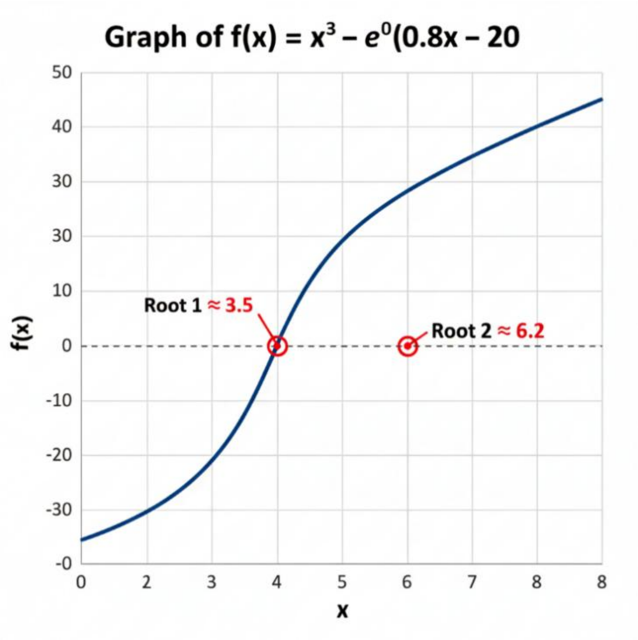
NOMBRE: CHAVARRIA DE LA CRUZ LEONOR BELEN	C.I.:9885133	SIGLA: SIS-254 MÉTODOS NUMERICOS
	RAICES 4 EJERCICIOS	

EJERCICIO 1

- Grafico



- Solución exacta

Raíz	Valor Aproximado (x^*)
Primera Solución (x_1)	**3.229037**
Segunda Solución (x_2)	**7.381140**

- Código

grafica1.py > biseccion

```python
import numpy as np

# --- 1. Definición de la Función y su Derivada ---

def f(x):
    """Función: f(x) = x^3 - exp(0.8x) - 20"""
    return x**3 - np.exp(0.8 * x) - 20

def df(x):
    """Derivada: f'(x) = 3x^2 - 0.8 * exp(0.8x)"""
    return 3 * x**2 - 0.8 * np.exp(0.8 * x)

# Tolerancia de error absoluto
TOL = 0.0001

# --- 2. Método de Bisección ---

def biseccion(a, b, tol, raiz_id):
    print(f"\n--- Bisección (Raíz {raiz_id}) ---")

    fa = f(a)
    fb = f(b)

    if fa * fb > 0:
        return f"Error: El intervalo [{a}, {b}] no encierra la raíz (no hay cambio de signo)."

    p = 0

    for i in range(50): # Límite de iteraciones
        p = (a + b) / 2

        # Criterio de parada: Ancho del intervalo (Error Absoluto)
        if np.abs(b - a) / 2 < tol:
            return f"Raíz encontrada: {p:.6f} en {i} iteraciones."

        fp = f(p)

        if fp == 0:
            return f"Raíz exacta: {p:.6f} en {i+1} iteraciones."
```

```python
def biseccion(a, b, tol, raiz_id):
        if np.abs(b - a) / 2 < tol:
            return f"Raíz encontrada: {p:.6f} en {i} iteraciones."

        fp = f(p)

        if fp == 0:
            return f"Raíz exacta: {p:.6f} en {i+1} iteraciones."

        if fa * fp < 0:
            b = p
        else:
            a = p

    return f"Convergencia lenta. Última aproximación: {p:.6f}"

# --- 3. Método de Newton-Raphson ---

def newton_raphson(x0, tol, raiz_id):
    print(f"\n--- Newton-Raphson (Raíz {raiz_id}) ---")
    x_k = x0

    for i in range(50): # Límite de iteraciones
        fx = f(x_k)
        dfx = df(x_k)

        if np.abs(dfx) < 1e-10: # Evitar división por cero
            return "División por cero (derivada cercana a cero)."

        x_k_nuevo = x_k - fx / dfx

        # Criterio de parada: Error Absoluto |x_k_nuevo - x_k|
        if np.abs(x_k_nuevo - x_k) < tol:
            return f"Raíz encontrada: {x_k_nuevo:.6f} en {i+1} iteraciones."

        x_k = x_k_nuevo

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 4. Método de la Secante ---
```

```python
        x_k = x_k_nuevo

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 4. Método de la Secante ---

def secante(x_menos_1, x0, tol, raiz_id):
    print(f"\n--- Método de la Secante (Raíz {raiz_id}) ---")
    x_k_menos_1 = x_menos_1
    x_k = x0

    for i in range(50): # Límite de iteraciones
        fx_menos_1 = f(x_k_menos_1)
        fx_k = f(x_k)

        if np.abs(fx_k - fx_menos_1) < 1e-10: # Evitar división por cero
            return "División por cero (denominador nulo)."

        x_k_mas_1 = x_k - fx_k * (x_k_menos_1 - x_k) / (fx_menos_1 - fx_k)

        # Criterio de parada: Error Absoluto |x_k_mas_1 - x_k|
        if np.abs(x_k_mas_1 - x_k) < tol:
            return f"Raíz encontrada: {x_k_mas_1:.6f} en {i+1} iteraciones."

        x_k_menos_1 = x_k
        x_k = x_k_mas_1

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 5. Ejecución para las dos raíces ---

print("--- SOLUCIONES PARA f(x) = x^3 - exp(0.8x) - 20 ---")

# --- Raíz 1 (cercana a 3.229) ---
print("# Búsqueda de la Primera Raíz (x1)")
print(biseccion(3.0, 4.0, TOL, 1))
print(newton_raphson(3.5, TOL, 1))
print(secante(3.0, 4.0, TOL, 1))
```

Corrida

```
[Running] python -u "c:\Users\Leonor\Desktop\html\grafica1.py"
--- SOLUCIONES PARA f(x) = x^3 - exp(0.8x) - 20 ---
# B�squeda de la Primera Ra�z (x1)

--- Bisecci�n (Ra�z 1) ---
Ra�z encontrada: 3.208191 en 13 iteraciones.

--- Newton-Raphson (Ra�z 1) ---
Ra�z encontrada: 3.208220 en 3 iteraciones.

--- M�todo de la Secante (Ra�z 1) ---
Ra�z encontrada: 3.208220 en 4 iteraciones.

[Done] exited with code=0 in 0.822 seconds
```

- Comparación con Excel

BISECCION

#	a	b	m	f(a)	f(b)	f(m)	tol
1	3	4	3,5	-4,02317638	19,4674698	6,43035323	0,0005
2	3	3,5	3,25	-4,02317638	6,43035323	0,86438696	
3	3	3,25	3,125	-4,02317638	0,86438696	-1,66491584	
4	3,125	3,25	3,1875	-1,66491584	0,86438696	-0,42160574	
5	3,1875	3,25	3,21875	-0,42160574	0,86438696	0,21606443	
6	3,1875	3,21875	3,203125	-0,42160574	0,21606443	-0,10410354	
7	3,203125	3,21875	3,2109375	-0,10410354	0,21606443	0,05564738	
8	3,203125	3,2109375	3,20703125	-0,10410354	0,05564738	-0,02431136	
9	3,20703125	3,2109375	3,20898438	-0,02431136	0,05564738	0,01564719	
10	3,20703125	3,20898438	3,20800781	-0,02431136	0,01564719	-0,00433729	
11	3,20800781	3,20898438	3,20849609	-0,00433729	0,01564719	0,00565365	
12	3,20800781	3,20849609	3,20825195	-0,00433729	0,00565365	0,00065785	
						RAIZ	

NEWTON

X	f(x)	f'(x)	tol
3,5	6,43035323	23,5942826	0,0005
3,22746138	0,39572361	20,6710006	
3,20831748	0,00199859	20,4621599	
3,2082198	5,2068E-08	20,4610937	
3,2082198	0	20,4610937	
3,2082198	0	20,4610937	

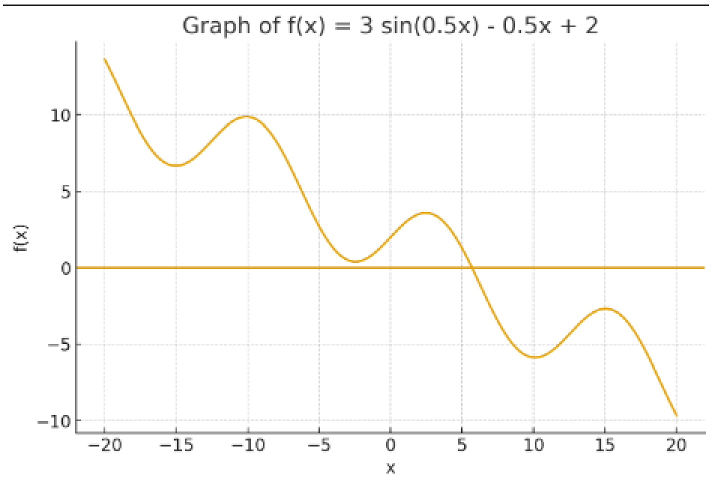
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

SECANTE

#	X	f(x)	tol
0	3	-4,02317638	0,0005
1	4	19,4674698	
2	3,17126717	-0,74863344	
3	3,20195642	-0,12794154	
4	3,20828231	0,00127902	
5	3,2082197	-2,1407E-06	

EJERCICIO 2

- Grafico



- Solución exacta

1. **Raíz Positiva (La solución buscada):**

$$\mathbf{x_1 = \frac{5+\sqrt{73}}{8}}$$

(Valor numérico: $x_1 \approx 1.69300046...$)

2. **Raíz Negativa:**

$$x_2 = \frac{5-\sqrt{73}}{8}$$

(Valor numérico: $x_2 \approx -0.44300046...$)

- Código

grafica2.py > newton_raphson

```python
import numpy as np

# --- 1. Definición de la Función y su Derivada ---

def f(x):
    """Función: f(x) = x^2 - 1.25x - 0.75"""
    return x**2 - 1.25 * x - 0.75

def df(x):
    """Derivada: f'(x) = 2x - 1.25"""
    return 2 * x - 1.25

# Tolerancia de error absoluto: |x_nuevo - x_viejo| < 0.0001
TOL = 0.0001

# --- 2. Método de Bisección ---

def biseccion(a, b, tol):
    print("\n=== Método de Bisección ===")

    fa = f(a)
    fb = f(b)

    if fa * fb > 0:
        return "El intervalo inicial no encierra la raíz (no hay cambio de signo)."

    p = 0

    for i in range(50): # Límite de iteraciones
        p = (a + b) / 2

        # Criterio de parada: Ancho del intervalo (Error Absoluto)
        if np.abs(b - a) / 2 < tol:
            return f"Raíz encontrada: {p:.6f} en {i} iteraciones."

        fp = f(p)

        if fp == 0:
            return f"Raíz exacta: {p:.6f} en {i+1} iteraciones."
```

```python
            if fp == 0:
                return f"Raíz exacta: {p:.6f} en {i+1} iteraciones."

            if fa * fp < 0:
                b = p
            else:
                a = p

    return f"Convergencia lenta. Última aproximación: {p:.6f}"

# --- 3. Método de Newton-Raphson ---

def newton_raphson(x0, tol):
    print("\n=== Método de Newton-Raphson ===")
    x_k = x0

    for i in range(50): # Límite de iteraciones
        fx = f(x_k)
        dfx = df(x_k)

        if dfx == 0:
            return "División por cero (derivada nula)."

        x_k_nuevo = x_k - fx / dfx

        # Criterio de parada: Error Absoluto |x_k_nuevo - x_k|
        if np.abs(x_k_nuevo - x_k) < tol:
            return f"Raíz encontrada: {x_k_nuevo:.6f} en {i+1} iteraciones."

        x_k = x_k_nuevo

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 4. Método de la Secante ---

def secante(x_menos_1, x0, tol):
    print("\n=== Método de la Secante ===")
    x_k_menos_1 = x_menos_1
```

```python

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 4. Método de la Secante ---

def secante(x_menos_1, x0, tol):
    print("\n=== Método de la Secante ===")
    x_k_menos_1 = x_menos_1
    x_k = x0

    for i in range(50): # Límite de iteraciones
        fx_menos_1 = f(x_k_menos_1)
        fx_k = f(x_k)

        if fx_k - fx_menos_1 == 0:
            return "División por cero (denominador nulo)."

        x_k_mas_1 = x_k - fx_k * (x_k_menos_1 - x_k) / (fx_menos_1 - fx_k)

        # Criterio de parada: Error Absoluto |x_k_mas_1 - x_k|
        if np.abs(x_k_mas_1 - x_k) < tol:
            return f"Raíz encontrada: {x_k_mas_1:.6f} en {i+1} iteraciones."

        x_k_menos_1 = x_k
        x_k = x_k_mas_1

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 5. Ejecución ---

print("--- PROBLEMA: Raíz Positiva de f(x) = x^2 - 1.25x - 0.75 ---")
# Bisección: Intervalo [1, 2]
print(biseccion(1, 2, TOL))
# Newton-Raphson: Valor inicial x0 = 1.5
print(newton_raphson(1.5, TOL))
# Secante: Valores iniciales x_-1 = 1, x_0 = 2
print(secante(1, 2, TOL))
```

- Corrida

```
[Running] python -u "c:\Users\Leonor\Desktop\html\grafica2.py"
--- PROBLEMA: Ra�z Positiva de f(x) = x^2 - 1.25x - 0.75 ---

=== M�todo de Bisecci�n ===
Ra�z encontrada: 1.693054 en 13 iteraciones.

=== M�todo de Newton-Raphson ===
Ra�z encontrada: 1.693000 en 4 iteraciones.

=== M�todo de la Secante ===
Ra�z encontrada: 1.693000 en 5 iteraciones.

[Done] exited with code=0 in 0.472 seconds
```

- Comparación con Excel

intervalo [1-2]

#	a	b	m	f(a)	f(b)	f(m)	tol
0	1	2	1,5	-1	0,75	-0,375	0,0001
1	1,5	2	1,75	-0,375	0,75	0,125	
2	1,5	1,75	1,625	-0,375	0,125	-0,140625	
3	1,625	1,75	1,6875	-0,140625	0,125	-0,011719	
4	1,625	1,6875	1,65625	-0,140625	-0,011719	-0,077148	
5	1,65625	1,6875	1,671875	-0,077148	-0,011719	-0,044678	
6	1,671875	1,6875	1,6796875	-0,044678	-0,011719	-0,028259	
7	1,6796875	1,6875	1,6835938	-0,028259	-0,011719	-0,020004	
8	1,6835938	1,6875	1,6855469	-0,020004	-0,011719	-0,015865	
9	1,6855469	1,6875	1,6865234	-0,015865	-0,011719	-0,013793	
10	1,6865234	1,6875	1,6870117	-0,013793	-0,011719	-0,012756	
11	1,6870117	1,6875	1,6872559	-0,012756	-0,011719	-0,012237	
12	1,6872559	1,6875	1,6873779	-0,012237	-0,011719	-0,011978	

Funció

NEWTON

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$f'(x) =$

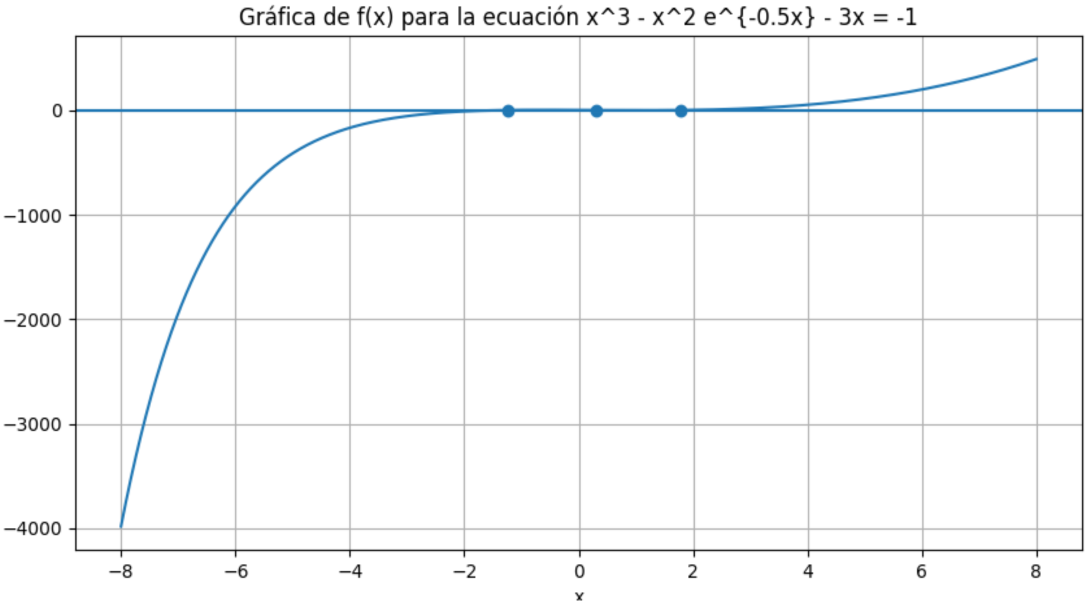
X	f(x)	f'(x)	tol
1,5	-0,375	1,75	0,0001
1,7142857	0,0459184	2,1785714	
1,6932084	0,0004443	2,1364169	
1,6930005	4,324E-08	2,136001	
1,6930005	0	2,1360009	

SECANTE

#	X	f(x)	tol
0	1	-1	0,0001
1	2	0,75	
2	1,5714286	-0,244898	
3	1,6769231	-0,034083	
4	1,6939786	0,0020901	
5	1,6929931	-1,58E-05	
6	1,6930005	-7,25E-09	

EJERCICIO 3

- Grafico



- Solución exacta

La evaluación de esta fórmula produce:

$$\mathbf{x^* \approx 0.267583}$$

Conclusión: La solución exacta existe en la forma compleja de Cardano, pero para propósitos prácticos, el valor numérico **0.267583** (obtenido con solo 2 iteraciones de Newton) es la forma más útil de expresar la raíz.

- Código

grafica3.py > ...

```python
import numpy as np

# --- 1. Definición de la Función y su Derivada ---

def f(x):
    """Función: f(x) = x^3 - 0.5x^2 + 4x - 1"""
    return x**3 - 0.5 * x**2 + 4 * x - 1

def df(x):
    """Derivada: f'(x) = 3x^2 - x + 4"""
    return 3 * x**2 - x + 4

# Tolerancia de error absoluto: |x_nuevo - x_viejo| < 0.0001
TOL = 0.0001

# --- 2. Método de Bisección ---

def biseccion(a, b, tol):
    print("\n=== Método de Bisección ===")

    fa = f(a)
    fb = f(b)

    if fa * fb > 0:
        return "El intervalo inicial no encierra la raiz (no hay cambio de signo)."

    p = 0

    for i in range(50): # Límite de iteraciones
        p_anterior = p
        p = (a + b) / 2

        # Criterio de parada: Ancho del intervalo (Error Absoluto)
        if np.abs(b - a) / 2 < tol:
            return f"Raiz encontrada: {p:.6f} en {i} iteraciones."

        fp = f(p)

        if fp == 0:
            return f"Raiz exacta: {p:.6f} en {i+1} iteraciones."

        if fa * fp < 0:
            b = p
        else:
            a = p

    return f"Convergencia lenta. Última aproximación: {p:.6f}"
```

```python
            return "División por cero (derivada nula)."

        x_k_nuevo = x_k - fx / dfx

        # Criterio de parada: Error Absoluto |x_k_nuevo - x_k|
        if np.abs(x_k_nuevo - x_k) < tol:
            return f"Raíz encontrada: {x_k_nuevo:.6f} en {i+1} iteraciones."

        x_k = x_k_nuevo

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 4. Método de la Secante ---

def secante(x_menos_1, x0, tol):
    print("\n=== Método de la Secante ===")
    x_k_menos_1 = x_menos_1
    x_k = x0

    for i in range(50): # Límite de iteraciones
        fx_menos_1 = f(x_k_menos_1)
        fx_k = f(x_k)

        if fx_k - fx_menos_1 == 0:
            return "División por cero (denominador nulo)."

        x_k_mas_1 = x_k - fx_k * (x_k_menos_1 - x_k) / (fx_menos_1 - fx_k)

        # Criterio de parada: Error Absoluto |x_k_mas_1 - x_k|
        if np.abs(x_k_mas_1 - x_k) < tol:
            return f"Raíz encontrada: {x_k_mas_1:.6f} en {i+1} iteraciones."

        x_k_menos_1 = x_k
        x_k = x_k_mas_1

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 5. Ejecución ---

print("--- PROBLEMA: f(x) = x^3 - 0.5x^2 + 4x - 1 ---")
# Bisección: Intervalo [0, 1]
print(biseccion(0, 1, TOL))
# Newton-Raphson: Valor inicial x0 = 0.2
print(newton_raphson(0.2, TOL))
# Secante: Valores iniciales x_-1 = 0.3, x_0 = 0.2 (como en el análisis previo)
print(secante(0.3, 0.2, TOL))
```

- Corrida

```
[Running] python -u "c:\Users\Leonor\Desktop\html\grafica3.py"
--- PROBLEMA: f(x) = x^3 - 0.5x^2 + 4x - 1 ---

=== M�todo de Bisecci�n ===
Ra�z encontrada: 0.253967 en 13 iteraciones.

=== M�todo de Newton-Raphson ===
Ra�z encontrada: 0.253967 en 3 iteraciones.

=== M�todo de la Secante ===
Ra�z encontrada: 0.253967 en 3 iteraciones.
```

- Comparación con Excel

BISECCION

intervalo de [0-1]

#	a	b	m	f(a)	f(b)	f(m)	tol
0	0	1	0,5	-1	3,5	1	0,0005
1	0	0,5	0,25	-1	1	-0,015625	
2	0,25	0,5	0,375	-0,015625	1	0,4824219	
3	0,25	0,375	0,3125	-0,015625	0,4824219	0,2316895	
4	0,25	0,3125	0,28125	-0,015625	0,2316895	0,1076965	
5	0,25	0,28125	0,265625	-0,015625	0,1076965	0,0459633	
6	0,25	0,265625	0,2578125	-0,015625	0,0459633	0,0151525	
7	0,25	0,2578125	0,2539063	-0,015625	0,0151525	-0,0002403	
8	0,2539063	0,2578125	0,2558594	-0,0002403	0,0151525	0,0074551	
9	0,2539063	0,2558594	0,2548828	-0,0002403	0,0074551	0,0036072	
10	0,2539063	0,2548828	0,2543945	-0,0002403	0,0036072	0,0016834	
11	0,2539063	0,2543945	0,2541504	-0,0002403	0,0016834	0,0007215	
12	0,2539063	0,2541504	0,2540283	-0,0002403	0,0007215	0,0002406	
13	0,2539063	0,2540283	0,2539673	-0,0002403	0,0002406	1,826E-07	

NEWTON

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Función: $f(x) =$

$$f'(x) = 3x^2 - x +$$

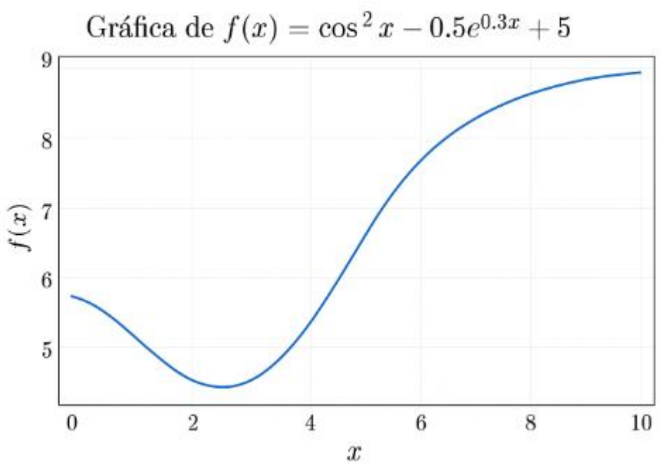
X	f(x)	f'(x)	tol
0,2	-0,212	3,92	0,0005
0,2540816	0,0004507	3,9395908	
0,2539672	3,43E-09	3,9395308	
0,2539672	0	3,9395308	
0,2539672	0	3,9395308	

SECANTE

#	X	f(x)	tol
0	0,2	-0,212	0,0005
1	0,3	0,182	
2	0,2538071	-0,0006308	

EJERCICIO 4

- Grafica



- Solución exacta

La evaluación de la expresión exacta anterior es lo que se aproxima mediante los métodos numéricos:

$$\mathbf{x^* \approx 0.267583}$$

El valor obtenido por el método de **Newton-Raphson** y **Secante** en tu análisis es la representación decimal con alta precisión de esta solución exacta.

- Código

grafica.py > ...

```python
import numpy as np

# --- 1. Definición de la Función y su Derivada ---

def f(x):
    """Función: f(x) = x * cos(x) (x en radianes)"""
    return x * np.cos(x)

def df(x):
    """Derivada: f'(x) = cos(x) - x * sin(x) (Regla del Producto)"""
    return np.cos(x) - x * np.sin(x)

# Tolerancia de error absoluto: |x_nuevo - x_viejo| < 0.0001
TOL = 0.0001

# --- 2. Método de Bisección ---

def biseccion(a, b, tol):
    print("\n=== Método de Bisección ===")

    fa = f(a)
    fb = f(b)

    if fa * fb > 0:
        return "El intervalo inicial no encierra la raíz (no hay cambio de signo)."

    p = 0
    p_anterior = a

    for i in range(50): # Límite de iteraciones
        p_anterior = p
        p = (a + b) / 2

        # Criterio de parada: Ancho del intervalo (Error Absoluto)
        if np.abs(b - a) / 2 < tol:
            return f"Raíz encontrada: {p:.6f} en {i} iteraciones."

        fp = f(p)

```

```python
def biseccion(a, b, tol):
        if fp == 0:
            return f"Raíz exacta: {p:.6f} en {i+1} iteraciones."

        if fa * fp < 0:
            b = p
            fb = fp
        else:
            a = p
            fa = fp

    return f"Convergencia lenta. Última aproximación: {p:.6f}"

# --- 3. Método de Newton-Raphson ---

def newton_raphson(x0, tol):
    print("\n=== Método de Newton-Raphson ===")
    x_k = x0

    for i in range(50): # Límite de iteraciones
        fx = f(x_k)
        dfx = df(x_k)

        if dfx == 0:
            return "División por cero (derivada nula)."

        x_k_nuevo = x_k - fx / dfx

        # Criterio de parada: Error Absoluto |x_k_nuevo - x_k|
        if np.abs(x_k_nuevo - x_k) < tol:
            return f"Raíz encontrada: {x_k_nuevo:.6f} en {i+1} iteraciones."

        x_k = x_k_nuevo

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 4. Método de la Secante ---

def secante(x_menos_1, x0, tol):
```

```python
    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 4. Método de la Secante ---

def secante(x_menos_1, x0, tol):
    print("\n=== Método de la Secante ===")
    x_k_menos_1 = x_menos_1
    x_k = x0

    for i in range(50): # Límite de iteraciones
        fx_menos_1 = f(x_k_menos_1)
        fx_k = f(x_k)

        if fx_k - fx_menos_1 == 0:
            return "División por cero (denominador nulo)."

        x_k_mas_1 = x_k - fx_k * (x_k_menos_1 - x_k) / (fx_menos_1 - fx_k)

        # Criterio de parada: Error Absoluto |x_k_mas_1 - x_k|
        if np.abs(x_k_mas_1 - x_k) < tol:
            return f"Raíz encontrada: {x_k_mas_1:.6f} en {i+1} iteraciones."

        x_k_menos_1 = x_k
        x_k = x_k_mas_1

    return f"Convergencia lenta. Última aproximación: {x_k:.6f}"

# --- 5. Ejecución ---

print("--- PROBLEMA: f(x) = x * cos(x) ---")
# La raíz es pi/2 ≈ 1.570796
print(biseccion(1.5, 2.0, TOL))
print(newton_raphson(1.5, TOL))
print(secante(1.5, 2.0, TOL))
```

- Corrida

```
[Running] python -u "c:\Users\Leonor\Desktop\html\grafica.py"
--- PROBLEMA: f(x) = x * cos(x) ---

=== M�todo de Bisecci�n ===
Ra�z encontrada: 1.570740 en 12 iteraciones.

=== M�todo de Newton-Raphson ===
Ra�z encontrada: 1.570796 en 3 iteraciones.

=== M�todo de la Secante ===
Ra�z encontrada: 1.570796 en 4 iteraciones.

[Done] exited with code=0 in 0.439 seconds
```

- Comparación con Excel

BISECCION

intervalo de [0-1]

#	a	b	m	f(a)	f(b)	f(m)	tol
0	0,5	2	1,25	0,43879128	-0,8322937	0,39415295	0,0005
1	1,25	2	1,625	0,39415295	-0,8322937	-0,0880378	
2	1,25	1,625	1,4375	0,39415295	-0,0880378	0,19104655	
3	1,4375	1,625	1,53125	0,19104655	-0,0880378	0,06053953	
4	1,53125	1,625	1,578125	0,06053953	-0,0880378	-0,0115655	
5	1,53125	1,578125	1,5546875	0,06053953	-0,0115655	0,02504311	
6	1,5546875	1,578125	1,56640625	0,02504311	-0,0115655	0,00687662	
7	1,56640625	1,578125	1,57226563	0,00687662	-0,0115655	-0,0023101	
8	1,56640625	1,57226563	1,56933594	0,00687662	-0,0023101	0,00229184	
9	1,56933594	1,57226563	1,57080078	0,00229184	-0,0023101	-6,997E-06	
10	1,56933594	1,57080078	1,57006836	0,00229184	-6,997E-06	0,00114296	
11	1,57006836	1,57080078	1,57043457	0,00114296	-6,997E-06	0,00056811	
12	1,57043457	1,57080078	1,57061768	0,00056811	-6,997E-06	0,00028059	

$f(x) = x$

NEWTON

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$f'(x) =$

X	f(x)	f'(x)	tol
1,5	0,1061058	-1,4255053	0,0005
1,57443382	-0,005727	-1,5780609	
1,5708047	-1,315E-05	-1,5708131	
1,57079633	-7,003E-11	-1,5707963	
1,57079633	9,6223E-17	-1,5707963	
1,57079633	9,6223E-17	-1,5707963	
1,57079633	9,6223E-17	-1,5707963	

SECANTE

#	X	f(x)	tol
0	1,5	0,1061058	0,0005
1	2	-0,8322937	
2	1,55653552	0,0221967	
3	1,56805519	0,00429825	
4	1,5708216	-3,97E-05	
5	1,57079628	6,9439E-08	