Problem 1

```
syms t

A = [];

for i = 1:4
    A = [A ; row(i, 3)];
end

b = [11, 29, 65, 125]'

x = A\b
T = row (t, 3)

P = T * x

vpa (subs( P, t, 3.3))

function [ A ] = row ( x, order )
    A = [];
    for i = 0:order
        A = [A, x^i];
    end
end
```

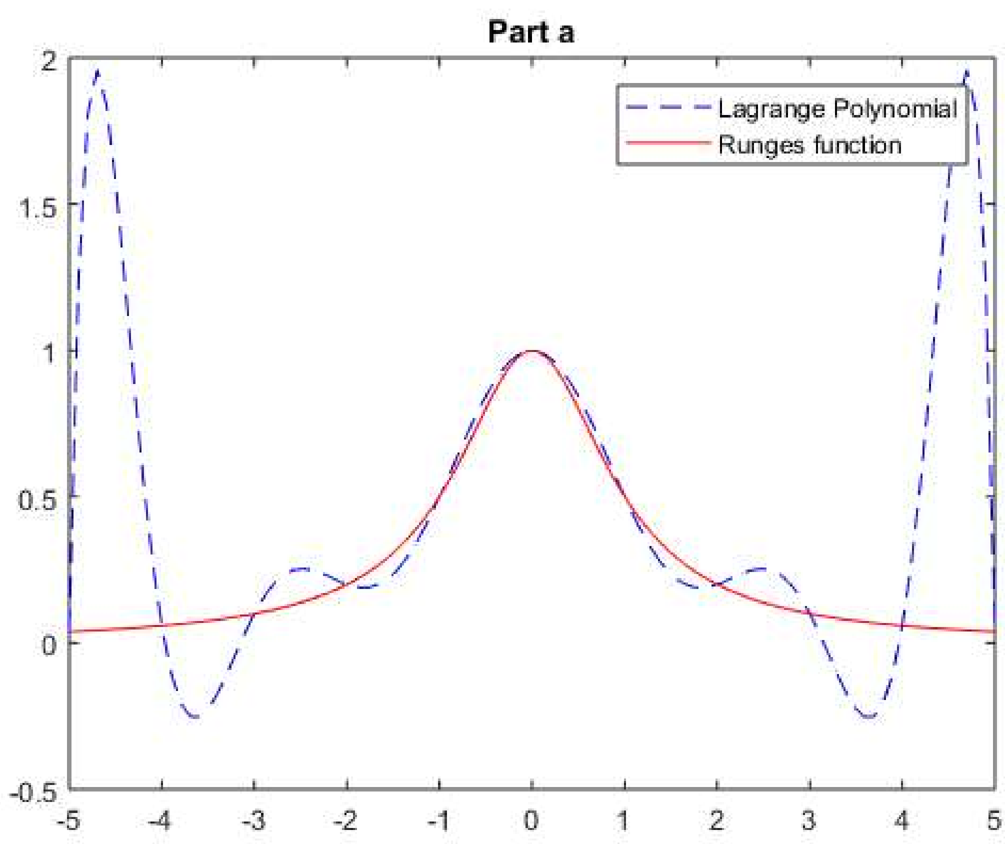
```
b =

    11
    29
    65
   125


x =

    5.0000
    2.0000
    3.0000
    1.0000


T =

[ 1, t, t^2, t^3]


P =

t^3 + 3*t^2 + 2*t + 5


ans =
```

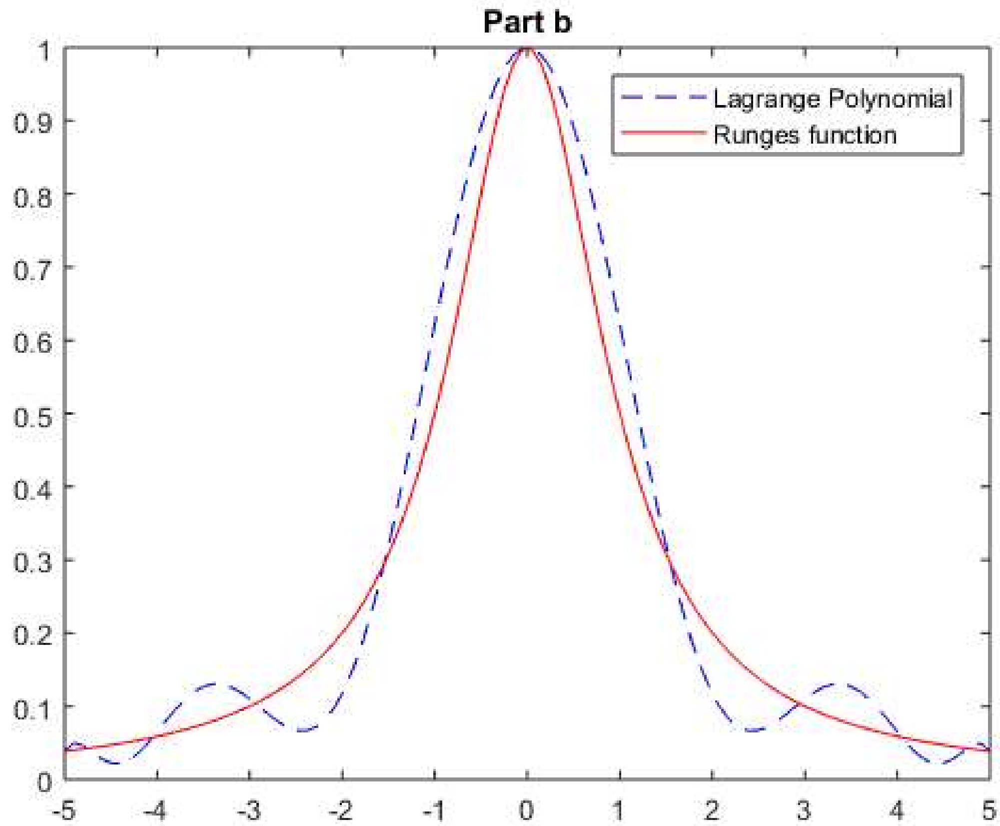
```
80.207
```

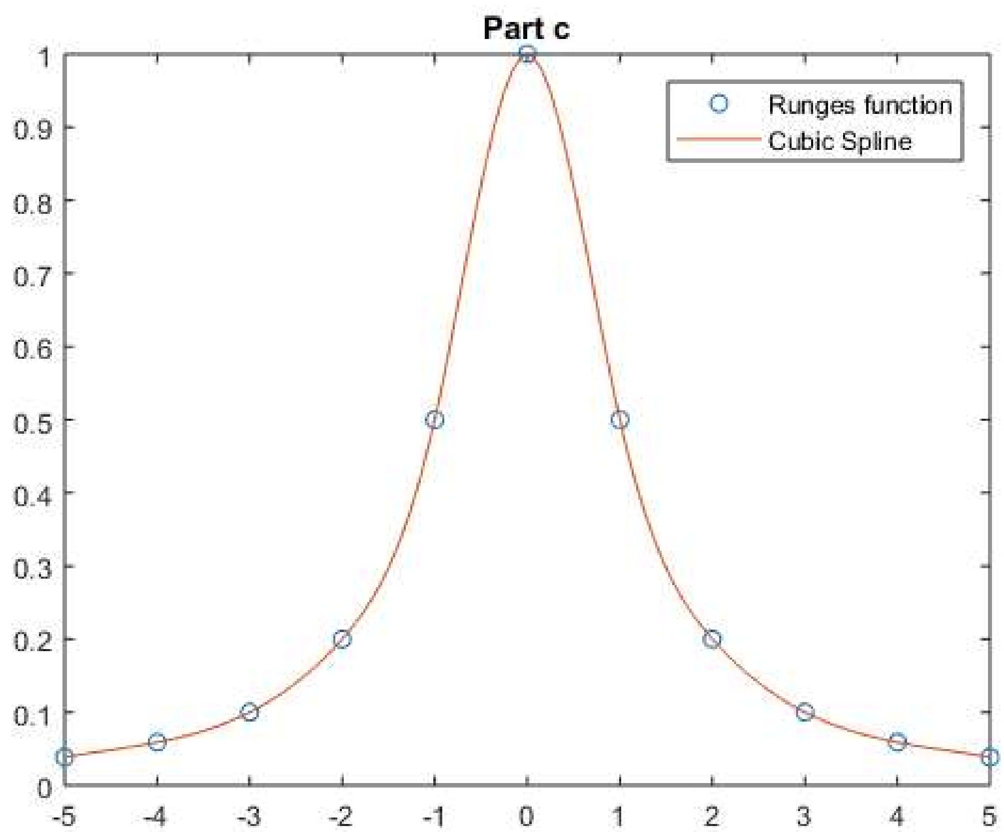
Problem 2

```
syms t

% Part a

X = -5:5;
Y = runge(X);

A = [];
for i = 1:11
    A = [A ; row(X(i), 10)];
end

x = A\Y';
T = row (t, 10);
P = T * x;

X = linspace(-5, 5);
y1 = vpa (subs( P, t, X));
y2 = runge(X);

figure
plot(X,y1,'b--',X,y2,'r')
legend('Lagrange Polynomial','Runges function')
title('Part a')

% Part b

X = myCos(10);
Y = runge(X);

A = [];
for i = 1:11
    A = [A ; row(X(i), 10)];
end

x = A\Y';
T = row (t, 10);
P = T * x;

X = linspace(-5, 5);
y1 = vpa (subs( P, t, X));
y2 = runge(X);

figure
plot(X,y1,'b--',X,y2,'r')
legend('Lagrange Polynomial','Runges function')
title('Part b')

% Part c

x = -5:5;  y = runge(x);
xx = linspace(-5, 5);
```

```
yy = spline(x,y,xx);
figure
plot(x,y,'o',xx,yy)
legend('Runges function','Cubic Spline')
title('Part c')

function [ y ] = runge ( x )
    y = 1 ./ (1 + x.^2);
end

function [ C ] = myCos ( n )
    J = 0:n;
    C = 5 .* cos(J .* pi ./ n);
end

function [ A ] = row ( x, order )
    A = [];
    for i = 0:order
        A = [A, x^i];
    end
end
```




Part b
Lagrange Polynomial
Runges function
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-5
-4
-3
-2
-1
0
1
2
3
4
5


Part c
Runges function
Cubic Spline
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-5
-4
-3
-2
-1
0
1
2
3
4
5

2a. I observe that the interpolating Lagrange polynomial is not very good at replicating the behaviour of the Runge's function. The aproximation is specially poor toward the boundaries {-5, 5} of the selected range

b. This interpolating polynomial behaves much more similarly to the original Runge's function, specially toward the boundaries of the range. Still, it is bad at replicating the behaviour of the function

Problem 3

```
syms t;

% Part a
fprintf(' # part A\n')

% We are changing the range to have smaller values to avoid large operations like 2000^7
X = [1978:2:1992] - 1976;
Y = [12, 12.7, 13, 15.2, 18.2, 19.8, 24.1, 28.1];

P = Lagrange(X, Y)

XX = linspace(2, 16);
y1 = vpa (subs( P, t, XX));

figure
plot(XX,y1,'b--',X,Y,'o')
xticklabels(1978:2:1992)
legend('Lagrange Polynomial','Data Point')
title('\nPart a')

fprintf('Estimate for 1994 using Lagrange: %f \n', vpa (subs( P, t, 18)));

% Part b
fprintf('\n # Part B\n\n')

x = [1978:2:1992];  y = [12, 12.7, 13, 15.2, 18.2, 19.8, 24.1, 28.1];
xx = linspace(1978, 1992);
yy = spline(x,y,xx);
figure
plot(x,y,'o',xx,yy)
xticklabels(1978:2:1992)
legend('Data Point','Cubic Spline')
title('Part b')

fprintf('Estimate for 1994 using Cubic Spline: %f \n', spline(x,y,1994));

% Part c
fprintf('\n # part C\n')

x = [1978:2:1992] - 1976;  y = [12, 12.7, 13, 15.2, 18.2, 19.8, 24.1, 28.1];
clear = [3, 4];
clear = sort(clear, 'descend');
for i = 1:length(clear)
    x(clear(i)) = [];
    y(clear(i)) = [];
end

P = Lagrange(x, y)

XX = linspace(2, 16);
y1 = vpa (subs( P, t, XX));
```

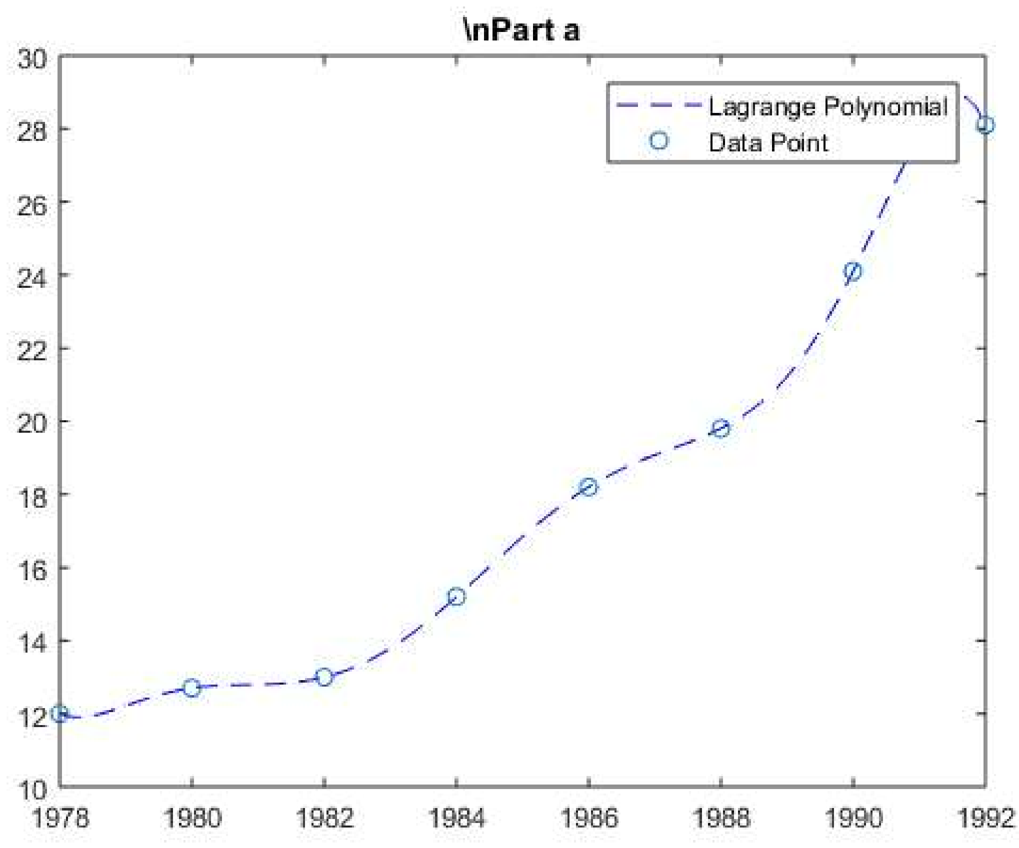
```
figure
plot(XX,y1,'b--',x,y,'o')
xticklabels(1978:2:1992)
legend('Lagrange Polynomial','Data Point')
title('Part c1')

fprintf('Estimate for 1982 and 1984 using Lagrange: %f, %f \n', vpa (subs( P, t, 6)), vpa (su
bs( P, t, 8)));

x = x + 1976;
xx = linspace(1978, 1992);
yy = spline(x,y,xx);
figure
plot(x,y,'o',xx,yy)
xticklabels(1978:2:1992)
legend('Data Point','Cubic Spline')
title('Part c2')

fprintf('Estimate for 1982 and 1984 using Spline: %f, %f \n', spline(x,y,1982), spline(x,y,19
84));


% Function Definitions

function [ LP ] = Lagrange ( X, Y )
    t = sym('t');
    order = length(X);
    A = [];
    for i = 1:order
        A = [A ; row(X(i), order - 1)];
    end

    x = A\Y';
    T = row (t, order - 1);
    LP = T * x;

    function [ A ] = row ( x, order )
        A = [];
        for j = 0:order
            A = [A, x^j];
        end
    end
end

function [ A ] = row ( x, order )
    A = [];
    for i = 0:order
        A = [A, x^i];
    end
end
```

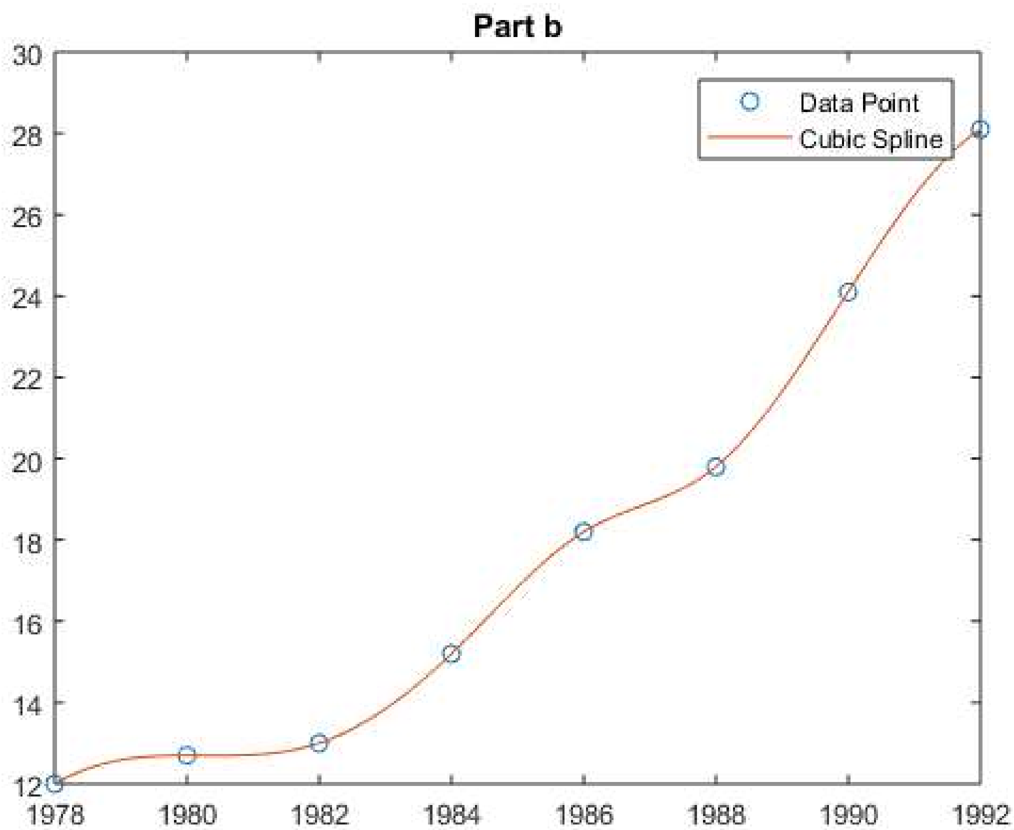
```
# part A

P =
```

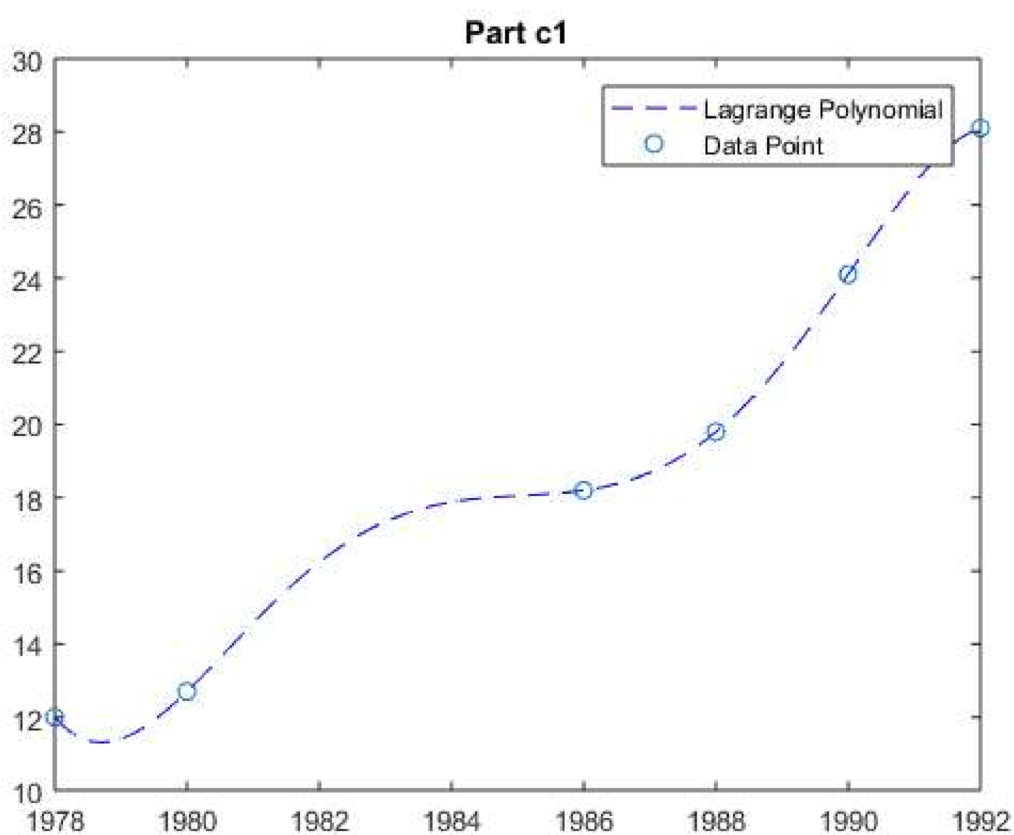
```
- (5924735509799193*t^7)/147573952589676412928 + (2719673774979989*t^6)/1152921504606846976 -
 (8035797835136663*t^5)/144115188075855872 + (6126381055618849*t^4)/9007199254740992 - (51456
75324261571*t^3)/1125899906842624 + (4729932874285817*t^2)/281474976710656 - (434867110895739
1*t)/140737488355328 + 4771000855260093/140737488355328

Estimate for 1994 using Lagrange: -38.400000
```

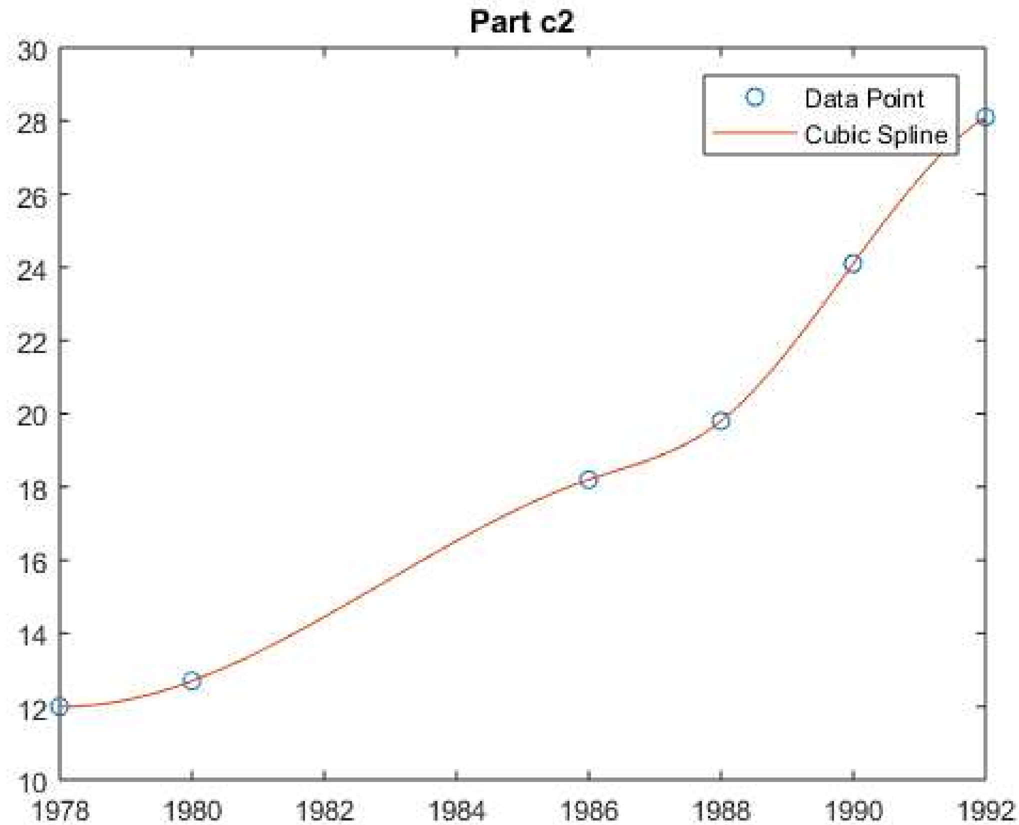
This prediction is extremely wrong, probably due to the fact that any polynomial increases its number of maximuns and minimums as it increases order. This means that high order polynomial such as this one have lots of wiggles which make the prediction worse, specially toward the edge.

```
 # Part B

Estimate for 1994 using Cubic Spline: 26.440670

 # part C

P =

- (485387959838791*t^5)/576460752303423488 + (2810996767416921*t^4)/72057594037927936 - (5941
311258413367*t^3)/9007199254740992 + (5601117474061161*t^2)/1125899906842624 - (8609224912669
087*t)/562949953421312 + 7703031862648063/281474976710656

Estimate for 1982 and 1984 using Lagrange: 16.233333, 17.880000
Estimate for 1982 and 1984 using Spline: 14.448187, 16.522280
```




Part b
Data Point
Cubic Spline
30
28
26
24
22
20
18
16
14
12
1978
1980
1982
1984
1986
1988
1990
1992


Part c1
Lagrange Polynomial
Data Point
30
28
26
24
22
20
18
16
14
12
10
1978
1980
1982
1984
1986
1988
1990
1992



Published with MATLAB® R2017a

a.

A function s that satisfies:

- on each interval $[x_j, x_{j+1}]$

$$s(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2$$

- s interpolates f at $x_0, x_1, \dots, x_n$
- s continuous on $[x_0, x_n]$
- s' continuous on $[x_0, x_n]$

$$\begin{cases} s_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 \\ s_j'(x) = b_j + 2c_j(x - x_j) \end{cases}$$

(1) $s_j(x_j) = y_j \Rightarrow \boxed{a_j = y_j}$ $\forall j$ ← interpolates the y's

(2) $s_j(x_{j+1}) = s_{j+1}(x_{j+1})$ ← s continuous

$$a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 = a_{j+1}$$

(3) $s_j'(x_{j+1}) = s_{j+1}'(x_{j+1})$ ← s' continuous

$$b_j + 2c_j(x_{j+1} - x_j) = b_{j+1}$$

$\therefore \boxed{c_j = \dfrac{b_{j+1} - b_j}{2h_j}}$ $\overset{(2)}{\Longrightarrow} a_{j+1} = a_j + b_j h_j + \dfrac{(b_{j+1} - b_j)}{2\cancel{h_j}} h_j^{\cancel{2}} =$

$= a_j + b_j\left(h_j - \dfrac{h_j}{2}\right) + \dfrac{b_{j+1}h_j}{2} \Rightarrow b_{j+1} = \dfrac{2}{h_j}\left[a_{j+1} - a_j - \dfrac{b_j h_j}{2}\right]$

$$\therefore \boxed{b_j = 2\frac{a_{j+1} - a_j}{h_j} - b_{j+1}}$$

$\therefore \quad b_j + b_{j+1} = \left(\frac{y_{j+1} - y_j}{h_j}\right) 2 = f_j$

$$\begin{bmatrix} 1 & 1 & 0 & \cdots & & & 0 \\ 0 & 1 & 1 & 0 & \cdots & & 0 \\ & & & & & & \\ 0 & & \cdots & & 0 & 0 & 1 & 1 \\ 0 & & \cdots & & & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-2} \\ f_{n-1} - b_n \end{Bmatrix}$$

b.

Note that b_n remains as a free paramete that needs to be chosen somehow, the end condition (which is 1 instead of 2 as in cubic spline)

$\therefore \quad b_j = S_j'(x_j)$ thus choosing b_n is equivalent to specifying the first derivative of S at point n

In the light of the above it would seem that "clamped spline" would be a good choice and we should set

$S'(x_n) = y_n'$

Problem 5

```
A = [
    2 -3 3
    3 3 9
    3 3 5
    ];

U = [
    2 -3 3
    0 7.5 4.5
    0 0 -4
    ];

L = [
    1 0 0
    1.5 1 0
    1.5 1 1
    ];

% Checking my answers
L * U

[LM UM PM] = lu(A)
```

```
ans =

     2    -3     3
     3     3     9
     3     3     5


LM =

    1.0000         0         0
    0.6667    1.0000         0
    1.0000         0    1.0000


UM =

     3     3     9
     0    -5    -3
     0     0    -4


PM =

     0     1     0
     1     0     0
     0     0     1
```

Published with MATLAB® R2017a

5a.

$$\left[\begin{array}{ccc|c} 2 & -3 & 3 & 2 \\ 3 & 3 & 9 & 15 \\ 3 & 3 & 5 & 11 \end{array}\right] \begin{array}{l} \\ \leftarrow -\frac{3}{2}R_1 \quad L_{21} \\ \leftarrow -\frac{3}{2}R_1 \quad L_{31} \end{array}$$

$$\left[\begin{array}{ccc|c} 2 & -3 & 3 & 2 \\ 0 & 7.5 & 4.5 & 12 \\ 0 & 7.5 & 0.5 & 8 \end{array}\right] \begin{array}{l} \\ \\ \leftarrow -\frac{7.5}{7.5}R_2 \quad L_{32} \end{array}$$

$$\left[\begin{array}{ccc|c} 2 & -3 & 3 & 2 \\ 0 & 7.5 & 4.5 & 12 \\ 0 & 0 & -4 & -4 \end{array}\right] \rightarrow \begin{cases} x_1 = 1 \\ x_2 = 1 \\ x_3 = 1 \end{cases}$$

$$U = \begin{bmatrix} 2 & -3 & 3 \\ 0 & 7.5 & 4.5 \\ 0 & 0 & -4 \end{bmatrix} \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ 1.5 & 1 & 0 \\ 1.5 & 1 & 1 \end{bmatrix}$$

The differenca lies in the fact that Matlab will allway switch colums before performing any operations so that the pivot element is the highest in the rows. This leads to a diff orde of the final answe, which is encoded in the P matrix.