

## Tarea 3 - Sistema de Ecuaciones Lineales, Regresión lineal e Interpolación

Realizado por:

Jonathan Andrés Rincon Ruiz

Cod: 1 111 199 510

Grupo

86

Tutor

Carlos Alberto Álvarez

Universidad Nacional Abierta y a Distancia – UNAD

Ciencias Básicas, tecnología e ingeniería

Ingeniería de sistemas

Métodos Numéricos

03 Nov 2020

## Introducción

En este trabajo se realiza ejercicios de sistemas de ecuaciones lineales, regresión lineal e interpolación utilizando el lenguaje de programación Python y Jupyter notebook, los sistemas de ecuaciones lineales, se desarrollaron utilizando tres métodos que son Gauss – Seidel, Jacobi y S.O.R, se determino que el SEL, es de solución única, al final se realizo un breve video explicando lo que trataba el ejercicios, para el tema de regresión lineal y el ultimo ejercicio es de interpolación y se desarrolló utilizando los métodos de newton y trazadores cúbicos

## Objetivo

- Aplicar métodos de solución utilizando Python para resolver sistemas de ecuación lineales
- Aplicar métodos de solución utilizando Python para resolver ajuste de curvas empleando técnicas de regresión lineal
- Aplicar métodos de solución utilizando Python para resolver problemas de interpolación

## Desarrollo de la actividad

### Ejercicio 1 – E

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1E.ipynb

jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Dado el siguiente Sistema de Ecuaciones Lineales (SEL)

$$\begin{aligned}4x_1 + x_2 + x_3 + x_4 &= 6 \\x_1 + 3x_2 + x_3 + x_4 &= 6 \\x_1 + x_2 + 5x_3 - x_4 - x_5 &= 6 \\x_2 - x_3 + x_4 &= 6 \\x_1 - x_3 + 4x_5 &= 6\end{aligned}$$

Tenemos la matriz de coeficiente:

$$\begin{bmatrix}4 & 1 & 1 & 1 & 0 \\1 & 3 & 1 & 1 & 0 \\1 & 1 & 5 & 1 & 1 \\1 & 1 & 1 & 0 & 0 \\1 & 1 & 4 & 0 & 0\end{bmatrix}$$

Vector de termino independientes

$$\begin{bmatrix}6 \\6 \\6 \\6 \\6\end{bmatrix}$$

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1E.ipynb

jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

$\begin{bmatrix}6 \\6 \\6\end{bmatrix}$

- Determinar si tiene, o no, solución. ¿Si la tiene, es única? Realice una breve explicación basándose en la teoría revisada.

```
In [1]: import numpy as np

a = np.array([[4, 1, 1, 0, 1], [1, 3, 1, 1, 0], [1, 1, 5, -1, -1], [0, 1, -1, 1, 0], [1, 0, -1, 0, 4]])
b = np.array([6, 6, 6, 6, 6])
UniAB = np.column_stack([a,b])

deterA=np.linalg.det(a)
rangoA=np.linalg.matrix_rank(a)
MAumentadaAB=np.linalg.matrix_rank(a)
print("La Matriz [A] tiene una determinante de", deterA, "> 0")
print("El rango de la matriz [A] es :", rangoA, "=", "al rango de la matriz aumentada de [A|B]:", MAumentadaAB)
print("El numero de incognitas es:", len(b))

La Matriz [A] tiene una determinante de 45.000000000000014 > 0
El rango de la matriz [A] es : 5 = al rango de la matriz aumentada de [A|B]: 5
el numero de incognitas es: 5
```

la matriz  $[A]$  son matrices cuadradas de coeficientes siendo de **solucion unica**, porque el número de ecuaciones (que corresponde a los renglones) y el número de incógnitas (que corresponde a las columnas) debe ser igual para que sea posible tener una **solución única**, en otras palabras la matriz  $[A] > 0$  y el rango de  $A$  es igual al número de incognitas  $X_n = 5$ . Chapra, Steven. Canale, Raymond (2007) Metodos numericos para ingenieros. pag 237.

Resolver el SEL por cada uno de los siguientes métodos, realizando una descripción detallada de cómo elaboró el algoritmo:

Windows taskbar: Llaves que abren puertas Espiritu, Documents/metodos numericos, G86\_Jonathan Andres Rincon\_Ej1E, (3) WhatsApp

Browser: localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter: G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

- Gauss – Seidel
- Jacobi
- S.O.R.

**METODO GAUSS - SEIDEL**

```
In [21]: g_xi = np.zeros((5,1), dtype=float)
g_x0 = np.zeros((5,1), dtype=float)
g_normaA = np.array([], dtype=float)
g_itr = 0
g_ea = 100.0
g_v_ea = []
sep = "\t"
n = len(a)
tol = 10**(-6)
print(1)
while float(np.linalg.norm(np.dot(a, g_xi)-b)) >= tol:
    g_itr = g_itr + 1

    for i in range(0, n):
        g_x0[i] = float(g_xi[i])
        g_xi[i] = float(b[i])/float(a[i][i])
        for j in range(0, n):
            if i != j:
                g_xi[i] = g_xi[i] - a[i][j] * g_xi[j] / a[i][i]
        g_ea = float(abs((g_xi[i] - g_x0[i]) / g_xi[i])) * 100
        g_v_ea.append(g_ea)
    print("Iteracion n°:", g_itr)
    print("Xi: ", g_xi.T)
    print("Norma: ", g_ea)
```

Windows taskbar: Llaves que abren puertas Espiritu, Documents/metodos numericos, G86\_Jonathan Andres Rincon\_Ej1E, (3) WhatsApp

Browser: localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter: G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
print("error: ", g_ea, "\n")

g_normaA = np.hstack([g_normaA, [float(np.linalg.norm(np.dot(a, g_xi)-b))]])

1
Iteracion n°: 1
Xi: [[1.5  1.5  0.6  5.1  1.275]]
error: 100.0

Iteracion n°: 2
Xi: [[ 0.65625 -0.11875  2.3675  8.48625  1.9278125]]
error: 33.86286270059978

Iteracion n°: 3
Xi: [[ 0.45585938 -1.76986979  3.54561458 11.31548437  2.2724388 ]]
error: 15.165482202089922

Iteracion n°: 4
Xi: [[ 0.4879541 -3.11635102  4.44326402 13.55961504  2.48882748]]
error: 8.694402448253765

Iteracion n°: 5
Xi: [[ 0.54606488 -4.18298131  5.13707179 15.3200531  2.64775173]]
error: 4.444444444444444
```

**METODO JACOBI**

```
In [3]: j_xold = np.zeros((5,1), dtype = float)
j_xi = np.zeros((5,1), dtype = float)
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

### METODO JACOBI

```
In [3]: j_xold = np.zeros((5,1), dtype = float)
j_xi = np.zeros((5,1), dtype = float)
j_n_a = []
j_n_a.append(float(np.linalg.norm(np.dot(a, j_xi) - b)))
j_itr = 0
j_ea = np.zeros((5,1), dtype = float)
mj_ea = np.array([], dtype = float)

while j_n_a[-1] >= tol:
    j_itr = j_itr + 1
    j_xold = j_xi
    j_xi = np.zeros((5,1), dtype=float)
    for i in range(0, n):
        for j in range(0, n):
            if i == j:
                j_xi[i] = j_xi[i] + b[i] / a[i][i]
            else:
                j_xi[i] = j_xi[i] - a[i][j] * j_xold[j] / a[i][i]
        j_ea[i] = abs((j_xi[i] - j_xold[i]) / j_xi[i]) * 100
    mj_ea = np.append(mj_ea, j_ea)
    j_n_a.append(float(np.linalg.norm(np.dot(a, j_xi) - b)))
    print("Iteracion n°: ", j_itr)
    print("Xi: ", j_xi.T)
    print("el error de tolerancia es: ", j_n_a[-1], "\n")

Iteracion n°: 1
Xi: [[1.5 2. 1.2 6. 1.5]]
el error de tolerancia es: 23.928017051147386
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
print("Xi: ", j_xi.T)
print("el error de tolerancia es: ", j_n_a[-1], "\n")

Iteracion n°: 1
Xi: [[1.5 2. 1.2 6. 1.5]]
el error de tolerancia es: 23.928017051147386

Iteracion n°: 2
Xi: [[ 0.325 -0.9 2. 5.2 1.425]]
el error de tolerancia es: 13.027255083094058

Iteracion n°: 3
Xi: [[ 0.86875 -0.50833333 2.64 8.9 1.91875 ]]
el error de tolerancia es: 13.5767410830987

Iteracion n°: 4
Xi: [[ 0.48739583 -2.13625 3.29166667 9.14833333 1.9428125 ]]
el error de tolerancia es: 7.951505998576577

Iteracion n°: 5
Xi: [[ 0.72544271 -2.30913194 3.748 11.42791667 2.20106771]]
el error de tolerancia es: 8.858775540502917
```

### METODO DE S.O.R

```
In [10]: import math as m

S_xi = np.zeros((5,1), dtype=float)
S_X0 = np.zeros((5,1), dtype=float)
S_NormaA = np.array([], dtype=float)
S_itr = 0
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```

METODO DE S.O.R

In [10]: import math as m

S_Xi = np.zeros((S,1), dtype=float)
S_X0 = np.zeros((S,1), dtype=float)
S_NormaA = np.array([], dtype=float)
S_itr = 0
S_Ea = 100.0
S_V_Ea = []
sep = "\t"
S_lambda = 1.4

while float(np.linalg.norm(np.dot(a, S_Xi)-b)) >= tol:
    S_itr = S_itr + 1

    for i in range(0, n):
        S_X0[i] = float(S_Xi[i])
        S_Xi[i] = float(b[i])/float(a[i][i])
        for j in range(0, n):
            if i != j:
                S_Xi[i] = S_Xi[i] - a[i][j] * S_Xi[j] / a[i][i]
        S_Xi[i] = S_lambda * S_Xi[i] + (1 - S_lambda) * S_X0[i]
        S_Ea = float(abs((S_Xi[i] - S_X0[i]) / S_Xi[i])) * 100
        S_V_Ea.append(S_Ea)
        S_NormaA = np.hstack((S_NormaA, [float(np.linalg.norm(np.dot(a, S_Xi)-b))]))

    print("Iteracion n°:", S_itr)
    print("Xi: ", S_Xi.T)
    print("error: ", S_Ea, "\n")

```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```

Iteracion n°: 1
Xi: [[2.1      1.82      0.5824      6.66736      1.56884]]
error: 100.0

Iteracion n°: 2
Xi: [[-0.129934 -1.25058547  4.13972145 13.27948568  2.96684341]]
error: 47.12090311505205

Iteracion n°: 3
Xi: [[ 0.10238081 -4.87650686  5.90983886 18.18908972  2.94587295]]
error: 0.7118587788528543

Iteracion n°: 4
Xi: [[ 0.66632594 -6.80651604  6.95310723 20.38783669  3.12202427]]
error: 5.642214904542866

Iteracion n°: 5
Xi: [[ 0.68945421 -7.55824605  7.40477989 21.19310163  3.20155428]]
error: 2.4841061895572403

Iteracion n°: 6
Xi: [[ 0.75738747 -7.87582711  7.5417548  21.50737402  3.19390685]]
error: 0.23943800573169152

Iteracion n°: 7
Xi: [[ 0.79610292 -7.97744397  7.59043222 21.59207705  3.20045251]]
error: 0.20452296841148268

Iteracion n°: 8

```

```
Iteration n°: 12
Xi: [[ 0.80009019 -8.00056814  7.6002798  21.60059716  3.20001104]]
error: 0.003977401112691932

Iteration n°: 13
Xi: [[ 0.80006098 -8.00021045  7.60010023  21.60019608  3.20000932]]
error: 5.357594839439677e-05

Iteration n°: 14
Xi: [[ 0.80001092 -8.0000592  7.60003094  21.60004776  3.20000328]]
error: 0.00018891477294240012

Iteration n°: 15
Xi: [[ 0.80000437 -8.00001509  7.60000491  21.6000089  3.19999888]]
error: 0.00013746361669895313

Iteration n°: 16
Xi: [[ 0.8000022 -8.00000144  7.6  21.59999845  3.19999968]]
error: 2.4958001966873862e-05

Iteration n°: 17
Xi: [[ 0.79999974 -7.99999858  7.59999915  21.59999744  3.19999993]]
error: 7.766810348821538e-06

Iteration n°: 18
Xi: [[ 0.79999993 -7.99999895  7.59999933  21.59999861  3.19999982]]
error: 3.3354353117770407e-06

Iteration n°: 19
Xi: [[ 0.79999996 -7.99999944  7.59999968  21.59999932  3.19999998]]
error: 4.921280253691352e-06
```

• Debe establecer los criterios de convergencia en cada método. Itere la solución hasta que ésta converja a una tolerancia de  $\|A\| \leq 10^{-6}$ , donde  $\|A\|$  es una norma matricial para la matriz  $A$  (escoja la que considere más adecuada). Valide que el vector  $x$  es solución del SEL (mediante la operación  $Ax = b$ )

**Prueba de la solución obtenida  $Ax_i = B$ , por el método de Gauss – Seidel**

```
In [12]: M a_inv = np.linalg.inv(a)
x = np.dot(a_inv, b)
print("Respuesta real X: ", x)
print("Respuesta hallada Xi: ", g_xi.T)
print("*****")
s = np.dot(a, g_xi)
print("B: ", b.T)
print("Prueba A Xi = B: ", s.T)
print("Norma Error = ", g_normaA[-1])

Respuesta real X: [ 0.8 -8.  7.6 21.6 3.2]
Respuesta hallada Xi: [[ 0.79999997 -7.9999995  7.59999967 21.59999917 3.19999993]]
*****
B: [ 6 6 6 6]
Prueba A Xi = B: [[5.99999997 6.00000032 5.99999975 6. 6. ]]
Norma Error = 9.237060990882992e-07
```

**• Prueba de la solución obtenida  $AX_i = B$ , por el método de Jacobi**

```
In [13]: M Ainv = np.linalg.inv(a)
Xreal=np.dot(Ainv, b)
print("Respuesta real X:", sep, Xreal)
```



Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

**- Prueba de la solución obtenida  $AX_i = B$ , por el método de Jacobi**

```
In [13]: Ainv = np.linalg.inv(a)
Xreal = np.dot(Ainv, b)
print("Respuesta real X:", sep, Xreal)
print("Respuesta hallada Xi:", sep, j_xi.T)
print("*****")
S = np.dot(a, j_xi)
print("B: ", b.T)
print("Prueba AXi=B: ", S.T, sep, "Norma Error=", j_h_a[-1])

Respuesta real X: [ 0.8 -8. 7.6 21.6 3.2]
Respuesta hallada Xi: [[ 0.79999999 -7.99999901 7.59999952 21.59999867 3.19999987]]
*****
B: [ 6 6 6 6]
Prueba AXi=B: [[ 5.99999999 6.00000024 5.99999972 5.99999985 5.99999994]] Norma Error= 9.084809484218036e-07
```

**Prueba de la solución obtenida  $AX_i = B$ , por el método S.O.R.**

```
In [17]: Ainv = np.linalg.inv(a)
Xreal = np.dot(Ainv, b)
print("Respuesta real X:", sep, Xreal)
print("Respuesta hallada Xi:", sep, s_xi.T)
print("*****")
S = np.dot(a, s_xi)
print("B: ", b.T)
print("Prueba AXi = B: ", S.T, sep, "Norma Error = ", s_NormaA[-1])

Respuesta real X: [ 0.8 -8. 7.6 21.6 3.2]
Respuesta hallada Xi: [[ 0.79999994 -7.99999973 7.59999987 21.59999972 3.19999999]]
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej1Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej1E Last Checkpoint: hace 21 horas (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

- Realizar una gráfica de la forma como va convergiendo la solución (Número de iteraciones vs norma del error). Realice una breve explicación, sustentándose en la teoría revisada, acerca de los resultados. ¿Cuál considera que es el mejor método para el SEL seleccionado en particular? ¿por qué?

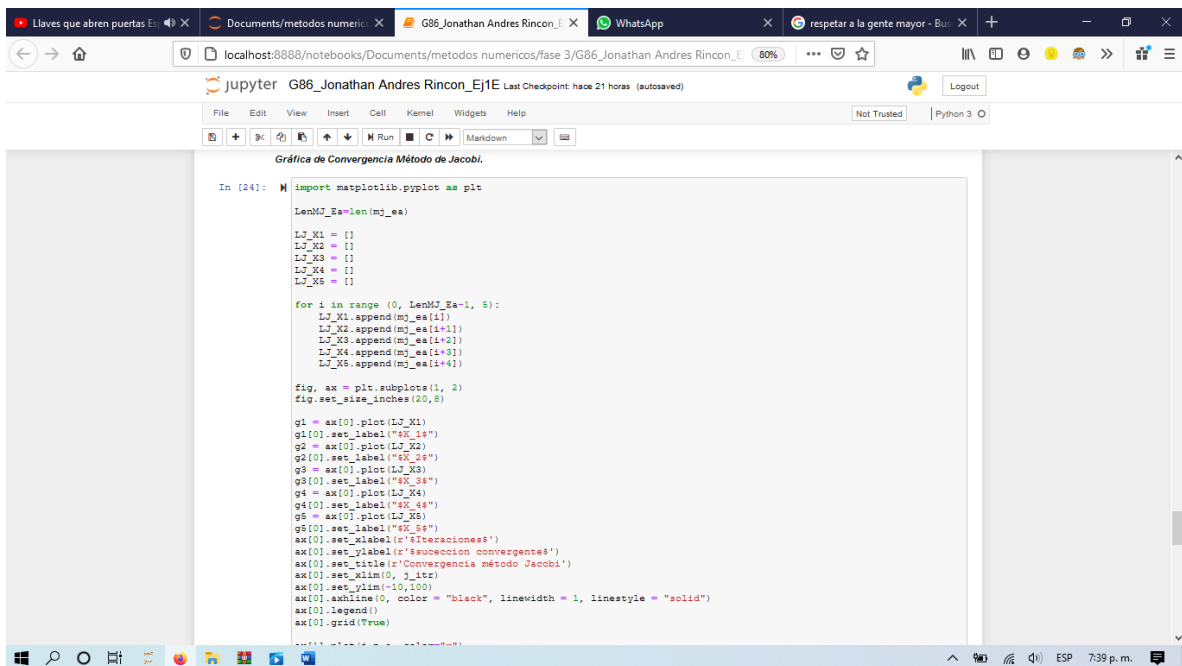
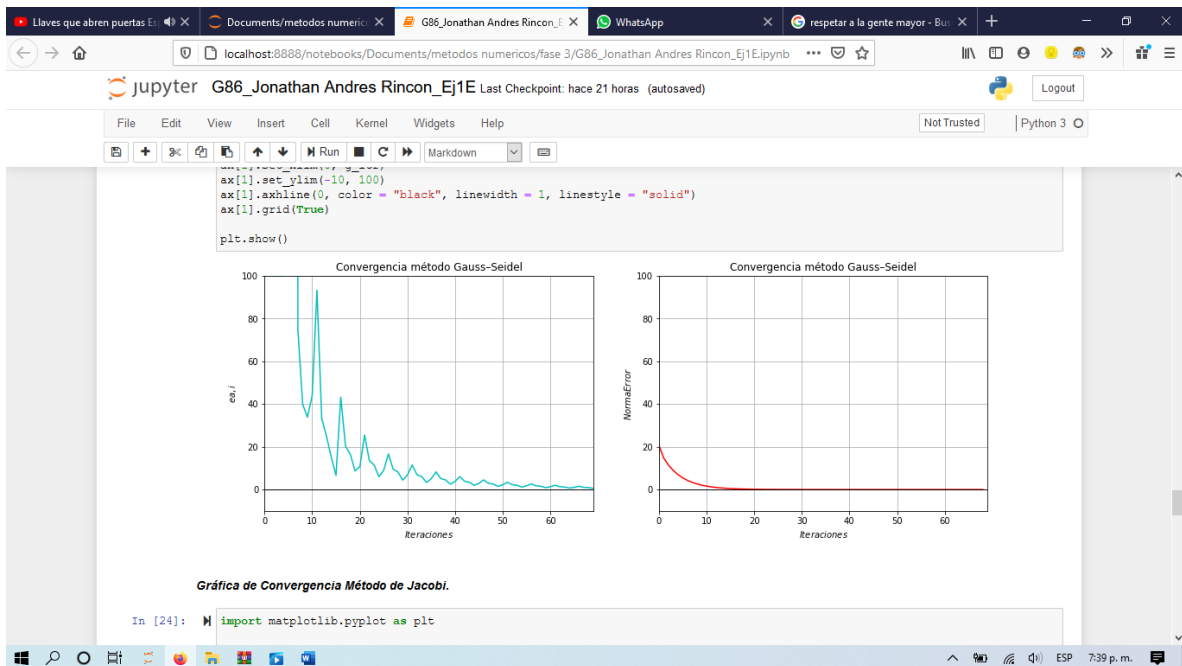
**Gráfica de Convergencia método Gauss – Seidel.**

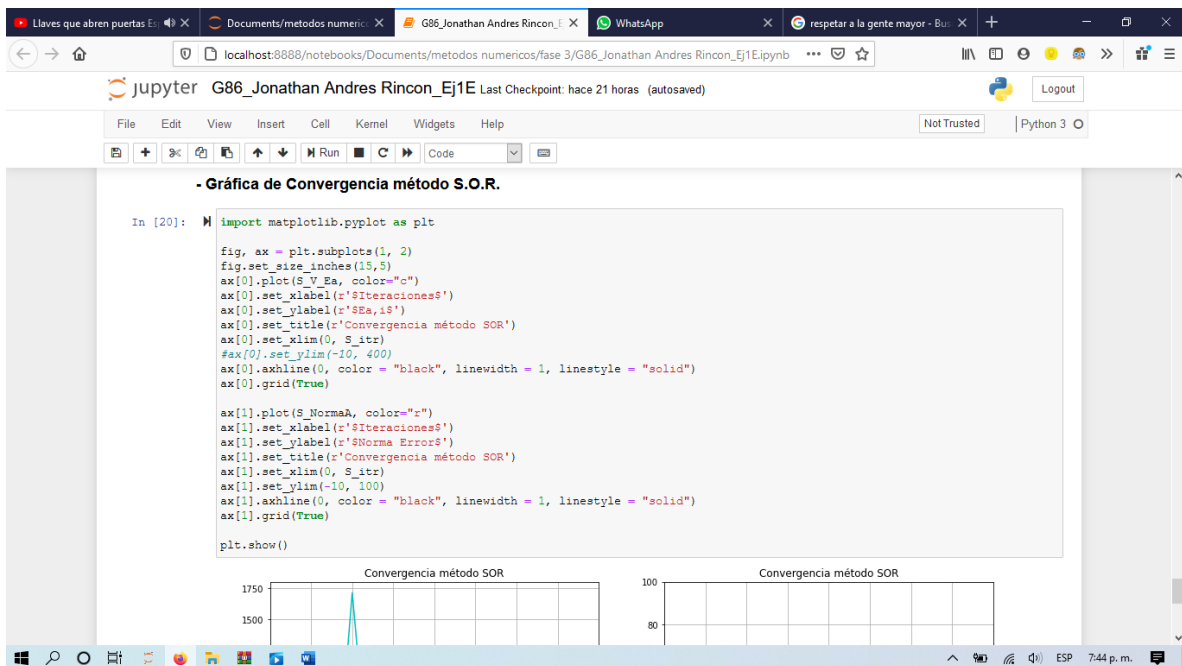
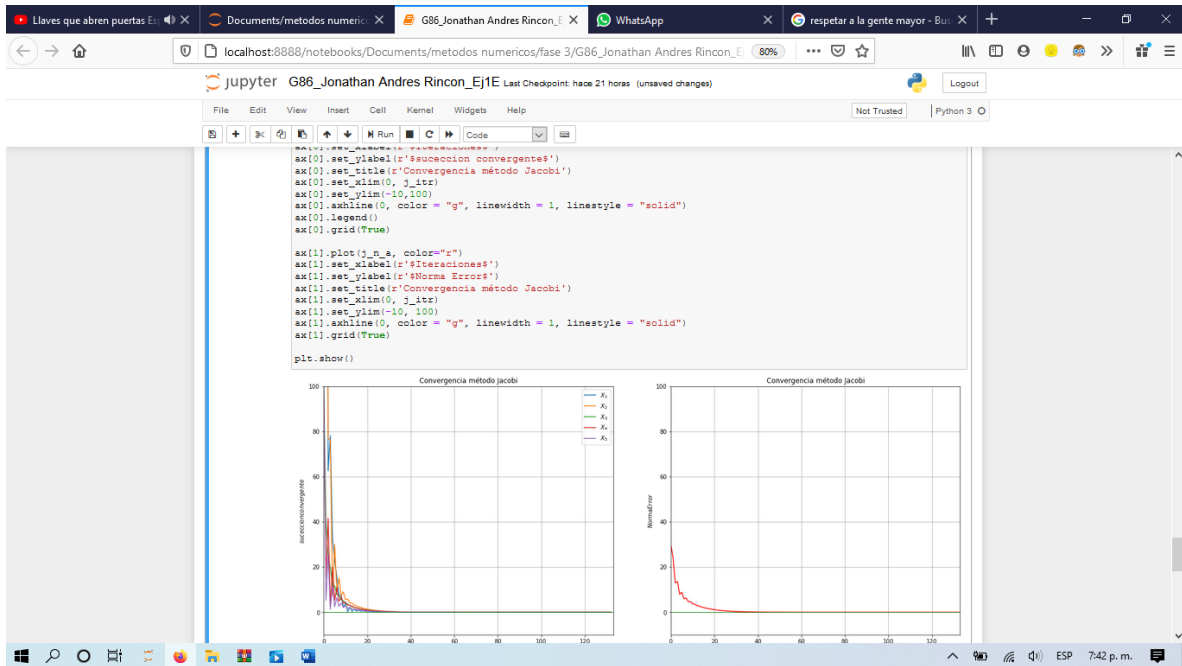
```
In [18]: import matplotlib.pyplot as plt

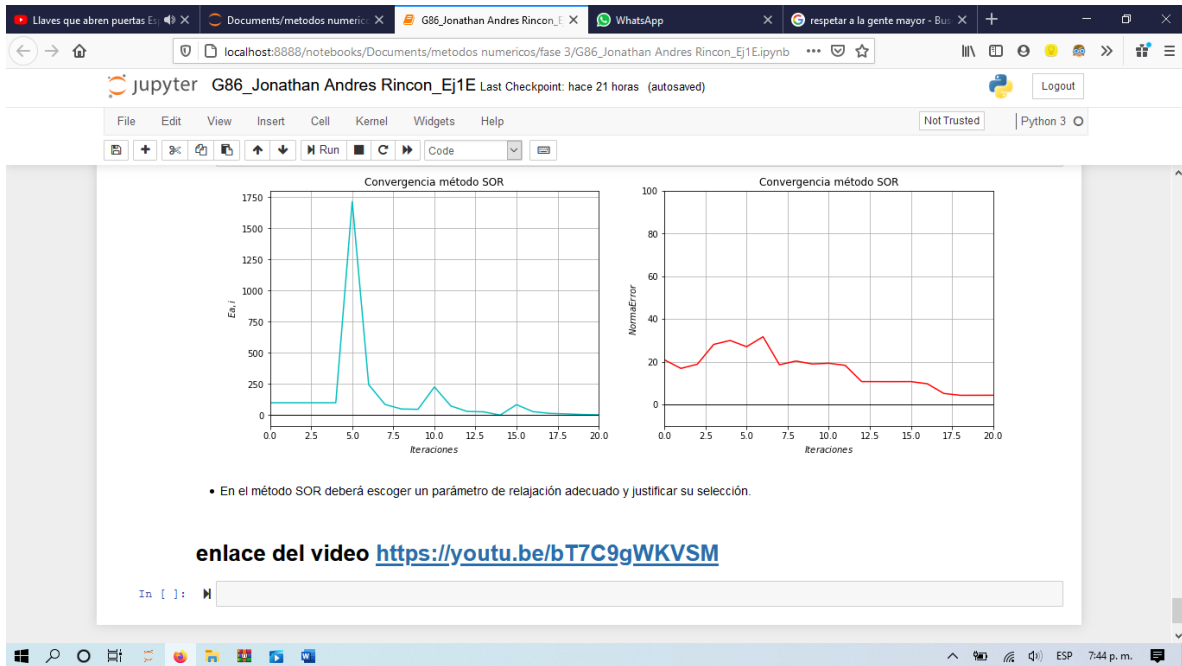
fig, ax = plt.subplots(1, 2)
fig.set_size_inches(15,5)
ax[0].plot(g_gauss, color="c")
ax[0].set_xlabel(r'$Iteraciones$')
ax[0].set_ylabel(r'$\epsilon$')
ax[0].set_title(r'Convergencia método Gauss-Seidel')
ax[0].set_xlim(0, g_itr)
ax[0].set_ylim(-10, 100)
ax[0].axhline(0, color = "black", linewidth = 1, linestyle = "solid")
ax[0].grid(True)

ax[1].plot(g_normaA, color="r")
ax[1].set_xlabel(r'$Iteraciones$')
ax[1].set_ylabel(r'$\epsilon$Norma Error$')
ax[1].set_title(r'Convergencia método Gauss-Seidel')
ax[1].set_xlim(0, g_itr)
ax[1].set_ylim(-10, 100)
ax[1].axhline(0, color = "black", linewidth = 1, linestyle = "solid")
ax[1].grid(True)

plt.show()
```







## Ejercicio 2 – E

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej2E.ipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej2E Last Checkpoint: ayer a las 18:36 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Kernel starting, please wait... Trusted Python 3

### Regresión lineal

Descripción del ejercicio: Para el conjunto de datos:

x	0	30	10	15	5	25	35	40
y	4	1	2	2	3	1	0	1

- Ajuste los datos a una línea recta empleando el método de los mínimos cuadrados, realizando una descripción teórica detallada.

```
In [1]: import numpy as np
import math as m

x = [0, 30, 10, 15, 5, 25, 35, 40]
y = [4, 1, 2, 2, 3, 1, 0, 1]
n = len(x)
sep="\t"

print("x:", x)
print("y:", y)
print("Cantidad de datos (n):", n)

x: [0, 30, 10, 15, 5, 25, 35, 40]
y: [4, 1, 2, 2, 3, 1, 0, 1]
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej2E.ipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej2E Last Checkpoint: ayer a las 18:36 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

```
y: [4, 1, 2, 2, 3, 1, 0, 1]
Cantidad de datos (n): 8
```

Las listas de datos no están ordenadas. Se pasan las listas [x],[y] a tuplas (x,y) con la función zip de python, para ordenarlas con luego con respecto a x con el método sort de python, y se devuelven luego como listas ordenadas. (Moracho, 2017)

```
In [2]: tuplelist = [e for e in zip(x, y)]
tuplelist.sort(key=lambda tuplelist: tuplelist[0])
xord = []
yord = []
xord, yord = [list(e) for e in zip(*tuplelist)]

print("x:", xord)
print("y:", yord)

x: [0, 5, 10, 15, 25, 30, 35, 40]
y: [4, 3, 2, 2, 1, 1, 0, 1]
```

- Calcule la pendiente y la intersección, así como el error estándar y el coeficiente de correlación.

Se crea la función de regresión por mínimos cuadrados, basándose en la codificación propuesta en el material de referencia (Canale, 2007). Posteriormente se evalúa usando los datos del ejercicio.

De acuerdo con la teoría, lo que trata este método es buscar la ecuación de una recta, con la cual la distancia de la nube de puntos dados a ésta recta, sea mínima y de esta manera explicar aproximadamente el comportamiento de los datos (asociados a algún fenómeno). Teniendo en cuenta que la ecuación de una recta (en forma pendiente - intercepción) es  $y = a_0 + a_1x + e$ , donde  $a_1$  es la pendiente de la recta,  $a_0$  el intercepción y  $e$  el error. Despejando  $e$ , se calculan de la siguiente forma:

$$e = y - a_0 - a_1x$$

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej2Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej2E Last Checkpoint: ayer a las 18:36 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

$$\sum_{i=1}^n a_i = \sum_{i=1}^n (y_i - a_0 - a_1 x_i), \quad a_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2}, \quad a_0 = \bar{y} - a_1 \bar{x}$$

Asociados con el metodo tambien se calculan; error estándar  $S_{y/x} = \sqrt{\frac{s_r}{n-2}}$  que cuantifica la dispersión de los datos, la magnitud del error residual  $S_y$  asociado con la variable dependiente (y) antes de la regresión y el error después de realizar la regresión  $S_y$ . Ambos sirven para calcular además, el coeficiente de determinación  $r^2 = \frac{S_y - S_{y/x}}{S_y}$  que es la norma de  $S_y$  por diferencia de los anteriores  $r^2 = \frac{S_y - S_{y/x}}{S_y}$  que explica cuanto explica el modelo lineal (regresión) el conjunto de datos dados.

```
In [7]: def Regress(x, y, n):
sumx = 0
sumxy = 0
st = 0
sumy = 0
sumx2 = 0
sz = 0

for i in range(1, n):
    sumx = sumx + x[i]
    sumy = sumy + y[i]
    sumxy = sumxy + x[i]*y[i]
    sumx2 = sumx2 + x[i]*x[i]

xm = sumx / n
ym = sumy / n
a1 = (n * sumxy - sumx * sumy) / (n * sumx2 - sumx * sumx)
a0 = ym - a1 * xm
for i in range(1, n):
    st = st + (y[i] - ym) ** 2
    sr = sr + (y[i] - a1 * x[i] - a0) ** 2
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej2Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej2E Last Checkpoint: ayer a las 18:36 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
syx = (sr / (n - 2)) ** 0.5
r2 = (st - sr) / st
Reg_Lin = [a1, a0, st, sr, syx, r2]
return Reg_Lin

Reg_Lin = Regress(xord, yord, n)

print("Pendiente a1:", Reg_Lin[0])
print("Intercepto a0:", Reg_Lin[1])
print("Error estandar St:", Reg_Lin[2])
print("Error residual Sr:", Reg_Lin[3])
print("Error estandar estimado Sy/x:", Reg_Lin[4])
print("R2:", Reg_Lin[5])

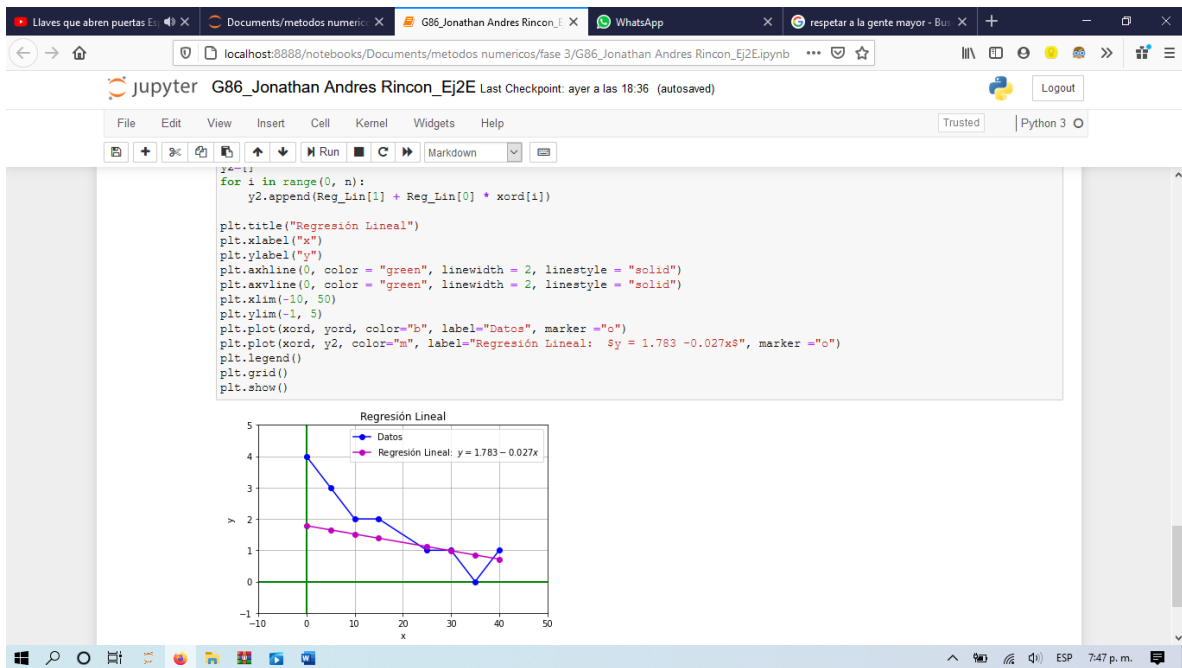
Pendiente a1: -0.02666666666666667
Intercepto a0: 1.7833333333333332
Error estandar St: 5.9375
Error residual Sr: 3.2530555555555556
Error estandar estimado Sy/x: 0.7363259644518356
R2: 0.4521169590643275
```

De acuerdo con los resultados de la regresión, la ecuación de la recta que representa el modelo lineal de los datos dados es:  $y = 1.783 - 0.027x$ ; y de acuerdo con el coeficiente de correlación  $r^2 = 0.45$ , se indica que el modelo lineal sólo explica el 45% del comportamiento de los datos...

- Realice una gráfica en donde se muestren los datos originales y la recta que ajusta

```
In [13]: import matplotlib.pyplot as plt

y2=[]
```



## Ejercicio 3 – E

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 4 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

### Interpolacion

x	1	2	3	4	5	6	7	8	9	10	11
y	-0,5	0,26873	0,70424	1,00412	1,23127	1,23127	?	0,70424	0	-0,5	0

Ajuste a un polinomio de interpolación los datos dados, empleando cada uno de los siguientes métodos:

- Newton
- Lagrange
- Trazadores cúbicos

• Con el polinomio encontrado, determine el valor en  $x = 7$

### Newton

La fórmula general para un polinomio de n-ésimo grado es:

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

fórmula para calcular valores intermedios (Canale 2007). La forma más simple de interpolación consiste en unir dos puntos con una línea recta para ello se

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 4 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

usa la siguiente fórmula:

$$f_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

Donde  $x$  es un punto entre  $x_0$  y  $x_1$  que queremos encontrar

El análisis anterior puede generalizarse para ajustar un polinomio de n-ésimo grado a  $n+1$  datos. El polinomio de n-ésimo grado es:

$$f_n(x) = b_0 + b_1(x - x_0) + \dots + b_n(x - x_0)(x - x_1) \dots (x - x_{n-1})$$

Usamos estos datos y las siguientes ecuaciones para evaluar los coeficientes:

$$\begin{aligned} b_0 &= f(x_0) \\ b_1 &= f[x_1, x_0] \\ b_2 &= f[x_2, x_1, x_0] \\ b_n &= f[x_n, x_{n-1}, \dots, x_1, x_0] \end{aligned}$$

la primera diferencia dividida finita en forma general se representa como:

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

segunda diferencia dividida finita, que representa la diferencia de las dos primeras diferencias divididas, se expresa en forma general como:

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

Estas diferencias sirven para evaluar los coeficientes en las ecuaciones  $b_0$  a  $b_n$ , los cuales se sustituirán en la ecuación



Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej3Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 4 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run

Las diferencias sirven para evaluar las diferencias en las evaluaciones de  $u_i$ , las cuales se sustituyen en la ecuación

$$f_n(x) = b_0 + b_1(x-x_0) + \dots + b_n(x-x_0)(x-x_1) \dots (x-x_{n-1})$$

polinomio de interpolación de Newton en diferencias divididas (Canale 2007)

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

In [2]: x1= [1,2,3,4,5,6,8,9,10,11]
y2= [-0.5,0.26873,0.70424, 1.00412, 1.23127, 1.23127, 0.70424,0,-0.5,0]
xi=7
n=len(x1)

In [3]: def NewtInt (x, y, n, xi):

    fdd=[None for x in range(n)] for x in range (n)
    yint=[None for x in range(n)]
    ea=[None for x in range(n)]

    for i in range(n):
        fdd[i][0]=y[i]

    print("#i \t x_i \t f(x_i) \t diferencia")
    for j in range(1,n):
        for i in range(n-j):
            fdd[i][j] = (fdd[i+1][j-1]-fdd[i][j-1])/(x[i+j]-x[i])
            print ("(0) \t(1) \t(2:.5f) \t(3:,.5f)".format(i,x[i],y[i],fdd[i][j]))

        xterm = 1
        yint[0]=fdd[0][0]

        for order in range(1,n):
            xterm = xterm * (xi-x[order-1])
            yint2 = yint[order-1] + fdd[0][order] * xterm

            yint = yint[order-1] + fdd[0][order] * xterm
            ea[order-1] = yint2-yint[order-1]
            yint[order] = yint2

        #print(ea[order-1])
        return yint[order], ea[order-1]
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej3Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 4 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run

```
yint = yint[order-1] + fdd[0][order] * xterm
ea[order-1] = yint2-yint[order-1]
yint[order] = yint2

#print(ea[order-1])
return yint[order], ea[order-1]
```

```
In [4]: yi,ea=NewtInt(x1, y2, n, xi)
```

#i	x_i	f(x_i)	diferencia
0	1	-0.50000	0.76873
1	2	0.26873	0.43551
2	3	0.70424	0.29988
3	4	1.00412	0.22715
4	5	1.23127	0.00000
5	6	1.23127	-0.26352
6	8	0.70424	-0.70424
7	9	0.00000	-0.50000
8	10	-0.50000	0.50000
0	1	-0.50000	-0.16661
1	2	0.26873	-0.06782
2	3	0.70424	-0.03636
3	4	1.00412	-0.11358
4	5	1.23127	-0.08784
5	6	1.23127	-0.14691
6	8	0.70424	0.10212
7	9	0.00000	0.50000
0	1	-0.50000	0.03293
1	2	0.26873	0.01048
2	3	0.70424	-0.02574
3	4	1.00412	0.00643
4	5	1.23127	-0.01477
5	6	1.23127	0.06226

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej3Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 5 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

1 2 0.26873 -0.00004  
0 1 -0.50000 -0.00001

Con el polinomio encontrado, determine el valor en  $x = 7$

```
In [5]: print("el valor de x=7 es f(x)=",round(y1,5))
el valor de x=7 es f(x)= 1.06292
```

```
In [6]: print("el valor del error es",ea)
el valor del error es 0.03000673809523824
```

Como podemos ver ya tenemos todas las y correspondientes

x	1	2	3	4	5	6	7	8	9	10	11
y	-0,5	0,26873	0,70424	1,00412	1,23127	1,23127	1,06292	0,70424	0	-0,5	0

```
In [16]: x2= [1,2,3,4,5,6,7,8,9,10,11]
y3= [-0.5,0.26873,0.70424, 1.00412, 1.23127, 1.23127,1.06292, 0.70424,0,-0.5, 0]
```

```
In [17]: import numpy as np
import matplotlib.pyplot as plt

plt.xlabel("eje x")
plt.ylabel("eje y")
```

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej3Eipynb

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 5 minutos (autosaved)

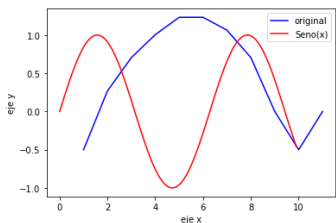
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
plt.plot(x2,y3,"blue", label="original")
plt.legend()

t = np.arange(0.0, 10.0, 0.10)
s = np.sin(t)

plt.plot(t,s,"red", label="Seno(x)")
plt.legend()

plt.show()
```



$x=7$  es en realidad  $\text{seno}(7)$ .

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej3Eipyb

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 5 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

$$f_n(x) = \sum_{i=0}^n L_i(x)f(x_i)$$

donde

$$L_i(x) = \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j}$$

donde  $\prod$  designa el "producto de" (chapa 2007)

```
In [9]: def Lagrng(x1, y2, n, xi):
sum = 0
for i in range(n):
    product = y2[i]
    for j in range(n):
        if i != j:
            product = product*(xi-x1[j])/(x1[i]-x1[j])
    sum = sum + product
#Lagrng = sum
return sum

In [10]: y1=Lagrng(x1, y2, n,xi)

In [11]: print("el valor de x=7 es f(x)=",y1)

el valor de x=7 es f(x)= 1.062919238095238
```

Windows Taskbar: 7:54 p. m.

Localhost:8888/notebooks/Documents/metodos numericos/fase 3/G86\_Jonathan Andres Rincon\_Ej3Eipyb

Jupyter G86\_Jonathan Andres Rincon\_Ej3E Last Checkpoint: hace 6 minutos (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

### Trazadores cúbicos

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

Las incógnitas se evalúan empleando la siguiente ecuación

$$\begin{aligned} & (x_i - x_{i-1})f''(x_{i-1}) + 2(x_{i+1} - x_{i-1})f''(x_i) + (x_{i+1} - x_i)f''(x_{i+1}) \\ &= \frac{6}{x_{i+1} - x_i}[f(x_{i+1}) - f(x_i)] + \frac{6}{x_i - x_{i-1}}[f(x_{i-1}) - f(x_i)] \end{aligned}$$

```
In [12]: def traza(x,y):
h = np.zeros(n-1, dtype = float)
for j in range(n-1):
    h[j] = x[j+1] - x[j]

A = np.zeros(shape=(n-2,n-2), dtype = float)
B = np.zeros(n-2, dtype = float)
S = np.zeros(n, dtype = float)

A[0,0] = 2*(h[0]+h[1])
A[0,1] = h[1]
B[0] = 6*((y[2]-y[1])/h[1] - (y[1]-y[0])/h[0])

for i in range(1,n-3):
    A[i,i-1] = h[i]
```

Windows Taskbar: 7:54 p. m.

```

G86_Jonathan Andres Rincon_Ej3E Last Checkpoint: hace 6 minutos (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
A[i+1,i] = h[i]
A[i,i] = 2*(h[i]+h[i+1])
A[i,i+1] = h[i+1]
factor21 = (y[i+2]-y[i+1])/h[i+1]
factor10 = (y[i+1]-y[i])/h[i]
B[i] = 6*(factor21 - factor10)

A[n-3,n-4] = h[n-3]
A[n-3,n-3] = 2*(h[n-3]+h[n-2])
factor12 = (y[n-1]-y[n-2])/h[n-2]
factor23 = (y[n-2]-y[n-3])/h[n-3]
B[n-3] = 6*(factor12 - factor23)

r = np.linalg.solve(A,B)
for j in range(1,n-1):
    S[j] = r[j-1]
S[0] = 0
S[n-1] = 0

a = np.zeros(n-1, dtype = float)
b = np.zeros(n-1, dtype = float)
c = np.zeros(n-1, dtype = float)
d = np.zeros(n-1, dtype = float)

for j in range(n-1):
    a[j] = (S[j+1]-S[j])/(6*h[j])
    b[j] = S[j]/2
    factor10 = (y[j+1]-y[j])/h[j]
    c[j] = factor10 - (2*h[j]*S[j]+h[j]*S[j+1])/6
    d[j] = y[j]

```

```

G86_Jonathan Andres Rincon_Ej3E Last Checkpoint: hace 6 minutos (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
return(tabla1)

In [13]: M tabla1 = traza(x1,y2)

print("tramos:")
for i in range(1,n):
    print(i, ' x = [' +str(x1[i-1])+'+', '+str(x1[i])+'+', ' ', ' ',str(tabla1[i-1]))

tramos:
1 x = [1,2] 0.26873
2 x = [2,3] 0.70424
3 x = [3,4] 1.00412
4 x = [4,5] 1.23127
5 x = [5,6] 1.23127
6 x = [6,8] 0.70424
7 x = [8,9] 0.0
8 x = [9,10] -0.5
9 x = [10,11] -5.551115123125783e-17

In [14]: M print("El valor en el tramo donde se encuentra el valor x=7 es: ",tabla1[xi-1])

El valor en el tramo donde se encuentra el valor x=7 es: 0.0



| x | 1    | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9 | 10   | 11 |
|---|------|---------|---------|---------|---------|---------|---------|---------|---|------|----|
| y | -0,5 | 0,26873 | 0,70424 | 1,00412 | 1,23127 | 1,23127 | 0,70424 | 0,70424 | 0 | -0,5 | 0  |



In [15]: M import numpy as np

```

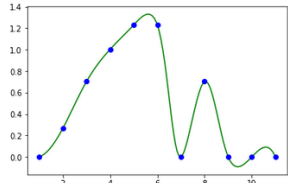
Localhost: 8888 / Documents / metodos numericos / fase 3 / G86\_Jonathan Andres Rincon\_EJ3E

Jupyter G86\_Jonathan Andres Rincon\_EJ3E Last Checkpoint: hace 9 minutos (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

x	1	2	3	4	5	6	7	8	9	10	11
y	-0.5	0.26873	0.70424	1.00412	1.23127	1.23127	0	0.70424	0	-0.5	0

```
In [18]: import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
points = [(1,0.0005), (2,0.26873), (3,0.70424), (4,1.00412), (5,1.23127), (6,1.23127), (7,0)]
points = points + [(8,0.70424), (9,0), (10,0.0005), (11,0)]
data = np.array(points)
tck,u = interpolate.splprep(data.transpose(), s=0)
unew = np.arange(0, 1.01, 0.01)
out = interpolate.splev(unew, tck)
plt.figure()
plt.plot(out[0], out[1], color='g')
plt.plot(data[:,0], data[:,1], 'ob')
plt.show()
```



### Referencias bibliográficas

Canale, R. P. y P. Canale, R. (2007). Métodos numéricos para ingenieros (5a. ed.). México D.F, Mexico: McGraw-Hill Interamericana. Recuperado de <https://elibro-net.bibliotecavirtual.unad.edu.co/es/lc/unad/titulos/73710>