Matematik med MATLAB®

En handledning

Matematik CTH/GU

Version J.L. 11 augusti 1997
Reviderad J-E.A. augusti 2002

Matematik med MATLAB®

En handledning

Matematik CTH/GU

Version J.L. 11 augusti 1997
Reviderad J-E.A. augusti 2002

Detta är en handledning till MATLAB, ursprungligen för version 5, men här åtminstone delvis modifierad för version 6. Användaren tänks gå igenom handledningen genom att själv arbeta igenom textens exempel.

Syftet med handledningen är att hjälpa användaren att snabbt komma igång med att använda MATLAB och att därefter på egen hand utveckla sin förmåga att använda MATLAB som ett avancerat generellt beräkningsprogram och som ett programmeringsspråk.

I handledningen används följande konventioner vad gäller stilsorter.
Kursiv text för MATLAB-kommandon och svar på sådana kommandon.

Handledningen har sju sektioner, nämligen

1 Grunderna

1.1 Hur man startar och slutar

Man startar MATLAB genom att dubbelklicka på ikonen för MATLAB. Då öppnas så småningom ett fönster med tre delar. Till höger finns "Command Window" i vilket man sedan arbetar. Högst upp till vänster finns "Launch pad" som är ett hjälpfönster och under detta "Command History ". Detta är ett historiefönster där de kommandon man skriver i Command Window sparas och kan återanvändas genom att man dubbelklickar på dem. Det finns möjlighet att ta fram några andra fönster som man hittar under rubriken "View".

När man startat MATLAB får man promptern
>>
i "Command Window" som talar om att programmet är beredd att ta emot instruktioner.
Man lämnar programmet antingen under rubriken "File" välja "Exit MATLAB" eller genom att mata in *quit* eller *exit* i "Command Window".
Man avbryter pågående beräkningar med kommandot $Control - c$, (dvs tryck på knappen *Control* och bokstaven c samtidigt).
All inmatning avslutas med vagnretur RETURN (eller ENTER).

Det kan också vara bra att veta att om man skriver långa kommandon som inte får plats på en rad kan man fortsätta till ny rad genom att avsluta raden med
...
följt av RETURN.

1.2 Aritmetiska operationer

De vanliga aritmetiska operationerna mellan tal ser ut så här :

+	addition
−	subtraktion
*	multiplikation
/	division
^	exponentiering

Talet π skrivs pi medan talet e skrivs $exp(1)$
Några exempel:
Skriver du
$>> 100 * pi$
blir svaret
$ans =$
314.1593 (*ans* står för det senaste svaret).
MATLAB tillämpar den vanliga prioriteringsordningen mellan de aritmetiska operationerna :
>> 8^2/3
ger svaret 21.3333 d.v.s 64/3, medan
>> 8^(2/3)
ger svaret 4.0000.

1.3 Elementära funktioner

MATLAB har alla de vanliga elementära grundfunktionerna , till exempel exponential- och logaritmfunktionerna, de trigonometriska funktionerna, kvadratrotsfunktionen och flera andra. Här följer en lista som visar hur dessa skrivs i MATLAB :
exp , $log(= \ln)$, $log10(= \lg)$
sin , cos , tan , $sqrt$
Det finns flera inbyggda funktioner. Vi återkommer till hur du skall ta reda på vilka.
Exempel: Vi beräknar $\ln(\sqrt{e})$:
$>> log(sqrt(exp(1)))$
$ans = 0.5000$

1.4 Variabler

Variabler kan ges namn innehållande bokstäver (dock ej å,ä och ö) och siffror. Första tecknet måste vara en bokstav. Man bör inte använda namn på inbyggda funktioner eller MATLAB-kommandon som variabelnamn. Programmet skiljer mellan stora och små bokstäver.
Tecknet = används för att tilldela variabler värden.
$>> x = pi/3$
ger variabeln x värdet $\pi/3$.
Man kan ge instruktioner på samma rad om man skiljer dem åt med ett komma (eller ett semikolon).
$>> x = pi/4, X = sin(x)$
ger
$x = 0.7854$ och $X = 0.7071$.
Instruktionen
$>> x = x + 1$
ger sedan $x = 1.7854$.

1.5 Redigering och formatering

Man hindrar programmet att skriva ut svar genom att avsluta en instruktion med semi-kolon
(före RETURN).
$>> z = exp(1);$
>>
Du kan kontrollera att variabeln z verkligen har fått värdet e genom att skriva
$>> z$

Man kan återfå tidigare givna kommandorader genom att trycka på piltangenten upp ▲. Senare kommandon får man tillbaka genom att trycka på ▼.
Man kan korrigera ett givet kommando genom att flytta markören med piltangenterna ◀ och ▶ och sedan ta bort tecknen till vänster om markören antingen med knappen Back Space eller med Delknappen. (Du får kontrollera själv vilket som fungerar.) Ny text kan skrivas in vid markörens plats.
Man kan dirigera antal decimaler som skrivs ut och formen på utskriften med hjälp av kommandot $format$. De vanligaste varianterna är
$format\ short$
som ger fem signifikanta siffror och
$format\ long$
som ger femton signifikanta siffror. Exempel
$>>\ format\ long;\ 10 * pi$
$ans = 31.41592653589793$
$>>\ format\ short;\ ans$
ger sedan
$ans = 31.4159$

2 Hjälp i MATAB

2.1 Beskrivning av *help*−kommandot

MATLAB är väl utrustat med direkthjälp på skärmen. Hjälpfunktionen är inte främst till för att ge hjälp i nödsituationer. Den är istället tänkt som ett redskap att med vars hjälp du på egen hand kan utvidga ditt kunnande om MATLAB. Hjälpfunktionen är organiserad i nivåer. Pröva med att dubbelklicka på MATLAB i "Launch pad"

Förutom i hjälpfönstret kan man få onlinehjälp med hjälp av kommandot *help*. Skriver man bara detta erhålls en lista med ämnesrubriker, där varje rad anger ett underbibliotek och dess innehåll. Här reproduceras några av de första raderna.

$>>\ help$

```
HELP topics:
matlab/general   - General purpose commands.
matlab/ops       - Operators and special characters.
matlab/lang      - Language constructs and debugging.
matlab/elmat     - Elementary matrices and matrix
                   manipulation.
matlab/specmat   - Specialized matrices.
matlab/elfun     - Elementary math functions.
```

Om man dubbelklickar med musen på en rubrik i hjälpfönstret eller ger kommandot *help* följt av ett biblioteks namn, får man information om vad som finns i detta bibliotek och hur du skall få veta mer. Pröva att dubbelklicka på *help ops* (eller ge detta kommando i matlab-fönstret). Du får då en lång lista med information om vilka operationer MATLAB har. Dubbelklicka på en operation så får du veta vad den utför. Man kan t.ex. få informationen

```
+   Plus.
    X + Y adds matrices X and Y. X and Y must
    have the same dimensions unless one is a scalar (a
    1-by-1 matrix). A scalar can be added to anything.
```

2.2 Exempel på användning av *help*

Låt oss ta reda på vilka inbyggda elementära funktioner som finns i MATLAB och sedan undersöka vad två av dem gör. De elementära funktionerna finns i biblioteket $elfun$. Börja med att gå tillbaka i hjälpfönstret och dubbelklicka på $>>\ help\ elfun$ (eller ge detta kommando).
Du får en lång lista av funktioner. Du känner förmodligen igen flera av dem, men fix är kanske en nyhet. För att ta reda på vad fix−funktionen gör ge kommandot
$>>\ help\ fix$

```
FIX   Round towards zero.
      FIX(X) rounds the elements of X
      to the nearest integers towards zero.

      See also FLOOR, ROUND, CEIL.
```

Pröva vad funktionen gör genom att exempelvis låta datorn räkna ut några värden.
$>>\ fix(1.5)$
$ans = 1$
$>>\ fix(-1.5)$
$ans = -1$

Innan man är van kan det vara svårt att begripa vad en *help*−information innebär. Kommandot
$>>\ help\ abs$
ger flera rader information, varav de första ser ut så här:

```
ABS   Absolute value and string to numeric
      conversion.
      ABS(X) is the absolute value of the elements of
      X.
      When X is complex, ABS(X) is the complex
      modulus (magnitude) of the elements of X.
```

Om man nu inte vet vad som menas med ordet "string" känner man sig kanske blåst. Bättre är att lägga märke till det man begriper. På andra raden där det står att det handlar om absolutbelopp. Kommandot
$>>\ abs(-pi)$

ger det väntade svaret
$ans = 3.1416$
Anledningen till att informationstexten talar om "the elements of X" är att MATLAB nästan alltid accepterar matriser (rektangulära scheman av tal) som argument. Vi återkommer till detta i nästa avsnitt och i avsnitt 6. Komplexa tal och innebörden av ordet "string" skall vi komma in på i avsnitt 4.

2.3 Felmeddelanden

Använder man funktioner eller kommandon som inte finns inbyggda i MATLAB och inte heller finns definierade på annat sätt, så kommer programmet att pipa och ge felmeddelande. Samma sak händer om man använder en variabel som inte tidigare givits ett värde. Det är lätt hänt att man råkar stava fel eller att det smyger in sig en stor bokstav där det skall vara en liten. Här några exempel.
$>> plutten$
??? Undefined function or variable 'plutten'.
MATLAB vet inte vad *plutten* är. Du måste först ge *plutten* ett värde, exempelvis
$>> plutten = 0;$
Nu går det bra att skriva $>>$ *plutten* utan att råka ut för protester.
Ännu ett exempel:
$>> x = sin$
??? Error using ==¿ sin
Incorrect number of inputs.
Man måste naturligtvis ge något argument som sinus-funktionen kan räkna ut.
$>> x = sin(plutten)$
går till exempel bra.
Här ett exempel där antalet argument är för stort:
$>> x = sin(2, 3)$
??? Error using ==¿ sin
Incorrect number of inputs.
Här ett exempel där man felaktigt använder stora bokstäver.
$>> X = SIN(2)$
??? Undefined variable .... ;
Caps Lock may be on
Förklaringen till den sista raden är följande. En vanlig orsak till att man av misstag får stora bokstäver är att man råkat trycka på knappen *Caps Lock*. Då lyser en liten lampa på knappen. Tryck ner den igen i så fall!

2.4 Kommandot *lookfor*

Om du vill ha reda på MATLABs namn på en funktion kan du pröva *lookfor*.
$>> lookfor\ logarithm$
ger till exempel upplysning om vilka logaritmer som finns. Kommandot *lookfor abc* letar upp alla rutiner som innehåller texten *abc* på första raden i *help*−texten. Kommandot $lookfor\ -all\ abc$ letar igenom hela *help*−texten, inte bara första raden. Kommandot
$>> lookfor\ -all\ arctan$
ger (bl.a.) svaret
ATAN Inverse tangent
medan enbart $>> lookfor\ arctan$ inte ger någon information alls.
Anm.: Du avbryter MATLABs letande med *Ctrl c*.

3 Radvektorer och kurvritning

3.1 Operationer med radvektorer

MATLAB kan som nämnts arbeta med flera tal samtidigt, t.ex. skrivna som en matris eller en radvektor $(x_1, x_2, \ldots, x_n)$. Talen n kalls vektorns längd (*length*) och $(1, n)$ dess matrisstorlek (*size*). Följande kommandon definierar två radvektorer x och y av längden 3:
$>>\ x = [1\ \ 2\ \ 3];\ y = [4\ \ 5\ \ 6];$
Om man vill kan man också skriva komma mellan talen.
$>>\ x = [1, 2, 3];$
Om x och y är två radvektorer av samma längd kan man utföra koordinatvis addition och subtraktion genom att skriva $x+y$ resp. $x-y$. Man kan också utföra koordinatvis multiplikation och division med kommandona $x.*y$ resp. $x./y$. Man kan även använda de elementära funktionerna på radvektorer. Allmän hjälp om detta område får du med *help elmat* och *help ops*.
Exempel (med utskrift i *format short*):

$>> x = [1\ \ 2\ \ 3];\ y = [4\ \ 5\ \ 6];$		
$>> x + y$	ger	5 7 9
$>> x.*y$	ger	4 10 18
$>> x./y$	ger	0.2500 0.4000 0.5000
$>> x.\hat{}\, y$	ger	1 32 729
$>> exp(x)$	ger	2.7183 7.3891 20.0855

Däremot ger
$>> x * y$
felmeddelandet
??? Error using ==¿ *
Inner matrix dimensions must agree.

Symbolen $*$ betyder nämligen matrismultiplikation vilket är något annat än elementvis multiplikation.
Instruktionen
$>> x/y$
ger däremot inget felmeddelande utan det gåtfulla svaret $ans = 0.4156$. Förklaringen ges i avsnitt 6.

Viktiga specialkommandon är *ones* och *zeros*. De användes för att generera matriser bestående av enbart ettor respektive nollor. T.ex. ger kommandot $ones(1,3)$ eller $ones(size(x))$ svaret 1 1 1 om x är en vektor av längden 3. Andra exempel:

$>> x = [1\ \ 2\ \ 3\ \ 4];$
$>> ones(size(x))./x$
ger svaret 1.0000 0.5000 0.3333 0.2500
$>> x + ones(size(x))$ ger svaret 2 3 4 5
$>> 2 * ones(size(x))$ ger svaret 2 2 2 2

Det näst sista svaret hade man också kunnat få genom att skriva $x + 1$. I det andra kommandot kan man istället skriva $1./x$
Kommandot $ones(1,n)$ ger en radvektor med n stycken ettor.

3.2 Generering av aritmetiska följder

Om a, h och b är givna tal kan man bilda radvektorn $x = (a, a+h, a+2h, \ldots, b)$ med hjälp av kommandot $x = a : h : b$. Detta är ett av MATLABs mest användbara kommandon.
$>> x = -5 : 2 : 5$
$x =$
$-5 \quad -3 \quad -1 \quad 1 \quad 3 \quad 5$
Analogt ger kommandot
$>> x = -pi : 0.1 : pi;$
radvektorn $x = (-\pi, -\pi + 0.1, \ldots, \pi)$, vilket är en vektor av längden 63. Kolla genom att mata in x enligt ovan och sedan skriva
$>> length(x)$
Låt också datorn skriva ut vektorn x genom att skriva
$>> x$ (utan semi-kolon)
Vill man ha steglängden $h = 1$ räcker det att skriva $>> x = a : b$ t.ex.
$>> x = 0 : 10$
$x = 0 \ \ 1 \ \ 2 \ \ 3 \ \ 4 \ \ 5 \ \ 6 \ \ 7 \ \ 8 \ \ 9 \ \ 10$
Om $b < a$ så kan man ha $h < 0$.
Ytterligare information får du med *help colon*.

3.3 Kurvritning

Om $x = (x_1, \ldots, x_n)$ och $y = (y_1, \ldots, y_n)$ är två radvektorer av samma längd så kommer ritkommandot $plot(x, y)$ att rita en kurva som sammanbinder punkterna $(x_1, y_1), \ldots, (x_n, y_n)$. Om man inte ger ett särskilt kommando kommer MATLAB att välja koordinataxlarna så att samtliga punkter syns. (Detta kallas autoscaling.) Man kan dirigera på vilket sätt kurvan ritas genom att ge order om särskild linjetyp. Det går också att rita ut punkterna utan att sammanbinda dem genom att bestämma vilken punkttyp som skall användas. Här några exempel:

$>> x = -3 : 0.5 : 3; y = sin(x);$
$>> plot(x, y)$
$>> plot(x, y, ':')$ (Prickad kurva.)
$>> plot(x, y, 'o')$ (Punkterna som små cirklar.)

Allmän hjälp för tvådimensionell grafik får du under rubriken *help plotxy*. Du kan få bl.a. en förteckning av alla linjetyper och punkttyper som finns genom att kalla på *help plot*. Se också avsnittet 2.5 ovan.

3.4 Funktionskurvor

Av exemplen ovan framgår det redan hur man kan rita funktionskurvor. Säg att vi vill rita funktionen $f(x)$, på intervallet $a \le x \le b$. Man börjar då med att välja en lämplig steglängd h och bildar sedan vektorn $x = a : h : b$. Därefter skriver man in funktionen på formen funktionsnamn = funktionsuttryck. Därefter får man funktionskurvan uppritad med kommandot $plot(x, \text{funktionsuttryck})$. Ett exempel gör detta klarare:
$>> x = -2 * pi : 0.1 : 2 * pi;$
$>> f = sin(3 * x) + cos(5 * x); plot(x, f)$
Här ritas funktionen $f(x) = \sin 3x + \cos 5x$ på intervallet $[-2\pi, 2\pi]$. På samma sätt ritar man funktionen $g(x) = x \sin x^2$ på samma intervall med hjälp av kommandot
$>> g = x. * sin(x. * x); plot(x, g)$

Ett annat sätt att rita kurvor är att använda kommandot *fplot*. Detta kommando beskrivs i nästa avsnitt. Se också kommandot *ezplot*.

3.5 Flera kurvor i samma figur

Antag att vi vill rita funktionerna f och g ovan i samma figur. Vi kan då ge kommandot

$plot(x, f, x, g)$
Vill man ha båda kurvorna heldragna kan man skriva
$plot(x, f, '-', x, g, '-')$
Det finns också en annan möjlighet. Antag att vi först ritar funktionen f med hjälp av kommandot
$>> plot(x, f)$
Man kan då först ge kommandot
$>> hold\ on$
Då kommer de gamla kurvorna att behållas när nya kurvor ritas. Till exempel
$>> plot(x, g)$
När man inte längre vill behålla gamla kurvor ger man kommandot
$>> hold\ off$
Kommandot *hold* ensamt innebär att man skiftar från "hold on-läge" till "hold off-läge", (om man tidigare gett kommandot *hold on*) eller från "hold off-läge" till "hold-on-läge". Titta på *help hold* och läs.

3.6 Dimensionering av koordinataxlarna

Normalt väljer MATLAB ett koordinatsystem så att alla punkter som skall ritas syns på skärmen. Man kan styra valet av koordinataxlar med hjälp av kommandot *axis*. För att ta reda på hur *axis* fungerar skriv
$>> help\ axis$
Pröva följande exempel:
$>>\ x = 0 : 0.1 : pi; y = sin(x); plot(x, y)$
$>>\ axis([-1\ 4\ \ -1\ 2])$
$>>\ axis('off')$
$>>\ axis([0\ pi\ 0\ 1])$
$>>\ axis('on')$
$>>\ axis(axis); hold\ on$
$>>\ x = -1 : 0.1 : 2; y = cos(x); plot(x, y)$
$>>\ hold\ off$
$>>\ x = -1 : 0.1 : 2; y = cos(x); plot(x, y)$

4 Bra att veta

4.1 Strängar och kommandot *eval*

Elementen i en matris kan även vara teckensträngar (eng. "string"), dvs följder av tecken placerade mellan två apostrofer ', (ibland kallade fnuttar).
$>>\ A = 'Tjosan\ ,\ Hoppsan\ !\ '$
$A\ =$
$Tjosan\ ,\ Hoppsan\ !$
Detta använder man bland annat då man vill ha text i en graf (se nedan) men också för att bilda funktionsuttryck såsom $f = '\ x.\hat{}a. * exp(-x)\ '$;
Kommadot *eval* används för att "klä av" en textsträng till ett aritmetiskt uttryck.
$>> f = '\ x.\hat{}a. * exp(-x)\ ';$
$>> x = 0 : 0.1 : 10;$
$>> a = 0; plot(x, eval(f))$
Medan f här är en sträng blir $eval(f)$ en vektor, nämligen $x.\hat{}a. * exp(-x)$
Genom att ändra a får man en annan vektor $eval(f)$.
$>> hold\ on;$
$>> a = 0.5;\ plot(x, eval(f));$
$>> a = 1.0;\ plot(x, eval(f));$
$>> hold\ off$

4.2 Kommandot *fplot*

Detta är ett enkelt sätt att rita en funtion definierad med ett uttryck inom strängfnuttar på ett givet intervall $[a, b]$. Pröva
$>> fplot('sin(3 * x) + cos(5 * x)', [-2 * pi, 2 * pi])$
OBS: Variabeln måste heta just x.

4.3 Komplexa tal

Komplexa tal representeras på formen $a+bi$ (eller $a+bj$). Man räknar med dem som med reella tal. Exempel:
$>> (1 + 2i) * (3 + 4i)$
$ans\ =\ -5.0000 + 10.0000i$
$>> (1 + 2i)/(3 + 4i)$
$ans\ =\ 0.4400 + 0.0800i$
Man kan även använda komplexa tal i radvektorer, matriser och elementära funktioner.

Konjugering, beräkning av absolutbelopp, reaaldel och imaginärdel görs med kommandona *conj*, *abs*, *real* resp. *imag*.

En varning : Symbolen i är reserverad för den imaginära enheten. Därför är det olämpligt att använda i som variabelnamn. Om man har råkat använda i som variabel, kan man återställa det gamla värdet som imaginär enhet med kommandot $i = sqrt(-1)$.

4.4 Polynom

Man kan mata in polynom genom att ange polynomets koefficienter som en radvektor i fallande ordnig från högstagradskoefficienten nedåt. Polynomets värde kan sedan beräknas med kommandot *polyval* och dess rötter med *roots*. Exempel:
$>> myPol = [1\ 2\ 3]; roots(myPol)$

Här matar man polynomet $myPol(x) = x^2 + 2x + 3$, vars rötter är $-1.0000 \pm 1.4142i$.
Värdet i x beräknas med kommandot $polyval(myPol, x)$
$>> polyval(myPol, [2\ 3\ 4])$
$ans\ =\ 11\ 18\ 27$
Vill man rita ett polynom får man först beräkna dess värden i ett antal punkter och sedan använda $plot-$kommandot.
$>>\ x\ =\ -5\ :\ 0.1\ :\ 5; y\ =\ polyval(myPol, x); plot(x, y)$
För ytterliga specialkommandon för polynom se *help polyfun*.

4.5 Spara, rensa och återvinna data

Kommandot *who* ger en lista på alla aktuella variabler. När man går ut MATLAB (*quit* eller *exit*) kommer dessa variabler att försvinna. Man kan spara dem (namn och data) genom att ge kommandot *save*. Data kommer då att sparas i en fil som heter "matlab.mat". Samma data kan sedan återvinnas med kommandot *load*.
Vill man att data skall lagras i en fil med annat namn skriver man *save* följt av filnamn, till exempel
>> save temp
Data lagras då i en fil med namnet "temp.mat" i aktuellt bibliotek.

Samma data återvinns med kommandot *load temp*. Vill man att endast vissa variabler, t.ex. A,B,C , skall sparas skriver man
>> save A B C temp

Man kan rensa minnet från alla aktuella variabler genom att ge kommandot *clear* . Samma kommando följt av en lista på variabler rensar ut de listade variablerna, t.ex. *clear A*. Sådana utrensningar kan vara befogade att göra om man har för avsikt att börja med ett nytt problem och vill undvika sammanblandningar med tidigare variabler.

Om man vill rensa grafikfönstret ger man kommandot clf , (clear figure).

4.6 Text i figurer

Man kan placera in text i grafikfönstret med kommandot *text* så här:
$>>\ text(xpos, ypos, 'Själva\ texten\ inom\ apostrofer')$
Här är $(xpos, ypos)$ textens startposition i det koordinatsystem som gäller för aktuell figur. Man kan sätta namn på axlarna med kommandot *xlabel* och *ylabel*. Se också kommandona *title* och *gtext* .

4.7 Tredimensionell grafik

Detta är ett omfattande kapitel. Här får du bara en kort-kort introduktion men med *help plotxyz* får du reda på ytterligare ritmöjligheter.
Man kan rita funktionsytan av en given funktion av två variabler med hjälp av kommandona *meshgrid* och *mesh*. Vi ger ett exempel där vi ritar funktionsytan till funktionen

$$f(x, y) = xye^{-x^2-y^2}$$

över rektangeln

$$(x, y) : -2 \le x \le 2\ ,\ -3 \le y \le\ 3\ .$$

$>>\ f =' x. * y. * exp(-x. * x - y. * y)\ ';$
(OBS Apostroferna $'\ \ '$!.)
$>>\ [x, y] = meshgrid(\ -2 : .2 : 2\ ,\ -3 : .2 : 3\ );$
(Definierar området.)
$>>\ mesh(x, y, eval(f))$
(Ritar funktionssytan.)

5 Att skapa egna kommandon

En av de viktigaste finesserna med MATLAB är att man kan utvidga arsenalen av kommandon med sina egna program. Detta kan ske på två sätt, antingen i form av s.k. script-filer eller i form av funktionsfiler. Båda filtyperna kallas M-filer eftersom deras namn alltid slutar med ".m". Script-filerna består av en uppsättning vanliga MATLAB-kommandon medan funktionsfiler fungerar som egendefinierade kommandon. Allmän information får man med *help script* och *help function*.

För att kunna skriva egna filer måste du öppna din editor och flytta editeringsfönstret till lämplig plats på skärmen. Du kan skriva in filer i editeringsfönstret och testa dem direkt från matlab-fönstret samtidigt som du kan ge UNIX-kommandon från terminalfönstret.

Innan du provar en ".m"-fil måste du spara den. Det är viktigt eftersom MATLAB inte läser från skrivbufferten. Du måste ovillkorligen spara den med i formatet "filnamn.m".

5.1 Textfiler

Textfiler (eller script-filer) består av en samling vanliga MATLAB-kommandon. Följande fil är skriven för att rita $\sin(x)^N$ på intervallet $[a, b]$ för

olika a och b och för olika heltalsvärden på N. Till en början sätts $a = -4\pi,\ b = 4\pi$ och $N = 2$. Så här ser filen ut:

"mySinus.m"

```
a=-4*pi;b=4*pi;
N=2;
Steg=(b-a)/1000;
x=a:Steg:b;
y=sin(x).^N;
plot(x,y);
```

Kommentarer:

De två första raderna ger värden år intervalländpunkterna a och b samt heltalet N.

Nästa rad bestämmer steglängden $Steg$ till en tusendel av intervallets längd.

Därefter definieras radvektorerna x och $y = sin(x)^N$ som sedan plottas i sista raden.

OBS. Man använder elementvis exponentiering . eftersom x och $\sin(x)$ kan vara radvektorer.

Skriv in filen ovan i editeringsfönstret (exklusive kommentarerna) och spara den under namnet 'mySinus.m'. Kontrollera att filen sinusN.m finns i ditt aktuella bibliotek (matlab-biblioteket) genom att ge kommandot *what* i matlab-fönstret. Testa sedan följande kommando från matlab-fönstret :

$>> mySinus;$

Du kan nu gå tillbaka till editeringsfönstret och ändra (exempelvis) värdet på N. Ändra andra raden i filen "mySinus.m" till

N=3;

Spara filen (glöm inte det) och kör den på nytt

$>> mySinus;$

Du kan naturligtvis också ändra intervalländpunkterna a och b, liksom steglängden h och funktionen sin. Sätt igång och experimentera! Vill du se kurvorna på samma bild använd kommandot *hold on*. Glöm i så fall inte att avsluta med $hold\ off$.

Skriv gärna något medvetet fel i filen och se vilket felmeddelande du får.

En viktig observation är att alla variabler som förekommer i en script-fil är globala, d.v.s. de är tillgängliga utanför filen. Man kan testa detta genom att ge kommandot

$>> Steg$

OBS: Stort S som första bokstav.

5.2 Funktionsfiler

En funktionsfils namn skall vara "funktionsnamn.m". Det första ordet i en funktionsfil är $function$. Det följs sedan av en beskrivning av funktionens utvariabler (output) , om någon sådan finns, funktionsnamn och funktionens invariabler (input).

Låt oss skriva en enkel funktion. Den skall heta "myfunk", ha en invariabel och en utvariabel. Filen skall alltså heta "myFunk.m".

Den första raden skall då se ut så här:

```
function ut=myFunk(in)
```

Härefter kommer beräkningsdelen av funktionen. Alla beräkningsinstruktioner bör avslutas med ; för att undvika att mellan-kalkyler visas på skärmen.

Till sist kommer raden där utvariabelns värde anges. Därefter skall inget mer finnas i filen (utom möjligen kommentarer).

Här följer ett första exempel på en funktion.

"myFunk.m"

```
function y=myFunk(x)
taeljaren=x.*x-2*x-1;
naemnaren=1+x.*x;
y=taeljaren./naemnaren;
```

Kommentarer:

Funktionen är $myfunk(x) = (x^2 - 2x - 1)/(1 + x^2)$

I första beräkningsraden beräknas täljaren $x^2 - 2x - 1$.

Man använder .* eftersom input kan vara en radvektor

I andra beräkningen beräknas nämnaren $1 + x^2$

Därefter beräknas output y .

(OBS. Divisionen betecknas ./)

Skriv in denna funktion (exkl. kommentarerna) i editeringsfönstret, spara under namnet "myFunk.m" och provkör direkt med följande kommandon

$>> x = 0 : 0.01 : 5; y = myFunk(x); plot(x, y);$

Observera att varken invariabler eller utvariabler måste ha samma namn som de har i funktionsdefinitionen. Man hade alltså lika gärna kunnat skriva

$>> t = 0 : 0.01 : 5; z = myFunk(t); plot(t, z);$

Till skillnad mot situationen med script-filer är alla variabler som används inuti en funktion är lokala, d.v.s. de är inte åtkomliga utifrån. Kontrollera genom att ge kommandot

$>> taeljaren$

De enda variabler som kan användas utifrån är de som förekommer i funktionens utvariabler.

Observera också att om man har givit en variabel ett visst värde i matlab-fönstret, så kommer detta värde inte att påverkas även om man använder samma variabelnamn innuti en funktion.

För ytterligare information om funktionsfiler se avsnitt 7.

5.3 Hur man skriver egna *help*−kommandon

Man kan skriva egna kommentarer var som helst i en M-fil, bara man låter kommentaren inledas med tecknet %. MATLAB ignorerar då resten av raden. Om man sätter in sådana kommentarer överst i en script-fil eller omedelbart efter funktionsdeklaratioen i en funktionsfil, så kommer dessa kommentarer att skrivas ut om man ger kommandot *help* följt av filnamnet (exklusive ".m"). Här ges två exempel där filerna "mySinus.m" och "myFunk.m" kompletterats med kommentarer.

"mySinus.m"

```
% mySinus
%       Denna script-fil är till för att rita grafen av
%       y=sin(x).^N på intervallet (a,b).
%
a=-4*pi;b=4*pi;       % Val av intervallets ändpunkter
N=2;                  % Val av exponententen N
Steg=(b-a)/1000;      % Val av steglängd
x=a:Steg:b;
y=sin(x).^N;          % OBS : elementvis exponentiering
plot(x,y);
```

Skriv in kommentarerna och provkör sedan filen för att se om allt blivit rätt. Ge sedan kommandot
$>>$ *help mySinus*

Nedan har på liknande sätt funktionsfilen "myFunk.m" försetts med kommentarer och hjälptext.

"myFunk.m"

```
function y=myFunk(x)
% myFunk
%       myFunk(x)=(x.*x-2*x-1)./(1+x.^2)
taeljare=x.*x-2*x-1;
% använd .* eftersom input x t.ex. kan vara en radvektor
naemnare=1+x.*x;
y=taeljare./naemnare;         % obs ./
```

Skriv in kommentarerna och provkör. Testa också
$>>$ *help myFunk*

Det är klokt att ta för vana att sätta in kommentarer och hjälp-information. Man glömmer lättare än vad man tror. Dessutom är det ofta hopplöst svårt att begripa vad okommenterade program egentligen gör till och med om man själv har skrivit dem.

6 Matrishantering

Detta avsnitt förutsätter att läsaren är bekant med de grundläggande begreppen matrismultiplikation, system av linjära ekvationer och invers matris. Hjälp till detta kapitel hittar du i biblioteken *ops, elmat, specmat* och *polyfun*.

6.1 Grundläggande matrisoperationer

Man matar in matriser radvis med semi-kolon mellan raderna och mellanslag eller komma mellan elementen i en rad. Så till exempel ger inmatningen
$>> A = [\ 1,\ 2,\ 3,\ ;\ 4,\ 5,\ 6,\ ;\ 7,\ 8,\ 0\ ]$
resultatet

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{matrix}$$

Symbolen för *transponering* är $'$. (Apostrofen har alltså en dubbel funktion.) Transponering av en radvektor ger till exempel kolonnvektor:
$>> x = [-1\ 0\ 2\ ]'$
ger alltså resultatet

$$x = \begin{matrix} -1 \\ 0 \\ 2 \end{matrix}$$

Addition, subtraktion och multiplikation av matriser betecknas med $+$, $-$ resp. $*$.
Exempel:
$>> b = A * x$

$$b = \begin{matrix} 5 \\ 8 \\ -7 \end{matrix}$$

6.2 Ekvationssytem (matrisdivision)

Det finns två symboler för matrisdivision i MATLAB, nämligen \ och / . Om A är en icke-singulär (d.v.s. inverterbar) kvadratisk matris, kan man entydigt lösa systemet $A * X = b$ för varje matris b som har lika många rader som A. Lösningen X får man med kommandot $X = A\backslash b$. Exempel, med ovanstående matriser A och b :
$>> X = A\backslash b$

$$X = \begin{matrix} -1 \\ 0 \\ 2 \end{matrix}$$

Analogt ger kommandot $Y = c/A$ lösningen till systemet $Y * A = c$ för varje matris c som har lika många kolonner som A. Exempel:
$>> c = [1\ 2\ 3\ ]\ ;\ Y = c/A$

$Y = 1\ 0\ 0$
Låt oss sammanfatta:

$X = A\backslash b$	ger en lösning till	$A * X = b$
$Y = c/A$	ger en lösning till	$Y * A = c$

Divisionskommandona \ och / kan användas även för icke kvadratiska matriser. Pröva till exempel att lösa systemet

$$\begin{aligned} x_1 + 2x_2 &= b_1 \\ 3x_1 + 4x_2 &= b_2 \\ 5x_1 + 6x_2 &= b_3 \end{aligned}$$

Vi kallar systemets matris för W. I matrisbeteckningar blir systemet alltså $W * x = b$. Så här löser vi det:
$>> W = [\ 1\ 2\ ;3\ 4\ ;\ 5\ 6\ ]\ ;\ b = [1\ 3\ 5]\ ';W\backslash b$
$ans =$
1
0
Kontrollera att svaret är rätt genom att ge kommandot:
$>> W * ans$
$ans =$
1
3
5

Om ett givet ekvationssystem inte är entydigt lösbart beräknar MATLAB en approximativ lösning med hjälp av den s.k. minsta kvadratmetoden. Vi avstår från att gå in på detta här, men ger ett exempel med samma system som ovan.
Sätter vi $b = (1\ ,\ 0\ ,\ 0)$ så är systemet $W * x = b$ inte längre lösbart. Likväl levererar MATLAB ett svar :
$>> b = [1\ 0\ 0\ ]\ ';W\ \backslash b$
$ans =$
−1.3333
1.0833
Svaret är dock inte en exakt lösning, vilket man ser så här :
$>> W * ans$
$ans =$
0.8333
0.3333
−0.1667

Observera att om ett ekvationssystem har oändligt många lösningar så ger $A\backslash b$ bara en av dem, utan att varna för att det kan finnas flera lösningar. Man kan emellertid använda kommandot $rref$ för att få systemets utvidgade matris på radreducerad trappstegsform och sedan bestämma alla lösningar. Om vi t.ex. vill lösa $A * X = b$, där $A = [\ 1\ 1\ 1\ ;\ 1\ 2\ 3\ ]$ och $b = [\ 2\ 3\ ]'$ så ger
$>> A\backslash b$
enbart lösningen $(1.5, 0.0, 0.5)$. Om vi låter $U = [A\ b]$ vara den utvidgade matrisen, så ger
$>> rref(U)$
$ans =$
1 0 −1 1
0 1 2 1
Av detta kan man dra slutsatsen att systemets allmänna lösning är $(x, y, z) = (1, 1, 0) + t(1, -2, 1)$.
Pröva också $>> rrefmovie(U)$

6.3 Rader, kolonner och enskilda matriselement

Om A är en given matris betecknar $A(r, :)$ den r:te raden i A, $A(:, k)$ är den k:te kolonnen och $A(r, k)$ är elementet på plats (r,k). Om u och v är två vektorer med heltalskomponenter så betecknar $A(u, v)$ den undermatris av A som består av de rader vars index ges av vektorn u och vars kolonner är kolonnerna i A med index givna av vektorn v. Låt oss exemplifiera:
$>> A = [\ 11 : 15\ ;\ 21 : 25\ ;\ 31 : 35\ ]$
$A =$
11 12 14 13 15
21 22 23 24 25
31 32 33 34 35
$>> A(1, :)$
$ans =$
11 12 13 14 15
$>> A(:, 1)$
$ans =$
11
21
31
$>> A(3, 4)$
$ans =$
34
$>> B = A([2,\ 3], [1,\ 3,\ 5])$
$B =$
21 23 25
31 33 35
Man kan ändra enstaka element eller hela rader och kolonner genom att direkt tilldela dem nya värden. Exempel:
$>> A(3, 4) = 134$
$A =$
11 12 13 14 15
21 22 23 24 25
31 32 33 134 35
$>> A(:\ ,\ 5) = [1 : 3]\ '$

$A =$

11	12	13	14	1
21	22	23	24	2
31	32	33	134	3

Man kan ändra storlek på en matris med direkta tilldelningskommandon. Matrisen utökas eventuellt med tillägg av extra nollor. Exempel:
$>> B(4,:) = ones(1,3)$
$B =$

21	23	25
31	33	35
0	0	0
1	1	1

Man kan även bygga matriser med andra matriser som "block". Exempel:
$>> C = [[A\ ;\ (-1)\ .\hat{}(1:5)\ ], B]$
$C =$

11	12	13	14	1	21	23	25
21	22	23	24	2	31	33	35
31	32	33	34	3	0	0	0
−1	1	−1	1	−1	1	1	1

Ytterligare några användbara kommandon för att bygga upp nya matriser är

$tril(A)$	ger den undre triangulärmatrisen i den givna matrisen A
$triu(A)$	ger den övre triangulärmatrisen i A
$ones(n)$	nxn-matris med bara ettor
$ones(n,m)$	nxm-matris med bara ettor
$zeros(n)$	nxn-matris med bara nollor
$zeros(n,m)$	nxm-matris med bara nollor

Om man exempelvis vill addera talet 2 till alla element i matrisen C ovan, kan man ge kommandot
$>> C + 2. * ones(size(C))$
Matrisen $ones(size(C))$ är den matris av samma storlek som C som består av idel ettor.

6.4 Ett råd om ett bra arbetssätt

Det är praktiskt att skriva in matriser i en särskild textfil. För att göra det öppnar du ett editeringsfönster. Skriv in matrisen där, så här till exempel:
$A = [\ 1\ \ 1; 1\ \ -1; 0\ \ 0\ ];$
eller överskådligare

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 0 & 0 \end{bmatrix};$$

Spara sedan filen under något lämpligt namn, t.e.x "matrisen.m". Du kör sedan filen med kommandot $>>$ *matrisen*. Kontrollera att du har skrivit in rätt matris genom att ge kommandot $>> A$.
Du kan naturligtvis skriva in flera matriser och vektorer i filen "matrisen.m".

6.5 Invers och enhetsmatris

Inversen till en kvadratisk matris beräknas med kommandot *inv*. Med matrisen $A = [\ 1,\ 2\ 3,\ ;\ 4,\ 5,\ 6\ ;\ 7,\ 8,\ 0]$ ger kommandot
$>> C = inv(A)$
$C =$

−1.7778	0.8889	−0.1111
1.5556	−0.7778	0.2222
−0.1111	0.2222	−0.1111

Enhetsmatriser skrivs *eye* enligt följande:

$eye(n)$	enhetsmatris av typ nxn
$eye(n,m)$	nxm-matris med ettor i diagonalen och nollor för övrigt

Kontrollera genom att ge kommandot
$>> C * A - eye(3)$
Observera att MATLABs beräkning av inversen C är behäftad med ett visst numeriskt fel, vilket gör att svaret inte blir exakt noll, som det borde blivit.

6.6 Determinant, egenvärden och egenvektorer

Om A är en kvadratisk matris kan man beräkna dess determinant med hjälp av kommandot $det(A)$. Vi tar ett exempel:
$>> A = [\ 7,\ 2\ 0,\ ;\ 2,\ 6,\ -2\ ;\ 0,\ -2,\ 5\ ];$
$>> det(A)$
$ans =$
162

Egenvärden till A (reella och komplexa) med hjälp av kommandot $eig(A)$.
$>> eig(A)$
$ans =$
9.0000
6.0000
3.0000
Egenvärdena är alltså 9, 6, 3. Man kan även beräkna egenvektorer med hjälp av kommandot *eig*. Man skriver då $[V, D] = eig(A)$. Matrisen D är då en diagonalmatris innehållande egenvärden medan kolonnerna i matrisen V är motsvarande egenvektorer. Exempel:
$>> [V, D] = eig(A)$
$V =$

0.6667	−0.6667	0.3333
0.6667	0.3333	−0.6667
−0.3333	−0.6667	−0.6667

$D =$

9.0000 0 0
0 6.0000 0
0 0 3.0000

Här är alltså vektorn (0.6667 , 0.6667 , −0.3333) en egenvektor till egenvärdet 9. Kontrollera att matrisen V verkligen diagonaliserara A genom att ge kommandot $inv(V) * A * V$. Resultatet skall då bli D. MATLAB använder generella numeriska rutiner som endast ger närmevärden. För många matriser får man därför inte de exakta egenvärdena och egenvektorerna. Exempel:

$>> A = [2, 0, -2\ ;\ 1, 1, -2\ ;\ -2, 2, 1];$
$>> format\ long; eig(A)$
$ans =$
2.00000000000000
1.00000001924483
0.99999998075517

De exakta egenvärdena är (naturligtvis?) 2, 1, 1. Beräknar man egenvektorerna med hjälp av kommandot $[V, D] = eig(A)$ får man här resultatet (i $format\ short$):

$V =$
0.7071 0.6667 −0.6667
0.7071 0.6667 −0.6667
0.0000 0.3333 −0.3333

Observera att matrisen A inte är diagonaliserbar i detta exempel. Det är lätt att kontrollera för hand. Den exakta matrisen V är inte inverterbar, men eftersom MATLAB räknar med närmevärden uppfattas V trots allt som inverterbar. Därför kommer kommandot $inv(V)*A*V$ att ge svaret D (med sju korrekta decimaler). Man kan upptäcka att det är något skumt genom att låta MATLAB beräkna determinanten av V. Om denna determinant har ett mycket litet värde (här -1.5120e-09) måste man misstänka att den givna matrisen inte är diagonaliserbar.

Det finns flera andra numeriska rutiner för faktorisering av matriser. Se lu, svd, qr, $rref$.

6.7 Funktioner av matriser

Man kan beräkna potenser A^n av en given kvadratisk matris A med kommandot $A\hat{}n$. Här är n ett godtyckligt heltal, även negativt om A har invers. Om

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

kan man lätt beräkna matrisen

$$p(A) = a_n A^n + a_{n-1} A^{n-1} + \cdots + a_1 A + a_0 I$$

Vi tar ett exempel med polynomet $p(z) = z^2 + 2z + 3$ och en enkel 2x2-matris. Först matar vi in A och polynomets koefficienter :

$>> A = [\ 1,\ 2\ ;\ 3,\ 4\ ]; p = [\ 1,\ 2,\ 3\ ];$

Sedan beräknar vi $p(A)$ med hjälp av kommandot $polyvalm$ så här :

$>> polyvalm(p, A)$
$ans =$
12 14
21 33

Om A är en kvadratisk matris kommer kommandot $poly(A)$ att ger en vektor vars koordinater är koefficienterna i det karakteristiska polynomet för A med början med högstagradskoefficienten. Följaktligen kommer kommandot $roots(poly(A))$ att ge egenvärdena till matrisen A.

Det är också möjligt att beräkna mer komplicerade funktioner av en given kvadratisk matris A. Låt oss här bara nämna exponentialfunktionen

$$\exp A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \cdots + \frac{A^n}{n!} + \cdots$$

Den beräknas med hjälp av kommandot $expm(A)$ (inte $exp(A)$ som ger den elementvisa exponentialfunktionen). Observera att om c är en given kolonnvektor så är $x = expm(t * A) * c$ lösningen vid tiden t (ett givet tal) till systemet $x' = Ax\ ,\ x(0) = c$.

7 Programmering i MATLAB

7.1 Allmänna funktionsfiler

En funktionsfil kan ha ingen, en eller flera invariabler. En in-variabel kan vara ett tal, en rad- eller kolonnvektor eller en matris. På samma sätt kan funktionen ha ingen, en eller flera ut-variabler. Om funktionen skall ha flera utvariabler så skall detta anges i deklarationen genom att man räknar upp ut-variablerna inom parenteser [ ]. Nedan ges ett exempel på den allmänna utformningen av en funktion kallad *exempel* med fyra in-variabler och tre ut-variabler.

"exempel.m"

```
function [ut1,ut2,ut3]=exempel(in1,in2,in3,in4)
%      [u,v,w]=exempel(a,b,c,d)) beräknar
%      funktionsvärden u och v och w
%      av funktionen exempel i punkten (a,b,c,d).
beräkningar;
ut1=beräkningsresultat1;
ut2=beräkningsresultat2;
ut3=beräkningsresultat3;
```

7.2 Villkor och val

Den enklaste villkorssatsen ser ut så här:
if villkor
    instruktioner (satser) åtskillda av ;
end
Innebörden är att satserna bara utförs om villkoret är uppfyllt. Villkoren uttrycks med hjälp av jämförelser i vilka olika relationsoperatorer ingår, till exempel relationen <. Information om dessa får man med *help relop*. Här några exempel:
if a == b betyder "om a är lika med b"
if a~= b betyder "om a inte är lika med b"
if a >= b betyder "om a är större än eller lika med b"
if a > b betyder "om a är större än b"
if (a>0) & (b>0) betyder "om a>0 och b>0"
if (a>0) — (b>0) betyder "om a>0 eller b>0"
Man kan använda matriser i relationsoperationerna. Läs om detta med *help relop*.
Villkorssatser kan ha en mer komplicerad form, t.ex.
if villkor
    satser1;
else
    satser2;
end
Om villkoret är uppfyllt kommer de första satser1 att utföras. I annat fall utförs satser2.
Se vidare *help if*. Se också *any*, *all* m.fl. under *help ops*.

7.3 Slingor

En programslinga syftar till att upprepa en beräkning flera gånger. Den enklaste slinga görs med *for*. Programslingan ser ut så här i princip:
for variabel = vektor
    satser;
end
Den oftast förekommande slingan ser ut så här:
for k=1:n
   satser;
end
Här är n ett positivt heltal som antas specificerat från början. Resultatet av slingan blir att satserna kommer att utföras för $k = 1, 2, \cdots, n$.(Exempel kommer nedan).
En annan typ av slinga har formen
while villkor
    satser;
end
där satserna utförs så länge som villkoret är uppfyllt.

7.4 Inmatning och utmatning

Ett MATLAB-program, (alltså en funktion eller en script-fil) kan skriva ut text, data eller felmedelanden.
Kommandot *disp* används för att beordra utskrift av en matris eller text.
disp('Matrisen har egenvärdena');
disp([2 3]);
ger utskriften
Matrisen har egenvärdena
    2   3

Med kommandot *input* får man programmet att stanna och vänta på att användare av programmet skall mata in ett tal. Programraden
$a = input$('Mata in ett positivt tal !');
leder till att programmet skriver ut texten och väntar på användarens svar. Det inmatade talet blir sedan värdet på variabeln a.

Man kan också temporärt stoppa programflödet med *pause*. Programexekveringen fortsätter när användaren trycker på valfri knapp.

En programrad som innehåller kommandot *error* får MATLAB att skriva ut ett felmeddelande och gå ur programmet. Exempel
error('Du får inte mata in ett negativt tal, dummer!')

Kommandona *nargin* (number of arguments in) och *nargout* (number of arguments out) kontrollerar hur många in-variabler resp. ut-variabler som givits explicit. För exempel på användning se nedan.

Det finns ett antal ytterligare kommandandon för att styra hur program skall arbeta. För information se *help lang*.

7.5 Funktioner som in-variabler

In-variabler i en MATLAB-funktion kan vara filnamn på andra funktioner. För att innuti en funktion beräkna värdet av en annan funktion, vars namn är givet, använd kommandot $feval$. Låt oss ge ett exempel. Vi vill skriva en funktion vars uppgift är att beräkna summan av två andra funktioner. Man vill alltså beräkna $f(x)+g(x)$ där f och g är två funktioner vars namn är 'f_namn' resp. 'g_namn'.
"sumfg.m"
function s=sumfg(f_namn,g_namn,x)
% sumfg
%   s=sumfg(f_namn,g_namn,x)
%   Om f and g är två funktioner vars namn är

```
%      f_namn resp. g_namn, så är s = f(x) + g(x)
s = feval(f_namn,x)+feval(g_namn,x);
```
Denna funktion kan nu användas för att beräkna summan av godtyckliga funtioner, till exempel:

$>>\ x = 0 : 0.1 : 2 * pi;$

$>>\ y = sumfg('myFunk','cos', x); plot(x, y)$

7.6 Om effektiv programmering

Vi avslutar med några tips för att skriva effektiva MATLAB-program. Programslingor löper i princip långsamt i MATLAB. Därför bör man ta alla tillfällen att transformera slingor till vektor eller matrisoperationer. Vill man t.ex. beräkna $sin(n)$ för $n = 1, \ldots, 1000$ bör man inte skriva

```
n=0;
for k=0:999
  n=n+1;
  y(n)=sin(k);
end;
```

I stället bör man skriva slingan i vektoriserad form:

```
n=1:1000;
y=sin(n);
```

Om man ändå måste använda sig av programslingor kan det vara klokt att skapa minnesutrymme för slingans variabler genom att till exempel ge dem värdet noll. Exempel:

```
y=zeros(1,100);
for n=1:100
  y(n)=sum(x^n);
end;
```

Om man inte skapar minnesutrymme i förväg måste MATLAB utvidga vektorn y varje gång slingan genomlöps.

7.7 Om sökvägar och sånt

När MATLAB läser ett kommando som inte tillhör de inbyggda, så kommer MATLAB att leta bland filer som bär kommandots namn och slutar med ".m". MATLAB letar efter M-filer i följande ordning:

MATLABs programbibliotek.
nuvarande (aktuellt) bibliotek
ditt eget matlab-bibliotek (om du har ett sådant)

Observera att om det finns två M-filer med samma namn kommer MATLAB att använda den fil som hittas först.

Man får en lista över alla M-filer i aktuellt bibliotek genom att i matlab-fönstret ge kommandot *what*. Vill man byta bibliotek kan man skriva *cd* följt av bibliotekets namn (inklusive sökväg).

Det finns två typer av MATLAB-funktioner, dels s.k. inbyggda, dels sådana som definieras i M-filer. Exempel på en inbyggd funktion är *exp*, medan *sinh* ges i en M-fil. Skriver man *type* följt av funktionsnamnet (t.ex. *type sinh*) så får man antingen en utskrift av M-filen eller en information som talar om att funktionen är inbyggd.

Kommandot *path* ger en lista (inklusive sökvägar) över bibliotek där MATLAB söker efter filer. Programbibliotekets filer kan man fritt kopiera över till sitt eget matlab-bibliotek för inspektion och ev. modifikation.

7.8 Några programexempel

Exempel 1 : "myFunk2.m"

```
function y = myfunk2(x)
% myfunk2
%      y = myfunk2(x) beräknar y = exp(-x)-log(1+x)
%      Om inte alla x-värden är > −1 så
%      skrivs ett felmeddelande ut.
M=min(x);
if M <= −1
  error('Inget variabelvärde får vara mindre än 1!');
  % Funktionen är inte definierad om min(x) <= -1
else
  y=exp(-x)-log(1+x);
end
```

Exempel 2 : "mySum.m"

```
function s=mySum(z,maxN)
% mySum
%      s=mySum(z,maxN)
%      Beräknar den geometriska summan
%      s=1+z+z^2+...+z^n
%      för n=1,2,...,maxN och skriver ut summorna.
%      Stopp för varje nytt värde på n.
%      För att fortsätta tryck på godtycklig tangent.
%      Vill du avbryta tryck Ctrl-C
s=1;
for n=1:maxN
    s=1+z*s; n=n+1;
    disp('Antal termer och motsvarande summa');
    disp([n,s]); pause;
end
```

Exempel 3 : "myPowers.m"

```
% myPowers
%      Script-fil för att rita grafen av
%      funktionen y=x ^a
```

```
%      på intervallet (0,1)
%      där a matas in från tangentbordet
%      Potensen a måste vara positiv
%
%


continue=1;
% continue = 1 så länge som användaren vill fortsätta
a=1;
while continue==1
    a=input('Mata in en potens ');
    if a>=0
        x=0:0.05:1; y=x.^a; plot(x,y);
    else
        disp('Potensen måste vara positiv');
    end
    disp('Vill du fortsätta tryck 1, annars tryck 0');      % Användaren matar in 0 när han vill sluta.      continue=input('Tryck 1 eller 0 ');
    if continue==1
        hold on;
    end
end
hold off;
```

Exempel 4 : "plotpoly.m"

Detta exempel kräver en (begränsad) kunskap från avsnitt 6 om matriser.

```
function plotpoly(thePoly,linetype)
% PLOTPOLY
%    plotpoly(P) plots the polygon
%    defined by the 2xM-matrix P, where M > 2
%    using the linetype '-'
%    The matrix P holds the coordinates
%    of the vertices of the polygon in its columns
%    plotpoly(P,ltype) plots the same polygon
%    using the linetype ltype
%

[ N  M ] = size(thePoly);
if N==2 & M>2
    x=[thePoly(1,:) thePoly(1,1)];
    y=[thePoly(2,:) thePoly(2,1)];
    if nargin¡2
        plot(x,y,'-');
    else
        plot(x,y,linetype);
    end;
else
    error('Input skall vara en 2xM-matris med M>2');
end
```

7.9 Lista över de viktigaste kommandorubrikerna

Dessa huvudrubriker kommer man åt direkt genom att ge kommandot *help* följt av kommandorubriken, exemeplvis >> *help ops*.

Kommandorubrik	**Innehåll**	**Exempel**
general	allmänna kommandon	help, clear, load, save
ops	elementära matematiska operationer	+,*,.*,./,.
lang	programmeringskommandon	if, else, end, feval
elmat	grundläggande matriskommandon	zeros, ones, size
elfun	elementära matematiska funktioner	sin, exp, abs
matfun	matrisfunktioner	det, inv, rref, eig
datafun	funktioner för analys av data	min, max, sum
polyfun	polynom och interpolation	roots, polyval
funfun	nollställen, minimiering, integrering av funktioner	fmin, fzero
graph2d	tvådimensionell grafik	plot, axis, title
graph3d	tredimensionell grafik	mesh, surf
graphics	allmänna grafikkommandon	figure, clf, subplot
strfun	manipulation av strängar	eval, num2str
demos	demonstrationsfiler	demo, intro