UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
Métodos Numéricos
Professor: Thales Vieira
Curso: Engenharia de Computação

Lista 1

Dupla: Gabriel Vitorino de Andrade e Saulo Roberto dos Santos
Link do repositório: https://github.com/saulolv/Metodos-Numericos/tree/main/Lista%201

1. **(Questão teórica)** Considerando a representação em ponto flutuante com 64 bits, determine a representação decimal dos números abaixo.

a) 0 10000001010 1001001100000000000000001000000000000100000000001000000
b) 1 10000001010 1011001100000000000100000000000010000000000000000000
c) 0 01101011111 0001001100000000000000001000000000000000000000001000000
d) 1 01011111011 0101001100000000000000000000000001000000000100000000001

SOLUÇÃO:

Letra a)

1) a) 0 10000001010 1001001100000000000000001000000000000100000000001000000

$(-1)^S \cdot 2^{C-1023} \cdot (1+f)$

10000001010 $\quad 1 \cdot 2^{10} + 1 \cdot 2^{3} + 1 \cdot 2^{1} = 1 \cdot (1024) + 1 \cdot (8) + 1 \cdot (2) = 1024 + 8 + 2 = 1034$

$C = 1034$

$f = 1 \cdot (2)^{-1} + 1 \cdot (2)^{-4} + 1 \cdot (2)^{-7} + 1 \cdot (2)^{-8} + 1 \cdot (2)^{-25} + 1 \cdot (2)^{-37} + 1 \cdot (2)^{-46}$

$f = \frac{1}{2} + \frac{1}{16} + \frac{1}{128} + \frac{1}{256} + \frac{1}{33554432} + \frac{1}{137438953500} + \frac{1}{70368744180000}$

$f = 0,5 + 0,0625 + 7,8125 \cdot 10^{-3} + 3,90625 \cdot 10^{-3} + 2,980232239 \cdot 10^{-8} + 7,275957614 \cdot 10^{-12} + 1,421085472 \cdot 10^{-14}$

$f = 0,5 + 0,0625 + 0,0078125 + 0,00390625 + 0,00000002980232239 + 0,000000000007275957614 + 0,00000000000001421085472 = 0,574218752980304998576 14$

0,5
0,0625
0,0078125
0,00390625
0,00000002980232239
+ 0,000000000007275957614
0,574218752980304998576 14

$S = 0$
$C = 1034$
$f = 0,57421875298030499857614$

$(-1)^{(0)} \cdot 2^{(1034)-1023} \cdot (1 + 0,57421875298030499857614)$

$(1) \cdot 2^{11} \cdot (1,57421875298030499857614)$

$2048 \cdot (1,57421875298030499857614) \approx \boxed{3224,000006}$

Letra b)

b) 1 10000001010 1011001100000000001000000000000001000000000000000000000

10 3 2 1 0 -1 -3 -4 -7 -8 -19 -33

S=1

$C = 1.2^{10} + 1.2^{3} + 1.2^{1} = 1024 + 8 + 2 = 1034$

$f = 1.(2)^{-1} + 1.(2)^{-3} + 1.(2)^{-4} + 1.(2)^{-7} + 1.(2)^{-8} + 1.(2)^{-19} + 1.(2)^{-33} = 1.\frac{1}{2} + 1.\frac{1}{2^3} + 1.\frac{1}{2^4} + 1.\frac{1}{2^7} + 1.\frac{1}{2^8} + 1.\frac{1}{2^{19}} + 1.\frac{1}{2^{33}}$

$\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{128} + \frac{1}{256} + \frac{1}{524288} + \frac{1}{8589934592} = 0{,}5 + 0{,}125 + 0{,}0625 + 7{,}8125.10^{-3} + 3{,}90625.10^{-3} + 1{,}9073486133.10^{-6}$

$+ 1{,}164153218.10^{-10} \cong 0{,}6992206575$

$(-1)^{(1)} . 2^{(1034)-1023} . (1 + (0{,}6992206575)) = (-1) . 2^{11} . (1{,}699220658) \cong \boxed{-3480{,}003908}$

tilibra

Letra c)

c) 0 01110101111 0001001100000000000000100000000000000000000000001000000

8 7 5 3 2 1 0 -4 -7 -8 -23 -46

S=0

$C = 1.2^{0} + 1.2^{1} + 1.2^{2} + 1.2^{3} + 1.2^{5} + 1.2^{7} + 1.2^{8} = 1 + 2 + 4 + 8 + 32 + 128 + 256 = 431$

$f = 1.\frac{1}{2^4} + 1.\frac{1}{2^7} + 1.\frac{1}{2^8} + 1.\frac{1}{2^{23}} + 1.\frac{1}{2^{46}} = \frac{1}{16} + \frac{1}{128} + \frac{1}{256} + \frac{1}{8388608} + \frac{1}{7{,}036874418.10^{13}}$

$0{,}0625 + 7{,}8125.10^{-3} + 3{,}90625.10^{-3} + 1{,}192092896.10^{-7} + 1{,}421085472.10^{-14} \cong$

$0{,}07421886921$

$S=0$ | $(-1)^{S} . 2^{C-1023} . (1+f)$

$C=431$ | $(-1)^{(0)} . 2^{(431)-1023} . (1 + (0{,}07421886921))$

$f=0{,}07421886921$ | $1.2^{-592} . (1{,}07421886 9)$

$\frac{1}{2^{592}} . (1{,}074218869) =$

$\boxed{= 6{,}627280.10^{-179}}$

Letra d)

d) 1 01011111011 0101001100000000000000000000000000000001000000001000000001

9 7 6 5 4 3 1 0 -2 -4 -7 -8 -34 -43 -52

S=1

$C = 1.2^{0} + 1.2^{1} + 1.2^{3} + 1.2^{4} + 1.2^{5} + 1.2^{6} + 1.2^{7} + 1.2^{9} = 1 + 2 + 8 + 16 + 32 + 64 + 128 + 512 = 763$

$f = 1.\frac{1}{2^2} + 1.\frac{1}{2^4} + 1.\frac{1}{2^7} + 1.\frac{1}{2^8} + 1.\frac{1}{2^{34}} + 1.\frac{1}{2^{43}} + 1.\frac{1}{2^{52}} = 0{,}25 + 0{,}0625 + 0{,}0078125 +$

$0{,}00390625 + 5{,}82076609134674072265625.10^{-11} + 1{,}136868377216160297393798828125.10^{-13} +$

$2{,}220446049250313080847263336181640625.10^{-16} = 0{,}324218750058321569795793948287609964609146118164 0625$

$(-1)^{(1)} . 2^{(763)-1023} . (1 + (0{,}3242187500583215697957939482876099646091461181640625))$

$(-1) . 2^{-260} . (1{,}3242187500583215697957939482876099646091461181640625) \cong \boxed{-7{,}147670.10^{-79}}$

2. Implemente algoritmos para calcular o erro absoluto e o erro relativo das aproximações de p por p^* nos casos abaixo:

a) $p = 1$ e $p^* = 0.9994$
b) $p = 124$ e $p^* = 7$
c) $p = e^{10}$ e $p^* = 22000$
d) $p = 8!$ e $p^* = 39900$

CÓDIGO:

```python
def AbsoluteError(p, p_star):
    return abs(p - p_star)

def RelativeError(p, p_star):
    return abs((p - p_star)) / abs(p)

def Fatorial(n):
    if n == 1:
        return 1
    return n * Fatorial(n - 1)

def main():
    valores = [
        ('a', 1, 0.9994),
        ('b', 124, 7),
        ('c', 0.0001, 22000),
        ('d', Fatorial(8), 39900),
    ]

    for desc, p, p_star in valores:
        print(f'Erro absoluto de {desc}: {AbsoluteError(p, p_star)}')
        print(f'Erro relativo de {desc}: {RelativeError(p, p_star)}\n')

if __name__ == '__main__':
    main()
```

SAÍDA DO CÓDIGO: (Letra a, b, c e d na figura abaixo)

```
Erro absoluto de a: 0.0006000000000000449
Erro relativo de a: 0.0006000000000000449

Erro absoluto de b: 117
Erro relativo de b: 0.9435483870967742

Erro absoluto de c: 21999.9999
Erro relativo de c: 219999998.99999997

Erro absoluto de d: 420
Erro relativo de d: 0.010416666666666666
```

3. Suponha que p^* deva aproximar p com erro relativo máximo de 10^{-2}. Determine o maior intervalo no qual p^* pode estar para os valores de p abaixo:

a) $p = 150$
b) $p = 900$
c) $p = 1500$
d) $p = 90$

CÓDIGO:

```python
def CalculateIntervalePstar(p, error):
    limite_superior = p + error * p
    limite_inferior = p - error * p
    return limite_inferior, limite_superior

def main():
    error = 10 ** -2
    valores = [
        ('a', 150),
        ('b', 900),
        ('c', 1500),
        ('d', 90),
    ]

    for desc, p in valores:
        limite_inferior, limite_superior = CalculateIntervalePstar(p, error)
        print(f'Intervalo de {desc}: [{limite_inferior}, {limite_superior}]')


if __name__ == '__main__':
    main()
```

SAÍDA DO CÓDIGO: (Letra a, b, c e d na figura abaixo)

```
Intervalo de a: [148.5, 151.5]
Intervalo de b: [891.0, 909.0]
Intervalo de c: [1485.0, 1515.0]
Intervalo de d: [89.1, 90.9]
```

4. Implemente algoritmos para realizar os cálculos abaixo usando truncamento com 3 dígitos, truncamento com 4 dígitos, arredondamento com 3 dígitos e arrendondamento com 4 dígitos. Realize o cálculo exato e calcule os erros absoluto e relativo de cada aproximação para cada item abaixo. Obs.: considere como cálculo exato aquele realizado em Python com sua precisão padrão. Você pode usar a biblioteca *decimal* do Python para obter aproximações dos números[1].

a) $133 - 0,499$

b) x, tal que $\frac{1}{3}x^2 - \frac{123}{4}x + \frac{1}{6} = 0$

c) $\dfrac{\frac{13}{14} - \frac{6}{7}}{2e - 5,4}$

CÓDIGO:

```python
from decimal import Decimal, getcontext


getcontext().prec = 8

def calcular_aproximacoes(calc_exato):
    # Realizar calculos com truncamento de 3 digitos
    resultado_truncado_3 = Decimal(calc_exato).quantize(Decimal('0.000'), rounding='ROUND_DOWN')

    # Realizar calculos com truncamento de 4 digitos
    resultado_truncado_4 = Decimal(calc_exato).quantize(Decimal('0.0000'), rounding='ROUND_DOWN')

    # Realizar calculos com arredondamento de 3 digitos
    resultado_arredondado_3 = Decimal(calc_exato).quantize(Decimal('0.000'), rounding='ROUND_HALF_UP')

    # Realizar calculos com arredondamento de 4 digitos
    resultado_arredondado_4 = Decimal(calc_exato).quantize(Decimal('0.0000'), rounding='ROUND_HALF_UP')

    erros_3 = {
        'truncado_absoluto': float(abs(calc_exato - resultado_truncado_3)),
        'arredondado_absoluto': float(abs(calc_exato - resultado_arredondado_3)),
        'truncado_relativo': float(abs(calc_exato - resultado_truncado_3) / abs(calc_exato)),
        'arredondado_relativo': float(abs(calc_exato - resultado_arredondado_3) / abs(calc_exato))
    }
    erros_4 = {
        'truncado_absoluto': float(abs(calc_exato - resultado_truncado_4)),
        'arredondado_absoluto': float(abs(calc_exato - resultado_arredondado_4)),
        'truncado_relativo': float(abs(calc_exato - resultado_truncado_4) / abs(calc_exato)),
        'arredondado_relativo': float(abs(calc_exato - resultado_arredondado_4) / abs(calc_exato))
    }

    return {
        'truncado_3': resultado_truncado_3,
        'truncado_4': resultado_truncado_4,
        'arredondado_3': resultado_arredondado_3,
        'arredondado_4': resultado_arredondado_4,
        'erros_3': erros_3,
        'erros_4': erros_4
    }

def imprimir_resultados(titulo, calc_exato, resultados):
    print(f'{titulo}\n Calculo exato: {calc_exato}\n')
    for k, v in resultados.items():
        print(f'{k}: {v}')
    print('\n')




def main():
    # Exemplo a) 133 - 0.499
    calc_exato_a = Decimal('133') - Decimal('0.499')
    resultados_a = calcular_aproximacoes(calc_exato_a)
    imprimir_resultados('Exemplo a) 133 - 0.499', calc_exato_a, resultados_a)

    # Exemplo b) x, tal que (1/3)*x**2 - (123/4)*x + (1/6) = 0
    calc_exato_b =  Decimal('1') / Decimal('3')
    resultados_b = calcular_aproximacoes(calc_exato_b)
    imprimir_resultados('Exemplo b) x, tal que (1/3)*x**2 - (123/4)*x + (1/6) = 0', calc_exato_b, resultados_b)

    # Exemplo c) ((13/4) - (6/7)) / (2e - 5.4)
    e = Decimal('2.71828182845904523536028747135 27')
    calc_exato_c = (Decimal(13) / Decimal(4) - Decimal(6) / Decimal(7)) / ((Decimal('2') * Decimal(e)) - Decimal('5.4'))
    imprimir_resultados('Exemplo c) ((13/4) - (6/7)) / (2e - 5.4)', calc_exato_c, calcular_aproximacoes(calc_exato_c))
main()
```

SAÍDA DO CÓDIGO:

```
Exemplo a) 133 - 0.499
 Calculo exato: 132.501

truncado_3: 132.501
truncado_4: 132.5010
arredondado_3: 132.501
arredondado_4: 132.5010
erros_3: {'truncado_absoluto': 0.0, 'arredondado_absoluto': 0.0, 'truncado_relativo': 0.0, 'arredondado_relativo': 0.0}
erros_4: {'truncado_absoluto': 0.0, 'arredondado_absoluto': 0.0, 'truncado_relativo': 0.0, 'arredondado_relativo': 0.0}


Exemplo b) x, tal que (1/3)*x**2 - (123/4)*x + (1/6) = 0
 Calculo exato: 0.33333333

truncado_3: 0.333
truncado_4: 0.3333
arredondado_3: 0.333
arredondado_4: 0.3333
erros_3: {'truncado_absoluto': 0.00033333, 'arredondado_absoluto': 0.00033333, 'truncado_relativo': 0.00099999001, 'arredondado_relativo': 0.00099999001}
erros_4: {'truncado_absoluto': 3.333e-05, 'arredondado_absoluto': 3.333e-05, 'truncado_relativo': 9.9990001e-05, 'arredondado_relativo': 9.9990001e-05}


Exemplo c) ((13/4) - (6/7)) / (2e - 5.4)
 Calculo exato: 65.443516

truncado_3: 65.443
truncado_4: 65.4435
arredondado_3: 65.444
arredondado_4: 65.4435
erros_3: {'truncado_absoluto': 0.000516, 'arredondado_absoluto': 0.000484, 'truncado_relativo': 7.8846619e-06, 'arredondado_relativo': 7.3956907e-06}
erros_4: {'truncado_absoluto': 1.6e-05, 'arredondado_absoluto': 1.6e-05, 'truncado_relativo': 2.4448564e-07, 'arredondado_relativo': 2.4448564e-07}
```

5. Devido ao problema da representação, é possível que uma simples adição do tipo $1.0 + \epsilon$ retorne o valor 1.0. Implemente um algoritmo que determina qual o menor inteiro positivo n tal que $1.0 + 2^{-n} = 1.0$ na sua configuração de computador/linguagem de programação.

CÓDIGO:

```python
def find_smallest_n():
    n = 0
    resultado = 1.0 + 2**(-n)
    while resultado != 1.0:
        n += 1
        resultado = 1 + 2**(-n)
    return n

def main():
    print(f'O menor n que satisfaz 1 + 2**(-n) = 1 é {find_smallest_n()}')

if __name__ == '__main__':
    main()
```

SAÍDA DO CÓDIGO:

```
O menor n que satisfaz 1 + 2**(-n) = 1 é 53
```

6. Implemente o Método da Bisecção e o Método de Newton para resolver $f(x) = 0$, de modo que f seja um dos parâmetros de entrada. Para o método da bisseção, use a diferença entre as aproximações consecutivas como critério de parada nos dois métodos.

CÓDIGO:

```python
def f(x):
    return x**3 - 9*x + 3

def df(x):
    return 3*x**2 - 9

def MetodoBissecao(f, a, b, tolerance=1e-6, max_iterations=100):
    if f(a) * f(b) > 0:
        return None

    previous_midpoint = None

    while max_iterations > 0:
        midpoint = (a + b) / 2
        if f(midpoint) == 0 or (previous_midpoint is not None and abs(midpoint - previous_midpoint) < tolerance):
            return midpoint  # Encontrou uma raiz exata ou convergiu
        elif f(midpoint) * f(a) < 0:
            b = midpoint
        else:
            a = midpoint

        previous_midpoint = midpoint
        max_iterations -= 1

    return (a + b) / 2


def MetodoNewton(f, df, x0, tolerance=1e-6, max_iterations=100):
    x = x0
    previous_x = None

    while max_iterations > 0:
        x = x - f(x) / df(x)

        if previous_x is not None and abs(x - previous_x) < tolerance:
            return x  # Convergiu
        elif abs(f(x)) < tolerance:
            return x  # Encontrou uma raiz exata

        previous_x = x
        max_iterations -= 1

    return x

def main():
    resultado_bissecao = MetodoBissecao(f, 0, 1)
    print(f'Raiz encontrada pelo método da bisseção: {resultado_bissecao}')
    resultado_newton = MetodoNewton(f, df, 0.5)
    print(f'Raiz encontrada pelo método de Newton: {resultado_newton}')

if __name__ == '__main__':
    main()
```

SAÍDA DO CÓDIGO:

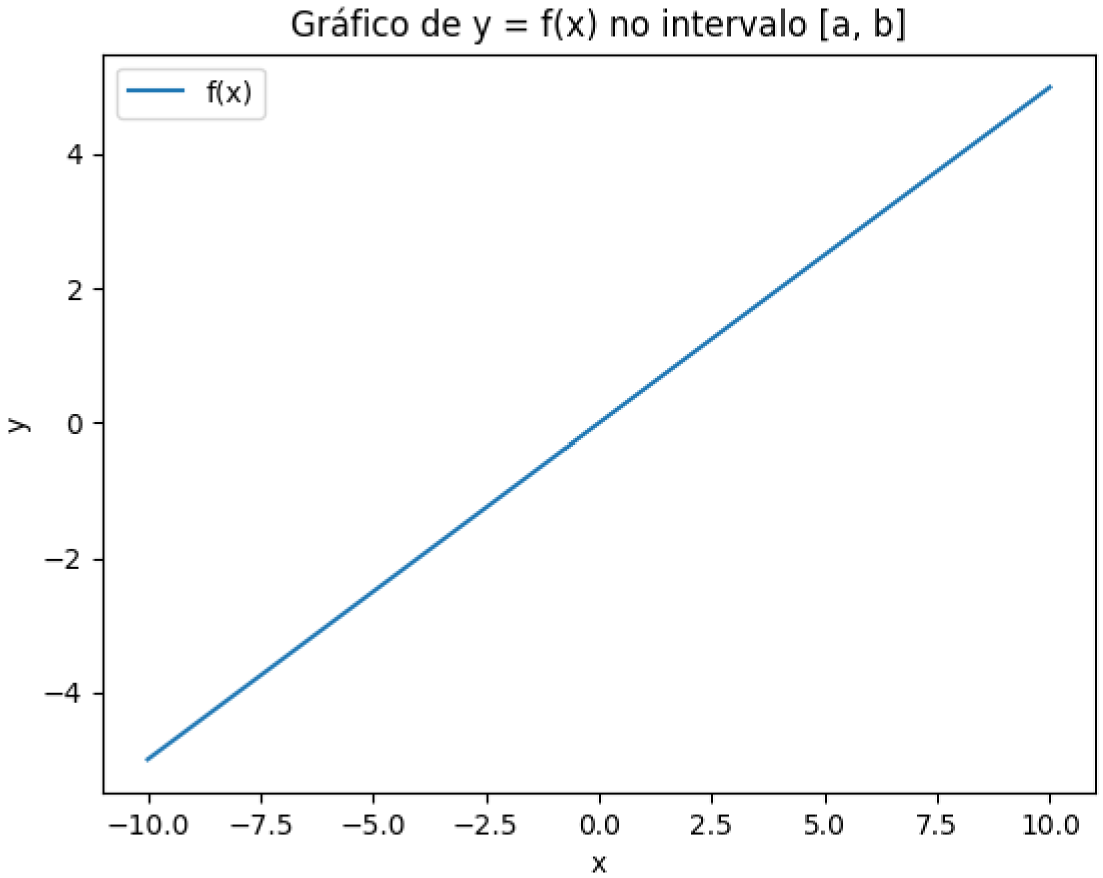
```
Raiz encontrada pelo método da bisseção: 0.33760929107666016
Raiz encontrada pelo método de Newton: 0.3376089559653128
```

7. Implemente uma função que receba f, a, b e Δ e plote o gráfico de $y = f(x)$ restrito ao intervalo $[a, b]$, amostrando uniformemente o domínio entre a e b com passo Δ, e usando segmentas de reta.

CÓDIGO:

```python
import matplotlib.pyplot as plt

def generate_x_values(a, b, delta):
    x_values = []
    current_x = a

    while current_x <= b:
        x_values.append(current_x)
        current_x += delta

    return x_values

def plot_function(f, a, b, delta):
    x_values = generate_x_values(a, b, delta)
    y_values = [f(x) for x in x_values]

    # Plota o gráfico usando segmentos de reta
    plt.plot(x_values, y_values, label='f(x)')

    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Gráfico de y = f(x) no intervalo [a, b]')
    plt.legend()

    plt.show()

# Exemplo de uso:
def f(x):
    return x**1/2

def main():
    plot_function(f, -10, 10, 0.001)

main()
```

SAÍDA DO CÓDIGO:



8. Implemente uma função que receba f e uma tolerância (TOL), e retorne uma lista com a sequência p_n de aproximações da raiz.

CÓDIGO:

```python
def f(x):
    return x**3 - 9*x + 3

def bissecao(f, a, b, tol):
    seq_aprox = []
    if f(a) * f(b) > 0:
        return None

    while abs(b - a) > tol:
        c = (a + b) / 2
        seq_aprox.append(c)
        if f(c) == 0:
            break
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return seq_aprox

def main():
    print(bissecao(f, 0, 1, 1e-6))

main()
```

SAÍDA DO CÓDIGO:

```
[0.5, 0.25, 0.375, 0.3125, 0.34375, 0.328125, 0.3359375, 0.33984375, 0.
337890625, 0.3369140625, 0.33740234375, 0.337646484375, 0.3375244140625
, 0.33758544921875, 0.337615966796875, 0.3376007080078125, 0.3376083374
0234375, 0.3376121520996094, 0.33761024475097656, 0.33760929107666016]
```

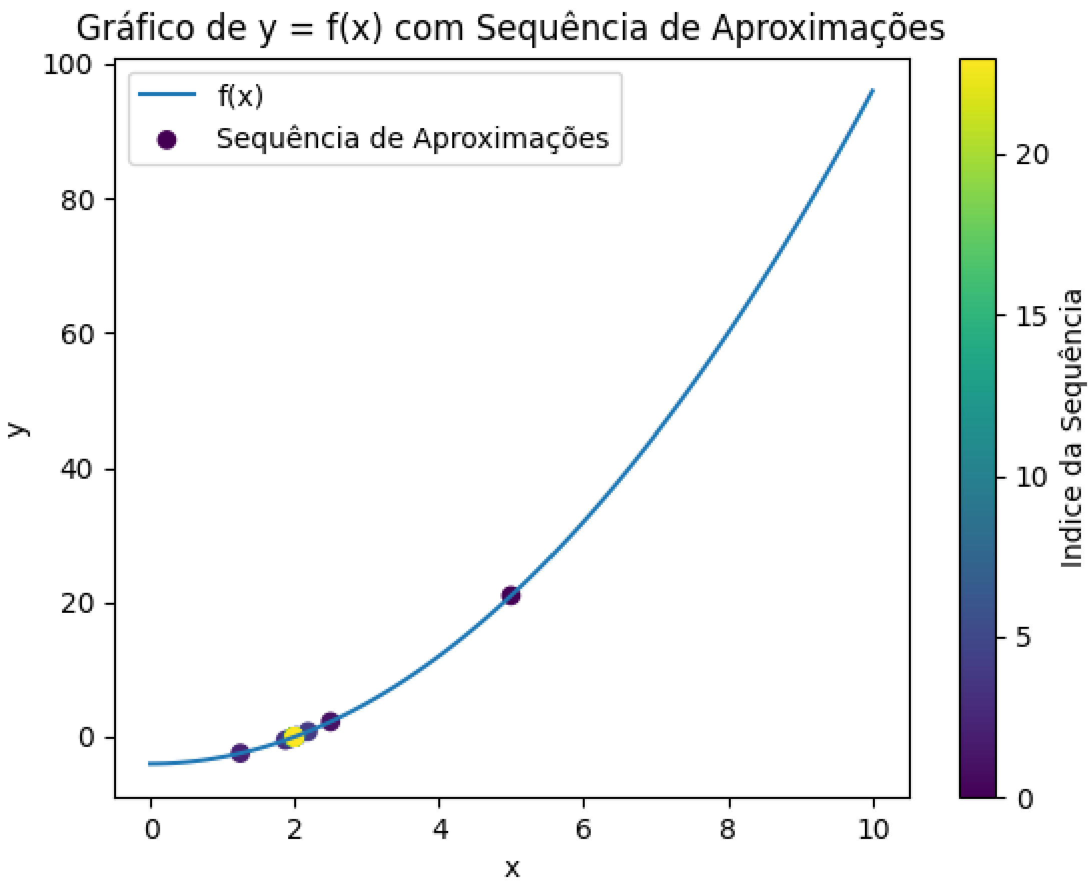
9. Combinando os resultados das questões 6 e 7, implemente um algoritmo que plote o gráfico de $y = f(x)$ junto com a sequência de pontos $(p_n, f(p_n))$. O gráfico deve ser uma curva poligonal (i.e. segmentos de reta conectados). Os pontos da sequência devem ser coloridos de acordo com seu índice.

[1] https://docs.python.org/pt-br/3/library/decimal.html

CÓDIGO:

```python
import matplotlib.pyplot as plt


def f(x):
    return x**2 - 4

def bissecao(f, a, b, tol):
    seq_aprox = []
    if f(a) * f(b) > 0:
        return None

    while abs(b - a) > tol:
        c = (a + b) / 2
        seq_aprox.append(c)
        if f(c) == 0:
            break
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return seq_aprox

def linspace(a, b, n):
    delta = (b - a) / (n - 1)
    return [a + i * delta for i in range(n)]

def Solve():
    x_values = linspace(0, 10, 100)
    y_values = [f(x) for x in x_values]

    a = 0
    b = 10
    tol = 1e-6
    seq_aprox = bissecao(f, a, b, tol)

    plt.plot(x_values, y_values, label='f(x)')
    plt.scatter(seq_aprox, [f(x) for x in seq_aprox], c=range(len(seq_aprox)), cmap='viridis', label='Sequência de Aproximações')
    plt.colorbar(label="Indice da Sequência")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Gráfico de y = f(x) com Sequência de Aproximações')
    plt.legend()
    plt.show()

Solve()
```

SAÍDA DO CÓDIGO:



9. Gere resultados das questões 5, 6, 7 e 8 para as seguintes equações. Gerar resultado significa resolver a equação (com cada um dos dois métodos) e plotar os gráficos da questão 8. Use uma tolerância adequada.

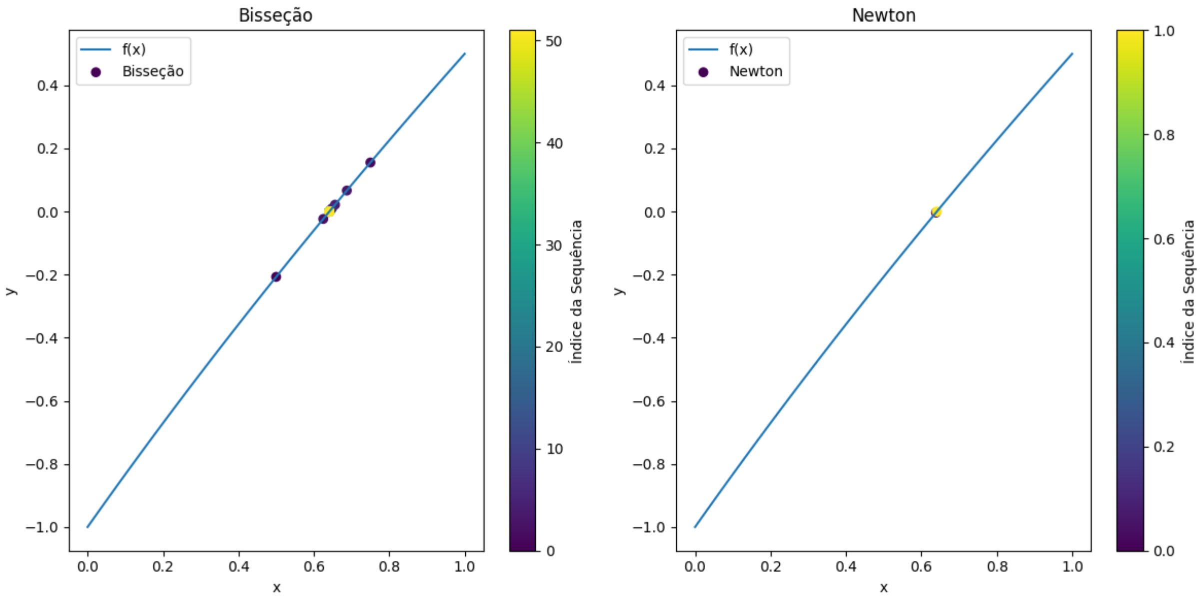
a) $x - 2^{-x} = 0, \quad [0, 1]$
b) $x + 1 - 2\,\text{sen}(\pi x) = 0, \quad [0, 0.5]$
c) $2x\cos(2x) - (x+1)^2 = 0, \quad [-3, -2]$
d) $\ln(x) - 2^x + x^2 = 0, \quad [3, 5]$

CÓDIGO:

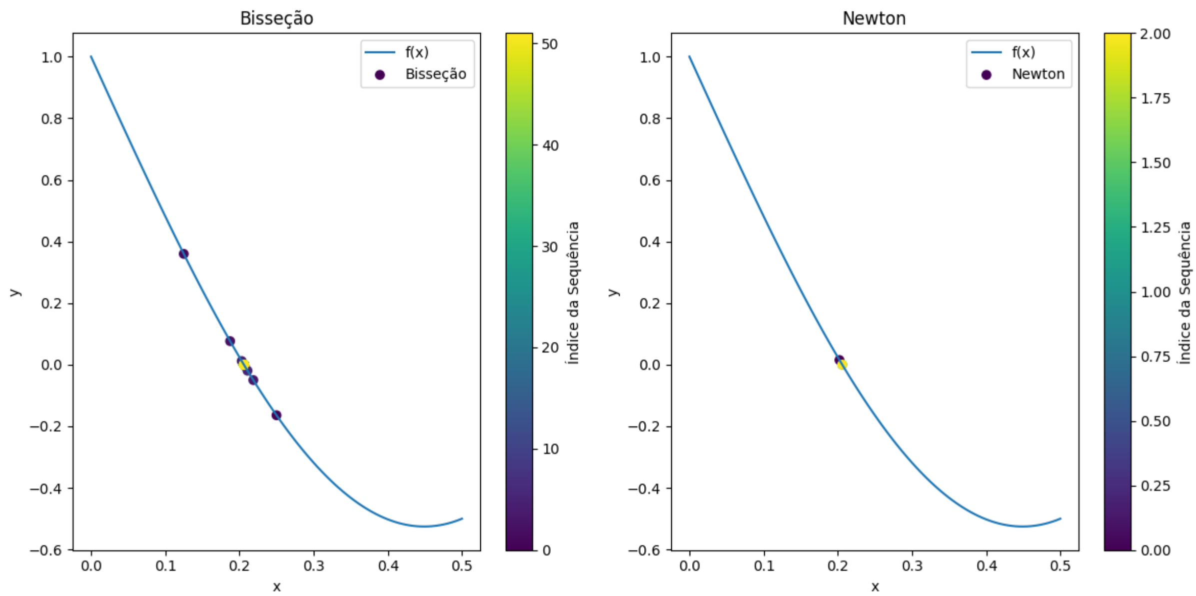
```python
import matplotlib.pyplot as plt
import math as m

def f_a(x):
    return x - 2**(-x)

def df_a(x):
    return (2**(x) + m.log(2)) / 2**(x)

def f_b(x):
    return x + 1 - 2 * m.sin(m.pi * x)

def df_b(x):
    return 1 - 2 * m.pi * m.cos(m.pi * x)

def f_c(x):
    return 2 * x * m.cos(2 * x) - (x + 1)**2

def df_c(x):
    return 2 * m.cos(2 * x) - 4 * x * m.sin(2 * x) - 2 * x - 2

def f_d(x):
    return m.log(abs(x)) - 2**x + x**2

def df_d(x):
    return (1/x) - (m.log(2) * (2)**x) + 2 * x

def MetodoBissecao(f, a, b, tolerance=1e-6, max_iterations=100):

    seq_aprox = []

    if f(a) * f(b) > 0:
        return []

    while max_iterations > 0:
        midpoint = (a + b) / 2
        seq_aprox.append(midpoint)
        if f(midpoint) == 0 or midpoint < tolerance:
            break  # Encontrou uma raiz exata ou convergiu
        elif f(midpoint) * f(a) < 0:
            b = midpoint
        else:
            a = midpoint

        max_iterations -= 1

    return seq_aprox

def MetodoNewton(f, df, x0, tolerance=1e-6, max_iterations=100):

    seq_aprox = []
    x = x0

    while max_iterations > 0:
        x = x - f(x) / df(x)
        seq_aprox.append(x)

        if abs(x - x0) < tolerance or abs(f(x)) < tolerance:
            break  # Convergiu
        elif abs(f(x)) < tolerance:
            break  # Encontrou uma raiz exata

        max_iterations -= 1

    return seq_aprox

def linspace(a, b, n):
    return [a + i * (b - a) / n for i in range(n + 1)]


def Solve(f, df, intervalo):
    a, b = intervalo
    tol = 1e-6

    seq_aprox_bissecao = MetodoBissecao(f, a, b, tol)
    seq_aprox_newton = MetodoNewton(f, df, (a + b) / 2, tol)

    x_values = linspace(a, b, 100)
    y_values = [f(x) for x in x_values]

    # Plotar para o Método da Bisseção
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)  # Subplot 1 (1 linha, 2 colunas, primeiro subplot)
    plt.plot(x_values, y_values, label='f(x)')
    plt.scatter(seq_aprox_bissecao, [f(x) for x in seq_aprox_bissecao], c=range(len(seq_aprox_bissecao)),
    cmap='viridis', label='Bisseção')
    plt.colorbar(label="Índice da Sequência")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Bisseção')
    plt.legend()

    # Plotar para o Método de Newton
    plt.subplot(1, 2, 2)  # Subplot 2 (1 linha, 2 colunas, segundo subplot)
    plt.plot(x_values, y_values, label='f(x)')
    plt.scatter(seq_aprox_newton, [f(x) for x in seq_aprox_newton], c=range(len(seq_aprox_newton)),
    cmap='viridis', label='Newton')
    plt.colorbar(label="Índice da Sequência")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Newton')
    plt.legend()

    plt.tight_layout()  # Ajusta a layout para evitar sobreposição
    plt.show()


def main():
    Solve(f_a, df_a, (0, 1))
    Solve(f_b, df_b, (0, 0.5))
    Solve(f_c, df_c, (-3, -2))
    Solve(f_d, df_d, (3, 5))

main()
```

SAÍDA DO CÓDIGO:

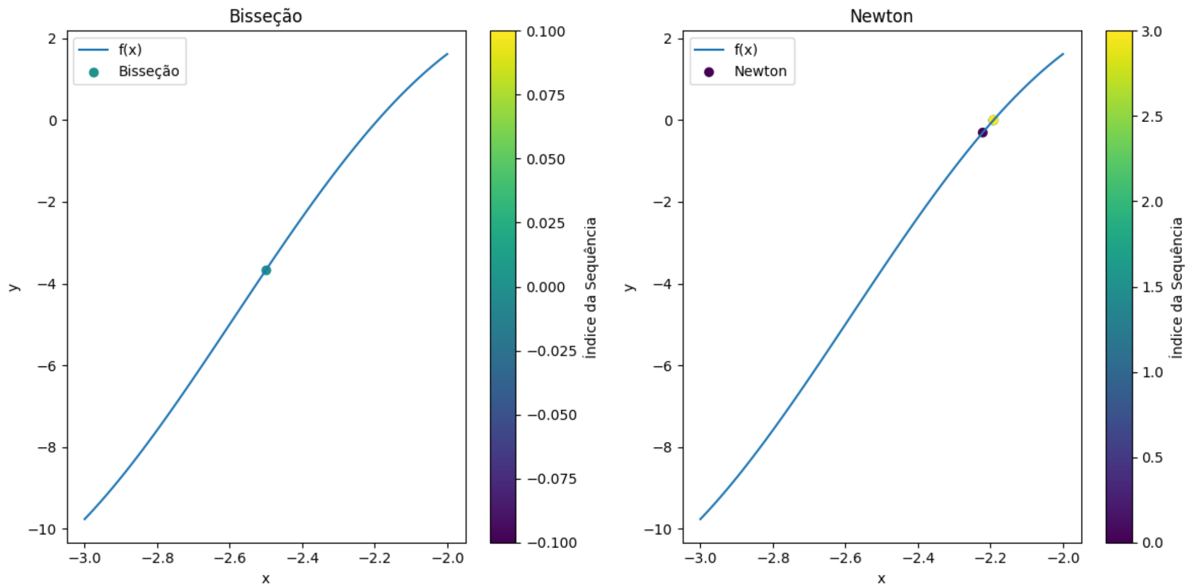
Letra a)



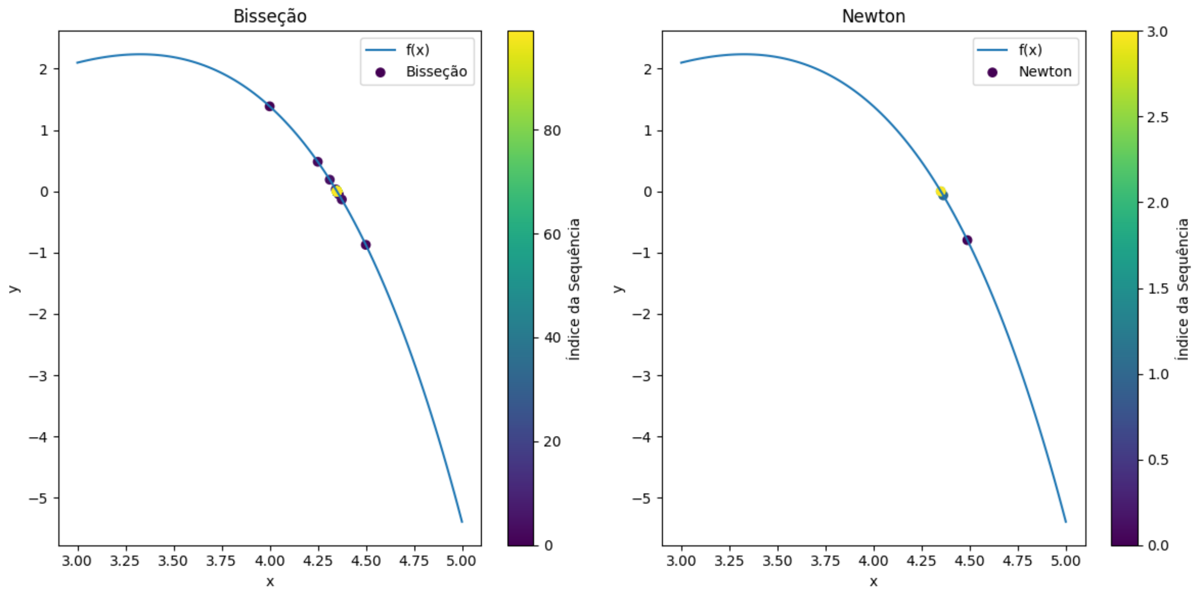
Letra b)



Letra c)



Letra d)



10. Determine o número mínimo de iterações necessário para resolver cada um dos itens da questão 9, com precisão absoluta 10^{-4}.

CÓDIGO:

```python
import matplotlib.pyplot as plt
import math as m

def f_a(x):
    return x - 2**(-x)

def df_a(x):
    return (2**(x) + m.log(2)) / 2**(x)

def f_b(x):
    return x + 1 - 2 * m.sin(m.pi * x)

def df_b(x):
    return 1 - 2 * m.pi * m.cos(m.pi * x)

def f_c(x):
    return 2 * x * m.cos(2 * x) - (x + 1)**2

def df_c(x):
    return 2 * m.cos(2 * x) - 4 * x * m.sin(2 * x) - 2 * x - 2

def f_d(x):
    return m.log(abs(x)) - 2**x + x**2

def df_d(x):
    return (1/x) - (m.log(2) * (2)**x) + 2 * x

def MetodoBissecao(f, a, b, tolerance=1e-6, max_iterations=100):

    seq_aprox = []

    if f(a) * f(b) > 0:
        return []

    while max_iterations > 0:
        midpoint = (a + b) / 2
        seq_aprox.append(midpoint)
        if f(midpoint) == 0 or midpoint < tolerance:
            break  # Encontrou uma raiz exata ou convergiu
        elif f(midpoint) * f(a) < 0:
            b = midpoint
        else:
            a = midpoint

        max_iterations -= 1


    return seq_aprox

def MetodoNewton(f, df, x0, tolerance=1e-6, max_iterations=100):

    seq_aprox = []
    x = x0

    while max_iterations > 0:
        x = x - f(x) / df(x)
        seq_aprox.append(x)

        if abs(x - x0) < tolerance or abs(f(x)) < tolerance:
            break  # Convergiu
        elif abs(f(x)) < tolerance:
            break  # Encontrou uma raiz exata

        max_iterations -= 1

    return seq_aprox


def Solve(f, df, intervalo, alternative):
    a, b = intervalo
    tol = 1e-4

    min_iterations_bissecao = len(MetodoBissecao(f, a, b, tol))
    min_iterations_newton = len(MetodoNewton(f, df, (a + b) / 2, tol))
    print(f"Para a função {alternative}: O método da bisseção convergiu ou chegou no limite de interações em: {min_iterations_bissecao} iterações.")
    print(f"Para a função {alternative}: O método de Newton convergiu em ou chegou no limite de interações em: {min_iterations_newton} iterações.\n")

def main():
    Solve(f_a, df_a, (0, 1), "A")
    Solve(f_b, df_b, (0, 0.5), "B")
    Solve(f_c, df_c, (-3, -2), "C")
    Solve(f_d, df_d, (3, 5), "D")

main()
```

SAÍDA DO CÓDIGO:

```
Para a função A: O método da bisseção convergiu ou chegou no limite de interações em: 52 iterações.
Para a função A: O método de Newton convergiu em ou chegou no limite de interações em: 2 iterações.

Para a função B: O método da bisseção convergiu ou chegou no limite de interações em: 52 iterações.
Para a função B: O método de Newton convergiu em ou chegou no limite de interações em: 2 iterações.

Para a função C: O método da bisseção convergiu ou chegou no limite de interações em: 1 iterações.
Para a função C: O método de Newton convergiu em ou chegou no limite de interações em: 3 iterações.

Para a função D: O método da bisseção convergiu ou chegou no limite de interações em: 100 iterações.
Para a função D: O método de Newton convergiu em ou chegou no limite de interações em: 4 iterações.
```