Introduktion till Matlab och Maple

Anders Holtsberg

Matematikcentrum
Lunds Tekniska Högskola

Version 1.21
20 augusti 1999

Innehåll

Figurer

Tabeller

Förord

Bakgrund

Den här handledningen är en kortfattad introduktion till programmet Matlab. Matlab är både en interaktiv matematikmiljö och ett programmeringsspråk, som idag används vid i stort sett alla tekniska högskolor världen över, och är stort även i industrin.

Ett litet smakprov på symbolhanteringsprogrammet Maple finns också med, men vad som sägs i fortsättningen i detta förord gäller Matlab, och enbart Matlab version 5, inte tidigare versioner.

Innehållet är tänkt att användas under datorstugan för nya F-teknologer och under det första studieåret. Häftet rekommenderas som lämplig brevidläsning till kursen i linjär algebra så att studenten kan befästa den matematiska lärdomen om matrisalgebra med hjälp av egna experiment och kontrollräkningar.

Tillsammans med träningen i grundläggande programmering (som i Lund sker med programspråket Java) så kan man med häftet som grund börja programmera sina egna matlabprogram.

Notera alltså att avsikten inte är att man ska gå igenom allt på datorstugan. Det finns mycket i det som inte kan förstås utan kunskaper från kurserna linjär algebra och programmering. Avsikten är att läsaren skall behålla häftet och läsa vid behov genom hela utbildningen.

Jag har tagit formatering och formuleringar från Per Forebys handledning till operativsystemet Unix. Vad gäller innehållet har jag lånat vissa smådelar från Bo Bernhardssons datorstugepapper från förra året och också använt följande källor.

- *Matlab Primer* av Kermit Sigmond finns gratis att hämta på Internet, se
  `www.yale.edu/secf/software/matlab.html`
  Den är 35 sidor lång och rätt kompakt skriven – den förutsätter att man kan linjär algebra, och det kan läsaren förhoppningsvis vid juletid.

- *An introductory course in Matlab* av Fredrik Gustafsson. Det häftet är kursmaterial i Linköping och finns att hämta på Internet, se
  `www.control.isy.liu.se/~fredrik/`

- *Using Matlab* från The Mathworks.

Jag har skrivit häftet såsom jag själv skulle ha velat ha det när jag började på tekniska högskolan en gång i tiden. Jag har hoppat över långrandiga uppräkningar av en massa funktioner och likaså hoppat över de mer avancerade matematiska eller fysikaliska modeller som brukar förekomma i andra handledningar på universitetsnivå. Å andra sidan är häftet betydligt mer än ett smakprov, det är faktiskt ett komplett studiematerial för grundläggande matlabprogrammering.

Texten om Maple är däremot bara ett smakprov. Orsaken till prioriteringen är tvåfaldig. För det första är kunskaper i Matlab mycket väsentliga för den högre utbildningen på F och E-sektionerna, möjligen också på de flesta övriga sektioner. Det är inte kunskaper i Maple. För det andra kan jag inte Maple själv.

Häftet skrevs under sommaren 1999 när jag var introduktionsansvarig för F-sektionen.

Matlab på egen dator

Matlab och Maple finns för många operativsystem men kostar en hel del pengar.

Universitetet har ett avtal som gör att studenter får använda programmen gratis hemma så länge man är inskriven vid LTH. En CD-skiva kan lånas på UB2. Det kostar 50 kronor per tillfälle att låna en skiva (om skivan kommer bort kan det bli *mycket* dyrt, så var aktsam). På skivan finns versioner av Matlab och Maple för operativsystemen Linux, MacOS och MS Windows. Även Acrobat reader finns med så att man skall kunna läsa medföljande manualer på pdf-format om man vill.

Mer om Matlab

Det finns en nyhetsgrupp om matlab som heter `comp.soft-sys.matlab`. Tillverkaren av Matlab – The Mathworks – finns på Internet med adress `www.mathworks.com`. Matlab är väldigt dominerande i sin nisch – matrisberäkningsprogram – men det finns andra med olika inriktning och grad av matlabkompatibilitet, t ex Mideva, Matrix-X, Rlab, Scilab, Splus och Gauss. I många Linux-distributioner finns Octave som är fri programvara och relativt matlabkompatibel.

Handledningens uppläggning

Studiematerial

Avsnitten 1.1 – 1.6 är avsedda för två övningspass på datastugan. Avsnitt 1.7 behandlar även saker som man omöjligt kan förstå den första veckan men är rekommenderad bredvidläsning till kursen i linjär algebra. Den som vill kan hoppa över hela detta avsnitt på datorstugan.

Avsnitten 1.8 och 1.9 kan läsas som orientering under datorstugan (utan krav på att förstå allt innehåll; tillämpliga delar av javakursen ger klarhet under hösten).

Avsnitt 1.10 är avsett att betona skillnaden i filosofi mellan matlabprogrammering och annan programmering och innehåller tips även för matlabexperten. Detta sagt för att återigen betona att häftet som helhet är avsett att ha som referens och under hela utbildningen. Avsnitt 1.11 hoppas lämpligen över helt eftersom det likaså innehåller begrepp som är tämligen avancerade.

Simulerad text

När handledningen visar exempel på in- och utmatning i terminalfönstret ser det ut så här:

```
>>3+4
ans =
     7
```

Den text som användaren själv skriver in markeras med fetstil. Övrig text är sådant som datorn skriver till skärmen.

Övningsexempel

I texten finns det en del övningsuppgifter. I appendix A finns lösningar till uppgifterna. Det finns dock inte övningar på allting som handledningen omfattar, så läsaren uppmanas att själv prova kommandon allt eftersom de introduceras i texten.

De övningar som finns insprängda i texten ser ut så här:

Övning 0.1
Hur beräknar man i Matlab snabbt summan av logaritmen på alla udda tvåsiffriga tal?

För de övningar där det är tillämpligt finns svar. I appendix A finns svaret till ovanstående exempel.

Om handledningen

Denna handledning är producerad med typsättningsprogrammet LaTeX, och utskriven på en vanlig laserskrivare. De teckensnitt som har använts är times, avant garde och courier. Källkoden till hela handledningen finns på nätadressen
`www.maths.lth.se/matstat/staff/andersh/`

Kommentarer är välkomna

Handledningen är alldeles nyskriven och innehåller säkerligen feltryck och möjligen förbiseenden och luckor och andra ojämnheter.

Jag tar tacksamt emot förslag på förändringar, helst via datorpost till `andersh@maths.lth.se`.

Kapitel 1

Matlab

1.1 Starta

På datorer som kör MS Windows startas Matlab i startmenyn under ”Programs”→”Matlab for Windows”.

På Unixdatorer startas Matlab genom att skriva `matlab` i ett terminalfönster. Ett annat sätt att starta är att använda en meny i rotfönstret. Det är olika på olika maskiner, under ”utilities” eller ”mathematics” med vänster eller höger musknapp.

Man avslutar Matlab genom att skriva kommandot `quit` eller `exit`.

1.2 Kom igång

Till att börja med kan man tänka på Matlab som en avancerad räknedosa som beräknar uttryck. Man skriver in vad man vill ha gjort och Matlab svarar:

```
>> 3*11.5+2.3^2/4
ans =
   35.8225
```

Decimalpunkt används, inte komma.

De vanliga funktionerna som finns på varje räknedosa av klass finns så klart i Matlab också, t ex

```
sqrt exp log sin asin cos acos tan atan
```

Observera att `log` är den naturliga logaritmen, inte logaritmen med 10 som bas.

Variabler tilldelas värden med tecknet = och finns sedan kvar i minnet:

```
>> a = 1;
>> A = sqrt(36);
>> width = 3.89;
>> who
Your variables are:
A         a         ans       width
```

Kommandot `who` visar alltså vilka variabler som finns i minnet. Variabeln som heter `ans` tilldelas resultatet av en programrad om man inte har tilldelat någon annan variabel det. Vissa variabelnamn är speciella, t ex `pi` som har värdet π. Vi ser också i exemplet att Matlab betraktar `a` och `A` som olika variabler, det är alltså skillnad på små och stora bokstäver.

En variabel, t ex `a`, kan raderas ur minnet med `clear a`. Om man ger kommandot `clear` utan argument så raderas alla variabler ur minnet.

Läggs ett semikolon, `;`, till efter en kommandorad som ovan i exemplet, så skrivs resultatet inte ut på skärmen.

Matlab använder så kallad *dynamisk typkontroll*, det vill säga variabler behöver inte deklareras i förväg och de kan ändra typ under programkörningen. Kontrollen av att man gör något vettigt med dem sker i exekveringsögonblicket. I språk med så kallad *statisk typkontroll*, t ex Java, så måste man deklarera i förväg vad en variabel skall innehålla för sorts data.

Övning 1.1
Beräkna $e^2 / \sin \frac{\pi}{9}$?

Om man skriver fel kan man ta tillbaka tidigare kommandorader med piltangenterna. Man kan också editera kommandoraden ungefär som i Emacs. Gör nästa övning genom att editera föregående kommando. Pröva `CONTROL-A`, `CONTROL-E`, `CONTROL-L` och `CONTROL-R` förutom piltangenterna under editeringen och se vad som händer.

Övning 1.2
Beräkna $e^3 / \sin \frac{\pi}{9}$?

1.3 Hjälp

Använd hjälpkommandot `help` närhelst du befinner dig i nöd, det är mycket användbart!

Övning 1.3
Hur fungerar arctangensfunktionen (`help atan`)? Kan man anropa `atan` med två argument för att få vinklar runt hela varvet eller finns det någon annan funktion för det?

En mycket användbar funktion är `lookfor` som letar efter en teckensträng i första raden av alla hjälptexter som finns.

Övning 1.4
Finns det någon funktion eller operator som beräknar resten vid heltalsdivition? Ledning: Rest heter *remainder* på engelska.

Om programmet tar lång tid eller verkar ha fastnat så kan man alltid avbryta med `CONTROL-C`.

Det är viktigt att våga experimentera sig fram och att kunna leta efter information med hjälpkommandot. Det är så de flesta lär sig att använda Matlab.

En mindre användbar men desto lättsammare funktion att rådfråga är `why`, utan argument.

1.4 Matriser

Matris

Cleve Moler, matematikern som grundade The Mathworks och skrev programmet Matlab, sa en gång ”Matlab is a highly typed language. It has one type. The matrix!” Det är inte sant numera eftersom Matlab har utvecklats väldigt sedan den första versionen började säljas för pengar 1984, men Matlab är en sammandragning av orden ”Matrix laboratory” och grunden är just matrisberäkningar.

I det här avsnittet skall vi bara tala om vad en matris är och hur man kan indexera den i Matlab. Matrisoperationer finns i avsnitt 1.7.

En matris är en ruta med tal. Man använder hakparenteser i Matlab för att bilda matriser. Vi bildar en 2×3-matris `x`, alltså en matris med 2 rader och 3 kolonner. För att skilja tal på samma rad används mellanslag eller kommatecken. För att ange ny rad används returtecken eller semikolon. Ett sätt att skriva in en matris är följande.

```
>> x = [8 3 4; 6 9 17]
x =
     8     3     4
     6     9    17
>> size(x)
ans =
     2     3
```

Man kan använda hakparenteser för att bygga matriser av matriser. I följande exempel utnyttjas funktionerna `zeros`, `ones` och `eye` som ger matris av nollor, matris av ettor och enhetsmatris, för att bygga en matris `x`.

```
>> z = [eye(2) zeros(2,3)
ones(2,3) [3 5; 2 7]]
z =
     1     0     0     0     0
     0     1     0     0     0
     1     1     1     3     5
     1     1     1     2     7
```

En *radvektor* är en rad med tal, d v s en matris med bara en rad. En *kolonnvektor* är en kolonn med tal, d v s en matris med bara en kolonn. En vektor är en radvektor eller en kolonnvektor.

Aprostrof `'` används som transponatoperator, den gör om en matris så att rad 1 blir kolonn 1 i stället, rad 2 blir kolonn 2 och så vidare. En radvektor görs då om till en kolonnvektor eller vice versa.

Notationen `a:b:c` betyder en radvektor från `a` till `c` i steg om `b`. I notationen `a:c` förutsätts steget vara 1.

```
>> 5:9
ans =
     5     6     7     8     9
>> y = (5:0.5:6)'
y =
    5.0000
    5.5000
    6.0000
```

Man kan plocka ut ett element ur en matris genom att ange rad och kolonn inom vanliga parenteser. Man kan ta flera rader eller kolonner också genom att indexera med en vektor. Man kan plocka ut hela rader eller kolonner genom ett ensamt kolontecken.

Övning 1.5
Skriv in en 2×4-matris `z`. Pröva följande operationer. Fundera över resultaten tills du förstår dem.
`z(2,1)  z(1,[1 3])  z(2,2:4)  z(:,1)  z(2,:)`

Det fungerar också att ändra en del element i matrisen genom att indexera i tilldelningssatser:

```
>> x(1,3) = 0
x =
     8     3     0
     6     9    17
>> x(1,[1 2]) = [99 98]
x =
    99    98     0
     6     9    17
```

Övning 1.6
Bilda en 7×5-matris med namn `z` som bara innehåller nollor (använd funktionen `zeros`). Sätt därefter det mittersta elementet och dess 8 närmaste grannar till 1.

Teckensträngar

Teckensträngar anges inom apostroftecken, alltså `'`, och inte inom citattecken `"` som i andra programspråk:

```
>> s = 'kålrot';
```

Variabeln `s` är nu i själva verket en radvektor bestående av sex tal, koderna för `'k'`, `'å'` o s v, men variabeln vet själv att den skall skrivas ut som bokstäver. För att se koderna kan man bara addera 0.

```
>> s
s =
kålrot
>> s+0
ans =
   107   229   108   114   111   116
```

Eftersom strängar bara är vanliga radvektorer så kan man bygga längre strängar med hakparenteser precis som man kan göra med numeriska vektorer:

```
>> t = ['En god ', s, 'ssoppa']
t =
En god kålrotssoppa
```

Logisk indexering

I Matlab kodas falskt som 0 och sant som 1. Antag att vi vill simulera 4 stycken slumptal mellan noll och ett och sedan plocka ut de som blev mindre än 0.4. Det verkar vettigt att skriva följande.

```
>> x = rand(1,4)
x =
    0.7975    0.6220    0.1516    0.1031
>> x(x<0.4)
ans =
    0.1516    0.1031
```

Till vår stora förvåning fungerade det. Men betrakta följande.

```
>> k = x<0.4
k =
     0     0     1     1
```

Resultatet är en vektor av samma längd som `x`. Men den innehåller ettor och nollor, och noll är inte något nummer på ett

element i x, som ju indexeras från 1 till 4. Anledningen till att det fungerar ändå är att det finns en dold flagga som gör att vektorn minns att den är resultatet från en logisk operation (de logiska operatorerna är ==, ~=, <, >, <= och >=). Funktionen `islogical` avslöjar detta:

```
>> islogical(x)
ans =
     0
>> islogical(k)
ans =
     1
```

Ett annat exempel är om vi vill ta bort alla element i en vektor v som är negativa. Då kan vi indexera med en logisk vektor på vänstersidan i en tilldelningssats och ha en tom matris på högersidan:

```
>> v(v<0) = [];
```

Och, eller och inte.

Och-operatorn heter & och eller-operatorn heter |. De fungerar elementvis på vektorer och matriser.

En märklig sak är att de två operatorerna har haft samma prioritet i Matlab, till skillnad från alla andra programmeringsspråk. Uppenbarligen kommer detta att ändras i framtiden. Matlab varnar i version 5 med en lång varningstext om man skriver `a|b&c`, som Matlab hittills tolkar från vänster till höger som `(a|b)&c`, men som borde tolkas `a|(b&c)`, som är brukligt i alla andra programmeringsspråk och i matematisk notation.

Icke-operatorn heter ~. Uttrycket `~x` ger ett för varje element i x som är noll, annars ger uttrycket noll. Exempelvis är uttrycket `~~x` samma som uttrycket `x~=0`.

Se även avsnitt 1.8 om `all` och `any`.

Komplexa tal

Komplexa tal erbjuder inga problem. Följande exempel illustrerar komplexa tal.

```
>> i = sqrt(-1)
i =
        0 + 1.0000i
>> z = 5 + 7*i
z =
   5.0000 + 7.0000i
>> real(z)
ans =
     5
>> imag(z)
ans =
     7
>> conj(z)
ans =
   5.0000 - 7.0000i
>> q = z*conj(z)
q =
    74
```

1.5 Grafik

Matlab innehåller numera ett mycket avancerat grafiksystem, men här skall vi bara gå igenom det allra enklaste.

Enkel grafik

Figur 1.1 är skapad på följande sätt. Först ritas den heldragna kurvan:

```
>> x = 0:0.1:7;
>> f = sin(x);
>> plot(x,f)
```

Det som hände var att vektorn x skapades som en vektor från 0 till 7 i steg om 0.1. Man kan också använda funktionen `linspace` för samma ändamål. När man skall plotta en kurva så gäller det att ta lagom många punkter, inte för få så att det blir hackigt, och inte för många så att det tar för mycket minnesutrymme och tid. I vektorn `f` lagras sedan sinus av motsvarande element i x. Funktionen `plot` ritar en kurva.

I nästa steg ritas den streckade kurvan:

```
>> g = sin(sqrt(x));
>> hold on
>> plot(x,g,'--')
```

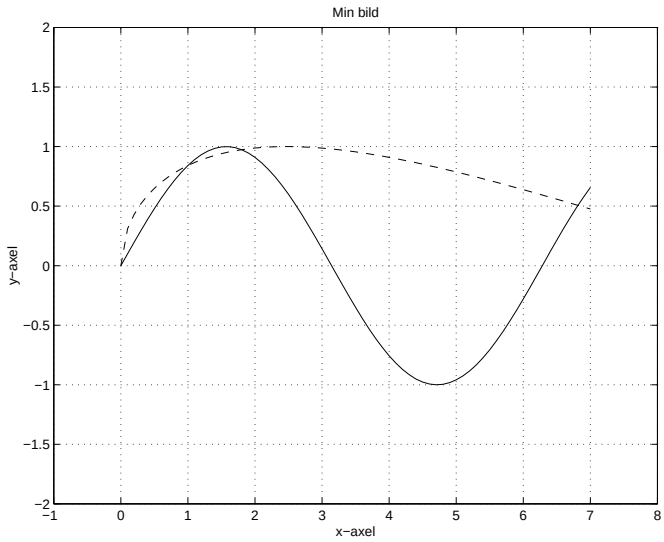
Funktionen `hold` används för att inte den heldragna kurvan skall försvinna när vi ritar den streckade. Funktionen `plot` ges ett tredje argument, teckensträngen `'-'`, som talar om hur kurvan skall ritas, här streckad. Andra möjligheter är att rita med olika färger och olika plotsymboler, t ex optionen `'o'` för små ringar, i stället för en kurva (se hjälpfunktionen till `plot`).

```
>> axis([-1 8 -2 2])
>> grid
```

Axlarna justerades, med `axis`, och hjälplinjer lades till, med `grid`, för att få ett estetiskt utseende.

```
>> xlabel('x-axel')
>> ylabel('y-axel')
>> title('Min bild')
```

Man skall ha texter på axlarna och figuren skall ha en titel, det ordnades med `xlabel`, `ylabel` och `titel`. Nu är bilden klar.



Figur 1.1: Två kurvor

Peka och zooma

Man kan peka med musen i en bild och få koordinaterna som svar, använd `ginput`. Man kan också lägga till text på något lämpligt ställe i bilden, använd `gtext`.

Övning 1.7
Rita kurvan $\sin(x)$ med följande kommandon.

```
x = 0:0.01:pi;
y = sin(x);
plot(x,y)
```

Klicka någonstans i bilden efter att du skrivit följande kommando.

```
p = ginput(1)
```

Du fick x- och y-koordinaten för punkten du pekade på. Lägg nu till text genom att först skriva kommandot

```
gtext('Citronil och Pommac')
```

och sedan klicka någonstans i bilden.

Man kan zooma i en figur genom kommandot `zoom`.

Övning 1.8
Ge kommandot `zoom`. Klicka därefter med musen och håll ner musknappen och dra den en bit och släpp knappen sedan. Se till att få med något intressant i rutan som syns. Pröva att zooma ut med högerknappen också.

Subplot

Det är lätt att plotta flera grafer i figurfönstret. Man använder funktionen `subplot(m,n,p)` som delar in figurfönstret i `m`×`n` grafer och sätter aktuell graf till nummer `p`. Man kan strunta i kommatecknet också och skriva t ex `subplot(313)` om man vill dela in figurfönstret i tre avlånga grafer och rita i den nedersta.

Övning 1.9
Pröva följande kommandon och försök förstå vad som händer.

```
>> x = 0:50;
>> y = sqrt(x);
>> z = rand(1,51);
>> subplot(211)
>> plot(x,y)
>> subplot(223)
>> plot(x,z,'o')
>> subplot(224)
>> plot(x,y+z,'.-r')
>> grid
```

Man kan ha flera figurfönster igång samtidigt också, se hjälptexten till `figure`.

Papperskopior

Det finns två sätt att skriva ut en figur direkt till skrivaren. Det ena är kommandot `print` och det andra är att klicka med musen på menyn "File" i ovankanten av figurfönstret, och sedan klicka på "Print".

Man kan också spara figuren som en fil, bland annat på formaten postscript, jpeg och tiff om man vill infoga dem i textdokument eller lägga dem på sin hemsida på Internet eller något annat roligt. Se hjälptexten till funktionen `print` för exakt information om hur man gör.

Om man använder MS Windows kan man (om man har tur) också använda klippytan för att skicka bilderna direkt till ordbehandlingsprogram. Enligt uppgift fungerar kommandot `print -zbuffer -dmeta` för att skicka till klippytan i Windows. "The -zbuffer [flag] is necessary with Word 2000, but not necessary with Word-97" enligt en användare på nätet.

1.6 Spara och läsa in

Man sparar alla sina variabler med kommandot `save`. Då skapas en fil som heter `matlab.mat` (om det finns en tidigare sparad fil med samma namn så skrivs den över). För att få tillbaka variablerna använder man `load`. Man kan också ange ett explicit filnamn, t ex `save minfil3` Då lägger Matlab automatiskt till fyra tecken så att filens namn blir `minfil3.mat` Om man bara vill spara vissa variabler i filen så anger man dem efter filnamnet, exempelvis `save minfil5 a x`

Aktuell katalog

Var hamnar filen i filträdet? Jo, i Matlabs aktuella katalog. Med kommandot `cd` med ett argument byter man aktuell katalog. Om man använder `cd` utan argument får man reda på aktuell katalog (observera här skillnaden mot Unix-kommandot `cd`). Kommandot `pwd` ger också aktuell katalog. Med kommandot `dir` eller `ls` kan man se vilka filer som finns. Tre av dessa kommandon heter likadant som i Unix och det fjärde heter likadant som i Msdos, vilket gör att man lättare kommer ihåg dem (På Unix-maskiner skiljer sig `dir` och `ls`, prova båda). Det finns till och med ett kommando `mkdir` som gör samma sak som i Unix, det vill säga skapar en ny katalog, även under MS Windows.

Övning 1.10
Ta reda på vad aktuell katalog heter. Skapa en underkatalog med namnet `matlab` i din hemkatalog. Gör så att den nyss skapade katalogen blir aktuell katalog. Titta efter vilka variabler som finns i arbetsminnet. Spara variablerna. Titta efter att en fil har skapats. Radera alla variabler och titta efter att arbetsminnet är tomt. Ladda tillbaka variablerna. Titta efter att de finns igen.

Man kan också spara en matris med läsbara siffror. Det gör man genom att ange filnamn, variabelnamn och optionen `-ascii` när man sparar. Betrakta följande exempel.

```
>> a = [1 2; 3 4]
a =
     1     2
     3     4
>> save afil.txt a -ascii
>> type afil.txt
   1.0000000e+00   2.0000000e+00
   3.0000000e+00   4.0000000e+00
>> clear all
>> load afil.txt
>> who
Your variables are:
a         afil
>> afil
afil =
     1     2
     3     4
```

Matlabkommandot `type` skriver ut en fil i kommandofönstret (det motsvarar alltså unixkommandona `cat` och `more`). För att inte långa texter bara skall svischa förbi kan man använda kommandot `more on`.

Notera att variabelnamnet efter inladdning av filen blev samma som filnamnet (med undantag av `.txt`).

På det viset kan man enkelt exportera och importera siffror till och från egna program, skrivna i till exempel Java.

Om man skall mata in många siffror för hand gör man det bäst i en text-fil med sin favoriteditor för att sedan läsa in den i Matlab på det beskrivna sättet.

1.7 Matrisoperationer

Det finns tre matlaboperatorer som betecknas `*`, `/` och `^` Operatorerna är så kallade matrisoperatorer. Det finns tre andra matlaboperatorer som har en punkt först och som kallas elementvisa operatorer. De är `.*` `./` och `.^`

Multiplikation

Vi skapar två matriser för att illustrera skillnaden mellan matrismultiplikation och elementvis multiplikation.

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

Matrismultiplikation är inte symmetrisk, `A*B` är inte samma sak som `B*A`:

```
>> A*B
ans =
    49    29
    62    40
>> B*A
ans =
    11    18    46
     8    18    40
    12    27    60
```

Om man vill multiplicera varje element i matrisen `A` med motsvarande element i matrisen `B'` använder man operatorn `.*` så här:

```
>> A.*B'
ans =
     1    12    36
    10     6    24
```

För exponentoperatorn `^` gäller samma sak. Notationen `A^2` betyder samma sak som `A*A`, medan `A.^2` betyder att varje element i `A` skall kvadreras.

Det finns också en operator `.'` som behövs när man arbetar med komplexa matriser. Operatorn `'` betyder Hermitskt transponat, d v s transponat av komplexkonjugatet, medan operatorn `.'` betyder transponat. Men för reella matriser så är det ju ingen skillnad.

Invers och division

Låt oss illustrera de olika divisionsoperatorerna med de tre matriserna A, B och C:

```
>> A = [1 2; 5 3]
A =
     1     2
     5     3
>> B = [5 2;3 1]
B =
     5     2
     3     1
>> C = A*B
C =
    11     4
    34    13
```

Invers heter i Matlab `inv` och enhetsmatrisen heter `eye`. Om man vet att relationen $AB = C$ gäller och man känner C och B men inte A så kan man multiplicera med B^{-1} från höger för att få fram $A = CB^{-1}$, eller i matlabformulering:

```
>> C*inv(B)
ans =
    1.0000    2.0000
    5.0000    3.0000
```

Man kan också göra på följande sätt, vilket är bättre av numeriska skäl. Minnesregeln är att om $AB = C$ så borde ju $A = C/B$. Så får man inte skriva i matematisk text men i Matlab bör man av beräkningstekniska skäl skriva precis så:

```
>> C/B
ans =
    1.0000    2.0000
    5.0000    3.0000
```

Om vi hade menat att varje element i C skulle divideras med motsvarande element i B så hade vi använt den elementvisa divisionsoperatorn `./` i beräkningen:

```
>> C./B
ans =
    2.2000    2.0000
   11.3333   13.0000
```

Om C och A vore kända men B okänd så kan vi multiplicera relationen $AB = C$ med A^{-1} från vänster och få fram att $B = A^{-1}C$.

```
>> inv(A)*C
ans =
    5.0000    2.0000
    3.0000    1.0000
```

Men ännu hellre gör man som nyss. Eftersom matrismultiplikation inte är symmetrisk så behövs en ny operator \ för vänsterdivision. Minnesregeln är att om $AB = C$ så borde ju $B = A\backslash C$, som tolkas som C dividerat med A från vänster:

```
>> A\C
ans =
    5.0000    2.0000
    3.0000    1.0000
```

Att siffrorna presenteras med ett antal nollor beror på att avrundning under beräkningarna har gjort att den sista signifikanta siffran inte är noll. I Matlab görs alla beräkningar med dubbel precision, vilket betyder att varje tal tar åtta byte av minnet och att antalet signifikanta siffror ungefär är femton. (För bildbehandling har man ofta behov av andra format, se `help datatypes`.)

MK-metoden

Antag att vi har ett ekvationsystem $AX = B$ där det finns fler rader än kolonner i matrisen A (och ingen kolonn i A kan

skrivas som en linjärkombination av de övriga). Då saknar ekvationssystemet i allmänhet lösning (om inte vektorn B är så speciell att ekvationssystemet ändå går att lösa).

Istället vill man ofta ha minstakvadratlösningen, det X som minimerar avståndet mellan B och AX (med avstånd menas här normen $||B - AX||$). Lösningen ges av de så kallade normalekvationerna, vilka i matrisformulering fås genom att multiplicera på A^T från vänster, $A^T AX = A^T B$. Lösningen blir $X = (A^T A)^{-1} A^T B$. I Matlab skulle koden för detta bli kort, men man skall inte skriva så. Matlabs matrisdivisionsoperator levererar i stället minstakvadratlösningen direkt på ett numeriskt bättre sätt:

```
>> A = [1 2; 3 4; 5 0]
A =
     1     2
     3     4
     5     0
>> B = [0 1 2]'
B =
     0
     1
     2
>> X = A\B
X =
    0.4048
   -0.0833
>> Bhatt = A*X
Bhatt =
    0.2381
    0.8810
    2.0238
```

Observera att det var till vänster A stod så det var vänsterdivision \ som användes.

Notationen i Matlab att använda matrisdivisionsoperatorerna får både vanlig lösning och minstakvadratlösning är helt konsekvent eftersom den vanliga lösningen är ett specialfall av minstakvadratlösningen, när matrisen A är kvadratisk med full rang.

Diverse

Skalärprodukten $u^T v$ mellan två kolonnvektorer, eller inre produkten som den ibland kallas, skrivs precis som väntat i Matlab, likaså yttre produkten uv^T. För normen $||u|| = \sqrt{u^T u}$ finns funktionen `norm`.

```
>> u = [4 3 7]'
u =
     4
     3
     7
>> v = [5 2 1]'
v =
     5
     2
     1
>> u'*v
ans =
    33
>> u*v'
ans =
    20     8     4
    15     6     3
    35    14     7
>> norm(u)
ans =
    8.6023
```

Determinanten beräknas med `det` och rangen av en matris med `rank`. För egenvärdena finns funktionen `eig` och funktionen `diag` tar ut diagonalelementen ur en matris. Se i följande exempel hur man kan få ut fler än ett utargument från en funktion i Matlab:

```
>> x = rand(3)
x =
    0.8462    0.6721    0.6813
    0.5252    0.8381    0.3795
    0.2026    0.0196    0.8318
>> [v, e] = eig(x)
v =
    0.7510    0.8135   -0.3483
    0.6246   -0.5268   -0.6320
    0.2142   -0.2464    0.6922
e =
    1.5996         0         0
         0    0.2046         0
         0         0    0.7119
>> v*e*inv(v)
ans =
    0.8462    0.6721    0.6813
    0.5252    0.8381    0.3795
    0.2026    0.0196    0.8318
>> diag(e)
ans =
    1.5996
    0.2046
    0.7119
```

Ordningen mellan egenvärdena blev inte i storleksordning, tyvärr, och vi hade tur som fick reella egenvärden. De hade kunnat bli komplexa eftersom matrisen med slumptal inte är symmetrisk.

1.8 Slingor

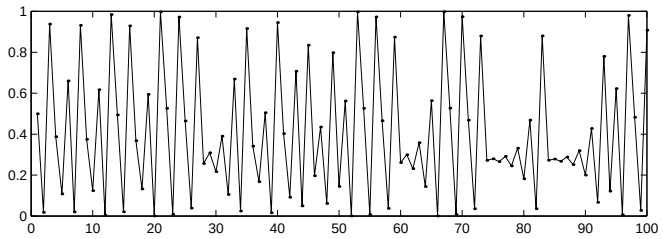
De reserverade orden `if`, `else`, `while`, `for`, `end`, `break`, `switch`, `case` och `otherwise` styr programflödet. Följande är ett exempel på `while`- och `if`-konstruktioner.

```
>> k = 3;
>> while k>1
      if rem(k,2) == 0
         k = k/2;
      else
         k = 3*k+1;
      end
      disp(k)
   end
    10
     5
    16
     8
     4
     2
     1
```

Matlab har inte något reseverat ord `begin` som man har i till exempel Pascal. Till varje `while`, `if` och `for` hör ett `end` som kommer senare.

Ett så långt exempel som det ovan skrivs egentligen bäst i en fil med en editor, se avsnitt 1.9, men ibland är det bra att skriva korta programslingor direkt i kommandofönstret. För att kunna editera lätt skriver man då flera satser på samma rad, som separeras med kommatecken eller semikolon. Följande kommandon genererade figur 1.2.

```
>> n = 100;
>> a = 1.73;
>> x = 0.5;
>> for i=2:n, x(i)=(1-a*x(i-1))^2; end
>> subplot(211)
>> plot(x,'.-')
```



Figur 1.2: Kaos

Man kan bryta en `for` eller `while`-slinga med `break` precis som i Java eller C. Däremot saknas konstigt nog motsvarande `continue`-instruktion.

En flervägs hoppinstruktion finns också, se hjälptexten till `switch`.

Ingen och alla

I en `if`- eller `while`-konstruktion vill man ofta testa om alla (eller minst ett) element i en vektor har en viss egenskap. Använd då `all` och `any`.

```
>> all(x<100)
ans =
     1
```

Ja, tydligen var alla element i vektorn `x` mindre än 100.

För en matris med minst två rader fungerar de två funktionerna på varje kolonn för sig och lämnar en radvektor som svar. Om man vill testa om det finns något element i en matris `A` som är negativt så skriver man `any(any(A<0))`.

Om villkoret i en `if`- eller `while`-sats har någon annan storlek än en 1×1-matris, dvs en skalär, så kan man fråga sig vad som händer. Svaret är att villkoret betraktas som sant bara om alla elementen är icke-nollor. Man kan alltså testa om två matriser är lika genom att skriva så här:

```
>> if A==B, disp('Yes equal'), end
```

Däremot kan man inte använda

```
>> if A~=B, disp('Fooled! Ha ha!'), end
```

för att testa att de två matriserna inte är lika, eftersom testet som utförs kräver att *alla* element är olika för att skriva ut teckensträngen. Man menade nog

```
>> if any(any(A~=B)), disp('Not equal'), end
```

eller ekvivalent

```
>> if A==B, else, disp('Not equal'), end
```

1.9 Egna program

Matlabprogram är filer med matlabkod. Namnet skall sluta med punkt och bokstaven m, t ex `mittprog.m`, och de kallas därför m-filer. Det finns två varianter av m-filer: skript och funktioner. Skriptfiler är helt enkelt matlabkommandon som utförs som om man hade skrivit dem direkt i kommandofönstret.

Funktioner har inargument och utargument och de börjar alltid med det reserverade ordet `function`. Betrakta följande egna funktion som finns i en fil med namnet `randint.m`. Om någon nödvändigtvis vill ha den så finns den under adressen
`www.maths.lth.se/matstat/staff/andersh/`
så slipper man knappa in den själv.

```
function x = randint(m,n,a,b)
%RANDINT Random integral matrix.
%
%        randint(m,n,a,b)
%
%        Returns an m-by-n matrix with
%        entries between a and b (default
%        is 0 and 9).
if nargin==0, m=1; end
if nargin<2, n=m; end
if nargin<3, a=0; b=9; end
x = floor((b-a+1).*rand(m,n)) + a;
```

Funktionen `floor` avrundar nedåt, till närmast lägre heltal. Några kommentarer om koden i funktionen följer.

- Variabeln `nargin` ger antalet inargument vid anropet. Det kan vara färre än antalet deklarerade argument, men inte fler.
- Koden visar hur man använder `nargin` för att ge värden till inargument som anroparen inte har angivit. Programmet ovan är skrivet för 0, 1, 2 eller 4 argument.
- Det finns ingen instruktion som måste avsluta funktionen, men man kan använda det reserverade ordet `return` om man vill avsluta på något speciellt ställe i koden.
- Utargumentet, `x` i exemplet, måste tilldelas ett värde.
- Hjälptexten till funktionen – den text som visas med kommandot `help` – är det första blocket med kommentartext. Som kommentartext räknas allt som följer efter ett procenttecken på varje rad.
- Funktionsnamnet bör stämma överens med filnamnet – i själva verket bryr sig Matlab bara om filnamnet men man skall ju inte förvirra sig själv eller andra med olika namn.

Det bästa sättet att lära sig hur man skriver bra matlabprogram är att studera kod som finns. Kommandot `type` skriver ut hela programtexten på skärmen.

Övning 1.11
Skriv ut hjälptexten till funktionen `mean` på skärmen. Skriv sedan ut hela programkoden.

För att editera programkod använder man sin favoriteditor. Det finns också en inbyggd editor i Matlab som startas med kommandot `edit minfil.m`, men i Unixmiljö är den så usel så att alla använder Emacs eller någon annan editor istället. Det finns också styrfiler för matlabkod till Emacs att tillgå som gör automatisk indragning och använder tjusiga färger för teckensträngar, kommentarer och liknande.

Varning och fel

När man skriver snygga program som andra skall använda så skall man kontrollera indata och ge varningar eller avbryta om något är orimligt. Man använder då funktionerna `warning` för att varna och `error` för att avbryta, på exempelvis det här sättet:

```
if any(any(x<0))
   error('Illegal x')
end
```

Sökvägar

Hur kör man programmet då? Finns m-filen i aktuell katalog så är det bara att köra. Skriv kommandot som om det vore inbyggt i Matlab. I de flesta andra programmeringsmiljöer måste man kompilera eller ladda in eller göra något speciellt men inte i Matlab. Varje gång man ger ett kommando så kollar Matlab med diverse sofistikerade metoder att programkoden inte har ändrats. Det kan dock hända om man har otur att Matlab har en gammal kopia i minnet och inte upptäcker att man har ändrat i programfilen. Matlab kan till och med köra en gammal version samtidigt som den glatt skriver ut den nya koden när man använder kommandot `type`. För att vara bombsäker begär man då `clear functions` så laddar den in *alla* programfiler på nytt när man kör dem.

Om man vill att Matlab skall köra program från flera olika underkataloger använder man funktionerna `path` och `addpath` för att ta reda på och sätta alla sökvägar.

startup.m

Om man har en fil `startup.m` där man startar Matlab eller i en underkatalog som heter `matlab` så körs den automatiskt. Det kan man utnyttja för att ställa in sökvägar och annat. Unixhackaren Hilbert Älg har en fil `startup.m` i en underkatalog `matlab` med följande innehåll. Tildetecknet på andra raden tolkas liksom i Unix som den egna hemkatalogen. Hilbert har lagt dit den raden därför att han vill att Matlab alltid skall starta med underkatalogen `matlab` som aktuellt bibliotek.

```
disp('GODMORGON HILBERT ÄLG')
cd ~/matlab
addpath('/h/d9/a/f99hia/mymlib')
```

1.10 Tips

Under denna rubrik är såväl viktiga saker som diverse kuriosa samlade.

Vi börjar med förallokering och vektorisering, som helt enkelt är sätt att skriva snabba och oftast mer läsbara matlabprogram. För att mäta tid använder man funktionen `tic`, som nollställer en intern klocka, och `toc` som visar hur mycket tid i sekunder som har gått.

Förallokering

I ett exempel på sidan 7 beräknade vi en kaosprocess med en slinga. Vi lägger till `tic` och `toc` för att mäta tiden och beräknar 10000 steg i stället:

```
>> n = 10000;
>> a = 1.73;
>> x = 0.5;
>> tic, for i=2:n, ...
        x(i)=(1-a*x(i-1))^2; end, toc
elapsed_time =
   46.6477
```

De tre punkterna ovan betyder att satsen fortsätter på nästa rad.

Beräkningen tog 47 sekunder. Det var illa. Anledningen är att Matlab får flytta data hela tiden eftersom `x` blir längre och längre efter hand. Men vi kan förallokera `x` från början så att den aldrig växer:

```
>> x = zeros(1,n);
>> x(1) = 0.5;
>> tic,for i=2:n,x(i)=(1-a*x(i-1))^2;end,toc
elapsed_time =
    3.7569
```

Det tog ju mindre än fyra sekunder! Ändå är fyra sekunder rätt mycket jämfört med vad det skulle ha tagit i ett språk med statisk typkontroll som C, Fortran, eller Java. Anledningen är att Matlab gör en massa kontroller och tidsödande saker mellan varje sats. Beräkningen ovan går inte att skriva om på ett mycket snabbare sätt, men med många andra sorters beräkningar går det. Om det handlar nästa tips.

Vektorisering

Vektorisering är att använda matrisoperationer i stället för programslingor. Antag att vi har en kolonnvektor (med längden 10000) och vill skapa en ny genom att multiplicera varje element med dess index i vektorn. Om man har skrivit mycket kod i konventionella programspråk så skulle man kanske skriva så här (där vi har lagt till `tic` och `toc` för att mäta tiden):

```
>> tic, for i=1:10000, x(i)=x(i)*i; end, toc
elapsed_time =
    3.4519
```

Så skall man inte skriva i Matlab. I stället skall man skriva så här:

```
>> tic, x=x.*(1:10000)'; toc
elapsed_time =
    0.0526
```

Skillnaden i tidsåtgång är avsevärd som synes.

Logiktricket

Antag att vi i en vektor v vill multiplicera ett element med motsvarande element i vektorn u om elementet i u är mindre än ett, annars dividera med det. Nu är det lätt att falla i fällan och skriva en slinga med en `for`-konstruktion innehållande en `if`-sats om man är van vid andra programmeringsspråk. Men i Matlab ser en bra lösning ut så här:

```
>> v = v.*((u<1).*u + (u>=1)./u);
```

Logiska operatorer som `<`, ger vektorer med nollor och ettor som svar och det kan vi utnyttja för att skriva vektoriserad kod som i exemplet.

Stapla kolonner

Ofta vill man ha alla elementen i sin matris eller vektor i en kolonnvektor i stället. Konstruktionen `A(:)` kan användas för att skapa en lång kolonnvektor där kolonnerna i `A` är staplade ovanpå varandra. Om man alltså skriver `x=x(:)` så har man gjort om `x` till en kolonnvektor oavsett om det var en matris eller radvektor från början.

Man kan också använda den speciella notationen på vänstersidan i en tilldelningssats, och då får det betydelsen att formatet på A behålls:

```
>> A = zeros(3,4);
>> A(:) = 1:12
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

Se vidare `reshape`, `repmat`, `fliplr`, `flipud`, `kron`, `meshgrid`, `toeplitz` och `hankel` för snabba sätt att möblera om elementen i matriser och skapa nya matriser med speciella strukturer.

Tonys trick

Många funktioner opererar kolonnvis på matriser, t ex `mean`, `median`, `max`, `min` och `sort`. Med det menas att exempelvis medelvärdesfunktionen `mean` beräknar medelvärdet i varje kolonn för sig och lägger resultetet i en radvektor så att första elementet är medelvärdet från första kolonnen i inargumentet och så vidare.

Om vi vill dra bort medelvärdet från varje kolonn i en matris kan man förstås göra det med en programslinga:

```
>> m=500; n=800;
>> x = rand(m,n);
>> y = zeros(m,n); % Förallokering
>> for i = 1:n
      y(:,i) = x(:,i)-mean(x(:,i));
   end
```

Ett bättre och vektoriserat sätt att göra det är följande.

```
>> y = x - ones(m,1)*mean(x)
```

Man kan emellertid göra ännu bättre. Följande mästerstycke kallas Tonys trick. För att förstå det måste man komma ihåg att en radvektor ju kan indexeras av två index, det är ju samtidigt en matris, låt vara med bara en rad.

```
>> mx = mean(x);
>> y = x - mx(ones(m,1),:);
```

Tonys trick exekverar snabbare än att använda matrismultiplikation, som i förra rutan.

Man kan också använda funktionen `repmat` för renare kod. Den funktionen utnyttjar Tonys trick internt.

```
>> y = x - repmat(mean(x),m,1)
```

Diagonaltricket

Hur kan man nollställa diagonalen i en kvadratisk matris utan att använda en programslinga? Jo, med diagonaltricket:

```
>> n = 5; A = rand(n);
>> J = 1:n+1:n^2;
>> D = A(J);         % Samma som diag(A)
>> A(J) = 0;         % Går ej med diag(A)
```

Tricket bygger på att Matlab accepterar att indexera en matris med bara ett index. Matriselementen numreras då kolonnvis, vilket t ex innebär att `A(2*n+3)` är det tredje diagonalelementet, som ju vanligtvis skulle indexeras `A(3,3)`.

Diagonaltricket är bara ett exempel på hur man kan laborera med att indexera matriser med bara ett enda index.

Sista elementet

Det reserverade ordet `end`, som används vid programslingor, kan också användas på ett helt annat vis vid indexeringar. Då betyder det sista elementet, eller sista raden respektive sista kolonnen. Det sista elementet i en vektor `v` är då `v(end)` och om man vill ha de tre sista elementen i sjunde kolonnen i en matris `A` så skriver man `A(end-2:end,7)`.

Sortering

Sortering illustreras av följande exempel.

```
>> x = [3 7 1 2 5];
>> [y,i] = sort(x)
y =
     1     2     3     5     7
i =
     3     4     1     5     2
```

Det andra elementet i vektorn `i` är fyra, vilket betyder att det fjärde elementet i `x` var det näst minsta (nummer två) i storleksordning. Vektorn `y` är då samma som vektorn `x(i)`. Men hur beräknar man en indexvektor `j` så att `x` är samma som `y(j)`, dvs hur beräknar man inversa avbildningen. Ett alternativ är följande.

```
>> j(i) = 1:length(i)
j =
     3     5     1     2     4
```

Man kan göra många sorteringar sammtidigt genom att utnyttja att funktionen `sort` sorterar varje kolonn för sig om man ger den en matris med minst två rader. Sorteringsordningen `i` blir då en matris där varje kolonn för sig är en sorteringsordning.

Om man vill ha inversa sorteringsordningen för varje kolonn för sig så fungerar inte ovanstående kod. I stället för att ta till en slinga över varje kolonn separat så kan man skriva följande.

```
>> [dummy, j] = sort(i);
```

Polygonarea

Här är ett värstingexempel på den vektoriseringssport som man kan ägna sig åt vid matlabprogrammering.

Om x och y är kolonnvektorer med koordinater för en polygon, vad är då ytan hos polygonen?

Svar:

```
>> abs(sum(diff([x;x(1)]) .*  ...
          (y+[y(2:end);y(1)]))/2)
```

Det tar ett tag att smälta både matematiken bakom lösningen och dess kompakta kodning i Matlab, men när man är van så är det lättare att läsa vektoriserad kod jämfört med kod som innehåller många programslingor (nåja, oftast i alla fall).

Hjälp igen

Hjälpfunktionen och flera andra funktioner fungerar utan att man sätter argumentet inom parenteser. Det kommer sig av att Matlab själv sätter dit apostrofer och parenteser så att `help sin` betyder samma sak som `help('sin')`. Med detta i bakhuvudet kan man efter en stund begripa följande bisarra beteende.

```
>> log pi
ans =
    4.7185    4.6540
```

1.11 Mer

Verktygslådor

Många verktygslådor (*toolboxes*) finns till Matlab. En översikt ges med kommandot `help` (utan argument). Många demonstrationer kan avnjutas med kommandot `demo`.

Exempelvis finns stöd för bildbehandling. Läs mer i texterna till funktionerna `image`, `imread` och `imwrite`. För listor över funktioner och demonstrationsprogram för bildbehandling se `help images` och `help imdemos`.

Glesa matriser

Matlab har ett omfattande stöd för glesa matriser, matriser som mest består av nollor. Man jobba med enormt stora matriser om man bara lagrar värden för de element som inte är noll. Se hjälptexten till `sparse` för vidare kunskaper.

Flerdimensionella matriser

Numera kan man i Matlab använda variabler som indexeras av fler än två index. Exempelvis är `rand(2,3,4)` en $2\times3\times4$-matris med slumptal. Se funktionerna `reshape` och `squeeze`.

Avancerad grafik

Vi har knappt nosat på grafikmöjligheterna. Man kan göra hur mycket som helst (även om vissa delar är långsamma jämfört med andra grafikmiljöer). Se vidare `help graphics`. Ett tips är att läsa programmen till demonstrationsrutinerna och klura ut hur de har gjort. Koden till exempelvis `demo` finns i en katalog som heter `demo`. Sökvägen till den katalogen kan du hitta om du skriver `path` utan argument. Se också funktionerna `get` och `set`. Prova `set(1)` och `set(gca)` efter att ha ritat en bild.

Cellmatriser

Vi har i detta häfte bara talat om matriser med tal. Det finns också cellmatriser. Det är matriser vars element kan vara vad som helst. Syntaxen är att man indexerar sådana matriser med krullparenteser istället för vanliga parenteser. I följande exempel bildas en cellvektor, eller en lista som det skulle heta i andra språk, med tre element: en skalär, en vanlig radvektor och en cellvektor.

```
>> x = {9, [6 7 2], {[], 'hej'}}
x =
    [9]    [1x3 double]    {1x2 cell}
>> x{2}
ans =
     6     7     2
```

Strukturmatriser

Det finns en datatyp som kallas strukturmatris. Ett sätt att se det är att man med en punktnotation kan samla flera variabler i en:

```
>> q.x = 2.6;
>> q.y = 9.1;
>> q.namn = 'Storspov';
>> q
q =
       x: 2.6000
       y: 9.1000
    namn: 'Storspov'
```

Med funktionerna `getfield` och `setfield` kan man använda strukturmatriser som så kallade associativa arrayer (dictionaries) om man vill.

Men då finns det å andra sidan bättre språk att jobba i, interpreterande språk med inbyggt stöd för avancerade datastrukturer, som Icon eller Python, eller kompilerande språk med skräpsamling (garbage collection) och väl utvecklade klassbibliotek, som Java. Matlab är gjort för matrisberäkningar, inte avancerade datastrukturer. Alla tilldelningssatser i Matlab resulterar i så kallad djupkopiering. Det finns inget sätt att få en cellmatris eller strukturmatris att referera till sig själv eftersom det inte finns något som kan kallas referens i Matlab.

Kapitel 2

Maple

Maple är ett matematikprogram, som i första hand inte är konstruerat för att räkna med siffror utan för att beräkna formler med symboler, med bokstäver. Det kan också räkna med siffror förstås och med extremt hög noggrannhet, så många siffrors noggrannhet som man har plats för i datorn och tid att vänta på svaret.

Under MS Windows så startas Maple i menyn ”Programs”→”Maple V Release 4”→”Maple V Release 4” Under Unix startar man i menyn ”System”→”Utilities” som man får fram genom att trycka ner en musknapp i rotfönstret (det ser olika ut i olilka datorer). Man kan också skriva `maple` i ett kommandofönster.

Man avslutar genom att skriva `quit;`. Notera semikolonet. I Maple måste varje kommandorad avslutas med semikolon, annars gör programmet ingenting mer än att vänta på fler knapptryckningar.

2.1 Hjälp

I Maple används ett inledande frågetecken för att fråga om hjälp, t ex betyder `?sin` att man vill ha hjälp med `sin`-kommandot.

Övning 2.1
Vad kandlar kommandot `simplify` om?

2.2 Derivata

Derivatan av en funktion kan beräknas med `diff`:

```
> diff(log(x),x);
                                1/x
```

Övning 2.2
Antag att $f(x) = sin^2(x)$. Vad är derivatan av f?

Gör nästa övning genom att ta tillbaka föregående kommando med piltangenterna och editera det.

Övning 2.3
Antag att $f(x) = cos^2(x)$. Vad är derivatan av f?

2.3 Integral

En primitiv funktion kan hittas med hjälp av kommandot `int`:

```
> int(1/x,x);
                                ln(x)
```

Övning 2.4
Hitta en primitiv funktion till $ln(x)$.

En integral beräknas genom att ange gränser till andra argumentet med hjälp av ett likhetstecken och två punkter. Integralen $\int_1^2 xln(1+x)dx$ beräknas alltså så här:

```
> int(x*ln(1+x),x=1..2);
                              3/2 ln(3) - 1/4
```

2.4 Tilldelning

Tilldelningsoperatorn är tvåteckenkombinationen `:=` till skillnad från Matlab och Java. Förra exemplet hade kunnat skrivas så här:

```
> f := x*ln(1+x);
                              f := x ln(1 + x)

> c := int(f,x=1..2);
                           c :=  3/2 ln(3) - 1/4
```

2.5 Numeriskt svar

Ett numeriskt svar kan man få med kommandot `evalf`. Vill vi ha ett numeriskt svar till förra exemplet kan vi skriva så här:

```
> evalf(c);
                                1.397918434
```

Räcker inte antalet siffror så kan man tilldela den inbyggda variabeln `Digits` ett högre tal.

```
> Digits := 40;
                                Digits := 40

> evalf(c);
  1.3979184330021645370928678553837885569 71
```

Övning 2.5
Beräkna $\sqrt{2}$ med tusen siffrors noggrannhet.

2.6 Sök själv

Det gäller precis som i Matlab att använda sin fantasi och hjälpfunktionen för att komma snabbt fram.

Övning 2.6
Försök att faktorisera ditt telefonnummer och personnummer.

Bilaga A

Lösningar till övningsuppgifterna

0.1 `sum(log(11:2:99))`

1.1 `21.6042`

1.2 `58.7262`

1.3 Nej, men det finns en funktion `atan2` som klarar just den saken.

1.4 `mod` och `rem`.

1.5 Fråga övningsledaren om det är oklart.

1.6
```
z = zeros(7,5)
z(3:5,2:4) = 1
```

1.7 Fråga övningsledaren om något är oklart.

1.8 Fråga övningsledaren om något är oklart.

1.9 Fråga övningsledaren om något är oklart.

1.10
```
pwd
mkdir matlab
cd matlab
who
save minfil
dir
clear
who
load minfil
who
```

1.11 `type mean`

2.1 `simplify` förenklar algebraiska uttryck.

2.2 `diff(sin(x)^2,x);`
vilket ger
$$2\sin(x)\cos(x)$$

2.3 `diff(cos(x)^2,x);`
vilket ger
$$-2\sin(x)\cos(x)$$

2.4 `int(ln(x),x);`
vilket ger $xln(x) - x$.

2.5 1.41421356237309504880168872420...

2.6 Funktionen `ifactor` kan användas.