

ESCUELA POLITECNICA NACIONAL

Materia: Métodos Numéricos

Nombre: Stiv Quishpe

link repositorio

<https://github.com/stiv001/CorreccionB2/blob/b91c06cac097be9b7262307622c8317a59019ac3/Correccion/MN24B-prueba02/Ejercicios-determinante.ipynb>

Ejercicio Mínimos cuadrados

Interpole los siguientes conjuntos de datos con la función correspondiente.

```
# 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
     $c_1 * a_1 + c_0 * a_0 = c_{ind}$ 

    ## 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_ind = sum(ys)
    c_1 = sum(xs)
    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
     $c_1 * a_1 + c_0 * a_0 = c_{ind}$ 

    ## 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):
        c_ind += xi * yi
        c_1 += xi * xi
        c_0 += xi

    return (c_1, c_0, c_ind)

```

Conjunto de datos de ejemplo

```
xs = [  
    -5.0000,  
    -3.8889,  
    -2.7778,  
    -1.6667,  
    -0.5556,  
    0.5556,  
    1.6667,  
    2.7778,  
    3.8889,  
    5.0000,  
]  
ys = [  
    -12.7292,  
    -7.5775,  
    -7.7390,  
    -4.1646,  
    -4.5382,  
    2.2048,  
    4.3369,  
    2.2227,  
    9.0625,  
    7.1860,  
]
```

```
from src import ajustar_min_cuadrados # no modificar esta función  
  
pars = ajustar_min_cuadrados(xs, ys, gradiente=[der_parcial_0, der_parcial_1])
```

```
[01-11 11:46:53] [INFO] 2025-01-11 11:46:53.918058  
[01-11 11:46:53] [INFO] 2025-01-11 11:46:53.921568  
[01-11 11:46:53] [INFO] Se ajustarán 2 parámetros.  
[01-11 11:46:53] [INFO]  
[[101.8525926    0.          209.87476711]  
 [  0.           10.         -11.7356    ]]
```

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

```

m, b = ajustar_min_cuadrados(xs, ys, gradiente=[der_parcial_0, der_parcial_1])

x = np.linspace(-5, 5, 100)

y = [m * xi + b for xi in x]

plt.scatter(xs, ys, label="Datos")
plt.plot(x, y, color="red", label=r"$ y = a_1 x + a_0 $" )
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ajuste por mínimos cuadrados ")
plt.legend()
plt.show()

```

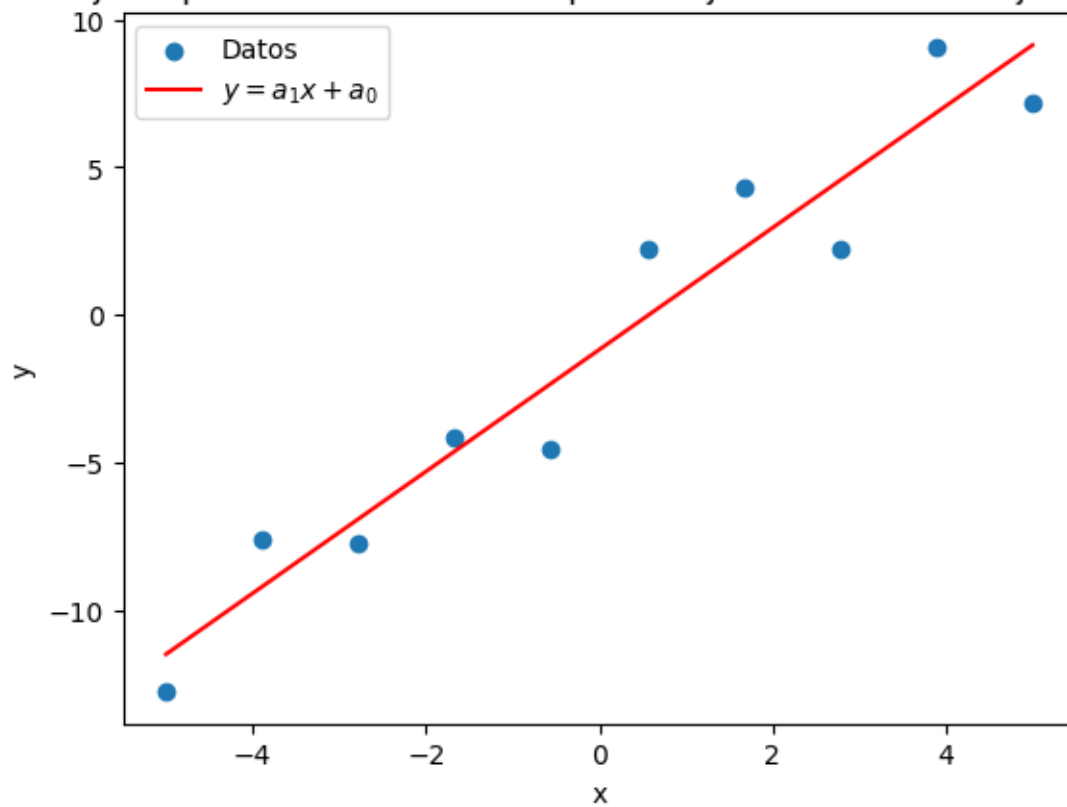
[01-11 11:46:56] [INFO] Se ajustarán 2 parámetros.

[01-11 11:46:56] [INFO]

[[101.8525926 0. 209.87476711]

[0. 10. -11.7356]]

Ajuste por mínimos cuadrados para conjunto de datos de ejemplo



Conjunto de datos 1

```
xs1 = [  
    -5.0000,  
    -3.8889,  
    -2.7778,  
    -1.6667,  
    -0.5556,  
     0.5556,  
     1.6667,  
     2.7778,  
     3.8889,  
     5.0000,  
]  
ys1 = [
```

```

57.2441,
33.0303,
16.4817,
7.0299,
0.5498,
0.7117,
3.4185,
12.1767,
24.9167,
44.2495,

```

```

]
```

```

def der_parcial_2(xs: list, ys: list) -> tuple[float, float, float, float]:
    """Retorna los coeficientes de la ecuación de la derivada parcial con respecto al parámetro 2.

    
$$c_2 * a_2 + c_1 * a_1 + c_0 * a_0 = c_{ind}$$


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

    ## Return
    ``c_2``: coeficiente del parámetro 2.
    ``c_1``: coeficiente del parámetro 1.
    ``c_0``: coeficiente del parámetro 0.
    ``c_ind``: coeficiente del término independiente.
    """
    c_2 = sum(xi**4 for xi in xs)
    c_1 = sum(xi**3 for xi in xs)
    c_0 = sum(xi**2 for xi in xs)
    c_ind = sum(yi * xi**2 for yi, xi in zip(ys, xs))
    return (c_2, c_1, c_0, c_ind)

def der_parcial_1(xs: list, ys: list) -> tuple[float, float, float, float]:
    """Retorna los coeficientes de la ecuación de la derivada parcial con respecto al parámetro 1.

    
$$c_2 * a_2 + c_1 * a_1 + c_0 * a_0 = c_{ind}$$


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

```

```

## Return
``c_2``: coeficiente del parámetro 2.
``c_1``: coeficiente del parámetro 1.
``c_0``: coeficiente del parámetro 0.
``c_ind``: coeficiente del término independiente.
"""

c_2 = sum(xi**3 for xi in xs)
c_1 = sum(xi**2 for xi in xs)
c_0 = sum(xi for xi in xs)
c_ind = sum(yi * xi for yi, xi in zip(ys, xs))
return (c_2, c_1, c_0, c_ind)

def der_parcial_0(xs: list, ys: list) -> tuple[float, float, float, float]:
    """Retorna los coeficientes de la ecuación de la derivada parcial con respecto al parámetro 0.

    
$$c_2 * a_2 + c_1 * a_1 + c_0 * a_0 = c_{ind}$$


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

    ## Return
    ``c_2``: coeficiente del parámetro 2.
    ``c_1``: coeficiente del parámetro 1.
    ``c_0``: coeficiente del parámetro 0.
    ``c_ind``: coeficiente del término independiente.
    """

    c_2 = sum(xi**2 for xi in xs)
    c_1 = sum(xi for xi in xs)
    c_0 = len(xs)
    c_ind = sum(ys)
    return (c_2, c_1, c_0, c_ind)

def parabola(x: float, pars: tuple[float, float, float]) -> float:
    """Ecuación de la parábola  $y = a_2 * x^2 + a_1 * x + a_0$ .

    ## Parameters
    ``x``: valor de x.
    ``pars``: parámetros de la parábola. Deben ser de la forma (a2, a1, a0).

    ## Return
    ``y``: valor de y.
    """

```

```

"""
a2, a1, a0 = pars
return a2 * x**2 + a1 * x + a0

xs2 = [
    -5.0000,
    -3.8889,
    -2.7778,
    -1.6667,
    -0.5556,
    0.5556,
    1.6667,
    2.7778,
    3.8889,
    5.0000,
]
ys2 = [
    57.2441,
    33.0303,
    16.4817,
    7.0299,
    0.5498,
    0.7117,
    3.4185,
    12.1767,
    24.9167,
    44.2495,
]

from src import ajustar_min_cuadrados
pars2 = ajustar_min_cuadrados(xs2, ys2, gradiente=[der_parcial_2, der_parcial_1, der_parcial_0],
pars2

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 100)
y = [parabola(xi, pars2) for xi in x]

plt.scatter(xs2, ys2, label="Datos")
plt.plot(x, y, color="red", label=r"$ y = a_2 x^2 + a_1 x + a_0 $" )
plt.xlabel("x")

```



```

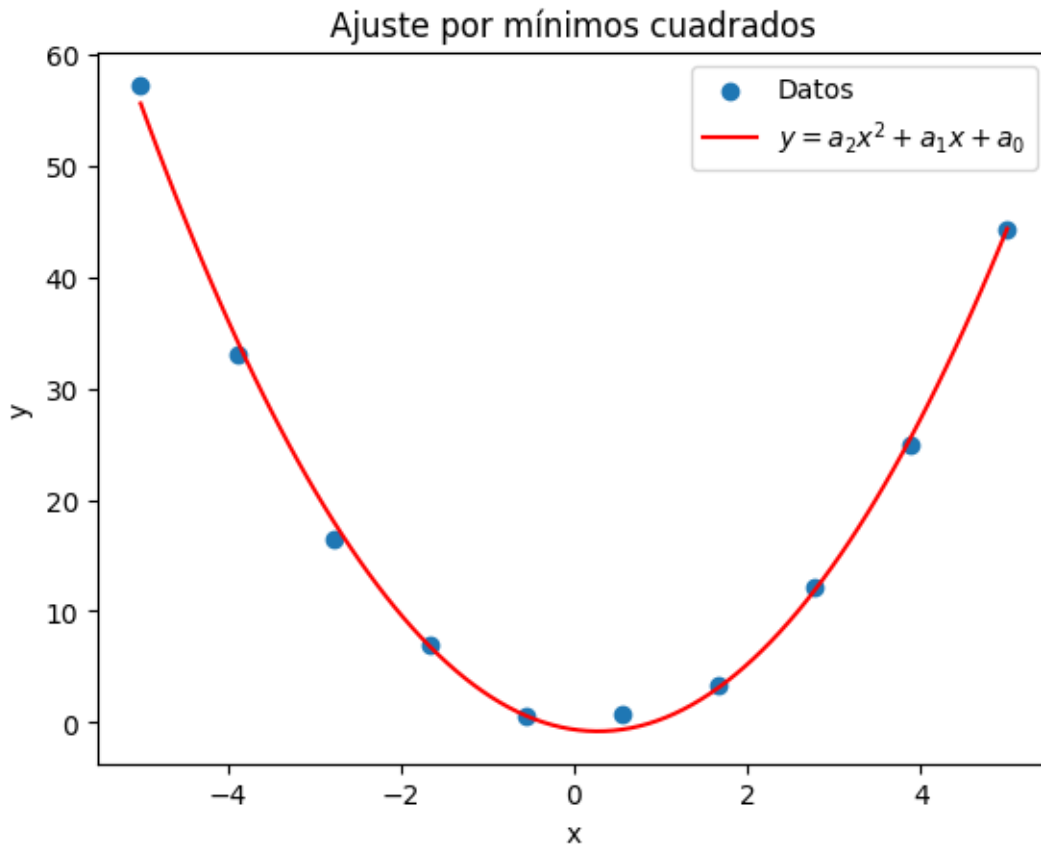
plt.ylabel("y")
plt.title("Ajuste por mínimos cuadrados")
plt.legend()
plt.show()
a2, a1, a0 = pars2
print(f"Coeficiente a2: {a2}")
print(f"Coeficiente a1: {a1}")
print(f"Coeficiente a0: {a0}")

```

```

[01-11 12:23:03] [INFO] Se ajustarán 3 parámetros.
[01-11 12:23:03] [INFO]
[[ 1.01852593e+02  0.00000000e+00  1.00000000e+01  1.99808900e+02]
 [ 0.00000000e+00  1.01852593e+02  0.00000000e+00 -1.14413577e+02]
 [-2.27373675e-13  0.00000000e+00 -7.90113041e+01  5.04294087e+01]]
[01-11 12:23:03] [INFO]
[[ 1.01852593e+02  0.00000000e+00  1.00000000e+01  1.99808900e+02]
 [ 0.00000000e+00  1.01852593e+02  0.00000000e+00 -1.14413577e+02]
 [-2.27373675e-13  0.00000000e+00 -7.90113041e+01  5.04294087e+01]]

```



Coeficiente a2: 2.024410482925083
Coeficiente a1: -1.1233251295755429
Coeficiente a0: -0.6382556172537739

```
import numpy as np

a2 = 2.024410482925083
a1 = -1.1233251295755429
a0 = -0.6382556172537739

x_val = 2.25
y_eval = a2 * x_val**2 + a1 * x_val + a0
print(f"El valor de y para x = {x_val} es: {y_eval}")
```

El valor de y para x = 2.25 es: 7.082840911009487

```
x_val = -2.25
y_eval = a2 * x_val**2 + a1 * x_val + a0
print(f"El valor de y para x = {x_val} es: {y_eval}")
```

El valor de y para x = -2.25 es: 12.13780399409943

```
import numpy as np

a2 = 2.024410482925083
a1 = -1.1233251295755429
a0 = -0.6382556172537739

y_val = -2.25

coefficients = [a2, a1, a0 - y_val]
roots = np.roots(coefficients)

roots
```

```
array([0.277445+0.84804434j, 0.277445-0.84804434j])
```

Interpole el conjunto de datos 1 usando la función cuadrática.

Conjunto de datos 2

```
xs2 = [
    0.0003,
    0.0822,
    0.2770,
    0.4212,
    0.4403,
    0.5588,
    0.5943,
    0.6134,
    0.9070,
    1.0367,
    1.1903,
    1.2511,
```

```

1.2519,
1.2576,
1.6165,
1.6761,
2.0114,
2.0557,
2.1610,
2.6344,
]
ys2 = [
1.1017,
1.5021,
0.3844,
1.3251,
1.7206,
1.9453,
0.3894,
0.3328,
1.2887,
3.1239,
2.1778,
3.1078,
4.1856,
3.3640,
6.0330,
5.8088,
10.5890,
11.5865,
11.8221,
26.5077,
]

```

```

import numpy as np
from scipy.optimize import curve_fit

import matplotlib.pyplot as plt

def func_exp(x, a, b):
    return a * np.exp(b * x)

popt, pcov = curve_fit(func_exp, xs2, ys2)

```

```

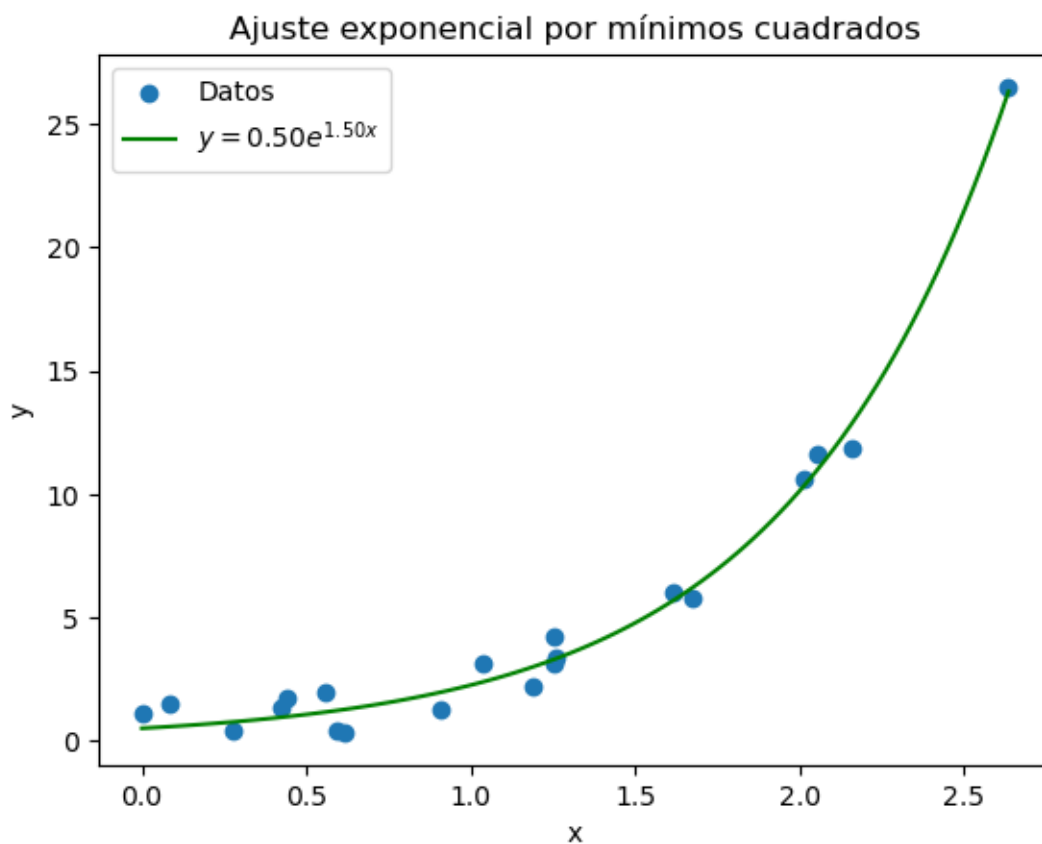
a, b = popt

x_fit = np.linspace(min(xs2), max(xs2), 100)

y_fit = func_exp(x_fit, a, b)

plt.scatter(xs2, ys2, label="Datos")
plt.plot(x_fit, y_fit, color="g", label=f"$y = {a:.2f} e^{{b:.2f} x}$")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Ajuste exponencial por mínimos cuadrados")
plt.legend()
plt.show()
x_val = 5
y_val = func_exp(x_val, a, b)
print(f"El valor de y para x = {x_val} es: {y_val}")

```



El valor de y para x = 5 es: 924.5446340088477

```
x_val = 1
y_val = func_exp(x_val, a, b)
print(f"El valor de y para x = {x_val} es: {y_val}")
```

El valor de y para x = 1 es: 2.2518190038794166

Interpole el conjunto de datos 2 usando la función exponencial.