En version av

Anders Holtsbergs Introduktion till Matlab och Maple

utan Maple.

av Dan Mattsson

Den här texten har tillkommit i stor hast. Som framgår av titeln är större delen stulet från Anders Holtsbergs *Introduktion till Matlab och Maple*[1] från 1999. Den har strukturerats om och försetts med en del nya exempel. Andra exempel har borttagits och avsnittet om Maple har utgått helt. De fel som finns i texten är med all sannolikhet mina.

Introduktion

När MatLab startar visas ett terminalfönster med en inmatrningsprompt. När handledningen visar exempel på in- och utmatning i terminalfönstret ser det ut så här:

```
>>  2 + 3
ans =
     5
```

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

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

Vill man undersöka lite vad MatLab kan göra så finns det en mängd färdiga program att köra. Dessa färdiga program kommer man åt genom att skriva `demo`. Man avslutar MatLab genom att skriva kommandot `quit` eller `exit`.

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

```
>>  2.3*1.2 - (7e-1)
ans =
    2.0600
>>  sin(2.1*pi)
ans =
    0.3090
```

Notera att decimalpunkt används och att talet $7{\cdot}10^{-1}$ skrivs `7e-1`. Förutom de vanliga räknesätten `+`, `-`, `*`, `/` och `^` finns även de vanligaste funktionerna fördefinierade i MatLab, till exempel

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

Funktionen `sqrt` är kvadratroten. Observera att `log` är den naturliga logaritmen och att de trigonometriska funktionerna tar indata i radianer.

Tecknet % inleder en kommentar och eventuell inmatning efter tecknet kommer inte att tolkas av MatLab.

```
>>  2+3 % + 13
ans =
     5
```

Variabler deklareras och tilldelas värden med operatorn `=` och finns sedan kvar i minnet:

```
>>  a = 1
a =
     1
>>  A = sqrt(36)
A =
     6
>>  b = pi
b =
    3.1416
>>  who
Your variables are:
A    a    ans  b
```

Kommandona `who` och `whos` visar vilka variabler som finns i minnet. Ett variabelnamn måste inledas med en bokstav men kan sedan även innehålla siffror och tecknet `_`. Om resultatet av en uträkning inte tilldelas någon namngiven variabel skapas variabeln `ans` innehållandes resultatet. Vissa namn är fördefinierade, t ex `pi` med värdet π och `i` (imaginära enheten). Andra namn är reserverade ord, t ex `exit`, `if`, och kan inte vara variabelnamn. Notera att MatLab gör skillnad på VERSALER och gemener i variabelnamn.

En variabel, t ex `a`, kan raderas ur minnet med `clear a`. Enbart `clear` rensar alla variabler ur minnet.

Om man vill ge flera kommandon på samma rad kan `,` användas.

```
>>  a = 1, A = sqrt(36), b = 3.1415, who
```

ger samma resultat som tidigare. Används istället `;` skrivs inget resultat ut.

```
>>  a = 1; A = sqrt(36); b = 3.1415;
```

utför samma beräkningar, men ger ingen utskrift.

I MatLab deklareras inte variablerna i förväg utan de skapas efter hand och kan ändra typer efter behov.

Övning 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.

Det mest användbara kommandot är `help`. Med det-

[1] finns på http://www.maths.lth.se/matstat/staff/andersh/mlabintr.ps

ta kan du se vilka kommandon/funktioner som är tillgängliga och hur de fungerar.

Övning 2. Använd kommandot `help` för se en lista av hjälpområden. Använd `more on` för att lägga in en paus efter varje helsida med text. Ange t ex `help elfun` för att se vilka elementära funktioner som är tillgängliga. Använd sedan `help` med ett funktionsnamn för att få hjälp om funktionen, till exempel `help help` för att se hur `help` fungerar.

Övning 3. Hur fungerar arcus sinus? Använd `help asin`. Vad händer om man beräknar $\sin^{-1}(x)$ då $x = 1.2$?

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

Övning 4. Finns det någon funktion eller operator som beräknar resten vid heltalsdivision? (Tips: ”rest” heter ”remainder” på engelska).

Man kan avbryta en programkörning genom att trycka 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.

Matriser

Det MatLab är speciellt bra på är att hantera och räkna med matriser. Man använder hakparenteser, `[` och `]`, för att deklarera matriser. En 2×3-matris $\boldsymbol{X}$ (en matris med 2 rader och 3 kolonner) kan skapas på flera olika sätt. Här följer tre ekvivalenta deklarationer:

```
>>  X = [8 3 4; 6 9 17]
X =
     8     3     4
     6     9    17

>>  X = [[8, 6]', [3, 9]', [4, 17]']
X =
     8     3     4
     6     9    17

>>  X = [
8 3 4
6 9 17
]
X =
     8     3     4
     6     9    17
```

Tal på samma rad separeras med blanksteg (eller kommatecken) och rader separeras med semikolon (eller returtecken). En radvektor är en matris med bara en rad och en kolonnvektor är en matris med bara en kolonn. Apostrof `'` används för att transponera en matris och omvandlar en kolonn/rad-vektor till en rad/kolonn-vektor[2].

I följande exempel byggs en matris upp av andra matriser. Här utnyttjas funktionerna `zeros`, `ones` och `eye` som ger en matris av nollor, en matris av ettor resp. en enhetsmatris.

```
>>  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
```

Tecknet `:` är en sekvensgenerator. Att skriva `a:b` ger vektorn $[a, a+1, a+2, \ldots, b]$. Om $b < a$ fås en tom lista.

```
>>  1:3
ans =
     1     2     3
>>  3:0
ans =
   Empty matrix: 1-by-0
>>  3:-1:0
ans =
     3     2     1     0
```

Med `a:h:b` erhålls den aritmetiska sekvensen från `a` till `b` med steglängd `h` som en radvektor.

Elementen i matrisen kan kommas åt med hjälp av vanliga parenteser, `(` och `)`. Om $\boldsymbol{Z} = [z_{ij}]_{i,j}$ så skrivs element z_{34} (rad 3, kolonn 4) som `Z(3,4)` i MatLab.

```
>>  Z(3,4)
ans =
     3
```

Listor kan användas för att referera till flera element samtidigt i en matris. `Z([1 2],1)` är en kolonnvektor med elementen z_{11} och z_{21}. `Z(2,[1 3])` är en radvektor med elementen z_{21} och z_{23}. `Z(1:4,1:2)` är en matris bestående av alla rader och kolonn 1 och 2. Önskas alla rader/kolonner kan gränserna utlämnas i sekvensgenereringen, `Z(1:4,1:2)` är samma sak som `Z(:,1:2)` och `Z(:,:)` är samma sak som `Z`.

Övning 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(:,:)
```

Man kan även modifiera en matris elementvis.

[2] Om matrisen har komplexvärda element så ger `'` det konjugerade transponatet, det vill säga den transponerade matrisen där alla element är komplexkonjugerade, medan operatorn `.'` ger enbart transponatet. För reella matriser är det ingen skillnad mellan dessa.

```
>>  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 6. Bilda en 7×5-matris med namn `Z` som bara innehåller nollor (använd funktionen `zeros`). Sätt därefter elementet i mitten och dess 8 närmaste grannar till 1.

De flesta fördefinierade elementära funktionerna i MatLab kan operera elementvis på vektorer (matriser). Om man skall beräkna $\sin(t)$ eller e^t för flera värden på t kan man deklarera en vektor med dessa värden,

```
t=[0 1 2 3 4 5 10 20 30 40 50 100]
```

och sedan anropa `sin(t)` resp `exp(t)`.

Övning 7. Använd vektorsyntaxen för att beräkna summan av logaritmen av alla udda positiva tvåsiffriga tal.

Statistikfunktionerna `mean` resp `std` beräknar medelvärde och stickprovsstandardavvikelse för talen i en vektor.

Övning 8. Använd funktionen `rand` för att skapa 100 stycken slumptal, likformigt valda på intervallet $[-10, 10]$. (Om U är likformigt fördelad på $[0, 1]$ så är $20U - 10$ likformigt fördelat på intervallet $[-10, 10]$.)
Beräkna sedan medelvärdet och stickprovsstandardavvikelsen för dessa tal.

För matriser opererar funktionerna `mean`, `std` och `sum` kolonnvis och returnerar en radvektor med medelvärden, stickprovsstandardavvikelser resp. kolonnsummor.

Övning 9. Skapa en 12×10000-matris $\boldsymbol{U} = [u_{ij}]_{i,j}$ med slumptal, likformigt valda på intervallet $[0, 1]$. Glöm inte `;` vid tilldelningen!
Skapa en radvektor $\boldsymbol{z}$ av längd 10000 där elementen

$$z_j = \left(\sum_{i=1}^{12} u_{ij} \right) - 6, \quad j = 1, \ldots, 10000.$$

Rita upp ett histogram över de erhållna z-värdena med `hist(z,50)`. (Se `hist`.)

De vanliga räkneoperationerna $+$, $-$ och $\cdot$ är definierade även för matriser. Addition och subtraktion av matriser är som vanligt definierade elementvis. Nu skall vi istället betrakta matrismultiplikation, `*`, och, vad som är speciellt för MatLab, matrisdivision, `/` och `\`. Dessa tre operationer har även elementvisa motsvarigheter, `.*`, `./` resp. `.\`.

Vi skapar två matriser, $\boldsymbol{A}$ och $\boldsymbol{B}$, 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 kommuterar inte, $\boldsymbol{AB}$ är inte samma sak som $\boldsymbol{BA}$.

```
>>  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 $\boldsymbol{A}$ med motsvarande element i matrisen $\boldsymbol{B}^T$ använder man operatorn `.*` såhär:

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

Notera att både $\boldsymbol{A}$ och $\boldsymbol{B}^T$ har samma storlek (2×3-matriser), en förutsättning för att elementvis multiplikation skall fungera.

Det finns en även exponentiering, `^`, och där gäller samma sak. `A^2` är samma sak som `A*A`, medan `A.^2` betyder att varje element i $\boldsymbol{A}$ skall kvadreras.

Låt oss illustrera de olika divisionsoperatorerna med de tre matriserna $\boldsymbol{A}$, $\boldsymbol{B}$ och $\boldsymbol{C}$:

```
>>  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`. Om $\boldsymbol{AB} = \boldsymbol{C}$ så är $\boldsymbol{ABB}^{-1} = \boldsymbol{CB}^{-1}$ eller $\boldsymbol{A} = \boldsymbol{CB}^{-1}$ eftersom $\boldsymbol{BB}^{-1} = \boldsymbol{I}$, enhetsmatrisen. I MatLab får vi

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

Notera att för reella tal gäller att då $B \neq 0$ och $AB = C$ så är $A = C/B$. För matriser är inte division definierat i matematisk text, men är det i MatLab.

```
>>  C/B
ans =
     1     2
     5     3
```

Kommandona `C*inv(B)` och `C/B` är konceptuellt desamma men av beräkningstekniska och numeriska skäl bör man använda det sistnämnda.

På liknande sätt kan man erhålla $\boldsymbol{B}$ som $\boldsymbol{A}^{-1}\boldsymbol{C}$, dvs

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

eller med vänsterdivision

```
>>  A\C
ans =
     5     2
     3     1
```

Notera att då funktionen `inv` användes presenteras svaren med ett antal nollor vilket beror på att avrundning under beräkningarna har gjort att den sista signifikanta siffran inte är noll, medan motsvarande numeriska lapsus inte återfinns då `/` eller `\` användes.

I tillämpningar är det ofta ekvationssystem på formen $\boldsymbol{Ax} = \boldsymbol{b}$ som skall lösas. Enligt det ovanstående erhålls lösningen med $\boldsymbol{x} = \boldsymbol{A}\backslash\boldsymbol{b}$. Antag nu att ekvationsystemet $\boldsymbol{Ax} = \boldsymbol{b}$ är överbestämt, dvs det finns fler rader än kolonner i matrisen $\boldsymbol{A}$ (och ingen kolonn i $\boldsymbol{A}$ kan skrivas som en linjärkombination av de övriga). Då saknar ekvationssystemet i allmänhet lösning. Minstakvadratlösningen till ekvationssystemet är det $\boldsymbol{x}$ som minimerar avståndet mellan $\boldsymbol{b}$ och $\boldsymbol{Ax}$ (med avstånd menas här normen $||\boldsymbol{b}-\boldsymbol{Ax}||$). Lösningen ges av de så kallade normalekvationerna, vilka i matrisformulering fås genom att vänstermultiplicera med $\boldsymbol{A}^T$,

$$\boldsymbol{A}^T\boldsymbol{Ax} = \boldsymbol{A}^T\boldsymbol{b}.$$

Lösningen av detta är således $\boldsymbol{x} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}$. I MatLab skulle koden för detta bli kort, `x = inv(A'*A)*A'*b` eller `(A'*A)\(A'*b)`. Dock, koden `A\b` ger minstakvadratlösningen till $\boldsymbol{Ax} = \boldsymbol{b}$ i fallet att ekvationssystemet är överbestämt. Man kan se den vanliga lösningen som ett specialfall av minstakvadratlösningen i fallet då $\boldsymbol{A}$ är kvadratisk med full rang.

Övning 10. Lös ekvationssystemet

$$\begin{cases} 3x+2y-z &= 0 \\ 2x+-y+z &= 1 \\ x+y+z &= 2 \end{cases}$$

genom att skriva det på formen $\boldsymbol{Ab} = \boldsymbol{c}$ där $\boldsymbol{c} = (0\ 1\ 2)^T$, $\boldsymbol{b} = (x\ y\ z)^T$ och

$$\boldsymbol{A} = \begin{bmatrix} 3 & 2 & -1 \\ 2 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Övning 11. Med

```
x = (1:10)';
y = 2 + 5*x  + 5*randn(10,1);
```

bestäm minsta-kvadratlösningen (α, β) till

$$\begin{cases} \alpha+\beta x_1 &= y_1 \\ \alpha+\beta x_2 &= y_2 \\ &\vdots \\ \alpha+\beta x_n &= y_n. \end{cases}$$

Skalärprodukten (inre produkten) $\boldsymbol{u}^T\boldsymbol{v}$ mellan två kolonnvektorer skrivs `u'*v`, och yttre produkten `u*v'`. För normen $||\boldsymbol{u}|| = \sqrt{\boldsymbol{u}^T\boldsymbol{u}}$ finns funktionen `norm`.

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.

Teckensträngar

Teckensträngar anges inom apostroftecken, alltså `'`, och inte inom citationstecken `"` som annars är brukligt.

```
>>  s = 'kålrot';
```

Variabeln `s` är nu i själva verket en radvektor bestående av tal, koderna för 'k', 'å' o s v., men MatLab håller koll på att dessa värden skall tolkas som tecken. Kommandot `whos s` avslöjar att `s` är en `char array`, alltså en character array eller teckensträng. För att se koderna kan man addera 0 eller tala om för MatLab att koderna skall tolkas som numeriska värden med funktionen `double`.

Övning 12. Prova detta. Deklarera `s` som en sträng innehållande text. Evaluera sedan `s`, `s+0` och `double(s)`.

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

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

Notera blanksteget efter ordet "god".

Boolska uttryck och matriser

I MatLab kodas falskt som 0 och sant som 1. Antag att vi vill plocka ut alla tal mindre än 0.4 av 4 stycken simulerade slumptal.

```
>>  x = rand(1,4)
x =
    0.9501    0.2311    0.6068    0.4860
>>  k = x<0.4
k =
     0     1     0     0
```

Resultatet av `x<0` är en vektor av samma längd som

x med (boolska) värden ettor och nollor (sant och falskt).

```
>>  whos k
Name  Size  Bytes  Class

k      1x4     32  double array (logical)

Grand total is 4 elements using 32 bytes

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

En logisk (boolsk) vektor kan användas för att indexera en annan vektor. Om den logiska (boolska) vektorn är sann (1) så väljs motsvarande element ut, annars inte. Vi kan således skriva `x(k)` eller `x(x<0.4)` för att erhålla en vektor bestående av de element i `x` som är mindre än 0.4.

Övning 13. Jämför reslutatet av `k` med `k+0`. Varför fungerar `x(k)` men inte `x(k+0)`?

En logisk vektor skapas av någon av jämförelseoperatorerna

```
<   <=   ==   =>   >   ~=
```

(där `==` är LIKHET och `~=` är OLIKHET) eller de logiska operatorerna

```
&   |   ~
```

OCH, ELLER samt NEGATION. Samtliga operatorer fungerar elementvis på vektorer och matriser. Uteslutande eller (symmetrisk differens) heter `xor` i MatLab.

Exempel: Antag att vi vill ta bort de element i en vektor `v` som är negativa eller större än 2. Då är `v(v<0 | v>2)` de element som skall tas bort och vi kan antingen skriva

```
>>  v(v<0 | v>2) = []
```

eller

```
>>  v = v(v>=0 & v<=2)
```

Komplexa tal

Följande exempel illustrerar hur MatLab hanterar komplexa tal.

```
>>  sqrt(-1)
ans =
        0 + 1.0000i
>>  i
ans =
        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
>>  z*conj(z)
ans =
    74
```

Notera att `i` var fördefinierad som den imaginära enheten.

Grafritning

Figur 1 är skapad på följande sätt. Först ritas den heldragna kurvan:

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

Vektorn `x` skapades som en vektor från 0 till 7 i steg om 0.1. I vektorn `f` lagras sinus av motsvarande element i `x`. Funktionen `plot` ritar en kurva där de 71 punkterna (x_i, f_i), $i = 1, \ldots, 71$, är förbundna med linjer. När en kurva skall ritas så gäller det att att välja lagom många punkter, inte för få så att grafen blir hackig, och inte för många så att det tar för mycket minnesutrymme och tid.

Övning 14. Experimentera med vad som händer med grafen om den ritas för `x = 0:7;`, `x = 0:0.1:7;` resp. `x = 0:0.01:7;`.

I nästa steg ritas den streckade kurvan:

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

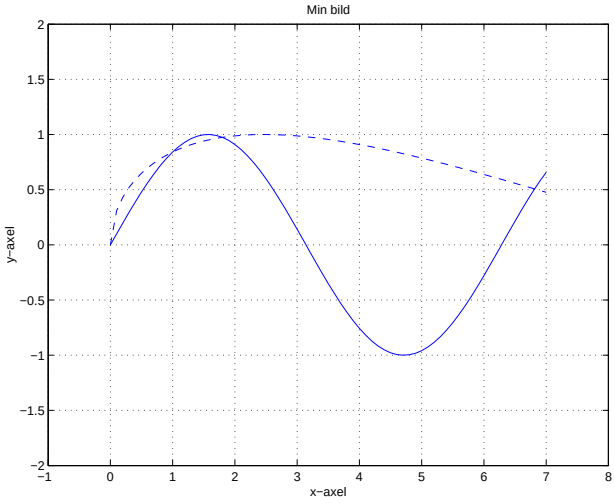
Funktionen `hold` används för att inte den heldragna kurvan skall försvinna när vi ritar den streckade. Funktionen `plot` ges ett tredje argument, teckensträngen `'--'`, som talar om hur kurvan skall ritas, här streckad. Andra möjligheter är att rita med olika färger och olika plotsymboler och linjetyper, se `help plot`.

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

Axlarna justerades, med `axis` (se `help axis`), och ett stödraster lades till med `grid on`.

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

Det tillhör god ton att ha texter på axlarna och en titel på figuren, vilket ordnades med `xlabel`, `ylabel` och `title`. Nu är bilden klar.



Figur 1.

Om man söker koordinaten för en punkt i bilden kan man använda `ginput(1)` och klicka i bilden med musen. Se `help ginput` om man önskar fler punkter. Man kan också lägga till text på lämpligt ställe i bilden med `gtext`. Prova t ex `gtext('sin(x)')`.

Övning 15. 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('Funktionen sin(x)')
```

och sedan klicka någonstans i bilden.

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

Övning 16. Ge kommandot `zoom` eller klicka på förstoringsglaset i figurfönstret. Zooma in på en punkt genom att klicka på punkten. Zooma in på ett område genom att hålla ned musknappen och flytta markören en bit och sedan släppa knappen. Se till att få med något intressant i rutan som syns. Zooma ut med högerknappen.
Använd `zoom`-funktionen för att approximativt bestämma lösningen till $\sin(x) = 0.2$ då $0 < x < 1$.

Det är lätt att plotta flera grafer i figurfönstret. Man använder funktionen `subplot(m,n,p)` som delar in figurfönstret i $m \times n$ grafer och sätter aktuell graf till nummer p, numrerade från vänster till höger uppifrån och ned. Parametern p kan även vara en vektor med nummer av delgrafer.

Övning 17. Pröva följande kommandon och försök förstå vad som händer.

```
x = 0:50;
y = sqrt(x);
z = rand(1,length(x));

subplot(2,2,[1 2])
plot(x,y)

subplot(2,2,3)
plot(x,z,'o')

subplot(2,2,4)
plot(x,y+z,'.-r')
grid on
```

För att öppna fler figurfönster kan man använda `figure`.

För att skriva ut en figur på papper kan man ge kommandot `print` eller välja ”Print” från menyn ”File” i figurfönstret. Med kommandot `print` kan man även spara figuren i en fil, bland annat i filformaten postscript, jpeg och tiff. Exempelvis

```
print -dtiff minfigur.tiff
```

sparar aktuell figur som en tiff-fil och

```
print -dpsc2 -f2 minfigur.ps
```

sparar figur 2 som en postscriptfil (level 2 i färg), vilken man sedan kan skicka till en skrivare.

Spara och återskapa

Kommandot `save` sparar alla variabler i filen `matlab.mat` i aktuell mapp (eventuell tidigare sparad fil med samma namn skrivs över). För att få tillbaka variablerna använder man kommandot `load`. Genom att ange ett explicit filnamn, t ex `save minfil3` vilket sparar variablerna i filen `minfil3.mat`, kan man skapa flera MatLab-filer i samma mapp . Om man vill spara bara vissa variabler kan man t ex skriva `save minfil5 a x` för att enbart spara `a` och `x`. De skapade filerna hamnar i MatLabs aktuella mapp. Man byter aktuell mapp med kommandot `cd` följd av önskat mappnamn. Kommandot `pwd` visar aktuell mapp. Kommandot `what` listar vilka MatLab-filer som finns i aktuellt mapp medan kommandot `dir` eller `ls` visar alla filer i mappen. Prova både `dir`, `ls` och `what`. Kommandot `mkdir` skapar en ny underkatalog i aktuell mapp.

Övning 18. Ta reda på vad aktuell mapp heter. Skapa en underkatalog med namnet ”matlab” i din hemkatalog. Gör så att den nysskapade mappen blir aktuell. Titta efter vilka variabler som finns i arbetsminnet med `who`. Spara variablerna i en namngiven fil och kontrollera att filen skapats. Radera alla variabler från arbetsminnet (`clear`) och kontrollera därefter att arbetsminnet är tomt. Ladda tillbaka variablerna. Verifiera att de finns igen.

Med kommandot `load` kan man även läsa in en fil med numeriska värden givna i textformat. Filen måste då definiera en matris av tal och den i MatLab skapade variabeln får samma namn som filnamnet. Exempelvis, en fil `mydata.txt` med innehållet

```
8 9 10
7 -1.2 83
4e-2 0 0
```

läses in i MatLab med:

```
>>  load mydata.txt
>>  mydata
mydata =
    8.0000    9.0000   10.0000
    7.0000   -1.2000   83.0000
    0.0400         0         0
```

Man kan även med `save` spara textfiler. Se `help save` för information.

MatLabkommandot `type` visar innehållet av en textfil precis som Unix kommandot `cat`. För att lägga in en paus vid varje skärmsida kan man använda `more on`.

Programkonstruktion

MatLabprogram är filer med MatLabkod. Filnamnet skall sluta med ”.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 inleds alltid med det reserverade ordet `function`.

Exempelvis, filen `mittprog.m` i aktuellt bibliotek innehåller texten

```
U = rand(1,100);
B = -5*log(U);
medel = sum(B)/length(U)
```

Exekvera kommandona i filen genom att skriva

```
>>  mittprog
medel =
    5.7601
```

Man kan editera filen med `edit mittprog.m` och betrakta innehållet med `type mittprog.m`.

Exempel på egna funktioner kommer i nästa stycke.

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
```

Varje `while`, `if` och `for` avslutas med ett `end`.

Ett så långt exempel som det ovan skrivs egentligen bäst i en fil med en editor, men ibland är det bra att skriva korta programslingor direkt i kommandofönstret.

Man kan bryta en `for` eller `while`-slinga med `break` och man kan hoppa direkt till avbrottsvillkoret med `continue` precis som i Java eller C. En flervägs hoppinstruktion finns också, se hjälptexten till `switch`.

Funktioner

Betrakta följande egna funktion som finns i en fil med namnet `randffg.m`

```
function x = randffg(p,m,n)
%RANDFFG Genererar ffg-fördelade
%observationer.
%
% randffg(p,m,n)
%
% Returnerar en (m,n)-matris med slumptal
% från en ffg(p)-fördelning.

if (prod(size(p))) ~= 1
  disp('p skall vara en skalär')
  return
end

if (p<=0 | p>=1)
  disp('p skall vara mellan 0 och 1')
  return
end

if nargin<2, m=1; end
if nargin<3, n=1; end

lambda = -log(p);
x = 1+floor(-rand(m,n)/lambda);
```

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 1, 2 eller 3 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 skall stämma överens med filnamnet. (I själva verket bryr sig MatLab bara om filnamnet men man skall ju inte förvirra sig själv eller andra)

- Variabler deklarerade i funktionen syns inte utanför densamma. Funktionsanropet skapar inte variabeln `lambda`.

Det bästa sättet att lära sig hur man skriver bra MatLabprogram är att studera kod som finns.

Övning 19. Skriv ut hjälptexten till funktionen `mean` på skärmen. Skriv sedan ut hela programkoden.

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.

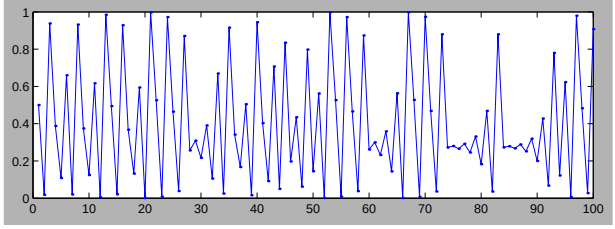
Programmeringstips

Använd `tic` och `toc` för att klocka exekveringstid för ett skript eller en funktion. Kommandot `tic` nollställer en intern klocka och `toc` visar hur lång tid (i sekunder) som förflutit sedan nollställningen.

Övning 20. Baserat på ett startvillkor x_1 och en konstant a beräknas $x_{n+1} = (1 - ax_n)^2$ för $n \geq 1$. Tiden det tar att beräkna de 100 första värdena då $x_1 = 0.5$ och $a = 1.7$ kan mätas med

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

Följande figur erhålls som reslutat av `plot`.



Processen i övningen uppvisar ett närmaste kaotiskt beteende.

Övning 21. Genom att modifiera programmet ovan, bestäm tiden det tar att beräkna de 100000 första värdena för denna process. Utelämna då `subplot` och `plot`.
Notera hur lång tid koden tog att köra.

Eftersom vi vet att $\boldsymbol{x}$ kommer att bli en vektor av längd n kan vi förallokera utrymme för denna. På så vis slipper MatLab flytta data allt eftersom vektorn växer.

Övning 22. Kör programmet

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

och notera hur lång tid koden tog att köra.

Vektorisering är att använda matrisoperationer i stället för programslingor.

Övning 23. Antag att vi har en kolonnvektor (med längden n) och vill skapa en ny genom att multiplicera varje element med dess index i vektorn. Ett kodförslag som liknar det man skulle skrivit i ett konventionellt programspråk skulle kunna se ut såhär (där vi har lagt till `tic` och `toc` för att mäta tiden)

```
tic, for i=1:n, x(i)=x(i)*i; end, toc
```

Gör detta! Jämför sedan detta med

```
tic, x = x.*(1:n)'; toc
```

För att undvika villkorssatser när man vektoriserar funktioner kan man använda boolska variabler.

Övning 24. Antag att vi i en vektor $\boldsymbol{v}$ vill multiplicera element med motsvarande element i vektorn $\boldsymbol{u}$ ifall elementet i $\boldsymbol{u}$ är mindre än ett, annars dividera med det.
Skapa $\boldsymbol{u}$ och $\boldsymbol{v}$ med

```
n = 100000;
v = rand(n,1);
u = 2*rand(n,1);
```

Gör en tidsjämförelse med `tic` och `toc` mellan programmen

```
for i=1:n
  if u(i)<1
    v(i) = v(i)*u(i);
  else
    v(i) = v(i)/u(i);
  end
end
```

och

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

Övertyga först dig själv om att programmen gör samma sak! Minns att de logiska operatorerna ger vektorer av 0:or och 1:or för falska resp. sanna villkor.