

SOLUCIONES COMPUTACIONALES A PROBLEMAS EN INGENIERÍA

- **Professor:** Lic. Ing. Alfonso M. Mancilla Herrera, M.Sc. Dr.
- **Estudiantes:** María Isabel Solá Valle 200169403 María José Duque Polo 200138289



Figure 1: Fotografía de María Isabel Solá y María José Duque Polo

Topics:

1. Series de Taylor
2. Búsqueda de raíces
3. Álgebra de matrices
4. Interpolación Funcional
5. Ajuste de Curvas
6. Solución Numérica de Ecuaciones Diferenciales Ordinarias
7. Matrices, Funciones y Gráficas en Matlab©.

INFORME

0.1 Sistemas Numéricos de base $b > 1$

Parte entera

$$x_{(b)} \sum_{i=0}^n c_i \cdot b^i$$

Parte fraccionaria

$$x_{(b)} \sum_{i=-1}^{\infty} c_i \cdot b^i$$

Integral

$$x_{(b)} \int_{i=0}^n c_i \cdot b^i$$

1 El Error

El error absoluto representa la diferencia entre dos magnitudes denominadas el valor real x , y el valor aproximado x^* .

$$E_A = |x - x^*|$$

1.1 Error Relativo

Se define como el cociente entre el error absoluto y el valor real, expresado en terminos porcentuales

$$E_R = 100 \cdot \frac{E_A}{x} \cdot \frac{1}{100} = 100 \cdot \frac{E_A}{x} \%$$

1.2 Ejemplos:

Hallar el error absoluto y el valor relativo que se incurre cuando aproximamos $\frac{1}{3} = 0, \bar{3}$ mediante una cifra significativa.

$$x = \frac{1}{3} \text{ y } x^* = \frac{3}{10}$$

$$E_A = \left| \frac{1}{3} - \frac{3}{10} \right| = \left| \frac{10 - 9}{3 \cdot 10} \right| = \frac{1}{3 \cdot 10} = 10^{-1} \cdot x$$

$$E_R = \frac{E_A}{x} = \frac{10^{-1} \cdot x}{x} = 10^{-1} = 10\%$$

1.3 Ejemplos:

Hallar el error absoluto y el valor relativo que se incurre cuando aproximamos $\frac{1}{3} = 0, \bar{3}$ mediante una cifra significativa.

$$x = \frac{1}{3} \text{ y } x^* = \frac{33}{100}$$

$$E_A = \left| \frac{1}{3} - \frac{33}{100} \right| = \left| \frac{100 - 99}{3 \cdot 100} \right| = \frac{1}{3 \cdot 100} = 10^{-2} \cdot x$$

$$E_R = \frac{E_A}{x} = \frac{10^{-2} \cdot x}{x} = 10^{-2} = 1\%$$

1.4 contexto

$$x = 0.333333$$

$$x^* = 0.310\% = 10^{-1}$$

$$x^* = 0.331\% = 10^{-2}$$

$$x^* = 0.3330.1\% = 10^{-3}$$

De una forma generica, sería la tolerancia:

$$tol = 10^{-5}$$

Ejemplo:

$$pi = 3,1416$$

2 Polinomios. Notación 1. Taylor

$$P_n(x) = \sum_{i=0}^n c_i \cdot (x - x_0)^i$$

2.1 Ejemplos

$$P_n(x) = \sum_{i=0}^3 c_i \cdot (x - x_0)^i = c_0 \cdot (x - x_0)^0 + c_1 \cdot (x - x_0)^1 + c_2 \cdot (x - x_0)^2 + c_3 \cdot (x - x_0)^3$$

$$[-2, 4, 0.1, 5] \rightarrow -2 \cdot x^0 + 4 \cdot x^1 + \frac{1}{10} \cdot x^2 + 5 \cdot x^3$$

$$\left[\frac{2}{5}, 0, 0, 5, 0, -1\right], x_0 = \frac{1}{2}$$

3 Polinomios. Notación 2 . McClaurin

$$P_N(X) = \sum_{i=0}^n c_i \cdot x^i$$

3.1 Ejemplos

$$P_n(x) = \sum_{i=0}^3 c_i \cdot x^i = c_0 \cdot x^0 + c_1 \cdot x^1 + c_2 \cdot x^2 + c_3 \cdot x^3$$

$$[-2, 4, 0.1, -5] \rightarrow -2 \cdot x^0 + 4 \cdot x^1 + \frac{1}{10} \cdot x^2 + 5 \cdot x^3$$

$$\left[\frac{2}{5}, 0, 0, 5, 0, -1\right]$$

4 El Polinomio de Taylor

Búsqueda de Raíces

4.1 Método de bisección

4.2 Método de Regula falsi

4.3 Método de Newton

$$(x_0, f(x_0)) \rightarrow (x_1, 0) \quad (4.1)$$

$$m = \frac{0 - f(x_0)}{x_1 - x_0} \quad (4.2)$$

$$x_1 - x_0 = \frac{0 - f(x_0)}{m} \quad (4.3)$$

$$x_1 = x_0 + \frac{0 - f(x_0)}{m} \quad (4.4)$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (4.5)$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \quad (4.6)$$

$$\vdots \quad (4.7)$$

$$(4.8)$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad (4.9)$$

4.4 Método de Secante

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad (4.10)$$

$$x_{k+1} = x_k - \frac{f(x_k)}{\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}} \quad (4.11)$$

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1}) f(x_k)}{f(x_k) - f(x_{k-1})} \quad (4.12)$$

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})} \quad (4.13)$$

4.4.1 El Algoritmo

1. Asignar valores iniciales para x_0 y x_1
2. En la k -ésima iteración, calcular $x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$
3. El algoritmo para cuando el error absoluto entre dos iteraciones sucesivas $|x_{k+1} - x_k|$ (o el error relativo $|x_{k+1} - x_k|/x_k$) sea menor o igual a la tolerancia especificada.

5 Matriz, determinante, cofactores, adjunta

5.1 Inversa de una matriz

5.2 Ecuaciones matriciales

Cómo se despeja A en la ecuación $P \cdot A = L \cdot U$

$$P \cdot A = L \cdot U \rightarrow A = \frac{L \cdot U}{P} \quad (5.1)$$

$$(P^{-1} \cdot P) \cdot A = P^{-1} \cdot L \cdot U \quad (5.2)$$

$$I \cdot A = P^{-1} \cdot L \cdot U \quad (5.3)$$

$$A = P^{-1} \cdot L \cdot U \quad (5.4)$$

6 Sistemas de Ecuaciones lineales

Halla las soluciones del siguiente sistema:

$$8x_1 + 3x_2 - 3x_3 = 14$$

$$-2x_1 - 8x_2 + 5x_3 = 5$$

$$3x_1 + 5x_2 + 10x_3 = -8$$

$$A = \begin{pmatrix} 8 & 3 & -3 \\ -2 & -8 & 5 \\ 3 & 5 & 10 \end{pmatrix} \quad b = \begin{pmatrix} 14 \\ 5 \\ -8 \end{pmatrix}$$

7 Métodos Directos

7.1 Método Clásico. $Ax = b \rightarrow x = A^{-1} \cdot b$

7.2 Método de Gauss-Jordan. $rref(...)$

7.2.1 Operaciones elementales

- Escalamiento: Multiplicar una fila, o columna de la matriz por un escalar diferente de cero.
- Intercambio: Intercambiar dos filas o columnas de la matriz
- Sustitución: Cambiar una fila, o columna, por un escalar multiplicando a otra fila o columna

8 Factorización de Matrices

Estudiaremos tres métodos que factorizan una matriz cuadrada A , como producto de dos matrices triangulares L y U ; entonces se tiene que verificar que $P \cdot A = L \cdot U \rightarrow A = P^{-1} \cdot L \cdot U$

$$\begin{aligned}
 A \cdot x &= b \\
 (P^{-1} \cdot L \cdot U) \cdot x &= b \\
 (P^{-1} \cdot L) \cdot (U \cdot x) &= b \quad \text{sea } U \cdot x = y \quad \text{Entonces,} \\
 (P^{-1} \cdot L) \cdot y &= b \\
 y &= ((P^{-1} \cdot L))^{-1} * b \quad \text{Se resuelve mediante Sustituciones Progresivas} \\
 y &= ((L^{-1} \cdot (P^{-1})^{-1}) * b \\
 y &= ((L^{-1} \cdot P) * b \\
 Ux &= y \\
 x &= U^{-1} * y \rightarrow \text{Se resuelve mediante Sustituciones Regresivas}
 \end{aligned}$$

8.1 Doolittle-Crout. $P, L, U = \text{scipy.linalg.lu}(A)$

8.2 Cholesky. $\text{np.linalg.cholesky}(A @ A.T)$

Requiere que la matriz A sea simétrica y definida positiva. Cholesky hace $U = L^T$ entonces la factorización queda como $A = L * U = L * L^T$. El método es eficiente ya que sólo requiere $n(n+1)/2$ ecuaciones en lugar de las n^2 que se necesitan en los métodos de Doolittle y Crout. $A = L * L^T$. Dos matrices son iguales si, y solamente si, sus elementos correspondientes son iguales, esto es: $A[i, j] = L[i, :] \cdot U[:, j]$.

8.3 Sustituciones regresivas

$$\begin{aligned}
 Ux &= b, \\
 U &= \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & u_{nn} \end{pmatrix}
 \end{aligned}$$

8.4 Algoritmo

1. $x_n = \frac{b_n}{u_{nn}}$
2. Para $i = n - 1, \dots, 1$

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^n u_{ij} x_j \right).$$

9 Sustituciones progresivas

$$\begin{aligned}
 Lx &= b, \\
 L &= \begin{pmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{pmatrix}
 \end{aligned}$$

9.1 Algoritmo

1. $x_1 = \frac{b_1}{l_{11}}$
2. Para $i = 2, \dots, n$
 $x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right).$

10 Métodos Iterativos

10.1 Gauss-Jacobi

$$\begin{aligned}x_1 &= \frac{(14 - 3 \cdot x_2 + 3 \cdot x_3)}{8} \\x_2 &= \frac{(5 + 2 \cdot x_1 - 5 \cdot x_3)}{-8} \\x_3 &= \frac{(-8 - 3 \cdot x_1 - 5 \cdot x_2)}{10}\end{aligned}$$

$$x_0 = \begin{pmatrix} 1 \\ -1 \\ \frac{1}{2} \end{pmatrix} \quad x_1 = \begin{pmatrix} \frac{37}{16} \\ \frac{9}{16} \\ -\frac{3}{5} \end{pmatrix} \quad Ea = \begin{pmatrix} \frac{21}{7} \\ \frac{16}{11} \\ \frac{16}{10} \end{pmatrix}$$

10.1.1 Algoritmo

```
xk=[1, -1, 1/2]
sw=True
while sw==True:
    despejar las incógnitas y evaluarlas en xk
    Calcular xk1 reemplando xk en las incógnitas
    Ea=abs(xk1-xk)
    if Ea <tol:
        sw=False, return xk1, break
```

10.2 Gauss-Seidel

$$\begin{aligned}x_1 &= \frac{(14 - 3 \cdot x_2 + 3 \cdot x_3)}{8} \\x_2 &= \frac{(5 + 2 \cdot x_1 - 5 \cdot x_3)}{-8} \\x_3 &= \frac{(-8 - 3 \cdot x_1 - 5 \cdot x_2)}{10}\end{aligned}$$

$$x_0 = \begin{pmatrix} 1 \\ -1 \\ \frac{1}{2} \end{pmatrix}$$

$$x_1 = \begin{pmatrix} \frac{37}{16} \\ -\frac{64}{671} \\ -\frac{64}{640} \end{pmatrix}$$
$$Ea = \begin{pmatrix} \frac{21}{16} \\ \frac{7}{11} \\ \frac{16}{10} \end{pmatrix}$$

10.3 Convergencia de los algoritmos de Gauss Jacobi y Gauss Seidel

Para que los algoritmos convergan Se requiere que la matriz A sea estrictamente dominante en sentido Diagonal.

$$abs(A[i, i]) > \sum_{j=0, j \neq i}^{n-1} abs(A(i, j))$$

11 El Polinomio de interpolación

Problema: Dados un conjunto de puntos $n + 1$, en 2D encontrar el polinomio, de grado n , que pasa por cada uno de ellos.

$$P_n(x) = c_n x^n + c_{n-1} x^{n-1} + \dots + c_1 x + c_0, c_n \neq 0.$$

11.1 Soluciones

11.1.1 Solución Matricial (Vandermonde)

$$\begin{aligned} y &= P_n(x) \\ y_1 &= P_n(x_1) \\ y_2 &= P_n(x_2) \\ &\vdots \\ y_n &= P_n(x_n) \\ y_{n+1} &= P_n(x_{n+1}) \end{aligned}$$

Reemplazamos

$$\begin{aligned} y_1 &= c_n x_1^n + c_{n-1} x_1^{n-1} + \dots + c_1 x_1 + c_0 \\ y_2 &= c_n x_2^n + c_{n-1} x_2^{n-1} + \dots + c_1 x_2 + c_0 \\ &\vdots \\ y_n &= c_n x_n^n + c_{n-1} x_n^{n-1} + \dots + c_1 x_n + c_0 \\ y_{n+1} &= c_n x_{n+1}^n + c_{n-1} x_{n+1}^{n-1} + \dots + c_1 x_{n+1} + c_0 \end{aligned}$$

Recordemos que este sistema se puede escribir en forma matricial. $A \cdot x = b$, donde A es la matriz de coeficientes de las incógnitas. x es el vector columna de las incógnitas y b es el vector columna de los términos independientes.

$$A = X = \begin{pmatrix} x_1^n & x_1^{n-1} & x_1^{n-2} & \dots & x_1^0 \\ x_2^n & x_2^{n-1} & x_2^{n-2} & \dots & x_2^0 \\ x_3^n & x_3^{n-1} & x_3^{n-2} & \dots & x_3^0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \dots & x_n^0 \\ x_{n+1}^n & x_{n+1}^{n-1} & x_{n+1}^{n-2} & \dots & x_{n+1}^0 \end{pmatrix}$$

$$x_0 = \begin{pmatrix} 1 \\ -1 \\ \frac{1}{2} \end{pmatrix} \quad x_1 = \begin{pmatrix} \frac{37}{16} \\ -\frac{9}{16} \\ -\frac{3}{5} \end{pmatrix} \quad Ea = \begin{pmatrix} \frac{21}{7} \\ \frac{16}{11} \\ \frac{1}{10} \end{pmatrix}$$

$$A \cdot x = b \rightarrow X \cdot c = y \rightarrow c = \text{inv}(X) \cdot y; \text{vander}(x) \cdot c = y \rightarrow c = \text{inv}(\text{vander}(x)) \cdot y;$$

12 El Polinomio de Newton

13 El Polinomio de Lagrange

14 Integración Numérica

$$\int_a^b f(x)$$

15 Regla generalizada del Trapecio

$$\int_a^b f(x) \cdot dx = \frac{f(x_0) + \sum_{i=1}^{n-1} f(x_i)}{2n}$$

16 Regla generalizada de Simpson

$$\int_{a=x_0}^{b=x_{2n}} f(x) \cdot dx = h \cdot \frac{f(x_0) + f(x_{2n}) + 4 \cdot \sum_{i=1}^n f(x_{2 \cdot i-1}) + 2 \cdot \sum_{i=1}^{n-1} f(x_{2 \cdot i})}{3}$$

17 Ajuste de Curvas

Un problema resuelto en análisis de datos es encontrar la recta que mejor se ajusta a una nube de puntos. Lo anterior no significa que este sea el mejor ajuste posible!

$$Y = m \cdot X + b$$

Los parámetros m y b se hallan mediante el método de mínimos cuadrados (Regresión Lineal).

18 El proceso de "Linealización"

Consiste en transformar el modelo propuesto, (potencial, exponencial, logístico, polinómico y otros), a uno de la forma

$$F(xoy) = f(\beta_0, \beta_1) \cdot G(xoy) + g(\beta_0, \beta_1)$$

de tal manera que: $f(\beta_0, \beta_1) \equiv m$ y $g(\beta_0, \beta_1) \equiv b$ para encontrar, en forma indirecta, el valor de los parámetros del modelo propuesto (β_0, β_1) en función de los parámetros del modelo Lineal (m, b)

18.1 El Ajuste Potencial.

$$y = A \cdot x^M$$

$$\log(y) = \log(A \cdot x^M)$$

$$\log(y) = M \cdot \log(x) + \log(A)$$

Equivalencias

$$Y = m \cdot X + b$$

$$Y \equiv \log(y)$$

$$m \equiv M$$

$$X \equiv \log(x)$$

$$b \equiv \log(A)$$

18.2 El Ajuste racional

$$y = \frac{D}{x + C}$$

$$D = x \cdot y + C \cdot y$$

$$y = -\frac{1}{C} \cdot (x \cdot y) + \frac{D}{C}$$

→ Equivalencias.

$$Y = m \cdot X + b$$

$$Y \equiv y$$

$$m \equiv -\frac{1}{C}$$

$$X \equiv x \cdot y$$

$$b \equiv \frac{D}{C}$$

18.3 El Ajuste Logístico

$$y = \frac{L}{1 + C \cdot e^{A \cdot x}}$$
$$L = y(1 + C \cdot e^{A \cdot x})$$
$$\frac{L}{y} = 1 + C \cdot e^{A \cdot x}$$
$$\frac{L}{y} - 1 = C \cdot e^{A \cdot x}$$
$$\log\left(\frac{L}{y} - 1\right) = \log(C \cdot e^{A \cdot x})$$

$$\log\left(\frac{L}{y} - 1\right) = A \cdot x + \log(C)$$

→ Equivalencias.

$$Y = m \cdot X + b$$
$$Y \equiv \ln\left(\frac{L}{y} - 1\right)$$
$$X \equiv x$$
$$m \equiv A$$
$$b \equiv \log(C)$$

18.4 El Ajuste recíproco de la recta

$$y = \frac{1}{A \cdot x + B}$$
$$\frac{1}{y} = A \cdot x + B$$

→ Equivalencias.

$$Y = m \cdot X + b$$
$$Y \equiv \frac{1}{y}$$
$$X \equiv x$$
$$m \equiv A$$
$$b \equiv B$$

18.5 El Ajuste exponencial

$$y = C \cdot e^{A \cdot x}$$

18.6 Otros Ajustes

HONESTIDAD

- El proyecto se desarrollará en equipos de dos estudiantes.
- La entrega está programada para el Sábado 08 de julio de 2023, hasta las 11:59 p.m.
- Cada entrega parcial tiene que incluir:
 1. Recuerde incluir solamente a los integrantes del equipo que aportaron a la solución de los problemas.
 2. Entregar un sólo archivo .zip. Éste debe estar estructurado de la siguiente manera:
 - Informe en Látex + Desarrollo de los problemas en Python©.
 3. Fotografía de los miembros del equipo que trabajaron.
 4. Vídeo con la solución de dos problemas. Duración aproximada entre 5 y diez minutos por estudiante. (Utilice como referencia el que se muestra en este enlace:
<https://www.youtube.com/watch?v=ox09Jko1ErM/>.
Cada estudiante selecciona un problema y presenta la solución, previa aprobación del profesor.
 5. Todas las componentes de la evaluación son de carácter obligatorio

RÚBRICA

Las soluciones a los problemas aquí propuestos deben estar detalladas.

Componente		Porcentaje
Problemas	Todos los problemas tienen el mismo peso	75 %
Informe	Debe estar hecho con normas IEEE, APA o EasyChair u otro.	20 %
Autoevaluación	Individual. Cada miembro del equipo debe realizar su propia autoevaluación.	5 %

18.7 Problema 01: Newton-Raphson(Alfonso Mancilla-El Manci)

A partir de un análisis gráfico determine el número de soluciones de los sistemas mostrados a continuación.

18.8 Sistemas de Ecuaciones 01

$$f(x, y) = 7x^3 - 10x - y - 1 = 0$$

$$g(x, y) = 8y^3 - 11y + x - 1 = 0$$

Solución: Sistemas de Ecuaciones 01

Código

```
\begin{minted}[breaklines]{python}
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

def f(x, y):
    return (7*x**3) - (10*x) - (y) - (1)

def g(x, y):
    return (8*y**3) - (11*y) + (x) - (1)

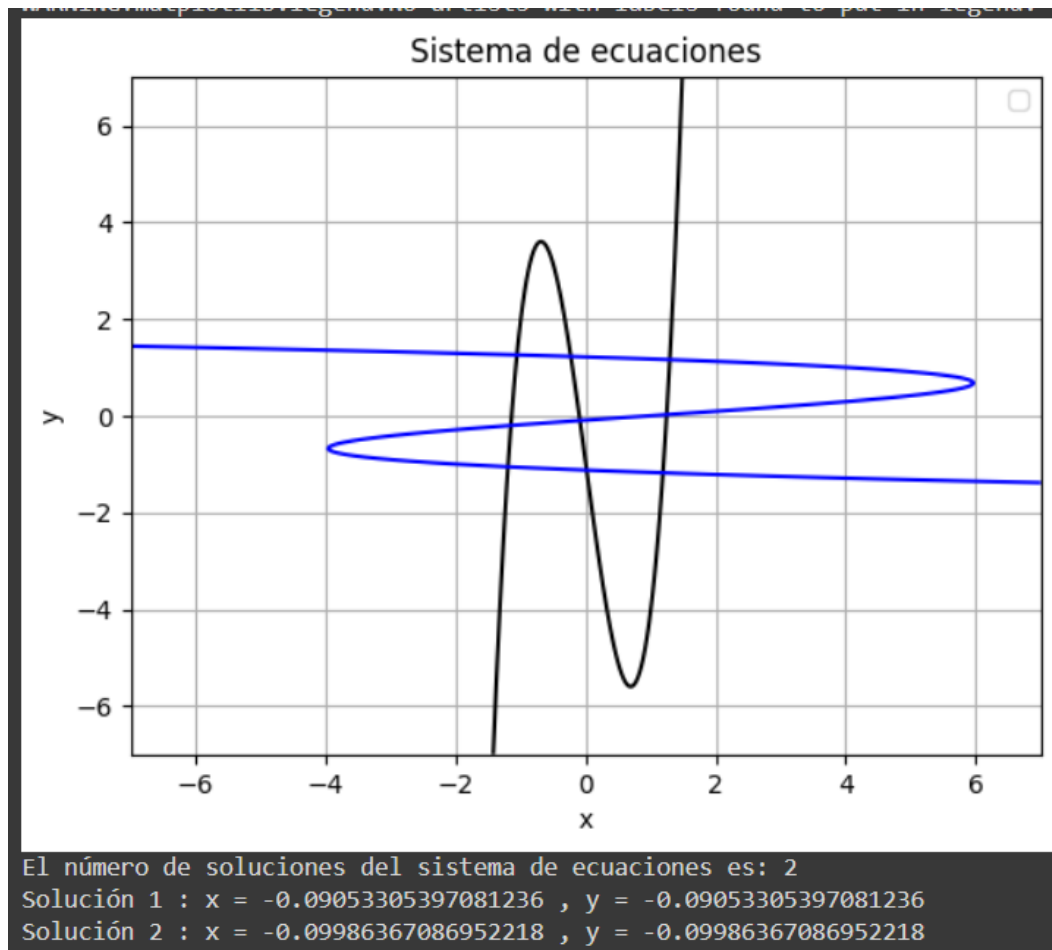
x = np.linspace(-7, 7, 500)
y = np.linspace(-7, 7, 500)
X, Y = np.meshgrid(x, y)
Z1 = f(X, Y)
Z2 = g(X, Y)

plt.contour(X, Y, Z1, levels=[0], colors='black')
plt.contour(X, Y, Z2, levels=[0], colors='blue')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Sistema de ecuaciones')
plt.legend()
plt.grid(True)
plt.show()

def ecuaciones(vars):
    x, y = vars
    return [f(x, y), g(x, y)]

soluciones = fsolve(ecuaciones, (0, 0))
numero_de_soluciones = len(soluciones)
print("El número de soluciones del sistema de ecuaciones es:", numero_de_soluciones)
for i, solucion in enumerate(soluciones):
    print("Solución", i+1, ": x =", solucion, ", y =", solucion)
\end{minted}
```

Resultado



18.9 Sistemas de Ecuaciones 02

$$f(x, y) = x^2 + x \cdot y^3 - 9$$

$$g(x, y) = 3x^2 \cdot y - y^3 - 4$$

Posteriormente Encuentre las soluciones al mediante el algoritmo de Newton-Raphson. Haga una gráfica encerrando en un círculo cada una de las soluciones obtenidas a partir de una punto cualquiera en las vecindades de las soluciones.

El archivo .ipynb tiene que contener el cálculo de TODAS las soluciones.

Solución: Sistemas de Ecuaciones 02

Código

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

def f(x, y):
```

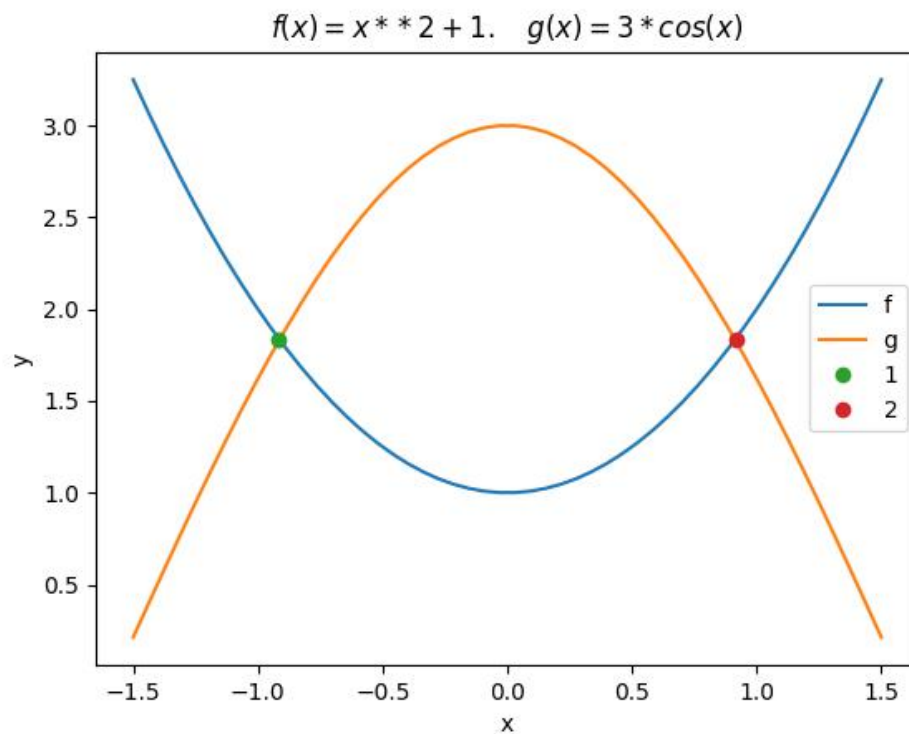



Figure 2: Ejemplo de la solución que tiene que presentar

```

    return x**2 + x * y**3 - 9

def g(x, y):
    return 3 * x**2 * y - y**3 - 4

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

X, Y = np.meshgrid(x, y)

Z_f = f(X, Y)
Z_g = g(X, Y)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.contour(X, Y, Z_f, levels=[0], colors='red')
plt.title('$f(x, y) = x^2 + x \cdot y^3 - 9$')

plt.subplot(1, 2, 2)
plt.contour(X, Y, Z_g, levels=[0], colors='blue')
plt.title('$g(x, y) = 3x^2 \cdot y - y^3 - 4$')

plt.tight_layout()

```

```
plt.show()

x0_1 = 2.0
y0_1 = 2.0

x0_2 = -2.0
y0_2 = -2.0

solution_1 = fsolve(ecuaciones, (x0_1, y0_1))
solution_2 = fsolve(ecuaciones, (x0_2, y0_2))

if len(solution_1) == 2 and len(solution_2) == 2:
    plt.figure(figsize=(6, 6))
    plt.contour(X, Y, Z_f, levels=[0], colors='red')
    plt.contour(X, Y, Z_g, levels=[0], colors='blue')
    plt.scatter(solution_1[0], solution_1[1], color='green', marker='o', s=100)
    plt.scatter(solution_2[0], solution_2[1], color='green', marker='o', s=100)
    plt.title('Soluciones encontradas')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.xlim(-5, 5)
    plt.ylim(-5, 5)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()

def ecuaciones(vars):
    x, y = vars
    return [f(x, y), g(x, y)]

soluciones = fsolve(ecuaciones, (0, 0))
numero_de_soluciones = len(soluciones)
print("\nEl número de soluciones del sistema de ecuaciones es:", numero_de_soluciones)
for i, solucion in enumerate(soluciones):
    print("Solución", i+1, ": x =", solucion, ", y =", solucion)
```

Resultado

El número de soluciones del sistema de ecuaciones es: 2
Solución 1 : x = 7.712242731443736e-05 , y = 7.712242731443736e-05
Solución 2 : x = -0.00010133814023443048 , y = -0.00010133814023443048

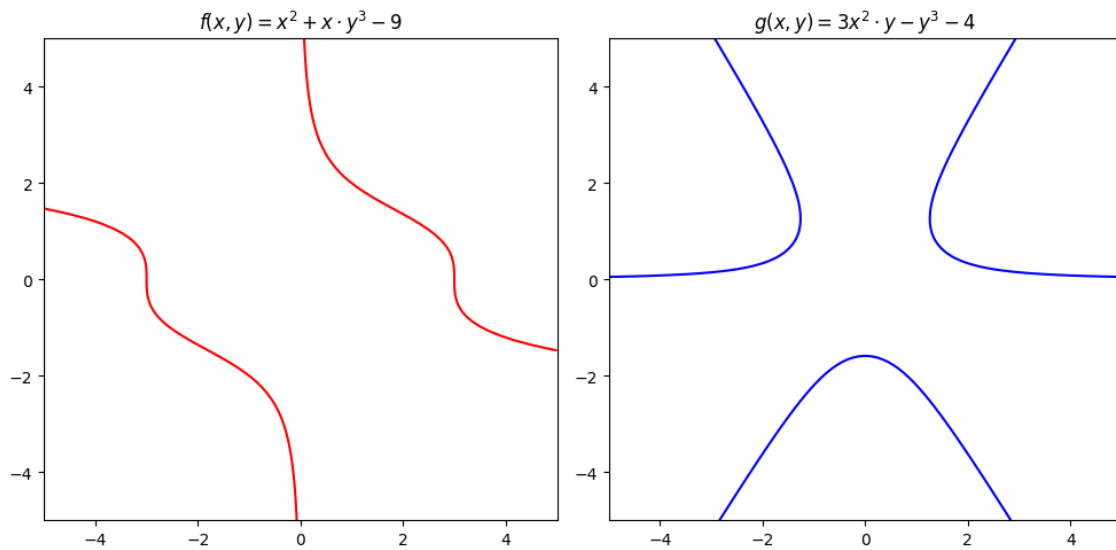


Figure 3: Grafica de las funciones $f(x)$ y $g(x)$

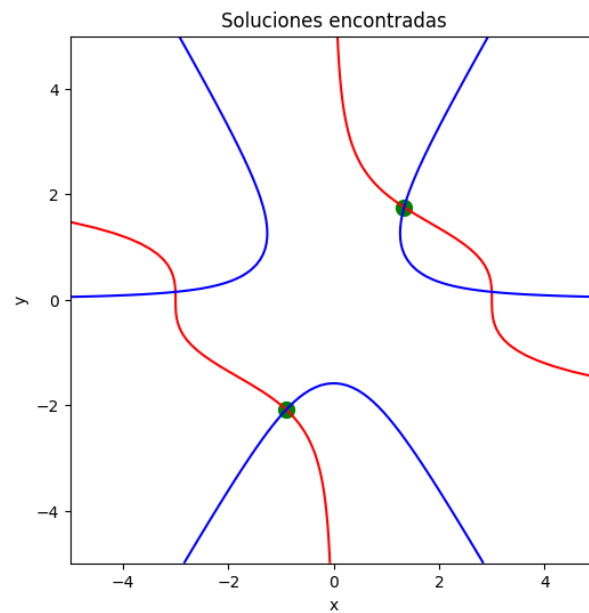


Figure 4: Puntos de las soluciones

Problema 02: Búsqueda de Raíces

Para cada uno de los métodos de búsqueda de raíces estudiados, usted debe diseñar e implementar tres gráficos que ilustren la acción del algoritmo cuando $k = 0$ (condiciones iniciales) y en las iteraciones $k = 1$ y $k = 2$. Usted tiene la libertad para escoger la función en cada caso.

$$c_k = \frac{a_k + b_k}{2}$$

$$c_k = b_k - \frac{f(b_k) \cdot (b_k - a_k)}{f(b_k) - f(a_k)}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Ayuda: A continuación encontrará la solución del caso $k = 0$ y $k = 1$ para el método de la falsa posición. Sólo le faltaría el caso $k = 2$. Regula Falsi

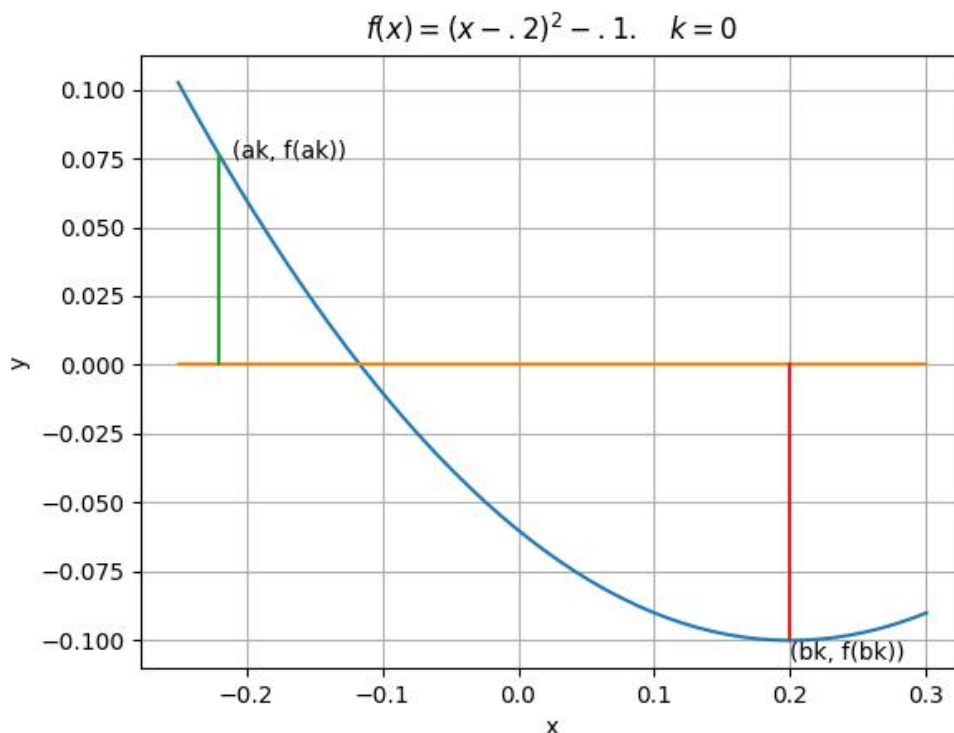


Figure 5: Ejemplo de la solución que tiene que presentar. *regula_falsi(f, a0, b0, tol)*

Solución: Problema 02: Búsqueda de Raíces

Código

-Metodo Regula falsi

```
def regula_falsi(f, ak, bk, tol):
    print('\nMétodo de Regula Falsi\n')
```

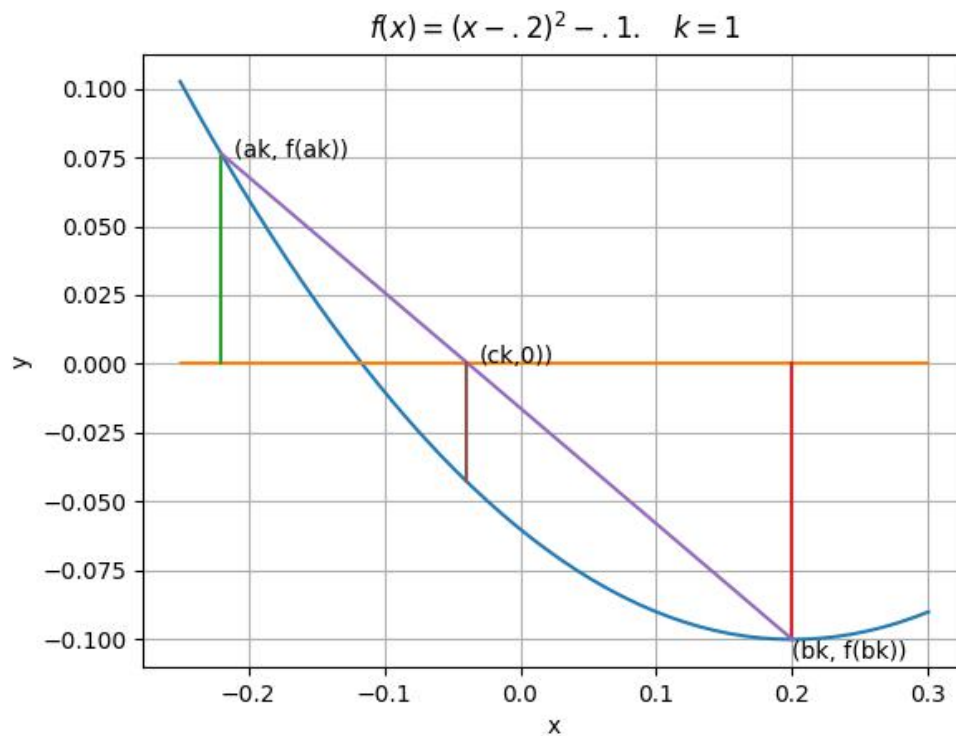


Figure 6: Ejemplo de la solución que tiene que presentar

```
print('k\t ak\t bk\t ck\tEa\n')

Ea = abs(bk - ak)
k = 0
c_ant = 0

while k <= 2:

    ck = bk - (f(bk) * (bk - ak)) / (f(bk) - f(ak))
    Ea = abs(ck - c_ant)

    print("k =", k, "\t ak =", ak, "\t bk =", bk, "\t ck =", ck, "\t Ea =", Ea)

    x = np.linspace(ak, bk, 1000)
    plt.grid()
    plt.plot(x, f(x), label='f(x)')

    plt.plot([ak, bk], [0, 0])
    plt.plot([ak, ak], [0, f(ak)], label='(ak,f(ak))')
    plt.plot([bk, bk], [f(bk), 0], label='(bk,f(bk))')
    plt.plot([ck, ck], [0, f(ck)], label='(ck,f(ck))')
    plt.plot([ak, bk], [f(ak), f(bk)], color='orange')
```

```
plt.text(ak, f(ak), '(ak,f(ak))')
plt.text(bk, f(bk), '(bk,f(bk))')
plt.text(ck, f(ck), '(ck,f(ck))')
plt.legend()
plt.title(r'$f(x) = ((x-0.2)^2)-0.1, k={}$'.format(k))
plt.suptitle('Método de Regula Falsi')
plt.show()

if f(ck) == 0:
    return ck
else:
    if f(ak) * f(ck) > 0:
        ak = ck
    else:
        bk = ck
    k += 1
    c_ant = ck
print('El valor de la raíz es =', ck)
return ck
```

Graficación

```
x = sp.symbols("x")
f = ((x-0.2)**2)-0.1
f = sp.lambdify(x, f)
x = np.linspace(-0.3, 0.3, 1000)

plt.plot(x, f(x))
plt.plot([-0.3, 0.3], [0, 0])
plt.text(-0.23,f(-0.25),'(ak,f(ak))')
plt.plot([-0.25, -0.25], [0, f(-0.25)], label='(ak, f(ak))')
plt.text(0.2,f(0.3),'(bk,f(bk))')
plt.plot([0.2, 0.2], [0, f(0.2)], label='(bk, f(bk))')
plt.title("$f(x) = ((x-0.2)^2)-0.1, k = 0$")
plt.suptitle("Método de Regula Falsi")
plt.ylim(-0.12, 0.12)
plt.grid()
plt.legend()
plt.show()
```

Resultado

```
def f(x):
    return ((x - 0.2) ** 2) - 0.1
regula_falsi(f,-0.25,0.2,10**-3)
```

-Metodo Bisección

```
def biseccion(f, ak, bk, tol):
    print('\nMÉTODO DE BISECCIÓN\n')
    print('k\t ak\t bk\t ck\tEa\n')
    Ea=abs(bk-ak)
```

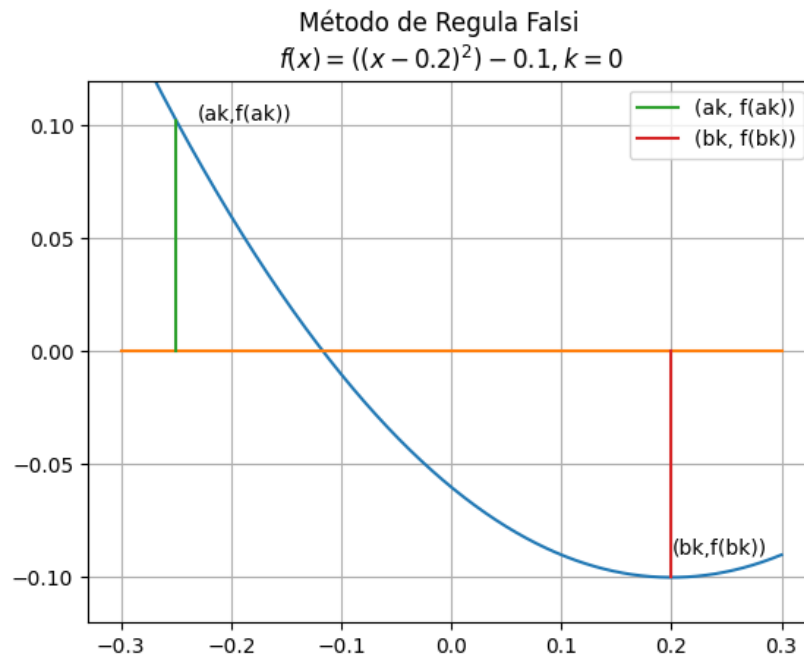


Figure 7: Grafica Modelo

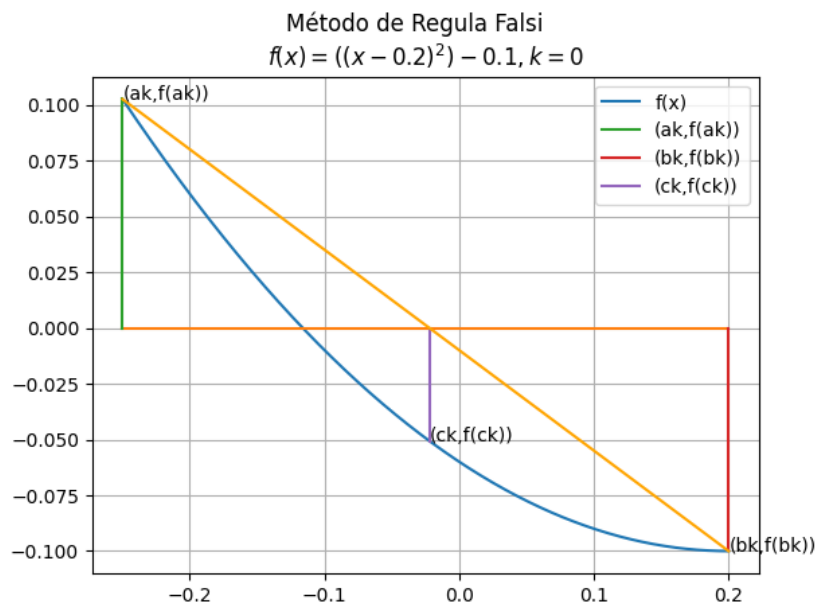


Figure 8: $k = 0$ $a_k = -0.25$ $b_k = 0.2$ $c_k = -0.02222222222222227$ $E_a = 0.02222222222222227$

```
k=0
while Ea>=tol:
    ck=(ak+bk)/2
    print("k = ", k, "\t ak = ", ak, "\t bk = ", bk, "\t ck = ", ck, "\t Ea = ", Ea)
```

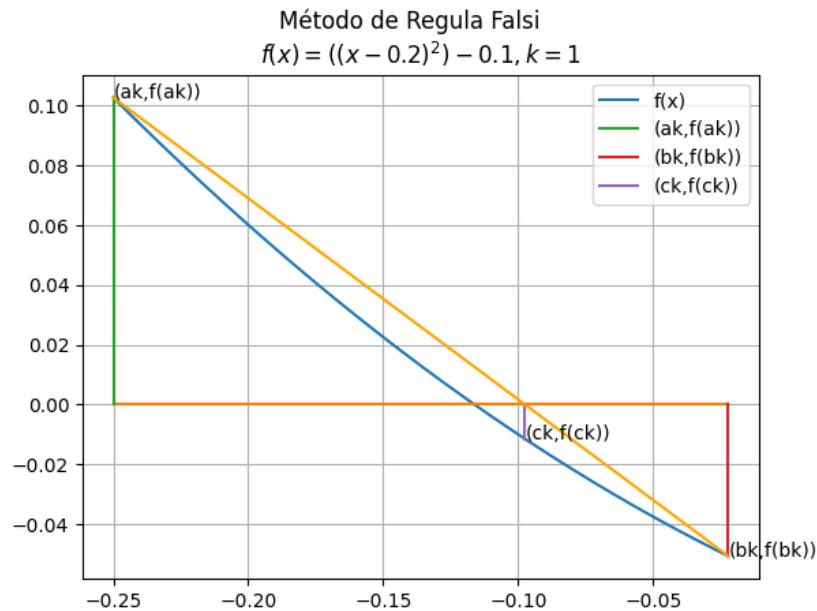


Figure 9: $k = 1$ $a_k = -0.25$ $b_k = -0.02222222222222227$ $c_k = -0.09752066115702479$ $E_a = 0.07529843893480256$

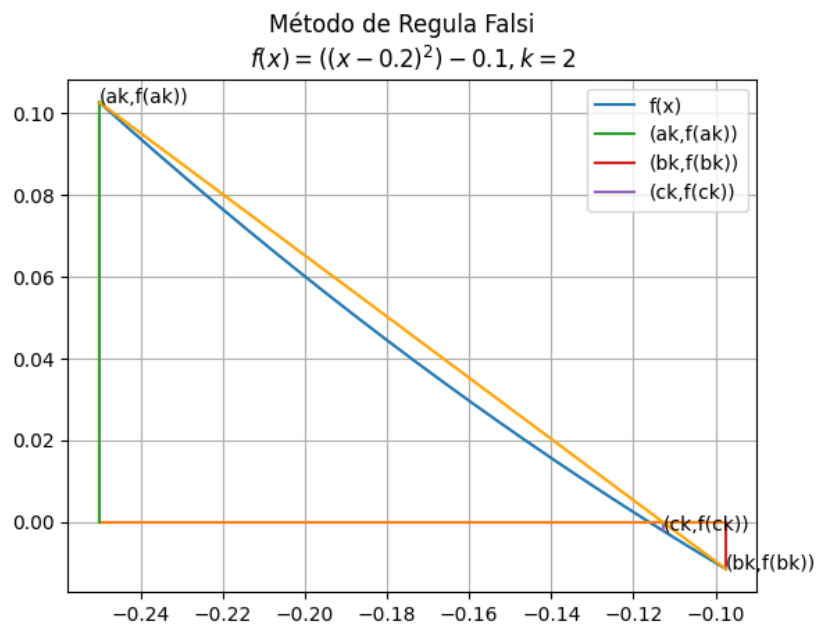


Figure 10: $k = 2$ $a_k = -0.25$ $b_k = -0.09752066115702479$ $c_k = -0.1128800442233278$ $E_a = 0.015359383066303009$

```
x = np.linspace(ak, bk, 1000)
plt.grid()
plt.plot(x, f(x), label='f(x)')
```



```
plt.plot([ak, bk], [0, 0])
plt.plot([ak, ak], [0, f(ak)], label = '(ak,f(ak))')
plt.plot([bk, bk], [f(bk), 0], label = '(bk,f(bk))')
plt.plot([ck, ck], [0, f(ck)], label= '(ck,f(ck))')
plt.plot([ak, bk], [f(ak), f(bk)], color='orange')

plt.text(ak,f(ak),'(ak,f(ak))')
plt.text(bk,f(bk),'(bk,f(bk))')
plt.text(ck,f(ck),'(ck,f(ck))')
plt.legend()
plt.title(r'$f(x) = ((x-0.2)^2)-0.1, k={}\$'.format(k))
plt.suptitle("Método de Bisección")
plt.show()

if f(ck)==0:
    return ck
else:
    if f(ak)*f(ck)>0:
        ak=ck
    else:
        bk=ck
    Ea=abs(bk-ak)
    k+=1
print("El valor de la raíz es = ", ck)
return ck
```

Graficación:

```
def f(x):
    return ((x - 0.2) ** 2) - 0.1
biseccion(f,-0.25,0.2,10**-3)
```

-Metodo Secante

```
def Secante(f, xk, xk_1, tol, maxiter):
    print('\nMÉTODO DE LA SECANTE\n')

    for k in range(maxiter):
        xk_1 = xk - ((f(xk) * (xk - xk_1)) / (f(xk) - f(xk_1)))
        Ea = abs(xk_1 - xk)
        print("k= ", k, "\t\t xk=", xk, "\t\t Ea= ", Ea)

        x = np.linspace(xk_1, xk, 1000)
        plt.grid()
        plt.plot(x, f(x), label='f(x)')

        plt.plot([xk_1, xk], [0, 0])
        plt.plot([xk_1, xk_1], [0, f(xk_1)], label='(xk_1, f(xk_1))')
        plt.plot([xk, xk], [f(xk), 0], label='(xk, f(xk))')
        plt.plot([xk_1, xk_1], [0, f(xk_1)], label='(xk_1, f(xk_1))')
        plt.plot([xk_1, xk], [f(xk_1), f(xk)], color='orange')
```

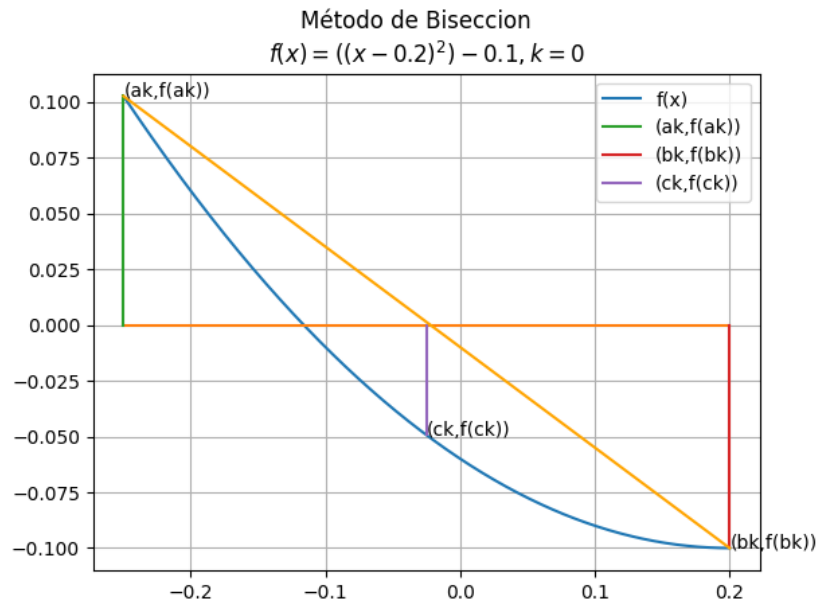


Figure 11: $k = 0$ $a_k = -0.25$ $b_k = 0.2$ $c_k = -0.02499999999999994$ $E_a = 0.45$

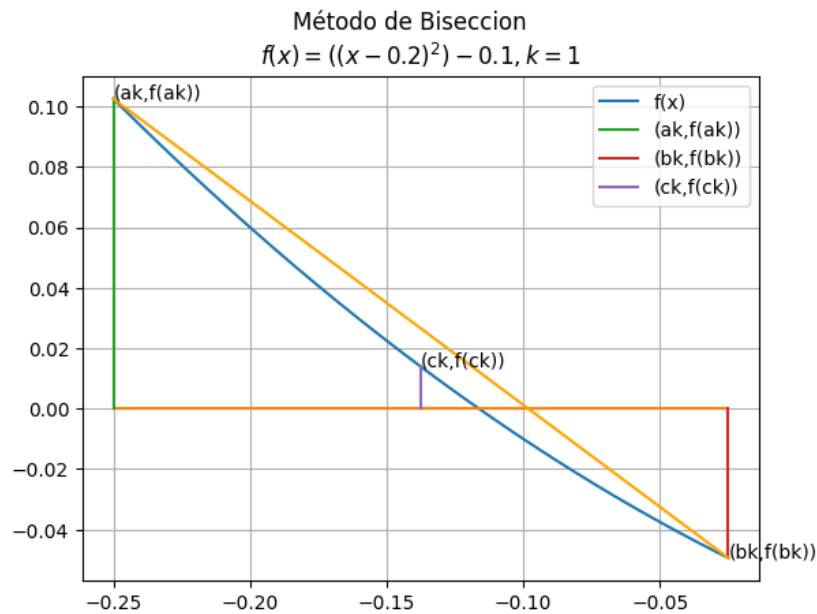


Figure 12: $k = 1$ $a_k = -0.25$ $b_k = -0.02499999999999994$ $c_k = -0.1375$ $E_a = 0.225$

```
plt.text(xk__1, f(xk__1), '(xk__1, f(xk__1))')
plt.text(xk, f(xk), '(xk, f(xk))')
plt.text(xk_1, f(xk_1), '(xk_1, f(xk_1))')
plt.legend()
plt.title(r'$f(x) = ((x-0.2)^2)-0.1, k={}$'.format(k))
plt.suptitle("Método de la Secante")
```

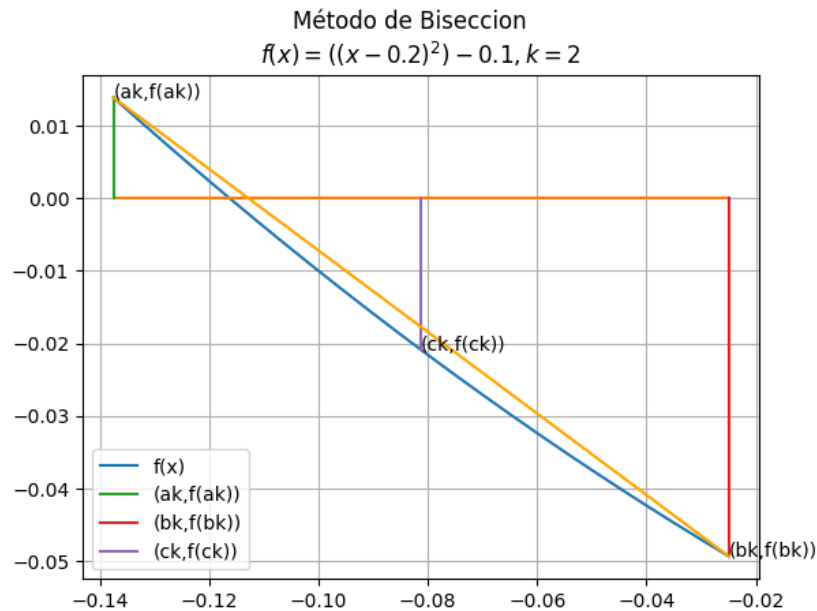


Figure 13: $k = 2$ $a_k = -0.1375$ $b_k = -0.024999999999999994$ $c_k = -0.08125$ $E_a = 0.11250000000000002$

```
plt.show()

if Ea < tol:
    print("El valor de la raíz es =", xk_1)
    return xk_1

xk__1 = xk
xk = xk_1

return [None]
```

Graficación:

```
def f(x):
    return ((x - 0.2) ** 2) - 0.1
Secante(f, -0.25, 0.2, 1e-3, 5)
```

-Metodo de newton

```
def newton(f, df, xk, tol, maxiter):
    print('\nMÉTODO DE NEWTON\n')

    for k in range(maxiter):
        xk1 = xk - f(xk) / df(xk)
        Ea = abs(xk1 - xk)
        print(k, "\t\t", xk, "\t\t", Ea)
```

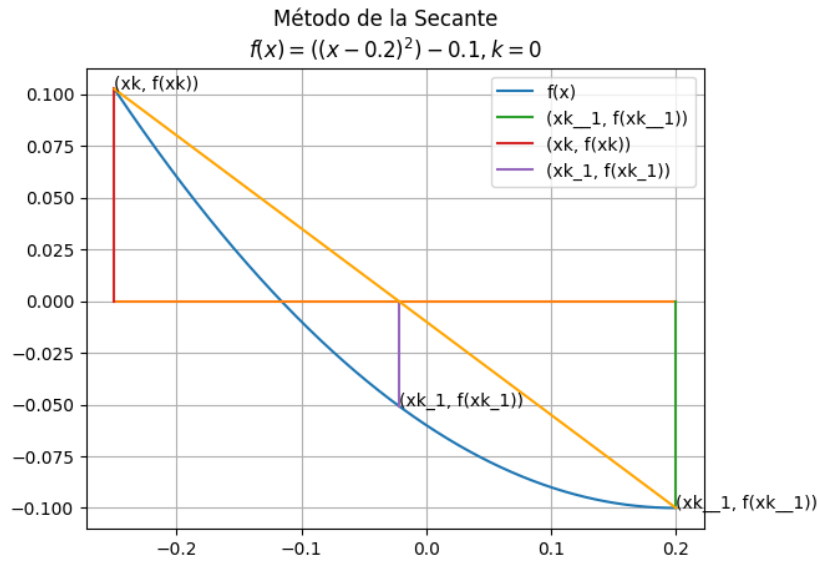


Figure 14: $k = 0$ $x_k = -0.25$ $E_a = 0.2277777777777778$

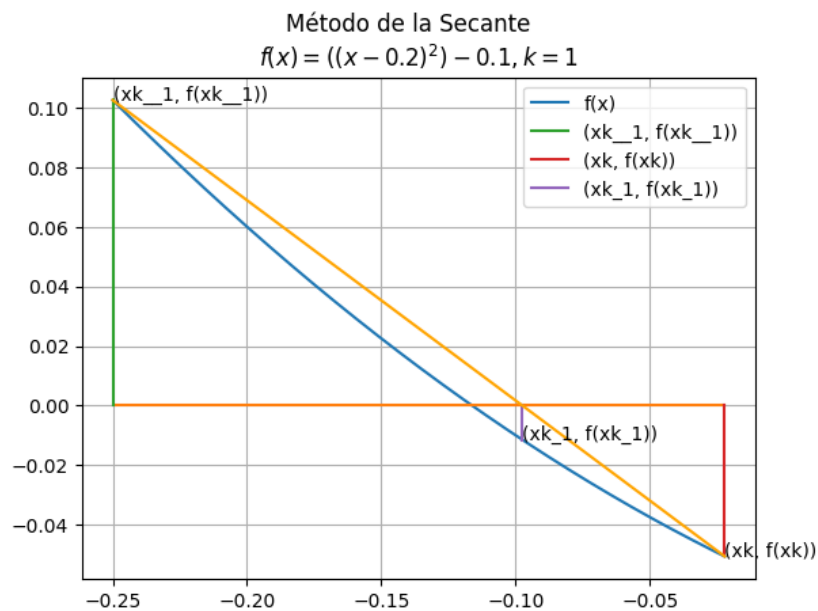


Figure 15: $k = 1$ $x_k = -0.022222222222222$ $E_a = 0.07529843893480258$

```
x = np.linspace(xk - 1, xk + 1, 1000)
plt.grid()
plt.plot(x, f(x), label='f(x)')
plt.axhline(y=0, color = "orange")
plt.axvline(x=xk,color='green', label='xk')
plt.axvline(x=xk1,color = "red", label='xk1')
plt.legend()
```

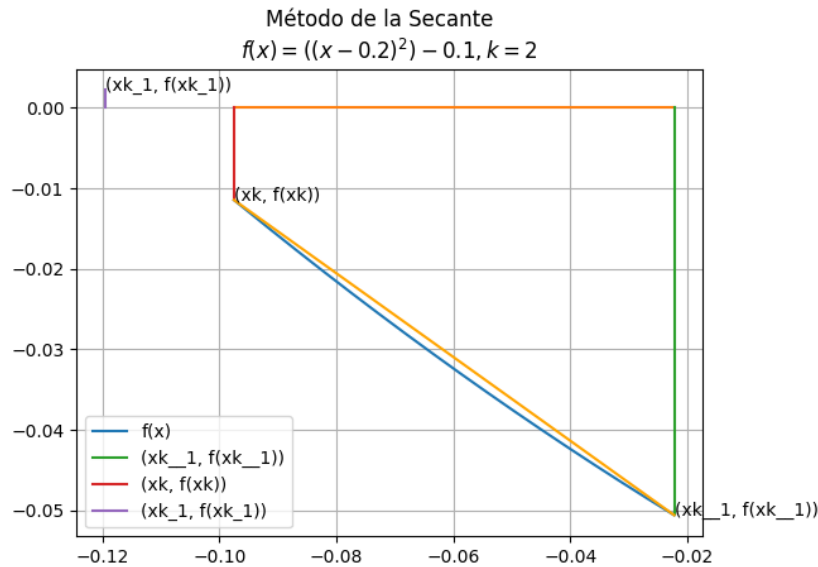


Figure 16: $k = 2$ $x_k = -0.09752066115702478$ $E_a = 0.022090646263469904$

```
plt.title(r'$f(x) = ((x-0.2)^2)-0.1, k={}$'.format(k))
plt.suptitle("Método de Newton")
plt.show()

if Ea < tol:
    print("El valor de la raíz es =", xk1)
    return xk1

xk = xk1

return [None]
```

Graficación:

```
x= sp.symbols('x')
f = ((x - 0.2) ** 2) - 0.1
df = sp.diff(f, x)
f= sp.lambdify(x, f)
df= sp.lambdify(x, df)
newton(f, df, -2, 10**-3, maxiter=100)
```

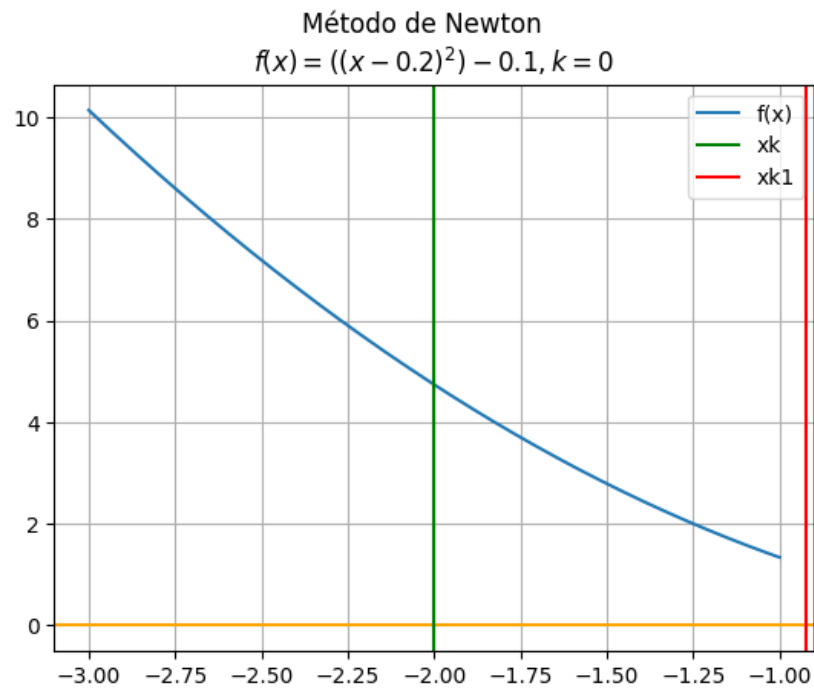


Figure 17: $k = 0$ $x_k = -2$ $E_a = 1.07727272727274$

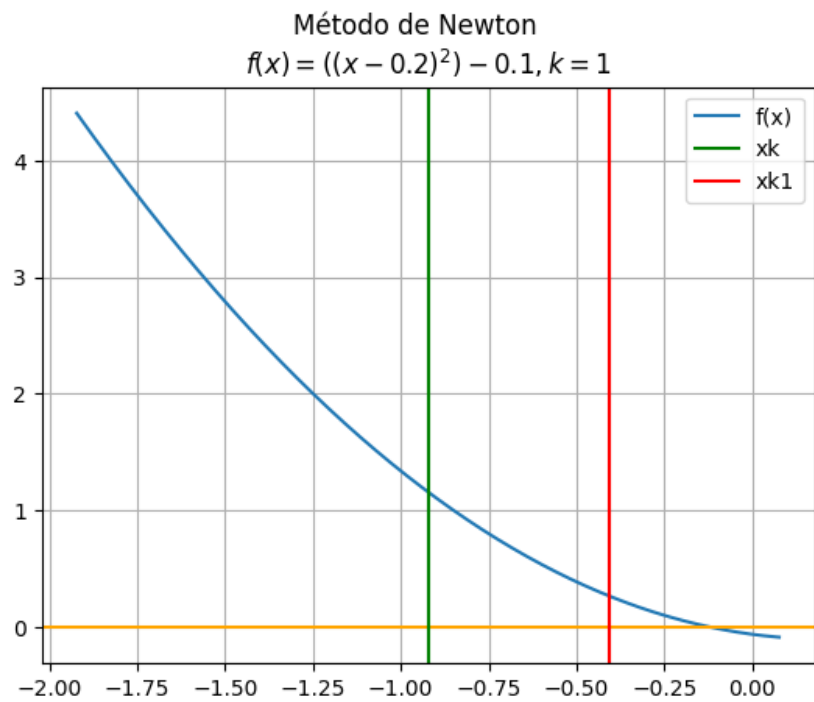


Figure 18: $k = 1$ $x_k = -0.92272727272726$ $E_a = 0.5168292234081706$

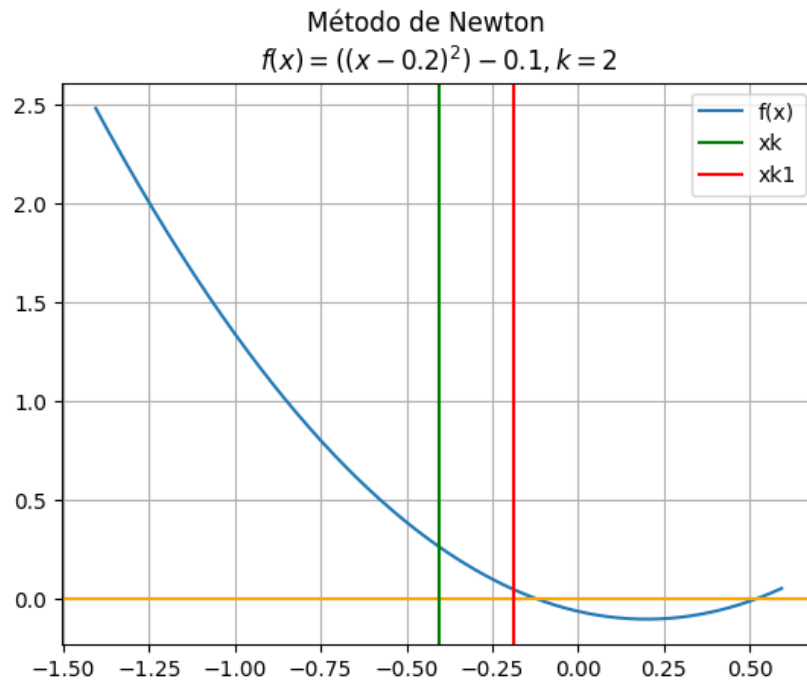


Figure 19: $k = 2$ $x_k = -0.405898049319102$ $E_a = 0.2204268906863699$

Problema 03: Ejercicio acordado del taller de Algebra de matrices

Solución: Problema 03: Ejercicio acordado del taller de Algebra de matrices

1.1 Solución ecuaciones de forma matricial

```
L = np.array([[1, 0, 0, 0, 0],
              [1, 1, 0, 0, 0],
              [1, 2, 1, 0, 0],
              [1, 4, 6, 4, 1]])

b = np.random.randint(-5, 10, size=(4, 1)) #Aquí genero los valores aleatorios

A_0 = np.dot(L, L.T) #Aquí obtengo lo que es A aplicando la formula donde A = L * L^T
x = np.linalg.solve(A_0, b)

print("Matriz A: \n", A_0)
print("Matriz b: \n", b)
print("Matriz x: \n", x)
```

Matriz **A**:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 5 \\ 1 & 3 & 6 & 15 \\ 1 & 5 & 15 & 70 \end{pmatrix}$$

Matriz **b**:

$$\begin{pmatrix} 3 \\ 8 \\ 0 \\ 6 \end{pmatrix}$$

Matriz **x**:

$$\begin{pmatrix} -25.76470588 \\ 59.70588235 \\ -34.52941176 \\ 3.58823529 \end{pmatrix}$$

Definiendo **b**:

```
b = np.array([3, 8, 0, 6])
```

Punto 1.1:

```
A = np.array([[ 1, 1, 1, 1],[ 1, 2, 3, 5],[ 1, 3, 6, 15],[ 1, 5, 15, 70]])
b = np.array([3, 8, 0, 6])

x1,x2,x3,x4 = sp.symbols('x1,x2,x3,x4')

x=sp.Matrix([x1,x2,x3,x4])
b=sp.Matrix([3, 8, 0, 6])
A = sp.Matrix([[ 1, 1, 1, 1],[ 1, 2, 3, 5],[ 1, 3, 6, 15],[ 1, 5, 15, 70]])
print(A*x+b,'\n\n')
```


Sistema de ecuaciones:

$$\begin{pmatrix} x_1 + x_2 + x_3 + x_4 + 3 \\ x_1 + 2x_2 + 3x_3 + 5x_4 + 8 \\ x_1 + 3x_2 + 6x_3 + 15x_4 \\ x_1 + 5x_2 + 15x_3 + 70x_4 + 6 \end{pmatrix}$$

Punto 1.2: Método clásico

```
A = np.array([[ 1, 1, 1, 1],[ 1, 2, 3, 5],[ 1, 3, 6, 15],[ 1, 5, 15, 70]])  
b = np.array([3, 8, 0, 6])
```

```
x = np.linalg.inv(A)@b  
print('Los valores de x son: ', x)
```

Solucion:

Los valores de x son: $[-25.76470588 \quad 59.70588235 \quad -34.52941176 \quad 3.58823529]$

Punto 1.3 Método de Gauss Jordan

```
A = np.matrix([[ 1, 1, 1, 1, 3],  
               [ 1, 2, 3, 5, 8],  
               [ 1, 3, 6, 15, 0],  
               [ 1, 5, 15, 70, 6]])  
print("Matriz Aumentada \n", A)
```

```
Aumentada = sp.Matrix(A).rref()  
print('Método de Gauss-Jordan [I x] \n', Aumentada)
```

Solución:

Matriz Aumentada:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 3 \\ 1 & 2 & 3 & 5 & 8 \\ 1 & 3 & 6 & 15 & 0 \\ 1 & 5 & 15 & 70 & 6 \end{pmatrix}$$

Método de Gauss-Jordan $[I-x]$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & -\frac{438}{17} \\ 0 & 1 & 0 & 0 & \frac{1015}{17} \\ 0 & 0 & 1 & 0 & -\frac{587}{17} \\ 0 & 0 & 0 & 1 & \frac{61}{17} \end{pmatrix}, (0, 1, 2, 3)$$

Punto 1.4 Factorización de Matrices

```
# Código desarrollado por: profesor Alfonso Mancilla  
def Sustituciones_Progresivas(L,b):  
    n=len(b)  
    x=np.zeros(b.shape)  
    if L[0,0]==0:  
        return None
```

```

else:
    x[0]=b[0]/L[0,0]
    for i in range(1,n):
        if L[i,i]==0:
            break
        else:
            x[i]=(b[i]-np.dot(L[i,0:i],x[0:i]))/L[i,i]
    return x

def Sustituciones_Regresivas(U,b):
    n=len(b)
    x=np.zeros(b.shape)
    if U[0,0]==0:
        return None
    else:
        x[n-1]=b[n-1]/U[n-1,n-1]
        print(x[n-1])
        for i in range(n-2,-1,-1):
            if U[i,i]==0:
                break
            else:
                x[i]=(b[i]-np.dot(U[i,i+1:n],x[i+1:n]))/U[i,i]
    return x

import numpy as np, sympy as sp,matplotlib.pyplot as plt, scipy as sc

A = np.array([[ 1, 1, 1, 1],[ 1, 2, 3, 5],[ 1, 3, 6, 15],[ 1, 5, 15, 70]])
b = np.array([3, 8, 0, 6])
# print(np.eye(4,4), '\n') #explicacion de los intercambios
P, L, U = sc.linalg.lu(A) #DOLITTLE-CROUT
print('Matriz de permutación P: \n', P, '\n\n Matriz triangular inferior L (Dolittle): \n',L, '\n\n Ma

y= np.linalg.inv(L)@P@b.T

x_fm= np.linalg.inv(U)@y.T #factorizacion de matrices
x_mc=np.linalg.inv(A)@b.T #metodo clasico

y=Sustituciones_Progresivas(L,P@b)
print('\nFactorización de Matrices, Sustituciones Progresivas. L.y=P.b \ny=',y)
x=Sustituciones_Regresivas(U,y)
print('\nFactorización de Matrices, Sustituciones Regresivas. U.x=y \nx=',x)

```

Solución:

Matriz de permutación **P**:

$$\begin{pmatrix} 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 0. & 1. & 0. \\ 0. & 1. & 0. & 0. \end{pmatrix}$$

Matriz triangular inferior **L** (Dolittle):

$$\begin{pmatrix} 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. \\ 1. & 0.5 & 1. & 0. \\ 1. & 0.25 & 0.75 & 1. \end{pmatrix}$$

Matriz triangular superior **U**:

$$\begin{pmatrix} 1. & 1. & 1. & 1. \\ 0. & 4. & 14. & 69. \\ 0. & 0. & -2. & -20.5 \\ 0. & 0. & 0. & 2.125 \end{pmatrix}$$

Factorización de Matrices, Sustituciones Progresivas. $\mathbf{L} \cdot \mathbf{y} = \mathbf{P} \cdot \mathbf{b}$

$$\mathbf{y} = [3. \quad 3. \quad -4.5 \quad 7.625]$$

Factorización de Matrices, Sustituciones Regresivas. $\mathbf{U} \cdot \mathbf{x} = \mathbf{y}$

$$\mathbf{x} = [-25.76470588 \quad 59.70588235 \quad -34.52941176 \quad 3.58823529]$$

Punto 1.5 Métodos Iterativos - Gauss Seidel

```
A = np.array([[ 1, 1, 1, 1],[ 1, 2, 3, 5],[ 1, 3, 6, 15],[ 1, 5, 15, 70]])
DP = np.diag(abs(A),0)
NDP = np.sum(abs(A),axis=1).T-DP
```

```
print('A = \n', A ,'\n\n','DP = \n', DP,'\n\n','NDP = \n', NDP ,'\n')
```

```
def cgjygs(A):
    DP = np.diag(np.abs(A),0)
    NDP = np.sum(np.abs(A),axis=1).T-DP
    if np.all(DP > NDP, axis=0):
        print("La matriz es estrictamente dominante en sentido diagonal")
        return True
    else:
        print("La matriz No es estrictamente dominante en sentido diagonal")
        return False
```

```
print(cgjygs(A))
```

Solución:

Matriz **A**:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 5 \\ 1 & 3 & 6 & 15 \\ 1 & 5 & 15 & 70 \end{pmatrix}$$

Vector DP:

$$\begin{pmatrix} 1 \\ 2 \\ 6 \\ 70 \end{pmatrix}$$

Vector NDP:

$$\begin{pmatrix} 3 \\ 9 \\ 19 \\ 21 \end{pmatrix}$$

La matriz **no** es estrictamente dominante en sentido diagonal: **False**

Problema 04: Ejercicio acordado del taller del taller de interpolación Funcional

La intensidad de la corriente en un circuito eléctrico obedece la ecuación $I(t) = 10 * e^{-t/10} * \text{sen}(2 * t)$. Para la calibración de un dispositivo eléctrico se hacen cinco mediciones de la intensidad de la corriente, en el laboratorio.

Los valores obtenidos están registrados en la siguiente tabla.

t	I(t)
1.0	8.2277
1.1	7.2428
1.2	5.9908
1.3	4.5260
1.4	2.9122

Table 1: Muestras de $I(t)$ en un lapso t

Solución: Problema 4 Interpolación Funcional

```
#Valores obtenidos de I en un lapso t
t = np.array([1.0, 1.1, 1.2, 1.3, 1.4])
I = np.array([8.2277, 7.2428, 5.9908, 4.5260, 2.9122])
```

2.1 El polinomio de interpolación

Coloque aquí, en un sólo lienzo, la gráfica del Polinomio de Interpolación (con guiones azules), Los puntos de la tabla (con círculos negros) y la función $I(t)$ (con puntos magenta). Etiquete los ejes y coloque la leyenda correspondiente.(30%).

Solución:

```
#sabemos que t es x y I(t) es y
def f(t):
    return 10 * np.exp(-t / 10) * np.sin(2 * t)

x = np.linspace(1, 1.4, 100)
y = np.array([f(t) for t in x])

PL = lagrange(t, I) # Lagrange
print(PL)

t_interpolation = np.linspace(1, 1.4, 100)
I_interpolation = PL(t_interpolation)
```

```
plt.plot(x, y, 'm.', label='Función I(t)')
plt.plot(t, I, 'ko', label='Puntos de la tabla')
plt.plot(t_interpolation, I_interpolation, 'b-', label='Polinomio de Interpolación')

#aqui nombro los ejes
plt.xlabel('t')
plt.ylabel('I(t)')

plt.legend()
plt.title('Polinomio de Interpolación y I(t)= 10*e^-t/10*sen(2*t)')
plt.grid(True)
plt.show()
```

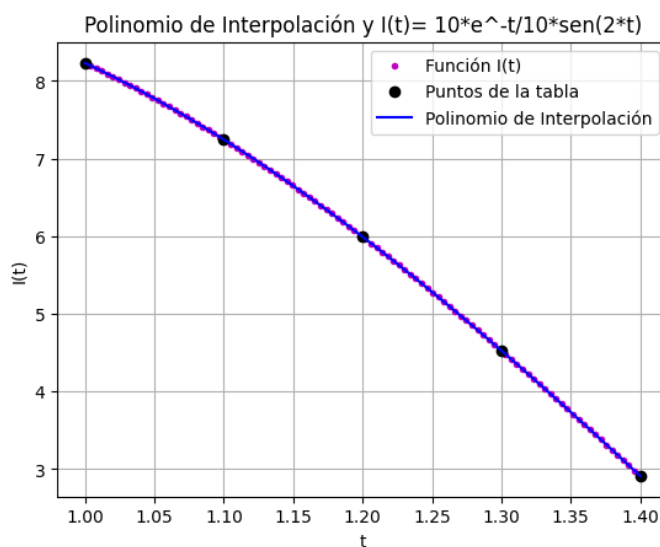


Figure 20: $3.958x^4 - 9.158x^3 - 11.91x^2 + 27.1x - 1.767$

2.2 El polinomio de Lagrange

Determine dos de los PCL, polinomios cocientes de Lagrange. Graficarlos en un lienzo. En el título especifique cuales son, etiquete los ejes y coloque la leyenda correspondiente. (30%).

$$P_n(x) = \sum_{k=0}^n y_k \cdot L_{n,k}(x),$$

DONDE

$$L_{n,k}(x) = \frac{\prod_{j=0, j \neq k}^n (x - x_j)}{\prod_{j=0, j \neq k}^n (x_k - x_j)} \text{ son los Polinomios coeficientes de Lagrange}$$

EJEMPLO. POLINOMIO DE LAGRANGE DE GRADO 2.

$$P_2(x) \sum_{k=0}^n y_k \cdot L_{n,k}(x) = y_0 \cdot L_{2,0}(x) + y_1 \cdot L_{2,1}(x) + y_2 \cdot L_{2,2}(x)$$

lo que esta en la funcion coeficients lo saque con ayuda de chatgpt y utilizando la productoria dada en el overleaf

$$L_{n,k}(x) = \frac{\prod_{j=0, j \neq k}^n (x - x_j)}{\prod_{j=0, j \neq k}^n (x_k - x_j)} \text{ son los Polinomios coeficientes de Lagrange}$$

Figure 21: Referencia

Codigo

```
def f(t):
    return 10 * np.exp(-t / 10) * np.sin(2 * t)
t = np.array([1.05, 1.15, 1.25, 1.35])
I_t = f(t)

# Polinomios Cocientes de Lagrange
def coeficients(t, k):
    n = len(t) - 1
    coef = 1
    for j in range(n + 1):
        if j != k:
            coef *= np.poly1d([1, -t[j]]) / (t[k] - t[j])
    return coef

x = np.linspace(min(t), max(t), 100)
pcl_0 = np.polyval(coeficients(t, 0), x)
pcl_1 = np.polyval(coeficients(t, 1), x)

plt.figure()
plt.plot(x, pcl_0, label='PCL 0')
plt.plot(x, pcl_1, label='PCL 1')

plt.xlabel('t')
plt.ylabel('PCL(t)')
plt.title('PCL 0 y PCL 1')
plt.legend()
plt.grid(True)
plt.show()
```

2.3 Error en la interpolación

Utilice Interpolación para estimar los valores de la intensidad de corriente en $t = [1.05, 1.15, 1.25, 1.35]$. Imprima aquí una tabla con los Errores Absolutos y relativos correspondientes a la interpolación.

```
import numpy as np

def f(t):
    return 10 * np.exp(-t / 10) * np.sin(2 * t)

t = np.array([1.05, 1.15, 1.25, 1.35])
I = f(t)
```

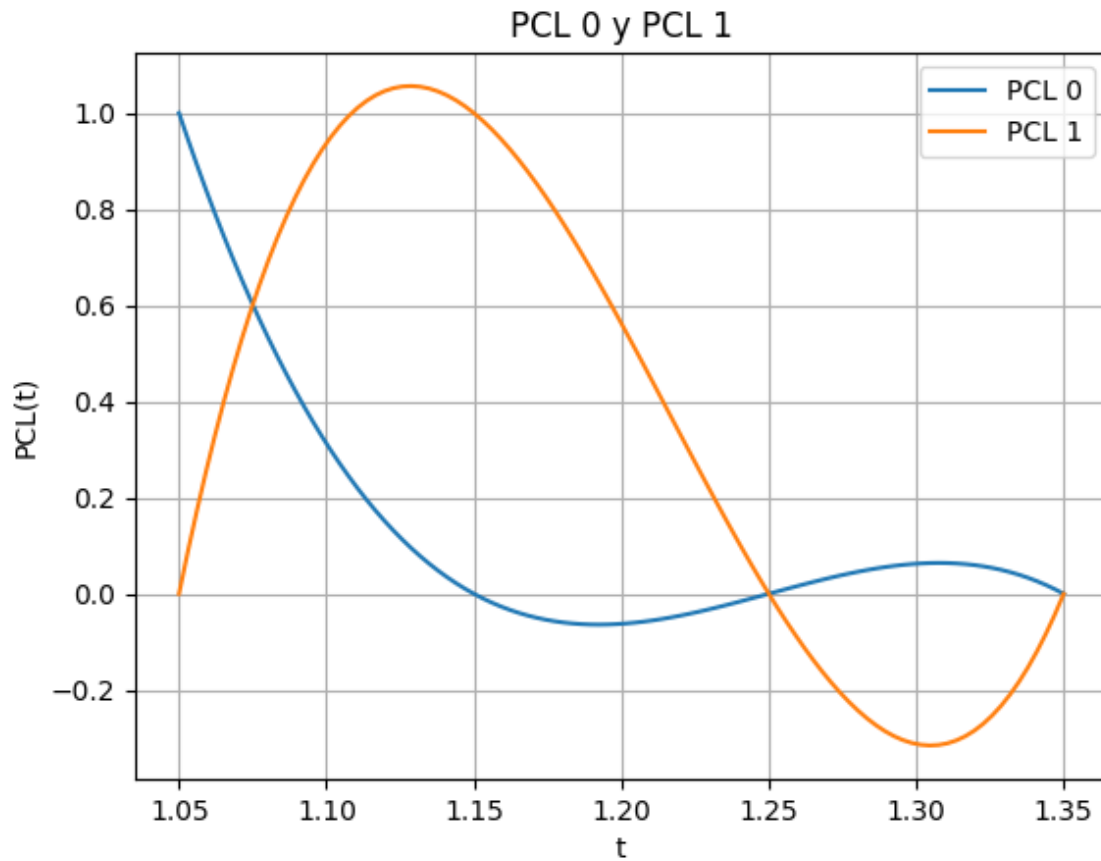


Figure 22: Lagrange

```
x = np.array([1, 1.1, 1.2, 1.3, 1.4])
y = f(x)

table = "  t   |  I(t)  |  P_n(t)  |  E_a(t)  |  E_r(t)  \n"
table += "-----+-----+-----+-----+-----\n"
for i in range(len(t)):
    coef_P = np.polyfit(x, y, i+1)
    P_t = np.polyval(coef_P, t)
    Ea = np.abs(I - P_t)
    Er = Ea / np.abs(I)
    table += f" {t[i]:.2f} | {I[i]:.4f} | {P_t[i]:.4f} | {Ea[i]:.4f} | {Er[i]:.4f} \n"

print(table)
```

Solución:

t	$I(t)$	$P_n(t)$	$E_a(t)$	$E_r(t)$
1.05	7.7717	7.7821	0.0104	0.0013
1.15	6.6470	6.6313	0.0157	0.0024
1.25	5.2815	5.2812	0.0003	0.0001
1.35	3.7341	3.7340	0.0001	0.0000

Table 2: Valores de t , $I(t)$, $P_n(t)$, $E_a(t)$ y $E_r(t)$ **Problema 05**

Utilizando las técnicas de integración numérica estudiadas, los algoritmos, script y funciones desarrollados en clases, genere las tablas estadísticas correspondientes a las siguientes distribuciones de probabilidad:

- t de Student
- Chi cuadrado

Escoger sólo una entre: distribución acumulada, de cola derecha o central.

t student:

```
import numpy as np, sympy as sp, scipy as sc, matplotlib.pyplot as plt
import scipy.stats as stats
def fdp_t_estandar(t, df):
    return stats.t.pdf(t, df)
df = 10
f = lambda x: fdp_t_estandar(x, df)
I = stats.t.cdf(0.72, df) - stats.t.cdf(0, df)
col0 = np.arange(0, 4.1, 0.1)
fil0 = np.arange(0, 0.1, 0.01)
nc = fil0.size
nf = col0.size
N = np.zeros((nf+1, nc+1))
N[0, 1:] = fil0
N[1:, 0] = col0
for i in range(1, nf+1):
    for j in range(1, nc+1):
        t = N[i, 0] + N[0, j]
        N[i, j] = stats.t.cdf(t, df)
N
```

Solucion propuesta en el colab**Chi cuadrado**

```
import numpy as np, sympy as sp, scipy as sc, matplotlib.pyplot as plt
import scipy.stats as stats
def fdp_chi_cuadrado(x, df):
    return stats.chi2.pdf(x, df)
df = 10
```



```
f = lambda x: fdp_chi_cuadrado(x, df)
I = stats.chi2.cdf(0.72, df) - stats.chi2.cdf(0, df)
col0 = np.arange(0, 4.1, 0.1)
fil0 = np.arange(0, 0.1, 0.01)
nc = fil0.size
nf = col0.size
N = np.zeros((nf+1, nc+1))
N[0, 1:] = fil0
N[1:, 0] = col0
for i in range(1, nf+1):
    for j in range(1, nc+1):
        x = N[i, 0] + N[0, j]
        N[i, j] = stats.chi2.cdf(x, df)
N
```

Solucion propuesta en el colab

Problema 06: Implementar el método de interpolación de Newton en Python

Solución: Problema 06: Implementar el método de interpolación de Newton en Python

```
# Profesor Alfonso Mancilla
# Diferencias Divididas de Newton
import numpy as np, sympy as sp, matplotlib.pyplot as plt
def f(x):
    return x+(2/x)

X=np.array([1,2,2.5])
Y=np.array([3,3,3.3])#Y=np.array([f(l) for l in X])
n=len(X)-1; #Grado del Polinomio
D=np.zeros((n+1,n+2));#Matriz para el cálculo de las DD0
D[:,0]=X; D[:,1]=Y;# Primera columna para X y segunda para Y
for j in range(2,n+2):
    for i in range(j-1,n+1):
        D[i,j]=(D[i,j-1]-D[i-1,j-1])/(X[i]-X[i-(j-1)])

DD=np.diag(D,1);#Las DD quedan en la diagonal principal número 1.
print(D,'\n\nDiferencias Divididas:\n', DD)
```

Solucion:

1.0	3.0	0.0	0.0
2.0	3.0	0.0	0.0
2.5	3.3	0.6	0.4

Diferencias Divididas: [3.0, 0.0, 0.4]

Problema 07: Ejercicio acordado del taller de Derivación numérica**Solución: Problema 07: Derivación numérica****Linealización modelo 1.1****Con a:**

$$f(t) = a \cdot e^{-b \cdot (e^{-c \cdot t})} \quad (18.1)$$

Primero ya sabemos que eso tiene forma de modelo exponencial

$$y = C \cdot e^{A \cdot x} \quad (18.2)$$

$$\ln(y) = A \cdot \ln(x) + \ln(C)$$

Tenemos según el libro que:

$$X = x$$

$$Y = \ln(y)$$

$$C = e^B$$

Entonces tenemos que:

1. Realizar la transformación:

$$X = t, \quad Y = \ln(f(t))$$

$$x = X, \quad y = e^Y$$

2. aplicamos la transformación y colocamos los ln

$$y = a \cdot e^{-b \cdot (e^{-c \cdot X})}$$

$$Y = \ln(y) = \ln \left(a \cdot e^{-b \cdot (e^{-c \cdot X})} \right)$$

$$Y = \ln(a) - b \cdot (e^{-c \cdot X})$$

3. Definimos variable J

$$J = e^{-c \cdot X}$$

4. la reescribimos

$$Y = \ln(a) - b \cdot J$$

5. Organizamos la expresión, separando cada dato

$$Y = \ln(f(t)), \quad X = t, \quad J = e^{-c \cdot X},$$

$$\ln(a) = \beta_0, \quad b = \beta_1$$

6. al final deje fijada la a como constante7. entonces si lo queremos dejar de la forma $y = mx + b$

$$y = -\beta_1 \cdot J + \beta_0$$

$$y = b \cdot J + \ln(a)$$

codigo:

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

f = lambda t, a, b, c: a * np.exp(-b * np.exp(-c * t))

a = 1 # Constante fijada
b = 2
c = 0.5
t = np.linspace(0, 5, 100)

a_values = [np.exp(1)/2, 1, -1/2]

fig, ax = plt.subplots()
ax.plot(t, f(t, a, b, c), label='a = 1')
ax.plot(t, f(t, a_values[0], b, c), label='a = e/2')
ax.plot(t, f(t, a_values[2], b, c), label='a = -1/2')

ax.legend()
plt.show()
```

solución:**Con b:**

$$b = \text{cte} \quad (18.3)$$

$$y = a \cdot e^{-b \cdot (e^{-c \cdot t})} \quad (18.4)$$

$$\ln(y) = \ln\left(a \cdot e^{-b \cdot (e^{-c \cdot t})}\right) \quad (18.5)$$

$$\ln(y) = \ln(a) + \ln\left(e^{-b \cdot (e^{-c \cdot t})}\right) \quad (18.6)$$

$$\ln(y) = \ln(a) + -b \cdot (e^{-c \cdot t}) \cdot \ln(e) \quad (18.7)$$

$$\ln(y) = \ln(a) + -b \cdot (e^{-c \cdot t}) \quad (18.8)$$

$$(18.9)$$

Codigo:

```
import numpy as np, sympy as sp, matplotlib.pyplot as plt
def f(t, a, b, c):
    return a * np.exp(-b * (np.exp(-c * t)))
def F(x, y, a, b, c, b0, b1):
    return f(0, a, -1/4, c) * g(x, y) + g(b0, b1)
def g(x, y):
    return f(x, -1, y, c)
a = 2
b = 0.2
c = 0.5
t = np.linspace(1,10, 100)
y1 = f(t, a, -1/4, c)
y2 = f(t, -1, b, c)
```

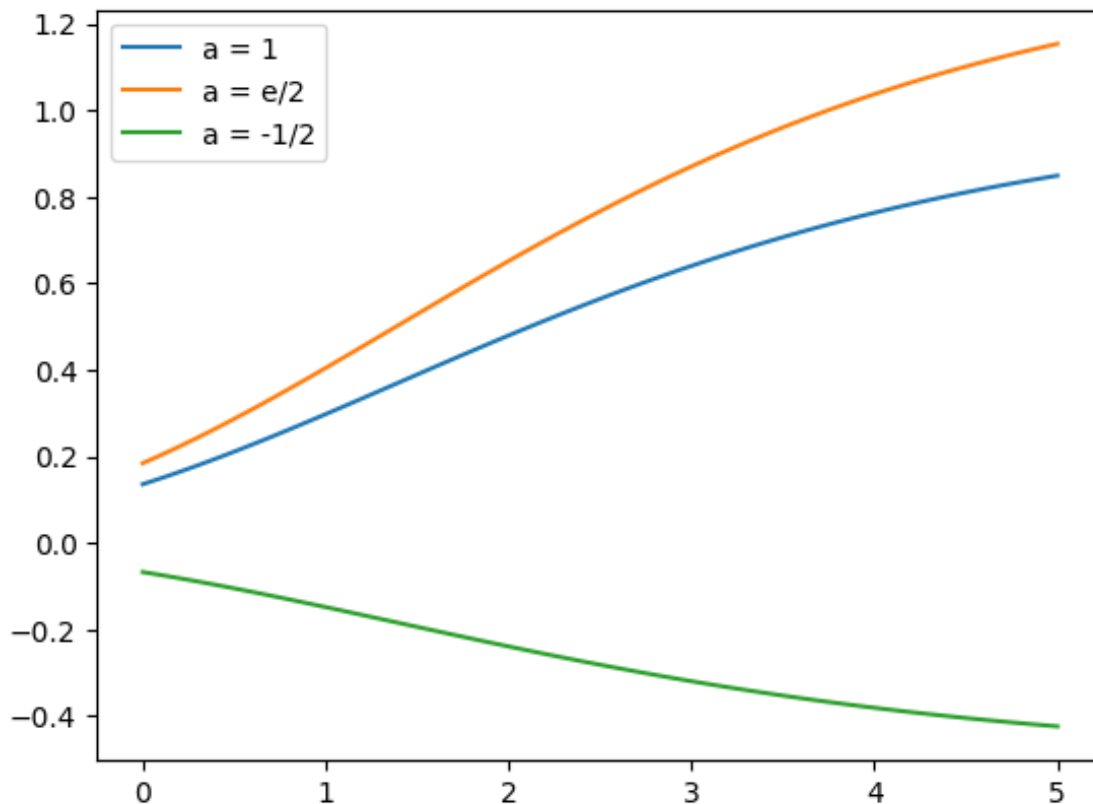


Figure 23: Con a fija

```

y3 = f(t, a, -4, c)
plt.plot(t, y1, label='f1')
plt.plot(t, y2, label='f2')
plt.plot(t, y3, label='f3')
plt.xlabel('t')
plt.ylabel('f(t)')
plt.title('Gráfico de las funciones f1, f2 y f3')
plt.legend()
plt.show()

```

Solución: Modelo 2

Esto es, la transformación de éste en la forma $F(xoy) = f(\beta_0 o \beta_1) \cdot G(xoy) + g(\beta_0, \beta_1)$.

Tenemos que

$$i = \frac{a \cdot T^b}{t_d^c}$$

$$\ln(i) = \ln\left(\frac{a \cdot T^b}{t_d^c}\right)$$

$$\ln(i) = \ln(a \cdot T^b) - \ln(t_d^c)$$

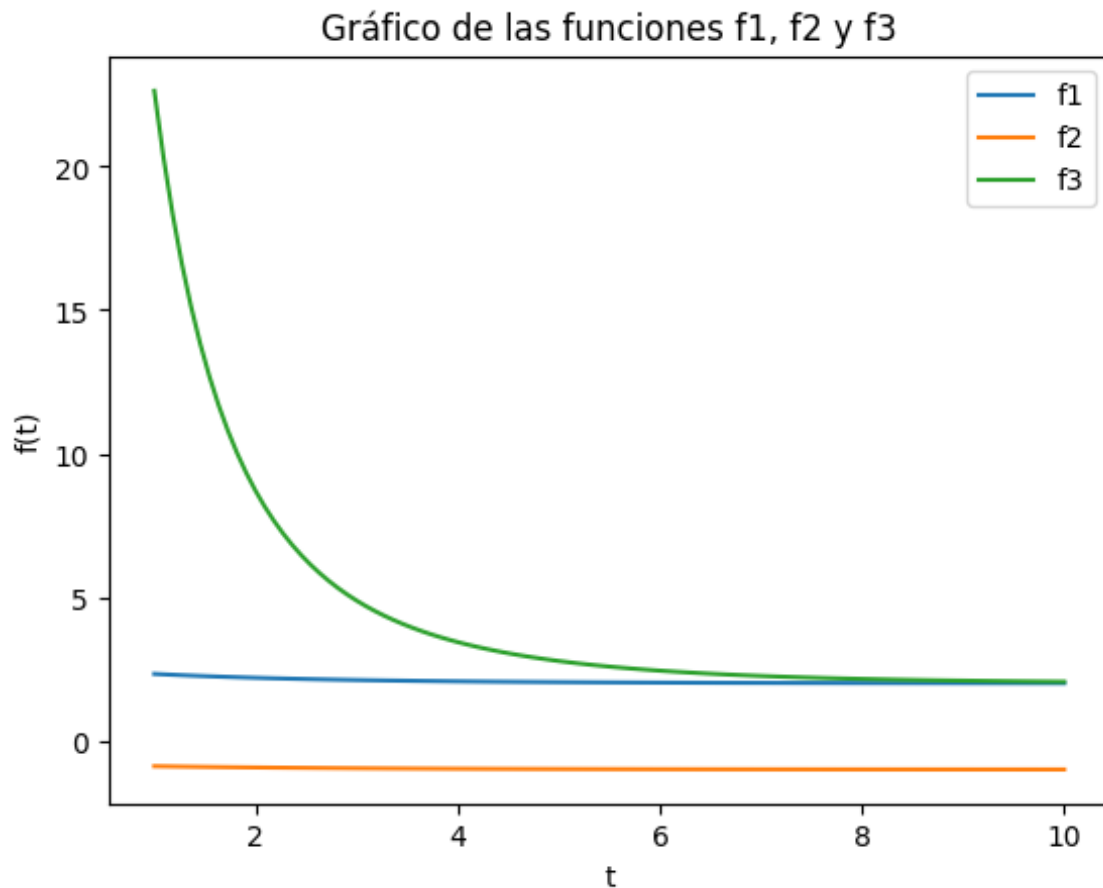


Figure 24: Con b fija

$$\ln(i) = \ln(a \cdot T^b) - c \cdot \ln(t_d)$$

$$\ln(i) = \ln(a) + \ln(T^b) - c \cdot \ln(t_d)$$

$$\ln(i) = \ln(a) + b \cdot \ln(T) - c \cdot \ln(t_d)$$

Donde:

$$F(x \text{ o } y) = \ln(i)$$

$$f(\beta_0 \text{ o } \beta_1) = \ln(a)$$

$$G(x \text{ o } y) = \ln(T)$$

$$g(\beta_0, \beta_1) = -c \cdot \ln(t_d)$$

Problema 08: Ejercicio acordado del taller de Integración numérica

Solución Ajuste de curvas

Codigo:

```
import numpy as np, sympy as sp, matplotlib.pyplot as plt
td = np.array([5, 10, 20, 45, 80, 120])
i = np.array([68, 41, 21, 9, 5, 4])
```

```

y1c = lambda td, a, c: a / (td ** c)
X = np.log(td)
Y = np.log(i)
X = np.array([np.ones(len(td)), X]).T
mb, _, _ = np.linalg.lstsq(X, Y)
b = mb[0]
m = mb[1]
c = -m
a = np.exp(b)
ejex = np.linspace(min(td), max(td))
fig, ax = plt.subplots()
ax.plot(td, i, 'r', label="Nube de puntos")
xl = plt.xlim()
yl = plt.ylim()
print(plt.xlim(), plt.ylim())
ax.plot(ejex, y1c(ejex, a, c), 'k', label="Curva ajustada")
ax.plot(ejex, np.exp(b) * ejex ** m, 'm', label="Recta de regresión")
plt.xlim(xl)
plt.ylim(yl)
plt.title("Curva IDF")
plt.xlabel("eje x")
plt.ylabel("eje y")
plt.legend()
plt.show()

```

Solución:

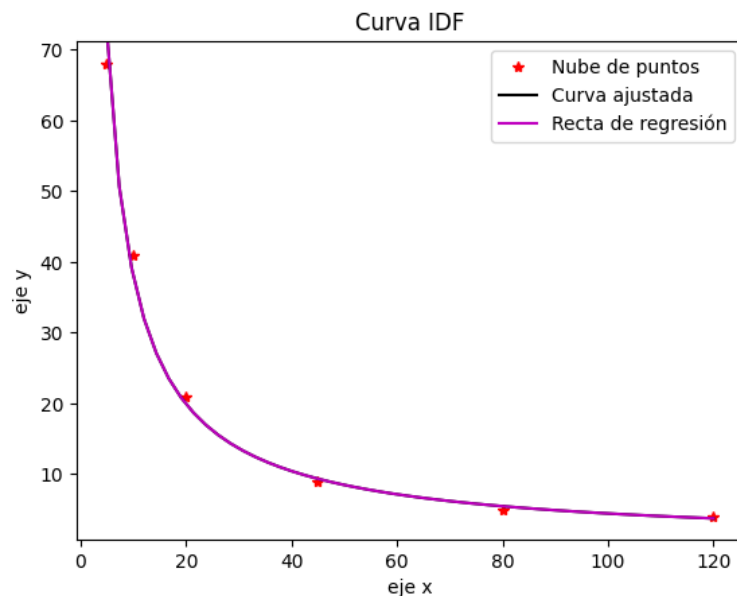


Figure 25: Curva

Problema 09: Ejercicio acordado del taller de Solución numérica de Ecuaciones Diferenciales ordinarias. ODE45(...)

Solución: Integración Numérica

Lo que chatgpt suministroo:

Codigo:

Cuando $h = 0.1$ y se utilizó $n1 = 17$

```
import scipy.integrate as spi
import numpy as np

def f(t):
    n1 = 17
    return np.cos(0.6 * np.sin((t + 1) * np.pi / 2 + 2 * n1 * np.pi) + 2 * n1 * np.pi)
a = 17 - 1
b = 17 + 1
integral_exacta, _ = spi.quad(f, a, b)
print("Valor exacto de la integral:", integral_exacta)
```

Resultado:

Valor exacto de la integral: 1.8240097269944222

```
import numpy as np
import scipy.integrate as spi

def f(t):
    n1 = 17
    return np.cos(0.6 * np.sin((t + 1) * np.pi / 2 + 2 * n1 * np.pi) + 2 * n1 * np.pi)

a = 17 - 1
b = 17 + 1
h = 0.1 # Incremento
n = int((b - a) / h)
def trapecio(f, a, b, n):
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = f(x)
    integral = (h / 2) * (y[0] + 2 * np.sum(y[1:n]) + y[n])
    return integral

def simpson(f, a, b, n):
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = f(x)
    integral = (h / 3) * (y[0] + 2 * np.sum(y[2:n-1:2]) + 4 * np.sum(y[1:n:2]) + y[n])
    return integral

def trapz_numpy(f, a, b, n):
    x = np.linspace(a, b, n+1)
```

```
y = f(x)
integral = np.trapz(y, x)
return integral

def quad_integration(f, a, b):
    integral, _ = spi.quad(f, a, b)
    return integral

integral_trapecio = trapecio(f, a, b, n)
integral_simpson = simpson(f, a, b, n)
integral_trapezoidal = trapz_numpy(f, a, b, n)
integral_quad = quad_integration(f, a, b)

# Calcular el valor exacto de la integral (utilizando ChatGPT) esto esta mal
# integral_exacta = -0.35006604147326735
integral_exacta = 1.8240097269944222

error_relativo_trapecio = abs(integral_trapecio - integral_exacta) / abs(integral_exacta)
error_absoluto_trapecio = abs(integral_trapecio - integral_exacta)

error_relativo_simpson = abs(integral_simpson - integral_exacta) / abs(integral_exacta)
error_absoluto_simpson = abs(integral_simpson - integral_exacta)

error_relativo_trapezoidal = abs(integral_trapezoidal - integral_exacta) / abs(integral_exacta)
error_absoluto_trapezoidal = abs(integral_trapezoidal - integral_exacta)

error_relativo_quad = abs(integral_quad - integral_exacta) / abs(integral_exacta)
error_absoluto_quad = abs(integral_quad - integral_exacta)

print("Integral utilizando método del trapecio:", integral_trapecio)
print("Error relativo (método del trapecio):", error_relativo_trapecio)
print("Error absoluto (método del trapecio):", error_absoluto_trapecio, '\n')
print("Integral utilizando regla de Simpson:", integral_simpson)
print("Error relativo (regla de Simpson):", error_relativo_simpson)
print("Error absoluto (regla de Simpson):", error_absoluto_simpson, '\n')
print("Integral utilizando numpy.trapz:", integral_trapezoidal)
print("Error relativo (numpy.trapz):", error_relativo_trapezoidal)
print("Error absoluto (numpy.trapz):", error_absoluto_trapezoidal, '\n')
print("Integral utilizando scipy.integrate.quad:", integral_quad)
print("Error relativo (scipy.integrate.quad):", error_relativo_quad)
print("Error absoluto (scipy.integrate.quad):", error_absoluto_quad)
```

Solución:

Método	Integral	Error relativo	Error absoluto
Método del trapecio	1.8240097269944233	6.086716579382318e-16	1.1102230246251565e-15
Regla de Simpson	1.8240097269944227	2.4346866317529273e-16	4.440892098500626e-16
numpy.trapz	1.824009726994423	4.869373263505855e-16	8.881784197001252e-16
scipy.integrate.quad	1.8240097269944222	0.0	0.0

Errores:**Codigo**

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

errores_relativos = [error_relativo_trapecio, error_relativo_simpson, error_relativo_trapezoidal, error_relativo_quad]
metodos = ['Trapecio', 'Simpson', 'numpy.trapz', 'scipy.integrate.quad']
fig, ax = plt.subplots()

ax.bar(metodos, errores_relativos)
ax.set_xlabel('Método de Integración')
ax.set_ylabel('Error Relativo')
ax.set_title('Errores Relativos en la Integración Numérica')
plt.show()
```

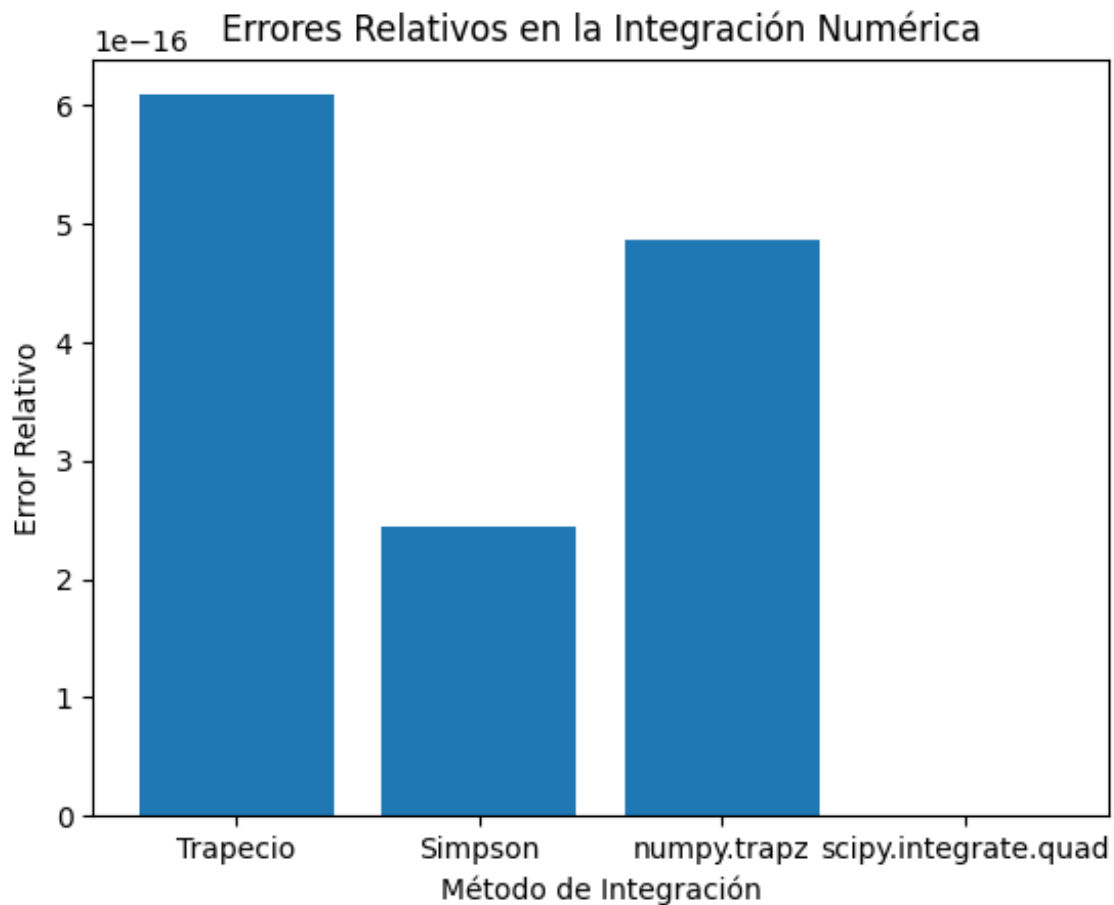


Figure 26: Grafico Error

cuando $h = 2$

Codigo:

```
import numpy as np
import scipy.integrate as spi
def f(t):
    nl = 17
    return np.cos(0.6 * np.sin((t + 1) * np.pi / 2 + 2 * nl * np.pi) + 2 * nl * np.pi)
a = 17 - 1
b = 17 + 1
h = .2 # Incremento
n = int((b - a) / h)
def trapecio(f, a, b, n):
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = f(x)
    integral = (h / 2) * (y[0] + 2 * np.sum(y[1:n]) + y[n])
    return integral

def simpson(f, a, b, n):
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = f(x)
    integral = (h / 3) * (y[0] + 2 * np.sum(y[2:n-1:2]) + 4 * np.sum(y[1:n:2]) + y[n])
    return integral

def trapz_numpy(f, a, b, n):
    x = np.linspace(a, b, n+1)
    y = f(x)
    integral = np.trapz(y, x)
    return integral

def quad_integration(f, a, b):
    integral, _ = spi.quad(f, a, b)
    return integral
integral_trapecio = trapecio(f, a, b, n)
integral_simpson = simpson(f, a, b, n)
integral_trapezoidal = trapz_numpy(f, a, b, n)
integral_quad = quad_integration(f, a, b)

# Calcular el valor exacto de la integral (utilizando ChatGPT) esto esta mal
# integral_exacta = -0.35006604147326735
integral_exacta = 1.8240097269944222

error_relativo_trapecio = abs(integral_trapecio - integral_exacta) / abs(integral_exacta)
error_absoluto_trapecio = abs(integral_trapecio - integral_exacta)

error_relativo_simpson = abs(integral_simpson - integral_exacta) / abs(integral_exacta)
error_absoluto_simpson = abs(integral_simpson - integral_exacta)

error_relativo_trapezoidal = abs(integral_trapezoidal - integral_exacta) / abs(integral_exacta)
error_absoluto_trapezoidal = abs(integral_trapezoidal - integral_exacta)
```

```
error_relativo_quad = abs(integral_quad - integral_exacta) / abs(integral_exacta)
error_absoluto_quad = abs(integral_quad - integral_exacta)
```

```
print("Integral utilizando método del trapecio:", integral_trapecio)
print("Error relativo (método del trapecio):", error_relativo_trapecio)
print("Error absoluto (método del trapecio):", error_absoluto_trapecio, '\n')
print("Integral utilizando regla de Simpson:", integral_simpson)
print("Error relativo (regla de Simpson):", error_relativo_simpson)
print("Error absoluto (regla de Simpson):", error_absoluto_simpson, '\n')
print("Integral utilizando numpy.trapz:", integral_trapezoidal)
print("Error relativo (numpy.trapz):", error_relativo_trapezoidal)
print("Error absoluto (numpy.trapz):", error_absoluto_trapezoidal, '\n')
print("Integral utilizando scipy.integrate.quad:", integral_quad)
print("Error relativo (scipy.integrate.quad):", error_relativo_quad)
print("Error absoluto (scipy.integrate.quad):", error_absoluto_quad)
```

Solución:

Método	Integral	Error relativo	Error absoluto
Método del trapecio	1.8240097269944244	1.2173433158764637e-15	2.220446049250313e-15
Regla de Simpson	1.8240097269965772	1.181431688058108e-12	2.154942890797429e-12
numpy.trapz	1.8240097269944244	1.2173433158764637e-15	2.220446049250313e-15
scipy.integrate.quad	1.8240097269944222	0.0	0.0

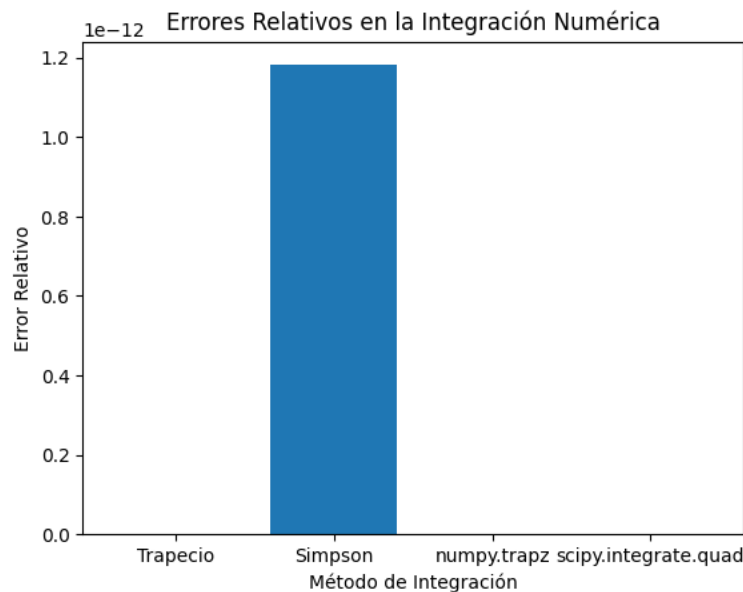
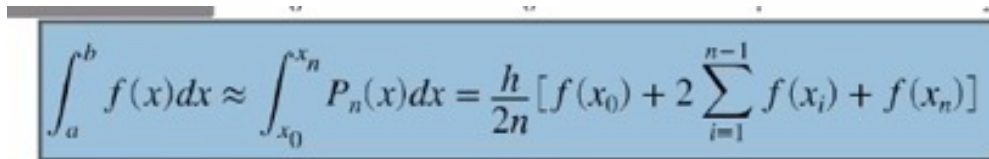


Figure 27: Grafico Error

Formula generalizada del trapecio teniendo en cuenta que:

$$\int_a^b f(x) \cdot dx = \frac{f(x_0) + 2 \sum_{i=1}^{n-1} f(x_i) + f(x_n)}{2n}$$



$$\int_a^b f(x) dx \approx \int_{x_0}^{x_n} P_n(x) dx = \frac{h}{2n} [f(x_0) + 2 \sum_{i=1}^{n-1} f(x_i) + f(x_n)]$$

Figure 28: Referencia - Clase

```
import numpy as np

def generalizada(f, a, b, n):
    h = (b - a) / n
    x = np.linspace(a, b, n + 1)
    y = f(x)
    integral = (y[0] + 2 * np.sum(y[1:n]) + y[n]) * h / 2
    return integral
resultado = generalizada(f, a, b, n)
print("Resultado de la integral:", resultado)
```

solución:

El resultado de la integral es: 1.8240097269944244.

Problema 10

Diseñe un programa en python que permita leer una matriz, de tamaño $n \times (m + 2)$ en el siguiente formato

$$\begin{pmatrix} x_1 & y_{1,1} & y_{1,2} & \cdots & y_{1,n} & y_1 \\ x_2 & y_{2,1} & y_{2,2} & \cdots & & y_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_{n,1} & y_{n,2} & \cdots & y_{n,n} & y_n \end{pmatrix} \quad (18.10)$$

donde $y_i = \sum_{j=1}^m \frac{y_{ij}}{m}$, $i = 1, 2, \dots, n$. Hacer lo siguiente:

- Ajustar la nube de puntos, mediante tres de los modelos registrados en la página 292 del texto guía, Mathews-Fink, la salida tiene que ser las gráficas correspondientes tal como se muestran en la página 291.
Para este ítem usted debe proveer al usuario de las opciones correspondientes a cada modelo identificando a éstos por su nombre, por ejemplo Modelo exponencial, Modelo potencial, Modelo Logarítmico, Modelo Logístico de crecimiento de poblaciones, etc.
- Determine analíticamente cuál de todos los modelos del numeral a) es el que mejor se ajusta al conjunto de datos dados.

Solución: Problema 10

Revisar el colab, ya que ahí esta todo con más especificaciones, sin embargo, aquí adjunto los principales codigos del inicio y sus respectivas graficas.

Modelo Ajuste Racional

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

x = np.array([0, 1, 2, 3, 4])
y = np.array([1.5, 2.5, 3.5, 5.0, 7.5])

y1calc = lambda x, C, D: D / (x + C)
X = np.multiply(x, y)
Y = y

X = (np.array([np.ones(len(x)), X])).T
mb, _, _ = np.linalg.lstsq(X, Y)
b = mb[0]
m = mb[1]
C = -1 / m
D = C * b
ejex = np.linspace(min(x), max(x))
fig, ax = plt.subplots()
ax.plot(x, y, '*k')
xl = plt.xlim()
yl = plt.ylim()
ax.plot(ejex, y1calc(ejex, C, D), 'oc')
plt.xlim(xl)
plt.ylim(yl)
plt.title("Modelo de Ajuste Racional")
plt.show()

print(D, C)
```

Grafica:

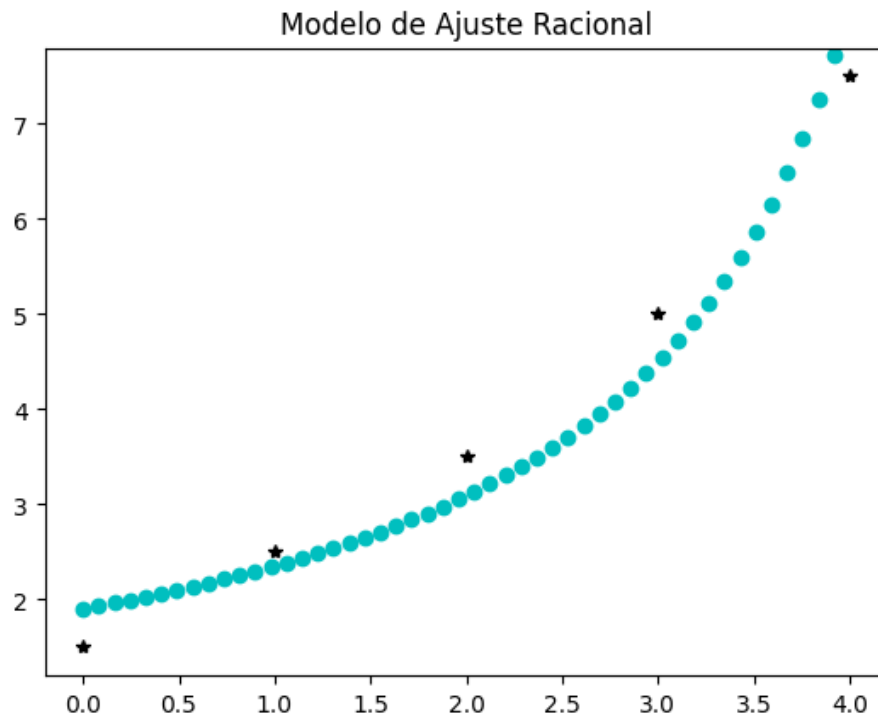


Figure 29: Modelo racional

Modelo Potencial:

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

x = np.array([0, 1, 2, 3, 4])
y = np.array([1.5, 2.5, 3.5, 5.0, 7.5])
y2calc = lambda x, A, B: 1 / (A * x + B)
X = x
Y = 1 / y
X = (np.array([np.ones(len(x)), x])).T
mb, _, _ = np.linalg.lstsq(X, Y)
b = mb[0]
m = mb[1]
A = m
B = b
ejex = np.linspace(min(x), max(x))

fig, ax = plt.subplots()
ax.plot(x, y, '*k')
xl = plt.xlim()
yl = plt.ylim()
ax.plot(ejex, y2calc(ejex, A, B), 'oc')
```

```
plt.xlim(xl)  
plt.ylim(yl)  
plt.title("Modelo Potencial")  
plt.show()
```

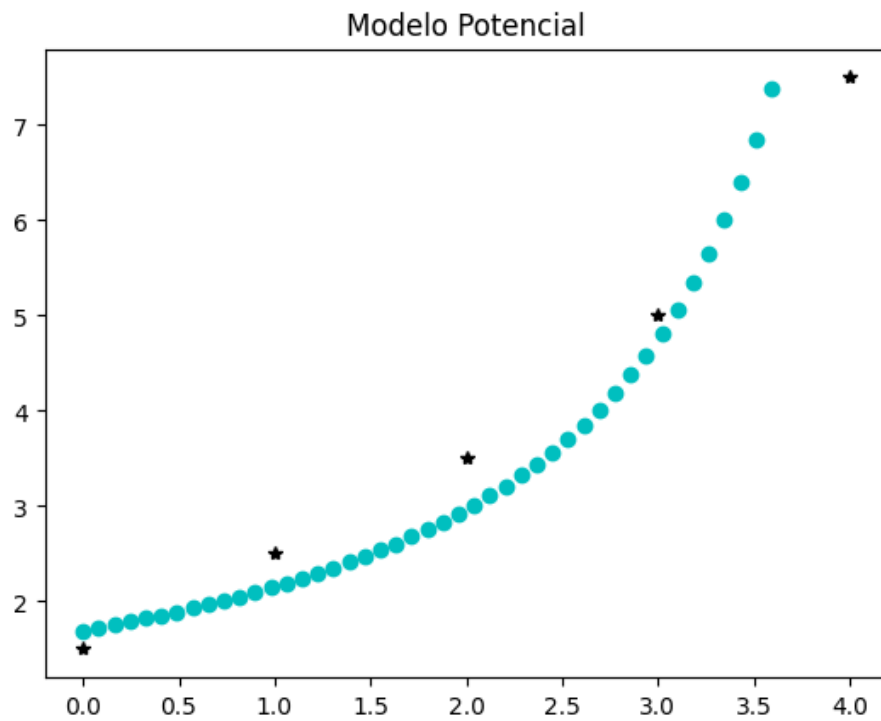


Figure 30: Modelo potencial

Modelo Exponencial

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

x = np.array([0, 1, 2, 3, 4])
y = np.array([1.5, 2.5, 3.5, 5.0, 7.5])
y_calc = lambda x, C, A: C * np.exp(A * x)

X = x
Y = np.log(y)

X = (np.array([np.ones(len(x)), X])).T
mb, _, _, _ = np.linalg.lstsq(X, Y)

b = mb[0]
B = b
C = np.exp(mb[0])
A = mb[1]
print('C =', C, ' A =', A, ' B =', B)
ejex = np.linspace(min(x), max(x))
fig, ax = plt.subplots()
ax.plot(x, y, '*k')
ax.plot(ejex, y_calc(ejex, C, A), 'oc')
plt.title("Modelo Exponencial")

plt.show()
```

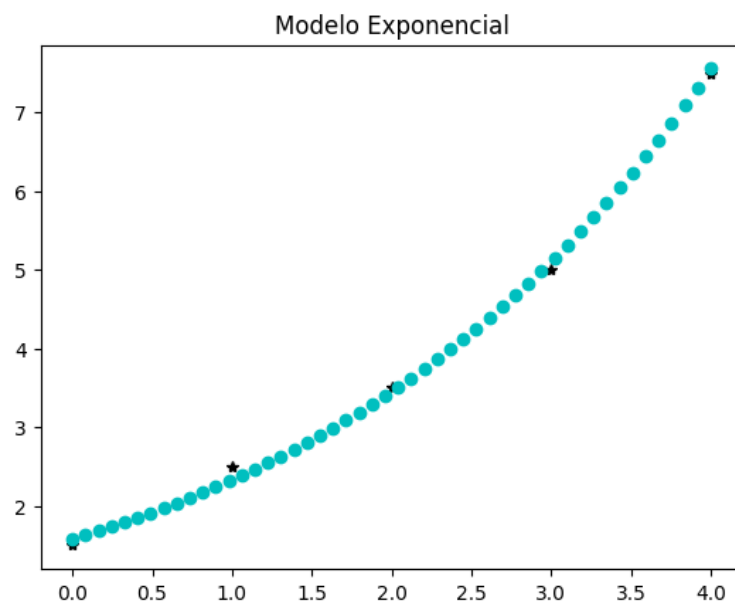


Figure 31: Modelo potencial

Las soluciones a los problemas aquí propuestos deben estar detalladas.

Componente		Porcentaje
Problemas	Problemas escogidos	65 %
Informe	Debe estar hecho con normas IEEE, APA o EasyChair u otro.	20 %
Vídeo	Individual. Cada miembro del equipo debe realizar su propio vídeo.	10 %
Autoevaluación	Individual. Cada miembro del equipo debe realizar su propia autoevaluación.	5 %

AUTOEVALUACIÓN:

1. María Isabel Solá Valle — 5/5
2. María José Duque Polo — 5/5