Fecha entrega: Viernes 2 Mayo 2025

Taller 1 - corte 2

Modelo de regresión lineal con el uso de matrices

Al ajustar un modelo de regresión lineal múltiple, en particular cuando el número de variable pasa de dos, el conocimiento de la teoría matricial puede facilitar las manipulaciones matemáticas. Supongamos que el experimentador tiene x1, x2, ..., xk variables independientes y n observaciones y1, y2, .. yn.

$$y = X\beta + \varepsilon$$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \cdot \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{pmatrix}$$

Donde obtendriamos que los Betas soluciones al sistema de ecuaciones seria:

$$\beta = (X^T X)^{-1} X^T y$$

Se debe utilizar esta ecuacion para encontrar los Betas solucion de los siguientes set de datos

puntos a evaluar:

- Aplicar el modelo de regresion lineal matricial
- Encontrar Betas
- Graficar la solucion (predicciones) en conjunto con el set de datos

In [2]:

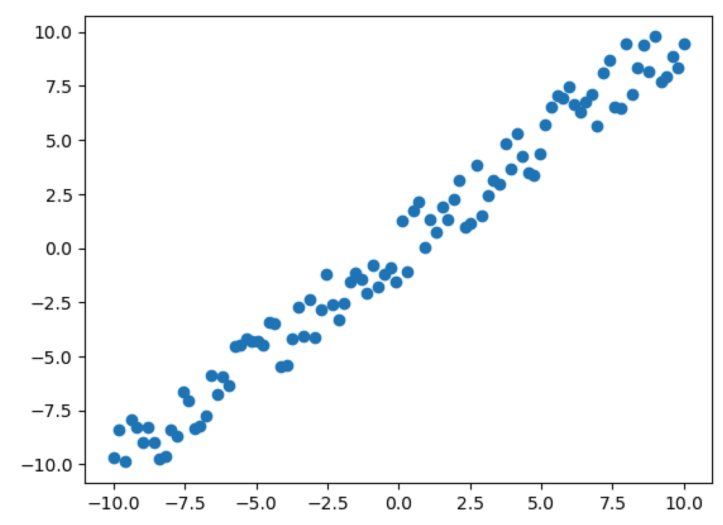
```python
import random
import math
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm
import numpy as np
```

punto 1

In [4]:

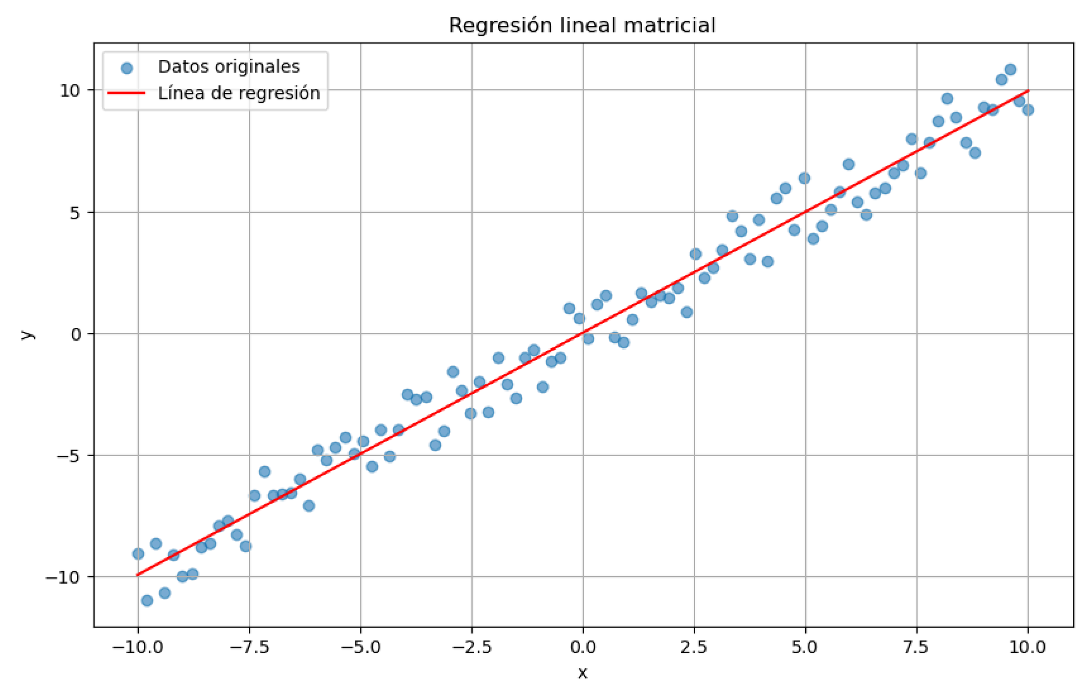
```python
data_x = np.linspace(-10,10,100)
data_y = np.array( [ x+(random.random()-0.5)*3 for x in data_x ] )
plt.scatter(data_x, data_y)
```

Out[4]: <matplotlib.collections.PathCollection at 0x198afda9be0>



In [1]:

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

data_x = np.linspace(-10, 10, 100)
data_y = np.array([x + (random.random() - 0.5) * 3 for x in data_x])

X = np.vstack([np.ones_like(data_x), data_x]).T

y = data_y.reshape(-1, 1)

beta = np.linalg.inv(X.T @ X) @ X.T @ y

y_pred = X @ beta

plt.figure(figsize=(10, 6))
plt.scatter(data_x, data_y, label='Datos originales', alpha=0.6)
plt.plot(data_x, y_pred, color='red', label='Línea de regresión')
plt.xlabel("x")
plt.ylabel("y")
plt.title("Regresión lineal matricial")
plt.legend()
plt.grid(True)
plt.show()

beta.flatten()
```



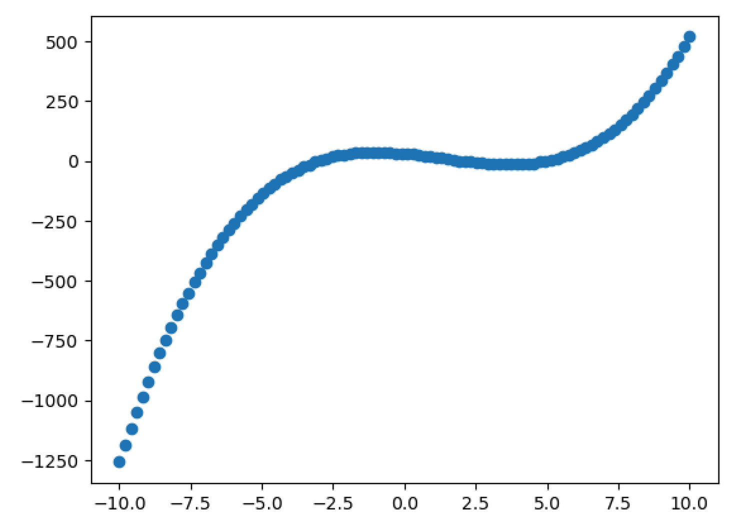
Out[1]: array([-0.00136275, 0.99507862])

punto 2

In [7]:

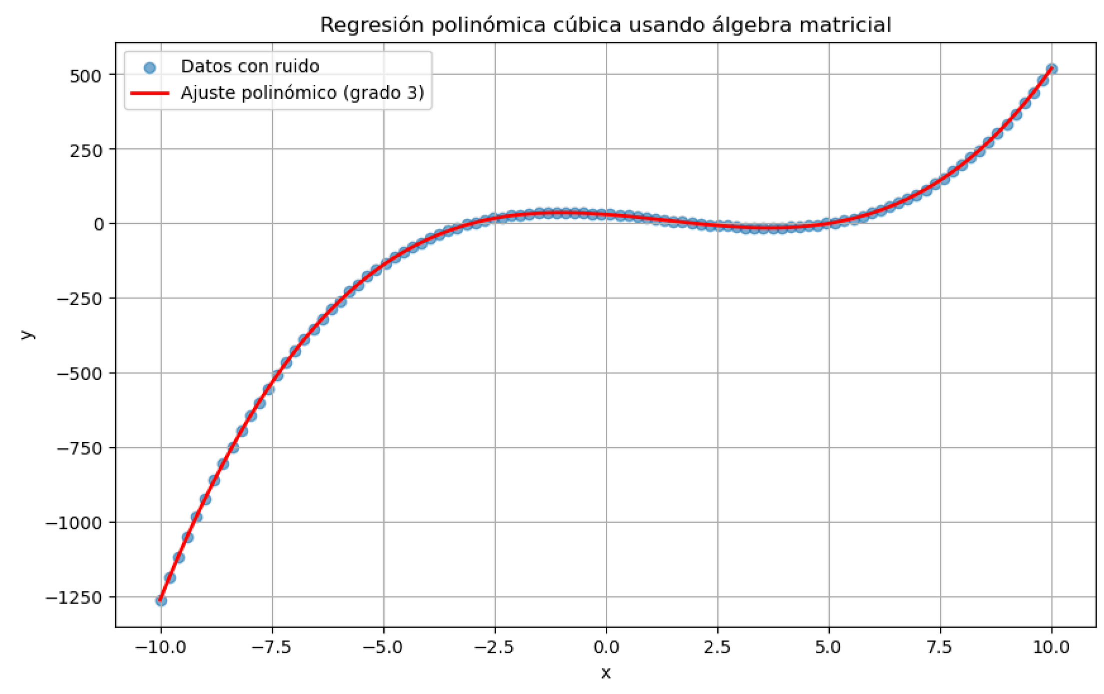
```python
data_x = np.linspace(-10,10,100)
data_y = np.array( [ (x-2)*(x-5)*(x+3)+(random.random()-0.5)*3 for x in data_x ] )
plt.scatter(data_x, data_y)
```

Out[7]: <matplotlib.collections.PathCollection at 0x198afdff470>



In [3]:

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

data_x = np.linspace(-10, 10, 100)
data_y = np.array([(x - 2)*(x - 5)*(x + 3) + (random.random() - 0.5) * 3 for x in data_x])

X = np.vstack([np.ones_like(data_x), data_x, data_x**2, data_x**3]).T
y = data_y.reshape(-1, 1)

beta = np.linalg.inv(X.T @ X) @ X.T @ y

y_pred = X @ beta

plt.figure(figsize=(10, 6))
plt.scatter(data_x, data_y, label='Datos con ruido', alpha=0.6)
plt.plot(data_x, y_pred, color='red', label='Ajuste polinómico (grado 3)', linewidth=2)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Regresión polinómica cúbica usando álgebra matricial')
plt.legend()
plt.grid(True)
plt.show()

# Mostrar coeficientes
for i, b in enumerate(beta.flatten()):
    print(f"β{i} = {b:.4f}")
```



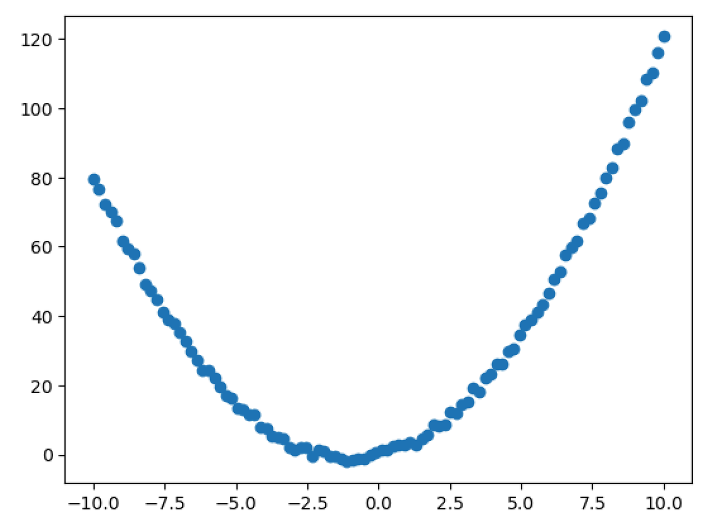

```
β0 = 30.0027
β1 = -11.0240
β2 = -4.0033
β3 = 1.0002
```

punto 3

In [10]:

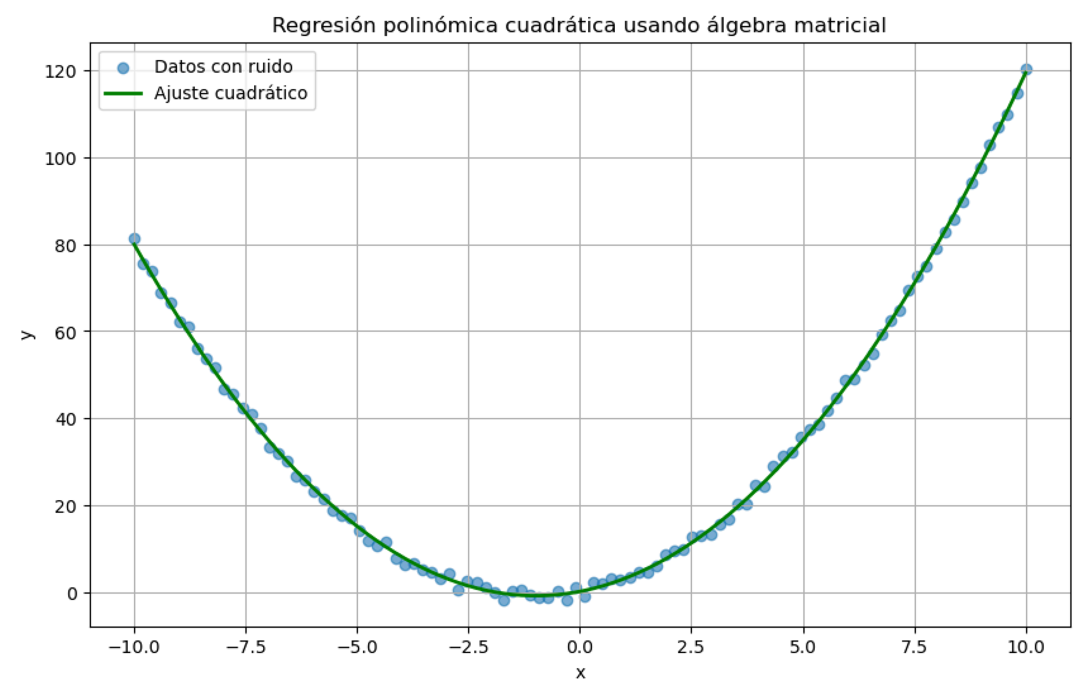
```python
data_x = np.linspace(-10,10,100)
data_y = np.array( [ x*(x+2)+(random.random()-0.5)*3 for x in data_x ] )
plt.scatter(data_x, data_y)
```

Out[10]: <matplotlib.collections.PathCollection at 0x198afeae450>



In [7]:

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

data_x = np.linspace(-10, 10, 100)
data_y = np.array([x*(x + 2) + (random.random() - 0.5)*3 for x in data_x])

X = np.vstack([np.ones_like(data_x), data_x, data_x**2]).T
y = data_y.reshape(-1, 1)

beta = np.linalg.inv(X.T @ X) @ X.T @ y

y_pred = X @ beta

plt.figure(figsize=(10, 6))
plt.scatter(data_x, data_y, label='Datos con ruido', alpha=0.6)
plt.plot(data_x, y_pred, color='green', label='Ajuste cuadrático', linewidth=2)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Regresión polinómica cuadrática usando álgebra matricial')
plt.legend()
plt.grid(True)
plt.show()

for i, b in enumerate(beta.flatten()):
    print(f"β{i} = {b:.4f}")
```



```
β0 = 0.0871
β1 = 1.9760
β2 = 0.9963
```

In [ ]: