João Pedro Martins Cruz. 202365552C
Júlia Zoffoli Caçador. 202365520B

Descrição e Objetivos

Durante esse trabalho, foi desenvolvido um código computacional para calcular pontos de integração da quadratura de Gauss em qualquer intervalo.

O código implementa um método numérico para a quadratura de Gauss, utilizando um processo iterativo baseado no Método de Newton para determinar os pesos w_i e os nós t_i da quadratura. Ele faz uso da decomposição LU para resolver sistemas lineares, garantindo maior precisão nos cálculos. Além disso, integra funções utilizando a Regra de 1/3 de Simpson para comparação com a solução analítica, permitindo avaliar a eficiência do método. Os testes realizados mostram que, à medida que N aumenta, a aproximação da quadratura de Gauss se torna mais precisa, reduzindo o erro e validando a eficácia da técnica.

Redirecionamento para o repositório no GitHub: Clique aqui para acessar

Algoritmos

Definição das Condições Iniciais

Algoritmo 1: Definição das condições iniciais w_0 e t_0

```
def calculo_w0_t0(a, b, N):
    w = [0] * N
    t = [0] * N

    mid = N // 2  # Ponto m dio (arredondado para baixo)

    for i in range(N):
        if i < mid:
            w[i] = (i + 1) * (b - a) / (2 * N)
            t[i] = a + (i + 1) * w[i] / 2
        elif i == mid and N % 2 != 0:
            # Caso especial para N  mpar : ponto m dio
            w[i] = (b - a) / N
            t[i] = (a + b) / 2
        else:
            # Espelha os valores da primeira metade
            mirrored_index = N - i - 1
            w[i] = w[mirrored_index]
            t[i] = (a + b) - t[mirrored_index]
```

```
    return w, t
```

Método de Aproximação das Integrais

Algoritmo 2: Método de Aproximação das Integrais com uma partição de $m = 1000$

```
def Simpson_1_3_Para_Integral(a, b, N):
    m = 1000
    h = (b - a) / m

    result = [0] * 2 * N
    c = 1

    # Para cada expoente de 0 at  2N-1
    for exp in range(0, 2 * N):
        integral = 0

        # Em intervalo sim trico e fun  o par, a integral vale 0
        if b == -a and (exp + 1) % 2 == 0:
            result[exp] = 0
            continue

        for i in range(0, m + 1):
            x = a + i * h

            # Ajuste de coeficientes
            if i == 0 or i == m:
                c = 1
            elif i % 2 == 0:
                c = 2
            else:
                c = 4

            integral += c * pow(x, exp)
        result[exp] = (h / 3) * integral

    return result
```

Montagem dos Sistemas de Equações

Algoritmo 3: Montagem dos Sistemas de Equações

```
def Definir_Funcoes(a, b, N):
    w_sym = sp.symbols(f'w0:{N}')
    t_sym = sp.symbols(f't0:{N}')

    # Calculando os valores de g_j
```

```
    g_values = Simpson_1_3_Para_Integral(a, b, N)

    funcoes = []
    for j in range(1, 2 * N + 1):
        expr = 0
        for i in range(N):
            expr += w_sym[i] * (t_sym[i] ** (j - 1))  # w_i * t_i^(j-1)

        # Subtraindo g_j
        expr -= g_values[j - 1]
        funcoes.append(expr)

    return funcoes
```

Aproximação de Derivadas pelo Método Quasi-Newton

Algoritmo 4: Aproximação de Derivadas pelo Método Quasi-Newton

```
def Aproxima_Derivada_QuasiNewton(w, t, N):
    e = 10e-9
    dxW = [[0] * N for _ in range(2 * N)]
    dxT = [[0] * N for _ in range(2 * N)]

    # Para a quantidade de equa  es 2N, o expoente de t varia de 0 a 2N-1
    for power in range(0, 2 * N):
        # Calcula as derivadas de todos os w e t com k indo de 0 a N
        for k in range(0, N):
            fx = 0
            fxW = 0
            fxT = 0

            # Para todos os wi e ti, identifica o elemento k que estamos derivando e faz os
            for i in range(0, N):
                # C lculo do fx normal
                Fa = w[i] * pow(t[i], power)
                fx += Fa

                # Quando estamos derivando o termo k
                if k == i:
                    fxW += (w[i] + e) * pow(t[i], power)
                    fxT += w[i] * pow((t[i] + e), power)
                else:
                    fxW += Fa
                    fxT += Fa

            # Matriz com 2N linhas e N colunas que guarda os N elementos derivados
            dxW[power][k] = (fxW - fx) / e
            dxT[power][k] = (fxT - fx) / e
```

```
    return dxW, dxT
```

Montagem da Matriz Jacobiana

Algoritmo 5: Montagem da Matriz Jacobiana de Derivadas Parciais

```
def Monta_Matriz(dW, dT, N):
    M = [[0] * 2 * N for _ in range(2 * N)]
    for linha in range(0, 2 * N):
        for k in range(0, N):
            M[linha][k] = dW[linha][k]
            M[linha][k + N] = dT[linha][k]
    return M
```

Método de Newton

Algoritmo 6: Método de Newton

```
def Metodo_Newton(a, b, N, TOL=1e-8, max_iter=100):
    w, t = calculo_w0_t0(a, b, N)

    for num_iter in range(max_iter):
        funcoes = Definir_Funcoes(a, b, N)

        # Calculando os valores de f(wk, tk) com os valores atuais de w e t
        f_values = np.array([float(f.subs({f'w{i}': w[i] for i in range(N)} | {f't{i}': t[i
        dW, dT = Aproxima_Derivada_QuasiNewton(w, t, N)
        J = np.array(Monta_Matriz(dW, dT, N))

        # Decomposi  o LU da matriz Jacobiana
        L, U = Decomposicao_LU(J)

        # Resolvendo o sistema linear J(wk, tk) * s = -f(wk, tk) usando decomposi  o LU
        s = ResolveSistemaLU(L, U, -f_values)

        w = [w[i] + s[i] for i in range(N)]
        t = [t[i] + s[i + N] for i in range(N)]

        # Norma infinito de s para crit rio de parada
        norma_s = max(abs(s))
        print(f"Itera  o {num_iter + 1}: Norma infinito de s = {norma_s}")

        # Crit rio de parada
        if norma_s < TOL:
            print("Converg ncia alcan ada!")
            break
```

```
    else:
        print("N mero m ximo de itera   es atingido sem converg ncia.")

    return w, t
```

Quadratura Gaussiana

Algoritmo 7: Quadratura Gaussiana

```
def Quadratura_Gaussiana(N, w, t, a, b, func):
    sum = 0
    for i in range(0, N):
        sum += w[i] * func(a, b, t[i])
    return sum
```

Resultados

Tabela 1: valores dos pesos e pontos de integração para o intervalo $[-1, 1]$ e N = 3.

i	$\mathbf{w_i}$	$\mathbf{t_i}$
1	0.55555556	-0.77459667
2	0.88888889	0.0
3	0.55555556	0.77459667

Tabela 2: valores dos pesos e pontos de integração para o intervalo $[-1, 1]$ e N = 4.

i	$\mathbf{w_i}$	$\mathbf{t_i}$
1	0.34785485	-0.86113631
2	0.65214515	-0.33998104
3	0.65214515	0.33998104
4	0.34785485	0.86113631

Tabela 3: valores dos pesos e pontos de integração para o intervalo $[-1, 1]$ e N = 5.

i	$\mathbf{w_i}$	$\mathbf{t_i}$
1	0.23692688	-0.90617985
2	0.47862867	-0.53846931
3	0.56888889	0.0
4	0.47862867	0.53846931
5	0.23692688	0.90617985

Tabela 4: Valores dos pesos e pontos de integração para o intervalo $[-3, 3]$ e N = 3.

i	$\mathbf{w_i}$	$\mathbf{t_i}$
1	1.66666667	-2.32379001
2	2.66666667	0.0
3	1.66666667	2.32379001

Tabela 5: Valores dos pesos e pontos de integração para o intervalo $[-3, 3]$ e N = 4.

i	$\mathbf{w_i}$	$\mathbf{t_i}$
1	1.04356454	-2.58340893
2	1.95643546	-1.01994313
3	1.95643546	1.01994313
4	1.04356454	2.58340893

Tabela 6: Valores dos pesos e pontos de integração para o intervalo $[-3, 3]$ e N = 5.

i	$\mathbf{w_i}$	$\mathbf{t_i}$
1	0.71078065	-2.71853954
2	1.43588601	-1.61540793
3	1.70666667	0.0
4	1.43588601	1.61540793
5	0.71078065	2.71853954

Tabela 7: comparação exata aproximada da integral da função $f(x) = exp(-x + 1)$ no intervalo $[-1, 1]$.

N	exata	aproximação	erro
1	6.3890561	5.43656366	0.14908187
2	6.3890561	6.36810821	0.00327872
3	6.3890561	6.38887816	2.785e-05
4	6.3890561	6.3890553	1.3e-07
5	6.3890561	6.3890561	5.2e-10
6	6.3890561	6.3890561	1.7e-10
7	6.3890561	6.3890561	1.7e-10

Tabela 8: comparação exata aproximada da integral da função $f(x) = exp(-3x + 3)$ no intervalo $[-3, 3]$.

N	exata	aproximação	erro
1	6.3890561	120.51322154	0.99777862
2	6.3890561	10881.23884626	0.79943007
3	6.3890561	35728.24339601	0.34143425
4	6.3890561	49512.27963246	0.08735811
5	6.3890561	53427.94182762	0.01518212
6	6.3890561	54147.7958676	0.00191332
7	6.3890561	54241.69211382	0.00018256

Tabela 9: comparação exata aproximada da integral da função $f(x) = exp(-x + 3)$ no intervalo $[-1, 3]$.

N	exata	aproximação	erro
1	53.59815	29.5562244	0.44855887
2	53.59815	51.54937983	0.03822464
3	53.59815	53.53034867	0.00126499
4	53.59815	53.5969482	2.24e-05
5	53.59815	53.59813676	2.5e-07
6	53.59815	53.59814993	1.24e-09
7	53.59815	53.59815003	-6.0e-10

Tabela 10: comparação exata aproximada da integral da função $f(x) = x^5$ no intervalo $[-1, 1]$.

N	exata	aproximação	erro
1	0.0	0.0	0.0
...	...	...	...
7	0.0	0.0	0.0

Tabela 11: comparação exata aproximada da integral da função $f(x) = x^5$ no intervalo $[-1, 3]$.

N	exata	aproximação	erro
1	121.333333	4.0	0.967032966
2	121.333333	92.88888889	0.23443223
3	121.333333	121.33333333	-2.75e-09
4	121.333333	121.33333333	-2.75e-09
5	121.333333	121.33333333	-2.75e-09
6	121.333333	121.33333333	-2.75e-09
7	121.333333	121.33333333	-2.75e-09