Matematik med MATLAB

1 Matrishantering

Detta avsnitt förutsätter att läsaren är bekant med den linjära algebrans mest grundläggande begrepp (t.ex. matrismultiplikation, matrisinvers och system av linjära ekvationer). Hjälp till detta kapitel hittar du i `helpwin` under `matlab/ops`,`matlab/elmat` och `matlab/matfun`. I *Pärt-Enander, Sjöberg: Användarhandledning för MATLAB*, se kapitel 5-6.

1.1 Grundläggande matrisoperationer

Man matar in matriser radvis med mellanslag eller kommatecken mellan radelementen och med semi-kolon eller tryck på return/enter-tangenten mellan raderna.

Exempel 1: Om man i Matlabs kommandofönster skriver

```
>>M=[1 2 3; 4 5 6; 7 8 0]
```

så erhålls resultatet

```
M =
   1   2   3
   4   5   6
   7   8   0
```

□

En radvektor (radmatris) med konstant differens h mellan elementen kan inmatas enligt mönstret: `R=[a:h:b]`.

Exempel 2:

```
>>R=[3:2:15]
```

ger resultatet

```
 R=
   3   5   7   9   11   13   15
```

Om ingen differens anges, sätts den till 1:

```
>>S=[3:9]
```

ger resultatet

```
S=
   3   4   5   6   7   8   9
```

□

Addition, subtraktion och multiplikation av matriser betecknas med `+`, `-` respektive `*`.

Exempel 3: Om vi skriver

```
>>A=[1 2 4; 3 7 2], B=[4 7 8;-1 3 2]
```

så returnerar MATLAB

```
A =
   1   2   4
   3   7   2
B =
     4   7   8
    -1   3   2
```

Vi kan sedan beräkna summan

```
>>S=A+B
```

med resultatet

```
S =
   5    9   12
   2   10   10
```

(Notera att addition utförs elementvis.)
Med M som ovan i exempel 1 så kan vi bilda matrisprodukten AM (elementet på plats (i,j) ges av skalärprodukten mellan i:te raden i A och j:e kolonnen i M):

```
>>P=A*M
```

ger resultatet

```
P =
   37   44   15
   45   57   51
```

□

Övning 1: Definiera egna matriser A och B av typ 2×3,
C av typ 3×3, D av typ 1×3 samt F och G av typ 3×1.

a. Testa kommandot *size* på några av matriserna. ($size(A)$)

b. Beräkna $A + B$, $-A$, $2A$ och $3A - 5B$.

c. Försök beräkna $A + C$.

d. Beräkna AC.

e. Beräkna AG.

f. Försök beräkna AB.

g. Jämför DG och GD.

h. Beräkna C^2 och C^{10}.

□

1.2 Transponering och inversmatris

Symbolen för *transponering* är `'`. Vid transponering av en matris så byter rader och kolonner plats. T.ex. så blir en radvektor efter transponering en kolonnvektor,

Exempel 4:

```
>> x=[-1 0 2]'
```

ger alltså resultatet

```
x =
   −1
    0
    2
```

□

Vissa kvadratiska matriser A är inverterbara, vilket innebär att det finns en matris C sådan att $AC = CA = I$ (identitetsmatrisen). I MATLAB beräknas inversen (betecknas A^{-1}) med kommandot $inv(A)$.

Exempel 5: Med M som i exempel 1 ovan får man dess invers $N = M^{-1}$:

```
>> N=inv(M)
N =
```

$$\begin{matrix} -1.7778 & 0.8889 & -0.1111 \\ 1.5556 & -0.7778 & 0.2222 \\ -0.1111 & 0.2222 & -0.1111 \end{matrix}$$

Om man skriver `format rat` i kommandofönstret så skrivs sedan alla tal som bråk istället för som decimaltal. (Du kan återställa till det vanliga formatet genom att skriva `format short`.)

```
>> format rat, N=inv(M)    (det är tillåtet att ge flera kommandon på samma rad!)
N =
```

$$\begin{matrix} -16/9 & 8/9 & -1/9 \\ 14/9 & -7/9 & 2/9 \\ -1/9 & 2/9 & -1/9 \end{matrix}$$

□

Multiplikation med inversmatris från höger respektive vänster kan skrivas kortare med bråkstreck (slash och backslash): M/A respektive $A\backslash N$ i stället för $M * inv(A)$ respektive $inv(A) * N$.

Övning 2: Låt A och G vara matriserna ur uppgift 1.

a. Beräkna A^t (transponatet av A). Jämför med A.

b. Inför en tredje rad i A genom att sätta den lika med G^t. Låt A vara den nya matrisen.

c. Beräkna A^{-1}. Om den skulle sakna invers, ändra något av elementen i A!

d. Testa om vi verkligen har fått inversen till `A`.

e. Jämför $(A^t)^{-1}$ och $(A^{-1})^t$.

f. Testa M/A och $A\backslash N$, där M och N är lämpligt valda matriser.
Jämför med $M * inv(A)$ respektive $inv(A) * N$.

□

1.3 Rader, kolonner och enskilda matriselement. Nya matriser av gamla.

Om M är en given matris betecknar $M(r,:)$ den r:te raden i M, $M(:,k)$ är den k:te kolonnen och $M(r,k)$ är elementet på plats (r,k) i matrisen M. Notera att om $size(M) = [m,n]$ så innehåller listan $m \cdot n$ element. Om u och v är två vektorer med heltalskomponenter så betecknar $M(u,v)$ den undermatris av M som består av de rader vars index ges av vektorn u och vars kolonner är kolonnerna i M med index givna av vektorn v.

Exempel 6:

```
>> M=[11:15;21:25;31:35]
M =
```

```
   11   12   13   14   15
   21   22   23   24   25
   31   32   33   34   35
>> M(1,:)    (ger första raden i M)
ans =        (ans står för det senaste svaret)
   11   12   13   14   15
>> M(:,2)     (ger andra kolonnen i M)
ans =
   12
   22
   32
>> M(3,4)
ans =
   34
>> N=M([2 3],[1 3 5])
```

här plockas element i raderna 2 och 3 ut men endast från kolonnerna 1, 3 och 5

```
N =
   21   23   25
   31   33   35
```

□

Man kan ändra enstaka element eller hela rader och kolonner genom att direkt tilldela dem nya värden.

Exempel 7:

```
>> M(3,4)=134
```

här ersätts elementet på plats $(3, 4)$ med talet 134

```
M =
   11   12   13    14   15
   21   22   23    24   25
   31   32   33   134   35
>> M(:,5)=[1:3]'
```

kolonn 5 ersätts med kolonnen $[1 \quad 2 \quad 3]^T$

```
M =
   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 8:

```
>> N(4,:)=[1 1 1]
```

med `N` ovan som i exempel 6 ger detta

```
N =
   21   23   25
   31   33   35
    0    0    0
    1    1    1
```

(N ovan har två rader men ”tvingas” här till fyra, trots att man endast angivit sista raden; resterande fylls automatiskt med nollor.) □

Man kan även bygga matriser med andra matriser som ”block”.

Exempel 9:

```
>> P=[[M;2*[1:5]] N]
P =
   11   12   13   14    1   21   23   25
   21   22   23   24    2   31   33   35
   31   32   33  134    3    0    0    0
    2    4    6    8   10    1    1    1
```

□

Man kan också göra om formen på en $m \times n$-matris med $reshape(M, r, k)$, under förutsättning att $r \cdot k = m \cdot n$. $M(:)$ gör om M till en kolonnmatris.

1.4 Speciella matriser

Det finns ett antal kommandon som är avsedda för att bygga upp nya matriser.
Här följer ett urval:

`zeros(n)`	$n \times n$-matris med bara nollor
`zeros(n,m)`	$n \times m$-matris med bara nollor
`eye(n)`	enhetsmatris av typ $n \times n$
`eye(n,m)`	$n \times m$-matris med ettor i diagonalen och nollor för övrigt
`ones(n)`	$n \times n$-matris med bara ettor
`ones(n,m)`	$n \times m$-matris med bara ettor
`rand(n)`	$n \times n$-matris vars element är tal mellan 0 och 1, slumpmässigt utvalda.
`rand(n,m)`	$n \times m$-matris vars element är tal mellan 0 och 1, slumpmässigt utvalda.

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
`diag(V)`	ger, om V är en rad- eller kolonnmatris, en matris med V på diagonalen och 0 på övriga platser.
`diag(A)`	ger, om A är en matris, en kolonnvektor med diagonaleelementen i A som element.

Det finns även ett antal speciella matriser som den nyfikne läsaren kan inspektera, dessa hittar du i `helpwin` under `matlab/elmat Specilized matrices`

Övning 3: Stora matriser byggda av mindre block.
Bygg en valfri `6x8`-matris med hjälp av de tidigare matriserna `A-G`, specialmatriser ur detta avsnitt och de flesta av metoderna som ges i föregående avsnitt. □

2 Ekvationssystem

Betrakta ett linjärt ekvationssystem `A*X = b`, där `A` är en $m \times n$-matris, `X` en $n \times 1$-matris (n obekanta) och `b` en $m \times 1$-matris (högerled, m ekvationer). Systemet kan alltid lösas med hjälp av Gauss-elimination i ekvationssystemet tills koefficientmatrisen är trappstegsformad. Ofta väljer man att presentera räkningarna utan att ta med de obekanta. Man bildar då systemets *totalmatris* eller *utökade koefficientmatris* som består av koefficientmatrisen

följd av högerledet, ofta separerade med ett lodrätt streck för att förtydliga att det handlar om en totalmatris till ett ekvationssystem. På denna matris gör man sedan elementära radoperationer tills koefficientmatrisen är trappstegsformad, därefter kan man enkelt lösa ut de obekanta ev. på parameterform, eller se att systemet saknar lösning. I Matlab erhålls den reducerade trappstegsformen genom att man bildar systemets totalmatris `A1 = [A b]` och sedan beräknar `T = rref(A1)`, där `rref` är förkortning av *row reduced echelon form* = *radreducerad trappstegsform*. Alla pivotelement i trappstegsmatrisen `T` är då ettor, och alla element ovanför och under är nollor. Ur denna matris `T` kan man bestämma `X`, om lösning finns. Det finns en pedagogisk variant av detta kommando: `rrefmovie`. Man kan då genom tangenttryckningar steg för steg studera eliminationsprocessen. MATLAB utför s k pivotering: före varje eliminationssteg ser man till genom erfordeliga radbyten, att det pivotelement som står på tur har maximalt belopp. Därigenom hålls beräkningsfelen nere.

Om matrisen `A` är kvadratisk och inverterbar, så har systemet entydig lösning `X=inv(A)*b`, vilket som vi tidigare sett också kan skrivas `X=A\b`. Om matrisen `A` inte är inverterbar (men fortfarande kvadratisk), så finns ingen eller oändligt många lösningar. Kommandot `X=A\b` ger dock ett svar, tillsammans med en varning. Då bör man övergå till `rref`.
Om `A` inte är kvadratisk så är `X=A\b` en lösning till `A*X=b` med minstakvadratmetoden (som behandlas senare i kursen i linjär algebra). Det innebär att även om systemet saknar lösning så får man alltså ett svar ("det närmaste man kan komma en lösning").

I *Pärt-Enander, Sjöberg: Användarhandledning för MATLAB*, kapitel 7, finns ytterligare information om behandling av linjära ekvationssystem med MATLAB.

Exempel 10: Med

```
>>A = [1 2 3;4 5 6;7 8 0], b=[5;8;-7]
```

erhålls

```
A =
   1   2   3
   4   5   6
   7   8   0
b =
     5
     8
    -7
```

Lösningen till `A*X = b` ges av `inv(A)*b` (eller alternativt `A\b`)

```
>>X = inv(A)*b
X =
    -1
     0
     2
```

Vi kan också lösa ekvationssystemet via radreducering av `[A b]`,

```
>>U = rref([A b])
U =
   1   0   0  -1
   0   1   0   0
   0   0   1   2
```

ur vilken vi direkt kan utläsa samma lösning som ovan. □

Exempel 11: Om vi skriver
```
>>A = [1 2 3;-2 4 1;3 -2 2]; b = [4 5 2]';
```
så skapas matriserna `A` och `b` i, men p.g.a att kommandona avslutas med semikolon så skriver MATLAB inte ut resultatet av inmatningen (vilket ju inte heller alltid är önskvärt!)
Nu ger
```
T = rref([A b]), format rat
T =
   1   0   5/4   0
   0   1   7/8   0
   0   0     0   1
```
Den sista raden ger oss ekvationen `0 = 1` vilket visar att ekvationssystemet `A*X = b` saknar lösning. Däremot ger
```
>>X = A\b
X =
   1.0e + 15*

        4.2221
        2.9555
       -3.3777
```
Här ges emellertid en varning som säger oss att detta kanske är fel.

Om vi ändrar högerledet till
```
>>b = [4 5 -1]'
b =
     4
     5
    -1
```
så ger
```
>>T = rref([A b])
T =
   1   0   5/4    3/4
   0   1   7/8   13/8
   0   0     0      0
```
Ur denna matris kan vi se att systemet har oändligt många lösningar. Observera att lösningarna mycket lätt kan formuleras i parameterform, med hjälp av trappstegsmatrisen ovan!

Nu ger
```
>>X = A\b
X =
   -1/2
    3/4
      1
```
Även nu ges en varning, som måste tas på allvar: den angivna lösningen är ju bara en bland oändligt många. □

Övning 4: I denna uppgift skall du undersöka några olika ekvationssystem och jämföra lösningarna som erhålls med de olika metoderna som beskrivs ovan. Definiera först följande matriser (I står här för enhetsmatrisen):

$$A = \begin{bmatrix} 3 & -3 & 1 & 0 & 4 \\ 6 & 0 & 2 & 5 & 5 \\ -7 & 7 & 0 & 4 & 1 \\ 4 & 2 & 1 & 6 & 1 \\ -3 & 1 & 1 & 4 & 10 \end{bmatrix}, \text{ B=A-I}, \quad C = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix}, \quad D = B * \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \end{bmatrix}.$$

Talen p_1 - p_5 och q_1 - q_5 är de 5 sista siffrorna i era personnummer.
I varje deluppgift, testa alla metoderna: `rref`, `rrefmovie`, användning av `inv(A)` och användning av bråkstreck (backslash). Ange också lösningen/lösningarna där sådana finns. Behandla följande ekvationssystem:

a. `AX=C`.

b. `BX=C`.

c. `BX=D`.

□

3 Aritmetik och elementära funktioner

3.1 Aritmetiska operationer, (P-E,S 2.5)

De vanliga aritmetiska operationerna mellan tal ser ut så här :

+	addition
−	subtraktion
*	multiplikation
/	division
^	exponentiering (OBS! För att få denna symbol skriver man ^ följt av mellanslag!)

Talet π skrivs `pi` medan talet e skrivs `exp(1)`

Exempel 12: Skriver du
`>> 100*pi` (utan ; före avslutande RETURN)
blir svaret
```
ans =
     314.1593
```
(`ans` står för det senaste svaret).
Man bör för det mesta ge namn till beräkningarna. I ovanstående exempel skriver man då
`>> a=100*pi`
och får svaret
```
a =
    314.1593
```

MATLAB tillämpar den vanliga prioriteringsordningen mellan de aritmetiska operationerna :

```
>> 8^1/3
```

ger svaret

```
ans =
     2.6667
```

(dvs $8^1/3$)

medan

```
>> 8^(1/3)
```

ger svaret

```
ans =
    2
```

(dvs $8^{1/3}$) □

Man får allmän hjälp om dessa begrepp i `helpwin` genom att gå till biblioteket `matlab/ops` och i `helpdesk` välja `functions by subject` och sedan `Operators and Special Characters`.

3.2 Elementära funktioner, (P-E,S 2.5)

MATLAB har alla de vanliga elementära grundfunktionerna, alltså exponential- och logaritmfunktionerna, de trigonometriska funktionerna, arcusfunktionerna, absolutbelopp, kvadratrot, och flera andra. Här följer en lista på några av MATLABs funktioner:

`exp, log` (= ln), `log10` (= 10-logaritmen), `sin, cos, tan, atan` (= arctan),
`asin, abs, sqrt  sinh, cosh, tanh`

Observera att man alltid måste ha () runt variabeln som i `sin(pi/3)`.

Det finns ytterligare funktioner t.ex.

`sign , round , floor , ceil`

Man får en fullständig lista i `helpwin` under `matlab/elfun`.

Exempel 13: Vi beräknar $\ln(\sqrt{e})$:

```
>>y = log(sqrt(exp(1)))
y =
    0.5000
```

Vi löser ekvationen $e \tan x = 5, \quad -\pi/2 < x < \pi/2$:

```
>> x = atan(5/exp(1))
x =
    1.0728
```

□

Notera att man inte kan skriva `e^x` för exponentialfunktionen.

3.3 Formatering av utskrift, (P-E,S 2.6)

Man kan dirigera antal decimaler som skrivs ut och formen på utskriften med hjälp av kommandot `format`. De vanligaste varianterna är

format short	ger fem signifikanta siffror
format long	ger femton signifikanta siffror
format short e	ger fem signifikanta siffror i flyttalsnotering
format long e	ger femton signifikanta siffror i flyttalsnotering
format rat	talen approximeras (med hög noggrannhet) med bråk

En fullständig lista av de olika utskriftsformaten finner man under `format` i `matlab/general`. Prova med att skriva ut 10π i de olika utskriftsformaten, t.ex.

```
>> format long
>> 10*pi
ans =  31.41592653589793
```

OBSERVERA :
I vissa av följande uppgifter förekommer variablerna $p_1, \ldots, p_{10}$. De avser siffrorna i ditt personnummer $p_1p_2p_3p_4p_5p_6 - p_7p_8p_9p_{10}$ (välj ett av era personnummer om ni är två som gör övningen tillsammans).

Övning 5: Låt c vara $p_7p_8p_9p_{10}/1000$ och skriv in detta i Matlabs kommandofönster:
`>>c=ditt värde;`
Beräkna därefter med hjälp av MATLAB

a) roten till ekvationen $10^x = 21$. (Se ovan hur ekv $e \tan x = 5, \quad -\pi/2 < x < \pi/2$ kunde lösas.)

b) den positiva roten till ekvationen $ln(1 + x^2) = 1/c$.

□

3.4 Operationer med radmatriser, (P-E,S 3.1-7)

För att utnyttja MATLAB effektivt skall nästan alla variabler man använder vara radmatriser eller större matriser. Detta innebär en viss komplikation. Multiplikationen `x*y` betyder matrismultiplikation. Om `x` och `y` är enstaka tal så är det den vanliga produkten. Är `x` och `y` matriser så finns inte produkten såvida inte typerna stämmer överens.
Vi kan som nämnts ovan föreställa oss en matris lagrad som en lång lista av tal tillsammans med uppgift om matrisens typ. Ofta vill vi beräkna elementvisa (**punkt**-visa) produkter av tal i sådana listor. Den produkten skrivs `.*` alltså med en **punkt** framför `*`-tecknet. Med `x = [1,2,3]` är `x.*x= [1,4,9]`. På samma sätt skrivs division `x./y` och potenser `x.^y`.
Man kan även använda de elementära funktionerna på matriser. Allmän hjälp om detta område får du i `helpwin` under `matlab/elmat` och `matlab/ops`.

Exempel 14: (med utskrift i `format short`):

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

□

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 radmatris av längden 3.

Exempel 15:

```
>>x=[1 2 3 4]; z=ones(size(x))./x
```

ger svaret

```
>>z =
    1.0000 0.5000 0.3333 0.2500
>>x+ones(size(x))
```

ger svaret

```
ans =
     2 3 4 5
>>2*ones(size(x))
```

ger svaret

```
ans =
     2 2 2 2
```

Det näst sista svaret hade man också kunnat få genom att skriva `x+1`.
`z` kan man erhålla med `z = 1./x`. □

Övning 6: Skriv in radmatriserna $x = [p_1\ p_2\ p_3]$ och $y = [p_4\ p_5\ p_6]$.
Beräkna `x+y, x-y, x.*y, x.^y, x./y, x.\y, x.*sin(y).`

Vad svarar Matlab på $x * y$? □

3.5 Generering av aritmetiska följder, (P-E,S 4.3)

Om a, h och b är givna tal kan man bilda radmatrisen `x=[a,a+h,a+2h,... ,b]` med hjälp av kommandot `x=a:h:b`.

Exempel 16:

```
>>x=-5:2:5
```

ger

```
x =
   -5 -3 -1 1 3  5
```

Analogt ger kommandot

```
>>x=-pi:0.1:pi
```

radmatrisen

```
x =
  [-3.1416 -3.0416 -2.9416 ...  2.9584  3.0584]
```

vilket är en matris av längden 63. Kolla genom att mata in `x` enligt ovan och sedan skriva `length(x)`. Låt Matlab också skriva ut matrisen `x` på skärmen (inget semikolon efter `x`). Vill man ha steglängden `h = 1` räcker det att skriva `x=a:b`

```
>>x=0:10
x =
  0 1 2 3 4 5 6 7 8 9 10
```

Om `b < a` så kan (bör) man välja `h < 0`

```
>>x=2:-0.5:-1
x =
  2.0000  1.5000  1.0000 0.5000  0  -0.5000  -1.0000
```
□

Ytterligare information får du under `matlab/ops`. Läs den informationen, kolon-operatorn är en mycket viktig ingrediens i Matlab.

Exempel 17: För att beräkna den geometriska summan

$$\sum_{k=0}^{10} \frac{1}{2^k} = 1 + \frac{1}{2} + \frac{1}{4} \ldots + \frac{1}{2^{10}}$$

genererar vi först en radvektor med alla k-värden:
`>>k=0:10;`
sedan beräknar vi samtliga 2^{-k} genom att skapa radvektorn
`>>ak = 2.^(-k);` (detta ger $ak = [1\ \frac{1}{2}\ \ldots\ \frac{1}{1024}]$)
Till sist beräknar vi summan med kommandot `sum`.

```
>>S = sum(ak)
```

vilket ger

```
S =
   1.9990
```

Ovanstående beräkningar kan också genomföras direkt med följande kommando

```
>>S = sum(2.^-[0:10])
S =
   1.9990
```
□

Övning 7: Hur många termer n behövs för att den harmoniska summan

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

skall bli större än 10 ?
(Experimentera med hjälp av ”pil upp” !)
□

4 Kurvritning

4.1 Allmänt om plotkommandot, (P-E,S 13.1-2)

Om $x = [x_1, \ldots, x_n]$ och $y = [y_1, \ldots y_n]$ är två radmatriser av samma längd så kommer ritkommandot `plot(x,y)` att rita en kurva som sammanbinder de `n` stycken punkterna (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$ med räta linjestycken.

Om man inte ger ett särskilt kommando kommer MATLAB att välja koordinataxlarna så att samtliga punkter syns. (Detta kallas auto-scaling.) Man kan dirigera på vilket sätt kurvan ritas genom att ge order om särskild linjetyp. Man kan också rita ut punkterna utan att sammanbinda dem genom att bestämma vilken punkttyp som skall användas.

Exempel 18:

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

Allmän hjälp för tvådimensionell grafik ges i `matlab/graph2d`. □

4.2 Funktionskurvor

Av exemplet ovan framgår det redan hur man kan rita funktionskurvor. Säg att vi vill rita funktionen $f(x)$, på intervallet $a \leq x \leq b$. Man börjar då med att välja en lämplig steglängd h och bildar sedan radmatrisen `x=a:h:b`. Därefter låter man MATLAB beräkna funktionsvärden samt ger dessa ett namn. Man får funktionskurvan uppritad med kommandot `plot(x,funktionsnamn)`.

Exempel 19:

```
>> x=-2*pi:0.1:2*pi;
>> y=sin(3*x)+cos(5*x);
>>plot(x,y)
```

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)` □

Övning 8: Rita ovanstående figurer. □

Vid kurvritning bör man tänka på att det blir bättre graf om man har fler punkter, men bara upp till en viss gräns. Radmatriserna bör inte ha fler än ca 400 element. Däröver blir det bara slöseri med beräkningstid och minnesutrymme. I synnerhet om bilden skall skrivas ut på papper så är detta mycket viktigt.

4.3 Flera kurvor i samma figur, `(P-E,S 13.4)`.

Antag att vi vill rita funktionerna f och g ovan i samma figur. Vi kan då ge kommandot `plot(x,f,'-',x,g,'-.')` eller bara `plot(x,f,x,g)`. (I de här fallen ger MATLAB automatiskt olika färger till graferna.) Det finns också en annan möjlighet. Antag att vi först ritar funktionen f med hjälp av kommandot

```
>>plot(x,f)
```

Man kan sedan 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”.

4.4 Dimensionering av koordinataxlarna, (P-E,S 13.4).

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 kan du läsa hjälptexten i `matlab/graph2d`.

Övning 9: Ett bekvämt sätt att ge ett plotkommando där det är lätt att ändra är att på en enda rad skriva enligt följande exempel.
`>> a=0; b=1; h=(b-a)/400; x = a:h:b; y = sin(1./x); plot(x,y)`
Med vänster och höger piltangenter kan du flytta markören till den plats du vill ändra. Backspace och del raderar, pröva vad de tar bort.
Rita en figur i vilken de tre kurvorna

$$y = sin(x)/x \ , \ y = cos(x) \text{ och } y = 1$$

förekommer samtidigt.
Zooma in för att titta på gränsvärdet $\lim_{x\to 0} \dfrac{\sin(x)}{x}$. □

5 Funktionsfiler

5.1 Hur man skriver en funktionsfil, (P-E,S 2.8)

Programmering i matlab sker med hjälp av m-filer. dessa filer skall ges namn som slutar med .m (t.ex. funk.m, kvadrat.m etc.) Det finns två sorters m-filer: *funktionsfiler* och *kommandofiler (alt. scriptfiler)*. Vi kommer här att koncentrera oss på funktionsfiler.
Första raden i en funktionsfil inleds alltid med ordet `function`, sedan följer utparametrarna och därefter funktionsnamnet följt av inparametrarna:

```
function [y1,y2,...,yn] = filnamn(x1,x2,...xm)
% kommentarer (gärna en beskrivning av funktionen)
satser som beskriver funktionen
```

För att skriva en m-fil så behövs en bra texteditor, t.ex. MATLABs egna (startas med kommandot `edit`), *emacs* eller *winedt* (PC). För att hålla reda på alla m-filer så är det lämpligt att skapa en katalog med namnet ”Matlab” och sedan därunder en beskrivande katalogstruktur.

Exempel 20: Skriv av exakt enligt nedanstående och spara det du skrivit med namnet `fun.m` i katalogen `Matlab/`.
Första två (och enda raderna) skall vara:

```
function y=fun(x)
% indata är variabeln x och utdata är y=x^2*(1 - sin(x))
y = x.*x.*(1-sin(x));
```

Nu kan du kontrollera att filen finns i det aktuella biblioteket med kommandot `what`. Testa sedan filen med

```
>>help fun
>> fun(1)
```

och

```
>> fun([1,2,3])
```

Fungerade detta så kan du rita grafen med

```
>> fplot(@fun,[-1,1])    (fplot = funktionsplot)
```

Dessa fyra steg är ett bra sätt att kontrollera att en sådan här funktonsfil fungerar. `fplot` ovan är ett exempel på en fuktion i MATLAB vars ena indata-argument är en ”pekare” (kallas *funktionshandtag*) till en annan m-funktion. Ovan så är `@fun` funktionshandtaget till filen `fun.m`. □

Övning 10: Gör en funktionsfil *fun2.m* som ger funktionen $y = sin(x)/x$.
Spara den som `fun2.m`. Testa den som ovan. □

Övning 11: Gör en funktionsfil *summa.m* som beräknar summan

$$\sum_{k=1}^{n} \frac{1}{k^2} = 1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2}$$

(Här är n funktionens indata.) Testa genom

```
>> summa(3)
>> summa(4)
```

Kan du se någon trend med ökande n ? Plotta gärna n mot $summa(n)$. □

Övning 12: Skriv en funktionsfil som har som indata två vektorer **u** och **v** i rummet. Och som utdata ger vinkeln mellan dessa. □