Matriser och linjära ekvationssystem

En matris är som ni vet ett rektangulärt talschema:

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

- Matrisen ovan är en $m \times n$-matris.
- Matriselementet på rad i i kolumn j betecknas a_{ij}

I Matlab används `size` för att avgöra antalet rader och kolumner i en matris.

```
>> A = [1 2; 3 4; 5 6];
>> [m,n] = size(A)
m =  3
n =  2
```

Matriselementet på rad i kolumn j i matrisen `A`

```
>> i = 2; j = 1;
>> A(i, j)
ans =  3
```

En $m \times 1$-matris kallas kolumvektor och en $1 \times n$ matris kallas radvektor.

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \quad \boldsymbol{c} = \begin{bmatrix} c_1 & \dots & c_n \end{bmatrix}$$

I Matlab:

```
>> b = [1; 2; 3]
b =  1
     2
     3

>> c = [4 5 6 7]
c =  4  5  6  7
```

Det finns många kommandon och andra sätt att skapa matriser. Vi har tidigare stött på `zeros`, `ones`. Men det finns många fler, t.ex. `eye` för enhetsmatrisen

```
>> I = eye(3)
I =  1  0  0
     0  1  0
     0  0  1
```

Kommandot `diag` för att skapa diagonalmatriser

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

Man kan skapa matriser med hjälp av mindre matriser

```
>> C = [A I A]
C =  2  0  0  1  0  0  2  0  0
     0  3  0  0  1  0  0  3  0
     0  0  7  0  0  1  0  0  7
```

(storlekarna måste stämma överens)

Operatorerna `^ , *, /, \, +, -` är matrisoperatorer.

Matrismultiplikation

```
>> A = [1 2; 3 4]; B = [1 0 1; 3 0 5];
>> C = A*B
C =  7     0    11
    15     0    23
```

Operatorn \ löser ett linjärt ekvationssystem.

```
>> b = [0; 1]
>> x = A\b
x = 1.0000
   -0.5000
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \Leftrightarrow$$

$$x_1 = 1, x_2 = -0.5$$

I Matlab används \ eller `rref` för att lösa linjära ekvationssystem.

$$\begin{cases} x_1 + 5x_2 + 9x_3 = 29 \\ \quad\; 2x_2 + 5x_3 = 26 \\ 3x_1 + 7x_2 + 11x_3 = 39 \end{cases}$$

```
>> A = [1 5 9; 0 2 5; 3 7 11]; b = [29; 26; 39];
>> x = A\b
x =  13.0000
    -22.0000
     14.0000
```

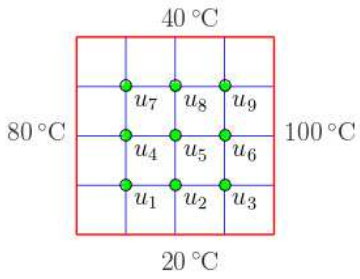
Operatorn \ fungerar bra om lösningen är entydig, men sämre om det finns fria variabler eller inga lösningar alls.

```
>> A = [1 5 9; 0 2 5; 3 7 11]; b = [29; 26; 39];
>> X = rref([A b])
X =  1     0     0    13
     0     1     0   -22
     0     0     1    14
```

Kommandot `rref` radreducerar den utökade matrisen $\left[\begin{array}{cc} \boldsymbol{A} & \boldsymbol{b} \end{array}\right]$ till reducerad trappstegsform.

Uppgift 6 lab3.1

Uppgift 6. Vi skall beräkna temperaturen på en stålplatta där plattans kanter hålls vid temperaturer enligt figuren.



Antag att temperaturen i en nodpunkt är medelvärdet av temperaturena i de närmsta nodpunkterna i *väster*, *öster*, *söder* och *norr*. Låt $u_1, u_2, \cdots, u_9$ beteckna temperaturerna i de olika nodpunkterna. Sätt upp de ekvationer som ger temperaturen i de olika nodpunkterna. Skriv det linjära ekvationssystemet på matrisform $\mathbf{Au} = \mathbf{b}$ och lös detta i MATLAB.

1: Formulera ekvationssystemet:

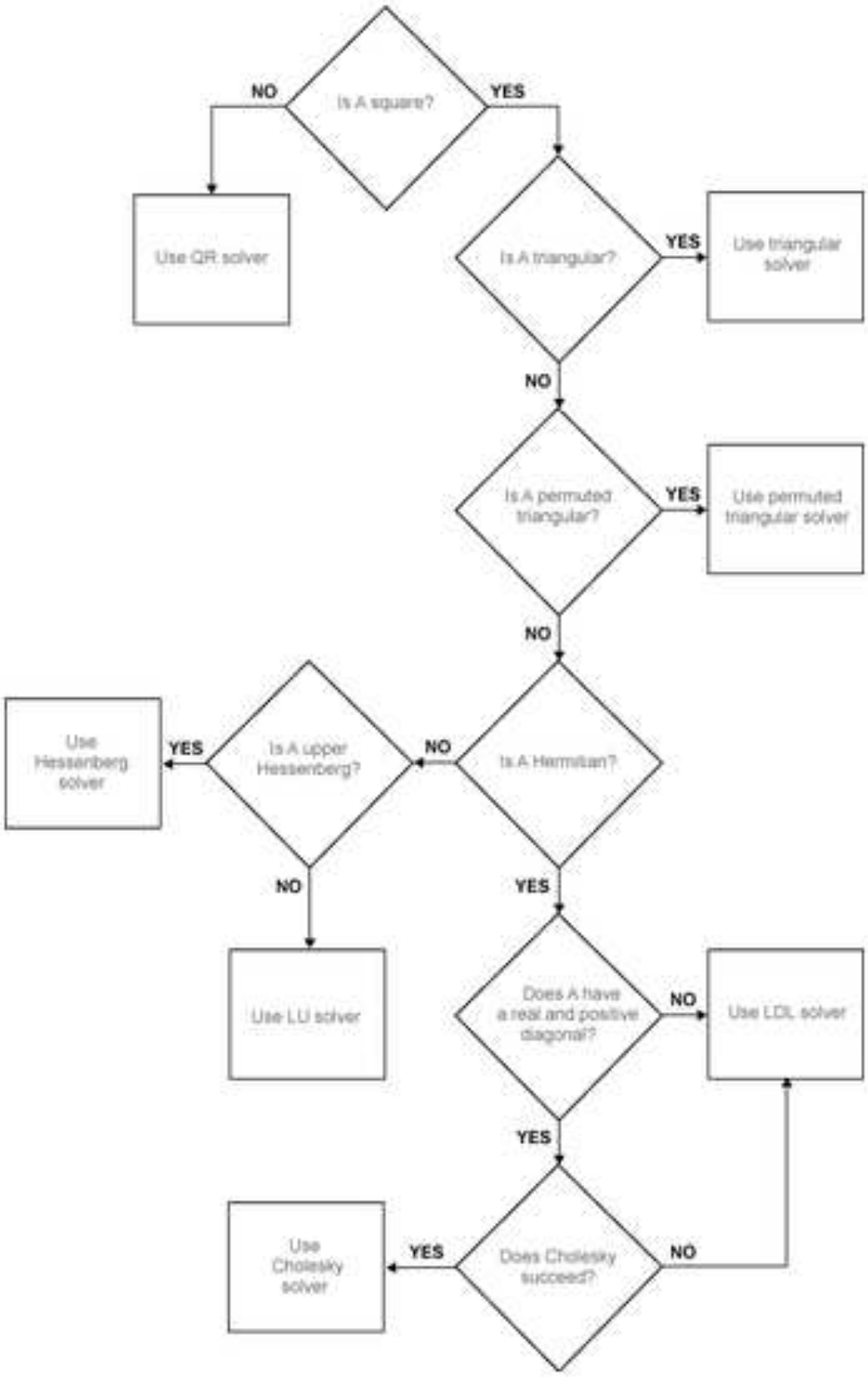
$$\begin{cases} u_1 = (u_4 + u_2 + 20 + 80)/4 \\ u_2 = (u_5 + u_3 + 20 + u_1)/4 \\ \vdots \end{cases} \Leftrightarrow \begin{cases} 4u_1 - u_2 - u_4 = 20 + 80 \\ 4u_2 - u_1 - u_3 - u_5 = 20 \\ \vdots \end{cases}$$

2: Formulera koefficientmatris och högerled:

$$\boldsymbol{A} = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & \dots & 0 \\ & \vdots & & \vdots & & \vdots & & \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} 100 \\ 20 \\ \vdots \end{bmatrix}$$

3: Lös ekvationssystemet i Matlab.

Kommandot \ använder olika algoritmer beroende på hur koefficientmatrisen ser ut. Flödesschemat nedan har jag hämtat från Matlabs dokumentation. Det beskriver vilka metoder som \ använder för att lösa ett linjärt ekvationssystem. Du kan själv titta på det om du vill, skriv i så fall `doc \` i kommandofönstret. **Man behöver inte lära sig de olika metoderna som finns i flödesschemat nedan, inte heller vilka metoder som används när**.



Matriser och linjära ekvationssystem

Om man löser ekvationssystemet i uppgift 6 (lab3.1) ovan med \ kommer en algoritm som heter LU-faktoriserng att användas.

Vi exemplifierar med ett mindre ekvationssystem.

$$\begin{cases} x_1 + x_2 + 6x_3 = 21 \\ -x_1 + 8x_2 + 5x_3 = 30 \\ x_1 - 2x_2 + x_3 = 0 \end{cases} \Leftrightarrow \begin{bmatrix} 1 & 1 & 6 \\ -1 & 8 & 5 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 21 \\ 30 \\ 0 \end{bmatrix}$$

När man löser ett linjärt ekvationssystem $\boldsymbol{Ax} = \boldsymbol{b}$ med LU-faktorisering börjar man med att LU-faktorisera matrisen $\boldsymbol{A}$. Man bestämmer två triangulära matriser $\boldsymbol{L}$ och $\boldsymbol{U}$ sådana att $\boldsymbol{A} = \boldsymbol{LU}$. Sedan löser man de två systemen $\boldsymbol{Ly} = \boldsymbol{b}$ och $\boldsymbol{Ux} = \boldsymbol{y}$

Kommandot `lu` i Matlab kan användas om man vill LU-faktorisera en matris.

```
>> A = [1 1 6; -1 8 5; 1 -2 1];
>> [L, U] = lu(A)
L =  1.0000         0         0
    -1.0000    1.0000         0
     1.0000   -0.3333    1.0000

U =  1.0000    1.0000    6.0000
          0    9.0000   11.0000
          0         0   -1.3333
```

Produkten av matriserna `L` och `U` blir `A`

```
>> L*U
ans = 1     1     6
     -1     8     5
      1    -2     1
```

Lösningen bestäms sedan genom att man löser de två ekvationssystemen $\boldsymbol{Ly} = \boldsymbol{b}$ och $\boldsymbol{Ux} = \boldsymbol{y}$. I Matlab kan man skriva

```
>> b = [21; 30; 0];
>> y = L\b;
>> x = U\y
x =  1
     2
     3
```

När man löser linjära ekvationssystem brukar man inte LU-faktorisera dem själv, som i exemplet ovan. Utan man brukar bara anropa \ som ju LU-faktoriserar koefficientmatrisen och bestämmer lösningarna på en gång.

```
>> A = [1 1 6; -1 8 5; 1 -2 1]; b = [21; 30; 0];
>> x = A\b
x =  1
     2
     3
```

Matriser och linjära ekvationssystem

Matriser med få nollskilda element kallas glesa.

Matrisen `B` i exemplet nedan innehåller 100 element, bara 3 element är nollskilda.

```
>> B = zeros(10,10);
>> B(1,1) = 9; B(2,3) = 5; B(3,2) = 27;
```

Kommandot `sparse` skapar en gles matris

```
>> Bs = sparse(B)
Bs =  (1,1)  9
      (3,2)  27
      (2,3)  5
```

Operatorer och kommandon fungerar som vanligt. Det är bara matrisens lagringsform som ändrats av Matlab. Istället för att lagra alla elementen i matrisen lagras bara de element som är nollskilda och platserna i matrisen de finns på.

Kommandot `spy` visar i en figur strukturen av en matris. De nollskilda elementen markeras med runda ringar.

Med kommandot `full` kan man göra en gles matris icke-gles igen.

Kommandot `spdiags` skapar en gles bandmatris (en bandmatris är en matris där de nollskilda elementen finns på några av diagonalerna).

```
>> n = 5;
>> e = ones(n,1);
>> A = spdiags([-e 2*e -e],[-2 0 1],n,n);
>> full(A)
ans =
     2    -1     0     0     0
     0     2    -1     0     0
    -1     0     2    -1     0
     0    -1     0     2    -1
     0     0    -1     0     2
```

Koefficientmatrisen i uppgift 6 lab3.1 är exempel på en bandmatris. Matrisen består dessutom av många nollor och relativt få nollskilda element. I uppgift 7, lab3.1, ska man använda `spdiags` och skapa koefficientmatrisen och sedan lösa ekvationssystemet med \.

Läshänvisning och rekommenderade övningar:

Kap 5.2-5.3. Kursen har ännu inte tagit upp `for`-loopar på allvar, så koncentrera dig på operatorer (`^ / \* + -`) och kommandon (`rref`, `lu`) i kapitlet så länge.

Övningar: 5.7–5.9, 5.18, 5.19

Kommandon och begrepp som berörts i föreläsningen:

Operatorer: `^ / * + -`

`size`

`diag`, `eye`

`sparse`, `full`, `spy`, `sqdiags`

`rref`, `lu`