UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE GEOCIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM GEOFÍSICA

Tópicos em Métodos Numéricos

Método de Resíduos Ponderados

Marcelo Lucas Almeida

marcelolucasif@gmail.com

Belém - Pará

20 de Janeiro de 2025

1 Introdução

O estudo das equações diferenciais é uma etapa fundamental na formação de cientistas e engenheiros. Por meio da formulação de princípios e leis físicas, diversos fenômenos naturais podem ser descritos matematicamente, e essa descrição muitas vezes conduz à formulação de equações diferenciais, sejam elas ordinárias ou parciais. Compreender esses tipos de equações e saber como resolvê-las é essencial para interpretar e modelar o mundo ao nosso redor.

Neste contexto, esta apostila tem como objetivo introduzir o estudante a algumas técnicas numéricas para a resolução de equações diferenciais com problema de valor de contorno, com foco nos métodos de resíduos ponderados, que constituem uma família de técnicas aproximadas para esse tipo de problema.

Apresentaremos os principais métodos desse grupo, como o Método de Colocação, o Método dos Mínimos Quadrados e o Método de Galerkin. Daremos ênfase especial a este último, por ser um ponto de partida importante para a compreensão de métodos mais avançados, como o Método dos Elementos Finitos (MEF).

Serão apresentadas as formulações matemáticas de cada método, sem, no entanto, exigir rigor formal nos conceitos teóricos envolvidos, de modo a facilitar a compreensão por parte do aluno.

Após a formulação matemática de cada técnica, incluiremos exemplos práticos com suporte computacional em linguagem Python, que tem se mostrado uma excelente escolha para aplicações em computação numérica e científica.

Como o foco desta apostila está nas técnicas numéricas, e não na linguagem de programação em si, será assumido que o leitor possui conhecimentos básicos de programação em Python. Para os que ainda não têm familiaridade com a linguagem, recomendamos as seguintes obras introdutórias: (**?**) e (**?**).

Para aqueles que desejam aprofundar seus conhecimentos em computação numérica com as bibliotecas NumPy, SciPy, Matplotlib e Pandas, recomendamos o livro de (**?**), que oferece uma excelente abordagem prática.

A idea principal usada no método de resíduos ponderados e substituir a nossa função incógnita $u(x)$ da nossa equação diferencial por um conjunto de funções linearmente independentes que aproximam a solução da nossa equação diferencial e minimizar o *resíduo* em um sentindo ponderado

Para entender com mais clareza o parágrafo anterior, partiemos de um problema piloto simples, porém, didático do ponto de vista da matemática.

Seja uma equação diferencial ordinária de 2 ordem linear, de coeficientes não constante e não homogênea sobre algum domínio Ω da reta real $\mathbb{R}$:

$$a(x)\frac{d^2u}{dx^2} + b(x)\frac{du}{dx} + c(x)u(x) = h(x) \quad \text{com} \quad u(a) = u_0, u(b) = u_1 \tag{1.1}$$

Com $a \leq x \leq b$. Para obtermos a expressão que caracteriza os métodos de ponderação de resíduos, trocamos $u(x)$ por um conjuntos de funções linearmnete independentes e que satsisfazem as condições de contorno do problema (**?**).

$$u(x) \approx u_n(x) \tag{1.2}$$

Onde $u_n(x)$ e a nossa solução aproximada do problema. Substituindo (1.2) em (1.1) não teremos mais uma igual, mas sim, um *resíduo* dado por :

$$R(x) = a(x)\frac{d^2u_n}{dx^2} + b(x)\frac{du_n}{dx} + c(x)u_n(x) - h(x) \neq 0 \tag{1.3}$$

A solução numérica do problema é obtida minimizando esse resíduo em um sentido ponderado :

$$\int_a^b R(x)w(x)dx = 0 \tag{1.4}$$

Onde $w(x)$ é a nossa *função peso*.

A forma como definimos $u_n(x)$ e a função $w(x)$ definem um técnica específica dos métodos de resíduos ponderados. Adiante veremos as principais técnicas usadas.

Observação

A ideias mostrada até o momento são facilmentes extendidas para resolver de forma numérica as equações diferencias em várias variáveis, ou seja, as Equações Diferencias Parciais (**?**), (**?**).

2 Método de Resíduos Ponderados

2.1 Método da Colocação - Problema Unidimensional

O método da Coloção é uma técnica pertencente aos métodos de ponderação de resíduos e é usada para se obter a solução aproximada de uma equação diferecial com condições de contorno, sendo ordinária ou parcial. Focaremos em desenvolver esse método aproximado para o caso mais simplório, ou seja, aplicando à equações diferencias ordinárias.

Consideremos o caso onde temos as condições de *contorno do tipo Direchlet Homogênea*, ou seja :

$$u(a) = u(b) = 0 \tag{2.1}$$

Partindo de uma equação diferencial de 2 ordem não homogênea :

$$a(x)\frac{d^2u}{dx^2} + b(x)\frac{du}{dx} + c(x)u(x) = h(x) \tag{2.2}$$

O resíduo será dado por :

$$R(x) = a(x)\frac{d^2u_n}{dx^2} + b(x)\frac{du_n}{dx} + c(x)u_n(x) - h(x) \tag{2.3}$$

Minimizando esse resíduo em um sentido ponderado, teremos :

$$\int_a^b R(x)w(x)dx = 0 \tag{2.4}$$

Agora espexíficamos a funçãao de peso $w(x)$, que no método da colocação é a *função generalizada de Dirac* (**?**).

$$\int_a^b R(x)\delta(x - x_i)dx = 0 \tag{2.5}$$

A equação (2.5) acima só será igual à zero, se o resíduo $R(x)$ for calculado nos pontos x_i.

$$R(x_i) = 0 \tag{2.6}$$

Os pontos x_i são conhecidos como *pontos de colocação* e podem ser gerados de várias maneiras ao londo do intervalo $[a, b]$: *pontos uniformemente espaçados*, *Polinômios ortogonais* e etc.

Como o resíduo é determinado pela equação (2.3), as funções de aproximação[1] $u_n(x)$ são um conjunto linearmente independentes que satisfazem as condições de contorno, ou seja :

[1]Também chamadas de *funções de base globais*.

$$u_n(x) = \sum_{j=1}^{n} c_j N_j(x) \tag{2.7}$$

Com

$$N_j(a) = N_j(b) = 0 \quad \text{para} \quad j = 1, 2, \ldots, n. \tag{2.8}$$

O sistema linear que é formado através da equação (2.6) depende da quantidade de pontos de colocação x_i. No caso de usarmos 2 pontos (x_1, x_2), temos :

$$\begin{bmatrix} \mathcal{L}(N_1(x_1)) & \mathcal{L}(N_2(x_1)) \\ \mathcal{L}(N_1(x_2)) & \mathcal{L}(N_2(x_2)) \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} h(x_1) \\ h(x_2) \end{bmatrix} \tag{2.9}$$

$$\mathbf{Kc} = \mathbf{b} \tag{2.10}$$

Onde temos que o símbolo $\mathcal{L}$ respresenta um operador diferencial de 2 ordem, que nesse caso em específico vale :

$$\mathcal{L}(N_1(x_1)) = a(x_1)\frac{d^2N_1}{dx^2}(x_1) + b(x_1)\frac{dN_1}{dx}(x_1) + c(x_1)N_1(x_1)$$

$$\mathcal{L}(N_2(x_1)) = a(x_1)\frac{d^2N_2}{dx^2}(x_1) + b(x_1)\frac{dN_2}{dx}(x_1) + c(x_1)N_2(x_1)$$

$$\mathcal{L}(N_1(x_2)) = a(x_2)\frac{d^2N_1}{dx^2}(x_2) + b(x_2)\frac{dN_2}{dx}(x_2) + c(x_2)N_1(x_2)$$

$$\mathcal{L}(N_2(x_2)) = a(x_2)\frac{d^2N_2}{dx^2}(x_2) + b(x_2)\frac{dN_2}{dx}(x_2) + c(x_2)N_2(x_2)$$

E assim por diante. Agora veremos como podemos aplicar o método a um problema de valor de contorno simples. Para isso, sempre iremos considerar no momento apenas condições de contorno homogêneas[2].

Seja a equação diferencial de 2 ordem :

$$\frac{d^2u}{dx^2} + u = x \quad \text{com} \quad u(0) = 0, u(1) = 0$$

Mediantes as técnicas aprendidas nos curso introdutórios de equações diferencias [3], obtemos a solução analítica da EDO acima, dada por :

[2]Quando estudarmos o método de Galerkin, veremos como podemos usar esse método para resolver condições de contorno não homogêneas.

[3]Para maiores detlahes sobre a teoria das equações diferencias ordinária e parciais, consulte as seguinte referências : (**?**), (**?**) e (**?**)

$$u(x) = x - \frac{\sin(x)}{\sin(1)}$$

Esse resultado pode ser facilmente obtido em python usando o módulo *SymPy*[4], que serve como uma *SAC* (Sistema de Computação Algébrica), que manipula símbolos de forma exata, similarmente como se estivessemos resolvendo o problema à mão pela matemática.

O seguinte código obtem a solução analítica da EDO do exemplo acima

Código Python

```
"""
============================================================
descrição : Solução da Equação Diferencial usando o SymPy
data : 05/01/25
programador : Marcelo L. Almeida
============================================================
"""
# modulos usados :
import numpy as np
import scipy as sp
from sympy import (symbols, diff,dsolve,Eq,init_printing,
    lambdify,sin,cos, integrate, Function)
import matplotlib.pyplot as plt
# deixando a saida simbolica em latex :
init_printing(use_latex= True)

## ===== codigo para resolução da edo ======== ##
# criando os símbolos
x = symbols("x", real = True)
u = Function("u")(x)

# criando a equação diferencial
eq_edo = Eq(diff(u,x,2) + u, x)

# criando as condições de contorno como dicionarios
pvc = {u.subs(x,0) : 0,
       u.subs(x,1) : 0}

# resolvendo a equação diferencial :
sol_analitic = dsolve(eq_edo,u, ics=pvc)

print(f"Solução␣Analítica␣:␣\n")

```

[4]Para aprender sobre esse magnífico pacote, conulte o seguinte endereço de sua documentação : SymPy

```
33 print(sol_analitic)
```

Ao executar o código acima, a saída que reprenseta a nossa solução analítica é :

Saída do Código

$$u(x) = x - \frac{\sin(x)}{\sin(1)}$$

Onde podemos observar que a solução obtida em Python bate com a nossa solução em (2.1).

Agora que estamos de posse da solução analítica, nos resta aplicar o método da colocação para aproximar a solução exata.

Antes disso, precisamos definir duas coisas importantes : *quais funções de base global usar ?* e *quais pontos de colocação* x_i *usar no intervalo de* $[0, 1]$?. Para responder a primeira pergunta, devemos considerar o seguinte Teorema

Teorema das Funções de Bases Globais

Para que as nossas funções de base aproximem de forma satisfatória as nossa solução analítica, elas devem satisfazer os seguintte itens :

1. **Propriedade da Completude** : As funções de aproximação devem representar de forma exata a solução analítica quando $n \longrightarrow \infty$. isso siginifica satisfazer :

$$\lim_{n \longrightarrow \infty} \left|\left| u(x) - \sum_{j=1}^{n} c_j \phi_j(x) \right|\right| = 0$$

2. **Linearidade** : As funções de base devem formar um conjunto de funções linearmente independentes dentro do intervalo de definição da nossa EDO/EDP.

3. **Satisfazer as Condições de Contorno** : O conjunto de funções L.I devem satisfazer todas as condições de contorno do porblema.

4. **Diferenciabilidade** : Se as nossa EDO/EDP contiver derivadas de ordem k, então as nossas funções de base devem ser no mínimo de classe $\mathcal{C}^k$(contínuas e diferenciáveis até a ordem k).

Mediante à esses fatos, podemos escolher o nosso conjunto por :

$$\phi_j(x) = x^j(1-x)$$

que para o nosso problema diferencial, satisfaz todas as condições de contorno :

$$\phi_j(0) = \phi_j(1) = 0 \quad \text{para} \quad j = 1, 2, \ldots, n.$$

Agora por último, precisamos decidir como gerar os pontos de colocação, que determinará o tamanho do sistema linear que será resolvido para obter a solução numérica do problema. Uma das forma mais eficientes de gerar os pontos de colocação para o método é usando os pontos de *chebyshev*.

Esse pontos são obtidos calculando as raízes dos polinômios de *chebyshev* $T_n(x)$, que são usualmentes usados em técnicas de quaadradura numérica ((**?**)) , (**?**) e (**?**).

Esse pontos são obtidos para um intervalo de $[-1, 1]$, mas usando a fórmula a seguir podemos gerar para um intervalo $[a, b]$.

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(2i-1)\,\pi}{2N}\right) \tag{2.11}$$

Agora que temos todas as ferramentas necessárias para aplicar o método aproximado, um exemplo de código a seguir é usado para computar a solução aproximada para o problema diferencial acima. Ao final do código, é plotado na mesma figura a solução analítica e numérica pelo método da colocação.

Código Python

```
"""
============================================================
descrição : Solução da Equação Diferencial usando o
método da colocação
data : 05/01/25
programador : Marcelo L. Almeida
============================================================
"""
import numpy as np
import scipy as sp
from sympy import (symbols, diff,dsolve,Eq,init_printing,
    lambdify,
                   sin,cos, integrate, Function, Matrix,
                       zeros)
import matplotlib.pyplot as plt
from scipy.special import roots_chebyc
# deixando a saida simbolica em latex :
init_printing(use_latex= True)
```

```
"""
============================================================
descrição :  Solução da Equação Diferencial usando o
método da colocação.
data : 05/01/25
programador : Marcelo L. Almeida
============================================================
"""
# criando os símbolos
x = symbols("x", real = True)
u = Function("u")(x)

# criando a equação diferencial
eq_edo = Eq(diff(u,x,2) + u, x)

# criando as condições de contorno como dicionarios
pvc = {u.subs(x,0) : 0,
       u.subs(x,1) : 0}

# resolvendo a equação diferencial :
sol_analitic = dsolve(eq_edo,u, ics=pvc)

# criando o vetor de pontos no eixo x :
a , b = 0 , 1
vetx = np.linspace(a,b,100)

# avaliando a nossa solução analitica nesse vetor
sol_analitic = lambdify(x,sol_analitic.rhs,"numpy")
u_analtic = sol_analitic(vetx)

## ================== computando o metodo da colocação
   ======== ##

N = 4  # quantidade de pontos de colocação
i = np.arange(1,N+1)
vet_xi = (a + b)/2 + ((b - a)/2)*np.cos(((2*i -1)/(2*N))*np.
   pi)

# criando as funções de base : phi_j
lista_phi_j = Matrix([x**(i+1)*(1 - x) for i in  range(N)])

K = zeros(N,N)
F = zeros(N,1)

# montando a matriz global do sistema :

for i in range(N) :

```

```
        for j in range(N) :

                expr = diff(lista_phi_j[j],x,2).subs(x,vet_xi[i
                    ]) + lista_phi_j[j].subs(x,vet_xi[i])

                K[i,j] = expr


for i in range(N) :

        F[i] = vet_xi[i]

# resolvendo o sistema linear usando a fatoração LU
cj = K.LUsolve(F)

## ================= montando a solução geral ============== ##

u_n = sum(cj[i]*lista_phi_j[i] for i in range(N))

u_n = lambdify(x,u_n,"numpy")

## ================= criando o plote ================== ##
fig, ax = plt.subplots(figsize = (8,6))

ax.plot(vetx, sol_analitic(vetx),"-b", label = "Solução Analítica", zorder = 1)
ax.scatter(vetx[::2], u_n(vetx[::2]), facecolors = "none", edgecolors = "r",zorder = 2,
           label = "Solução NUmérica (Colocação)")
ax.set_xlabel(r"$x$")
ax.set_ylabel(r"$u(x)$")
ax.set_title(r"Plote da Solução analítica e Numérica para a EDO : $\dfrac{d^2u}{dx^2} + u = x$")
ax.grid(True)
ax.legend(frameon = False , loc = 0)
ax.minorticks_on()

## ============ salvando os dados em arquivos ========== ##
M_analitic = np.vstack((vetx,u_analtic)).T
np.savetxt("dados_analitic_coloc.txt", M_analitic,fmt="%.5f", header="Solução analítica - Método da colocação")

M_numeric = np.vstack((vetx,u_n(vetx))).T
np.savetxt("dados_numeric_coloc.txt", M_numeric,fmt="%.5f", header="Solução numérica - Método da colocação")
```

Após executar o código acima, temos o seguinte gráfico na página a seguir, que mostra a solução analítica e numérica usando o método da colocação.

Na figura (1), temos em linha contínua e azul a solução analítica da equação diferencial e em círculos furados e vermelhos a solução numérica via colocação (apenas com alguns valores).

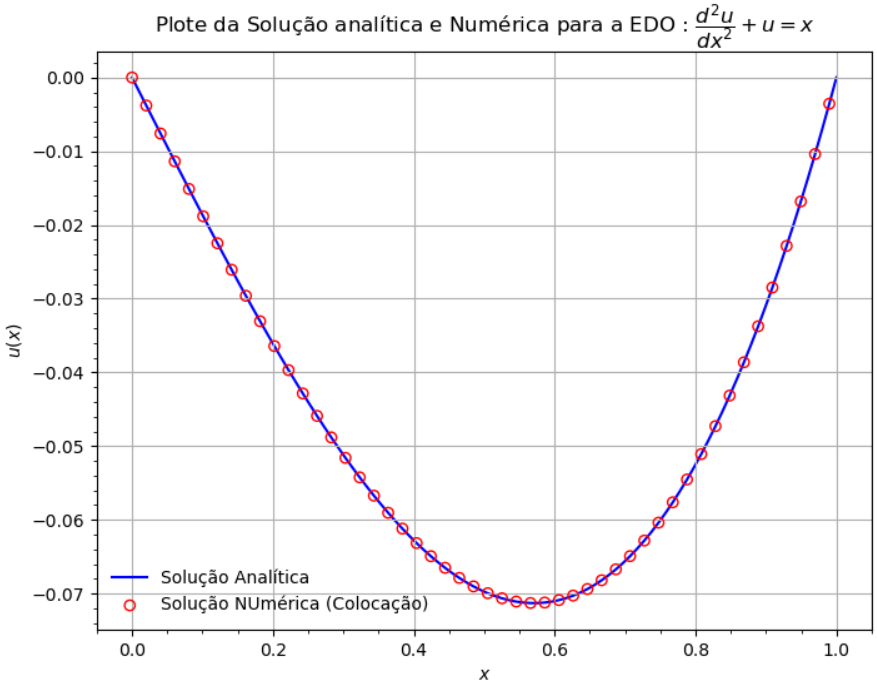

Figura 1: Solução Numérica e Analítica

Tão importante como a implementação do método numérico, é a implementação da análise do erro entre a solução exata e aproximada. Para isso, usaremos duas métricas para quantificar essa discrepância : *Erro Absoluto*.

O código a seguir cria a função para cálculo do erro e plota seus gráficos para comparação.

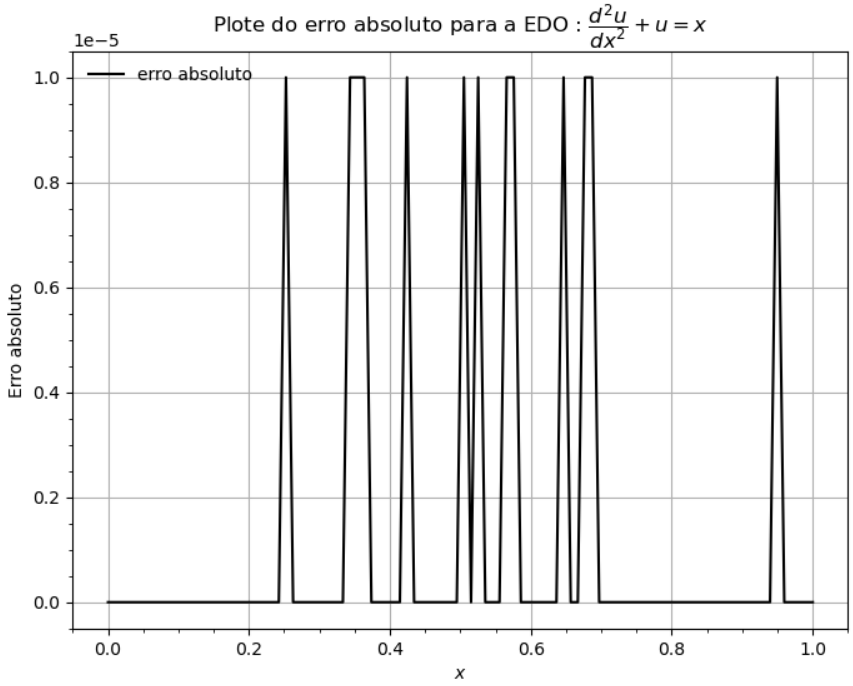

Figura 2: Erro Absoluto

2.2 Método de Mínimos Quadrados

Outro Método bastante importante é o ...