Curso de Optimización (DEMAT)

Tarea 2

Descripción:	Fechas
Fecha de publicación del documento:	Febrero 10, 2022
Fecha límite de entrega de la tarea:	Febrero 20, 2022

Indicaciones

El propósito de esta tarea es poner en prática lo que hemos revisado sobre Python, por lo que los ejercicios son de programación.

Puede escribir el código de los algoritmos que se piden en una celda de este notebook o si lo prefiere, escribir las funciones en un archivo `.py` independiente e importar la funciones para usarlas en este notebook. Lo importante es que en el notebook aparezcan los resultados de la pruebas realizadas y que:

- Si se requieren otros archivos para poder reproducir los resultados, para mandar la tarea cree un archivo ZIP en el que incluya el notebook y los archivos adicionales.
- Si todos los códigos para que se requieren para reproducir los resultados están en el notebook, no hace falta comprimirlo y puede anexar sólo el notebook en la tarea del Classroom.
- Exportar el notebook a un archivo PDF y anexarlo en la tarea del Classroom como un archivo independiente. **No lo incluya dentro del ZIP** , porque la idea que lo pueda accesar directamente para poner anotaciones y la calificación de cada ejercicio.

En la descripción de los ejercicios se nombran algunas variables para el algoritmo, pero sólo es para facilitar la descripción. En la implementación pueden nombrar sus variables como gusten.

En los algoritmos se describen las entradas de las funciones. La intención es que tomen en cuenta lo que requiere el algoritmo y que tiene que haber parámetros que permitan controlar el comportamiento del algoritmo, evitando que dejen fijo un valor y que no se puede modificar para hacer diferentes pruebas. Si quieren dar esta información usando un tipo de dato que contenga todos los valores o usar variables por separado, etc., lo pueden hacer y no usen variables globales si no es necesario.

Lo mismo para los valores que devuelve una función. Pueden codificar como gusten la manera en que regresa los cálculos. El punto es que podamos tener acceso a los resultados, sin usar variables globales, y que la función no sólo imprima los valores que después no los podamos usar.

Ejercicio 1 (4 puntos)

Calcular las raíces de los polinomios dados y generar las gráficas de los polinomos para mostrar las raíces reales encontradas.

1. Escriba una función que reciba un arreglo c que contiene los coeficientes del polinomio, de modo que si c es un arreglo de longitud n el valor del polinomio de grado $n-1$ en x se calcula mediante

$$c[0] * x * * (n-1) + c[1] * x * * (n-2) + \ldots + c[n-2] * x + c[n-1]$$

1. Revise la documentación de la función [roots()](#) de Numpy y úsela para obtener un arreglo con las raíces del polinomio e imprima las raíces encontradas.
2. Las raíces pueden ser complejas. Obtenga un arreglo r que contenga sólo las raíces reales, imprimalo y calcule las raíz real r_{min} más pequeña y la raíz real r_{max} más grande.
3. Use la función `linspace()` para generar un arreglo x con 100 valores que corresponden a una partición del intervalo $[r_{min}-1, r_{max}+1]$. Evalúe el polinomio en los valores de x. Puede usar la función de Numpy [polyval()](#) para evaluar el polinomio y generar un arreglo y.

4. Use los arreglos x y y para generar la gráfica del polinomio.
5. Agregue a la gráfica los puntos que representan las raíces reales r. Para esto evalue el polinomio en r para generar un arreglo $polr$ con esos valores. Use los arreglos r y $polr$ para graficar como puntos en la gráfica anterior
para ver que coinciden con los ceros de las gráfica.
6. Pruebe la función con los siguientes polinomios:

$$f_1(x) = -4x^3 + 33x^2 + 97x - 840$$

$$f_2(x) = -2x^4 + 15x^3 - 36x^2 + 135x - 162$$

Solución:

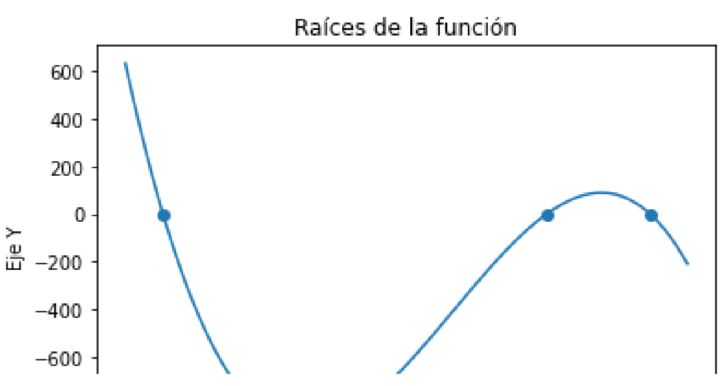
In [4]:

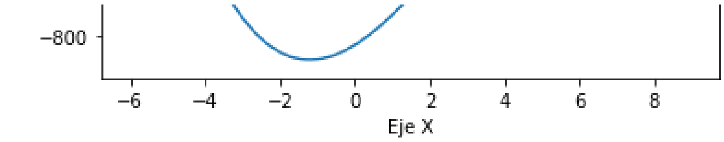
```
from numpy.core.function_base import linspace
# En esta celda puede poner el código de la función pedida
# o poner la instrucción para importarlas de un archivo .py

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def roots(c):
    r = np.roots(c)
    real = np.array([m.real for m in r if m.imag == 0])
    min = real.min()
    max = real.max()
    print(r)
    print("La raíces reales son: " ,real)
    print("El valor mínimo es: ",min)
    print("El valor máximo es: ",max)
    x = np.linspace(min-1, max+1, 100)
    y = np.polyval(c,x)
    polr = np.polyval(c,real)
    plt.plot(x, y)
    plt.scatter(real,polr)
    plt.title('Raíces de la función')
    plt.xlabel('Eje X')
    plt.ylabel('Eje Y')
    plt.show()
    return r, real, min, max
```

In [4]:

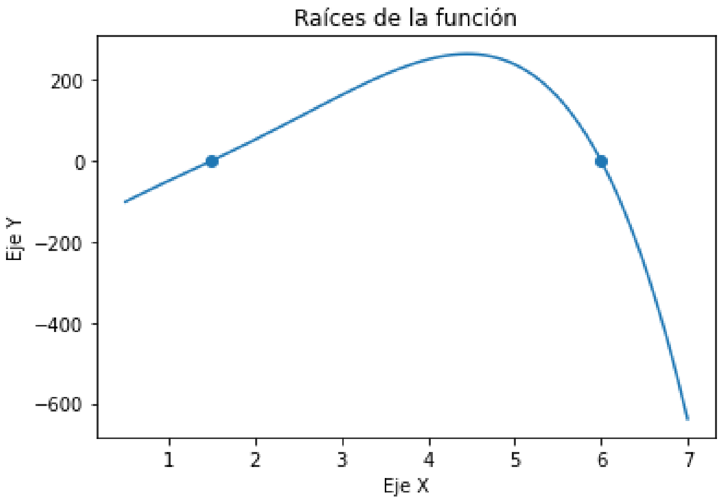
```
# Realice las pruebas que se indican
print("Las raíces de la función f1 son: ")
roots ([-4, 33, 97, -840])

print("\nLas raíces de la función f2 son: ")
roots ([-2, 15, -36, 135, -162])
```

```
Las raíces de la función f1 son: 
[-5.    8.    5.25]
La raíces reales son:  [-5.    8.    5.25]
El valor mínimo es:  -4.9999999999999964
El valor máximo es:  7.999999999999994
```






```
Las raíces de la función f2 son:
[6.00000000e+00+0.j 8.32667268e-16+3.j 8.32667268e-16-3.j
 1.50000000e+00+0.j]
La raíces reales son:  [6.  1.5]
El valor mínimo es:  1.5
El valor máximo es:  5.999999999999993
```



Out[4]:

```
(array([6.00000000e+00+0.j, 8.32667268e-16+3.j, 8.32667268e-16-3.j,
        1.50000000e+00+0.j]), array([6. , 1.5]), 1.5, 5.999999999999993)
```

El calculo de las raíces es muy preciso.

Ejercicio 2 (6 puntos)

Programar la función que resuelve el problema de ajustar un polinomio a un conjunto de puntos $\{(x_0, y_0), (x_1, y_1), \ldots, (x_m, y_m)\}$ usando mínimos cuadrados lineales.

Si revisan las notas del curso de métodos numéricos, se ve que para ajustar el polinomio de grado n $p(x) = c_n x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$ mediante mínimos cuadrados, hay que plantear el problema de minimizar la suma de diferencias elevadas al cuadrado:

$\sum_{i=0}^{m} (p(x_i) - y_i)^2$

Esto nos lleva a construir la matriz A y el vector de términos independientes

$$A = \begin{bmatrix} x_1^n & x_1^{n-1} & \cdots & x_1 & 1 \\ x_2^n & x_2^{n-1} & \cdots & x_2 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ x_m^n & x_m^{n-1} & \cdots & x_m & 1 \end{bmatrix}, \qquad b = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

Entonces el vector c con los coeficientes del polinomio se obtiene resolviendo el sistema de ecuaciones

$A^\top A c = A^\top y$.

1. **Programe la función que recibe como argumento un arreglo 2D que contiene los puntos $\{(x_0, y_0), (x_1, y_1), \ldots, (x_m, y_m)\}$ y el grado del polinomio n, que construya y resuelva el sistema de ecuaciones**

para obtener el vector de coeficientes c usando las funciones de Numpy y que devuelva este arreglo y el número de condición de la matriz del sistema. Este último dato lo puede obtener usando la función `numpy.linalg.cond()`.

2. Escriba una función que reciba como argumentos el nombre de un archivo que contiene los datos, el valor n del grado del polinomio que se quiere ajustar y un entero $r > 0$.

- La función debe leer el archivo, cargar los datos en una matriz y usar la función del inciso anterior para obtener el vector de coeficientes c y el número de condición. El archivo contiene una matriz con dos columnas. La primer columna tiene las abscisas $x_0, x_1, \ldots, x_m$ y la segunda columna tiene las ordenadas $y_0, y_1, \ldots, y_m$ de los puntos.
- Obtenga el valor mínimo $x_{\min}$ y máximo $x_{\max}$ de las abscicas x_i.
- Genere una partición $z_0, z_1, \ldots, z_{r-1}$ del intervalo $[x_{\min}, x_{\min}]$ con r puntos y use la función `numpy.polyval()` para evaluar el polinomio $p(x)$ en los puntos de la partición del intervalo.
- Haga que la función imprima el grado n del polinomio, los coeficientes c del polinomio y el número de condición. También haga que la función genere una gráfica que muestre los puntos $\{(x_0, y_0), (x_1, y_1), \ldots, (x_m, y_m)\}$ (como puntos) y los puntos $(z_i, p(z_i))$ con un trazo continuo para comparar los datos con la gráfica del polinomio.

1. Pruebe la función del inciso anterior usando los archivos `npy` que se encuentran dentro del archivo `datosTarea02.zip`. Para cada caso, use $r = 100$ y $n = 1, 2, 3, 4, 5$ y 6 (puede poner un ciclo para generar los resultados de cada caso).

Solución:

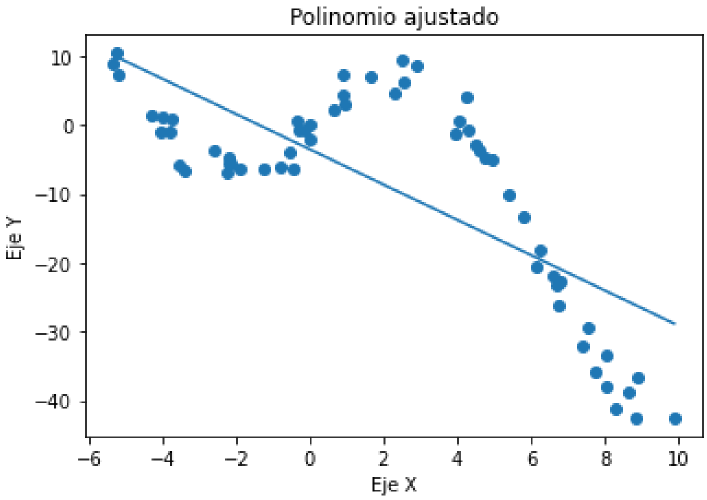
In [17]:

```python
from numpy.core.fromnumeric import transpose
# En esta celda puede poner el código de las funciones
# o poner la instrucción para importarlas de un archivo .py

def polyajuste (a, n):  #a es el arreglo de pares (xi,yi)
    x = np.array([m[0] for m in a])
    y = np.array([m[1] for m in a])
    expon = [n-i for i in range(n+1)]
    AT = np.array([x**j for j in expon])
    A = transpose(AT)
    ATA = np.matmul(AT, A)
    ATy = np.matmul(AT, y)
    c = np.linalg.solve(ATA, ATy)  #c son mis coeficientes del polinomio ajustado
    cond = np.linalg.cond(ATA)
    return c, cond

def ReadDataAndAdjust(archivo, n, r):
    datos = np.load(archivo)
    c, cond = polyajuste(datos, n)
    x = np.array([m[0] for m in datos])
    y = np.array([m[1] for m in datos])
    min = x.min()
    max = x.max()
    z = np.linspace(min, max, r)
    pz = np.polyval(c,z)
    plt.plot(z, pz)
    plt.scatter(x, y)
    print("El grado del polinomio:", n)
    print("Los coeficientes del polinomio:", c)
    print("El número de la condición:", cond)
    plt.title('Polinomio ajustado')
    plt.xlabel('Eje X')
    plt.ylabel('Eje Y')
    plt.show()
```

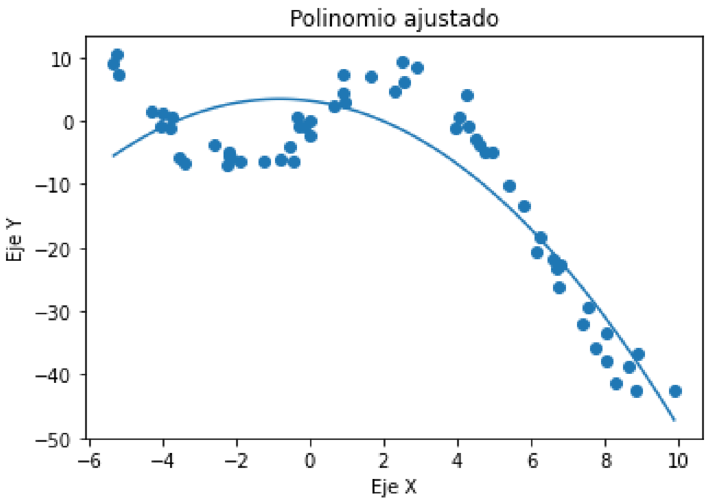
In [18]:

```
# Lectura de datos y pruebas realizadas
#Datos 1
for n in range (1,7):
    ReadDataAndAdjust("datos1.npy", n, 100)
```

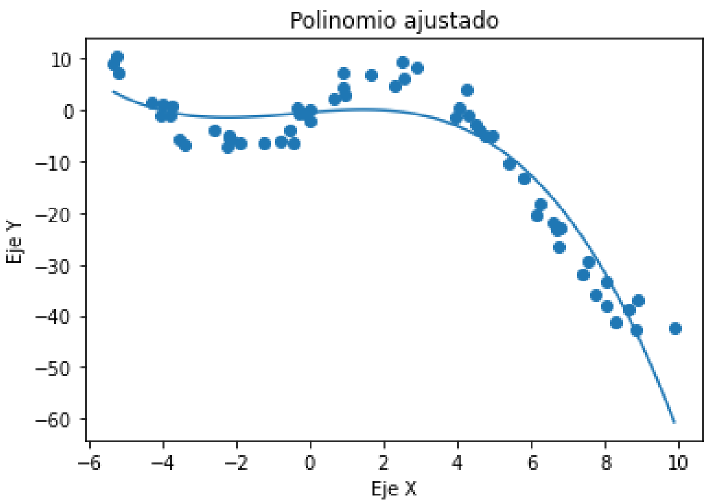
```
El grado del polinomio: 1
Los coeficientes del polinomio: [-2.55916204 -3.52034006]
El número de la condición: 29.545976262286963
```




```
El grado del polinomio: 2
Los coeficientes del polinomio: [-0.44159543 -0.71962797  3.15854673]
El número de la condición: 2458.72898553788
```

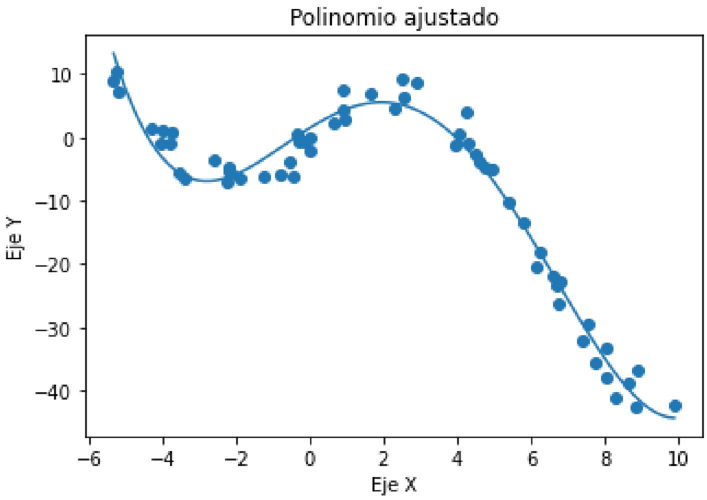



```
El grado del polinomio: 3
Los coeficientes del polinomio: [-0.06109608 -0.07256856  0.59937312 -0.38787334]
El número de la condición: 210402.57743870522
```

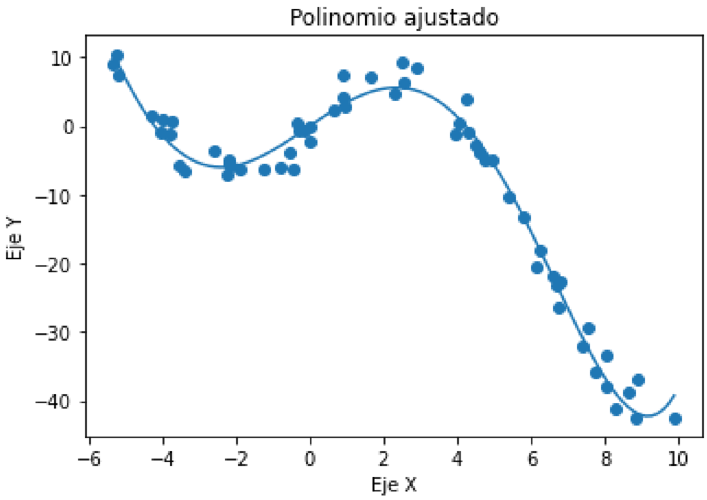



```
El grado del polinomio: 4
Los coeficientes del polinomio: [ 0.01704531 -0.205856   -0.47063422  3.66843303  1.45936
```

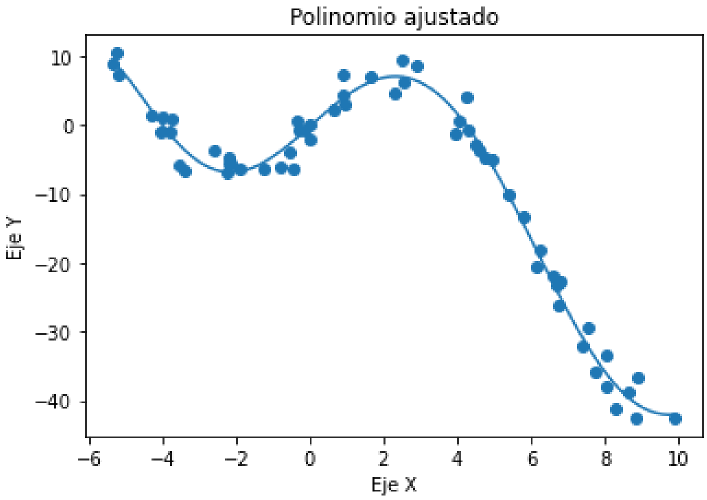
```
167]
El número de la condición: 15816940.818922417
```




```
El grado del polinomio: 5
Los coeficientes del polinomio: [ 1.29949313e-03  3.05302727e-03 -2.30789433e-01 -4.86830
173e-02
  3.70211559e+00 -4.01552133e-03]
El número de la condición: 1627465651.9693878
```



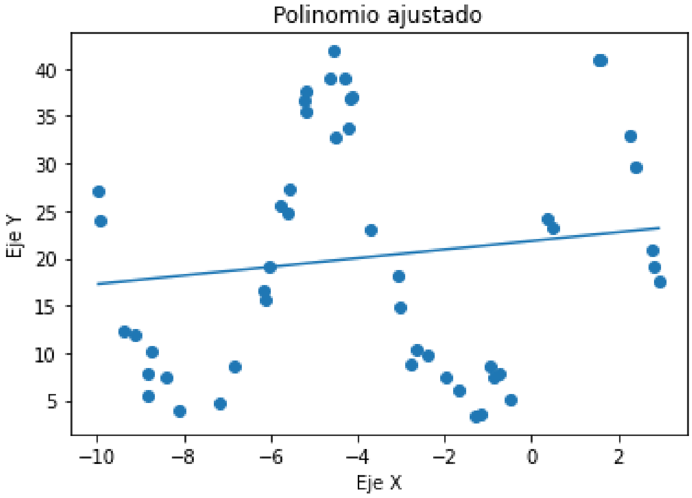

```
El grado del polinomio: 6
Los coeficientes del polinomio: [-2.41323682e-04  4.51463183e-03  5.43313400e-03 -3.58263
464e-01
  4.27818731e-02  4.78029052e+00 -2.22723051e-01]
El número de la condición: 132393662527.03316
```



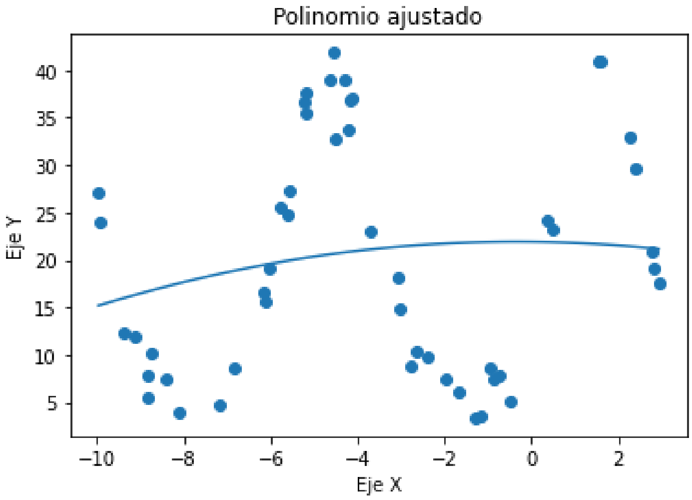
In [19]:

```
#Datos 2
for n in range (1,7):
    ReadDataAndAdjust("datos2.npy", n, 100)
```

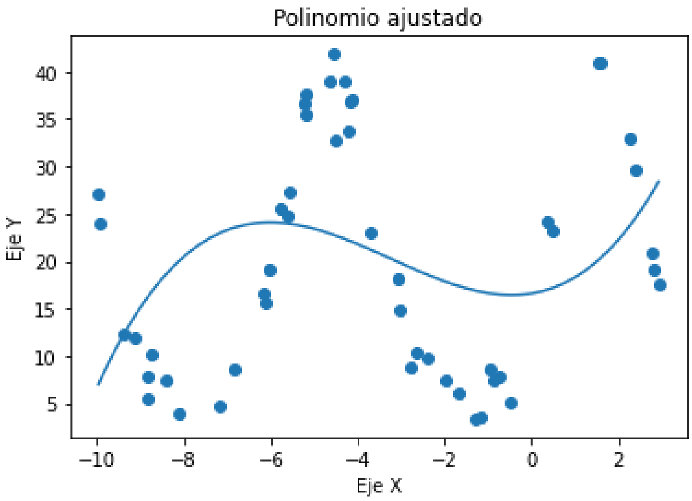
```
El grado del polinomio: 1
Los coeficientes del polinomio: [ 0.45503209 21.82332473]
El número de la condición: 57.66624448080778
```



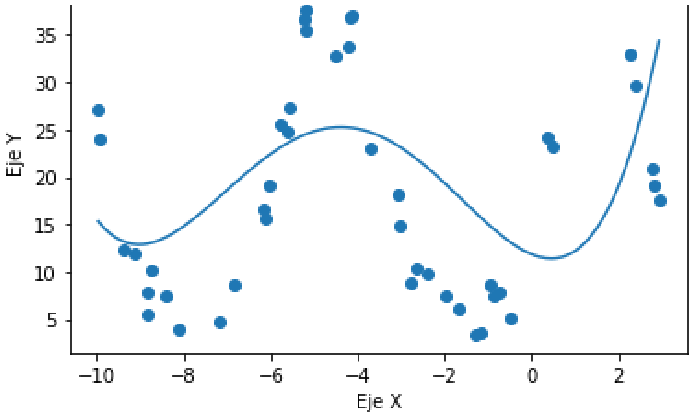

```
El grado del polinomio: 2
Los coeficientes del polinomio: [-0.07278501 -0.04994346 21.92621783]
El número de la condición: 3340.5363276776425
```



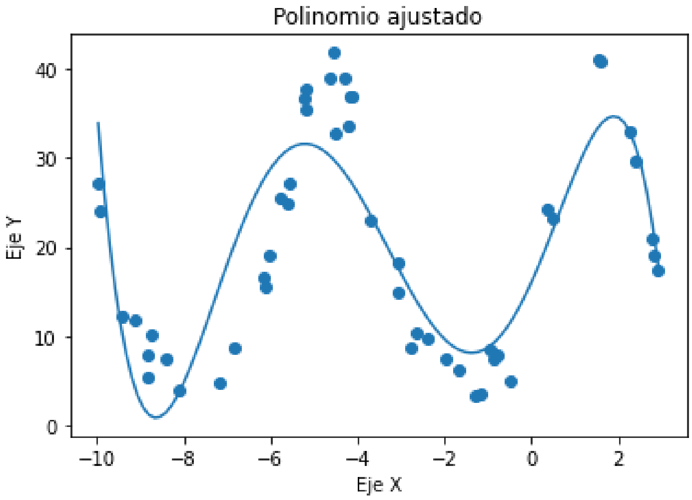

```
El grado del polinomio: 3
Los coeficientes del polinomio: [ 0.0889206   0.86350359  0.73317477 16.60591695]
El número de la condición: 458979.9355156497
```



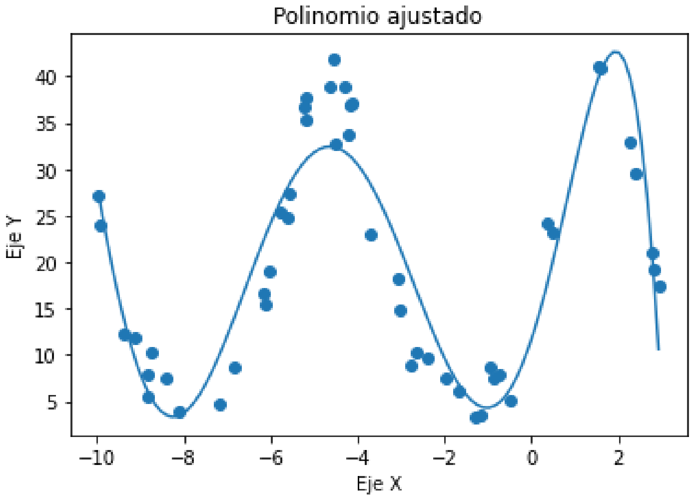

```
El grado del polinomio: 4
Los coeficientes del polinomio: [ 0.02585022  0.44675804  1.7319045  -1.87060715 11.85052
242]
El número de la condición: 57617346.05445348
```




```
El grado del polinomio: 5
Los coeficientes del polinomio: [-0.01844394 -0.30702364 -1.0774548   2.76387466 10.82311
663 16.00554059]
El número de la condición: 5519382925.375805
```




```
El grado del polinomio: 6
Los coeficientes del polinomio: [-2.68260451e-03 -7.61104642e-02 -6.60842886e-01 -1.20964
489e+00
  6.29804761e+00  1.42854727e+01  1.16810636e+01]
El número de la condición: 537838254508.4178
```



Notemos que el ajuste es muy bueno para los primeros datos a partir del polinomio de grado 4.

Los valores de los datos2 no es tan bueno, hasta el polinomio de grado 6 se ajusta de una forma más aproximada.