystemet som exemplet ovan består av två ekvationer kan man se vilket av fallen som gäller genom att rita motsvarande linjer.

4.4 Exempel: populationsmodeller och övergångsmatriser

Som exempel på användning av matriser och matrisberäkningar visas här en tillämpning från biologin.

Hur en population förändras över tiden kan beskrivas med hjälp av så kallade övergångsmatriser.

Populationen, dvs befolkningen, delas in i fyra grupper:

G_1: alla barn som ännu ej börjat skolan,

G_2: skolbarn och studerande,

G_3: förvärvsarbetande,

G_4: pensionärer.

Man vill studera hur gruppernas storlek förändras efter ett antal år. För enkelhetens skull antar man att ingen inflyttning eller utflyttning sker. Låt $P_i(t)$ vara antalet individer i grupp i vid tidpunkt t, f_i är andelen av G_i som flyttas till G_{i+1} och r_i är den andel som stannar kvar. Alltså är $1-(f_i+r_i)$ den andel individer som av någon anledning lämnar samhället, se Figur 15. Vidare är b_i antalet barn som varje individ i G_i förväntas få under tidsperioden. För G_1 gäller att vid tiden $t+1$ är

$$P_1(t+1) = r_1P_1(t) + b_1P_1(t) + b_2P_2(t) + b_3P_3(t) + b_4P_4(t).$$

För övriga grupper gäller

$$P_i(t+1) = f_{i-1}P_{i-1}(t) + r_iP_i(t), \quad i = 2, 3, 4.$$

På matrisform blir det

$$\begin{pmatrix} P_1(t+1) \\ P_2(t+1) \\ P_3(t+1) \\ P_4(t+1) \end{pmatrix} = \begin{pmatrix} r_1+b_1 & b_2 & b_3 & b_4 \\ f_1 & r_2 & 0 & 0 \\ 0 & f_2 & r_3 & 0 \\ 0 & 0 & f_3 & r_4 \end{pmatrix} \begin{pmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ P_4(t) \end{pmatrix},$$

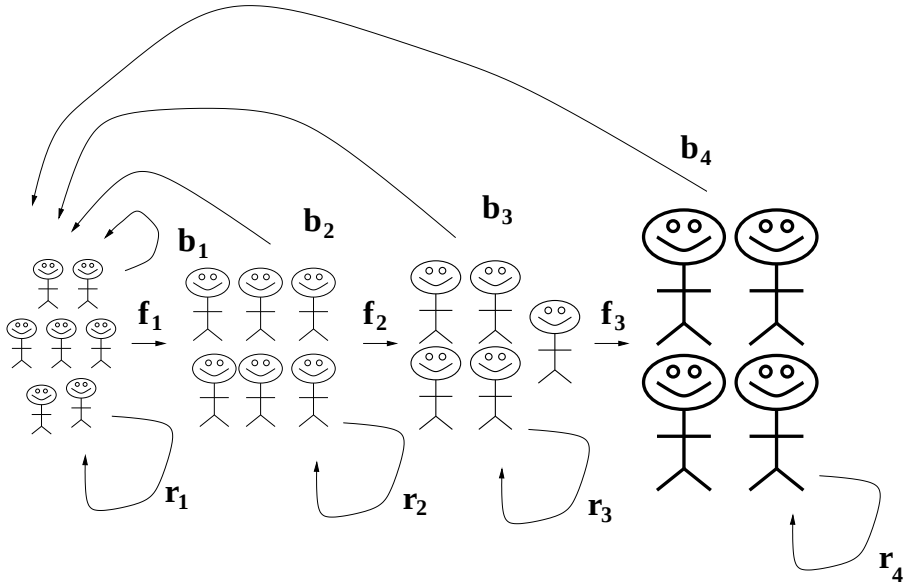
dvs man har $X_{t+1} = LX_t$, för

$$X_{t+1} = \begin{pmatrix} P_1(t+1) \\ P_2(t+1) \\ P_3(t+1) \\ P_4(t+1) \end{pmatrix}, \quad L = \begin{pmatrix} r_1+b_1 & b_2 & b_3 & b_4 \\ f_1 & r_2 & 0 & 0 \\ 0 & f_2 & r_3 & 0 \\ 0 & 0 & f_3 & r_4 \end{pmatrix} \textit{ och } \quad X_t = \begin{pmatrix} P_1(t) \\ P_2(t) \\ P_3(t) \\ P_4(t) \end{pmatrix}.$$

L är övergångsmatrisen. Om man har en startvektor X_0,

$$X_0 = \begin{pmatrix} P_1(0) \\ P_2(0) \\ P_3(0) \\ P_4(0) \end{pmatrix},$$

och värden på koefficienterna $r_1, \ldots, r_4$, $b_1, \ldots, b_4$ och $f_1, \ldots, f_3$ kan man beräkna fördelningen vid en viss tidpunkt t. I Figur 15 visas en schematisk bild över den beskrivna modellen.



Figur 15: Populationsmodellen beskriven i Kapitel 4.4

4.5 Egenvektorer och egenvärden

Egenvärden och egenvektorer bestäms i MATLAB med kommandot `eig`. Om man har en matris A får man egenvärdena genom `eig(A)`. För att få både egenvärden och egenvektorer används `[eigVec,eigVal] = eig(A)`. I detta fall hamnar egenvektorerna som kolonner i matrisen `eigVec` och egenvärdena på diagonalen i matrisen `eigVal`. Man behöver inte använda just namnen `eigVec` och `eigVal`, utan kan namnge dessa s k utparametrar till vad som helst.

Matrisen

$$A = \begin{pmatrix} 1 & -2 \\ 1 & 4 \end{pmatrix}$$

har egenvärden $\lambda_1 = 2$ och $\lambda_2 = 3$ och tillhörande egenvektorer $x_1 = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$ och $x_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. Om man normerar egenvektorerna, dvs dividerar med

vektorernas längd fås

$$x_1 = \frac{1}{\sqrt{(-2)^2+1^2}} \begin{pmatrix} -2 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.8944 \\ 0.4472 \end{pmatrix}$$
$$x_2 = \frac{1}{\sqrt{1^2+(-1)^2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0.7071 \\ -0.7071 \end{pmatrix}$$

I MATLAB blir beräkningarna

```
>> A = [1 -2; 1 4]
A =
     1    -2
     1     4
>> [eigVec,eigVal] = eig(A)
eigVec =
   -0.8944    0.7071
    0.4472   -0.7071
eigVal =
     2     0
     0     3
```

Notera att MATLAB normerar vektorerna.

Egenvärden och egenvektorer kan användas för att undersöka övergångsmatriser. Låt A vara en övergångsmatris med ett positivt egenvärde λ och tillhörande egenvektor X_0. Om X_0 används som startvektor så händer följande då tiden t går mot oändligheten.

1. 0m $0 < \lambda < 1$, så dör populationen ut.

2. 0m $\lambda > 1$, så växer populationen obegränsat.

3. 0m $\lambda = 1$, så är populationens storlek densamma, dvs X_0 är en stabil egenvektor.

Sammanfattning av matriskommandon	
`inv(A)`	ger inversen av matrisen A.
`A'`	ger transponatet av A.
`det(A)`	ger determinanten av A.
`[eigVec, eigVal] = eig(A)`	ger egenvärden (diagonalen i matrisen `eigVal`) och egenvektorer (kolonnerna i matrisen `eigVec`) till A.
`eye(n)`	ger en enhetsmatris av storlek $n \times n$.
`eye(m,n)`	ger en enhetsmatris av storlek $m \times n$.
`ones(n)`	ger en $n \times n$-matris med bara ettor.
`ones(m,n)`	ger en $m \times n$-matris med bara ettor.
`zeros(n)`	ger en $n \times n$-matris med bara nollor.
`zeros(m,n)`	ger en $m \times n$-matris med bara nollor.

4.6 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

4.1 Skapa matrisen

$$A = \begin{pmatrix} 2 & 2 & 7 & 2 \\ 3 & 6 & 4 & 1 \\ 5 & 1 & 3 & 10 \end{pmatrix}.$$

4.2 Skapa en matris B bestående av de första två raderna i A.

4.3 Skapa tre nya kolonnvektorer $c1$, $c2$ och $c3$ som är lika med den första, den andra respektive den tredje kolonnen i matrisen A.

4.4 Skapa en matris C som har kolonnerna $c2$, $c3$ och $c1$ i denna ordning.

4.5 Skapa matriserna

$$A = \begin{pmatrix} 2 & 3 & 1 & -1 \\ 1 & 2 & -1 & 1 \\ 1 & 1 & 2 & -1 \\ 1 & 1 & 1 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 4 & 3 & 2 \\ 2 & 3 & 1 & 4 \\ 3 & 2 & 4 & 1 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

och beräkna `A+B`.

4.6 Skapa matriserna

$$A = \begin{pmatrix} 3 & 5 & 1 & 0 \\ 2 & -2 & 1 & -1 \\ 8 & 0 & 1 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & -1 & -1 \\ 2 & 1 & 0 \\ 3 & 7 & 6 \\ 4 & 4 & 1 \end{pmatrix}$$

och beräkna `A*B`.

4.7 Skapa matriserna

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 1 & 1 & -1 \\ 0 & 2 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{pmatrix}$$

och beräkna `A*B` och `A.*B`. Vilken är skillnaden?

4.8 Skapa matriserna

$$A = \begin{pmatrix} 3 & 4 & 5 \\ 5 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{pmatrix}.$$

Försök att beräkna inversen av A och B. Vad händer och varför? Vad blir `A*inv(A)`?
Ändra utskriftsformatet till `format long e` och beräkna `A*inv(A)` igen. Vilket blir resultatet? Varför blir det inte nollor i element utanför diagonalen? Tips: Om orsaken till detta kan du läsa i avsnitt 1.4, sid 12.

4.9 Skapa matriserna

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & -1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} -2 & 1 \\ 8 & 11 \\ 1 & 2 \end{pmatrix}.$$

Beräkna transponaten av A och B. Beräkna också `A*B`, `(A*B)'` och `B'*A'`.

4.10 Lös ekvationssystemet

$$\begin{cases} x + 3y + 2z - w = 9 \\ 2x + 4y + z + w = 17 \\ 4x + y + z + 2w = 17 \\ -x + 4y + z + w = 14 \end{cases}$$

4.11 Undersök följande tre ekvationssystem grafiskt

$$\begin{cases} 2x_1 + 3x_2 = 1 \\ x_1 - x_2 = 2 \end{cases}, \quad \begin{cases} 2x_1 - 2x_2 = 4 \\ x_1 - x_2 = 2 \end{cases}, \quad \begin{cases} x_1 - x_2 = 3 \\ x_1 - x_2 = 2 \end{cases}$$

4.12 Antag att X_0 för exemplet på sidan 39 är

$$\begin{pmatrix} 10000 \\ 36000 \\ 90000 \\ 18000 \end{pmatrix}$$

och övergångsmatrisen

$$\begin{pmatrix} 0 & \frac{1}{18} & \frac{1}{3} & 0 \\ \frac{9}{10} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{2}{3} & 0 \\ 0 & 0 & \frac{1}{6} & \frac{1}{3} \end{pmatrix}.$$

Tidsperioden är 10 år, dvs $t = 0$ motsvarar år 0, $t = 1$ år 10, $t = 2$ år 20 osv. Beräkna antalet individer i varje grupp vid år 10, 20, 30 och 40. Vad tror du kommer att hända med populationen i framtiden?

4.13 Undersök övergångsmatrisen i övning 3.12 med hjälp av egenvärden och egenvektorer. Vad är populationens tillväxthastighet, dvs vilket är det dominanta egenvärdet? Om något av egenvärdena är komplexa, så får du jämföra med absolutbeloppet av egenvärdet, dvs för $z = a + ib$ med $|z| = \sqrt{a^2 + b^2}$. Absolutbeloppet får du med hjälp av `abs`. Vad kommer att hända om du använder motsvarande egenvektor som startvektor? Undersök om teorin stämmer genom att beräkna antalet individer vid olika år med egenvektorn som startvektor.

4.14 Antag att den högsta ålder som uppnås av honor i en viss djurpopulation är 15 år och man delar in populationen i tre lika klasser om fem år. Låt övergångsmatrisen för denna population vara

$$\begin{pmatrix} 0 & 4 & 3 \\ 0.5 & 0 & 0 \\ 0 & 0.25 & 0 \end{pmatrix}$$

Om det initialt är 1000 honor per åldersklass gäller

$$X_0 = \begin{pmatrix} 1000 \\ 1000 \\ 1000 \end{pmatrix}$$

$$X_1 = LX_0 = \begin{pmatrix} 0 & 4 & 3 \\ 0.5 & 0 & 0 \\ 0 & 0.25 & 0 \end{pmatrix} \begin{pmatrix} 1000 \\ 1000 \\ 1000 \end{pmatrix} = \begin{pmatrix} 7000 \\ 500 \\ 250 \end{pmatrix}$$

$$X_2 = LX_1 = \begin{pmatrix} 0 & 4 & 3 \\ 0.5 & 0 & 0 \\ 0 & 0.25 & 0 \end{pmatrix} \begin{pmatrix} 7000 \\ 500 \\ 250 \end{pmatrix} = \begin{pmatrix} 2750 \\ 3500 \\ 125 \end{pmatrix}$$

$$X_3 = LX_2 = \begin{pmatrix} 0 & 4 & 3 \\ 0.5 & 0 & 0 \\ 0 & 0.25 & 0 \end{pmatrix} \begin{pmatrix} 2750 \\ 3500 \\ 125 \end{pmatrix} = \begin{pmatrix} 14375 \\ 1375 \\ 875 \end{pmatrix}$$

eftersom man har att

$$X_k = LX_{k-1},\ k = 1, 2, \ldots$$

Det finns alltså efter femton år 14375 honor mellan 0 och 5 år, 1375 honor mellan 5 och 10 år och 875 honor mellan 10 och 15 år.

Även om rekursionen ovan medför att

$$X_k = LX_{k-1} = L^k X_0$$

och beskriver populationens fördelning vid en godtycklig tidpunkt ger den inte utan vidare en allmän bild av tillväxtprocessens dynamik. Undersök istället egenvärdena och egenvektorerna hos övergångsmatrisen.

4.15 Här ska man använda födelse- och dödsparametrar från ett visst år (1965) och en viss population kvinnor. Eftersom få kvinnor över femtio år är gravida gör man en begränsningen till andelen av populationen, som är mellan 0 och 50 år. Data är insamlade för femåriga åldersklasser och det är därför 10 åldersklasser. Istället för att skriva ut övergångsmatrisen, som är 10×10, listar man födelse- och dödsparametrar på följande sätt:

Åldersintervall	a_i	b_i
$[0, 5)$	0.00000	0.99651
$[5, 10)$	0.00024	0.99820
$[10, 15)$	0.05861	0.99802
$[15, 20)$	0.28608	0.99729
$[20, 25)$	0.44791	0.99694
$[25, 30)$	0.36399	0.99621
$[30, 35)$	0.22259	0.99460
$[35, 40)$	0.10457	0.99184
$[40, 45)$	0.02826	0.98700
$[45, 50)$	0.00240	-

övergångsmatrisen blir på formen:

$$L = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_9 & a_{10} \\ b_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & b_2 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & b_9 & 0 \end{pmatrix}$$

Beräkna egenvärden och egenvektorer till matrisen. Vilket är det dominerande egenvärdet och motsvarande egenvektor?

Från egenvärdet ser man att om kvinnorna fortsatte att reproducera sig och dö som de gjorde 1965, skulle slutligen deras antal öka med 7.622% var femte år.

Från egenvektorn kan man se att för varje hundratusental kvinnor mellan 0 och 5 år kommer det att finnas 92594 kvinnor mellan 5 och 10 år, 85881 kvinnor mellan 10 och 15 år osv. Multiplicera egenvektorn med ett tal så att första elementet är 1. Är vektorn fortfarande en egenvektor? Kontrollera genom $Av = \lambda v$. Jämför talen i vektorn med att för varje hundratusental kvinnor mellan 0 och 5 år kommer det att finnas 92594 kvinnor mellan 5 och 10 år, 85881 kvinnor mellan 10 och 15 år osv.

5 Polynom

Polynom har många användningsområden och MATLAB har både kommandon för att hantera polynom och kommandon där polynom används, kurvanpassning är ett exempel.

5.1 Skapa och hantera polynom

Ett polynom, $p(x)$, dvs en funktion på formen

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

lagras i MATLAB som

$$p = \begin{pmatrix} a_n & a_{n-1} & \ldots & a_1 & a_0 \end{pmatrix}$$

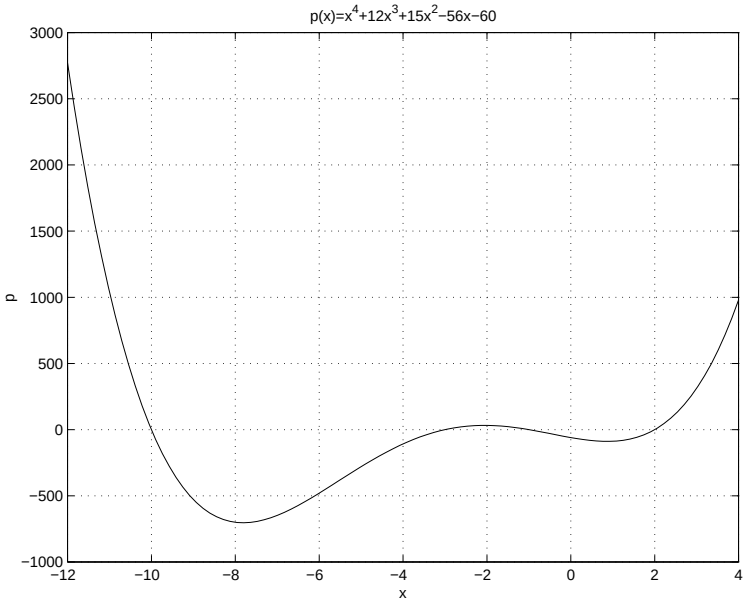
dvs som en radvektor (radmatris) p med längden $n+1$. Elementen i p representerar koefficienterna $a_n, a_{n-1}, \ldots, a_1, a_0$ i polynomet och anges i fallande ordning på potenserna.
För att beräkna värdet av $p(x)$ i en punkt x_0 använder man kommandot `polyval`. Antag att man har polynomet $p(x) = 3x^2 + 2x + 1$. För att med hjälp av MATLAB beräkna värdet i $x = 2$ skriver man

```
>> p = [3 2 1];
>> x = 2;
>> px = polyval(p,x)
```

Det går bra att ange en vektor i stället för bara en enstaka punkt. Då beräknas polynomets värde för varje element i vektorn. Resultet blir en vektor av samma storlek.

För att rita ett polynom använder man sig också av `polyval`. Antag att man har ett polynom $p(x) = x^4 + 12x^3 + 15x^2 - 56x - 60$. För att rita grafen till polynomet i intervallet $-12 < x < 4$ skriver man

```
x = linspace(-12,4);      % Skapa intervall
p = [1 12 15 -56 -60];    % Definiera polynom
y = polyval(p,x);         $ Beräkna polynomets värden i x-värdena
plot(x, y);
xlabel('x'); ylabel('p');
title('p(x)=x^4+12x^3+15x^2-56x-60');
grid
```



Figur 16: Polynomet $p(x) = x^4 + 12x^3 + 15x^2 - 56x - 60$ ritat med hjälp av kommandot `polyval`.

Resultatet visas i Figur 16.

Samma resultat får man om man använder metoden från Kapitel 2, dvs att direkt beräkna en vektor med polynomets värden:

```
x = linspace(-12,4);
p = x.^4+12*x.^3+15*x.^2-56*x-60

plot(x,p)
xlabel('x')
ylabel('p')
title('p(x) = x^4+12x^3+15x^2-56x-60')
grid
```

Det fiffiga med att representera polynom med en radvektor är att det finns en mängd olika kommandon som inte går att använda när man bara har en vektor med polynomets värden. De olika kommandona finns sammanfattade i rutan *Kommandon för polynom*.

Kommandon för polynom	
`polyval(p,x)`	evaluerar polynomet p. Om x är en skalär så returneras värdet av polynomet i punkten p. Om x är en matris så evalueras polynomet för varje element i x.
`roots(p)`	ger en kolonnmatris av längd n med nollställena till polynomet p av grad n, dvs lösningen till $p(x) = 0$.
`conv(p,q)`	multiplicerar polynomet p med polynomet q.
`polyder(p)`	ger en radmatris som representerar derivatan av polynomet p.
`polyder(p,q)`	ger en radmatris som representerar derivatan av polynomet bildat av `conv(p,q)`.
`polyint(p)`	ger en radmatris som representerar integralen av polynomet p.

5.2 Kurvanpassning med polynom

Att anpassa funktioner till mätdata är ett mycket vanligt inom naturvetenskap och teknik, men också samhällsvetenskap och ekonomi. Med hjälp av anpassade funktioner kan man dra slutsatser om samband och göra uppskattningar där mätdata saknas.

I de följande tre avsnitten används tre metoder för att anpassa polynom till mätdata. Oavsett varifrån mätdata kommer antas det bestå av ett antal talpar som ritats in som punkter i ett koordinatsystem. Alla tre metoder försöker i någon mening fånga sambandet genom att dra en kurva som ansluter till punkterna.

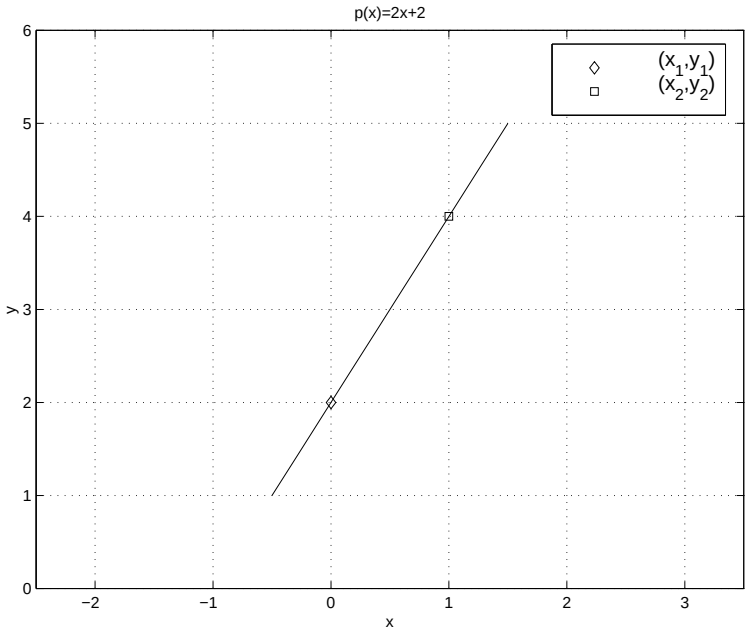
De tre metoderna är:

- *Interpolation med polynom.* Här beräknar man ett polynom som går genom alla punkterna.
- *Interpolation med styckvisa polynom.* Här beräknar man en uppsättning polynom av låg grad, som var för sig bara passerar genom två punkter var. Polynomen sätts sedan samman till en kurva.
- *Minstakvadratanpassning.* Här beräknas ett polynom av låg grad som normalt inte går genom någon enda mätpunkt, men som minimerar

det sammanlagda felet (i minstakvadratmening) mellan kurvan och mätpunkterna.

5.3 Interpolation med polynom

Antag att man har två punkter (x_1, y_1) och (x_2, y_2) där $x_1 \neq x_2$. Då finns det exakt en rät linje, dvs ett förstagradspolynom, som går genom dessa punkter, se Figur 17.



Figur 17: Interpolation med förstagradspolynom

Den räta linjen kan bestämmas direkt genom

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1),$$

dvs räta linjens ekvation i tvåpunktsform. Alternativt kan ett 1:a gradspolynom ansättas, dvs $y(x) = a + bx$ ansätts där a och b ska bestämmas. Eftersom polynomet ska gå exakt genom punkterna får man för de två talparen

$$\begin{cases} 1 \cdot a + x_1 \cdot b = y_1 \\ 1 \cdot a + x_2 \cdot b = y_2 \end{cases} \iff \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}.$$

Man letar ju efter funktionen $y(x)$ för vilken $y(x_1) = y_1$ och $y(x_2) = y_2$. Lösningen av det linjära ekvationssystemet ger a och b.

Om man istället har tre punkter, (x_1, y_1), (x_2, y_2) och (x_3, y_3), kan man normalt inte bestämma en rät linje som går genom samtliga tre punkter. Det finns däremot ett entydigt bestämt polynom av grad två som går genom de tre punkterna. Man letar då efter ett polynom på formen $y(x) = a + bx + cx^2$ som uppfyller $y(x_1) = y_1$, $y(x_2) = y_2$ och $y(x_3) = y_3$. Man kan hitta a, b och c genom att lösa ekvationssystemet

$$\begin{cases} 1 \cdot a + x_1 \cdot b + x_1^2 \cdot c = y_1 \\ 1 \cdot a + x_2 \cdot b + x_2^2 \cdot c = y_2 \\ 1 \cdot a + x_3 \cdot b + x_3^2 \cdot c = y_3 \end{cases} \Leftrightarrow \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Det gäller generellt att om man har $n + 1$ olika x-värden $x_0, x_1, \ldots, x_n$ och motsvarande y-värden $y_0, y_1, \ldots, y_n$, finns det exakt ett polynom av grad n så att interpolationsvillkoren

$$p(x_k) = y_k, \quad k = 0, 1, \ldots, n$$

är uppfyllda, dvs som går genom alla punkterna (x_k, y_k), $k = 0, 1, \ldots, n$.

I MATLAB kan man enklast anpassa polynom till punkter med kommandot `polyfit`. Om exempelvis följande mätvärden är givna

x	2	3	4
y	0.3010	0.4771	0.6021

skriver man t ex

```
>> xValues = [2 3 4];
>> yValues = [0.3010 0.4771 0.6021];
>> p = polyfit(xValues,yValues,2)
p =
   -0.0255    0.3038   -0.2045
```

Polynomet som går genom de tre punkterna blir alltså
$p(x) = -0.0255x^2 + 0.3038x - 0.2045$.

I komandot `p = polyfit(xValues,yValues,2)` står 2:an för polynomgrad. I detta fall vill man ju ha ett polynom av grad 2. Om man vill att polynomet ska gå genom alla punkter måste vi, enligt resonemanget ovan, ange gradtal = antalet punkter − 1. I avsnitt 5.5, visas exempel på vad som händer om man anger ett lägre gradtal.

Man kan kontrollera att polynomet verkligen går genom alla punkter genom att rita det. Man använder då `polyval` för att beräkna polynomets värden

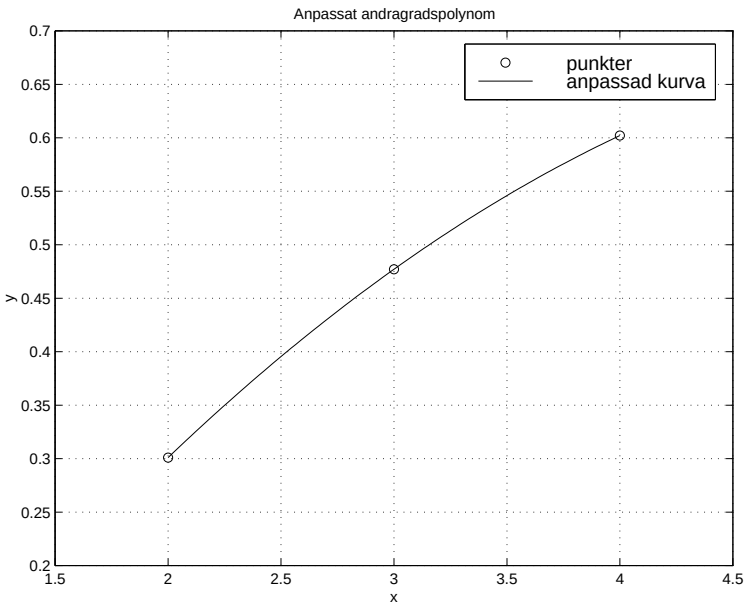
även mellan de givna punkerna. Man får grafen i Figur 18 med följande rader MATLAB-kod

```
xValues = [2 3 4];                % x-värden
yValues = [0.3010 0.4771 0.6021]; % y-värden

p = polyfit(xValues,yValues,2);   % anpassa polynom till punkterna

x = linspace(2,4);                % skapa x-axel
y = polyval(p,x);                 % beräkna polynomets värden för
                                  % alla värden i x

plot(xValues,yValues,'o')         % rita punkterna
hold on
plot(x,y)                         % rita polynomet
grid on
axis([1.5 4.5 0.2 0.7])
title('Anpassat andragradspolynom')
legend('punkter','anpassad kurva')
xlabel('x')
ylabel('y')
```

5.4 Interpolation med styckvisa polynom

En nackdel med att beräkna ett polynom som passerar alla punkter är att gradtalet blir högt när man har många mätpunkter. Det medför att man ofta får ”oroliga” polynom med stora svängningar upp och ned.

En lösning på problemet med svängiga polynom av hög grad är att istället sätta samman styckvisa polynom av låg grad, något som kallas för *splines*. Då låter man varje polynom passera två punkter var på följande sätt: det första polynomet låter man passera den första och andra punkten, det andra polynomet den andra och tredje punkten, det tre polynomet den tredje och fjärde punkten osv. Polynomen skapas så att de uppfyller vissa villkor, bl a att första och andraderivatorna ska överenstämma i mätpunkterna. På så sätt får man en mjuk sammansatt kurva där inga skarvar syns.



Figur 18: Interpolation med andragradspolynom

Man kan välja några olika sätt att skapa de styckvisa polynomen, t ex måste man bestämma sig för vilket gradtal de ska ha. En vanligt förekommande metod är att använda tredjegradspolynom, s k *kubiska splines*. Skälet att det blev just kubiska är att det praktiskt visat sig att dessa approximerar verkliga förlopp väldigt bra.

I MATLAB finns kommandot `spline` som använder ovan beskrivna metod. Antag att en kurva ska anpassas till föjande fem punkter:

x	0	1	2	3	5
y	1	2	1	2	3

Då kommer följande kommandosekvens att skapa en kurva m h a kommandot `spline`.

```
xValues = [0 1 2 3 5];          % x-värden
yValues = [1 2 1 2 3];          % y-värden

x = linspace(-1,6);             % Skapa x-axel
p = spline(xValues, yValues); % Beräkna kubiska splines
y = ppval(p,x);                 % Beräkna polynomets värde för alla
                                % x-värden
```

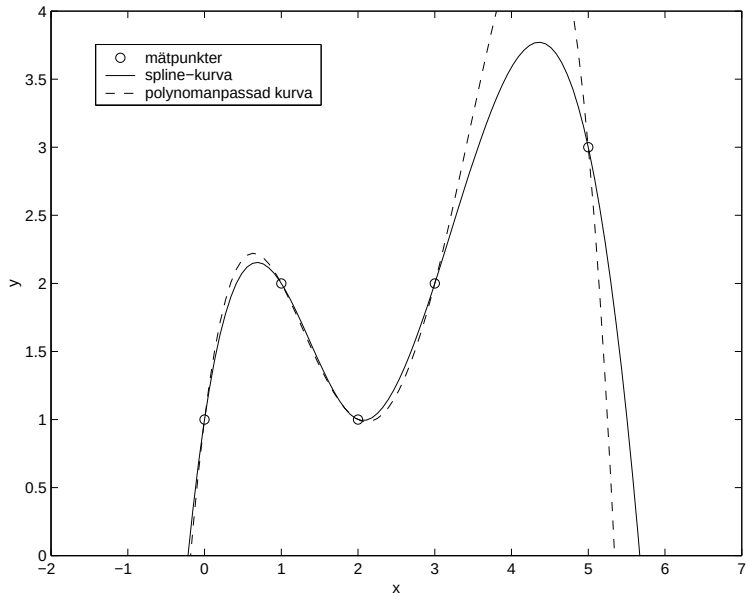
```
plot(xValues,yValues,'o')      % Plotta mätpunkter
hold on
plot(x,y)                      % Plotta polynomet
axis([-2 7 0 4])               % Ändra axlarna
legend('mätpunkter','spline-kurva','polynomanpassad kurva')
xlabel('x')
ylabel('y')
```

Som jämförelse ritas också ett vanligt interpolerande polynom in, se Figur 19. Observera att spline-kurvan inte svänger lika mycket som det andra polynomet.
Koden för de båda interpolationsmetoderna överensstämmer med några små skillnader. Istället för `polyfit` används `spline`, och istället för `polyval` används `ppval`. Det finns också möjlighet att utelämna `ppval` och utföra den beräkningen direkt i kommandot `spline`. Man ersätter då raden

```
p = spline(xValues,yValues);
```

med

```
y = spline(xValues,yValues,x)
```



Figur 19: En spline-kurva och ett polynom anpassade till fem punkter.

5.5 Minstakvadratmetoden

De givna punkterna är i allmänhet mätvärden och innehåller så gott som alltid mätfel. Det är därför inte alltid rimligt att tvinga kurvan att gå exakt genom alla punkter. Dessutom kan antalet punkter i praktiken vara mycket stort, t ex i storleksordningen 1000 punkter. Det är då varken rimligt eller möjligt att ansätta polynom som går genom alla punkter.

I stället för att "tvinga" ett polynom att gå genom varje punkt kan man leta efter ett polynom (eller en annan funktion) som går så nära punkterna som möjligt. Ett sätt är att minimera felkvadratsumman. Det innebär att om man har punkterna (x_1, y_1), (x_2, y_2), ..., (x_m, y_m) och vill hitta en rät linje, $y = c_0 + c_1 x$, som går så nära punkterna som möjligt, minimerar man

$$F(c_0, c_1) = \sum_{i=1}^{m} (y_i - c_0 - c_1 x_i)^2.$$

Funktionen $F(c_0, c_1)$ som minimeras är summan av kvadraterna på de lodräta avstånden mellan punkterna och räta linjen, se exemplet i Figur 20.

Det visar sig att minimum ges av de så kallade normalekvationerna $A^T A c = A^T b$, där

$$A = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix}, \quad b = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \text{ och } \quad c = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix}$$

I MATLAB kan man beräkna ett minstakvadratanpassat polynom till en punktmängd på (minst) tre sätt:

- genom att använda `polyfit`-kommandot och ange ett gradtal som är lägre än antalet punkter-1 (annars blir det interpolation). Detta är det enklaste sättet eftersom man slipper skapa matrisen A.

- genom att bilda matrisen A sedan skriva $x = A\backslash b$. Vänsterdivision "känner av" att matrisen A är överbestämd, dvs att den har fler rader än kolonner och utför då minstakvadratapproximation.

- i princip kan man också lösa normalekvationerna $A^T A c = A^T b$, med vänsterdivision, dvs `c = (A'*A)\(A'*b)`. Detta är korrekt, men lite jobbigare metod.

Om man istället vill ha ett polynom av högre grad, exempelvis grad 3, dvs

$$F(c_0, c_1, c_2, c_3) = \sum_{i=1}^{m} (y_i - c_0 - c_1 x_i - c_2 x_i^2 - c_3 x_i^3)^2,$$

får man istället

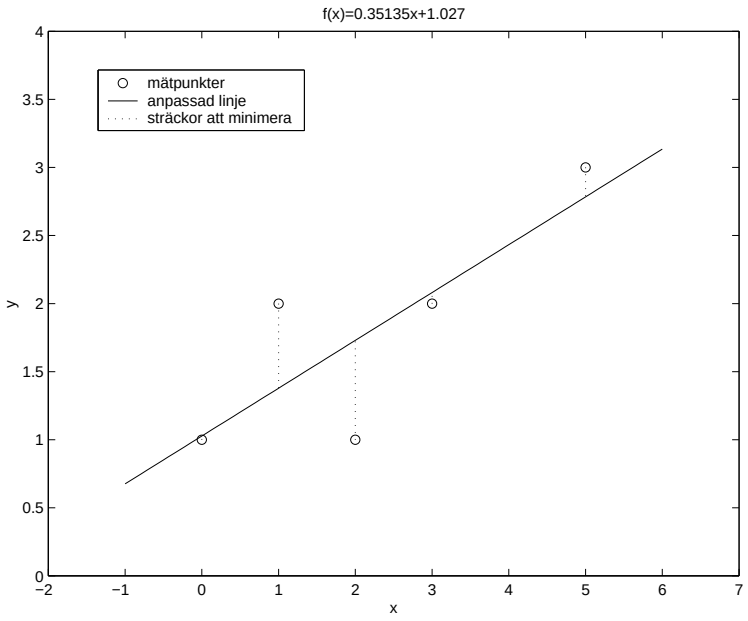
$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & & \\ 1 & x_m & x_m^2 & x_m^3 \end{pmatrix} \quad \text{och} \quad c = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.$$

Vektorn b är som ovan. Metoden går att generalisera till hur höga gradtal som helst (begränsade av antalet punkter förstås).

Antag att en linje ska minstakvadratanpassas till de fem punkter som gavs i exemplet i föregående avsnitt. I MATLAB kan man lösa problemet så här:

```
x = [0 1 2 3 5];          % x-värden
y = [1 2 1 2 3];          % y-värden

xTal = linspace(-1,6);    % skapa x-axel
f = polyfit(x,y,1);       % anpassa polynom av grad 1
yTal = polyval(f,xTal);   % beräkna polynomets värden för hela
                          % x-axeln
p1 = plot(x,y,'o');       % plotta x- och y-värden
hold on
p2 = plot(xTal,yTal);     % plotta polynomet

axis([-2 7 0 4])          % ändra axlar

legend([p1 p2],'mätpunkter','anpassad linje')
xlabel('x')
ylabel('y')
title(['f(x)=' num2str(f(:,1)) 'x+' num2str(f(:,2))])
```

Grafen som generaras av kommandona ovan visas i Figur 20. I grafen har för tydlighets skull även lagts till avståndet mellan punkter och linje, dvs de avstånd som ska minimeras.



Figur 20: En rät linje anpassad till fem punkter i minstakvadrat-mening.

Kurvanpassning	
`polyfit(x,y,n)`	ger en radvektor med koefficienter till det polynom av grad n som antingen är minstakvadratanpassning till talmängden $\{(x_i, y_i)\}$ eller interpolation.
`yy = spline(x,y,xx)`	returnerar en vektor yy som är en splinekurva till punkterna vars koordinater ges i talföljderna x och y. Vektorn yy beräknas för de x-värden som ges av vektorn xx.

5.6 Exempel: kaniner och rävar

Som tillämpning på kurvanpassning visas här ett större exempel där man ska undersöka sambandet mellan antalet kaniner och antalet rävar inom ett område.

Antag att man har gjort ett experiment där man under fyra år har räknat antalet rävar och kaniner inom ett begränsat område. Man har räknat dem vid fyra tillfällen per år och därvid erhållit två dataserier med 16 värden i

varje. För att studera tidsutvecklingen för antalet djur gör man två diagram.

Lägg märke till att ett långt kommando kan skrivas över två rader genom att avsluta den första raden med tre punkter.

```
manad = 1:3:(4*12);                % Ger manad = [1 4 7 10 ...]
                                   % dvs radmatrisen
                                   % innehåller första
                                   % månaden i varje kvartal.
kaniner = [39 55 224 302 283 430 435 354 333 272 196 192 ...
          101 257 222 404];
ravar =   [71 59 49 37 30 19 22 24 26 39 44 59 56 57 44 15];

figure(1)
clf
subplot(221)
plot(manad, kaniner)
title('Kaninvariation')
xlabel('Månad')
ylabel('Antal')

subplot(222)
plot(manad, ravar, ':')
title('Rävvariation')
xlabel('Månad')
ylabel('Antal')
```

Det kan vara intressant att se de två kurvorna i samma diagram.

```
subplot(223)
plot(manad, kaniner)
hold on
plot(manad, ravar, ':')
title('Kaniner och rävar')
xlabel('Månad')
ylabel('Antal')
legend('Kaniner', 'Rävar')
```

För att tydligare se hur antalet rävar och kaniner samvarierar kan man rita antalet kaniner mot antalet rävar istället för att rita dem mot tiden.

```
subplot(224)
plot(ravar, kaniner, 'o')
title('Samvariation mellan kaniner och rävar')
xlabel('Antal rävar')
ylabel('Antal kaniner')
```

Man ser i plottarna att det verkar råda ett linjärt samband mellan antalet rävar och antalet kaniner. Med hjälp av minstakvadrat anpassning kan man hitta den linje som i minstakvadratmening bäst passar datamängden. Linjen kallas ofta för regressionslinje. I MATLAB beräknas regressionslinjen med hjälp av kommandot `polyfit`.

```
regrLinje = polyfit(ravar, kaniner, 1);
hold on
x = linspace(min(ravar), max(ravar)); % med ''min'' och ''max''
                                      % hittar man enkelt det
                                      % minsta respektive
                                      % det största värdet
                                      % i en vektor
plot(x, polyval(regrLinje, x),'r')
title('Regression mellan rävar och kaniner')
```

De resulterande graferna finns i Figur 21.

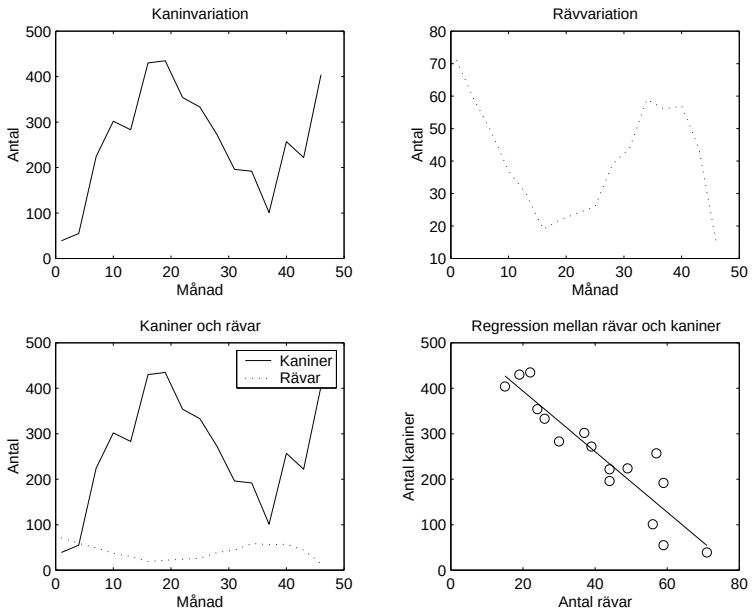
Med hjälp av regressionslinjen kan man göra "intelligenta" gissningar om t ex hur många kaniner det kommer att finnas om antalet rävar är 50 eller kanske 15. Det är bara att räkna ut värdet för den räta linjen för det antal rävar man är intresserad av.

```
ravAntal = 15
kaninAntal = polyval(regrLinje, ravAntal)
```

Som svar fås

```
ravAntal =
    15
kaninAntal =
  427.2426
```

vilket kanske bör avrundas till heltal.



Figur 21: Exempel på användning av kurvanpassning: redovisning av experiment med förhållandet mellan antal rävar och antal kaniner.

5.7 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

För att kunna lösa några av övningarna nedan kan det vara bra att repetera ett par satser för funktioner (inklusive polynom):

1. Om $y = f(x)$ är deriverbar i ett intervall I och

   a) $f'(x) > 0$ för alla $x \in I$ så är $y = f(x)$ strängt växande i intervallet I.

   b) $f'(x) < 0$ för alla $x \in I$ så är $y = f(x)$ strängt avtagande i intervallet I.

   c) f har ett max- eller minvärde i x_0 (som ligger i det inre av I) så gäller att $f'(x_0) = 0$. En sådan punkt kallas en *stationär punkt*.

2. Om funktionen f har en stationär punkt i x_0 och f'' existerar i x_0, så gäller att

   a) $f''(x_0) > 0 \;\Rightarrow\; f$ har en minpunkt i x_0.

   b) $f''(x_0) < 0 \;\Rightarrow\; f$ har en maxpunkt i x_0.

5.1 Beräkna alla extrempunkter, dvs lokala minima och lokala maxima, till polynomet

$$r(x) = -x^4 + 0.25x^3 + 1.375x^2 - 0.25x - 0.375$$

Beräkna också polynomets värden i dessa punkter. Rita polynomet och undersök om dina beräkningar stämmer överens med grafen.

5.2 Bestäm var polynomet

$$p(x) = x^4 + 12x^3 + 15x^2 - 56x - 60$$

är strängt växande och var det är strängt avtagande. Bestäm också de stationära punkterna till polynomet och avgör om de är max-, min- eller terasspunkter. Kontrollera resultatet genom att rita en graf.

5.3 Metan kan reagera med vattenånga och ger då kolmonoxid och vätgas.

$$\mathrm{CH_4} + \mathrm{H_2O} \rightleftharpoons \mathrm{CO} + 3\,\mathrm{H_2}$$

Om $\mathrm{CH_4}$ (g) och $\mathrm{H_2O}$ (g), vardera av trycket 1.00 atm tillförs ett reaktionskärl vid 300 K kommer trycket vid jämvikt att uppfylla:

$$\frac{p_{\mathrm{CO}} \cdot p_{\mathrm{H_2}}^3}{p_{\mathrm{CH_4}} \cdot p_{\mathrm{H_2O}}} = 26$$

Ur stökiometrin för reaktionen och initialtrycken kan man få följande samband mellan de olika jämviktstrycken: om $p_{\mathrm{CO}} = x$ blir övriga tryck $p_{\mathrm{CH_4}} = p_{\mathrm{H_2O}} = 1 - x$ och $p_{\mathrm{H_2}} = 3x$. Beräkna trycket för de olika gaserna i reaktionen vid jämvikt.

5.4 Vid reaktioner eller tillståndsändringar som innehåller gaser används ofta idealgaslagen ($PV = nRT$) för att beskriva sambandet mellan tryck, volym, temperatur och antalet mol (n). Konstanten R kallas allmänna gaskonstanten. För mer noggranna beräkningar nära kokpunkten eller vid höga tryck används andra s k tillståndsekvationer. Exempel på ett sådant samband är van der Waals ekvation

$$\left(P + \frac{a}{V_m^2}\right)(V_m - b) = RT$$

där V_m är volymen per mol gas. Konstanterna a och b har olika värden för olika gaser. Till skillnad från idealgaslagen kan denna ekvation beskriva både gas och vätskeformen av ett ämne. Använd van der

Waals tillståndsekvation för att bestämma molvolymen, dvs V_m, för CO vid 200 K och 1000 bar. För CO är $a = 1.4734\,\text{bar}\,\text{dm}^6\,\text{mol}^{-2}$ och $b = 0.039523\ \text{dm}^3\,\text{mol}^{-1}$. Gaskonstanten har värdet $R = 0.083145\,\text{bar}\,\text{dm}^3\,\text{K}^{-1}\,\text{mol}^{-1}$.

Det experimentella värdet är $0.04009\,\text{dm}^3\,\text{mol}^{-1}$. Hur stor blev skillnaden?

5.5 Gör samma beräkning som i uppgift 4 för Redlich-Kwongs tillståndsekvation

$$P = \frac{RT}{V_m - B} - \frac{A}{\sqrt{T}V_m(V_m + B)}$$

med $A = 17.208\,\text{bar}\,\text{dm}^6\,\text{K}^{1/2}\,\text{mol}^{-2}$ och $B = 0.027394\,\text{dm}^3\,\text{mol}^{-1}$.

5.6 Vid 400 K kan det experimentellt uppmätta trycket för propan vid olika koncentrationer, ρ, passas till uttrycket

$$P(\rho) = 33.258\rho - 7.5884\rho^2 + 1.0306\rho^3 - 0.058757\rho^4 - 0.0033566\rho^5 + 0.00060696\rho^6$$

där trycket är givet i bar och koncentrationen i $\text{mol}\,\text{dm}^{-3}$. Använd van der Waals ekvation (i uppgift 4) och Redlich-Kwongs ekvation (i uppgift 5) för att beräkna P som funktion av $\rho = 1/V_m$ upp till $\rho = 12.3\,\text{mol}\,\text{dm}^{-3}$. Rita resultatet och jämför med det experimentellt bestämda trycket. Uppgiften består alltså av att plotta P för tre olika fall:

(i) Det polynom som approximerar experimentellt bestämda tryck-konc. data.

(ii) Trycket beräknat med van der Waals ekvation.

(iii) Trycket beräknat med Redlich-Kwongs ekvation.

För propan är $A = 183.02\,\text{bar}\,\text{dm}^6\,\text{K}^{1/2}\,\text{mol}^{-2}$ och $B = 0.062723\,\text{dm}^3\,\text{mol}^{-1}$. För konstanterna i van der Waals ekvation gäller $a = 9.3916\,\text{bar}\,\text{dm}^6\,\text{mol}^{-2}$ och $b = 0.090494\,\text{dm}^3\,\text{mol}^{-1}$.

5.7 a) En person kastar en kula vertikalt upp i luften och under kulans luftfärd mäts kulans höjd över den punkt där den släpptes. Följande värden erhålls

tid	[s]	0	1	2	3	4	5	6	7	8	9
sträcka	[m]	1	10	16	21	24	24	23	21	15	7

Anpassa en kurva till punkterna som går genom varje punkt. Rita kurvan och punkterna i samma plot. Kurvan ritas som en linje och punkterna som punkter.

b) Man vet från fysiken att en kastkurva följer en andragradskurva, om man bortser från luftmotstånd. Anpassa därför istället ett polynom av grad 2 till mätpunkterna.

5.8 Priserna i 1000-tals kronor per kvadratmeter för bostadsrätter i Uppsala kan troligen uppskattas med formeln

$$\text{pris}(s) = as^2 + bs + c, \quad \text{där } s = \frac{1}{r}$$

och r är avståndet mellan bostaden och centrum räknat i 100-tals meter och a, b, c är konstanter. Givetvis påverkar även andra faktorer, men för ett urval av jämförbara bostadsrätter kan ansatsen ändå duga. Data från en sådan undersökning finns i tabellen nedan.

avstånd [km]	5	10	15	20	25	30	35	40
pris [kr/kvm]	5.40	4.30	3.85	3.25	2.60	1.90	1.60	1.50

Anpassa ett polynom av grad 2 till uppgifterna i tabellen.

5.9 Experimentera fram ett polynom som är en bra anpassning till nedanstående data. Använd minstakvadratanassning och börja med gradtal 1. Rita punkter och alla polynomen. Vad händer om du väljer ett för högt gradtal?

x	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5	6.0	6.5
y	1.2	2.3	3.2	4.3	5.0	5.7	6.0	5.9	5.0	4.8	4.0	3.7

x	7.0	7.5	8.0	8.5	9.0	9.5	10.0	10.5	11.0	11.5	12.0
y	3.8	3.9	3.8	3.4	2.3	0.3	-0.9	-1.6	-1.8	-2.3	-2.1

x	12.5	13.0	13.5	14.0
y	-1.7	-1.1	-0.1	1.2

5.10 Utgå från en mätserie som uppmättes i ett laboratorium

x	0.5	1.5	2.1	3.4	4.0	5.3	6.1	7.5	8.8	9.7	10.5
y	1.3	2.3	3.4	4.2	4.8	5.1	4.7	4.0	3.3	2.4	1.4

och bestäm det polynom som går genom samtliga punkter. Rita också polynomet.

Grafen blir väldigt "orolig". Polynomet kan knappast utgöra någon rimlig modell över skeendet. En mer trolig modell är $p(x) = ax^2 + bx + c$. Utgå från mätserien och bestäm a, b och c genom att göra en minstakvadrat-anpassning.

5.11 Vid ultracentrifugering av proteiner eller andra makromolekyler kan man bestämma den s k sedimentationskonstanten, S, vilken definieras som

$$S = \frac{dr/dt}{r\omega^2}$$

där r är avståndet till centrifugens rotoraxel, t är tiden och ω är vinkelhastigheten ($2\pi\cdot$antalet varv per sekund). Om man integrerar detta uttryck får man

$$\ln r = S\omega^2 \cdot t + A$$

Bestäm sedimentationskonstanten för en viss polymer ur följande data

$t/(\text{min})$	0	16.3	32.3	48.3	64.3	80.3
$r/(\text{cm})$	6.190	6.348	6.497	6.649	6.806	6.962

om rotationshastigheten är 52000 varv per minut.

5.12 För att bestämma ångbildningsvärmet för en vätska (dvs den värmeenergi som åtgår vid förångning) kan man mäta vätskans ångtryck vid olika temperaturer och passa data till följande funktion (Clausius-Clapeyrons ekvation):

$$\ln P = -\frac{\Delta H_m}{R} \cdot \frac{1}{T} + A$$

där ΔH_m är ångbildningsvärmet per mol, T är temperaturen i Kelvin (K), R är gaskonstanten ($R = 8.3145\,\text{J}\,\text{K}^{-1}\,\text{mol}^{-1}$) och A är en konstant. Bestäm ångbildningsvärmet för kvicksilver m h a följande data

$P \cdot 10^4/(\text{mmHg})$	28.01	61.18	127.2	252.6
$t/^\circ\text{C}$	30.0	40.0	50.0	60.0

5.13 Den elektriska ledningsförmågan *per mol* (den molära ledningsförmågan, Λ) för lösningar av starka (fullständigt dissocierade) elektrolyter minskar med ökande koncentration eftersom de laddade jonerna påverkar varandra mer och mer när medelavståndet mellan dem minskar. För att bestämma jonernas eget bidrag till ledningsförmågan kan man extrapolera till oändlig utspädning, dvs koncentrationen noll, och får då

molkonduktiviteten vid oändlig utspädning, Λ_0. Ofta brukar man kunna anpassa data till följande funktion

$$\Lambda = \Lambda_0 - k\sqrt{c}$$

där c är koncentrationen och k är en konstant. Använd följande data för kaliumjodat (KIO_3) för att bestämma molkonduktiviteten vid oändlig utspädning.

$c/(\mathrm{mol\,m^{-3}})$	0.130	0.317	0.576	0.998
$\Lambda \cdot 10^4/(\mathrm{m^2\,\Omega^{-1}\,mol^{-1}})$	113.25	112.68	112.13	111.46

5.14 Det som bestämmer om en process är spontan vid ett visst tryck och en viss temperatur är *både* energiomsättningen och molekylernas "rörelsefrihet" (entropin). Man kan t ex torka tvätt vid rumstemperatur (vilket är under kokpunkten för vatten) även om det "kostar energi" att förånga vattenmolekylerna. Drivkraften för detta är molekylernas ökade rörelsefrihet i gasfasen och detta beskrivs med entropiökningen, ΔS. För att kunna räkna ut entropiändringar för ett ämne kan man använda sambandet

$$\Delta S = \int_{T_0}^{T} \frac{C_p}{T} dT$$

Ett sätt att utföra denna integrering är att ersätta C_p med ett polynom genom att ansätta ett polynom till data.
Beräkna på detta sätt entropiändringen då man ökar temperaturen för N_2O från 15.17 K till 182.26 K, vilket är smältpunkten, genom att utnyttja följande data där $C_{p,m}$ är värmekapaciteten per mol.

T (K)	$C_{p,m}(\mathrm{J\,K^{-1}\,mol^{-1}})$	T (K)	$C_{p,m}(\mathrm{J\,K^{-1}\,mol^{-1}})$
15.17	2.90	87.06	38.87
19.95	6.19	98.34	41.13
25.81	10.89	109.12	42.84
33.81	16.98	120.29	45.10
42.61	23.13	130.44	47.32
52.02	28.56	141.07	48.91
57.35	30.75	154.71	52.17
68.05	34.18	164.82	54.02
76.67	36.57	174.90	56.99
		180.75	58.83

6 Integraler och differentialekvationer

Att beräkna bestämda integraler och hitta lösningar till differentialekvationer är två typiska problem där numeriska metoder är nödvändiga. För många problem kan man nämligen inte använda analytiska metoder, dvs metoder där man försöker bestämma en explicit formel, på grund av att en analytisk lösning inte existerar eller är för svår att hitta.

I MATLAB finns inbyggda kommandon som använder sig av numeriska metoder för integralberäkning och lösande av differentialekvationer.

6.1 Integralberäkning

Vid beräkning av en bestämd integral beräknas värdet för ett givet intervall. Antag att funktionen $f(x)$ ska integreras i intervallet $x \in [a, b]$. Då ges integralen q av:

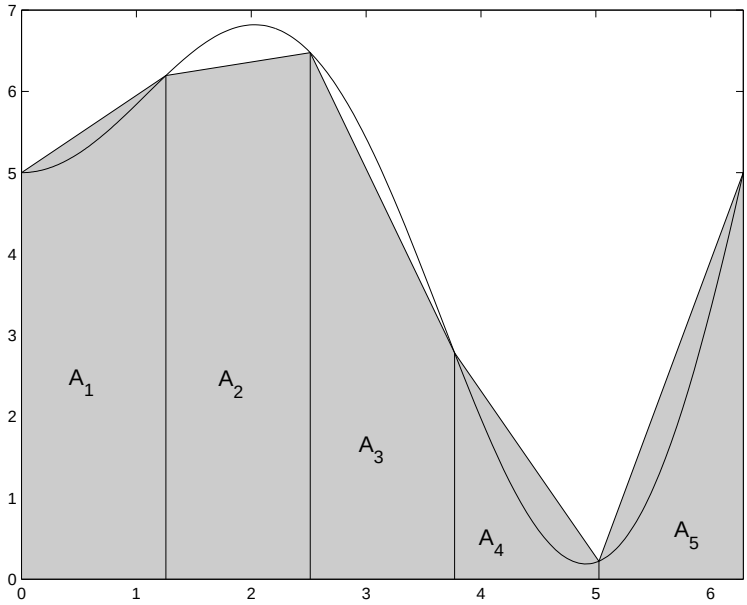
$$q = \int_a^b f(x)dx$$

Om integranden, dvs $f(x)$, är positiv på hela intervallet motsvarar q arean mellan x-axeln och grafen till funktionen $f(x)$ i det givna intervallet. Om integranden är negativ för något delintervall subtraheras den area som ligger nedanför x-axeln (arean mellan axeln och grafen).

För att numeriskt bestämma q kan man approximera arean under kurvan med en summa av parallelltrapetser. I Figur 22 visas som exempel hur integralen av funktionen $5 + x \sin x$ i intervallet $[0, 2\pi]$ uppskattas med hjälp av de fem parallelltrapetserna A_1, A_2, A_3, A_4, A_5. Som man kan se i figuren överenstämmer inte arean av parallelltrapetserna helt med arean under kurvan och resultatet blir inte exakt. Men genom att öka antalet trapetser kan felet göras tillräckligt litet. Denna metod kallas *Trapetsmetoden*.

I MATLAB finns kommandot `trapz` för att uppskatta en bestämd integral med Trapetsmetoden. För att göra uppskattningen ovan med fem trapetser skriver man:

```
>> x = linspace(0,2*pi,6);
>> y = 5 + x.*sin(x);
>> trapz(x,y)
ans =
   25.9822
```



Figur 22: Numerisk integration av $5 + x \sin x$ i intervallet $[0, 2\pi]$.

Att det blir just 5 trapetser beror på att man i `linspace` valt 6 punkter. Funktionen $5 + x \sin x$ kan integreras analytiskt (en primitiv funktion är $5x - x \cos x - \sin x$) och i det givna intervallet har den bestämda integralen värdet 25.1327 avrundat till fyra decimaler. Trots att bara fem trapetser användes fick man alltså en någorlunda bra approximation. Om antalet trapetser ökas till 100 blir uppskattningen 25.1349.

Det finns i MATLAB två ytterligare kommandon för integrering, nämligen `quad` och `quadl` (i äldre versioner av MATLAB finns `quad8` istället för `quadl`). Dessa baseras på mer effektiva metoder än Trapetsmetoden ovan och man bör använda någon av dessa när man löser integraler. Metoden bakom `quadl` har högre *noggrannhetsordning* än `quad`. Begreppet *noggrannhetsordning* förklaras inte här, eftersom det kräver ytterligare kunskaper i beräkningsvetenskap, men man kan säga att `quadl` bygger på en mer avancerad algoritm. Båda kommandona används på samma sätt och kräver att man sparar integranden som en funktion i en m-fil, som skickas in som inparameter till `quad` respektive `quadl`. Hur man skriver funktioner visas i avsnitt 7.8, men vi visar ändå här med ett exempel hur det ser ut.

Om vi vill integrera samma integrand som tidigare, dvs $5+x \sin x$ med `quadl` eller `quad` måste alltså integranden definieras i form av en funktion som lagras

i en m-fil. Vi kan exempelvis ge denna m-fil namnet `integrand.m` och följande innehåll

```
function y = integrand(x)
y = 5 + x.*sin(x);
```

Att m-filen inleds med `function y = integrand(x)` medför att den av MATLAB betraktas som en funktion, att `x` är en s k inparameter och `y` en utparameter. Namnet på funktionen är `integrand`.
Vi visar här hur man använder `quadl`. För att noggrannheten hos metoden ska framgå ökas antalet utskrivna decimaler med kommandot `format long` (som enbart påverkar hur många decimaler som skrivs ut, inte noggrannheten i beräkningarna).

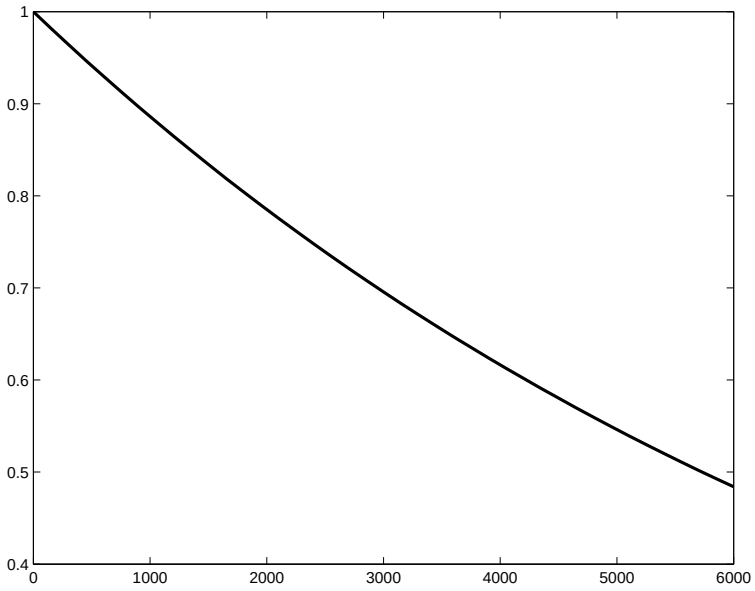
```
>> quadl(@integrand,0,2*pi)
ans =
  25.13274122908970
```

Detta kan jämföras med värdet från den analytiska lösningen som avrundat till lika många decimaler är 25.13274122871834. Observera att integranden "skickas in" i `quadl` genom att man skriver `@integrand` som första argument (filen där integranden definierades heter `integrand.m` i det här fallet). Man "talar om" för `quadl` att den ska arbeta på den funktion som finns definierad i filen `integrand.m`.

Integralberäkning	
`trapz(x,y)`	beräknar integralen av y som funktion av x. Variablerna x och y ska vara talföljder av samma längd där (x_i, y_i) representerar punkter på en kurva.
`quadl(@fkn,a,b)`	beräknar integralen mellan a och b av funktionen som finns definierad i fkn.m
`quad(@fkn,a,b)`	som `quadl` men använder en mindre avancerad metod.

6.2 Differentialekvationer

En *differentialekvation* är ett samband mellan en okänd funktion och dess derivator. De beskriver någon typ av förändring t ex i tiden. Ett enkelt exempel



Figur 23: Lösning av $y(t)' = -ky$ för kol-14 med begynnelsevärde $y(t_0) = 1$.

är radioaktivt sönderfall. Om $y(t)$ är andelen av ett radioaktivt ämne vid tidpunkten t, så avtar avtar denna andel proportionellt mot andelen radioaktivt ämne som finns kvar. Detta kan beskrivas med en differentialekvation

$$y'(t) = -ky$$

där k är en konstant som beror av vilket radioaktivt ämne det handlar om. För att man ska kunna hitta en specifik lösning till denna måste man ha ett *begynnelsevärde*, dvs man måste veta andelen radioaktivt ämne vid en viss starttidpunkt, $y(t_0) = \hat{y}$. Denna typ av differentialekvationer kallas därför för *begynnelsevärdesproblem*. I detta fall blir den

$$\begin{cases} y'(t) = -ky \\ y(t_0) = \hat{y} \end{cases}$$

För kol-14 gäller att $k \approx 0.000120968$. I Figur 23 visas lösningen till differentialekvationen då andelen radioaktivt material är 100% vid år noll, dvs begynnelsevärdet $y(t_0) = 1$ På den horisontella axeln visas år och man kan t ex se att andelen radioaktivt material material har halverats någonstans kring 5700 år (5730 för att vara exakt). Detta är halveringstiden för kol-14.

Hur man löser denna differentialekvation i MATLAB visas i avsnitt 6.4, sid 75.

Allmänt kan man skriva begynnelsevärdesproblem på följande sätt

$$\begin{cases} y'(t) = f(t, y) \\ y(t_0) = \hat{y} \end{cases}$$

där den sökta funktionen är $y(t)$. Om funktionen är av en variabel, som i exemplet, kallas differentialekvationen *ordinär*, i annat fall är den *partiell*. En ordinär differentialekvation är av *ordning* n om den har derivator av den sökta funktionen av maximalt ordning n.

6.3 System av ordinära differentialekvationer

Ofta har man inte en enda differentialekvation utan ett antal ekvationer som beror av varandra, ett *system* av differentialekvationer. Ett exempel på system av differentialekvationer är

$$\begin{cases} y_1'(t) = \dfrac{dy_1}{dt} = c_1 y_1 - c_2 y_1 y_2 \\ \\ y_2'(t) = \dfrac{dy_2}{dt} = -c_3 y_2 + c_4 y_1 y_2 \end{cases}$$

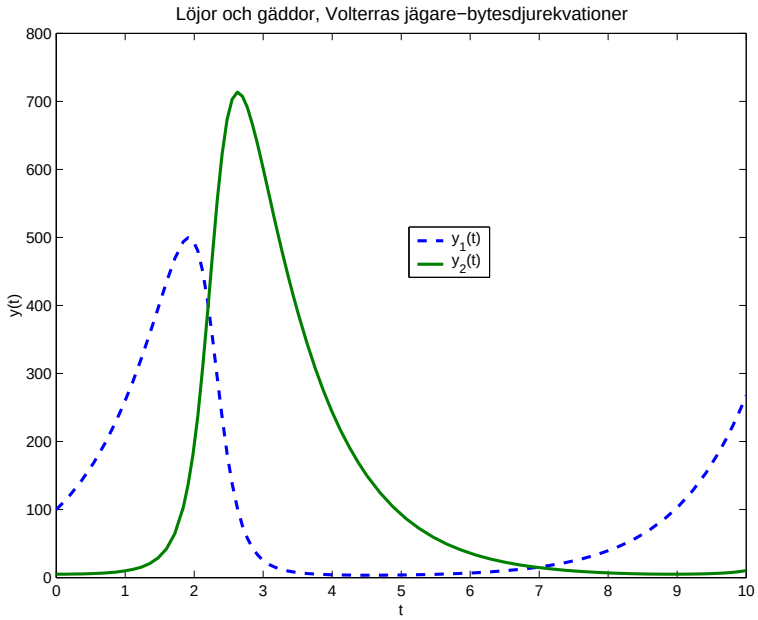
Detta system kallas Volterras jägare-bytesdjursekvationer, där $y_1(t)$ är antalet djur av arten som är byte, $y_2(t)$ är antalet djur av arten som är jägare, och $c_1, \ldots c_4$ är konstanter. Systemet beskriver alltså hur populationsantalet varierar med tiden (t=dagar). Den första termen i ekvationerna kommer från Malthus lag om en populations tillväxt under förutsättning att det finns obegränsat med föda. Den andra termen beskriver vad som händer vid möte mellan jägare och byte.

Vi kan anta att bytena i detta fall är löjor och jägarna gäddor och konstanterna i systemet är

$$c_1 = 1, \quad c_2 = 0.007, \quad c_3 = 1 \text{ och } \quad c_4 = 0.01.$$

Även här krävs begynnelsevärden för att man ska kunna hitta en specifik lösning. Om vi antar att det från början 100 löjor och 5 gäddor fås begynnelsevärdena $y_1(0) = 100$ och $y_2(0) = 5$. Begynnelsevärdesproblemet blir då

$$\begin{cases} y_1'(t) = y_1 - 0.007 y_1 y_2 \\ y_2'(t) = -y_2 + 0.01 y_1 y_2 \\ x_1(0) = 100 \\ x_2(0) = 5 \end{cases}$$



Figur 24: Antalet löjor och gäddor under dag 0 till 10.

Lösningen till detta problem visas i Figur 24. Hur man löser det i MATLAB presenteras i avsnitt 6.4, sid 76.
Allmänt kan ett system av två differentialekvationer skrivas

$$\begin{cases} y_1'(t) = f_1(t, y_1, y_2) \\ y_2'(t) = f_2(t, y_1, y_2) \\ y_1(t_0) = \hat{y}_1, \quad y_2(t_0) = \hat{y}_2 \end{cases}$$

Ett system av differentialekvationer behöver givetvis inte bestå av just två ekvationer utan kan i princip innehålla hur många som helst.

6.4 Lösning av ordinära differentialekvationer med MATLAB

Det finns i MATLAB en hel rad med färdiga kommandon för att numeriskt lösa differentialekvationer, t ex `ode23`, `ode45`, `ode15s`, etc. Av `ode` i första delen av namnet ser man att kommandot löser ordinära differentialekvationer. De olika kommandona baseras på lite olika metoder, och är lämpliga för olika typer av problem. För s k *styva problem*, ett begrepp som inte förklaras i detta kompendium, är t ex `ode15s` lämplig.

Den metod man bör välja som förstahandsval är `ode45`, och vi kommer därför att använda den här. Användningen av de övriga kommandona fungerar på exakt samma sätt.

För att kunna använda `ode45` måste man först skapa en MATLAB-funktion där högerledet $f(t, y)$ (för system högerleden $f_1(t, y_1, y_2)$ och $f_2(t, y_1, y_2)$), i differentialekvationen definieras. Mer om hur man skriver funktioner finns i avsnitt 7.8, men vi ska redan här visa hur det ser ut.

Därefter anger man tre parametrar till kommandot `ode45`: namnet på MATLAB-funktionen som definierar högerledet, intervallet $[t_0, t_{max}]$ där lösningen ska beräknas, samt begynnelsevärdet $y(t_0)$.

Lösning av en differentialekvation

Som exempel tar vi den differentialekvation som beskriver radioaktivt sönderfall (se sid 72 i avsnitt 6.2), dvs

$$\begin{cases} y'(t) = -ky, \;\; 0 \leq t \leq 6000 \\ y(0) = 1 \end{cases}$$

med $k = 0.000120968$.

Första steget är att skapa en funktion som t ex kan namnges till `diffEkv.m`:

```
function yprim = diffEkv(t,y)
k = 0.000120968;  % Konstanten k
yprim = -k*y;
```

Denna funktion definierar alltså högerledet i vår differentialekvation.

Nu sätter vi samman en kommandosekvens där anropet av `ode45` görs. Här krävs begynnelsevärdet $y_0 = 1$ och intervallet som beräkningen ska utföras på som i detta fall är $t_0 = 0$ och slutpunkt $t_{max} = 6000$.

```
y0 = 1; t0 = 0;
tmax = 6000;
[t, y] = ode45(@diffEkv,[t0 tmax],y0);
plot(t,y,'-')
title('Lösning av y''=-ky med y(0)=1') % För att få y' i titel,
                                       % skriv y''
xlabel('t')
ylabel('y(t)')
```

Utparametrar från `ode45` är en kolonnvektor t med t-värden och motsvarande $y(t)$-värden i kolonnvektorn y. Dessa kolonnvektorer används för att rita grafen. Resultatet visas i Figur 23 (fast där utan `title`, `xlabel`, `ylabel`).

Lösning av system av differentialekvationer

System av differentialekvationer löses på principiellt samma sätt i MATLAB. Skillnaden är att man måste definiera alla högerled i differentialekvationen och begynnelsevillkor ska ges för varje ekvation. Utparametrarna är fortfarande `t` och `y`, men `y` är nu en matris med lika många kolonner som det finns ekvationer i systemet.

Som exempel tar vi Volterras jägare-bytesdjursekvationer som presenterades på sid 73 i avsnitt 6.3, dvs

$$\begin{cases} y_1'(t) = \dfrac{dy_1}{dt} = c_1 y_1 - c_2 y_1 y_2 \\ y_2'(t) = \dfrac{dy_2}{dt} = -c_3 y_2 + c_4 y_1 y_2 \end{cases}$$

där $y_1(t)$ var djur av arten som är byte och $y_2(t)$ antalet djur av arten som är jägare, och t är tiden i dagar.

Liksom tidigare ska högerledet definieras i en funktion i MATLAB. När det är fler än en ekvation måste det skrivas om på vektorform. Om man sätter

$$y'(t) = \begin{pmatrix} y_1'(t) \\ \\ y_2'(t) \end{pmatrix}, \quad f(t, y_1, y_2) = f(t, y) = \begin{pmatrix} c_1 y_1 - c_2 y_1 y_2 \\ \\ -c_3 y_2 + c_4 y_1 y_2 \end{pmatrix}$$

så kan systemet formuleras som

$$y'(t) = f(t, y)$$

dvs det blir samma formulering som när det var en ekvation, med den skillnaden att y och $f(t, y)$ nu är vektorer. Detta högerled kan nu matas in i MATLAB.

För att göra detta skapar man en funktion i MATLAB, som vi exempelvis kan kalla för `diffEkvSys.m`. Precis som tidigare (sid 73, avsnitt 6.3) använder vi här konstanterna $c_1 = 1$, $c_2 = 0.007$, $c_3 = 1$ och $c_4 = 0.01$ och får följande MATLAB-kod:

```
function yprim = diffEkvSys(t,y)
c1 = 1; c2 = 0.007; c3 = 1; c4 = 0.01;
xprim = [c1*y(1) - c2*y(1)*y(2);
        -c3*y(2) + c4*y(1)*y(2)];
```

Funktionen skickar man sedan till `ode45`. Som inparametrar har man också en radvektor för tidsintervallet, och en kolonnvektor där de två elementen är begynnelsevärdena. Kommandosekvensen för att lösa problemet blir då

```
y0 = [100; 5];   % Begynnelsevärden
t0 = 0;
tmax = 10;
[t y] = ode45(@diffEkvSys,[t0 tmax],y0);
plot(t,y)
title('Löjor och gäddor, Volterras jägare-bytesdjurekvationer')
xlabel('t')
ylabel('y(t)')
legend('y_1(t)','y_2(t)')
```

Lösningsgrafen visas i Figur 24.

Lösning av differentialekvationer	
`[t,y]=ode45(` `@fkn,tspan,y0)`	beräknar lösningen till $y' = f(t, y)$ Inparametern fkn är en MATLAB-funktion där högerledet $f(t, y)$ definieras, `tspan` är en vektor som innehåller startpunkt och slutpunkt för beräkningen, `y0` är begynnelsevärdet. Metoden är lämplig som förstahandval om man inte har några speciella skäl att använda någon annan metod.
`[t,y]=ode23(` `@fkn,tspan,y0)`	Samma som `ode45`, men använder en metod av lägre noggranhetsordning. Lämplig då man har lägre noggranhetskrav.
`[t,y]=ode15s(` `@fkn,tspan,y0)`	Samma som `ode45`, men använder en metod som är lämplig att använda då man har s k styva problem.

6.5 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

6.1 Beräkna integralen

$$\int_0^1 e^{-x^2} dx$$

med `quad` och `quadl`.

6.2 Hastigheten för en löpare har under tio sekunder mätts upp till följande:

t (s)	1	2	3	4	5	6	7	8	9	10
v (m/s)	5.2	8.3	10.1	10.2	10.0	8.7	9.9	11.0	11.3	11.2

Hur lång sträcka sprang löparen under de tio sekunderna?

6.3 Antag att man har differentialekvationen $y'(t) = y(t)$. Den exakta lösningen är $y(x) = e^x + C$, där C är en konstant. Lös ekvationen numeriskt med `ode45` på $0 \leq t \leq 2$. Använd $C = 0$ och rita lösningskurvan för den exakta lösningen och den numeriska lösningen. Olika begynnelsevärden medför lösning för olika värden på C (testa i MATLAB). Vilket begynnelsevärde ska användas om $C = 0$?

6.4 Granbarkborren är en av de värre skadedjuren för granar i Sverige. Den söker sig till skadade träd, där den utför näringsgnag och förökar sig. Vid tillräckligt stora populationer sprider sig även borrarna till friska träd och utgör därmed en stor ekonomisk katastrof för markägaren.

Man vill konstruera en modell för ökningen av population av granbarkborre. Till att börja med antas att tillväxten är konstant proportionell mot storleken på populationen. Om $W(t)$ betecknar storleken av populationen vid tiden t, då leder antagandet till ekvationen

$$\frac{dW}{dt} = rW.$$

Lösningen blir $W(t) = W_0 e^{rt}$, där W_0 betecknar populationens storlek från början, $W(0)$.

a) Rita lösningskurvan för $r = 0.48$, $W_0 = 4$ och $0 \leq t \leq 20$. Vad händer om du ökar tiden?

   Använd `ode45` för att lösa problemet. Rita lösningskurvorna i ett och samma grafikfönster för följande värden på begynnelsevillkoret W_0: 2, 4, 5, 6, 8, 10, 12, 14, 16 och 20.

I de flesta fall är en exponentiell tillväxtmodell orealistiskt. Ett skäl är att populationer inte kan öka i oändlighet. Man lägger därför till antagandet att om populationen blir för stor, så blir tillväxten negativ. Det kan man göra genom att lägga till en "strypningsterm". Den ska inte påverka tillväxten mycket när populationen är liten (nära noll), eftersom små populationer inte behöver räkna med några problem att hitta träd. När populationen blir för stor, då ska tillväxten bli negativ. Alltså ska tillväxten blir negativ när populationsstorlekten överskrider en viss tröskel, K, vilken biologer kallar *omgivningens bärförmåga*. Den bestäms av livsvillkorsfaktorer som mängden träd. Den nya modellen blir

$$\frac{dW}{dt} = rW(1 - \frac{W}{K}).$$

r och K kallas parametrarna i modellen. Denna modell kallas ofta logistisk tillväxtmodell eller Verhults logistiska ekvation och skrivs ofta på formen $dW/dt = aW - bW^2$.

b) För $K = 15$ rita lösningskurvorna i samma grafikfönster för begynnelsevillkoren angivna i a). Använd `ode45`.

   Rita lösningkurvorna även för $K = 10$. Hur påverkar valet av K populationen?

Man kan också tänka oss situationen att det dessutom finns fåglar som äter granbarkborre. Låt $P(W)$ representera antalet granbarkborrar som fåglarna äter upp. Man vet följande:

1. $P(0) = 0$, eftersom fåglarna inte kan äta några maskar om det inte finns några att äta.
2. Om W är litet, så är också P litet av samma orsak som ovan.
3. Fåglarnas aptit har en tröskel. Efter den tröskel kommer fåglarna äta några, men troligen inte alla maskar.

Resultatet blir

$$P(W) = \frac{aW^2}{b^2 + W^2},$$

där a och b är positiva parametrar. I detta fall antar man att $a = b = 2$. Populationsmodellen blir nu

$$\frac{dW}{dt} = rW(1 - \frac{W}{K}) - \frac{2W^2}{4 + W^2} = f(W)$$

c) Rita lösningskurvorna för samma K och W_0 som i b).

6.5 I modeller av konkurrerande arter antas ibland att konkurrensen utgörs av negativa interaktioner, dvs arterna missgynnas av att de finns samtidigt. Modellerna som beskriver sådana system är ofta formulerade som kopplade differentialekvationer, dvs differentialekvationer som beror av varandra. I detta exempel tittar man på mängden gråsparvar (g) och pilfinkar (p). I frånvaron av konkurrens inom eller mellan arterna antas de växa exponentiellt. Det beskrivs av differentialekvationen

$$\begin{cases} \frac{dg}{dt} = k_1 \cdot g \\ \frac{dp}{dt} = k_2 \cdot p \end{cases}$$

eftersom lösningen till $y' = k \cdot y$ ges av $y(t) = C \cdot e^{k \cdot t}$. Dessa ekvationer förutspår alltså en accelererande tillväxt av de båda arterna. Även om detta ibland hyfsat beskriver den första perioden när en ny nisch exploateras, kommer snart den ökade mängden individer leda till att resurserna (mat eller dylikt) blir begränsande.

När de två arterna inte konkurrerar med varandra beskrivs ofta processen som

$$\begin{cases} \frac{dg}{dt} = k_1 \cdot g - k_3 \cdot g^2 \\ \frac{dp}{dt} = k_2 \cdot p - k_4 \cdot p^2 \end{cases}$$

De kvadratiska termerna betyder att man antar att antalet interaktioner, dvs möten inom arten, är proportionellt mot kvadraten av antalet individer och att konkurrensen inom arten är proportionellt mot antalet interaktioner. Detta får man från att ett möte kräver två individer. Tre gånger så många individer ger tre gånger så många kandidater av båda och nio gånger fler möten, därav den kvadratiska effekten.

Konkurrens *mellan* arter representeras på samma sätt men med andra interaktionskonstanter

$$\begin{cases} \dfrac{dg}{dt} = k_1 \cdot g - k_3 \cdot g^2 - k_5 \cdot g \cdot p \\ \dfrac{dp}{dt} = k_2 \cdot p - k_4 \cdot p^2 - k_6 \cdot p \cdot g \end{cases}$$

Icke-linjära kopplade dynamiska system, som systemet ovan är ett exempel på, är ofta svåra att förståoch lösa. Många system är olösbara analytiskt. I MATLAB kan man däremot generera en numerisk lösning för givna värden av konstanterna. Samma modell beskriver också vissa kemiska reaktioner.

Uppgiften är att beräkna hur ett sådant system utvecklar sig i tiden för olika startvärden. Använd en uppsättning av konstanterna $k_1, \ldots, k_6$ så att

$$\begin{cases} \frac{dg}{dt} = 3 \cdot g - g^2 - 2 \cdot g \cdot p \\ \frac{dp}{dt} = 2 \cdot p - p^2 - p \cdot g \end{cases}$$

Denna uppsättning innebär att gråsparvar växer snabbare om resurserna är obegränsade, att den interna konkurrensen inom de båda arterna är samma samt att pilfinkarna påverkas minst av konkurrensen mellan arterna. Testa följande startvärden

gråsparvar	pilfinkar
0.1	0.1
0.02	0.1
0.034	0.1
0.036	0.1
5	2.2
1.9	2.5
3.5	2.5
3.65	2.5

Observera att värdena inte direkt ger antalet. I denna modell kan g och p anta vilka värden som helst. Vill man ta hänsyn till att det inte kan finnas t ex 2.3 gråsparvar måste modellen redigeras radikalt. Det första steget i sådant mer komplicerat modellarbete är ändå att först förstå modellen ovan.

Gör en graf med båda arterna för varje startvärde, dvs totalt åtta stycken med två kurvor i varje. Använd kommandot `subplot`.

Lös uppgiften med hjälp av `ode45` och givetvis använder du en m-fil där olika parametrar sätts, anropet av `ode45` görs och plottning sker. Prova gärna vad som händer om du anger $p(0) = 1$ och $g(0) = 1$ som startvärden.

7 Programmering i MATLAB

I detta kapitel visas hur man skriver program i MATLAB. Genom att skriva program kan man lösa mer komplicerade problem än när man bara använder MATLAB interaktivt. Det ger också en möjlighet att skapa nya kommandon.

7.1 Grundläggande begrepp

I föregående kapitel har många exempel på *kommandosekvenser* givits, sekvenser med kommandon som kan skrivas in direkt ett efter ett vid MATLAB-promptern, eller i en m-fil. Sådana filer brukar man också kalla *kommandofiler*.

En kommandosekvens som lagras i en m-fil, dvs en kommandofil är ett enkelt exempel på ett *program*. Det finns i MATLAB också en annan typ av program som också lagras i m-filer, nämligen *funktioner*. Mer om funktioner i avsnitt 7.8. Program i MATLAB skrivs alltså i m-filer, på samma sätt som beskrivs i avsnitt 3, antingen i form av *kommandofiler* eller *funktioner*.

Ett program innehåller dock normalt också speciella satser (ord), s k *styrande satser*, som styr i vilken ordning kommandon ska utföras. Med hjälp av dessa satser kan man bryta den linjära strukturen i en kommandosekvens, vissa kommandon kan hoppas över medan andra kan upprepas.

De styrande satserna kan delas in i de två grupperna *villkorssatser*, där vissa kommandon bara utförs om något villkor är uppfyllt, och *repetitionssatser*, där vissa kommandon upprepas enligt något villkor. Dessa två grupper av satser presenteras i avsnitt 7.3 respektive avsnitt 7.4.

För båda grupperna av styrande satser finns alltså ett behov av att kunna uttrycka *villkor*, dvs det som avgör vilka kommandon som ska utföras. Ett sätt att uttrycka villkor är med *logiska uttryck*, uttryck som resulterar i *sant* eller *falskt*. Hur man bildar logiska uttryck beskrivs i avsnitt 7.2.

7.2 Logiska uttryck

Ett logiskt uttryck är ett uttryck som antingen är sant eller falskt. I MATLAB motsvaras falskt av talet 0 och sant av talet 1.

Det vanligaste fallet är att ett logiskt uttryck består av att en relation testas, t ex två variabler jämförs för att undersöka om de är lika (har samma värde).

För det ändamålet finns s k *relationsoperatorer*.

Relationsoperatorer

Relationsoperatorer är till för att undersöka relationen mellan *operanderna*, dvs det som står till vänster och höger om operatorn. Resultatet blir 1 om relationen är sann och 0 om den är falsk.

Om man t ex vill skriva ett logiskt uttryck som testar om variabeln x är större än tre skriver man `x > 3`. Beroende på vilket värde x har, kommer resultatet bli antingen 1 eller 0.

I rutan *Relationsoperatorer* visas vilka relationsoperatorer som finns.

Relationsoperatorer	
`==`	likhet
`~=`	olikhet (skilt från)
`<`	mindre än
`<=`	mindre än eller lika med
`>`	större än
`>=`	större än eller lika med

Logiska operatorer

Det går att kombinera flera villkor och då måste man använda något som kallas *logiska operatorer*.

Antag att man vill undersöka om både uttrycket "x är större än tre" och uttrycket "y är lika med ett" är sanna. Då kan man sammanfoga de båda logiska uttrycken med den logiska operatorn *och* som i MATLAB skrivs som en ampersand (`&`), t ex `x > 3 & y == 1`.

I ett uttryck har aritmetiska operationer högst prioritet, därefter kommer relationsoperatorerna, och lägst prioritet har de logiska operatorerna. För att ändra prioritetsordningen kan man använda parenteser.

De logiska operatorerna finns sammanfattas i rutan *Logiska operatorer*.

Logiska operatorer	
`&`	logiskt *och*, när man vill att flera villkor måste uppfyllas. Uttrycket blir sant om *båda* ingående operanderna/villkoren är sanna.
`|`	logiskt *eller*, när man vill att något av villkoren ska uppfyllas. Uttrycket blir sant om någon eller båda av de ingående operanderna är sanna.
`~`	logisk *negation*, när man inte vill att ett villkor ska uppfyllas. Uttrycket blir sant när villkoret inte är uppfyllt.
`xor`	logisk *exklusivt eller*, betyder *antingen eller*, **ej** både och. Uttrycket blir sant om någon av de båda ingående operanderna, men *inte båda*, är sanna.

7.3 Villkorssatser

Med villkorssatser kan man styra vilka kommandon som ska utföras. I MATLAB finns det två villkorssatser: `if` och `switch`.

If-satsen

If-satsen är en villkorssats som fritt kan översättas till ”om så - annars” -satsen. Den generella formen av en if-sats ser ut så här

```
if logiskt utryck
  MATLAB-kommandon
else
  MATLAB-kommandon
end
```

Kommandona mellan `if` och `else` kommer utföras enbart om det logiska uttrycket är sant. Är det inte det så utförs satserna efter `else` istället. Det måste vara ett mellanslag mellan `if` och det logiska uttrycket. En if-sats avslutas med ett `end`.
Det är inte nödvändigt att använda en else-gren i en if-sats. Behöver man den inte går det lika bra att skriva

```
if logiskt uttryck
  MATLAB-kommandon
end
```

Antag att man har skapat två variabler x och y. En if-sats kan då se ut så här:

```
if x >= 0
  y = sqrt(x);
else
  disp('Talet negativt')
end
```

Ovanstående if-sats undersöker om variabeln x är större än eller lika med noll. Om den är det så sätts y-variabeln till kvadratroten av x, dvs om x är 4 kommer y att bli 2. Annars skrivs ett meddelande om att talet är negativt ut på skärmen. Om man inte vill utföra något när x är mindre än 0 kan man hoppa över else-grenen.

switch-satsen

En switch-sats kan välja ut en av flera grupper av kommandon som ska utföras. Den generella formen av en switch-sats ser ut så här

```
switch uttryck
case värde 1
  MATLAB-kommandon
case värde 2
  MATLAB-kommandon
case värde 3
  MATLAB-kommandon
...
otherwise
  MATLAB-kommandon
end
```

Uttrycket som står efter `switch` beräknas och ska ge en skalär, dvs ett tal som resultat. (Det fungerar också med en sträng som resultat, för mer information om strängar se avsnitt 7.6). Resultatet jämförs sedan med de olika värdena som står efter raderna som börjar med `case`. Om värdena är lika utförs de efterföljande kommandona. Om inget värde överenstämmer utförs

kommandona efter `otherwise`.

Antag att man har definierat en variabel *val*.

```
switch val
case 0
  x = 1 ; y = 2 ; z = 3 ;
case 1
  x = 3 ; y = 1 ; z = 2 ;
case 2
  x = 2 ; y = 3 ; z = 1 ;
otherwise
  x = 0 ; y = 0 ; z = 0 ;
end
```

Om *val* har något av värdena 0, 1 eller 2 kommer variablerna x, y och z att tilldelas värdena som står efter motsvarande `case`. Annars tilldelas variablerna värdet 0.

7.4 Repetitionssatser

Med repetitionssatser kan man styra ett program så att vissa kommandon upprepas ett antal gånger. En sådan slinga där ett antal kommandon upprepas kallas för en *loop*. I MATLAB finns det två repetitionssatser: `for` och `while`.

for-satsen

Med hjälp av en for-loop kan man göra en beräkning flera gånger utan att behöva skriva ner den mer än en gång. Man talar helt enkelt om för MATLAB hur många gånger man vill utföra beräkningen. Den generella formen för en for-loop ser ut så här

```
for variabel = uttryck
  MATLAB-kommandon
end
```

där `variabel` är namnet på den s k *loop-variabeln* och `uttryck` talar om vilka värden loop-variabeln ska anta. Uttrycket skrivs på formen

```
startvärde : tillväxtvärde : slutvärde
```

Tillväxtvärdet kan hoppas över och kommer då automatiskt att sättas till 1.

Första gången kommandona utförs är loop-variabeln satt till startvärdet. Sedan läggs tillväxtvärdet succesivt till loop-variablen innan kommandona utförs, och repetitionen pågår tills loop-variabeln antar slutvärdet.

Antag att man vill räkna ut fakultet av ett tal n, dvs $n!$. Det kan se ut så här

```
y = 1;
for x = 1:1:n
  y = y*x;
end
y
```

I uttrycket `x = 1:1:n` betyder första 1:an att startvärdet är 1. Andra 1:an betyder att man ökar x med 1 i varje varv av loopen. I detta fall hade det inte spelat någon roll om man skrivit uttrycket som `x = 1:n` och hoppat över 1:an i mitten eftersom tillväxtvärdet ändå satts till 1. Jämför med kolonnotation. Sista bokstaven n talar om vid vilket värde på x som man ska sluta vid.

Om man vill räknat ut 5! så sätter man $n = 5$ och kör for-loopen ovan. Följande beräkningar utförs varv för varv i loopen:

varv 1:	$y = 1 * 1$
varv 2:	$y = 1 * 2$
varv 3:	$y = 2 * 3$
varv 4:	$y = 6 * 4$
varv 5:	$y = 24 * 5$

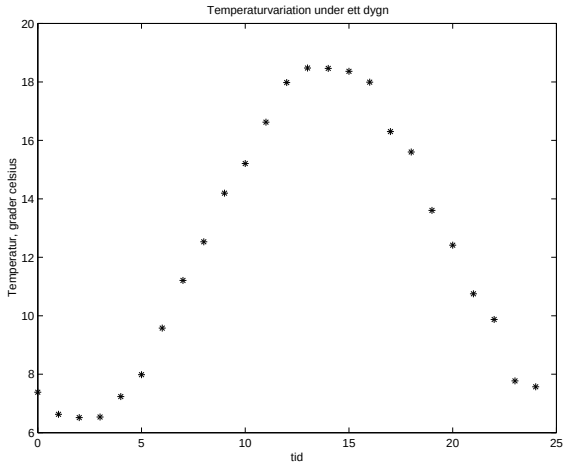
Resultatet blir $y = 120$ $(= 1 * 2 * 3 * 4 * 5 = 5!)$.

Som ytterligare exempel på en for-loop kan man t ex rita upp hur lufttemperaturen under ett dygn i maj varierade enligt funktionen

$$T = 12 + 6sin(\tfrac{\pi(t-8)}{12})$$

där t är tiden i timmar mätt från midnatt. En slumpvariabel är inlagd för att simulera små fluktuationer. Resultatet visas i Figur 25.

```
for t = 0:24
```



Figur 25: Temperaturvariation under ett dygn i maj.

```
  T = 12 + 6*sin(pi*(t - 8)/12) + rand;
  plot(t,T,'*');
  hold on;
end
title('Temperaturvariation under ett dygn')
xlabel('tid')
ylabel('Temperatur, grader celsius')
```

while-satsen

En annan variant av repetition är while-loopen där kommandon upprepas så länge ett fortsättningsvillkor är uppfyllt. Den generella formen för en while-loop är

```
while logiskt uttryck
  MATLAB-kommando
end
```

Precis som for-loop och if-sats måste även while-loopen avslutas med ett *end*. Till skillnad från for-loopen finns det ingen loopvariabel, utan det är det logiska uttrycket som talar om när loopen ska upphöra. Det är alltså viktigt att se till att det logiska uttrycket någon gång blir falskt för att inte loopen ska bli oändlig. Exempel på en while-loop

```
sum = 0;
```

```
while sum < 10
  sum = sum + sum
end
```

Eftersom $sum = 1$ innan loopen kommer först $1 < 10$ att vara korrekt och man går in i loopen. I loopen sker sedan 1+1=2 och $2 < 10$ så loopen utförs igen o s v. Om *sum* sätts till 0, får *sum* värdet 0 hela tiden och $sum < 10$ blir alltid uppfyllt. Man får en oändlig loop. Sätts *sum* däremot till 10 eller mer kommer man aldrig in i loopen eftersom villkoret inte uppfylls.

Som exempel på en while-loop kan man testa följande lilla program som gör en räknekontroll i MATLAB.

```
count = 0;              % räknar antalet varv i while-loopen
x = 0.1                 % startvärde på x
while x > 0             % så länge x är större än 0
  x = x^2               % kvadrera x
  count = count + 1     % ett nytt varv inne i loopen
end                     % avslutar while-loopen
```

Denna lilla programsnutt borde resultera i en oändlig loop eftersom x kommer närma sig 0 hela tiden men aldrig bli exakt 0, dvs villkoret borde alltid uppfyllas. Så är nu emellertid inte fallet eftersom MATLAB räknar med ett visst antal decimaler, se avsnitt 1.4 på sid 12. Avrundningen kommer så småningom göra att villkoret inte längre uppfylls eftersom x till slut sätts till 0.

7.5 Nästlade villkor och loopar

Det går att nästla styrande satser i varandra. Både loopar och villkorssatser kan innehålla en eller flera andra loopar och villkorssatser.

Som exempel på nästlade satser används loopen från sidan 88 som beräknar fakultet för ett tal n. Genom att nästla loopen med en if-sats kan man beräkna fakultet om $n > 0$, annars sätta resultatet till 0.

```
if n > 0
  y = 1
  for x = 1:1:n
    y = y*x
  end
```

```
else
  y = 0
end
```

I kommande avsnitt ges ytterligare exempel på nästlade satser.

7.6 Strängar

Förutom numeriska data går det också att hantera text i MATLAB. På liknande sätt som man skapar variabler som inneåller tal kan man skapa variabler som innehåller text. Vid programmering har man nytta av att kunna behandla text, t ex när man vill läsa in något från användaren av programmet eller skriva ut något på skärmen.

Text lagras som s k *strängar* och definieras med apostrofer runt texten, t ex

```
>> stad = 'Uppsala'
stad =
Uppsala
```

Texten lagras i en radvektor med ett tecken i varje element. I exemplet är alltså variabeln *stad* en radvektor med 7 element.

Strängar har använts i många exempel i kompendiet för att skriva ut ledtexter i grafikfönster. För att t ex ge ett grafikfönster en titel har kommandot `title('Namn på grafikfönster')` getts. Som alternativ kan man först skapa en strängvariabel `str = 'Namn på grafikfönster'` och sedan ange namnet på strängen i `title`-kommandot: `title(str)`.

Strängar går att sätta ihop på samma sätt som man sätter ihop vanliga radvektorer med tal. Om man skriver `['Hej' 'på' 'dig']` tolkar MATLAB det som strängen 'Hejpådig'. Ett alternativ är att använda kommandot `strcat('Hej','på','dig')`.

Det går att konvertera ett tal till en sträng med kommandona `num2str` (2:an i kommandot ska uttalas "to", vilket ger "num-to-string") och `int2str`. Kommandot `num2str` konverterar talet till en sträng i så kallat flyttalsformat, med fyra siffror och exponent. Med kommandot `int2str` blir resultatet ett heltal. Ett exempel på när det kan vara användbart att konvertera tal till strängar är när man vill göra titlar till grafer mer informativa genom att ange vissa variablers värden. I exemplet på sidan 54 anpassas ett polynom till några

punkter. Istället för titeln "Anpassat andragradspolynom" kan man lägga in själva polynomet i titeln med kommandot

```
title(['p(x)=' num2str(p(1)) 'x^2+' num2str(p(2)) 'x' num2str(p(3))])
```

Titeln på grafen blir då

$$p(x) = -0.02555x^2 + 0.30385x - 0.2045$$

När man skriver program vill man ibland skriva ut ledtexter i kommandofönstret till användaren. Till det ändamålet finns det två kommandon: `disp` och `input`.

Kommandot `disp(str)`skriver ut strängen *str* som ledtext till användaren.

```
>> disp('Det här programmet beräknar integraler numeriskt.')
Det här programmet beräknar integraler numeriskt.
```

Kommandot `in = input(out)` skriver ut strängen *out* på skärmen och väntar på att användaren ska mata in ett tal. Det tal användaren matar in lagras i *in*. Exempel:

```
>> x = input('Hur många tal vill du ange? ')
Hur många tal vill du ange? 3
x =
     3pwd
```

För att läsa in text lägger man till strängen 's' som andra parameter till `input`-kommandot, dvs kommandot `in = input(out,'s')` läser in en sträng till variabeln *in*. Exempel:

```
>> funk = input('Ge en funktion: ','s')
Ge en funktion: sin(x)*x
funk =
sin(x)*x
```

I program kan det också finnas behov av att jämföra två strängar med varandra. Kommandot `strcmp(str1,str2)` jämför om strängarna *str*1 och *str*2 är lika, isåfall returneras 1 (= sant), annars 0 (= falskt). Antag som exempel att man i en del av ett större program vill fråga användaren om en ny funktion ska läsas in:

```
disp(['Följande funktion är given: ' funk])
svar = input('Vill du ange en ny funktion (ja/nej)? ','s');

if (strcmp(svar,'ja'))
   funk = input('Ange funktion: ','s');
end
```

Vilket resulterar i följande när denna del av programmet körs:

```
Följande funktion är given: sin(x)*x
Vill du ange en ny funktion (ja/nej)? ja
Ange funktion: exp(x)/x - 1
```

Se rutan *strängkommandon* för en sammanfattning av de strängkommandon som gåtts igenom i det här avsnittet. Många fler kommandon finns dock, ge kommandot `help strfun` för mer information.

Strängkommandon	
`int2str(n)`	konverterar heltalet n till en sträng i heltalsformat.
`num2str(f)`	konverterar skalären f till en sträng i flyttalsformat, med fyra siffror och exponent.
`strcat(str1, str2,...)`	sätter samman strängarna $str1$, $str2$, osv till en enda sträng.
`strcmp(str1, str2)`	jämför strängarna $str1$ och $str2$. Om de är lika returneras 1, annars 0.
`disp(str)`	skriver ut strängen str på skärmen.
`in = input(out)`	skriver ut strängen out på skärmen och väntar på att användaren ska mata in ett tal. Det tal användaren matar in lagras i in.
`in = input(out,'s')`	som ovan, men det användaren matar in lagras som en sträng i in.

7.7 Exempel: Monte Carlo simulering

Monte Carlo simuleringar är vanliga i t ex tillämpade naturvetenskapliga ämnen som kemi och biologi. Man tar hjälp av slumpen för att simulera ett komplext förlopp. En mycket enkel variant av en Monte Carlo simulering är att bestämma integralen för någon funktion med hjälp av slumptal. I exemplet nedan bestämmer man arean av en kvartscirkel med radien 1 (area=$\pi/4$).

Man tänker sig att någon kastar pil mot en måltavla i form av en kvadrat med sidan 1. På tavlan finns en kvartscirkel med radie 1 och centrum i tavlans nedre vänstra hörn.

Följande ritar upp måltavlan i ett grafikfönster.

```
nollTillEtt = linspace(0,1);
kvartsCirkel = sqrt(1 - nollTillEtt.^2);

clear
figure(1)                % Skapar nytt grafikfönster
clf

plot(nollTillEtt, kvartsCirkel)
title('Approximation av \pi')
axis('square')           % Gör ritytan kvadratisk. Det har ingen
hold on                  % inverkan på resultatet men cirkeln
                         % liknar mer en cirkel.
```

Pilkastaren antas kasta helt slumpmässigt inom kvadraten. Då kommer antalet kast som ligger inuti kvartscirkeln dividerat med totala antalet kast att approximera $\pi/4$. Genom att multiplicera kvoten med 4 får man ett närmevärde för π. Man låter pilkastaren kasta så många pilar att man får π med två korrekta decimaler. Slutligen skriver man ut hur många kast som behövdes samt närmevärdet för π.

Efter varje kast kontrollerar man om kastet hamnade i cirkeln. Det gör man genom att beräkna avståndet till origo (= nedre vänstra hörnet) med pythagoras sats. Om avståndet är mindre än 1 så är kastet innanför cirkeln. För att illustrera pilkastningen ritas alla kasten ut.

```
maxFel = 0.5e-3;         % Största tillåtna fel.
ok = 0;                  % Logisk variabel som sätts till 1 när
                         % tillräcklig noggrannhet har uppnåtts.
antalKast = 0;           % Inga pilar kastade ännu.
antalICirkel = 0;        % Inga pilar i cirkeln ännu.
maxKast = 1000;          % Maximalt 1000 kast.

while (antalKast < maxKast & ~ok)
  slumpX = rand;          % Koordinater för pilkastet
  slumpY = rand;          % rand ger ett tal mellan 0 och 1
```

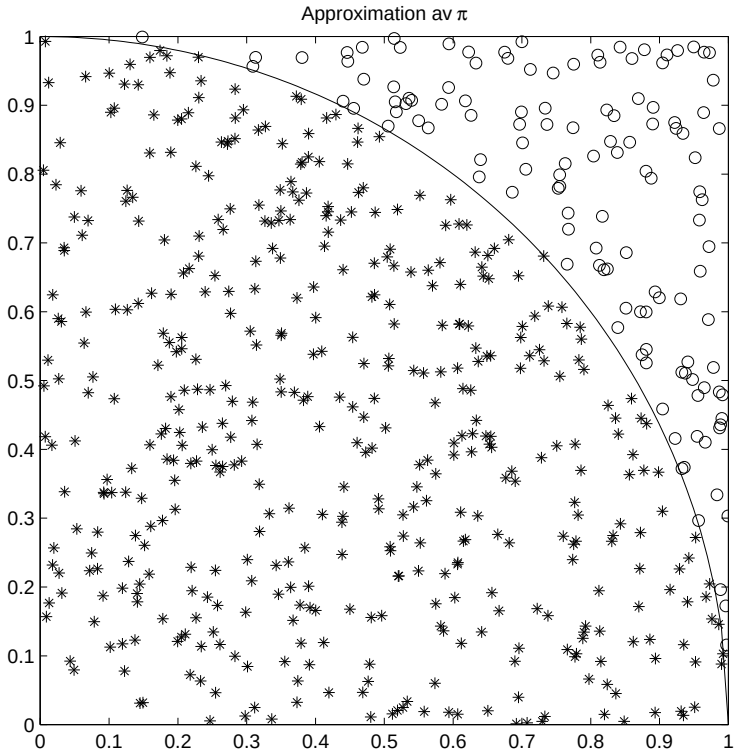
```
  % Komtroll om kastet hamnade i cirkeln. Det gör vi genom att beräkna
  % avståndet till origo (= nedre vänstra hörnet) med Pythagoras sats.
  % Om avståndet är mindre än 1 så är vi innanför cirkeln.

  avst = slumpX*slumpX + slumpY*slumpY;
  if (avst < 1)
    plot(slumpX, slumpY, '*')
    antalICirkel = antalICirkel+1;
  else
    plot(slumpX, slumpY, 'o')

  end

  antalKast = antalKast + 1;
  piApprox = 4*antalICirkel/antalKast;
  ok = abs(pi - piApprox) < maxFel;
end

hold off

if (antalKast == maxKast)
  disp('Maximala antalet kast har uppnåtts.')
else
  piApprox
  antalKast
end
```

Utskriften från MATLAB blir

```
piApprox =
    3.1414
antalKast =
   587
```

Vid denna körning av programmet behövdes alltså 587 kast och π approximerades då till 3.1414. Det kan jämföras med att π avrundat till 4 decimaler är 3.1416. I Figur 26 visas grafen som programmet ritade.

I programmet visas också exempel på användandet av en logisk variabel, dvs en variabel som representerar värdet sant eller falskt (se avsnitt 7.2 om



Figur 26: Pilkastning för bestämning av arean av en kvartscirkel med radien 1 (area=$\pi/4$) med hjälp av Monte Carlo-simulering.

logiska uttryck.). Variabeln används för att komma ihåg om något villkor är uppfyllt. I det här fallet används variabeln *ok*, som anger om det uppskattade värdet på π är tillräckligt noggrannt. När *ok* blir sann kommer while-loopen att avslutas.

7.8 Egna funktioner

I kapitel 7.1 nämndes att program också kan skrivas som egna funktioner (som också lagras i m-filer). Dessa funktioner är jämförbara med vilken i MATLAB inbyggd funktion som helst, t ex `sin` och `polyval`. I själva verket är alla kommandon i MATLAB funktioner.

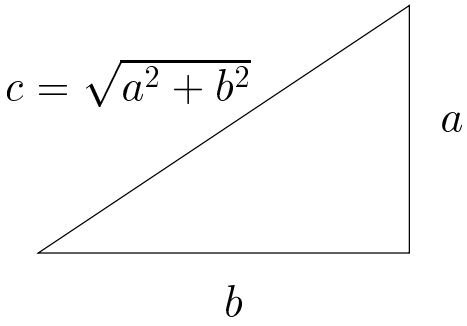
En funktion ska ligga i en egen m-fil som namnges med ett lämpligt namn avslutat med `.m`, t ex `MinFunktion.m`. Filnamnet utom `.m`, dvs här `MinFunktion` måste överensstämma med funktionsnamnet, annars hittar inte MATLAB funktionen. Första raden i filen måste innehålla ordet `function` och se ut på

följande sätt

```
function utparametrar = funktionsnamn(inparametrar)
```

Funktionen kan sedan anropas med `funktionsnamn(inparametrar)`.

Antag som exempel att man vill skriva en funktion som givet de två kateterna beräknar hypotenusan i en rätvinklig triangel.



Man skapar en m-fil, `hypotenusa.m`, med följande innehåll

```
function hyp = hypotenusa(kat1,kat2)
hyp = sqrt(kat1^2 + kat2^2);
```

Funktionen kan sedan anropas från ett program med `c = hypotenusa(a,b)`. Efter anropet har variabeln c värdet $\sqrt{a^2 + b^2}$. Observera att funktionsnamnet överensstämmer med filnamnet. Observera också att variablerna inte behöver ha samma namn i funktionsfilen som i filen som funktionen anropas ifrån.

Lokala och globala variabler

En funktion kan betraktas som en svart låda, där all kommunikation med omvärlden sker via in- och utparametrarna. Variabler som enbart skapas och finns inne i funktionen existerar inte utanför (t ex i kommandofönstret). Man säger då att alla variabler inne i funktion är *lokala*. Detta till skillnad från kommandofiler där alla variabler är *globala*, dvs variablerna som skapas i kommandofilen finns också i MATLAB:s kommandofönster.

Ibland vill man ändå komma åt variabler i funktionsfilen utan att de är angivna i parameterlistan till funktionen. Det som behövs är att deklarera variablerna som `global`. Det gör man genom att skriva `global` och sedan en lista med alla varibler som man vill ska vara globalt deklarerade. Variablerna ska skiljas åt med mellanslag. Detta måste göras innan variablerna används

och i alla filer där de ska användas. Variabler som en gång deklarerats som `global` är globala värden ända till du avslutar programmet, alternativt tar bort variablerna med `clear global`. Om man använder globala variabler i exemplet på sidan 73 så skulle funktionen kunna se ut så här

```
function xprim = diffEkvSys(t,x)
global c1
global c2
global c3
global c4
xprim = [c1*x(1) - c2*x(1)*x(2) ;
         -c3*x(2) + c4*x(1)*x(2)];
```

I kommandofilen kan man sedan skriva

```
global c1
global c2
global c3
global c4
c1 = 1;
c2 = 0.007;
c3 = 1;
c4 = 0.01;
[t x] = ode45(@diffEkvSys,[0 10],[100;5]);
```

Om man nu vill ändra en av c_1, c_2, c_3 eller c_4 behöver man bara göra det i kommandofilen.

En lite varning är här på sin plats. Samtidigt som globala variabler kan förenkla programmeringsarbetet kan de också ställa till problem och de bör användas sparsamt. Om man använder många globala variabler i stora program blir programmet lätt oöverskådligt och svårläst. Speciellt för andra programmerare, men även för en själv när det har gått en tid. I de flesta situationer finns det bättre alternativ än att använda globala variabler i funktioner.

7.9 Exempel: lösning av icke-linjära ekvationssystem

Ett problem som är lämpligt att lösa med MATLAB-programmering är att beräkna rötterna till en icke-linjär ekvation $f(x) = 0$. För att göra det kan

man t ex använda Newton-Raphsons metod.

Det kan här tilläggas att det finns ett inbyggt kommando i MATLAB, `fzero`, som kan användas för att beräkna rötter till icke-linjära ekvationer. Se rutan *Funktionsevaluering och nollställen* på sidan 102.

Numeriskt ser lösningen för en ekvation, $f(x) = 0$, med Newton-Raphsons metod ut som

$$\begin{cases} x_0 & \text{givet startvärde} \\ x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)} \end{cases}$$

där x_0 om möjligt sätts till ett värde nära x, dvs om man vet ungefär var nollstället är väljer man ett x_0 i närheten, och $f(x)$ är en godtycklig funktion och $f'(x)$ är dess derivata. Om den enkla roten till $f''(x)$ är kontinuerlig blir konvergensen minst kvadratisk. Figur 27 visar ett mycket enkelt exempel där man når ett resultat ganska nära det verkliga redan efter 2 beräkningar. Ekvationen man försöker räkna ut är $1 - x^2 = 0$ där de verkliga nollställena ju är $x = -1$ och $x = 1$. I figuren har ett startvärde på $x = -0.5$ valts för att hitta det lägre av de två nollställena och redan efter två beräkningar har resultatet $x = -1.025$ erhållits. Vill man ha bättre noggrannhet kan man utföra ytterligare beräkningar.

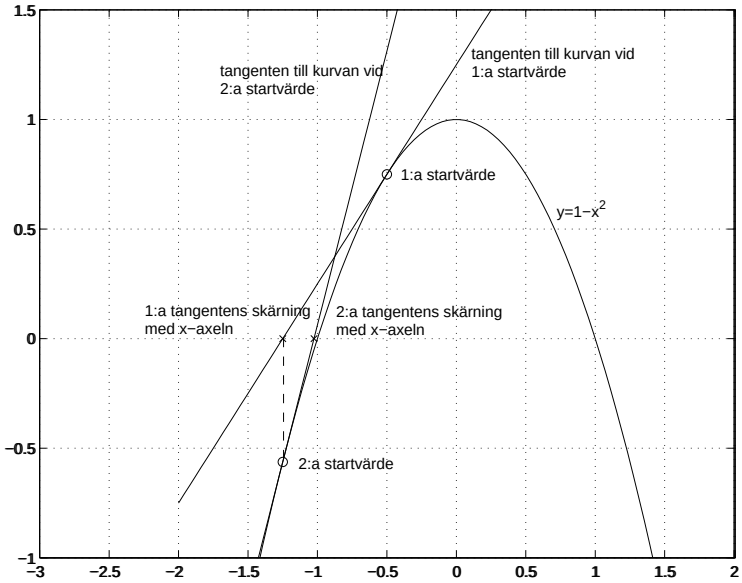
Antag att man är intresserad av att hitta nollställen till funktionen $f(x) = \sin(x)\cos(x)^2$. Man skapar därför en MATLAB-funktion (se avsnitt 7.8) och sparar den i en fil, exempelvis med namnet `funk.m`.

```
function y = funk(x)
y = sin(x)*cos(x)^2;
```

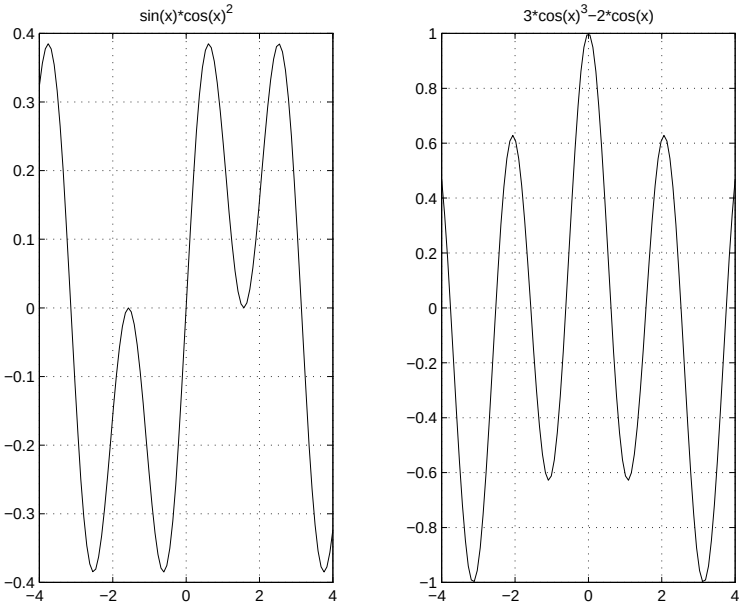
Sedan deriverar man (här gör vi det analytiskt, men det går givetvis att göra numeriskt i MATLAB) funktionen till $f'(x) = 3\cos(x)^3 - 2\cos(x)$ och lägger derivatan i en fil som vi kallar för `funkder.m`.

```
function yp = funkder(x)
yp = 3*cos(x).^3 - 2*cos(x);
```

I Figur 28 ser man hur funktionen och dess derivata ser ut. Där kan man också lätt se ungefär vid vilka x-värden som funktionen har sina nollställen.



Figur 27: Ett exempel på hur uträkningen av ekvationen $1 - x^2 = 0$ kan se ut med hjälp av Newton-Raphsons metod.



Figur 28: Funktionerna man använt i det här exemplet om Newton-Raphson.

Nu har man alla funktioner som definerar $f(x)$ och $f'(x)$ och det som nu behövs är en funktion för att Newton-Raphsons metod. Metoden skrivs här som MATLAB-funktionen *Newton-Raphson* och läggs därför i en m-fil med namnet `NewtonRaphson.m`.

```
function [xzero, fzero] = NewtonRaphson(fx,fprim,intervall)

% [xzero, fzero] = NewtonRaphson(fx,fprim,intervall)
% Funktion som beräknar nollstället, xzero, och motsvarande
% funktionsvärde, fzero, med Newton-Raphsons metod till f(x)=0.
%
% fx        -  funktion där f(x) definieras
% fprim     -  funktion där f'(x) defineras
% intervall -  radvektor med två tal [x1 x2] som talar om
% lägsta och högsta värde på ett uppskattat intervall.
% Startvärdet i Newton-Raphsons metod tas i mitten av
% intervallet.

xold = (intervall(1) + intervall(2))/2;
xnew = 0; % variabel att spara det nya x-värdet i
klar = 0; % kontrollvariabel så att loopen kan brytas
tol  = 10^(-3); % Tolerans anger noggrannheten i beräkningen
% Fortsätt så länge vi är inom intervallet utan att ett
% tillräckligt bra svar erhållits.
% Om skillnaden är tillräckligt liten - sätt klar = 1 (= sant)

while (xold > intervall(1) & xold < intervall(2) & ~klar)
  % räknar ut nästa tal
  xnew = xold - feval(fx,xold)/feval(fprim,xold);
  if ((xnew - xold) < tol)
    klar = 1;   % loopen bryts.
  end           % avslutar if-sats
  xold = xnew;  % uppgradera det gamla x-värdet
end             % avslutar while-loop

xzero = xnew;            % lägg resultatet i variabeln xzero
fzero = feval(fx,xzero); % lägg funktionsvärdet i fzero
```

Kommandot `feval` i programmet ovan evaluerar (beräknar) funktionerna fx respektive $fprim$:s värden för det gamla x-värdet x_{old}. Se rutan *Funktionse-*

valuering och nollställen.

Funktionsevaluering och nollställen	
`feval(fkn,x1, ... ,xn)`	evaluerar funktionen som anges i strängen fkn. Evaluering sker för de givna parametrarna $x_1, ..., x_n$. Funktionen *feval* används vanligen inuti funktioner som har andra funktioner som parametrar.
`[y1,y2,...] = feval(fkn,...)`	returnerar flera utvärden. Annars som `feval`.
`fzero(fkn,x0)`	returnerar ett av nollställena till funktionen som är angiven i strängen fkn. Kommandot kräver att man ger en startgissning x_0.

För att testa funktionen söks ett nollställe i intervallet $[2, 4]$:

```
>> [nollst, funkvarde] = NewtonRaphson(@funk,@funkder,[2 4])
nollst =
    3.1416
funkvarde =
   7.9778e-07
```

Funktionen hittar alltså ett nollstället för $x = 3.1416$ och man ser också att motsvarande funktionsvärde verkar litet, vilket verkar korrekt. Vill man höja noggrannheten i beräkningen får man ändra på variabeln `tol` i funktionen `NewtonRaphson`. Ett alternativ är att även låta `tol` ingå i parameterlistan så kan den ändras via anropet till funktionen.

Nu kan man skriva en kommandofil som frågar efter funktionen $f(x)$, dess derivata och ett intervall och sedan anropar `NewtonRaphson`

```
% nollstalle -- söker nollställen till en funktion i ett givet
% intervall.

f = input('Funktionens namn: ', 's');
fp = input('Derivatans namn : ', 's');
interv = input('Intervall (radvektor med två tal): ');

x = NewtonRaphson(@f,@fp,interv) % Anropa funktionen NewtonRaphson
```

Kommandot `input` används för inläsning (se sidan 92) och funktionerna läses in som strängar medan intervallet läses in som en radvektor. För att inläsningen av en vektor ska fungera måste användaren förstås komma ihåg hakparenteserna runt elementen.

Testkörning med samma funktion och intervall som i exemplet ovan:

```
>> nollstalle
Funktionens namn: funk
Derivatans namn : funkder
Intervall (radvektor med två tal): [2 4]
x =
    3.1416
```

7.10 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

7.1 Använd kommandot `rand` för att skapa slumpvariabeln x. Kontrollera med hjälp av en if-sats

a) om det x du skapat är större än 0.5. I så fall, multiplicera x med 10, annars multiplicerar du talet med 100.

b) om polynomet $y = 7x^2 + 3x - 1$ har $y \leq 0$ eller $y > 100$, sätt i så fall x=100.

c) vilken av variablerna x och y från uppgift a & b ovan som är störst. Det största värdet ska läggas i variabeln *max*.

7.2 Skriv en eller flera if-satser som avgör vilket av tre tal, x, y och z, som är störst. Det största talet ska läggas i variabeln *max*. Se till att alla variabler är definierade.

7.3 Använd en for-loop för att beräkna

a) 100!

b) summan av 1+2+...+99+100

c) det tal som blir om du multiplicerar ihop alla tal i en valfri vektor med 5 tal. *Tips! vektor(i).*

7.4 Skapa en 20 tal lång slumpvektor m h a *randn(1,20)*. Loopa över vektorn och spara det sista negativa tal som dyker upp i variabel *x*. *Tips! vektor(i)*

7.5 Antag att du har formeln $y = y * (-y)$. Sätt y till lämpligt startvärde och repetera beräkningen så länge du har ett tal mellan -100 och 100. Spara alla värden av y i en vektor. Prova t ex startvärde y=2. Glöm inte att skapa vektorn och lägga dit startvärdet innan du använder den. Tips! `vektor=[vektor y]` alternativt `vektor(i) = y, i=i+1`.

7.6 Talet π kan beräknas med formeln $\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + ...)$. Ta reda på hur många termer i summan som behövs för att du ska få π med 3 korrekta decimaler.

7.7 De flesta mutationer som sker i naturen är *neutrala* för organismen, dvs mutationen påverkar inte individens livsduglighet. Sannolikheten att en neutral mutation fixeras i en population är beroende av populationens storlek. Med fixeras menas att populationen slutligen endast innehåller mutanter. Antag att den totala populationen är konstant ($=N$), dvs att antal födslar och dödsfall tar ut varandra, och att det finns *mut* mutanter (och därmed $N - mut$ av vildtyp). Sannolikheten att man vid nästa förändring får ytterligare en mutant beror då av mut. Om det finns 30% mutanter, dvs $mut/N = 0.3$, så är det 30% chans att nästa förändring innebär en till mutant och 70% chans att det blir en mutant färre.

Du ska med hjälp av Monte Carlo simulering i MATLAB simulera spridningen av en neutral mutation i en population.

1. Antag $N = 40$, starta med $n = 10$ mutanter och gör en figur som illustrerar hur antalet mutanter förändrar sig från start tills den fixerats eller försvunnit helt.
2. Gör som i 1. fast byt ut antalet mutanter till $n = 20$ istället.

Kör programmen **tre** gånger och lägg varje körning i en separat subplot. Jämför resultaten från de olika körningarna.

Några tips
Varje beslut om det ska bli en mutant mer eller mindre görs i en if-sats. Generera ett likformigt fördelat slumptal mellan 0 och 1 m h a *MATLAB*-funktionen `rand`. Om talet är mindre än mut/N ska det

bli $mut + 1$ mutanter, annars $mut - 1$. Eftersom denna process ska genomföras tills dess att $mut = N$ eller $mut = 0$, är det lämpligt att använda en `while`-loop. Algoritmen kan sammanfattas som

0. Startvärdet är `N` och `mut`=n.
1. Generar ett slumptal med `rand`
   om talet är mindre än `mut/N` så sätt `mut=mut+1`
   om talet är större än `mut/N` så sätt `mut=mut-1`
2. Spara värdet på `mut` i en vektor, `mutVekt`, t ex genom att skriva `mutVekt=[mutVekt mut]`.

Upprepa steg 1 och 2 tills `mut` är 0 eller N. Vektorn `mutVekt` ritas mot tiden.

8 Symbolisk hantering med MATLAB

MATLAB är ett program för *numeriska* beräkningar. Det innebär att lösningar beräknas i form av siffror. Man kan alltså egentligen inte arbeta med analytiska uttryck, t ex förenklingar av formler. Sådan hantering brukar kallas *symboliska beräkningar* och det finns särskilda programvaror för detta, där Maple är ett exempel. Till MATLAB finns det dock olika tilläggsprogram, s k *toolboxar*, vilka fungerar som extra verktygslådor med kommandon och verktyg som är anpassade för olika tillämningar. En sådan verktygslåda är *Symbolic Toolbox* som gör att MATLAB även kan hantera symboliska variabler. *Symbolic Toolbox* använder sig av ett annat programs beräkningskärna, nämligen Maple. Detta gör att MATLAB kan göra mycket inom området, men inte allt som Maple kan.

Med hjälp av kommandot `help symbolic` får man en översikt över de kommandon som finns i verktygslådan.

8.1 Skillnaden symbolisk – numerisk

När man utför numeriska beräkningar, t ex använder miniräknare eller ett vanligt beräkningsprogram, så presenteras varje lösning i form av siffror. Lösningen är inte beräknad på de sätt som man lär sig i matematiken, utan istället används *numeriska metoder*. Dessa beräknar lösningar i form av approximationer. Normalt märker man inte av detta eftersom approximationen är så liten. Med symbolisk hantering kan matematiska uttryck lagras som just uttryck och inte som numeriska approximationer. Man kan på det sättet hantera matematiska formeluttryck, t ex utföra förenklingar. Man kan också få lösningar i form av formeluttryck. Ett litet exempel på detta är nedanstående egenvärdesberäkning av en generell 2×2-matris.

```
>> syms a b c d;
>> eig([a b; ...
        c d])
ans =
[ 1/2*a+1/2*d+1/2*(a^2-2*a*d+d^2+4*b*c)^(1/2)]
[ 1/2*a+1/2*d-1/2*(a^2-2*a*d+d^2+4*b*c)^(1/2)]
```

Kommandot `syms` ”talar om” för MATLAB att `a`, `b`, `c` och `d` ska fungera som symboliska variabler. Egenvärdena beräknas då som ett formeluttryck.

Det främsta arbetsområdet för symboliska hanteringen är att ta fram och förenkla uttryck för analytiska lösningar. Men i de flesta tillämpningar är

beräkningarna så pass komplicerade att de inte går att lösa analytiskt. Man kan exempelvis lösa banorna för två kroppar i rymden analytiskt, dvs om man bara beaktar jorden och månen, så vet man exakt hur de rör sig relativt varandra. Om man tar in solen i ekvationen blir problemet däremot omöjligt att lösa analytiskt! Däremot är det enkelt att lösa kropparnas banor numeriskt. En annan nackdel är att de symboliska lösningar tar längre tid att beräkna, medan numeriska metoder ofta är snabba och effektiva.

Det här gör att symboliska hanteringen används som ett matematiskt verktyg, medan den numeriska hanteringen är ett tekniskt verktyg för stora (och realistiska) beräkningsproblem.

8.2 Grundläggande symbolisk hantering i MATLAB

För att kunna arbeta symboliskt i MATLAB, måste man tala om att en eller flera variabler eller uttryck ska vara symboliska. Detta gör man med kommandot `sym` eller `syms`.

```
>> x = sym(1/3)
x =
   1/3
```

Jämför detta med ”vanlig” MATLAB

```
>> x = 1/3
x =
  0.3333
```

I den symboliska varianten lagras talet `x` exakt i form av symboler, och i det andra exemplet utförs en avrundning.

Om man definerar flera symboliska variabler är det enklare att använda `syms`. Man kan då definiera alla på en rad

```
>> syms a b c
>> len = sqrt(a^2 + b^2 + c^2)
len =
 (a^2+b^2+c^2)^(1/2)
```

Detta var ett exempel på en variabel, `len`, som blev ett symboliskt uttryck. Ett annat exempel är

```
>> syms x;
>> f = 1/x*exp(x)*(2*sin(2*x)^2+cos(4*x)+1)
f =
  1/x*exp(x)*(2*sin(2*x)^2+cos(4*x)+1)
```

För att göra utskriften lite mer tydlig används `pretty`

```
>> pretty(len)
                                2    2    2 1/2
                               (a  + b  + c )
>> pretty(f)
                                        2
                      exp(x) (2 sin(2 x)  + cos(4 x) + 1)
                      -----------------------------------
                                      x
```

Uttryck kan förenklas med kommandot `simple`. Om ingen svarsparameter ges så visas flera möjliga förenklingar, annars returneras bara den ”bästa” förenklingen:

```
>> fsim = simple(f)
fsim =
  2/x*exp(x)
```

Ett annat sätt att förenkla uttryck är att samla eller bryta ut termer. Detta görs med kommandot `collect`

```
>> syms x y; p = sym(x^2*y + y*x - x^2 - 2*x)
p =
  x^2*y+y*x-x^2-2*x

>> collect(p,x)
ans =
  (y-1)*x^2+(y-2)*x

>> collect(p,y)
ans =
  (x^2+x)*y-x^2-2*x
```

I det första fallet skrivs uttrycket om i termer av x-potenser, i det andra i termer av y-potenser. Man bryter alltså ut x respektive y ur uttrycket.

I tabellen nedan sammanfattas de grundläggande kommandona som gåtts igenom hittills.

Grundläggande symboliska kommandon	
`sym(uttryck)`	returnerar det symboliska uttrycket `uttryck`.
`sym(x)`	skapar en symbolisk variabel `x`.
`syms x y z`	skapar flera symboliska variabler, `x`, `y` och `z`.
`pretty(utr)`	skriver ut `utr` på ett mer lättläsligt format.
`simple(utr)`	om en svarsvariabel ges så returneras en förenkling av uttrycket `utr`. Om svarsvariabel inte ges visas flera möjliga förenklingar av uttrycket.
`simplify(utr)`	samlar de olika termerna och förenklar `utr`.
`collect(utr,v)`	bryter ut variabeln `v` ur `uttryck`.
`findsym(utr)`	returnerar de symboliska variabler som finns i `utr`.

8.3 Evaluering av symboliska uttryck

Det finns kommandon så att de symboliska uttryckens värde kan beräknas i olika punkter. För att få en översikt över hela funktionen kan man rita den, vilket visas i avsnittet 8.7.
För att exempelvis ta reda på en funktions värde i punkten 1 (ett) används `subs` enligt

```
>> f= sym(1/x*exp(x)*(2*sin(2*x)^2+cos(4*x)+1));
>> subs(f,x,1)
ans =
    5.4366
```

Eller varför inte lära sig de första 100 decimalerna på π:

```
>> vpa(sym(pi),101)
ans =
3.14159265358979323846264338327950288419716939937510582097494459
23078164062862089986280348253421170680
```

Kommandot `vpa` skriver ut symboliska uttrycks värde i ett vissta angivet antal värdesiffror. Här är det uttrycket π i 101 värdesiffor som skrivs ut.

Evaluering av symboliska uttryck	
`vpa(utr,d)`	vpa står för variabel precisionsaritmetik och visar uttrycket `utr` med `d` värdesiffror. Om `d` inte ges används normalt 32 siffror.
`digits(d)`	med `digits` anges hur många värdesiffror `vpa` normalt skall visa.
`subs(utr,x,y)`	ersätter variabeln `x` med `y`. Observera att `y` kan vara både en symbolisk eller numerisk variabel.

8.4 Derivering, integration och gränsvärden

Kommandona `diff`, `int` och `limit` används för derivering, integrering respektive gränsvärdesberäkning av symboliska uttryck. I tabellen listas dessa kommandon

Derivering, integration och gränsvärden	
`diff(utr,var)`	deriverar utrycket `utr` med avseende på `var`. Om `var` inte ges används `findsym` för att ta reda på vilken variabel det skall derivaras med avseende på.
`diff(utr,var,n)`	returnerar den `n`:te derivatan av `utr`.
`int(utr,var)`	integrerar uttrycket `utr` med avseende på `var`.
`int(utr,var,a,b)`	integrerar uttrycket `utr` med avseende på `var` över intervallet $[a, b]$.
`limit(utr,var)`	returnerar gränsvärdet för `utr` då `var` går mot 0.
`limit(utr,var, a,'left')`	returnerar gränsvärdet för `utr` då `var` går mot `a` från vänster. Om `'left'` byts mot `'right'` tas gränsvärdet från höger.

Gränsvärdesberäkning

Vi använder MATLAB för att beräkna följande gränsvärde

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

som är definitionen av f'. Om vi använder $f = sin(x)$ fås

```
>> syms x h;
>> limit((sin(x+h)-sin(x))/h,h,0)
ans =
   cos(x)
```

dvs $f' = cos(x)$, vilket ju är korrekt. I kommandot specificeras vilken variabel det gäller (`h` i detta fall) och vad denna varibel ska gå mot (noll i detta fall).

Om gränsvärdet går mot ∞ används `inf` som följande exempel visar

```
>> v = sym((1+a/x)^x)
 v =
  (1+a/x)^x
>> limit(v, x, inf)
ans =
   exp(a)
```

dvs vi har i MATLAB visat att gränsvärdet

$$\lim_{x\to\infty}(1+\frac{a}{x})^x = e^a.$$

Symbolisk derivering

Ett enkelt exempel på symbolisk derivering är derivering av uttrycket x^n:

```
>> syms x; fprim = diff(x^n, x)
fprim =
   x^n*n/x
```

Även här måste man ange vilken variabel som man ska derivera med avseende på. Om uttrycket förenklas känns det lättare igen

```
 >> simplify(fprim)
ans =
   x^(n-1)*n
```

eller

```
 >> pretty(simplify(fprim))
                                    (n - 1)
                                   x        n
```

Symbolisk integrering

Vid symbolisk integrering använder man kommandot `int` ungefär på samma sätt deriveringskommandot ovan. Ett enkelt exempel är uttrycket $f(x) = x^n$ som i MATLAB blir

```
>> syms x n; f = x^n;
>> pretty(int(f,x))
                                    (n + 1)
                                   x
                                   --------
                                    n + 1
```

`pretty` används för att få lite snyggare utskrift. Samma uttryck på integrerat på intervallet [0 1] blir

```
>> pretty(int(f,x,0,1))
                                      (n + 1)
                                     x        - 1
                          lim     - ------------
                        x -> 0+        n + 1
```

Lösningen blir alltså ett formeluttryck för integralen.

Det är inte säkert att integraler kan uttryckas som ett formeluttryck, dvs analytiskt, inte ens för relativt enkla integraler. Två exempel är

$$\int_0^1 e^{-x^2} dx \quad \text{och} \quad \int_0^{\pi/2} \sqrt{1 + cos^2(x)} dx$$

Om vi provar med symbolisk lösning till den första fås

```
>> f1=exp(-x^2); int(f,x,0,1)
ans =
   1/2*erf(1)*pi^(1/2)
```

där `help erf` ger beskedet att `erf` är en felfunktion som definieras med integralen själv. MATLAB kan alltså inte hitta ett formeluttryck för denna integral, vilket beror på att det inte existerar något sådant. Om vi istället provar med numerisk lösning med exempelvis `quadl` (se avsnitt 6.1, sid 69) är det inga problem utan man får det numeriska värdet 0.7468.

8.5 Serier

Serier med symboliska uttryck är möjliga att skapa med hjälp av kommandot `symsum` och det är enkelt att ta fram Taylorutvecklingar med `taylor`.

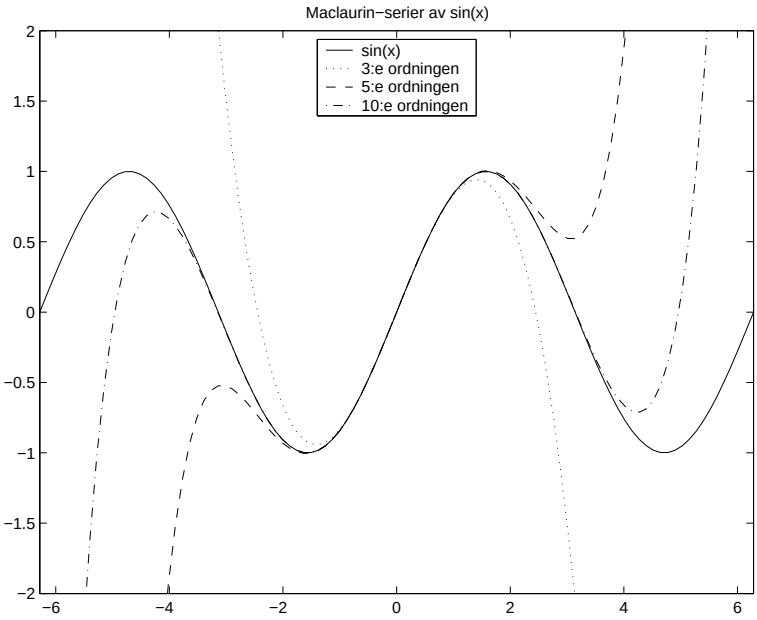
Serier	
`symsum(utr)`	returnerar summan av `utr` från 1 till n.
`symsum(utr,n, a,b)`	returnerar summan av `utr` där n går från a till b. Observera att `inf` motsvarar ∞.
`taylor(utr)`	returnerar Maclaurinserieav ordning 4 för uttrycket `utr`.
`taylor(utr,n)`	returnerar Maclaurinserie av ordning $n-1$ för uttrycket `utr`.
`taylor(utr,var, a,n)`	returnerar Taylorserie av ordning $n-1$ för `utr` med variabeln `var` kring punkten a.

MacLaurinutveckling är en Taylorutveckling kring punkten noll. Vi skall se hur MacLaurinutveckling ser ut för sinusfunktionen. Med hjälp av funktionen `taylor` tar vi fram två Maclaurinutvecklingen av femte respektive elfte ordningen.

```
>> syms x; f = sin(x);
>> f3 = taylor(f,4), f5 = taylor(f), f10 = taylor(f,11)
f3 =
x-1/6*x^3
f5 =
x-1/6*x^3+1/120*x^5
f10 =
x-1/6*x^3+1/120*x^5-1/5040*x^7+1/362880*x^9
```

För att se skillnaderna ritar vi upp serierna och sinuskurvan i 100 punkter i intervallet $[-2\pi\ 2\pi]$. Detta producerar bilden som syns i figur 29. Vi använder `subs` för att evaluera funktionerna på intervallet (se 8.3, sid 110)

```
>> xx = linspace(-2*pi,2*pi,100);
>> plot(xx,subs(f,x,xx),xx,subs(f3,x,xx),':', ...
        xx,subs(f5,x,xx),'--',xx,subs(f10,x,xx),'-.')
>> axis([-2*pi, 2*pi,-2, 2])
>> title('Maclaurin-serier av sin(x)')
>> legend('sin(x)','3:e ordningen','5:e ordningen', ...
          '10:e ordningen',0)
```



Figur 29: Maclaurinserier av $sin(x)$. Ju fler termer man har i serien, desto bättre approximation är serien till funktionen kring 0.

8.6 Ekvationslösning

Med hjälp av Symbolic toolbox kan MATLAB lösa symboliska ekvationer, ekvationssystem och differentialekvationer.

Ekvationslösning	
`solve(ekv)`	löser ekvationen `ekv` analytiskt.
`solve(e1,...,eN, var1,...,varN)`	löser ekvationssystemet ekvationerna `e1 ... eN` med avseende på de symboliska variablerna `var1 ... varN`.
`dsolve(str)`	löser differentialekvationen som uttrycks i strängen *str*. Se `help dsolve` för hur ekvationen skall skrivas.

En vanlig andragradekvation har två lösningar:

```
>> syms a b c x; solve('a*x^2 + b*x + c')
ans =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
```

Observera att med lösning menas nollställen till ekvationen, dvs x sådana att ekvationen är lika med noll. Det går också att hitta lösningar då ekvationen är lika med något annan än noll:

```
>> syms a b c x; solve('a*x^2 + b*x + c=1')
ans =
[ 1/2/a*(-b+(b^2-4*a*c+4*a)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c+4*a)^(1/2))]
```

Lösningarna till tredje- och fjärdegradsekvationer är så pass långa att de inte får vara med här, men de är lätta att ta fram. Försöker man lösa en femtegradsekvation symboliskt visar sig en av begränsningarna med symbolisk hantering, den går nämligen inte att lösa generellt.

```
>> syms a b c d e f x;
>> femte = solve(a*x^5 + b*x^4 + c*x^3 + d*x^2 + e*x + f)
??? Error using ==> solve
Error, (in allvalues/rootseq) cannot evaluate with symbolic
coefficients
Error in ==> /it/sw/matlab/6.5/toolbox/symbolic/@sym/solve.m
On line 49  ==> [varargout{1:max(1,nargout)}] = solve(S{:});
```

Däremot går det att lösa numeriskt med `fzero` (se sid 102).
För att lösa ekvationssystem anger man de olika ekvationerna i följd och för vilka variabler ekvationerna ska lösas. Att hitta lösningen till ekvationerna

$$\begin{cases} x^2 + y^2 - a^2 = 0 \\ -x^2 - y - a = 0 \end{cases}$$

där a är en parameter, dvs vi ska lösa för variablerna x och y görs på följande sätt

```
>> [X Y] = solve('x^2 + y^2 - a^2','-x^2 - y - a',x,y)
X =
[               0]
[               0]
[  (-2*a-1)^(1/2)]
[ -(-2*a-1)^(1/2)]

Y =
[  -a]
[  -a]
[ a+1]
[ a+1]
```

Lösningen ska tolkas så att de två lösningarna till den första ekvationen är $(x_1, y_1) = (x_2, y_2) = (0, -a)$ (rad ett och två i `X` resp `Y`). Lösningarna till den andra ekvationen är $(x_1, y_1) = (\sqrt{-1-2a}, a+1)$ och $(x_2, y_2) = (-\sqrt{-1-2a}, a+1)$.

8.7 Grafikkommandon

Det finns några grafikkommandon som är lämpade för att använda i symboliska sammanhang. De liknar motsvarande grafikkommandon som inte har prefixet `ez`.

Grafikkommandon för symboliska uttryck	
`ezplot(utr)`	ritar det symboliska uttrycket `utr` i ett figurfönster.
`ezplot(utr, [a b])`	ritar det symboliska uttrycket `utr` då den symboliska variabeln går från a till b.
`ezplot(x,y, [a b])`	ritar ut kurvan $x = x(t)$ och $y = y(t)$ för t i intervallet $[a\ b]$ ($[0\ 2\pi]$ om intervall inte anges).
`ezpolar(r,[a b])`	ritar den polära funktionen $r = r(\theta)$. Intervall som ovan.

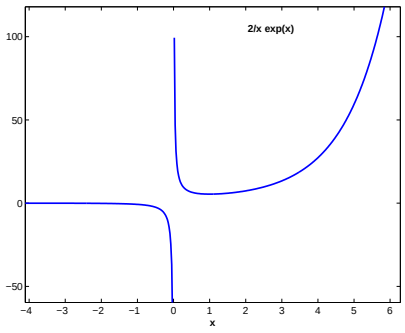
För att exempelvis rita funktion $f(x) = 2e^x/x$ används följande kommando

```
>> syms x; ezplot(2/x*exp(x))
```

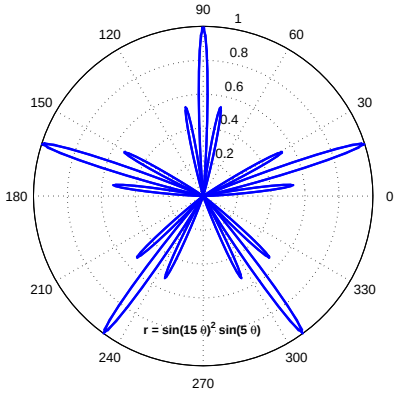
vilket resulterar i Figur 30.
Om vi vill rita funktionen $r(\theta) = \sin^2(15\theta)\cos(5(\theta - 0.1\pi))$ (polära koordinater), skriver vi följande kommandon

```
>> syms theta; ezpolar(sin(15*theta)^2*cos(5*(theta-0.1*pi)))
```

vilket ger Figur 31.
Det finns också kommandon för att rita funktioner med två variabler.



Figur 30: Funktionen $f(x) = \frac{2e^x}{x}$ ritad med kommandot `ezplot`.



Figur 31: Funktionen $r(\theta) = \sin^2(15\theta)\cos(5(\theta - 0.1\pi))$ ritad med kommandot `ezpolar`.

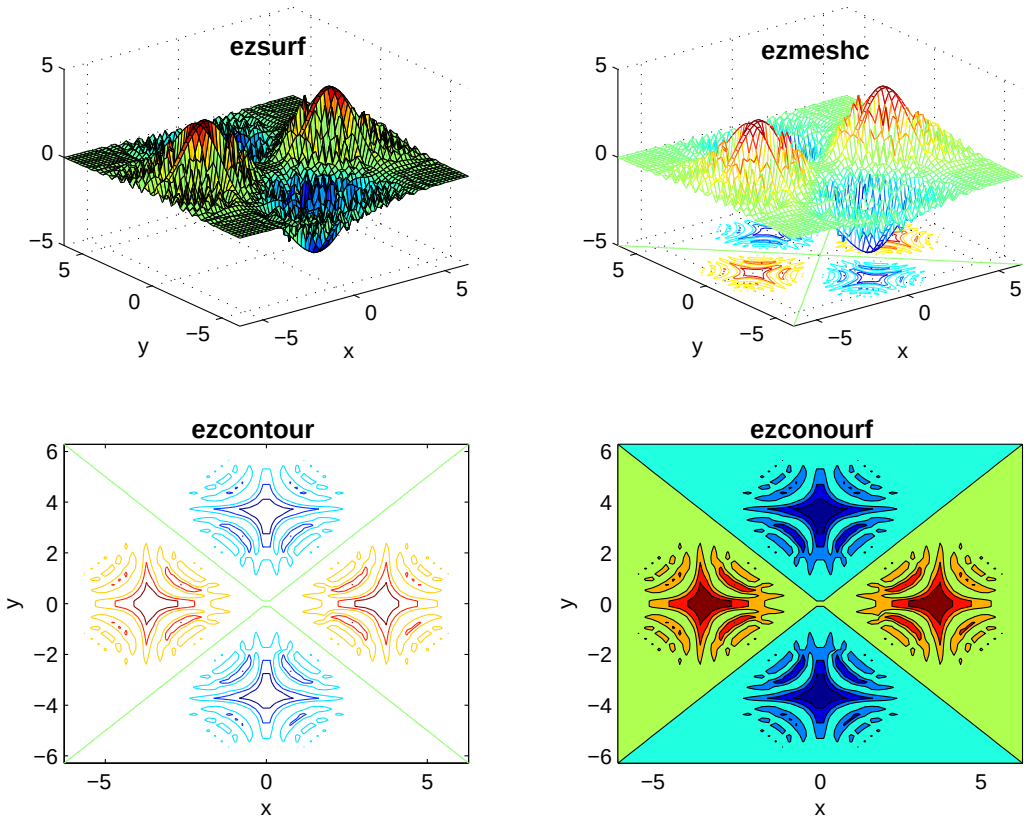
Grafikkommandon för symboliska uttryck med två variabler	
`ezplot3(x,y,z)`	som `ezplot`, men $x = x(t)$, $y = y(t)$ och $z = z(t)$.
`ezsurf(f, [a b c d])`	ritar ut ytan där $z = f(x, y)$ där a och b anger intervallet för x och c och d för y. Om bara ett intervall ges, gäller det för bägge variablerna.
`ezsurf(x,y,z)`	ritar ut ytan där $x = x(s,t)$, $y = y(s,t)$ och $z = z(s,t)$. Intervallen anges som ovan.
`ezsurfc(...)`	som `ezsurf`, men ritar även nivåkurvor i botten av ritområdet.
`ezmesh(...)`	som `ezsurf`, men ritar ett nät i stället för en yta.
`ezmeshc(...)`	som `ezsurfc`, men ritar ett nät i stället för en yta.
`ezcontour(f, [a b c d])`	ritar ut nivåkurvor för funktionen $f = f(x, y)$ intervallet ges som ovan och om det inte anges används $[-2\pi\ 2\pi]$.
`ezcontourf(...)`	som ovan, men fyller områdena mellan kurvorna.

Vi tittar på en funktion av två variabler med hjälp av fyra olika kommandon, där funktionen är

$$f(x, y) = \left(x^2 - y^2\right) \sin^2(4\,xy) e^{-0.1(x^2+y^2)}$$

Vi ritar funktione med kommandona `ezsurf`, `ezmeshc`, `ezcontour` och `ezcontourf`. Resultatet syns i Figur 32.

```
>> syms x y; f = (x^2-y^2)*sin(4*x*y)^2 * exp(-0.1*(x^2+y^2));
>> subplot(2,2,1); ezsurf(f);
>> subplot(2,2,2); ezmeshc(f);
>> subplot(2,2,3); ezcontour(f);
>> subplot(2,2,4); ezcontourf(f);
```



Figur 32: Funktionen $f(x,y) = (x^2 - y^2)\sin^2(4\,xy)e^{-0.1(x^2+y^2)}$ ritad med kommandona `ezsurf`, `ezmeshc`, `ezcontour` och `ezcontourf`

8.8 Övningsuppgifter

För lösningsförslag, se *http://www.it.uu.se/edu/bookshelf/LillaMatlab/*

8.1 Skriv in uttrycket $f = \frac{(x+1)^2}{x^2-1}$ och förenkla i MATLAB. Gör detta med och utan att använda `pretty` på resultatet.

8.2 Under lång tid användes 22/7 som närmevärde till π. Skriv ut uttrycket som decimaltal med 51 värdesiffror. Jämför med π genom att skriva ut π med lika många värdesiffror.

8.3 Beräkna följande summor

a) $\sum_{k=1}^{10} \frac{1}{k^2}$

b) $\sum_{k=1}^{\infty} \frac{1}{k^2}$

c) $\sum_{k=1}^{\infty} \frac{k^3}{3^k}$

8.4 Lös ekvationerna

a) $2 * cos(v) = \sqrt{3}$

b) $3^z = 5$

c) $4x^5 + 8x^4 + x^3 - 7x^2 - 5x - 1 = 0$

8.5 Lös ekvationssystemen

a)

$$\begin{cases} 2x + 3y = 12 \\ 3x - 2y = 0 \end{cases}$$

b)

$$\begin{cases} 3x + y + 2z = 1 \\ x + 2y + z = 0 \\ 2x - 3y - z = -1 \end{cases}$$

8.6 Beräkna $f'(x)$ då

a) $f(x) = e^x(3x^3 - 3x + 2x^2)$

b) $f(x) = sin(x)cos(x)$

c) $f(x) = \frac{sin(3x^2)+cos(x)}{1-tan(x)}$

Använd även gärna `pretty` för att få lite snyggare utskrift

8.7 Beräkna $f''(x)$ på funktionerna i föregående uppgift.

8.8 Beräkna följande integraler

a) $\int x^2 e^x dx$

b) $\int_0^2 x^2 e^x dx$

c) $\int_0^\infty e^{-t^2} dt$

d) $\int \sqrt{a^2 + x^2} dx$

e) $\int \sqrt{a^2 + x^2} da$, dvs integrera m a p a.

Använd även gärna `pretty` för att få lite snyggare utskrift

A Statistikfunktioner i MATLAB

I det här appendixet sammanfattas kortfattat MATLABs enklaste statistikkommandon: beräkning av läges- och spridningsmått, generering av slumptal och grafisk presentation.

En del statistikfunktioner finns inbyggda i grundversionen av MATLAB. MATLAB kan också utökas med s k *toolboxar* som är tillägg för olika tillämpningsområden. En sådan är *Statistics Toolbox* med mängder av avancerade statistikkommandon. Om toolboxen är installerad ges en översikt av kommandona med `help stats`.

De flesta av de kommandon som visas här kräver inte någon toolbox utan ingår i grundversionen. I de fall *Statistics Toolbox* krävs anges det.

A.1 Lägesmått och spridningsmått

Det finns färdiga funktioner för att beräkna olika läges- och spridningsmått med hjälp av MATLAB, se rutan nedan.

Grundläggande statistikfunktioner	
`mean(x)`	ger det aritmetiska medelvärdet av elementen i vektorn x.
`max(x)`	ger det största elementet i vektorn x.
`[m,i]=max(x)`	ger det största elementet m och dess position i i vektorn x.
`min(x)`	ger det minsta elementet i vektorn x.
`[m,i]=min(x)`	ger det minsta elementet m och dess position i i vektorn x.
`sort(x)`	sorterar elementen i vektorn x i stigande ordning.
`mean(X)`	ger en radmatris med det aritmetiska medelvärdet av elementen i varje kolonn av matrisen X.
`median(x)`	ger medianen av elementen i vektorn x.
`std(x)`	ger standardavvikelsen av elementen i vektorn x.
`cov(x)`	ger variansen av elementen i vektorn x.

A.2 Slumptal

Slumptal används inom många områden, och en sådan tillämpning finns i avsnitt 7.7.

Det går att generera slumptal från likformig- och normalfördelning i MATLAB med kommandona `rand` respektive `randn`. Med Statistics Toolbox kan man genera slumptal från de flesta fördelningar, t ex binomial- , χ^2- , exponential- , poisson-fördelning.

Som exempel visas hur man skapar en 3×2-matris med likformigt fördelade slumptal mellan 0 och 10:

```
>> 10*rand(3,2)
ans =
    8.9365    8.1317
    0.5789    0.0986
    3.5287    1.3889
```

Slumptal	
`rand`	ger likformigt fördelade slumptalsvärden mellan 0 och 1.
`rand(n)`	ger en $n \times n$-matris med likformigt fördelade slumptalsvärden mellan 0 och 1.
`rand(m,n)`	ger en $m \times n$-matris med likformigt fördelade slumptalsvärden mellan 0 och 1.
`randn`	ger normalfördelade slumptal med väntevärde 0 och varians 1.
`randn(n)`	ger en $n \times n$-matris med normalfördelade slumptal, väntevärde 0 och varians 1.
`randn(m,n)`	ger en $m \times n$-matris med normalfördelade slumptal, väntevärde 0 och varians 1.

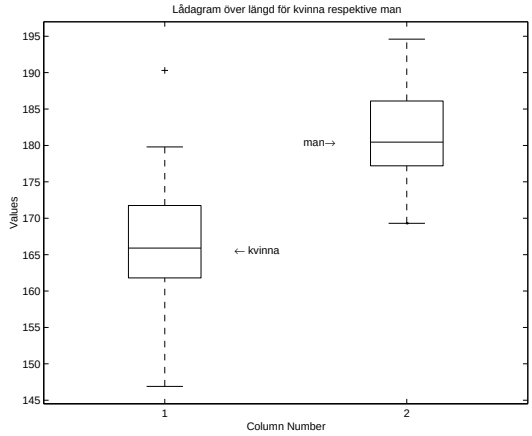
A.3 Box-dagram

Box-diagram är ett sätt att grafiskt sammanfatta minsta värde, största värde, kvartiler och median. Man får en överblick över både läge och spridning. Ett exempel på box-diagram visas nedan. Kommando för box-diagram ingår i *Statistics Toolbox*.

Antag som exempel att man mätt 100 stycken elever som går i årskurs 3 på gymnasiet och fått följande mätvärden

Kvinnor				Män			
146.9	161.8	165.9	172.0	169.3	177.2	180.5	187.1
149.4	163.3	167.3	172.3	172.2	177.4	181.0	187.2
154.4	163.3	167.8	173.4	172.7	177.4	181.4	187.2
154.7	163.5	168.0	174.1	174.5	177.5	181.5	187.6
157.3	163.7	168.3	175.0	175.0	178.0	182.2	188.2
157.8	163.9	168.7	178.3	175.3	178.0	183.1	188.2
158.4	164.2	168.9	178.4	175.4	178.2	183.2	189.2
159.3	164.5	170.0	178.7	175.9	178.6	183.3	190.1
160.2	164.7	170.1	178.9	176.1	179.0	183.5	190.8
160.3	164.9	170.3	178.9	176.3	179.4	183.7	191.0
160.4	165.3	171.1	179.5	176.5	179.7	184.3	191.2
160.9	165.5	171.2	179.8	176.7	179.8	185.1	192.0
161.8	165.9	171.5	190.3	177.2	180.4	185.1	194.6

I MATLAB låter man de olika grupperna vara varsin kolonn, `x1`, `x2`, i en matris, `X = [x1;x2]`. Ett box-diagram fås med kommandot `boxplot(X)`. Resultatet visas i Figur 33.



Figur 33: Exempel på en grafisk presentation med hjälp av ett box-diagram.

Om det finns värden som avviker extremt mycket, s k outliers, markeras dessa med `+`. Strecket dras alltså inte ut så långt.

Box-Diagram	
`boxplot(A)`	ger ett box-diagram för varje kolonn i matrisen A. Kräver *Statistics Toolbox*.

A.4 Histogram och stapeldiagram

Om man har en vektor med data kan man rita ett histogram i MATLAB med hjälp av kommandot `hist`. Man kan också använda kommandot `bar` för att göra stapeldiagram.

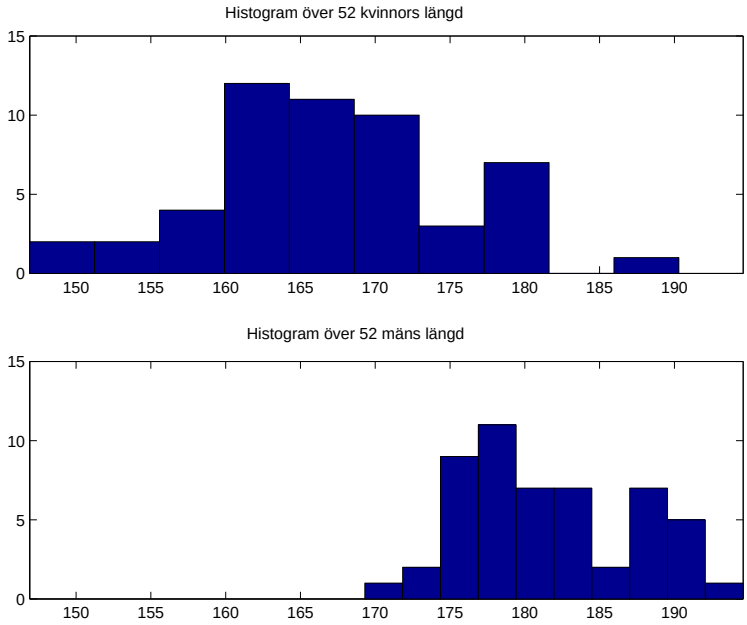
Histogram och stapeldiagram	
`hist(x)`	ritar ut ett 10-intervalls histogram för elementen i vektorn x.
`hist(x,n)`	ritar ut ett n-intervalls histogram för elementen i x.
`hist(x,y)`	ritar ut ett histogram för elementen i x med de intervall som bestäms av y. Värdena i y bör ligga i stigande ordning.
`bar(x)`	ritar ut ett stapeldiagram över värdena i vektorn x.
`bar(y,x)`	ritar ut ett stapeldiagram över värdena i vektorn x i positionerna givna av vektorb y.
`bar(y,x,l)`	ritar ut ett stapeldiagram över värdena i vektorn x i positionerna givna av vektorn y med längden l.

För att få en bild över hur elevernas längder, från föregående exempel, är fördelade kan det vara lämpligt att göra ett histogram.

```
figure(1)
clf

subplot(2,1,1)
hist(kvinna)
axis([min([kvinna; man]) max([kvinna; man]) 0 15])
gtext('Histogram över 52 kvinnors längd')

subplot(2,1,2)
hist(man)
axis([min([kvinna; man]) max([kvinna; man]) 0 15])
gtext('Histogram över 52 mäns längd')
```

Histogrammen visas i Figur 34.

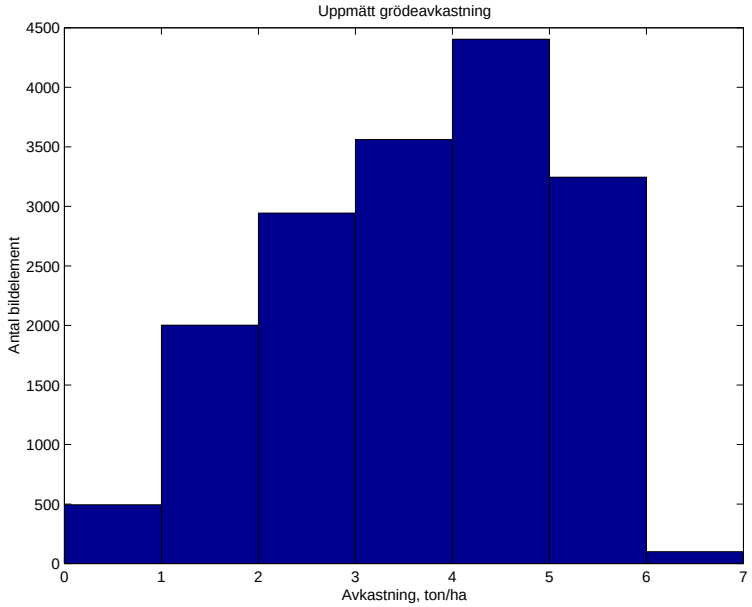
För att ge exempel på stapeldiagram används en annan undersökning. Man vill undsersöka hur mängden avkastning är korrelerat till spektral reflektans. Den spektrala reflektansen får man från en satellitbild och mängden



Figur 34: Exempel på en grafisk presentation med hjälp av ett histogram.

avkastning är uppmätt på marken. Medelvärdet och standardavvikelsen är beräknat för den spektrala reflektansen vid olika avkastning. För att sammanfatta det grafiskt gör man ett stapeldiagram. Kommandot `bar` används för detta ändamål, MATLAB-koden visas nedan och det resulterande stapeldiagrammet i Figur 35.

```
clear

y = [494 2002 2944 3562 4404 3245 100];
x = [0.5 1.5 2.5 3.5 4.5 5.5 6.5]

figure(1)
clf

bar(x,y,1)

title('Uppmätt grödeavkastning')
xlabel('Avkastning, ton/ha')
ylabel('Antal bildelement')
```



Figur 35: Exempel på en grafisk presentation med hjälp av ett stapeldiagram.

```
axis([0 7 0 4450])
```

A.5 Rita punkter med felmarginaler

Ytterligare ett kommando för grafisk presentation av data är `errorbar` som ritar ut en felmarginal kring varje varje värde.

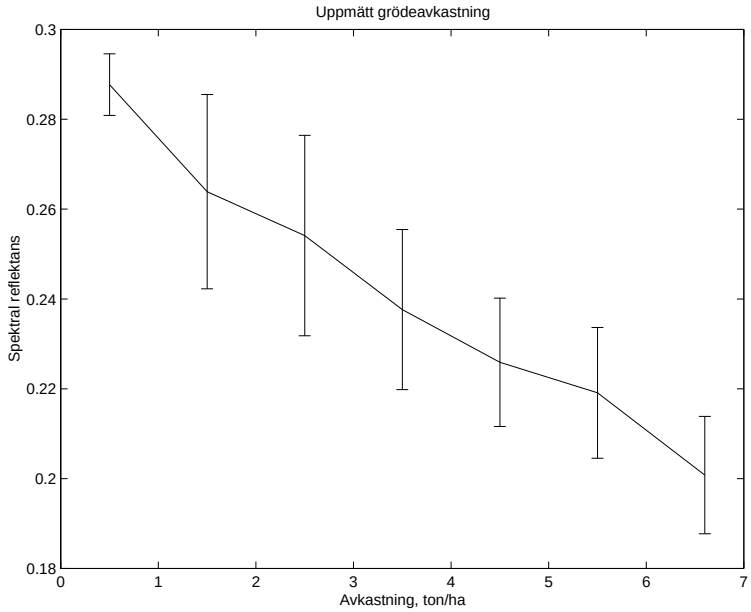
Undersökningen från föregående exempel om hur mängden avkastning är korrelerat till spektral reflektans fortsätter här för att exemplifiera användandet av `errorbar`. Nu har man bilder från en annan satellit och vill framställa resultaten på ett lite annorlunda sätt. Återigen har man beräknat medelvärdet och standardavvikelsen för den spektrala reflektansen vid olika avkastning. För att sammanfatta det grafiskt gör man denna gång en graf över medelvärdena med dess felmarginaler markerade. Det kan man göra med kommandot `errorbar`.

```
avkastning = [0.5 1.5 2.5 3.5 4.5 5.5 6.6];
spektralMedelv = [0.287703 0.263879 0.254113 ...
      0.237631 0.225904 0.219105 0.200781];
spektralStd = [0.006859 0.021629 0.022302 ...
      0.017824 0.01428 0.0145603 0.013069];
```

```
figure(1)
clf
errorbar(avkastning, spektralMedelv, spektralStd)
title('Uppmätt grödeavkastning')
xlabel('Avkastning, ton/ha')
ylabel('Spektral reflektans')
axis([0 7 0.18 0.30])
```

Resultatet ser man i Figur 36.

Errorbar	
`errorbar(x,y,e)`	ritar vektorn y mot vektorn x på samma sätt som med kommandot `plot`. Dessutom ritas felmarginalen e_i ut symmetriskt kring y_i.



Figur 36: Exempel på en grafisk presentation av felmarginaler med medelvärde och standardavvikelser.

B Lite av varje

I detta avslutande appendix är några skilda ämnen samlade: hur man sparar variabler på fil, hur man får matematiska symboler i titlar och diverse användbara kommandon.

B.1 Spara variabler på fil

Variabler kan sparas på fil mellan arbetspass. Följande exempel visar hur:

```
>> A = [1 2; 3 4];
>> x = [1; 2];
>> y = A*x;
>> who

Your variables are:

A         x         y

>> save exempelvariabler A x y
>> clear
>> who
>> load exempelvariabler
>> who

Your variables are:

A         x         y
```

De tre variablerna A, x och y sparas med kommandot `save` i filen `exempelvariabler.mat` (observera att suffixet `.mat` inte behöver anges). För att visa att kommandot fungerar raderas variablerna med `clear`, och kommandot `who` ger till svar att ingen variabel är definierad. Därefter laddas variablerna in med kommandot `load` och `who` visar att variablerna A, x och y åter finns för användning. Variablerna sparas i aktuell arbetskatalog (se sidan 31).

Variabler sparas med `save` i ett särskilt MATLAB-format. Det innebär att de enbart kan laddas in i MATLAB igen. Man kan inte ”titta på” variablerna utanför MATLAB. Man kan emellertid också spara variablerna i textformat till en fil som sedan kan öppnas i någon editor eller ordbehandlingsprogram. Detta görs genom att lägga till ’-ascii’ sist i kommandot. Det är då lämpligt

att namnge filen med suffixet `.txt` som visar att det är en textfil enligt detta exempel:

```
save exempelvariabler.txt A x y -ascii
```

Spara variabler på fil	
`save filnamn`	sparar alla definierade variabler i filen `filnamn.mat`.
`save filnamn x y`	sparar variablerna x och y i filen `filnamn.mat`.
`save filnamn -ascii`	sparar alla definierade variabler i textformat i filen `filnamn`.
`load filnamn`	hämtar variabler sparade i filen `filnamn.mat`

B.2 Matematiska formler

Det finns särskilda teckensekvenser som gör det möjligt att skriva grekiska tecken, som α, β och γ, och upphöjt till, x^2, i t ex titeln till graferna. Några av dessa tas upp nedan. Hela listan kan man hitta i *MATLAB:s hjälp-fönster*, se sidan 7.

Matematiska formler					
Grekiska tecken (små)		Grekiska tecken (stora)		Sub-/superscript	
`\alpha`	α			`x_1`	x_1
`\beta`	β			`x_{11}`	x_{11}
`\gamma`	γ	`\Gamma`	Γ	`x^2`	x^2
`\delta`	δ	`\Delta`	Δ	`x^{22}`	x^{22}
`\pi`	π	`\Pi`	Π		
⋮	⋮	⋮	⋮	Pilar	
				`\rightarrow`	$\rightarrow$
Olikheter		`\infty`	∞	`\Rightarrow`	$\Rightarrow$
`\leq`	$\leq$			`\leftarrow`	$\leftarrow$
`\geq`	$\geq$			`\Leftarrow`	$\Leftarrow$

Sakregister