Resumos

Gauss

```
import numpy as np


def gauss_elimination(A, b):
    n = len(b)

    # Etapa de eliminação
    for pivot_row in range(n-1):  # Para cada linha pivô
        for row in range(pivot_row+1, n):  # Para cada linha abaixo da linha
pivô
            factor = A[row, pivot_row] / \
                A[pivot_row, pivot_row]  # Fator de eliminação
            A[row, pivot_row:] -= factor * A[pivot_row,
                                             pivot_row:]  # Operação de
eliminação
            b[row] -= factor * b[pivot_row]  # Operação de eliminação

    # Etapa de substituição de volta
    x = np.zeros(n) # Vetor de solução
    for i in range(n-1, -1, -1): # Para cada linha, de baixo para cima
        x[i] = (b[i] - np.sum(A[i, i+1:] * x[i+1:])) / A[i, i] # Operação de
substituição

    return x


# Sistema de equações a)
A_a = np.array([[3, 2, 4], [1, 1, 2], [4, 3, -2]], dtype=float)
b_a = np.array([1, 2, 3], dtype=float)
x_a = gauss_elimination(A_a, b_a)

print("Solução do sistema a):")
print("x =", x_a[0])
print("y =", x_a[1])
print("z =", x_a[2])

# Sistema de equações b)
A_b = np.array([[3, 2, 0, 1], [9, 8, -3, 4],
                [-6, 4, -8, 0], [3, -8, 3, -4]], dtype=float)
```

```
b_b = np.array([3, 6, 16, 18], dtype=float)
x_b = gauss_elimination(A_b, b_b)

print("\nSolução do sistema b):")
print("x =", x_b[0])
print("y =", x_b[1])
print("z =", x_b[2])
print("w =", x_b[3])


# TESTANDO AS SOLUÇÕES

# Sistema de equações a)
print("\nTestando as soluções do sistema a):")
print("3x + 2y + 4z =", 3*x_a[0] + 2*x_a[1] + 4*x_a[2])
print("x + y + 2z =", x_a[0] + x_a[1] + 2*x_a[2])
print("4x + 3y - 2z =", 4*x_a[0] + 3*x_a[1] - 2*x_a[2])

# Sistema de equações b)
print("\nTestando as soluções do sistema b):")
print("3x + 2y + w =", 3*x_b[0] + 2*x_b[1] + x_b[3])
print("9x + 8y - 3z + 4w =", 9*x_b[0] + 8*x_b[1] - 3*x_b[2] + 4*x_b[3])
print("-6x + 4y - 8z =", -6*x_b[0] + 4*x_b[1] - 8*x_b[2])
print("3x - 8y + 3z - 4w =", 3*x_b[0] - 8*x_b[1] + 3*x_b[2] - 4*x_b[3])
```

Saida

```
"""

Solução do sistema a):
x = -3.0
y = 5.0
z = 5.551115123125783e-17

Solução do sistema b):
x = 2.0
y = 7.0
z = 0.0
w = -17.0

Testando as soluções do sistema a):
3x + 2y + 4z = 1.0000000000000002
x + y + 2z = 2.0
4x + 3y - 2z = 3.0

Testando as soluções do sistema b):
3x + 2y + w = 3.0
9x + 8y - 3z + 4w = 6.0
-6x + 4y - 8z = 16.0
3x - 8y + 3z - 4w = 18.0
```

```
"""
```

Newton-Raphson

```
def newton_raphson(f, f_prime, x0, tol):
    """
    Implementa o método de Newton-Raphson.

    Args:
      f: A função a ser resolvida.
      f_prime: A derivada da função a ser resolvida.
      x0: A aproximação inicial.
      tol: O critério de parada.

    Returns:
      Uma aproximação da raiz.
    """

    x = x0
    while True:  # Loop infinito
        # Calcula a próxima aproximação -> significa que x' = x - f(x)/f'(x)
        x_prime = x - (f(x) / f_prime(x))
        if abs(f(x_prime)) < tol:  # Critério de parada
            return x_prime  # Retorna a aproximação da raiz
        x = x_prime  # Atualiza a aproximação


if __name__ == "__main__":
    def f(x): return 0.5 * x ** 2 - 1
    def f_prime(x): return x  # Derivada de f(x)
    x0 = (-2 + -1) / 2  # Ponto médio do intervalo [-2, -1]
    tol = 0.01  # Tolerância desejada
    x_star = newton_raphson(f, f_prime, x0, tol)  # Aproximação da raiz
    print("A raiz é", x_star)
```

Saída

```
"""A raiz é -1.4166666666666667"""
```

Lista PV1

```
def q1():
    def erro_absoluto(valor_real, valor_aproximado) -> float:
        return abs(valor_real - valor_aproximado)

    def erro_relativo(valor_real, valor_aproximado):
        return erro_absoluto(valor_real, valor_aproximado) /
float(valor_real)

    valor_real = 2.718281828
    valor_aproximado = 2.718

    calc_erro_absoluto = erro_absoluto(valor_real, valor_aproximado)
    calc_erro_relativo = erro_relativo(valor_real, valor_aproximado)

    print("Erro absoluto:", calc_erro_absoluto)
    print("Erro relativo:", calc_erro_relativo)


def q2():
    def erro_absoluto(valor_real, valor_aproximado):
        return abs(valor_real - valor_aproximado)

    def erro_relativo(valor_real, valor_aproximado):
        return erro_absoluto(valor_real, valor_aproximado) / valor_real

    valor_real = 96485.33289
    valor_aproximado = 96485

    calc_erro_absoluto = erro_absoluto(valor_real, valor_aproximado)
    calc_erro_relativo = erro_relativo(valor_real, valor_aproximado)

    print("Erro absoluto:", calc_erro_absoluto)
    print("Erro relativo:", calc_erro_relativo)


def q3():

    def f(x):
        return x**3 - 2*x**2 - 3*x + 1

    def encontrar_intervalos_com_raizes(a, b, passo):
        intervalos_com_raizes = []

        while a < b:
            fa = f(a)
            fb = f(a + passo)

            if fa * fb < 0:
                intervalos_com_raizes.append((a, a + passo))

            a += passo

        return intervalos_com_raizes

    a_inicial = -4
    b_inicial = 4
```

```
    passo = 1  # Ajuste conforme necessário

    intervalos = encontrar_intervalos_com_raizes(a_inicial, b_inicial,
passo)

    if len(intervalos) > 0:
        print("Intervalos com raízes encontrados:")
        for intervalo in intervalos:
            print(f"Intervalo: [{intervalo[0]}, {intervalo[1]}]")
    else:
        print("Não foram encontrados intervalos com mudança de sinal no
intervalo dado.")


def q4():

    def f(x):
        return x**3 - 2*x**2 - 3*x + 1

    def bisseccao(a, b, tolerancia, max_repeticoes):
        if f(a) * f(b) > 0:
            print(
                "Não há mudança de sinal no intervalo. O método da Bissecção
não se aplica.")
            return None

        for i in range(max_repeticoes):
            c = (a + b) / 2.0
            if f(c) == 0 or (b - a) / 2.0 < tolerancia:
                return c
            elif f(a) * f(c) < 0:
                b = c
            else:
                a = c

        return (a + b) / 2.0

    # Intervalo [a, b] com mudança de sinal
    a = -4
    b = 0
    tolerancia = 0.0001  # Tolerância desejada
    max_repeticoes = 4  # Máximo de repetições

    raiz_aproximada = bisseccao(a, b, tolerancia, max_repeticoes)

    if raiz_aproximada is not None:
        print(f"Raiz aproximada encontrada: {raiz_aproximada:.5f}")
    else:
        print("Não foi possível encontrar uma raiz no intervalo dado ou
atingir a tolerância desejada.")


def q5():

    def f(x):
        return x**3 - 2*x**2 - 3*x + 1
```

```
    def df(x):
        return 3*x**2 - 4*x - 3

    def newton_raphson(x0, tolerancia, max_repeticoes):
        for i in range(max_repeticoes):
            fx = f(x0)
            dfx = df(x0)

            if abs(fx) < tolerancia:
                return x0

            x1 = x0 - fx / dfx
            x0 = x1

        return x0

    # Valor inicial igual ao ponto médio do intervalo [0, 4]
    x0 = 2
    tolerancia = 0.0001  # Tolerância desejada
    max_repeticoes = 4  # Máximo de repetições

    raiz_aproximada = newton_raphson(x0, tolerancia, max_repeticoes)

    if raiz_aproximada is not None:
        print(f"Segunda raiz aproximada encontrada: {raiz_aproximada:.5f}")
    else:
        print(
            "Não foi possível encontrar uma segunda raiz ou atingir a
tolerância desejada.")


def q6():

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

    def encontrar_intervalos_com_raizes(a, b, passo):
        intervalos_com_raizes = []

        while a < b:
            fa = f(a)
            fb = f(a + passo)

            if fa * fb < 0:
                intervalos_com_raizes.append((a, a + passo))

            a += passo

        return intervalos_com_raizes

    # Intervalo inicial [-3, 3]
    a_inicial = -3
    b_inicial = 3
    passo = 0.1  # Ajuste conforme necessário

    intervalos = encontrar_intervalos_com_raizes(a_inicial, b_inicial,
passo)
```

```
    if len(intervalos) > 0:
        print("Intervalos com raízes encontrados:")
        for intervalo in intervalos:
            print(f"Intervalo: [{intervalo[0]:.2f}, {intervalo[1]:.2f}]")
    else:
        print("Não foram encontrados intervalos com mudança de sinal no
intervalo dado.")


def q7():
    def f(x):
        return -2*x**2 + 4*x + 2

    def bisseccao(a, b, tolerancia, max_repeticoes):
        if f(a) * f(b) > 0:
            print(
                "Não há mudança de sinal no intervalo. O método da Bissecção
não se aplica.")
            return None

        for i in range(max_repeticoes):
            c = (a + b) / 2.0
            if f(c) == 0 or (b - a) / 2.0 < tolerancia:
                return c
            elif f(a) * f(c) < 0:
                b = c
            else:
                a = c

        return (a + b) / 2.0

    # Intervalo [a, b] com mudança de sinal
    a = -3
    b = 0
    tolerancia = 0.0001  # Tolerância desejada
    max_repeticoes = 4  # Máximo de repetições

    raiz_aproximada = bisseccao(a, b, tolerancia, max_repeticoes)

    if raiz_aproximada is not None:
        print(f"Raiz aproximada encontrada: {raiz_aproximada:.5f}")
    else:
        print("Não foi possível encontrar uma raiz no intervalo dado ou
atingir a tolerância desejada.")


def q8():
    def f(x):
        return -2*x**2 + 4*x + 2

    def df(x):
        return -4*x + 4

    def newton_raphson(x0, tolerancia, max_repeticoes):
        for i in range(max_repeticoes):
            fx = f(x0)
```

```
            dfx = df(x0)

            if abs(fx) < tolerancia:
                return x0

            x1 = x0 - fx / dfx
            x0 = x1

        return x0

    # Valor inicial igual ao ponto médio do intervalo [0, 3]
    x0 = 1.5
    tolerancia = 0.0001  # Tolerância desejada
    max_repeticoes = 4  # Máximo de repetições

    raiz_aproximada = newton_raphson(x0, tolerancia, max_repeticoes)

    if raiz_aproximada is not None:
        print(f"Segunda raiz aproximada encontrada: {raiz_aproximada:.5f}")
    else:
        print(
            "Não foi possível encontrar uma segunda raiz ou atingir a
tolerância desejada.")


"""
1ª)Na solução de diversos problemas matemáticos, uma constante bastante
utilizada é o número de Neper,
um número irracional representado por e. Considerando apenas as nove casas
decimais da calculadora Cassio fx-82,
o mesmo pode ser representado por 2,718281828. Ao realizar certo
cálculo, se eu aproximar este valor dado
por 2,718, qual o erro absoluto e qual o erro relativo introduzido nessa
aproximação?
"""
print("Questão 1")
q1()

"""
Questão 1
Erro absoluto: 0.00028182799999987296
Erro relativo: 0.00010367872716392709
"""

"""
2ª)No estudo da eletroquímica, o valor 96485,33289 C/mol é conhecido por
constante de Faraday. Ao realizar certo
cálculo, se eu aproximar este valor dado por 96485,qual o erro
absoluto
e qual o erro relativo introduzido nessa aproximação?
"""
print("\nQuestão 2")
q2()

"""
Questão 2
Erro absoluto: 0.33289000000513624
```

```
Erro relativo: 3.4501616985107365e-06
"""

"""
(Enunciado para as questões 3a 5)Seja a função f(x) = x^3-2x^2-3x + 1.
Sabendo que esta função possui suas raízes no intervalo [-4, 4], responda:
"""

"""
3ª)Usando o método T.E.U., localize os intervalos que se encontram cada uma
das raízes reais da função.
"""
print("\nQuestão 3")
q3()

"""
Questão 3
Intervalos com raízes encontrados:
Intervalo: [-2, -1]
Intervalo: [0, 1]
Intervalo: [2, 3]

"""

"""
4ª)Usando o Método da  Bissecção e  um dos  intervalos  encontrados na
questão 3,
encontre  uma  raiz aproximada da função.Considerando no máximo 4repetições
(critério de parada).
"""
print("\nQuestão 4")
q4()
"""
Questão 4
Raiz aproximada encontrada: -1.12500

"""

"""
5ª)Usando o Método de Newton-Raphson e um outro dos intervalos encontrados
na questão 3,
encontre uma segunda raiz aproximada da função, utilizando como valor
inicial o
ponto médio do intervalo usado.Considerando no máximo 4repetições (critério
de parada).
"""
print("\nQuestão 5")
q5()
"""
Questão 5
Segunda raiz aproximada encontrada: 3.20867
"""

"""
(Enunciado para as questões 6a 8)Seja a função f(x) = -2^x2+ 4x + 2. Sabendo
que esta função possui suas raízes no intervalo [-3, 3], responda:
"""
```

```
"""
6ª)Usando o método T.E.U., localize os intervalos que se encontram cada uma
das raízes reais da função.
"""
print("\nQuestão 6")
q6()

"""
Questão 6
Intervalos com raízes encontrados:
Intervalo: [-0.50, -0.40]
Intervalo: [2.40, 2.50]
"""


"""
7ª)Usando o Método da Bissecção e o primeiro dos intervalos encontrados na
questão 6, encontre uma raiz aproximada da função.
Considere no máximo4repetições (critério de parada).
"""
print("\nQuestão 7")
q7()
"""
Questão 7
Raiz aproximada encontrada: -0.46875

"""

"""
8ª) Usando o Método de Newton-Raphson e o segundo dos intervalos encontrados
na questão 6, encontre uma segunda raiz aproximada da função,
utilizando como valor inicial o ponto médio do intervalo usado.Considerando
no máximo 4repetições (critério de parada).
"""
print("\nQuestão 8")
q8()

"""

Questão 8
Segunda raiz aproximada encontrada: 2.41423
"""
```

Seidel e Jacobi

```
import numpy as np

solucao_final = np.array([1, -2, 1], dtype=float)
```

```
def gaus_jacobi():

    def gauss_jacobi(A, b, x0, epsilon, max_iterations):
        n = len(b)
        x = np.copy(x0)  # Vetor de solução inicial

        for iteration in range(max_iterations):
            x_old = np.copy(x)  # Copia o vetor de solução anterior
            for i in range(n):  # Para cada linha da matriz
                # Somatório dos elementos da linha
                sigma = np.dot(A[i, :n], x_old[:n])
                x[i] = (b[i] - sigma + A[i, i] * x_old[i]) / A[i, i]

            if np.linalg.norm(x - x_old, np.inf) < epsilon:  # Critério de
parada
                return x  # Retorna o vetor de solução

        return x

    # Sistema de equações
    A = np.array([[10, 2, 1], [1, 5, 1], [2, 3, 10]], dtype=float)
    b = np.array([7, -8, 6], dtype=float)

    # Vetor de solução inicial
    x0 = np.array([0, 0, 0], dtype=float)

    # Critério de parada e número máximo de iterações
    epsilon = 0.01
    max_iterations = 10

    # Resolver o sistema utilizando o método de Gauss-Jacobi
    solution = gauss_jacobi(A, b, x0, epsilon, max_iterations)

    # Imprimir a solução
    print("Solução do sistema: | Método de Gauss-Jacobi |")
    print("x =", round(solution[0], 4))
    print("y =", round(solution[1], 4))
    print("z =", round(solution[2], 4))

    x = round(solution[0], 4)
    y = round(solution[1], 4)
    z = round(solution[2], 4)

    print("Avaliando a solução:")
    print("10x + 2y + z =", 10*x + 2*y + z)
    print("x + 5y + z =", x + 5*y + z)
    print("2x + 3y + 10z =", 2*x + 3*y + 10*z)

    print("Diferença entre a solução encontrada e a solução real:")
    print("x =", x - solucao_final[0])
    print("y =", y - solucao_final[1])
    print("z =", z - solucao_final[2])


def gaus_seidel():

    def gauss_seidel(A, b, x0, epsilon, max_iterations):
```

```
        n = len(b)
        x = np.copy(x0)

        for iteration in range(max_iterations):
            x_old = np.copy(x)  # Copia o vetor de solução anterior
            for i in range(n):
                # Somatório dos elementos anteriores a x[i]
                sigma1 = np.dot(A[i, :i], x[:i])
                # Somatório dos elementos posteriores a x[i]
                sigma2 = np.dot(A[i, i+1:], x_old[i+1:])
                # Calcula o novo valor de x[i]
                x[i] = (b[i] - sigma1 - sigma2) / A[i, i]

            if np.linalg.norm(x - x_old, np.inf) < epsilon:  # Critério de
parada
                return x

        return x

    # Sistema de equações
    A = np.array([[10, 2, 1], [1, 5, 1], [2, 3, 10]], dtype=float)
    b = np.array([7, -8, 6], dtype=float)

    # Vetor de solução inicial
    x0 = np.array([0, 0, 0], dtype=float)

    # Critério de parada e número máximo de iterações
    epsilon = 0.01
    max_iterations = 10

    # Resolver o sistema utilizando o método de Gauss-Seidel
    solution = gauss_seidel(A, b, x0, epsilon, max_iterations)

    # Imprimir a solução
    print("Solução do sistema: | Método de Gauss-Seidel |")
    print("x =", round(solution[0], 4))
    print("y =", round(solution[1], 4))
    print("z =", round(solution[2], 4))

    x = round(solution[0], 4)
    y = round(solution[1], 4)
    z = round(solution[2], 4)

    print("Avaliando a solução:")
    print("10x + 2y + z =", 10*x + 2*y + z)
    print("x + 5y + z =", x + 5*y + z)
    print("2x + 3y + 10z =", 2*x + 3*y + 10*z)

    print("Diferença entre a solução encontrada e a solução real:")
    print("x =", x - solucao_final[0])
    print("y =", y - solucao_final[1])
    print("z =", z - solucao_final[2])



gaus_jacobi()
print('='*50)
```

```
gaus_seidel()
```

Saída:

```
"""
Solução do sistema: | Método de Gauss-Jacobi |
x = 1.0002
y = -1.9989
z = 1.0003
Avaliando a solução:
10x + 2y + z = 7.004499999999999
x + 5y + z = -7.993999999999999
2x + 3y + 10z = 6.0067
Diferença entre a solução encontrada e a solução real:
x = 0.00019999999999997797
y = 0.001100000000000101
z = 0.00029999999999996696
==================================================
Solução do sistema: | Método de Gauss-Seidel |
x = 1.0
y = -2.0002
z = 1.0
Avaliando a solução:
10x + 2y + z = 6.9996
x + 5y + z = -8.001
2x + 3y + 10z = 5.9994
Diferença entre a solução encontrada e a solução real:
x = 0.0
y = -0.00019999999999997797
z = 0.0

"""
```