Tarea 9 - Métodos de interpolación
Métodos Numéricos

Rafael Alejandro García Ramírez

22 de octubre de 2023

1. Introducción

Los métodos de interpolación son ampliamente utilizados para determinar un valor que se desconoce siendo que se conocen varios diferentes dentro del sistema. Esto puede resultar bastante útil para encontrar valores que no se pueden medir o calcular, o si simplemente no se midió el dato. Otra aplicación resulta al momento de completar registos con uno o más campos vacíos en bases de datos.

Esta última aplicación nos deja en claro la problemática principal al momento de interpolar los datos: ¿bajo qué comportamiento suponemos que existirían los datos si se tiene una cantidad enorme (por no decir continua) de ellos?

Entre dos puntos (x_i, y_i) y (x_j, y_j) uno puede en principio trazar una infinidad de curvas diferentes entre ambos puntos, por lo que la decisión entre que tipo de curva trazar entre ellos determinará la exactitud entre los intervalos $[x_i, x_j]$. Imaginemos que queremos conectar (o interpolar datos entre) los puntos $(1, 1)$ y $(4, 1)$, en la figura 1 podemos ver tres ejemplos para realizar predicciones entre los datos, todos conteniendo los mismos tres puntos:

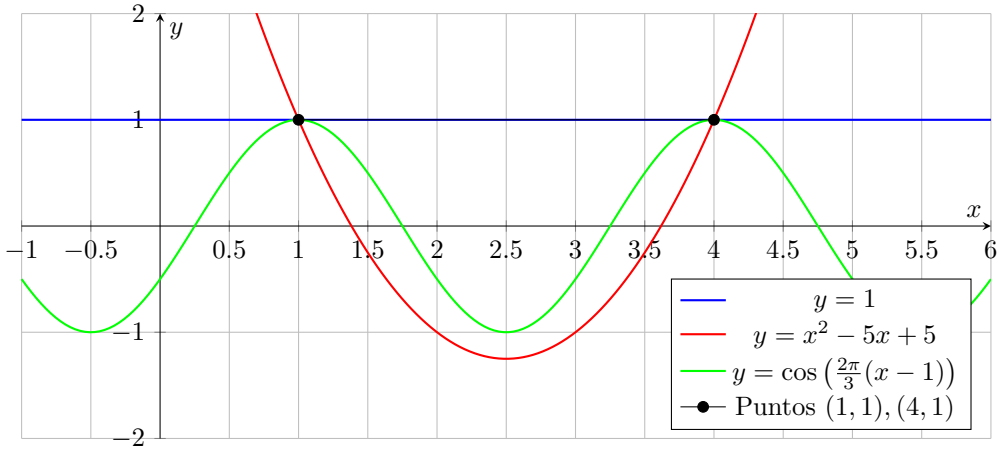


Figura 1: Intersección de tres funcines diferentes.

De este ejercicio, podemos ver que entre más datos se tengan más fácil será escoger el tipo de función con la que se debe (o se desea) describir el sistema. Entre dos puntos se carece de la información suficiente para dicha tarea. A continuación presentamos la elaboración de cuatro pseudocódigos para métodos que numéricamente encuentran un valor de interpolación tomando en cuenta un polinomio de grado n generador por $n + 1$ puntos.

2. Pseudocódigo

Los pseudocódigos que se utilizaron para realizar la implementación en `C` para los algoritmos los diferentes métodos en esta tarea son los siguientes:

2.1. Interpolador de Taylor

Algorithm 1 Interpolación de Taylor

Require: N, derivadas, x, x_0
Ensure: Valor interpolado P

function InterpolacionTaylor(N, derivadas[N], x, x_0)
    $diferencia \leftarrow x - x_0$
    $diferencia\_previa \leftarrow 1$
    $factorial \leftarrow 1$
    $P \leftarrow$ derivadas[0]
    for $i \leftarrow 1$ **hasta** $N - 1$ **do**
        $diferencia\_previa \leftarrow diferencia\_previa \cdot diferencia$
        $factorial \leftarrow factorial \cdot i$
        $P \leftarrow P +$ (derivadas[i] $\cdot\ diferencia\_previa)/factorial$
    end for
    return P
end function

2.2. Interpolador de Lagrange

Algorithm 2 Interpolador de Lagrange

Require: N, x, y, punto
Ensure: Valor interpolado P

function InterpoladorLagrange(N, x[N], y[N], punto)
    $P \leftarrow 0.0$
    for $i \leftarrow 0$ **hasta** $N - 1$ **do**
        $L \leftarrow 1$
        for $j \leftarrow 0$ **hasta** $N - 1$ **do**
            if $i \neq j$ **then**
                $L \leftarrow (L \cdot (punto - x[j]))/(x[i] - x[j])$
            end if
        end for
        $P \leftarrow P + y[i] \cdot L$
    end for
    return P
end function

2.3. Interpolador de Neville

Algorithm 3 Interpolación de Neville

Require: $N, x, y, Q, punto$
Ensure: Valor interpolado $Q[N-1][N-1]$

```
function InterpolacionNeville(N, x[N], y[N], Q[N][N], punto)

    for i ← 0 hasta N − 1 do                    ▷ Inicializar la tabla de diferencias divididas a cero
        for j ← 0 hasta N − 1 do
            Q[i][j] ← 0.0
        end for
    end for

    for i ← 0 hasta N − 1 do                    ▷ Asignar los valores iniciales a la tabla
        Q[i][0] ← y[i]
    end for

    for i ← 1 hasta N − 1 do                    ▷ Calcular los valores de la tabla
        for j ← 1 hasta i do
            Q[i][j] ← ((punto − x[i − j]) · Q[i][j − 1] − (punto − x[i]) · Q[i − 1][j − 1])/(x[i] − x[i − j])
        end for
    end for
    return Q[N − 1][N − 1]
end function
```

2.4. Interpolador de Newton

Algorithm 4 Interpolación de Newton

Require: $N, x, y, punto, tabla$
Ensure: Valor interpolado P

```
function InterpoladorNewton(N, x[N], y[N], punto, tabla[N][N])

    for i ← 0 hasta N − 1 do                    ▷ Inicializar la tabla de diferencias divididas a cero
        for j ← 0 hasta N − 1 do
            tabla[i][j] ← 0.0
        end for
    end for

    for i ← 0 hasta N − 1 do                    ▷ Asignar los valores iniciales a la tabla
        tabla[i][0] ← y[i]
    end for

    for i ← 1 hasta N − 1 do                    ▷ Calcular las diferencias divididas
        for j ← 1 hasta i do
            tabla[i][j] ← (tabla[i][j − 1] − tabla[i − 1][j − 1])/(x[i] − x[i − j])
        end for
    end for

    L, P ← 0.0
    for i ← 1 hasta N − 1 do                    ▷ Calcular el polinomio interpolador
        L ← 1
        for j ← 0 hasta N − 1 do
            if j ≤ i − 1 then
                L ← L · (punto − x[j])
            end if
        end for
        P ← P + tabla[i][i] · L
    end for
    return P + tabla[0][0]
end function
```

3. Resultados

A continuación presentamos la resolución de los problemas presentados en la tarea:

1. La tabla de valores para diferentes z y diferentes n para la función $f(z) = e^z$ en $x_0 = 0$ es la siguiente

z	Orden de interpolación	Valor de interpolación	Error absoluto
0.5	1	1.0	0.648721270700
	3	1.0	0.648721270700
	5	1.0	0.648721270700
	10	1.0	0.648721270700
1.0	1	1.0	1.718281828459
	3	2.50	0.218281828459
	5	2.50	0.218281828459
	10	2.50	0.218281828459
1.50	1	1.0	3.481689070338
	3	3.6250	0.856689070338
	5	4.39843750	0.083251570338
	10	4.39843750	0.083251570338
2.0	1	1.0	6.389056098931
	3	5.0	2.389056098931
	5	7.0	0.389056098931
	10	7.388712522046	0.000343576885

Estos datos los podemos ver en las siguientes gráficas: la figura 2 nos muestra los valores de interpolación que adquiere según el orden de interpolación y la figura 3 que muestra los el comportamiento de los errores absolutos de cada orden.

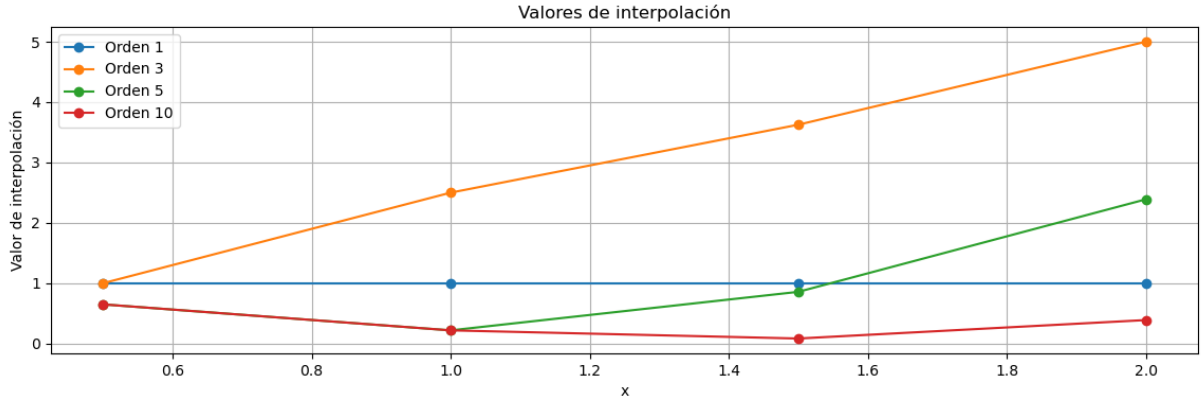


Figura 2: Gráfica de valores de interpolación según el orden.

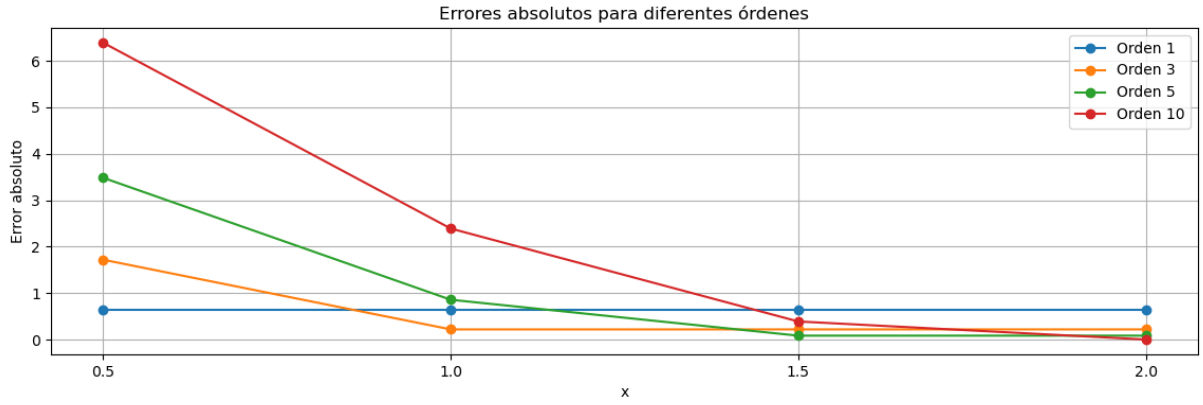


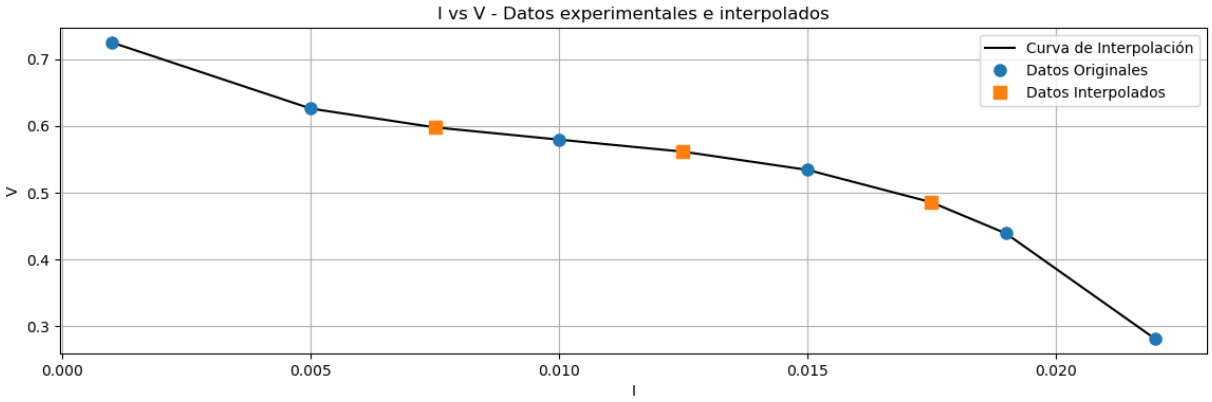
Figura 3: Gráfica de errores de interpolación según el orden.

(2,3,4). Para los incisos 2,3 y 4 se adjunta el código correspondiente en la paquetería `solvers.h` y con uso en el archivo `main.c`

5. Para este problema se realizó la siguiente tabla de datos como se pide:

Valor z	$f(z)$	I. Lagrange	error L.	I. Neville	I. Diferencias divididas	error DD
0.4	1.491825	1.490929	0.000896	1.490929	1.490929	0.000896
0.8	2.225541	2.226452	0.000911	2.226452	2.226452	0.000911
1.2	3.320117	3.319144	0.000973	3.319144	3.319144	0.000973
1.6	4.953032	4.954125	0.001093	4.954125	4.954125	0.001093
1.9	6.685894	6.688510	0.002615	6.688510	6.688510	0.002615

6. Para este ejercicio se utilizó el método de newton para realizar la interpolación, esto dado a que si bien no se cuenta con una cantidad de valores grandes en general el método de newton presenta menor oscilación, se obtuvieron como interpolación los siguiente datos para $V = [0.598054, 0.561617, 0.485709]$, graficando estos datos junto con los ya conocidos tenemos



Podemos decir de la solución que es una con valores interpolados que se comportan bien, es decir, que no generan una oscilación grande entre los datos. Esto coincide bastante bien con la idea de que se tiene un sistema "natural."[en] el sentido de que obedecen leyes físicas.

4. Conclusiones

Los métodos de interpolación son métodos bastante útiles para comenzar a realizar predicciones sobre un conjunto de datos, estos pueden ser desde datos que no conocen hasta datos que no se pueden tomar, sin embargo cabe resaltar y recordar la importancia de que este tipo de métodos parte de la suposición de conocer en buena medida el comportamiento de los datos.

Quizás para sistemas aplicados físicos, químicos y demás, los métodos no sufran de problemas se suposición, pues estos en su mayoría se comportan bien; pero, por ejemplo, en el caso de interpolar valores como el precio de cambio de la moneda, al interpolar los datos entre dos fechas resulta muy importante esta suposición, puedes puede existir oscilaciones o cambios muy bruscos dentro de la gráfica (es decir, que se tienen datos mal comportados).

5. Referencias

1. Kong, Q., Siauw, T., & Bayen, A. (2020). Python programming and numerical methods: A guide for engineers and scientists. Academic Press.
2. Richard. L. Burden y J. Douglas Faires, Análisis Numérico, 7a Edición, Edito-rial Thomson Learning, 2002.
3. Samuel S M Wong, Computational Methods in Physics and Enginering, Ed. World Scientific, 3rd Edition, 1997.
4. Teukolsky, S. A., Flannery, B. P., Press, W. H., & Vetterling, W. T. (1992). Numerical recipes in C. SMR, 693(1), 59-70.
5. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd Edition, Cambridge University Press, 2007