# Universidad Autónoma de Querétaro

## Facultad de Ingeniería

# Diferencias divididas y polinomial

*Análisis numérico*


Autor:
David Gómez Torres


13 Octubre del 2021

# ÍNDICE GENERAL

# CAPÍTULO 1

# INTRODUCCIÓN

## 1.1. Interpolación polinomial de Newton en diferencias dividas

La **interpolación polinomial** consiste en determinar el polinomio único de n-ésimo grado que se ajuste a $n+1$ puntos asociados con datos.

### 1.1.1. Interpolación lineal

La forma más simple de interpolación, llamada **interpolación lineal**, consiste en unir dos puntos asociados con datos con una línea recta.
utilizando triángulos semejantes,

$$\frac{f_1(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

En general, cuanto menor sea el intervalo entre los puntos asociados con datos, mejor será la aproximación. Esto se debe al hecho de que, conforme el intervalo disminuye, una función continua estará mejor aproximada por una línea recta.

L fórmula de interpolación lineal

$$f_1(x) = f(x_0) + \frac{f(x_1 - f(x_0))}{x_1 - x_0}(x - x_0) \tag{1.1}$$

### 1.1.2. Interpolación cuadrática

Una estrategia para mejorar la estimación consiste en introducir alguna curvatura a la línea que une los puntos.Si se tienen tres puntos asociados con datos, éstos pueden ajustarse en un polinomio de segundo grado (también conocido como **polinomio cuadrático o parábola**).

Estos pueden escribirse como

$$f_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) \tag{1.2}$$

donde

$$a_0 = b_0 - b_1 x_0 + b_2 x_0 x_1$$
$$a_1 = b_1 - b_2 x_0 - b_2 x_1$$
$$a_2 = b_2$$

Después en la ecuación (1.2) se sustituye $x = x_1$ para tener

$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{1.3}$$

Después en la ecuación (1.3) se sustituye para obtener

$$b_2 = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} \tag{1.4}$$

donde $b_1$ representa la pendiente de la línea que une los puntos $x_0$ y $x_1$ y $b_2$ determina la curvatura de segundo grado en la fórmula.

### 1.1.3. Forma general de los polinomios de interpolación de Newton

El análisis anterior puede generalizarse para ajustar un polinomio de n-ésimo grado a $n+1$ puntos asociados con datos. El polinomio de n-ésimo grado es

$$f_n(x) = b_0 + b_1(x - x_0) + \cdots + b_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

Para un polinomio de n-ésimo grado se requieren $n+1$ puntos asociados con datos: $[x_0, f(x_0)]$, $[x_1, f(x_1)]$,$\cdots$, $[x_n, f(x_n)]$.
Esto nos da la n-ésima **diferencia dividida**

$$f[x_n, x_{n-1}, \cdots, x_1, x_0] = \frac{f[x_n, x_{n-1}, \cdots, x_1] - f[x_{n-1}, x_{n-2}, \cdots, x_0]}{x_n - x_0}$$

Estas diferencias sirven para evaluar los coeficientes en las ecuaciones,los cuales se sustituirán en la ecuación (1.1.3) para obtener el polinomio de interpolación.

$$f_n(x) = f(x0_) + (x{-}x_0)f[x_1, x_0] + (x{-}x_0)$$
$$(x{-}x_1)f[x_2, x_1, x_0] + \cdots + (x{-}x_0)(x{-}x_1)\cdots(x{-}x_{n-1})f[x_n, x_{n-1}, \cdots, x_0] \tag{1.5}$$

que se conoce como **polinomio de interpolación de Newton en diferencias divididas**.

### 1.1.4. Errores de la interpolación polinomial de Newton

Como ocurrió con la serie de Taylor, si la función verdadera es un polinomio de n-ésimo grado, entonces el polinomio de interpolación de n-ésimo grado basado en $n+1$ puntos asociados con datos dará resultados exactos.

Para un polinomio de interpolación de n-ésimo grado, una expresión para el error es

$$R_n = \frac{f^{(n+1)}(\xi}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n) \tag{1.6}$$

donde $\xi$ está en alguna parte del intervalo que contiene la incógnita y los datos.

### 1.1.5. Algoritmo computacional

**Algorithm 1** Polinomio de Interpolación de Newton

**Entrada:** array x, y; entero n, yint; real ea;
**Salida:** array yint2; real error;
**while** error > es **do**
  **for** i→n **do**
    fdd$_{1,0}$ → y$_i$

    **for** j=1 → n **do**
      **for** i=1 → n-j **do**
        fdd$_{i,j}$ = (fdd$_{i+1,j-1}$ - fdd$_{i,j-1}$)/(x$_{i+j}$-x$_i$)
      **end for**
    **end for**
    xterm = 1
    $yint_0$ = fdd$_{0,0}$
    **for** order = 1 → n **do**
      xterm = xterm * (xi - $x_{\text{order-1}}$
      yint2 = $yint_{\text{order-1}}$ + $fdd_{0,order}$ * xterm
      $ea_{\text{order-1}}$ = yint2 - $yint_{\text{order-1}}$
      $yint_{\text{order}}$ = yint2
    **end for**
  **end while**
**return** print(yint, error)

## 1.2. Polinomio de interpolación de Lagrange

El **polinomio de interpolación de Lagrange** es simplemente una reformulación del polinomio de Newton que evita el cálculo de las diferencias divididas, y se representa de manera concisa como

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i) \tag{1.7}$$

donde

$$L_i(x) = \prod_{j=0}^{n} \frac{x - x_j}{x_i - x_j} \tag{1.8}$$

donde $\Pi$ designa el "producto de".

Como en el método de Newton, la forma de Lagrange tiene un error estimado por

$$R_n = f[x, x_n, x_{n-1}, \ldots, x_0] \prod_{j=0}^{n} (x - x_i) \tag{1.9}$$

De este modo, si se tiene un punto adicional en $x = x_{n+1}$ , se puede obtener un error estimado.

En los casos donde se desconoce el grado del polinomio, el método de Newton tiene ventajas debido a la comprensión que proporciona respecto al comportamiento de las fórmulas de diferente grado.De esta manera, para cálculos exploratorios, a menudo se prefiere el método de Newton.

## 1.3. Interpolación inversa

Como la nomenclatura implica, los valores de f($x$) y $x$ en la mayoría de los problemas de interpolación son las variables dependiente e independiente, respectivamente.En consecuencia, los valores de las x con frecuencia están espaciados uniformemente.

A ese problema se le conoce como **interpolación inversa**. En un caso más complicado, se podría intentar intercambiar los valores f($x$) y $x$ [es decir, tan sólo graficar $x$ contra f($x$)] y usar un procedimiento como la interpolación de Lagrange para determinar el resultado. Por desgracia, cuando se invierte las variables no hay garantía de que los valores junto con la nueva abscisa [las f($x$)] estén espaciados de una manera uniforme.

# CAPÍTULO 2

# METODOLOGÍA

## 2.1. Interpolación de Newton

**Problema 18.5**

Dados los datos

| x | 1.6 | 2 | 2.5 | 3.2 | 4 | 4.5 |
|---|---|---|---|---|---|---|
| f($x$) | 2 | 8 | 14 | 15 | 8 | 2 |

a) Calcule f(2.8) con el uso de polinomios de interpolación de Newton de grados 1 a 3. Elija la secuencia de puntos más apropiada para alcanzar la mayor exactitud posible para sus estimaciones.

b) Utilice la ecuación (18.18) para estimar el error de cada predicción.

1. Gráfica del ajuste polinomial a los datos

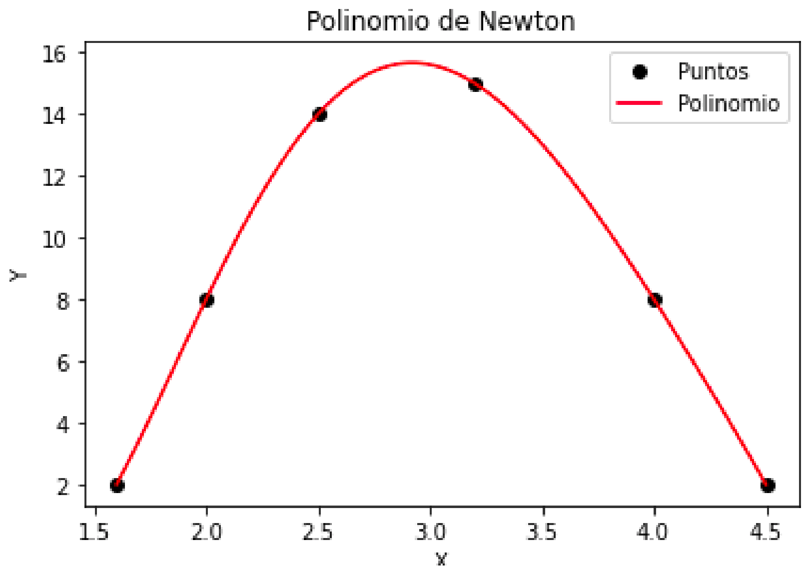


Figura 2.1: Línea ajustada a los datos

$$-0.481x^5 + 8.247x^4 - 53.767x^3 + 159.836x^2 - 204.523x + 91.285$$

2.
```python
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt

#Datos de prueba
xi = np.array([1.6, 2, 2.5, 3.2,4,4.5])
fi = np.array([2, 8, 14, 15,8,2])

# Tabla de Diferencias
n = len(xi)
ki = np.arange(0,n,1)
tabla = np.concatenate(([ki],[xi],[fi]),axis=0)
tabla = np.transpose(tabla)

# diferencias divididas vacia
dfinita = np.zeros(shape=(n,n),dtype=float)
tabla = np.concatenate((tabla,dfinita), axis=1)

# Calcula tabla, inicia en columna 3
[n,m] = np.shape(tabla)
diagonal = n-1
j = 3
while (j < m):
    titulo.append('F['+str(j-2)+']')

    i = 0
    paso = j-2 # inicia en 1
    while (i < diagonal):
        denominador = (xi[i+paso]-xi[i])
        numerador = tabla[i+1,j-1]-tabla[i,j-1]
        tabla[i,j] = numerador/denominador
        i = i+1
    diagonal = diagonal - 1
    j = j+1
dDividida = tabla[0,3:]
n = len(dfinita)

# expresion del polinomio
x = sym.Symbol('x')
polinomio = fi[0]
for j in range(1,n,1):
    factor = dDividida[j-1]
    termino = 1
    for k in range(0,j,1):
        termino = termino*(x-xi[k])
    polinomio = polinomio + termino*factor
```

```
#evaluacion numerica
px = sym.lambdify(x,polisimple)

# Puntos para la gr fica
muestras = 101
a = np.min(xi)
b = np.max(xi)
pxi = np.linspace(a,b,muestras)
pfi = px(pxi)

print('Polinomio: ' )
print(polisimple)

# Gr fica
plt.plot(xi,fi,'o', label = 'Puntos',color='black')
plt.plot(pxi,pfi, label = 'Polinomio',color='red')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Polinomio de Newton')
plt.show()
```

## 2.2. Polinomio de Newton

**Problema 18.9**

Use el polinomio de interpolación de Newton para determinar y en $x = 3.5$ con la mayor exactitud posible. Calcule las diferencias divididas finitas como en la figura 18.5 y ordene sus puntos para obtener exactitud óptima y convergencia.

| x | 0 | 1 | 2.5 | 3 | 4.5 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| y | 2 | 5.4375 | 7.3516 | 7.5625 | 8.4453 | 9.1875 | 12 |

1. Gráfica

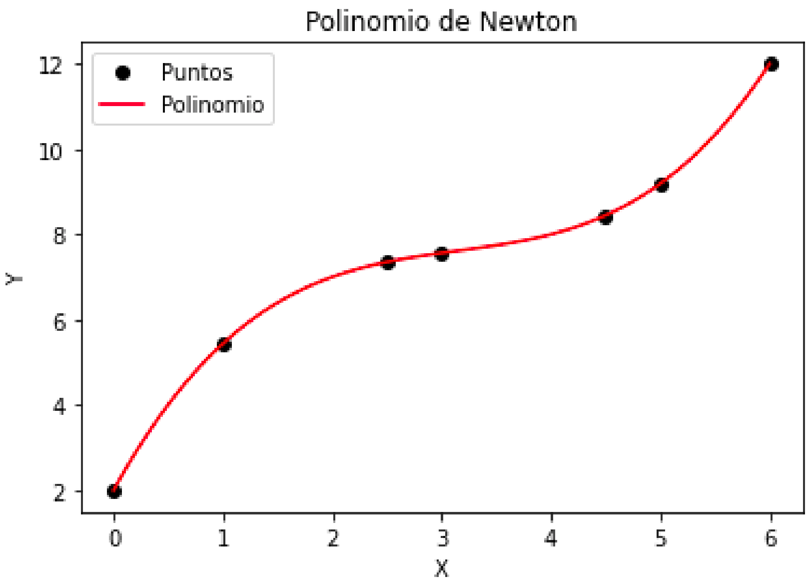


Figura 2.2: Polinomio de Newton

$$7.901e^{-7}x^6 - 1.611e^{-5}x^5 + 0.00012x^4 + 0.145x^3 - 1.374x^2 + 4.666x + 2.0$$

2.

```
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt

#Datos de prueba
xi = np.array([0, 1, 2.5, 3,4.5,5,6])
fi = np.array([2, 5.4375, 7.3516,7.5625, 8.4453,9.1875,12])

# Tabla de Diferencias
n = len(xi)
ki = np.arange(0,n,1)
tabla = np.concatenate(([ki],[xi],[fi]),axis=0)
tabla = np.transpose(tabla)

# diferencias divididas vacia
dfinita = np.zeros(shape=(n,n),dtype=float)
tabla = np.concatenate((tabla,dfinita), axis=1)

# Calcula tabla, inicia en columna 3
[n,m] = np.shape(tabla)
diagonal = n-1
j = 3
while (j < m):
    titulo.append('F['+str(j-2)+']')

    i = 0
    paso = j-2 # inicia en 1
    while (i < diagonal):
        denominador = (xi[i+paso]-xi[i])
        numerador = tabla[i+1,j-1]-tabla[i,j-1]
        tabla[i,j] = numerador/denominador
        i = i+1
    diagonal = diagonal - 1
    j = j+1
dDividida = tabla[0,3:]
n = len(dfinita)

# expresion del polinomio
x = sym.Symbol('x')
polinomio = fi[0]
for j in range(1,n,1):
    factor = dDividida[j-1]
    termino = 1
    for k in range(0,j,1):
        termino = termino*(x-xi[k])
    polinomio = polinomio + termino*factor
```

```
#evaluacion numerica
px = sym.lambdify(x,polisimple)

# Puntos para la gr fica
muestras = 101
a = np.min(xi)
b = np.max(xi)
pxi = np.linspace(a,b,muestras)
pfi = px(pxi)

print('Polinomio: ' )
print(polisimple)

# Gr fica
plt.plot(xi,fi,'o', label = 'Puntos',color='black')
plt.plot(pxi,pfi, label = 'Polinomio',color='red')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Polinomio de Newton')
plt.show()
```

## 2.3. Interpolación inversa

Ejercicio 18.11

Emplee interpolación inversa con el uso de un polinomio de interpolación cúbico y de bisección, para determinar el valor de x que corresponde a f($x$) = 0.23, para los datos tabulados que siguen:

| x | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| y | 0.5 | 0.333 | 0.25 | 0.2 | 0.1667 | 1.1429 |

1. Gráfica

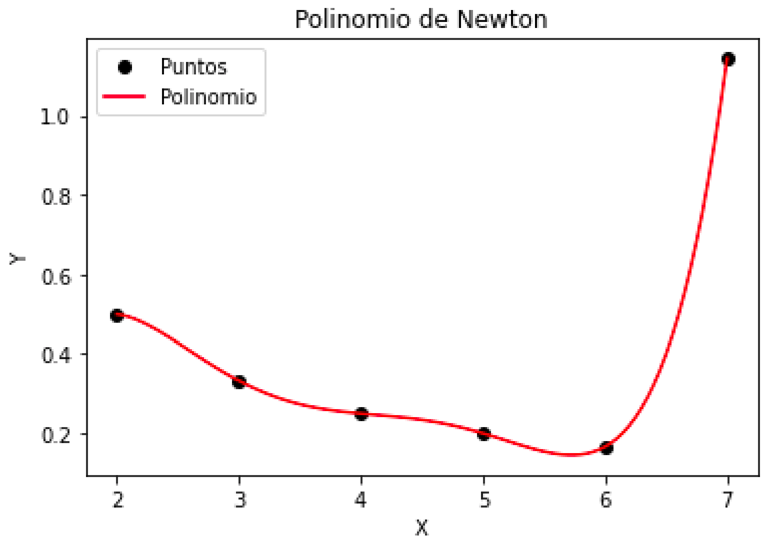


Figura 2.3: Modelo basado en la interpolación inversa

$$0.008x^5 - 0.160x^4 + 1.229x^3 - 4.488x^2 + 7.656x - 4.3829$$

2. Método de Newton

```
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt

#Datos de prueba
xi = np.array([0, 1, 2.5, 3,4.5,5,6])
fi = np.array([2, 5.4375, 7.3516,7.5625, 8.4453,9.1875,12])

# Tabla de Diferencias
n = len(xi)
ki = np.arange(0,n,1)
tabla = np.concatenate(([ki],[xi],[fi]),axis=0)
tabla = np.transpose(tabla)

# diferencias divididas vacia
dfinita = np.zeros(shape=(n,n),dtype=float)
tabla = np.concatenate((tabla,dfinita), axis=1)

# Calcula tabla, inicia en columna 3
[n,m] = np.shape(tabla)
diagonal = n-1
j = 3
while (j < m):
    titulo.append('F['+str(j-2)+']')

    i = 0
    paso = j-2 # inicia en 1
    while (i < diagonal):
        denominador = (xi[i+paso]-xi[i])
        numerador = tabla[i+1,j-1]-tabla[i,j-1]
        tabla[i,j] = numerador/denominador
        i = i+1
    diagonal = diagonal - 1
    j = j+1
dDividida = tabla[0,3:]
n = len(dfinita)

# expresion del polinomio
x = sym.Symbol('x')
polinomio = fi[0]
for j in range(1,n,1):
    factor = dDividida[j-1]
    termino = 1
    for k in range(0,j,1):
        termino = termino*(x-xi[k])
    polinomio = polinomio + termino*factor
```

```
#evaluacion numerica
px = sym.lambdify(x,polisimple)

# Puntos para la gr fica
muestras = 101
a = np.min(xi)
b = np.max(xi)
pxi = np.linspace(a,b,muestras)
pfi = px(pxi)

print('Polinomio: ' )
print(polisimple)

# Gr fica
plt.plot(xi,fi,'o', label = 'Puntos',color='black')
plt.plot(pxi,pfi, label = 'Polinomio',color='red')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Polinomio de Newton')
plt.show()
```

## 2.4. Interpolación de Lagrange

**Ejercicios 18.21**

Desarrolle, depure y pruebe un programa en el lenguaje de alto nivel o macros que elija, para implantar la interpolación de Lagrange. Utilice como base el pseudocódigo de la figura 18.11. Pruébelo con la duplicación del ejemplo 18.7.

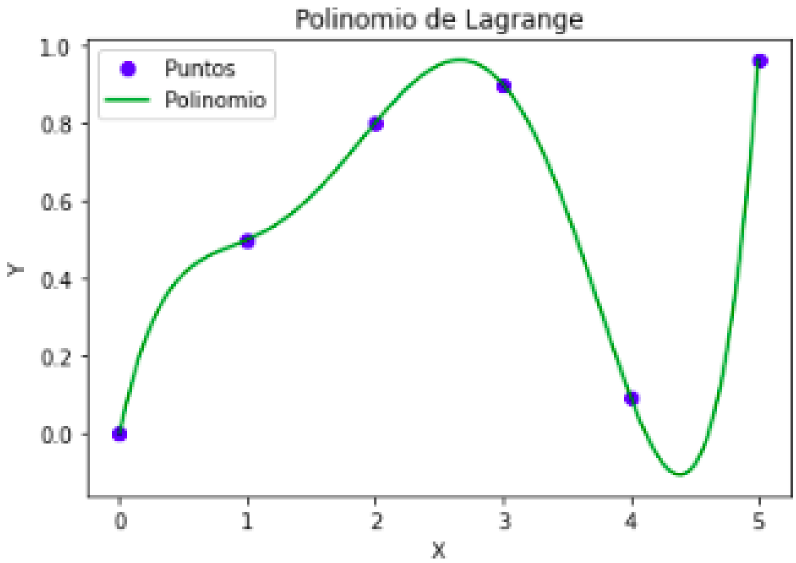


Figura 2.4: Interpolación de Lagrange

$$0.033x^5 - 0.361x^4 + 1.340x^3 - 2.086x^2 + 1.574x$$

```python
import numpy as np
import sympy as sym
import matplotlib.pyplot as plt

#Datos
x = np.array([0, 0.2, 0.3, 0.4])
y = np.array([1, 1.6, 1.7, 2.0])

n=len(x)
x = sym.Symbol('x')
polinomio = 0
divisorL = np.zeros(n, dtype = float)

for i in range(0,n,1):
    numerador = 1
    denominador = 1
    for j in range(0,n,1):
        if (j!=i):
            numerador = numerador*(x-x[j])
            denominador = denominador*(x[i]-x[j])
    terminoLi = numerador/denominador
    polinomio = polinomio + terminoLi*y[i]
    divisorL[i] = denominador
#Evaluacion numerica
px = sym.lambdify(x,polisimple)
#grafica
a = np.min(x)
b = np.max(x)
px = np.linspace(a,b,muestras)
py = px(px)
print('Polinomio de Lagrange: ')
print(polisimple)
# Grafica
plt.plot(x,y,'o', label = 'Puntos',color='blue')
plt.plot(px,py, label = 'Polinomio',color='green')
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Interpolacion Lagrange')
plt.show()
```

# CAPÍTULO 3

# ANEXOS

## 3.1. Anexo A: Evidencia de los código reportados

### 3.1.1. Ejercicio 18.5

```
Polinomio:
-0.481150793650793*x**5 + 8.2470238095238*x**4 -
73115079364*x**3 + 159.836011904762*x**2 - 204.5
98412*x + 91.2857142857141
```

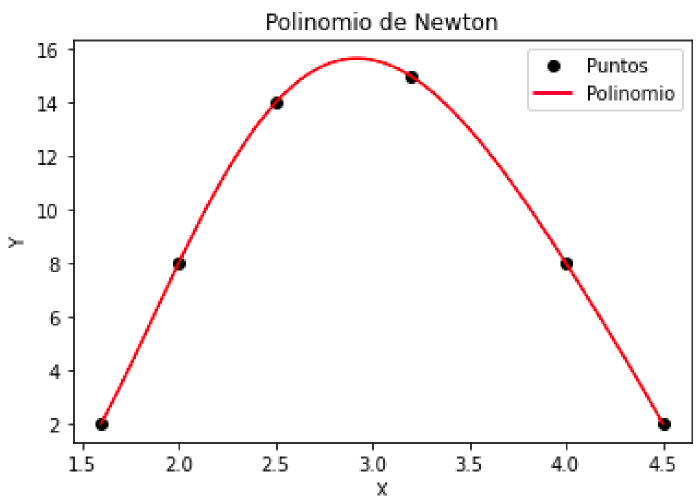


Figura 3.1: Interpolación de Newton

### 3.1.2. Ejercicio 18.9

```
Polinomio:
7.90123456788229e-7*x**6 - 1.61128747795301e-5*x**5
+ 0.000124754850088295*x**4 + 0.145384345679011*x**3
- 1.37427695238095*x**2 + 4.66628317460317*x + 2.0
```

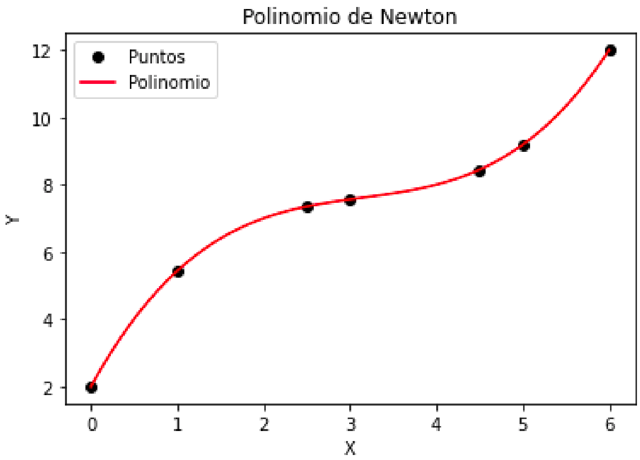


Figura 3.2: Método de Newton

### 3.1.3. Ejercicio 18.11

```
Polinomio:
0.00812*x**5 - 0.160954166666667*x**4 + 1.229858333
33*x**3 - 4.48844583333333*x**2 + 7.65662166666667*
4.3829
```

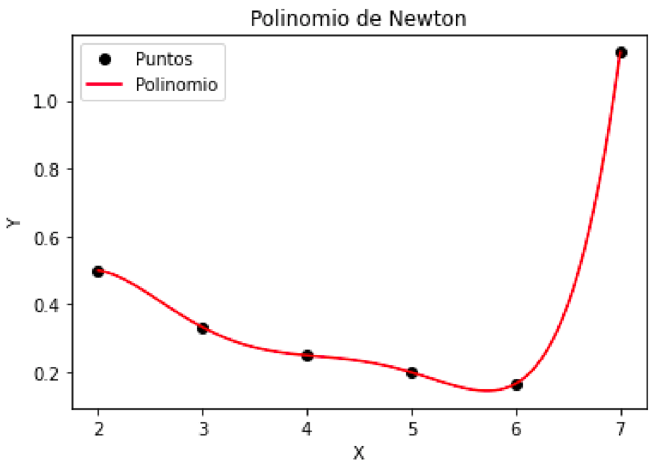


Figura 3.3: Método inverso

### 3.1.4. Ejercicio 18.21

```
Polinomio de Lagrange:
0.0332579166666667*x**5 - 0.3619909333333
33*x**4 + 1.34049768333333*x**3 - 2.08642
526666667*x**2 + 1.5746606*x
```

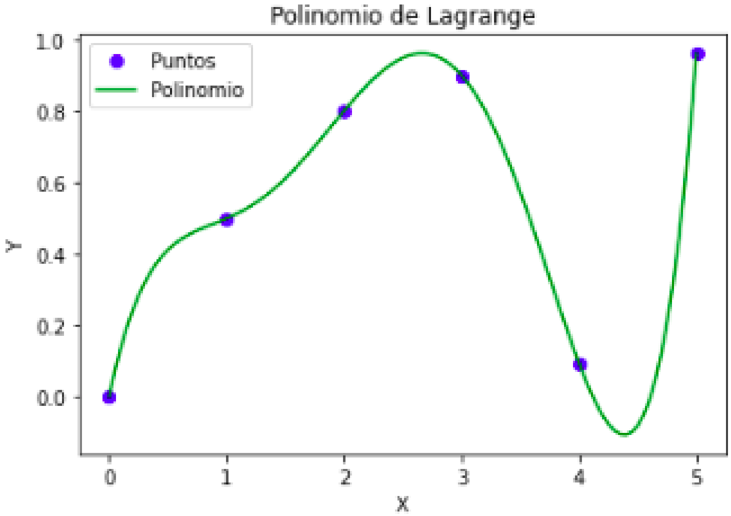


Figura 3.4: Método de Lagrange de prueba

CAPÍTULO 4

# BIBLIOGRAFÍA


a) Chapra, S. C.,  Canale, R. P. (2011). *Numerical methods for engineers* (Vol. 1221). New York: Mcgraw-hill.