Trabalho Prático 2: Métodos Numéricos (M2039)

Resolução de Equações Não Lineares

Autores:

- Eduardo Moura - up202406710
- Filipe Zheng - up202406753
- João Azevedo - up202404140
- Manuel Mota - up202403543

Introdução

Este notebook apresenta a resolução dos problemas propostos no Trabalho Prático 2. Os métodos utilizados baseiam-se na matéria exposta nos slides da unidade curricular, nomeadamente o **Método das Bisseções Sucessivas** e o **Método de Newton** para a resolução de equações não lineares, bem como a análise de convergência do **Método Iterativo Simples (ou do Ponto Fixo)**.

Exercício 1: Resolução de $F(x) = \text{sen}(x^2) + 1.1 - e^{-x} = 0$

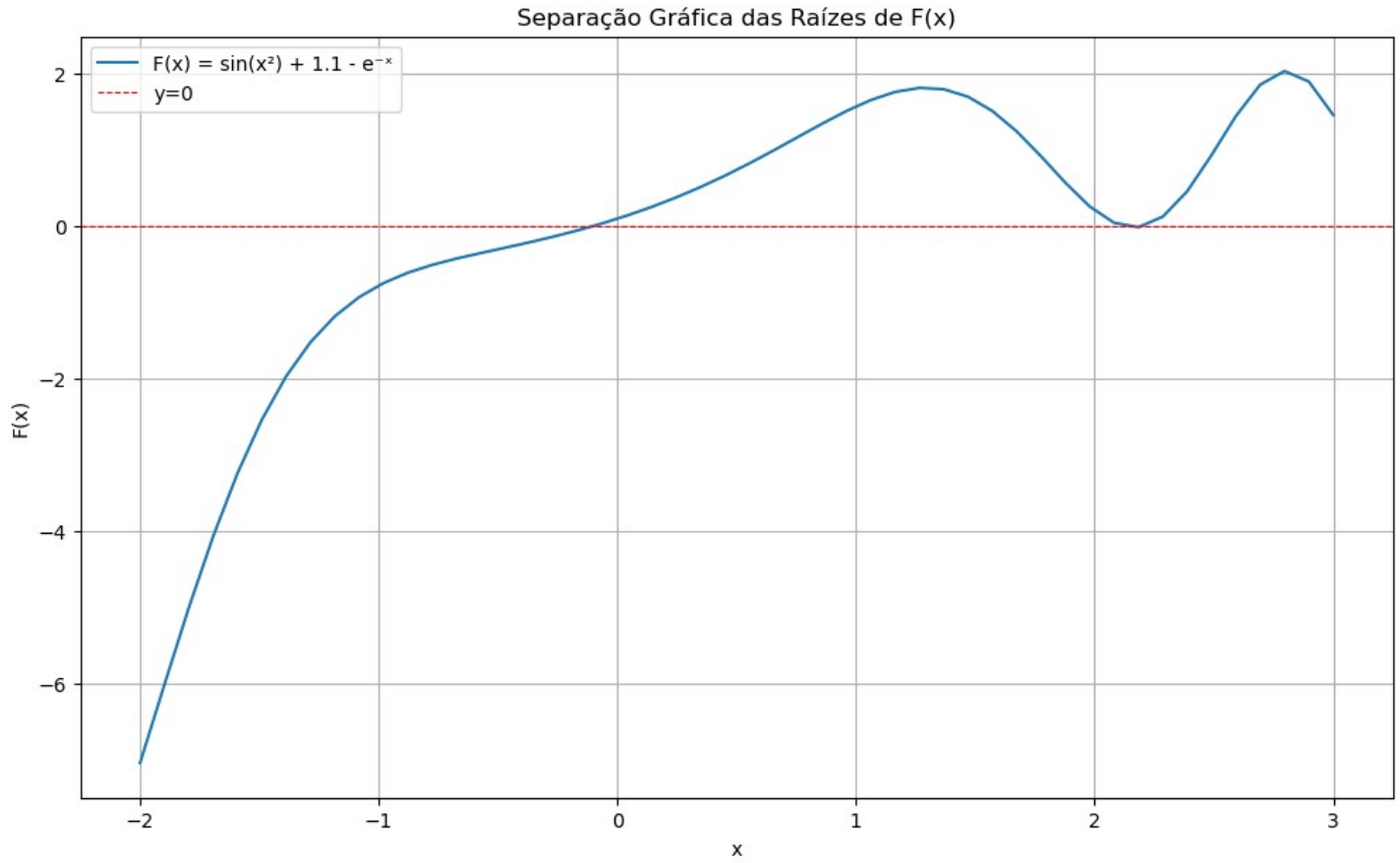
Neste exercício, pretendemos encontrar um valor aproximado para um zero da função F(x) dada.

(a) Separação Gráfica das Raízes e Determinação do Intervalo I

Objetivo: Separar graficamente as raízes e determinar um intervalo `I` de amplitude 10^{-1} que contenha a menor delas.

O método gráfico para a separação de raízes consiste em determinar os pontos de interseção do gráfico de F(x) com o eixo y=0. Uma raiz existe num intervalo `[a, b]` se `F(x)` for contínua e `F(a) * F(b) < 0` (uma consequência do Teorema de Bolzano).

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

def F(x):
    return np.sin(x**2) + 1.1 - np.exp(-x)

x_vals = np.linspace(-2, 3)
y_vals = F(x_vals)
plt.figure(figsize=(12, 7))
plt.plot(x_vals, y_vals, label='F(x) = sin(x²) + 1.1 - e⁻ˣ')
plt.axhline(0, color='red', linestyle='--', linewidth=0.8, label='y=0')
plt.title('Separação Gráfica das Raízes de F(x)')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.grid(True)
plt.legend()
plt.show()
```



Análise do Gráfico:

O gráfico mostra que a função interseta o eixo y=0 em dois pontos, indicando a existência de, pelo menos, duas raízes reais. Uma parece estar próxima de x=2.2 e a outra, a menor, parece estar ligeiramente à esquerda de x=0. O exercício pede para nos concentrarmos na menor delas.

Para os valores $x < -1$, como e^{-x} é decrescente, sabemos que:

$e^{-x} > e$

$<=> -e^{-x} < -e$

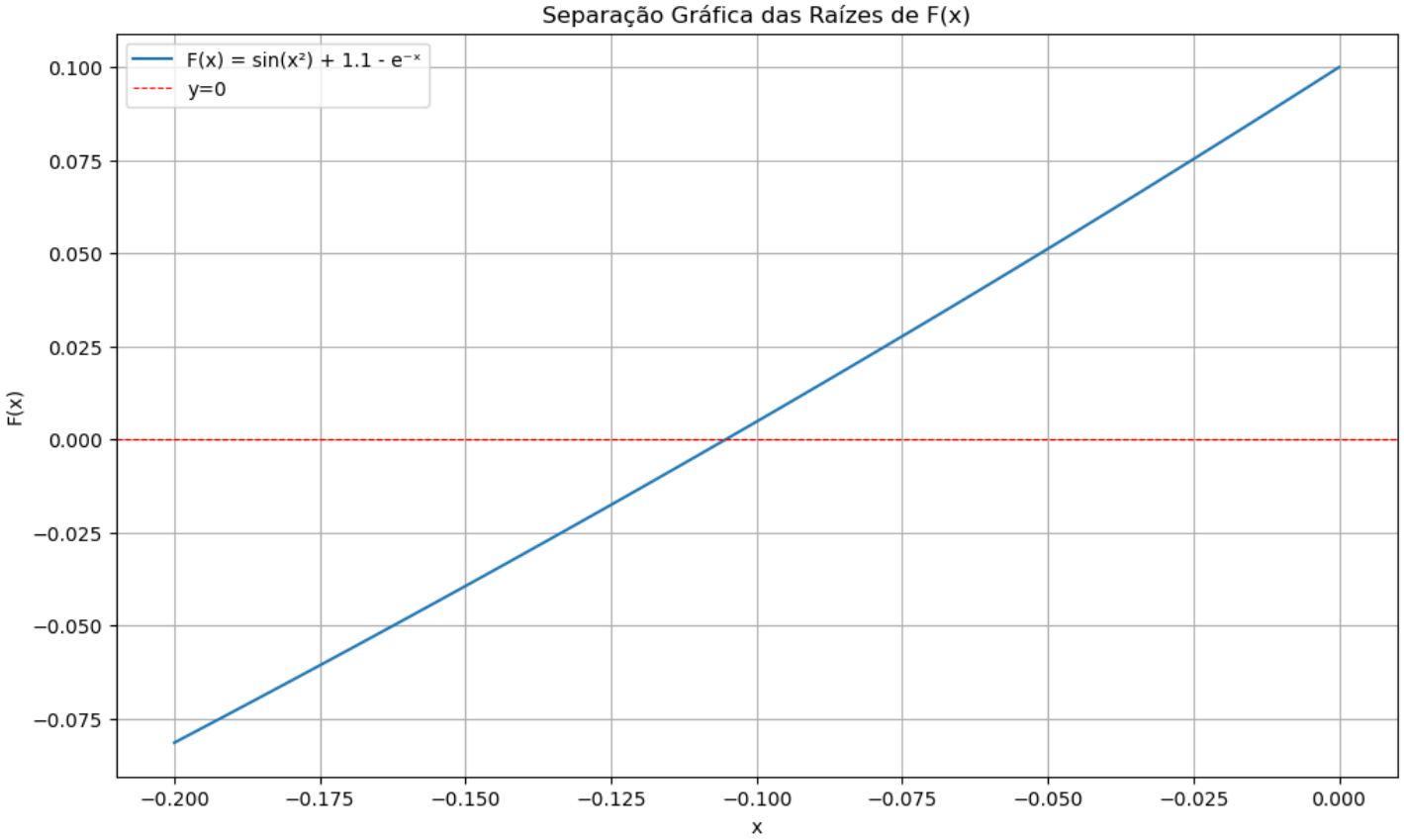
$<=> \sin(x^2) - e^{-x} < 1 - e$; Como $-1 \leq \sin(x^2) \leq 1$

$<=> \sin(x^2) + 1.1 - e^{-x} < 2.1 - e \approx -0.62 < 0$

$<=> F(x) < 0$

Logo sabemos que não existem raizes antes de x=-1. A menor das raizes é então aquela ligeiramente à esquerda de x=0 visto no gráfico.

```
x_vals = np.linspace(-0.2, 0)
y_vals = F(x_vals)

plt.figure(figsize=(12, 7))
plt.plot(x_vals, y_vals, label='F(x) = sin(x²) + 1.1 - e⁻ˣ')
plt.axhline(0, color='red', linestyle='--', linewidth=0.8, label='y=0')
plt.title('Separação Gráfica das Raízes de F(x)')
plt.xlabel('x')
plt.ylabel('F(x)')
plt.grid(True)
plt.legend()
plt.show()
```



Observamos que F(-0.150) < 0 e F(-0.050) > 0, logo F(-0.150)F(-0.050) < 0. Como a função F(x) é uma soma de funções contínuas (seno composta com polinomial, polinomial e exponencial), ela é contínua em todo o seu domínio.

Conclusão para (a):

As condições do Teorema de Bolzano estão satisfeitas. Portanto, existe pelo menos uma raiz no intervalo [-0.150, -0.050]. A amplitude deste intervalo é $0.1 = 10^{-1}$. O intervalo I escolhido é I = [-0.150, -0.050]

(b) Resolução Numérica da Raiz em I

Objetivo: Calcular um valor aproximado da raiz em `I` com erro absoluto majorado inferior a $\varepsilon = 5 \times 10^{-9}$.

i. Usando o Método das Bisseções Sucessivas

O método aplica-se pois, como vimos:

1. `F(x)` é contínua em `I = [-0.150, -0.050]`.
2. `F(-0.15) * F(-0.05) < 0`.

O erro absoluto após `n` iterações é majorado por $|\Delta x_n| \leq |b - a| / 2^n$. Para garantir que o erro seja inferior a ε, precisamos de um número `n` de iterações tal que: $(0.1) / 2^n \leq 5 \times 10^{-9} \iff 2^n \geq 0.1 / (5 \times 10^{-9}) \iff n \geq \log_2(2 \times 10^7)$ Calculando, $n \geq 24.25$. Portanto, necessitaremos de **pelo menos 25 iterações**.

```
a, b = -0.150, -0.050
epsilon = 5e-9

def bissecao(f, a, b, tol):
    if f(a) * f(b) >= 0:
        raise "Condição F(a) * F(b) < 0 não satisfeita."

    n_iter = 0
    vfa = F(a)
```

```
    ponto_medio = a
    erro_majorado = abs(b-a)
    while erro_majorado > tol:
        ponto_medio = (a + b) / 2
        f_medio = f(ponto_medio)

        if f_medio == 0:
            return ponto_medio, n_iter + 1

        if vfa * f_medio < 0:
            b = ponto_medio
        else:
            a = ponto_medio
        erro_majorado /= 2
        n_iter += 1
    return ponto_medio, erro_majorado, n_iter

# Execução do método
raiz_bis, erro_bis, iter_bis = bissecao(F, a, b, epsilon)

def round_error(error:float)->str:
  error = f"{error:e}"
  error_rounded = f"{math.ceil(float(error[:-4])*10)/10}{error[-4:]}"
  return error_rounded

def format_result(raiz,erro):
  if isinstance(erro,float): erro = round_error(erro)
  return f"{raiz:0.0{int(erro[-2:])+1}f} ± {erro}"

erro_bis = round_error(erro_bis)

print("--- Método das Bisseções Sucessivas ---")
print(f"Intervalo inicial (I): [{a}, {b}]")
print(f"Tolerância (ε): {epsilon}")
print(f"Raiz aproximada: {raiz_bis:.15f}")
print(f"Erro absoluto majorado: {erro_bis}")
print(f"Número de iterações efetuadas: {iter_bis}")
print(f"Raiz: X = {format_result(raiz_bis,erro_bis)}")
```

```
--- Método das Bisseções Sucessivas ---
Intervalo inicial (I): [-0.15, -0.05]
Tolerância (ε): 5e-09
Raiz aproximada: -0.105348852276802
Erro absoluto majorado: 3.0e-09
Número de iterações efetuadas: 25
Raiz: X = -0.1053488523 ± 3.0e-09
```

Resultados e Análise:

O método convergiu para uma solução dentro da tolerância especificada. O número de iterações (25) está de acordo com o nosso cálculo teórico.

ii. Usando o Método de Newton

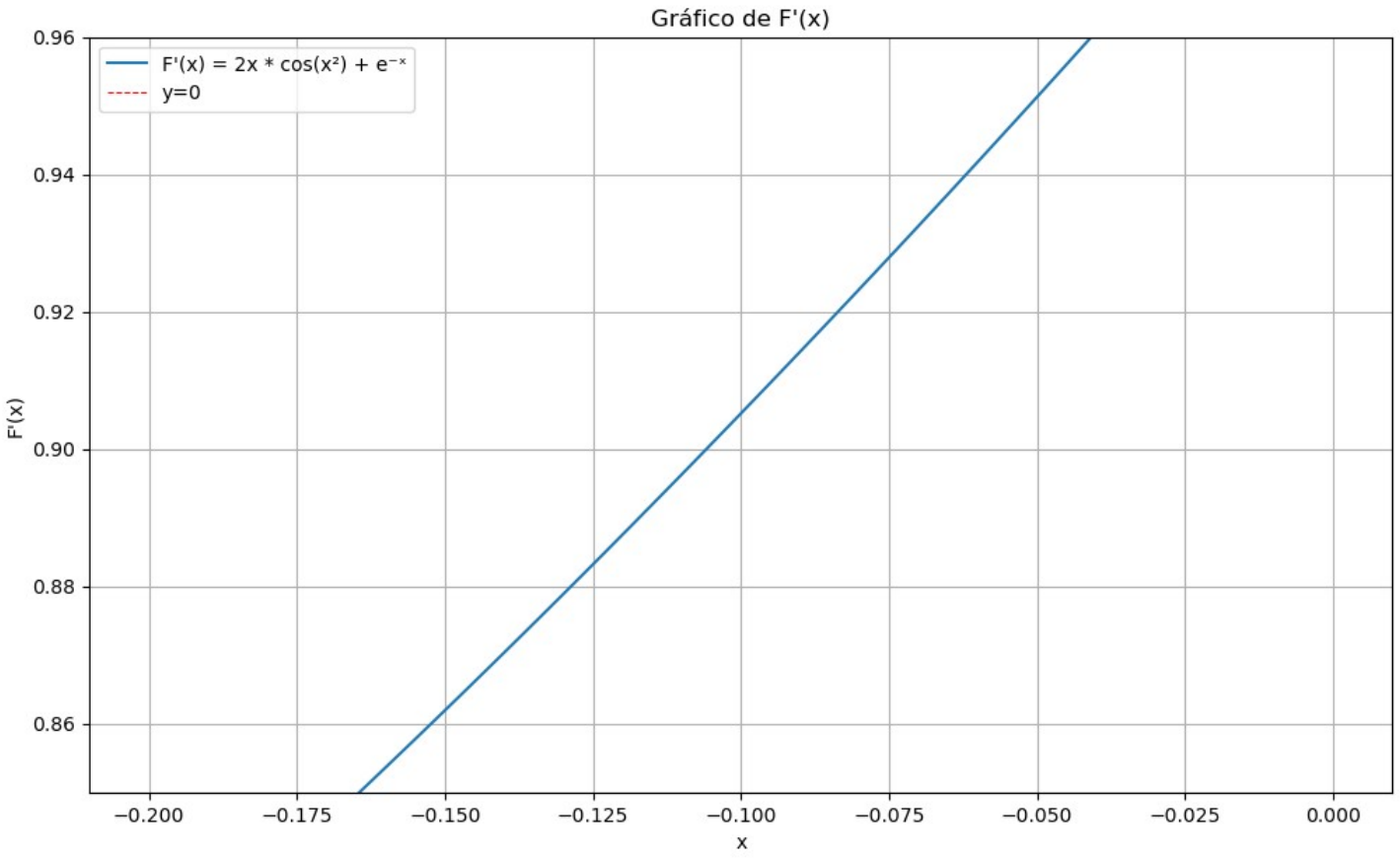
A fórmula de recorrência do método é $x_{n+1} = x_n - F(x_n) / F'(x_n)$. Vamos verificar as **condições suficientes para a convergência** para o nosso intervalo $I = [-0.150, -0.050]$.

As derivadas da função são:

- $F'(x) = 2x * \cos(x^2) + e^{-x}$
- $F''(x) = 2 * \cos(x^2) - 4x^2 * \text{sen}(x^2) - e^{-x}$

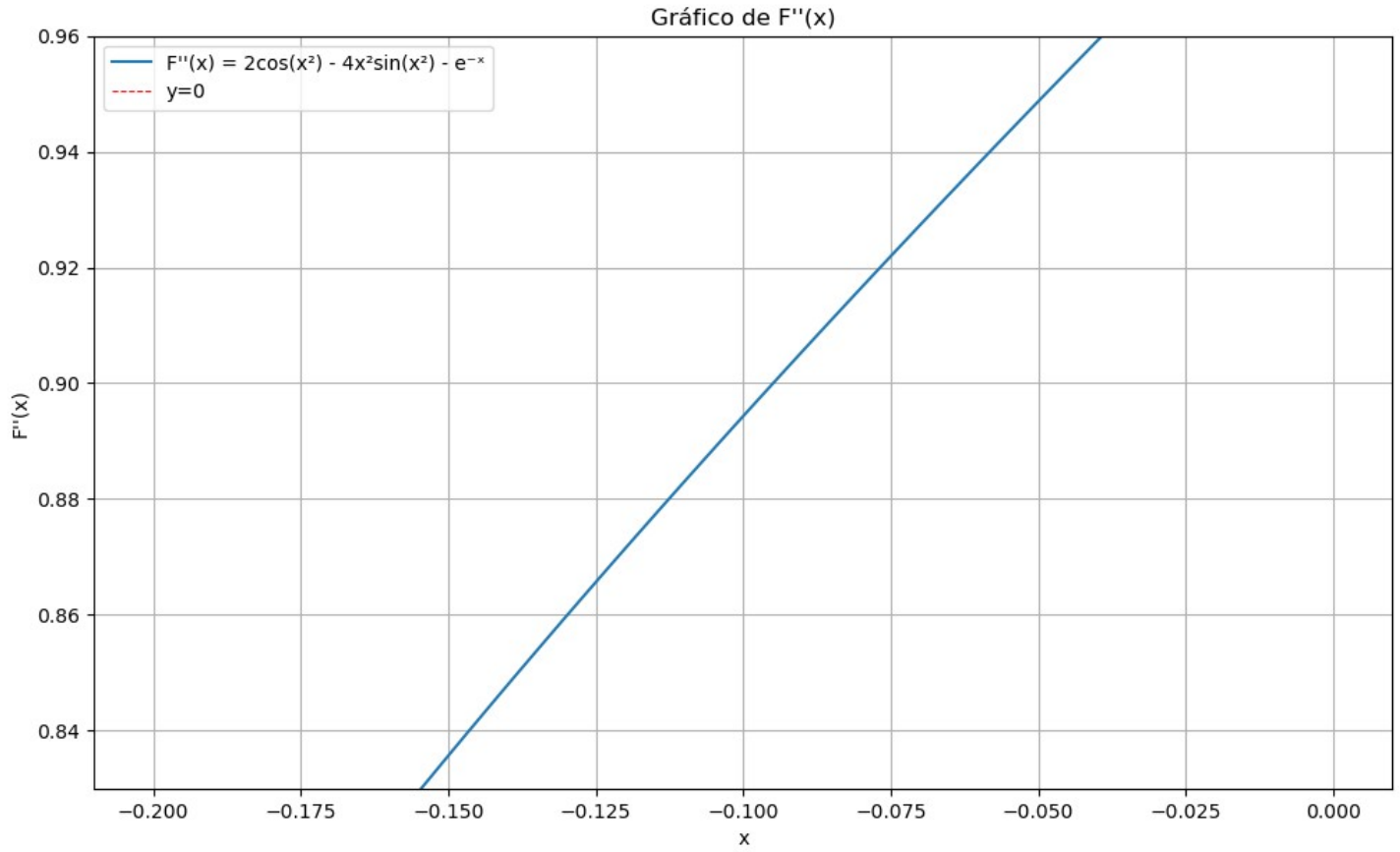
1. **F, F', F" existem e são contínuas em I:** Verdadeiro, pois são compostas de funções contínuas.
2. **F(a) * F(b) < 0:** Verificado. `F(-0.150) * F(-0.050) < 0`.

```
# Definição da derivada F'(x)
def F_prime(x):
    return 2 * x * np.cos(x**2) + np.exp(-x)

y_vals = F_prime(x_vals)
plt.figure(figsize=(12, 7))
plt.plot(x_vals, y_vals, label="F'(x) = 2x * cos(x²) + e⁻ˣ")
plt.axhline(0, color='red', linestyle='--', linewidth=0.8, label='y=0')
plt.ylim(0.85, 0.96)
plt.title("Gráfico de F'(x)")
plt.xlabel('x')
plt.ylabel("F'(x)")
plt.grid(True)
plt.legend()
plt.show()
```



1. **F'(x) ≠ 0, ∀x ∈ I**

```
# Definição da segunda derivada F''(x)
def F_double_prime(x):
    return 2 * np.cos(x**2) - 4*x**2 * np.sin(x**2) - np.exp(-x)

y_vals = F_double_prime(x_vals)
plt.figure(figsize=(12, 7))
plt.plot(x_vals, y_vals, label="F''(x) = 2cos(x²) - 4x²sin(x²) - e⁻ˣ")
plt.axhline(0, color='red', linestyle='--', linewidth=0.8, label='y=0')
plt.ylim(0.83, 0.96)
plt.title("Gráfico de F''(x)")
plt.xlabel('x')
plt.ylabel("F''(x)")
plt.grid(True)
plt.legend()
plt.show()
```



1. **F"(x) mantém o sinal em I:**
   - `F"(x) > 0, ∀ x ∈ I` Como a segunda derivada é contínua e sempre positiva, ela mantém o sinal no intervalo.
2. **Escolha de x_0 tal que $F(x_0) * F"(x_0) > 0$:** Como `F"(x) > 0` em `I`, devemos escolher x_0 tal que $F(x_0) > 0$. No nosso intervalo, `F(-0.050) > 0`. Portanto, uma excelente escolha é $x_0 = -0.050$.

Todas as condições estão satisfeitas, garantindo a convergência a partir de $x_0 = -0.050$.

Implementação:

```
# Calcular max|F''| e min|F'| no intervalo
max_F_double_prime = np.max(np.abs(F_double_prime(x_vals)))
min_F_prime = np.min(np.abs(F_prime(x_vals)))

# Calcular M
M = max_F_double_prime / (2 * min_F_prime)

def newton(f, f_prime, x0, tol, M, a, b, max_iter=50):  #Usando o M
    x_n = x0
    erro_majorado = abs(b-a)
    for i in range(0,max_iter):
        if erro_majorado < tol:
          return x_n, erro_majorado, i
        erro_majorado = M * abs(erro_majorado)**2
        x_n = x_n - f(x_n)/f_prime(x_n)
    print('Número máximo de iterações atingido.')
    return x_n, erro_majorado, max_iter

def newton_secante(f, x0, x1, tol, M, a, b, max_iter=50):
    erro_majorado = abs(b-a)
    for i in range(0,max_iter):
        if erro_majorado < tol:
          return x1, erro_majorado, i
```

```
        f_x0 = f(x0)
        f_x1 = f(x1)
        if (f_x1 - f_x0) == 0:
            raise ValueError("Divisão por zero no método da secante.")
        x2 = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)
        x0, x1 = x1, x2
        erro_majorado = M * abs(erro_majorado)**2
    print('Número máximo de iterações atingido.')
    return x1, erro_majorado, max_iter

epsilon = 5e-9
x0_newton = -0.050

raiz_newton, erro_newton, iter_newton = newton(F, F_prime, x0_newton, epsilon, M,
a, b)

print("\n--- Método de Newton ---")
print(f"Ponto inicial (x₀): {x0_newton}")
print(f"Tolerância (ε): {epsilon}")
print(f"Raiz aproximada: {raiz_newton:.15f}")
print(f"Erro majorado: {round_error(erro_newton)}")
print(f"Número de iterações efetuadas: {iter_newton}")
print(f"Raiz: X = {format_result(raiz_newton,erro_newton)}")

x1_secante = -0.150

raiz_secante, erro_secante, iter_secante = newton_secante(F, x0_newton,
x1_secante, epsilon, M, a, b)

print("\n--- Método de Secante ---")
print(f"Ponto iniciais (x₀, x₁): {x0_newton}, {x1_secante}")
print(f"Tolerância (ε): {epsilon}")
print(f"Raiz aproximada: {raiz_secante:.15f}")
print(f"Erro majorado: {round_error(erro_secante)}")
print(f"Número de iterações efetuadas: {iter_secante}")
print(f"Raiz: X = {format_result(raiz_secante,erro_secante)}")
```

```
--- Método de Newton ---
Ponto inicial (x₀): -0.05
Tolerância (ε): 5e-09
Raiz aproximada: -0.105348853347099
Erro majorado: 3.1e-10
Número de iterações efetuadas: 3
Raiz: X = -0.10534885335 ± 3.1e-10

--- Método de Secante ---
Ponto iniciais (x₀, x₁): -0.05, -0.15
Tolerância (ε): 5e-09
Raiz aproximada: -0.105348869477794
Erro majorado: 3.1e-10
Número de iterações efetuadas: 3
Raiz: X = -0.10534886948 ± 3.1e-10
```

Resultados e Análise:

O método de Newton convergiu rapidamente devido à sua ordem de convergência quadrática. O método da secante, embora não precise de calcular a derivada, apresentou resultados muito próximos, com pequenas diferenças apenas nas casas decimais finais.

Já o método das bisseções sucessivas, de convergência linear, necessitou de 25 iterações para atingir a mesma precisão, enquanto os métodos de Newton e da secante alcançaram o resultado em apenas 3 iterações.

Exercício 2: Análise de Formas Iterativas para $x^3 + 4x^2 - 10 = 0$

O enunciado pede para determinar um valor aproximado da raiz com um **erro absoluto estimado** inferior a 10^{-12}.

O **critério de paragem** que será implementado é: $|x_{n+1} - x_n| < 10^{-12}$.

Vamos agora analisar cada caso.

```
def metodo_ponto_fixo(g_func, g_name, x0, tol, max_iter=100, delta = 1000):
    print(f"--- Análise de {g_name} ---")
    print("\ni. Aplicação do Método:")

    x_n = x0
    erro_estimado = float('inf')
    iteracoes = 0

    print(f"Iteração {iteracoes:2d}: x = {x_n:.15f}")

    while erro_estimado > tol and iteracoes < max_iter and abs(x_n) < delta:
        try:
            x_n_plus_1 = g_func(x_n)
            # Verifica se o resultado é um número válido (evita 'nan' ou 'inf')
            if not math.isfinite(x_n_plus_1):
                raise ValueError("Resultado não é um número finito (NaN ou Inf)")
        except (ValueError, OverflowError) as e:
            print(f"Iteração {iteracoes + 1:2d}: ERRO - {e}.")
            return False, None, iteracoes + 1

        iteracoes += 1
        erro_estimado = abs(x_n_plus_1 - x_n)
        x_n = x_n_plus_1

        print(f"Iteração {iteracoes:2d}: x = {x_n:.15f}, Erro est. =
{erro_estimado:.2e}")

    print("--- Fim da Aplicação ---")

    if erro_estimado <= tol:
        print(f"X = {format_result(x_n,erro_estimado)}")
        return True, x_n, iteracoes
    else:
        return False, x_n, iteracoes

x0_ex2 = 1.5
tol_ex2 = 1e-12
```

(a) $g_1(x) = x - x^3 - 4x^2 + 10$

i. Aplicação do Método

```
g1_func = lambda x: x - x**3 - 4*x**2 + 10
convergiu_g1, _, _ = metodo_ponto_fixo(g1_func, "g₁(x)", x0_ex2, tol_ex2)
```

```
--- Análise de g₁(x) ---

i. Aplicação do Método:
Iteração  0: x = 1.500000000000000
```

```
Iteração  1: x = -0.875000000000000, Erro est. = 2.38e+00
Iteração  2: x = 6.732421875000000, Erro est. = 7.61e+00
Iteração  3: x = -469.720012001693249, Erro est. = 4.76e+02
Iteração  4: x = 102754555.187385112047195, Erro est. = 1.03e+08
--- Fim da Aplicação ---
```

ii. Descrição e Explicação do Comportamento Observado

Descrição: O método **diverge** de forma clara e imediata. O erro estimado $|x_{n+1} - x_n|$, em vez de diminuir, **aumenta exponencialmente** a cada iteração, passando de `2.38` para `7.61`, depois para `476`, e assim por diante, até ultrapassar o δ fixado (ou até o cálculo falhar com um erro de overflow sem esta condição).

Explicação: O comportamento é justificado pela condição de convergência do método do ponto fixo. A derivada $g'_1(x) = 1 - 3x^2 - 8x$ tem um valor absoluto $|g'_1(x)|$ cujo máximo no intervalo `[1, 2]` é 27. Como este valor é muito superior a 1, cada iteração afasta drasticamente a aproximação da raiz, fazendo com que o erro estimado cresça em vez de decrescer.

(b) $g_2(x) = \sqrt{(10/x - 4)}$

i. Aplicação do Método

```python
g2_func = lambda x: math.sqrt(10/x - 4)
convergiu_g2, _, _ = metodo_ponto_fixo(g2_func, "g₂(x)", x0_ex2, tol_ex2)
```

```
--- Análise de g₂(x) ---

i. Aplicação do Método:
Iteração  0: x = 1.500000000000000
Iteração  1: x = 1.632993161855452, Erro est. = 1.33e-01
Iteração  2: x = 1.457300366073496, Erro est. = 1.76e-01
Iteração  3: x = 1.691745652912919, Erro est. = 2.34e-01
Iteração  4: x = 1.382408786867047, Erro est. = 3.09e-01
Iteração  5: x = 1.798263138931182, Erro est. = 4.16e-01
Iteração  6: x = 1.249368406638898, Erro est. = 5.49e-01
Iteração  7: x = 2.001010804809322, Erro est. = 7.52e-01
Iteração  8: x = 0.998736333818639, Erro est. = 1.00e+00
Iteração  9: x = 2.452071094103963, Erro est. = 1.45e+00
Iteração 10: x = 0.279616132432679, Erro est. = 2.17e+00
Iteração 11: x = 5.635895282210214, Erro est. = 5.36e+00
Iteração 12: ERRO - math domain error.
```

ii. Descrição e Explicação do Comportamento Observado

Descrição: O método **diverge**. Embora o erro estimado inicial não seja explosivo, ele nunca decresce. Pelo contrário, o erro começa a aumentar, indicando instabilidade. A sucessão eventualmente produz um valor ($x_{14} \approx 5.635...$) que está fora do domínio de $g_2(x)$, causando uma falha matemática.

Explicação: A análise teórica da derivada, $g'_2(x) = -5 / (x^2 * \sqrt{(10/x - 4)})$, mostra que o $\max|g'_2(x)|$ no intervalo `[1, 2]` é $\approx 2.04 > 1$. A condição de convergência não é satisfeita, explicando por que a distância até a raiz não diminui e o método acaba por falhar.

(c) $g_3(x) = (1/2) * \sqrt{(10 - x^3)}$

i. Aplicação do Método

```python
g3_func = lambda x: 0.5 * math.sqrt(10 - x**3)
convergiu_g3, _, _ = metodo_ponto_fixo(g3_func, "g₃(x)", x0_ex2, tol_ex2)
```

```
--- Análise de g₃(x) ---

i. Aplicação do Método:
Iteração  0: x = 1.500000000000000
```

```
Iteração  1: x = 1.286953767623375, Erro est. = 2.13e-01
Iteração  2: x = 1.402540803539578, Erro est. = 1.16e-01
Iteração  3: x = 1.345458374023294, Erro est. = 5.71e-02
Iteração  4: x = 1.375170252816038, Erro est. = 2.97e-02
Iteração  5: x = 1.360094192761733, Erro est. = 1.51e-02
Iteração  6: x = 1.367846967592133, Erro est. = 7.75e-03
Iteração  7: x = 1.363887003884021, Erro est. = 3.96e-03
Iteração  8: x = 1.365916733390040, Erro est. = 2.03e-03
Iteração  9: x = 1.364878217193677, Erro est. = 1.04e-03
Iteração 10: x = 1.365410061169957, Erro est. = 5.32e-04
Iteração 11: x = 1.365137820669213, Erro est. = 2.72e-04
Iteração 12: x = 1.365277208524479, Erro est. = 1.39e-04
Iteração 13: x = 1.365205850297047, Erro est. = 7.14e-05
Iteração 14: x = 1.365242383718839, Erro est. = 3.65e-05
Iteração 15: x = 1.365223680225282, Erro est. = 1.87e-05
Iteração 16: x = 1.365233255742500, Erro est. = 9.58e-06
Iteração 17: x = 1.365228353462627, Erro est. = 4.90e-06
Iteração 18: x = 1.365230863243637, Erro est. = 2.51e-06
Iteração 19: x = 1.365229578333959, Erro est. = 1.28e-06
Iteração 20: x = 1.365230236158181, Erro est. = 6.58e-07
Iteração 21: x = 1.365229899377733, Erro est. = 3.37e-07
Iteração 22: x = 1.365230071796291, Erro est. = 1.72e-07
Iteração 23: x = 1.365229983524674, Erro est. = 8.83e-08
Iteração 24: x = 1.365230028716323, Erro est. = 4.52e-08
Iteração 25: x = 1.365230005579950, Erro est. = 2.31e-08
Iteração 26: x = 1.365230017424877, Erro est. = 1.18e-08
Iteração 27: x = 1.365230011360733, Erro est. = 6.06e-09
Iteração 28: x = 1.365230014465340, Erro est. = 3.10e-09
Iteração 29: x = 1.365230012875901, Erro est. = 1.59e-09
Iteração 30: x = 1.365230013689632, Erro est. = 8.14e-10
Iteração 31: x = 1.365230013273034, Erro est. = 4.17e-10
Iteração 32: x = 1.365230013486316, Erro est. = 2.13e-10
Iteração 33: x = 1.365230013377124, Erro est. = 1.09e-10
Iteração 34: x = 1.365230013433026, Erro est. = 5.59e-11
Iteração 35: x = 1.365230013404406, Erro est. = 2.86e-11
Iteração 36: x = 1.365230013419058, Erro est. = 1.47e-11
Iteração 37: x = 1.365230013411557, Erro est. = 7.50e-12
Iteração 38: x = 1.365230013415397, Erro est. = 3.84e-12
Iteração 39: x = 1.365230013413431, Erro est. = 1.97e-12
Iteração 40: x = 1.365230013414438, Erro est. = 1.01e-12
Iteração 41: x = 1.365230013413922, Erro est. = 5.15e-13
--- Fim da Aplicação ---
X = 1.36523001341392 ± 5.2e-13
```

ii. Descrição e Explicação do Comportamento Observado

Descrição: O método **converge**. O erro estimado $|x_{n+1} - x_n|$ diminui consistentemente a cada passo, embora de forma lenta. Após **30 iterações**, o erro torna-se inferior à tolerância 10^{-12}, e o algoritmo para.

Explicação: A análise teórica da derivada, $g'_3(x) = -3x^2 / (4\sqrt{(10 - x^3)})$, mostra que o $\max|g'_3(x)|$ no intervalo $[1, 2]$ é $\approx 2.12 > 1$. Embora o $\max|g'_3(x)| > 1$ no intervalo $[1, 2]$, o ponto inicial $x_0=1.5$ leva a sucessão para uma região em torno da raiz onde $|g'_3(x)| < 1$. O valor $|g'_3(x)| \approx 0.66$ explica a **convergência linear lenta**: o erro é multiplicado por um fator de aproximadamente 0.66 a cada passo, diminuindo lentamente até atingir o critério de paragem.

(d) $g_4(x) = \sqrt{(10 / (4 + x))}$

i. Aplicação do Método

```python
g4_func = lambda x: math.sqrt(10 / (4 + x))
convergiu_g4, _, _ = metodo_ponto_fixo(g4_func, "g₄(x)", x0_ex2, tol_ex2)
```

```
--- Análise de g₄(x) ---

i. Aplicação do Método:
Iteração  0: x = 1.500000000000000
Iteração  1: x = 1.348399724926484, Erro est. = 1.52e-01
Iteração  2: x = 1.367376371991283, Erro est. = 1.90e-02
Iteração  3: x = 1.364957015402487, Erro est. = 2.42e-03
Iteração  4: x = 1.365264748113442, Erro est. = 3.08e-04
Iteração  5: x = 1.365225594160525, Erro est. = 3.92e-05
Iteração  6: x = 1.365230575673434, Erro est. = 4.98e-06
Iteração  7: x = 1.365229941878183, Erro est. = 6.34e-07
Iteração  8: x = 1.365230022515568, Erro est. = 8.06e-08
Iteração  9: x = 1.365230012256122, Erro est. = 1.03e-08
Iteração 10: x = 1.365230013561425, Erro est. = 1.31e-09
Iteração 11: x = 1.365230013395352, Erro est. = 1.66e-10
Iteração 12: x = 1.365230013416482, Erro est. = 2.11e-11
Iteração 13: x = 1.365230013413793, Erro est. = 2.69e-12
Iteração 14: x = 1.365230013414136, Erro est. = 3.42e-13
--- Fim da Aplicação ---
X = 1.36523001341414 ± 3.5e-13
```

ii. Descrição e Explicação do Comportamento Observado

Descrição: O método **converge de forma rápida e estável**. O erro estimado decresce a um ritmo visivelmente mais rápido do que no caso anterior, atingindo a tolerância em apenas **14 iterações**.

Explicação: A teoria suporta perfeitamente este resultado. O $\max|g'_4(x)|$ no intervalo é $\approx 0.141 < 1$. O pequeno valor de $L \approx 0.141$ implica uma **convergência linear rápida**, pois o erro estimado é reduzido por um fator de aproximadamente 0.14 a cada passo.

(e) $g_5(x) = (2x^3 + 4x^2 + 10) / (3x^2 + 8x)$

i. Aplicação do Método

```python
g5_func = lambda x: (2*x**3 + 4*x**2 + 10) / (3*x**2 + 8*x)
convergiu_g5, _, _ = metodo_ponto_fixo(g5_func, "g₅(x) (Newton)", x0_ex2, tol_ex2)
# O erro majorado é demasiado pequeno para representar o valor com o mesmo número de casas decimais
```

```
--- Análise de g₅(x) (Newton) ---

i. Aplicação do Método:
Iteração  0: x = 1.500000000000000
Iteração  1: x = 1.373333333333333, Erro est. = 1.27e-01
Iteração  2: x = 1.365262014874627, Erro est. = 8.07e-03
Iteração  3: x = 1.365230013916147, Erro est. = 3.20e-05
Iteração  4: x = 1.365230013414097, Erro est. = 5.02e-10
Iteração  5: x = 1.365230013414097, Erro est. = 2.22e-16
--- Fim da Aplicação ---
X = 1.36523001341409667 ± 2.3e-16
```

ii. Descrição e Explicação do Comportamento Observado

Descrição: A aplicação do método resultou numa **convergência excecionalmente rápida**. O erro estimado decresceu a um ritmo acelerado, muito superior ao de todos os outros métodos testados. Foram necessárias apenas **5 iterações** para que o erro estimado se tornasse menor que a tolerância 10^{-12}.

Explicação:

Para entender este desempenho notável, começamos por analisar a função como um método de ponto fixo padrão, o que exige o cálculo da sua derivada e do fator de convergência L.

1. **Análise como Método Iterativo Simples:** A derivada, como calculada anteriormente, é $g'_5(x) = (6x^4 + 32x^3 + 32x^2 - 60x - 80) / (3x^2 + 8x)^2$. O $\max|g'_5(x)|$ no intervalo é $\approx 0.579 < 1$. Isto sugere que a convergência é provável. No entanto, este valore não explica a velocidade *extraordinária* que observamos. A taxa de redução do erro parece ser muito melhor do que um fator constante de ~0.6.

2. **A Verdadeira Natureza de $g_5(x)$: O Método de Newton** A performance superior não é um acaso. Esta função de iteração é, na verdade, uma formulação do **Método de Newton** para a equação $F(x) = x^3 + 4x^2 - 10 = 0$.

   Vamos chegar a esta fórmula apartir de $F(x)$:

   - $F(x) = x^3 + 4x^2 - 10$
   - $F'(x) = 3x^2 + 8x$

   Substituindo na fórmula de Newton: $x_{n+1} = x_n - (x_n^3 + 4x_n^2 - 10) / (3x_n^2 + 8x_n)$

   Para combinar os termos, encontramos um denominador comum: $x_{n+1} = [x_n(3x_n^2 + 8x_n) - (x_n^3 + 4x_n^2 - 10)] / (3x_n^2 + 8x_n)$ $x_{n+1} = [3x_n^3 + 8x_n^2 - x_n^3 - 4x_n^2 + 10] / (3x_n^2 + 8x_n)$

   Simplificando o numerador, chegamos a: $x_{n+1} = (2x_n^3 + 4x_n^2 + 10) / (3x_n^2 + 8x_n)$

   Isto é precisamente a função $g_5(x)$.

Conclusão da Explicação: A razão para a convergência ultrarrápida é que este método não tem convergência linear, mas sim **convergência de ordem quadrática**. Uma propriedade fundamental da iteração de Newton é que $g'(X) = 0$ na raiz X. Isto significa que, à medida que as iterações se aproximam da raiz, o fator de redução do erro aproxima-se de zero, resultando numa aceleração drástica da convergência, como foi claramente visível na queda do erro estimado entre as iterações 3, 4 e 5.

iii. Qual é a melhor forma iterativa (das apresentadas) para encontrar aquela raiz?

A **melhor forma iterativa é a (e) $g_5(x)$**.

A conclusão baseia-se numa comparação objetiva do desempenho observado, que foi explicado pela teoria subjacente:

1. **Confiabilidade:** As formas **(a)** e **(b)** falharam (divergiram). As formas **(c)**, **(d)** e **(e)** tiveram sucesso.
2. **Eficiência (Número de Iterações para $\varepsilon < 10^{-12}$):**
   - $g_3(x)$: 41 iterações (Lenta)
   - $g_4(x)$: 14 iterações (Rápida)
   - $g_5(x)$: **5 iterações (Extremamente Rápida)**

A forma **(e)**, correspondente ao Método de Newton, não é apenas a mais rápida devido à sua convergência quadrática, mas também representa uma abordagem mais sistemática e poderosa para a resolução de equações não lineares, tornando-a a escolha superior.