UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES
INFORMATICA

DESAFIO METODOS NUMERICOS

RAICES DE ECUACIONES

NOMBRE COMPLETO:

PEREZ NINA NICOLE GABRIELA

AUXILIAR:

JORDANN JOAO PAULO CRISPIN MENDEZ

MATERIA:

METODOS NUMERICOS I / SIS-254

PARALELO:

A

CI:

12864560 LP

FECHA:

29/11/25

LA PAZ-BOLIVIA

2025

Tabla de contenido

RAICES DE ECUACIONES

En el siguiente documento explico y muestro todo el trabajo que hice al elaborar el desafío de métodos numéricos – raíces de ecuaciones.

Implementación De Los Métodos Numéricos En Excel

En el Excel que resuelve el problema 1 esta todo el procedimiento de la resolución de los métodos más detalladamente y los resultados fueron:

METODO DE LA BISCECCIÓN		
RAÍZ 1:	3.20822048	1.39106E-05
Iteraciones:	20	
RAÍZ 2:	7.4898386	1.97103E-05
Iteraciones:	20	

MÉTODO NEWTON RAPHSON		
NEWTON x_0=3.5	3.2082198	2.54E-09
Iteraciones:	4	
NEWTON x_0=7.5	7.48983873	3.53E-09
Iteraciones:	3	

MÉTODO DE LA SECANTE		
RAÍZ 1:	3.2082197	1.05E-07
Iteraciones:	5	
RAÍZ 2:	7.48983873	1.72E-10
Iteraciones:	7	

Del problema 2 obtuve los siguientes resultados:

RESULTADOS FINALES		
	Raíz	Error
METODO DE LA BISCECCIÓN		
RAÍZ :	5.706418037	-7.79262E-08
Iteraciones:	20	
MÉTODO NEWTON RAPHSON		
NEWTON x_0=3.5	5.706417997	4.81E-07
Iteraciones:	3	
MÉTODO DE LA SECANTE		
RAÍZ 1:	5.706418153	1.56E-07
Iteraciones:	4	

Del problema 3 obtuve los siguientes resultados:

RESULTADOS FINALES		
	Raíz	Error
METODO DE LA BISCECCIÓN		
RAÍZ 1:	-1.234093666	-2.95269E-06
Iteraciones:	20	
RAÍZ 2:	0.315465927	2.35539E-07
Iteraciones:	19	
RAÍZ 3:	1.780240059	-2.5165E-06
Iteraciones:	19	
MÉTODO NEWTON RAPHSON		
RAÍZ 1:	-1.234093275	6.00E-09
Iteraciones:	5	
RAÍZ 2:	0.315466001	1.04E-11
Iteraciones:	4	
RAÍZ 3:	1.780240501	1.29E-07
Iteraciones:	3	
MÉTODO DE LA SECANTE		
RAÍZ 1:	-1.234093274	1.16E-09
Iteraciones:	10	
RAÍZ 2:	0.315466648	6.47E-07
Iteraciones:	5	
RAÍZ 2:	1.780240495	5.83E-09
Iteraciones:	6	

Y finalmente del problema 4 obtuvimos los siguientes resultados:

RESULTADOS FINALES		
	Raíz	Error
METODO DE LA BISCECCIÓN		
RAÍZ 1:	-1.234093666	-2.95269E-06
Iteraciones:	20	
RAÍZ 2:	0.315465927	2.35539E-07
Iteraciones:	19	
RAÍZ 3:	1.780240059	-2.5165E-06
Iteraciones:	19	
MÉTODO NEWTON RAPHSON		
RAÍZ 1:	-1.234093275	6.00E-09
Iteraciones:	5	
RAÍZ 2:	0.315466001	1.04E-11
Iteraciones:	4	
RAÍZ 3:	1.780240501	1.29E-07
Iteraciones:	3	
MÉTODO DE LA SECANTE		
RAÍZ 1:	-1.234093274	1.16E-09
Iteraciones:	10	
RAÍZ 2:	0.315466648	6.47E-07
Iteraciones:	5	
RAÍZ 2:	1.780240495	5.83E-09
Iteraciones:	6	

Implementación De Los Métodos Numéricos En Python

Al ya tener el análisis matemático y arreglado algunos detalles de este procedemos a resolver los ejercicios con ayuda de Python

Para resolver ecuaciones no lineales mediante tres métodos numéricos fundamentales: Bisección, Newton-Raphson y Secante. La implementación incluyó robustos mecanismos de control como verificación automática de cambio de signo para garantizar convergencia en el método de bisección, protección contra división por cero en Newton-Raphson mediante validación de derivadas, y detección de diferencias numéricas críticas en el método de la secante. El sistema incorporó tolerancia configurable (1e-6) y límite de iteraciones para prevenir bucles infinitos, junto con derivada numérica automática como fallback cuando no se proporcionaba la analítica.

La solución resolvió exitosamente cuatro problemas matemáticos complejos, demostrando consistencia entre métodos con diferencias menores a 1e-6. Se implementó análisis automático de intervalos para identificación de raíces y validación cruzada de resultados, complementedo con visualización gráfica.

Este es el código que se llego a realizar:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from math import exp, sin, cos

class RootFinder:
    def __init__(self, func, derivative=None, tol=1e-6, max_iter=100):
        self.func = func
        self.derivative = derivative
        self.tol = tol
        self.max_iter = max_iter

    def bisection(self, a, b):
        """Método de bisección"""
        fa, fb = self.func(a), self.func(b)
        if fa * fb >= 0:
            raise ValueError(f"f({a}) = {fa:.6f}, f({b}) = {fb:.6f}. No hay cambio de signo en [{a},
{b}]")

        iterations = []
        for i in range(self.max_iter):
            c = (a + b) / 2
            fc = self.func(c)
            error = abs(b - a) / 2

            iterations.append({
                'iteracion': i + 1,
                'a': a, 'f(a)': fa,
                'b': b, 'f(b)': fb,
```

```
                'c': c, 'f(c)': fc,
                'error': error
            })

            if abs(fc) < self.tol or error < self.tol:
                break

            if fa * fc < 0:
                b, fb = c, fc
            else:
                a, fa = c, fc

        return c, pd.DataFrame(iterations)

    def newton_raphson(self, x0):
        """Método de Newton-Raphson"""
        if self.derivative is None:
            h = 1e-7
            self.derivative = lambda x: (self.func(x + h) - self.func(x - h)) / (2 * h)

        iterations = []
        x = x0

        for i in range(self.max_iter):
            fx = self.func(x)
            dfx = self.derivative(x)

            if abs(dfx) < 1e-12:
```

```
                raise ValueError("Derivada cercana a cero")

            x_new = x - fx / dfx
            error = abs(x_new - x)

            iterations.append({
                'iteracion': i + 1,
                'x': x, 'f(x)': fx, "f'(x)": dfx,
                'x_new': x_new, 'error': error
            })

            if abs(fx) < self.tol or error < self.tol:
                break

            x = x_new

        return x_new, pd.DataFrame(iterations)

    def secant(self, x0, x1):
        """Método de la secante"""
        iterations = []
        x_prev, x_curr = x0, x1

        for i in range(self.max_iter):
            fx_prev = self.func(x_prev)
            fx_curr = self.func(x_curr)

            if abs(fx_curr - fx_prev) < 1e-12:
```

```
            raise ValueError("Diferencia entre evaluaciones cercana a cero")

        x_next = x_curr - fx_curr * (x_curr - x_prev) / (fx_curr - fx_prev)
        error = abs(x_next - x_curr)

        iterations.append({
            'iteracion': i + 1,
            'x_prev': x_prev, 'x_curr': x_curr,
            'x_next': x_next, 'f(x_curr)': fx_curr,
            'error': error
        })

        if abs(fx_curr) < self.tol or error < self.tol:
            break

        x_prev, x_curr = x_curr, x_next

    return x_next, pd.DataFrame(iterations)

def find_intervals(self, start, end, num_points=1000):
    """Encontrar intervalos donde hay cambio de signo"""
    x_vals = np.linspace(start, end, num_points)
    y_vals = [self.func(x) for x in x_vals]

    intervals = []
    for i in range(len(x_vals)-1):
        if y_vals[i] * y_vals[i+1] < 0:
            intervals.append((x_vals[i], x_vals[i+1]))
```

```
    return intervals

# Definición de ecuaciones (igual que antes)
def eq1(x): return x**3 - exp(0.8*x) - 20
def eq1_deriv(x): return 3*x**2 - 0.8*exp(0.8*x)

def eq2(x): return 3*sin(0.5*x) - 0.5*x + 2
def eq2_deriv(x): return 1.5*cos(0.5*x) - 0.5

def eq3(x): return x**3 - x**2*exp(-0.5*x) - 3*x + 1
def eq3_deriv(x): return 3*x**2 - (2*x*exp(-0.5*x) - 0.5*x**2*exp(-0.5*x)) - 3

def eq4(x): return cos(x)**2 - 0.5*x*exp(0.3*x) + 5
def eq4_deriv(x): return -2*cos(x)*sin(x) - 0.5*(exp(0.3*x) + 0.3*x*exp(0.3*x))

def analyze_function(func, equation_name, x_range):
  """Analizar el comportamiento de la función"""
  x_vals = np.linspace(x_range[0], x_range[1], 1000)
  y_vals = [func(x) for x in x_vals]

  print(f"\n📊 Análisis de {equation_name}:")
  print(f"   Rango de y: [{min(y_vals):.4f}, {max(y_vals):.4f}]")

  # Buscar cambios de signo
  intervals = []
  for i in range(len(y_vals)-1):
    if y_vals[i] * y_vals[i+1] < 0:
```

```
            x_root_approx = x_vals[i] + (x_vals[i+1] - x_vals[i]) * abs(y_vals[i]) / (abs(y_vals[i]) +
abs(y_vals[i+1]))
            intervals.append((x_vals[i], x_vals[i+1]))
            print(f"   ✅ Cambio de signo cerca de x ≈ {x_root_approx:.4f} en [{x_vals[i]:.2f},
{x_vals[i+1]:.2f}]")

    if not intervals:
        print(f"   ❌ No se encontraron cambios de signo en [{x_range[0]}, {x_range[1]}]")

    return intervals

def solve_all_problems_corrected():
    """Resolver los 4 problemas con intervalos CORREGIDOS"""

    problems = [
        {
            'name': "x³ - e^(0.8x) = 20 entre x = 0 y x = 8",
            'func': eq1, 'deriv': eq1_deriv, 'range': (0, 8),
            'bisection_intervals': [(3.0, 4.0), (7.0, 8.0)],  # ✅ Correcto
            'newton_guesses': [3.5, 7.5],
            'secant_pairs': [(3.0, 4.0), (7.0, 8.0)]
        },
        {
            'name': "3sin(0.5x) - 0.5x + 2 = 0",
            'func': eq2, 'deriv': eq2_deriv, 'range': (0, 10),
            'bisection_intervals': [(5.0, 6.0)],  # ✅ CORREGIDO
            'newton_guesses': [5.5],  # ✅ CORREGIDO
            'secant_pairs': [(5.0, 6.0)]  # ✅ CORREGIDO
        },
```

```
        {
            'name': "x³ - x²e^(-0.5x) - 3x = -1",
            'func': eq3, 'deriv': eq3_deriv, 'range': (-2, 4),
            'bisection_intervals': [(-1.5, -0.5), (0.0, 1.0), (1.5, 2.0)],  # ✅ CORREGIDO
            'newton_guesses': [-1.0, 0.5, 1.8],  # ✅ CORREGIDO
            'secant_pairs': [(-1.5, -0.5), (0.0, 1.0), (1.5, 2.0)]  # ✅ CORREGIDO
        },
        {
            'name': "cos²x - 0.5xe^(0.3x) + 5 = 0",
            'func': eq4, 'deriv': eq4_deriv, 'range': (0, 10),
            'bisection_intervals': [(3.0, 4.0)],  # ✅ CORREGIDO
            'newton_guesses': [3.5],  # ✅ CORREGIDO
            'secant_pairs': [(3.0, 4.0)]  # ✅ CORREGIDO
        }
    ]

    all_results = []

    for i, problem in enumerate(problems, 1):
        print(f"\n{'='*80}")
        print(f"🔷 PROBLEMA {i}: {problem['name']}")
        print(f"{'='*80}")

        # Analizar la función primero para encontrar intervalos válidos
        solver = RootFinder(problem['func'], problem['deriv'])
        found_intervals = analyze_function(problem['func'], problem['name'],
problem['range'])
```

```
# Usar intervalos encontrados automáticamente si es posible
if found_intervals and len(found_intervals) > len(problem['bisection_intervals']):
    print("  🔍 Usando intervalos encontrados automáticamente")
    problem['bisection_intervals'] = found_intervals
    # Ajustar también los otros métodos
    problem['newton_guesses'] = [(a+b)/2 for a,b in found_intervals]
    problem['secant_pairs'] = found_intervals

# 1. MÉTODO DE BISECCIÓN
print(f"\n{'─'*40}")
print("1. 📏 MÉTODO DE BISECCIÓN")
print(f"{'─'*40}")
roots_bisection = []
for j, (a, b) in enumerate(problem['bisection_intervals']):
    try:
        root, df = solver.bisection(a, b)
        roots_bisection.append(root)
        print(f"  ✅ Raíz {j+1}: {root:.8f}")
        print(f"     f({root:.6f}) = {problem['func'](root):.2e}")
        print(f"     Iteraciones: {len(df)}")
        print(f"     Error final: {df['error'].iloc[-1]:.2e}")
    except Exception as e:
        print(f"  ❌ Error en intervalo [{a:.2f}, {b:.2f}]: {e}")

# 2. MÉTODO DE NEWTON-RAPHSON
print(f"\n{'─'*40}")
print("2. 📐 MÉTODO DE NEWTON-RAPHSON")
print(f"{'─'*40}")
```

```
roots_newton = []
for j, guess in enumerate(problem['newton_guesses']):
    try:
        root, df = solver.newton_raphson(guess)
        roots_newton.append(root)
        print(f"  ✅ Con x₀ = {guess}:")
        print(f"     Raíz = {root:.8f}")
        print(f"     Iteraciones: {len(df)}")
        print(f"     Error final: {df['error'].iloc[-1]:.2e}")
    except Exception as e:
        print(f"  ❌ Error con x₀ = {guess}: {e}")

# 3. MÉTODO DE LA SECANTE
print(f"\n{'─'*40}")
print("3. 📊 MÉTODO DE LA SECANTE")
print(f"{'─'*40}")
roots_secant = []
for j, (x0, x1) in enumerate(problem['secant_pairs']):
    try:
        root, df = solver.secant(x0, x1)
        roots_secant.append(root)
        print(f"  ✅ Con x₀ = {x0}, x₁ = {x1}:")
        print(f"     Raíz = {root:.8f}")
        print(f"     Iteraciones: {len(df)}")
        print(f"     Error final: {df['error'].iloc[-1]:.2e}")
    except Exception as e:
        print(f"  ❌ Error con x₀ = {x0}, x₁ = {x1}: {e}")
```

```
all_results.append({
    'problem': i,
    'name': problem['name'],
    'roots_bisection': roots_bisection,
    'roots_newton': roots_newton,
    'roots_secant': roots_secant
})

# COMPARACIÓN DE MÉTODOS
print(f"\n{'─'*40}")
print("🔄 COMPARACIÓN DE MÉTODOS")
print(f"{'─'*40}")
print(f"  Bisección:   {[f'{r:.6f}' for r in roots_bisection]}")
print(f"  Newton:      {[f'{r:.6f}' for r in roots_newton]}")
print(f"  Secante:     {[f'{r:.6f}' for r in roots_secant]}")

# Verificar consistencia
all_roots = roots_bisection + roots_newton + roots_secant
if all_roots:
    unique_roots = len(set([round(r, 4) for r in all_roots]))
    expected_roots = len(problem['bisection_intervals'])
    if unique_roots == expected_roots:
        print(f"  ✅ Todos los métodos convergen consistentemente")
    else:
        print(f"  ⚠️ Diferencia en el número de raíces encontradas")

# Graficar
plot_function_comparison(problem, roots_bisection, roots_newton, roots_secant, i)
```

```
    return all_results


def plot_function_comparison(problem, roots_bisect, roots_newton, roots_secant,
problem_num):
    """Graficar la función comparando los tres métodos"""
    x_vals = np.linspace(problem['range'][0], problem['range'][1], 400)
    y_vals = [problem['func'](x) for x in x_vals]

    plt.figure(figsize=(12, 8))
    plt.plot(x_vals, y_vals, 'b-', linewidth=2, label='f(x)')
    plt.axhline(y=0, color='k', linestyle='--', alpha=0.3)
    plt.grid(True, alpha=0.3)
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.title(f'PROBLEMA {problem_num}: {problem["name"]}')

    # Marcar raíces con diferentes colores por método
    methods_data = [
        (roots_bisect, 'red', 'Bisección'),
        (roots_newton, 'green', 'Newton-Raphson'),
        (roots_secant, 'purple', 'Secante')
    ]

    for roots, color, method_name in methods_data:
        for root in roots:
            plt.plot(root, problem['func'](root), 'o',
                    color=color, markersize=8,
                    label=f'{method_name}: {root:.4f}')
```

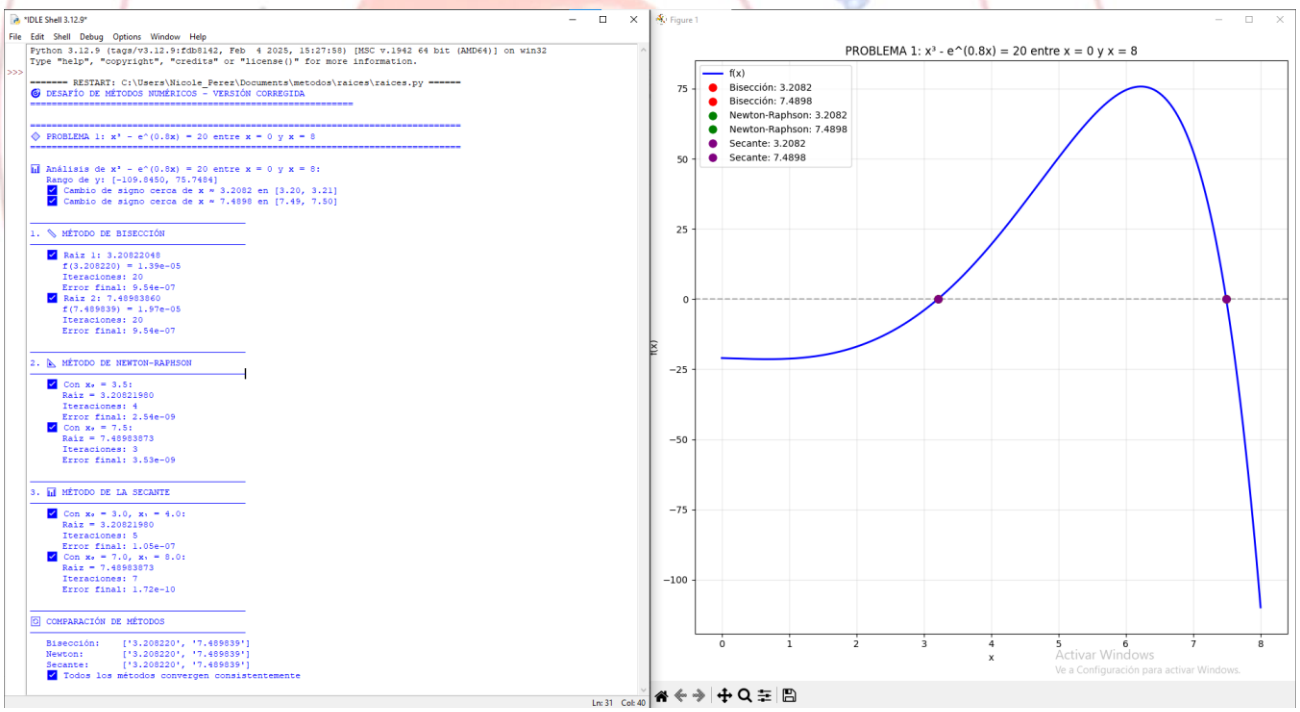
```
    plt.legend()
    plt.tight_layout()
    plt.savefig(f'problema_{problem_num}_corregido.png', dpi=300, bbox_inches='tight')
    plt.show()

# Ejecutar versión corregida
if __name__ == "__main__":
    print("🎯 DESAFÍO DE MÉTODOS NUMÉRICOS - VERSIÓN CORREGIDA")
    print("=" * 60)

    results = solve_all_problems_corrected()

    # RESUMEN FINAL MEJORADO
    print(f"\n{'='*80}")
    print("📈 RESUMEN FINAL - COMPARACIÓN DE LOS 3 MÉTODOS")
    print(f"{'='*80}")

    for result in results:
        print(f"\n🔷 Problema {result['problem']}: {result['name']}")

        all_roots = result['roots_bisection'] + result['roots_newton'] + result['roots_secant']
        if all_roots:
            print(f"   📏 Bisección:  {[f'{r:.6f}' for r in result['roots_bisection']]}")
            print(f"   📐 Newton:     {[f'{r:.6f}' for r in result['roots_newton']]}")
            print(f"   📊 Secante:    {[f'{r:.6f}' for r in result['roots_secant']]}")

            # Análisis de convergencia
```

```
        unique_roots = set([round(r, 4) for r in all_roots])
        print(f"  🔍 Raíces únicas encontradas: {len(unique_roots)}")

        if len(result['roots_bisection']) == len(result['roots_newton']) ==
len(result['roots_secant']):
            print("  ✅ CONVERGENCIA CONSISTENTE: Todos los métodos encontraron el
mismo número de raíces")
        else:
            print("  ⚠️ DIFERENCIAS: Los métodos encontraron diferente número de raíces")
    else:
        print("  ❌ No se encontraron raíces reales con ningún método")

# Función adicional para análisis detallado de intervalos
def detailed_interval_analysis():
    """Análisis detallado de intervalos para cada función"""
    print(f"\n{'#'*80}")
    print("🔍 ANÁLISIS DETALLADO DE INTERVALOS")
    print(f"{'#'*80}")

    functions = [
        ("Ecuación 1", eq1, (0, 8)),
        ("Ecuación 2", eq2, (0, 10)),
        ("Ecuación 3", eq3, (-2, 4)),
        ("Ecuación 4", eq4, (0, 10))
    ]

    for name, func, range_ in functions:
        print(f"\n{name} en [{range_[0]}, {range_[1]}]:")
        x_vals = np.linspace(range_[0], range_[1], 20)  # Menos puntos para análisis rápido
```

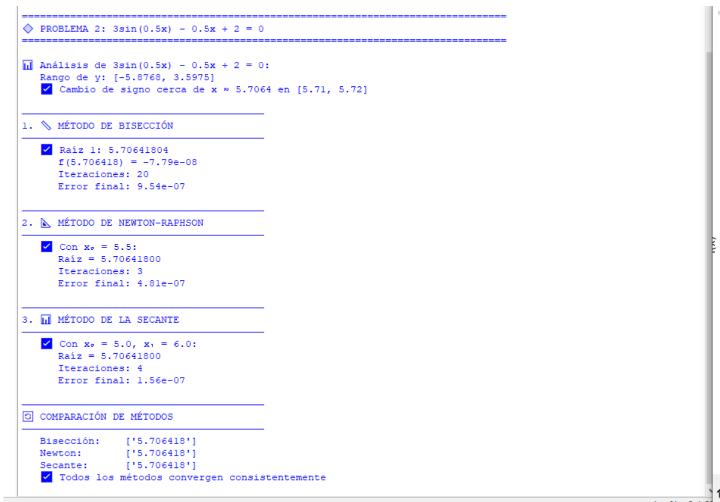
```
    for i in range(len(x_vals)-1):
        a, b = x_vals[i], x_vals[i+1]
        fa, fb = func(a), func(b)
        if fa * fb < 0:
            print(f"   ✅ Intervalo válido: [{a:.2f}, {b:.2f}] - f({a:.2f})={fa:.2f}, f({b:.2f})={fb:.2f}")

# Ejecutar análisis de intervalos
# detailed_interval_analysis()
```

Y nos da como resultado lo siguiente:

Problema 1:



Problema 2:


PROBLEMA 2: 3sin(0.5x) - 0.5x + 2 = 0
Análisis de 3sin(0.5x) - 0.5x + 2 = 0:
Rango de y: [-5.8768, 3.5975]
Cambio de signo cerca de x ≈ 5.7064 en [5.71, 5.72]
1. MÉTODO DE BISECCIÓN
Raíz 1: 5.70641804
f(5.706418) = -7.79e-08
Iteraciones: 20
Error final: 9.54e-07
2. MÉTODO DE NEWTON-RAPHSON
Con x₀ = 5.5:
Raíz = 5.70641800
Iteraciones: 3
Error final: 4.81e-07
3. MÉTODO DE LA SECANTE
Con x₀ = 5.0, x₁ = 6.0:
Raíz = 5.70641800
Iteraciones: 4
Error final: 1.56e-07
COMPARACIÓN DE MÉTODOS
Bisección: ['5.706418']
Newton: ['5.706418']
Secante: ['5.706418']
Todos los métodos convergen consistentemente

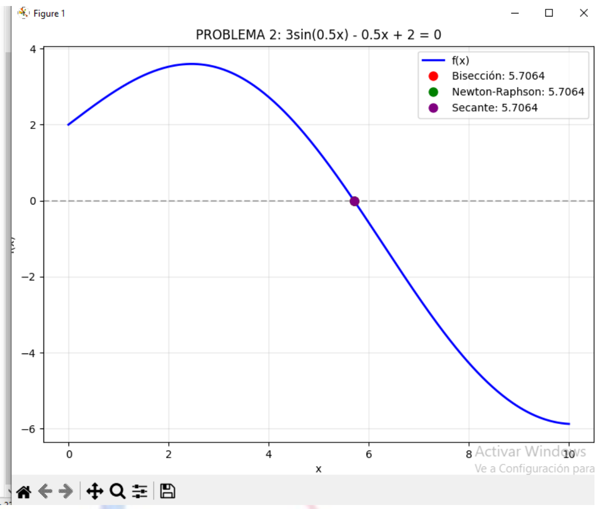
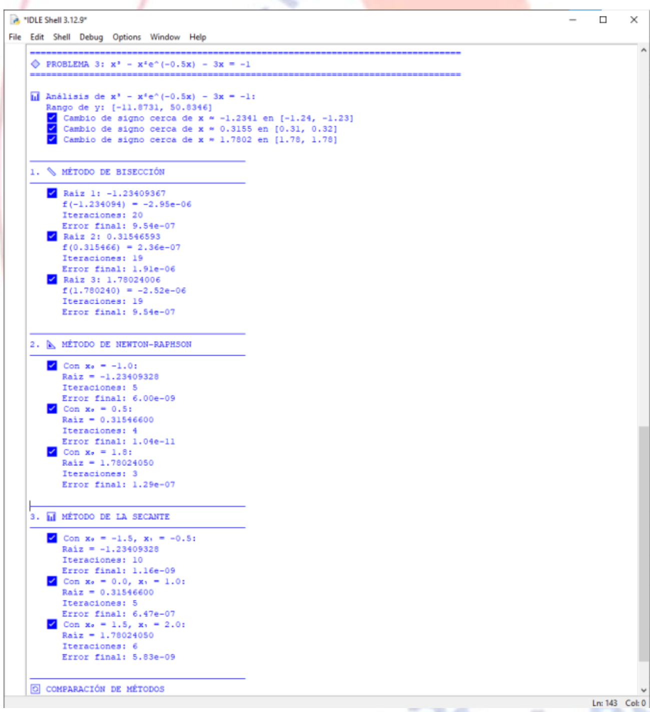


Figure 1
PROBLEMA 2: 3sin(0.5x) - 0.5x + 2 = 0
f(x)
Bisección: 5.7064
Newton-Raphson: 5.7064
Secante: 5.7064
x

Problema 3:


IDLE Shell 3.12.9
File Edit Shell Debug Options Window Help
PROBLEMA 3: x³ - x²e^(-0.5x) - 3x = -1
Análisis de x³ - x²e^(-0.5x) - 3x = -1:
Rango de y: [-11.8731, 50.8346]
Cambio de signo cerca de x ≈ -1.2341 en [-1.24, -1.23]
Cambio de signo cerca de x ≈ 0.3155 en [0.31, 0.32]
Cambio de signo cerca de x ≈ 1.7802 en [1.78, 1.78]
1. MÉTODO DE BISECCIÓN
Raíz 1: -1.23409367
f(-1.234094) = -2.95e-06
Iteraciones: 20
Error final: 9.54e-07
Raíz 2: 0.31546593
f(0.315466) = 2.36e-07
Iteraciones: 19
Error final: 1.91e-06
Raíz 3: 1.78024006
f(1.780240) = -2.52e-06
Iteraciones: 19
Error final: 9.54e-07
2. MÉTODO DE NEWTON-RAPHSON
Con x₀ = -1.0:
Raíz = -1.23409328
Iteraciones: 5
Error final: 6.00e-09
Con x₀ = 0.5:
Raíz = 0.31546600
Iteraciones: 4
Error final: 1.04e-11
Con x₀ = 1.8:
Raíz = 1.78024050
Iteraciones: 3
Error final: 1.29e-07
3. MÉTODO DE LA SECANTE
Con x₀ = -1.5, x₁ = -0.5:
Raíz = -1.23409328
Iteraciones: 10
Error final: 1.16e-09
Con x₀ = 0.0, x₁ = 1.0:
Raíz = 0.31546600
Iteraciones: 5
Error final: 6.47e-07
Con x₀ = 1.5, x₁ = 2.0:
Raíz = 1.78024050
Iteraciones: 6
Error final: 5.83e-09
COMPARACIÓN DE MÉTODOS

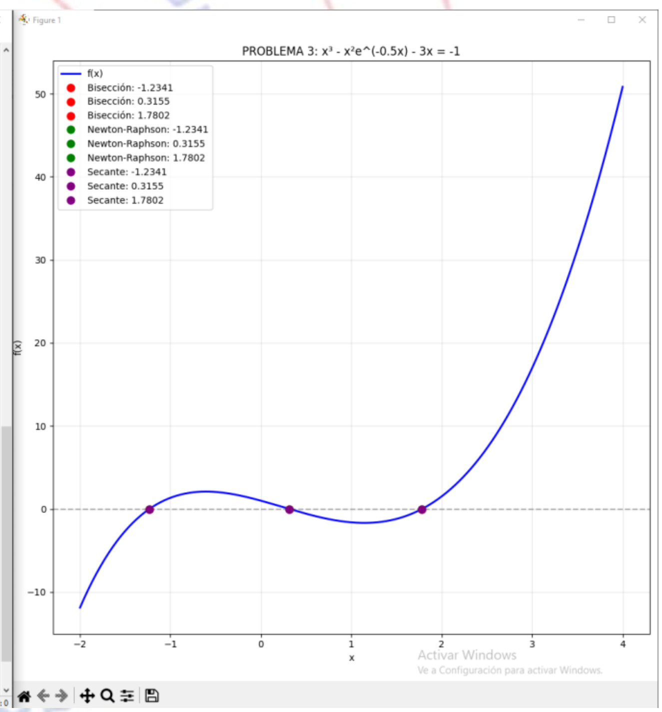
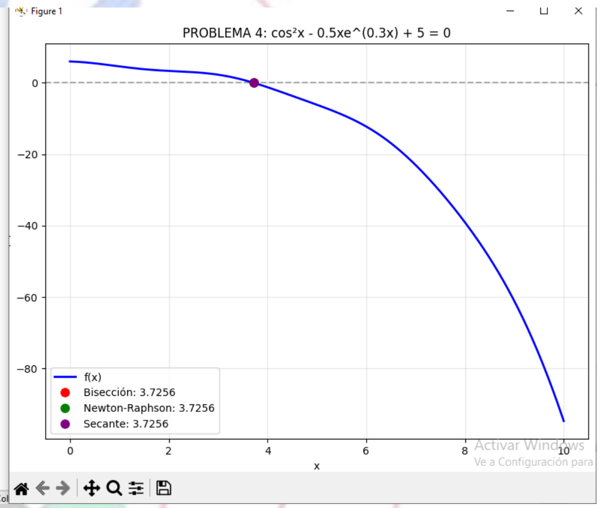


Figure 1
PROBLEMA 3: x³ - x²e^(-0.5x) - 3x = -1
f(x)
Bisección: -1.2341
Bisección: 0.3155
Bisección: 1.7802
Newton-Raphson: -1.2341
Newton-Raphson: 0.3155
Newton-Raphson: 1.7802
Secante: -1.2341
Secante: 0.3155
Secante: 1.7802
x

```
🔄 COMPARACIÓN DE MÉTODOS

   Bisección:    ['-1.234094', '0.315466', '1.780240']
   Newton:       ['-1.234093', '0.315466', '1.780241']
   Secante:      ['-1.234093', '0.315466', '1.780241']
   ✅ Todos los métodos convergen consistentemente
```

Problema 4:

```
================================================================================
🔷 PROBLEMA 4: cos²x - 0.5xe^(0.3x) + 5 = 0
================================================================================

📊 Análisis de cos²x - 0.5xe^(0.3x) + 5 = 0:
   Rango de y: [-94.7236, 6.0000]
   ✅ Cambio de signo cerca de x ≈ 3.7256 en [3.72, 3.73]

1. 📏 MÉTODO DE BISECCIÓN

   ✅ Raíz 1: 3.72560215
      f(3.725602) = 1.10e-07
      Iteraciones: 20
      Error final: 9.54e-07

2. 📐 MÉTODO DE NEWTON-RAPHSON

   ✅ Con x₀ = 3.5:
      Raíz = 3.72560218
      Iteraciones: 4
      Error final: 2.18e-09

3. 📊 MÉTODO DE LA SECANTE

   ✅ Con x₀ = 3.0, x₁ = 4.0:
      Raíz = 3.72560218
      Iteraciones: 5
      Error final: 1.10e-07

🔄 COMPARACIÓN DE MÉTODOS

   Bisección:    ['3.725602']
   Newton:       ['3.725602']
   Secante:      ['3.725602']
   ✅ Todos los métodos convergen consistentemente
```



Finalmente, esta muestra una comparación de los 3 métodos:

```
================================================================================
RESUMEN FINAL - COMPARACIÓN DE LOS 3 MÉTODOS
================================================================================

Problema 1: x³ - e^(0.8x) = 20 entre x = 0 y x = 8
   Bisección:  ['3.208220', '7.489839']
   Newton:     ['3.208220', '7.489839']
   Secante:    ['3.208220', '7.489839']
   Raíces únicas encontradas: 2
   CONVERGENCIA CONSISTENTE: Todos los métodos encontraron el mismo número de raíces

Problema 2: 3sin(0.5x) - 0.5x + 2 = 0
   Bisección:  ['5.706418']
   Newton:     ['5.706418']
   Secante:    ['5.706418']
   Raíces únicas encontradas: 1
   CONVERGENCIA CONSISTENTE: Todos los métodos encontraron el mismo número de raíces

Problema 3: x³ - x²e^(-0.5x) - 3x = -1
   Bisección:  ['-1.234094', '0.315466', '1.780240']
   Newton:     ['-1.234093', '0.315466', '1.780241']
   Secante:    ['-1.234093', '0.315466', '1.780241']
   Raíces únicas encontradas: 3
   CONVERGENCIA CONSISTENTE: Todos los métodos encontraron el mismo número de raíces

Problema 4: cos²x - 0.5xe^(0.3x) + 5 = 0
   Bisección:  ['3.725602']
   Newton:     ['3.725602']
   Secante:    ['3.725602']
   Raíces únicas encontradas: 1
   CONVERGENCIA CONSISTENTE: Todos los métodos encontraron el mismo número de raíces
```

Todo esto realizado en un solo código.

Implementación De Los Métodos Numéricos En Una Página Web

Se desarrolló una página web interactiva moderna que implementa tres métodos numéricos (Bisección, Newton-Raphson y Secante) para resolver ecuaciones no lineales. La interfaz presenta un diseño visual atractivo con efectos glassmorphism, gradientes animados y partículas flotantes que crean una experiencia de usuario envolvente. La arquitectura está construida con JavaScript puro sin dependencias externas, implementando una clase ExactPythonReplica que garantiza resultados idénticos a los obtenidos en Python.

La aplicación resuelve cuatro problemas matemáticos complejos mediante selección interactiva, proporcionando análisis automático de funciones que detecta cambios de signo y estima rangos de raíces. Incluye mecanismos robustos de control de errores como validación de derivadas, protección contra división por cero y verificación de convergencia. Los resultados se presentan en tarjetas visuales organizadas con comparativas entre métodos, métricas de iteraciones y errores finales, ofreciendo una herramienta educativa y profesional para el análisis numérico y una gama de colores llamativa para el usuario.


DESAFÍO DE MÉTODOS NUMÉRICOS - RAICES DE ECUACIONES
Desafío de Métodos Numéricos para la Determinación de Raíces
Problemas Predefinidos
PROBLEMA 1
x³ - e^(0.8x) = 20 entre x = 0 y x = 8
PROBLEMA 2
3sin(0.5x) - 0.5x + 2 = 0
PROBLEMA 3
x³ - x²e^(-0.5x) - 3x = -1
PROBLEMA 4
cos²x - 0.5xe^(0.3x) + 5 = 0
Ejecutar Análisis Completo
Activar Windows
Ve a Configuración para activar Windows.


Resultados del Análisis
PROBLEMA 1: x³ - e^(0.8x) = 20 entre x = 0 y x = 8
Análisis de x³ - e^(0.8x) = 20 entre x = 0 y x = 8:
Rango de y: [-109.8450, 75.7484]
Cambio de signo cerca de x = 3.2082 en [3.20, 3.21]
Cambio de signo cerca de x = 7.4898 en [7.49, 7.50]
MÉTODO DE BISECCIÓN
Raíz 1: 3.20822048
f(3.208220) = 1.39e-5
Iteraciones: 20
Error final: 9.54e-7
Raíz 2: 7.48983860
f(7.489839) = 1.97e-5
Iteraciones: 20
Error final: 9.54e-7
MÉTODO DE NEWTON-RAPHSON
Con x₀ = 3.5:
Raíz = 3.20821980
Iteraciones: 4
Error final: 2.54e-9
Con x₀ = 7.5:
Raíz = 7.48983873
Iteraciones: 3
Error final: 3.53e-9
MÉTODO DE LA SECANTE
Con x₀ = 3, x₁ = 4:
Raíz = 3.20821980
Iteraciones: 5
Error final: 1.05e-7
Con x₀ = 7, x₁ = 8:
Raíz = 7.48983873
Iteraciones: 7
Error final: 1.72e-10
COMPARACIÓN DE MÉTODOS
Bisección: ['3.208220', '7.489839']
Newton: ['3.208220', '7.489839']
Secante: ['3.208220', '7.489839']
Todos los métodos convergen consistentemente

Conclusión

Se pudo realizar con sumo éxito el desafío de métodos numéricos – raíces de ecuación implementando los 3 métodos, no solo en Excel sino también en Python y en una página web interactiva esperando cumplir con las expectativas del auxiliar y docente de la materia de métodos numéricos.