Interpolación

Objetivos

1. Programar el método de interpolación de Newton
2. Programar el método de interpolación de Lagrange
3. Programar el método de interpolación de Hermite
4. Programar el método de interpolación de Newton para 4 puntos dados de una función

Interpolación de Newton

La interpolación de Newton a través de $n+1$ puntos en un intervalo $[a,b]$ está dada por la función $P_n(x)$ en su forma de Newton con centros en $x_0, x_1, \ldots, x_{n-1}$:

$$P_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n(x-x_0)(x-x_1)\ldots(x-x_n) \quad (1)$$

o en su forma anidada:

$$P_n(x) = a_0 + (x-x_0)\{a_1 + (x-x_1)\{a_2 + \cdots + (x-x_{n-1})\{a_{n-1} + a_n(x-x_n)\}\ldots\}\} \quad (2)$$

Usando el método de diferencias divididas de Newton podemos rescribir el polinomio (1) de la forma:

$$\begin{aligned} P_n(x) = f(x_0) + f[x_0, x_1](x-x_0) + f[x_0, x_0, x_1](x-x_0)(x-x_0) + \ldots \\ + f[x_0, x_0, \ldots, x_n](x-x_0)(x-x_0)\ldots(x-x_{n-1}) \end{aligned} \quad (3)$$

dónde:

$$f(x_0) = a_0 \quad (4)$$

$$f[x_i, x_{i*1}] = \frac{f(x_{i*1}) - f(x_i)}{x_{i+1} - x_i} \quad (5)$$

y

$$f[x_i, x_{i*1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+1} - x_i} \quad (6)$$

Algoritmo

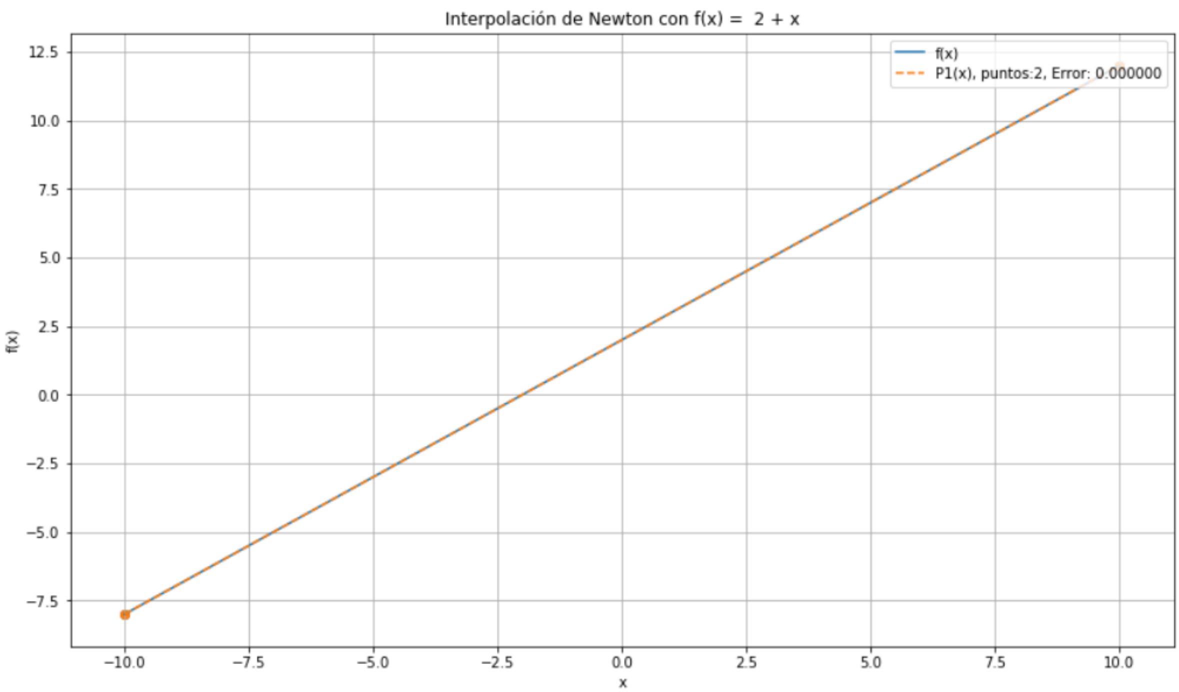
```
n <- Grado del polinomio
x[n+1] <- vector con las coordenadas en x de los n+1 puntos
y[n+1] <- vector con las coordenadas en y de los n+1 puntos

// Retorna los coeficientes correspondientes al polinomio de las diferencias finitas
function get_newton_coefs(x, y)
    ndots = (size * (size + 1)) / 2 // Número de diferencia finitas a calcular
    div_table[size] // Tabla de diferencias finitas
    coefs[size] // Vector de los coeficientes
    // Copiamos los valores de f(x) como primeras diferencias finitas
```

```
    for y_i, index in enumerate(y):
        div_table[index] = y_i
    // Calculo de las siguientes diferencias finitas
    coef_index = 0;
    for (i = 0, inc = size, index = size; i < ndots - 1; i+=inc, inc--)
        x1 = 0, x2 = coef_index + 1
        for (j = 0; j < inc - 1; j++, x1++, x2++)
            div_table[index++] = (div_table[i+j+1] - div_table[i+j]) / (x[x2] - x[x1])
        coefs[coef_index++] = div_table[i]
    coefs[coef_index] = div_table[ndots - 1]

    return coefs

// Retorna el valor del polinomio de Newton evaluado en algun valor
// Se evalua a través de la forma anidada de Newton
function interpolation(x, coefs, n, value):
    if n == 0
        return coefs[0]
    return coefs[0] + (value - x[0]) * evaluation(x[1:], coefs[1:], size - 1, value)


```

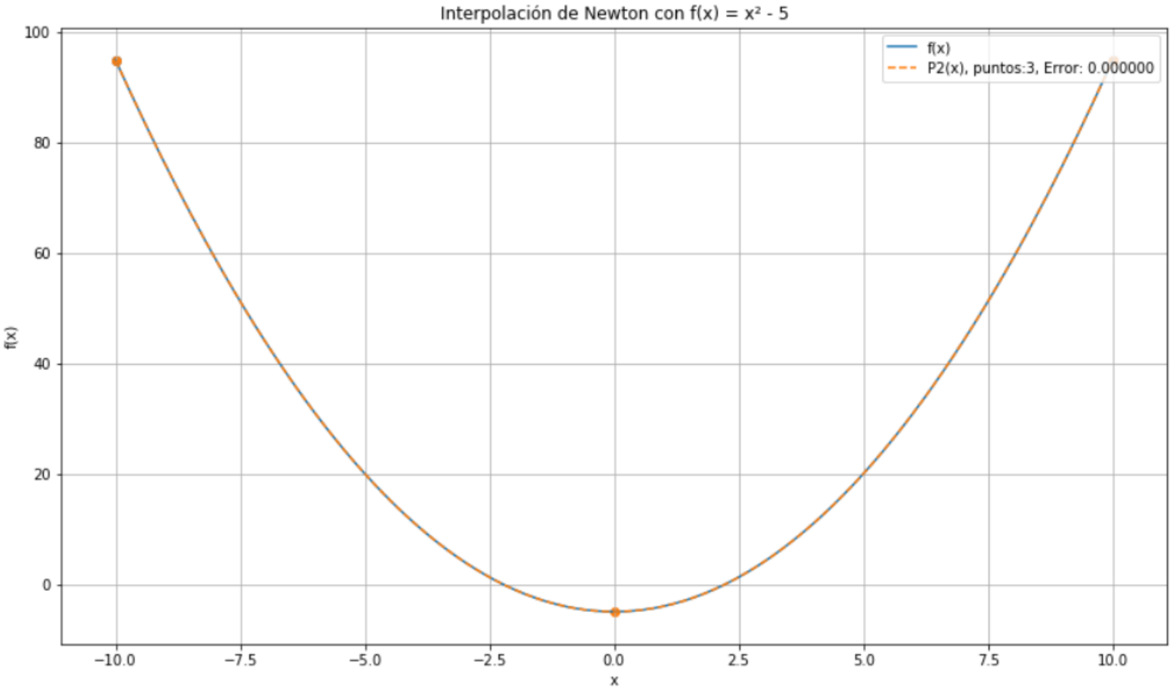
Resultados

Se presentan los resultados obtenidos para diferentes aproximaciones a las siguientes funciones:

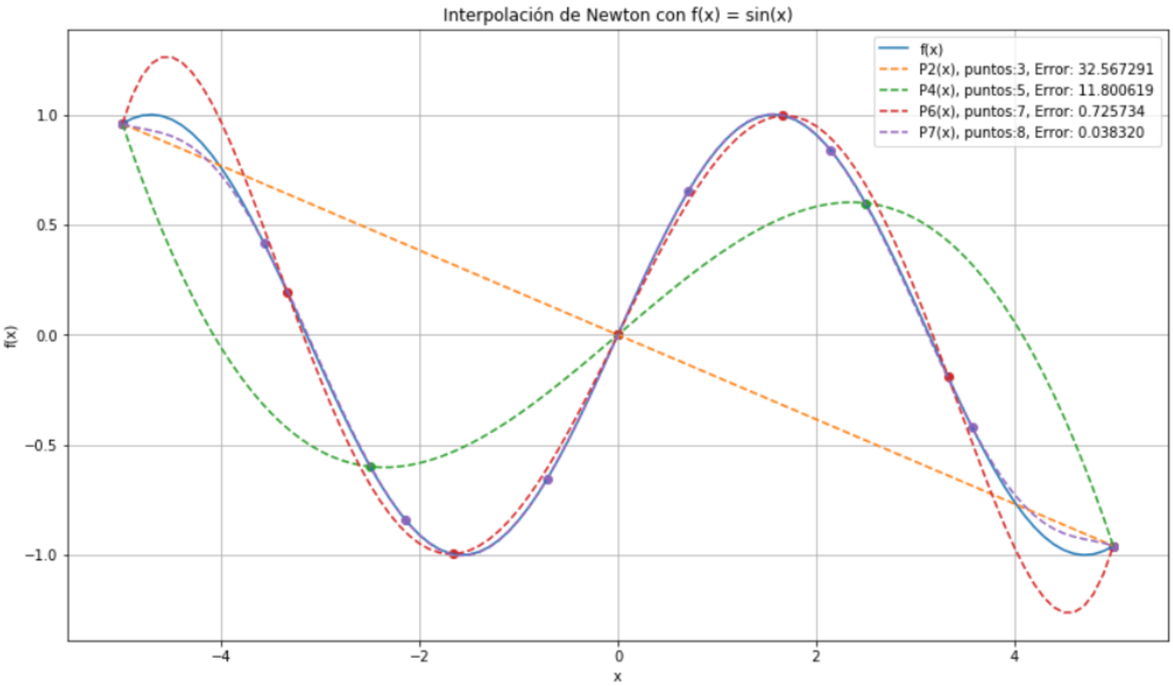
- Función: $f(x) = 2 + x$,
- Rango de Evaluación: $(-10, 10)$
- Número de puntos para interpolación: 2



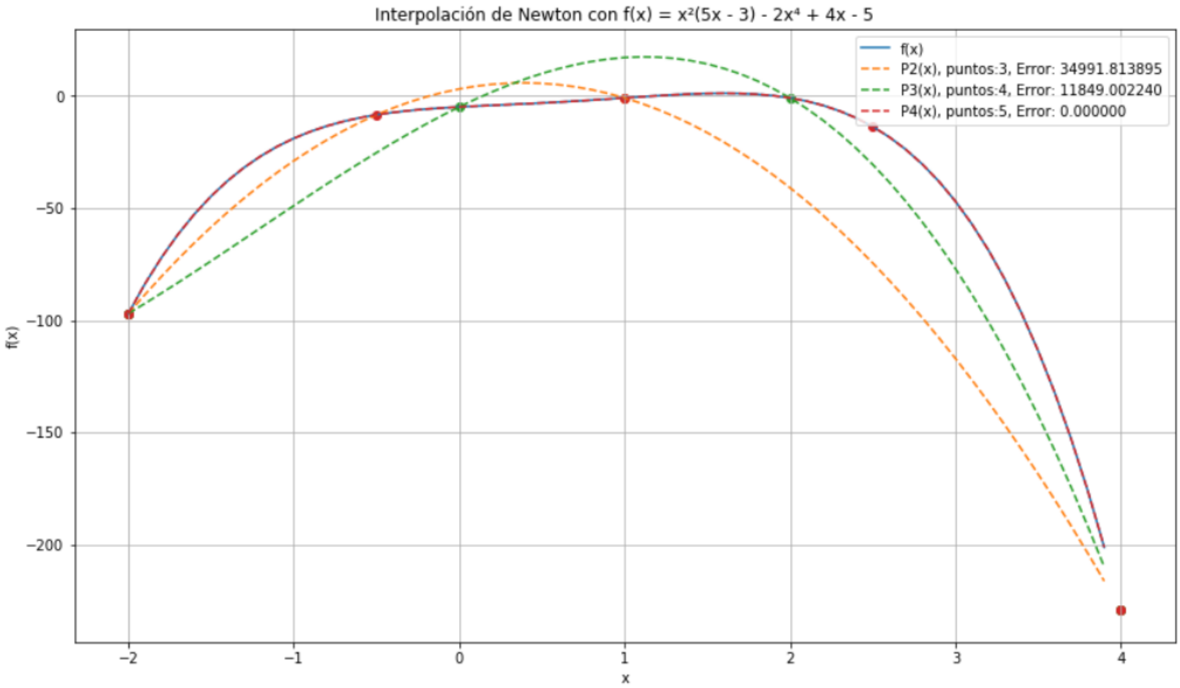
- Función: $f(x) = x^2 - 5$
- Rango de evaluación: $(-10, 10)$
- Número de puntos para interpolación: 3



- Función $f(x) = sin(x)$,
- Rango de evaluación: $(-5, 5)$
- Número de puntos para interpolación: 3, 5, 7 y 8



- Función: $f(x) = x^2(5X - 3) - 2X^4 + 4X - 5$
- Rango de evaluación: $(-2, 4)$
- Número de puntos para interpolación: 3, 4 y 5



Para el cálculo del error se utilizó el Error Cuadrático Medio con puntos en el intervalo en incrementos de 0.1. Los puntos son equidistantes en el intervalo de cada gráfica.

Interpolación de Lagrange

La interpolación de Lagrange a través de $n+1$ puntos en un intervalo $[a,b]$ está dada por la función:

$$P_n(x) = \sum_{i=1}^{n} a_i l_i(x) \tag{7}$$

dónde:

$$l_i(x) = \prod_{k=0,k\neq i}^{n} \frac{x - x_k}{x_i - x_k} \tag{8}$$

$$a_i = f(x_i) \tag{9}$$

Algoritmo

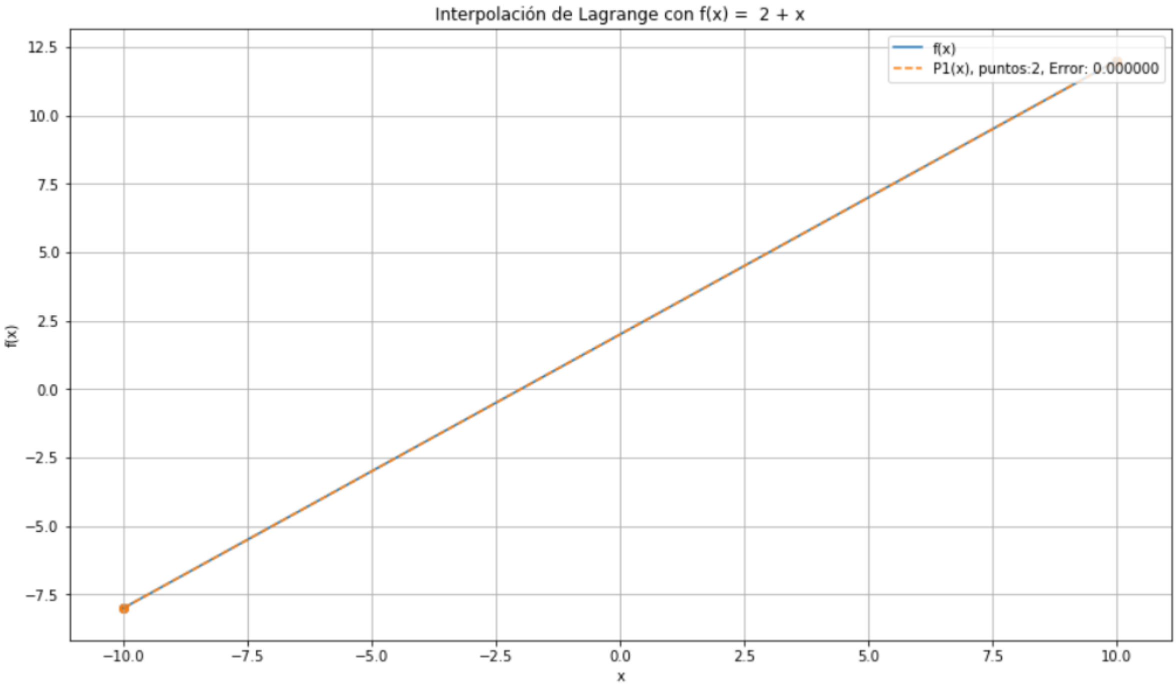
```
n <- Grado del polinomio
x[n+1] <- vector con las coordenadas en x de los n+1 puntos
y[n+1] <- vector con las coordenadas en y de los n+1 puntos

// Retorna la evaluacion en un valor para el polinomio de interpolación de lagrange
function interpolation(x, y, value)
    size = n + 1
    p_x = 0.0
    for (i = 0; i < size; i++)
        l_i = 1.0;
        for (int j = 0; j < size; j++)
            if (j == i) continue
            l_i *= (value - x[j]) / (x[i] - x[j])
        p_x += y[i] * l_i;
```

```
    return p_x;
```

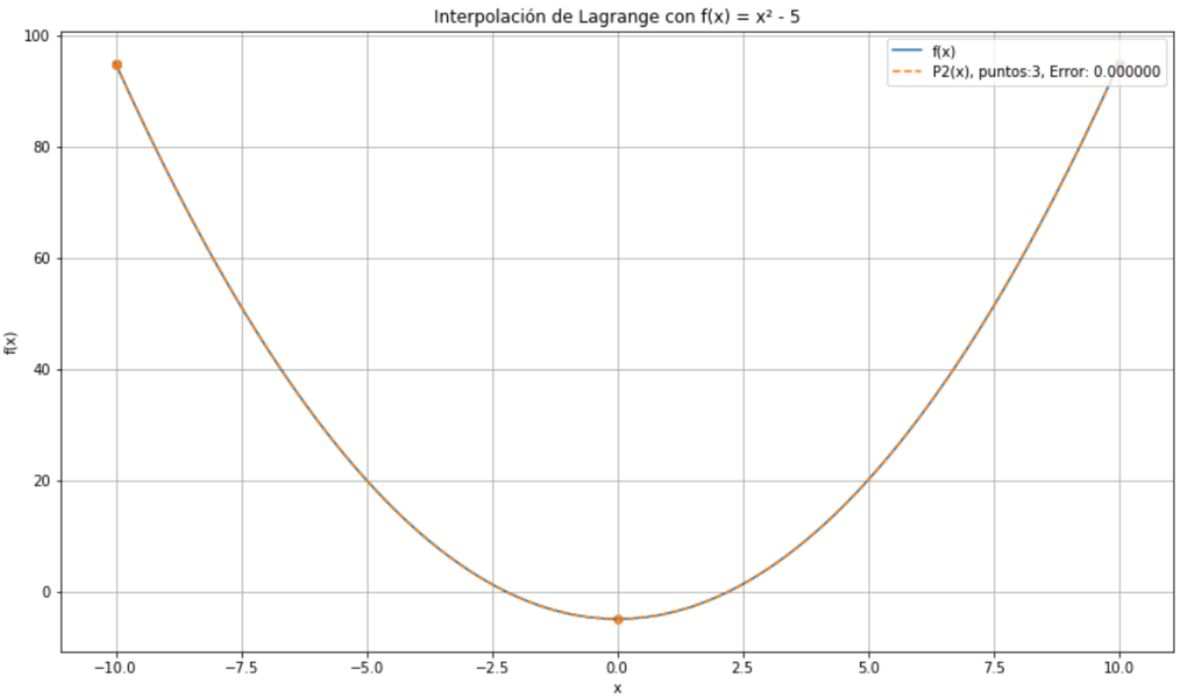
Resultados

Se presentan los resultados obtenidos para diferentes aproximaciones a las siguientes funciones:

- Función: $f(x) = 2 + x$,
- Rango de Evaluación: $(-10, 10)$
- Número de puntos para interpolación: 2

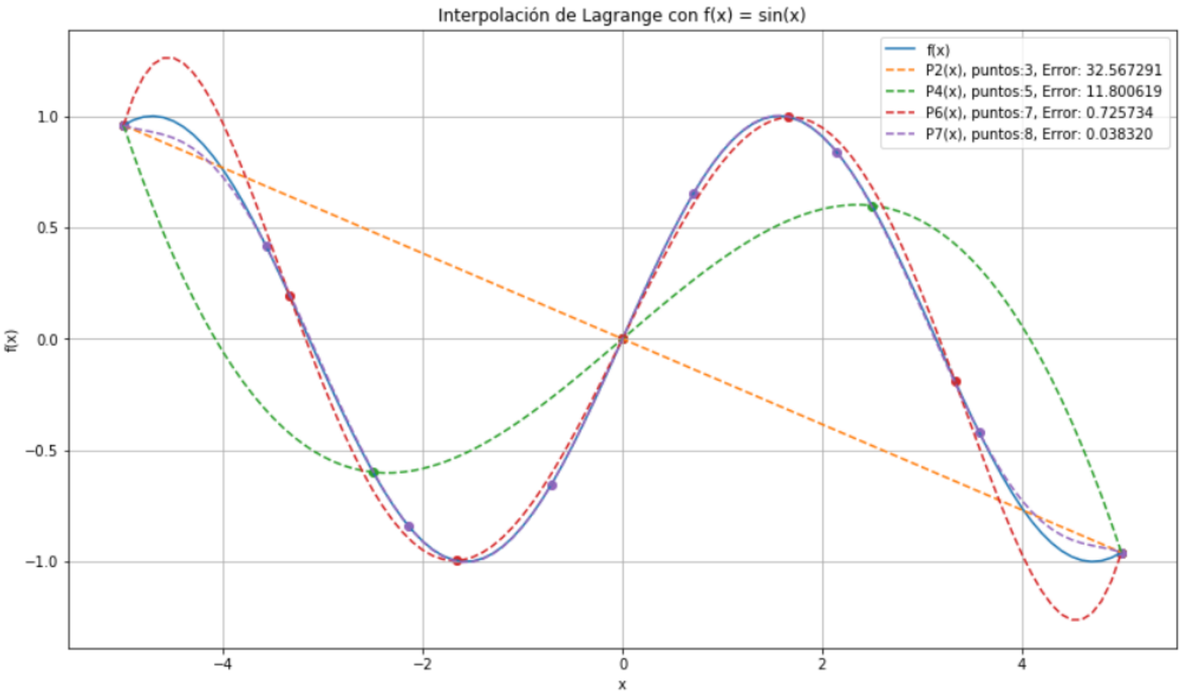


- Función: $f(x) = x^2 - 5$
- Rango de evaluación: $(-10, 10)$
- Número de puntos para interpolación: 3

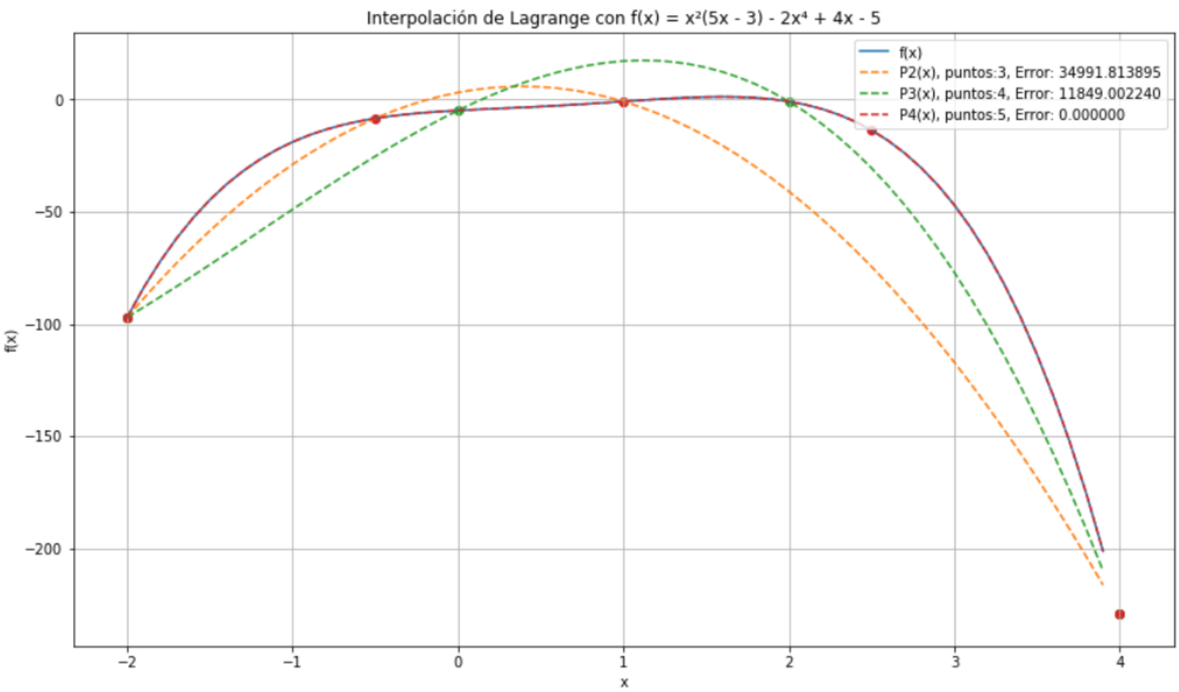


- Función $f(x) = sin(x)$,
- Rango de evaluación: $(-5, 5)$

- Número de puntos para interpolación: 3, 5, 7 y 8



- Función: $f(x) = x^2(5X - 3) - 2X^4 + 4X - 5$
- Rango de evaluación: $(-2, 4)$
- Número de puntos para interpolación: 3, 4 y 5



Para el cálculo del error se utilizó el Error Cuadrático Medio con puntos en el intervalo en incrementos de 0.1. Los puntos son equidistantes en el intervalo de cada gráfica.

Interpolación de Hermite

La interpolación de Hermite a través de $n + 1$ puntos en un intervalo $[a, b]$ está dada por la función:

$$P_{2n+1}(x) = \sum_{i=0}^{n} u_i(x)P_{2n+1}(x_i) + \sum_{i=0}^{n} v_i(x)P'_{2n+1}(x_i) \tag{10}$$
$$= \sum_{i=0}^{n} u_i(x)f(x_i) + \sum_{i=0}^{n} v_i(x)f'(x_i)$$

donde $f(x)$ es la función real evaluado en x y dónde

$$u_i(x) = (-2l_i'(x_i)x + 1 + 2x_i l_i'(x_i))l_i^2(x) \tag{11}$$

$$v_i(x) = (x - x_i)l_i^2(x) \tag{12}$$

con $l_i(x)$ como el coeficiente descrito en el método de Lagrange en la ecuación (8) y

$$l_i' = \sum_{j=0, j\neq i}^{n} [\frac{1}{x_i - x_j} \prod_{m=0, m\neq(i,j)}^{n} \frac{x - x_m}{x_i - x_m}] \tag{13}$$

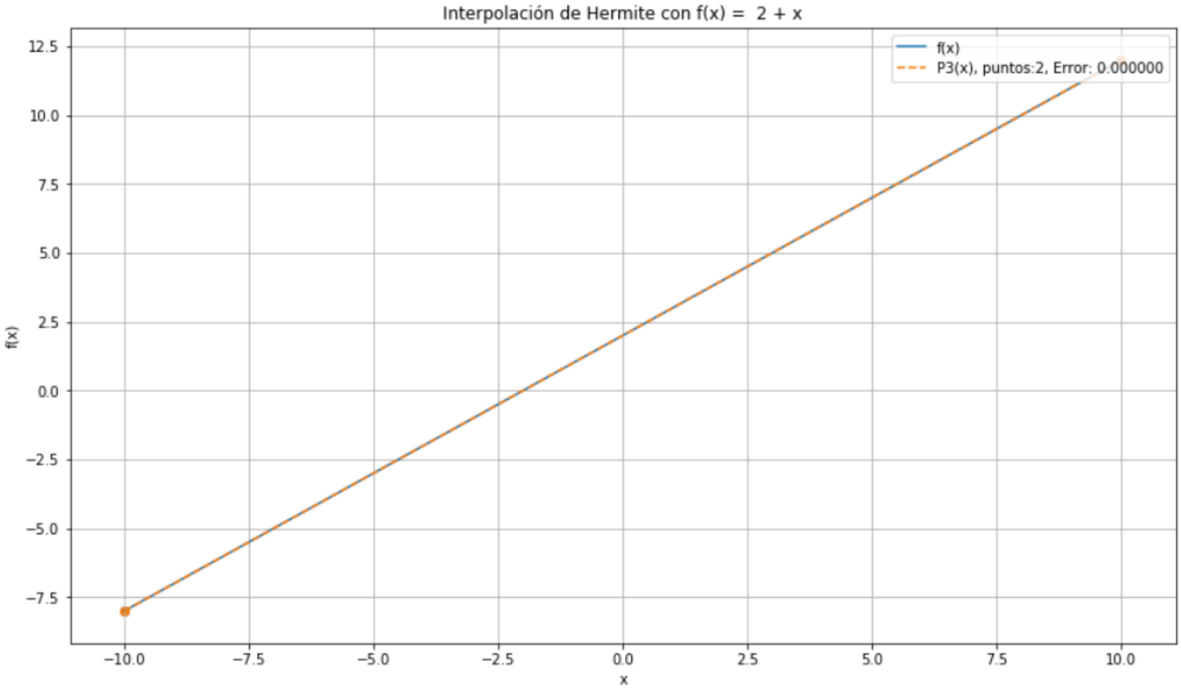
Algoritmo

```
n <- Grado del polinomio
x[n+1] <- vector con las coordenadas en x de los n+1 puntos
y[n+1] <- vector con las coordenadas en y de los n+1 puntos
dy[n+1] <- vector con los valores correspondientes a y' = f'(x) de los n+1 puntos

// Retorna el valor L_i
function get_li(i, x, size, value) {
    l_i = 1.0
    for (j = 0; j < size; j++){
        if (j == i) continue
        l_i *= (value - x[j]) / (x[i] - x[j])
    return l_i

// Retorna el valor de la deriva de l_i
function get_dlj(j, x, size, value)
    dl_j = 0.0;
    for (int i = 0; i < size; i++)
        if (i == j) continue
        sub1 = 1.0
        for (int m = 0; m < size; m++)
            if (m == i || m == j) continue
            sub1 *= (value - x[m]) / (x[j] - x[m])
        dl_j += sub1 / (x[j]  - x[i])
    return dl_j;

// Evaluación del polinomio de interpolación de Hermite en un punto dado
function interpolation(x, y, dy, size, value)
    double p_x_term1 = 0
    double p_x_term2 = 0
    for (int i = 0; i < size; i++)
        double li_d1 = get_dlj(i, x, size, x[i])
        double li = get_li(i, x, size, value)
        p_x_term1 += (-2 * li_d1 * value + 1 + 2 * x[i] * li_d1) * li * li * y[i]
        p_x_term2 += (value - x[i]) * li * li * dy[i]
    return p_x_term1 + p_x_term2
```

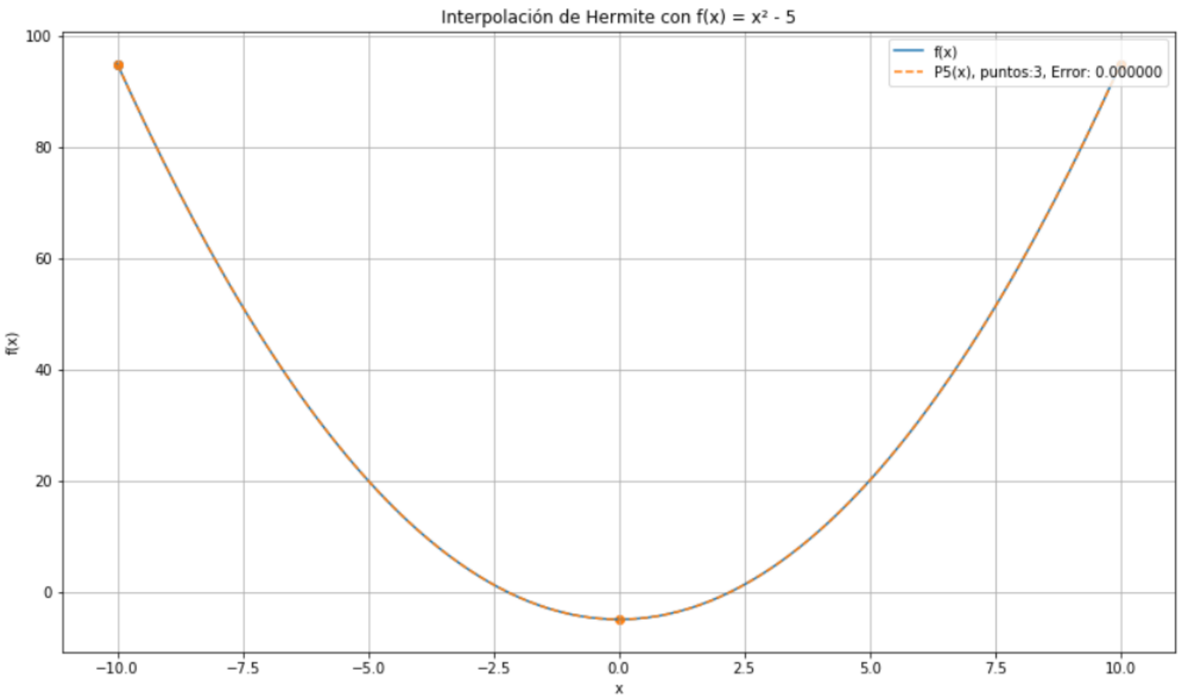
Resultados

Se presentan los resultados obtenidos para diferentes aproximaciones a las siguientes funciones:

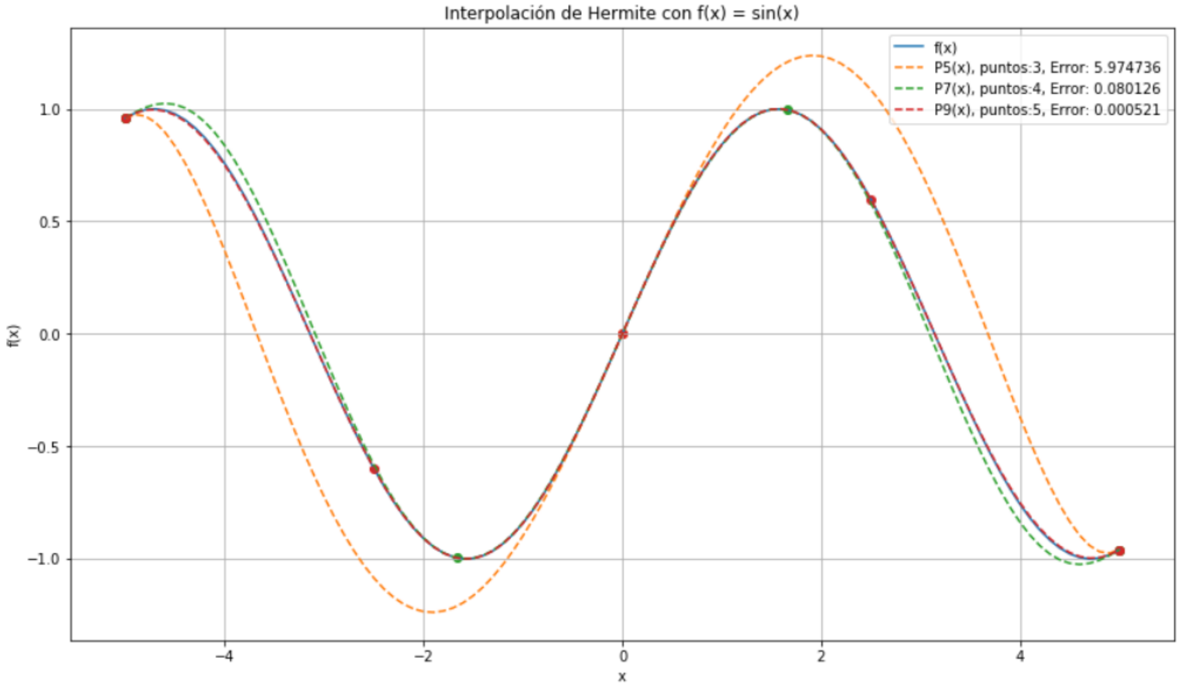
- Función: $f(x) = 2 + x$,
- Rango de Evaluación: $(-10, 10)$
- Número de puntos para interpolación: 2



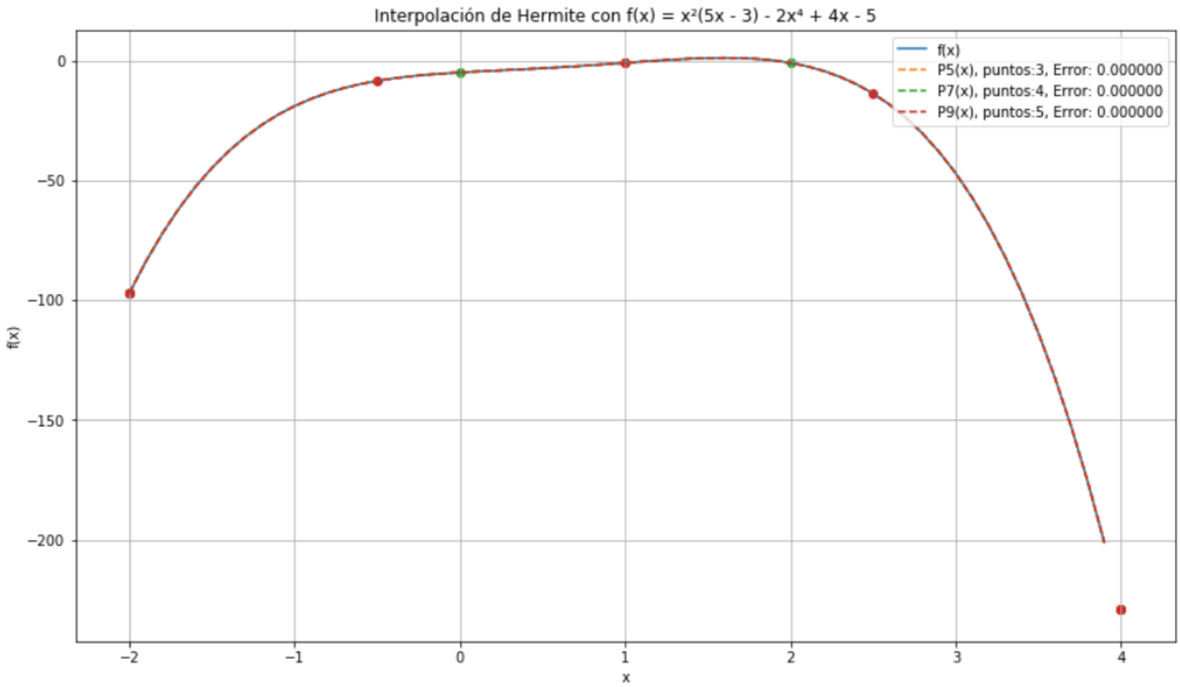
- Función: $f(x) = x^2 - 5$
- Rango de evaluación: $(-10, 10)$
- Número de puntos para interpolación: 3



- Función $f(x) = sin(x)$,
- Rango de evaluación: $(-5, 5)$
- Número de puntos para interpolación: 3, 4, y 5



- Función: $f(x) = x^2(5X - 3) - 2X^4 + 4X - 5$
- Rango de evaluación: $(-2, 4)$
- Número de puntos para interpolación: 3, 4 y 5



Para el cálculo del error se utilizó el Error Cuadrático Medio con puntos en el intervalo en incrementos de 0.1. Los puntos son equidistantes en el intervalo de cada gráfica.

Interpolación de Newton para 4 puntos dados de una función

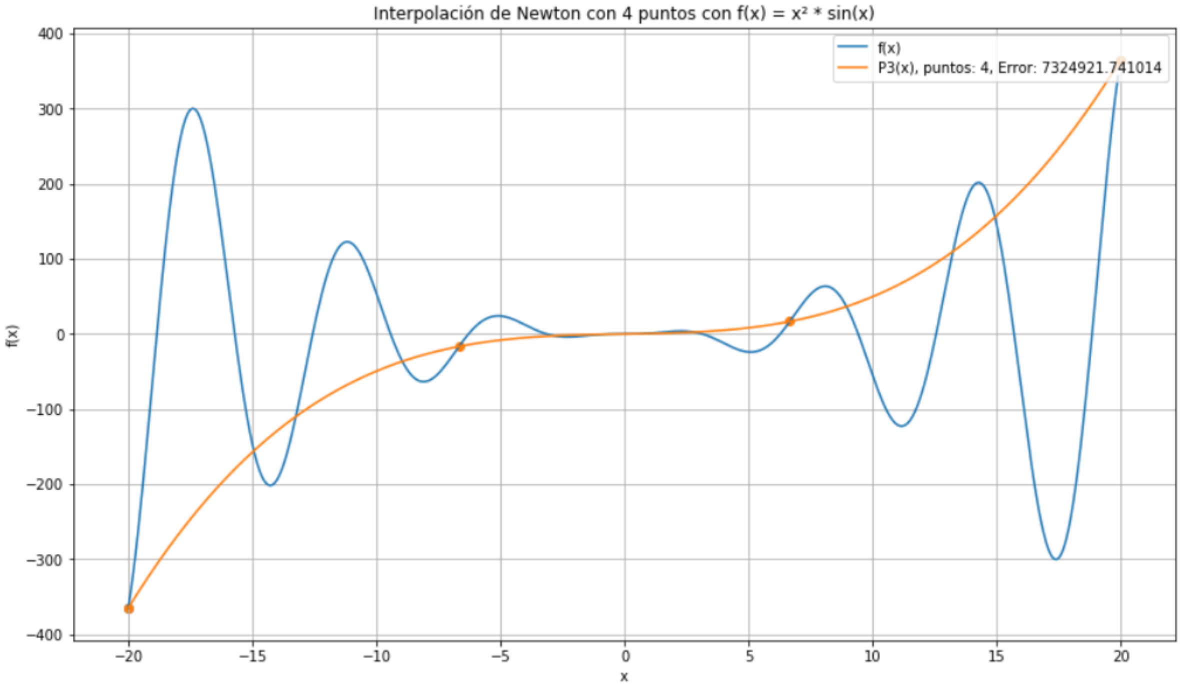
La función a evaluar para este caso especial de la Interpolación de Newton es la siguiente:

$$f(x) = x^2 sin(x) \tag{14}$$

en el intervalo de $(-20, 20)$. Los cuatro puntos de elección son:

```
x = [-20, -6.666666666666666, 6.666666666666668, 20.0]
Y = [-365.17810029105107, -16.62894358094307, 16.62894358094315,
365.17810029105107]
```

La siguiente gráfica nos muestra el polinomio de interpolación de grado 3 obtenido y la función original. Los puntos evaluados para la gráfica son equidistantes con incrementos de 0.1 en el intervalo, con un error de aproximación de *7324921.741013986*



El error al igual que los anteriores es calculado a través del Error cuadrático Medio

Tiempos de ejecución

La siguiente tabla muestra los tiempos de ejecución para los algoritmos antes descritos para los procesos de interpolación de las 4 funciones anteriores y la generación de una muestra en el intervalo dado.

Algoritmo	Tiempo Promedio	Repeticiones	Tiempo Promedio total
Newton	0.004000s	5000	8.972000s
Lagrange	0.005000s	5000	23.942000s
Hermite	0.014000s	5000	57.454000s

Tiempo promedio y error para la interpolación de la función $f(x) = sin(x)$ usando un polinomio de grado 7:

Algoritmo	ECM	Tiempo Promedio
Newton	0.038320	0.0015780s
Lagrange	0.038320	0.0016610s
Hermite	0.000000	0.0017120s

Conclusiones

Tanto en las gráficas como en la tabla errores se observa que los algoritmos de Newton como el de Lagrange el error de aproximación es similar para para polinomios del mismo grado, la diferencia se nota en los tiempos de ejecución, pues el algoritmo de interpolación de Newton es bastante más rápido que el de Lagrange.

El algoritmo de Hermite nos permite aproximar mucho mejor el polinomio de la función real con polinomios del mismo grado, como podemos observar el error usando el algoritmo de Hermite con un polinomio de grado 7 el error obtenido es cero, y para un polinomio de grado 7 usando los otros dos algoritmos el error aún es significativo. sin embargo hay que notar que el algoritmo de interpolación de Hermite requiere conocer los valores de las derivadas de los puntos dados y su tiempo de ejecución es aún más alto.