Actividad 2 – II Unidad: Interpolación de Newton

1. Introducción

La interpolación polinómica de Newton es una técnica para construir un polinomio que pasa exactamente por un conjunto de puntos. Se basa en las diferencias divididas y permite obtener el polinomio de forma incremental; es decir, si se agrega un nuevo punto no es necesario calcular todos los coeficientes desde cero.

Este laboratorio aborda la teoría, la práctica manual (tabla de diferencias divididas) y la implementación en Python para reforzar la comprensión y el uso práctico del método.

2. Objetivos

Generales

- Comprender y aplicar el método de interpolación de Newton usando diferencias divididas.
- Implementar el algoritmo de forma programática en Python.

Específicos

- Construir y llenar la tabla triangular de diferencias divididas manualmente.
- Generar el polinomio en forma progresiva y/o regresiva según corresponda.
- Evaluar la precisión del polinomio interpolante en puntos intermedios y extremos.
- Actualizar el polinomio al añadir un nuevo punto y comparar el coste computacional.

4. Materiales requeridos

- Computadora con Python 3.8+ (recomendado 3.11) y librerías: numpy.
- Visual Studio Code, Sublime Text, Editor de texto, o Jupyter Notebook.
- Calculadora (para los desarrollos manuales).

5. Fundamentos teóricos

Dado un conjunto de $n+1$ puntos conocidos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, se busca una función $P_n(x)$ tal que $P_n(x_i) = y_i$ para todo $i = 0, 1, \ldots, n$.

La **interpolación** busca construir una función polinómica $P_n(x)$ que pase exactamente por un conjunto de $n+1$ puntos conocidos:

El polinomio interpolante de Newton se basa en las diferencias divididas para construir $P_n(X)$ de forma incremental, aprovechando cálculos previos.

5.1 Diferencias Divididas

Las diferencias divididas son coeficientes que se calculan a partir de los puntos dados, y representan las pendientes de orden superior entre los puntos.

Se definen recursivamente:

$$f[x_i] = y_i$$

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

En general:

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}$$

Cada nivel usa los resultados del nivel anterior, formando una tabla triangular de diferencias divididas

5.2 Polinomio interpolante de Newton (forma progresiva)

Una vez calculadas las diferencias divididas, el polinomio de Newton se construye como:

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

Esta es la forma progresiva, ya que se desarrolla a partir de x_0 hacia adelante.

Cada término agrega una corrección basada en las diferencias divididas de mayor orden.

5.3 Forma regresiva de Newton

Si en lugar de comenzar con X0, se comienza desde el extremo derecho con Xn, se obtiene la forma regresiva:

$$P_n(x) = f[x_n] + f[x_n, x_{n-1}](x - x_n) + f[x_n, x_{n-1}, x_{n-2}](x - x_n)(x - x_{n-1}) + \cdots + f[x_n, x_{n-1}, \ldots, x_0](x - x_n)(x - x_{n-1})$$

Esta forma es útil cuando se interpolan valores cercanos al extremo superior de los datos.

Ejercicio 1 — Reconstrucción de un polinomio conocido

Enunciado. Dado el conjunto de puntos $(1, 1), (2, 4), (3, 9), (4, 16)$, construir el polinomio interpolante mediante la tabla de diferencias divididas, mostrar la forma progresiva y la regresiva y comprobar en $x = 2.5$.

```
import numpy as np


def divided_differences(xs, ys):
n = len(xs)
# Construimos una tabla n x n, diagonal superior almacenará las diferencias
table = np.zeros((n, n))
```

```
table[:,0] = ys
for j in range(1, n):
for i in range(n - j):
table[i,j] = (table[i+1,j-1] - table[i,j-1]) / (xs[i+j] - xs[i])
# Los coeficientes de Newton están en table[0, :]
coef = table[0, :]
return coef, table




def newton_eval(coef, xs, x):
# Horner para forma factorizada de Newton (evaluación eficiente)
n = len(coef)
result = coef[n-1]
for k in range(n-2, -1, -1):
result = result * (x - xs[k]) + coef[k]
return result


# Datos
xs = np.array([1.0, 2.0, 3.0, 4.0])
ys = np.array([1.0, 4.0, 9.0, 16.0])


coef, table = divided_differences(xs, ys)
print('Coeficientes (diferencias divididas, desde f[x0]):', coef)
print('\nTabla completa de diferencias divididas:\n', table)


# Evaluación en x=2.5
x_test = 2.5
y_test = newton_eval(coef, xs, x_test)
print(f'P_3({x_test}) =', y_test)
```

Ejercicio 2 — Interpolación de una función polinómica

Probar en X = 1.5

$$f(x) = x^2 + 2x + 1$$

x	f(x)
0	1
1	4
2	9

Solución Manual

$$f[x_0, x_1] = (4-1)/(1-0) = 3$$

$$f[x_1, x_2] = (9-4)/(2-1) = 5$$

$$f[x_0, x_1, x_2] = (5-3)/(2-0) = 1$$

$$P(x) = 1 + 3(x-0) + 1(x-0)(x-1)$$

$$P(x) = 1 + 3x + x(x-1) = x^2 + 2x + 1$$

```
#EJERCICIO 2

import numpy as np

def newton_interpolation(x_data, y_data, x):
n = len(x_data)
coef = np.copy(y_data).astype(float)

for j in range(1, n):
    coef[j:n] = (coef[j:n] - coef[j-1:n-1]) / (x_data[j:n] - x_data[0:n-j])

result = coef[-1]
for k in range(n-2, -1, -1):
    result = result * (x - x_data[k]) + coef[k]
return result, coef
x_points = np.array([0, 1, 2])
y_points = np.array([1, 4, 9])

val, coef = newton_interpolation(x_points, y_points, 1.5)
print("Coeficientes:", coef)
print("P(1.5) =", val)
```

Ejercicio 3 — Interpolación de una función polinómica

$$f(x) = e^x$$

x	f(x)
0	1.000
1	2.718
2	7.389

Solución Manual

$$f[x_0, x_1] = (2.718 - 1)/(1 - 0) = 1.718$$

$$f[x_1, x_2] = (7.389 - 2.718)/(2 - 1) = 4.671$$

$$f[x_0, x_1, x_2] = (4.671 - 1.718)/(2 - 0) = 1.4765$$

$$P(x) = 1 + 1.718x + 1.4765x(x - 1)$$

```
# EJERCICIO 3
import numpy as np
import math

x_points = np.array([0, 1, 2])
y_points = np.array([math.e**x for x in x_points])

def newton_interp(x_data, y_data, x):
n = len(x_data)
coef = np.copy(y_data)
for j in range(1, n):
coef[j:n] = (coef[j:n] - coef[j-1:n-1]) / (x_data[j:n] - x_data[0:n-j])
result = coef[-1]
for k in range(n-2, -1, -1):
result = result * (x - x_data[k]) + coef[k]
return result

print("P(1.5) =", newton_interp(x_points, y_points, 1.5))
```

Ejercicio 4 — Interpolación de una función no polinómica

$$f(x) = \sin(x)$$

Usando radianes.

x	f(x)
0	0.000
π/4	0.707
π/2	1.000

Solución Manual

$$f[x_0, x_1] = (0.707 - 0)/(\pi/4 - 0) = 0.900$$

$$f[x_1, x_2] = (1 - 0.707)/(\pi/2 - \pi/4) = 0.744$$

$$f[x_0, x_1, x_2] = (0.744 - 0.900)/(\pi/2 - 0) = -0.099$$

$$P(x) = 0 + 0.900x - 0.099x(x - \pi/4)$$

```python
# EJERCICIO 4
import numpy as np
import math

x_points = np.array([0, math.pi/4, math.pi/2])
y_points = np.sin(x_points)

def newton_interp(x_data, y_data, x):
n = len(x_data)
coef = np.copy(y_data)
for j in range(1, n):
coef[j:n] = (coef[j:n] - coef[j-1:n-1]) / (x_data[j:n] - x_data[0:n-j])
result = coef[-1]
for k in range(n-2, -1, -1):
result = result * (x - x_data[k]) + coef[k]
return result

x_eval = math.pi / 3
print("P(pi/3) =", newton_interp(x_points, y_points, x_eval))
print("Valor real =", math.sin(x_eval))
```

Ejercicio 5 — Interpolación de una función no polinómica

Enunciado. Sea la función $f(x) = \sin(x)$. Tomar los puntos $x_0 = 0$, $x_1 = \pi/6$, $x_2 = \pi/3$, $x_3 = \pi/2$ y construir el polinomio interpolante de Newton. Evaluar en $x = \pi/4$ y comparar con $\sin(\pi/4) = \sqrt{2}/2$.

Realice su solución manual y haga código en Python