

AJUSTE DE CURVAS

✓ Ejercicio 4:

Dentro del archivo mediciones.txt se encuentran dos columnas de datos. La primera corresponde a mediciones del eje X, y la segunda corresponde a mediciones del eje Y. Utilizando dichos datos:

- a) Grafique los datos como puntos discretos
- b) Ajustar a los datos una curva polinómica de **primer grado** y graficarla sobre los puntos discretos
- c) Ajustar a los datos una curva polinómica de **segundo grado** y graficarla sobre los puntos discretos
- d) Ajustar a los datos una curva polinómica de **tercer grado** y graficarla sobre los puntos discretos

```

import matplotlib.pyplot as plt
import numpy as np

lista_x = np.loadtxt('mediciones.txt', float, usecols=0)
lista_y = np.loadtxt('mediciones.txt', float, usecols=1)

primer_x = lista_x[0]
ultimo_x = lista_x[-1]

lista_x_extendida = np.linspace(primer_x, ultimo_x, 100) #Para graficar suavemente

#Para curva primer grado:

lista_de_coeficientes_1er_grado = np.polyfit(lista_x, lista_y, 1)
p1 = np.poly1d(lista_de_coeficientes_1er_grado)

#pl.plot(lista_x_extendida, p1(lista_x_extendida), label = '1er grado')

#Para curva segundo grado:

lista_de_coeficientes_2do_grado = np.polyfit(lista_x, lista_y, 2)
p2 = np.poly1d(lista_de_coeficientes_2do_grado)

#pl.plot(lista_x_extendida, p2(lista_x_extendida), label = '2do grado')

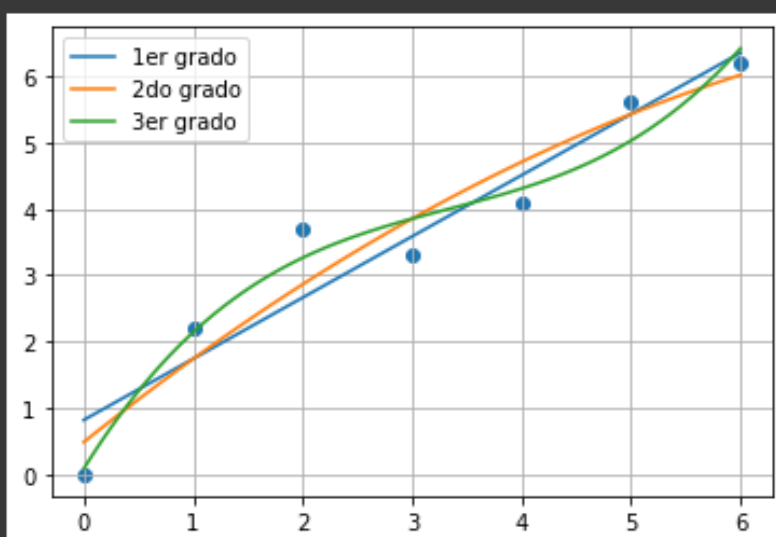
#Para curva tercer grado:

lista_de_coeficientes_3er_grado = np.polyfit(lista_x, lista_y, 3)
p3 = np.poly1d(lista_de_coeficientes_3er_grado)

#pl.plot(lista_x_extendida, p3(lista_x_extendida), label = '3er grado')

plt.scatter(lista_x, lista_y)
plt.plot(lista_x_extendida, p1(lista_x_extendida),
         lista_x_extendida, p2(lista_x_extendida),
         lista_x_extendida, p3(lista_x_extendida))
plt.legend(('1er grado', '2do grado', '3er grado',), loc='best')
plt.grid()
plt.show()

```



✓ Ejercicio 5:

Dentro del archivo **lineal.txt**, se encuentran dos columnas de datos, la primera corresponde a mediciones en el eje X y la segunda a mediciones en el eje Y. Con ellas, desarrollar los siguientes ejercicios:

- a) Grafique los datos como puntos discretos
- b) Utilizando la función **interp** de numpy, efectuar un Spline lineal sobre los datos discretos, de tal manera que sea posible dibujar una curva que atraviese a todos los puntos.

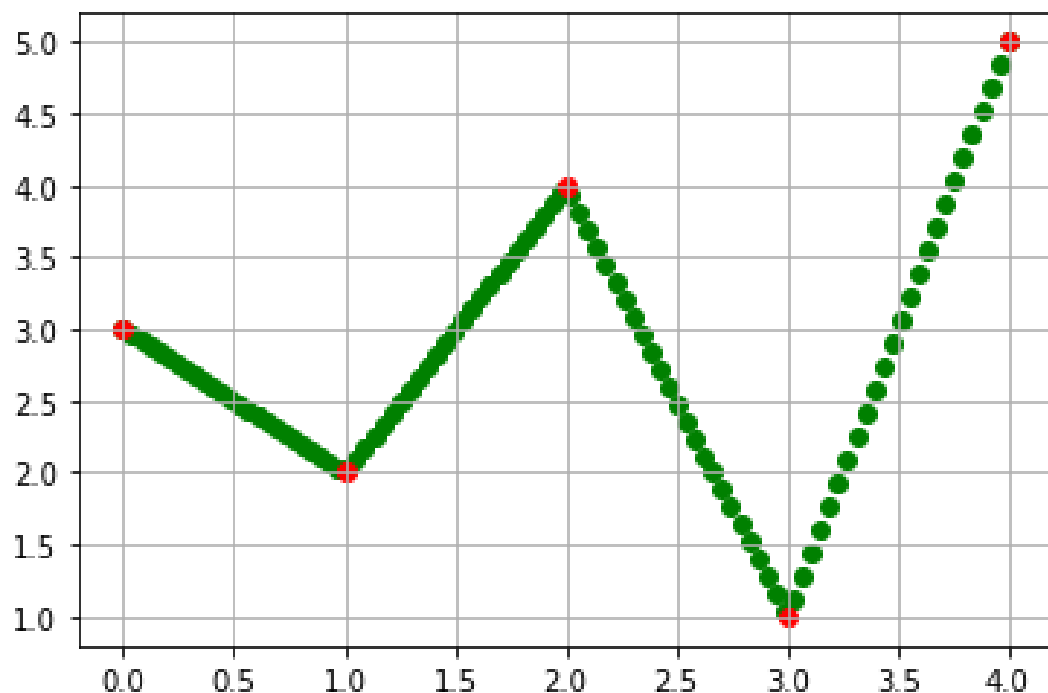
```
import matplotlib.pyplot as plt
import numpy as np

array_x = np.loadtxt('lineal.txt', float, usecols=0)
array_y = np.loadtxt('lineal.txt', float, usecols=1)

primer_x = array_x[0]
ultimo_x = array_x[-1]

array_x_fluido = np.linspace(primer_x, ultimo_x, 100)
array_y_fluido = np.interp(array_x_fluido, array_x, array_y)

plt.scatter(array_x_fluido, array_y_fluido, color='green')
plt.scatter(array_x, array_y, color='red')
#plt.plot(array_x, array_y)
plt.grid()
plt.show()
```



✓ Ejercicio 6:

Utilizando el mismo archivo **lineal.txt**, desarrollar lo siguiente:

- a) Grafique los datos como puntos discretos
- b) Definir una función $S(x)$ que permita dibujar un spline lineal sobre los datos discretos
- c) Graficar el spline lineal sobre los datos discretos.

```

import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d

array_x = np.loadtxt('lineal.txt', float, usecols=0)
array_y = np.loadtxt('lineal.txt', float, usecols=1)

primer_x = array_x[0]
ultimo_x = array_x[-1]

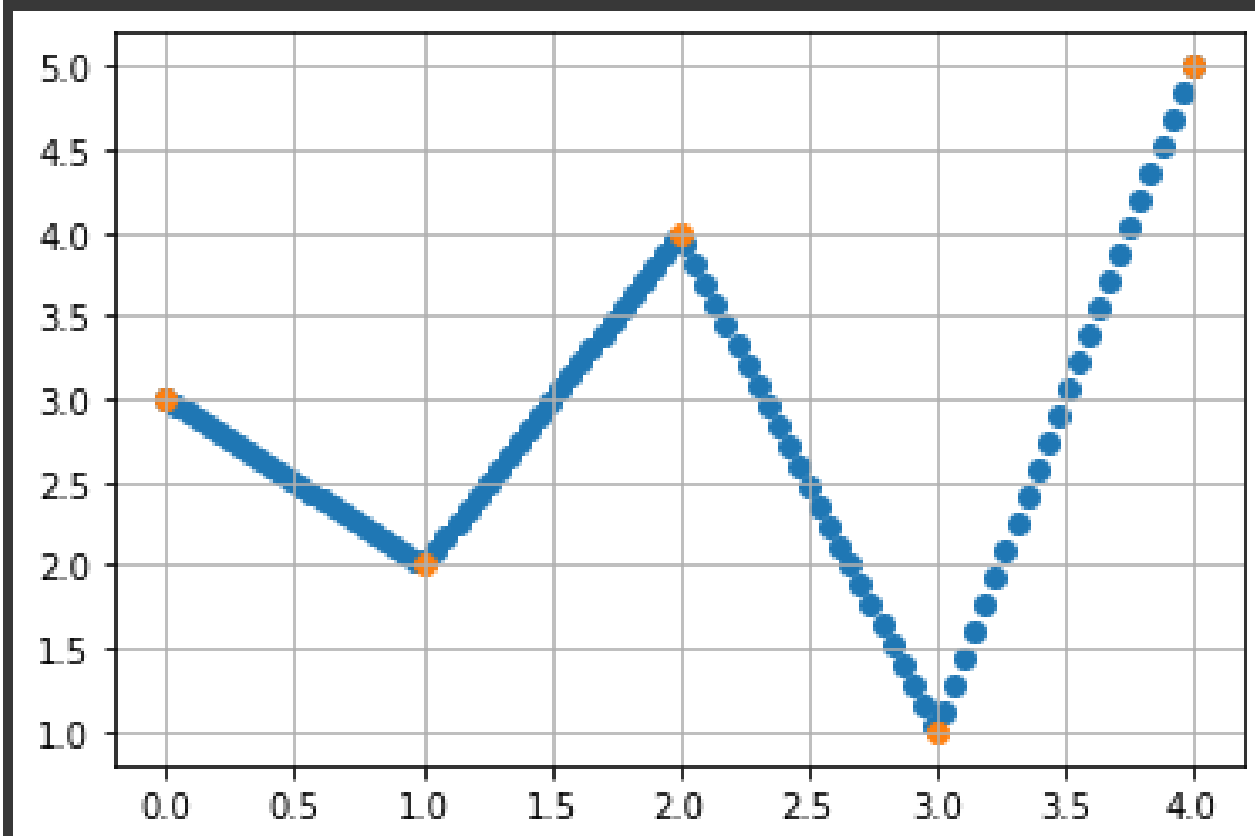
S = interp1d(array_x, array_y)

array_x_fluido = np.linspace(primer_x, ultimo_x, 100)

plt.scatter(array_x_fluido, S(array_x_fluido))
plt.scatter(array_x, array_y)
plt.grid()
plt.show()

print("El valor en Y cuando X es 2.2, es:", S(2.2))

```



El valor en Y cuando X es 2.2, es: 3.3999999999999995

✓ Ejercicio 7:

Dentro del archivo **cubic.txt**, se encuentran dos columnas de datos, la primera corresponde a mediciones en el eje X y la segunda a mediciones en el eje Y. Con ellas, desarrollar los siguientes ejercicios:

- a) Grafique los datos como puntos discretos.
- b) Definir una función $Z(x)$ que permita dibujar un spline cúbico sobre los datos discretos.
- c) Graficar el spline cúbico sobre los datos discretos.
- d) Graficar una curva interpolada junto con los puntos de datos discretos y ponerlo como subplot al lado del spline.

```

import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d

array_x = np.loadtxt('cubic.txt', float, usecols=0)
array_y = np.loadtxt('cubic.txt', float, usecols=1)

#Spline
primer_x = array_x[0]
ultimo_x = array_x[-1]

array_x_fluido = np.linspace(primer_x, ultimo_x, 100)
Z = interp1d(array_x, array_y, kind='cubic')
####

#Interpolación
cantidad_de_datos = len(array_x)

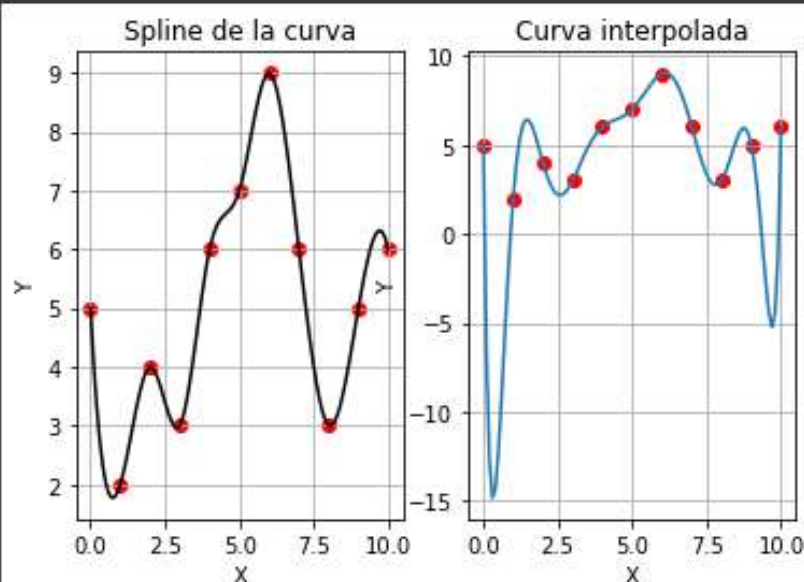
lista_de_coeficientes = np.polyfit(array_x, array_y, cantidad_de_datos - 1)

p = np.poly1d(lista_de_coeficientes)
####

plt.clf()
plt.subplot(1,2,1)
plt.scatter(array_x, array_y, color='red')
plt.plot(array_x_fluido, Z(array_x_fluido), '-k')
plt.title('Spline de la curva')
plt.xlabel('X');plt.ylabel('Y')
plt.grid()

plt.subplot(1,2,2)
plt.scatter(array_x, array_y, color='red')
plt.plot(array_x_fluido, p(array_x_fluido))
plt.title('Curva interpolada')
plt.xlabel('X');plt.ylabel('Y')
plt.grid()
plt.show()

```



Dentro del archivo **temp_pressure_PP2.txt**, se encuentran tres columnas de datos atmosféricos, la primera corresponde a mediciones de altura en $[km]$, la segunda a mediciones de temperatura en $[K]$ y la tercera a mediciones de presión en $[Pa]$ (asuma los datos como exactos).

Existe una propiedad llamada temperatura potencial, la cual denotaremos por θ . Esta propiedad depende tanto de la temperatura como de la presión:

$$\theta(T, p) = T \left(\frac{p}{p_0} \right)^{\frac{R}{c_p}}$$

Donde:

- T : temperatura en $[K]$
- p : presión en $[Pa]$
- p_0 : 100000 $[Pa]$
- R : $8.31[J * mol^{-1} * K^{-1}]$
- c_p : $2.91[J * mol^{-1} * K^{-1}]$

Con esta información, resuelva:

- a) Calcular cuál será el valor de la temperatura potencial θ a 2.2 $[km]$ y a 5.8 $[km]$ de altura.
- b) Elaborar un gráfico continuo que muestre cómo varía la temperatura potencial θ respecto a la altura en $[km]$.

```

import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

#EJERCICIO TIPO PRUEBA

altura = np.loadtxt('temp_pressure_PP2.txt', float, usecols=0)
T = np.loadtxt('temp_pressure_PP2.txt', float, usecols=1)
p = np.loadtxt('temp_pressure_PP2.txt', float, usecols=2)

R = 8.31
cp = 2.91

#Definimos la funcion de temperatura potencial con las constantes correspondientes:

def theta(temp, presion):
    return temp * ((presion / 100000)**(R / cp))

temp_potenciales = theta(T, p)

#Utilizo spline cubico para interpolar, porque ofrece mejores aproximaciones:

TP_Func = interp1d(altura, temp_potenciales, kind='cubic')

#Ahora puedo conocer los valores de temperatura potencial para:

print('Temperatura potencial a altura de 2.2 km: ', TP_Func(2.2), ' [K]');print()
print('Temperatura potencial a altura de 5.8 km: ', TP_Func(5.8), ' [K]');print()

#Luego, sólo falta graficar altura vs. temperatura potencial.

altura_suave = np.linspace(altura[0], altura[-1], 150)

plt.plot(altura_suave, TP_Func(altura_suave))
plt.title(r'Gráfico  $\theta(T,p)$ ')
plt.xlabel('Altura [km]');plt.ylabel(r' $\theta$  [K]')
plt.grid()
plt.show()

```

Temperatura potencial a altura de 2.2 km: 132.50021120073777 [K]

Temperatura potencial a altura de 5.8 km: 31.7654854284243 [K]

