

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Asignatura: Cálculo Científico (6105)

Estudiante: Naranjo Sthory Alexanyer Antonio

Cédula de identidad: V – 26.498.600

### **Tarea 7: Interpolación polinomial y Splines**

A continuación, se presentan las respuestas de cada una de las preguntas indicadas para la actual asignación. Se destaca también la carpeta cuyo nombre es **Codes**, que contiene el código fuente de la resolución de aquellos ejercicios que lo requieran. De igual manera, en el presente informe se indican aquellas preguntas que se solventaron a partir de una implementación en Matlab/Octave y también se adjuntan imágenes de aquellos fragmentos de código relevantes para la justificación de la respuesta.

## **Respuesta #2**

### ***a) Interpolación en la forma de Newton***

#### **Resumen Teórico:**

Dado  $(n + 1)$  puntos  $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ .

Los puntos definidos por  $(x_{y_0})_{0 \leq y_0 \leq n}$  se denominan **puntos de interpolación**.

Los puntos definidos por  $(y_{y_0})_{0 \leq y_0 \leq n}$  son los **valores de interpolación**. Para interpolar una función  $f$ , los valores de interpolación se definen como sigue:

$$y_{y_0} = f(x_{y_0}), \forall y_0 = 0, \dots, n.$$

Sabemos que se puede analizar e implementar el polinomio de interpolación de Lagrange de grado  $n$ . El cual se define de la siguiente forma,

$$P_n(x) = \sum_{k=0}^n l_k(x) f(x_k)$$

Donde los  $l_k$ 's, son polinomios de grado  $n$  formando una base de  $P_n$ .

$$l_k(x) = \prod_{y=0, y \neq k}^n \frac{x - x_{y0}}{x_k - x_{y0}} = \frac{x - x_0}{x_k - x_0} \cdots \frac{x - x_{k-1}}{x_k - x_{k-1}} \frac{x - x_{k+1}}{x_k - x_{k+1}} \cdots \frac{x - x_n}{x_k - x_n}$$

Tales polinomios no son convenientes, ya que numéricamente, es difícil de deducir  $l_{k+1}$  de  $l_k$ . Por esta razón, es que se opta por trabajar con el **polinomio de interpolación de Newton**.

Los polinomios de la base de Newton,  $e_k$ , se definen por,

$$e_k(x) = \prod_{y=0}^{k-1} (x - x_0) = (x - x_0)(x - x_1) \cdots (x - x_{k-1}), k = 1, \dots, n.$$

Con la siguiente convención,

$$e_0 = 1$$

Además,

$$e_1 = (x - x_0)$$

$$e_2 = (x - x_0)(x - x_1)$$

$$e_3 = (x - x_0)(x - x_1)(x - x_2)$$

⋮

$$e_n = (x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

Es importante destacar los siguientes aspectos:

- El conjunto de polinomios  $(e_k)_{0 \leq k \leq n}$  (*La base de Newton*) son una base de  $P_n$ , el espacio de polinomios de grado como máximo igual a  $n$ . De hecho, constituyen un conjunto de grados escalones de  $(n + 1)$  polinomios.
- Un polinomio de interpolación de grado  $n$  en la forma de Newton relacionado con la subdivisión  $\{(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)\}$  se define como,

$$P_n(x) = \sum_{k=0}^n \alpha_k e_k(x)$$

$$= \alpha_0 + \alpha_1(x - x_0) + \alpha_2(x - x_0)(x - x_1) + \dots + \alpha_n(x - x_0) \dots (x - x_{n-1}).$$

Donde,

$$P_n(x_{y_0}) = f(x_{y_0}), \forall y_0 = 0, \dots, n.$$

Para determinar los coeficientes  $(\alpha_k)_{0 \leq k \leq n}$  utilizamos el procedimiento de ***Diferencias Divididas***.

Consideremos un polinomio de interpolación de grado  $n$  en la forma de Newton  $P_n(x)$ . Si este es evaluado en  $x_0$ , obtenemos como resultado lo siguiente,

$$P_n(x_0) = \sum_{k=0}^n \alpha_k e_k(x_0) = \alpha_0 = f(x_0) = f[x_0]$$

En términos generales, escribimos,

$$f[x_{y_0}] = f(x_{y_0}), \forall y_0 = 0, \dots, n.$$

$f[x_0]$  se denomina *Diferencia Dividida de orden cero*.

Ahora, si tomamos el polinomio de interpolación de grado  $n$  en la forma de Newton y lo evaluamos en  $x_1$ , obtendríamos como resultado,

$$P_n(x_1) = \sum_{k=0}^n \alpha_k e_k(x_1)$$

$$\begin{aligned}
&= \alpha_0 + \alpha_1(x_1 - x_0) \\
&= f[x_0] + \alpha_1(x_1 - x_0) \\
&= f[x_1].
\end{aligned}$$

Por lo tanto,

$$\alpha_1 = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1]$$

$f[x_0, x_1]$  tiene por nombre *Primera diferencia dividida en orden*.

Realicemos el procedimiento una vez más evaluando el polinomio de interpolación en  $x_2$ ,

$$\begin{aligned}
P_n(x_2) &= \sum_{k=0}^n \alpha_k e_k(x_2) \\
&= \alpha_0 + \alpha_1(x_2 - x_0) + \alpha_2(x_2 - x_0)(x_2 - x_1) \\
&= f[x_0] + f[x_0, x_1](x_2 - x_0) + \alpha_2(x_2 - x_0)(x_2 - x_1) \\
&= f[x_2].
\end{aligned}$$

Obteniendo así,

$$\begin{aligned}
\alpha_2(x_2 - x_0)(x_2 - x_1) &= f[x_2] - f[x_0] - f[x_0, x_1](x_2 - x_0) \\
\alpha_2(x_2 - x_0)(x_2 - x_1) &= f[x_2] - f[x_0] - f[x_0, x_1](x_2 - x_0) - f[x_1] + f[x_1] \\
\alpha_2(x_2 - x_0)(x_2 - x_1) &= f[x_2] - f[x_1] + f[x_1] - f[x_0] - f[x_0, x_1](x_2 - x_0) \\
\alpha_2(x_2 - x_0)(x_2 - x_1) &= f[x_2] - f[x_1] + (x_1 - x_0)f[x_0, x_1] - f[x_0, x_1](x_2 - x_0) \\
\alpha_2(x_2 - x_0)(x_2 - x_1) &= f[x_2] - f[x_1] + (x_1 - x_2)f[x_0, x_1] \\
\alpha_2(x_2 - x_0) &= \frac{f[x_2] - f[x_1]}{x_2 - x_1} - f[x_0, x_1]
\end{aligned}$$

$$\alpha_2(x_2 - x_0) = f[x_1, x_2] - f[x_0, x_1].$$

Por lo tanto,

$$\alpha_2 = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = f[x_0, x_1, x_2].$$

$f[x_0, x_1, x_2]$  tiene por nombre *Segunda Diferencia Dividida en orden*.

Por recurrencia, obtenemos,

$$\alpha_k = \frac{f[x_1, \dots, x_k] - f[x_0, \dots, x_{k-1}]}{x_k - x_0} = f[x_0, \dots, x_k].$$

$f[x_0, \dots, x_k]$  por lo tanto, se denomina *k-ésima Diferencia Dividida en orden*.

Así, el Polinomio de interpolación de grado  $n$  en la forma de Newton se obtiene a través de las sucesivas diferencias divididas,


$$P_n(x) = f[x_0] + \sum_{k=1}^n f[x_0, \dots, x_k] e_k(x).$$

### Implementación (Matlab/Octave):

El código fuente (*Newton\_Interpolation.m*) se dividió en tres funciones, cada una de las cuales cumple con una tarea en específica que se indicará a continuación:

- *newton\_interpolation*: En ella se establece la definición de los datos, como lo son la función a trabajar, la cantidad de puntos y el intervalo a considerar para realizar el estudio.
- *interpolating\_polynomial*: Se desarrolla a nivel de código, todo lo explicado en el resumen teórico presentado anteriormente en esta sección del informe.
- *plot\_newton\_interpolation*: Por último, se realiza la gráfica de los resultados obtenidos luego de aplicar el procedimiento de interpolación de Newton.

A continuación, se presentan imágenes correspondientes a cada una de las funciones mencionadas y desarrolladas.



```
function newton_interpolation
    % newton_interpolation computes a interpolating
    % polynomial of the function
    %
    %       $f(x) = \log(x^1)$ 
    %
    % based on  $n$  linearly spaced
    % points in the interval  $[2, 10]$ 

    n = 10;
    X = linspace(2, 10, n);
    Y = log(X.^1);

    DD(1:n,1) = X;
    DD(1:n,2) = Y;

    P = interpolating_polynomial(n,X,Y,DD)
    plot_newton_interpolation(X,Y,P)
endfunction
```

```

function P = interpolating_polynomial(n,X,Y,DD)
    for j = 1:n-1
        for k = 1:n-j
            DD(k,j+2) = (DD(k+1,j+1)-DD(k,j+1))/(DD(k+j,1)-DD(k,1));
        endfor
    endfor

    Prod = zeros(n-1,n);
    p = conv(1,[1,-X(1)]); % poly (x-x(1))

    for i = 1:n-1
        Prod(i,(n-(length(p)-1)):n) = DD(1,i+2)*p;
        p = conv(p,[1,-X(i+1)]); % poly (x-x(1))*(x-x(2))....
    endfor

    P = sum(Prod,1);
    P(1,n) = P(1,n)+Y(1);
endfunction

```

```

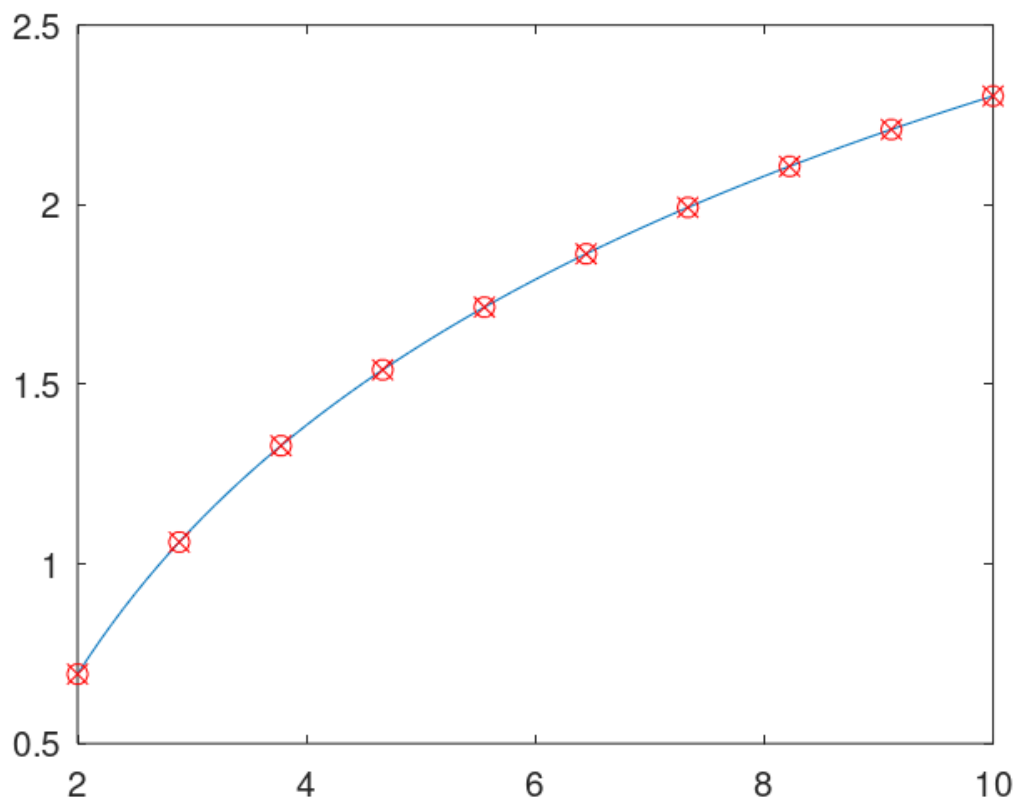
function plot_newton_interpolation(X,Y,P)
    % plot a interpolating polynomial

    x_eval = min(X):0.1:max(X);
    y_eval = polyval(P,x_eval);
    plot(x_eval,y_eval);
    hold on;
    plot(X,Y,"xr");
    hold on;
    plot(X,log(X),"or")
endfunction

```

### Resultado Gráfico:

Para finalizar, a continuación, se visualiza la gráfica obtenida de la función de ejemplo indicada en el código fuente facilitado, donde observamos en color azul la función establecida y, además, los puntos por el cual el polinomio interpolante realiza su trayectoria pasando por los diferentes puntos establecidos (color rojo en forma de X).



### b) Spline Cúbico Natural Interpolante

#### Resumen Teórico:

El *spline cúbico* es una función de interpolación muy popular. Para  $(n + 1)$  puntos de datos  $(x_0, y_0), \dots, (x_n, y_n)$  que obedecen a la condición  $a = x_0 < x_1 < \dots < x_n = b$ , el spline cúbico tiene las siguientes propiedades para  $k = 0, \dots, n - 1$ :



1.  $S_k(x) = s_{k,0} + s_{k,1}(x - x_k) + s_{k,2}(x - x_k)^2 + s_{k,3}(x - x_k)^3$ . El spline cúbico se construye con  $n$  polinomios cúbicos de esta forma.
2.  $S_k(x_k) = y_k$ . El spline cúbico pasa por cada punto de datos.
3.  $S_k(x_{k+1}) = S_{k+1}(x_{k+1})$ . El spline cúbico es continuo en los puntos de datos y, por tanto, es continuo en todo el intervalo de interpolación.
4.  $S'_k(x_{k+1}) = S'_{k+1}(x_{k+1})$ . El spline cúbico tiene una *primera* derivada continua.
5.  $S''_k(x_{k+1}) = S''_{k+1}(x_{k+1})$ . El spline cúbico tiene una *segunda* derivada continua.

Para construir un spline cúbico a partir de un conjunto dado de puntos de datos debemos resolver los coeficientes de los polinomios cúbicos  $\{S_k\}_{k=0}^{n-1}$ . Hay cuatro coeficientes en un polinomio cúbico, por lo que tenemos  $4n$  incógnitas. Los puntos de datos proporcionan  $(n + 1)$  restricciones. Las propiedades 3, 4 y 5 proporcionan cada una  $(n - 1)$  restricciones adicionales. Por lo tanto,  $(n + 1) + 3(n - 1) = 4n - 2$  restricciones se especifican con  $4n$  incógnitas. Para resolver los coeficientes necesitaremos especificar dos restricciones más. Lo hacemos para  $S''$  o  $S'$  en los extremos del intervalo de interpolación.

Con  $m_k = S''(x_k)$ ,  $h_k = x_{k+1} - x_k$ , y  $d_k = \frac{y_{k+1} - y_k}{h_k}$ , hay tres condiciones límite comunes: *natural*, *especificada* y *extrapolada*. En un *spline cúbico natural* la segunda derivada es cero en los puntos finales,  $m_0 = m_n = 0$ . En las condiciones de *contorno especificadas*  $m_0$  y  $m_n$  están de alguna manera especificados o proporcionados. En el *spline cúbico extrapolado*, extrapolamos la segunda derivada en los puntos finales utilizando puntos de datos cercanos,

$$m_0 = m_1 - \frac{h_0(m_2 - m_1)}{h_1}$$

$$m_n = m_{n-1} + \frac{h_{n-1}(m_{n-1} - m_{n-2})}{h_{n-2}}.$$

Independientemente de la elección de las condiciones de contorno, se mantiene la siguiente relación,

$$h_{k-1}m_{k-1} + 2(h_{k-1} + h_k)m_k + h_k m_{k+1} = 6(d_k - d_{k-1}).$$

Para  $k = 1, 2, 3, \dots, n$ . Los  $h_k$  y  $d_k$  se conocen a partir de los puntos de datos, por lo que todo lo que queda es resolver para  $\{m_k\}$ . Una vez que se determinan los  $\{m_k\}$ , los coeficientes para  $S_k$  vienen dados por,

$$S_{k,0} = y_k$$

$$S_{k,1} = d_k = \frac{h_k(2m_k + m_{k+1})}{6}$$

$$S_{k,2} = \frac{m_k}{2}$$

$$S_{k,3} = \frac{m_{k+1} - m_k}{6h_k}.$$

### Implementación (Matlab/Octave):

De igual forma al código fuente facilitado en el bloque anterior, en este (*Cubi\_Spline.m*) se han desarrollado tres funciones las cuales en cada una se realiza una tarea diferente para así modular y facilitar la comprensión del procedimiento implementado:

- *cubic\_driver*: Se realiza la definición de los datos relevante para desarrollar el spline cubico natural; definición de función, cantidad de puntos a trabajar e intervalo de interés a estudiar. Además de la invocación con los parámetros necesarios de las siguientes funciones a mencionar.
- *cubic\_spline*: Enfocamos esta función en el cálculo de los coeficientes  $S_0, S_1, S_2$  y  $S_3$  de los diferentes  $\{m_k\}$  establecidos, haciendo uso de las

igualdades indicadas al final del resumen teórico explicado en la sección anterior.

- *plot\_cubic\_spline*: Por último, se despliega la gráfica correspondiente según los datos calculados para visualizar el spline cúbico natural que se trabajó.

```
function cubic_driver
    % cubic_driver(n) computes a cubic spline
    % interpolant of the runge function
    %
    %       $f(x) = 1/(1+25*x^2)$ 
    %
    % based on n linearly spaced
    % points in the interval [-1,1]

    runge = @(x) 1./(1+ 25*x.^2);
    num_points = 5;
    x = linspace(-1,1,num_points);
    y = runge(x);
    [s0,s1,s2,s3] = cubic_spline(x',y');
    plot_points = 1000;
    xx = linspace(-1,1,plot_points);
    yy = runge(xx);
    plot(xx,yy,'g');
    hold on;
    plot_cubic_spline(x,s0,s1,s2,s3);
endfunction
```

```

function [s0,s1,s2,s3]=cubic_spline(x,y)
    % [s0,s1,s2,s3]=cubic_spline(x,y)
    %
    % computes the coefficients of a cubic spline
    % interpolant through the data points (x,y)
    %
    % The spline is defined as the piecewise cubic
    % polynomial
    %
    %  $S(x) = \begin{cases} S_k(x) & x(k) \leq x \leq x(k+1) \\ 0 & \text{otherwise} \end{cases}$ 
    %
    % The cubic polynomial  $S_k(x)$  is given by
    %
    %  $S_k(x) = sk0 + sk1*(x-x(k)) + sk2*(x-x(k))^2 + sk3*(x-x(k))^3$ 
    %
    % The coefficients  $sk0$ ,  $sk1$ ,  $sk2$ , and  $sk3$  for each of the
    % polynomials are returned in the vectors  $s0$ ,  $s1$ ,  $s2$ , and  $s3$ 
    % respectively.

    if any(size(x) ~= size(y)) || size(x,2) ~= 1
        error('inputs x and y must be column vectors of equal length');
    end
    n = length(x);
    h = x(2:n) - x(1:n-1);
    d = (y(2:n) - y(1:n-1))./h;
    lower = h(1:end-1);
    main = 2*(h(1:end-1) + h(2:end));
    upper = h(2:end);
    T = spdiags([lower main upper], [-1 0 1], n-2, n-2);
    rhs = 6*(d(2:end)-d(1:end-1));
    m = T\rhs;
    % Use natural boundary conditions where second derivative
    % is zero at the endpoints
    m = [ 0; m; 0];
    s0 = y;
    s1 = d - h.*(2*m(1:end-1) + m(2:end))/6;
    s2 = m/2;
    s3 =(m(2:end)-m(1:end-1))./(6*h);
endfunction

```

```

function plot_cubic_spline(x,s0,s1,s2,s3)
    % plot_cubic_spline(x,s0,s1,s2,s3)
    %
    % plots a cubic spline with break points x
    % and coefficients s0, s1, s2, s3

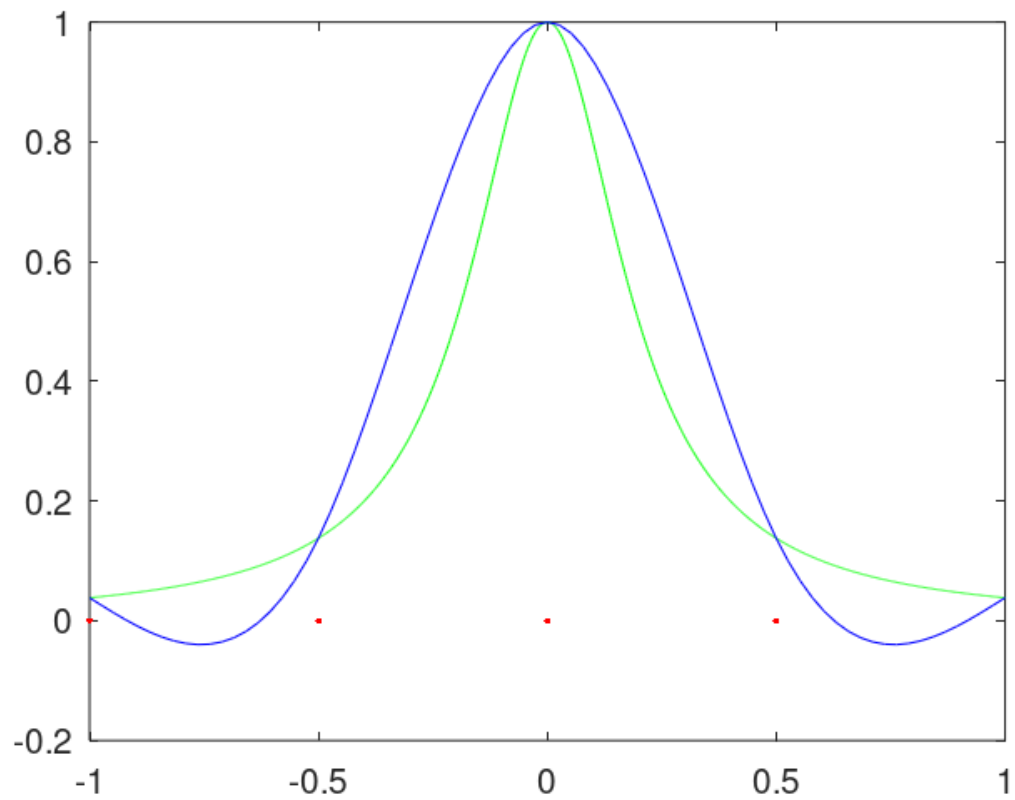
    n = length(x);
    inner_points = 20;
    for i=1:n-1
        xx = linspace(x(i),x(i+1),inner_points);
        xi = repmat(x(i),1,inner_points);
        yy = s0(i) + s1(i)*(xx-xi) + ...
            s2(i)*(xx-xi).^2 + s3(i)*(xx - xi).^3;
        plot(xx,yy,'b')
        plot(x(i),0,'r');
    end
endfunction

```

### Resultado Gráfico:

Algo importante a destacar es que para el ajuste de curvas, los splines se utilizan para aproximar formas complicadas. La simplicidad de la representación y la facilidad de cómputo de los splines los hacen populares para la representación de curvas en informática, particularmente en el terreno de los gráficos por ordenador. Dicho esto, a continuación, se presente el gráfico a obtener a partir del ejemplo indicado en el código fuente dentro de la función *cubic\_driver*.

De color verde tenemos la representación de la función original, mientras que de color azul obtenemos el resultado calculado del spline cúbico natural procesado en un intervalo de -1 a 1, junto con un total de  $n = 5$  puntos linealmente espaciados.



### Referencias:

- 1) Mathews, John H., 1943 – Numerical methods for mathematics, science and engineering.
- 2) De Boor, Carl. A practical guide to splines.