Løsningsforslag til eksamensopgaver i M3NUM1 sommer 2022

OPGAVE 1

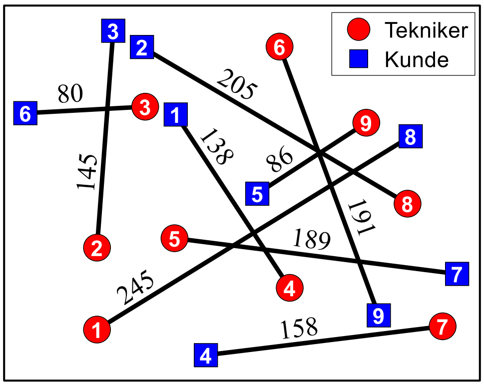
(1a)

Her følger to mulige versioner af funktionen `samletafst`:

```
function sa = samletafst(kundetildelt,afstande)
n = length(kundetildelt);
sa = 0;
for tekniker = 1:n
  sa = sa + afstande(tekniker,kundetildelt(tekniker));
end
```

```
function sa = samletafst(kundetildelt,afstande)
sa = sum(diag(afstande(:,kundetildelt)));
```

Funktionen afprøves med de i opgaven anførte data (figuren med kundetildeling og afstande til højre er blot til illustration og ikke et krav i besvarelsen):




```
kundetildelt = [8 3 6 1 7 9 4 2 5];
AfstandeTeknikereKunder; % Definér afstande-matricen
samletafstand = samletafst(kundetildelt,afstande)
```

```
samletafstand = 1437
```

Den samlede transportafstand mellem teknikerne og de tildelte kunder er således 1437 km.

(1b)

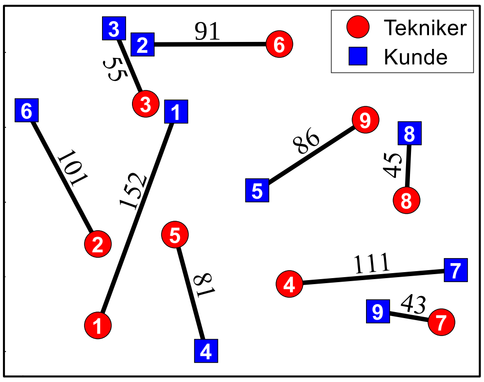
Her følger det redigerede script:

```
AfstandeTeknikereKunder;        % Definér afstandsmatricen afstande
n = length(afstande);
tildelingsmuligh = perms(1:n);  % Generér alle tildelingsmuligheder
N = factorial(n);               % Antal tildelingsmuligheder = n!
minsamletafstand = +inf;        % Sæt initielt min. afst. = uendelig
for i = 1:N
  kundetildelt = tildelingsmuligh(i,:);
  samletafstand = samletafst(kundetildelt,afstande);
  if samletafstand < minsamletafstand && kundetildelt(1) ~= 4 && kundetildelt(4) == 7
    optkundetildelt = kundetildelt;
    minsamletafstand = samletafstand;
  end
end
optkundetildelt                 % Vis indhold af optkundetildelt
minsamletafstand                % Vis værdi af minsamletafstand
```

Ved kørsel af scriptet fås:

```
optkundetildelt = 1×9
     1     6     3     7     4     2     9     8     5
minsamletafstand = 765
```

Dvs. at tekniker 1, 2, …, 9 tildeles henholdsvis kunde 1, 6, 3, 7, 4, 2, 9, 8 og 5, og at den samlede transportafstand bliver 765 km.



OPGAVE 2

(2a)

Løsning af differentialligningssystemet vha. Eulers metode kan gennemføres i Excel eller MATLAB.

Ved brug af Excel opnås følgende talresultater:

	A	B	C	D	E	F
1	Pmidl =	185	Pmax =	214	Lmax =	125
2	a =	1,2	k =	1,3	d =	0,03
3	S0 =	24	B0 =	66	h =	0,25
4						
5	i	t	S	B	dS/dt	dB/dt
6	0	0	24	66	68,16	-6,74135
7	1	0,25	41,04	64,31466	65,53088	-22,8863
8	2	0,5	57,42272	58,59309	56,60521	-35,8696
9	3	0,75	71,57402	49,62569	42,61608	-43,3678
10	4	1	82,22804	38,78374	25,70264	-43,1797
11	5	1,25	88,6537	27,98883	8,862572	-34,6718
12	6	1,5	90,86934	19,32088	-4,65943	-20,8559
13	7	1,75	89,70449	14,10691	-12,7932	-7,83662
14	8	2	86,50618	12,14775	-15,8495	0,317658
15	9	2,25	82,5438	12,22717	-15,7256	4,073429
16	10	2,5	78,6124	13,24552	-14,137	5,39863
17	11	2,75	75,07815	14,59518	-12,0315	5,599237
18	12	3	72,07027	15,99499	-9,84782	5,289281
19	13	3,25	69,60832	17,31731	-7,785	4,742485
20	14	3,5	67,66207	18,50293	-5,93543	4,090656
21	15	3,75	66,17821	19,52559	-4,34007	3,406572
22	16	4	65,09319	20,37724	-3,01151	2,736267
23	17	4,25	64,34032	21,0613	-1,94437	2,111334
24	18	4,5	63,85422	21,58914	-1,12094	1,553451
25	19	4,75	63,57399	21,9775	-0,5151	1,076052
26	20	5	63,44521	22,24651	-0,09544	0,685175

Her følger formlerne bag tallene:

	A	B	C	D	E	F
1	Pmidl =	185	Pmax =	214	Lmax =	125
2	a =	1,2	k =	1,3	d =	0,03
3	S0 =	24	B0 =	66	h =	0,25
4						
5	i	t	S	B	dS/dt	dB/dt
6	0	=i*h	24	66	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
7	=A6+1	=i*h	=C6+E6*h	=D6+F6*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
8	=A7+1	=i*h	=C7+E7*h	=D7+F7*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
9	=A8+1	=i*h	=C8+E8*h	=D8+F8*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
10	=A9+1	=i*h	=C9+E9*h	=D9+F9*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
11	=A10+1	=i*h	=C10+E10*h	=D10+F10*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
12	=A11+1	=i*h	=C11+E11*h	=D11+F11*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
13	=A12+1	=i*h	=C12+E12*h	=D12+F12*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S

	A	B	C	D	E	F
14	=A13+1	=i*h	=C13+E13*h	=D13+F13*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
15	=A14+1	=i*h	=C14+E14*h	=D14+F14*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
16	=A15+1	=i*h	=C15+E15*h	=D15+F15*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
17	=A16+1	=i*h	=C16+E16*h	=D16+F16*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
18	=A17+1	=i*h	=C17+E17*h	=D17+F17*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
19	=A18+1	=i*h	=C18+E18*h	=D18+F18*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
20	=A19+1	=i*h	=C19+E19*h	=D19+F19*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
21	=A20+1	=i*h	=C20+E20*h	=D20+F20*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
22	=A21+1	=i*h	=C21+E21*h	=D21+F21*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
23	=A22+1	=i*h	=C22+E22*h	=D22+F22*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
24	=A23+1	=i*h	=C23+E23*h	=D23+F23*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
25	=A24+1	=i*h	=C24+E24*h	=D24+F24*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S
26	=A25+1	=i*h	=C25+E25*h	=D25+F25*h	=a*(Pmidl-Pmax+k*B)	=Lmax*EKSP(-d*B)-S

Ved brug af den i undervisningen præsenterede MATLAB-funktion `eulsys` opnås samme resultater:

```
Pmidl = 185; Pmax = 214; Lmax = 125; a = 1.2; k = 1.3; d = 0.03;
S0 = 24; B0 = 66;
dYdt = @(t,Y) [a*(Pmidl - Pmax + k*Y(2)), Lmax*exp(-d*Y(2)) - Y(1)];
Y0 = [S0 B0];
tidsinterval = [0 5]; h = 0.25;
[t,Y] = eulsys(dYdt,tidsinterval,Y0,h);
S = Y(:,1); B = Y(:,2);
disp(table(t,S,B,'VariableNames',{'t, uger','S, enh./uge','B, enh.'}))
```

t, uger	**S, enh./uge**	**B, enh.**
0	24	66
0.25	41.04	64.315
0.5	57.423	58.593
0.75	71.574	49.626
1	82.228	38.784
1.25	88.654	27.989
1.5	90.869	19.321
1.75	89.704	14.107
2	86.506	12.148
2.25	82.544	12.227
2.5	78.612	13.246
2.75	75.078	14.595
3	72.07	15.995
3.25	69.608	17.317
3.5	67.662	18.503
3.75	66.178	19.526
4	65.093	20.377
4.25	64.34	21.061
4.5	63.854	21.589
4.75	63.574	21.978
5	63.445	22.247

Salget opnår ifølge løsningen sin højeste værdi på <u>90,87 enheder pr. uge</u> efter <u>1,5 uge</u>. PS! Det vil også være acceptabelt, hvis salget angives op- eller nedrundet til helt tal.

(2b)
Idet differentialligningssystemets afhængige variable, S og B, arrangeres på vektorform, $Y = [S, B]$, dvs. $Y_1 = S$ og $Y_2 = B$, kan ligningerne omskrives som følger:

$$S'(t) = a\,(P_{\text{midl}} - P_{\max} + k\,B(t)),\quad S(0) = S_0 \quad \Rightarrow \quad \frac{dY_1}{dt} = a\,(P_{\text{midl}} - P_{\max} + k\,Y_2),\quad Y_1(0) = S_0$$

$$B'(t) = L_{\max} e^{-d\,B(t)} - S(t), \quad B(0) = B_0 \qquad \Rightarrow \frac{dY_2}{dt} = L_{\max} e^{-d\,Y_2} - Y_1, \quad Y_2(0) = B_0$$

Dermed kan MATLAB-koden opskrives og køres og løsningsresultater opnås:

```
Pmidl = 185; Pmax = 214; Lmax = 125; a = 1.2; k = 1.3; d = 0.03;
S0 = 24; B0 = 66;
dYdt = @(t,Y) [a*(Pmidl - Pmax + k*Y(2)), Lmax*exp(-d*Y(2)) - Y(1)];
Y0 = [S0 B0];
tidsinterval = [0 5]; h = 0.25;
[t,Y] = rk4system(dYdt,tidsinterval,Y0,h);
S = Y(:,1); B = Y(:,2);
disp(table(t,S,B,'VariableNames',{'t, uger','S, enh./uge','B, enh.'}))
```

t, uger	S, enh./uge	B, enh.
0	24	66
0.25	40.46	62.428
0.5	54.885	55.741
0.75	66.295	47.196
1	74.248	38.261
1.25	78.869	30.3
1.5	80.728	24.191
1.75	80.61	20.151
2	79.276	17.883
2.25	77.323	16.897
2.5	75.16	16.735
2.75	73.037	17.054
3	71.094	17.626
3.25	69.399	18.303
3.5	67.972	18.996
3.75	66.811	19.653
4	65.893	20.244
4.25	65.19	20.755
4.5	64.671	21.182
4.75	64.301	21.527
5	64.052	21.798

Salget opnår ifølge løsningen sin højeste værdi på 80,73 enheder pr. uge efter 1,5 uge. PS! Det vil også være acceptabelt, hvis salget angives op- eller nedrundet til helt tal.

Da Eulers metode og den fjerdeordens Runge-Kutta-metode begge er gennemført med tidsskridt på 0,25 uge, vil løsningen fundet med sidstnævnte metode (delopgave (2a)) være mest troværdigt. Det skyldes, at en fjerdeordensmetoden må antages at være mere nøjagtig end en førsteordensmetode (Eulers metode).

OPGAVE 3

(3a)

Jacobi-matricen bestemmes ved partiel differentiation af de enkelte funktioner mht. hver enkelte ubekendte:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{x_1}{\sqrt{x_1{}^2+1}} & -x_3 & -x_2 \\ (x_2-2)^2 & 2x_1\,(x_2-2) & 2 \\ x_2x_3 - x_2 - x_3 & x_1x_3 - x_1 - x_3 & x_1x_2 - x_1 - x_2 \end{bmatrix}$$

Man kan evt. anvende den indbyggede MATLAB-funktion `jacobian` til bestemmelse J:

```
syms x_1 x_2 x_3 f J
f = [sqrt(x_1^2+1) - x_2*x_3 - 2
     x_1*(x_2-2)^2 + 2*x_3 - 2
     x_1*x_2*x_3 - x_1*x_2 - x_1*x_3 - x_2*x_3]
J = jacobian(f,[x_1;x_2;x_3])
```

f = $\begin{bmatrix} \sqrt{x_1{}^2+1} - x_2\,x_3 - 2 \\ 2\,x_3 + x_1\,(x_2-2)^2 - 2 \\ x_1\,x_2\,x_3 - x_1\,x_3 - x_2\,x_3 - x_1\,x_2 \end{bmatrix}$ J = $\begin{bmatrix} \frac{x_1}{\sqrt{x_1{}^2+1}} & -x_3 & -x_2 \\ (x_2-2)^2 & x_1\,(2\,x_2-4) & 2 \\ x_2\,x_3 - x_3 - x_2 & x_1\,x_3 - x_3 - x_1 & x_1\,x_2 - x_2 - x_1 \end{bmatrix}$

Da der er matrixelementer i Jacobi-matricen, der ikke er konstante, fordi de afhænger af de ubekendte, kan man konkludere, at ligningssystemet er ulineært.

(3b)

I forbindelse med første iteration i Newton-Raphsons metode skal man udregne den inverse af Jacobi-matricen for de værdier, der er givet i startgættet, dvs. i dette tilfælde for $x_1 = x_2 = x_3 = 0$. Indsættes disse x-værdier i Jacobi-matricen (jfr. løsningen til delopgave (3a)), fås matricen:

$$\begin{bmatrix} 0 & 0 & 0 \\ 4 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}$$

Denne matrix kan ikke inverteres (den er singulær), hvilket bekræftes af, at dens determinant er nul:

`inv([0 0 0; 4 0 2; 0 0 0])`	Warning: Matrix is singular to working precision. ans = 3×3 Inf Inf Inf Inf Inf Inf Inf Inf Inf
`det([0 0 0; 4 0 2; 0 0 0])`	ans = 0

Man kan således ikke anvende det givne startgæt som udgangspunkt for Newton-Raphsons metode.

PS! Hvis delopgave (3a) ikke er besvaret, og man derfor anvender den alternative Jacobi-matrix, fås ligeledes en singulær matrix ved indsættelse af det givne startgæt, og konklusionen bliver dermed den samme.

(3c)

Fremgangsmåde 1: Opgaven kan fx løses vha. et MATLAB-script:

`format long`
`x = [-2; 3; 2]`
`for i = 1:6`
`  f = [sqrt(x(1)^2+1) - x(2)*x(3) - 2`
`       x(1)*(x(2)-2)^2 + 2*x(3) - 2`
`       x(1)*x(2)*x(3) - x(1)*x(2) - x(1)*x(3) - x(2)*x(3)];`
	x = 3×1
	-2
	3
	2
	x = 3×1
	-3.583715866413271
	2.159626588952238

```
  J = [x(1)/sqrt(x(1)^2+1)  -x(3)                 -x(2)
       (x(2)-2)^2           2*x(1)*(x(2)-2)       2
       x(2)*x(3)-x(2)-x(3)  x(1)*x(3)-x(1)-x(3)  x(1)*x(2)-x(1)-x(2)];
  x = x - J\f
end
```

```
   1.111111111111111
x = 3×1
  -5.428534915600491
   2.666146263618940
   1.358919084389606
x = 3×1
  -5.020999340936641
   2.437310211818899
   1.286519818697206
x = 3×1
  -4.924247604183617
   2.352918362594540
   1.285555024186328
x = 3×1
  -4.915860578084753
   2.342369360058020
   1.287805924341830
x = 3×1
  -4.915772917325267
   2.342223345196957
   1.287859784253214
```

Det ses, at iteration 5 og 6 giver samme resultat med op til tre decimaler (afrundet) for alle tre ubekendte, og den fundne løsning med sikre decimaler er derfor: $\underline{\underline{x_1 = -4{,}916,\ x_2 = 2{,}342,\ x_3 = 1{,}288}}$.

Fremgangsmåde 2: Ved denne fremgangsmåde forudsættes det, at følgende funktion, `Jogf`, til beregning af Jacobi-matricen og funktionsvektoren, er defineret:

```
function [J,f] = Jogf(x)
f = [sqrt(x(1)^2+1) - x(2)*x(3) - 2
     x(1)*(x(2)-2)^2 + 2*x(3) - 2
     x(1)*x(2)*x(3) - x(1)*x(2) - x(1)*x(3) - x(2)*x(3)];
J = [x(1)/sqrt(x(1)^2+1)  -x(3)                 -x(2)
     (x(2)-2)^2           2*x(1)*(x(2)-2)       2
     x(2)*x(3)-x(2)-x(3)  x(1)*x(3)-x(1)-x(3)  x(1)*x(2)-x(1)-x(2)];
```

Opgaven kan herefter løses vha. funktionen `newtmult`, der har været anvendt i undervisningen:

```
format long
x0 = [-2; 3; 2];
es = 0.000001;
maxit = 6;
[x,f,ea,iter] = newtmult(@Jogf,x0,es,maxit)
xinterval = [x-x*ea/100 x+x*ea/100]
```

```
x =
  -4.915772917325267
   2.342223345196957
   1.287859784253214
f =
   1.0e-03 *
   0.024022435512361
  -0.609492849895421
   0.140747104591110
ea =
   0.006234028081150
iter =
     6
xintervaller =
  -4.915466466661195  -4.916079367989338
   2.342077330335894   2.342369360058020
   1.287779498712618   1.287940069793810
```

Antallet af sikre decimaler (afrundet) for de enkelte ubekendte bestemmes på grundlag af usikkerhedsintervallerne i `xintervaller`. Øvre og nedre intervalendepunkt bliver ens, når der afrundes til henholdsvis 2, 3 og 3 decimaler for hhv. x_1, x_2 og x_3, så den fundne løsning angivet med sikre decimaler er derfor: $\underline{\underline{x_1 = -4{,}92,\ x_2 = 2{,}342,\ x_3 = 1{,}288}}$.

PS! Hvis ovenstående fremgangsmåder gennemføres med den alternative Jacobi-matrix, der er opgivet i opgaven, fås:

Fremgangsmåde 1: $\underline{\underline{x_1 = -4{,}9157,\ x_2 = 2{,}342,\ x_3 = 1{,}288}}$

Fremgangsmåde 2: $\underline{\underline{x_1 = -4{,}92,\ x_2 = 2{,}342,\ x_3 = 1{,}29}}$

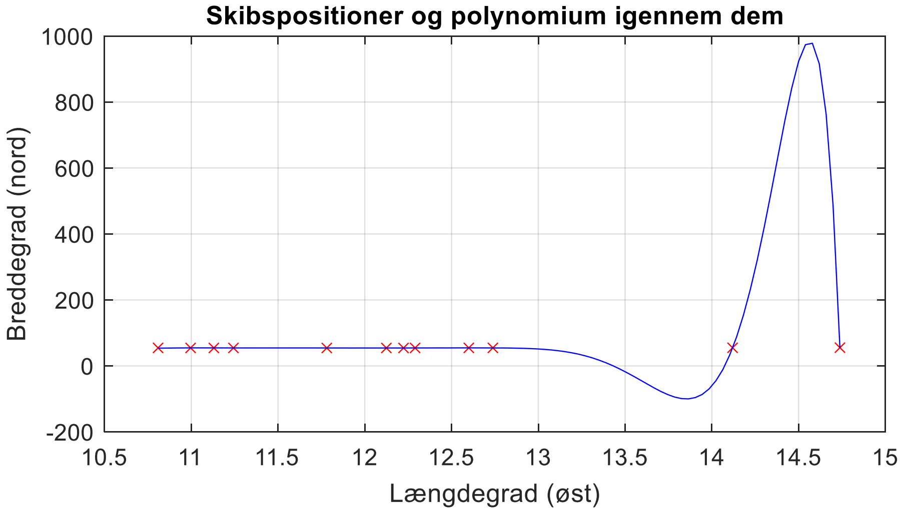
OPGAVE 4

(4a)

Da der er 12 punkter (positioner), skal der bestemmes et polynomium af 11. grad. Her følger MATLAB-kode og resultater (`polyfit` kommer med en advarsel, som ikke er gengivet):

```
format shortg
Skibspositioner; % Indlæs skibspos. (lgd og brd)
p = polyfit(lgd,brd,11);
disp('Polynomiumskoefficienter:')
disp(p')
lgd1 = linspace(min(lgd),max(lgd));
brd1 = polyval(p,lgd1);
plot(lgd,brd,'rx',lgd1,brd1,'b')
title('Skibspositioner og polynomium igennem dem')
xlabel('Længdegrad (øst)')
ylabel('Breddegrad (nord)')
grid
```

```
Polynomiumskoefficienter:
      -1.6768
       226.45
       -13888
   5.1058e+05
  -1.2502e+07
    2.141e+08
  -2.6164e+09
   2.2819e+10
  -1.3918e+11
   6.6548e+11
  -1.3773e+12
   1.5234e+12
```

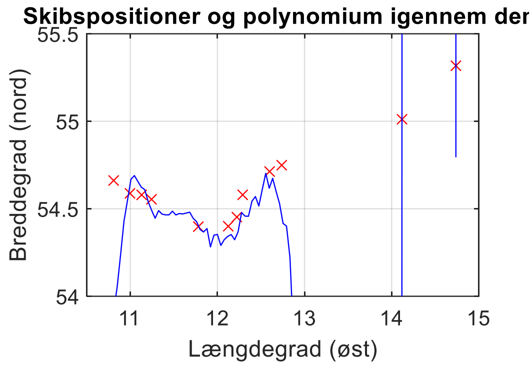


PS! Den høje polynomiumsgrad gør, at forskellige computere kan give forskellige koefficientværdier (afrundingsproblem). Måske får man:

```
Polynomiumskoefficienter:
      -1.7375
       235.62
       -14510
   5.3565e+05
  -1.3171e+07
    2.265e+08
  -2.7797e+09
   2.4346e+10
  -1.4914e+11
   6.0854e+11
  -1.4886e+12
   1.6537e+12
```

Polynomiet giver et decideret dårligt estimat at sejlrutens forløb, idet der forekommer voldsomme under- og oversving efter 13° østlig længde.

Yderligere kommentarer (ikke nødvendige for at besvarelsen kan bedømmes korrekt):
Under- og oversving er dog et velkendt problem for polynomier af høj grad. Funktionen `polyfit` udløser da også en advarsel: `Warning: Polynomial is badly conditioned ...`.

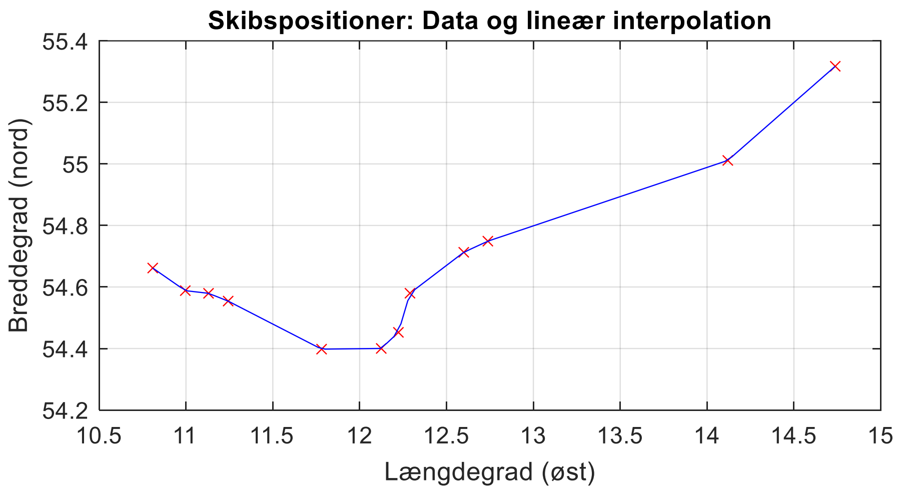


PS! Hvis man zoomer ind i y-aksens retning til en mere "naturlig" skalering (se figur til højre), ser man, at polynomiet faktisk ikke går igennem alle positioner, og at polynomiet desuden har et unaturligt forløb med hyppige oscillationer. Dette er også et velkendt problem for polynomier af høj grad, som skyldes afrundingsfejl i beregningsprocessen. Dette problem kan afhjælpes ved passende reskalering af x-værdierne (længdegraderne).

(4b)
Her følger MATLAB-kode og resultater:

```
Skibspositioner; % Indlæs skibspos. (lgd og brd)
lgd1 = linspace(min(lgd),max(lgd));
brd1 = interp1(lgd,brd,lgd1,'Linear');
plot(lgd,brd,'rx',lgd1,brd1,'b')
title('Skibspositioner: Data og lineær interpolation')
xlabel('Længdegrad (øst)')
ylabel('Breddegrad (nord)')
grid
breddegrad_ved_12_4_ost = interp1(lgd,brd,12.4,'Linear')
```

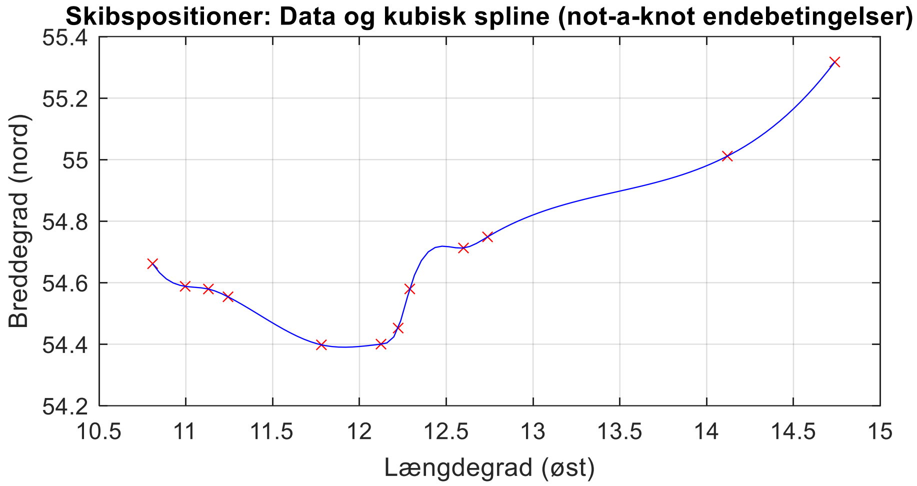
```
breddegrad_ved_12_4_ost
= 54.6271
```



Den estimerede breddegrad ved 12,4000° østlig længde er altså 54,6271°.

(4c)
Her følger MATLAB-kode og resultater:

```
Skibspositioner; % Indlæs skibspos. (lgd og brd)
lgd1 = linspace(min(lgd),max(lgd));
brd1 = spline(lgd,brd,lgd1);
plot(lgd,brd,'rx',lgd1,brd1,'b')
title('Skibspositioner: Data og kubisk spline (not-a-knot endebetingelser)')
xlabel('Længdegrad (øst)')
ylabel('Breddegrad (nord)')
grid
breddegrad_ved_12_4_ost = spline(lgd,brd,12.4)
```



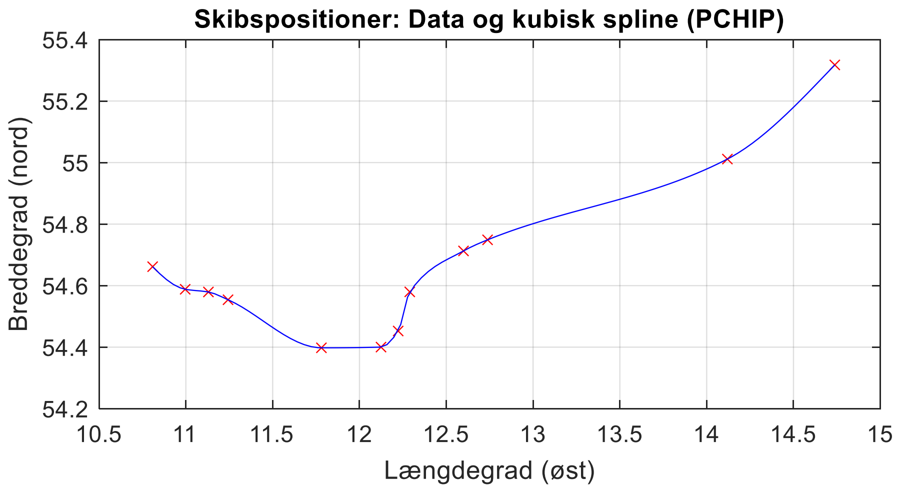

```
breddegrad_ved_12_4_ost = 54.7020
```

Den estimerede breddegrad ved 12,4000° østlig længde er altså 54,7020°

(4d)

Her følger MATLAB-kode og resultater:

```
Skibspositioner; % Indlæs skibspos. (lgd og brd)
lgd1 = linspace(min(lgd),max(lgd));
brd1 = interp1(lgd,brd,lgd1,'PCHIP');
plot(lgd,brd,'rx',lgd1,brd1,'b')
title('Skibspositioner: Data og kubisk spline (PCHIP)')
xlabel('Længdegrad (øst)')
ylabel('Breddegrad (nord)')
grid
breddegrad_ved_12_4_ost = interp1(lgd,brd,12.4,'PCHIP')
```



```
breddegrad_ved_12_4_ost = 54.6475
```

Den estimerede breddegrad ved 12,4000° østlig længde er altså 54,6475°

OPGAVE 5

(5a)

Kravet er, at $V_{\text{elm}j} = V_{\text{sek}j}$, $j = 1,2, \dots ,6$, dvs. $A_j L = \frac{\pi}{3}(r_j^2 + r_j r_{j+1} + r_{j+1}^2)L$ og dermed:

$$A_j = \frac{\pi}{3}\left(r_j^2 + r_j r_{j+1} + r_{j+1}^2\right), \qquad j = 1,2, \dots ,6$$

Tværsnitsarealerne kan dermed beregnes vha. MATLAB (tre mulige alternativer præsenteres):

```
% === Definition af størrelser til brug her og i resten af opgaven:
Ltot = 240e-3; L = 40e-3; % m
Ra = 2e-3; Rb = 6e-3;     % m
E = 210e9;                % Pa
F1 = 2000;                % N
n = 6;                    % Antal sektioner/elementer

% === Beregning af r1, r2, ..., r7 som vektor r:
for i = 1:n+1
  r(i) = Ra + (Rb-Ra)/6*(i-1); % m
end

% === Beregning af vektor A, alternativ 1:
for j = 1:n
  A(j) = pi*(r(j)^2 + r(j)*r(j+1) + r(j+1)^2)/3;
end

% === Beregning af vektor A, alternativ 2:
for j = 1:n
  A(j) = pi*(r(j)^2 + r(j)*r(j+1) + r(j+1)^2)/3;
end

% === Beregning af vektor A, alternativ 3:
A = pi*(r(1:n).^2 + r(1:n).*r(2:n+1) + r(2:n+1).^2)/3;

A_mm2 = A*1e6             % A-vektor omregnet til/vist i mm^2
```

De tre alternative kodningsmuligheder giver alle (resultater i mm^2):

```
A_mm2 = 1×6
    17.2206    28.3907    42.3533    59.1085    78.6562   100.9964
```

Dvs. $A_1 = 17{,}22 \text{ mm}^2$, $A_2 = 28{,}39 \text{ mm}^2$, $A_3 = 42{,}35 \text{ mm}^2$, $A_4 = 59{,}11 \text{ mm}^2$, $A_5 = 78{,}66 \text{ mm}^2$, $A_6 = 101{,}00 \text{ mm}^2$.

(5b)

Stivhedsmatricen, K, kan udregnes med følgende MATLAB-kode:

```
K = zeros(n+1,n+1);
for j = 1:n
  k = A(j)*E/L;           % Fjederkonstant (N/m) for stangelement j
  Kelem = [k -k; -k k];   % Stivhedsmatrix (N/m) for stangelement j
```

```
  K(j:j+1,j:j+1) = K(j:j+1,j:j+1) + Kelem; % Opdatér K
end
K_MN_pr_m = K/1e6         % K omregnet til/vist i MN/m
```

Som resultat fås den efterspurgte stivhedsmatrix i MN/m:

```
K_MN_pr_m = 7×7
   90.4081  -90.4081         0         0         0         0         0
  -90.4081  239.4592 -149.0511         0         0         0         0
         0 -149.0511  371.4061 -222.3549         0         0         0
         0         0 -222.3549  532.6745 -310.3195         0         0
         0         0         0 -310.3195  723.2644 -412.9449         0
         0         0         0         0 -412.9449  943.1759 -530.2310
         0         0         0         0         0 -530.2310  530.2310
```

PS! Som grundlag for ovenstående beregninger er `A`-vektoren udregnet i delopgave (5a) anvendt. Hvis man i stedet anvender de let afrundede arealværdier anført i indledningen til delopgave (5b), får man et resultat, der afviger en smule fra ovenstående stivhedsmatrix (afvigelse typisk på fjerde decimal).

(5c)

Forskydningerne af de frie knuder kan beregnes med følgende MATLAB-kode (stivhedsmatricen, `K`, fra delopgave (5b) anvendes i beregningerne):

```
Pfri = [-F1 0 0 0 0 0]';  % Eksterne kræfter på de frie knuder (N)
Kfri = K(1:6,1:6); % Del af stivhedsmatrix svarende til frie knuder (N/m)
D = Kfri\Pfri;            % Forskydninger af de frie knuder (m)
D_my = D'*1e6             % Forskydninger omregnet til/vist i mikrometer
```

Resultatet bliver:

```
D_my = 1×6
  -59.5949  -37.4730  -24.0548  -15.0602   -8.6152   -3.7719
```

De efterspurgte forskydninger er derfor:

$\underline{\Delta_1 = -59{,}6\ \mu\text{m},\ \Delta_2 = -37{,}5\ \mu\text{m},\ \Delta_3 = -24{,}1\ \mu\text{m}}$
$\underline{\Delta_4 = -15{,}1\ \mu\text{m},\ \Delta_5 = -8{,}6\ \mu\text{m},\ \Delta_6 = -3{,}8\ \mu\text{m}}$

PS! Hvis den alternative stivhedsmatrix anvendes som grundlag for beregningerne få lidt andre resultater:

$\underline{\Delta_1 = -59{,}0\ \mu\text{m},\ \Delta_2 = -39{,}0\ \mu\text{m},\ \Delta_3 = -25{,}7\ \mu\text{m}}$
$\underline{\Delta_4 = -15{,}7\ \mu\text{m},\ \Delta_5 = -9{,}0\ \mu\text{m},\ \Delta_6 = -4{,}0\ \mu\text{m}}$