

---

FACULTAD DE CIENCIAS

# Tarea 5

## ANÁLISIS NÚMÉRICO

Oscar Andrés Rosas Hernandez

Alarcón Alvarez Aylin

Pahua Castro Jesús Miguel Ángel

Diciembre 2018

# Índice

<b>1. Problemas de Computadora</b>	<b>2</b>
1.1. 12 . . . . .	3
1.2. 13 . . . . .	4
1.3. 14 . . . . .	6
1.4. 15 . . . . .	7
1.5. 16 . . . . .	8
1.6. 17 . . . . .	15
1.7. 18 . . . . .	17
<b>2. Anexo</b>	<b>19</b>
2.1. HermiteInterpolant . . . . .	19
2.2. LagrangeInterpolant . . . . .	20
2.3. NewtonInterpolant . . . . .	21
2.4. Spline3Interpolant . . . . .	23

## 1. Problemas de Computadora

Una nota importante es que al inicio de CADA script se incluyen los algoritmos, porfavor cambia el valor de la variable `Directory` para que sea un string que apunte desde donde estas a donde estan la carpeta de los algoritmos

Esta linea:

```
1 getd(pwd() + Directory);
```

Para ejecutar cada uno basta con hacer algo como:

```
1 exec("/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework4/Code/1.sce", -1)
```

## 1.1. 12

Ejecuta los scripts que esta dentro de Code llamado: 12a.sce y 12b.sce

En este código muestra justo lo que se nos pide, eso si, calcular este interpolante toma un par de segundos, ten paciencia.

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8 points = [
9     -0.5;
10    -0.25;
11     0;
12 ]
13
14 valuations = [
15    -0.024;
16     0.334;
17     1.101;
18 ]
19
20 derivatives = [
21     0.751;
22     2.189;
23     4.002;
24 ]
25
26 pointsToEvaluate = linspace(-1, 1, 20)';
27
28 pointsEvaluated = HermiteInterpolant(points, valuations, derivatives, pointsToEvaluate)
29
30 plot(points, valuations, "red")
31 plot(pointsToEvaluate, pointsEvaluated, "-blue")
32
33 xtitle("Hermite Interpolant", "x", "f(x)");

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Ángel
4
5 getd(pwd() + Directory);
6 clc;
7
8 points = [
9     0.1;
10     0.2;
11     0.3;
12     0.4;
13 ]
14
15 valuations = [
16    -0.620;
17    -0.283;
18     0.006;
19     0.248;
20 ]
21
22 derivatives = [
23     3.585;
24     3.140;
25     2.666;
26     2.165;
27 ]
28
29 pointsToEvaluate = linspace(-0.1, 0.6, 20)';
30
31 pointsEvaluated = HermiteInterpolant(points, valuations, derivatives, pointsToEvaluate)
32
33 plot(points, valuations, "red")
34 plot(pointsToEvaluate, pointsEvaluated, "-blue")
35
36 xtitle("Hermite Interpolant", "x", "f(x)");

```

## 1.2. 13

Ejecuta los scripts que esta dentro de Code llamado: 13a.sce, 13b.sce y 13c.sce

Creo que es obvio que gracias a la información en la derivada Hermite será mejor

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8
9 function [x] = f(x)
10     x = cos(x)
11 endfunction
12
13 points = [
14     0;
15     0.6;
16     0.9;
17 ]
18
19 valuations = [
20     f(points(1));
21     f(points(2));
22     f(points(3));
23 ]
24
25 derivatives = [
26     ( f(points(1) + 0.00001) - f(points(1)) ) / (0.00001);
27     ( f(points(2) + 0.00001) - f(points(2)) ) / (0.00001);
28     ( f(points(3) + 0.00001) - f(points(3)) ) / (0.00001);
29 ]
30
31 pointsToEvaluate = linspace(-0.1, 1, 30)';
32
33 pointsEvaluated1 = HermiteInterpolant(points, valuations, derivatives, pointsToEvaluate)
34 pointsEvaluated2 = LagrangeInterpolant(points, valuations, pointsToEvaluate)
35
36 plot(points, valuations, "red")
37 plot(pointsToEvaluate, pointsEvaluated1, "-blue")
38 plot(pointsToEvaluate, pointsEvaluated2, "-green")
39 plot(pointsToEvaluate, f(pointsToEvaluate), "-m")
40
41 hl=legend(['Points'; 'Hermite'; 'Lagrange'; 'Real']);
42 xtitle("Hermite Interpolant", "x", "f(x)");

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8
9 function [x] = f(x)
10     x = log(x + 1)
11 endfunction
12
13 points = [
14     0;
15     0.6;
16     0.9;
17 ]
18
19 valuations = [
20     f(points(1));
21     f(points(2));
22     f(points(3));
23 ]
24
25 derivatives = [
26     ( f(points(1) + 0.00001) - f(points(1)) ) / (0.00001);
27     ( f(points(2) + 0.00001) - f(points(2)) ) / (0.00001);
28     ( f(points(3) + 0.00001) - f(points(3)) ) / (0.00001);
29 ]
30
31 pointsToEvaluate = linspace(-0.1, 1, 30)';
32
33 pointsEvaluated1 = HermiteInterpolant(points, valuations, derivatives, pointsToEvaluate)
34 pointsEvaluated2 = LagrangeInterpolant(points, valuations, pointsToEvaluate)

```

```

35 plot(points, valuations, "red")
36 plot(pointsToEvaluate, pointsEvaluated1, "blue")
37 plot(pointsToEvaluate, pointsEvaluated2, "green")
38 plot(pointsToEvaluate, f(pointsToEvaluate), "m")
39
40 hl=legend(['Points'; 'Hermite'; 'Lagrange'; 'Real']);
41 xtitle("Hermite Interpolant", "x", "f(x)");
42

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pádua Castro Jesús Miguel Ángel
4
5 getd(pwd() + Directory);
6 clc;
7
8
9 function [x] = f(x)
10     x = sqrt(x + 1)
11 endfunction
12
13 points = [
14     0;
15     0.6;
16     0.9;
17 ]
18
19 valuations = [
20     f(points(1));
21     f(points(2));
22     f(points(3));
23 ]
24
25 derivatives = [
26     ( f(points(1) + 0.00001) - f(points(1)) ) / (0.00001);
27     ( f(points(2) + 0.00001) - f(points(2)) ) / (0.00001);
28     ( f(points(3) + 0.00001) - f(points(3)) ) / (0.00001);
29 ]
30
31 pointsToEvaluate = linspace(-0.1, 1, 30)';
32
33 pointsEvaluated1 = HermiteInterpolant(points, valuations, derivatives, pointsToEvaluate)
34 pointsEvaluated2 = LagrangeInterpolant(points, valuations, pointsToEvaluate)
35
36 plot(points, valuations, "red")
37 plot(pointsToEvaluate, pointsEvaluated1, "blue")
38 plot(pointsToEvaluate, pointsEvaluated2, "green")
39 plot(pointsToEvaluate, f(pointsToEvaluate), "m")
40
41 hl=legend(['Points'; 'Hermite'; 'Lagrange'; 'Real']);
42 xtitle("Hermite Interpolant", "x", "f(x)");

```

### 1.3. 14

Ejecuta los scripts que esta dentro de Code llamado: 14a.sce y 14b.sce

En este código muestra justo lo que se nos pide, eso si, grafico ambos interpolantes de Newton, para que veas que son iguales e incluso hice que dentro del interpolante de Newton nos genere un string con la representación del interpolante, se ve bonito.

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8 points = [
9     -0.5;
10    -0.25;
11     0;
12 ]
13
14 valuations = [
15    -0.024;
16     0.334;
17     1.101;
18 ]
19
20 pointsToEvaluate = linspace(-1, 1)';
21
22 pointsEvaluated = NewtonHomogeneousInterpolant(points, valuations, pointsToEvaluate)
23 pointsEvaluated2 = NewtonInterpolant(points, valuations, pointsToEvaluate)
24
25 plot(points, valuations, "red")
26 plot(pointsToEvaluate, pointsEvaluated2, "-green")
27 plot(pointsToEvaluate, pointsEvaluated, "-blue")
28
29 xtitle("Newton Homogeneous Interpolant", "x", "f(x)");

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8 points = [
9     0.1;
10     0.2;
11     0.3;
12     0.4;
13 ]
14
15 valuations = [
16    -0.620;
17    -0.283;
18     0.006;
19     0.248;
20 ]
21
22 pointsToEvaluate = linspace(-1, 1)';
23
24 pointsEvaluated = NewtonHomogeneousInterpolant(points, valuations, pointsToEvaluate)
25 pointsEvaluated2 = NewtonInterpolant(points, valuations, pointsToEvaluate)
26
27 plot(points, valuations, "red")
28 plot(pointsToEvaluate, pointsEvaluated2, "-green")
29 plot(pointsToEvaluate, pointsEvaluated, "-blue")
30
31 xtitle("Newton Homogeneous Interpolant", "x", "f(x)");

```

## 1.4. 15

Ejecuta los scripts que esta dentro de Code llamado: 15.sce

En este código muestra justo lo que se nos pide. Se puso todo en un solo archivo justo para comparar:

Ahora si:

- En [1,5] El mejor tiene que ser el polinomio por lo cerca que esta con respecto a la verdadera función gamma, digo no hay mucho mas que decir, fácil de deducir porque el polinomio tiene un grado mas que el Spline en este Castro
- En [1,2] Tiene mucho mejor rendimiento el Spline, por la misma razón, al solo representar un fragmento lo puede hacer mucho mejor, mientras que el polinomio tiene que ser el mismo para este fragmento y para todo lo demas, por lo que en giros bruscos es mucho mejor el spline. Como en este caso.

```

1
2 getd(pwd() + Directory);
3 clc;
4
5 x = [1; 2; 3; 4; 5];
6 y = [1; 1; 2; 6; 24];
7
8 A = [
9     1 x(1) x(1)^2 x(1)^3 x(1)^4
10    1 x(2) x(2)^2 x(2)^3 x(2)^4
11    1 x(3) x(3)^2 x(3)^3 x(3)^4
12    1 x(4) x(4)^2 x(4)^3 x(4)^4
13    1 x(5) x(5)^2 x(5)^3 x(5)^4
14 ]
15
16 Coefficients = inv(A) * y
17 a0 = Coefficients(1)
18 a1 = Coefficients(2)
19 a2 = Coefficients(3)
20 a3 = Coefficients(4)
21 a4 = Coefficients(5)
22
23 disp(Coefficients)
24
25 function [x] = f(x)
26     x = a0 + a1*x + a2*x^2 + a3*x^3 + a4*x^4
27 endfunction
28
29 points = linspace(1, 5)';
30
31 [estimations] = f(points);
32 plot(points, estimations, "black-");
33
34 disp(f(2))
35
36 [estimations] = Spline3Interpolant(x, y, points, 0);
37 plot(points, estimations, "blue-");
38 plot(points, gamma(points), "red-");
39 plot(x, y, "m*");
40
41 legend(['Poly'; 'Spline'; 'Real gamma'; 'Points'])
42
43 xtitle("Gamma", "x", "y");

```



## 1.5. 16

- a) En el inciso A hay solo que calcular condicionales y como dice que lo hagamos a través de una libreria lo hice usando la función de scilab. Donde si bien a todas las fue de la patada, tengo que admitir que la versión 4) es la mejor y por mucho.
- c) Pues hace lo que tiene que hacer :v
- d) Ahi notamos la ventaja del Spline, porque en los polinomio en los extremos tienen unas pendientes terriblemente feas. Pero el spline no :)  
No esta extremadamente cerca pero la idea esta bien.
- e) Toma su tiempo, no te preocupes, pero obviamente da el mismo polinomio.
- f) Da obviamente da el mismo polinomio.

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Ángel
4
5 getd(pwd() + Directory);
6 clc;
7
8 x = [
9     1900;
10    1910;
11    1920;
12    1930;
13    1940;
14    1950;
15    1960;
16    1970;
17    1980;
18 ];
19
20 y = [
21     076212168;
22     092228496;
23     106021537;
24     123202624;
25     132164569;
26     151325798;
27     179323175;
28     203302031;
29     226542199;
30 ];
31
32 function [y] = f1(t, j)
33     y = t^(j -1)
34 endfunction
35
36 function [y] = f2(t, j)
37     y = (t - 1900)^(j -1)
38 endfunction
39
40 function [y] = f3(t, j)
41     y = (t - 1940)^(j -1)
42 endfunction
43
44 function [y] = f4(t, j)
45     y = ((t - 1940) / 40)^(j -1)
46 endfunction
47
48 A1 = [
49     f1(x(1), 1)  f1(x(1), 2)  f1(x(1), 3)  f1(x(1), 4)  f1(x(1), 5)  f1(x(1), 6)  f1(x(1), 7)
50     f1(x(1), 8)  f1(x(1), 9)  f1(x(2), 3)  f1(x(2), 4)  f1(x(2), 5)  f1(x(2), 6)  f1(x(2), 7)
51     f1(x(2), 8)  f1(x(2), 9)  f1(x(3), 3)  f1(x(3), 4)  f1(x(3), 5)  f1(x(3), 6)  f1(x(3), 7)
52     f1(x(3), 8)  f1(x(3), 9)  f1(x(4), 3)  f1(x(4), 4)  f1(x(4), 5)  f1(x(4), 6)  f1(x(4), 7)
53     f1(x(4), 8)  f1(x(4), 9)  f1(x(5), 3)  f1(x(5), 4)  f1(x(5), 5)  f1(x(5), 6)  f1(x(5), 7)
54     f1(x(5), 8)  f1(x(5), 9)  f1(x(6), 3)  f1(x(6), 4)  f1(x(6), 5)  f1(x(6), 6)  f1(x(6), 7)
55     f1(x(6), 8)  f1(x(6), 9)  f1(x(7), 3)  f1(x(7), 4)  f1(x(7), 5)  f1(x(7), 6)  f1(x(7), 7)
56     f1(x(7), 8)  f1(x(7), 9)  f1(x(8), 3)  f1(x(8), 4)  f1(x(8), 5)  f1(x(8), 6)  f1(x(8), 7)
57     f1(x(8), 8)  f1(x(8), 9)

```

```

57      f1(x(9), 1) f1(x(9), 2) f1(x(9), 3) f1(x(9), 4) f1(x(9), 5) f1(x(9), 6) f1(x(9), 7)
58      f1(x(9), 8) f1(x(9), 9)
59  ]
60
61  A2 = [
62      f2(x(1), 1) f2(x(1), 2) f2(x(1), 3) f2(x(1), 4) f2(x(1), 5) f2(x(1), 6) f2(x(1), 7)
63      f2(x(1), 8) f2(x(1), 9)
64      f2(x(2), 1) f2(x(2), 2) f2(x(2), 3) f2(x(2), 4) f2(x(2), 5) f2(x(2), 6) f2(x(2), 7)
65      f2(x(2), 8) f2(x(2), 9)
66      f2(x(3), 1) f2(x(3), 2) f2(x(3), 3) f2(x(3), 4) f2(x(3), 5) f2(x(3), 6) f2(x(3), 7)
67      f2(x(3), 8) f2(x(3), 9)
68      f2(x(4), 1) f2(x(4), 2) f2(x(4), 3) f2(x(4), 4) f2(x(4), 5) f2(x(4), 6) f2(x(4), 7)
69      f2(x(4), 8) f2(x(4), 9)
70      f2(x(5), 1) f2(x(5), 2) f2(x(5), 3) f2(x(5), 4) f2(x(5), 5) f2(x(5), 6) f2(x(5), 7)
71      f2(x(5), 8) f2(x(5), 9)
72      f2(x(6), 1) f2(x(6), 2) f2(x(6), 3) f2(x(6), 4) f2(x(6), 5) f2(x(6), 6) f2(x(6), 7)
73      f2(x(6), 8) f2(x(6), 9)
74      f2(x(7), 1) f2(x(7), 2) f2(x(7), 3) f2(x(7), 4) f2(x(7), 5) f2(x(7), 6) f2(x(7), 7)
75      f2(x(7), 8) f2(x(7), 9)
76      f2(x(8), 1) f2(x(8), 2) f2(x(8), 3) f2(x(8), 4) f2(x(8), 5) f2(x(8), 6) f2(x(8), 7)
77      f2(x(8), 8) f2(x(8), 9)
78      f2(x(9), 1) f2(x(9), 2) f2(x(9), 3) f2(x(9), 4) f2(x(9), 5) f2(x(9), 6) f2(x(9), 7)
79      f2(x(9), 8) f2(x(9), 9)
80  ]
81
82  A3 = [
83      f3(x(1), 1) f3(x(1), 2) f3(x(1), 3) f3(x(1), 4) f3(x(1), 5) f3(x(1), 6) f3(x(1), 7)
84      f3(x(1), 8) f3(x(1), 9)
85      f3(x(2), 1) f3(x(2), 2) f3(x(2), 3) f3(x(2), 4) f3(x(2), 5) f3(x(2), 6) f3(x(2), 7)
86      f3(x(2), 8) f3(x(2), 9)
87      f3(x(3), 1) f3(x(3), 2) f3(x(3), 3) f3(x(3), 4) f3(x(3), 5) f3(x(3), 6) f3(x(3), 7)
88      f3(x(3), 8) f3(x(3), 9)
89      f3(x(4), 1) f3(x(4), 2) f3(x(4), 3) f3(x(4), 4) f3(x(4), 5) f3(x(4), 6) f3(x(4), 7)
90      f3(x(4), 8) f3(x(4), 9)
91      f3(x(5), 1) f3(x(5), 2) f3(x(5), 3) f3(x(5), 4) f3(x(5), 5) f3(x(5), 6) f3(x(5), 7)
92      f3(x(5), 8) f3(x(5), 9)
93      f3(x(6), 1) f3(x(6), 2) f3(x(6), 3) f3(x(6), 4) f3(x(6), 5) f3(x(6), 6) f3(x(6), 7)
94      f3(x(6), 8) f3(x(6), 9)
95      f3(x(7), 1) f3(x(7), 2) f3(x(7), 3) f3(x(7), 4) f3(x(7), 5) f3(x(7), 6) f3(x(7), 7)
96      f3(x(7), 8) f3(x(7), 9)
97      f3(x(8), 1) f3(x(8), 2) f3(x(8), 3) f3(x(8), 4) f3(x(8), 5) f3(x(8), 6) f3(x(8), 7)
98      f3(x(8), 8) f3(x(8), 9)
99      f3(x(9), 1) f3(x(9), 2) f3(x(9), 3) f3(x(9), 4) f3(x(9), 5) f3(x(9), 6) f3(x(9), 7)
100     f3(x(9), 8) f3(x(9), 9)
101  ]
102
103  A4 = [
104      f4(x(1), 1) f4(x(1), 2) f4(x(1), 3) f4(x(1), 4) f4(x(1), 5) f4(x(1), 6) f4(x(1), 7)
105      f4(x(1), 8) f4(x(1), 9)
106      f4(x(2), 1) f4(x(2), 2) f4(x(2), 3) f4(x(2), 4) f4(x(2), 5) f4(x(2), 6) f4(x(2), 7)
107      f4(x(2), 8) f4(x(2), 9)
108      f4(x(3), 1) f4(x(3), 2) f4(x(3), 3) f4(x(3), 4) f4(x(3), 5) f4(x(3), 6) f4(x(3), 7)
109      f4(x(3), 8) f4(x(3), 9)
110      f4(x(4), 1) f4(x(4), 2) f4(x(4), 3) f4(x(4), 4) f4(x(4), 5) f4(x(4), 6) f4(x(4), 7)
111      f4(x(4), 8) f4(x(4), 9)
112      f4(x(5), 1) f4(x(5), 2) f4(x(5), 3) f4(x(5), 4) f4(x(5), 5) f4(x(5), 6) f4(x(5), 7)
113      f4(x(5), 8) f4(x(5), 9)
114      f4(x(6), 1) f4(x(6), 2) f4(x(6), 3) f4(x(6), 4) f4(x(6), 5) f4(x(6), 6) f4(x(6), 7)
115      f4(x(6), 8) f4(x(6), 9)
116      f4(x(7), 1) f4(x(7), 2) f4(x(7), 3) f4(x(7), 4) f4(x(7), 5) f4(x(7), 6) f4(x(7), 7)
117      f4(x(7), 8) f4(x(7), 9)
118      f4(x(8), 1) f4(x(8), 2) f4(x(8), 3) f4(x(8), 4) f4(x(8), 5) f4(x(8), 6) f4(x(8), 7)
119      f4(x(8), 8) f4(x(8), 9)
120      f4(x(9), 1) f4(x(9), 2) f4(x(9), 3) f4(x(9), 4) f4(x(9), 5) f4(x(9), 6) f4(x(9), 7)
121      f4(x(9), 8) f4(x(9), 9)
122  ]
123
124  disp(cond(A1))
125  disp(cond(A2))
126  disp(cond(A3))
127  disp(cond(A4))
128
129  Coefficients = inv(A4) * y
130  a0 = Coefficients(1)
131  a1 = Coefficients(2)
132  a2 = Coefficients(3)
133  a3 = Coefficients(4)
134  a4 = Coefficients(5)
135  a5 = Coefficients(6)
136  a6 = Coefficients(7)
137  a7 = Coefficients(8)
138  a8 = Coefficients(9)
139
140  disp(Coefficients)

```

```

117 function [x] = f(x)
118     x = a0*f4(x, 1) + a1*f4(x, 2) + a2*f4(x, 3) + a3*f4(x, 4) + a4*f4(x, 5) + a5*f4(x, 6) +
        a6*f4(x, 7) + a7*f4(x, 8) + a8*f4(x, 9)
119 endfunction
120
121
122
123 points = linspace(1900, 1980)';
124
125 [estimations] = f(points);
126 plot(points, estimations, "black-");
127 plot(x, y, "m*");
128
129
130 legend(['Poly' ; 'Points'])
131
132 xtitle("Population data", "year", "population");

```

```

1
2 getd(pwd() + Directory);
3 clc;
4
5 x = [
6     1900;
7     1910;
8     1920;
9     1930;
10    1940;
11    1950;
12    1960;
13    1970;
14    1980;
15 ];
16
17 y = [
18     076212168;
19     092228496;
20     106021537;
21     123202624;
22     132164569;
23     151325798;
24     179323175;
25     203302031;
26     226542199;
27 ];
28
29 function [y] = f4(t, j)
30     y = ( (t - 1940) / 40 )^(j - 1)
31 endfunction
32
33 A4 = [
34     f4(x(1), 1)  f4(x(1), 2)  f4(x(1), 3)  f4(x(1), 4)  f4(x(1), 5)  f4(x(1), 6)  f4(x(1), 7)
        f4(x(1), 8)  f4(x(1), 9)
35     f4(x(2), 1)  f4(x(2), 2)  f4(x(2), 3)  f4(x(2), 4)  f4(x(2), 5)  f4(x(2), 6)  f4(x(2), 7)
        f4(x(2), 8)  f4(x(2), 9)
36     f4(x(3), 1)  f4(x(3), 2)  f4(x(3), 3)  f4(x(3), 4)  f4(x(3), 5)  f4(x(3), 6)  f4(x(3), 7)
        f4(x(3), 8)  f4(x(3), 9)
37     f4(x(4), 1)  f4(x(4), 2)  f4(x(4), 3)  f4(x(4), 4)  f4(x(4), 5)  f4(x(4), 6)  f4(x(4), 7)
        f4(x(4), 8)  f4(x(4), 9)
38     f4(x(5), 1)  f4(x(5), 2)  f4(x(5), 3)  f4(x(5), 4)  f4(x(5), 5)  f4(x(5), 6)  f4(x(5), 7)
        f4(x(5), 8)  f4(x(5), 9)
39     f4(x(6), 1)  f4(x(6), 2)  f4(x(6), 3)  f4(x(6), 4)  f4(x(6), 5)  f4(x(6), 6)  f4(x(6), 7)
        f4(x(6), 8)  f4(x(6), 9)
40     f4(x(7), 1)  f4(x(7), 2)  f4(x(7), 3)  f4(x(7), 4)  f4(x(7), 5)  f4(x(7), 6)  f4(x(7), 7)
        f4(x(7), 8)  f4(x(7), 9)
41     f4(x(8), 1)  f4(x(8), 2)  f4(x(8), 3)  f4(x(8), 4)  f4(x(8), 5)  f4(x(8), 6)  f4(x(8), 7)
        f4(x(8), 8)  f4(x(8), 9)
42     f4(x(9), 1)  f4(x(9), 2)  f4(x(9), 3)  f4(x(9), 4)  f4(x(9), 5)  f4(x(9), 6)  f4(x(9), 7)
        f4(x(9), 8)  f4(x(9), 9)
43 ]
44
45 disp(cond(A1))
46 disp(cond(A2))
47 disp(cond(A3))
48 disp(cond(A4))
49 // @Author: Rosas Hernandez Oscar Andres
50 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
51 // @Author: Pahua Castro Jesús Miguel Angel
52
53 Coefficients = inv(A4) * y
54 a0 = Coefficients(1)
55 a1 = Coefficients(2)
56 a2 = Coefficients(3)
57 a3 = Coefficients(4)
58 a4 = Coefficients(5)
59 a5 = Coefficients(6)

```

```

60 a6 = Coefficients(7)
61 a7 = Coefficients(8)
62 a8 = Coefficients(9)
63
64 disp(Coefficients)
65
66 function [x] = f(x)
67     x = a0*f4(x, 1) + a1*f4(x, 2) + a2*f4(x, 3) + a3*f4(x, 4) + a4*f4(x, 5) + a5*f4(x, 6) +
        a6*f4(x, 7) + a7*f4(x, 8) + a8*f4(x, 9)
68 endfunction
69
70 points = linspace(1900, 1980)';
71
72 [estimations] = f(points);
73 plot(points, estimations, "black-");
74 plot(x, y, "m*");
75
76 [estimations2] = Spline3Interpolant(x, y, points, 0);
77 plot(points, estimations2, "m-");
78
79 legend(['Poly' ; 'Points' ; 'Spline'])
80
81 xtitle("Population data", "year", "population");

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahuá Castro Jesús Miguel Ángel
4
5 getd(pwd() + Directory);
6 clc;
7
8 x = [
9     1900;
10    1910;
11    1920;
12    1930;
13    1940;
14    1950;
15    1960;
16    1970;
17    1980;
18 ];
19
20 y = [
21     076212168;
22     092228496;
23     106021537;
24     123202624;
25     132164569;
26     151325798;
27     179323175;
28     203302031;
29     226542199;
30 ];
31
32 function [y] = f4(t, j)
33     y = (t - 1940) / 40 ^ (j - 1)
34 endfunction
35
36 A4 = [
37     f4(x(1), 1) f4(x(1), 2) f4(x(1), 3) f4(x(1), 4) f4(x(1), 5) f4(x(1), 6) f4(x(1), 7)
38     f4(x(2), 1) f4(x(2), 2) f4(x(2), 3) f4(x(2), 4) f4(x(2), 5) f4(x(2), 6) f4(x(2), 7)
39     f4(x(3), 1) f4(x(3), 2) f4(x(3), 3) f4(x(3), 4) f4(x(3), 5) f4(x(3), 6) f4(x(3), 7)
40     f4(x(4), 1) f4(x(4), 2) f4(x(4), 3) f4(x(4), 4) f4(x(4), 5) f4(x(4), 6) f4(x(4), 7)
41     f4(x(5), 1) f4(x(5), 2) f4(x(5), 3) f4(x(5), 4) f4(x(5), 5) f4(x(5), 6) f4(x(5), 7)
42     f4(x(6), 1) f4(x(6), 2) f4(x(6), 3) f4(x(6), 4) f4(x(6), 5) f4(x(6), 6) f4(x(6), 7)
43     f4(x(7), 1) f4(x(7), 2) f4(x(7), 3) f4(x(7), 4) f4(x(7), 5) f4(x(7), 6) f4(x(7), 7)
44     f4(x(8), 1) f4(x(8), 2) f4(x(8), 3) f4(x(8), 4) f4(x(8), 5) f4(x(8), 6) f4(x(8), 7)
45     f4(x(9), 1) f4(x(9), 2) f4(x(9), 3) f4(x(9), 4) f4(x(9), 5) f4(x(9), 6) f4(x(9), 7)
46 ];
47
48 disp(cond(A1))
49 disp(cond(A2))
50 disp(cond(A3))
51 disp(cond(A4))
52
53 Coefficients = inv(A4) * y

```

```

54 a0 = Coefficients(1)
55 a1 = Coefficients(2)
56 a2 = Coefficients(3)
57 a3 = Coefficients(4)
58 a4 = Coefficients(5)
59 a5 = Coefficients(6)
60 a6 = Coefficients(7)
61 a7 = Coefficients(8)
62 a8 = Coefficients(9)
63
64 disp(Coefficients)
65
66 function [x] = f(x)
67     x = a0*f4(x, 1) + a1*f4(x, 2) + a2*f4(x, 3) + a3*f4(x, 4) + a4*f4(x, 5) + a5*f4(x, 6) +
68         a6*f4(x, 7) + a7*f4(x, 8) + a8*f4(x, 9)
69 endfunction
70
71 points = linspace(1900, 1990)';
72
73 [estimations] = f(points);
74 plot(points, estimations, "black-");
75 plot(x, y, "m*");
76
77 [estimations2] = Spline3Interpolant(x, y, points, 0);
78 plot(points, estimations2, "m-");
79
80 legend(['Poly' ; 'Points' ; 'Spline'])
81
82 xtitle("Population data", "year", "population");
83
84 disp("Poly:" + string(estimations(100)))
85 disp("Spline:" + string(estimations2(100)))

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahuá Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8 x = [
9     1900;
10    1910;
11    1920;
12    1930;
13    1940;
14    1950;
15    1960;
16    1970;
17    1980;
18 ];
19
20 y = [
21     076212168;
22     092228496;
23     106021537;
24     123202624;
25     132164569;
26     151325798;
27     179323175;
28     203302031;
29     226542199;
30 ];
31
32 function [y] = f4(t, j)
33     y = ( (t - 1940) / 40 )^(j - 1)
34 endfunction
35
36 A4 = [
37     f4(x(1), 1) f4(x(1), 2) f4(x(1), 3) f4(x(1), 4) f4(x(1), 5) f4(x(1), 6) f4(x(1), 7)
38     f4(x(1), 8) f4(x(1), 9)
39     f4(x(2), 1) f4(x(2), 2) f4(x(2), 3) f4(x(2), 4) f4(x(2), 5) f4(x(2), 6) f4(x(2), 7)
40     f4(x(2), 8) f4(x(2), 9)
41     f4(x(3), 1) f4(x(3), 2) f4(x(3), 3) f4(x(3), 4) f4(x(3), 5) f4(x(3), 6) f4(x(3), 7)
42     f4(x(3), 8) f4(x(3), 9)
43     f4(x(4), 1) f4(x(4), 2) f4(x(4), 3) f4(x(4), 4) f4(x(4), 5) f4(x(4), 6) f4(x(4), 7)
44     f4(x(4), 8) f4(x(4), 9)
45     f4(x(5), 1) f4(x(5), 2) f4(x(5), 3) f4(x(5), 4) f4(x(5), 5) f4(x(5), 6) f4(x(5), 7)
46     f4(x(5), 8) f4(x(5), 9)
47     f4(x(6), 1) f4(x(6), 2) f4(x(6), 3) f4(x(6), 4) f4(x(6), 5) f4(x(6), 6) f4(x(6), 7)
48     f4(x(6), 8) f4(x(6), 9)
49     f4(x(7), 1) f4(x(7), 2) f4(x(7), 3) f4(x(7), 4) f4(x(7), 5) f4(x(7), 6) f4(x(7), 7)
50     f4(x(7), 8) f4(x(7), 9)
51     f4(x(8), 1) f4(x(8), 2) f4(x(8), 3) f4(x(8), 4) f4(x(8), 5) f4(x(8), 6) f4(x(8), 7)
52     f4(x(8), 8) f4(x(8), 9)
53     f4(x(9), 1) f4(x(9), 2) f4(x(9), 3) f4(x(9), 4) f4(x(9), 5) f4(x(9), 6) f4(x(9), 7)
54     f4(x(9), 8) f4(x(9), 9)

```

```

46     f4(x(9), 8) f4(x(9), 9)
47 ]
48 disp(cond(A1))
49 disp(cond(A2))
50 disp(cond(A3))
51 disp(cond(A4))
52
53 Coefficients = inv(A4) * y
54 a0 = Coefficients(1)
55 a1 = Coefficients(2)
56 a2 = Coefficients(3)
57 a3 = Coefficients(4)
58 a4 = Coefficients(5)
59 a5 = Coefficients(6)
60 a6 = Coefficients(7)
61 a7 = Coefficients(8)
62 a8 = Coefficients(9)
63
64 disp(Coefficients)
65
66 function [x] = f(x)
67     x = a0*f4(x, 1) + a1*f4(x, 2) + a2*f4(x, 3) + a3*f4(x, 4) + a4*f4(x, 5) + a5*f4(x, 6) +
68         a6*f4(x, 7) + a7*f4(x, 8) + a8*f4(x, 9)
69 endfunction
70
71 points = linspace(1900, 1990)';
72
73 [estimations] = f(points);
74 plot(points, estimations, "black-");
75 plot(x, y, "m*");
76
77 [estimations2] = LagrangeInterpolant(x, y, points);
78 plot(points, estimations2, "m-");
79
80 legend(['Poly' ; 'Points' ; 'Lagrange'])
81
82 xtitle("Population data", "year", "population");

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Ángel
4
5 getd(pwd() + Directory);
6 clc;
7
8 x = [
9     1900;
10    1910;
11    1920;
12    1930;
13    1940;
14    1950;
15    1960;
16    1970;
17    1980;
18 ];
19
20 y = [
21     076212168;
22     092228496;
23     106021537;
24     123202624;
25     132164569;
26     151325798;
27     179323175;
28     203302031;
29     226542199;
30 ];
31
32 function [y] = f4(t, j)
33     y = ( (t - 1940) / 40 )^(j - 1)
34 endfunction
35
36 A4 = [
37     f4(x(1), 1) f4(x(1), 2) f4(x(1), 3) f4(x(1), 4) f4(x(1), 5) f4(x(1), 6) f4(x(1), 7)
38     f4(x(1), 8) f4(x(1), 9)
39     f4(x(2), 1) f4(x(2), 2) f4(x(2), 3) f4(x(2), 4) f4(x(2), 5) f4(x(2), 6) f4(x(2), 7)
40     f4(x(2), 8) f4(x(2), 9)
41     f4(x(3), 1) f4(x(3), 2) f4(x(3), 3) f4(x(3), 4) f4(x(3), 5) f4(x(3), 6) f4(x(3), 7)
42     f4(x(3), 8) f4(x(3), 9)
43     f4(x(4), 1) f4(x(4), 2) f4(x(4), 3) f4(x(4), 4) f4(x(4), 5) f4(x(4), 6) f4(x(4), 7)
44     f4(x(4), 8) f4(x(4), 9)
45     f4(x(5), 1) f4(x(5), 2) f4(x(5), 3) f4(x(5), 4) f4(x(5), 5) f4(x(5), 6) f4(x(5), 7)
46     f4(x(5), 8) f4(x(5), 9)
47     f4(x(6), 1) f4(x(6), 2) f4(x(6), 3) f4(x(6), 4) f4(x(6), 5) f4(x(6), 6) f4(x(6), 7)

```

```

43     f4(x(6), 8)  f4(x(6), 9)
44     f4(x(7), 1)  f4(x(7), 2)  f4(x(7), 3)  f4(x(7), 4)  f4(x(7), 5)  f4(x(7), 6)  f4(x(7), 7)
45     f4(x(7), 8)  f4(x(7), 9)
46     f4(x(8), 1)  f4(x(8), 2)  f4(x(8), 3)  f4(x(8), 4)  f4(x(8), 5)  f4(x(8), 6)  f4(x(8), 7)
47     f4(x(8), 8)  f4(x(8), 9)
48     f4(x(9), 1)  f4(x(9), 2)  f4(x(9), 3)  f4(x(9), 4)  f4(x(9), 5)  f4(x(9), 6)  f4(x(9), 7)
49     f4(x(9), 8)  f4(x(9), 9)
50 ]
51 disp(cond(A1))
52 disp(cond(A2))
53 disp(cond(A3))
54 disp(cond(A4))
55
56 Coefficients = inv(A4) * y
57 a0 = Coefficients(1)
58 a1 = Coefficients(2)
59 a2 = Coefficients(3)
60 a3 = Coefficients(4)
61 a4 = Coefficients(5)
62 a5 = Coefficients(6)
63 a6 = Coefficients(7)
64 a7 = Coefficients(8)
65 a8 = Coefficients(9)
66
67 disp(Coefficients)
68
69 function [x] = f(x)
70     x = a0*f4(x, 1) + a1*f4(x, 2) + a2*f4(x, 3) + a3*f4(x, 4) + a4*f4(x, 5) + a5*f4(x, 6) +
71         a6*f4(x, 7) + a7*f4(x, 8) + a8*f4(x, 9)
72 endfunction
73
74 points = linspace(1900, 1990)';
75
76 [estimations] = f(points);
77 plot(points, estimations, "black-");
78 plot(x, y, "m*");
79
80 [estimations2] = NewtonInterpolant(x, y, points);
81 plot(points, estimations2, "m-");
82
83 legend(['Poly' ; 'Points' ; 'Newton'])
84
85 xtitle("Population data", "year", "population");

```

## 1.6. 17

Ejecuta los scripts que esta dentro de Code llamado: 17.sce

En este código muestra justo lo que se nos pide, eso si, el snoppy se ve mas bonito se mueves la grafica, haciendola mas horizontal

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahua Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8 x1 = [
9     1;
10    2;
11    5;
12    6;
13    7;
14    8;
15   10;
16   13;
17   17;
18 ];
19
20 y1 = [
21     3.0;
22     3.7;
23     3.9;
24     4.2;
25     5.7;
26     6.6;
27     7.1;
28     6.7;
29     4.5;
30 ];
31
32 x2 = [
33     17;
34     20;
35     23;
36     24;
37     25;
38     27;
39    27.7;
40 ];
41
42 y2 = [
43     4.5;
44     7.0;
45     6.1;
46     5.6;
47     5.8;
48     5.2;
49     4.1;
50 ];
51
52 x3 = [
53    27.7;
54    28;
55    29;
56    30;
57 ];
58
59 y3 = [
60     4.1;
61     4.3;
62     4.1;
63     3.0;
64 ];
65
66 points1 = linspace(1, 17)';
67 plot(0, 0, "r");
68
69 [estimations1] = Spline3Interpolant(x1, y1, points1, [-1; +0.67]);
70 plot(points1, estimations1, "blue-");
71 plot(x1, y1, "m*");
72
73 points2 = linspace(17, 27.7)';
74
75 [estimations2] = Spline3Interpolant(x2, y2, points2, [-3; +4]);
76 plot(points2, estimations2, "red-");
77 plot(x2, y2, "m*");

```



```
78 points3 = linspace(27.7, 30, 20)';
79
80
81 [estimations3] = Spline3Interpolant(x3, y3, points3, [-0.33; 1.5]);
82 plot(points3, estimations3, "m-");
83 plot(x3, y3, "m*");
84
85 xtitle("Estimate a Snappy", "x", "y");
```

## 1.7. 18

Ejecuta los scripts que esta dentro de Code llamado: 18.sce

En este código muestra justo lo que se nos pide, eso si, el snoppy se ve mas bonito se mueves la grafica, haciendola mas horizontal y debo de admitir que con este spline me gusto mas como se ve

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Pahuá Castro Jesús Miguel Angel
4
5 getd(pwd() + Directory);
6 clc;
7
8 x1 = [
9     1;
10    2;
11    5;
12    6;
13    7;
14    8;
15   10;
16   13;
17   17;
18 ];
19
20 y1 = [
21     3.0;
22     3.7;
23     3.9;
24     4.2;
25     5.7;
26     6.6;
27     7.1;
28     6.7;
29     4.5;
30 ];
31
32 x2 = [
33     17;
34     20;
35     23;
36     24;
37     25;
38     27;
39    27.7;
40 ];
41
42 y2 = [
43     4.5;
44     7.0;
45     6.1;
46     5.6;
47     5.8;
48     5.2;
49     4.1;
50 ];
51
52 x3 = [
53    27.7;
54    28;
55    29;
56    30;
57 ];
58
59 y3 = [
60     4.1;
61     4.3;
62     4.1;
63     3.0;
64 ];
65
66 points1 = linspace(1, 17)';
67 plot(0, 0, "r");
68
69 [estimations1] = Spline3Interpolant(x1, y1, points1, 0);
70 plot(points1, estimations1, "blue-");
71 plot(x1, y1, "m*");
72
73 points2 = linspace(17, 27.7)';
74
75 [estimations2] = Spline3Interpolant(x2, y2, points2, 0);
76 plot(points2, estimations2, "red-");
77 plot(x2, y2, "m*");

```

```
78 points3 = linspace(27.7, 30, 20)';
79
80
81 [estimations3] = Spline3Interpolant(x3, y3, points3, 0);
82 plot(points3, estimations3, "m-");
83 plot(x3, y3, "m*");
84
85 xtitle("Estimate a Snappy", "x", "y");
```

## 2. Anexo

### 2.1. HermiteInterpolant

```
1 // Function to aproximate a function using the Lagrange method
2 // @param: points a vector of points
3 // @param: valuations a vector such valuations(i) = f(points(i))
4 // @param: pointsToEvaluate a vector of points to evaluate
5 // @return: pointsEvaluated such pointsEvaluated(i) = f(pointsToEvaluate(i))
6
7 // @Author: Rosas Hernandez Oscar Andres
8 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
9 // @Author: Pahuá Castro Jesús Miguel Angel
10
11 function [pointsEvaluated] = HermiteInterpolant(points, valuations, derivatives, pointsToEvaluate)
12     n = length(points)
13
14     numberOfEvaluations = length(pointsToEvaluate)
15     pointsEvaluated = zeros(numberOfEvaluations, 1)
16
17     HermiteEvaluations = zeros(numberOfEvaluations, n)
18     HermiteEvaluationsHat = zeros(numberOfEvaluations, n)
19
20     for evaluation = (1 : numberOfEvaluations)
21         for i = (1 : n)
22             HermiteEvaluationsHat(:, i) = HermiteHatPolynomial(points, i, pointsToEvaluate)
23             HermiteEvaluations(:, i) = HermitePolynomial(points, i, pointsToEvaluate)
24         end
25     end
26
27     for evaluation = (1 : numberOfEvaluations)
28         temporal = 0
29         for i = (1 : n)
30             temporal = temporal + valuations(i) * HermiteEvaluations(evaluation, i)
31             temporal = temporal + derivatives(i) * HermiteEvaluationsHat(evaluation, i)
32         end
33         pointsEvaluated(evaluation) = temporal
34     end
35 endfunction
36
37 function [pointsEvaluated] = HermitePolynomial(points, j, pointsToEvaluate)
38     numberOfEvaluations = length(pointsToEvaluate)
39     pointsEvaluated = zeros(numberOfEvaluations, 1)
40
41     for evaluation = (1 : numberOfEvaluations)
42         temporal = (LagrangePolynomial(points, j, points(j) + 0.0001) - LagrangePolynomial(points,
43             j, points(j))) / (0.0001)
44         pointsEvaluated(evaluation) = 1 - 2 * (pointsToEvaluate(evaluation) - points(j)) * temporal
45         pointsEvaluated(evaluation) = pointsEvaluated(evaluation) * LagrangePolynomial(points, j,
46             pointsToEvaluate(evaluation))^2
47     end
48 endfunction
49
50 function [pointsEvaluated] = HermiteHatPolynomial(points, j, pointsToEvaluate)
51     numberOfEvaluations = length(pointsToEvaluate)
52     pointsEvaluated = zeros(numberOfEvaluations, 1)
53
54     for evaluation = (1 : numberOfEvaluations)
55         pointsEvaluated(evaluation) = (pointsToEvaluate(evaluation) - points(j)) *
56             LagrangePolynomial(points, j, pointsToEvaluate(evaluation))^2
57     end
58 endfunction
```

## 2.2. LagrangeInterpolant

```

1 // Function to aproximate a function using the Lagrange method
2 // @param: points a vector of points
3 // @param: valuations a vector such valuations(i) = f(points(i))
4 // @param: pointsToEvaluate a vector of points to evaluate
5 // @return: pointsEvaluated such pointsEvaluated(i) = f(pointsToEvaluate(i))
6
7 // @Author: Rosas Hernandez Oscar Andres
8 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
9 // @Author: Pahuá Castro Jesús Miguel Angel
10
11 function [pointsEvaluated] = LagrangeInterpolant(points, valuations, pointsToEvaluate)
12     n = length(points)
13
14     numberOfEvaluations = length(pointsToEvaluate)
15     pointsEvaluated = zeros(numberOfEvaluations, 1)
16
17     LangrangeEvaluations = zeros(numberOfEvaluations, n)
18
19     for evaluation = (1 : numberOfEvaluations)
20         for i = (1 : n)
21             LangrangeEvaluations(:, i) = LagrangePolynomial(points, i, pointsToEvaluate)
22         end
23     end
24
25     for evaluation = (1 : numberOfEvaluations)
26         temporal = 0
27         for i = (1 : n)
28             temporal = temporal + valuations(i) * LangrangeEvaluations(evaluation, i)
29         end
30         pointsEvaluated(evaluation) = temporal
31     end
32 endfunction
33
34 function [pointsEvaluated] = LagrangePolynomial(points, k, pointsToEvaluate)
35     n = length(points)
36
37     numberOfEvaluations = length(pointsToEvaluate)
38     for evaluation = (1 : numberOfEvaluations)
39         temporal = 1
40         for i = (1 : n)
41             if (i ~= k)
42                 temporal = temporal * (pointsToEvaluate(evaluation) - points(i)) / (points(k) -
43                     points(i))
44             end
45         end
46         pointsEvaluated(evaluation) = temporal
47     end
48 endfunction

```

## 2.3. NewtonInterpolant

```

1 // Function to aproximate a function using the Newton method
2 // @param: points a vector of points
3 // @param: valuations a vector such valuations(i) = f(points(i))
4 // @param: pointsToEvaluate a vector of points to evaluate
5 // @return: pointsEvaluated such pointsEvaluated(i) = f(pointsToEvaluate(i))
6
7 // @Author: Rosas Hernandez Oscar Andres
8 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
9 // @Author: Pahuá Castro Jesús Miguel Angel
10
11 function [pointsEvaluated] = NewtonInterpolant(points, valuations, pointsToEvaluate)
12     n = length(points)
13     Differences = NewtonInterpolantCoefficients(points, valuations)
14
15     data = "I(x) = " + string(Differences(1))
16     for i = (2 : n)
17         data = data + " + " + string(Differences(i))
18         for j = (1 : i - 1)
19             if (points(j) > 0)
20                 data = data + "(x - " + string(points(j)) + ")"
21             else
22                 data = data + "(x + " + string(-1 * points(j)) + ")"
23             end
24         end
25     end
26
27     disp(data)
28
29     numberOfEvaluations = length(pointsToEvaluate)
30     pointsEvaluated = zeros(numberOfEvaluations, 1)
31
32     for evaluation = (1 : numberOfEvaluations)
33         temporal = Differences(n)
34         for j = (n - 1 : -1 : 1)
35             temporal = temporal * ( pointsToEvaluate(evaluation) - points(j) ) + Differences(j)
36         end
37         pointsEvaluated(evaluation) = temporal
38     end
39 endfunction
40
41 // Function to aproximate a function using the Newton method but the points should be
42 // from a homogenues partition
43 // @param: points a vector of points
44 // @param: valuations a vector such valuations(i) = f(points(i))
45 // @param: pointsToEvaluate a vector of points to evaluate
46 // @return: pointsEvaluated such pointsEvaluated(i) = f(pointsToEvaluate(i))
47
48 function [pointsEvaluated] = NewtonHomogeneousInterpolant(points, valuations, pointsToEvaluate)
49     n = length(points)
50     Differences = NewtonInterpolantCoefficients(points, valuations)
51
52     numberOfEvaluations = length(pointsToEvaluate)
53     pointsEvaluated = zeros(numberOfEvaluations, 1)
54     h = points(2) - points(1)
55
56     for evaluation = (1 : numberOfEvaluations)
57         s = ( pointsToEvaluate(evaluation) - points(1) ) / h
58
59         temporal = Differences(1)
60         for k = (2 : n)
61             temporal2 = Differences(k) * h^(k-1)
62             for term = (0 : k - 2)
63                 temporal2 = temporal2 * (s - term)
64             end
65             temporal = temporal + temporal2
66         end
67         pointsEvaluated(evaluation) = temporal
68     end
69 endfunction
70
71
72 // Function get all the Divided Differences for the Newton interpolant
73 // @param: points a vector of points
74 // @param: valuations a vector such valuations(i) = f(points(i))
75 // @return: Differences such Differences(i) = f[x_0 ... x_i]
76
77 function [Differences] = NewtonInterpolantCoefficients(points, valuations)
78     n = length(points)
79
80     for i = (1 : n)
81         Differences(i) = valuations(i)
82     end
83
84     for order = (1 : n - 1)
85         for i = (n : -1 : order + 1)

```

```
86         numerator = Differences(i) - Differences(i-1)
87         denominator = points(i) - points(i-order)
88         Differences(i) = numerator / denominator
89     end
90 end
91 endfunction
92
```

## 2.4. Spline3Interpolant

```

1 // Function to aproximate a function using the Spline (cubic) method
2 // @param: x a vector of points
3 // @param: y a vector such valuations(i) = f(x(i))
4 // @param: pointsToEvaluate a vector of points to evaluate
5 // @param: type if 0 then Spline natural if [a, b] then spline complete where
6 //           f(x_0)' = -a and f(x_n)' = -b
7 // @return: pointsEvaluated such pointsEvaluated(i) = f(pointsToEvaluate(i))
8
9 // @Author: Rosas Hernandez Oscar Andres
10 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
11 // @Author: Pádua Castro Jesús Miguel Angel
12
13
14 function [pointsEvaluated] = Spline3Interpolant(x, y, pointsToEvaluate, type)
15     [z, h] = Spline3Coefficients(x, y, type)
16
17     numberOfEvaluations = length(pointsToEvaluate)
18     pointsEvaluated = zeros(numberOfEvaluations, 1)
19
20     for evaluation = (1 : numberOfEvaluations)
21         point = pointsToEvaluate(evaluation)
22         i = FindPoint(point, x)
23
24         sum1 = z(i + 1) / (6 * h(i)) * (point - x(i))^3
25         sum2 = z(i) / (6 * h(i)) * (x(i+1) - point)^3
26         sum3 = ( y(i+1)/h(i) - z(i+1)/6 * h(i) ) * (point - x(i))
27         sum4 = ( y(i)/h(i) - z(i)/6 * h(i) ) * (x(i+1) - point)
28
29         pointsEvaluated(evaluation) = sum1 + sum2 + sum3 + sum4
30     end
31 endfunction
32
33
34 // Function to get the Z and H for the Spline method :v
35 // @param: t a vector of points
36 // @param: y a vector such valuations(i) = f(t(i))
37 // @return: z a vector of ?? well, of number to the next algorithm step
38 // @return: h a vector of distances
39
40
41 function [z, h] = Spline3Coefficients(t, y, type)
42     n = length(t);
43
44     h = zeros(n - 1, 1);
45     b = zeros(n - 1, 1);
46     u = zeros(n - 2, 1);
47     v = zeros(n - 2, 1);
48     z = zeros(n, 1);
49
50     for i = (1 : n - 1)
51         h(i) = t(i+1) - t(i);
52         b(i) = 6 * (y(i+1) - y(i)) / h(i);
53     end
54
55     u(2) = 2 * (h(1) + h(2));
56     v(2) = b(2) - b(1);
57
58     for i = (3 : n - 1)
59         u(i) = 2 * (h(i) + h(i-1)) - h(i-1)**2 / u(i-1);
60         v(i) = b(i) - b(i-1) - h(i-1) * v(i-1) / u(i-1);
61     end
62
63     if (length(type) == 1) z(n) = 0; else z(n) = type(2); end
64
65     for i = (n - 1 : -1 : 2)
66         z(i) = (v(i) - h(i) * z(i+1)) / u(i);
67     end
68
69     if (length(type) == 1) z(1) = 0; else z(1) = type(1); end
70 endfunction
71
72
73 // Function to find a interval in a distribution
74 // @param: point a point :v
75 // @param: data a vector of values
76 // @return: i a number such data(i) <= point <= data(i+1)
77
78 function [i] = FindPoint(point, data)
79     middle = 1, start = 1
80     final = length(data)
81
82     while (start < final)
83         middle = start + floor((final - start) / 2)
84
85         if (point < data(middle)) then final = middle;

```



```
86         else start = middle + 1;
87         end
88     end
89
90     i = start - 1;
91
92 endfunction
```