Matriser och linjära ekvationssystem

Matriser

En matris är som ni vet ett rektangulärt talschema:

$$\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \tag{1}$$

Matrisen ovan har m rader och n kolonner, vi säger att den är av typ $m \times n$. Ett matriselement i rad nr i, kolonn nr j tecknas a_{ij}, där i är radindex och j är kolonnindex. I MATLAB skrivs detta `A(i,j)` och `[m,n]=size(A)` ger matrisens typ, medan `m=size(A,1)` ger endast antal rader och `n=size(A,2)` ger endast antal kolonner.

Indexeringen i MATLAB är alltid som i (1), dvs. rad- och kolonnindex börjar alltid på ett och vi kan inte ändra på det.

En matris av typ $m \times 1$ kallas kolonnmatris (kolonnvektor) och en matris av typ $1 \times n$ kallas radmatris (radvektor):

$$\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_1 & \cdots & c_n \end{bmatrix} \tag{2}$$

Du kommer att se att vi använder oftast kolonnvektorer för att representera kvantiteter som vi beräknar. Element nr i ges i MATLAB av `b(i)` och antalet element ges av `m=length(b)`. Även för vektorer gäller att indexeringen alltid börjar på ett. Motsvarande gäller för radvektorn $\mathbf{c}$.

Som exempel tar vi

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 0 & 2 & 4 \end{bmatrix}$$

Vi skriver in detta i MATLAB enligt

```
>> A=[1 4 7 10; 2 5 8 11; 3 6 9 12]
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12
```

```
>> b=[1; 3; 5]
b =
     1
     3
     5

>> c=[0 2 4]
c =
     0     2     4
```

och ser på typerna och några element med

```
>> [m,n]=size(A)
m =
     3
n =
     4

>> A(2,3)
ans =
     8
```

Prova gärna `length` och `size` på `b` och `c`. Någon skillnad? Skriv ut något element också.

En matris kan betraktas som en kollektion av kolonner:

$$\mathbf{A} = \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{m1} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \cdots & \mathbf{a}_j & \cdots & \mathbf{a}_n \end{bmatrix}$$

med kolonnerna

$$\mathbf{a}_1 = \begin{bmatrix} a_{11} \\ \vdots \\ a_{m1} \end{bmatrix}, \quad \mathbf{a}_j = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix}, \quad \mathbf{a}_n = \begin{bmatrix} a_{1n} \\ \vdots \\ a_{mn} \end{bmatrix}$$

Man kan även betrakta den som en kollektion av rader, men vi använder oftast kolonnrepresentationen. I MATLAB plockar man ut kolonn nr j med `A(:,j)`. Här är j kolonnindex medan radindex $i = 1, \ldots, m$ representeras av tecknet kolon `:`. På liknande vis ges rad nr i av `A(i,:)`.

```
>> a1=A(:,1)
a1 =
     1
     2
     3

>> A2=A(2,:)
A2 =
     2     5     8    11
```

Uppgift 1. Skriv in följande matriser i MATLAB.

$$\mathbf{A} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 4 & 5 & 6 \\ 3 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} 0 & 2 & 4 \end{bmatrix}$$

(a). Skriv ut matriselementen a_{23}, b_{23}, c_2 och d_3. Prova `size` och `length`. Ändra b_{23} genom att skriva `B(2,3)=5`.

(b). Skriv ut kolonn nr 1, 2 och 3 ur matrisen $\mathbf{A}$. Sätt in kolonnvektorn $\mathbf{c}$ som 2:a kolonn i $\mathbf{A}$ genom att skriva `A(:,2)=c`.

(c). Radera matrisen $\mathbf{B}$ (`clear B`) och skriv in den igen genom att först bilda kolonnerna

$$\mathbf{b}_1 = \begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix}, \quad \mathbf{b}_2 = \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}, \quad \mathbf{b}_3 = \begin{bmatrix} 6 \\ 1 \\ 1 \end{bmatrix}$$

och sedan sätta in dem i matrisen $\mathbf{B} = [\mathbf{b}_1\ \mathbf{b}_2\ \mathbf{b}_3]$.

Vi kan ta ut ett block ur en matris med `A(iv,jv)` där `iv` är en vektor med radindex och `jv` är en vektor med kolonnindex. Resultatet blir en matris med `length(iv)` rader och `length(jv)` kolonner.

```
>> A=[1 3 5; 7 9 11; 2 4 6]
A =
     1     3     5
     7     9    11
     2     4     6
```

```
>> B=A([2 3],[1 3])
B =
     7    11
     2     6
```

Transponatet $\mathbf{A}^\mathsf{T}$ av en matris $\mathbf{A}$ ges av apostrof (').

```
>> A=[1 3 5; 7 9 11]
A =
     1     3     5
     7     9    11
```

```
>> B=A'
B =
     1     7
     3     9
     5    11
```

Uppgift 2. Låt som i uppgift 1

$$\mathbf{A} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 4 & 5 & 6 \\ 3 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 1 \\ 3 \\ 5 \\ 7 \end{bmatrix}, \quad \mathbf{d} = \begin{bmatrix} 0 & 2 & 4 \end{bmatrix}$$

(a). Sätt in kolonnvektorn $\mathbf{c}$ som 3:e kolonn i $\mathbf{A}$ och sätt in radvektorn $\mathbf{d}$ som 2:a rad i $\mathbf{B}$.

(b). Låt 1:a och 4:e raden i $\mathbf{A}$ byta plats och låt därefter den 2:a och 3:e kolonnen byta plats.

Bygga upp matriser

Med funktionerna `zeros` och `ones` kan man i MATLAB bilda matriser med nollor och ettor. Exempelvis `zeros(m,n)` ger en matris av storleken $m \times n$ fylld med nollor. Med `zeros(size(A))` får vi en matris fylld med nollor av samma storlek som `A`. Motsvarande gäller för `ones`.

Enhetsmatriser bildas med funktionen `eye`, med `eye(n)` får vi enhetsmatrisen av storleken $n \times n$. Man kan också använda `eye` för att bilda rektangulära matriser med ettor på huvuddiagonalen och nollor för övrigt. Med `eye(m,n)` får vi en sådan matris av storleken $m \times n$ och med `eye(size(A))` får vi en av samma storlek som `A`.

Med `diag` bildas diagonalmatriser. En vektor kan läggas in på en viss diagonal enligt

```
>> d=[2 3 7];
>> A=diag(d,0) % eller A=diag(d)
A =
     2     0     0
     0     3     0
     0     0     7
```

```
>> d=[2 3 7];
>> A=diag(d,1)
A =
     0     2     0     0
     0     0     3     0
     0     0     0     7
     0     0     0     0
```

Huvuddiagonalen markeras med `0`, diagonalen ovanför till höger med `1`, diagonalen nedanför till vänster med `-1`, osv. Matrisen blir så stor att vektorns alla element får plats längs angiven diagonal.

Vi kan sätta ihop två matriser `A` och `B` horisontellt med `[A B]` om antal rader i de två matriserna är lika. Två matriser `A` och `B` kan sättas ihop vertikalt med `[A; B]` om antal kolonner i de två matriserna är lika.

Matris-vektor-multiplikation

Vi definierar produkten av en radmatris och en kolonnmatris (med samma antal element) enligt:

$$\mathbf{ax} = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = a_1x_1 + a_2x_2 + \cdots + a_nx_n.$$

Observera att detta är samma formel som för skalärprodukt. Produkten $\mathbf{y} = \mathbf{Ax}$ av en matris av typen $m \times n$ och en kolonnvektor av typen $n \times 1$ definieras på liknande vis genom att vi multiplicerar matrisens rader i tur och ordning med kolonnvektorn. Vi får en kolonnvektor av typen $m \times 1$ och den ges av

$$\mathbf{y} = \mathbf{Ax} \tag{3}$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}, \tag{4}$$

$$y_i = \sum_{j=1}^{n} a_{ij}x_j = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n.$$

Observera att typerna måste stämma överens för att produkten ska vara definierad enligt regeln $m \times 1 = (m \times n)(n \times 1)$.

Ett alternativt sätt att introducera matris-vektor-multiplikation är att definiera $\mathbf{Ax}$ som en linjärkombination av kolonnerna i $\mathbf{A}$, (se Lay avsnitt 1.4)

$$\mathbf{Ax} = \begin{bmatrix} \mathbf{a_1} & \cdots & \mathbf{a_n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = x_1\mathbf{a_1} + x_2\mathbf{a_2} + \cdots + x_n\mathbf{a_n}.$$

Detta leder förstås till samma uttryck som i (4),

$$x_1\mathbf{a_1}+x_2\mathbf{a_2}+\cdots+x_n\mathbf{a_n} = \begin{bmatrix} a_{11} \\ \vdots \\ a_{m1} \end{bmatrix} x_1 + \begin{bmatrix} a_{12} \\ \vdots \\ a_{m2} \end{bmatrix} x_2 + \cdots + \begin{bmatrix} a_{1n} \\ \vdots \\ a_{mn} \end{bmatrix} x_n = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}.$$

I MATLAB skriver man helt enkelt `y=A*x`.

Uppgift 3. Skriv in följande matriser i MATLAB.

$$\mathbf{A} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 4 & 5 & 6 \\ 3 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Beräkna följande produkter, först för hand sedan med MATLAB.

$$\mathbf{Ax}, \quad \mathbf{Bx}, \quad \mathbf{ax}, \quad \mathbf{Aa}.$$

Observera att alla dessa produkter inte nödvändigtvis är definierade. Varför, och hur märks det i MATLAB?

Linjärt ekvationssystem

Matriser används bland annat för att skriva ned linjära ekvationssystem. Exempel: ekvationssystemet

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 14 \\ 3x_1 + 2x_2 + x_3 = 10 \\ 7x_1 + 8x_2 = 23 \end{cases}$$

kan skrivas på matrisform

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 7 & 8 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 14 \\ 10 \\ 23 \end{bmatrix},$$

dvs.

$$\mathbf{Ax} = \mathbf{b}, \quad \text{med } \mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 7 & 8 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 14 \\ 10 \\ 23 \end{bmatrix}.$$

Vi ska lära oss hur man löser sådana ekvationssystem. I MATLAB finns backslash-kommandot (\) eller alternativt kommandot `rref` (row-reduced-echelon form) som löser systemet, $\mathbf{Ax} = \mathbf{b}$:

```
>> x=A\b
>> rref([A b])
```

I det första fallet fungerar det bra om lösningen är entydig men sämre om det finns fria variabler eller inga lösningar alls. I det andra fallet reducerar MATLAB den utökade matrisen [**A b**] till reducerad trappstegsform.

Uppgift 4. Skriv följande ekvationssystem på matrisform (utökad matris!) och lös dom sedan med `\` respektive `rref`.

$$\begin{cases} x_1 + 5x_2 + 9x_3 = 29 \\ 2x_1 + 5x_3 = 26 \\ 3x_1 + 7x_2 + 11x_3 = 39 \end{cases} \qquad \begin{cases} x_1 + x_2 + 3x_3 + 4x_4 = 2 \\ -2x_1 + 2x_2 + 2x_3 = -4 \\ x_1 + x_2 + 2x_3 + 3x_4 = 1 \\ x_1 - x_2 - 2x_3 - x_4 = 1 \end{cases}$$

Ibland blir det något enklare att tolka svaret om vi ger kommandot `format rat` före beräkningarna, då skriver MATLAB svaret med rationella tal. Med `format short` får vi tillbaka standard formatet.

Skapa matris kolonnvis

Vi bygger ofta upp matriser med hjälp någon beräkningsalgoritm, ibland görs beräkningen kolonnvis. Vi ger ett exempel. Matrisen

$$\mathbf{C} = \begin{bmatrix} 1 & 2 & 3 & 4 & \cdots & 10 \\ 1 & 2 & 3 & 4 & \cdots & 10 \\ 1 & 2 & 3 & 4 & \cdots & 10 \\ 1 & 2 & 3 & 4 & \cdots & 10 \end{bmatrix}$$

skapas av `for`-loopen

```
clear C
ett=ones(4,1);
for i=1:10
    C(:,i)=i*ett;
end
```

Uppgift 5. Skriv en skriptfil som gör detta. Tag bort semikolon så att du ser vad som händer i varje steg.

Du kan också pröva att bilda `C` genom en s.k. ytterprodukt enligt `C=ett*[1:10]`. Lägg märke till att `ett` är en kolonnvektor medan `[1:10]` är en radvektor.

Vi kan använda `tic` och `toc` för att mäta den tid beräkningarna tar. Det blir tydligare om vi tar en större matris. Pröva följande som en skriptfil.

```
clear C
ett=ones(2000,1);
tic
for i=1:100
    C(:,i)=i*ett;
end
toc
```

Tänk på att sätta semikolon på alla beräkningar, så inte skärmen fylls med ointressanta utskrifter.

Matrisen `C` kommer successivt byggas upp och bli av typen 2000×100. Om vi nu istället för `clear C` skriver `C=zeros(2000,100);` så får matrisen `C` sin slutliga storlek på en gång. Vi börjar med en stor matris fylld med nollor, där vi i loopen successivt sätter in rätt kolonner. Kör nu den nya skriptfilen och se om beräkningstiden blev kortare.

Matrisiterationer

Avslutningsvis skall vi kort se på några matrisiterationer.

Uppgift 6. Låt

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

Skriv en `for`-loop som skapar matrisen $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_5]$ där

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k.$$

Uppgift 7. *Barnsley Fern* – Genom att rita ut punkter $\mathbf{x}_k$ i planet som genereras av iterationsformeln

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{b}$$

där matrisen $\mathbf{A}$ och vektorn $\mathbf{b}$ väljs slumpvis enligt vissa regler, kan vi producera en bild som liknar t.ex. ett ormbunksblad. (Detta är en s.k. fraktal.)

I varje iteration låter vi $\mathbf{A} = \mathbf{A}_i$ och $\mathbf{b} = \mathbf{b}_i$ med sannolikheten P_i, där

$$\mathbf{A}_1 = \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix},\ \mathbf{A}_2 = \begin{bmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{bmatrix},\ \mathbf{A}_3 = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix},\ \mathbf{A}_4 = \begin{bmatrix} 0 & 0 \\ 0 & 0.16 \end{bmatrix}$$

och

$$\mathbf{b}_1 = \mathbf{b}_2 = \begin{bmatrix} 0 \\ 1.6 \end{bmatrix},\ \mathbf{b}_3 = \begin{bmatrix} 0 \\ 0.44 \end{bmatrix},\ \mathbf{b}_4 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

samt

$$P_1 = 0.85,\ P_2 = P_3 = 0.07,\ P_4 = 0.01$$

Starta med $\mathbf{x}_0 = (0.5, 0.5)$ och rita ut punkterna $\mathbf{x}_k$ för $k = 0, 1, \cdots, 100$.

När man snabbt skall rita en snygg bild på skärmen (då krävs det ca 20000 punkter) använder man lämpligen ett grafikobjekt vars data uppdateras i varje iteration.

Lite hjälp på vägen om det behövs ...

```
figure('Name','Ormbunksblad','Menu','none','NumberTitle','off',...
       'Position',[256 412 320 512])
set(gcf,'Color',[0.1 0.1 0.1])
h = animatedline('color','g','linestyle','none','marker','.','markersize',1);
```

```
kmax=20000; x=[0.5;0.5];
addpoints(h,x(1),x(2))
axis([-3 3 0 10]), axis off, shg
p=[0.85 0.92 0.99 1.00];
A1=[ 0.85  0.04;-0.04 0.85]; b1=[0;1.6 ];
A2=[ 0.20 -0.26; 0.23 0.22]; b2=[0;1.6 ];
A3=[-0.15  0.28; 0.26 0.24]; b3=[0;0.44];
A4=[ 0     0   ; 0    0.16]; b4=[0;0   ];
for k=1:kmax
    r=rand;
    if r<p(1)
        x=A1*x+b1;
    elseif r<p(2)
        x=A2*x+b2;
    elseif r<p(3)
        x=A3*x+b3;
    else
        x=A4*x+b4;
    end
    addpoints(h,x(1),x(2))
    drawnow update
end
```