PROYECTO DE INTERPOLACIÓN

DAMIAN OSPINA - 2201296
JEICOR FLOREZ - 2231328
DANIEL MORALES - 2200812

INTERPOLACIÓN DE DATOS PARA UNA ECUACIÓN DE ESTADO

01 DESCRIBIR LOS DATOS

02 REALIZAR LAS INTERPOLACIONES

03 COMPARAR GRAFICAMENTE LA INTERPOLACIÓN

04 GRAFICAR LOS ERRORES DE CADA METODO

05 CONCLUSIÓN

DESCRIBIR LOS DATOS

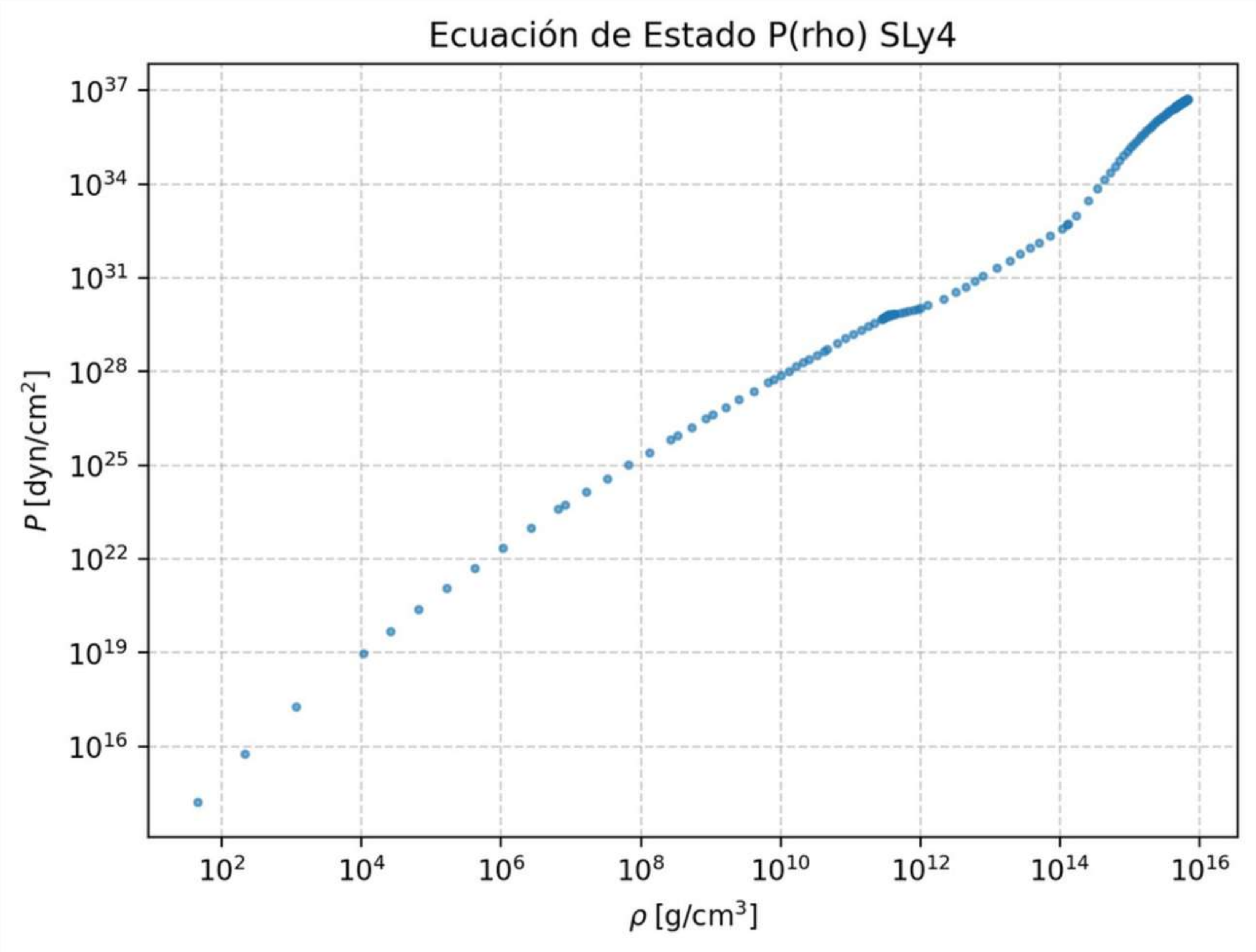
10 primeros datos = "sly4.dat"

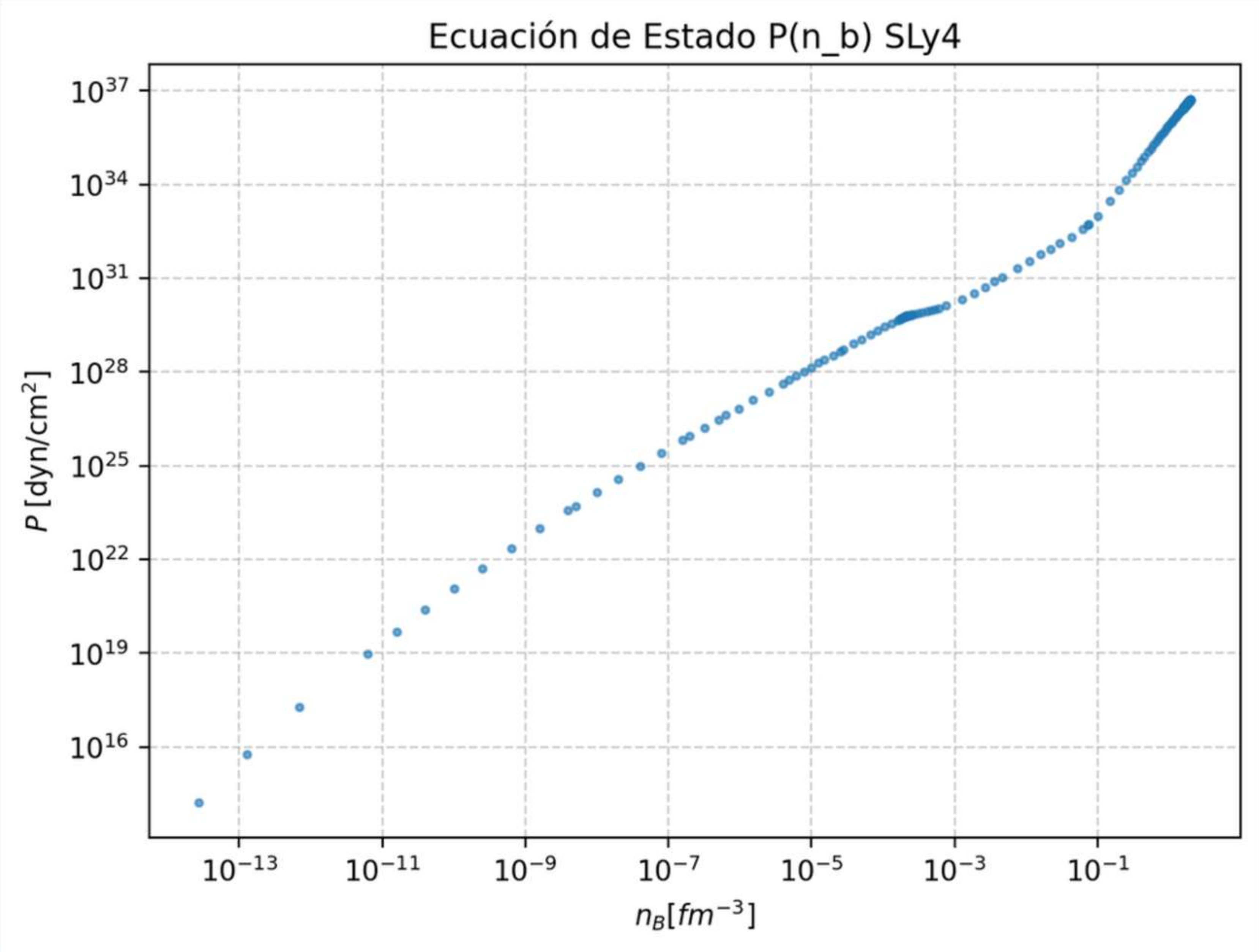
	n_B[fm^{-3}]	rho[g/cm^3]	P[dyn/cm^2]
0	2.720000e-14	45.1	1.700000e+14
1	1.270000e-13	212.0	5.820000e+15
2	6.930000e-13	1150.0	1.900000e+17
3	6.295000e-12	10440.0	9.744000e+18
4	1.581000e-11	26220.0	4.968000e+19
5	3.972000e-11	65870.0	2.431000e+20
6	9.976000e-11	165400.0	1.151000e+21
7	2.506000e-10	415600.0	5.266000e+21
8	6.294000e-10	1044000.0	2.318000e+22
9	1.581000e-09	2622000.0	9.755000e+22

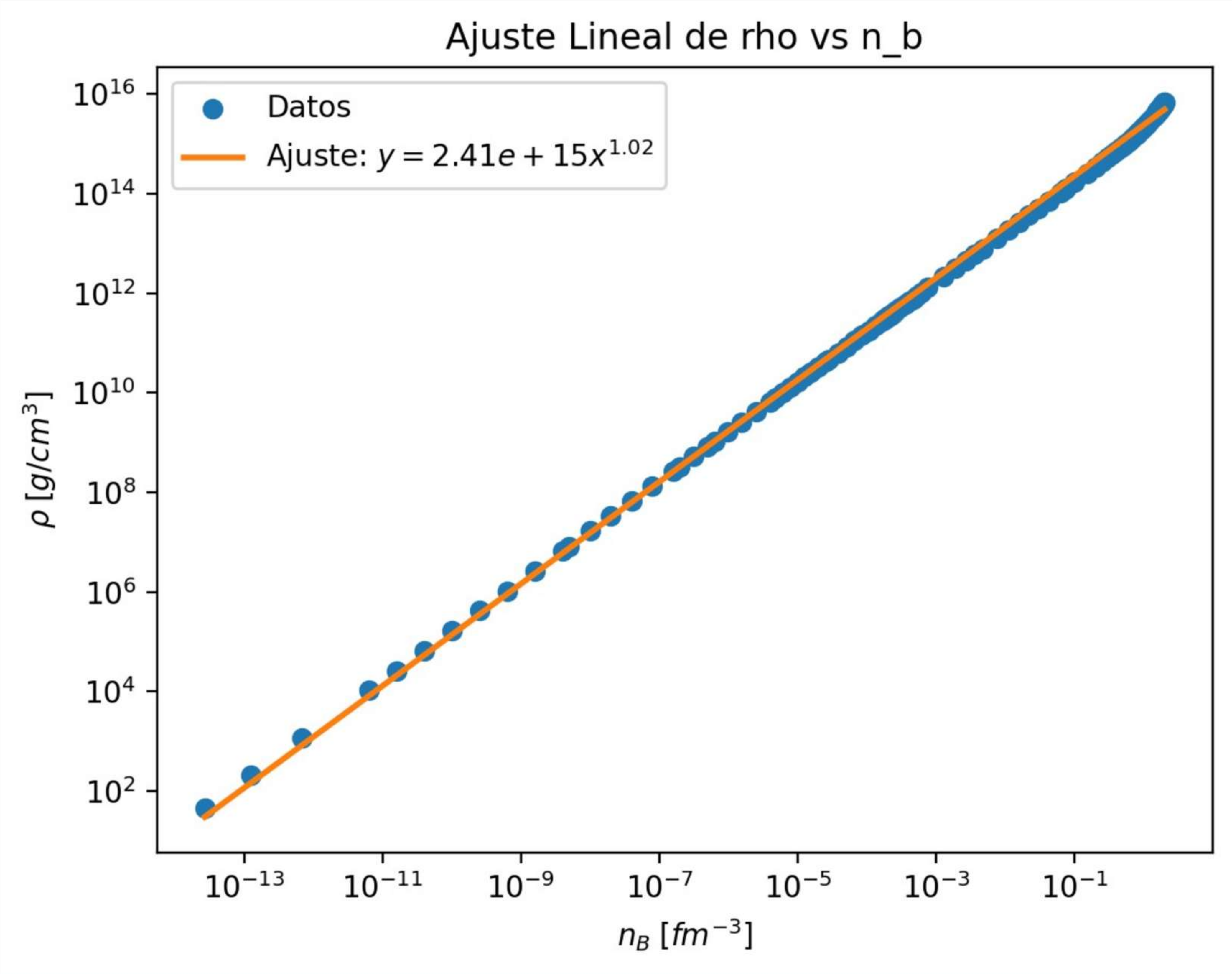
dataframe.describe()

	n_B[fm^{-3}]	rho[g/cm^3]	P[dyn/cm^2]
count	1.520000e+02	1.520000e+02	1.520000e+02
mean	6.912947e-01	1.992758e+15	1.332255e+36
std	7.984834e-01	2.463589e+15	1.829668e+36
min	2.720000e-14	4.510000e+01	1.700000e+14
25%	1.007325e-04	1.682500e+11	2.619750e+29
50%	7.456500e-02	1.260500e+14	5.208500e+32
75%	1.619500e+00	4.617500e+15	3.066750e+36
max	1.997000e+00	6.749000e+15	5.344000e+36

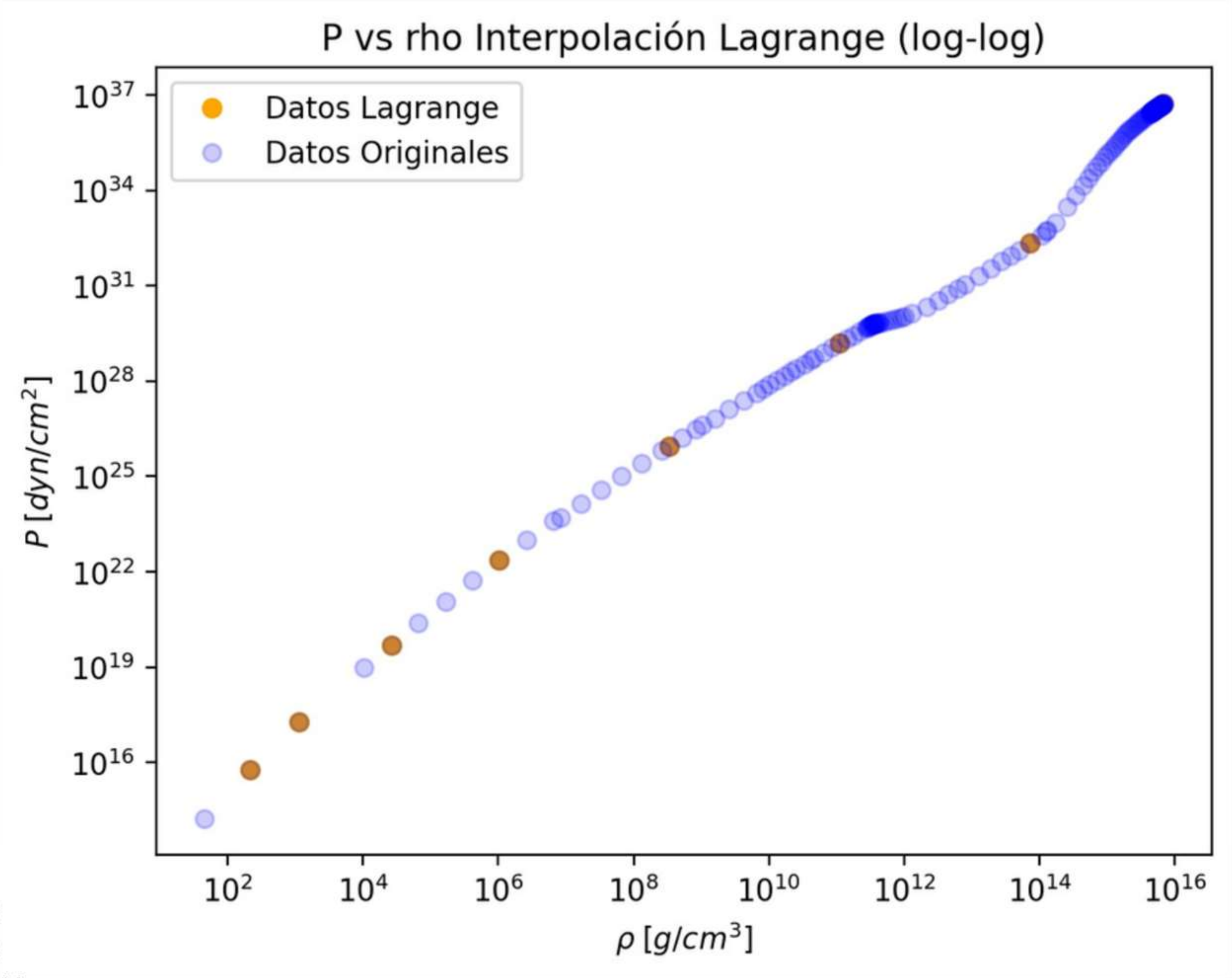
Ecuaciones de Estado

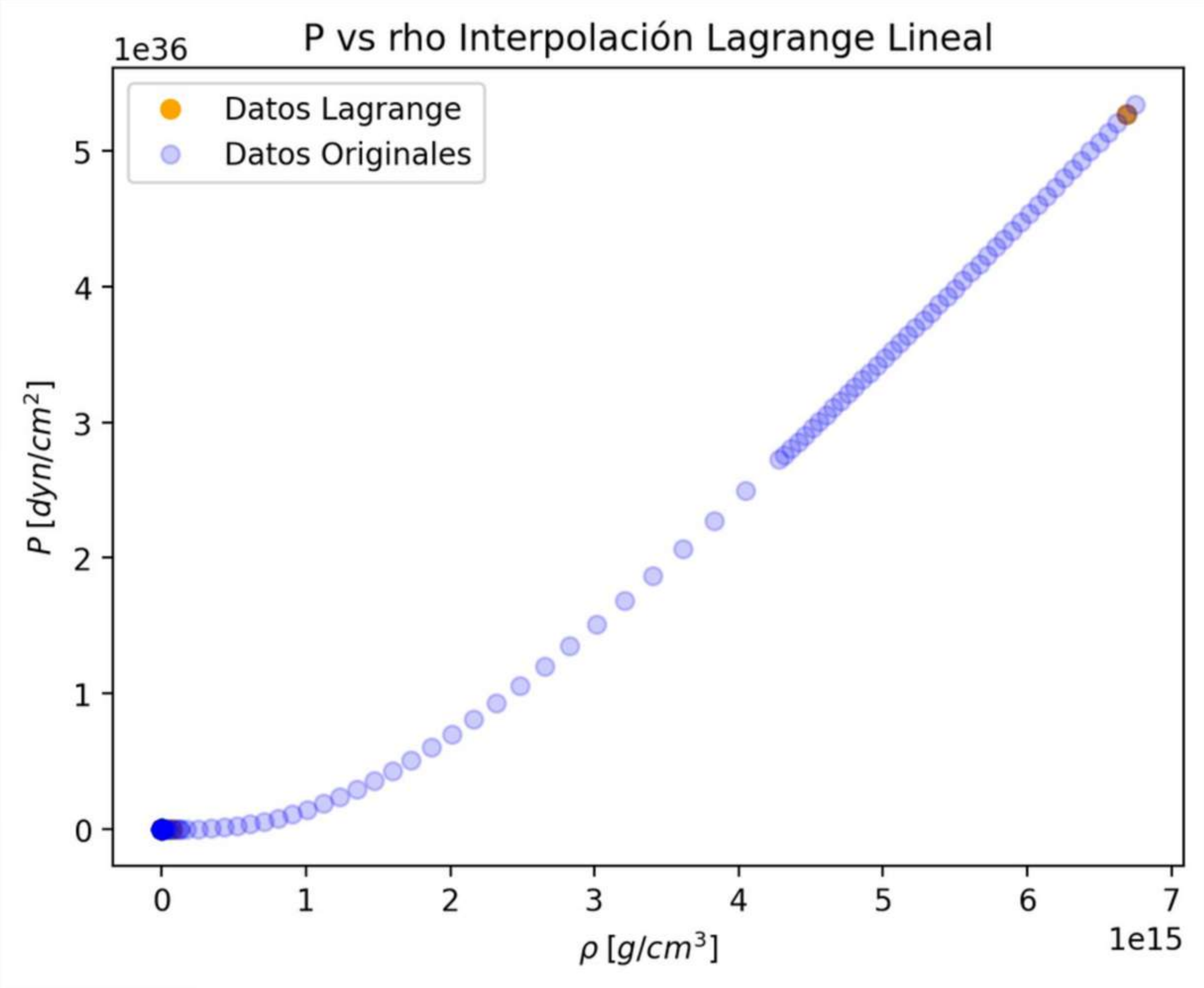




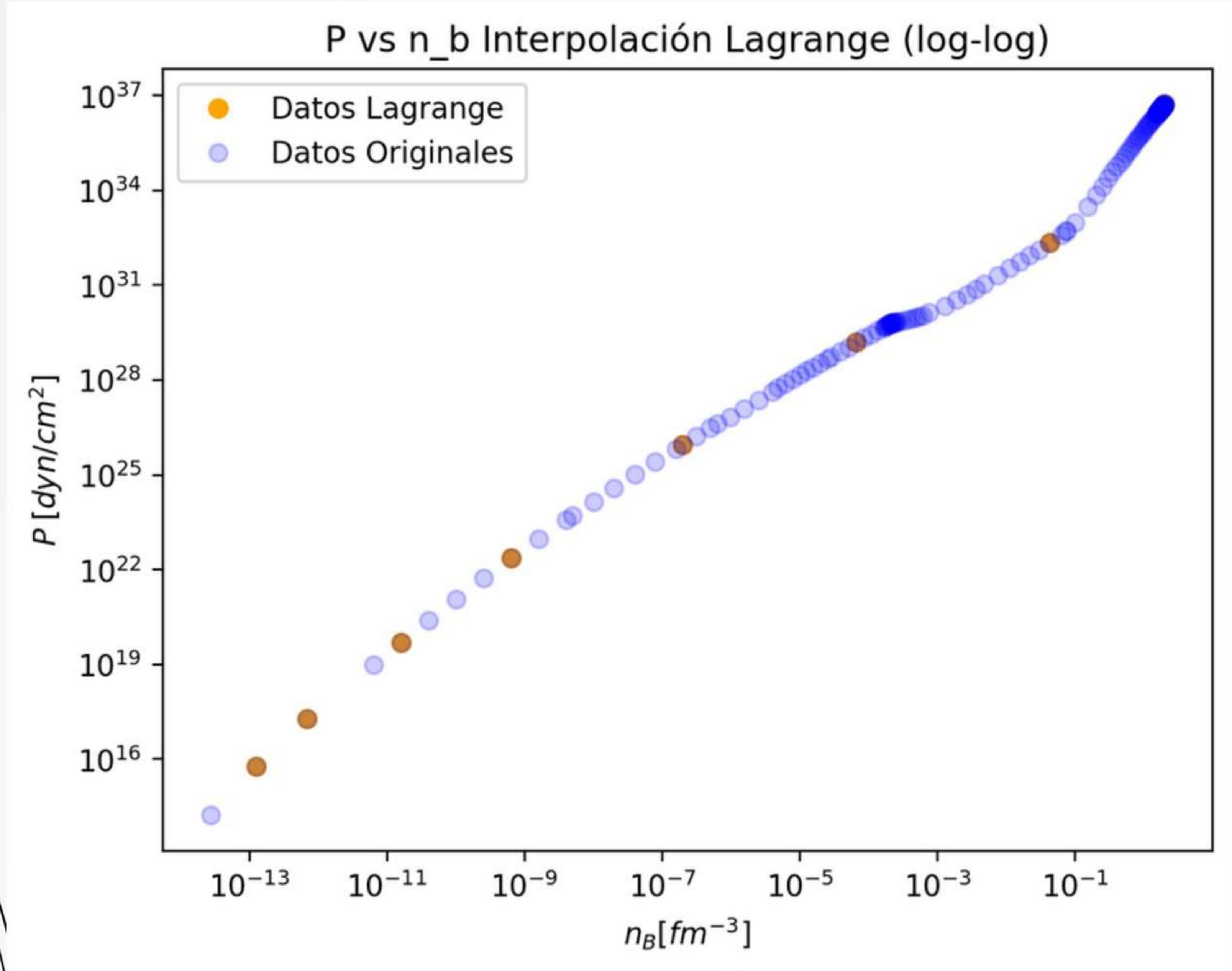

Ajuste Lineal de rho vs n_b
Datos
Ajuste: $y = 2.41e + 15x^{1.02}$
ρ $[g/cm^3]$
n_B $[fm^{-3}]$
10^{16}
10^{14}
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{2}
10^{-13}
10^{-11}
10^{-9}
10^{-7}
10^{-5}
10^{-3}
10^{-1}

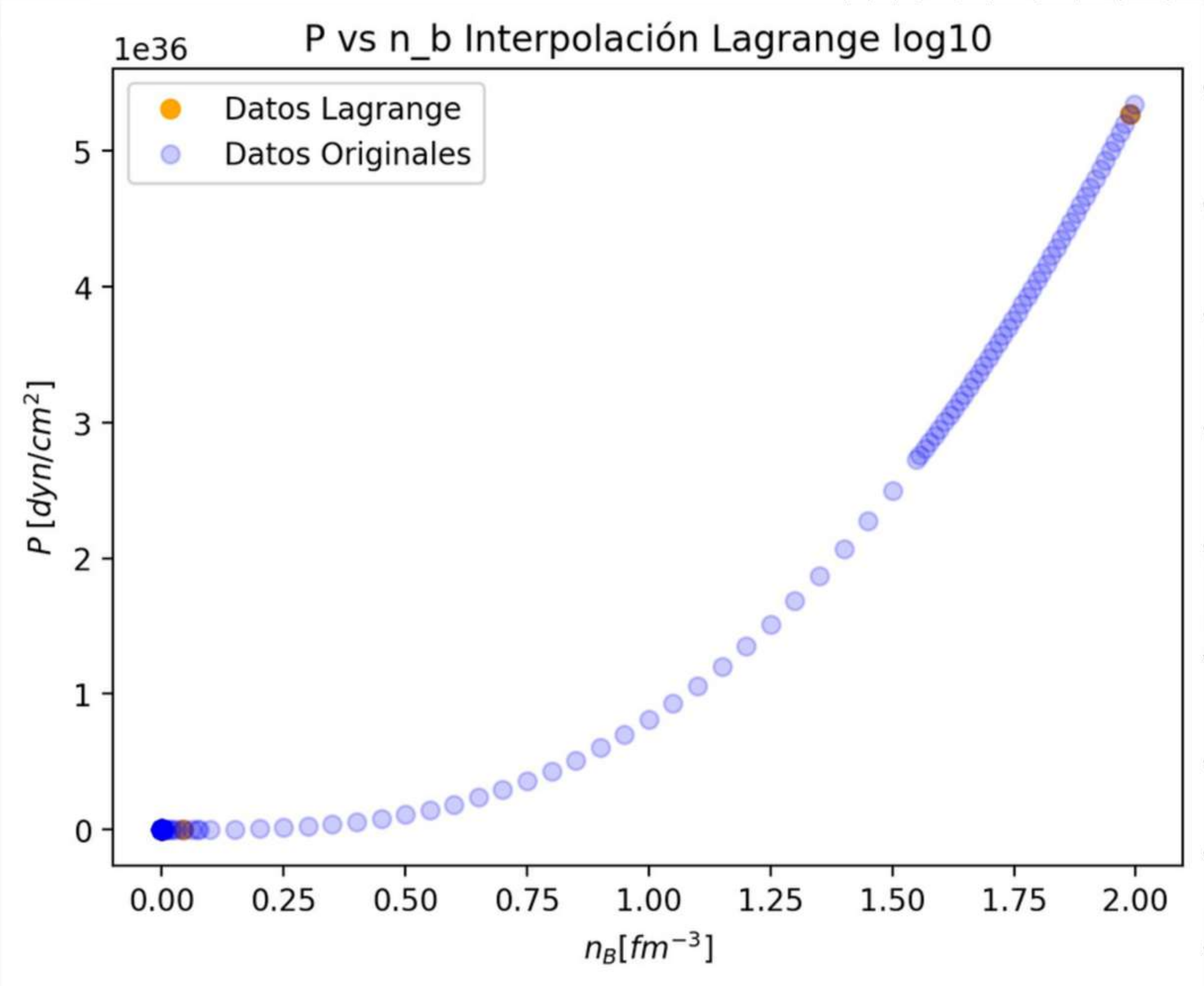
Ecuación de Estado P(rho)





Ecuación de Estado P(n_B)



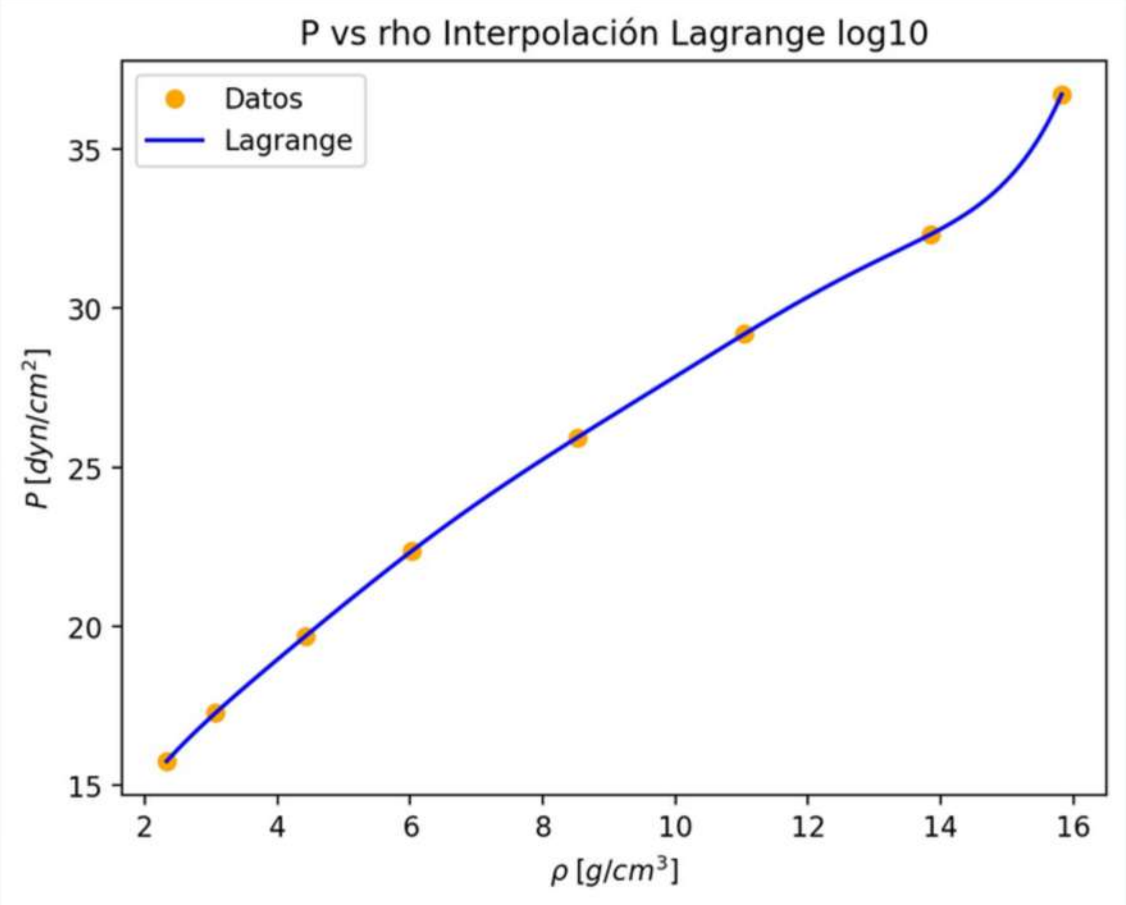


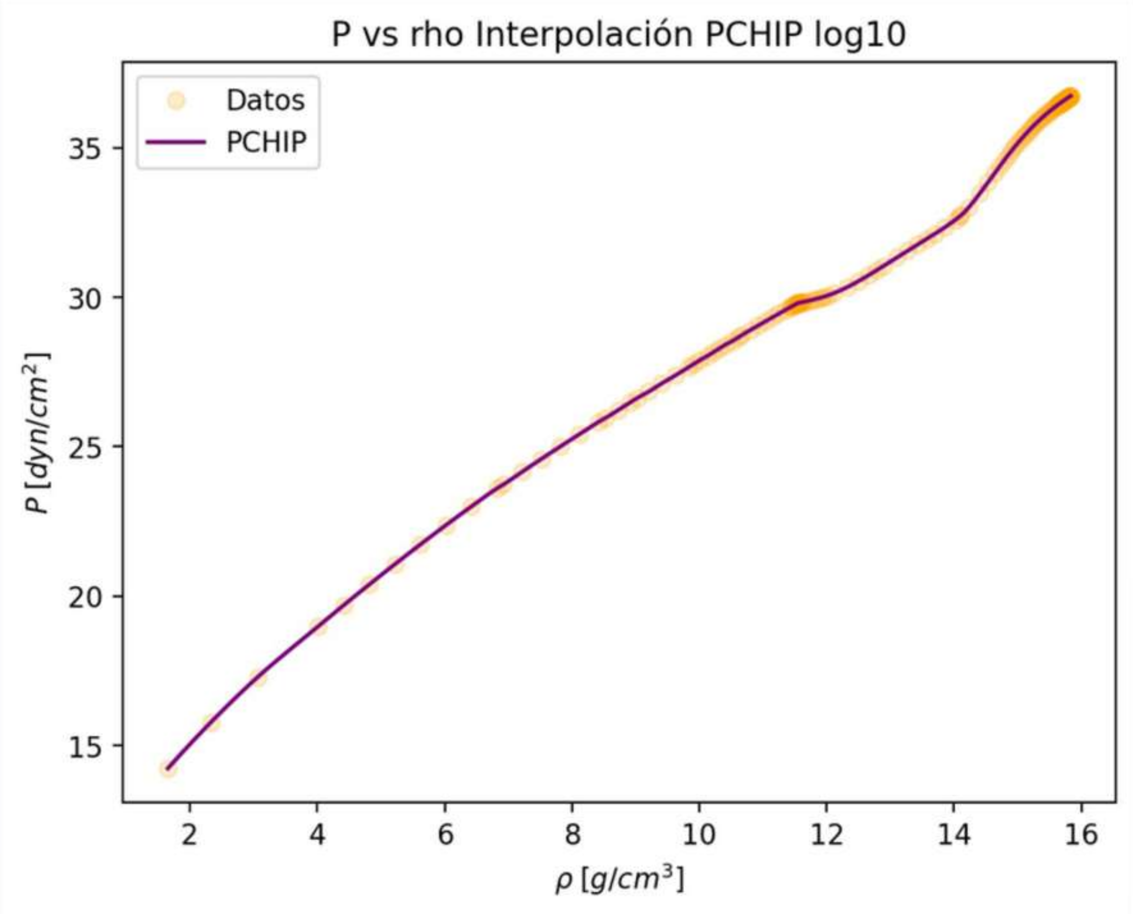
```python
# Interpolación con Spline cúbico natural
cs = CubicSpline(x_sc, y_sc, bc_type="natural")
x_new = np.linspace(min(x_sc), max(x_sc), 500)
y_new = cs(x_new)

plt.plot(x_sc, y_sc, marker='o', color='orange', linestyle='none', label="Datos", alpha=0.2)
plt.plot(x_new, y_new, label="Spline cúbico natural", color="green")
plt.legend()
plt.xlabel(r"$\rho\;[g/cm^3]$")
plt.ylabel(r"$P\;[dyn/cm^2]$")
plt.title("P vs rho Interpolación SC natural log10")
plt.show()
```

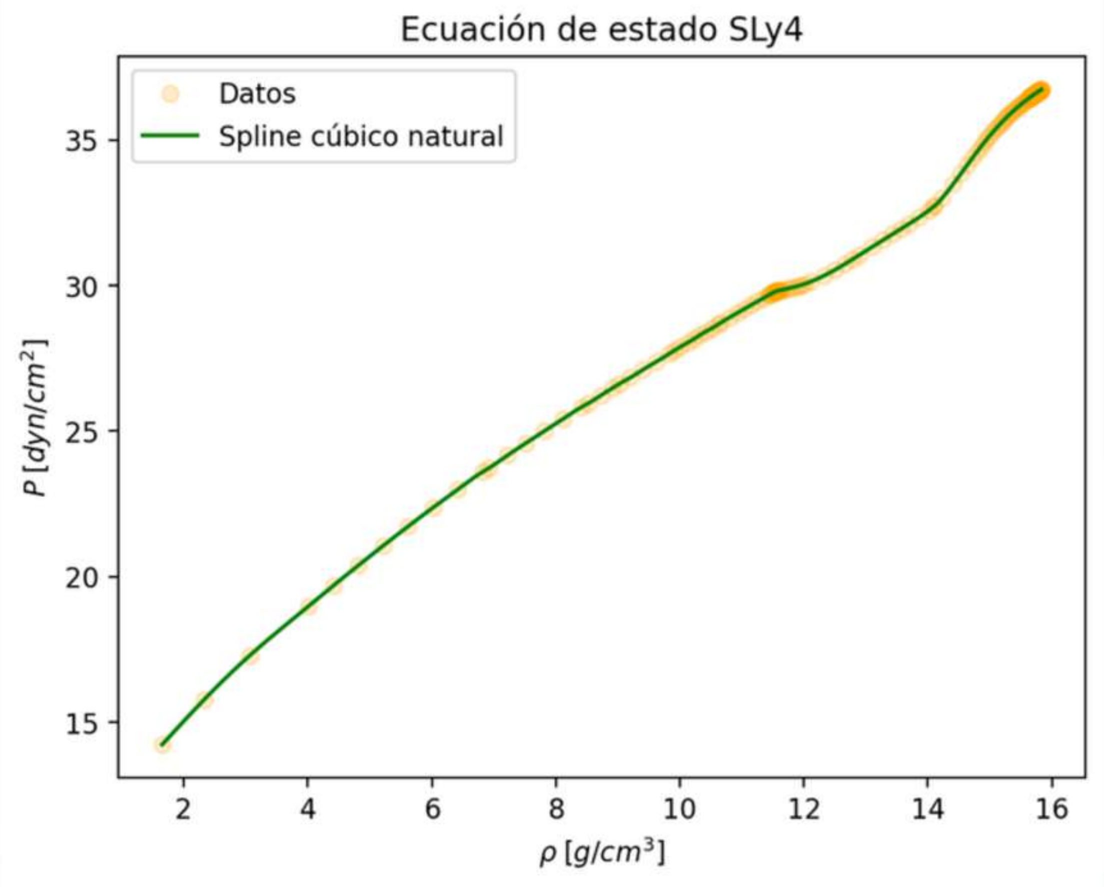
Ecuación de Estado P(rho)

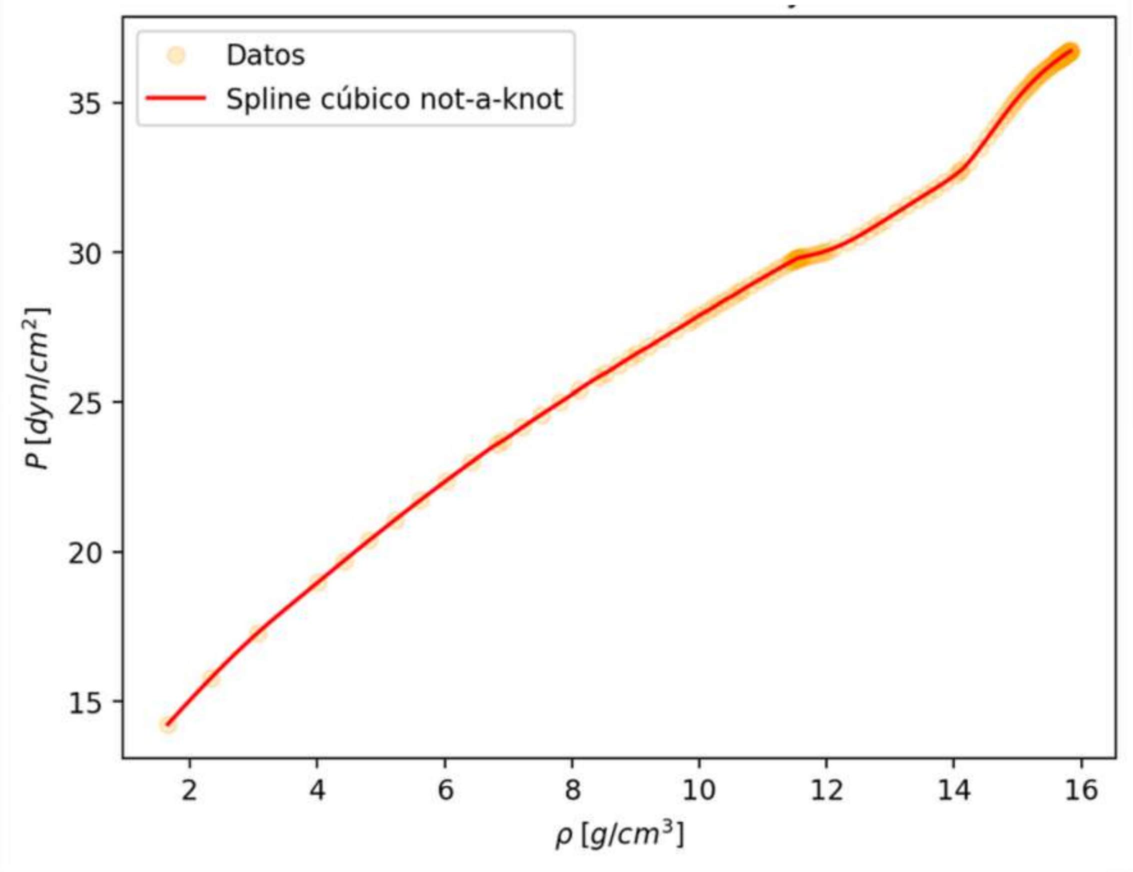
REALIZAR LAS INTERPOLACIONES

- Lagrange
- PCHIP
- Cubic Spline Natural
- Cubic Spline No Natural

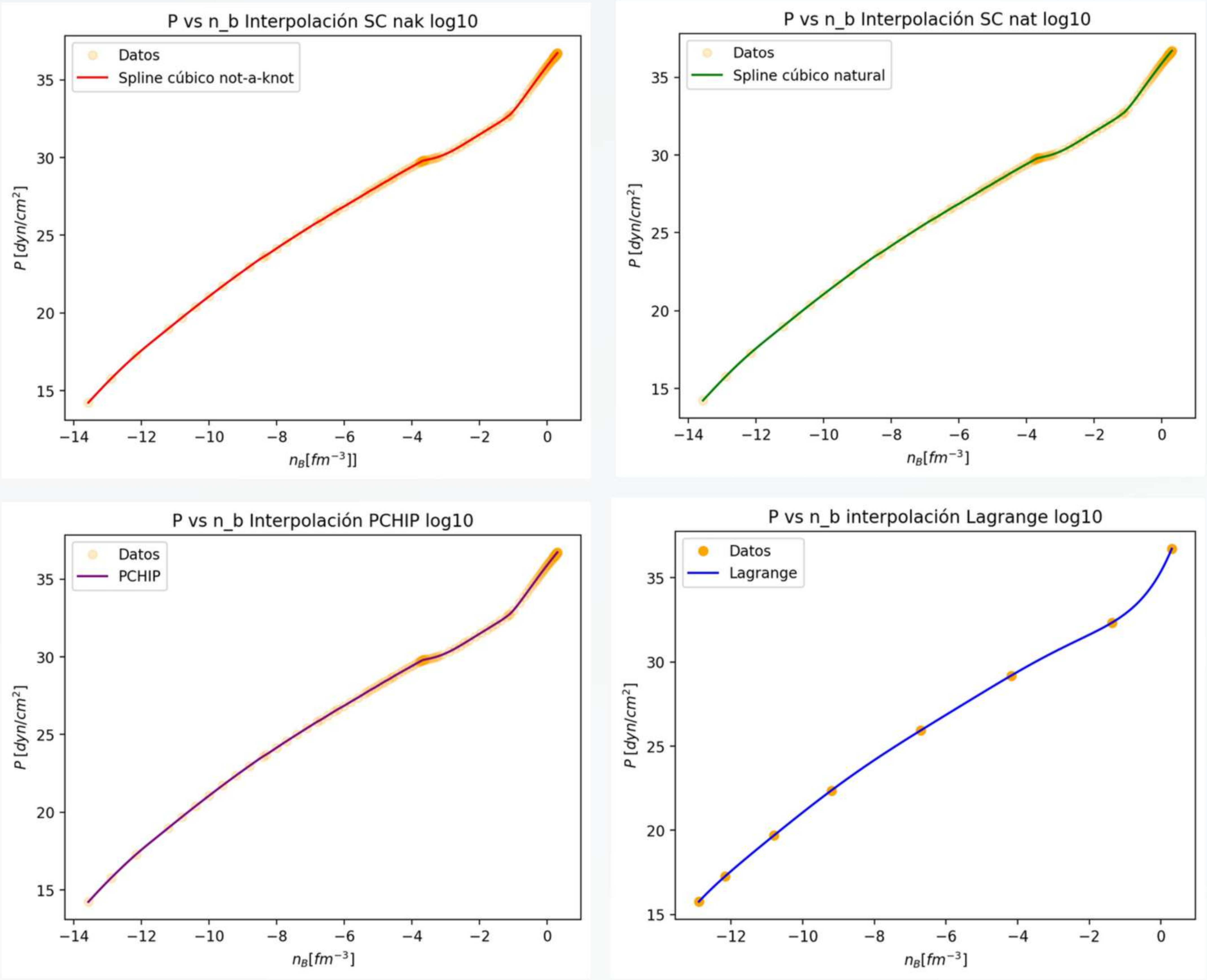








Ecuación de Estado P(n_B)



Crear grupo de entrenamiento y grupo de prueba. Cómo crearon cada grupo.

```python
#Grupos para la varibale independiente n_b

rng = np.random.default_rng(42) #Usamos siempre la misma semilla

n = 2 #Hay que reducir los intervalos pq son pocos datos
rango = np.linspace(x_l.min(), x_l.max(), n+1)
bin_idx = np.digitize(x_l, rango) - 1

train_mask = np.zeros(len(x_l), dtype=bool)
test_mask  = np.zeros(len(x_l), dtype=bool)

porcentaje = 0.7 # 70% en entrenamiento

for b in range(n):
    idx_in_bin = np.where(bin_idx == b)[0]
    if len(idx_in_bin) == 0:
        continue
    n_train = max(1, int(np.round(len(idx_in_bin) * porcentaje)))
    sel = rng.choice(idx_in_bin, size=len(idx_in_bin), replace=False)
    train_idx = sel[:n_train]
    test_idx  = sel[n_train:]
    train_mask[train_idx] = True
    test_mask[test_idx] = True

# conjuntos resultantes:
x_train_l, y_train_l = x_l[train_mask], y_l[train_mask]
x_test_l,  y_test_l  = x_l[test_mask],  y_l[test_mask]

print(f"Para x, los datos de entrenamiento son: \n\n {x_train_l} \n\n y los de testeo son: \n\n {x_test_l}")

print("\n ========================================= \n")
```

```python
#Grupos para la variable aleatoria

rng = np.random.default_rng(42) #Usamos siempre la misma semilla

rango = np.linspace(x_l2.min(), x_l2.max(), n+1)
bin_idx = np.digitize(x_l2, rango) - 1

train_mask = np.zeros(len(x_l2), dtype=bool)
test_mask  = np.zeros(len(x_l2), dtype=bool)

for b in range(n):
    idx_in_bin = np.where(bin_idx == b)[0]
    if len(idx_in_bin) == 0:
        continue
    n_train = max(1, int(np.round(len(idx_in_bin) * porcentaje)))
    sel = rng.choice(idx_in_bin, size=len(idx_in_bin), replace=False)
    train_idx = sel[:n_train]
    test_idx  = sel[n_train:]
    train_mask[train_idx] = True
    test_mask[test_idx] = True

# conjuntos resultantes:
x_train_l2, y_train_l2 = x_l2[train_mask], y_l2[train_mask]
x_test_l2,  y_test_l2  = x_l2[test_mask],  y_l2[test_mask]

print(f"Para x, los datos de entrenamiento son: \n\n {x_train_l2} \n\n y los de testeo son: \n\n {x_test_l2}")
```

```
Para x, los datos de entrenamiento son:

 [ 2.32633586  3.06069784  6.0187005   8.51917146 13.85745312]

 y los de testeo son:

 [ 4.41863269 11.04139269]

 =========================================

Para x, los datos de entrenamiento son:

 [-12.89619628 -12.15926677  -9.20107326  -6.70092874  -1.36977559]

 y los de testeo son:

 [-10.80106813  -4.1809828 ]
```

Grupos train y test solo para P(rho) y P(n_b) solo para lagrange

Grupos train y test para P(rho)

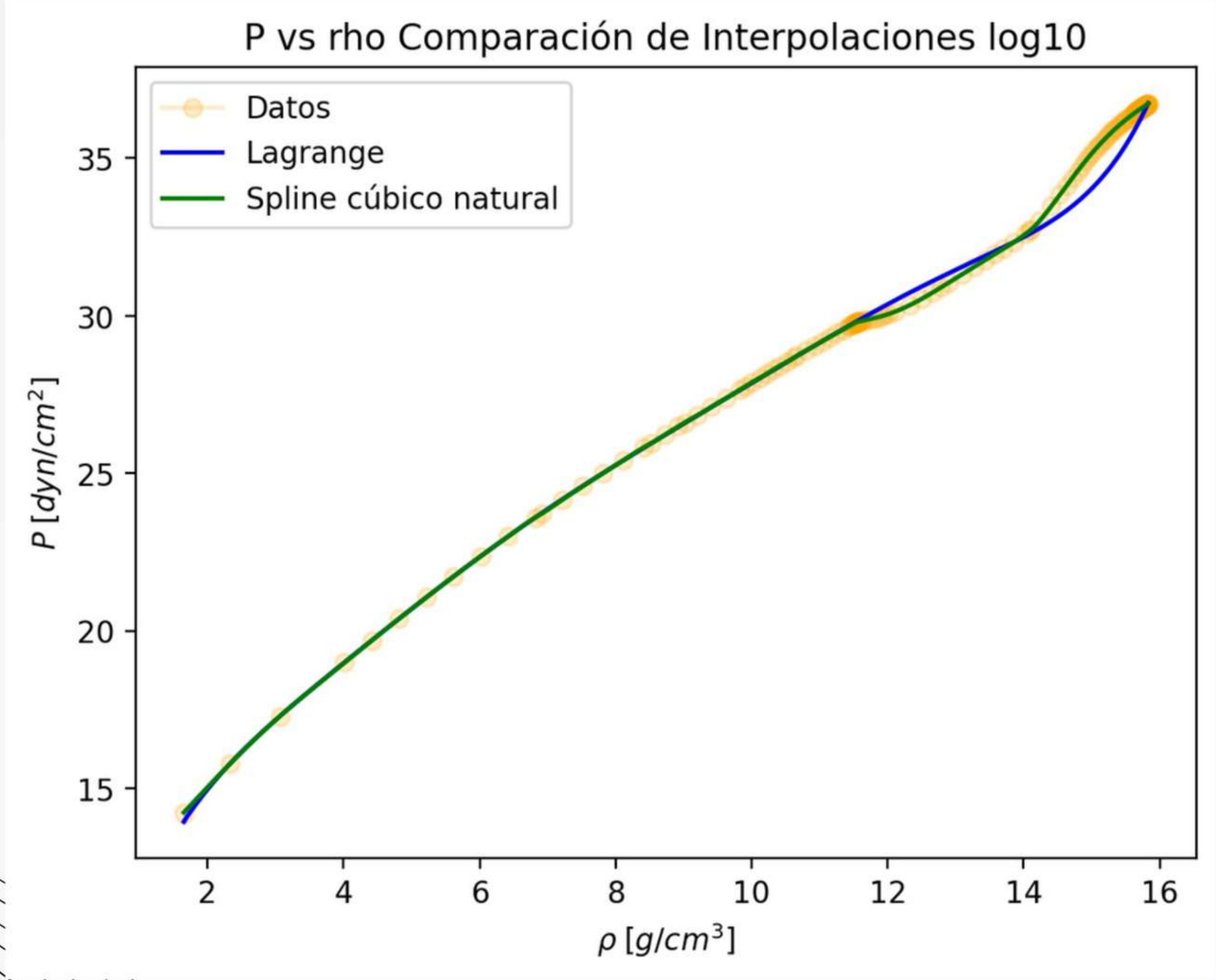
```python
rng = np.random.default_rng(42) #Usamos siempre la misma semilla

n = 8
rango = np.linspace(x_sc.min(), x_sc.max(), n+1)
bin_idx = np.digitize(x_sc, rango) - 1

train_mask = np.zeros(len(x_sc), dtype=bool)
test_mask  = np.zeros(len(x_sc), dtype=bool)

porcentaje = 0.8 # 80% en entrenamiento

for b in range(n):
    idx_in_bin = np.where(bin_idx == b)[0]
    if len(idx_in_bin) == 0:
        continue
    n_train = max(1, int(np.round(len(idx_in_bin) * porcentaje)))
    sel = rng.choice(idx_in_bin, size=len(idx_in_bin), replace=False)
    train_idx = sel[:n_train]
    test_idx  = sel[n_train:]
    train_mask[train_idx] = True
    test_mask[test_idx] = True

# conjuntos resultantes:
x_train_sc, y_train_sc = x_sc[train_mask], y_sc[train_mask]
x_test_sc,  y_test_sc  = x_sc[test_mask],  y_sc[test_mask]
```

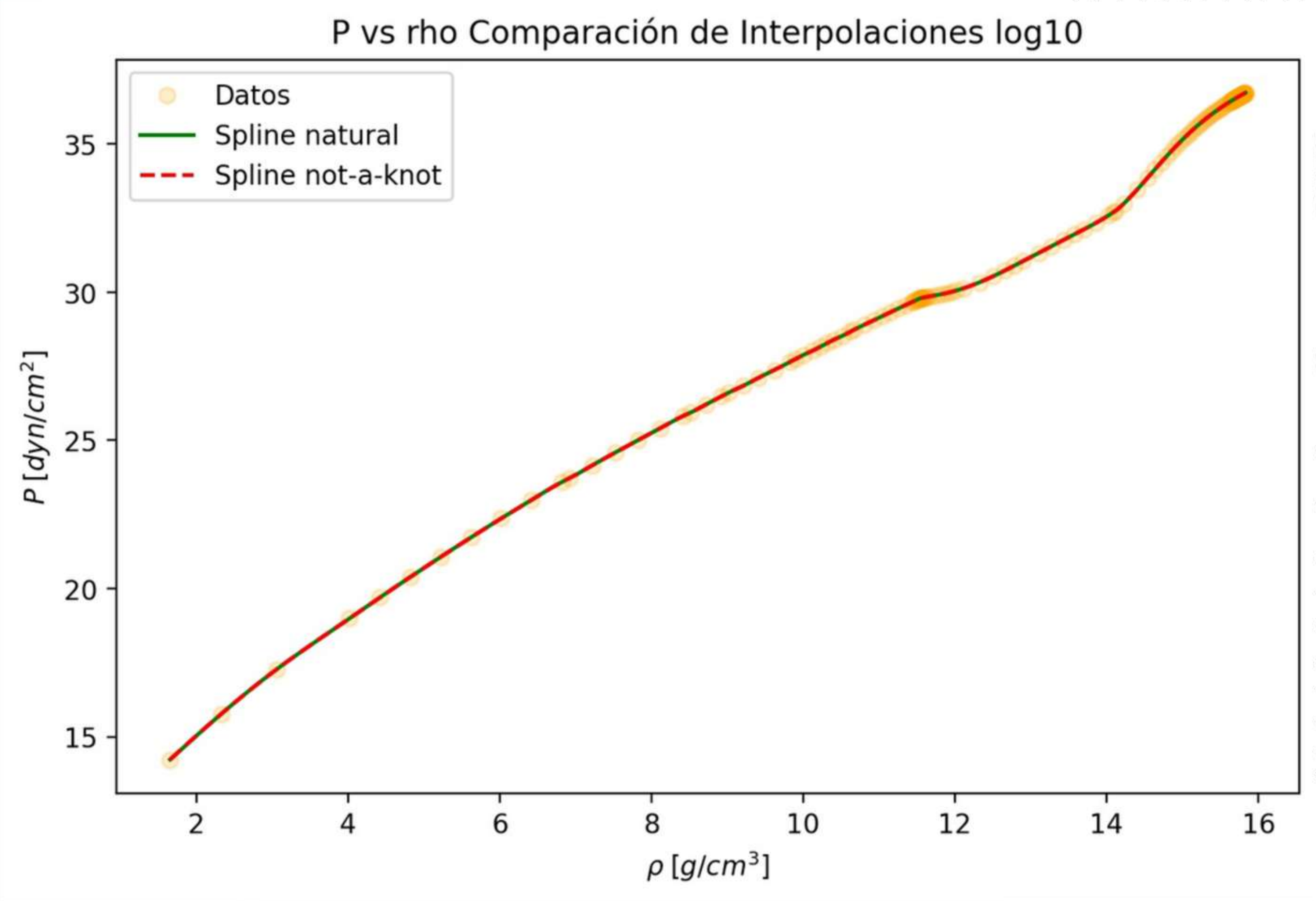
Grupos train y test para P(n_b)

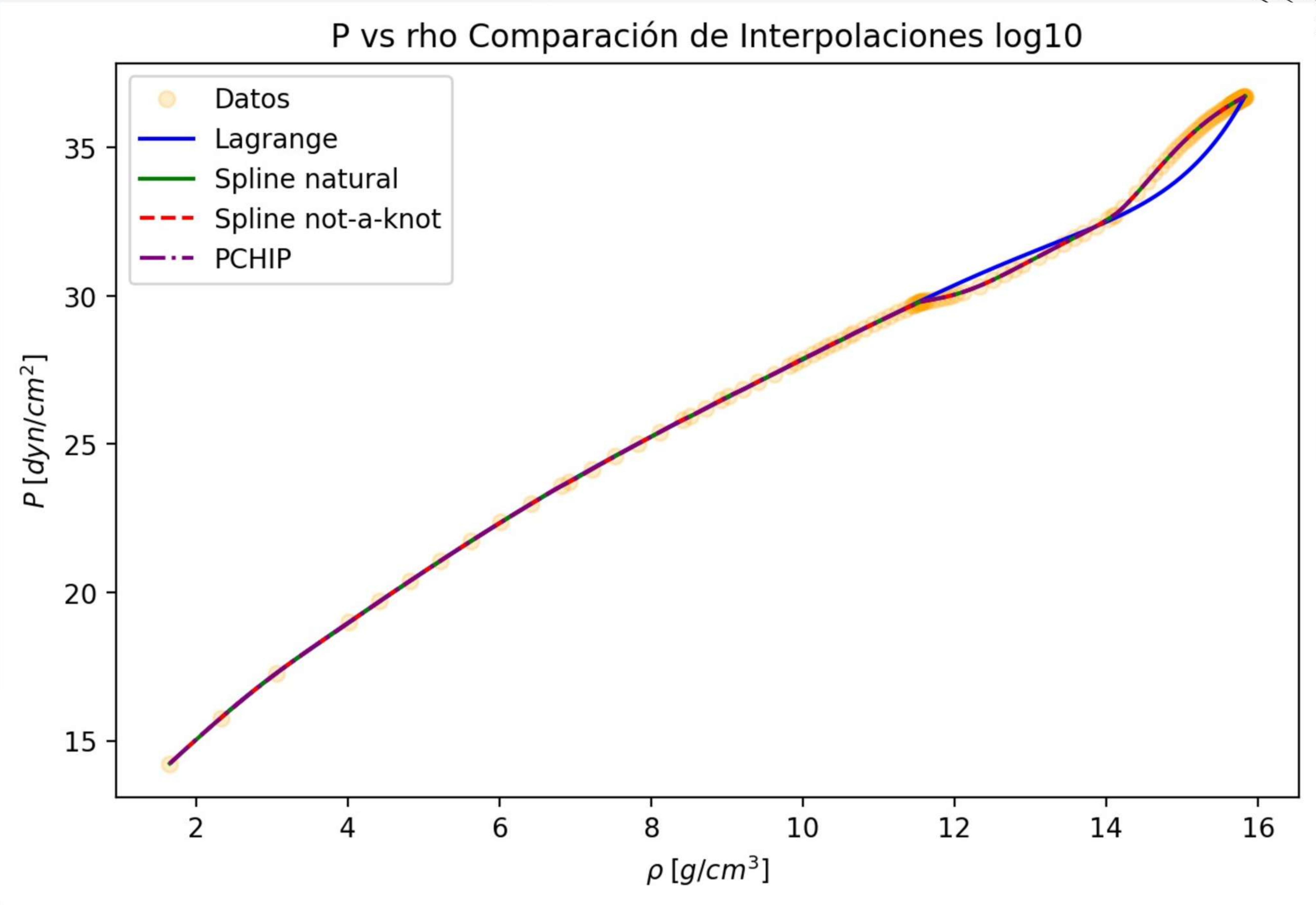
```python
rng = np.random.default_rng(42) #Usamos siempre la misma semilla

n = 8
rango = np.linspace(x_sc2.min(), x_sc2.max(), n+1)
bin_idx = np.digitize(x_sc2, rango) - 1

train_mask = np.zeros(len(x_sc2), dtype=bool)
test_mask  = np.zeros(len(x_sc2), dtype=bool)

porcentaje = 0.8 # 80% en entrenamiento

for b in range(n):
    idx_in_bin = np.where(bin_idx == b)[0]
    if len(idx_in_bin) == 0:
        continue
    n_train = max(1, int(np.round(len(idx_in_bin) * porcentaje)))
    sel = rng.choice(idx_in_bin, size=len(idx_in_bin), replace=False)
    train_idx = sel[:n_train]
    test_idx  = sel[n_train:]
    train_mask[train_idx] = True
    test_mask[test_idx] = True

# conjuntos resultantes:
x_train_sc2, y_train_sc2 = x_sc2[train_mask], y_sc2[train_mask]
x_test_sc2,  y_test_sc2  = x_sc2[test_mask],  y_sc2[test_mask]
```

Comparar graficamente la interpolación

Ecuación de Estado P(rho)



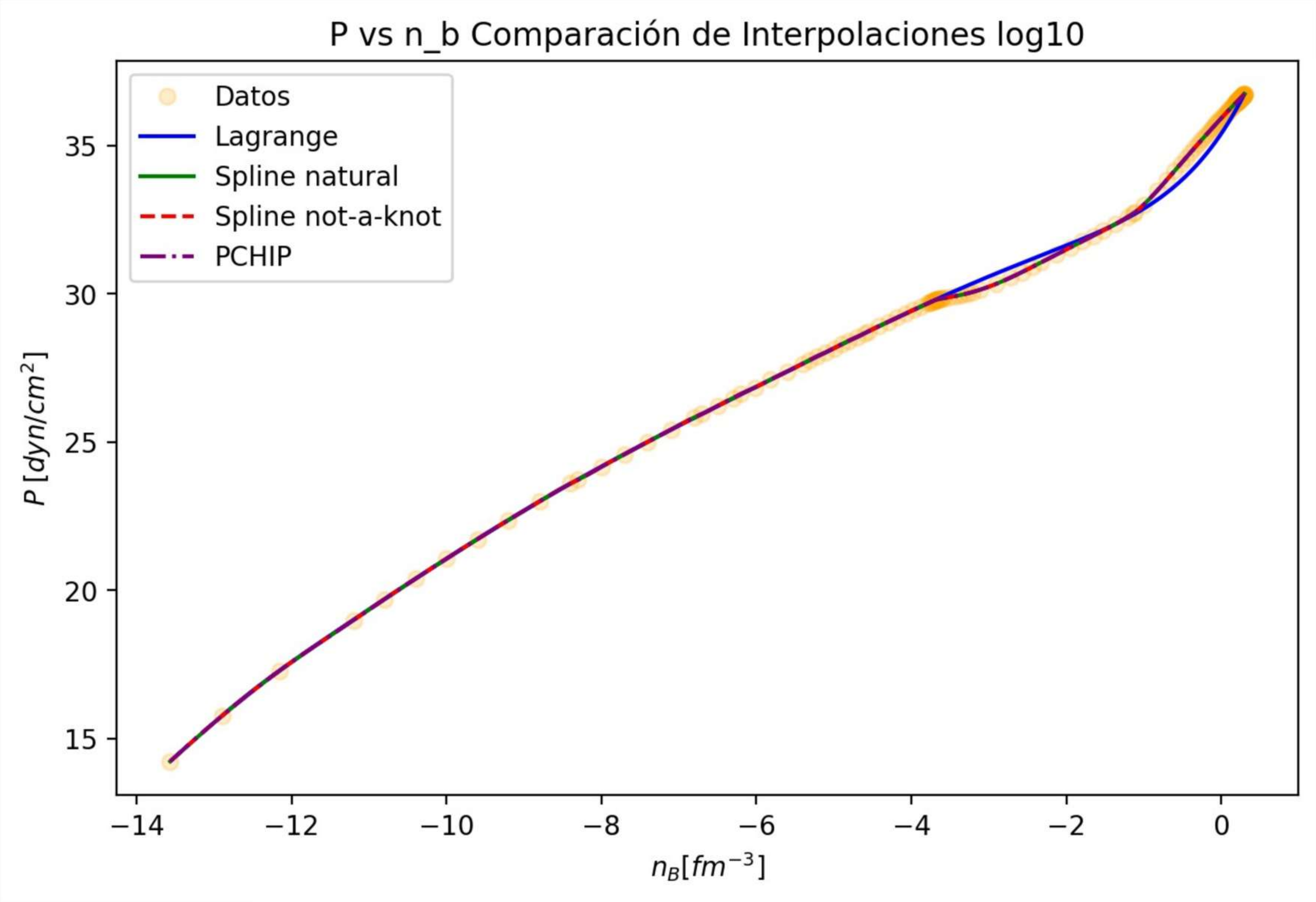




P vs rho Comparación de Interpolaciones log10
Datos
Lagrange
Spline natural
Spline not-a-knot
PCHIP
P [dyn/cm^2]
ρ [g/cm^3]
15
20
25
30
35
2
4
6
8
10
12
14
16

Comparar graficamente la interpolación

Ecuación de Estado P(n_B)



Calculo de Error Cuadrático y Absoluto

- Error Cuadrático P(rho)

  MSE Lagrange: 2.4343 e-21
  MSE Spline natural: 3.3215 e-31
  MSE Spline not-a-knot: 0
  MSE PCHIP: 0

- Error Absoluto P(rho)

  MSE Lagrange: 2.6432 e-11
  MSE Spline natural: 4.674 e-31
  MSE Spline not-a-knot: 0
  MSE PCHIP: 0

- Error Cuadrático P(n_b)

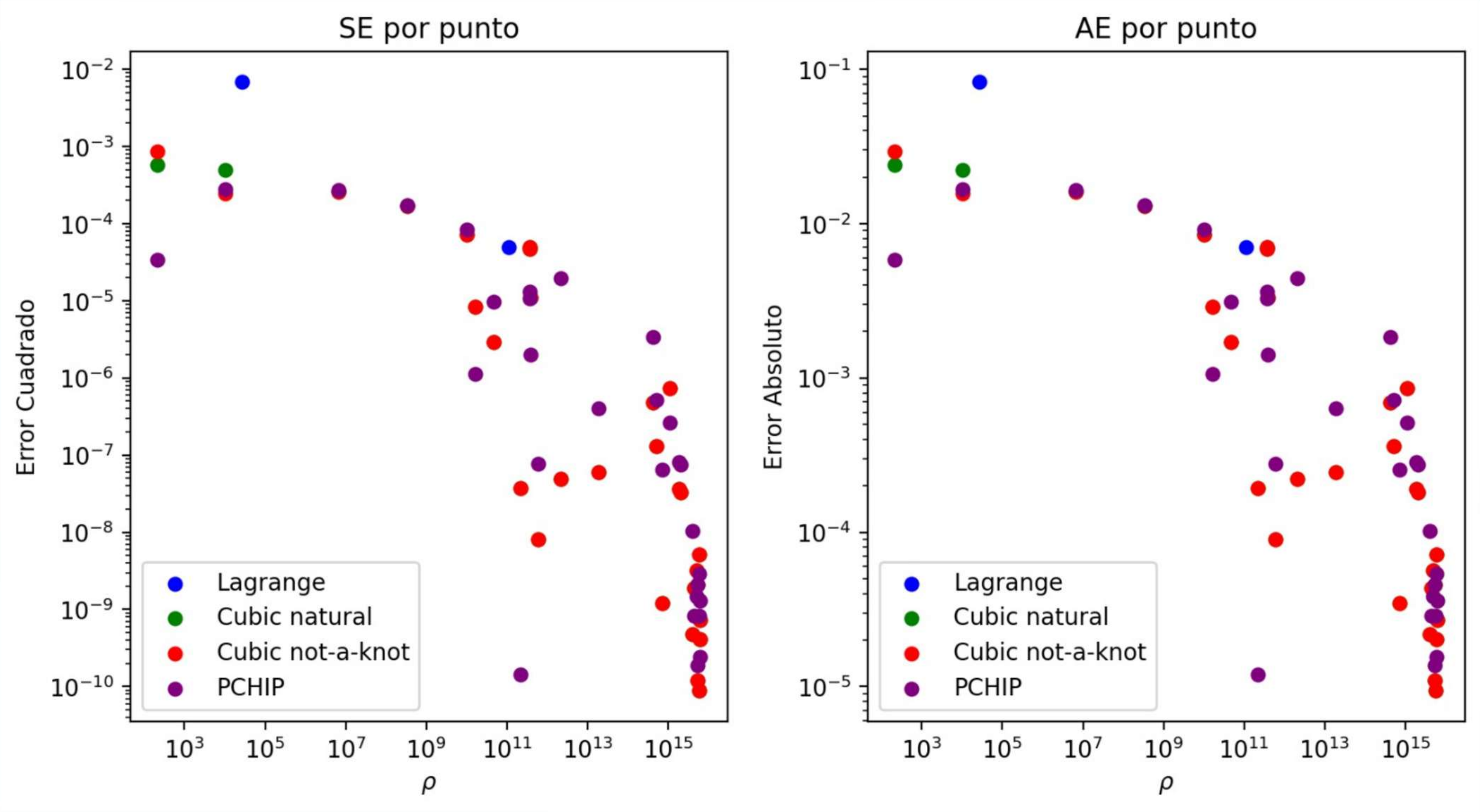
  MSE Lagrange: 3.028 e-21
  MSE Spline natural: 0
  MSE Spline not-a-knot: 0
  MSE PCHIP: 0

- Error Absoluto P(rho)

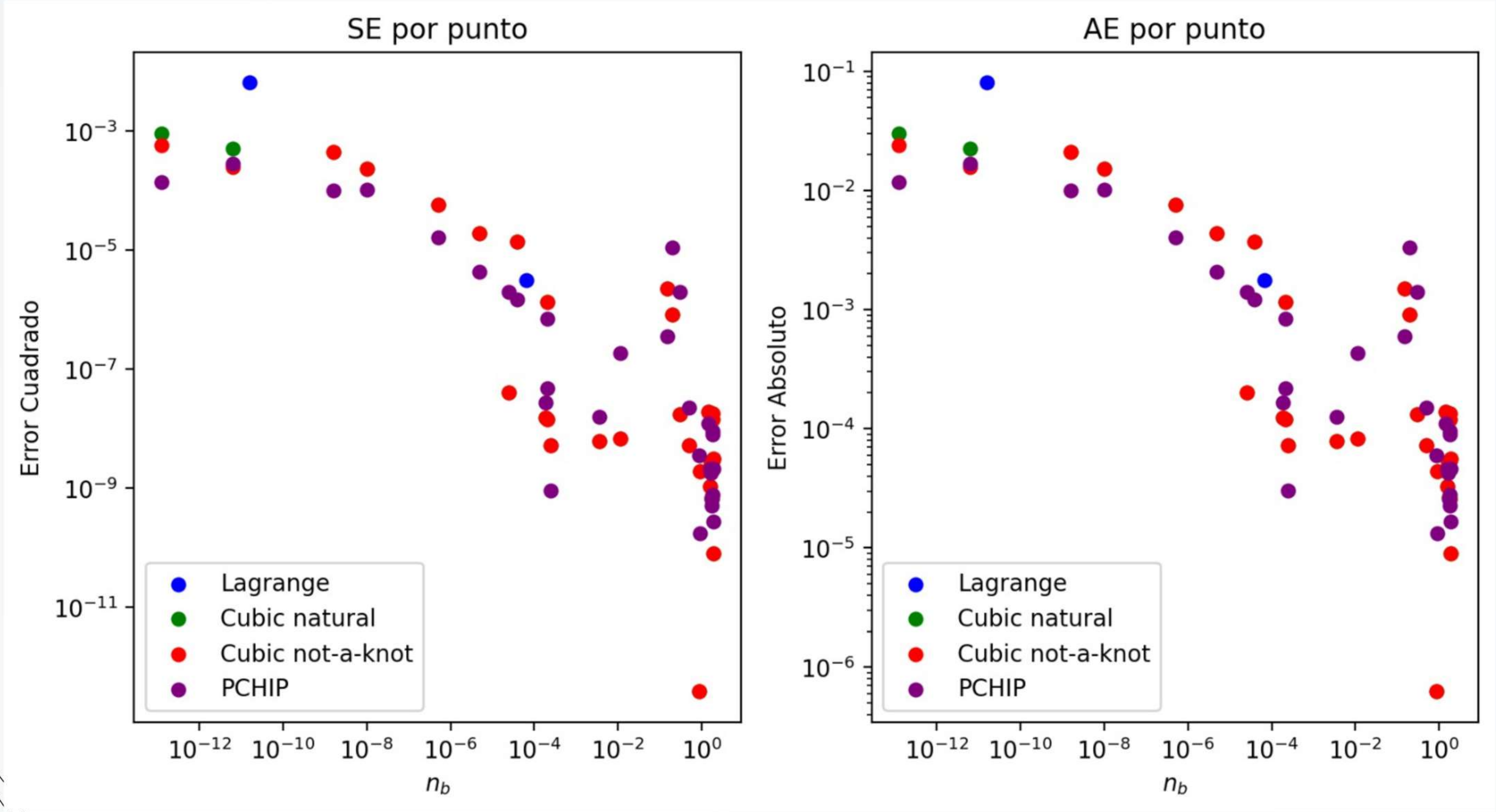
  MSE Lagrange: 3.4429 e-11
  MSE Spline natural: 0
  MSE Spline not-a-knot: 0
  MSE PCHIP: 0

GRAFICAR LOS ERRORES DE CADA METODO

Errores P(rho)



Errores P(n_b)



SE por punto
Error Cuadrado
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{-11}
10^{-12} 10^{-10} 10^{-8} 10^{-6} 10^{-4} 10^{-2} 10^{0}
n_b
Lagrange
Cubic natural
Cubic not-a-knot
PCHIP
AE por punto
Error Absoluto
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-12} 10^{-10} 10^{-8} 10^{-6} 10^{-4} 10^{-2} 10^{0}
n_b
Lagrange
Cubic natural
Cubic not-a-knot
PCHIP