Axl Silva de Andrade

Exercícios - Métodos dos Elementos de Contorno

São João de Meriti

Agosto de 2025

Axl Silva de Andrade

Exercícios - Métodos dos Elementos de Contorno

Relatório apresentado como avaliação da disciplina de Métodos dos Elementos de Contorno do Mestrado em Modelagem Matemática e Computacional da Universidade Federal Rural do Rio de Janeiro.

Universidade Federal Rural do Rio de Janeiro

Mestrado em Modelagem Matemática e Computacional

São João de Meriti

Agosto de 2025

Resumo

Este trabalho apresenta um relatório consolidado de exercícios da disciplina IC-7215 - Métodos Numéricos Computacionais, abrangendo um amplo espectro de técnicas para a resolução de equações diferenciais. O estudo inicia-se com a análise de Equações Diferenciais Ordinárias (EDOs) através do Método de Euler para Problemas de Valor Inicial (PVI). Em seguida, avança para o domínio das Equações Diferenciais Parciais (EDPs), onde se exploram e comparam métodos clássicos, como as Diferenças Finitas, e técnicas sem malha, como o Método das Soluções Fundamentais (MSF) e a interpolação com Funções de Base Radial (FBR). Finalmente, o trabalho investiga os fundamentos do Método dos Elementos Finitos através da derivação da formulação fraca e da aplicação do Método de Galerkin a problemas de contorno.

Nesse escopo, a análise foca no rigor matemático por trás de cada método, investigando não apenas a obtenção de soluções, mas também as condições sob as quais elas são exatas ou aproximadas. A investigação inclui o cálculo do erro de truncamento, a análise de convergência, e o estudo de casos patológicos, como a interpolação da função de Weierstrass, que desafia os limites dos métodos de aproximação suaves. Adicionalmente, explora-se como a escolha do espaço de aproximação no Método de Galerkin impacta diretamente a exatidão da solução final.

Por fim, o relatório consolida a teoria com a prática através de implementações computacionais em ambiente Octave, demonstrando a geração de soluções numéricas, a visualização gráfica de resultados complexos e a comparação quantitativa entre diferentes abordagens. Isso permite uma avaliação crítica da adequação, precisão e limitações de cada técnica em função da natureza do problema matemático.

Palavras-chaves: Análise Numérica. Método de Euler. Equações Diferenciais Parciais. Método de Galerkin. Formulação Fraca. Método das Diferenças Finitas. Funções de Base Radial (FBR). Método das Soluções Fundamentais (MSF). Função de Weierstrass.

Sumário

1 Introdução

Este relatório técnico apresenta a resolução e análise de uma série de exercícios da disciplina IC-7215 - Métodos Numéricos Computacionais, consolidando a aplicação de um vasto espectro de técnicas para a solução de equações diferenciais ordinárias e parciais. O objetivo é não apenas implementar os métodos, mas também investigar com rigor matemático os seus fundamentos teóricos, limitações e domínios de aplicabilidade.

O trabalho está estruturado em capítulos que correspondem aos conjuntos de exercícios propostos. O primeiro capítulo aborda a solução numérica de um Problema de Valor Inicial (PVI) para Equações Diferenciais Ordinárias (EDOs), utilizando o Método de Euler como ponto de partida para a análise de erro e convergência.

Os capítulos seguintes avançam para o domínio das Equações Diferenciais Parciais (EDPs), onde se exploram e comparam métodos clássicos baseados em malha, como as Diferenças Finitas, e técnicas modernas sem malha (*meshless*), como o Método das Soluções Fundamentais (MSF) e a interpolação com Funções de Base Radial (FBR). Um destaque desta parte é a aplicação destes métodos a problemas desafiadores, incluindo a interpolação da função patológica de Weierstrass.

Subsequentemente, o relatório investiga os fundamentos do Método dos Elementos Finitos através da derivação da formulação fraca e da aplicação do Método de Galerkin a problemas de contorno unidimensionais, explorando casos onde a solução numérica é exata e outros onde ela representa a melhor aproximação possível dentro de um subespaço.

Ao longo do texto, a teoria é consolidada com implementações práticas em ambiente Octave, cujos códigos são fornecidos e discutidos, permitindo uma avaliação crítica da performance, precisão e robustez de cada abordagem.

2 Método de Euler para Problemas de Valor Inicial

2.1 Análise de um Problema de Valor Inicial (PVI)

Este capítulo aborda a resolução do PVI definido por:

$$y'(t) = y(t) - t^2 + 1, \quad t \in [0, 2] \tag{2.1}$$

$$y(0) = 0.5 \tag{2.2}$$

A análise envolve a derivação da solução exata, a implementação do Método de Euler, e uma comparação quantitativa dos resultados.

2.1.1 Fundamentação Teórica do Método de Euler

O Método de Euler é o mais fundamental dos métodos numéricos para EDOs. Sua derivação advém diretamente da expansão em série de Taylor da solução $y(t)$ em torno de um ponto t_i:

$$y(t_{i+1}) = y(t_i) + y'(t_i)(t_{i+1} - t_i) + \frac{y''(\xi_i)}{2}(t_{i+1} - t_i)^2, \quad \xi_i \in (t_i, t_{i+1})$$

Definindo o passo de integração como $h = t_{i+1} - t_i$ e substituindo a EDO, $y'(t_i) = f(t_i, y(t_i))$, na expansão, temos:

$$y(t_{i+1}) = y(t_i) + hf(t_i, y(t_i)) + \frac{y''(\xi_i)}{2}h^2$$

O termo $\frac{y''(\xi_i)}{2}h^2$ é o **erro de truncamento local**, que representa o erro cometido em um único passo ao se aproximar a função por sua tangente. Ele é da ordem de $\mathcal{O}(h^2)$. Ao truncar a série e desconsiderar este termo, e ao substituir a solução exata $y(t_i)$ por sua aproximação computacional w_i, obtemos a célebre fórmula de Euler:

$$w_{i+1} = w_i + hf(t_i, w_i) \tag{2.3}$$

A acumulação do erro local ao longo de $N = (t_{fim} - t_{inicio})/h$ passos resulta em um **erro de truncamento global** da ordem de $\mathcal{O}(h)$.

2.1.2 Solução Analítica (Exata)

A EDO (Eq. 2.1) é linear de primeira ordem. Sua solução exata, obtida via técnica do fator integrante e aplicando a condição inicial (Eq. 2.2), é:

$$y(t) = (t + 1)^2 - 0.5e^t \tag{2.4}$$

2.1.3 Implementação Numérica e Resultados

Aplicando o Método de Euler com passo $h = 0.2$, obtemos a solução aproximada w_i em cada passo t_i. A Tabela 1 compara os resultados, que foram gerados pelo código em `euler_pvi.m`.

Tabela 1 – Comparação entre a solução exata e a aproximada por Euler ($h = 0.2$).

t_i	w_i **(Aproximada)**	y_i **(Exata)**	**Erro Absoluto** $\lvert y_i - w_i \rvert$
0.0	0.5000	0.5000	0.0000
0.2	0.8000	0.8293	0.0293
0.4	1.1520	1.2141	0.0621
0.6	1.5504	1.6489	0.0985
0.8	1.9885	2.1272	0.1387
1.0	2.4582	2.6409	0.1827
1.2	2.9498	3.1799	0.2301
1.4	3.4518	3.7324	0.2806
1.6	3.9501	4.2835	0.3333
1.8	4.4281	4.8152	0.3871
2.0	4.8658	5.3055	0.4397

2.1.4 Código Computacional

O código em Octave utilizado para resolver o problema e gerar a tabela e os gráficos correspondentes está listado abaixo, e deve ser salvo como `euler_pvi.m`.

```
% ------------------------------------------------------------------
% Arquivo: euler_pvi.m
% Descricao: Solucao do PVI y' = y - t^2 + 1 via Metodo de Euler.
% ------------------------------------------------------------------
clear; clc; close all;

%% --- Definicoes do Problema ---
f = @(t, y) y - t.^2 + 1; % Funcao da EDO
y_exata = @(t) (t+1).^2 - 0.5*exp(t); % Solucao exata
t_inicio = 0;
t_fim = 2;
h = 0.2;
y0 = 0.5;

%% --- Implementacao do Metodo de Euler ---
t_euler = t_inicio:h:t_fim;
w_euler = zeros(size(t_euler));
w_euler(1) = y0;
for i = 1:length(t_euler)-1
    w_euler(i+1) = w_euler(i) + h * f(t_euler(i), w_euler(i));
end
```

```
22
23 %% --- Analise de Resultados ---
24 y_nos_pontos_euler = y_exata(t_euler);
25 erro_abs = abs(y_nos_pontos_euler - w_euler);
26
27 % Impressao da Tabela de Resultados para o console
28 fprintf('
    ======================================================================\n
    ');
29 fprintf('                Tabela Comparativa de Resultados (Euler)\n');
30 fprintf('
    ======================================================================\n
    ');
31 fprintf('  t_i    |    w_i (Aproximado)  |    y_i (Exato)    |    Erro
    Absoluto\n');
32 fprintf('
    ----------------------------------------------------------------------\n
    ');
33 for i = 1:length(t_euler)
34     fprintf('  %.1f    |       %.4f        |      %.4f       |       %.4f\n',
    ...
35             t_euler(i), w_euler(i), y_nos_pontos_euler(i), erro_abs(i));
36 end
37 fprintf('
    ======================================================================\n
    \n');
38
39 %% --- Plotagem Grafica ---
40 figure('Position', [100, 100, 800, 600]);
41 % Grafico 1: Solucoes
42 subplot(2, 1, 1);
43 t_plot = linspace(t_inicio, t_fim, 200);
44 plot(t_plot, y_exata(t_plot), 'b-', 'LineWidth', 2, 'DisplayName', '
    Exata');
45 hold on;
46 plot(t_euler, w_euler, 'r--o', 'DisplayName', 'Euler (h=0.2)');
47 title('Solucao Exata vs. Solucao Aproximada (Euler)');
48 xlabel('t');
49 ylabel('y(t)');
50 legend show; grid on;
51
52 % Grafico 2: Erro Absoluto
53 subplot(2, 1, 2);
54 plot(t_euler, erro_abs, 'k-s', 'DisplayName', 'Erro Absoluto');
55 title('Erro Absoluto |y_i - w_i|');
56 xlabel('t');
57 ylabel('Erro');
```

```
58 legend show; grid on;
```

Listing 2.1 – Solução do PVI via Método de Euler.

2.1.5 Relatório e Conclusão

O Método de Euler, por ser de primeira ordem (erro global $\mathcal{O}(h)$), acumula um erro de truncamento a cada passo, resultando em um desvio progressivo da solução exata, como evidenciado na Tabela 1 e nos gráficos gerados. A análise de convergência, realizada sem o conhecimento da solução exata, dependeria da comparação de resultados com passos sucessivamente menores (e.g., $h, h/2, h/4$) para verificar empiricamente a taxa de redução do erro.

3 MDF para a Equação de Laplace

3.1 Análise de um Problema de Valor de Contorno (PVC)

3.1.1 Enunciado do Problema

Analisar a solução da Equação de Laplace, $\nabla^2 u = 0$, em uma placa quadrada $x, y \in [0, 0.5]$ com condições de contorno dadas pela função $u(x, y) = 400xy$. O objetivo é explicar por que a solução numérica pelo Método das Diferenças Finitas (MDF) coincide com a solução exata.

3.1.2 Fundamentação Teórica do MDF para o Laplaciano

A aproximação do Laplaciano $\nabla^2 u = u_{xx} + u_{yy}$ por diferenças centradas é obtida pela expansão em série de Taylor de $u(x, y)$ em torno de um ponto (x_i, y_j):

$$u(x_i \pm h, y_j) = u(x_i, y_j) \pm h\frac{\partial u}{\partial x} + \frac{h^2}{2}\frac{\partial^2 u}{\partial x^2} \pm \frac{h^3}{6}\frac{\partial^3 u}{\partial x^3} + \frac{h^4}{24}\frac{\partial^4 u}{\partial x^4} + \mathcal{O}(h^5)$$
$$u(x_i, y_j \pm h) = u(x_i, y_j) \pm h\frac{\partial u}{\partial y} + \frac{h^2}{2}\frac{\partial^2 u}{\partial y^2} \pm \frac{h^3}{6}\frac{\partial^3 u}{\partial y^3} + \frac{h^4}{24}\frac{\partial^4 u}{\partial y^4} + \mathcal{O}(h^5)$$

Somando as expansões para $u(x_i + h, y_j)$ e $u(x_i - h, y_j)$ e isolando u_{xx}, obtemos a aproximação para a segunda derivada em x, com um erro de truncamento τ_x:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{(i,j)} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} - \underbrace{\left.\frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}\right|_{(\xi, y_j)}}_{\tau_x}$$

De forma análoga para y. Somando as duas aproximações, obtemos o estêncil de 5 pontos para o Laplaciano, cujo erro de truncamento total $\tau_{ij} = \tau_x + \tau_y$ é:

$$\tau_{ij} = -\frac{h^2}{12}\left(\frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial y^4}\right)\Bigg|_{(\xi_i, \eta_j)} + \mathcal{O}(h^4)$$

3.1.3 Análise do Erro e Relatório Final

Para a solução exata $u(x, y) = 400xy$, todas as suas derivadas de ordem 2 e superiores são identicamente nulas. Em particular, $\frac{\partial^4 u}{\partial x^4} = 0$ e $\frac{\partial^4 u}{\partial y^4} = 0$. Isso implica que o erro de truncamento τ_{ij} é **exatamente zero**.

3.1.4 Conclusão:

Quando o erro de truncamento é nulo, a equação de diferenças finitas é uma representação algébrica exata da EDP para esta função específica. O código `mdf_laplace.m`

implementa a solução numérica, que, como previsto pela teoria, coincide com a solução exata, desprezando erros de ponto flutuante da máquina.

3.1.5 Código Computacional

O código em Octave que monta e resolve o sistema linear para o MDF está listado abaixo.

```
% ---------------------------------------------------------------------
% Arquivo: mdf_laplace.m
% Descricao: Solucao da Equacao de Laplace em uma placa quadrada
%            via Metodo das Diferencas Finitas (MDF).
% ---------------------------------------------------------------------
clear; clc; close all;

%% --- Definicoes do Problema ---
L = 0.5; % Dimensao da placa (0 a L)
h = 0.125; % Passo da malha
N_int = (L/h) - 1; % Numero de pontos internos em uma direcao (3)
N_vars = N_int^2; % Numero total de incognitas (9)

% Solucao exata e condicoes de contorno
u_exata = @(x,y) 400 * x .* y;

%% --- Montagem do Sistema Linear A*w = b ---
A = zeros(N_vars, N_vars);
b = zeros(N_vars, 1);
x_grid = linspace(0, L, N_int+2);
y_grid = linspace(0, L, N_int+2);

% Mapeamento de (i,j) 2D para k 1D
map = @(i,j) (j-1)*N_int + i;

for j = 1:N_int % Loop em y
    for i = 1:N_int % Loop em x
        k = map(i,j);

        % Diagonal principal (4*w_ij)
        A(k,k) = 4;

        % Vizinho da direita (w_{i+1,j})
        if i < N_int
            A(k, map(i+1,j)) = -1;
        else % Fronteira Direita (x=L)
            b(k) = b(k) + u_exata(x_grid(i+2), y_grid(j+1));
        end

```

```
        % Vizinho da esquerda (w_{i-1,j})
        if i > 1
            A(k, map(i-1,j)) = -1;
        else % Fronteira Esquerda (x=0)
            b(k) = b(k) + u_exata(x_grid(1), y_grid(j+1));
        end

        % Vizinho de cima (w_{i,j+1})
        if j < N_int
            A(k, map(i,j+1)) = -1;
        else % Fronteira Superior (y=L)
            b(k) = b(k) + u_exata(x_grid(i+1), y_grid(j+2));
        end

        % Vizinho de baixo (w_{i,j-1})
        if j > 1
            A(k, map(i,j-1)) = -1;
        else % Fronteira Inferior (y=0)
            b(k) = b(k) + u_exata(x_grid(i+1), y_grid(1));
        end
    end
end

%% --- Resolucao e Analise ---
w_vec = A \ b;
W_aprox = reshape(w_vec, N_int, N_int)'; % Remodelar para matriz 2D

% Calcular solucao exata nos mesmos pontos para comparacao
W_exata = zeros(N_int, N_int);
for j=1:N_int
    for i=1:N_int
        W_exata(j,i) = u_exata(x_grid(i+1), y_grid(j+1));
    end
end
erro = abs(W_exata - W_aprox);

%% --- Exibicao dos Resultados ---
disp('Solucao Numerica (W_aprox):');
disp(W_aprox);
disp('Solucao Exata (W_exata):');
disp(W_exata);
disp('Erro Absoluto (deve ser proximo de zero):');
disp(erro);
fprintf('\nErro maximo encontrado: %e\n', max(erro(:)));

%% --- Plotagem da Superficie ---
figure;
```

```
87 [X, Y] = meshgrid(x_grid, y_grid);
88 U_surf = u_exata(X, Y); % A solucao eh exata em todo lugar
89 U_surf(2:end-1, 2:end-1) = W_aprox; % Preenche o interior com a solucao
       numerica
90 surf(X, Y, U_surf);
91 title('Superficie da Solucao Numerica (MDF)');
92 xlabel('x'); ylabel('y'); zlabel('u(x,y)');
```

Listing 3.1 – Solução da Equação de Laplace via MDF.

4 Interpolação com Funções de Base Radial (FBR)

4.1 Desafio: Interpolação da Função de Weierstrass

4.1.1 Enunciado do Problema

Interpolar pontos da função de Weierstrass, $W(x) = \sum a^n \cos(b^n \pi x)$, que é contínua mas não-diferenciável, comparando FBRs suaves (Gaussiana, Multiquádrica e Invérsa Quadrática).

4.1.2 Fundamentação Teórica das FBRs

Funções de Base Radial são funções $\phi : \mathbb{R}^+ \to \mathbb{R}$ cuja resposta depende apenas da distância a um ponto central, $\phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|)$. O interpolante é uma combinação linear $s(\boldsymbol{x}) = \sum \lambda_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|)$. A garantia de uma solução única para o sistema linear $\mathbf{A}\boldsymbol{\lambda} = \mathbf{y}$ está ligada à definibilidade da matriz de interpolação $\mathbf{A}$. FBRs como a Gaussiana e a Invérsa Quadrática são positivas definidas, garantindo que $\mathbf{A}$ seja invertível.

4.1.3 O Dilema do Parâmetro de Forma (ε):

Muitas FBRs dependem de um parâmetro de forma ε. A sua escolha representa um compromisso fundamental (conhecido como *trade-off* ou princípio da incerteza):

- $\varepsilon \to 0$ (FBRs "planas"): Leva a matrizes $\mathbf{A}$ severamente mal-condicionadas, mas tende a produzir interpolantes mais precisos.
- $\varepsilon \to \infty$ (FBRs "pontudas"): Leva a matrizes bem-condicionadas, mas o interpolante tende a decair rapidamente para zero longe dos centros, perdendo sua qualidade de aproximação.

A escolha ótima de ε é um problema em aberto e depende dos dados.

4.1.4 Análise de Resultados e Relatório

A interpolação com FBR produz uma aproximação C^∞ de uma função não-diferenciável. O resultado, visualizado nos gráficos gerados pelo código em `fbr_weierstrass.m`, captura as oscilações de baixa frequência, mas falha em representar a natureza fractal de

alta frequência. O exercício expõe uma limitação teórica fundamental: a incompatibilidade de suavidade entre o interpolante e a função alvo.

4.1.5 Código Computacional

```
% ------------------------------------------------------------------
% Arquivo: fbr_weierstrass.m
% Descricao: Interpolacao da funcao de Weierstrass com FBRs.
% ------------------------------------------------------------------
clear; clc; close all;

%% --- Parametros do Problema ---
a = 0.5;
b = 13;
N_pontos = 41;
epsilon = 2.0; % Parametro de forma

%% --- Funcao de Weierstrass (soma finita) ---
function y = weierstrass(x, a, b)
    y = zeros(size(x));
    for n = 0:100 % Soma ate n=100 e suficiente para convergencia
        y = y + a^n * cos(b^n * pi * x);
    end
end

%% --- Geracao dos pontos de interpolacao ---
x_dados = linspace(-1, 1, N_pontos)';
y_dados = weierstrass(x_dados, a, b);

%% --- Pontos para plotagem (malha fina) ---
x_plot = linspace(-1, 1, 1000)';
y_verdadeiro = weierstrass(x_plot, a, b);

%% --- Definicao das FBRs ---
rbf_ga = @(r, ep) exp(-(ep*r).^2);
rbf_mq = @(r, ep) sqrt(1 + (ep*r).^2);
rbf_iq = @(r, ep) 1 ./ (1 + (ep*r).^2);

fbrs = {rbf_ga, rbf_mq, rbf_iq};
nomes_fbrs = {'Gaussiana', 'Multiquadrica', 'Inversa Quadratica'};
estilos = {'g-', 'm--', 'c-.'};
y_interpolado = zeros(length(x_plot), length(fbrs));

%% --- Loop para calcular a interpolacao para cada FBR ---
fprintf('--- Analise de Condicionamento ---\n');
for k = 1:length(fbrs)
    rbf = fbrs{k};
```

```
    dist_matrix = abs(x_dados - x_dados');
    A = rbf(dist_matrix, epsilon);
    lambda = A \ y_dados;

    dist_plot = abs(x_plot - x_dados');
    phi_plot = rbf(dist_plot, epsilon);
    y_interpolado(:, k) = phi_plot * lambda;

    fprintf('Condicionamento da matriz A para FBR %s: %e\n', nomes_fbrs{k}, cond(A));
end

%% --- Plotagem dos Resultados ---
figure('Position', [100, 100, 1000, 700]);
hold on;
plot(x_plot, y_verdadeiro, 'k-', 'LineWidth', 2, 'DisplayName', 'Weierstrass (Verdadeira)');
plot(x_dados, y_dados, 'ro', 'MarkerFaceColor', 'r', 'DisplayName', 'Pontos de Interpolacao');

for k = 1:length(fbrs)
    plot(x_plot, y_interpolado(:,k), estilos{k}, 'LineWidth', 1.5, 'DisplayName', nomes_fbrs{k});
end

hold off;
title(['Interpolacao da Funcao de Weierstrass (N=', num2str(N_pontos), ', epsilon=', num2str(epsilon), ')']);
xlabel('x'); ylabel('W(x)');
legend('show', 'Location', 'southoutside', 'Orientation', 'horizontal');
grid on; ylim([-2, 2]);
```

Listing 4.1 – Interpolação da função de Weierstrass com FBRs.

Parte I

O Método de Galerkin e a Formulação Fraca

5 Aproximação de Problemas de Contorno 1D

5.1 Fundamentação Teórica

5.1.1 Formulação Fraca

A formulação fraca de um problema de contorno $\mathcal{L}u = f$ transforma a equação diferencial em uma equação integral através do método dos resíduos ponderados, $\int_\Omega w(\mathcal{L}u - f)d\Omega = 0$. A aplicação de integração por partes ("transferindo"uma derivada de u para w) reduz os requisitos de continuidade da solução e incorpora as condições de contorno naturais na formulação.

5.1.2 Ortogonalidade de Galerkin

O Método de Galerkin encontra uma solução aproximada u_h em um subespaço de dimensão finita S_h. Sua propriedade mais importante é a **ortogonalidade de Galerkin**: o erro $e = u - u_h$ é ortogonal ao subespaço de aproximação S_h na norma de energia $a(\cdot, \cdot)$:

$$a(e, v_h) = a(u - u_h, v_h) = 0, \quad \forall v_h \in S_h$$

Isso implica que a solução de Galerkin é a melhor aproximação possível de u dentro do subespaço S_h. Se a solução exata u pertencer ao subespaço ($u \in S_h$), então o erro deve ser nulo, $e = 0$.

5.2 Exercício 1: Aproximação Linear Exata

Para o problema $u'' = 0$ em $(1, 3)$ com $u'(1) = 2, u(3) = 1$, a aproximação de Galerkin com uma função linear ($S_h = P_1$) foi utilizada. A solução encontrada, $u_{aprox}(x) = 2x - 5$, coincidiu com a solução exata.

5.2.1 Relatório:

A coincidência ocorreu porque a solução exata $u(x) = 2x - 5$ pertence ao espaço de aproximação ($u \in P_1$). Pela propriedade de ortogonalidade de Galerkin, o erro é necessariamente nulo.

5.3 Exercício 2: Aproximação com Base Enriquecida

Repetindo o problema anterior com uma base quadrática ($S_h = P_2$), o método de Galerkin encontrou coeficientes que zeraram o termo quadrático, recuperando a mesma

solução exata $u_{aprox}(x) = 2x - 5$.

5.3.1 Relatório:

Isso demonstra a consistência do método. Enriquecer a base com termos desnecessários não prejudica a solução; o método projeta corretamente a solução exata no subespaço, atribuindo coeficientes nulos aos componentes ortogonais.

5.4 Exercício 3: Problema Não-Homogêneo e Solução Inexata

Para o problema $u'' = 2$ em $(0, 1)$ com $u'(0) = -2, u(1) = 0$, a solução exata é a função quadrática $u_{exata}(x) = (x - 1)^2$. A aproximação de Galerkin com uma base linear $(S_h = P_1)$ resultou em $u_{aprox}(x) = 1 - x$.

5.4.1 Relatório:

A solução foi inexata porque o espaço de aproximação era insuficiente para representar a solução real $(u_{exata} \notin P_1)$. A solução encontrada é a projeção ortogonal da solução exata no espaço P_1, sendo a melhor aproximação linear possível, mas não a solução verdadeira. Ela não satisfez a EDO nem a condição de contorno natural.

5.5 Código Computacional de Verificação

O código abaixo verifica os cálculos dos sistemas lineares dos exercícios.

```
% ------------------------------------------------------------------
% Arquivo: galerkin_systems.m
% Descricao: Solucao computacional dos sistemas lineares do
%            Metodo de Galerkin.
% ------------------------------------------------------------------
clear; clc;

%% --- Exercicio 2: Aproximacao Quadratica ---
fprintf('--- Exercicio 2: Aproximacao Quadratica ---\n');
% Sistema linear A*alpha = b para encontrar [alpha1; alpha2]
% Equacao 1: 2*a1 - 4*a2 = 4
% Equacao 2: -4*a1 + (32/3)*a2 = -8
A_ex2 = [2, -4; -4, 32/3];
b_ex2 = [4; -8];
alpha_ex2 = A_ex2 \ b_ex2;

fprintf('Coeficientes encontrados:\n');
fprintf('alpha_1 = %f\n', alpha_ex2(1));
fprintf('alpha_2 = %f (como esperado, e zero)\n', alpha_ex2(2));
```

```
u_str = sprintf('1 + %.1f*(x-3) + %.1f*(x-3)^2', alpha_ex2(1), alpha_ex2(2));
fprintf('Solucao: u(x) = %s = 2x - 5\n\n', u_str);

%% --- Exercicio 3: Problema Nao-Homogeneo ---
fprintf('--- Exercicio 3: Problema Nao-Homogeneo ---\n');
% Equacao: integral(w''*u')dx = 2*w(0) - integral(2*w)dx
% u(x) = a1*(x-1) -> u' = a1
% w(x) = x-1      -> w' = 1

% Lado Esquerdo da equacao (LHS): integral(1*a1)dx de 0 a 1 = a1
% Lado Direito da equacao (RHS): 2*w(0) - integral(2*w)dx
w = @(x) x-1;
integral_2w = integral(@(x) 2*w(x), 0, 1);
rhs = 2*w(0) - integral_2w;
alpha1_ex3 = rhs; % Como o coeficiente do LHS e 1

fprintf('Calculo do lado direito (RHS):\n');
fprintf('2*w(0) = 2*(%d) = %d\n', w(0), 2*w(0));
fprintf('integral(2w)dx de 0 a 1 = %f\n', integral_2w);
fprintf('RHS = %d - (%f) = %f\n', 2*w(0), integral_2w, rhs);
fprintf('Coeficiente encontrado: alpha_1 = %f\n', alpha1_ex3);
fprintf('Solucao: u(x) = %.1f*(x-1) = %.1f - %.1fx\n\n', ...
        alpha1_ex3, -alpha1_ex3, alpha1_ex3);
```

Listing 5.1 – Verificação computacional dos sistemas de Galerkin.

6 Soluções Fundamentais e o MSF

6.1 A Solução Fundamental do Laplaciano

A solução fundamental $G(\boldsymbol{x}, \boldsymbol{\xi})$ de um operador diferencial $\mathcal{L}$ pode ser interpretada como a resposta do sistema em um domínio infinito a uma fonte pontual de intensidade unitária, modelada pela distribuição Delta de Dirac, $\delta(\boldsymbol{x} - \boldsymbol{\xi})$.

6.1.1 Derivação em 1D e 3D

A derivação formal, apresentada anteriormente, leva às soluções:

- **1D:** $G(x, \xi) = -\frac{1}{2}|x - \xi|$
- **3D:** $G(\boldsymbol{x}, \boldsymbol{\xi}) = \frac{1}{4\pi\|\boldsymbol{x}-\boldsymbol{\xi}\|}$

6.2 Análise Comparativa de Métodos para PVC

6.2.1 Problema Teste e Métodos

Para comparar o MSF, MDF e FBR, foi definido um problema teste: $\nabla^2 u = 0$ em um disco de raio 1 com condição de Dirichlet $u(1, \theta) = \cos(2\theta)$, cuja solução exata é $u(x, y) = x^2 - y^2$.

6.2.2 Código Computacional para o MSF

O código abaixo implementa o MSF para o problema teste.

```
% -------------------------------------------------------------------
% Arquivo: msf_laplace_circle.m
% Descricao: Solucao da Equacao de Laplace no disco unitario
%            via Metodo das Solucoes Fundamentais (MSF).
% -------------------------------------------------------------------
clear; clc; close all;

%% --- Definicoes do Problema ---
N = 30; % Numero de pontos de contorno e de fontes
R_dominio = 1.0; % Raio do dominio (disco)
R_fonte = 2.0;   % Raio do circulo das fontes ficticias

% Solucao Fundamental 2D e Solucao Exata
G = @(p1, p2) -1/(2*pi) * log(norm(p1-p2));
```

```
u_exata = @(x,y) x.^2 - y.^2;

%% --- Geracao dos Pontos ---
theta = linspace(0, 2*pi, N+1)'; % N+1 para nao repetir o ultimo ponto
theta = theta(1:end-1); % Remove o ultimo ponto duplicado

% Pontos de Colocacao (na fronteira do dominio)
colloc_pts = R_dominio * [cos(theta), sin(theta)];

% Pontos Fonte (na fronteira ficticia)
source_pts = R_fonte * [cos(theta), sin(theta)];

%% --- Montagem do Sistema Linear A*c = b ---
A = zeros(N, N);
for i = 1:N
    for j = 1:N
        A(i,j) = G(colloc_pts(i,:), source_pts(j,:));
    end
end

% Vetor b com as condicoes de contorno
b = u_exata(colloc_pts(:,1), colloc_pts(:,2));

%% --- Resolucao e Analise de Erro ---
c = A \ b; % Coeficientes da combinacao linear

% Funcao para avaliar a solucao do MSF em qualquer ponto (x,y)
u_msf = @(x,y,sources,coeffs) ...
    arrayfun(@(px,py) sum(coeffs .* -log(sqrt((px-sources(:,1)).^2 + (py
   -sources(:,2)).^2)) / (2*pi)), x, y);

% Criar uma malha de pontos internos para testar o erro
[X, Y] = meshgrid(linspace(-R_dominio, R_dominio, 51));
R_test = sqrt(X.^2 + Y.^2);
interior_mask = R_test < 0.999; % Apenas pontos estritamente dentro do
   disco

X_int = X(interior_mask);
Y_int = Y(interior_mask);

% Avaliar solucoes e erro nos pontos internos
U_aprox_vals = u_msf(X_int, Y_int, source_pts, c);
U_exata_vals = u_exata(X_int, Y_int);
erro_abs = abs(U_aprox_vals - U_exata_vals);
rmse = sqrt(mean(erro_abs.^2));

fprintf('--- Analise de Erro do MSF ---\n');
```

```
fprintf('Numero de pontos (N): %d\n', N);
fprintf('Raio das fontes (Rs): %.1f\n', R_fonte);
fprintf('Erro Maximo Absoluto no interior: %e\n', max(erro_abs));
fprintf('Erro Quadratico Medio (RMSE) no interior: %e\n\n', rmse);

%% --- Plotagem dos Resultados ---
figure('Position', [100, 100, 1200, 500]);
% Plot da solucao aproximada
subplot(1, 2, 1);
U_plot = nan(size(X));
U_plot(interior_mask) = u_msf(X_int, Y_int, source_pts, c);
surf(X, Y, U_plot);
title('Solucao Aproximada (MSF)');
xlabel('x'); ylabel('y'); zlabel('u(x,y)');
axis equal; view(30, 30);

% Plot do erro absoluto
subplot(1, 2, 2);
Erro_plot = nan(size(X));
Erro_plot(interior_mask) = erro_abs;
surf(X, Y, Erro_plot);
title('Erro Absoluto |u_{exata} - u_{aprox}|');
xlabel('x'); ylabel('y'); zlabel('Erro');
axis equal; view(30, 30);
```

Listing 6.1 – Solução da Equação de Laplace no disco via MSF.

6.2.3 Relatório e Tabela Comparativa

A Tabela 2 sumariza a comparação qualitativa dos três métodos.

Tabela 2 – Comparação qualitativa dos métodos numéricos para o problema do disco.

Critério	MSF	MDF	FBR (Kansa)
Geração de Malha	Nenhuma	Obrigatória (2D/3D)	Nenhuma
Flexibilidade Geométrica	Excelente	Ruim	Excelente
Tipo de Matriz	Densa, Simétrica	Esparsa, Estruturada	Densa, Assimétrica
Precisão Típica	Muito Alta	Média/Baixa	Alta
Principal Vantagem	Simplicidade e Precisão	Matriz Esparsa	Generalidade
Principal Desvantagem	Fronteira Fictícia	Fronteiras Curvas	Param. de Forma ε

6.2.4 Conclusão:

Para o problema da Equação de Laplace em um domínio com fronteira suave, os métodos sem malha (MSF e FBR) são vastamente superiores ao MDF. O MSF é conceitualmente elegante e preciso para a Equação de Laplace, enquanto o Método de Kansa (FBR) é mais geral e aplicável a uma gama maior de equações.

7 Conclusão Geral

Este relatório explorou um conjunto diversificado de métodos numéricos, revelando uma hierarquia de complexidade, precisão e aplicabilidade. O Método de Euler, como representante de métodos para EDOs, serviu para ilustrar os conceitos fundamentais de erro local e global.

No âmbito das EDPs, a inadequação de métodos de malha rígida, como as Diferenças Finitas, para geometrias complexas foi contrastada com a flexibilidade e poder dos métodos sem malha (MSF e FBR). Estes, por sua vez, introduzem seus próprios desafios teóricos e práticos, como a escolha de parâmetros livres e a gestão do mal-condicionamento.

Finalmente, o estudo do Método de Galerkin proporcionou um vislumbre dos fundamentos do Método dos Elementos Finitos, destacando a importância crucial do espaço de aproximação. A capacidade do método de encontrar soluções exatas quando estas pertencem ao espaço de busca, e de fornecer a melhor aproximação possível em caso contrário, através da propriedade de ortogonalidade, representa um dos pilares da análise numérica moderna.

A jornada através destes exercícios demonstra que não existe um "melhor método"universal. A escolha eficaz de uma técnica numérica exige uma compreensão profunda tanto do problema matemático em questão quanto das propriedades teóricas — convergência, estabilidade e rigor — do método a ser aplicado.

Apêndices

APÊNDICE A – Códigos-Fonte Adicionais

Esta seção contém os códigos-fonte para os exercícios que não foram diretamente inseridos nos capítulos anteriores.

A.1 Código para Interpolação com FBR

```
% ---------------------------------------------------------------------
% Arquivo: fbr_weierstrass.m
% Descricao: Interpolacao da funcao de Weierstrass com FBRs.
% ---------------------------------------------------------------------
clear; clc; close all;

%% --- Parametros do Problema ---
a = 0.5;
b = 13;
N_pontos = 41;
epsilon = 2.0; % Parametro de forma

%% --- Funcao de Weierstrass (soma finita) ---
function y = weierstrass(x, a, b)
    y = zeros(size(x));
    for n = 0:100 % Soma ate n=100 e suficiente para convergencia
        y = y + a^n * cos(b^n * pi * x);
    end
end

%% --- Geracao dos pontos de interpolacao ---
x_dados = linspace(-1, 1, N_pontos)';
y_dados = weierstrass(x_dados, a, b);

%% --- Pontos para plotagem (malha fina) ---
x_plot = linspace(-1, 1, 1000)';
y_verdadeiro = weierstrass(x_plot, a, b);

%% --- Definicao das FBRs ---
rbf_ga = @(r, ep) exp(-(ep*r).^2);
rbf_mq = @(r, ep) sqrt(1 + (ep*r).^2);
rbf_iq = @(r, ep) 1 ./ (1 + (ep*r).^2);

fbrs = {rbf_ga, rbf_mq, rbf_iq};
nomes_fbrs = {'Gaussiana', 'Multiquadrica', 'Inversa Quadratica'};
estilos = {'g-', 'm--', 'c-.'};
```

```
y_interpolado = zeros(length(x_plot), length(fbrs));

%% --- Loop para calcular a interpolacao para cada FBR ---
fprintf('--- Analise de Condicionamento ---\n');
for k = 1:length(fbrs)
    rbf = fbrs{k};
    dist_matrix = abs(x_dados - x_dados');
    A = rbf(dist_matrix, epsilon);
    lambda = A \ y_dados;

    dist_plot = abs(x_plot - x_dados');
    phi_plot = rbf(dist_plot, epsilon);
    y_interpolado(:, k) = phi_plot * lambda;

    fprintf('Condicionamento da matriz A para FBR %s: %e\n', nomes_fbrs{k}, cond(A));
end

%% --- Plotagem dos Resultados ---
figure('Position', [100, 100, 1000, 700]);
hold on;
plot(x_plot, y_verdadeiro, 'k-', 'LineWidth', 2, 'DisplayName', 'Weierstrass (Verdadeira)');
plot(x_dados, y_dados, 'ro', 'MarkerFaceColor', 'r', 'DisplayName', 'Pontos de Interpolacao');

for k = 1:length(fbrs)
    plot(x_plot, y_interpolado(:,k), estilos{k}, 'LineWidth', 1.5, 'DisplayName', nomes_fbrs{k});
end

hold off;
title(['Interpolacao da Funcao de Weierstrass (N=', num2str(N_pontos), ', epsilon=', num2str(epsilon), ')']);
xlabel('x'); ylabel('W(x)');
legend('show', 'Location', 'southoutside', 'Orientation', 'horizontal');
grid on; ylim([-2, 2]);
```

Listing A.1 – Interpolação da função de Weierstrass com FBRs.

A.2 Código para Verificação de Galerkin

```
% ------------------------------------------------------------------
% Arquivo: galerkin_systems.m
% Descricao: Solucao computacional dos sistemas lineares do
%            Metodo de Galerkin.
% ------------------------------------------------------------------
clear; clc;
```

```
7
8  %% --- Exercicio 2: Aproximacao Quadratica ---
9  fprintf('--- Exercicio 2: Aproximacao Quadratica ---\n');
10 % Sistema linear A*alpha = b para encontrar [alpha1; alpha2]
11 % Equacao 1: 2*a1 - 4*a2 = 4
12 % Equacao 2: -4*a1 + (32/3)*a2 = -8
13 A_ex2 = [2, -4; -4, 32/3];
14 b_ex2 = [4; -8];
15 alpha_ex2 = A_ex2 \ b_ex2;
16
17 fprintf('Coeficientes encontrados:\n');
18 fprintf('alpha_1 = %f\n', alpha_ex2(1));
19 fprintf('alpha_2 = %f (como esperado, e zero)\n', alpha_ex2(2));
20 u_str = sprintf('1 + %.1f*(x-3) + %.1f*(x-3)^2', alpha_ex2(1), alpha_ex2
       (2));
21 fprintf('Solucao: u(x) = %s = 2x - 5\n\n', u_str);
22
23 %% --- Exercicio 3: Problema Nao-Homogeneo ---
24 fprintf('--- Exercicio 3: Problema Nao-Homogeneo ---\n');
25 % Equacao: integral(w''*u')dx = 2*w(0) - integral(2*w)dx
26 % u(x) = a1*(x-1) -> u' = a1
27 % w(x) = x-1      -> w' = 1
28
29 % Lado Esquerdo da equacao (LHS): integral(1*a1)dx de 0 a 1 = a1
30 % Lado Direito da equacao (RHS): 2*w(0) - integral(2*w)dx
31 w = @(x) x-1;
32 integral_2w = integral(@(x) 2*w(x), 0, 1);
33 rhs = 2*w(0) - integral_2w;
34 alpha1_ex3 = rhs; % Como o coeficiente do LHS e 1
35
36 fprintf('Calculo do lado direito (RHS):\n');
37 fprintf('2*w(0) = 2*(%d) = %d\n', w(0), 2*w(0));
38 fprintf('integral(2w)dx de 0 a 1 = %f\n', integral_2w);
39 fprintf('RHS = %d - (%f) = %f\n', 2*w(0), integral_2w, rhs);
40 fprintf('Coeficiente encontrado: alpha_1 = %f\n', alpha1_ex3);
41 fprintf('Solucao: u(x) = %.1f*(x-1) = %.1f - %.1fx\n\n', ...
42         alpha1_ex3, -alpha1_ex3, alpha1_ex3);
```

Listing A.2 – Verificação computacional dos sistemas de Galerkin.

A.3 Código para o Método das Soluções Fundamentais

```
1 % ------------------------------------------------------------------
2 % Arquivo: msf_laplace_circle.m
3 % Descricao: Solucao da Equacao de Laplace no disco unitario
4 %            via Metodo das Solucoes Fundamentais (MSF).
5 % ------------------------------------------------------------------
6 clear; clc; close all;
```

```
7
8  %% --- Definicoes do Problema ---
9  N = 30; % Numero de pontos de contorno e de fontes
10 R_dominio = 1.0; % Raio do dominio (disco)
11 R_fonte = 2.0;   % Raio do circulo das fontes ficticias
12
13 % Solucao Fundamental 2D e Solucao Exata
14 G = @(p1, p2) -1/(2*pi) * log(norm(p1-p2));
15 u_exata = @(x,y) x.^2 - y.^2;
16
17 %% --- Geracao dos Pontos ---
18 theta = linspace(0, 2*pi, N+1)'; % N+1 para nao repetir o ultimo ponto
19 theta = theta(1:end-1); % Remove o ultimo ponto duplicado
20
21 % Pontos de Colocacao (na fronteira do dominio)
22 colloc_pts = R_dominio * [cos(theta), sin(theta)];
23
24 % Pontos Fonte (na fronteira ficticia)
25 source_pts = R_fonte * [cos(theta), sin(theta)];
26
27 %% --- Montagem do Sistema Linear A*c = b ---
28 A = zeros(N, N);
29 for i = 1:N
30     for j = 1:N
31         A(i,j) = G(colloc_pts(i,:), source_pts(j,:));
32     end
33 end
34
35 % Vetor b com as condicoes de contorno
36 b = u_exata(colloc_pts(:,1), colloc_pts(:,2));
37
38 %% --- Resolucao e Analise de Erro ---
39 c = A \ b; % Coeficientes da combinacao linear
40
41 % Funcao para avaliar a solucao do MSF em qualquer ponto (x,y)
42 u_msf = @(x,y,sources,coeffs) ...
43     arrayfun(@(px,py) sum(coeffs .* -log(sqrt((px-sources(:,1)).^2 + (py
    -sources(:,2)).^2)) / (2*pi)), x, y);
44
45 % Criar uma malha de pontos internos para testar o erro
46 [X, Y] = meshgrid(linspace(-R_dominio, R_dominio, 51));
47 R_test = sqrt(X.^2 + Y.^2);
48 interior_mask = R_test < 0.999; % Apenas pontos estritamente dentro do
    disco
49
50 X_int = X(interior_mask);
51 Y_int = Y(interior_mask);
```

```
52
53 % Avaliar solucoes e erro nos pontos internos
54 U_aprox_vals = u_msf(X_int, Y_int, source_pts, c);
55 U_exata_vals = u_exata(X_int, Y_int);
56 erro_abs = abs(U_aprox_vals - U_exata_vals);
57 rmse = sqrt(mean(erro_abs.^2));
58
59 fprintf('--- Analise de Erro do MSF ---\n');
60 fprintf('Numero de pontos (N): %d\n', N);
61 fprintf('Raio das fontes (Rs): %.1f\n', R_fonte);
62 fprintf('Erro Maximo Absoluto no interior: %e\n', max(erro_abs));
63 fprintf('Erro Quadratico Medio (RMSE) no interior: %e\n\n', rmse);
64
65 %% --- Plotagem dos Resultados ---
66 figure('Position', [100, 100, 1200, 500]);
67 % Plot da solucao aproximada
68 subplot(1, 2, 1);
69 U_plot = nan(size(X));
70 U_plot(interior_mask) = u_msf(X_int, Y_int, source_pts, c);
71 surf(X, Y, U_plot);
72 title('Solucao Aproximada (MSF)');
73 xlabel('x'); ylabel('y'); zlabel('u(x,y)');
74 axis equal; view(30, 30);
75
76 % Plot do erro absoluto
77 subplot(1, 2, 2);
78 Erro_plot = nan(size(X));
79 Erro_plot(interior_mask) = erro_abs;
80 surf(X, Y, Erro_plot);
81 title('Erro Absoluto |u_{exata} - u_{aprox}|');
82 xlabel('x'); ylabel('y'); zlabel('Erro');
83 axis equal; view(30, 30);
```

Listing A.3 – Solução da Equação de Laplace no disco via MSF.