clase_02

April 10, 2025

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

Si $det(A) <> 0$ entonces el sistema tiene una única solucion y se puede resolver el sistema $Ax = b$ tomando la inversa

$$A^{-1}Ax = A^{-1}b = Ix = A^{-1}b$$

por lo tanto

$$x = A^{-1}b$$

Un ejemplo es la interpolación polinómica donde se tienen diferentes pares ordenados (x, y) y se puede ajustar mediante un polinomia de grado n-1

$$p(x) = c_0 + c_1x + ... + c_nx^n$$

en forma matricial quedaría

$$A = \begin{bmatrix} 1 & X_1 & X_1^{n-1} \\ 1 & X_2 & X_2^{n-1} \\ 1 & X_3 & X_3^{n-1} \\ 1 & X_n & X_n^{n-1} \end{bmatrix}$$

Siendo $A * c = Y$ y los elementos que hay que determinar es el vector c

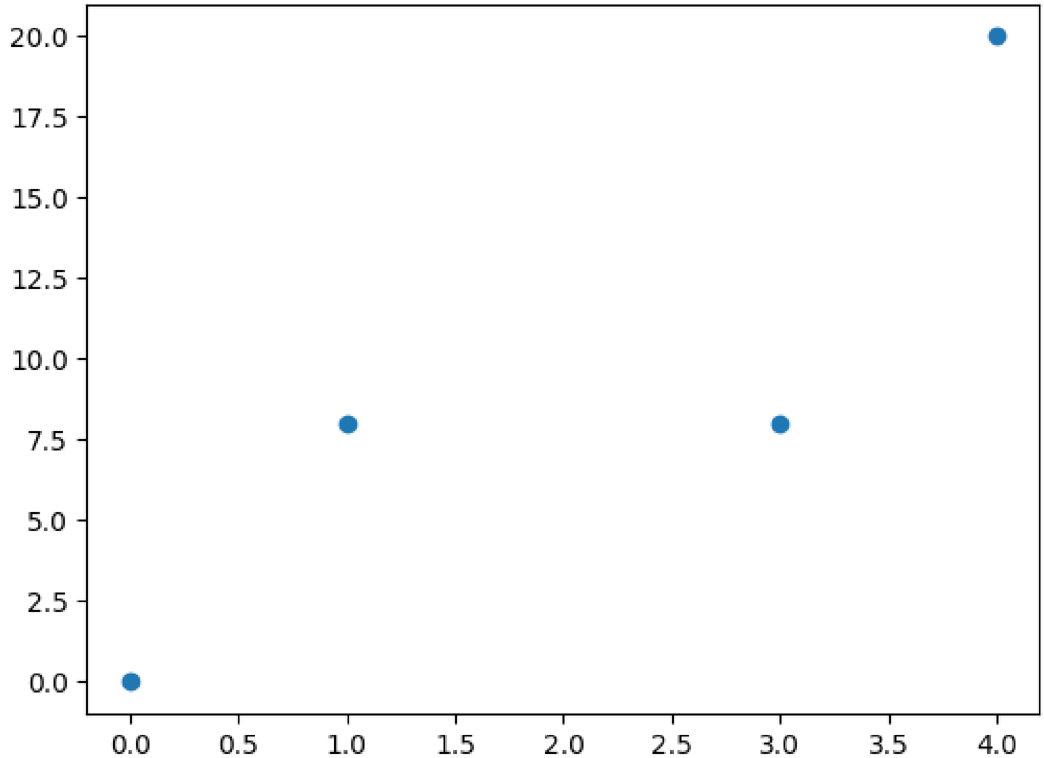
Dados los tiempos t=(0,1,3,4) y las mediciones b=(0.8.8.20)

Buscamos un polinomio de grado 3 (n-1 mediciones) porque esa es una matriz cuadrada y lo puedo resolver con la inversa

[9]:
```python
tiempos=[0,1,3,4]
mediciones=[0,8,8,20]

plt.plot(tiempos, mediciones, 'o', label='Datos')
```

[9]: [<matplotlib.lines.Line2D at 0x10be09110>]



[ ]:
```
#La matriz Ax=b ahora es Tx=m siendo T los tiempos y m las mediciones
# como es un una matriz de 4 necesito un polinomio de grado 3 para poder tener␣
↪una matriz cuadrada
#entonces p(x) =c0+c1*t1 + c2*t2^2+ c3*t3+^3c4*t4^4
# la matriz T quedaria

T=[[1,0,0,0],
   [1,1,1,1],
   [1,3,9,27],
    [1,4,16,64]]
# Ahora hago T^{-1}*mediciones y obtengo x
T_inv=np.linalg.pinv(T)
print(T)
print(T_inv)
x=T_inv@mediciones
print("el vector de soluciones para un polinomio de grado 3 es")
print(x)
```

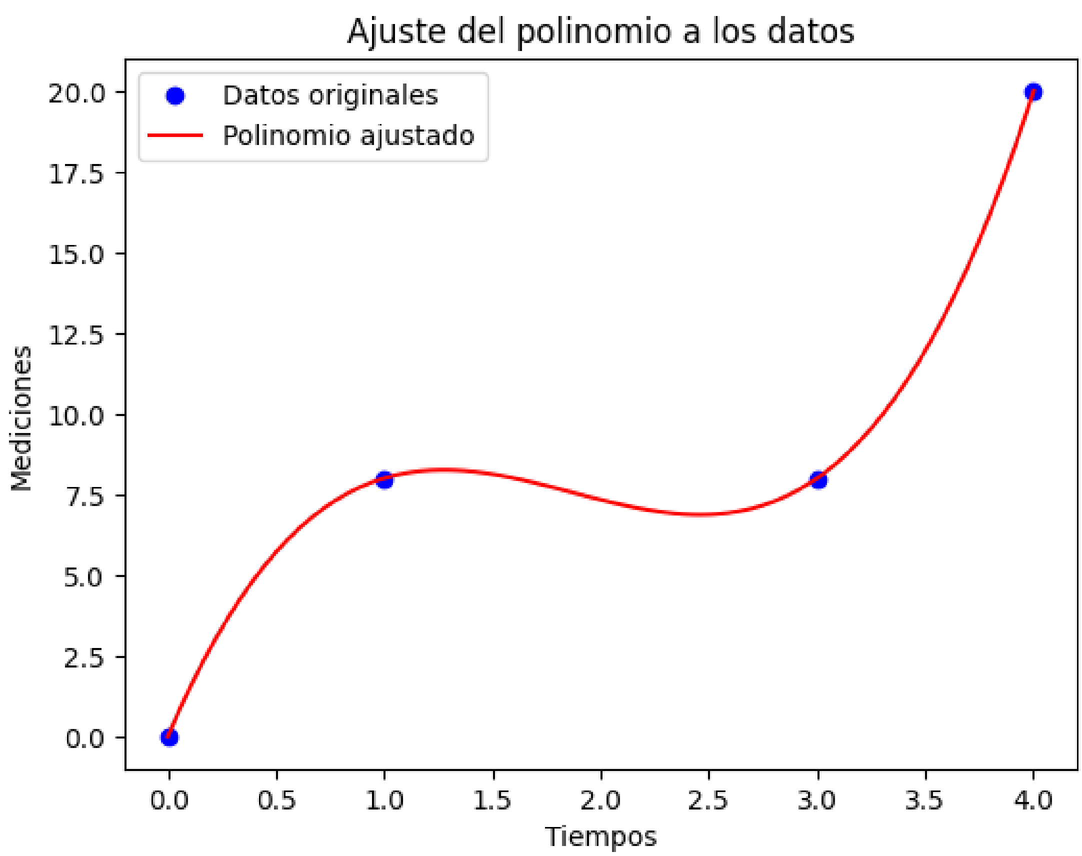
```
[[1, 0, 0, 0], [1, 1, 1, 1], [1, 3, 9, 27], [1, 4, 16, 64]]
[[ 1.00000000e+00  2.30699207e-14 -7.41825608e-15  3.09996116e-15]
 [-1.58333333e+00  2.00000000e+00 -6.66666667e-01  2.50000000e-01]
 [ 6.66666667e-01 -1.16666667e+00  8.33333333e-01 -3.33333333e-01]
```

```
 [-8.33333333e-02  1.66666667e-01 -1.66666667e-01  8.33333333e-02]]
el vector de soluciones para un polinomio de grado 3 es
[ 1.87212540e-13  1.56666667e+01 -9.33333333e+00  1.66666667e+00]
```

[ ]:
```python
## Graficamos el polinomio
```

[10]:
```python
# Generar valores para el polinomio
t_vals = np.linspace(min(tiempos), max(tiempos), 100)  # Valores de t para
 ↪graficar
polinomio_vals = x[0] + x[1]*t_vals + x[2]*t_vals**2 + x[3]*t_vals**3  #
 ↪Evaluar el polinomio

# Graficar los datos originales y el polinomio ajustado
plt.plot(tiempos, mediciones, 'bo', label='Datos originales')  # Datos
 ↪originales
plt.plot(t_vals, polinomio_vals, 'r-', label='Polinomio ajustado')  # Polinomio
 ↪ajustado
plt.xlabel('Tiempos')
plt.ylabel('Mediciones')
plt.legend()
plt.title('Ajuste del polinomio a los datos')
plt.show()
```



[11]:
```
#Ahora podemos calcular el error, sin embargo vemos que va a ser muy pequeño o
↪cero
def polinomio_grado_3(t_vals):
    return  x[0] + x[1]*t_vals + x[2]*t_vals**2 + x[3]*t_vals**3

for i in range(len(tiempos)):
    error=0
    error=error+(mediciones[i]-polinomio_grado_3(tiempos[i]))**2

print(error)
```

```
2.0194839173657902e-28
```

0.1 a) encontrar la recta y=c+dt que mejor se ajusta a los datos. Escriba las ecuaciones normales y resuelvalas.

[ ]:
```
#Ahora tengo el problema de que tengo menos ecuaciones que incógnitas, porque
↪tengo un polinomio de grado 1 o sea, 2 columnas
#Entonces uso las ecuaciones normales
```

[ ]:
```

```

[ ]:
```

```

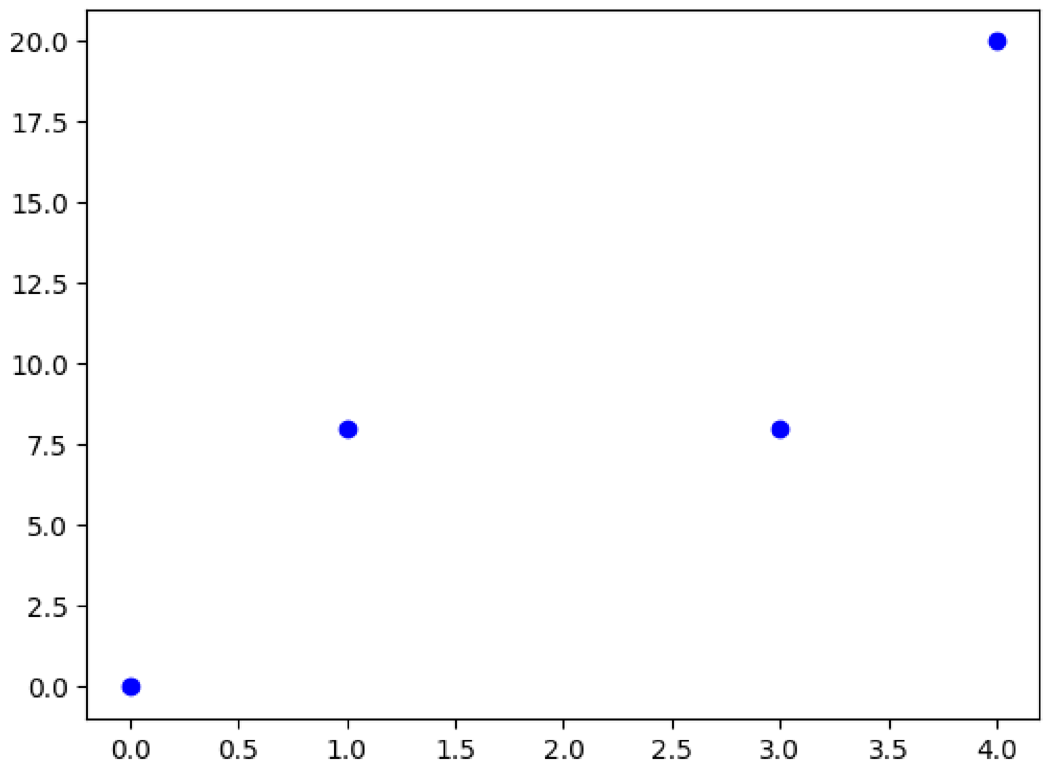
[2]:
```
#1. Dados los tiempos t= (0,1,3,4) y las mediciones b= (0,8,8,20).

# a)  Encontrar la recta y= c+ dt que mejor se ajusta a los datos. Para esto
↪escriba las ecuaciones
#normales y resuélvalas. Calcular el error.

tiempos= np.array([0, 1, 3, 4])
mediciones= np.array([0, 8, 8, 20])

## graficamos
plt.plot(tiempos, mediciones, 'bo', label='Datos')
```

[2]:
```
[<matplotlib.lines.Line2D at 0x10bb30250>]
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[57]:
```
# Se define la matriz de diseño
A= np.vstack((tiempos,np.ones(len(tiempos)))).T

print(A)
```

```
[[0. 1.]
 [1. 1.]
 [3. 1.]
 [4. 1.]]
```

[58]:
```
A.T
```

[58]:
```
array([[0., 1., 3., 4.],
       [1., 1., 1., 1.]])
```

[59]:
```
A.T @ A
```

[59]:
```
array([[26.,  8.],
       [ 8.,  4.]])
```

[60]:
```
solution=np.linalg.solve(A.T @ A, A.T @ mediciones)

print(solution)
```

```
[4. 1.]
```

[62]:
```
#Calculo el error de aproximar por medio de la recta con los parametros de␣
↪solucion
errorTotal= 0
for i in range(len(tiempos)):
    mediciones_ajustadas= solution[0]*tiempos[i] + solution[1]
    error= (mediciones[i]-mediciones_ajustadas)
    print(f"Error en el tiempo {tiempos[i]}: {error}")
    errorTotal+= error**2
print(errorTotal)
```

```
Error en el tiempo 0: -0.9999999999999964
Error en el tiempo 1: 3.0000000000000027
Error en el tiempo 3: -5.0
Error en el tiempo 4: 3.0
44.00000000000001
```

Repetir con una línea recta horizontal y= c.

[79]:
```
# tiempos y mediciones son iguales, lo que cambia es la matriz

#A= np.vstack((tiempos,np.ones(len(tiempos)))).T

A= np.vstack((np.ones(len(tiempos))))
print(A)
```

```
[[1.]
 [1.]
 [1.]
 [1.]]
```

[80]:
```
solution=np.linalg.solve(A.T @ A, A.T @ mediciones)
print(solution)
```

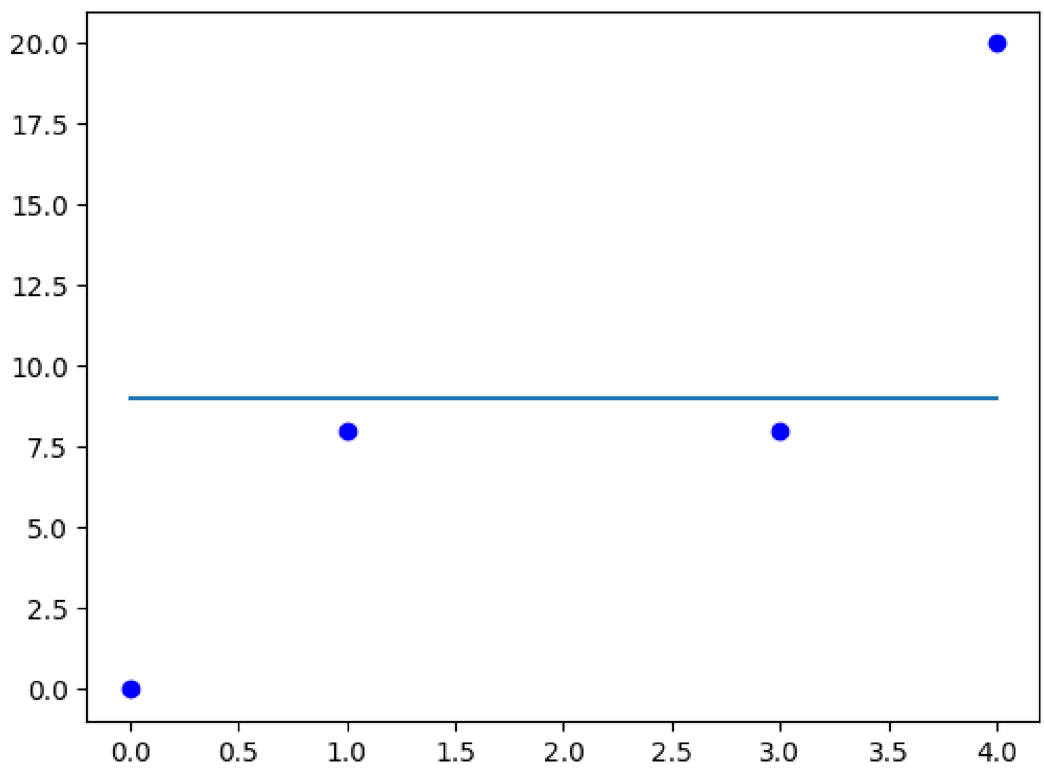
```
[9.]
```

[81]:
```
## calculo el error de aproximar por medio de la recta con los parametros de␣
↪solucion
errorTotal= 0
for i in range(len(tiempos)):
    mediciones_ajustadas=solution[0]
    error=mediciones[i]-mediciones_ajustadas
    print(f"Error en el tiempo {tiempos[i]}: {error}")
    errorTotal+= error**2
print(errorTotal)
```

```
Error en el tiempo 0: -9.0
Error en el tiempo 1: -1.0
Error en el tiempo 3: -1.0
Error en el tiempo 4: 11.0
204.0
```

[86]:
```
plt.plot(tiempos, mediciones, 'bo', label='Datos')
y=np.full_like(tiempos, solution[0])
plt.plot(tiempos, y)
```

[86]: [<matplotlib.lines.Line2D at 0x11bb50c10>]



c) Repetir con una línea recta que pasa por el origen y= dt.

[72]:
```
# Se define la matriz de diseño
A= np.vstack((tiempos,np.zeros(len(tiempos)))).T

print(A)

# esta matriz es no singular asi que busco la pseudo inversa
#solution=np.linalg.solve(A.T @ A, A.T @ mediciones)
solucion = np.linalg.pinv(A) @ mediciones



print(solution)
```

```
[[0. 0.]
 [1. 0.]
 [3. 0.]
 [4. 0.]]
[9.]
```

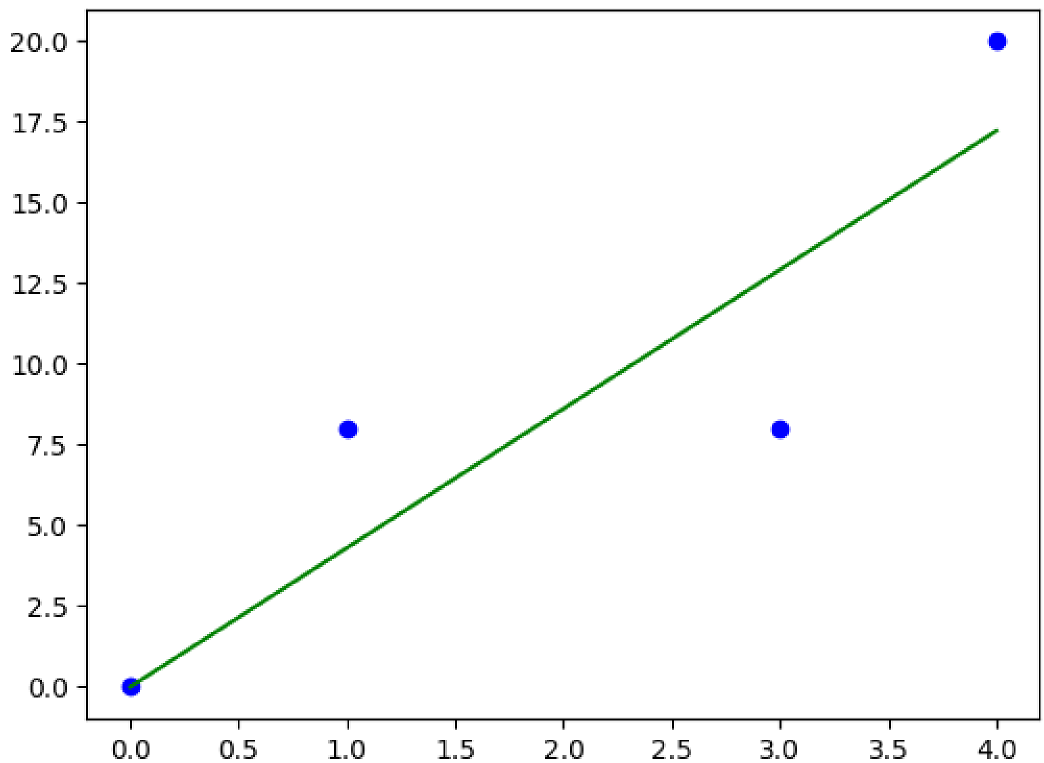
[78]:
```
## calculo el error de aproximar por medio de la recta con los parametros de␣
↪solucion
errorTotal= 0
for i in range(len(tiempos)):
    mediciones_ajustadas=solution[0]*tiempos[i]
    error=mediciones[i]-mediciones_ajustadas
    print(f"Error en el tiempo {tiempos[i]}: {error}")
    errorTotal+= error**2
print(errorTotal)
```

```
Error en el tiempo 0: 0.0
Error en el tiempo 1: -1.0
Error en el tiempo 3: -19.0
Error en el tiempo 4: -16.0
618.0
```

[77]:
```
plt.plot(tiempos, mediciones, 'bo', label='Datos')

t_recta = np.linspace(0, 4, 100)  # Valores de t para la línea
y_recta = solucion[0] * t_recta  # y = 9t
#plt.plot(t_recta, y_recta, color='blue', linestyle='--', label='y = 9t')  #␣
↪Recta
plt.plot(tiempos, solucion[0]*tiempos , "g-")
```

[77]:
```
[<matplotlib.lines.Line2D at 0x118a0e850>]
```


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0