Informe Laboratorio: Análisis Numérico
Práctica No. 6

Daniel Delgado
Código: 2182066
Grupo: B2
Escuela de Ingeniería de Sistemas e Informática
Universidad Industrial de Santander

4 de febrero de 2021

1. Introducción

La interpolación, específicamente la interpolación polinómica, es una práctica la cual permite el encontrar más una función a partir de un conjunto de puntos dados. Esto puede ser por seguimiento de la tendencia de una función o la generación de un polinómio que se ajuste a un set de puntos dados.

En el presente informe, tratará las interpolaciones polinómicas de Newton y LaGrange además el como estas operan y se diferencian entre ellas. De la misma manera, se buscará entender el como cada una de estas se comporta en un contexto computacional y las ventajas respectivas.

La compresión de las diferentes maneras de realizar interpolaciones, al igual que el desarrollo de la algoritmia relacionada, son los principales temas a a tratar durante el desarrollo del presente informe, así como la resolución de los problemas propuestos a manera de pregunta orientadora del componente práctico del mismo.

2. Desarrollo

1. Preguntas propuestas

   De manera inicial, la guía de trabajo plantea algunas preguntas base con el fin de establecer los conceptos básicos que serán trabajados a lo largo del desarrollo del presente informe.

   a) ¿Qué es la interpolación?

   En términos simples, la interpolación se define como la obtención de puntos de una función indeterminada a partir de algunos puntos conocidos. Esto se hace comúnmente para la búsqueda de tendencias dadas por algunos datos o con el fin de realizar una aproximación de alguna función hacia algo más simple.

   b) ¿Cómo se calcula un polinomio del Taylor de grado N?

   Los polinomios de Taylor de grado N, aunque existen varias maneras de calcularlos, dentro de un dominio determinado $[a, b]$, es posible calcularlos a partir de la siguiente sumatoria la cual es usada para aproximar una función:

$$f(x) \approx P(x) = \sum_{k=0}^{N} \frac{f^{(k)}(x)}{k!}(x - x_0)^k$$

   Es gracias a esto que podemos realizar aproximaciones polinómicas de algunas funciones más complejas como lo vienen siendo las funciones sinusoidales o exponenciales.

c) ¿Cómo se calculan las interpolaciones de Newton y LaGrange?

Estas interpolaciones son calculadas de dos maneras bastante diferentes diferentes pero toma datos similares para realizar este proceso.

La interpolación de LaGrange se realiza en dos partes, la primera, es el cálculo de los productos de los puntos que serán usados para realizar la generación del polinómio:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Seguidamente, empleando los valores obtenidos para L_i, se calculará el polinómio resultante de la interpolación a partir de la siguiente sumatoria:

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

En cuanto a la interpolación de Newton, también conocida como diferencias divididas, es calculada a partir del siguiente proceso recursivo:

$$f_i(x_0, x_1, x_2, ..., x_{i-1}, x_i) = \frac{f_{i-1}(x_0, x_1, ..., x_{i-1}, x_i) - f_i(x_0, x_1, x_2, ..., x_{i-1})}{x_i - x_0}$$

d) ¿Qué aplicaciones tiene la interpolación?

La interpolación tiene múltiples aplicaciones. Estas aplicaciones están principalmente relacionadas con la búsqueda de una función aproximada a partir de un polinómio de grado n. Estas aproximaciones pueden ser usadas principalmente para la simplificación de cálculos y la reducción de tiempos de computación para funciones donde la precisión no debe ser extremadamente alta.

2. Aplicando

a) Para encontrar el polinomio cuadrático de LaGrange $P_2(x)$ para la función $y = f(x) = \sqrt{x}$, debemos realizar siguiente sumatoria:

$$P_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

Con el fin de realizar esto, tenemos que encontrar los valores para $L_i(x)$ los cuales son obtenidos a partir del siguiente producto:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Partiendo de los valores para x_i dados en el enunciado, podemos realizar el cálculo de $L_0(x)$, $L_1(x)$, $L_2(x)$:

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = \frac{(x - 1.25)(x - 1.5)}{(1 - 1.25)(1 - 1.5)} = 8(x - 1.25)(x - 1.5)$$
$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = \frac{(x - 1)(x - 1.5)}{(1.25 - 1)(1.25 - 1.5)} = -16(x - 1)(x - 1.5)$$
$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_0)} = \frac{(x - 1)(x - 1.25)}{(1.5 - 1)(1.5 - 1.25)} = 8(x - 1)(x - 1.25)$$

Teniendo ya los valores para $L_i(x)$, podemos expandir la sumatoria planteada inicialmente:

$$P_2(x) = \sum_{i=0}^{2} L_i(x) f(x_i) = L_0(x) f(x_0) + L_1(x) f(x_1) + L_2(x) f(x_2)$$
$$P_2(x) = 8(x - 1.25)(x - 1.5) \cdot f(1) + [-16(x - 1)(x - 1.5)] \cdot f(1.25) + 8(x - 1)(x - 1.25) \cdot f(1.5)$$

Y evaluando los valores de la función en los diferentes valores de x dados:

$$P_2(x) = 8(x-1.25)(x-1.5)\cdot 1 + [-16(x-1)(x-1.5)]\cdot 1.1180339 + 8(x-1)(x-1.25)\cdot 1.2247448$$

Y simplificando, dando nuestro resultado final para el polinomio de LaGrange:

$$P_2(x) = 8(x-1.25)(x-1.5) - 17.8885438(x-1)(x-1.5) + 9.7979589(x-1)(x-1.25)$$

b) Para encontrar los valores indeterminados de la tabla dada, será necesario, a partir de los valores que tenemos en la tabla, usar la regla recursiva de las diferencias divididas. Esta está definida de la siguiente manera:

$$f[x_{k-j}, x_{k-j+1}, ..., x_k] = \frac{f[x_{k-j+1}, ..., x_k] - f[x_{k-j}, ..., x_{k-1}]}{x_k - x_{k-j}}$$

En este sentido, para poder encontrar los valores indeterminados en la tabla, vamos a requerir las siguientes "versiones"de las diferencias divididas:

$$k_0 \to f[x_0] = f(x_0)$$
$$k_1 \to f[x_1, x_0] = \frac{f[x_1] - f[x_0]}{x_1 - x_0}$$
$$k_2 \to f[x_2, x_1, x_0] = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$$
$$k_3 \to f[x_3, x_2, x_1, x_0] = \frac{f[x_3, x_2, x_1] - f[x_2, x_1, x_0]}{x_3 - x_0}$$

Seguidamente, vamos a darle nombres a cada uno de los valores indeterminados con el fin de ubicarnos mejor:

x_k	$f[x_k]$	$f[,]$	$f[,,]$	$f[,,,]$
$x_0 = 1.0$	3.5	-	-	-
$x_1 = 1.5$	x_a	x_b	-	-
$x_2 = 3.5$	103	45.5	11.4	-
$x_3 = 5.0$	491.5	259	61	x_c

El primer valor que podemos calcular es x_c debido a que conocemos los dos valores necesarios para el cálculo de este:

$$f[x_3, x_2, x_1, x_0] = \frac{f[x_3, x_2, x_1] - f[x_2, x_1, x_0]}{x_3 - x_0}$$
$$f[5, 3.5, 1.5, 1] = \frac{f[5, 3.5, 1.5] - f[3.5, 1.5, 1]}{5 - 1}$$
$$f[5, 3.5, 1.5, 1] = \frac{61 - 11.4}{5 - 1}$$
$$f[5, 3.5, 1.5, 1] = \frac{49.6}{4} = \frac{62}{5} = 12.4 = x_c$$

Para calcular los valores de x_a y x_b, tendremos, es posible desperjarlos de las siguientes ecuaciones:

$$11.4 = f[3.5, 1.5, 1] = \frac{f[3.5, 1.5] - f[1.5, 1]}{3.5 - 1}$$
$$11.4 = \frac{f[3.5, 1.5] - f[1.5, 1]}{2.5}$$

$$f[1.5, 1] = \frac{f[1.5] - f[1]}{1.5 - 1} \qquad f[3.5, 1.5] = \frac{f[3.5] - f[1.5]}{3.5 - 1.5}$$
$$x_b = 2(x_a - 3.5) \qquad 45.5 = \frac{103 - x_a}{2}$$

$$45.5 = \frac{103 - x_a}{2}$$
$$91 = 103 - x_a$$
$$x_a = 103 - 91 = 12$$
$$x_b = 2(x_a - 3.5)$$
$$x_b = 2((12) - 3.5) = 2(8.5) = 17$$

Ya con los valores despejados, podemos remplazar en la tabla para completarla:

x_k	$f[x_k]$	$f[,]$	$f[,,]$	$f[,,,]$
$x_0 = 1.0$	3.5	-	-	-
$x_1 = 1.5$	12	17	-	-
$x_2 = 3.5$	103	45.5	11.4	-
$x_3 = 5.0$	491.5	259	61	12.4

3. Implementando

a) De manera inicial, se plantea la necesidad de tener un algoritmo el cual realice el cálculo de un polinómio de LaGrange a partir de una cantidad n de puntos dada. Con el fin de dar solución a esto, se desarrolló la función `lagrangePoly(pointMatrix)` la cual permite calcular el polinomio de LaGrange correspondiente a unos puntos dados.

```
function output = lagrangePoly(pointMatrix)
    syms x
    [vS, hS] = size(pointMatrix);
    output = NaN;
    if(hS == 2 && vS > 1) %%Las entradas son correctas cuando los puntos son de (x,y)
        L = [];
        for index = 1:vS
            tempFun = 1;
            for pain = 1:vS
                if(pain == index)
                    %% Skip!
                else
                    tempFun = tempFun * ((x-pointMatrix(pain,1))/(pointMatrix(index,1) -
                        pointMatrix(pain,1)));
                end
            end
            L = [L tempFun];
        end
        fullFunc = 0;
        for finex = 1:vS
            fullFunc = fullFunc + pointMatrix(finex,2) * L(finex);
        end
        output = fullFunc;
    end
end
```

Este proceso se realiza en 3 partes. La primera es un verificación básica de la forma de la matriz de entrada, de tamaño $n \times 2$, donde n es mayor a 1. La segunda la cual se basa en el cálculo de los valores de $L_i(x)$. Esto se realiza de una manera, aunque poco eficiente por la necesidad de 2 ciclos iterativos, simple de entender.

Inicialmente se define una función anónima temporal `tempfun = @(x) 1`, esta se usará para poder guardar los valores de $L_i(x)$. Seguidamente, dentro del segundo ciclo iterativo, se realizará el cálculo de todos los valores que compondrán L_i y se evitarán los valores donde daría un valor dividido entre 0.

```
L = [];
for index = 1:vS
    tempFun = 1;
    for pain = 1:vS
        if(pain == index)
            %% Skip!
        else
            tempFun = tempFun * ((x-pointMatrix(pain,1))/(pointMatrix(index,1) - pointMatrix(pain,1)));
        end
    end
    L = [L tempFun];
end
```

Tras calcular todos los valores respectos de $L_i(x)$, en la tercera parte, se construirá la función final en el tercer y último ciclo iterativo en la cual se multiplicarán los valores calculados de $L_i(x)$ con los valores de $f(x)$ correspondientes.

```
fullFunc = 0;
for finex = 1:vS
    fullFunc = fullFunc + pointMatrix(finex,2) * L(finex);
end
output = fullFunc;
```

b) De igual manera, se pide la realización de un algoritmo el cual pueda realizar el cálculo de un polinomio de Newton a partir de un mismo set de n puntos dados. De la misma manera que con el problema anterior, se desarrolló la función `newtonPoly(pointMatrix)` la cual realiza el cálculo de un polinomio de Newton.

```
function output = newtonPoly(pointMatrix)
    syms x
    [vS, hS] = size(pointMatrix);
    output = NaN;
    if(hS == 2) %%Las entradas son correctas cuando los puntos son de (x,y)
        funTrix = zeros(vS,vS);
        for starex = 1:vS
            funTrix(starex,1) = pointMatrix(starex,2);
        end
        for index = 2:vS
            for pain = index:vS
                funTrix(pain,index) = ((funTrix(pain,index-1)-funTrix(pain-1,index-1))/
                    (pointMatrix(pain,1)-pointMatrix((pain - index + 1), 1)));
            end
        end
        fullFunc = funTrix(1,1);
        fullFuncAux = 1;
        for fintrex = 2:vS
            fullFuncAux = fullFuncAux * (x-pointMatrix(fintrex-1,1));
            fullFunc = fullFunc + funTrix(fintrex, fintrex) * fullFuncAux;
        end
        output = fullFunc;
    end
end
```

Este proceso se realiza en 4 ciclos iterativos los cuales pueden repartirse en 3 partes. La primera parte del proceso crea una matriz del tamaño de la cantidad de puntos dados como entrada, seguidamente llena la primera columna con los valores y respectivos para cada uno de los puntos. Esto se hace con el fin de calcular cada una de las diferencias divididas necesarias para el polinómio.

```
funTrix = zeros(vS,vS);
for starex = 1:vS
    funTrix(starex,1) = pointMatrix(starex,2);
end
```

En la segunda parte, se realizará como tal el cálculo de los valores de las diferencias divididas siguiendo la regla de iteración dada para los polinomios de Newton:

```
for index = 2:vS
    for pain = index:vS
        funTrix(pain,index) = ((funTrix(pain,index-1)-funTrix(pain-1,index-1))/
            (pointMatrix(pain,1)-pointMatrix((pain - index + 1), 1)));
    end
end
```

Finalmente, en la tercera etapa, se construirá la función final de tal manera en la que la función tenga como salida una función simbólica la cual tendrá la forma de un polinomio de Newton.

```
fullFunc = funTrix(1,1);
fullFuncAux = 1;
for fintrex = 2:vS
    fullFuncAux = fullFuncAux * (x-pointMatrix(fintrex-1,1));
    fullFunc = fullFunc + funTrix(fintrex, fintrex) * fullFuncAux;
end
output = fullFunc;
```

c) A continuación, aplicando las funciones desarrolladas en los puntos anteriores, se nos pide desarrollar un polinomio de LaGrange y de Newton basándose en la función $f(x) = 3\sin(\pi x/6)$ para los puntos $x_0 = 0$, $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, and $x_4 = 4$.

Con el fin de realizar esto, se desarrollaron 2 scripts para la ejecución de cada una de estas funciones. En el caso del polinómio de LaGrange, se empleó `lagrangeRunner.m`. Este script, en términos simples, definirá la función $f(x)$ al igual que los puntos a usar para el cálculo del polinomio y ejecutará `lagrangePoly(pointMatrix)` con los respectivos valores.

```
f = @(x) 3 * sin(pi*x/6);
pointMatrix = [0 f(0); 1 f(1); 2 f(2); 3 f(3); 4 f(4)];

lagrangeP = lagrangePoly(pointMatrix)
```

Tras la ejecución en la consola, el script da como resultado la siguiente función. Como es posible observar, es posible ver la forma de polinómio de LaGrange en el resultado.

```
lagrangeP =

(3*3^(1/2)*x*(x/2 - 2)*(x - 1)*(x - 3))/4 - x*(x/2 - 1/2)*(x - 2)*(x - 4) - (3*x*(x/2 - 3/2)*(x/3 -
    4/3)*(x - 2))/2 + (3*3^(1/2)*x*(x/2 - 1)*(x/3 - 1/3)*(x - 3))/8
```

En cuanto al polinomio de Newton, se desarrolló básicamente lo mismo pero para el script `newtonRunner.m` el cual cumple las mismas características del anterior.

```
f = @(x) 3 * sin(pi*x/6);
pointMatrix = [0 f(0); 1 f(1); 2 f(2); 3 f(3); 4 f(4)];

newtonP = newtonPoly(pointMatrix)
```

De igual manera, tras la ejecución en consola, el output de este da como resultado la siguiente función que cumple con la típica forma de un polinomio de Newton.

```
newtonP =

(3*x)/2 - (905051912390271*x*(x - 1))/4503599627370496 - (441695988904705*x*(x - 1)*(x -
    2))/9007199254740992 + (8960769158908629*x*(x - 1)*(x - 2)*(x - 3))/1152921504606846976
```

d) Finalmente, con el fin de comprobar el funcionamiento de las funciones, se pedía la graficación de cada una de las funciones comparándolas con la función original. En este sentido, nuevamente, se desarrollaron 2 scripts los cuales generarían las gráficas respectivas de cada uno de los polinomios obtenidos anteriormente.

Empezando con el de LaGrange, se desarrolló el script `lagraph.m`. Este script graficaría tanto la función calculada como la función original al igual que los puntos datos.

```
syms x

dom = linspace(-3, 7);

orig = @(x) 3 * sin(pi*x/6);

inter = matlabFunction((3*3^(1/2)*x*(x/2 - 2)*(x - 1)*(x - 3))/4 - x*(x/2 - 1/2)*(x - 2)*(x - 4) -
    (3*x*(x/2 - 3/2)*(x/3 - 4/3)*(x - 2))/2 + (3*3^(1/2)*x*(x/2 - 1)*(x/3 - 1/3)*(x - 3))/8);


plot(dom, inter(dom))
hold on
plot(dom, orig(dom))
hold on
axis([-3, 7, -4, 4])
line([0,0], ylim, 'Color', 'k', 'LineWidth', 2)
line(xlim, [0,0], 'Color', 'k', 'LineWidth', 2)
grid on

plot(0,orig(0),'*')
plot(1,orig(1),'*')
plot(2,orig(2),'*')
plot(3,orig(3),'*')
plot(4,orig(4),'*')
```

Tras la ejecución del script, se generará la figura 1 la cual muestra la cercanía entre la función original y la función interpolada.

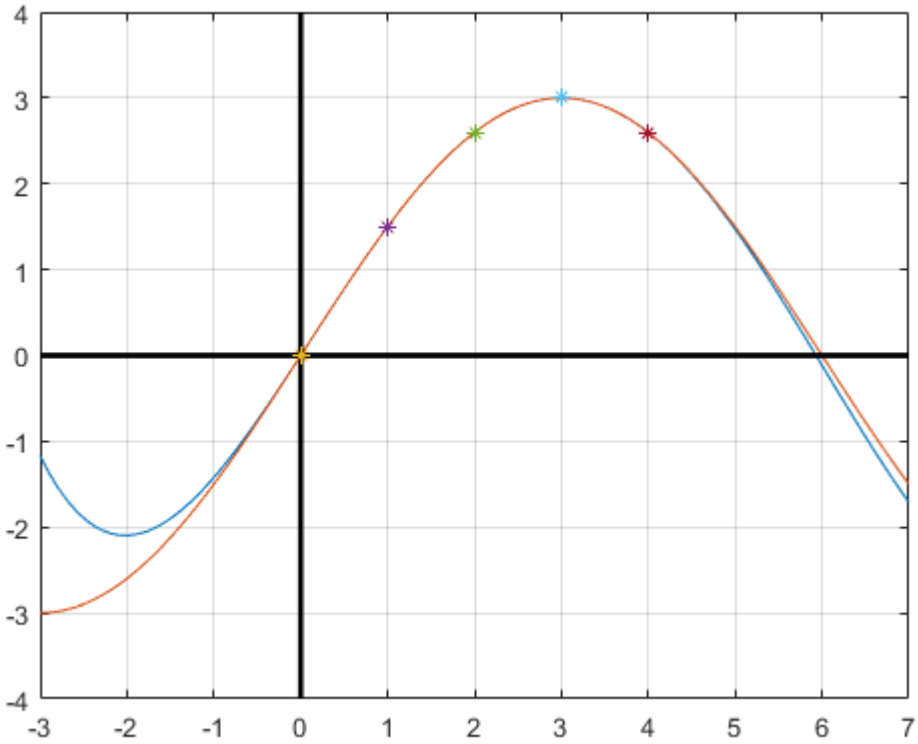

Figura 1: Gráfica del polinomio de LaGrange

Como es posible observar, ambas gráficas, especialmente entre los valores dados, existe una gran cercanía de las 2 gráficas. En este sentido, podemos afirmar que dentro de los puntos dados, la interpolación efectivamente tiene un alto nivel de precisión.

En cuanto al polinomio de Newton, se desarrolló `newtong.m` el cual, en términos simples, realiza las mismas instrucciones que `lagraph.m` pero para el resultado de `newtonPoly(pointMatrix)`.

```
syms x

dom = linspace(-3, 7);

orig = @(x) 3 * sin(pi*x/6);

inter = matlabFunction((3*x)/2 - (905051912390271*x*(x - 1))/4503599627370496 - (441695988904705*x*(x -
    1)*(x - 2))/9007199254740992 + (8960769158908629*x*(x - 1)*(x - 2)*(x - 3))/11529215046068469 76);


plot(dom, inter(dom))
hold on
plot(dom, orig(dom))
hold on
axis([-3, 7, -4, 4])
line([0,0], ylim, 'Color', 'k', 'LineWidth', 2)
line(xlim, [0,0], 'Color', 'k', 'LineWidth', 2)
grid on

```

```
plot(0,orig(0),'*')
plot(1,orig(1),'*')
plot(2,orig(2),'*')
plot(3,orig(3),'*')
plot(4,orig(4),'*')
```

Tras la ejecución del script, se genera la figura 2 en la cual, al igual que con le polinomio de LaGrange, podemos observar el como esta tiene una alta cercanía con la función original.

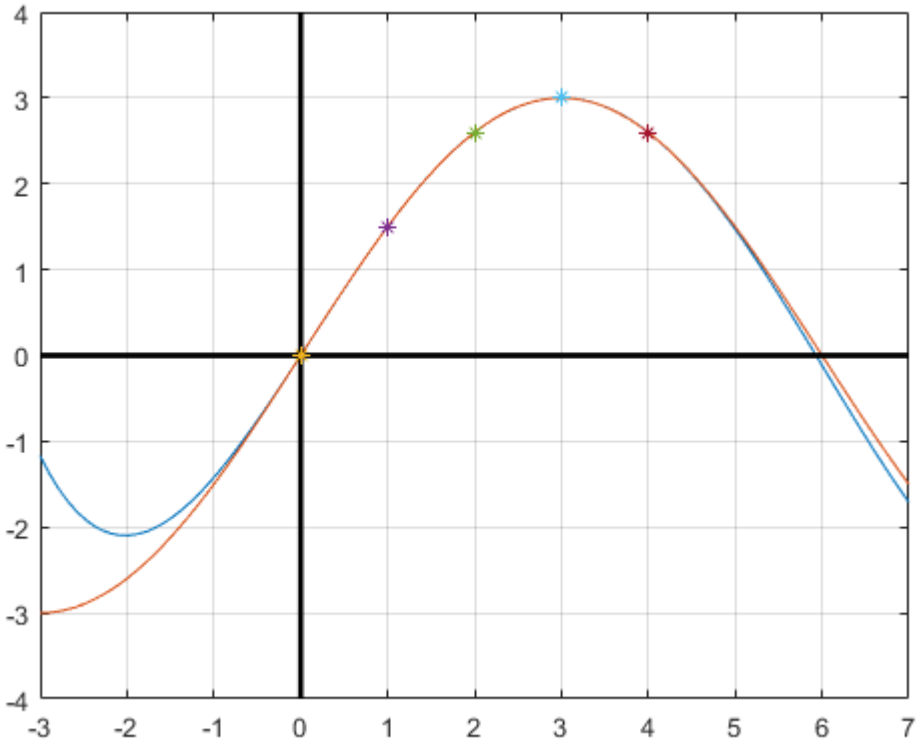

Figura 2: Gráfica del polinomio de Newton

Como es posible apreciar, y al igual que en la figura 1, la cercanía entre las 2 gráficas sugiere que las interpolaciones realizadas tienen un alto grado de precisión dentro de los puntos que fueron inicialmente dados para la construcción del polinomio de Newton.

4. Interpretando

Para finalizar, se pide encontrar el grado del polinomio de Taylor $P_N(x)$ el cual permita aproximar $\cos(33\pi/32)$ para un error menor de 10^{-6} cuando $x_o = \pi$. Para realizar esto, debemos partir de la función que nos permite calcular el error de manera inicial.

$$E_N(x) = \frac{f^{(N+1)}(c)}{(N+1)!}(x - x_0)^{(N+1)}$$

A partir de esto, podemos realizar algunos remplazos dentro de la ecuación. Lo primero es nuestra función y el valor de x, estos valores pueden ser fácilmente determiados mirando a la ecuación a aproximar. Para $\cos(33\pi/32)$, podemos ver que el valor de x es $33\pi/32$ y que nuestra función es $\cos(x)$.

$$E_N(\tfrac{33\pi}{32}) = \frac{\cos^{(N+1)}(c)}{(N+1)!}(\tfrac{\pi}{32})^{(N+1)}$$

Para N = 1 , para c se tomará el valor de π puesto es que es el máximo valor que el denominador puede tomar.

$$E_1(\tfrac{33\pi}{32}) = \frac{\cos''(\pi)}{2!}(\tfrac{\pi}{32})^{(2)}$$

$$E_1(\tfrac{33\pi}{32}) = \frac{-\cos(\frac{33\pi}{32})}{2!}(\tfrac{\pi}{32})^{(2)} = 0.00453149880662 > 10^{-6}$$

Para N = 2, c tomará el valor de $\frac{33\pi}{32}$ para así maximizar nuevamente el error.

$$E_2(\tfrac{33\pi}{32}) = \frac{\cos^{(3)}(\pi)}{(3)!}(\tfrac{\pi}{32})^{(3)}$$

$$E_2(\tfrac{33\pi}{32}) = \frac{\sin(\pi)}{(3)!}(\tfrac{\pi}{32})^{(3)} = 0.0000140947889013 > 10^{-6}$$

Para N = 3, c tomará el valor de π para así maximizar nuevamente el error.

$$E_3(\tfrac{33\pi}{32}) = \frac{\cos^{(4)}(\pi)}{(4)!}(\tfrac{\pi}{32})^{(4)}$$

$$E_3(\tfrac{33\pi}{32}) = \frac{\cos(\pi)}{(4)!}(\tfrac{\pi}{32})^{(4)} = 0.00000340593371579 > 10^{-6}$$

Para N = 4, c tomará el valor de $\frac{33\pi}{32}$ para así maximizar nuevamente el error.

$$E_4(\tfrac{33\pi}{32}) = \frac{\cos^{(5)}(\frac{33\pi}{32})}{(5)!}(\tfrac{\pi}{32})^{(5)}$$

$$E_4(\tfrac{33\pi}{32}) = \frac{-\sin(\frac{33\pi}{32})}{(5)!}(\tfrac{\pi}{32})^{(5)}) = 3.3545519141 \times 10^{-7} < 10^{-6}$$

Entonces, como podemos ver, el grado para el cual el error de la aproximación de Taylor para $\cos(\frac{33\pi}{32})$ es N = 4 en donde el error obtenido es $3.3545519141 \times 10^{-7}$.

3. Anexos

lagrangePoly.m

```
function output = lagrangePoly(pointMatrix)

    syms x

    [vS, hS] = size(pointMatrix);

    output = NaN;

    if(hS == 2 && vS > 1) %%Las entradas son correctas cuando los puntos son de (x,y)
        L = [];
        for index = 1:vS
            tempFun = 1;
            for pain = 1:vS
                if(pain == index)
                    %% Skip!
                else
                    tempFun = tempFun * ((x-pointMatrix(pain,1))/(pointMatrix(index,1) - pointMatrix(pain,1)));
                end
            end
            L = [L tempFun];
        end

        fullFunc = 0;

        for finex = 1:vS
            fullFunc = fullFunc + pointMatrix(finex,2) * L(finex);
        end

        output = fullFunc;

    end

end
```

newtonPoly.m

```
function output = newtonPoly(pointMatrix)

    syms x

    [vS, hS] = size(pointMatrix);

    output = NaN;

    if(hS == 2) %%Las entradas son correctas cuando los puntos son de (x,y)

        funTrix = zeros(vS,vS);

        for starex = 1:vS
            funTrix(starex,1) = pointMatrix(starex,2);
        end

        for index = 2:vS
            for pain = index:vS
                funTrix(pain,index) =
```

```
                    ((funTrix(pain,index-1)-funTrix(pain-1,index-1))/(pointMatrix(pain,1)-pointMatrix((pain - index
                    + 1), 1)));
            end
        end

        fullFunc = funTrix(1,1);
        fullFuncAux = 1;

        for fintrex = 2:vS
            fullFuncAux = fullFuncAux * (x-pointMatrix(fintrex-1,1));
            fullFunc = fullFunc + funTrix(fintrex, fintrex) * fullFuncAux;
        end

        output = fullFunc;
    end
end
```

lagrangeRunner.m

```
f = @(x) 3 * sin(pi*x/6);
pointMatrix = [0 f(0); 1 f(1); 2 f(2); 3 f(3); 4 f(4)];

lagrangeP = lagrangePoly(pointMatrix)
```

newtonRunner.m

```
f = @(x) 3 * sin(pi*x/6);
pointMatrix = [0 f(0); 1 f(1); 2 f(2); 3 f(3); 4 f(4)];

newtonP = newtonPoly(pointMatrix)
```

lagraph.m

```
syms x

dom = linspace(-3, 7);

orig = @(x) 3 * sin(pi*x/6);

inter = matlabFunction((3*3^(1/2)*x*(x/2 - 2)*(x - 1)*(x - 3))/4 - x*(x/2 - 1/2)*(x - 2)*(x - 4) - (3*x*(x/2 -
    3/2)*(x/3 - 4/3)*(x - 2))/2 + (3*3^(1/2)*x*(x/2 - 1)*(x/3 - 1/3)*(x - 3))/8);


plot(dom, inter(dom))
hold on
plot(dom, orig(dom))
hold on
axis([-3, 7, -4, 4])
line([0,0], ylim, 'Color', 'k', 'LineWidth', 2)
line(xlim, [0,0], 'Color', 'k', 'LineWidth', 2)
grid on

plot(0,orig(0),'*')
plot(1,orig(1),'*')
plot(2,orig(2),'*')
plot(3,orig(3),'*')
plot(4,orig(4),'*')
```

newtong.m

```
syms x

dom = linspace(-3, 7);

orig = @(x) 3 * sin(pi*x/6);

inter = matlabFunction((3*x)/2 - (905051912390271*x*(x - 1))/4503599627370496 - (441695988904705*x*(x - 1)*(x - 2))/9007199254740992 + (8960769158908629*x*(x - 1)*(x - 2)*(x - 3))/1152921504606846976);


plot(dom, inter(dom))
hold on
plot(dom, orig(dom))
hold on
axis([-3, 7, -4, 4])
line([0,0], ylim, 'Color', 'k', 'LineWidth', 2)
line(xlim, [0,0], 'Color', 'k', 'LineWidth', 2)
grid on

plot(0,orig(0),'*')
plot(1,orig(1),'*')
plot(2,orig(2),'*')
plot(3,orig(3),'*')
plot(4,orig(4),'*')
```