Regresión Lineal Taller 1

Trabajo Corte 2 - Computación 2

Nombre: Mariana Rodríguez Pérez

Fecha: Abril 2025

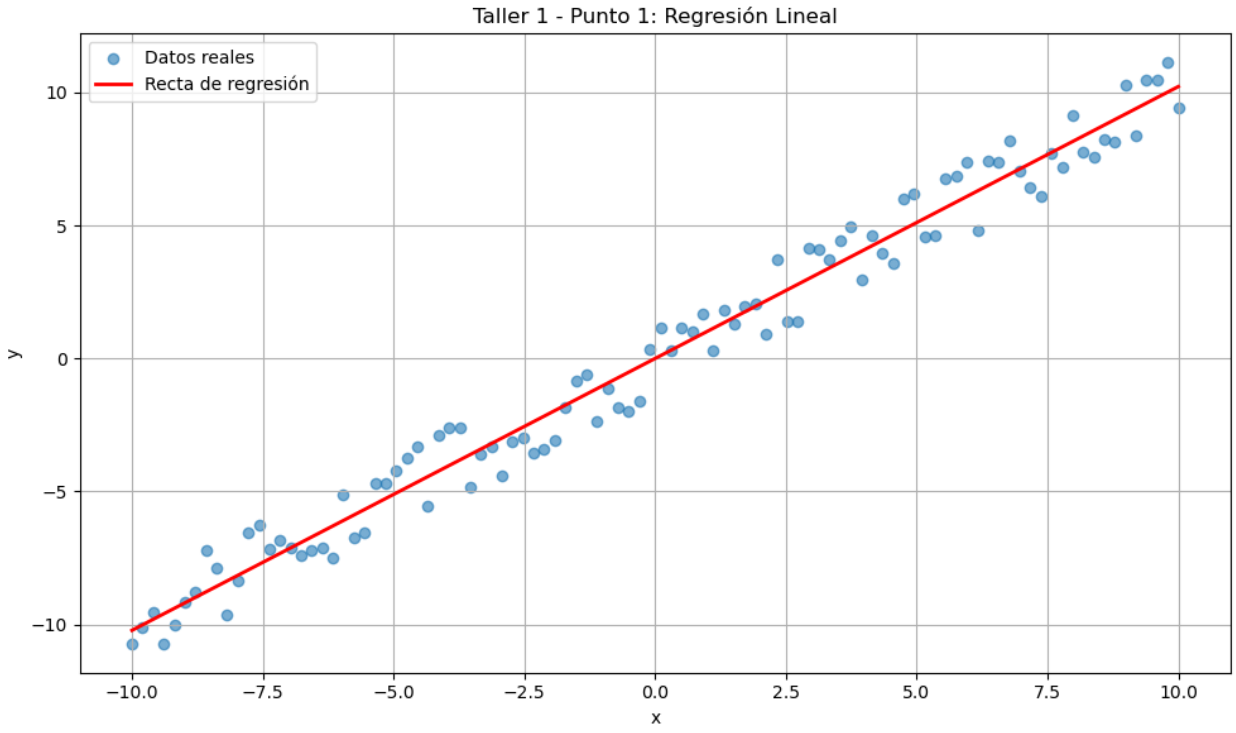
Punto 1

En este primer ejercicio se aplicó un modelo de regresión lineal simple utilizando álgebra matricial. El objetivo fue encontrar la recta que mejor se ajusta a un conjunto de datos generados artificialmente, donde y depende linealmente de x con algo de ruido aleatorio agregado para simular variabilidad real.

1. **Generación de datos**: Se creó un conjunto de 100 puntos donde x varía entre -10 y 10, y y es aproximadamente igual a x, pero con un pequeño ruido aleatorio.
2. **Construcción de la matriz X**: Para aplicar el modelo matricial, se formó la matriz X con dos columnas: una de unos (para el intercepto (\beta_0)) y otra con los valores de x (para la pendiente (\beta_1)).
3. **Aplicación de la fórmula matricial**: Se utilizó la ecuación [\boldsymbol{\beta} = (X^T X)^{-1} X^T y] para encontrar los coeficientes óptimos del modelo lineal. Esto nos da los valores de (\beta_0) y (\beta_1).
4. **Predicción**: Con los betas encontrados, se calcularon los valores estimados de y usando la expresión (\hat{y} = X \cdot \boldsymbol{\beta}).
5. **Visualización**: Finalmente, se graficaron los puntos originales (x vs y) junto con la línea de regresión ajustada, para observar qué tan bien se adapta el modelo a los datos.

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

# 1. Generar los datos con ruido
np.random.seed(0)
data_x = np.linspace(-10, 10, 100)
data_y = np.array([x + (random.random() - 0.5) * 3 for x in data_x])

# 2. Construir la matriz X con columna de unos (para beta_0) y columna
de x (para beta_1)
X = np.vstack((np.ones_like(data_x), data_x)).T
y = data_y.reshape(-1, 1)
```

```python
# 3. Calcular los coeficientes Beta con la fórmula matricial
# Beta = (X^T X)^-1 X^T y
beta = np.linalg.inv(X.T @ X) @ X.T @ y

# 4. Generar las predicciones
y_pred = X @ beta

# 5. Graficar los datos originales y la línea ajustada
plt.figure(figsize=(10, 6))
plt.scatter(data_x, data_y, label='Datos reales', alpha=0.6)
plt.plot(data_x, y_pred, label='Recta de regresión', color='red',
linewidth=2)
plt.title('Taller 1 - Punto 1: Regresión Lineal')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# 6. Mostrar los coeficientes
print(f"Coeficientes obtenidos:")
print(f"Beta_0 (intercepto): {beta[0][0]:.4f}")
print(f"Beta_1 (pendiente): {beta[1][0]:.4f}")
```



```
Coeficientes obtenidos:
Beta_0 (intercepto): 0.0017
Beta_1 (pendiente): 1.0218
```

Punto 2

En este segundo ejercicio aplicamos un modelo de regresión polinomial utilizando álgebra matricial. A diferencia del punto anterior, los datos no siguen una relación lineal, sino una forma cúbica con tres raíces reales. El objetivo fue ajustar un polinomio de grado 3 a estos datos usando la misma técnica matricial.

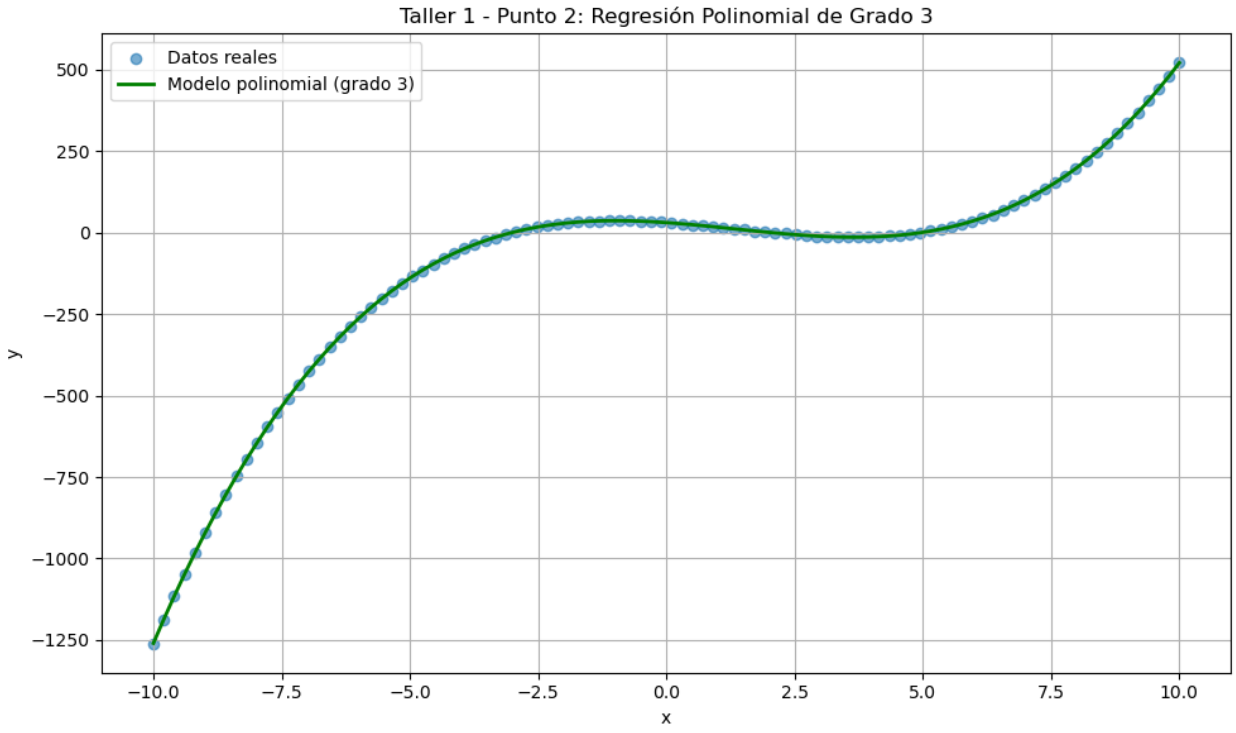
1. **Generación de datos**: Se creó un conjunto de datos en el que `y = (x - 2)(x - 5)(x + 3) + ruido`. Esto da como resultado una curva cúbica con ruido aleatorio para simular variabilidad real.
2. **Construcción de la matriz X**: Se construyó la matriz `X` con las siguientes columnas:
   - Una columna de unos (para el término independiente beta_0),
   - Una columna con `x`,
   - Una con `x^2`,
   - Y una con `x^3`.
3. **Aplicación de la fórmula matricial**: Se utilizó nuevamente la ecuación clásica:

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

   para encontrar los coeficientes óptimos del polinomio.
4. **Predicción**: Se generaron las predicciones `y_hat = X * beta` para evaluar qué tan bien el modelo representa los datos.
5. **Visualización**: Se graficaron los puntos originales y la curva ajustada por el modelo cúbico

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

# 1. Generar los datos con forma cúbica y algo de ruido
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])

# 2. Construir la matriz X para modelo polinomial de grado 3 (1, x,
x^2, x^3)
X = np.vstack((np.ones_like(data_x), data_x, data_x**2, data_x**3)).T
y = data_y.reshape(-1, 1)

# 3. Calcular los coeficientes Beta con fórmula matricial
```

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

# 4. Generar predicciones
y_pred = X @ beta

# 5. Graficar datos originales y curva ajustada
plt.figure(figsize=(10, 6))
plt.scatter(data_x, data_y, label='Datos reales', alpha=0.6)
plt.plot(data_x, y_pred, label='Modelo polinomial (grado 3)',
color='green', linewidth=2)
plt.title('Taller 1 - Punto 2: Regresión Polinomial de Grado 3')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# 6. Mostrar coeficientes Beta
print("Coeficientes obtenidos:")
print(f"Beta_0: {beta[0][0]:.4f}")
print(f"Beta_1: {beta[1][0]:.4f}")
print(f"Beta_2: {beta[2][0]:.4f}")
print(f"Beta_3: {beta[3][0]:.4f}")
```



```
Coeficientes obtenidos:
Beta_0: 29.9817
Beta_1: -10.9535
Beta_2: -3.9977
Beta_3: 0.9993
```

Punto 3

En este tercer ejercicio se aplicó un modelo de regresión polinomial cuadrático (grado 2) utilizando álgebra matricial. Los datos generados siguen una forma parabólica basada en la función `y = x(x + 2) + ruido`.

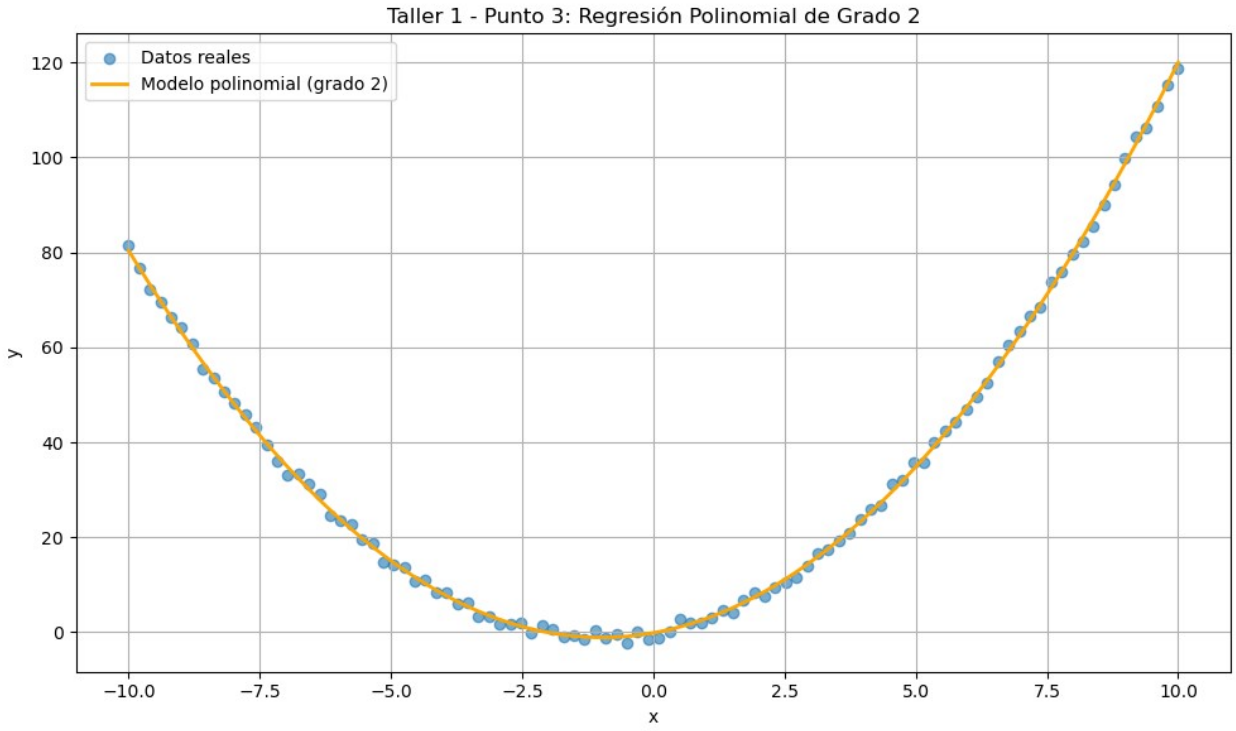
1. **Generación de datos**: Se generaron 100 puntos en el intervalo de `x = [-10, 10]` y se aplicó la función cuadrática `y = x(x + 2)` con un poco de ruido aleatorio.
2. **Construcción de la matriz X**: La matriz `X` se armó con tres columnas:
   - Una columna de unos (para el término independiente beta_0),
   - Una columna con `x`,
   - Una columna con `x^2`.
3. **Aplicación de la fórmula matricial**: Se utilizó la ecuación estándar:

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

   para obtener los coeficientes del modelo cuadrático.
4. **Predicción**: Se calculó la predicción `y_hat = X * beta` para cada valor de entrada `x`.
5. **Visualización**: Se graficaron los puntos originales y la curva ajustada con el modelo de grado 2.

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

# 1. Generar los datos con forma cuadrática y algo de ruido
data_x = np.linspace(-10, 10, 100)
data_y = np.array([x * (x + 2) + (random.random() - 0.5) * 3 for x in
data_x])

# 2. Construir la matriz X para modelo polinomial de grado 2 (1, x,
x^2)
X = np.vstack((np.ones_like(data_x), data_x, data_x**2)).T
y = data_y.reshape(-1, 1)

# 3. Calcular los coeficientes Beta con fórmula matricial
beta = np.linalg.inv(X.T @ X) @ X.T @ y
```

```python
# 4. Generar predicciones
y_pred = X @ beta

# 5. Graficar datos originales y curva ajustada
plt.figure(figsize=(10, 6))
plt.scatter(data_x, data_y, label='Datos reales', alpha=0.6)
plt.plot(data_x, y_pred, label='Modelo polinomial (grado 2)',
color='orange', linewidth=2)
plt.title('Taller 1 - Punto 3: Regresión Polinomial de Grado 2')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# 6. Mostrar coeficientes Beta
print("Coeficientes obtenidos:")
print(f"Beta_0: {beta[0][0]:.4f}")
print(f"Beta_1: {beta[1][0]:.4f}")
print(f"Beta_2: {beta[2][0]:.4f}")
```




```
Coeficientes obtenidos:
Beta_0: -0.1211
Beta_1: 1.9806
Beta_2: 1.0031
```