1)

Desarrolla un programa que solicite al usuario el nombre de un archivo de Excel en donde se tengan almacenados puntos experimentales; el programa deberá ser capaz de cargar esos datos, graficarlos y calcular el polinomio de Lagrange asociado tanto de manera compacta usando las funciones asociadas de MATLAB, como de manera procedimental; los resultados serán mostrados en pantalla y almacenados en un nuevo archivo de Excel.

In [2]:
```
# Importar librerias
import sympy as sp
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from openpyxl import Workbook
from numpy.polynomial import Polynomial
import warnings

warnings.filterwarnings("ignore")
```

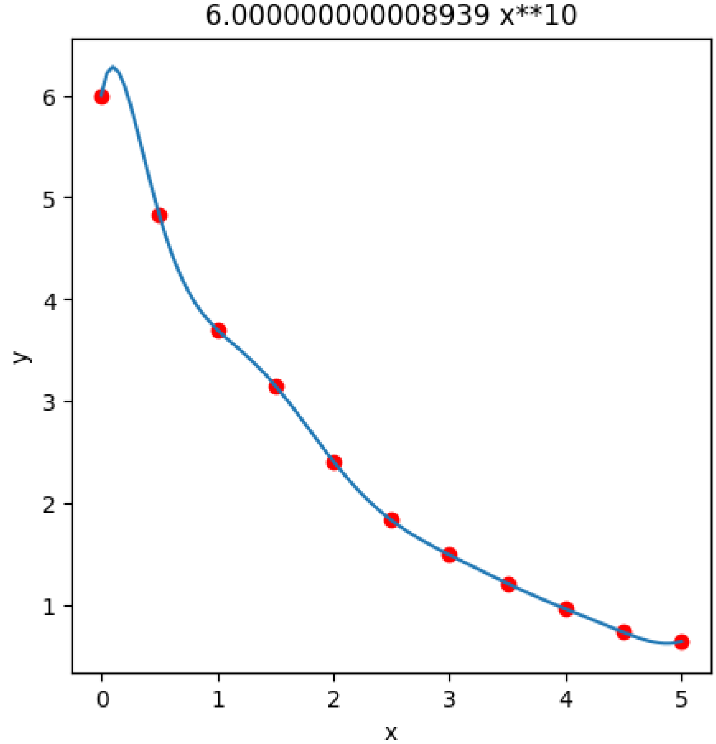
Leer datos de exel ¶

In [3]:
```
df = pd.read_excel("data.xlsx")

# Pedirle nombre del exel al usuario y cargarlo
while False:
    try:
        exel = input("Nombre del archivo de exel (data.xlsx)")
        df = pd.read_excel(exel)
        break
    except:
        print("Nombre invalido ")


# Leer puntos del exel
x = df["x"].values
y = df["y"].values
```

Usando polyfit()

In [4]:
```
INTERVAL = np.linspace(0, 5, 100)

# Calcular el polinomio de Lagrange con la funcion polyfit
p = np.polyfit(x, y, deg=len(x) - 1)

# Crear plot
fig, ax = plt.subplots(figsize=(5, 5))

# Configuracion de plot
ax.set(title=Polynomial(p), xlabel="x", ylabel="y")

# Plotear puntos de exel
ax.scatter(x, y, c="r")

# Plotear puntos con el polinomio de Lagrange
ax.plot(
    INTERVAL,
    np.polyval(p, INTERVAL),
)

fig.show()
```

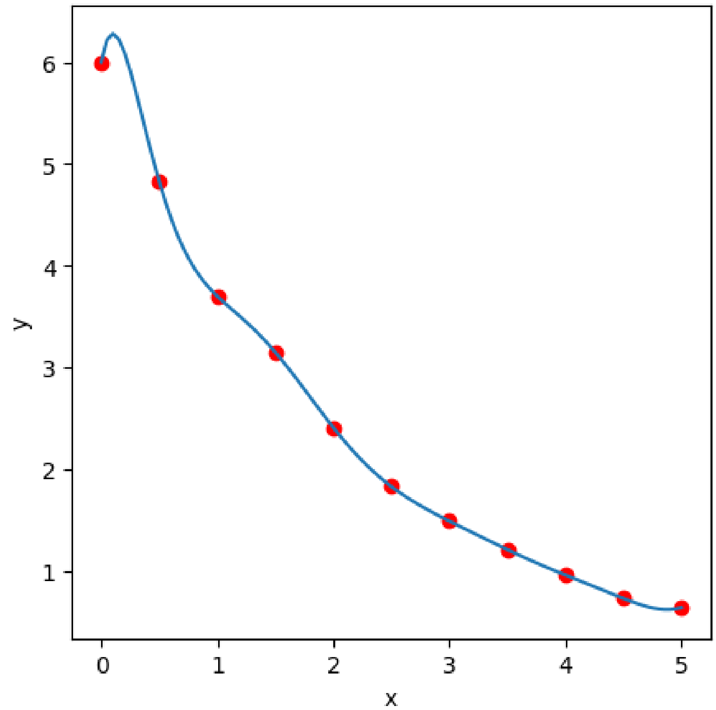
-0.0013037037032453652 + 0.038194003516085574 x**1 - 0.48253968242128487 x**2 + 3.434941798243592 x**3 - 15.09024444191331 x**4 + 42.074796290477096 x**5 - 73.5887962878591 x**6 + 76.52075043351839 x**7 - 41.45211586946449 x**8 + 6.246317459586915 x**9 + 6.000000000008939 x**10



De manera procedimental

In [5]:
```
def lagrange(eval):
    n = len(x)  # Número de puntos
    result = 0  # Resultado final

    for i in range(n):
        term = y[i]  # Término actual del polinomio de Lagrange
        for j in range(n):
            if j != i:
                term = (
                    term * (eval - x[j]) / (x[i] - x[j])
                )  # Producto de términos individuales

        result += term  # Agregar el término al resultado final

    return result  # Devolver el resultado


# Crear plot
fig, ax = plt.subplots(figsize=(5, 5))

# Configuracion de plot
ax.set(title=Polynomial(p), xlabel="x", ylabel="y")

# Plotear puntos de exel
ax.scatter(x, y, c="r")

# Plotear puntos con el polinomio de Lagrange
ax.plot(
    INTERVAL,
    list(
        map(lagrange, INTERVAL),
    ),
)

fig.show()
```

-0.0013037037032453652 + 0.038194003516085574 x**1 - 0.48253968242128487 x**2 + 3.434941798243592 x**3 - 15.09024444191331 x**4 + 42.074796290477096 x**5 - 73.5887962878591 x**6 + 76.52075043351839 x**7 - 41.45211586946449 x**8 + 6.246317459586915 x**9 + 6.000000000008939 x**10



Salvar resultados a exel

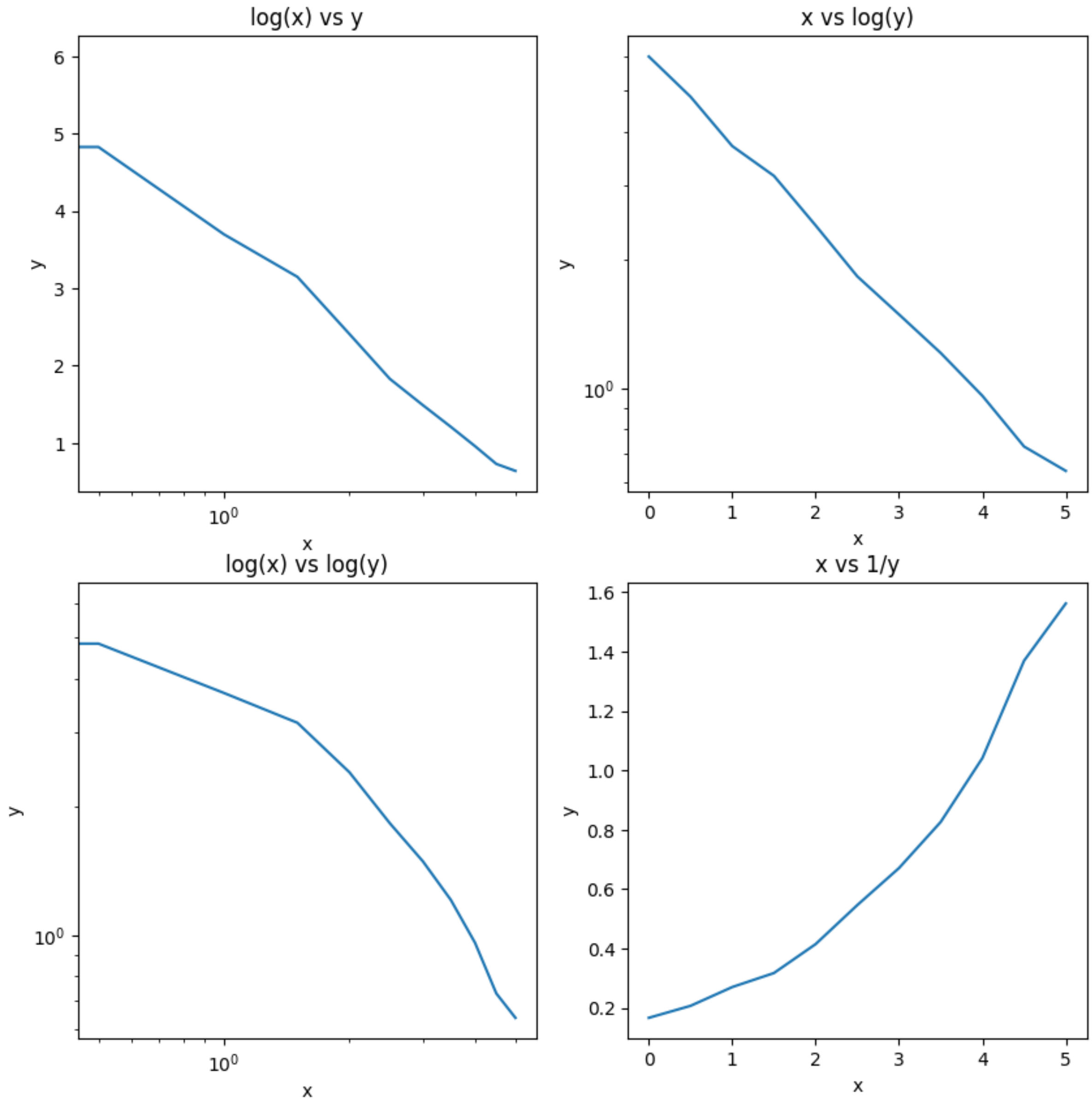
In [6]:
```
# Guardar los resultados en el archivo de Excel
pd.DataFrame({"x": INTERVAL, "y": list(map(lagrange, INTERVAL))}).to_excel(
    "output1.xlsx", index=False
)
```

2)

Realiza un programa que haga lo mismo que el punto anterior, pero que sea capaz de ajustar los datos a diversos modelos funcionales (potencias, logaritmos, exponenciales y funciones recíprocas) Los resultados deben de poderse almacenar en un archivo de Excel, de igual manera deben de graficarse, tanto los puntos experimentales como los ajustes, con etiquetas que indiquen la ecuación del modelo obtenido. El programa, a petición del usuario, debe ser capaz de dar un valor interpolado o extrapolado en base a un valor de x dado por el usuario.

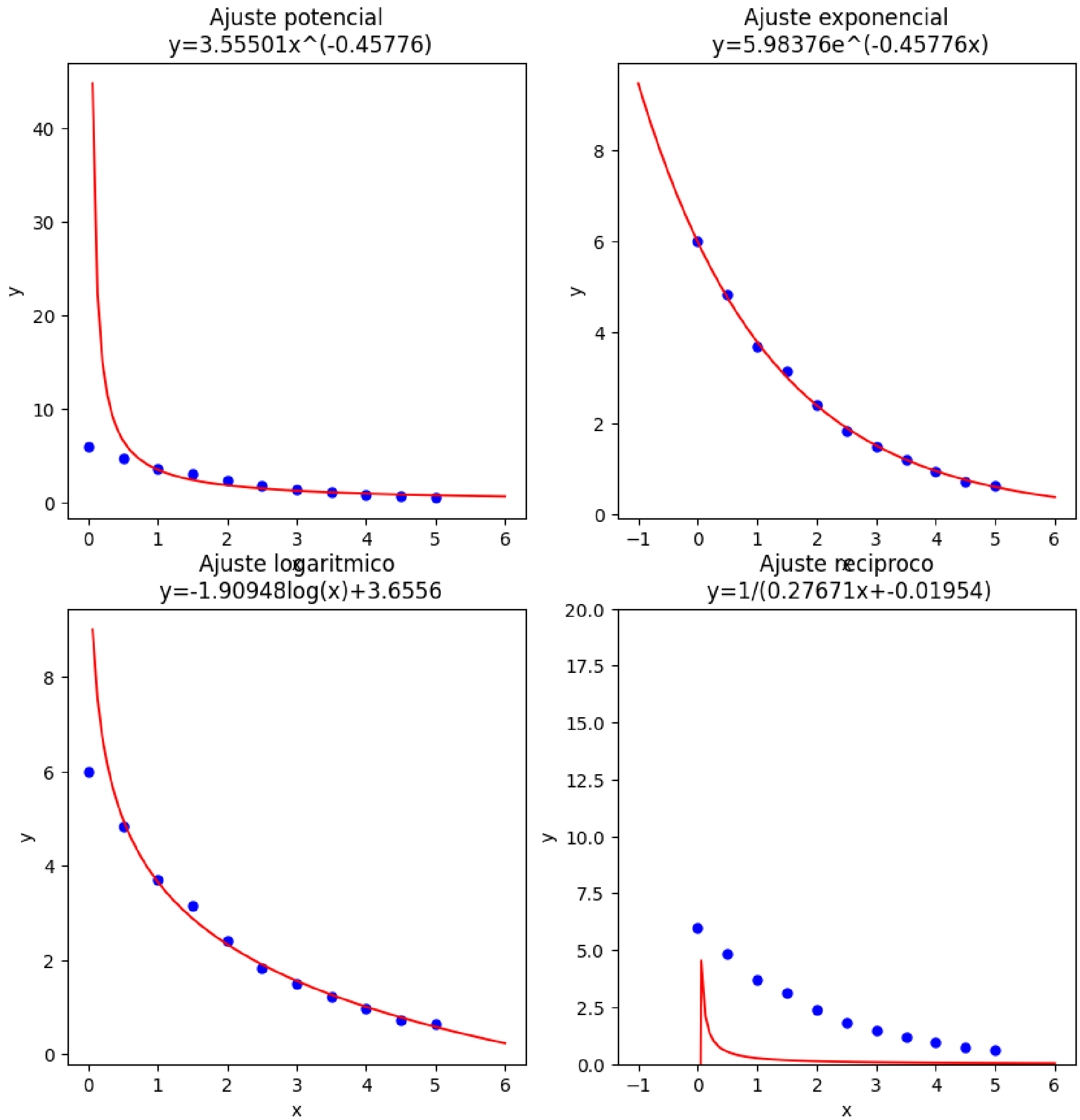
In [7]:
```
df = pd.read_excel("data.xlsx")

# Pedirle nombre del exel al usuario y cargarlo
while False:
    try:
        exel = input("Nombre del archivo de exel (data.xlsx)")
        df = pd.read_excel(exel)
        break
    except:
        print("Nombre invalido ")


# Leer puntos del exel
x = df["x"].values
y = df["y"].values
```

In [8]:
```
# Crear una figura con 2 filas y 2 columnas
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))

# Gráfico 1: log(x) vs y
ax[0, 0].set(title="log(x) vs y", xlabel="x", ylabel="y")
ax[0, 0].semilogx(x, y)

# Gráfico 2: x vs log(y)
ax[0, 1].set(title="x vs log(y)", xlabel="x", ylabel="y")
ax[0, 1].semilogy(x, y)

# Gráfico 3: log(x) vs log(y)
ax[1, 0].set(title="log(x) vs log(y)", xlabel="x", ylabel="y")
ax[1, 0].loglog(x, y)

# Gráfico 4: x vs 1/y
ax[1, 1].set(title="x vs 1/y", xlabel="x", ylabel="y")
ax[1, 1].plot(x, 1 / y)


fig.show()
```


log(x) vs y
x vs log(y)
log(x) vs log(y)
x vs 1/y
y
x
10^0
1
2
3
4
5
6
0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

In [9]:
```
# Ajuste potencial
p_po = np.polyfit(np.log(x[(x > 0) & (y > 0)]), np.log(y[(x > 0) & (y > 0)]),
1)
m_po = p_po[0]
b_po = np.exp(p_po[1])
x_po = np.linspace(-1, 6, 100)
y_po = b_po * x_po**m_po

# Ajuste exponencial
p_ex = np.polyfit(x[(x > 0) & (y > 0)], np.log(y[(x > 0) & (y > 0)]), 1)
m_ex = p_ex[0]
b_ex = np.exp(p_ex[1])
x_ex = np.linspace(-1, 6, 100)
y_ex = b_ex * np.exp(m_ex * x_ex)

# Ajuste logaritmico
p_lo = np.polyfit(np.log(x[(x > 0) & (y > 0)]), y[(x > 0) & (y > 0)], 1)
m_lo = p_lo[0]
b_lo = p_lo[1]
x_lo = np.linspace(-1, 6, 100)
y_lo = b_lo + m_lo * np.log(x_lo)

# Ajuste reciproco
p_re = np.polyfit(x[y != 0], 1 / y[y != 0], 1)
m_re = p_re[0]
b_re = p_re[1]
x_re = np.linspace(-1, 6, 100)
y_re = m_re / x_re + b_re


# Crear una figura con 2 filas y 2 columnas
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))

# Gráfico 1: Ajuste potencial
ax[0, 0].set(
    title=f"Ajuste potencial\n y={round(b_po,5)}x^({round(m_ex,5)})",
    xlabel="x",
    ylabel="y",
)
ax[0, 0].scatter(x, y, color="blue", marker="o", facecolors="blue", s=25)
ax[0, 0].plot(
    x_po,
    y_po,
    linewidth=1.3,
    color="red",
)


# Gráfico 2: Ajuste exponencial
ax[0, 1].set(
    title=f"Ajuste exponencial\n y={round(b_ex,5)}e^({round(m_ex,5)}x)",
    xlabel="x",
    ylabel="y",
)
ax[0, 1].scatter(x, y, color="blue", marker="o", facecolors="blue", s=25)
ax[0, 1].plot(
```

```
    x_ex,
    y_ex,
    linewidth=1.3,
    color="red",
)


# Gráfico 3: Ajuste logaritmico
ax[1, 0].set(
    title=f"Ajuste logaritmico\n y={round(m_lo,5)}log(x)+{round(b_lo,5)}",
    xlabel="x",
    ylabel="y",
)
ax[1, 0].scatter(x, y, color="blue", marker="o", facecolors="blue", s=25)
ax[1, 0].plot(
    x_lo,
    y_lo,
    linewidth=1.3,
    color="red",
)


# Gráfico 4: Ajuste reciproco
ax[1, 1].set(
    title=f"Ajuste reciproco\n y=1/({round(m_re,5)}x+{round(b_re,5)})",
    xlabel="x",
    ylabel="y",
    ylim=(0, 20),
)

ax[1, 1].scatter(x, y, color="blue", marker="o", facecolors="blue", s=25)
ax[1, 1].plot(
    x_re,
    y_re,
    linewidth=1.3,
    color="red",
)

fig.show()
```



Salvar resultados a exel

In [10]:
```
# Guardar los resultados de el ajuste exponencial ya que es el que mejor se aj
usta
pd.DataFrame({"x": x_ex, "y": y_ex}).to_excel("output2.xlsx", index=False)
```

3)

Construye un programa que ajuste datos leídos de Excel mediante el método de regresión por mínimos cuadrados. Tu programa debe de graficar la recta producida, mostrar los valores de los factores de ajuste, así como sus incertidumbres y graficar los residuos. Usando los datos de la siguiente tabla, comprueba el funcionamiento de tu programa:

x	-5	-4	-1	1	4	6	9	10
y	12	10	6	2	-3	-6	-11	-12
Δx	1	0.8	0.7	0.5	1.2	2.7	3.3	4.5
Δy	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Leer datos de Exel

In [11]:

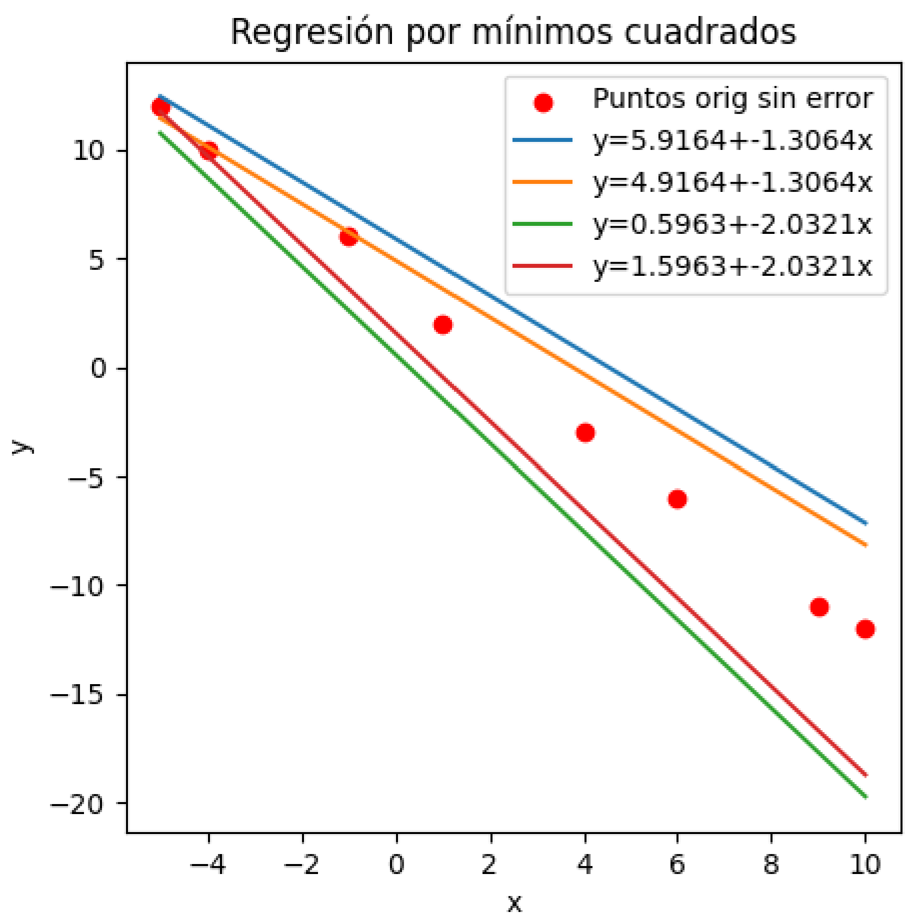
```
# Leer y extraer datos del exel
df = pd.read_excel("data2.xlsx")
data_x = df["x"].values
data_y = df["y"].values
data_err_x = df["dx"].values
data_err_y = df["dy"].values


# Preparar datos
sets = [
    list(zip(data_x + data_err_x, data_y + data_err_y)),
    list(zip(data_x + data_err_x, data_y - data_err_y)),
    list(zip(data_x - data_err_x, data_y - data_err_y)),
    list(zip(data_x - data_err_x, data_y + data_err_y)),
]
```

Calcular los valores B

In [12]:
```
# lista para guardar los valoores de b
b_vals = []

# Iterar sobre cada set de puntos
for data_set in sets:
    x_values, y_values = zip(*data_set)  # Separar x, y en 2 variables
    x_mat = np.ones((len(data_set), 2))  # Crear matriz con puros 1
    x_mat[:, 1] = x_values  # Cambiar valores de la 2nda columna a los de x
    x_mat_t = np.transpose(x_mat)  # crear matriz transpuesta

    y_mat = np.array(y_values)  # crear matriz de "y"

    # calcular B
    b = np.matmul(np.linalg.inv(np.matmul(x_mat_t, x_mat)), np.matmul(x_mat_t,
y_mat))

    # agregar los valores de b a la lista
    b_vals.append((b[0], b[1]))
```

Graficar resultados

In [13]:
```
# Crear plot
fig, ax = plt.subplots(figsize=(5, 5))

# Configuracion de plot
ax.set(title="Regresión por mínimos cuadrados", xlabel="x", ylabel="y")

# Plotear puntos de exel
ax.scatter(data_x, data_y, c="r", label="Puntos orig sin error")

# Plotear
for b in b_vals:
    x = np.linspace(np.min(data_x), np.max(data_x), 100)
    y = b[0] + b[1] * x
    ax.plot(x, y, label=f"y={round(b[0],4)}+{round(b[1],4)}x")

# Agregar leyenda a grafica
ax.legend()

fig.show()
```



4)

Investiga y usa la ecuación de Michaelis-Menten para hacer un ajuste por mínimos cuadrados no lineal de datos de concentración de un sustrato S contra la tasa de reacción $rate$ en una reacción mediada mediante una enzima. Construye un programa similar al del inciso anterior que resuelva este problema y pruébalo usando los siguientes datos:

x	-5	-4	-1	1	4	6	9	10
y	12	10	6	2	-3	-6	-11	-12
Δx	1	0.8	0.7	0.5	1.2	2.7	3.3	4.5
Δy	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

In [10]:
```
# Definir datos
s = np.array([0.038, 0.194, 0.425, 0.626, 1.253, 2.500, 3.740])
rate = np.array([0.050, 0.127, 0.094, 0.2122, 0.2729, 0.2665, 0.3317])


# Crear matriz M con los datos de la tabla
M = np.zeros((len(s), 2))
M[:, 0] = s
M[:, 1] = rate


# Valores iniciales de beta
b1 = 1
b2 = 0.5
beta = np.array([[1], [0.5]])


# Crear matrizes J y R
J = np.zeros((len(s), 2))
r = np.zeros(len(s))


# Calcular valor de J y R para cada dato
for i in range(len(s)):
    J[i, 0] = s[i] / (b2 + s[i])
    J[i, 1] = -b1 * s[i] / ((b2 + s[i]) ** 2)
    r[i] = rate[i] - (b1 * s[i]) / (b2 + s[i])


# Cambian las dimensiones de r
r = r.reshape((-1, 1))


# Calcular los nuevos valores de beta
B = beta + np.linalg.inv(J.T.dot(J)).dot(J.T.dot(r))
print(B)


# Plotear resultados
x = np.linspace(0, 5, 1000)
y = B[0] * x / (B[1] + x)
plt.plot(x, y)
plt.show()
```

```
[[0.36070283]
 [0.51736436]]
```