Interpolación polinómica

Importamos los módulos `matplotlib.pyplot` , `numpy` y las funciones de `numpy` para polinomios.

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.polynomial.polynomial as pol
```

Polinomio interpolante de Lagrange

Tenemos una serie de puntos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ y queremos encontrar un polinomio que pase por todos ellos.

Sea el conjunto de puntos:

$$C = \{(-1, 1), (0, 3), (2, 4), (3, 3), (5, 1)\}$$

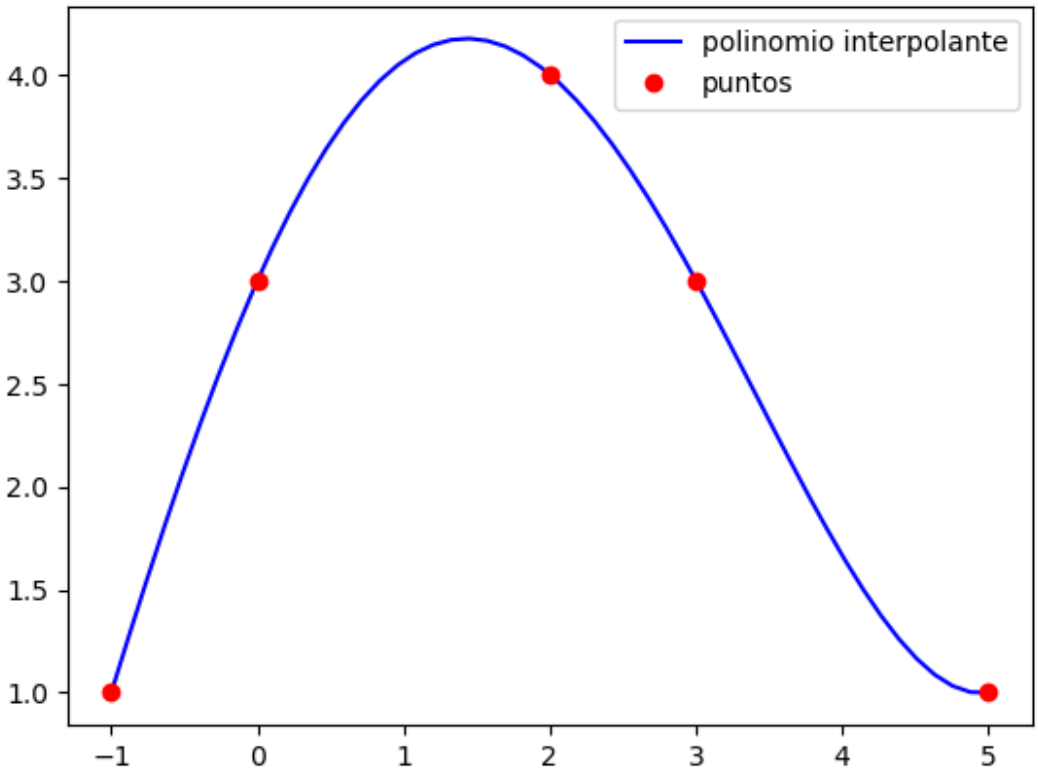
Los puntos los almacenaremos en matrices numpy (numpy arrays)

```
x = np.array([-1., 0, 2, 3, 5])
y = np.array([ 1., 3, 4, 3, 1])
```

Y dibujamos los puntos y el polinomio de Lagrange, que es el polinomio que pasa por todos ellos

```
xp = np.linspace(min(x),max(x))
p  = pol.polyfit(x,y,len(x)-1)
yp = pol.polyval(xp,p)

plt.figure()
plt.plot(xp,yp,'b-', label = 'polinomio interpolante')
plt.plot( x, y,'ro', label = 'puntos')
plt.legend()
plt.show()
```



Interpolación mediante los polinomios fundamentales de Lagrange

Para cada $k = 0, 1, \ldots, n$, existe un único polinomio ℓ_k de grado $\leq n$ tal que $\ell_k\left(x_j\right) = \delta_{kj}$ (uno si $k = j$, cero en caso contrario):

$$\ell_k\left(z\right) = \frac{\left(z - x_0\right)}{\left(x_k - x_0\right)} \cdots \frac{\left(z - x_{k-1}\right)}{\left(x_k - x_{k-1}\right)} \cdot \frac{\left(z - x_{k+1}\right)}{\left(x_k - x_{k+1}\right)} \cdots \frac{\left(z - x_n\right)}{\left(x_k - x_n\right)}$$

Por ejemplo, con nuestros 5 nodos, serían

$$\ell_0\left(z\right) = \left| ^{x_0} \frac{\left(z - x_1\right)}{\left(x_0 - x_1\right)} \cdot \frac{\left(z - x_2\right)}{\left(x_0 - x_2\right)} \cdot \frac{\left(z - x_3\right)}{\left(x_0 - x_3\right)} \cdot \frac{\left(z - x_4\right)}{\left(x_0 - x_4\right)}$$

$$\ell_1\left(z\right) = \frac{\left(z - x_0\right)}{\left(x_1 - x_0\right)} \left| ^{x_1} \frac{\left(z - x_2\right)}{\left(x_1 - x_2\right)} \cdot \frac{\left(z - x_3\right)}{\left(x_1 - x_3\right)} \cdot \frac{\left(z - x_4\right)}{\left(x_1 - x_4\right)}\right.$$

$$\ell_2\left(z\right) = \frac{\left(z - x_0\right)}{\left(x_2 - x_0\right)} \cdot \frac{\left(z - x_1\right)}{\left(x_2 - x_1\right)} \left| ^{x_2} \frac{\left(z - x_3\right)}{\left(x_2 - x_3\right)} \cdot \frac{\left(z - x_4\right)}{\left(x_2 - x_4\right)}\right.$$

$$\ell_3\left(z\right) = \frac{\left(z - x_0\right)}{\left(x_3 - x_0\right)} \cdot \frac{\left(z - x_1\right)}{\left(x_3 - x_1\right)} \cdot \frac{\left(z - x_2\right)}{\left(x_3 - x_2\right)} \left| ^{x_3} \frac{\left(z - x_4\right)}{\left(x_3 - x_4\right)}\right.$$

$$\ell_4\left(z\right) = \frac{\left(z - x_0\right)}{\left(x_4 - x_0\right)} \cdot \frac{\left(z - x_1\right)}{\left(x_4 - x_1\right)} \cdot \frac{\left(z - x_2\right)}{\left(x_4 - x_2\right)} \cdot \frac{\left(z - x_3\right)}{\left(x_4 - x_3\right)} \Bigg|^{x_4}$$

Los polinomios $\ell_0, \ell_1, \ldots, \ell_n$ son los **polinomios fundamentales de Lagrange**.

El polinomio interpolante de Lagrange en los puntos $x_{0,}x_1, \ldots, x_n$ relativo a los valores $y_0, y_1, \ldots, y_n$ es

$$P_n\left(z\right) = y_0\ell_0\left(z\right) + y_1\ell_1\left(z\right) + \cdots + y_n\ell_n\left(z\right)$$

Ejercicio 1

Escribir un programa que calcule los polinomios fundamentales de Lagrange. Para ello, crear una función `lagrange_fund(k,x,z)` que tendrá:

- ***Argumentos de entrada:*** el índice `k` del polinomio Fundamental, los nodos `x` del problema de interpolación y un punto (o puntos, vector) donde vamos a calcular los valores del polinomio, `z` .
- ***Argumentos de salida:*** el valor del $k-$ésimo polinomio fundamental correspondiente a los nodos `x` en el punto (o puntos, vector) `z` .

Dibujar todos los polinomios Fundamental de Lagrange.

Usar los nodos:

```
x = np.array([-1., 0, 2, 3, 5])
```

Nota:

- Puedes empezar haciendo la función para un punto z y luego modificarla para que admita un vector. Por ejemplo, la primera versión funcionaría

```
k = 2
z = 1.3
yz = lagrange_fundamental(k,x,z)
print(yz)
```

```
1.0448388888888889
```

> y ahora sabemos que $\ell_2\left(1.3\right) = 1.0448388888888889$. La segunda versión haría

```
k = 2
z = np.array([1.3, 2.1, 3.2])
yz = lagrange_fundamental(k,x,z)
print(yz)
```

```
[ 1.04483889  0.94395    -0.2688    ]
```

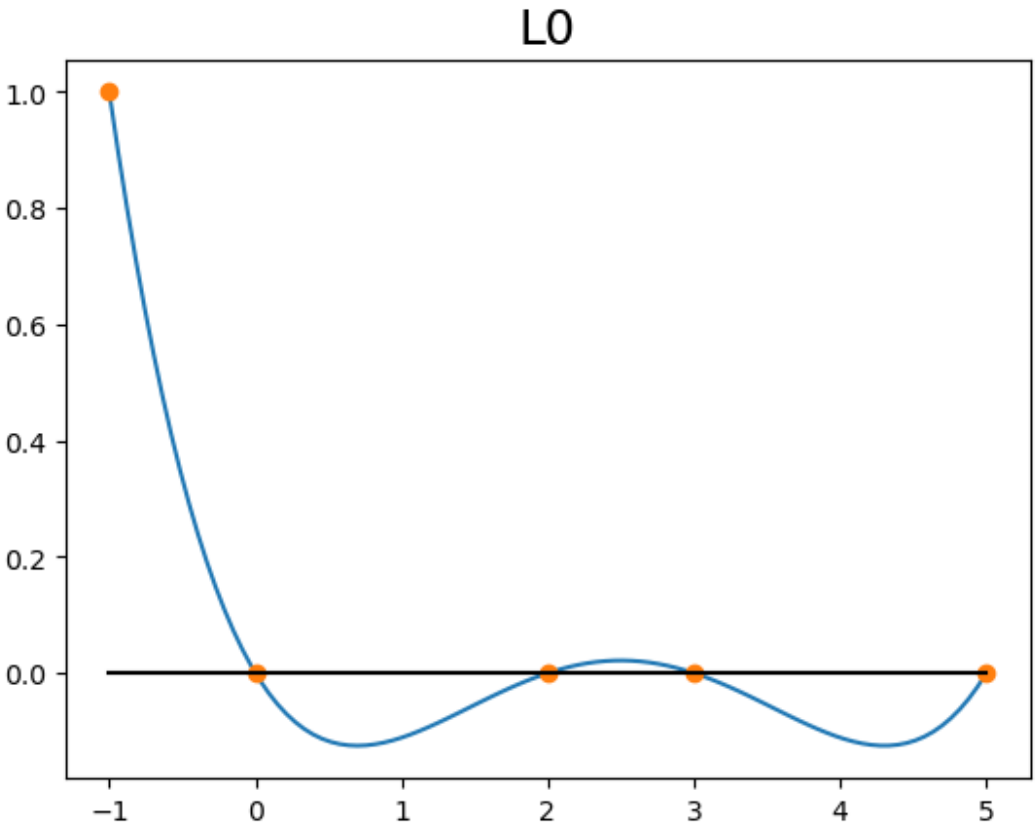
> y $\ell_2\left(\left[1.3,\, 2.1,\, 3.2\right]\right) = \left[1.04483889,\, 0.94395,\, -0.2688\right]$.

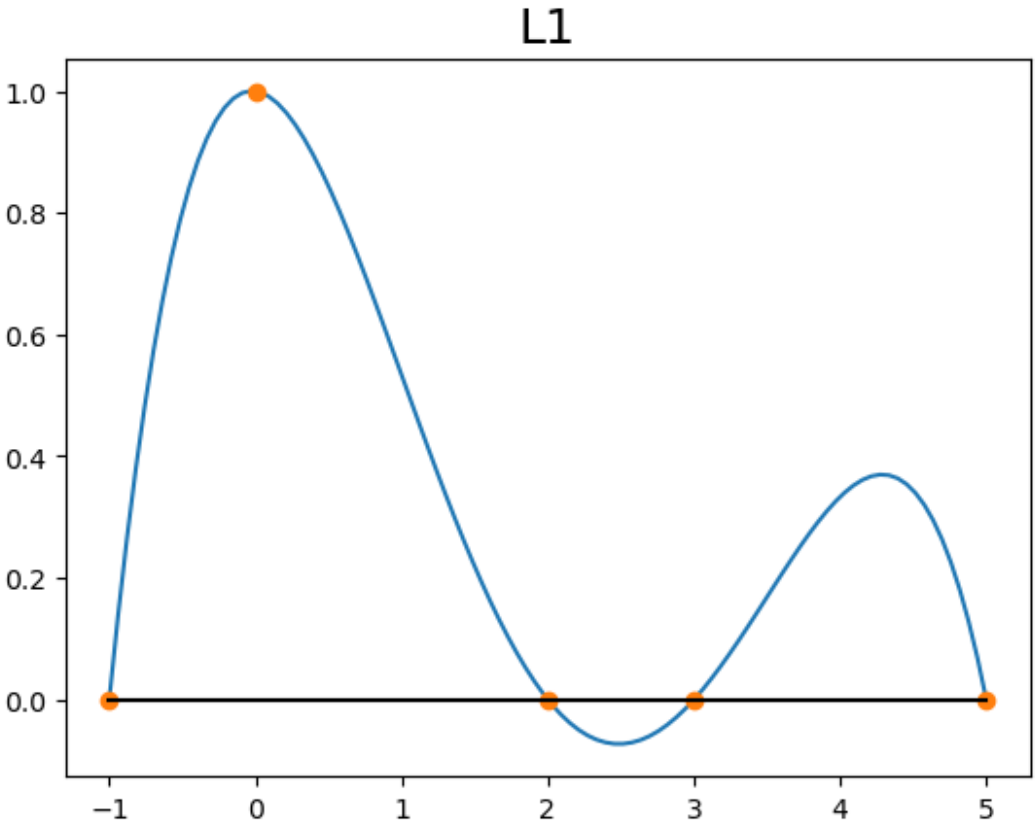
- Para dibujar los puntos, se puede tener en cuenta que sus xs están contenidas en el vector de nodos `x` y sus correspondientes ys se pueden obtener como las filas de la matriz identidad:

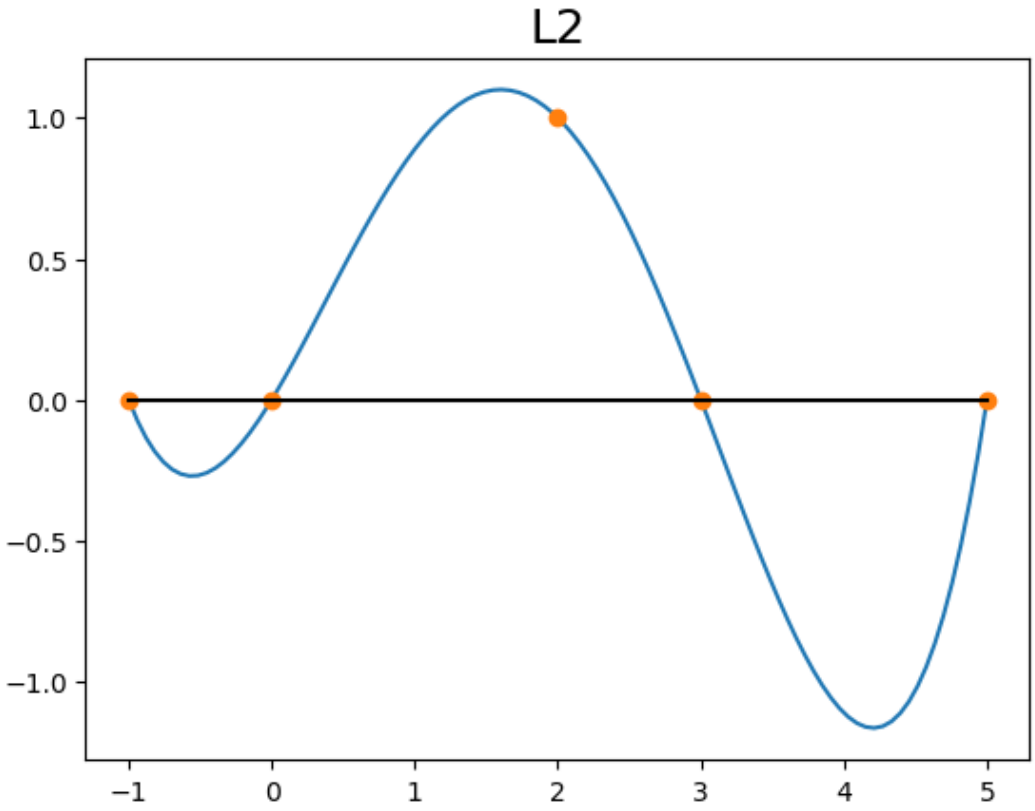
```
print(np.eye(len(x)))
```

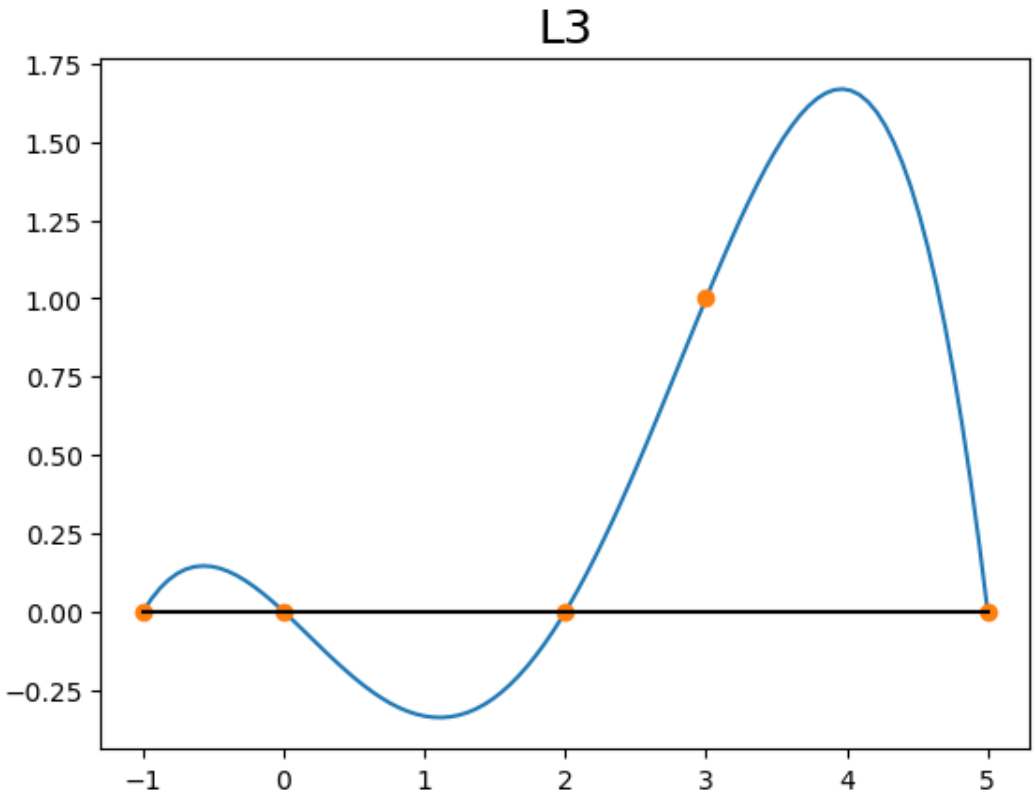
```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

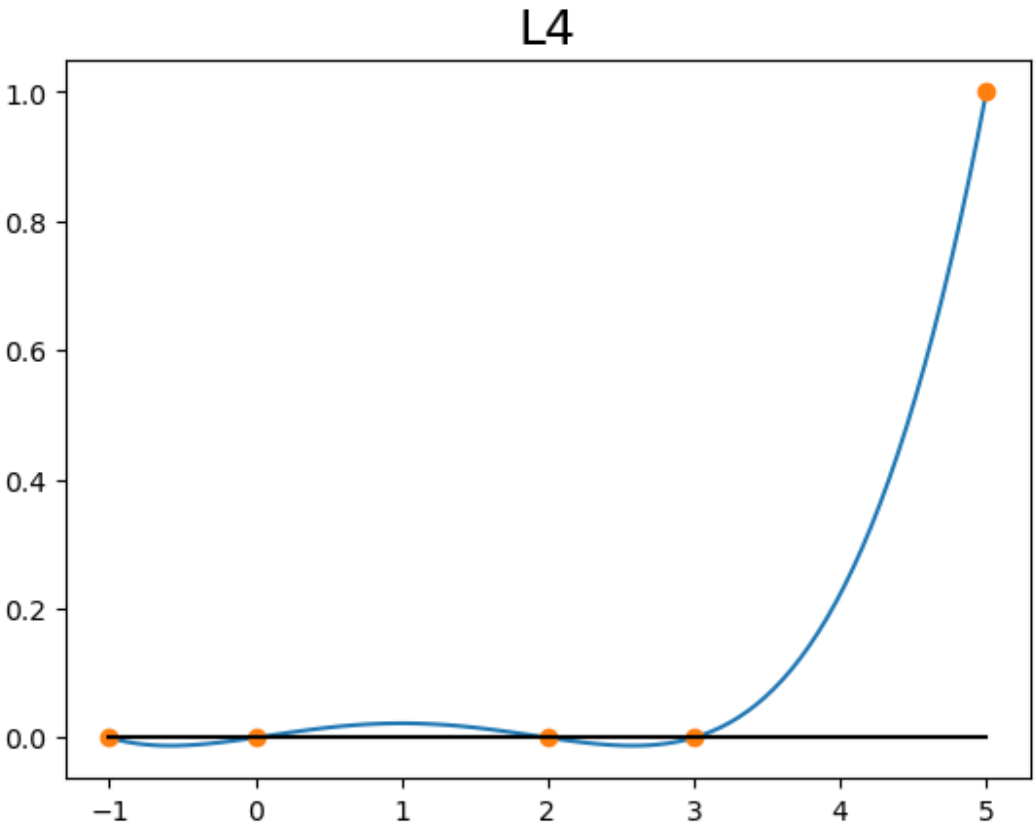
```
%run Ejercicio1.py
```


L0
1.0
0.8
0.6
0.4
0.2
0.0
−1
0
1
2
3
4
5


L1
1.0
0.8
0.6
0.4
0.2
0.0
−1
0
1
2
3
4
5


L2
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2
3
4
5


L3
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−1
0
1
2
3
4
5



Recordamos que **el polinomio interpolante de Lagrange** en los puntos $x_{0,}x_1, \ldots, x_n$ relativo a los valores $y_0, y_1, \ldots, y_n$ es

$$P_n(z) = y_0\ell_0(z) + y_1\ell_1(z) + \cdots + y_n\ell_n(z)$$

Ejercicio 2

Escribir un programa que calcule el polinomio interpolante de Lagrange utilizando los polinomios fundamentales de Lagrange. Para ello, usar la función **`lagrange_fund(k,x,z)`** y una funcion **`polinomio_lagrange(x,y,z)`** que tendrá:

- ***Argumentos de entrada:*** los nodos **`x`**, **`y`** del problema y un punto (o vector) a evaluar **`z`**.
- ***Argumentos de salida:*** el valor del polinomio interpolante de Lagrange en **`z`**.

Dibujar el polinomio interpolante en el intervalo $[-1, 5]$.

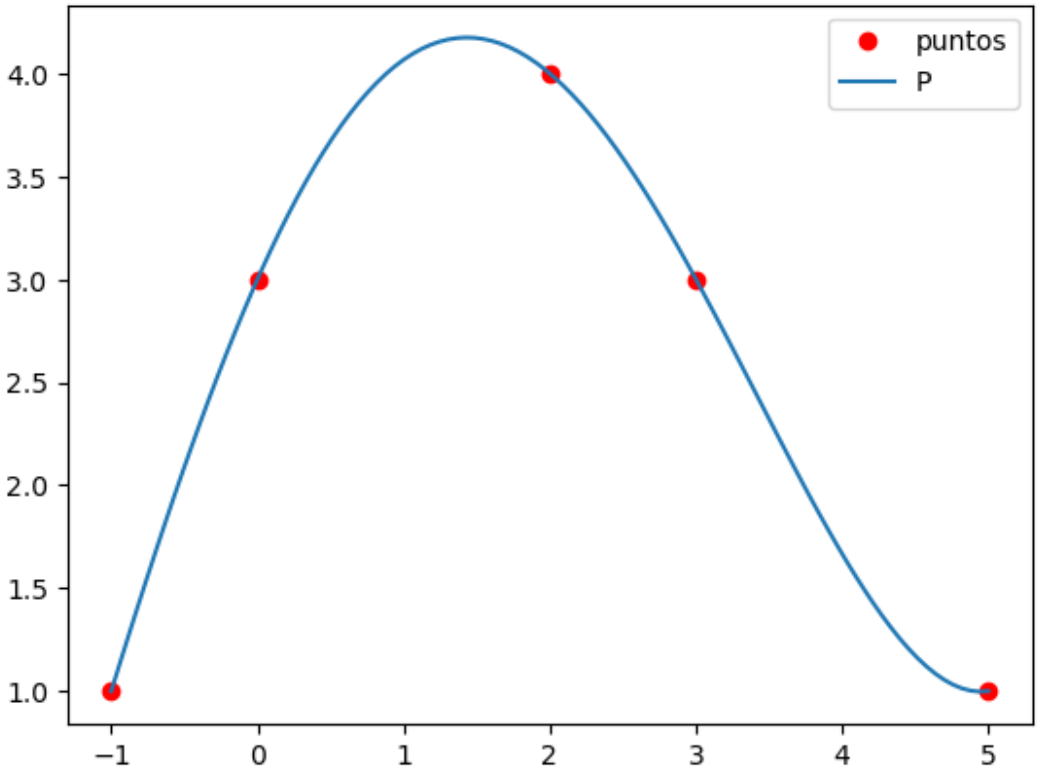
Usar los nodos:

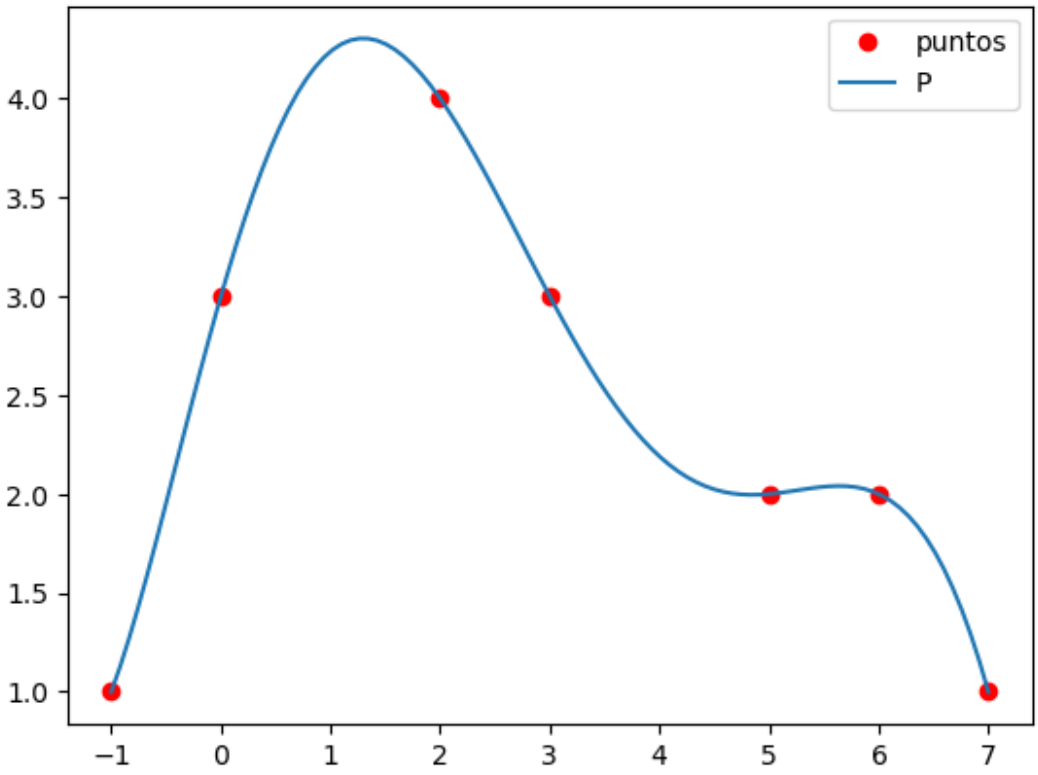
```
x = np.array([-1., 0, 2, 3, 5])
y = np.array([ 1., 3, 4, 3, 1])
```

Comprobar que también funciona con los nodos en el intervalo $[-1, 7]$:

```
x1 = np.array([-1., 0, 2, 3, 5, 6, 7])
y1 = np.array([ 1., 3, 4, 3, 2, 2, 1])
```

```
%run Ejercicio2.py
```





El inconveniente de este método es que si queremos incorporar un nodo nuevo tenemos que rehacer todos los cálculos.

Nodos de Chebyshev

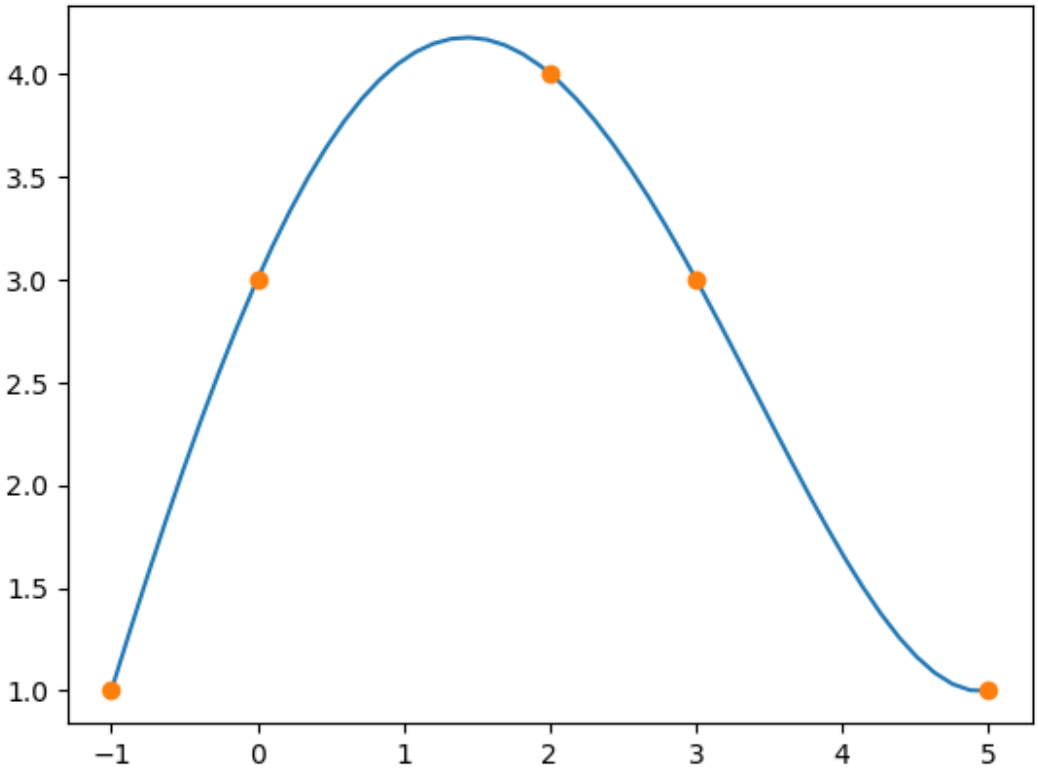
El polinomio de interpolación se puede obtener con la función de numpy `polyfit` escogiendo como grado del polinomio el número de puntos menos uno. Nos da los coeficientes del polinomio interpolante en un vector, empezando por el de mayor grado:

```
x = np.array([-1., 0, 2, 3, 5])
y = np.array([ 1., 3, 4, 3, 1])
p = pol.polyfit(x,y,len(x)-1)  # coeficientes del polinomio

xp = np.linspace(min(x),max(x))
yp = pol.polyval(xp,p)         # valor del polinomio en los puntos de xp
```

Representamos el polinomio

```
plt.figure()
plt.plot(xp, yp, '-',x, y, 'o')
plt.show()
```



Ejercicio 3

Escribir una función **`chebyshev(f,a,b,n)`** que interpole la función **`f`** en el intervalo **`[a,b]`** utilizando **`n`** nodos:

- Utilizando nodos equiespaciados que incluyan los extremos del intervalo.
- Utilizando nodos de Chebyshev, que se definen en el intervalo $[-1,1]$ según la fórmula:

$$x_i = \cos\frac{(2i-1)\,\pi}{2n} \quad i = 1, 2, \ldots, n$$

El número de nodos es $n = 11$, el itervalo $[a,b] = [-1,1]$ y la función f

$$f_1(x) = \frac{1}{1+25x^2}$$

Comprobar que también funciona con 15 nodos y con la función:

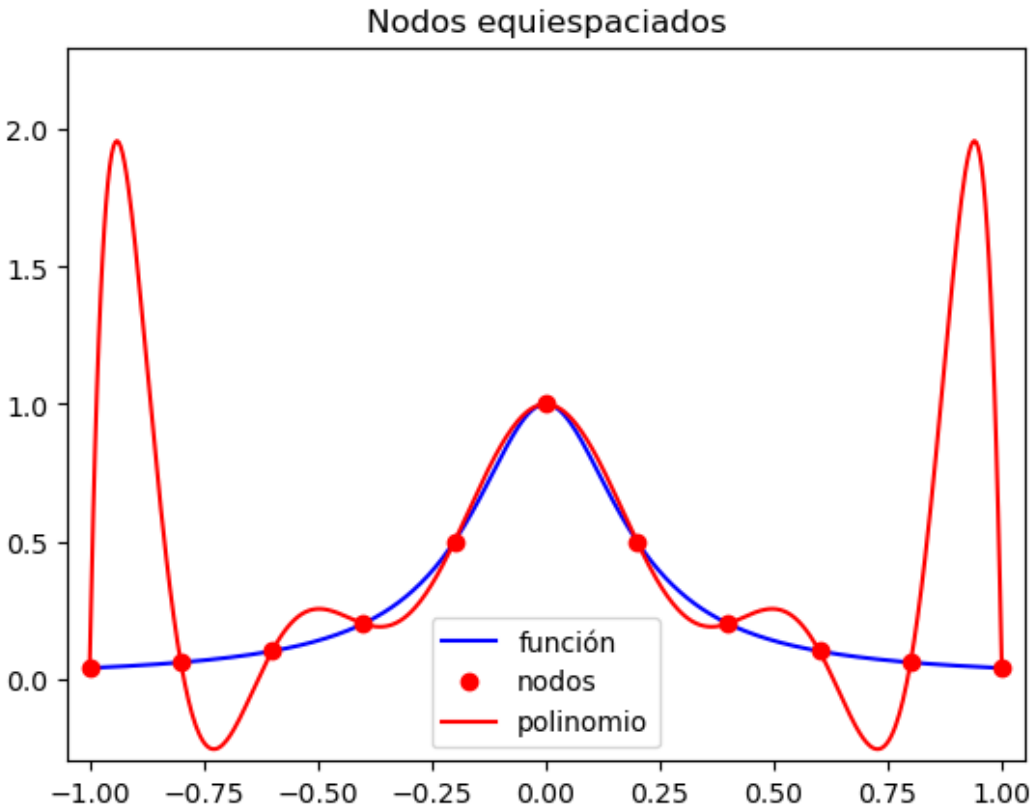
$$f_2(x) = e^{-20x^2}$$

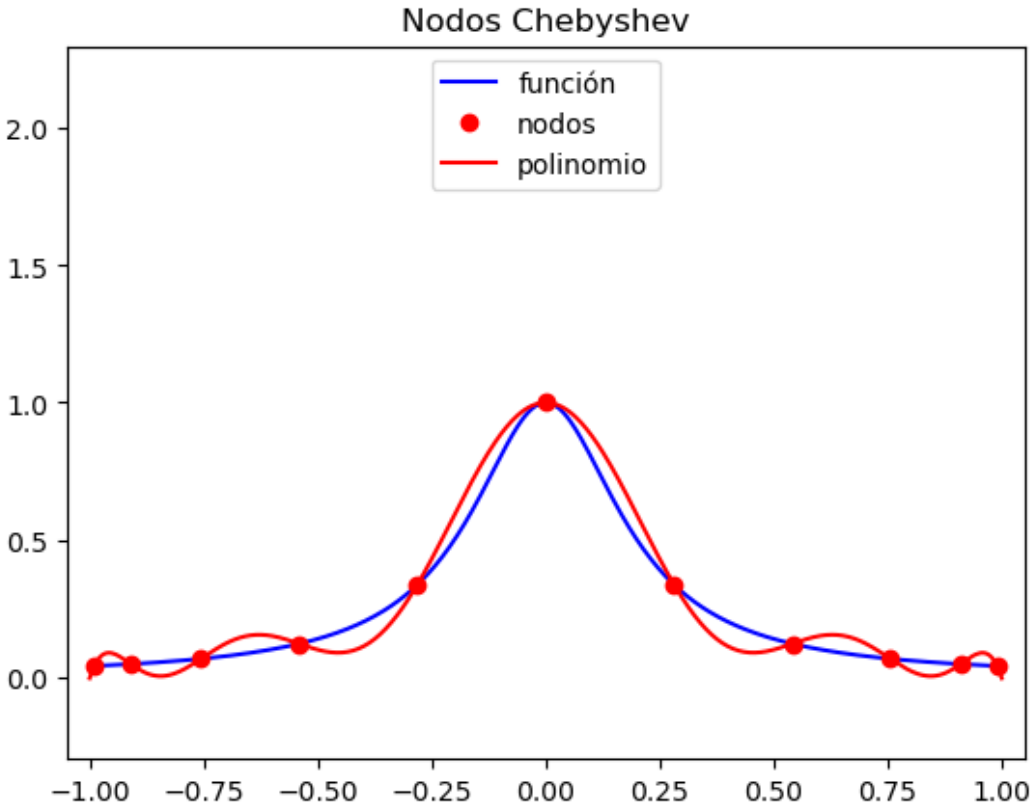
Seguir los pasos:

- **Construir los nodos:** los n nodos se almacenarán en un vector **`x`** de n elementos en ambos casos. Para los nodos equiespaciados se pueden usar **`np.linspace`** o **`np.arange`** . Las **`y`** correspondientes se obtienen utilizando **`f1`** y **`f2`** .
- **Construir los polinomios interpolantes:** Usar las funciones **`pol.polyfit`** y **`pol.polyval`** para calcular los polinomios interpolantes.
- **Dibujar la gráfica:** En ambos casos dibujar los puntos, la función y el polinomio interpolador. Usar **`plt.axis([-1.05, 1.05, -0.3, 2.3])`** para que dibuje todas las gráficas en el rectángulo $[-1.05, 1.05] \times [-0.3, 2.3]$. Usar más de 50 puntos (valor por defecto) en **`np.linspace`** porque el polinomio interpolante oscila mucho.

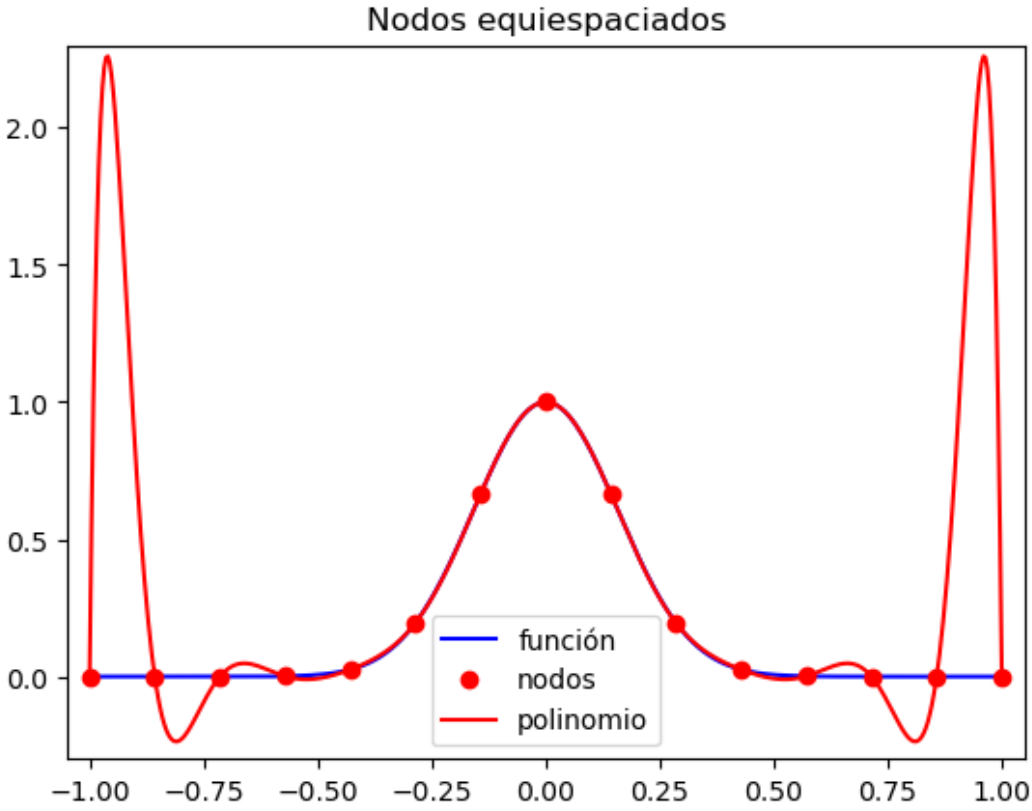
```
%run Ejercicio3.py
```

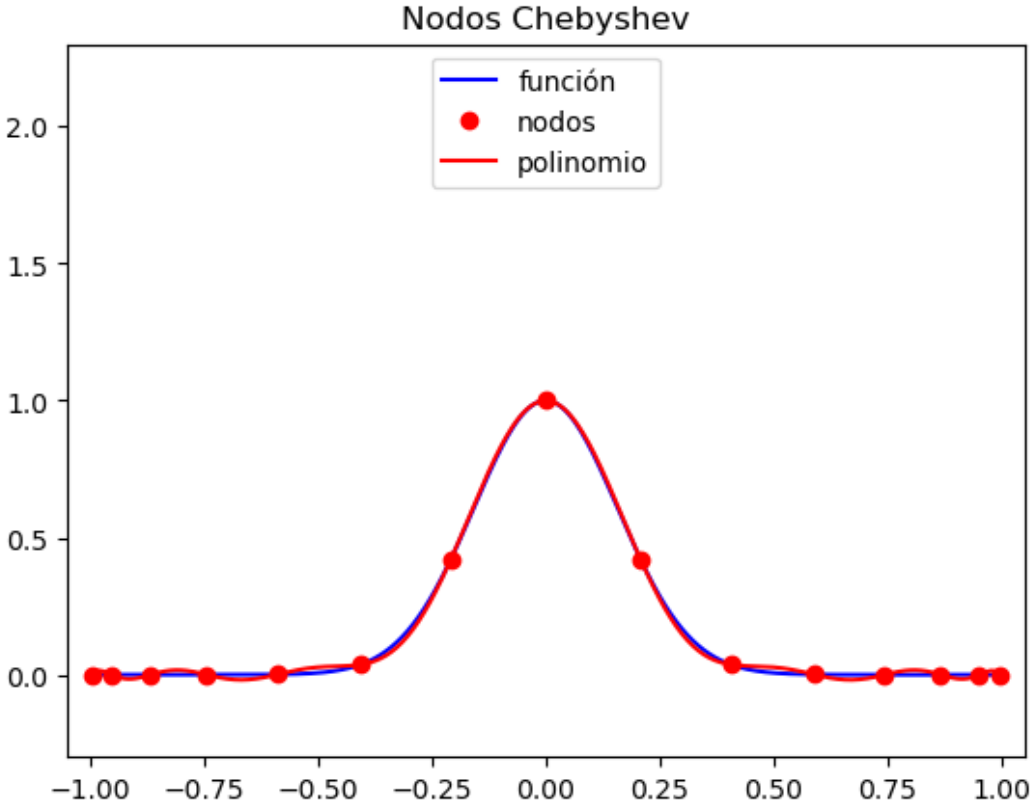
```
------------  Función f1  ------------
```






```
------------  Función f2  ------------
```





Ejercicios propuestos

Interpolación mediante diferencias divididas

Fórmula de Newton

El polinomio de interpolación de Lagrange en los puntos $x_{0,}x_1,\ldots,x_n$ relativo a los valores $y_0,y_1,\ldots,y_n$ es

$$P_n\left(z\right)=\left[y_0\right]+\left[y_0,y_1\right]\left(z-x_0\right)+\left[y_0,y_1,y_2\right]\left(z-x_0\right)\left(z-x_1\right)+\cdots+\left[y_0,y_1,\ldots,y_n\right]\left(z-$$

Tabla de diferencias divididas

Los coeficientes del polinomio anterior son la primera fila de la tabla:

$$\begin{array}{llllll} x_0 & y_0 & \left[y_0,y_1\right]=\frac{y_0-y_1}{x_0-x_1} & \left[y_0,y_1,y_2\right]=\frac{\left[y_0,y_1\right]-\left[y_1,y_2\right]}{x_0-x_2} & \cdots & \left[y_0,y_1,\ldots,y_n\right]=\frac{\left[y_0,..}{} \\ x_1 & y_1 & \left[y_1,y_2\right]=\frac{y_1-y_2}{x_1-x_2} & \left[y_1,y_2,y_3\right]=\frac{\left[y_1,y_2\right]-\left[y_2,y_3\right]}{x_1-x_3} & \cdots & \\ x_2 & y_2 & \left[y_2,y_3\right]=\frac{y_2-y_3}{x_2-x_3} & \left[y_2,y_3,y_4\right]=\frac{\left[y_2,y_3\right]-\left[y_3,y_4\right]}{x_2-x_4} & \cdots & \\ \ldots & \ldots & \ldots & \ldots & \ldots & \\ x_{n-1} & y_{n-1} & \left[y_{n-1},y_n\right]=\frac{y_{n-1}-y_n}{x_{n-1}-x_n} & & & \\ x_n & y_n & & & & \end{array}$$

Ejercicio 4

Escribir un programa que calcule el polinomio interpolante de Lagrange utilizando las diferencias divididas. Para ello, crear dos funciones. La primera será **`dif_div(x,y)`** que tendrá:

- ***Argumentos de entrada:*** los nodos **`x`**, **`y`** del problema de interpolación.
- ***Argumentos de salida:*** una matriz que contenga las diferencias divididas.

y una funcion **`polinomio_newton(x,y,z)`** que tendrá:

- ***Argumentos de entrada:*** los nodos **`x`**, **`y`** del problema y un punto (o vector) a evaluar **`z`**.
- ***Argumentos de salida:*** el valor del polinomio interpolante de Lagrange en **`z`**.

Escribir la matriz de diferencias divididas. Dibujar el polinomio interpolante en el intervalo de interpolación.

Usar los nodos:

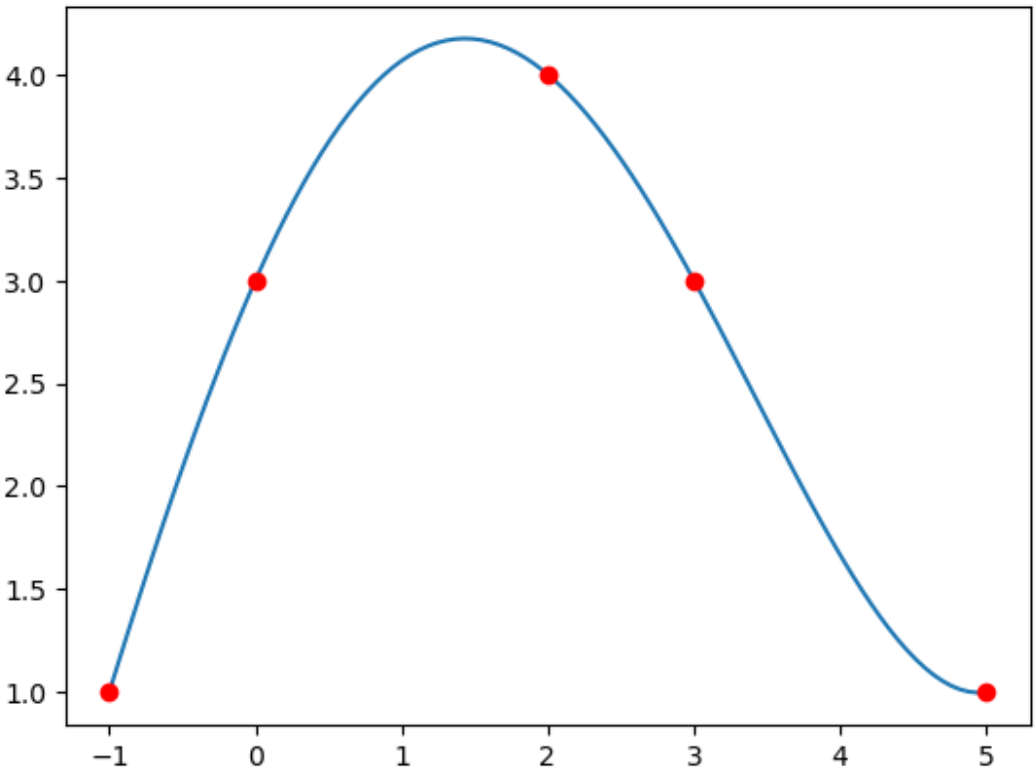
```
x = np.array([-1., 0, 2, 3, 5])
y = np.array([ 1., 3, 4, 3, 1])
```

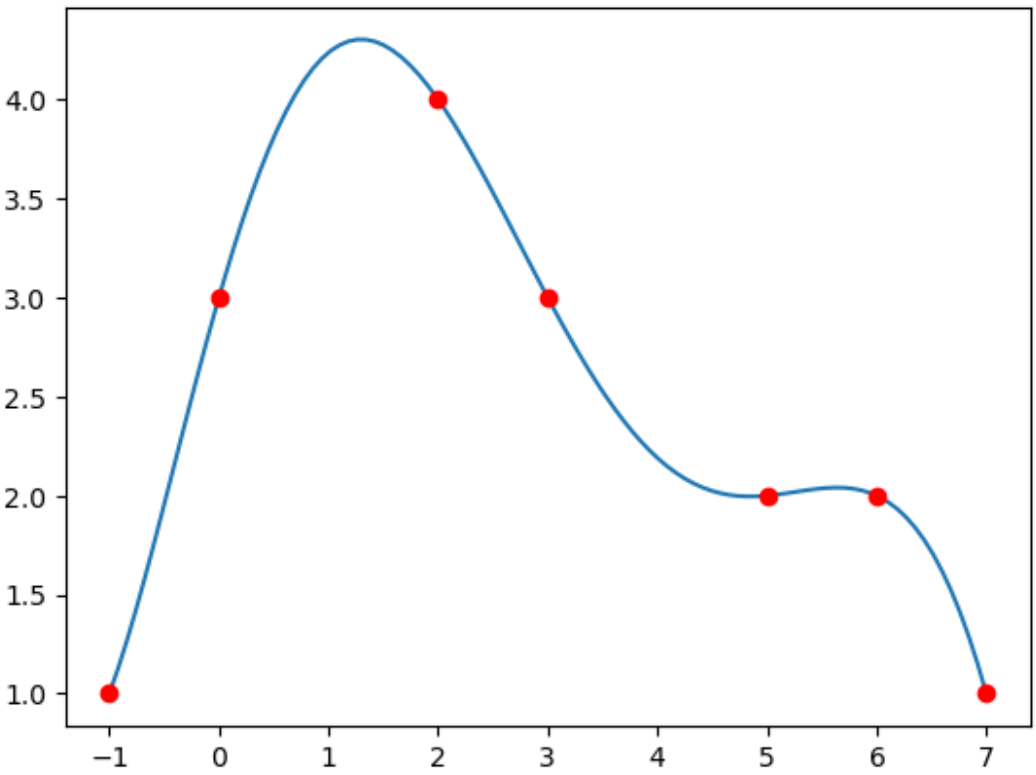
Comprobar que también funciona con los nodos:

```
x1 = np.array([-1., 0, 2, 3, 5, 6, 7])
y1 = np.array([ 1., 3, 4, 3, 2, 2, 1])
```

```
%run Ejercicio4.py
```

```
TABLA DE DIFERENCIAS DIVIDIDAS
[[-1.    1.    2.   -0.5   0.    0.02]
 [ 0.    3.    0.5  -0.5   0.1   0.  ]
 [ 2.    4.   -1.    0.    0.    0.  ]
 [ 3.    3.   -1.    0.    0.    0.  ]
 [ 5.    1.    0.    0.    0.    0.  ]]
```





Interpolación polinomial a trozos con funciones de python

Podemos obtener la **interpolación a trozos** con funciones de scipy.

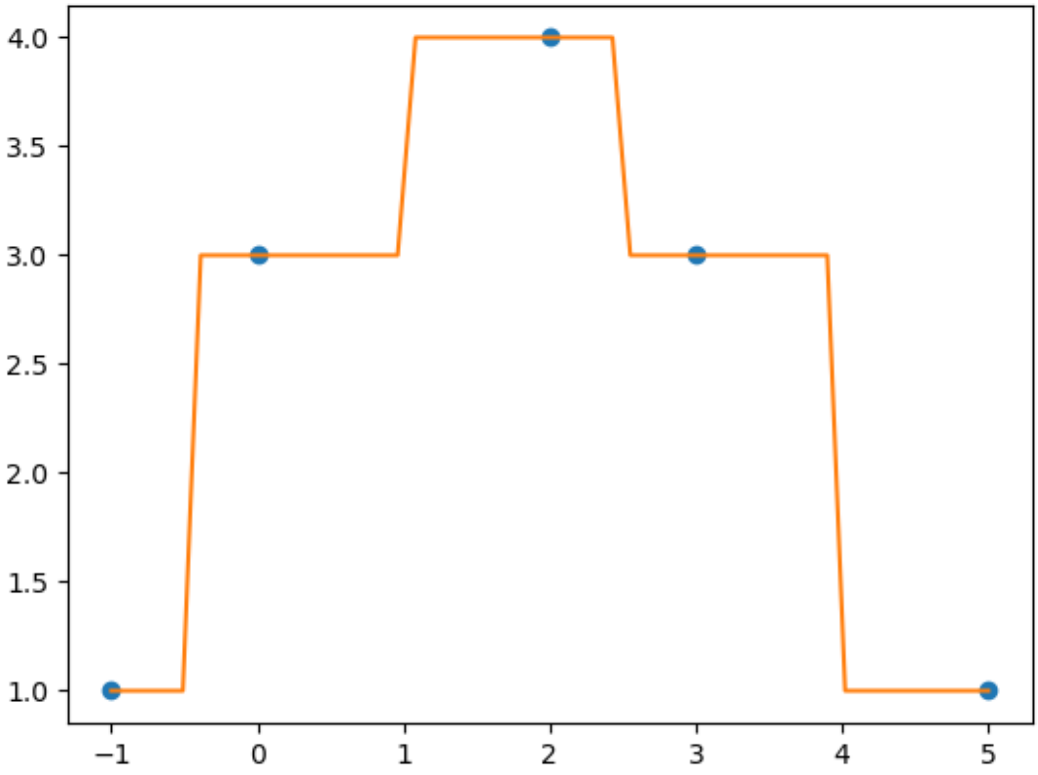
```
from scipy.interpolate import interp1d, CubicSpline

f1 = interp1d(x, y, kind='nearest')
f2 = interp1d(x, y, kind='linear')
f3 = CubicSpline(x, y, bc_type='natural')

xp = np.linspace(min(x),max(x))
```

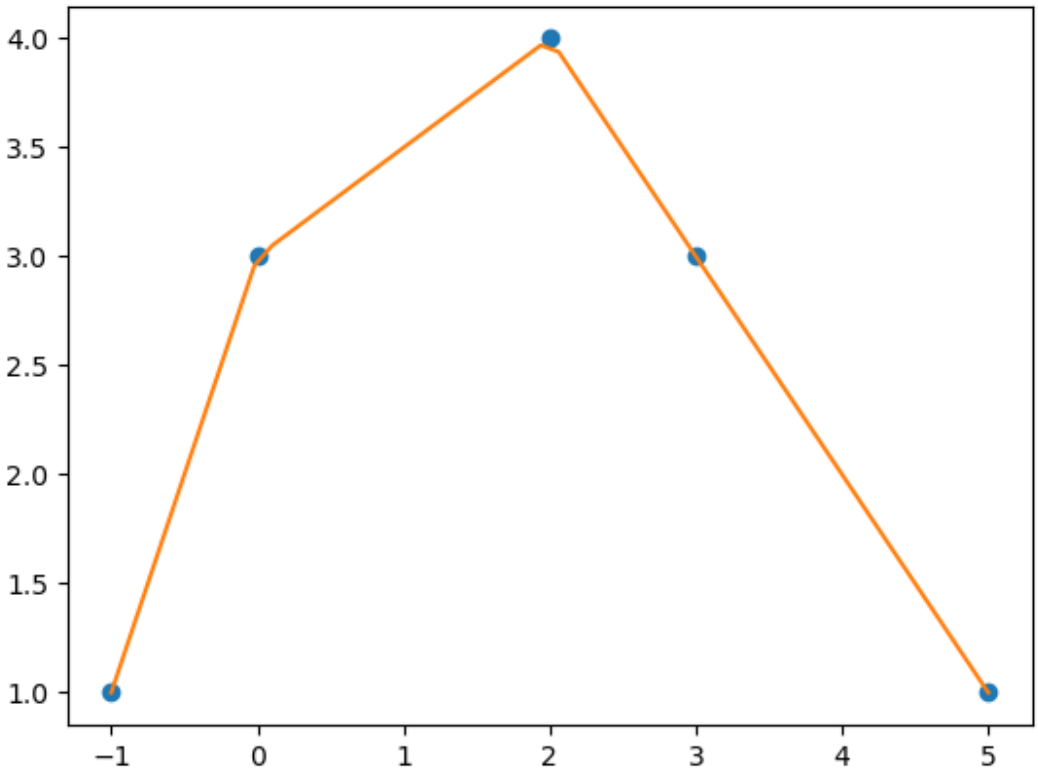
De grado cero

```
plt.figure()
plt.plot(x, y, 'o', xp, f1(xp), '-')
plt.show()
```



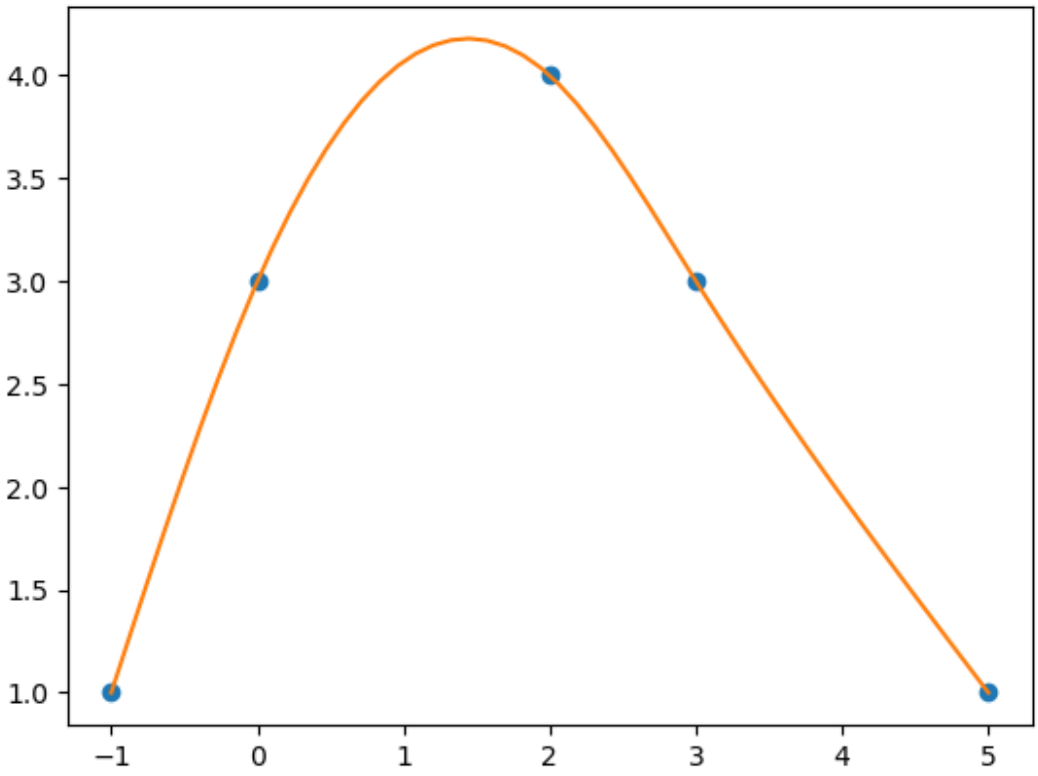
De grado uno o lineal

```
plt.figure()
plt.plot(x, y, 'o', xp, f2(xp), '-')
plt.show()
```



De grado tres, un spline cúbico natural

```
plt.figure()
plt.plot(x, y, 'o', xp, f3(xp), '-')
plt.show()
```



Ejercicio 5

Realizar una programa que, para la función $f(x) = \cos x$ en el intervalo $[0, 10]$ tomando como nodos $x = 0, 1, \ldots, 10$ dibuje los polinomios de interpolación:

- Lineal a trozos.
- Con splines cúbicas.

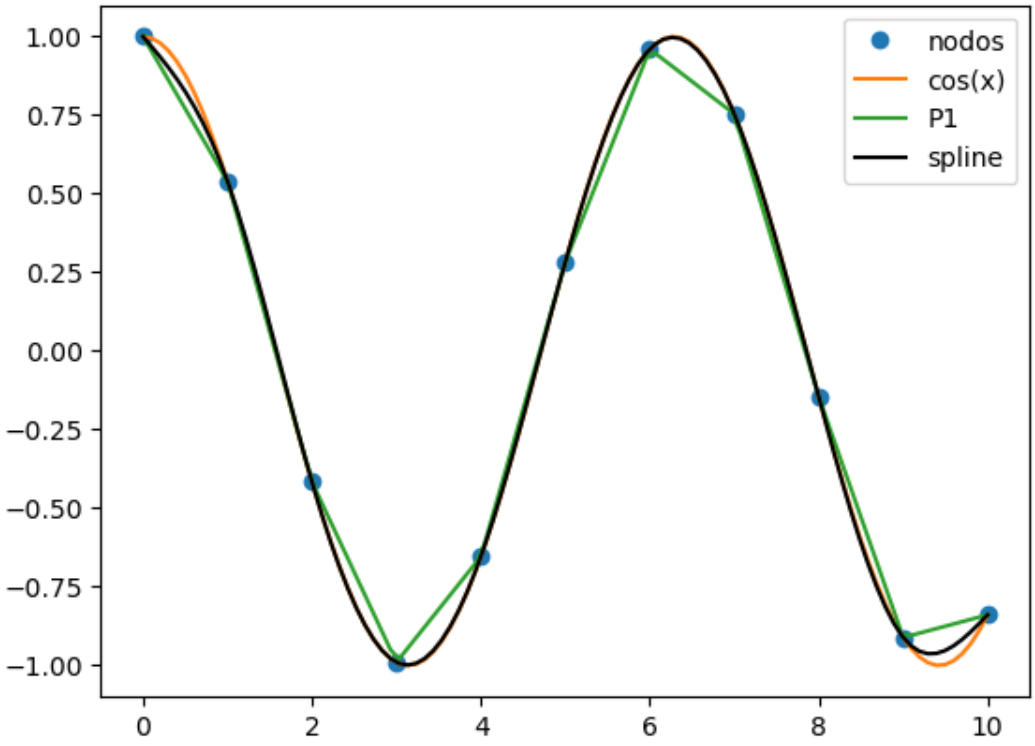
y calcule, para los puntos

```
xp = np.linspace(0,10,100)
```

el error absoluto en cada caso. Para evaluar el error utilizar la función `Ea = np.linalg.norm(yp-f(xp))` .

$$E_a = \|P(x_p) - f(x_p)\|$$

```
%run Ejercicio5.py
```




```
Error interpolación lineal a trozos = 0.64506
Error interpolación con splines     = 0.16625
```

Solución con la matriz de Vandermonde

Para los puntos

```
x = np.array([-1., 0, 2, 3, 5])
y = np.array([ 1., 3, 4, 3, 1])
```

si queremos calcular el polinomio que pasa por todos ellos, como tenemos 5 puntos, podemos plantear 5 ecuaciones y necesitamos 5 incógnitas que serán los coeficientes del polinomio. Por lo tanto:

$$P_4(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

es decir a_0, a_1, a_2, a_3, a_4, que son 5 incógnitas y el polinomio es, en principio, de grado 4. En general, para los puntos $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ el polinomio interpolante será de grado $\leq n$.

Las ecuaciones serían:

$$\begin{aligned}
P_4(x_0) = y_0 \qquad & a_0 + a_1 x_0 + a_2 x_0^2 + a_3 x_0^3 + a_4 x_0^4 = y_0 \\
P_4(x_1) = y_1 \qquad & a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + a_4 x_1^4 = y_1 \\
P_4(x_2) = y_2 \qquad & a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 + a_4 x_2^4 = y_2 \\
P_4(x_3) = y_3 \qquad & a_0 + a_1 x_3 + a_2 x_3^2 + a_3 x_3^3 + a_4 x_3^4 = y_3 \\
P_4(x_4) = y_4 \qquad & a_0 + a_1 x_4 + a_2 x_4^2 + a_3 x_4^3 + a_4 x_4^4 = y_4
\end{aligned}$$

Y el sistema a resolver es:

$$\begin{pmatrix}
1 & x_0 & x_0^2 & x_0^3 & x_0^4 \\
1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\
1 & x_2 & x_2^2 & x_2^3 & x_2^4 \\
1 & x_3 & x_3^2 & x_3^3 & x_3^4 \\
1 & x_4 & x_4^2 & x_4^3 & x_4^4
\end{pmatrix}
\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} =
\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

es decir, $Vp = y$, donde V es la matriz de coeficientes del sistema, *Matriz de Vandermonde*, y contiene los términos independientes del sistema y la solución del sistema son los coeficientes del polinomio, contenidos en p.

Ejercicio 6

1. Escribir una función **`Vandermonde(x)`** que tenga como argumento de entrada el vector **`x`** y como argumento de salida la matriz de Vandermonde **`V`**.
2. Escribir una función **`polVandermonde(x,y)`** que tenga como argumento de entrada los valores de las coordenadas **`x`** e **`y`** de los nodos y como salida **`p`**, que contiene los coeficientes del polinomio interpolante que pasa por los nodos, es decir, $p = (a_0, a_1, a_2, a_3, a_4)$. Para ello:
   - Calcular **`V`** llamando a la función **`Vandermonde`**.
   - Calcular los coeficientes del polinomio resolviendo el sistema lineal **`Vp = y`**. Usar la orden **`p = np.linalg.solve(V,y)`**.
3. Dibujar el polinomio interpolante con los nodos:
   - **`xp = np.linspace(min(x),max(x))`** nos da 50 puntos x en el intervalo de interpolación.
   - Utilizar la orden **`yp = pol.polyval(xp,p)`** para evaluar el polinomio en un punto o puntos **`xp`** a partir de los coeficientes del polinomio, **`p`**, obtenidos en el apartado anterior y obtener sus valores **`yp`**.

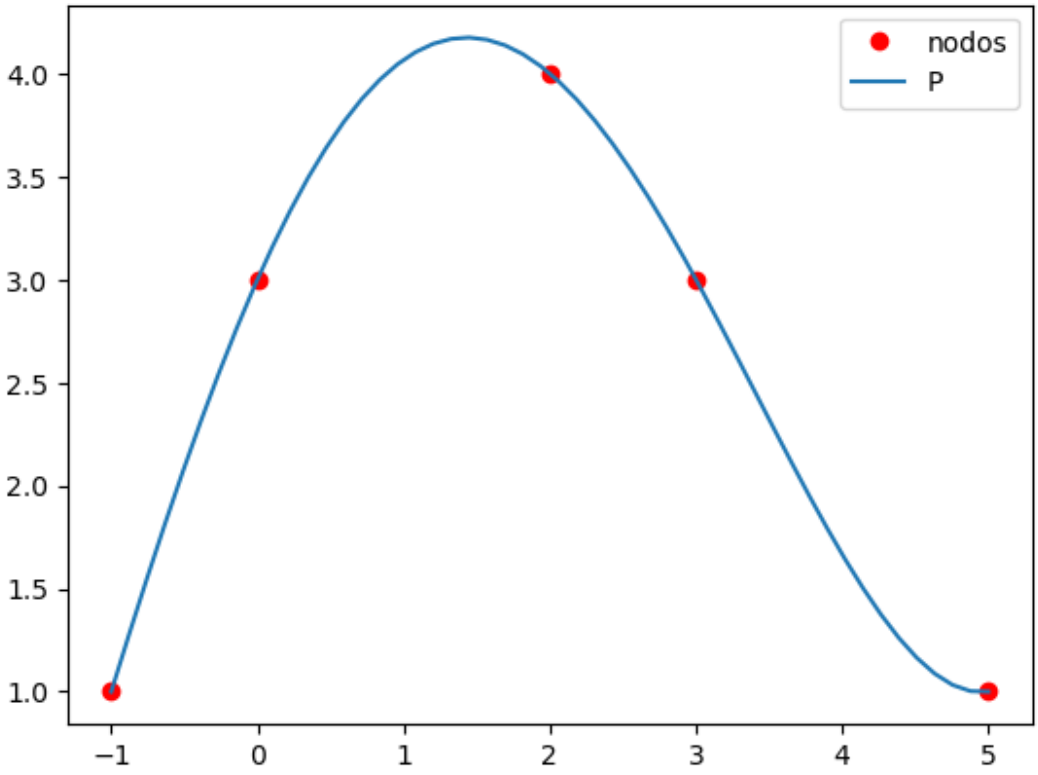
Usar los nodos:

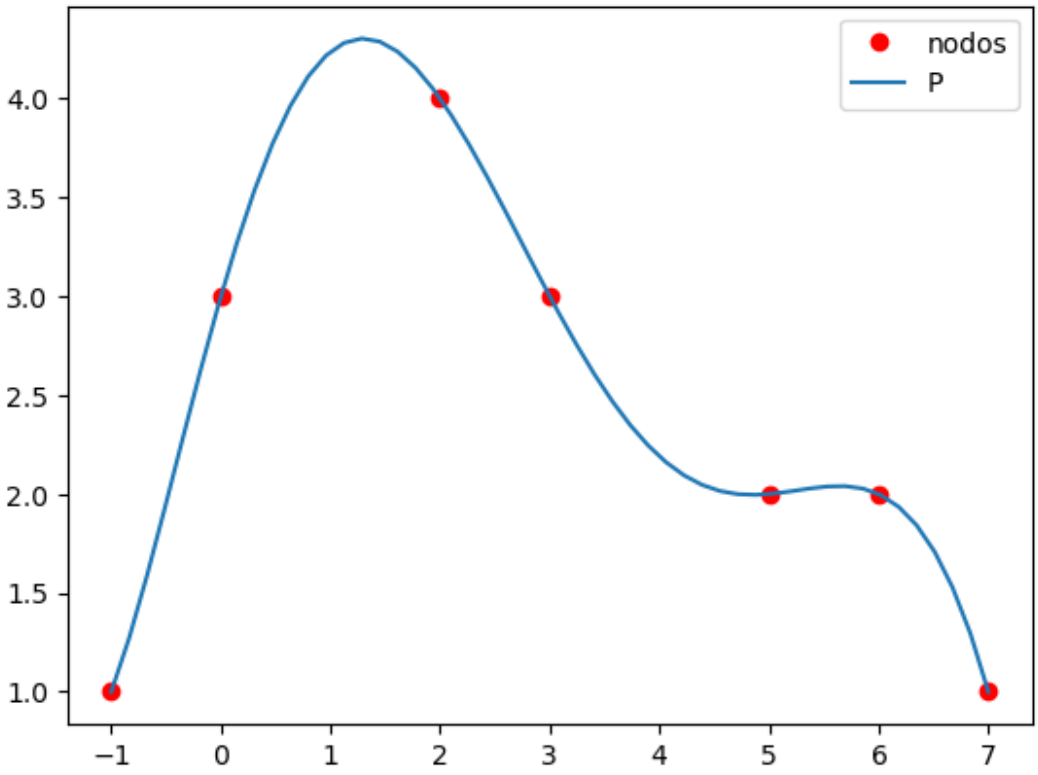
```
x = np.array([-1., 0, 2, 3, 5])
y = np.array([ 1., 3, 4, 3, 1])
```

Comprobar que también funciona con los nodos:

```
x1 = np.array([-1., 0, 2, 3, 5, 6, 7])
y1 = np.array([ 1., 3, 4, 3, 2, 2, 1])
```

```
%run Ejercicio6.py
```





Como resultados intermedios en este problema, podemos obtener la matriz de Vandermonde y los coeficientes del polinomio.

```
print('Matriz de Vandermonde V')
print(V)
```

```
Matriz de Vandermonde V
[[  1.  -1.   1.  -1.   1.]
 [  1.   0.   0.   0.   0.]
 [  1.   2.   4.   8.  16.]
 [  1.   3.   9.  27.  81.]
 [  1.   5.  25. 125. 625.]]
```

```
print('Coeficientes del polinomio')
print(p)
```

```
Coeficientes del polinomio
[ 3.    1.6  -0.48 -0.07  0.02]
```

Existe una función `numpy` que calcula la Matriz de Vandermonde

```
va = pol.polyvander(x, len(x)-1)
print(va)
```

```
[[  1.  -1.   1.  -1.   1.]
 [  1.   0.   0.   0.   0.]
 [  1.   2.   4.   8.  16.]
 [  1.   3.   9.  27.  81.]
 [  1.   5.  25. 125. 625.]]
```

Pero este no es un método aconsejable porque la matriz de Vandermonde, en general, está mal condicionada y puede dar lugar a grandes errores en la solución del sistema.