ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA EN SISTEMAS

METODOS NUMERICOS ICCD412

JUAN FRANCISCO PINTO ANDRANGO

Taller 05 Minimos Cuadrados

GR1CC

FECHA DE ENTREGA 15 DE DICIEMBRE DEL 2025

In [2]:
```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)


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)
```

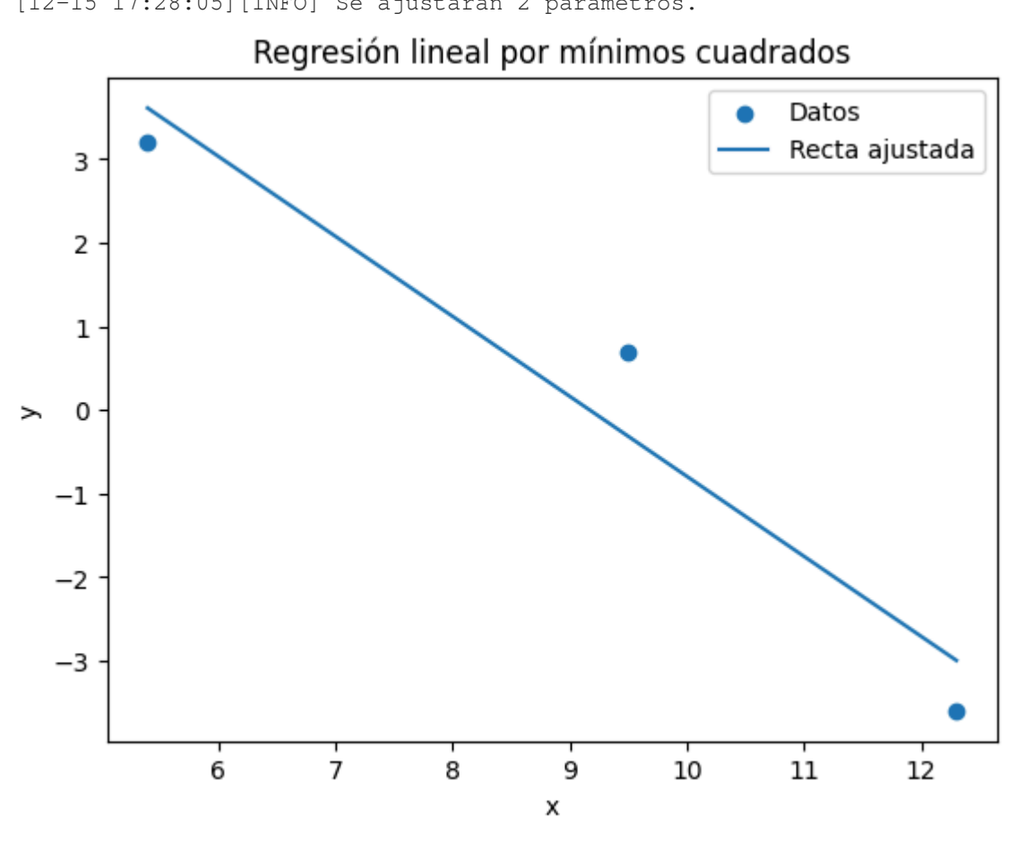
In [3]:
```python
from src import ajustar_min_cuadrados

# UTILICE:
ajustar_min_cuadrados([5.4,9.5,12.3], [3.2,0.7,-3.6], [der_parcial_1, der_parcial_0])
```

```
[12-15 17:27:57][INFO] Se ajustarán 2 parámetros.
```

Out[3]:
```
array([-0.95779131,  8.78397454])
```

In [4]:
```python
import numpy as np
import matplotlib.pyplot as plt

from src import ajustar_min_cuadrados

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

# Ajuste por mínimos cuadrados
a1, a0 = ajustar_min_cuadrados(xs, ys, [der_parcial_1, der_parcial_0])

# Función ajustada
def f(x):
    return a1 * x + a0

# Valores para la recta
x_linea = np.linspace(min(xs), max(xs), 100)
y_linea = f(x_linea)

# Gráfica
plt.figure()
plt.scatter(xs, ys, label="Datos")
plt.plot(x_linea, y_linea, label="Recta ajustada")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Regresión lineal por mínimos cuadrados")
plt.legend()
plt.show()
```

```
[12-15 17:28:05][INFO] Se ajustarán 2 parámetros.
```



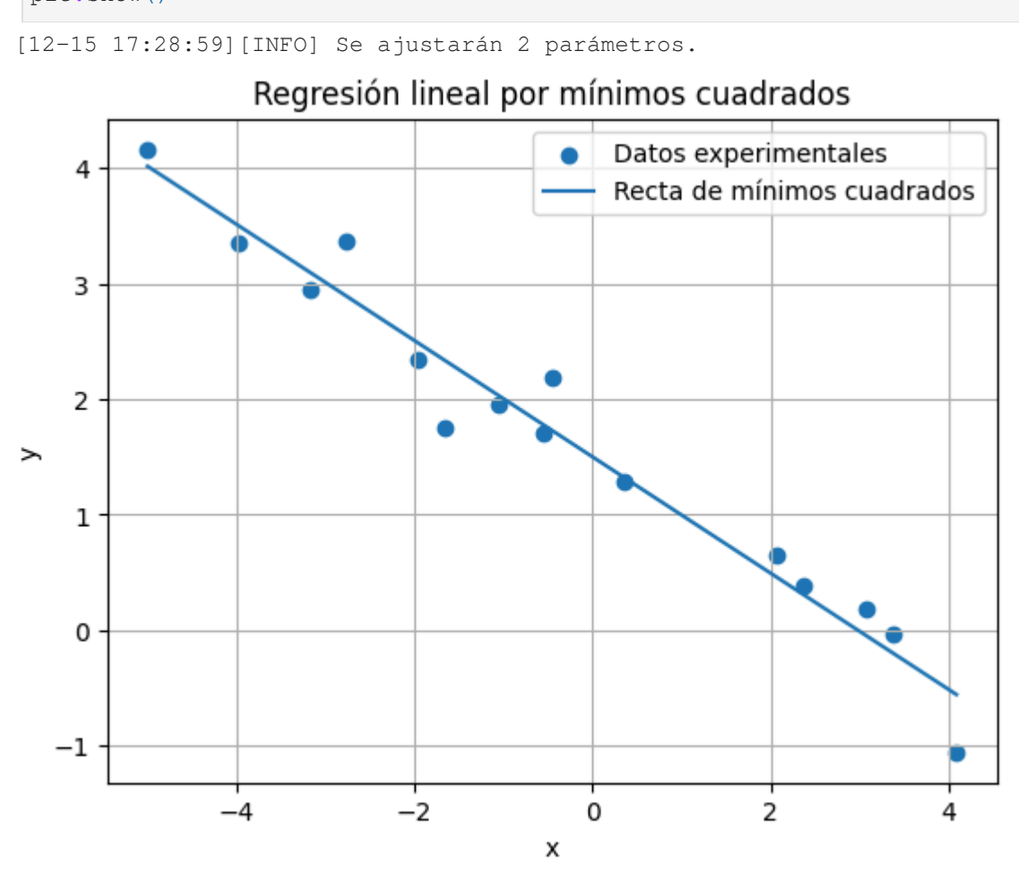
In [5]:
```python
from src import ajustar_min_cuadrados

# UTILICE:
ajustar_min_cuadrados([3.38,
0.35,
2.07,
-0.45,
-0.55,
-1.06,
-2.77,
3.08,
-3.98,
-5,
-3.18,
-1.96,
2.37,
-1.66,
4.09], [-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], [der_parcial_1, der_parcial_0])
```

```
[12-15 17:28:15][INFO] Se ajustarán 2 parámetros.
```

Out[5]:
```
array([-0.50323259,  1.49919762])
```

In [6]:
```python
import numpy as np
import matplotlib.pyplot as plt

from src import ajustar_min_cuadrados

# Datos
xs = [
    3.38, 0.35, 2.07, -0.45, -0.55, -1.06, -2.77,
    3.08, -3.98, -5, -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
]

# Ajuste por mínimos cuadrados
a1, a0 = ajustar_min_cuadrados(xs, ys, [der_parcial_1, der_parcial_0])

# Función ajustada
def f(x):
    return a1 * x + a0

# Valores para la recta
x_linea = np.linspace(min(xs), max(xs), 200)
y_linea = f(x_linea)

# Gráfica
plt.figure()
plt.scatter(xs, ys, label="Datos experimentales")
plt.plot(x_linea, y_linea, label="Recta de mínimos cuadrados")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Regresión lineal por mínimos cuadrados")
plt.legend()
plt.grid(True)
plt.show()
```

```
[12-15 17:28:59][INFO] Se ajustarán 2 parámetros.
```



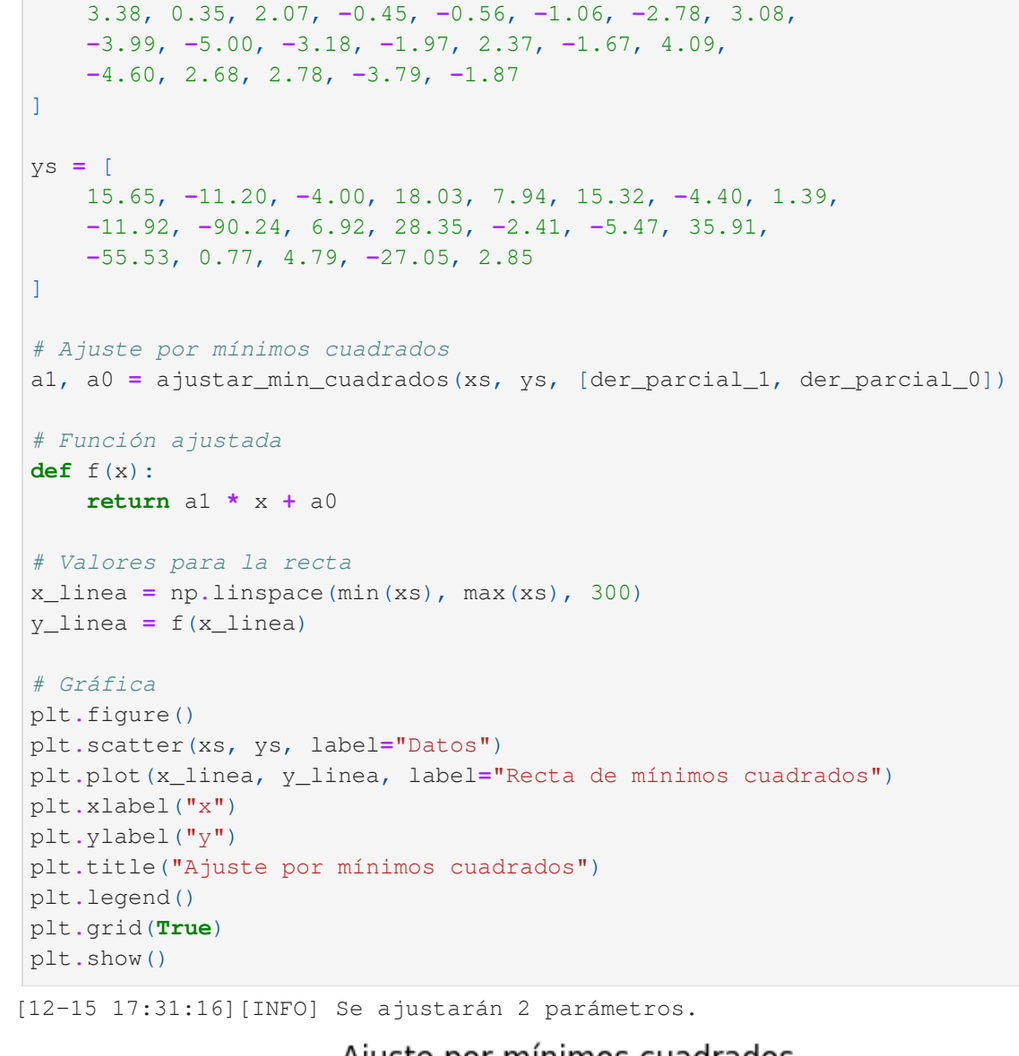
In [7]:
```python
from src import ajustar_min_cuadrados

# UTILICE:
ajustar_min_cuadrados(
    [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],
    [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],
    [der_parcial_1, der_parcial_0]
)
```

```
[12-15 17:30:51][INFO] Se ajustarán 2 parámetros.
```

Out[7]:
```
array([ 5.68666556, -0.83754722])
```

In [8]:
```python
import numpy as np
import matplotlib.pyplot as plt

from src import ajustar_min_cuadrados

# Datos
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
]

# Ajuste por mínimos cuadrados
a1, a0 = ajustar_min_cuadrados(xs, ys, [der_parcial_1, der_parcial_0])

# Función ajustada
def f(x):
    return a1 * x + a0

# Valores para la recta
x_linea = np.linspace(min(xs), max(xs), 300)
y_linea = f(x_linea)

# Gráfica
plt.figure()
plt.scatter(xs, ys, label="Datos")
plt.plot(x_linea, y_linea, label="Recta de mínimos cuadrados")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ajuste por mínimos cuadrados")
plt.legend()
plt.grid(True)
plt.show()
```

```
[12-15 17:31:16][INFO] Se ajustarán 2 parámetros.
```



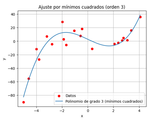
Para una funcion de ornden 3

In [6]:
```python
# Regresión polinómica de orden 3 (mínimos cuadrados)
# Derivadas parciales y ecuaciones normales
# ==================================================

def der_parcial_0_orden3(xs: list, ys: list) -> tuple[float, float, float, float, float]:
    """
    Derivada parcial respecto a a_0
    c_3*a_3 + c_2*a_2 + c_1*a_1 + c_0*a_0 = c_ind
    """
    c_3 = sum(x**3 for x in xs)
    c_2 = sum(x**2 for x in xs)
    c_1 = sum(xs)
    c_0 = len(xs)
    c_ind = sum(ys)

    return (c_3, c_2, c_1, c_0, c_ind)


def der_parcial_1_orden3(xs: list, ys: list) -> tuple[float, float, float, float, float]:
    """
    Derivada parcial respecto a a_1
    """
    c_3 = sum(x**4 for x in xs)
    c_2 = sum(x**3 for x in xs)
    c_1 = sum(x**2 for x in xs)
    c_0 = sum(xs)
    c_ind = sum(x * y for x, y in zip(xs, ys))

    return (c_3, c_2, c_1, c_0, c_ind)


def der_parcial_2_orden3(xs: list, ys: list) -> tuple[float, float, float, float, float]:
    """
    Derivada parcial respecto a a_2
    """
    c_3 = sum(x**5 for x in xs)
    c_2 = sum(x**4 for x in xs)
    c_1 = sum(x**3 for x in xs)
    c_0 = sum(x**2 for x in xs)
    c_ind = sum((x**2) * y for x, y in zip(xs, ys))

    return (c_3, c_2, c_1, c_0, c_ind)


def der_parcial_3_orden3(xs: list, ys: list) -> tuple[float, float, float, float, float]:
    """
    Derivada parcial respecto a a_3
    """
    c_3 = sum(x**6 for x in xs)
    c_2 = sum(x**5 for x in xs)
    c_1 = sum(x**4 for x in xs)
    c_0 = sum(x**3 for x in xs)
    c_ind = sum((x**3) * y for x, y in zip(xs, ys))

    return (c_3, c_2, c_1, c_0, c_ind)


# ####################################################
# Graficar ajuste por mínimos cuadrados (orden 3)
# ####################################################
import numpy as np
import matplotlib.pyplot as plt
from typing import Callable

def ajustar_min_cuadrados_orden3(
    xs: list[float],
    ys: list[float],
    gradiente: list[Callable[[list[float], list[float]], tuple]],
) -> np.ndarray:
    """
    Resuelve el sistema de ecuaciones normales del método de mínimos cuadrados
    usando las derivadas parciales suministradas.

    Cada función en `gradiente` debe retornar una tupla de la forma:
    (c_n, c_{n-1}, ..., c_0, c_ind)

    donde:
        c_n * a_n + ... + c_0 * a_0 = c_ind
    """

    assert len(xs) == len(ys), "xs y ys deben tener la misma longitud."

    num_pars = len(gradiente)

    # Matriz aumentada [A | b]
    Ab = np.zeros((num_pars, num_pars + 1), dtype=float)

    for i, der_parcial in enumerate(gradiente):
        assert callable(der_parcial), "Cada derivada parcial debe ser una función."
        Ab[i, :] = der_parcial(xs, ys)

    # Separar A y b
    A = Ab[:, :-1]
    b = Ab[:, -1]

    # Resolver sistema lineal
    return np.linalg.solve(A, b)
```

In [7]:
```python
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
]

# --------------------------------------------------
# Ajuste por mínimos cuadrados (polinomio grado 3)
# --------------------------------------------------
a3, a2, a1, a0 = ajustar_min_cuadrados_orden3(
    xs,
    ys,
    [der_parcial_3_orden3, der_parcial_2_orden3, der_parcial_1_orden3, der_parcial_0_orden3]
)
print(f"\n valor a3: {a3}, valor a2: {a2}, valor a1: {a1}, valor a0: {a0}")
print(f"\ny = {a3}x^3 + {a2}x^2 + {a1}x + {a0}")
```

```
 valor a3: 1.0449779168247026, valor a2: 0.03132740575355141, valor a1: -8.880663970075563, valor a0: 2.7622899198768462

y = 1.0449779168247026x^3 + 0.03132740575355141x^2 + -8.880663970075563x + 2.7622899198768462
```

In [8]:
```python
# --------------------------------------------------
# Datos
# --------------------------------------------------
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
]

# --------------------------------------------------
# Ajuste por mínimos cuadrados (polinomio grado 3)
# --------------------------------------------------
a3, a2, a1, a0 = ajustar_min_cuadrados_orden3(
    xs,
    ys,
    [der_parcial_3_orden3, der_parcial_2_orden3, der_parcial_1_orden3, der_parcial_0_orden3]
)

# --------------------------------------------------
# Función ajustada
# --------------------------------------------------
def f(x):
    return a3 * x**3 + a2 * x**2 + a1 * x + a0

# --------------------------------------------------
# Valores para la curva
# --------------------------------------------------
x_curva = np.linspace(min(xs), max(xs), 300)
y_curva = f(x_curva)

# --------------------------------------------------
# Gráfica
# --------------------------------------------------
plt.figure()
plt.scatter(xs, ys, label="Datos", color='red')
plt.plot(x_curva, y_curva, label="Polinomio de grado 3 (mínimos cuadrados)")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ajuste por mínimos cuadrados (orden 3)")
plt.legend()
plt.grid(True)
plt.show()
```



Link-Del-Repositorio-de-Git_hub