```
% Föreläsning 2, den 8/2

clear

% Vi återvänder till uppgift 1.9 i övningshäftet, och kommandot A\Y som vi
% såg förra gången
A=[ 1 1 ; 1 -2 ; 3 4 ] % Systemmatrisen
Y=[ -4 ; 2 ; 1 ] % Högerledet
A\Y
% Detta system skulle ju som sagt inte ha någon lösning!
% Nu blir det lite "vanlig" föreläsning som förklarar vad som händer.

% Här kommer kurvanpassningsexemplet från föreläsningen:
xv=[ 1 2 3 4 5 ] % x-värdena
yv=[ 3 5 8 9 13 ] % Motsvarande y-värden
A=[ xv' ones(5,1)] % Skapar A
Y=yv'
A'*A
A'*Y
Sol = A'*A\A'*Y % Löser normalekvationen
plot(xv,yv,'+') % Plottar in mätpunkterna
xlim([0,6]), ylim([2,14]), axis equal % Nu blev det bättre
hold on
k=Sol(1), m=Sol(2) % Plockar ut k och m ur kolonnmatrisen som är lösningen
x=linspace(0,6);
y=k*x+m;
plot(x,y,'k') % Ritar den anpassade linjen (svart)

% Kommandot A\Y gör precis
% detta då systemet är överbestämt. Vi provar:
A\Y
% Stämmer!



% Efter att ha läst om Taylorutvecklingar så inser du nog att polynom är
% speciellt viktiga funktioner. Vi tittar nu på hur man kan hantera polynom
% i Matlab.

% Ett polynom representeras som en vektor där koefficienterna är listade,
% så exempelvis x^4 - 2 x^3 + 4 x^2 - x + 7 blir
p=[ 1 -2 4 -1 7 ]
polyval(p,1) % Beräknar värdet i x=1 (kolla!)
roots(p) % Ger alla rötterna. Här blev de visst komplexa.
polyder(p) % Deriverar (kolla!)
polyint(p) % Integrerar (kolla!)
q=[ 1 -1 ] % Polynomet x-1
conv(p,q) % Beräknar produkten p*q

% Det finns ett färdigt kommando för att anpassa ett polynom till ett antal
% punkter: "polyfit(x,y,n)" ger polynomet av grad n som är bäst anpassat
% till punkterna som ges av vektorerna x och y.
% Vi illustrerar genom att återvända till exmplet med kurvanpassning ovan,
% där vi ville anpassa en linje till ett antal mätdata.
% Alternativt kan vi här skriva:
hold off
```

```
xv=[ 1 2 3 4 5 ]; % x-värdena
yv=[ 3 5 8 9 13 ]; % Motsvarande y-värden
plot(xv,yv,'+') % Plottar in mätpunkterna
xlim([0,6]), ylim([2,14]), axis equal
hold on
P=polyfit(xv,yv,1) % Beräknar koefficienterna i aktuellt 1:a-gradspolynom
x=linspace(0,6);
y=polyval(P,x); % Räknar ut motsvarande y-värden
plot(x,y,'k') % Ritar den anpassade linjen
% Nu kan vi exempelvis lätt även rita in det anpassade 2:a-gradspolynomet
P=polyfit(xv,yv,2)
x=linspace(0,6);
y=polyval(P,x);
plot(x,y,'--')
legend('mätpunkter','1:a-grads','2:a-grads') % Så här kan man skriva
% förklaringar till punkterna/kurvorna. De kommer i rätt ordning.

% Vi diskuterar nu (igen) på tavlan, genom att tänka på ekvationssystem, hur
hög
% grad ett polynom måste ha för att dess graf exakt skall gå genom ett
% visst antal punkter.
hold off
xv=[ 1 2 3 4 5 ];
yv=[ 3 5 8 9 13 ];
plot(xv,yv,'+')
xlim([0,6]), ylim([2,14]), axis equal
hold on
n=length(xv)
P=polyfit(xv,yv,n-1)
x=linspace(0,6);
y=polyval(P,x);
plot(x,y,'k')
% Beteendet vid den första mätpunkten är kanske inte det vi vill ha
% Att anpassa en funktion så att den går exakt genom givna punkten
% kallas "interpolation"

% Vi kan också interpolera genom att "fixa till en kurvbit mellan varje par
% av punkter".
hold off
xv=[ 1 2 3 4 5 ];
yv=[ 3 5 8 9 13 ];
plot(xv,yv,'+')
xlim([0,6]), ylim([2,14]), axis equal
hold on
x=linspace(0,6);
yIl=interp1(xv,yv,x,'linear'); % Ger en linjär funktion mellan punkterna,
% dvs. ett linjestycke från punkt till punkt
yIk=interp1(xv,yv,x,'spline'); % Ger ett tredjegradspolynom mellan punkterna,
% och fixar dessutom till helheten så att det blir deriverbart och fint.
plot(x,yIl,'k',x,yIk,'--')
legend('mätpunkter','linjär','kubisk')
% Resultatet blev lite bättre till höger om första punkten.

% Hinner vi så tittar vi även lite på de färdiga kommandon som man kan
% klicka på exempelvis i grafikfönstret.
```