Signaler och system – med tillämpningar

Introduktion till MATLAB

1 Inledning

MATLAB (för Matrix laboratory) är ett interaktivt programpaket som är mycket användbart i rad olika vetenskapliga och tekniska tillämpningar, däribland signalbehandling och reglerteknik. MATLAB kan kompletteras med s.k. verktygslådor (*toolboxes*). För övningarna i boken krävs SIGNAL PROCESSING TOOLBOX och CONTROL SYSTEM TOOLBOX. Vad som gör MATLAB förnämligt är dels den rika tillgången på olika verktygslådor och dels att man med de verktyg som finns kan utvidga med egna funktioner och manuskript (s.k. m-filer; program skrivna i MATLAB). MATLAB arbetar med i huvudsak en typ av objekt: rektangulära numeriska matriser med matriselement som är reella eller komplexa. En vanlig skalär blir specialfallet 1×1-matris. Detsamma gäller för vektorer: en radvektor blir en $1 \times n$-matris och en kolumnvektor en $n \times 1$-matris. Viktigt att notera är MATLAB gör skillnad mellan stora och små bokstäver i beteckningar på matriser.

2 Grundprinciper

2.1 Inmatning

Matriser kan skapas och inträda i MATLAB på en mängd olika sätt:

- En lista av element.
- Genererade av inbyggda programsatser och funktioner.
- Skapade i m-filer.
- Hämtade från externa datafiler.

Det enklaste sättet att skapa en matris på är genom en explicit lista. Listan av matriselement separeras genom mellanslag eller kommatecknet. Den omges av hakparanteser och man använder semikolon för att indikera slutet på en rad. Satsen

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

resulterar i följande utdata:

```
A =
     1     2     3
     4     5     6
     7     8     9
```

Matrisen `A` finns nu sparad i arbetsminnet för framtida bruk. Ett alternativt sätt att mata in matrisen `A` är rad för rad på följande sätt:

```
>> A = [1   2   3
        4   5   6
        7   8   9]
```

2.2 Matriselement

Ett matriselement kan vara ett godtyckligt MATLAB-uttryck, t.ex.

```
>> x = [-1.3 sqrt(3) (1+2+3)*4/5]
```

vilket resulterar i utskriften

```
x =
    -1.3000    1.7321    4.8000
```

Individuella matriselement kan refereras med index inuti parenteser. Om vi fortsätter ovanstående exempel:

```
>> x(5) = abs(x(1))
```

ger

```
x =
    -1.3000    1.7321    4.8000    0.0000    1.3000
```

Observera att storleken på `x` automatiskt ökar för att kunna innehålla det nya elementet. Det odefinierade elementet `x(4)` sätts till noll. Större matriser kan konstrueras genom att använda små matriser som element. T.ex. så kan vi lägga till en rad till matrisen `A` med följande satser:

```
>> r = [10 11 12];
>> A  =[A; r]
```

vilket resulterar i utskriften

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

Små matriser kan extraheras ur större matriser genom att använda kolon. Exempelvis ger

```
>> B = A(1:3, 2:3)
```

resultatet

```
B =
     2     3
     5     6
     8     9
```

2.3 Några viktiga funktioner för hantering av matriser

Inte så sällan uppkommer behovet av att ta reda på vilken dimension en matris har. Då använder man funktionen `size`, som ger tillbaka en radvektor med två element: antal rader och antal kolumner i matrisen:

```
>> s = size(A)
s =
     4     3
```

Vill man istället veta längden på en vektor skriver man

```
>> l = length(x)
l =
     5
```

som ger rätt resultat oavsett om x är en rad- eller kolumnvektor. Några andra viktiga funktioner är

- max, min resp. mean, som beräknar maximum, minimum resp. medelvärde av elementen i en vektor. Är argumentet en matris sker beräkning över varje kolumn. Exempel:

```
>> m = max(A)
m =
    10    11    12
```

- abs, sign resp. angle, som beräknar absolutbelopp, tecken resp. vinkel (i komplexa talplanet) elementvis. Resultatet är en matris med samma dimension som argumentet. Exempel:

```
>> xabs = abs(x)
xabs =
    1.3000    1.7321    4.8000         0    1.3000
```

2.4 Satser, variabler och uttryck

MATLAB är ett språk för uttryck. Uttryck som skrivs av användaren tolkas och utvärderas av MATLAB. Satser i MATLAB är ofta på formen:

```
variabel = uttryck
```

eller helt enkelt

```
uttryck
```

Om variabelnamn och likhetstecknet utelämnas så skapas automatiskt en variabel med namnet ans (för *answer*). Skriv t.ex. uttrycket

```
>> 1900/81+1
```

så fås resultatet

```
ans =
   24.4568
```

En sats avslutas normalt med ENTER-tangenten, men om det sista tecknet i ett uttryck är semikolon, så undertrycks utskriften. Detta är användbart då man skriver många satser som mellansteg innan man når slutresultatet. Observera att ans alltid innehåller det sist beräknade värdet. Detta kan utnyttjas genom att dela upp ett uttryck som ska beräknas i deluttryck. Anta att man vill beräkna $e^{-1.5^2}$. Detta skulle kunna göras på följande sätt:

```
>> 1.5^2
ans =
    2.25
>> exp(-ans)
ans =
    0.1054
```

Om uttrycket är så stort att det inte ryms på en rad, så kan man skriva tre eller fler punkter som då indikerar att satsen fortsätter på nästa rad. T.ex. så beräknar

```
>> s = 1 - 1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
    -1/8 + 1/9 - 1/10 + 1/11 -1/12;
```

summan $\sum_{n=1}^{12}(-1)^{n+1}/n$ och tilldelar variabeln `s` resultatet, men ingen utskrift fås på skärmen eftersom uttrycket avslutas med semikolon.

2.5 Information om arbetsarean (*workspace*)

För att ta reda på vilka variabler man har skapat kan man skriva:

```
>> who
Your variables are:
A         ans       r         x
B         p         s
```

Av detta ser vi att sju variabler har skapats genom våra exempel. Mer detaljerad information fås genom att skriva:

```
>> whos
                 Name        Size            Bytes     Class

                    A       4 by 3             12        No
                    B       3 by 2              6        No
                  ans       1 by 1              1        No
                    p       1 by 5              5        No
                    r       1 by 3              3        No
                    s       1 by 1              1        No
                    x       1 by 5              5        No

Grand total is 7 elements using 264 bytes
```

Varje element i en reell matris tar 8 bytes. Matris `A` som är en 4×3-matris tar alltså $12 \times 8 = 96$ bytes av minnet.

2.6 Tal och aritmetiska uttryck

Tal anges med konventionell decimalnotation med decimalpunkt. Här följer några exempel på notation som tillåts i MATLAB:

```
3           -99         0.0001
9.63972328  1.6021E-20  6.02252e23
```

2.7 Komplexa tal och matriser

Komplexa tal är tillåtna att använda i alla operationer och funktioner i MATLAB. Komplexa tal skrivs med hjälp av beteckningarna `i` och `j`, som bägge motsvarar $\sqrt{-1}$. Beroende på tycke och smak skriver en del av oss komplexa tal som

```
z = 3 + 4*i
```

medan andra föredrar (såsom i boken)

```
z = 3 + 4*j
```

Ett annat exempel är

```
>> r = 2;
>> theta = 45*pi/180;
>> w = r*exp(i*theta)
w =
   1.4142 + 1.4142i
>> abs(w)
ans =
     2
>> angle(w)*180/pi
ans =
    45
```

Det finns två bra sätt att mata in en komplex matris på:

```
>> A = [ 1 2; 3 4 ] + i*[ 5 6; 7 8 ]
```

eller

```
>> A = [ 1+5*i 2+6*i; 3+7*i 4+8*i ]
```

som ger samma resultat.

2.8 Utskriftsformat

För formatering av utskrifter finns `format`-kommandot. Nedan följer några exempel på formatering.

```
>> X = [4/3 1.2345e-6];
>> format short
>> X
X =
    1.3333    0.0000
>> format short e
>> X
X =
   1.3333e+00   1.2345e-06
>> format long
>> X
X =
   1.33333333333333   0.00000123450000
>> format hex
>> X
X =
   3ff5555555555555   3eb4b6231abfd271
>> format +
>> X
X =
++
```

2.9 Hjälpfunktionen `help`

Med `help`-funktionen kan man få information om det mesta som rör MATLAB. För att få en lista över tillgängliga ämnen (*topics*) som man kan söka hjälp på så skriver man

```
>> help
```

För att få hjälp på ett specifikt ämne så skriver man

```
>> help topic
```

Anta t.ex. att vi söker information om funktionen `inv`, som inverterar en matris. Man skriver då

```
>> help inv
INV INV(X) is the inverse of the square matrix X. A warning
    message is printed if X is badly scaled or nearly
    singular.
```

2.10 Avsluta och spara arbetsarean: `quit, save, load`

För att avsluta MATLAB räcker det med att skriva `quit` eller `exit`. Innan man avslutar vill man kanske spara alla variabler i arbetsarean. Detta görs genom att skriva

```
>> save
```

Detta sparar alla variabler i en fil med namnet `matlab.mat`. Nästa gång MATLAB startas upp kan man ladda in denna fil genom att skriva

```
>> load
```

Kommandot

```
>> save temp
```

sparar alla variabler i en fil med namnet `temp.mat`. Denna kan hämtas in genom att skriva kommandot

```
>> load temp
```

3 Matriser och arrayer

Matrisalgebra och matrisoperationer är fundamentalt i MATLAB och fungerar på precis samma sätt som i matematiken. Förutom detta finns det som benämns med termen *arrayoperationer:* aritmetik som sker elementvis.

3.1 Matrisalgebra

För att transponera en matris används specialtecknet `'` (apostrof). Följande satser i MATLAB visar hur transponering verkar:

```
>> A = [1 2 3; 4 5 6; 7 8 0]
A =
     1     2     3
     4     5     6
     7     8     0
>> B = A'
B =
     1     4     7
     2     5     8
     3     6     0
>> x = [-1 0 2]'
x =
    -1
     0
     2
```

Om Z är en komplex matris är Z' transponering och komplext konjugat av Z. Addition och subtraktion av matriser görs med matriser av samma dimension.

```
>> C = A+B
C =
     2     6    10
     6    10    14
    10    14     0
```

Addition och subtraktion är definierade även om en av operanderna är skalär. I dessa fall adderas/subtraheras alla element i den andra operanden:

```
>> A
A =
     1     2     3
     4     5     6
     7     8     0
>> A+1
ans =
     2     3     4
     5     6     7
     8     9     1
>> y = x-1
y =
    -2
    -1
     1
```

Multiplikation av två matriser, t.ex. X*Y, kräver att antalet kolumner i X är lika med antalet rader i Y:

```
>> x'*y
ans =
     4
>> x*y'
ans =
     2     1    -1
     0     0     0
    -4    -2     2
>> y*x'
ans =
     2     0    -4
     1     0    -2
    -1     0     2
>> b = A*x
b =
     5
     8
    -7
```

En skalär kan multipliceras med en godtycklig matris, varvid varje element multipliceras med skalären, exempelvis

```
>> pi*x
ans =
   -3.1416
         0
    6.2832
```

I MATLAB används divisionssymbolerna / och \ på matriser enligt följande: Om A är en icke singulär kvadratisk matris, så ger A\B och B/A vänster- resp. högermultiplikation av B med inversen av A, d.v.s. inv(A)*B resp. B*inv(A):

- X = A \ B är lösningen till ekvationen A*X = B, d.v.s. X = inv(A)*B.
- X = A / B är lösningen till ekvationen X*A = B, d.v.s. X = B*inv(A).

I tidigare exempel har vi beräknat vektorn b som A*x. Satsen

```
>> z = A\b
z =
    -1
     0
     2
```

gör att x återfås i form av vektorn z.

3.2 Elementvisa beräkningar (arrayoperationer)

I vanlig linjär matrisalgebra i MATLAB använder man symbolerna * / \ ^ och ' för att ange att man vill multiplicera, "dividera" (ta vänster- resp. högerinvers), upphöja (ta gånger sig själv ett antal gånger) eller transponera en matris. För att ange elementvis aritmetik föregås operatorerna av en punkt, det vill säga .* ./ .\ .^ och .' (Var noggrann med att i aritmetiska uttryck där arrayoperatorer används alltid ha ett blanktecken före punkten för att slippa problemet med att den kan betraktas som en decimalpunkt!) För addition och subtraktion är matrisoperationerna och arrayoperationerna lika. Vi exemplifierar användningen:

```
>> x = [1 2 3]; y = [4 5 6];
>> z = x.*y
z =
     4    10    18
```

Notera skillnaden mellan de två divisionsoperatorerna i nedanstående exempel.

```
>> z = x.\y
z =
    4.0000    2.5000    2.0000
>> z = x./y
z =
    0.2500    0.4000    0.5000
```

Här följer några exempel som illustrerar användningen av arraypotens:

```
>> z = x.^y
z =
     1    32   729
>> z = x.^2
z =
     1     4     9
>> z = 2.^[x, y]
z =
     2     4     8    16    32    64
```

3.3 Generering av vektorer med kolonnotation

Med kolonnotation kan man skapa vektorer på ett enkelt sätt. `J:K` är detsamma som `[J, J+1, ..., K]`. (`J:K` är tom om `J > K`.) `J:I:K` är detsamma som `[J, J+I, J+2*I, ..., K]`. (`J:I:K` är tom om `I > 0` och `J > K` eller om `I < 0` och `J < K`.) Exempel:

```
>> t = 1:10
t =
     1     2     3     4     5     6     7     8     9    10
>> y = 0:pi/4:pi
y =
         0    0.7854    1.5708    2.3562    3.1416
```

3.4 Att plocka ut rader och kolumner med kolonnotation

Kolonnotation kan användas för att plocka ut utvalda rader och kolumner i en vektor eller matris. `A(:)` är alla elementen i `A`, som en enda kolumnvektor. `A(:,J)` är den `J`:te kolumnen av `A`. `A(J:K)` är `A(J), A(J+1), ..., A(K)` och `A(:,J:K)` är `A(:,J), A(:,J+1), ..., A(:,K)`. Följande exempel visar hur man plockar ut vart tredje element i en vektor `y`:

```
>> y = [0 1 2 3 4 5 6 7 8 9 10];
>> x = y(1:3:10)
x =
     0     3     6     9
```

3.5 Egenvärden

Egenvärdena för en matris `A`, d.v.s. lösningarna till karakteristiska ekvationen $\det(\lambda I - A)$, beräknas med funktionen `eig` (för *eigenvalues*). Exempel:

```
>> A = [1 2 3; 4 5 6; 7 8 0]
A =
     1     2     3
     4     5     6
     7     8     0
>> eig(A)
ans =
   12.1229
   -0.3884
   -5.7345
```

4 Polynom

Polynom i MATLAB representeras i av radvektorer som innehåller polynomets koefficienter i avtagande ordning. Polynomet $p(x) = x^3 - 6x^2 - 72s - 27$ matas in i MATLAB som radvektorn

```
>> p = [1 -6 -72 -27];
```

Rötterna till denna ekvation fås genom

```
>> r = roots(p)
r =
   12.1229
   -5.7345
   -0.3884
```

Dessa kan användas för att rekonstruera det ursprungliga polynomet genom

```
>> p2 = poly(r)
p2 =
    1.0000   -6.0000  -72.0000  -27.0000
```

Den karakteristiska ekvationen $\det(\lambda I - A) = \lambda^3 - 6\lambda^2 - 72\lambda - 27$ till matrisen

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

kan beräknas med `poly`-funktionen enligt

```
>> p = poly(A)
p =
    1.0000   -6.0000  -72.0000  -27.0000
```

som är lite enklare än det ekvivalenta `poly(eig(A))`. Multiplikation av polynom görs med `conv`-funktionen. Anta att vi har två polynom: $a(x) = x^2+2x+3$ och $b(x) = 4x^2+5x+6$. Multiplikation beräknas enligt

```
>> a = [1 2 3]; b = [4 5 6];
>> c = conv(a, b)
c =
     4    13    28    27    18
```

I klartext blir resultatet $c(x) = a(x)b(x) = 4x^4+13x^3+28x^2+27x+18$. Division av polynom kan göras med `deconv`-funktionen:

```
>> [q, r] = deconv(c, a)
q =
     4     5     6
r =
     0     0     0     0     0
```

där vektorerna `q` och `r` innehåller koefficienterna för kvot- och restpolynomen.

5 Grafik

”En bild säger mer än tusen ord” är ett uttryck som är sant i många sammanhang. Bl.a. då man vill avläsa information ur stora datamängder så görs detta oftast bäst med hjälp av grafiska bilder. I MATLAB finns en mängd av funktioner som understödjer detta.

5.1 x–y-diagram

Anta att man vill plotta funktionen $\sin t$ för $0 \leq t \leq 10$. För att göra detta måste man först skapa en tidsvektor, med starttidpunkt i `0` och sluttidpunkt i `10`. Upplösningen bestämmer vi till `0.5` tidsenheter (sekunder):

```
>> t = 0:0.5:10
t =
  Columns 1 through 7
         0   0.5000   1.0000   1.5000   2.0000   2.5000   3.0000
  Columns 8 through 14
    3.5000   4.0000   4.5000   5.0000   5.5000   6.0000   6.5000
  Columns 15 through 21
    7.0000   7.5000   8.0000   8.5000   9.0000   9.5000  10.0000
```

För att skapa linjära grafer använder man `plot`-funktionen på följande sätt:

```
>> plot(t, sin(t))
```

som gör att `t`-vektorn plottas mot `sin(t)`-vektorn i grafikfönstret.

5.2 Logaritmiska och polära diagram samt stapeldiagram

För logaritmiska plottar finns funktionerna `loglog`, `semilogx` och `semilogy`, samt för polära plottar `polar`. Ritning av trappstegskurvor görs med `stairs` och för stapeldiagram används `bar` och `hist`.

5.3 Yt- och konturdiagram

För att generera tredimensionella perspektivplottar kan funktionen `mesh` användas. Anta t.ex. att man vill åskådliggöra funktionen $xe^{-x^2-y^2}$ i en tredimensionell graf:

```
>> [x, y] = meshgrid( -2:0.2:2, -2:0.2:3);
>> z = x .* exp(-x.^2-y.^2);
>> mesh(x, y, z)
```

6 Flödeskontroll

I MATLAB finns repetitiva och villkorliga satser som påminner om motsvarande i högnivåspråk som C och Java.

6.1 `for`-loop

Det allmänna utseendet på en `for`-loop är

```
for variabel = uttryck
    sats
end
```

Anta att summan $\sum_{n=1}^{1000} \frac{1}{n}$ ska beräknas. Detta kan göras på följande sätt i MATLAB:

```
>> s = 0;
>> for n = 1:1000
   s = s+1/n;
 end
>> s
s =
    7.4855
```

I exemplet ovan så har vi semikolon efter satsen i `for`-loopen för att undertrycka utskrift. Uttrycket i en `for`-loop är, som allting annat i MATLAB, en matris. Formen för denna är `m:n` eller `m:i:n`. I exemplet ovan blir resultatet av `n = 1:1000` att en vektor med 1000 element skapas: `n = [1 2 3 ... 998 999 1000]`. I MATLAB ska man helst försöka undvika att göra loop-strukturer vid beräkningar, då dessa tar mycket längre tid att utföra än om man hade löst problemet med matris/array-beräkningar. Ovanstående summation hade kunnat göras på följande sätt istället:

```
>> n = 1:1000;
>> sum(1./n);
```

6.2 `while`-loop

En `while`-loop tillåter en grupp av satser att bli repeterade under kontroll av ett logiskt villkor:

```
while uttryck
    sats
end
```

Uttrycket i `while`-satsen är en matris och repetition sker så länge alla elementen i matrisen är skilda från noll. Anta igen att summan ovan ska beräknas. Detta görs på följande sätt med `while`-loopen:

```
>> s = 0; n = 0;
>> while n <= 1000
    n = n+1;
    s = s+1/n;
 end
>> s
s =
    7.4865
```

6.3 `if`- och `break`-satser

Följande problem från talteorin är olöst: Ta ett positivt heltal. Om det är jämnt, dela det med 2 och om det är udda, multiplicera det med 3 och addera till 1. Upprepa denna process ända tills heltalet blir 1. Frågan är nu: finns det något heltal som kan hålla igång denna loop för evigt? Programmet nedan illustrerar `while` och `if`-satser. Det visar också `input`-funktionen från vilken tangentbordsdata kan hämtas in. En `break`-sats ger ett uthopp ur loopen då ett negativt heltal matas in.

```
>> %Klassiskt problem från talteorin
>> while 1
  n = input('Mata in ett tal n, negativa avslutar ');
  if n < 0
    break;
  end
  while n > 1
    if rem(n, 2) == 0
      n = n/2
    else
      n = 3*n+1
    end
  end
end
```

7 Manuskript och funktioner

Normalt arbetar MATLAB i en kommandodriven mod, där man rad för rad matar in kommandon via tangentbordet. MATLAB kan också exekvera en sekvens av kommandon som samlas i en fil. Filer som innehåller MATLAB-satser kallas m-filer på grund av att sista delen av filnamnet (*extension*) är ".m". En användning av m-filer är att automatisera långa sekvenser av kommandon i s.k. manuskriptfiler (*script files*). Den andra typen av m-filer erbjuder till en utvidgning av MATLAB genom s.k. funktionsfiler, där nya funktioner skapas.

7.1 Manuskriptfiler

När ett manuskript anropas i MATLAB så exekveras kommandona i denna fil. Programsatserna i en manuskriptfil opererar globalt på variablerna i arbetsarean. Skapa en fil med namnet `sum1.m` genom att öppna MATLABs egen texteditor under menyn `File` och `New M-file`. Skriv sedan in följande programsatser. (Notera att procenttecknet inleder en kommentar.)

```
% M-fil som beräknar summan av serien
% 1 + 1/2 + 1/9 + 1/16 + 1/25 + ...
% Vi summerar de 1000 första elementen i denna serie:
s = 0;
for n = 1:1000
   s = s + 1/(n*n);
end
s  % Visa resultatet av beräkningarna
```

Spara filen och gå över till MATLABs kommandofönster. Genom att skriva satsen `sum1` kommer alla satser i denna fil att utföras och resultera i följande:

```
>> sum1
s =
    1.6439
```

7.2 Funktionsfiler

En funktionsfil påminner mycket om en manuskriptfil. Det som skiljer är att i första raden innehåller m-filen ordet `function` och att argument kan skickas med. Alla variabler som definieras inuti en funktionsfil är lokala, d.v.s. de opererar ej globalt på arbetsarean. Betrakta nedanstående funktion:

```
function y = kvadrat(x)
% Kvadrat beräknar elementvis kvadraten av x.
y = x .* x;
```

Här följer några funktionsanrop på `kvadrat`-funktionen:

```
>> kvadrat(3)
ans =
     9
>> kvadrat([8, 2, 4 ,5])
ans =
    64     4    16    25
>> z = 4;
>> z2 = kvadrat(z)
z2 =
    16
>> help kvadrat
  Kvadrat beräknar elementvis kvadraten av x.
```