TallerN5LeninAmangandi

December 15, 2025

1 Escuela Politécnica Nacional

1.1 Métodos Numéricos

Nombre: Lenin Amangandi

Tema: Mínimos cuadrados

Link al repositorio Taller N5

[1]:
```python
# Derivadas parciales para regresión lineal
# ######################################################################
def der_parcial_1(xs: list, ys: list) -> tuple[float, float, float]:
    """Retorna los coeficientes de la ecuación de la derivada parcial con␣
 ↪respecto al parámetro 1 al reemplazar los valores ``xs`` y ``ys``. La␣
 ↪ecuación es de la forma:
    c_1 * a_1 + c_0 * a_0 = c_ind

    ## Parameters
    ``xs``: lista de valores de x.
    ``ys``: lista de valores de y.

    ## Return
    ``c_1``: coeficiente del parámetro 1.
    ``c_0``: coeficiente del parámetro 0.
    ``c_ind``: coeficiente del término independiente.
    """

    # coeficiente del término independiente
    c_ind = sum(ys)

    # coeficiente del parámetro 1
    c_1 = sum(xs)

    # coeficiente del parámetro 0
    c_0 = len(xs)

    return (c_1, c_0, c_ind)
```

```python
def der_parcial_0(xs: list, ys: list) -> tuple[float, float, float]:
    """Retorna los coeficientes de la ecuación de la derivada parcial con␣
 ↪respecto al parámetro 0 al reemplazar los valores ``xs`` y ``ys``. La␣
 ↪ecuación es de la forma:
    c_1 * a_1 + c_0 * a_0 = c_ind

    ## Parameters

    ``xs``: lista de valores de x.
    ``ys``: lista de valores de y.

    ## Return

    ``c_1``: coeficiente del parámetro 1.
    ``c_0``: coeficiente del parámetro 0.
    ``c_ind``: coeficiente del término independiente.
    """
    c_1 = 0
    c_0 = 0
    c_ind = 0
    for xi, yi in zip(xs, ys):
        # coeficiente del término independiente
        c_ind += xi * yi

        # coeficiente del parámetro 1
        c_1 += xi * xi

        # coeficiente del parámetro 0
        c_0 += xi

    return (c_1, c_0, c_ind)
```

1.2 A) Interpole los puntos:

$p_1 = (5.4, 3.2)$

$p2_i = (9.5, 0.7)$

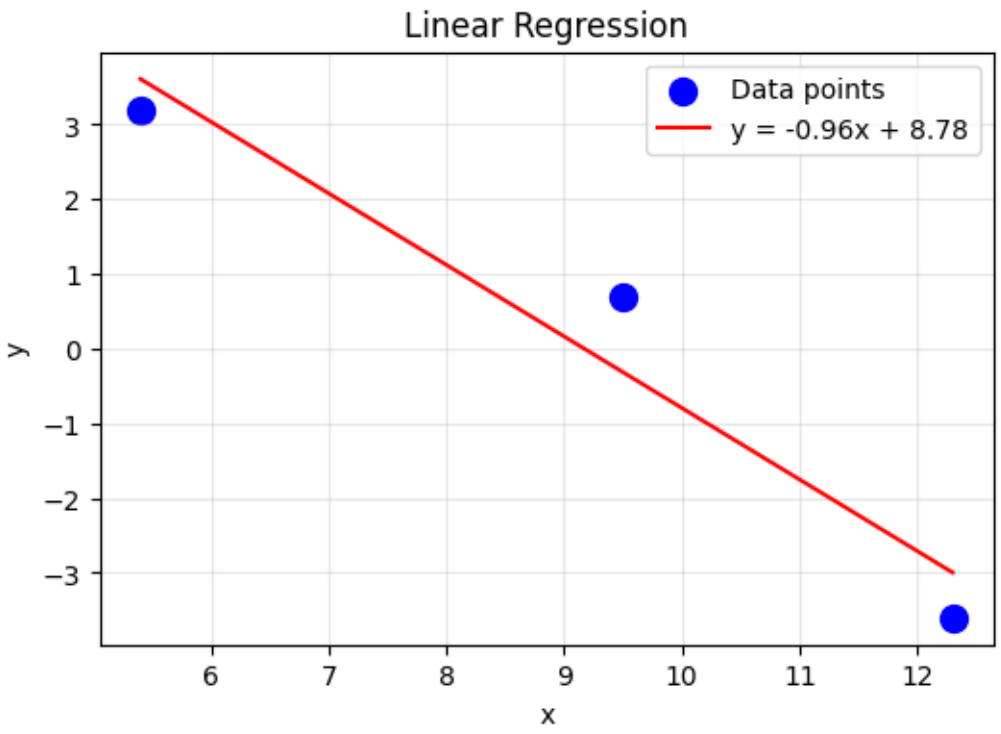
$p3 = (12.3, -3.6)$

[2]:
```python
from src import ajustar_min_cuadrados

m, b = ajustar_min_cuadrados(
    xs=[5.4, 9.5, 12.3], ys=[3.2, 0.7, -3.6], gradiente=[der_parcial_1,␣
 ↪der_parcial_0]
)
print(f"m: {m}, b: {b}")
```

```
[12-15 22:23:20][INFO] User| 2025-12-15 22:23:20.831414
```

```
[12-15 22:23:20][INFO] Se ajustarán 2 parámetros.
m: -0.9577913091613628, b: 8.783974536396357
[12-15 22:23:20][INFO] Se ajustarán 2 parámetros.
m: -0.9577913091613628, b: 8.783974536396357
```

[3]:
```python
import numpy as np

import matplotlib.pyplot as plt

xs = [5.4, 9.5, 12.3]
ys = [3.2, 0.7, -3.6]

# Create the plot
plt.figure(figsize=(6, 4))

# Plot the original data points
plt.scatter(xs, ys, color="blue", label="Data points", s=100)

# Plot the fitted line
x_line = np.linspace(min(xs), max(xs), 100)
y_line = m * x_line + b
plt.plot(x_line, y_line, color="red", label=f"y = {m:.2f}x + {b:.2f}")

plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Regression")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```



1.3 B) Conjunto de datos 1

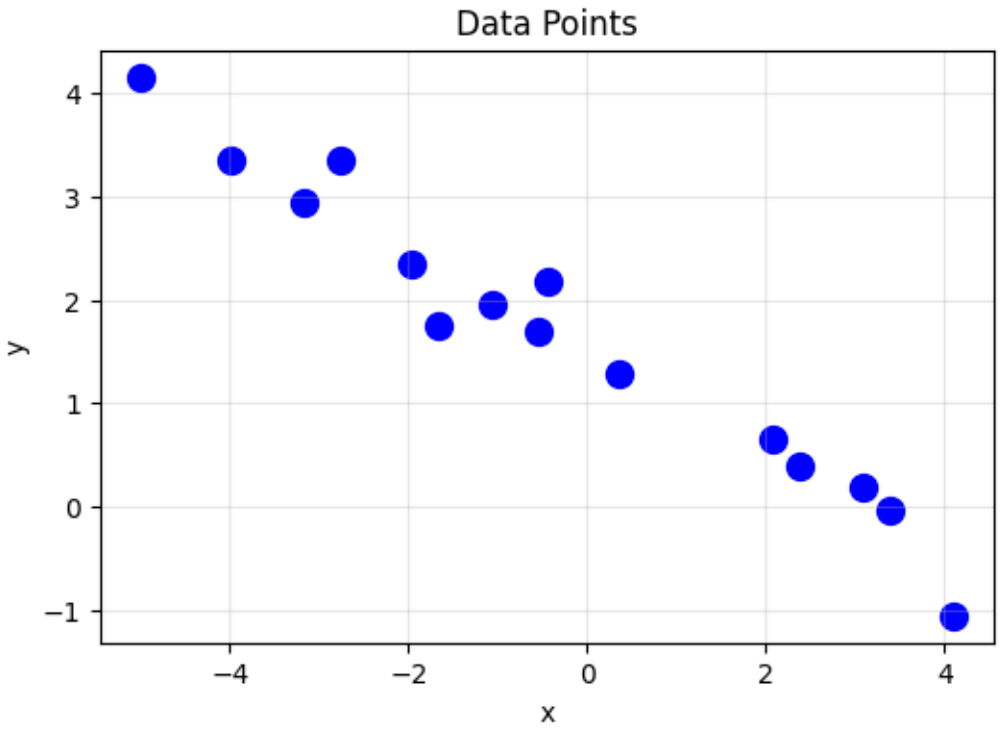
[4]:
```
xs = [
    3.38,
    0.35,
    2.07,
    -0.45,
    -0.55,
    -1.06,
    -2.77,
    3.08,
    -3.98,
    -5.0,
    -3.18,
    -1.96,
    2.37,
    -1.66,
    4.09,
]

ys = [
    -0.04,
    1.28,
```

```
    0.65,
    2.18,
    1.70,
    1.96,
    3.36,
    0.18,
    3.35,
    4.16,
    2.95,
    2.34,
    0.38,
    1.75,
    -1.06,
]
```

[5]:
```
len(xs), len(ys)
```

[5]:
```
(15, 15)
```

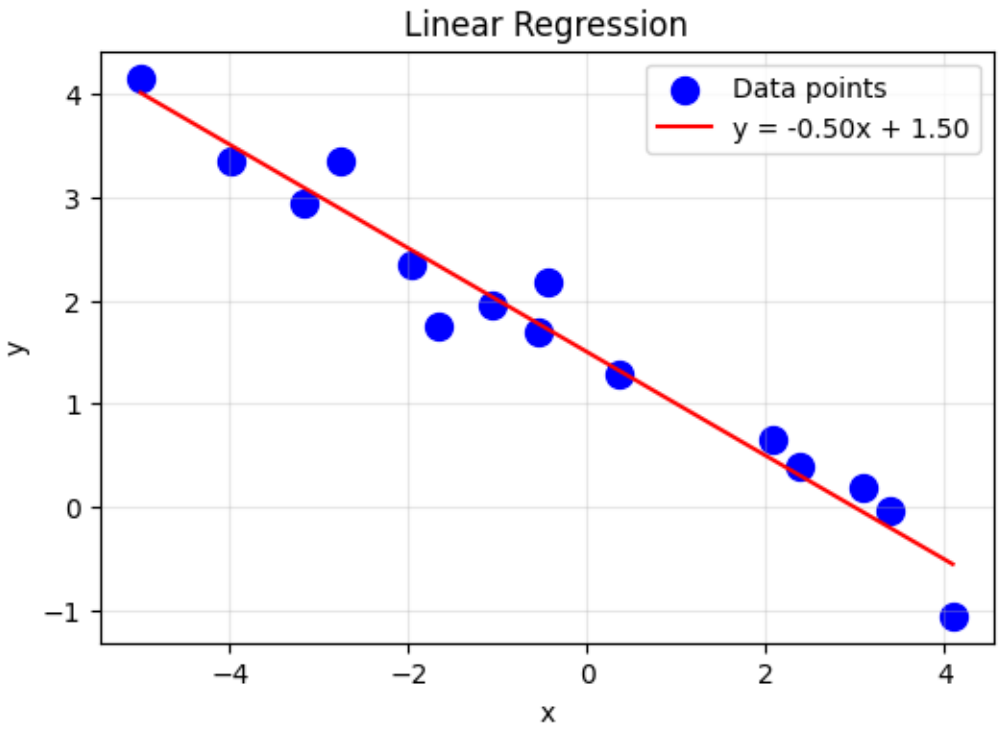
[6]:
```
plt.figure(figsize=(6, 4))
plt.scatter(xs, ys, color="blue", s=100)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Data Points")
plt.grid(True, alpha=0.3)
plt.show()
```



[7]:
```
from src import ajustar_min_cuadrados

m, b = ajustar_min_cuadrados(xs, ys, gradiente=[der_parcial_1, der_parcial_0])
print(f"m: {m}, b: {b}")
```

```
[12-15 22:23:21][INFO] Se ajustarán 2 parámetros.
m: -0.503232585436377, b: 1.4991976183166862
m: -0.503232585436377, b: 1.4991976183166862
```

[8]:
```
plt.figure(figsize=(6, 4))

plt.scatter(xs, ys, color="blue", label="Data points", s=100)

x_line = np.linspace(min(xs), max(xs), 100)
y_line = m * x_line + b
plt.plot(x_line, y_line, color="red", label=f"y = {m:.2f}x + {b:.2f}")

plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Regression")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

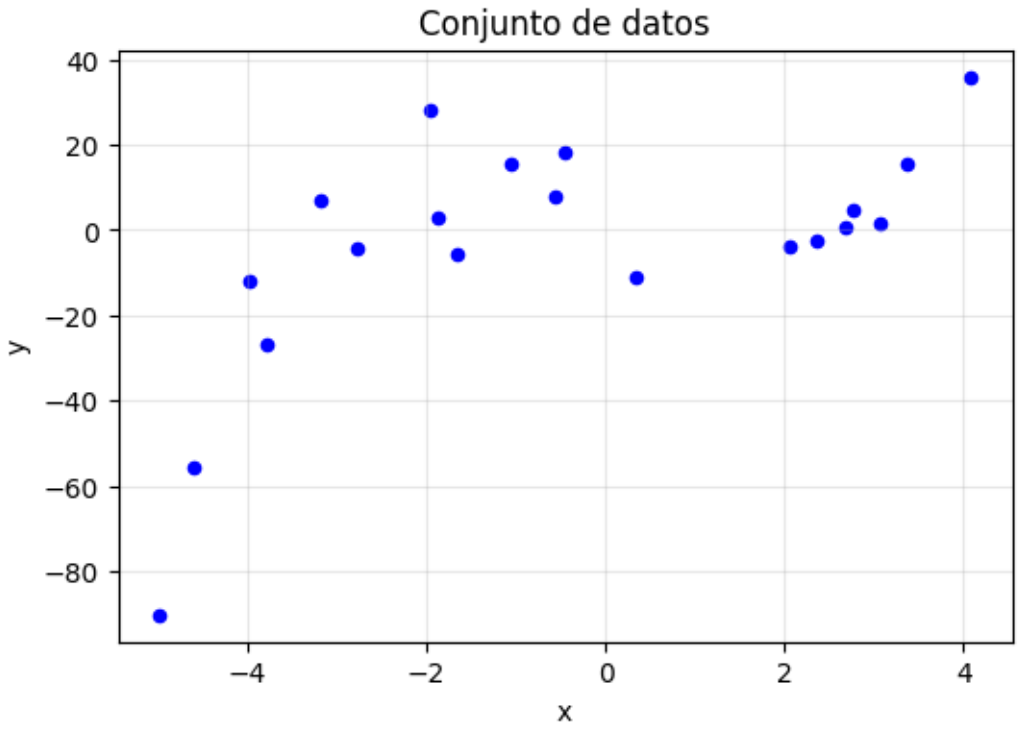


1.4 C) Conjunto de datos 2

[9]:
```
xs = [
    3.38,
    0.35,
    2.07,
    -0.45,
    -0.56,
    -1.06,
    -2.78,
    3.08,
    -3.99,
    -5.00,
    -3.18,
    -1.97,
    2.37,
    -1.67,
    4.09,
    -4.60,
    2.68,
    2.78,
    -3.79,
    -1.87,
]
ys = [
    15.65,
    -11.20,
    -4.00,
    18.03,
    7.94,
    15.32,
    -4.40,
    1.39,
    -11.92,
    -90.24,
    6.92,
    28.35,
    -2.41,
    -5.47,
    35.91,
    -55.53,
    0.77,
    4.79,
    -27.05,
```

```
    2.85,
]
```

[10]:
```
plt.figure(figsize=(6, 4))
plt.scatter(xs, ys, color="blue", s=20)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Conjunto de datos")
plt.grid(True, alpha=0.3)
plt.show()
```

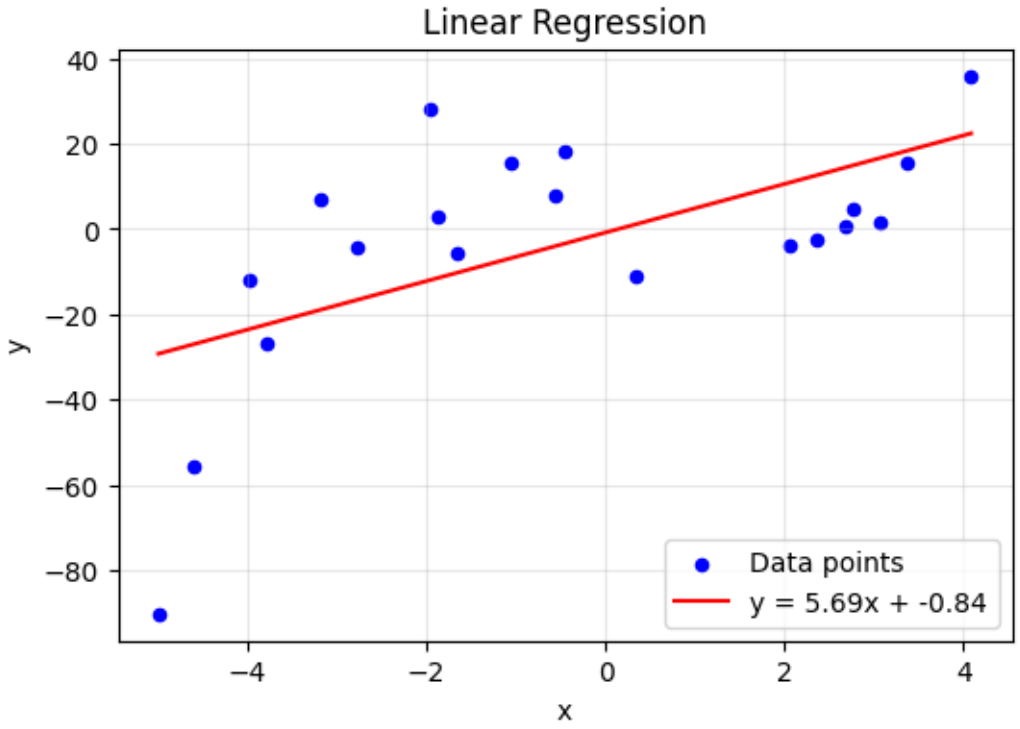


[11]:
```
m, b = ajustar_min_cuadrados(xs, ys, gradiente=[der_parcial_1, der_parcial_0])
print(f"m: {m}, b: {b}")
```

```
[12-15 22:23:21][INFO] Se ajustarán 2 parámetros.
m: 5.686665564419458, b: -0.8375472244037535
m: 5.686665564419458, b: -0.8375472244037535
```

[12]:
```
plt.figure(figsize=(6, 4))

plt.scatter(xs, ys, color="blue", label="Data points", s=20)

x_line = np.linspace(min(xs), max(xs), 100)
```

```
y_line = m * x_line + b
plt.plot(x_line, y_line, color="red", label=f"y = {m:.2f}x + {b:.2f}")

plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Regression")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```



Mejor regresión lineal con una expresión cuadratica

[13]:
```
def derivada_parcial_a(xs, ys):
    xs = np.array(xs)
    ys = np.array(ys)
    return [
        np.sum(xs**6),
        np.sum(xs**5),
        np.sum(xs**4),
        np.sum(xs**3),
        np.sum(xs**3 * ys)
    ]
```

```python
def derivada_parcial_b(xs, ys):
    xs = np.array(xs)
    ys = np.array(ys)
    return [
        np.sum(xs**5),
        np.sum(xs**4),
        np.sum(xs**3),
        np.sum(xs**2),
        np.sum(xs**2 * ys)
    ]

def derivada_parcial_c(xs, ys):
    xs = np.array(xs)
    ys = np.array(ys)
    return [
        np.sum(xs**4),
        np.sum(xs**3),
        np.sum(xs**2),
        np.sum(xs),
        np.sum(xs * ys)
    ]

def derivada_parcial_d(xs, ys):
    xs = np.array(xs)
    ys = np.array(ys)
    return [
        np.sum(xs**3),
        np.sum(xs**2),
        np.sum(xs),
        len(xs),
        np.sum(ys)
    ]
```

[14]:
```python
a, b, c, d = ajustar_min_cuadrados(
    xs,
    ys,
    gradiente=[
        derivada_parcial_a,
        derivada_parcial_b,
        derivada_parcial_c,
        derivada_parcial_d
    ]
)

print(f"a={a}, b={b}, c={c}, d={d}")
```

```
[12-15 22:23:21][INFO] Se ajustarán 4 parámetros.
a=1.044977916824698, b=0.03132740575354256, c=-8.880663970075506,
```

```
d=2.762289919876904
a=1.044977916824698, b=0.03132740575354256, c=-8.880663970075506,
d=2.762289919876904
```

[15]:
```
plt.figure(figsize=(6, 3))
plt.scatter(xs, ys, color="blue", label="Puntos", s=20)

x_line = np.linspace(min(xs), max(xs), 400)
y_line = a*x_line**3 + b*x_line**2 + c*x_line + d

plt.plot(x_line, y_line, color="red",label=f"y = {a:.2f}x³ + {b:.2f}x² + {c:.
↪2f}x + {d:.2f}")

plt.xlabel("x")
plt.ylabel("y")
plt.title("Regresion cúbica")
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```