Laboratorio 6: Interpolación polinomial.

Sebastián Valencia Calderón
201111578

1 Introducción

El proceso mediante el cual se toman valores de datos en ciertos puntos es llamado interpolación. En ciertos casos, este problema y la aproximación por interpolación, puede tratarse como un problema de aproximación discreta. La interpolación, puede usarse para construir de manera constructiva ciertas funciones extraídas a partir del comportamiento de datos en $\mathbb{R}^2$, de manera más puntual, proporciona una metodología para la integración numérica, el tratamiento numérico de ecuaciones diferenciales y la discretización de un conjunto más grande de ecuaciones.

Los polinomios interpolantes, producto final de la interpolacion polinomial, rara vez son el producto de un proceso numérico, sino mas bien la base para la solución de un conjunto mas amplio de problemas solubles mediante esta metodología. Por lo tanto, estos polinomios aparecen frecuentemente en el análisis y desarrollo de algoritmos numéricos. Dado un conjunto de puntos, se desea explorar el comportamiento de ciertos algoritmos de interpolación, de manera numérica a través de implementaciones de los mismos algoritmos en el lenguaje de programación MATLAB. Ademas de los algoritmos, se presenta la solución a problemas aplicados, en los cuales se explora el comportamiento real de los algoritmos, contrastando su comportamiento, y el error generado por la ejecución de los mismos. El desarrollo de estos ejercicios, pretende comprometerse con los siguientes objetivos:

1. Identificar y entender algunos algoritmos de interpolación con polinomios y Splines.
2. Resolver algoritmos de interpolación mediante el uso de herramientas de computación matricial disponibles en MATLAB.
3. Contrastar la complejidad de implementación y funcionamiento de los algoritmos implementados.

2 Procedimiento

Para cumplir los objetivos enumerados anteriormente, se implementan tres algoritmos clásicos de interpolación, el de los polinomios interpoladores de Lagrange y el de diferencias de Newton, además, de una variación a los Splines cúbicos naturales, para reducir la oscilación de los polinomios sobre puntos ajenos al conjunto dado como argumento. Con los dos primeros polinomios, se explora un problema de predicción y modelado de datos discretos. Con el ultimo algoritmos, se pretende dibujar una figura con los polinomios resultantes dado un conjunto partido de datos discretos.

El desarrollo de cada uno de estos ejercicios, requiere comprender el funcionamiento y concepto de cada uno de los algoritmos usados para la interpolación de los datos dados. El desarrollo del laboratorio, comienza con la comprensión de cada uno de estos métodos o algoritmos, pero se centra fundamentalmente en la exploración de estos mismos a través de problemas prácticos.

3 Resultados

A continuación, se exponen los resultados, las metodologías propuestas para los análisis y las herramientas de ejecución para cada uno de los problemas propuestos.

3.1 Implementación de algoritmos en MATLAB

A continuación, se exponen las implementaciones para cada uno de los algoritmos básicos para la interpolación polinomial sobre los datos. Además, se compara cada resultado con la matriz natural, la cual surge al plantear de manera general el problema (esta aproximación, posee todos los problemas sobre representación y dimensionalidad estudiados anteriormente al estudiar la solución de sistemas lineales en un computador).

3.1.1 Sistema lineal para el polinomio interpolante

$$\begin{bmatrix} 1 & x_0^1 & x_0^2 & \dots & x_0^n \\ 1 & x_1^1 & x_1^2 & \dots & x_1^n \\ 1 & x_2^1 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n^1 & x_n^2 & \dots & x_n^n \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Figura 1: La solución de este sistema lineal, proporciona los coeficientes del polinomio interpolante para un conjunto de datos bi-dimensionales.

Script 1: Implementación en MATLAB del sistema lineal natural para los coeficientes del polinomio interpolante

```
function [A, sol] = validation(x, y)

    n = size(x, 2);     % Cardinality of data set degree
    A = zeros(n, n);    % The dependent-data matrix
    sol = y;            % We're supossed to solve A * a' = sol

    % Matrix population A(i, j) = x(i)^(j - 1)
    for i =1:n
        for j=1:n, A(i, j) = x(i)^(j - 1); end
    end;
end
```

3.1.2 Interpolación polinomial de Lagrange

A continuación, se anexa la funcion en MATLAB, que implementa el algoritmo para la obtención de polinomio interpolante de Lagrange. La metodología para este algoritmos, fue estudiada y basada en las de las referencias [2] y [5].

Script 2: Implementación en MATLAB del polinomio interpolador de Lagrange.

```
function lag = lagrange(x,y)
% Find the coefficients of the interpolant polynomial by Lagrange's method

    N = length(x);      % Data dimension
    lag = 0;            % Result polynomial

    for m = 1:N
      P = 1;            % Interpolant fraction
      for k = 1:N
        if k ~= m
            % The shape of the fraction
            P = conv(P, [1 - x(k)]);
            P = P / (x(m) - x(k));
        end
      end
      % Polynomial update by current term
      lag = lag + y(m)*P;
    end;
end
```

3.1.3 Interpolación polinomial de Newton

A continuación, se anexa la funcion en MATLAB, que implementa el algoritmo para la obtención de polinomio interpolante a partir del metodo de diferencias divididas de Newton-Gregory. La metodología para este algoritmos, fue estudiada y basada en las de las referencias [2] y [5].

Script 3: Implementación en MATLAB del polinomio de interpolacion haciendo uso el método de diferencias divididas de Newton-Gregory.

```
function n = newton(x,y)
    N = length(x)-1;
    DD = zeros(N + 1,N + 1);
    DD(1:N + 1,1) = y';
    for k = 2:N + 1
        for m = 1: N + 2 - k
            DD(m,k) = (DD(m + 1,k - 1) - DD(m,k - 1))/(x(m + k - 1)- x(m));
        end;
    end;

```

```
    a = DD(1,:);
    n = a(N+1);

    for k = N:-1:1
        n = [n a(k)] - [0 n*x(k)];
    end

end
```

3.1.4 Verificación de la implementación

Para la validación de los algoritmos frente a la formulación natural del problema, se propone validarlos contra un vector de datos aleatorios, obtener los coeficientes del polinomio interpolante haciendo uso de cada uno de los algoritmos, y obtener el error frente al polinomio obtenido al resolver el sistema lineal. El sistema lineal, es obtenido de acuerdo al esquema mostrado en la figura 1.

Script 4: Obtención del sistema lineal para la formulación natural del problema, esto es, de forma matricial.

```
function [A, sol] = validation(x, y)

    n = size(x, 2);     % Cardinality of data set degree
    A = zeros(n, n);    % The dependent-data matrix
    sol = y;            % We're supossed to solve A * a' = sol

    % Matrix population A(i, j) = x(i)^(j - 1)
    for i =1:n
        for j=1:n, A(i, j) = x(i)^(j - 1); end
    end;
end
```

Script 5: Obtención del sistema lineal para la formulación natural del problema, esto es, de forma matricial.

```
SIMULATIONS = 100;          % Number of simulations

lerror = zeros(SIMULATIONS, 1);     % Error by Lagrange's method
nerror = zeros(SIMULATIONS, 1);     % Error by Newton-Gregory method

for counter=1:SIMULATIONS

    n = randi(10, 1, 1);            % Random dimension
    x = rand(1, n);                 % Random data
    y = rand(1, n);                 % Random data

```

```
    lpol = lagrange(x, y);          % Polynomial by Lagrange method
    npol = newton(x, y);            % Polynomial by Newton method

    % Linear system formulation
    [A, sol] = validation(x, y);
    mpol = fliplr(linsolve(A, sol')');

    % Measuring absolute error in each case
    lerror(counter) = norm(lpol - mpol) / norm(mpol);
    nerror(counter) = norm(npol - mpol) / norm(mpol);
end;

axis = 1:SIMULATIONS;
Y = [lerror, nerror];
h = stem(axis, Y);

h(1).Color = 'green';
h(2).Marker = 'square';

xlabel('Simulaciones');
ylabel('Error obtenido');
title('Errores obtenidos en la interpolacion');
legend('Error obtenido en interpolacion de Lagrange', 'Error obtenido en
    interpolacion de Newton', 'location','southoutside');
set(gcf, 'Position', [400 400 700 700]);
saveas(gcf, 'interpo.png');
```

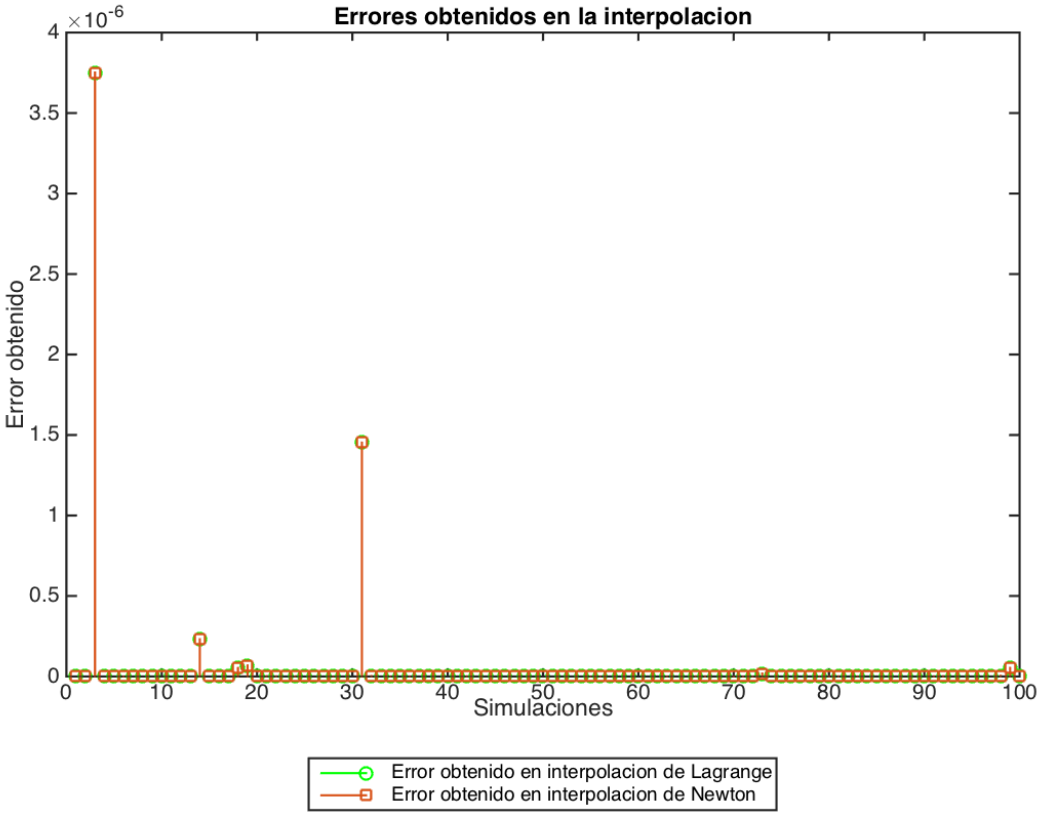


Figura 2: El resultado a la simulación, arroja valores alentadores sobre el error obtenido en cada uno de los algoritmos tomando como valor real, la solución del sistema lineal de la formulación general del problema. Puede verse la similitud en resultados de ambos algoritmos, y la baja magnitud del error de cada uno de los algoritmos.

3.2 Problema de aplicacion

Años después de 1900	Población en millones
0	76.0
20	105.7
40	131.7
60	179.3
80	226.5
100	281.4

Figura 3: Muestra los datos a usar, el año y la población de Estados Unidos correspondiente a ese año.

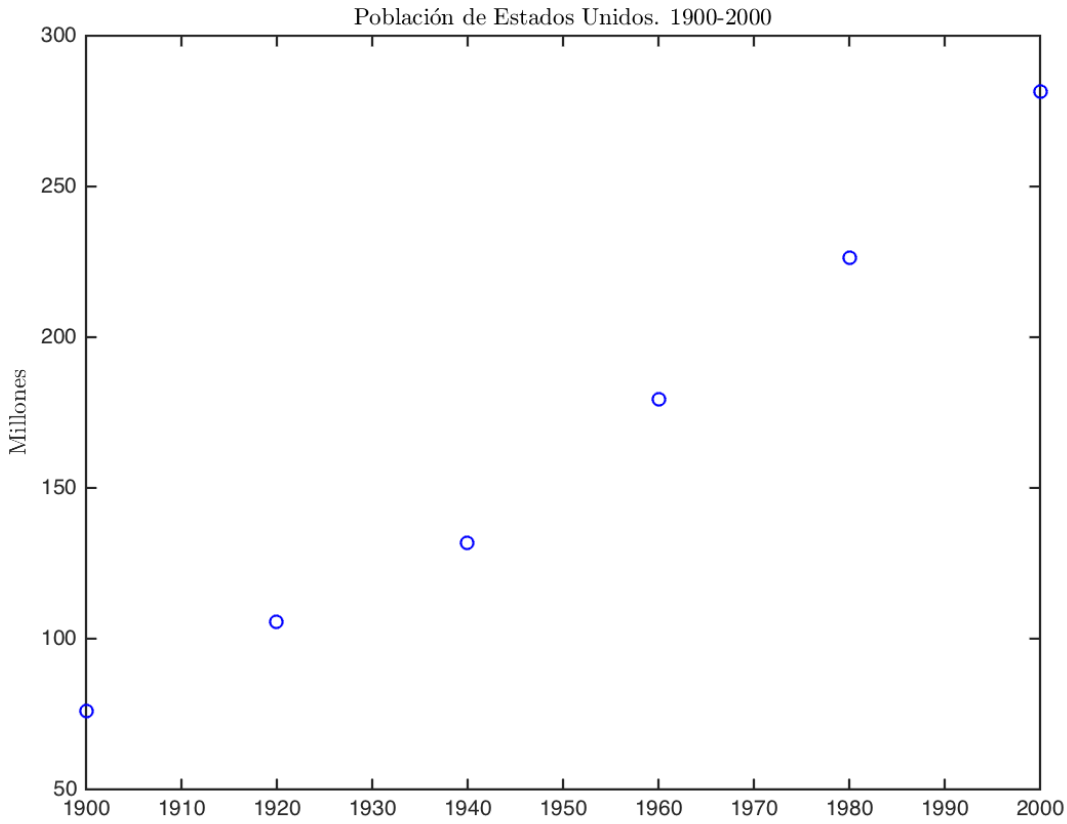
- Graficar la población contra el tiempo después de 1900 de acuerdo a la tabla presentada.

Script 6: Script para graficar los puntos de población en cada año.

```
year = [1900 1920 1940 1960 1980 2000];
pop = [76.0 105.7 131.7 179.3 226.5 281.4];

plot(year, pop,'bo');
title('Poblaci\''on de Estados Unidos. 1900--2000','interpreter','latex'
    )
ylabel('Millones','interpreter','latex');

set(gcf, 'Position', [400 400 1000 400]);
saveas(gcf, '../img/populationplot.png');
```



- Encuentre mediante cada uno de los métodos implementados el polinomio de quinto grado que pasa por cada uno de los puntos en la tabla.

  Script 7: Script para obtener los polinomios de interpolación usando cada método implementado anteriormente.

```
lag = lagrange(year, pop);
```

```
2 new = newton(year, pop);
```

Los coeficientes de los polinomios obtenidos son:

`lag` =

$$(2.0156e-07 \quad -0.00196 \quad 7.6812 \quad -1.4992e+04 \quad 1.4630e+07 \quad -5.7105e+09)$$

`new` =

$$(2.0156e-07 \quad -0.00196 \quad 7.6812 \quad -1.499e+04 \quad 1.463e+07 \quad -5.7105e+09)$$

- Determine mediante cada uno de los polinomios el valor estimado de la población de los Estados Unidos para el año 2020.

Script 8: Script para obtener los valores correspondientes al año 2020.

```
1 lval = polyval(lag, 2020);
2 nval = polyval(new, 2020);
```

`lval` = 459.5999 `nval` = 459.6000

- Grafique los polinomios encontrados. ¿Cree que el estimado encontrado es razonable?

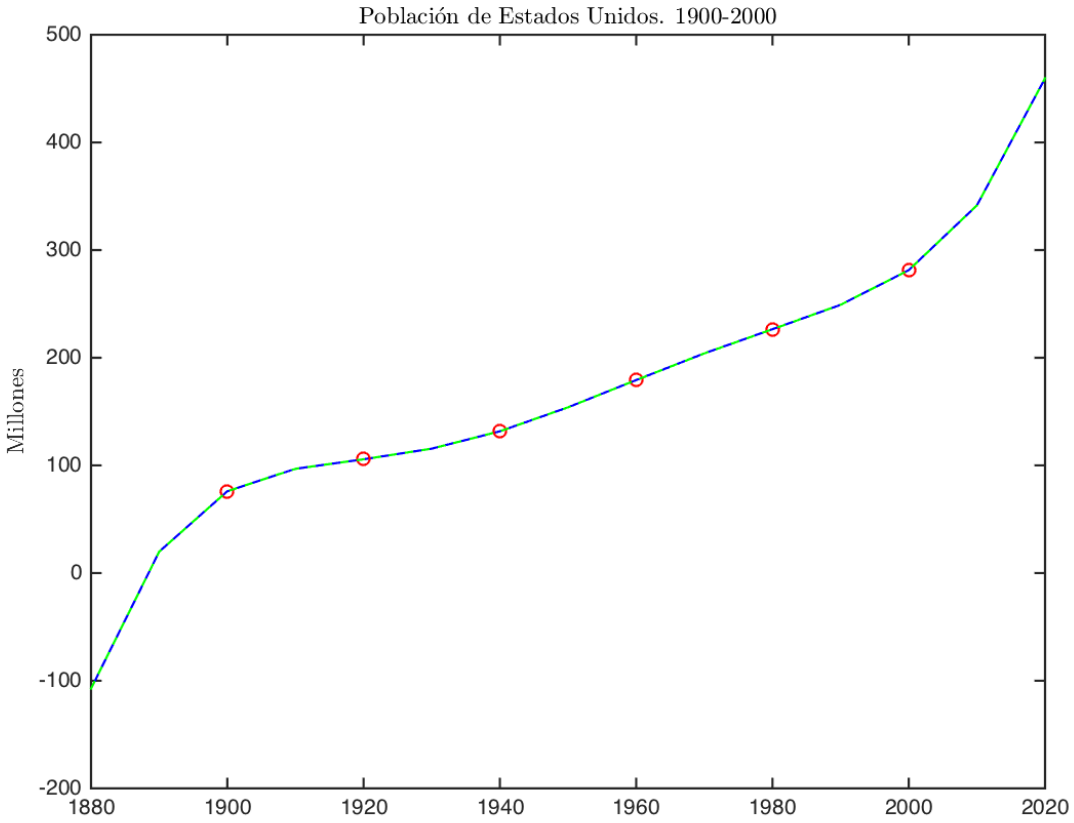


Figura 4: La gráfica de los puntos y los polinomios encontrados, confirma los resultados obtenidos anteriormente, primero, el comportamiento de ambos algoritmos es similar, de la misma manera, se tienen dos polinomios que incluyen los puntos de los cuales parte el polinomio resultante. Sobre la razón de este polinomio, si se compara el valor actual real con el pronosticado por el polinomio, se tiene que el valor es aproximado, si embargo, la aproximación del polinomio para valores menores a 1900, es mala, ya que por la naturaleza de la variable, este valor es positivo, y esto no se incorpora de ninguna manera al algoritmo usado.

- Encuentre mediante cada uno de los métodos implementados el polinomio de segundo grado que pasa por los tres primeros punto en la tabla. Estime los valores de los tres puntos restantes en la tabla con estos polinomios. Compare y concluya.

  Script 9: Script para comparar las predicciones realizadas, sobre un conjunto predictor de menor dimensión.

```
lval = polyval(lag, 2020);
nval = polyval(new, 2020);

year2 = [1900 1920 1940];
pop2 = [76.0 105.7 131.7];

```

```
lag2 = lagrange(year2, pop2);

lag2p1 = polyval(lag2, year(4));
lag2p2 = polyval(lag2, year(5));
lag2p3 = polyval(lag2, year(6));

abserrorp1 = abs(lag2p1 - pop(4)) / pop(4);
abserrorp2 = abs(lag2p2 - pop(5)) / pop(5);
abserrorp3 = abs(lag2p3 - pop(6)) / pop(6);
```

$$\texttt{abserrorp1} = 0.1411042$$

$$\texttt{abserrorp2} = 0.2379690$$

$$\texttt{abserrorp3} = 0.3336886$$

Puede verse, que al no incorporar la información de los otros años, no es posible inferir nada sobre datos fuera del rango con el cual se alimentó el algoritmo de interpolación. Esto, define que entre más datos se tenga, una mayor precisión puede tenerse.

3.3 Implementación de algoritmos en MATLAB (Splines)

A continuación, se anexa la función en MATLAB, que implementa el algoritmo para la obtención de polinomio interpolante por medio de splines cubicos con ajuste diferencial en los extremos. La metodología para este algoritmos, fue estudiada y basada en las de la referencia [5].

Script 10: Implementación en MATLAB de los splines cúbicos con criterios extremos sobre las derivadas.

```
function [S] = clamped(x, f, fpo, fpn)

    n = size(x, 2);
    h = zeros(n - 1, 1);

    % Companion tems of each polynomial temrm
    a = f'; b = zeros(n - 1, 1); c = zeros(n, 1); d = zeros(n - 1, 1);

    % Resulting polynomials
    S = zeros(n - 1, 4);

    % h's for natural spline
    for i=1:n-1
        h(i) = x(i + 1) - x(i);
    end;

    % alphas for coefficient population
```

```
18      alpha = zeros(n, 1);
19      alpha(1) = 3 * (f(2) - f(1)) / h(1) - 3*fpo;
20      alpha(n) = 3 * fpn - 3 * (f(n) - f(n - 1)) / h(n - 1);
21
22      for i=2:n-1
23          ls = 3 / h(i) * (f(i + 1) - f(i));
24          rs = 3 / h(i - 1) * (f(i) - f(i-1));
25          alpha(i) = ls - rs;
26      end;
27
28      l = zeros(n, 1);
29      mu = zeros(n, 1);
30      z = zeros(n, 1);
31
32      l(1) = 2 * h(1);
33      mu(1) = 0.5;
34      z(1) = alpha(1) / l(1);
35
36      for i=2:n-1
37          l(i) = 2*(x(i+1) - x(i-1)) - h(i-1)*mu(i-1);
38          mu(i) = h(i) / l(i);
39          z(i) = (alpha(i) - h(i-1)*z(i-1)) / l(i);
40      end;
41
42      l(n) = h(n-1)*(2 - mu(n-1));
43      z(n) = (alpha(n) - h(n-1)*z(n-1)) / l(n);
44      c(n) = z(n);
45
46      for j=n-1:-1:1
47          c(j) = z(j) - mu(j) * c(j+1);
48          b(j) = (f(j+1) - f(j))/h(j) - h(j)*(c(j+1) + 2*c(j)) / 3;
49          d(j) = (c(j+1) - c(j)) / (3*h(j));
50      end;
51
52      for j=1:n-1
53          syms y;
54          pol = a(j) + b(j) * (y - x(j)) + c(j) * (y - x(j))^2 + d(j) * (y - x
                (j))^3;
55          S(j,:) = sym2poly(pol);
56      end;
57
58  end
```

3.4 Problema de aplicación (Splines)

Curve 1				Curve 2				Curve 3			
i	x_i	$f(x_i)$	$f'(x_i)$	i	x_i	$f(x_i)$	$f'(x_i)$	i	x_i	$f(x_i)$	$f'(x_i)$
0	1	3.0	1.0	0	17	4.5	3.0	0	27.7	4.1	0.33
1	2	3.7		1	20	7.0		1	28	4.3	
2	5	3.9		2	23	6.1		2	29	4.1	
3	6	4.2		3	24	5.6		3	30	3.0	−1.5
4	7	5.7		4	25	5.8					
5	8	6.6		5	27	5.2					
6	10	7.1		6	27.7	4.1	−4.0				
7	13	6.7									
8	17	4.5	−0.67								

Figura 5

```
x1 = [1 2 5 6 7 8 10 13 17];
f1 = [3.0 3.7 3.9 4.2 5.7 6.6 7.1 6.7 4.5];
fpo1 = 1.0;
fpn1 = −0.67;

S1 = clamped(x1, f1, fpo1, fpn1);

slice1 = @(i) x1(i):0.001:x1(i + 1);
value1 = @(i) polyval(S1(i,:), slice1(i));

plot(slice1(1), value1(1), slice1(2), value1(2), slice1(3), value1(3),
    slice1(4), value1(4), slice1(5), value1(5), slice1(6), value1(6), slice1
    (7), value1(7), slice1(8), value1(8));
saveas(gcf, '../img/curve1.png');

x2 = [17 20 23 24 25 27 27.7];
f2 = [4.5 7.0 6.1 5.6 5.8 5.2 4.1];
fpo2 = 3;
fpn2 = −4;

S2 = clamped(x2, f2, fpo2, fpn2);

slice2 = @(i) x2(i):0.001:x2(i + 1);
value2 = @(i) polyval(S2(i,:), slice2(i));

plot(slice2(1), value2(1), slice2(2), value2(2), slice2(3), value2(3),
    slice2(4), value2(4), slice2(5), value2(5), slice2(6), value2(6));
saveas(gcf, '../img/curve2.png');

x3 = [27.7 28 29 30];
```

```
f3 = [4.1 4.3 4.1 3.0];
fpo3 = 0.33;
fpn3 = -1.5;

S3 = clamped(x3, f3, fpo3, fpn3);

slice3 = @(i) x3(i):0.001:x3(i + 1);
value3 = @(i) polyval(S3(i,:), slice3(i));

plot(slice3(1), value3(1), slice3(2), value3(2), slice3(3), value3(3));
saveas(gcf, '../img/curve3.png');

plot(slice1(1), value1(1), slice1(2), value1(2), slice1(3), value1(3),
    slice1(4), value1(4), slice1(5), value1(5), slice1(6), value1(6), slice1
    (7), value1(7), slice1(8), value1(8));
hold on
plot(slice2(1), value2(1), slice2(2), value2(2), slice2(3), value2(3),
    slice2(4), value2(4), slice2(5), value2(5), slice2(6), value2(6));
hold on
plot(slice3(1), value3(1), slice3(2), value3(2), slice3(3), value3(3));
saveas(gcf, '../img/beastcurve.png');
```

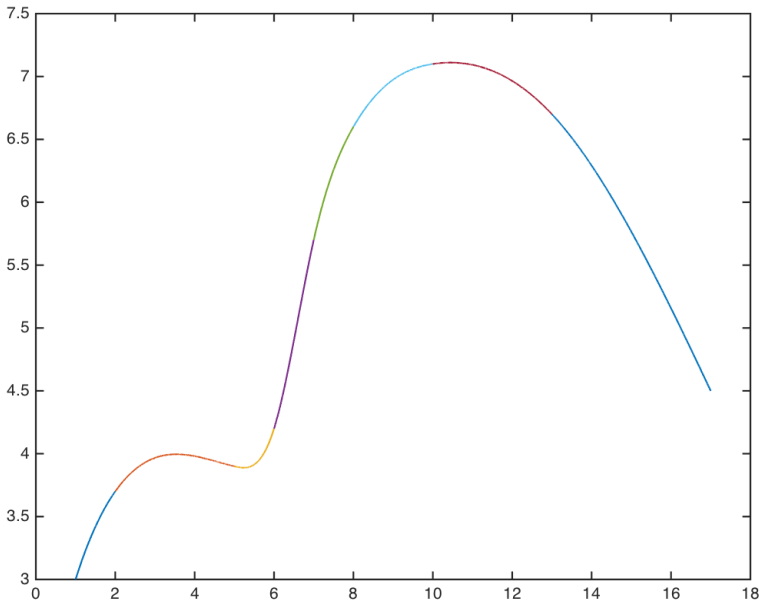


Figura 6: Aproximación para la curva 1.

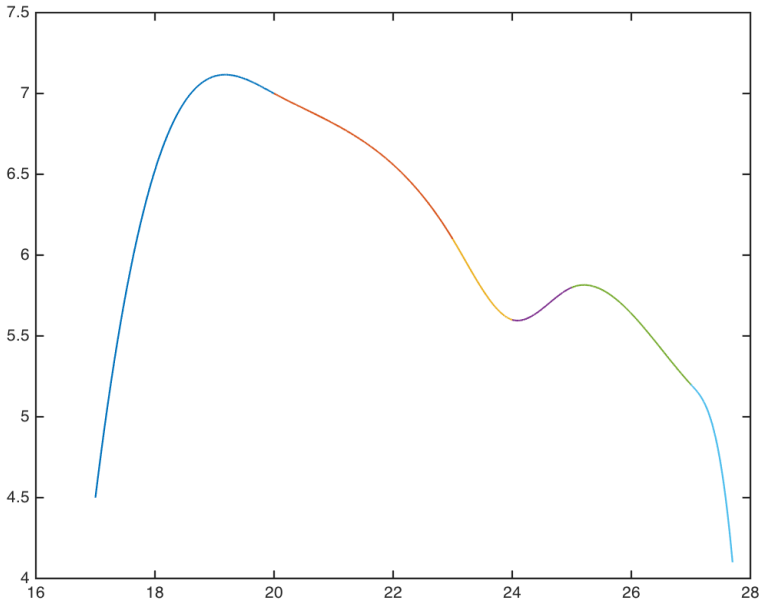


Figura 7: Aproximación para la curva 2.

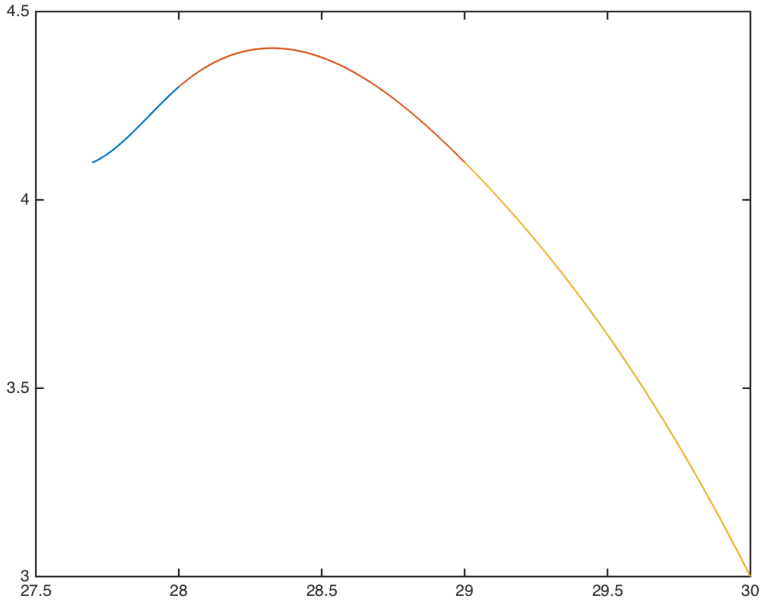


Figura 8: Aproximación para la curva 3.

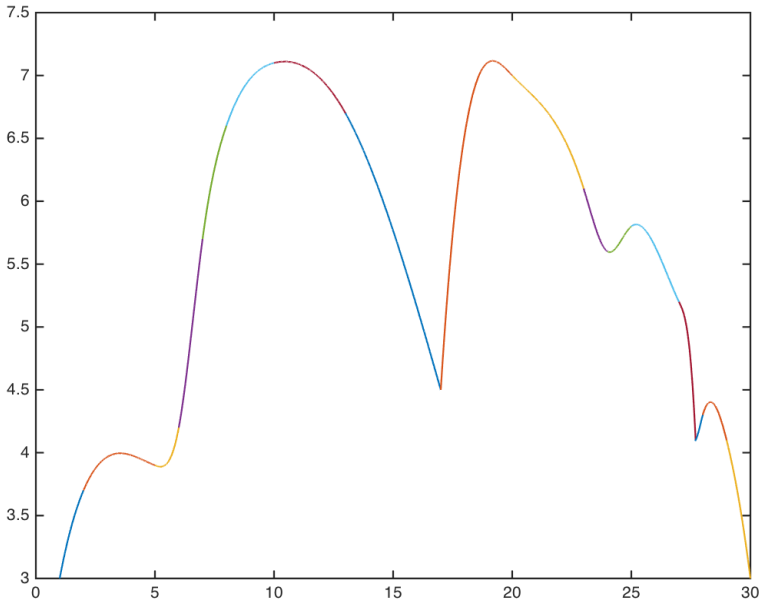


Figura 9: Unión de las curvas anteriores, esto completa la gráfica requerida según el enunciado.

4 Conclusiones

El desarrollo de los ejercicios propuestos para el laboratorio, sirvieron para conceptualizar los algoritmos fundamentales para la obtención de funciones analíticas a partir de una serie de datos discretos, es decir, los algoritmos para la interpolación polinomial con pares discretos de datos. Entender estos algoritmos, sus ventajas y modos de uso, permite extender la solución de otros problemas mucho más prácticos de análisis numérico, por ejemplo, se sabe que muchos problemas en diferenciación, integración numérica, y tratamiento de ecuaciones diferenciales, se pueden aproximar por medio de polinomios sobre pares de datos discretos.

Por medio del desarrollo del laboratorio, se identificaron algunos de estos algoritos (Lagrange, Newton-Gregory y Splines) para el tratamiento del problema de interpolación, sobre estos, se estudió la pertinencia de uso, su exactitud, y el problema dimensional que poseen, además, se aplicaron casos prácticos de predicción y de aproximación analítica a la interpolación de ciertos datos, ajustando el algoritmo con las derivadas de los extremos. Cada uno de estos algoritmos, fueron estudiados haciendo uso de MATLAB.

5 Bibliografía

[1] Ascher, U. M, Greif, C. *A First Course in Numerical Methods.* 2011. Society for Industrial & Applied Mathematics - Computational Science and Engineering. Página 93-108.

[2] Yang, W.Y. and Cao, W. and Chung, T.S. and Morris, J. *Applied Numerical Methods Using MATLAB.* 2005. Wiley. Pags. 50 - 57.

[3] Lopez, C. *MATLAB Symbolic Algebra and Calculus Tools.* 2014. MATLAB Solutions Series - Apress. Pags. 73 - 75.

[4] Bradie, B. *A Friendly Introduction to Numerical Analysis.* 2006. Pearson Prentice Hall - Featured Titles for Numerical Analysis Series. Página 150 - 174.

[5] Burden, R.L. and Faires, J.D. *Numerical Analysis.* 2010. Cengage Learning. Pags. 105 - 160.