1 Inledning

Vi skall se på lösning av linjära ekvationssystem $\boldsymbol{Ax} = \boldsymbol{b}$, speciellt då matrisen $\boldsymbol{A}$ är stor och gles vilket är vanligt i tekniska tillämpningar. Sedan ser vi på minsta-kvadratmetoden för att anpassa linjära modeller till mätdata.

2 Linjära ekvationssystem

Linjära ekvationssystem med små koefficientmatriser, löser vi i MATLAB med backslash-kommandot `x=A\b` eller alternativt med kommandot `rref` enligt `rref([A b])`.

Backslash-kommandot används när $\boldsymbol{A}$ är en kvadratisk matris och vi vet att vi har en entydig lösning. Har vi fria variabler så använder vi `rref` som reducerar den utökade matrisen $[\boldsymbol{A}\ \boldsymbol{b}]$ till reducerad trappstegsform.

Vid numeriska beräkningar skall man *inte* bilda $\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}$, dvs. man skall inte bilda inversen och multiplicera med den. Det blir mindre effektivt och mindre noggrant, speciellt för lite större matriser.

2.1 Stora och glesa linjära ekvationssystem

En del problem har väldigt många obekanta och ger stora matriser. Finns det få kopplingar blir det många nollor i matrisen, vi får en s.k. gles matris. Vi skall se hur MATLAB hanterar detta. När väl matrisen är lagrad känner MATLAB av att det är en gles matris när vi t.ex. vill beräkna lösningen till ett linjärt ekvationssystem.

Som exempel tar vi matrisen

$$\boldsymbol{A} = \begin{bmatrix} 7 & 0 & 0 & 5 & 0 \\ 0 & 2 & -8 & 0 & 0 \\ 3 & 0 & 0 & 0 & 11 \\ 1 & 0 & 0 & 4 & 0 \\ 0 & -5 & 9 & 0 & 6 \end{bmatrix}$$

Detta är en gles matris och en lagringsmetod är att lagra tripplar (i, j, a_{ij}) för de element som är skilda från noll. Vi bildar en tabell över nollskilda element och deras rad- respektive kolonnindex.

i	1	1	2	2	3	3	4	4	5	5	5
j	1	4	2	3	1	5	1	4	2	3	5
a_{ij}	7	5	2	−8	3	11	1	4	−5	9	6

I MATLAB bildar man tre vektorer av rad- och kolonnindex samt matriselement.

```
>> ivec=[1 1 2 2 3 3 4 4 5 5 5];
>> jvec=[1 4 2 3 1 5 1 4 2 3 5];
>> aijvec=[7 5 2 -8 3 11 1 4 -5 9 6];
```

Den glesa matrisen bildas med `sparse` enligt

```
>> A=sparse(ivec,jvec,aijvec)
A =
   (1,1)         7
   (3,1)         3
   (4,1)         1
   (2,2)         2
   (5,2)        -5
   (2,3)        -8
   (5,3)         9
   (1,4)         5
   (4,4)         4
   (3,5)        11
   (5,5)         6
```

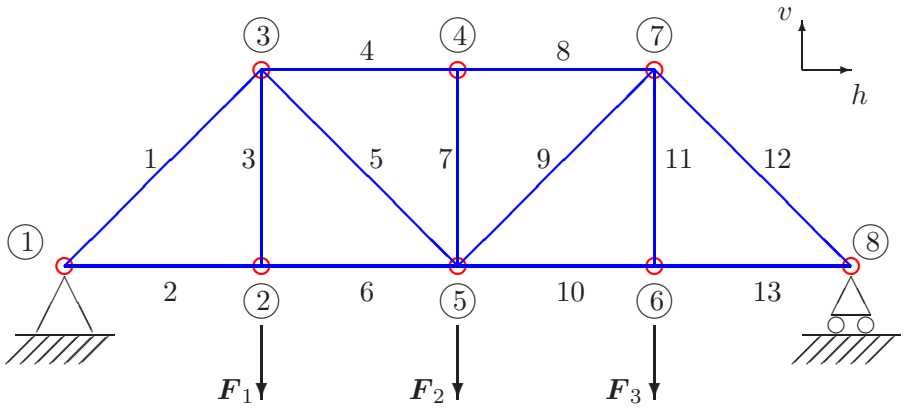
Utskriften visar alla nollskilda element och deras plats i matrisen. Med `full` omvandlar vi en gles matris till en vanlig fylld matris.

```
>> FA=full(A)
FA =
     7     0     0     5     0
     0     2    -8     0     0
     3     0     0     0    11
     1     0     0     4     0
     0    -5     9     0     6
```

Här får vi en kontroll att vi bildat rätt matris.

Det här var ingen stor matris, men vi ville bara visa principerna för hur man bildar en gles matris med `sparse`.

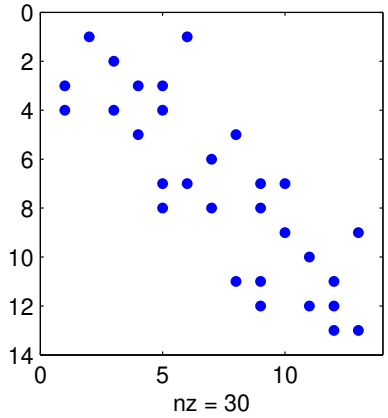
Som exempel på en tillämpning där vi får en gles matris tar vi: *Fackverk* – Krafterna i de olika grenarna av det *statiskt bestämda* fackverket i figuren nedan skall bestämmas då angivna yttre krafter är anbringade.

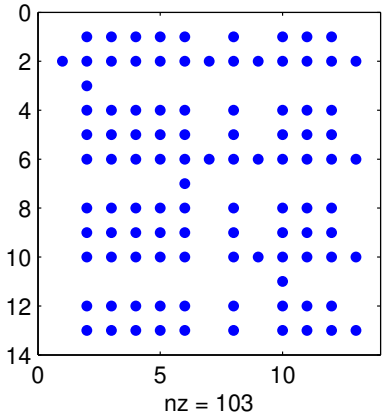


Genom att ansätta kraftjämvikt i horisontal- och vertikalled i knutpunkterna får vi ett linjärt ekvationssystem $\boldsymbol{Ax} = \boldsymbol{b}$ för de sökta krafterna i fackverkets grenar. Beroende på om vi använder friläggning eller inte blir matrisen $\boldsymbol{A}$ av storleken 13×13 eller 16×16, vi har alltså 13 eller 16 obekanta. Gör det gärna om ni hinner.

Efter att i MATLAB ha bildat den glesa matrisen $\boldsymbol{A}$, högerledsvektorn $\boldsymbol{b}$ så beräknar vi lösningen $\boldsymbol{x}$ med `x=A\b`. Eftersom matrisen är lagrad som en gles matris kommer MATLAB lösa ekvationssystemet med metoder som utnyttjar gleshetsstrukturen för att mycket effektivt och noggrant beräkna lösningen.

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 $\boldsymbol{A}^{-1}$ är en nästan fylld matris, detta är typiskt för inversen till glesa matriser. Detta är ett av skälen att man inte skall bilda inverser och multiplicera med dem, utan istället direkt lösa med lämplig metod.

Vid riktiga tillämpningar är 1000-tals eller 10000-tals obekanta inget ovanligt, även 100000-tals obekanta förekommer titt som tätt. Då kan vi prata om stort.

2.2 Bandmatriser

Linjära ekvationssystem från tekniska tillämpningar har ofta en matris med diagonaler av nollskilda element, en s.k. *bandmatris*. En sådan matris kan man bilda med `sparse` men enklare och mer effektivt är att använda funktionen `spdiags`.

Som ett första litet exempel tar vi en matris (som ni känner igen från laboration 2)

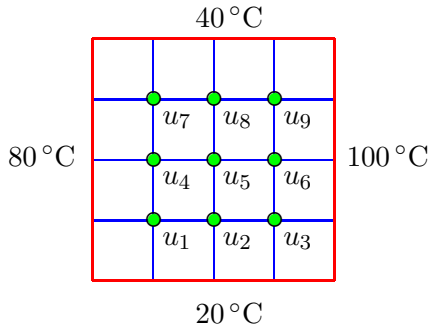
$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Detta är en bandmatris med tre diagonaler (ofta kallad en tridiagonal matris). I MATLAB bildar vi en vektor fylld med 1:or, därefter placerar vi in denna vektor (multiplicerad med 2 respektive -1) som diagonaler med `spdiags` enligt

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

Diagonalerna sätts ihop som kolonner i en matris med `[-ett 2*ett -ett]`, de måste därför vara lika långa. Kolonnerna placeras som diagonaler i ordningen som ges av vektorn `[-1 0 1]` och det hela skall bli en $n \times n$-matris, därav `n,n` allra sist.

Som exempel på ett lite större linjärt ekvationssystem med en gles matris där `spdiags` är lämplig att använda tar vi: *Värmeledning* – Vi skall beräkna temperaturen på en stålplatta där plattans kanter hålls vid temperaturer enligt figuren.



Vi lägger ett gitter (grid) över området med $n = 3$ punkter horisontellt respektive vertikalt och betecknar temperaturerna i de sammanlagt $n^2 = 9$ olika gitterpunkterna med $u_1, u_2, \cdots, u_9$. Antar vi att temperaturen i en gitterpunkt är medelvärdet av temperaturena i de närmsta gitterpunkterna i *väster*, *öster*, *söder* och *norr* så kan vi skriva upp de ekvationer som ger temperaturerna.

$$\begin{cases} u_1 = \frac{1}{4}(80 + u_2 + 20 + u_4) \\ u_2 = \frac{1}{4}(u_1 + u_3 + 20 + u_5) \\ u_3 = \frac{1}{4}(u_2 + 100 + 20 + u_6) \\ \vdots \end{cases} \quad \Leftrightarrow \quad \begin{cases} 4\,u_1 - u_2 - u_4 = 20 + 80 \\ 4\,u_2 - u_1 - u_3 - u_5 = 20 \\ 4\,u_3 - u_2 - u_6 = 20 + 100 \\ \vdots \end{cases}$$

Vi skriver på matrisform $\boldsymbol{Au} = \boldsymbol{b}$ enligt

$$\begin{bmatrix} 4 & -1 & 0 & \cdots \\ -1 & 4 & -1 & \cdots \\ 0 & -1 & 4 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} 20+ & 80 \\ 20+ & 0 \\ 20+ & 100 \\ & \vdots \end{bmatrix}$$

Uppgift 1. Skriv ned alla ekvationer på papper och gör färdigt matrisformen. Bilda matrisen med `spdiags`, ungefär på sammma sätt som i exemplet högst upp på sidan. Tänk på att ni inte har -1:or överallt på diagonalerna precis nedanför och ovanför huvuddiagonalen. Ni måste justera matrisen ni bildat. Bilda sedan högerledsvektorn och lös ekvationssystemet i MATLAB. Får ni rimliga temperaturvärden?

När ni är klara med uppgiften ovan har ni förhoppningsvis kommit fram till följande resultat:

$$\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} 20 \\ 20 \\ 20 \\ 0 \\ 0 \\ 0 \\ 40 \\ 40 \\ 40 \end{bmatrix} + \begin{bmatrix} 80 \\ 0 \\ 100 \\ 80 \\ 0 \\ 100 \\ 80 \\ 0 \\ 100 \end{bmatrix}$$

Vi har markerat en blockstruktur, block som beskriver samband horisontellt respektive vertikalt. Med n inre punkter i gittrets båda led får vi $n \times n$ stycken block av storleken $n \times n$. Matrisen $\boldsymbol{A}$ blir av storleken $n^2 \times n^2$. Högerledet $\boldsymbol{b}$ blir en kolonnvektor med n^2 element.

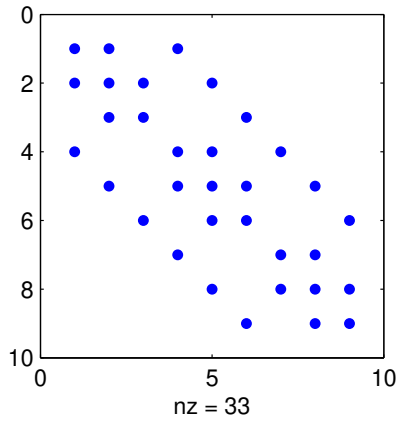
Om vi väljer n större så får vi ett tätare gitter och därmed en bättre approximation av temperaturen på plattan.

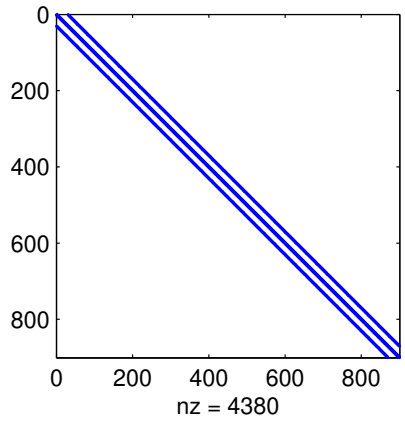
Utgående från strukturen hos matrisen $\boldsymbol{A}$ och högerledet $\boldsymbol{b}$ för $n = 3$ kan vi sluta oss till hur $\boldsymbol{A}$ och $\boldsymbol{b}$ ser ut för allmännt n.

Här är ett skript som fungerar för godtyckligt n. Testa genom att sätta $n = 3$ och använda `full` för att se efter att resultatet blir samma som förut.

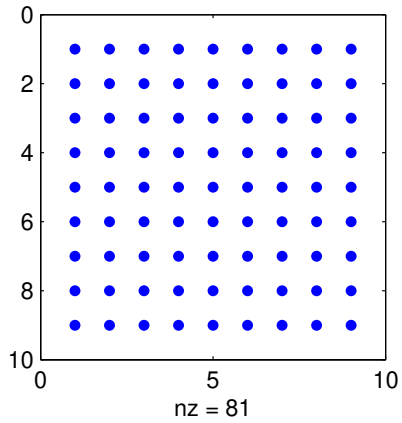
```
n=30; n2=n^2; e=ones(n2,1);
% Bilda A
A=spdiags([-e -e 4*e -e -e],[-n -1 0 1 n],n2,n2);  % Bra början, men vi har fått
for k=n:n:(n-1)*n   % några -1:or där det skall vara 0:or (röda i matrisen ovan)
    A(k,k+1)=0;     % justera diagonal +1
    A(k+1,k)=0;     % justera diagonal -1
end
% Temperaturen på kanterna (nedre, övre, vänstra, högra)
g1=20*ones(n,1); g3=40*ones(n,1); g2=80*ones(n,1); g4=100*ones(n,1);
% Bilda b
b=[g1                   % Block med känd temp från nedre kant ovanför
   zeros((n-2)*n,1)     % n-2 block med 0:or och
   g3];                 % underst block med känd temp från övre kant
for j=1:n
    j1=(j-1)*n+1;       % Första index för block j och
    jn=j*n;             % sista index för block j
    b(j1)=b(j1)+g2(j);  % Addera i början av varje block och
    b(jn)=b(jn)+g4(j);  % i slutet av varje block
end
% Beräkna u, dvs lös Au=b
u=A\b;
```

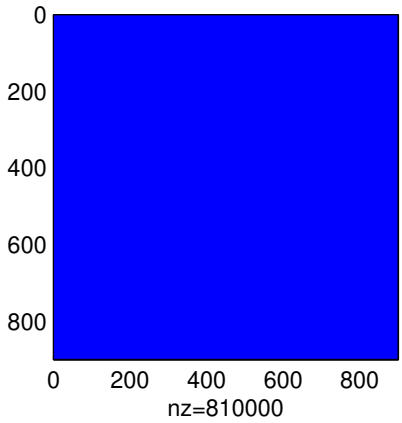
Här ser vi gleshetsstrukturen för $\boldsymbol{A}$ då $n = 3$, dvs. då $\boldsymbol{A}$ är en 9×9-matris, och då $n = 30$, dvs. då $\boldsymbol{A}$ är en 900×900-matris. Vi ser att vi har glesa matriser och förstår varför man kallar dem bandmatriser. Utskriften av `nz` ger antal element skilda från noll.





Motsvarande inverser är fyllda matriser. Bara att multiplicera med dem är kostsamt (då n är lite större), för att inte tala om att beräkna dem. Även lagra så mycket data i onödan är dumt.





Här ser vi beräkningstider (sekunder på typisk labdator) för olika n.

Gittertäthet n	3	10	20	30	60	100
Antal obekanta n^2	9	100	400	900	3600	10000
Gles `u=A\b`	0.001	0.002	0.002	0.003	0.01	0.03
Fylld `u=A\b`	0.00008	0.0004	0.008	0.03	1.2	26.3
Invers `u=inv(A)*b`	0.00008	0.0009	0.01	0.2	14.0	290

Vi ser att då matriserna blir lite större är det viktigt att utnyttja glesheten hos matriserna.

Vi skall se rita upp temperaturen $U(x, y)$ i de två variablerna x (läget horisontellt) och y (läget vertikalt), se figuren nedan. Motsvarigheten till en funktionskurva, som vi ritar med `plot`, blir en funktionsyta, som vi ritar med `surf`. Ytans nivåer (isotermer) ritar vi med `contourf`.

För att rita plattans temperaturer måste vi göra en rekonstruktion. Vi har placerat temperaturen i de olika gitterpunkterna i en kolonnvektor `u` och vi skall bilda en matris `U` för att efterlikna plattan enligt

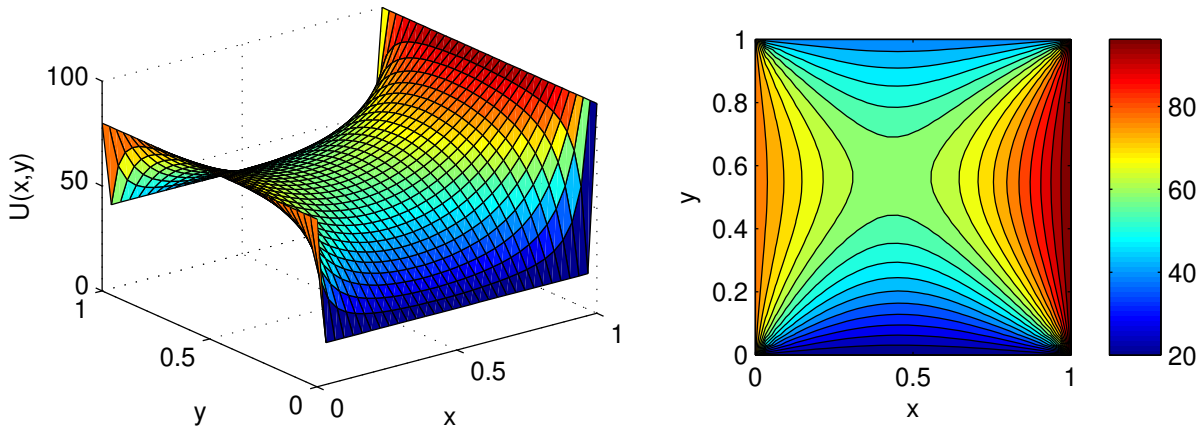
$$\begin{bmatrix} 80 & 40 & 40 & 40 & 100 \\ 80 & u_7 & u_8 & u_9 & 100 \\ 80 & u_4 & u_5 & u_6 & 100 \\ 80 & u_1 & u_2 & u_3 & 100 \\ 80 & 20 & 20 & 20 & 100 \end{bmatrix}$$

Vi har satt in plattans kanttemperaturer som ram i matrisen och i mitten fyllt på med temperaturen i de olika gitterpunkterna. Så här gör vi i MATLAB

```
% rekonstruktion  -- bilda matris U med u-värden på plattan
U=reshape(u,n,n)';   % Här styckar vi upp u
U=[g2(1) g1' g4(1)
   g2    U   g4
   g2(n) g3' g4(n)];
% plottning -- rita upp matrisen U
```

```
L=1; h=L/(n+1); x=0:h:L; y=0:h:L;
figure(1)
surf(x,y,U)
xlabel('x'), ylabel('y'), zlabel('U(x,y)')
figure(2)
contourf(x,y,U,20)
axis equal, axis tight, colorbar
xlabel('x'), ylabel('y')
```

Här ser vi resultatet av den beräkning där vi tog $n = 30$.



Vi ser hög temperaturen som rött och låg temperatur som blått. Isotermerna ser vi till höger, där stapeln ger en översättning mellan färg och temperatur.

2.3 LU-faktorisering

En matris $\boldsymbol{A}$ kan alltid faktoriseras som $\boldsymbol{PA} = \boldsymbol{LU}$, där $\boldsymbol{P}$ är en s.k. permutationsmatris som ordnar om raderna i $\boldsymbol{A}$ på lämpligt sätt, $\boldsymbol{L}$ är en nedåt triangulär matris med ettor på diagonalen och $\boldsymbol{U}$ är en uppåt triangulär matris. Läs i Lay kapitel 2.5 hur man LU-faktoriserar en matris $\boldsymbol{A}$ med systematisk Gauss-elimination. (För enkelhets skull antas att inga radbyten behövs.)

Man kan lösa ett ekvationssystem $\boldsymbol{Ax} = \boldsymbol{b}$ genom att först faktorisera $\boldsymbol{A}$ på formen $\boldsymbol{PA} = \boldsymbol{LU}$, därefter lösa systemet $\boldsymbol{Ly} = \boldsymbol{Pb}$ (framåtsubstitution), för att sedan avslutningsvis lösa systemet $\boldsymbol{Ux} = \boldsymbol{y}$ (bakåtsubstitution).

För en fylld matris kräver faktoriseringen $\sim \frac{n^3}{3}$ operationer medan lösning av de triangulära systemen kräver endast $\sim \frac{n^2}{2}$ operationer vardera. 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.

En matris $\boldsymbol{A}$ som är symmetrisk, dvs. $\boldsymbol{A}^T = \boldsymbol{A}$, och positivt definit, dvs. $\boldsymbol{x}^T\boldsymbol{Ax} > 0$ för alla $\boldsymbol{x} \neq 0$, kan faktoriseras $\boldsymbol{A} = \boldsymbol{C}^T\boldsymbol{C}$, där $\boldsymbol{C}$ är uppåt triangulär. Detta kallas för Choleski-faktorisering och kräver för en fylld matris $\sim \frac{n^3}{6}$ operationer.

För glesa matriser beror antalet operationer på hur gles matrisen är och vilken gleshetsstruktur den har, men det gäller alltid att faktoriseringen är det mest arbetskrävande.

Med MATLAB beräknas faktoriseringarna enligt `[L,U]=lu(A)` respektive `C=chol(A)` oavsett om matrisen är fylld eller gles.

3 Minsta-kvadratproblem

Ett ofta förekommande problem inom teknik och vetenskap är modellering av data, dvs. att koppla samman mätdata med en formel eller kurva som man vill verifiera eller bygga upp.

Som exempel tar vi det klassiska problemet att anpassa en rät linje $y = a + b\,t$ till givna mätdata $(t_i, y_i),\ i = 1, \cdots, n$. Hur skall vi välja a och b? Vi kan ju inte få den räta linjen att gå igenom mer än högst två punkter. Problemet vi skall lösa är följande överbestämda ekvationssystem

$$\begin{cases} a + b\,t_1 = y_1 \\ a + b\,t_2 = y_2 \\ \quad\vdots \\ a + b\,t_n = y_n \end{cases} \qquad \underbrace{\begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_n \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\boldsymbol{y}}$$

Grundidén i minsta-kvadratmetoden är att projicera vektorn $\boldsymbol{y}$ ortogonalt på kolonnrummet $\mathrm{Col}(\boldsymbol{A})$ och sedan lösa ekvationen $\boldsymbol{Ax} = \boldsymbol{p}$ där $\boldsymbol{p}$ är projektionen. Vi får då en lösning $\hat{\boldsymbol{x}}$ där avståndet $\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}\|$ är det minsta möjliga och väljer vi nu $a = \hat{x_1},\ b = \hat{x_2}$ så har vi minimerat summan av kvadraterna på avvikelserna:

$$\sum_{i=1}^{n} (a + b\,t_i - y_i)^2$$

Lösningen $\hat{\boldsymbol{x}}$ till problemet $\boldsymbol{Ax} = \boldsymbol{p}$ ovan säges vara *minsta-kvadratlösningen* till det ursprungliga ekvationssystemet $\boldsymbol{Ax} = \boldsymbol{y}$ och ges som lösningen till normalekvationen

$$\boldsymbol{A}^T\boldsymbol{Ax} = \boldsymbol{A}^T\boldsymbol{y}$$

Det *kvadratiska medelfelet*, den genomsnittliga avvikelsen, ges av

$$\varepsilon = \|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}\|/\sqrt{n}$$

där n är antalet mätdata.

I praktiken bildar man sällan normalekvationen, utan man gör en s.k. QR-faktorisering av $\boldsymbol{A}$ så att $\boldsymbol{A} = \boldsymbol{QR}$, där $\boldsymbol{Q}$ har ortonormala kolonner och $\boldsymbol{R}$ är kvadratisk och uppåt triangulär. Minsta-kvadratlösningen beräknas sedan som lösningen till systemet $\boldsymbol{Rx} = \boldsymbol{Q}^T\boldsymbol{y}$. Läs om detta i Lay kapitel 6.5.

MATLAB beräknar lösningen på detta sätt om vi använder backslash-kommandot enligt `x=A\y`. Notera att vi skriver på samma sätt som när vi löser ett vanligt kvadratiskt ekvationssystem.

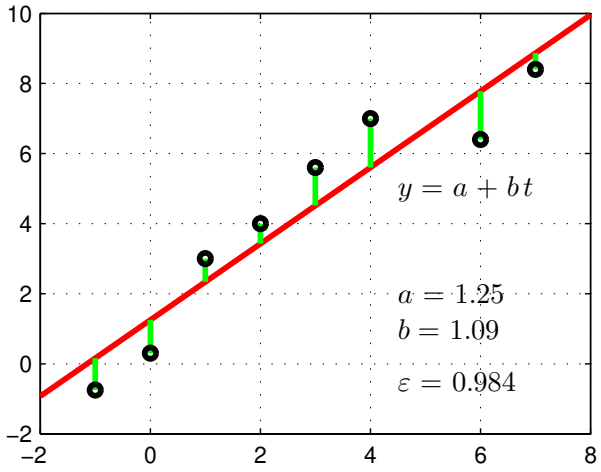
Låt oss bestämma den räta linje som i minsta-kvadratmening är bäst anpassad till följande data:

$\boldsymbol{t}$	-1	0	1	2	3	4	6	7
$\boldsymbol{y}$	-0.75	0.3	3	4	5.6	7	6.4	8.4

I MATLAB bildar vi $\boldsymbol{A}$, $\boldsymbol{t}$ och $\boldsymbol{y}$ samt beräknar $\hat{\boldsymbol{x}}$ enligt

```
>> td=[-1 0 1 2 3 4 6 7]';              % t-data
>> yd=[-0.75 0.3 3 4 5.6 7 6.4 8.4]';   % y-data
>> A=[ones(size(td)) td];               % Designmatrisen
>> x=A\yd; a=x(1); b=x(2);              % Minsta-kvadratlösningen
>> n=length(td);                        % Antalet mätdata
>> e=norm(A*x-yd)/sqrt(n);              % Kvadratiska medelfelet
```

Vi kan nu rita upp följande figur



Vi minimerar summan av kvadraterna på de lodräta sträckorna.

3.1 Allmän formulering

Vi skall nu bestämma den kurva

$$y = c_1 f_1(t) + c_2 f_2(t) + \cdots + c_k f_k(t)$$

där $f_1,\ f_2,\ \cdots, f_k$ är kända funktioner, som i minsta-kvadratmening är bäst anpassad till givna mätdata

$\boldsymbol{t}$	t_1	t_2	$\cdots$	t_n
$\boldsymbol{y}$	y_1	y_2	$\cdots$	y_n

Matrisformuleringen av det överbestämda ekvationssystem vi intresserar oss för ges av

$$\underbrace{\begin{bmatrix} f_1\left(\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}\right) & f_2\left(\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}\right) & \cdots & f_k\left(\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}\right) \end{bmatrix}}_{\boldsymbol{A}\ \text{(designmatrisen)}} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{bmatrix} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\boldsymbol{y}}$$

Låt oss återgå till vårt första exempel ovan och istället anpassa ett andragradspolynom

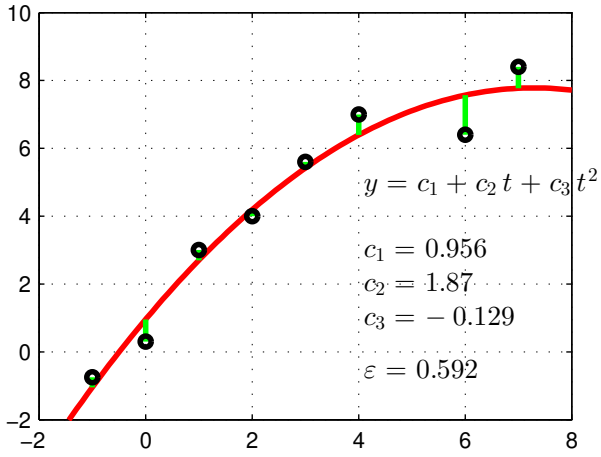
$$y = c_1 + c_2 t + c_3 t^2$$

till givna mätpunkter.

Här är $f_1(t) = 1$, $f_2(t) = t$, $f_3(t) = t^2$ och designmatrisen $\boldsymbol{A}$ och minsta-kvadratlösningen skapas i MATLAB på liknande sätt som tidigare

```
>> td=[-1 0 1 2 3 4 6 7]';              % t-data
>> yd=[-0.75 0.3 3 4 5.6 7 6.4 8.4]';   % y-data
>> A=[ones(size(td)) td td.^2];         % Designmatrisen
>> x=A\yd;                              % Minsta-kvadratlösningen
>> n=length(td);                        % Antalet mätdata
>> e=norm(A*x-yd)/sqrt(n);              % Kvadratiska medelfelet
```

Vi kan nu rita figuren



Åter minimerar vi summan av kvadraterna på de lodräta sträckorna, men nu får vi en bättre anpassning. Det kvadratiska medelfelet blir betydligt mindre.

Uppgift 2. Tabellen nedan visar medeltemperaturen under tidsperioden 1961-1990 för årets tolv månader i Göteborg.

Månad	1	2	3	4	5	6	7	8	9	10	11	12
Temp °C	-0.9	-0.9	2.0	6.0	11.6	15.5	16.6	16.2	12.8	9.1	4.4	1.0

Eftersom detta är ett periodiskt förlopp skall vi anpassa följande modell till mätdata:

$$y(t) = c_1 + c_2\cos(\omega t) + c_3\sin(\omega t)$$

Välj ett lämpligt värde på ω (tänk på periodlängder) och bestäm c_1, c_2 och c_3 med minsta-kvadratmetoden. Rita grafen av funktionen $y(t)$ och rita ut mätdata i samma bild så att man kan se om modellen ansluter väl till mätdata.

Uppgift 3. Värmeförlusten hos den som vistas i kyla beror inte enbart på temperaturen, utan även på hur mycket det blåser.

v \ t	10	6	0	-6	-10	-16	-26	-30	-36
2	9	5	-2	-9	-14	-21	-33	-37	-44
6	7	2	-5	-13	-18	-26	-38	-44	-51
10	6	1	-7	-15	-20	-28	-41	-47	-55
14	6	0	-8	-16	-22	-30	-44	-49	-57
18	5	-1	-9	-17	-23	-31	-45	-51	-59

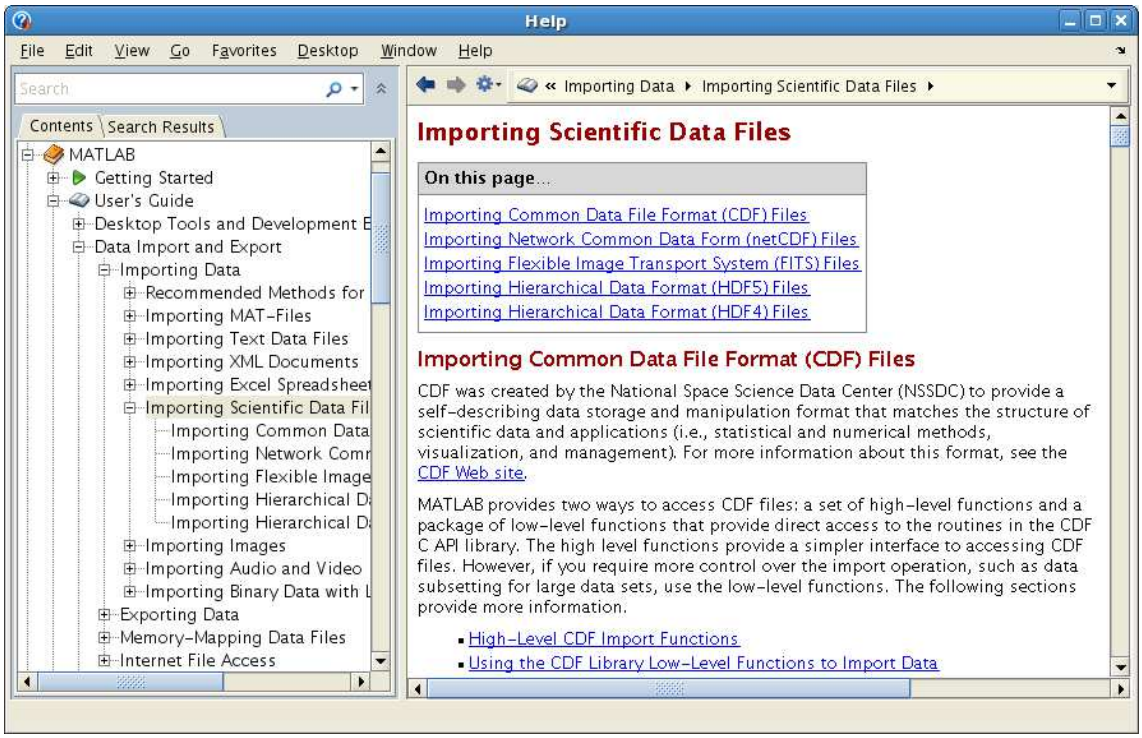
Tabellen ovan visar vilken *effektiv temperatur* det blir vid olika temperaturer t [°C] och vindhastigheter v [m/s].

(a). På data-filen `avkylningseffekt.mat` på kurshemsidan finns tabellen lagrad i tre variabler `t`, `v` och `ET`. Hämta data-filen och ladda in i MATLAB med `load('avkylningseffekt')`.

(b). Anpassa en linjär modell $y(t, v) = c_1 + c_2 t + c_3 v$ till data med minsta-kvadratmetoden. Rita ut data med markörer och ett plan som beskriver modellen.

3.2 Import av data

För att hantera mätdata behöver man bl.a. kunna hantera olika fil-format. För import av data har MATLAB en Import Wizard.



Vid behov läser man lämpligen mer om detta i Helpdesk.