5 Linjära ekvationssystem.

Vi skall studera numerisk lösning av system av n linjära ekvationer i n obekanta,

$$\left\{\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &=& b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &=& b_2 \\ \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &=& b_n \end{array}\right.$$

Som med matris- och vektorbeteckningar kan skrivas

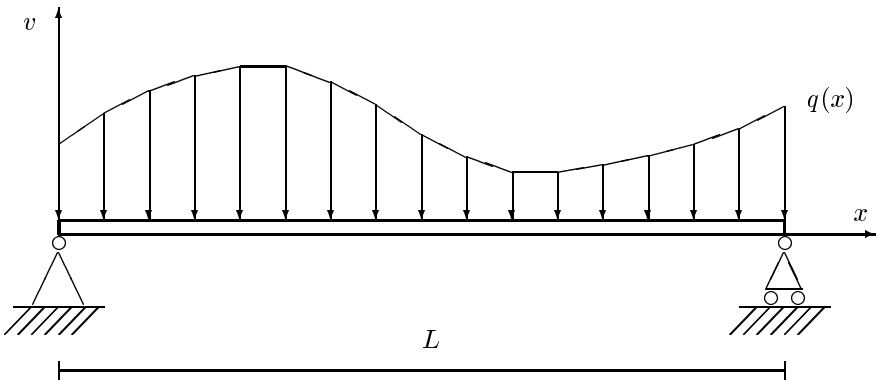
$$Ax = b,$$

där

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Tydligen är A en kvadratisk $n \times n$-matris, x och b är n-vektorer. Att lösa linjära ekvationssystem ingår ofta som delproblem vid numeriska beräkningar av olika slag. Vi kommer t.ex. i senare kapitel att lösa system av ickelinjära ekvationer genom att lösa en följd av linjära ekvationssystem. Nu skall vi se på ett exempel där vi löser ett randvärdesproblem för en ordinär differentialekvation genom att approximera derivatan med differenskvoter och lösa några linjära ekvationssystem.

Exempel 5.1. En rak balk, med variabelt yttröghetsmoment $I(x)$ och elasticitetsmodul E, ligger på två stöd och påverkas av ett tryck $q(x)$.



Moment $M(x)$ och utböjning $v(x)$ uppfyller differentialekvationen

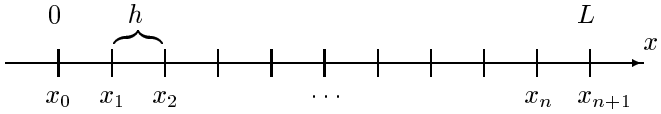
$$\left\{\begin{array}{l} M''(x) = -q(x) \\ EI(x)\,v''(x) = -M(x) \end{array}\right.$$

med randvillkoren

$$\left\{\begin{array}{l} M(0) = M(L) = 0 \\ v(0) = v(L) = 0 \end{array}\right.$$

Vi vill bestämma $M(x)$ och $v(x)$.

Låt $x_i = ih$, $i = 0, 1, \cdots, n+1$ med $h = \frac{L}{n+1}$ vara en indelning av intervallet $0 \le x \le L$.



Låt M_i och v_i beteckna approximationer av $M(x_i)$ respektive $v(x_i)$. Vi ersätter $M''(x)$ och $v''(x)$ med centraldifferenskvoter i punkterna $x_1, x_2, \cdots, x_n$, (se kapitel 3)

$$\left\{\begin{array}{l} \frac{M_{i+1}-2M_i+M_{i-1}}{h^2} = -q(x_i) \quad i = 1, 2, \cdots, n \\ M_0 = M_{n+1} = 0 \\ \frac{v_{i+1}-2v_i+v_{i-1}}{h^2} = -\frac{M_i}{E\,I(x_i)} \quad i = 1, 2, \cdots, n \\ v_0 = v_{n+1} = 0 \end{array}\right.$$

Med matriser kan detta skrivas

$$\begin{cases} AM = b \\ Av = r(M) \end{cases}$$

där

$$A = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}, \quad M = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix},$$

$$b = -\begin{bmatrix} q(x_1) \\ q(x_2) \\ \vdots \\ q(x_{n-1}) \\ q(x_n) \end{bmatrix}, \quad r(M) = -h^2 \begin{bmatrix} \frac{M_1}{EI(x_1)} \\ \frac{M_2}{EI(x_2)} \\ \vdots \\ \frac{M_{n-1}}{EI(x_{n-1})} \\ \frac{M_n}{EI(x_n)} \end{bmatrix}$$

Vi skall se hur vi kan använda Gauss-elimination på ett sätt som effektivt löser vårt problem. ■

Vi skall alltså lösa det linjära ekvationssystemet

$$Ax = b, \tag{5.1}$$

där A är en kvadratisk matris.

Om koefficientmatrisen A är reguljär (ickesingulär) så har (5.1) en entydigt bestämd lösning. Då A är singulär så har (5.1) oändligt många resp. inga lösningar beroende på om b kan skrivas som en linjär kombination av kolonnerna i A eller inte. För exempel 5.1 är det uppenbart att om koefficientmatrisen vore singulär, så hade vi gjort en felaktig formulering av problemet. I resten av det här avsnittet antar vi att A är reguljär.

Elementära transformationsmatriser.

En elementär transformationsmatris är en matris av typen

$$M = I - \sigma uv^T,$$

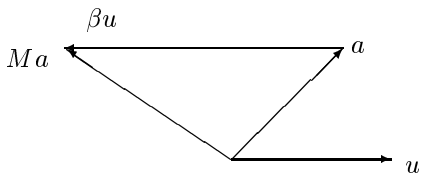
där I är enhetsmatrisen, dvs. matrisen med 1:or på diagonalen och 0:or för övrigt, σ är en skalär och u och v är vektorer. Dess invers är

$$M^{-1} = I - \alpha uv^T,$$

med $\alpha = -\sigma/(1 - \sigma v^T u)$.

Låt oss se vad som händer då vi multiplicerar en vektor a med transformationsmatrisen M. Vi får

$$Ma = a - \sigma uv^T a = a - \beta u \quad (\beta = \sigma v^T a)$$



Vi skall använda sådana här matriser även i senare kapitel. Men nu skall vi använda transformationen till elimination, då har vi $\sigma = 1$, $v = e_k$ (k:te enhetsvektorn, dvs. vektorn med en 1:a på plats k och 0:or för övrigt) och $u = m_k$, där de k första elementen i m_k är noll och övriga väljs på lämpligt sätt. Vi har alltså

$$M_k = I - m_k e_k^T$$

och då gäller att

$$M_k^{-1} = I + m_k e_k^T.$$

Nu skall vi se vad som händer då vi multiplicerar en matris A med transformationsmatrisen M_k. Beteckna raderna i A med $A_1, \cdots, A_n$ och observera att $e_k^T A = A_k$, dvs. multiplikation med e_k från vänster plockar ut k:te raden k ur A. Vi får

$$M_k A = A - m_k e_k^T A = A - m_k A_k =$$

$$= \begin{bmatrix} A_1 \\ \vdots \\ A_k \\ A_{k+1} \\ \vdots \\ A_n \end{bmatrix} - \begin{bmatrix} 0 \cdot A_k \\ \vdots \\ 0 \cdot A_k \\ m_{k+1k} \cdot A_k \\ \vdots \\ m_{nk} \cdot A_k \end{bmatrix} = \begin{bmatrix} A_1 \\ \vdots \\ A_k \\ A_{k+1} - m_{k+1k} \cdot A_k \\ \vdots \\ A_n - m_{nk} \cdot A_k \end{bmatrix},$$

där

$$e_k = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ och } m_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ m_{k+1k} \\ \vdots \\ m_{nk} \end{bmatrix}.$$

5.1 Gauss-elimination och LU-faktorisering.

En matris A kan alltid faktoriseras som

$$PA = LU,$$

där P är en s.k. permutationsmatris som ordnar om raderna i A på lämpligt sätt, L är en nedåt triangulär matris med ettor på diagonalen och U är en uppåt triangulär matris.

Vi skall se hur man LU-faktoriserar en matris A med en systematisk Gauss-elimination. För enkelhets skull gör vi inga radbyten.

Så här gör man.

Vi låter $A^{(1)} = A$ och sedan gör vi elementära transformationer tills vi får en triangulär form.

$$A^{(1)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix}$$

Första steget i Gauss-eliminationen ger matrisen

$$A^{(2)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix},$$

där

$$a_{ik}^{(2)} = a_{ik}^{(1)} - m_{i1} a_{1k}^{(1)},\ m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, i = 2, \cdots, n, k = 1, \cdots, n.$$

Låter vi

$$m_1 = [\, 0\, m_{21}\ m_{31} \cdots m_{n1}\,]^T$$

så gäller $A^{(2)} = M_1 A^{(1)}$ med

$$M_1 = I - m_1 e_1^T = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -m_{21} & 1 & 0 & \cdots & 0 \\ -m_{31} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -m_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Andra steget ger

$$A^{(3)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} \end{bmatrix},$$

där

$$a_{ik}^{(3)} = a_{ik}^{(2)} - m_{i2}a_{2k}^{(2)}, m_{i2} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}, i = 3, \cdots, n, k = 2, \cdots, n.$$

Vi låter

$$m_2 = [\, 0\ 0\ m_{32} \cdots m_{n2}\,]^T$$

och får $A^{(3)} = M_2 A^{(2)}$ med

$$M_2 = I - m_2 e_2^T = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & -m_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -m_{n2} & 0 & \cdots & 1 \end{bmatrix}.$$

Vi fortsätter på detta sätt tills vi får en uppåt triangulär matris U

$$A^{(n)} = M_{n-1}A^{(n-1)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{bmatrix} = U.$$

Det gäller nu att

$$A^{(n)} = M_{n-1}A^{(n-1)} = M_{n-1}M_{n-2}\cdots M_1 A^{(1)},$$

och därmed har vi

$$A = A^{(1)} = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1}A^{(n)} = LA^{(n)} = LU,$$

där

$$L = M_1^{-1}M_2^{-1}\cdots M_{n-1}^{-1}.$$

Vi påminner oss att

$$M_k^{-1} = I + m_k e_k^T.$$

Nu gäller att

$$M_1^{-1}M_2^{-1} = I + m_1 e_1^T + m_2 e_2^T + m_1 e_1^T m_2 e_2^T = I + m_1 e_1^T + m_2 e_2^T,$$

eftersom $e_1^T m_2 = 0$.

På samma sätt kan vi fortsätta att analysera produkten som bygger upp L och finner då att

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & 0 \\ m_{31} & m_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n3} & \cdots & 1 \end{bmatrix},$$

dvs. L är nedåt triangulär. De element som ingår i L är multiplikatorerna från Gauss-eliminationen.

Normalt ordnar man om raderna i A under eliminationens gång, så att avrundningsfel tillväxer så lite som möjligt. Detta för att vi skall få en så noggrann lösning som möjligt.

Hur får man fram lösningen?

Man kan lösa ekvationen $Ax = b$ genom att:

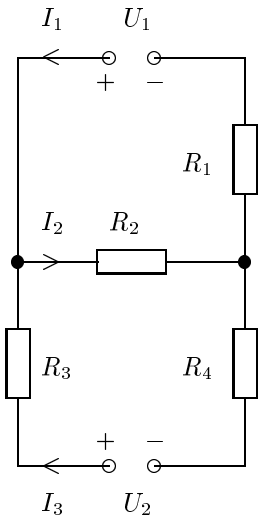
(1). Faktorisera A på formen $PA = LU$

(2). Lösa systemet $Ly = Pb$ (Framåtsubstitution)

(3). Lösa systemet $Ux = y$ (Bakåtsubstitution)

Faktoriseringen (1) kräver $\sim \frac{n^3}{3}$ operationer medan lösning av de triangulära systemen (2) och (3) kräver endast $\sim \frac{n^2}{2}$ operationer. Vi observerar att om man har flera olika högerled men med samma matris, så görs LU-faktoriseringen en gång och arbetet per högerled blir sedan endast $\sim n^2$ operationer.

Exempel 5.2. Antag att vi vill bestämma strömmarna I_1, I_2 och I_3 i följande likströmskrets



$U_1 = 8$ volt

$U_2 = 9$ volt

$R_1 = 4$ ohm

$R_2 = 2$ ohm

$R_3 = 3$ ohm

$R_4 = 2$ ohm

Vi får då det linjära ekvationssystemet

$$\begin{bmatrix} 1 & -1 & 1 \\ 4 & 2 & 0 \\ 0 & 2 & 5 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \\ 9 \end{bmatrix}$$

och vi ser vad vi kan göra med systemet i MATLAB.

```
>> A=[1 -1 1; 4 2 0; 0 2 5]
A =
     1    -1     1
     4     2     0
     0     2     5
>> b=[0; 8; 9]
b =
     0
     8
     9
>> x=A\b
x =
     1
     2
     1
>> [L,U]=lu(A)
L =
    0.2500   -0.7500    1.0000
    1.0000         0         0
         0    1.0000         0
```

```
U =
    4.0000    2.0000         0
         0    2.0000    5.0000
         0         0    4.7500
>> y=L\b
y =
    8.0000
    9.0000
    4.7500
>> x=U\y
x =
     1
     2
     1
```

Vi ser att MATLAB ger faktoriseringen $A = \tilde{L}U$ där $\tilde{L} = P^{-1}L$, i det här fallet har vi

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

eftersom $P\tilde{L}$ blir uppåt triangulär. ■

En matris A som är symmetrisk, dvs. $A^T = A$, och positivt definit, dvs. $x^T Ax > 0$ för alla $x \neq 0$, kan LU-faktoriseras stabilt utan radbyten. Detta gäller även diagonaldominanta matriser, dvs. matriser med $|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$ för alla i och med sträng olikhet för något i.

Om matrisen A är symmetrisk och positivt definit så kan den faktoriseras

$$A = C^T C$$

där C är uppåt triangulär. Detta kallas för Choleski-faktorisering och kräver $\sim \frac{n^3}{6}$ operationer.

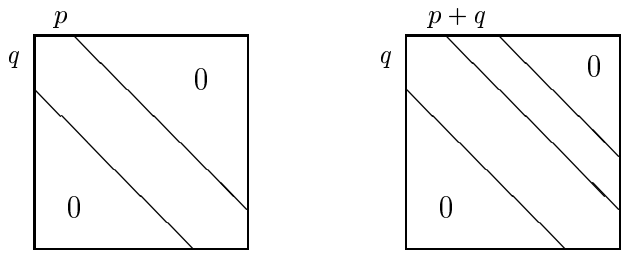
Exempel 5.3. Vi ser på matrisen A från exempel 5.1. Denna matris är symmetrisk och negativt definit ($x^T Ax < 0$), så $B = -A$ är symmetrisk och positivt definit. Vi kan alltså Choleski-faktorisera B, en kontroll med MATLAB ger

```
>> n=5;
>> B=eye(n,n)-diag(ones(n-1,1),1); B=B+B'
B =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
     0     0     0    -1     2
>> C=chol(B)
C =
    1.4142   -0.7071         0         0         0
         0    1.2247   -0.8165         0         0
         0         0    1.1547   -0.8660         0
         0         0         0    1.1180   -0.8944
         0         0         0         0    1.0954
>> C'*C
ans =
    2.0000   -1.0000         0         0         0
   -1.0000    2.0000   -1.0000         0         0
         0   -1.0000    2.0000   -1.0000         0
         0         0   -1.0000    2.0000   -1.0000
         0         0         0   -1.0000    2.0000
```

■

Glesa matriser.

En matris där många element är noll sägs vara en gles matris. Om de från noll skilda elementen ligger samlade runt huvuddiagonalen så sägs matrisen ha en bandstruktur. För bandmatriser finns det varianter av Gauss-elimination som tar vara på bandstrukturen. Många linjära ekvationssystem från tekniska tillämpningar har en bandstruktur, t.ex. matrisen i exempel 5.1.



En matris med bandbredden $d = p+q+1$ kan faktoriseras till en kostnad av $\sim nqp$ operationer, om inga radbyten görs. Kostnaden blir $\sim nq(p+q)$ operationer om man byter rader. Lösningen av de triangulära systemen kräver $\sim nd$ operationer.

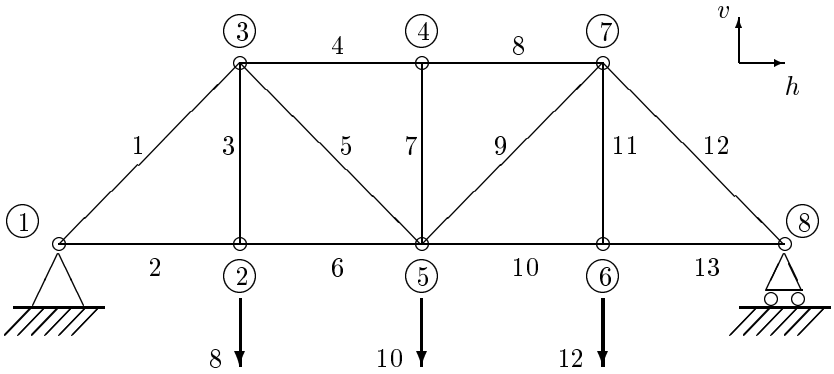
Exempel 5.4. I exempel 5.1 såg vi på utbuktningen hos en balk. Vi fick två linjära ekvationssystem med samma koefficientmatris

$$\begin{cases} AM = b \\ Av = r(M) \end{cases}$$

Nu faktoriserar vi $A = LU$ med $\sim n$ räkneoperationer. Löser först $Lz = q$ och $UM = z$, därefter löser vi $Lw = r(M)$ och $Uv = w$. ■

För allmänna glesa matriser finns speciella metoder som bygger på Gauss-elimination, som tar vara på glesheten. För att beskriva dessa metoder behöver man bl.a. använda grafteori. Vi går inte in på dessa metoder i denna kursen, men vi skall se på ett exempel på en allmänt gles matris.

Exempel 5.5. Krafterna i de olika grenarna av fackverket i figuren skall bestämmas då angivna yttre krafter är anbringade.



Genom att ansätta kraftjämvikt i horisontal- och vertikalled i knutpunkterna får vi ett linjärt ekvationssystem för de sökta krafterna i fackverkets grenar. Att detta blir välbestämt för aktuellt fackverk följer av resultat i mekaniken.

Låt $u = \sin(\frac{\pi}{4}) = \frac{1}{\sqrt{2}}$ och låt x vara vektorn med de sökta krafterna. Vi får då ekvationerna

$-x_2 + x_6 = 0$	h knut 2	$-ux_1 + x_4 + ux_5 = 0$	h knut 3
$x_3 - 8 = 0$	v knut 2	$-ux_1 - x_3 - ux_5 = 0$	v knut 3
$-x_4 + x_8 = 0$	h knut 4	$-ux_5 - x_6 + ux_9 + x_{10} = 0$	h knut 5
$-x_7 = 0$	v knut 4	$ux_5 + x_7 + ux_9 - 10 = 0$	v knut 5
$-x_{10} + x_{13} = 0$	h knut 6	$-x_8 - ux_9 + ux_{12} = 0$	h knut 7
$x_{11} - 12 = 0$	v knut 6	$-ux_9 - x_{11} - ux_{12} = 0$	v knut 7
$-ux_{12} - x_{13} = 0$	h knut 8		

Detta blir med matrisbeteckningar $Ax = b$ med

$$A = \begin{bmatrix}
0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-u & 0 & 0 & 1 & u & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-u & 0 & -1 & 0 & -u & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -u & -1 & 0 & 0 & u & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & u & 0 & 1 & 0 & u & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -u & 0 & 0 & u & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -u & 0 & -1 & -u & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -u & -1
\end{bmatrix}, \quad b = \begin{bmatrix}
0 \\ 8 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 10 \\ 0 \\ 12 \\ 0 \\ 0 \\ 0
\end{bmatrix}.$$

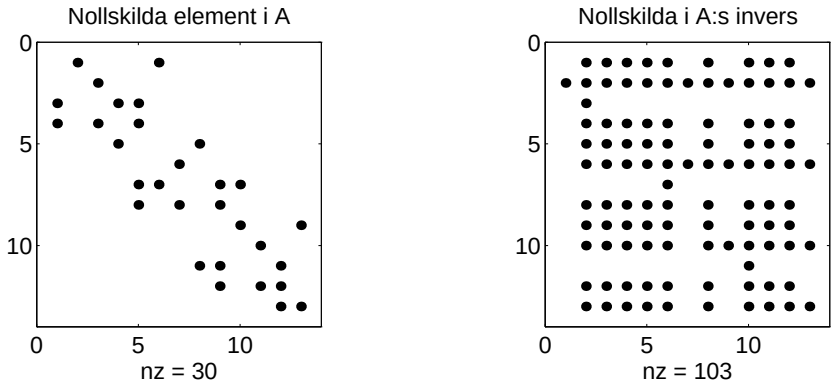
Vi matar in de nollskilda elementen med motsvarande rad- och kolonnindex

```
>> u=1/sqrt(2);
>> rad=[ 1 1 2  3 3 3  4  4  4  5 5  6  7  7 7  7 8 8 8  9  9 10 11 11 11 12 12 12 13 13 ];
>> kol=[ 2 6 3  1 4 5  1  3  5  4 8  7  5  6 9 10 5 7 9 10 13 11  8  9 12  9 11 12 12 13 ];
>> ele=[-1 1 1 -u 1 u -u -1 -u -1 1 -1 -u -1 u  1 u 1 u -1  1  1 -1 -u  u -u -1 -u -u -1 ];
```

Med följande kommandon genererar vi i MATLAB den glesa matrisen A, högerledsvektorn b och beräknar lösningen.

```
>> A=sparse(rad,kol,ele);
>> b=[0 8 0 0 0 0 0 10 0 12 0 0 0]';
>> x=A\b
x =
  -19.7990
   14.0000
    8.0000
  -20.0000
    8.4853
   14.0000
         0
  -20.0000
    5.6569
   16.0000
   12.0000
  -22.6274
   16.0000
```

Vänsterdivisionen löser ekvationssystemet genom att göra en faktorisering av A som tar vara på dess gleshetsstruktur. Med `>> spy(A)` ser vi att bara 30 av de 169 elementen i matrisen är skilda från noll. Vi gör `>> spy(inv(A))` och ser att inversen A^{-1} är en nästan fylld matris, detta är typiskt för inversen till glesa matriser.



■

Gauss-elimination är en s.k. direkt metod. En annan huvudtyp av metoder är s.k. iterativa metoder, dessa används bl.a. i samband med lösning av partiella differentialekvationer. Vi går dock inte in på dessa metoder i denna kursen.

5.2 Lösningsnoggrannhet.

Indata, dvs. koefficientmatrisen A och högerledet b, i det linjära ekvationssystemet

$$Ax = b$$

är ofta behäftade med osäkerheter. De kan t.ex. bero på uppmätta storheter som inte kan bestämmas exakt eller att talen inte kan representeras exakt i datorn.

I stället för att lösa $Ax = b$ kommer vi lösa systemet

$$\bar{A}\bar{x} = \bar{b}$$

där $\bar{A} \approx A$ och $\bar{b} \approx b$.

Vektor- och matrisnormer.

Normen av en vektor eller matris är ett positivt tal som ger information om vektorns eller matrisens storlek. Normer används för att jämföra olika vektorer eller matriser.
Några vektornormer är (x är en vektor med n komponenter);

$$\| x \|_1 = \sum_{i=1}^{n} |x_i| \quad \text{summanormen,}$$

$$\| x \|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} \quad \text{Euklidiska normen,}$$

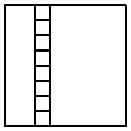
$$\| x \|_\infty = \max_{1 \le i \le n} |x_i| \quad \text{maximumnormen.}$$

För varje vektornorm kan man definiera en matrisnorm enligt

$$\| A \| = \max_{\|x\| \neq 0} \frac{\| Ax \|}{\| x \|}. \tag{5.2}$$

Man kan visa att

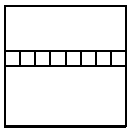
$$\| A \|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|,$$



$$\| A \|_2 = \sqrt{\lambda_{\max}(A^T A)},$$

där $\lambda_{\max}(A^T A)$ är det största egenvärdet[1] till $A^T A$,

$$\| A \|_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|.$$



En viktig egenskap hos matrisnormer definierade enligt (5.2) är att

$$\| Ax \| \le \| A \| \| x \|,$$

för alla vektorer x. En matrisnorm med denna egenskap kallas subordinerad.

Störningsresultat.

Låt $\delta A = \bar{A} - A$, $\delta b = \bar{b} - b$ och $\delta x = \bar{x} - x$. Om vi först antar att $\delta A = 0$ så gäller

$$A(x + \delta x) = b + \delta b,$$

$$Ax = b$$

dvs. δx uppfyller ekvationen $A\delta x = \delta b$ så

$$\delta x = A^{-1}\delta b.$$

[1] Vad ett egenvärde till en matris är skall vi lära oss senare i kursen.

Vi har således

$$\| \delta x \| \leq \| A^{-1} \| \| \delta b \| .$$

Vidare gäller

$$\| b \| = \| Ax \| \leq \| A \| \| x \|,$$

varav följer att

$$\frac{\| \delta x \|}{\| x \|} \leq \| A^{-1} \| \| A \| \frac{\| \delta b \|}{\| b \|}$$

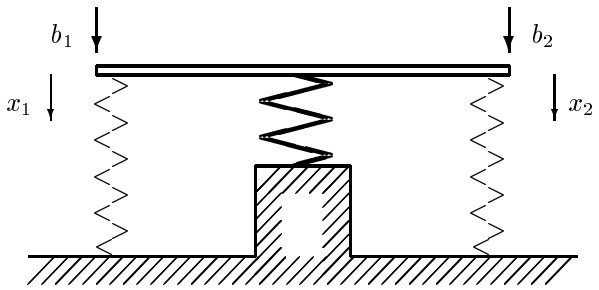
dvs.

$$\frac{\| \delta x \|}{\| x \|} \leq \kappa(A) \frac{\| \delta b \|}{\| b \|},$$

där $\kappa(A) = \| A^{-1} \| \| A \|$ är konditionstalet[2] för matrisen A. I ord kan denna olikhet formuleras; Relativa felet i lösningen är mindre än relativa felet i högerledet multiplicerat med konditionstalet för koefficientmatrisen A.

Ett linjärt ekvationssystem är välkonditionerat om konditionstalet är litet (dvs. lösningen är väl bestämd) och det är illakonditionerat om konditionstalet är stort (dvs. lösningen är dåligt bestämd).

Exempel 5.6. Betrakta fjädersystemet



En balk är upplagd på en central fjäder med styvhet $k_0 = 4000$ och två sidofjädrar med styvhet $k_1 = k_2 = 1$, de senare för att motverka rotation. Vid lasterna b_1 och b_2 får vi förskjutningarna x_1 och x_2 från balkens neutralläge.

Systemets totala energi ges av

$$E(x_1, x_2) = \frac{k_1}{2} x_1^2 + \frac{k_2}{2} x_2^2 + \frac{k_0}{2} \left(\frac{x_1 + x_2}{2} \right)^2 - b_1 x_1 - b_2 x_2$$

Jämvikt får vi då energin har sitt minimum[3], dvs. då $\nabla E(x_1, x_2) = 0$. I vårt fall får vi

$$\nabla E(x_1, x_2) = \begin{bmatrix} k_1 + \frac{k_0}{4} & \frac{k_0}{4} \\ \frac{k_0}{4} & k_2 + \frac{k_0}{4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

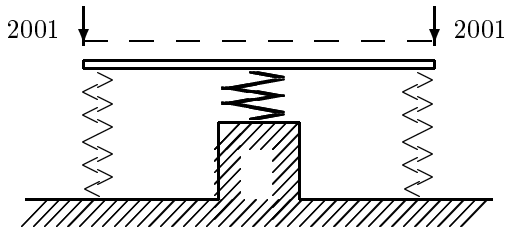
dvs.

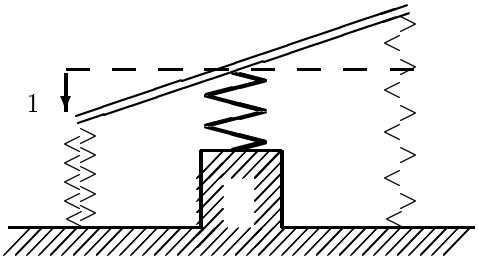
$$\begin{bmatrix} 1001 & 1000 \\ 1000 & 1001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

När krafterna är lika eller nästan lika är systemet instabilt, små förändringar i krafterna ger avsevärda förändringar i förskjutningarna. För vissa andra förhållande mellan krafterna, som till höger i figuren, är problemet stabilt. Stora förändringar i krafterna ger då marginella förändringar i förskjutningarna.

[2]Konditionstalet $\kappa(A)$ beräknas normalt inte, det skulle kosta för mycket. Men det finns program, i bl.a. LINPACK, som på ett effektivt sätt uppskattar konditionstalet.

[3]Om E bara beror av en variabel x så gäller att $E'(x) = 0$ i minpunkten, då E beror av två (eller fler) variabler så motsvaras ekvationen av $\nabla E(x_1, x_2) = 0$, där ∇E betecknar den s.k. gradienten som är en vektor som består av derivator med avseende på de olika variablerna. Mer om detta lär du dig i flervariabel analysen.





T.ex. ger $b_1 = b_2 = 2001$ lösningen $x_1 = x_2 = 1$ medan $b_1 = 2002$, $b_2 = 2000$, en förändring med 0.05%, ger lösningen $x_1 = 2$, $x_2 = 0$, som har förändrats med 100%.
För $b_1 = 1$, $b_2 = -1$ blir lösningen $x_1 = 1$, $x_2 = -1$ och för $b_1 = 2.001$, $b_2 = 0$, en förändring med 100 % har vi lösningen $x_1 = 1.001$, $x_2 = -1$. Förändringen blir endast 0.07% i det här fallet. ■

Om vi nu istället antar att $\delta b = 0$ så får vi

$$(A + \delta A)(x + \delta x) = b,$$

$$Ax = b$$

dvs. δx uppfyller ekvationen $A\delta x + \delta A(x + \delta x) = 0$ så

$$\delta x = -A^{-1}\delta A(x + \delta x).$$

Detta ger

$$\| \delta x \| \leq \| A^{-1} \| \| \delta A \| \| x + \delta x \|,$$

så vi har

$$\frac{\| \delta x \|}{\| x + \delta x \|} \leq \| A^{-1} \| \| A \| \frac{\| \delta A \|}{\| A \|} = \kappa(A) \frac{\| \delta A \|}{\| A \|}.$$

I vänsterledet har vi ungefär relativa felet i lösningen och i högerledet relativa felet i koefficientmatrisen A multiplicerat med dess konditionstal.

Gauss-elimination och avrundningsfel.

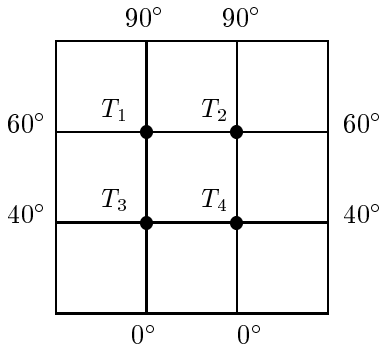
Gauss-elimination med lämpliga radpermutationer är stabil, den ger exakt lösning till

$$(A + E)x^* = b,$$

där elementen i E oftast är av ungefär samma storleksordning som avrundningsfelen i elementen i A. Något bättre än detta kan man inte förvänta sig av någon numerisk metod, eftersom ett sådant fel får vi bara vi lagrar matrisen A i datorn.

5.3 Övningar.

1. Betrakta temperaturfördelningen i en kvadratisk platta. Temperaturen i plattans kanter varierar som angivet i figuren. På sidorna är plattan isolerad. Man önskar bestämma temperaturen i de inre punkterna T_1, T_2, T_3 och T_4. Man antar att temperaturen i en inre punkt är medelvärdet av temperaturerna i de fyra grannpunkterna (i norr, söder, väster och öster). Sätt upp och lös det linjära ekvationssystem som ger de sökta temperaturerna. Hur blir matrisens struktur om fler likformigt fördelade inre punkter betraktas? (Då måste givetvis temperaturen vara känd i fler punkter på kanterna.)



2. Vi vill lösa det linjära ekvationssystemet $Ax = b$ där,

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 2 & 3 \\ -1 & -3 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 3 \\ 2 \end{bmatrix}.$$

LU-faktorisera A utan pivotering. Använd sedan dessa faktorer för att beräkna lösningen x genom att först lösa $Ly = b$ och sedan $Ux = y$.

3. Ett datorprogram löser ett tridiagonalt ekvationssystem med 300 obekanta på 3 sekunder. Hur lång tid tar det att lösa ett tridiagonalt system med 30 000 obekanta med samma program?

4. För att lösa ett triangulärt ekvationssystem med 100 obekanta krävdes 1 sekunds körtid på en viss dator. Uppskatta körtiden vid lösandet (på samma dator) av

(a) ett triangulärt system med 300 obekanta

(b) ett allmänt (fyllt) system med 100 resp. 300 obekanta?

5. Choleski-faktorisera matrisen

$$A = \begin{bmatrix} 4 & 2 & 6 \\ 2 & 2 & 5 \\ 6 & 5 & 17 \end{bmatrix},$$

dvs. $A = CC^T$, där C vänstertriangulär.

6. Antag att vi har en LU-faktorisering av en matris A och vill lösa ekvationssystemet $A^T x = c$. Hur kan vi då utnyttja faktorerna L och U?

7. Vi vill lösa randvärdesproblemet

$$\begin{cases} y'' + y = \sin(x),\ 0 < x < \pi/2 \\ y(0) = 1,\ y(\pi/2) = 0 \end{cases}$$

Inför en indelning av intervallet $0 < x < \pi/2$, som i exempel 5.1, och använd en centraldifferenskvot för att approximera y'' i indelningspunkterna. Vi får då ett linjärt ekvationssystem vars lösning ger approximationer av y i dessa punkter. Vilken struktur har det linjära ekvationssystemet och hur många operationer åtgår för att lösa det?

8. Betrakta systemet

$$\begin{bmatrix} A & b \\ c^T & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} e \\ f \end{bmatrix}$$

där A är en $n \times n$-matris, b, c och e är n-vektorer, d och f är skalärer.

Visa att $x = v - yw$ där

$$Av = e, \quad Aw = b$$

och

$$y = \frac{c^T v - f}{c^T w - d}$$

om A ickesingulär och $c^T w \neq d$. Detta kallas block-elimination.

9. Bestäm utböjningen $v(x)$ för balken i exempel 5.1 för $E = 2 \cdot 10^{11}$ N/m^2, $L = 1$ m , $q(x) = (10 - 9x) \cdot 500$ N/m och $I(x) = (2 - x) \cdot 10^{-7}$ m^4. Tag t.ex. $h = 0.2, 0.1, 0.05, 0.025$.

10. Vad blir konditionstalet för matrisen $A = \begin{bmatrix} 2 & 3 \\ 2 & 2 \end{bmatrix}$? Använd vilken matrisnorm du vill.

5.4 Lösningar.

1. Medelvärdesbildningen ger

$$\begin{cases} T_1 = \frac{T_2+90+60+T_3}{4} \\ T_2 = \frac{60+90+T_1+T_4}{4} \\ T_3 = \frac{T_4+T_1+40+0}{4} \\ T_4 = \frac{40+T_2+T_3+0}{4} \end{cases} \Leftrightarrow \begin{cases} 4T_1 - T_2 - T_3 = 150 \\ -T_1 + 4T_2 - T_4 = 150 \\ -T_1 + 4T_3 - T_4 = 40 \\ -T_2 - T_3 + 4T_4 = 40 \end{cases}$$

Med matrisbeteckningar

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 150 \\ 150 \\ 40 \\ 40 \end{bmatrix}$$

med lösningen

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 61.25 \\ 61.25 \\ 33.75 \\ 33.75 \end{bmatrix}$$

2. Direkt med MATLAB

```
>> A=[1 2 1; 2 2 3; -1 -3 0]
A =
     1     2     1
     2     2     3
    -1    -3     0
>> b=[0; 3; 2]
b =
     0
     3
     2
>> [L,U]=lu(A)
L =
    0.5000   -0.5000    1.0000
    1.0000         0         0
   -0.5000    1.0000         0

U =
    2.0000    2.0000    3.0000
         0   -2.0000    1.5000
         0         0    0.2500
>> y=L\b
y =
    3.0000
    3.5000
    0.2500
>> x=U\y
x =
     1
    -1
     1
>> r=A*x-b
r =
     0
     0
     0
```

3. 300 sek.

4.(a) 9 sek.

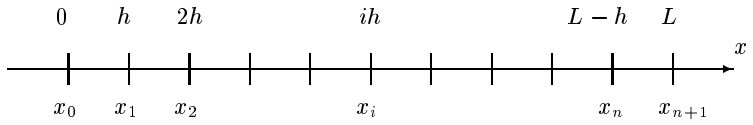
(b) 67 resp. 1800 sek.

5. Direkt med MATLAB

```
>> A=[4 2 6; 2 2 5; 6 5 17]
A =

     4     2     6
     2     2     5
     6     5    17
>> C=chol(A)
C =
     2     1     3
     0     1     2
     0     0     2
>> C'*C
ans =
     4     2     6
     2     2     5
     6     5    17
```

6. Eftersom $A = LU$ har vi $A^T = U^T L^T$. Lösningen till $A^T x = c$ ges därför av att först lösa $U^T y = c$ och sedan lösa $L^T x = y$.

7. Låt $x_i = ih$, $i = 0, 1, \cdots, n+1$, $h = \frac{L}{n+1}$ vara ett nät på $0 \le x \le L$



Låt y_i beteckna en approximation av $y(x_i)$ och ersätt $y''(x_i)$ med centraldifferenskvoten $D_+D_-y(x_i)$

$$\begin{cases} \frac{y_{i+1}-2y_i+y_{i-1}}{h^2} + y_i = f_i \;\; i = 1, \cdots, n \\ y_0 = 1, \;\; y_{n+1} = 0 \end{cases}$$

där $f_i = \sin(x_i)$.

Vi kan skriva om ekvationerna lite

$$\begin{cases} y_0 = 1 \\ y_{i-1} + (-2+h^2)y_i + y_{i+1} = h^2 f_i \;\; i = 1, \cdots, n \\ y_{n+1} = 0 \end{cases}$$

och på matrisform blir detta

$$\begin{bmatrix} -2+h^2 & 1 & & & & \\ 1 & -2+h^2 & 1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 1 & -2+h^2 & 1 & \\ & & & 1 & -2+h^2 & \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} h^2 f_1 - 1 \\ h^2 f_2 \\ \vdots \\ h^2 f_{n-1} \\ h^2 f_n \end{bmatrix}$$

Detta linjära ekvationssystem kan man lösa med $\mathcal{O}(n)$ räkneoperationer.

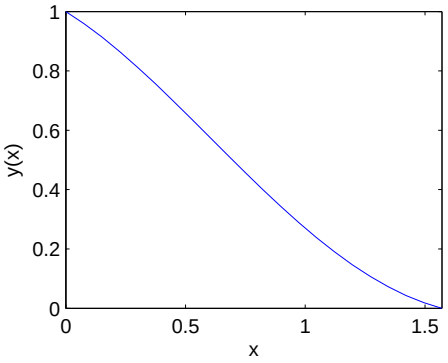
Lösning med MATLAB. Först ser vi på hjälptexten till **spdiags** hur man lagrar en bandmatris.

```
>> help spdiags
 SPDIAGS Sparse matrix formed from diagonals.
    SPDIAGS, which generalizes the function "diag", deals with three
    matrices, in various combinations, as both input and output.

    [B,d] = SPDIAGS(A) extracts all nonzero diagonals from the m-by-n
    matrix A.  B is a min(m,n)-by-p matrix whose columns are the p
```

```
    nonzero diagonals of A.  d is a vector of length p whose integer
    components specify the diagonals in A.

    B = SPDIAGS(A,d) extracts the diagonals specified by d.
    A = SPDIAGS(B,d,A) replaces the diagonals of A specified by d with
        the columns of B.  The output is sparse.
    A = SPDIAGS(B,d,m,n) creates an m-by-n sparse matrix from the
        columns of B and places them along the diagonals specified by d.

    Roughly, A, B and d are related by
        for k = 1:p
            B(:,k) = diag(A,d(k))
        end

    Example: These commands generate a sparse tridiagonal representation
    of the classic second difference operator on n points.
        e = ones(n,1);
        A = spdiags([e -2*e e], -1:1, n, n)

    Some elements of B, corresponding to positions "outside" of A, are
    not actually used.  They are not referenced when B is an input and
    are set to zero when B is an output.  If a column of B is longer than
    the diagonal it's representing, elements of super-diagonals of A
    correspond to the lower part of the column of B, while elements of
    sub-diagonals of A correspond to the upper part of the column of B.
...
```

Och nu lösning av vårt problem (lämpligen som en scriptfil).

```
n=20; L=pi/2;
h=L/(n+1); x=h*[0:n+1]';
% generera koefficientmatrisen
e=ones(n,1);
A=spdiags([e (-2+h^2)*e e],[-1 0 1],n,n);
% generera högerledet
f=sin(x(2:end-1));
b=h^2*f; b(1)=b(1)-1;
% lös ekvationssystemet
y=A\b;
% fyll på randvärdena och rita
y=[1;y;0];
plot(x,y)
xlabel('x'), ylabel('y(x)')
```



8. Insättning ger

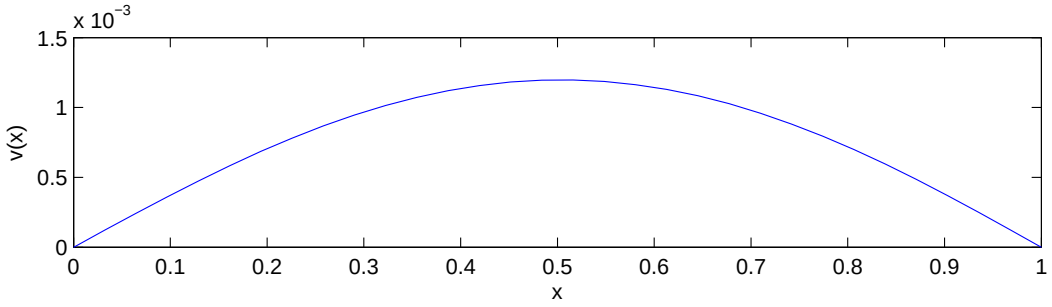
$$Ax = A(v - yw) = Av - yAw = e - yb \;\Rightarrow\; Ax + by = e$$

och

$$c^T x + dy = c^T(v - yw) + dy = c^T v - y(c^T w - d) = c^T v - (c^T v - f) = f$$

9. Vi gör en scriptfil till MATLAB och ökar värdet på n tills skillnaden mellan beräkningsresultaten är tillräckligt liten.

```
E=2e11; L=1;
n=30;
h=L/(n+1);
e=ones(n,1);
A=spdiags([-e 2*e -e],[-1 0 1],n,n);
C=chol(A);
x=[h:h:L-h]';
q=h^2*(10-9*x)*500;
M=C\(C'\q);
r=h^2/E*M./((2-x)*1e-7);
v=C\(C'\r);
X=[0;x;L];
V=[0;v;0];
plot(X,V)
xlabel('x'), ylabel('v(x)')
```



6 Överbestämda linjära ekvationssystem.

I det här kapitlet skall vi se på numerisk lösning av överbestämda system av m linjära ekvationer i n obekanta ($m > n$),

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{cases}$$

Som med matris- och vektorbeteckningar kan skrivas

$$Ax = b,$$

där

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}.$$

Matrisen A är rektangulär med fler rader än kolonner.

Exempel 6.1. För bestämning av längdutvidgningskoefficienten λ för en metall gjordes ett experiment, där en metallstång upphettades och längden L mättes vid olika temperaturer T

L	8.78	8.93	9.06	9.25	9.40
T	20.0	25.5	30.2	36.8	41.0

Man använder en linjär modell för att beskriva sambandet mellan längd och temperatur

$$L(T) = L_0 + \lambda T$$

Detta leder till ett överbestämt linjärt ekvationssystem

$$L(T_i) = L_0 + \lambda T_i = L_i, \quad i = 1, 2, \cdots, 5$$

eller

$$Ax = b$$

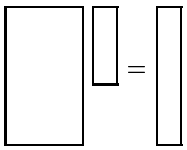
där

$$A = \begin{bmatrix} 1 & 20.0 \\ 1 & 25.5 \\ 1 & 30.2 \\ 1 & 36.8 \\ 1 & 41.0 \end{bmatrix}, \; x = \begin{bmatrix} L_0 \\ \lambda \end{bmatrix}, \; b = \begin{bmatrix} 8.78 \\ 8.93 \\ 9.06 \\ 9.25 \\ 9.40 \end{bmatrix}.$$

■

Vi skall alltså söka en numerisk lösning av överbestämda system av m linjära ekvationer i n obekanta ($m > n$),

$Ax = b$



Det kommer normalt inte att finnas någon vektor x som exakt uppfyller alla ekvationerna. Minstakvadratlösningen är den vektor x som gör den s.k. residualvektorn $r = Ax - b$ så kort som möjligt i den Euklidiska normen, dvs. vi skall lösa problemet

$$\min_x \| Ax - b \|_2$$

Ofta är antal ekvationer m mycket större än antalet obekanta n, speciellt då vi anpassar en modell till mätdata. Då är modellens parametrar de obekanta som skall bestämmas.

Om koefficientmatrisen A har full rang (dvs. alla kolonner i A är linjärt oberoende) så har minstakvadratproblemet en entydigt bestämd lösning som man kan räkna fram med hjälp av en s.k. QR-faktorisering[4].

[4] Om rangen för A är k (dvs. antalet linjärt oberoende kolonner är k) och $k < n$, så är minstakvadratlösningen inte entydigt bestämd utan $n - k$ parametriga familjer av lösningar existerar. Man kan då använda en variant av QR-faktorisering där man sorterar om kolonnerna i A, eller så använder man sig av en singulärvärdesfaktorisering, SVD. Vid anpassning av en modell till mätdata kan det hända att någon parameter i modellen är överflödig. Detta leder till att matrisen får linjärt beroende kolonner, dvs. att rangen k blir mindre än n.

Ortogonala vektorer och matriser.

Två vektorer x och y är ortogonala (vinkelräta) om $x^T y = 0$.

En kvadratisk matris Q kallas ortogonal om dess kolonner är ortogonala mot varandra och har var och en längden 1.

Om Q är en ortogonalmatris så gäller det att $Q^T Q = I$ och $QQ^T = I$, där I är enhetsmatrisen. Vidare gäller att $\| Qx \|_2 = \| x \|_2$, där x är en godtycklig vektor.

Vi kommer att behöva en speciell typ av ortogonala matriser, s.k. elementära speglingar

$$U = I - 2uu^T, \| u \|_2 = 1.$$

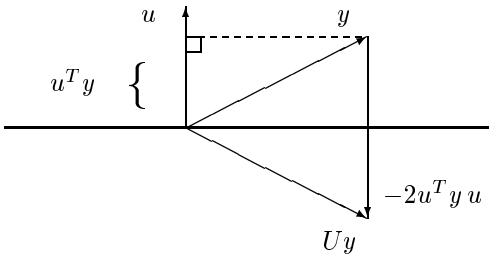
För dessa gäller att;

$$\begin{array}{ll} U^T = U & \text{(symmetri)} \\ U^T U = I & \text{(ortogonalitet)} \\ U^{-1} = U & \end{array}$$

Låt oss se vad som händer då vi multiplicerar en vektor y med matrisen U;

$$Uy = (I - 2uu^T)y = y - 2uu^T y = y - 2u^T yu.$$

Vi får en spegling i det plan som har u som enhetsnormal.



Vi kommer vilja avbilda en vektor y på en vektor z (då måste $\| z \|_2 = \| y \|_2$), dvs. vi vill bestämma U så att

$$Uy = z.$$

Utnyttjar vi att $U = (I - 2uu^T)$ får vi

$$y - 2u^T yu = z,$$

eller

$$u = \frac{1}{2u^T y}(y - z).$$

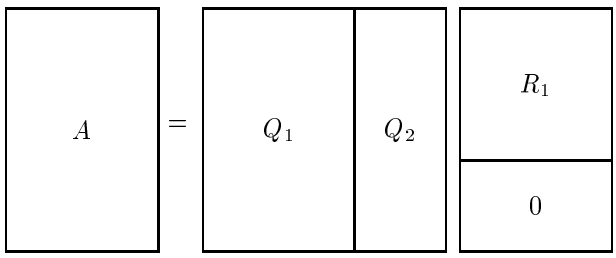
Om vi nu tar $u = (y - z)/ \| y - z \|_2$ så blir precis som vi ville $Uy = z$.

6.1 QR-faktorisering.

En $m \times n$-matris A kan faktoriseras

$$A = QR$$

där Q är en ortogonal $m \times m$-matris och R är en uppåt triangulär $m \times n$-matris.



Om A har full rang (alla kolonner linjärt oberoende) så är R_1 reguljär.

Så här gör man.

Vi börjar med att låta $A^{(1)} = A$ och multiplicerar med elementära speglingar tills vi har en uppåt triangulär matris.

$$A^{(1)} = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} & \cdots & a_{3n}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1}^{(1)} & a_{m2}^{(1)} & a_{m3}^{(1)} & \cdots & a_{mn}^{(1)} \end{bmatrix}$$

Vi vill att $U_1 a_1 = r_{11} e_1$ där $r_{11} = \pm \parallel a_1 \parallel_2$ och a_1 är 1:a kolonnen i $A^{(1)}$. Så vi tar

$$U_1 = I - 2u_1 u_1^T$$

med

$$u_1 = (a_1 - r_{11} e_1) / \parallel a_1 - r_{11} e_1 \parallel_2,$$

där vi väljer tecknet på r_{11} så att kancellation undviks, och får då

$$A^{(2)} = U_1 A^{(1)} = \begin{bmatrix} r_{11} & a_{12}^{(2)} & a_{13}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{m2}^{(2)} & a_{m3}^{(2)} & \cdots & a_{mn}^{(2)} \end{bmatrix}$$

Vid nästa transformation vill vi inte förstöra 0:orna i 1:a kolonnen i $A^{(2)}$. Detta undviker vi genom att ta $U_2 = I - 2u_2 u_2^T$, med första komponenten i u_2 lika med 0. Då kommer 1:a kolonnen och första raden i $A^{(2)}$ inte att ändras i transformationen. Resten av u_2 väljs på liknade sätt som för u_1. Vi får

$$A^{(3)} = U_2 A^{(2)} = \begin{bmatrix} r_{11} & a_{12}^{(2)} & a_{13}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & r_{22} & a_{23}^{(3)} & \cdots & a_{2n}^{(3)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{m3}^{(3)} & \cdots & a_{mn}^{(3)} \end{bmatrix}$$

Sedan fortsätter vi på samma sätt. Efter n transformationer har vi

$$A^{(n+1)} = U_n A^{(n)} = \begin{bmatrix} r_{11} & a_{12}^{(2)} & a_{13}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & r_{22} & a_{23}^{(3)} & \cdots & a_{2n}^{(3)} \\ 0 & 0 & r_{33} & \cdots & a_{3n}^{(4)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & r_{nn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = R,$$

där R_1 och därmed R uppåt triangulär. Nu gäller att

$$A = A^{(1)} = U_1^T A^{(2)} = \cdots = U_1^T U_2^T \cdots U_n^T A^{(n+1)} = Q A^{(n+1)} = QR.$$

Man behöver inte göra radbyten, som för Gauss-elimination, däremot förekommer det att man gör kolonnbyten så att r_{kk} blir så stor som möjligt, i varje steg. Kostnaden för en QR-faktorisering är $\sim mn^2$ operationer.

Hur får vi fram lösningen?

Vi vill minimera $\| r \|_2 = \| Ax - b \|_2$ med avseende på x.

$$\| r \|_2^2 = \| Ax - b \|_2^2 = \| QRx - b \|_2^2 = \| Q(Rx - Q^T b) \|_2^2 =$$

$$= \| Rx - Q^T b \|_2^2 = \| R_1 x - Q_1^T b \|_2^2 + \| Q_2^T b \|_2^2 .$$

Om A har full rang så kan den första termen nollställas genom att vi tar

$$x = R_1^{-1} Q_1^T b.$$

Den andra termen är oberoende av x.

Exempel 6.2. I exempel 6.1 ville vi bestämma längdutvidgningskoefficienten λ för en metall. I MATLAB matar vi in data och löser problemet så här:

```
>> L=[8.78; 8.93; 9.06; 9.25; 9.40];
>> T=[20.0; 25.5; 30.2; 36.8; 41.0];
>> A=[ones(size(T)), T];
>> x=A\L
x =
     8.1876
     0.0292
```

dvs. $L_0 = 8.1876$ och $\lambda = 0.0292$. Vi ser efter att vi får samma resultat genom att använda QR-faktorisering

```
>> [Q,R]=qr(A)
Q =
   -0.4472   -0.6337   -0.4193   -0.3538   -0.3121
   -0.4472   -0.3080    0.0366    0.4478    0.7094
   -0.4472   -0.0296    0.8456   -0.1926   -0.2169
   -0.4472    0.3613   -0.2128    0.6116   -0.5001
   -0.4472    0.6100   -0.2500   -0.5130    0.3196
R =
   -2.2361  -68.6473
         0   16.8843
         0         0
         0         0
         0         0
>> [m,n]=size(A); Q1=Q(:,1:n); R1=R(1:n,1:n);
>> x=R1\(Q1'*L)
x =
    8.1866
    0.0292
```

Slutligen ser vi på residualvektorn

```
>> r=A*x-L
r =
   -0.0088
    0.0020
    0.0094
    0.0123
   -0.0149
```

Att komponenterna i r blir både positiva och negativa är typiskt för minstakvadratmetoden. ■

6.2 Övningar.

1. Strålningsintensiteten I hos en blandning av två radioaktiva ämnen avtar med tiden enligt

$$I(t) = A_0 \mathrm{e}^{-\lambda t} + B_0 \mathrm{e}^{-\mu t}$$

Man känner halveringstiderna för ämnena och därav vet man att $\lambda = 0.29$ och $\mu = 0.17$. Genom mätningar har följande data erhållits

t	1	2	3	4	5	6
I	8.01	6.18	4.71	3.68	2.86	2.20

Bestäm halterna A_0 och B_0 av ämnena i blandningen med minstakvadratmetoden.

2. Strålningsintensiteten I hos ett radioaktivt preparat avtar med tiden t enligt

$$I(t) = I_0 \mathrm{e}^{-\lambda t}$$

Genom mätningar har följande data erhållits

t	1	2	3	4	5	6
I	6.32	4.76	3.51	2.67	2.01	1.48

Bestäm I_0 och λ med minstakvadratmetoden. Ledning: Logaritmera så att ni får ett linjärt problem.

3. För bestämning av fjäderkonstanten k för en fjäder gjordes ett försök, där fjädern belastades med en kraft F och dess längd l mättes. Vid försöket fick man följande mätdata

F	1	2	3	4	5
l	7.97	10.2	14.2	16.0	21.2

Hookes lag säger att fjäderns längd beror linjärt på den kraft som belastar fjädern enligt

$$l = e + \frac{F}{k}$$

där e är fjäderns längd då den är obelastad. Bestäm k med minstakvadratmetoden.

4. Man vill bestämma amplitud A och fasvinkel v hos en harmonisk svängning

$$u = A \sin(t + v)$$

med hjälp av följande mätvärden

t (grader)	0	30	60	90
u	2.86	10.2	14.8	15.4

Linjärisera och använd minstakvadratmetoden.

5. Använd minstakvadratmetoden för att anpassa ett polynom av lämpligt gradtal till följande mätdata för vattenångas maximala partialtryck p i luft vid temperaturen T

T	40	45	50	55	60	65	70	75	80	85	90	95	100
p	55.3	71.9	92.5	118	149.4	187.5	233.7	289.1	355.1	433.6	525.8	633.9	760

Gör en s.k. residualanalys: Anpassa ett polynom av grad 1. Rita upp residualen, ser den ut som ett 2:a gradspolynom, så anpassa ett polynom av grad 2. Rita upp residualen osv.

6.3 Lösningar.

1. Vi får ett överbestämt linjärt ekvationssystem

$$\begin{cases} A_0 \mathrm{e}^{-\lambda t_1} + B_0 \mathrm{e}^{-\mu t_1} = I_1 \\ A_0 \mathrm{e}^{-\lambda t_2} + B_0 \mathrm{e}^{-\mu t_2} = I_2 \\ \vdots \end{cases}$$

Lösning med MATLAB

```
>> t=[1 2 3 4 5 6]';
>> I=[8.01 6.18 4.71 3.68 2.86 2.20]';
>> lambda=0.29; mu=0.17;
>> A=[exp(-lambda*t) exp(-mu*t)];
>> x=A\I;
>> A0=x(1), B0=x(2)
A0 =
    8.4199
B0 =
    2.0301
```

2. Logaritmering ger $\ln(I(t)) = \ln(I_0) - \lambda t = a - \lambda t$ och vi får ett överbestämt linjärt ekvationssystem

$$\begin{cases} a - \lambda t_1 = \ln(I_1) \\ a - \lambda t_2 = \ln(I_2) \\ \qquad \vdots \end{cases}$$

Vi bestämmer sedan $a = \ln(I_0)$ och λ.

```
>> t=[1 2 3 4 5 6]';
>> I=[6.32 4.76 3.51 2.67 2.01 1.48]';
>> A=[ones(size(t)) -t]; b=log(I);
>> x=A\b;
>> I0=exp(x(1)), lambda=x(2)
I0 =
   8.4469e+00
lambda =
   2.8909e-01
```

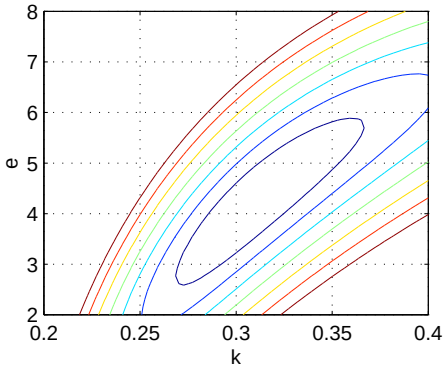
3. Vi får ett överbestämt linjärt ekvationssystem

$$e + \frac{F_i}{k} = l_i \quad i = 1, \cdots, 5$$

```
>> F=[1 2 3 4 5]';
>> l=[7.97 10.2 14.2 16.0 21.2]';
>> A=[ones(size(F)) F];
>> x=A\l;
>> e=x(1)
e =
    4.2360
>> k=1/x(2)
k =
    0.3100
>> Residual=norm(A*x-l)
Residual =
    1.6041
```

Vi ritar en s.k. responsyta

```
>> N=40; M=45;
>> Res=zeros(N,M);
>> emin=2; emax=8; kmin=0.2; kmax=0.4;
>> e=linspace(emin,emax,N);
>> k=linspace(kmin,kmax,M);
>> for n=1:N
     for m=1:M
       Res(n,m)=norm(A*[e(n); 1/k(m)]-l);
     end
   end
>> Resmin=min(min(Res));
>> contour(k,e,Res,[1:0.4:4]*Resmin)
>> xlabel('k'), ylabel('e')
```



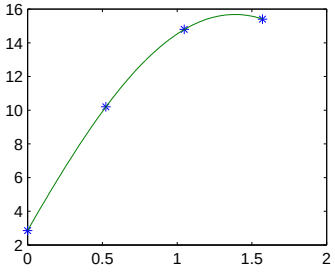
4. Sambandet $\sin(\alpha + \beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$ ger $u = A\sin(t + v) = A\cos(v)\sin(t) + A\sin(v)\cos(t)$ eller

$$u = c_0 \sin(t) + c_1 \cos(t)$$

där $c_0 = A\cos(v)$ och $c_1 = A\sin(v)$.

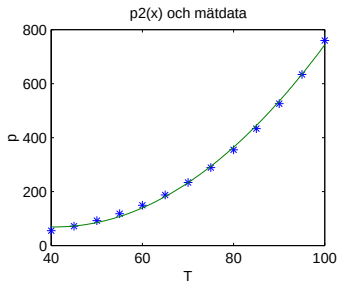
Bestämmer vi sedan c_0 och c_1 så har vi $A = \sqrt{c_0^2 + c_1^2}$ och $v = \arctan\left(\frac{c_1}{c_0}\right)$.

```
>> t_deg=[0 30 60 90]'; u=[2.86 10.2 14.8 15.4]';
>> t=t_deg*pi/180;        % Omvandla till radianer
>> M=[sin(t) cos(t)];     % Kefficientmatris
>> c=M\u;                 % Lösning
>> A=norm(c); v=atan(c(2)/c(1));
>> w=inline('A*sin(t+v)','A','v','t');  % Modellen
>> disp([A v norm(u-w(A,v,t))])
   1.5680e+01   1.8419e-01   2.4473e-02
>> te=linspace(t(1),t(end),200);  % Täta t-värden
>> plot(t,u,'*',te,w(A,v,te))     % för snygg graf
```



5. Vi prövar med ett 2:a gradspolynom.

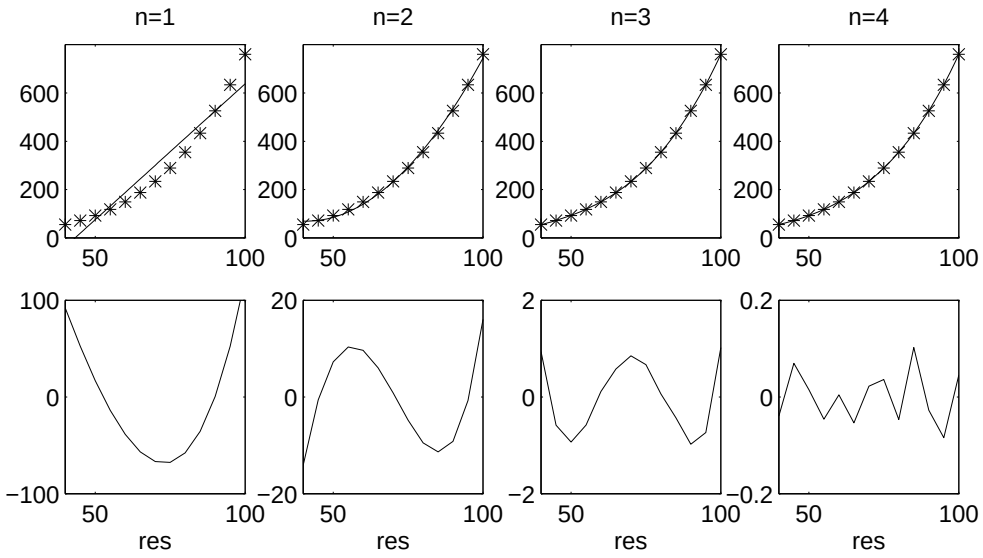
```
>> T=[40; 45; 50; 55; ... ];
>> p=[55.3; 71.9; 92.5; 118; ... ];
>> c=polyfit(T,p,2);
>> Tm=linspace(40,100,200);
>> plot(T,p,'*',Tm,polyval(c,Tm))
>> xlabel('T'), ylabel('p')
>> title('p2(x) och mätdata')
>> res=p-polyval(c,T);
>> norm(res)
ans =
   32.7487
```



Systematisk undersökning av gradtal $n = 1, 2, \cdots, 6$

```
>> for n=1:6
      c=polyfit(T,p,n); disp([n norm(p-polyval(c,T))])
   end

    1.0000  218.5073
    2.0000   32.7487
    3.0000    2.5804
    4.0000    0.1896
    5.0000    0.1430
    6.0000    0.1422
```

Residualanalys:



7 Egenvärdesproblem.

Vi skall studera hur man löser egenvärdesproblem av typen

$$Ax = \lambda x,$$

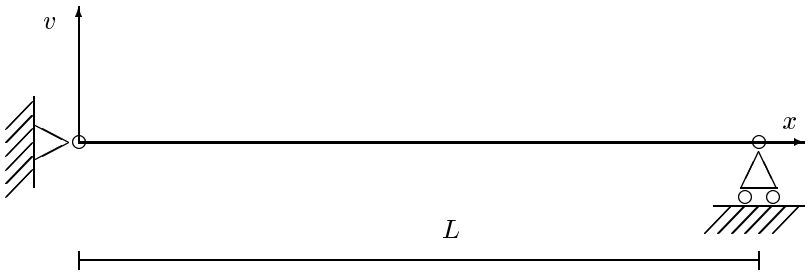
där A är en kvadratisk matris, x är en vektor skild från nollvektorn och λ är en skalär (ett tal). x kallas för egenvektor och λ kallas för egenvärde.

Att lösa sådana egenvärdesproblem ingår ibland som delproblem vid numeriska beräkningar. Vissa tillämpningar, t.ex. vibrationsproblem, leder direkt till ett egenvärdesproblem för en matris.

Exempel 7.1. Utböjningen v hos en ledlagrad stav av längd L som belastas med lasten P beskrivs av följande egenvärdesproblem för en ordinär differentialekvation

$$\begin{cases} EI\, v'' + Pv = 0,\ 0 < x < L \\ v(0) = v(L) = 0 \end{cases}$$

där EI är böjstyvheten.



Vi diskretisera problemet genom att införa en ekvidistant indelning av intervallet, precis som i exempel 5.1, och använda en centraldifferensapproximation av derivatan. Låter vi v_i beteckna approximationen av $v(x_i)$ får vi

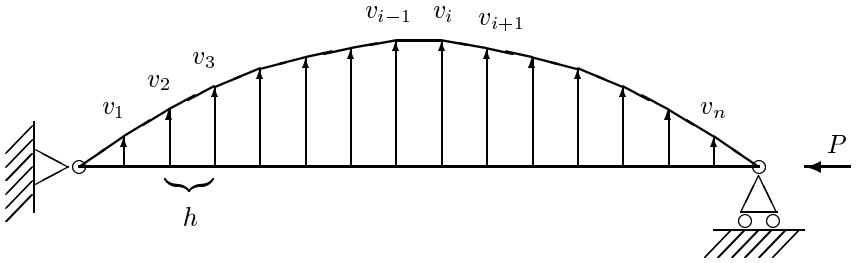
$$\begin{cases} v_{i+1} - 2v_i + v_{i-1} = -h^2 \frac{P}{EI} v_i \quad i = 1, 2, \cdots, n \\ v_0 = v_{n+1} = 0 \end{cases}$$

Med matriser kan detta skrivas

$$Av = \lambda v$$

där

$$A = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \quad v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} \quad \text{och } \lambda = -h^2 \frac{P}{EI}$$



■

Ett egenvärdesproblem för en $n \times n$-matris har n stycken egenvärden med tillhörande egenvektorer. Är matrisen osymmetrisk och egenvärdena multipla, så kan egenvektorerna tillhörande samma egenvärde sammanfalla. Matrisen kallas då defekt.

Egenvektorer tillhörande olika egenvärden är alltid linjärt oberoende. En reell symmetrisk matris har alltid reella egenvärden och egenvektorerna kan väljas ortogonala.

Om x är en egenvektor så är den en icketrivial lösning till det homogena linjära ekvationssystemet $(A-\lambda I)x = 0$ för något λ, dvs. matrisen $A-\lambda I$ är singulär då λ är ett egenvärde till A. Speciellt om T är en triangulär matris så är $T-\lambda I$ singulär precis då $\lambda = t_{ii}$, dvs. på diagonalen i matrisen T finner vi dess egenvärden.

Om V är en kvadratisk ickesingulär matris så har A och $B = V^{-1}AV$ samma egenvärden. En sådan transformation (som bevarar egenvärdena) kallas en likformighetstransformation. Matriserna A och B säges vara similära. Om x är en egenvektor till A så är $y = V^{-1}x$ en egenvektor till B. Man kan visa att det finns ortogonala likformighetstransformationer så att A är similär med en uppåt triangulär matris T.

Vi skall nu se på några metoder för att bestämma egenvärden och egenvektorer. För den viktigaste metoden, den s.k. QR-metoden, gäller att de beräknade egenlösningarna är exakta för ett stört problem.

I detta sammanhang kan det vara bra att känna till ett störningsresultat av Bauer-Fike;

Om $\lambda_1, \lambda_2, \cdots, \lambda_n$ är egenvärden till A och vi samlar ihop motsvarande egenvektorer $x_1, x_2, \cdots, x_n$ som kolonner i en matris X så gäller

$$\min_i |\lambda_i - \mu| \leq \kappa(X) \parallel \delta A \parallel$$

där μ egenvärde till $A + \delta A$.

Speciellt om A är symmetrisk så är $\kappa(X) = 1$ (eftersom egenvektorerna kan väljas ortogonala) och vi får

$$\min_i |\lambda_i - \mu| \leq \parallel \delta A \parallel$$

dvs. en liten störning av en symmetrisk matris ger alltid endast en liten störning av dess egenvärden.

7.1 Potensmetoden.

Om x är en egenvektor till matrisen A så kan motsvarande egenvärde λ beräknas genom att man bildar den s.k. Rayleigh-kvoten

$$\lambda = \frac{x^T A x}{x^T x}.$$

Potensmetoden går ut på att man bildar en följd av vektorer

$$u_{k+1} = Au_k, k = 0, 1, \cdots,$$

som förhoppningvis konvergerar mot en egenvektor till A. Rayleigh-kvoterna

$$\sigma_k = \frac{u_k^T A u_k}{u_k^T u_k}$$

ger en approximation av motsvarande egenvärde.

Potensmetoden kan nu formuleras;

Givet en vektor u_0, upprepa för $k = 0, 1, \cdots,$

(1). $w_k = u_k / \parallel u_k \parallel_2$

(2). $u_{k+1} = Aw_k$

(3). $\sigma_k = w_k^T u_{k+1}$

Normaliseringen i steg (1) görs för att undvika överspill, normen på vektorerna kan annars växa mycket snabbt när vi upprepade gånger multiplicerar med matrisen.

Antag att för egenvärdena λ_i till A gäller

$$|\lambda_1| \leq \cdots \leq |\lambda_{n-1}| < |\lambda_n|$$

och att motsvarande egenvektorer x_i, $i = 1, 2, \cdots, n$ är linjärt oberoende.

Då kan u_0 skrivas som en linjär kombination av dessa egenvektorer,

$$u_0 = \sum_{i=1}^{n} \alpha_i x_i.$$

Nu gäller att

$$u_k = Au_{k-1} = A^2 u_{k-2} = \cdots = A^k u_0 =$$
$$= \sum_{i=1}^{n} \alpha_i A^k x_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k x_i,$$

där vi har utnyttjat att $Ax_i = \lambda_i x_i$. En omskrivning ger

$$u_k = \alpha_n \lambda_n^k \left(x_n + \sum_{i=1}^{n-1} \frac{\alpha_i}{\alpha_n} \left(\frac{\lambda_i}{\lambda_n} \right)^k x_i \right).$$

Eftersom λ_n är det till belopp största egenvärdet kommer $(\frac{\lambda_i}{\lambda_n})^k \to 0$ då $k \to \infty$, och därmed kommer u_k att gå mot $\alpha_n \lambda_n^k x_n$, dvs. u_k kommer att konvergera mot egenvektorn x_n tillhörande det till belopp största egenvärdet λ_n. Vidare gäller att $\sigma_k \to \lambda_n$ då $k \to \infty$.

Inversiteration.

Med potensmetoden kunde vi ju bestämma det till belopp största egenvärdet och motsvarande egenvektor. För att bestämma ett annat egenvärde och motsvarande egenvektor kan vi utnyttja att om A har egenvärdena λ_i, $i = 1, 2, \cdots, n$ så har matrisen $(A - \mu I)^{-1}$ egenvärdena $(\lambda_i - \mu)^{-1}$. Så det egenvärde till $(A - \mu I)^{-1}$ som är till belopp störst är det egenvärde till A som ligger närmast μ. Vi kan därför om vi vill bestämma det egenvärde som ligger närmast μ genom att använda potensmetoden på matrisen $(A - \mu I)^{-1}$.

Inversiterationsmetoden kan nu formuleras;

Bestäm ett skift μ nära det sökta egenvärdet λ_j och låt u_0 vara en slumpvektor, upprepa för $k = 0, 1, \cdots,$

(1). $w_k = u_k / \parallel u_k \parallel_2$

(2). $(A - \mu I)u_{k+1} = w_k$

(3). $\sigma_k = w_k^T u_{k+1}$

Nu kommer σ_k att konvergera mot $(\lambda_j - \mu)^{-1}$, så λ_j approximeras av $\mu + 1/\sigma_k$, och u_k konvergerar mot x_j. I steg (2) skall man ju lösa ett linjärt ekvationssystem för u_{k+1}, då gör man givetvis så att man LU-faktoriserar matrisen $A - \mu I$ en gång för alla och använder sedan dessa faktorer vid lösning av systemen.

Exempel 7.2. Vi ser på exempel 7.1. Låt oss bestämma den lägsta kritiska lasten med inversiteration. Det räcker med $n = 20$ nodpunkter för att ge en hygglig approximation av egenfunktionen till det första egenvärdet. Vi använder skiftet $\mu = 0$, eftersom det är egenvärdet närmast noll vi vill bestämma, och får med MATLAB

```
>> n=20;  h=1/(n+1);
>> A=eye(n,n)-diag(ones(n-1,1),1); A=A+A';  % Generera matrisen A
>> C=chol(A);                               % Choleski-faktorisera A så att A=C'*C
>> v=randn(n,1);                            % Startvektor av slumptal
>> for k=1:10,
      y=C'\v; v=C\y;
      disp(norm(v)), v=v/norm(v);
   end
```

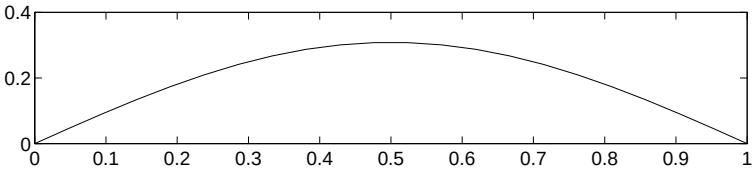
Från MATLAB får vi utskriften

```
   9.42386636347751
  17.67049115870390
  38.29286193928083
```

```
  44.25364411129021
  44.73318008568442
  44.76398842110944
  44.76593717395706
  44.76606034281692
  44.76606812749792
  44.76606861951934
```

Vi ser att approximationerna konvergerar. Nu ritar vi första bucklingsmoden

```
>> v=[0; v; 0];          % Vi fyller på randvärdena
>> x=[0:h:1]';
>> plot(x,v)             % Ritar ut bucklingsmoden
```



I nästa avsnitt skall vi plocka fram metoder som kan beräkna alla egenvärden och motsvarande egenvektorer på en gång. ■

7.2 QR-metoden.

Den s.k. QR-metoden bygger på att man gör en följd av ortogonala likformighetstransformationer av matrisen A m.h.a. QR-faktoriseringar tills man får en matris på triangulär form (diagonal form i symmetriska fallet).

Metoden kan formuleras;

> Låt $A_1 = A$. Upprepa för $k = 1, 2, \cdots$,
>
> (1). $A_k = Q_k R_k$ (QR-faktorisering)
>
> (2). $A_{k+1} = R_k Q_k$

Man kan visa att $A_{k+1} \to T$, där T triangulär (diagonal i symmetriska fallet), då $k \to \infty$ och vi finner alltså egenvärdena till A längs diagonalen på T.

Nu är A_{k+1} similär med A_k, ty

$$A_{k+1} = R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T A_k Q_k,$$

eftersom $Q_k^T Q_k = I$. Egenvektorerna till T kan vi sedan bestämma med t.ex. inversiteration som vi skall beskriva senare. Om y är en egenvektor till T så är $x = Vy$, där $V = Q_1 Q_2 \cdots$, en egenvektor till A.

Varje gång vi upprepar steg (1) måste vi utföra en ny QR-faktorisering. Detta kostar mycket beräkningsarbete. Med Householders metod kan vi med $(n-2)$ multiplikationer med elementära speglingar föra över matrisen på Hessenbergform[5] (tridiagonal form i symmetriska fallet). Denna Hessenberg matris kan vi sedan använda QR-metoden på till låg kostnad.

Låt oss se på hur Householders metod går till på ett 4×4-exempel: Vi använder oss av elementära speglingar $P = I - 2uu^T$.

$$A^{(1)} = \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix}, P_1 = I - 2u_1 u_1^T, u_1 = \begin{bmatrix} 0 \\ u_{21} \\ u_{31} \\ u_{41} \end{bmatrix}.$$

Vid vänstermultiplikation vill vi nollställa i 1:a kolonnen. Vid högermultiplikation måste vi lämna 1:a kolonnen orörd så att vi inte förstör de nollor vi skapat. Första speglingen måste därför ha $u_{11} = 0$, därmed kan vi bara nollställa nedanför den subdiagonala platsen. 1:a raden lämnas orörd vid vänstermultiplikationen.

[5] En matris är på Hessenbergform om den innehåller idel 0:or under subdiagonalen.

$$P_1^T A^{(1)} = \begin{bmatrix} x & x & x & x \\ y & y & y & y \\ 0 & y & y & y \\ 0 & y & y & y \end{bmatrix}, A^{(2)} = P_1^T A^{(1)} P_1 = \begin{bmatrix} x & z & z & z \\ y & z & z & z \\ 0 & z & z & z \\ 0 & z & z & z \end{bmatrix},$$

I nästa steg är det dax för den del av $A^{(2)}$ vi får om vi bortser från 1:a raden och kolonnen. Nu måste vi ha $u_{12} = u_{22} = 0$.

$$P_2 = I - 2u_2 u_2^T, u_2 = \begin{bmatrix} 0 \\ 0 \\ u_{32} \\ u_{42} \end{bmatrix}.$$

Och vi får vid vänster- och högermultiplikation följande

$$P_2^T A^{(2)} = \begin{bmatrix} x & z & z & z \\ y & z & z & z \\ 0 & s & s & s \\ 0 & 0 & s & s \end{bmatrix}, A^{(3)} = P_2^T A^{(2)} P_2 = \begin{bmatrix} x & z & t & t \\ y & z & t & t \\ 0 & s & t & t \\ 0 & 0 & t & t \end{bmatrix},$$

Vi ser att vi fått $A^{(3)}$ på Hessenbergform.

Allmänt går det till så att vi startar med $A^{(1)} = A$, och utför $(n-2)$ speglingar tills vi har $A^{(n-1)} = H$ en Hessenbergmatris.

Nu är A och H similära eftersom

$$A^{(n-1)} = P_{n-2}^T A^{(n-2)} P_{n-2} = P_{n-2}^T P_{n-3}^T A^{(n-3)} P_{n-3} P_{n-2} = \cdots = V^T A^{(1)} V,$$

med

$$V = P_1 P_2 \cdots P_{n-2}. \tag{7.1}$$

För att få snabbare konvergens i QR-metoden kan man införa skift, dvs. man ersätter A_k med $A_k - \mu_k I$ i metoden. Detta ger en förskjutning, skiftning, av egenvärdena. Om μ_k väljs på lämpligt sätt i varje steg så fås snabbare konvergens. Hur detta görs går vi dock inte in på här.

Vi kan nu formulera QR-metoden med skift;

Utför Householders metod på A så att vi får en similär Hessenbergmatris H. Låt $A_1 = H$.
Upprepa för $k = 1, 2, \cdots,$

(1). $(A_k - \mu_k I) = Q_k R_k$ (QR-faktorisering)

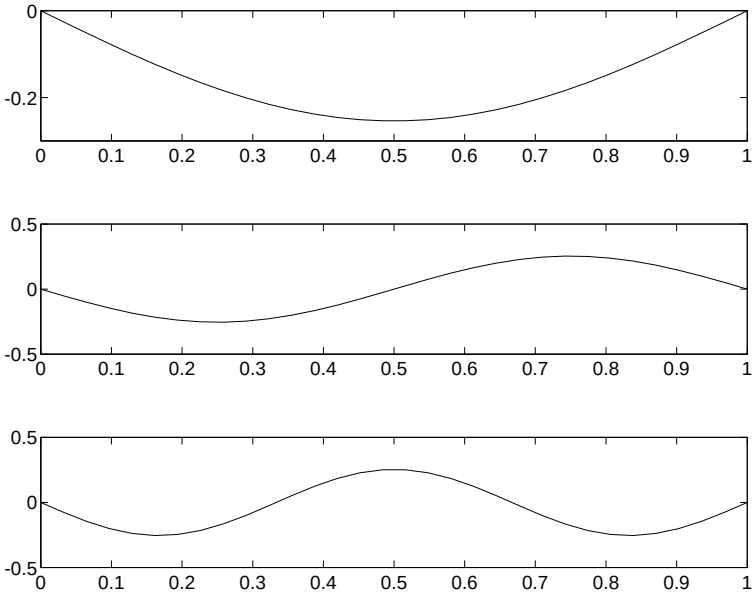
(2). $A_{k+1} = R_k Q_k + \mu_k I$

tills triangulär form (diagonal form i symmetriska fallet).

Alla matriserna $A_1, A_2, \cdots$ är på Hessenbergform, så QR-faktoriseringen i steg (1) kan utföras med litet beräkningsarbete $\sim 3n^2$ operationer. Egenvektorerna till A kan vi få som $x = Vy$, där V är likformighetstransformationen (7.1) och y är en egenvektor till H bestämd med t.ex. inversiteration. H kan LU-faktoriseras till låg kostnad.

Exempel 7.3. Vi ser på exempel 7.1 igen. Med MATLAB beräknar vi och ritar upp de tre första bucklingsmoderna. Det räcker med $n = 30$ nodpunkter för att ge en hygglig approximation av egenfunktionerna till det första, andra respektive tredje egenvärdet.

Här ser vi resultatet



Nu skall vi se hur vi fick fram resultatet. Först genererar vi matrisen

```
>> n=30;  h=1/(n+1);
>> A=eye(n,n)-diag(ones(n-1,1),1); A=A+A';
```

Sedan tar vi och beräknar egenlösningarna med

```
>> [V,D]=eig(A);
```

Egenvärdena finner vi på diagonalen i D och motsvarande egenvektorer är kolonner i V. Vi plockar ut diagonalen i D och sorterar egenvärdena i stigande ordning, därefter sorterar vi om egenvektorerna i samma ordning.

```
>> D=diag(D);
>> [D,index]=sort(D);
>> V=V(:,index);
```

Vi fyller på randvärdena med

```
>> V=[zeros(1,n); V; zeros(1,n)];
```

Och ritar ut de tre första bucklingsmoderna

```
>> x=[0:h:1]';
>> subplot(3,1,1), plot(x,V(:,1))
>> subplot(3,1,2), plot(x,V(:,2))
>> subplot(3,1,3), plot(x,V(:,3))
```

I det här enkla fallet kan man lösa problemet analytiskt och får de kritiska lasterna

$$P_{cr} = \frac{n^2\pi^2 EI}{L^2},\ n = 1, 2, \cdots$$

Men den numeriska metoden duger även till problem som inte kan lösas analytiskt. ■

7.3 Övningar.

1. Visa att $\max_i |\lambda_i| \leq \| A \|$.
2. Visa att om $Ax = \lambda x$ så gäller följande samband mellan de olika matriserna och deras egenvärdena.

Matris	αA	$A - \mu I$	A^{-1}	$(A - \mu I)^{-1}$
Egenvärde	$\alpha\lambda$	$\lambda - \mu$	λ^{-1}	$(\lambda - \mu)^{-1}$

3. Egenvärdesproblem för ordinär differentialekvation

$$\begin{cases} -u'' = \lambda u,\ 0 \le x \le L \\ u(0) = u(L) = 0 \end{cases}$$

beskriver även egensvängningar hos en sträng som är fast inspänd i båda ändar.

(a) Bestäm den lägsta frekvensen med inversiteration. Det räcker med $n = 20$ nodpunkter för att ge en hygglig approximation av egenfunktionen till det första egenvärdet. Använd skiftet $\mu = 0$, eftersom det är egenvärdet närmast noll vi vill bestämma. Rita upp approximationen av motsvarande egenfunktion.

(b) Bestäm några av de lägsta egenfrekvenserna och motsvarande egenvektorer till det diskreta problemet med **eigs** i MATLAB. Rita upp approximationerna av egenfunktionerna.

7.4 Lösningar.

1. Tag ett egenvärde λ och motsvarande egenvektor v. Då gäller att $Av = \lambda v$ och därmed

$$\frac{\| Av \|}{\| v \|} = \frac{\| \lambda v \|}{\| v \|} = \frac{|\lambda| \, \| v \|}{\| v \|} = |\lambda|$$

Alltså har vi

$$|\lambda| = \frac{\| Av \|}{\| v \|} \le \max_{x \neq 0} \frac{\| Ax \|}{\| x \|} = \| A \|$$

2. Vi tar ett godtyckligt egenvärde λ och motsvarande egenvektor v så att $Av = \lambda v$ och får

I. $B = \alpha A$: $Bv = \alpha Av = \alpha(\lambda v) = (\alpha\lambda)v$

II. $B = A - \mu I$: $Bv = (A - \mu I)v = Av - \mu v = \lambda v - \mu v = (\lambda - \mu)v$

III. $B = A^{-1}$: $Bv = A^{-1}v = \lambda^{-1}A^{-1}(\lambda v) = \lambda^{-1}A^{-1}Av = \lambda^{-1}v$

IV. $B = (A - \mu I)^{-1}$: $Bv = (\lambda - \mu)^{-1}v$ följer av II och III.

3 Vi har samma matris som i exempel 7.1. Så här kan man göra i MATLAB

```
>> n=20;
>> h=1/(n+1); e=ones(n,1);
>> A=spdiags([-e 2*e -e],[-1 0 1],n,n);   % Generera tridiag. matrisen
```

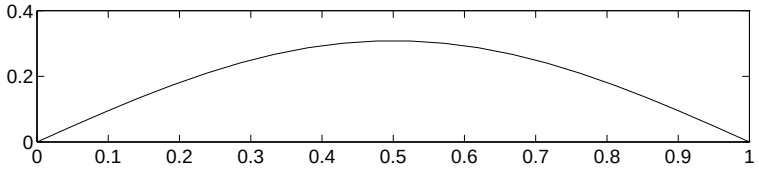
(a)
```
>> C=chol(A);                             % Choleski-faktorisera den
>> v=randn(n,1);                          % Startvektor av normalf. slumptal
>> for k=1:10,                            % Iterationen
      y=C'\v; v=C\y;
      disp(norm(v)), v=v/norm(v);
   end
```

och utskriften blir t.ex. (olika körningar ger olika startvärden)

```
  9.42386636347751
 17.67049115870390
 38.29286193928083
 44.25364411129021
 44.73318008568442
 44.76398842110944
 44.76593717395706
 44.76606034281692
 44.76606812749792
 44.76606861951934
```

Vi ser att approximationerna konvergerar. Nu ritar vi första egenmoden

```
>> v=[0; v; 0];                % Vi fyller på randvärdena
>> x=[0:h:1]';                 % Genererar nätpunkterna
>> plot(x,v)                   % Ritar ut moden
```

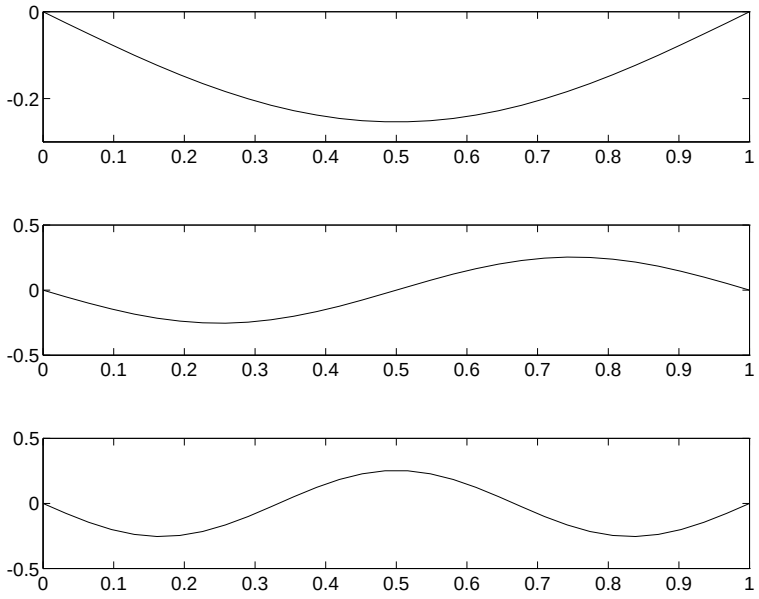


(b) Vi tar och beräknar egenlösningarna med (**eigs** är den glesa versionen av **eig**)

```
>> k=3;                        % antal egenvärden
>> [V,D]=eigs(A,k,'SM');       % SM = närmast nollan
```

Egenvärdena finner vi på diagonalen i D och motsvarande egenvektorer är kolonner i V. Vi plockar ut diagonalen i D och sorterar egenvärdena i stigande ordning, därefter sorterar vi om egenvektorerna i samma ordning. Sedan fyller vi på randvärden och ritar ut de tre första moderna

```
>> D=diag(D);
>> [D,index]=sort(D);
>> V=V(:,index);
>> V=[zeros(1,k); V; zeros(1,k)];  % randvärden
>> x=[0:h:1]';
>> subplot(3,1,1), plot(x,V(:,1))
>> subplot(3,1,2), plot(x,V(:,2))
>> subplot(3,1,3), plot(x,V(:,3))
```




```
>> help eigs
 EIGS  Find a few eigenvalues and eigenvectors of a matrix using ARPACK.
    D = EIGS(A) returns a vector of A's 6 largest magnitude eigenvalues.
    A must be square and should be large and sparse.

    [V,D] = EIGS(A) returns a diagonal matrix D of A's 6 largest magnitude
    eigenvalues and a matrix V whose columns are the corresponding eigenvectors.

    [V,D,FLAG] = EIGS(A) also returns a convergence flag.  If FLAG is 0
    then all the eigenvalues converged; otherwise not all converged.

    EIGS(AFUN,N) accepts the function AFUN instead of the matrix A.
    Y = AFUN(X) should return Y = A*X.  N is the size of A.  The matrix A
    represented by AFUN is assumed to be real and nonsymmetric.  In all these
    calling sequences, EIGS(A,...) may be replaced by EIGS(AFUN,N,...).
```

```
EIGS(A,B) solves the generalized eigenvalue problem A*V == B*V*D.  B must
be symmetric (or Hermitian) positive definite and the same size as A.
EIGS(A,[],...) indicates the standard eigenvalue problem A*V == V*D.

EIGS(A,K) and EIGS(A,B,K) return the K largest magnitude eigenvalues.

EIGS(A,K,SIGMA) and EIGS(A,B,K,SIGMA) return K eigenvalues based on SIGMA:
   'LM' or 'SM' - Largest or Smallest Magnitude
For real symmetric problems, SIGMA may also be:
   'LA' or 'SA' - Largest or Smallest Algebraic
   'BE' - Both Ends, one more from high end if K is odd
For nonsymmetric and complex problems, SIGMA may also be:
   'LR' or 'SR' - Largest or Smallest Real part
   'LI' or 'SI' - Largest or Smallest Imaginary part
If SIGMA is a real or complex scalar, EIGS finds the eigenvalues closest to
SIGMA.  In this case, B need only be symmetric (or Hermitian) positive
semi-definite and the function Y = AFUN(X) must return Y = (A-SIGMA*B)\X.

EIGS(A,K,SIGMA,OPTS) and EIGS(A,B,K,SIGMA,OPTS) specify options:
OPTS.issym: symmetry of A or A-SIGMA*B represented by AFUN [{0} | 1]
OPTS.isreal: complexity of A or A-SIGMA*B represented by AFUN [0 | {1}]
OPTS.tol: convergence: abs(d_comp-d_true) < tol*abs(d_comp) [scalar | {eps}]
OPTS.maxit: maximum number of iterations [integer | {300}]
OPTS.p: number of Lanczos vectors: K+1<p<=N [integer | {2K}]
OPTS.v0: starting vector [N-by-1 vector | {randomly generated by ARPACK}]
OPTS.disp: diagnostic information display level [0 | {1} | 2]
OPTS.cholB: B is actually its Cholesky factor CHOL(B) [{0} | 1]
OPTS.permB: sparse B is actually CHOL(B(permB,permB)) [permB | {1:N}]

EIGS(AFUN,N,K,SIGMA,OPTS,P1,...) and EIGS(AFUN,N,B,K,SIGMA,OPTS,P1,...)
provide for additional arguments which are passed to AFUN(X,P1,...).

Examples:
   A = delsq(numgrid('C',15));  d1 = eigs(A,5,'SM');
Equivalently, if dnRk is the following one-line function:
   function y = dnRk(x,R,k)
   y = (delsq(numgrid(R,k))) * x;
then pass dnRk's additional arguments, 'C' and 15, to eigs:
   n = size(A,1);  opts.issym = 1;  d2 = eigs(@dnRk,n,5,'SM',opts,'C',15);

See also EIG, SVDS, ARPACKC.
```