UNIVERSIDADE FEDERAL DO PARÁ

INSTITUTO DE GEOCIÊNCIAS

PROGRAMA DE PÓS-GRADUAÇÃO EM GEOFÍSICA

Tópicos em Métodos Computacionais

Método de Resíduos Ponderados Para Problemas de Valor de Contorno em EDOs

Marcelo Lucas Almeida

marcelolucasif@gmail.com

Belém - Pará

20 de Janeiro de 2025

Sumário

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: [7] e [10].

Para aqueles que desejam aprofundar seus conhecimentos em computação numérica com as bibliotecas NumPy, SciPy, Matplotlib e Pandas, recomendamos o livro de [5], 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 [11].

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

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 [11], [4].

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* [1].

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

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 : [13], [2] e [6]

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

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, cosulte 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 ([9]) , [3] e [8].

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.

No código a seguir usaremos o Sympy para realizar os cálculos matemáticos exegidos no método de aproximação. Como se trata de um exemplo relativamente simples, podemos usar o paradigma simbólico para computar as contas de forma exata, entretanto, para problemas mais desafiadores, a computação simbólica pode ser ineficiente e extremamente lenta, portanto , analisar o problema em questão é fundamental.

Polinômios de Chebyshev

Os polinômios de *Chebyshev* formam um conjunto ortogonal de funções com aplicações em teoria da aproximação e análise numérica. Existem dois tipos de polinômios de Chebyshev : **1° Tipo** ($T_n(x)$) e **2° Tipo** ($U_n(x)$).

Polinômios de Chebyshev de Primeiro Tipo:
Definidos por:

$$T_0(x) = 1, \quad T_1(x) = x$$
$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad (n \geq 1)$$

Propriedades: Ortogonais em $[-1, 1]$ com peso $w(x) = \dfrac{1}{\sqrt{1-x^2}}$:

$$\int_{-1}^{1} \frac{T_m(x)T_n(x)}{\sqrt{1-x^2}}dx = \begin{cases} 0 & n \neq m \\ \pi & n = m = 0 \\ \dfrac{\pi}{2} & n = m \neq 0 \end{cases}$$

medskip **Polinômios de Chebyshev de Segundo Tipo:**
Definidos por:

$$U_0(x) = 1, \quad U_1(x) = 2x$$
$$U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x) \quad (n \geq 1)$$

Propriedades: Ortogonais em $[-1, 1]$ com peso $w(x) = \sqrt{1-x^2}$:

$$\int_{-1}^{1} T_m(x)T_n(x)\sqrt{1-x^2}dx = \begin{cases} 0 & n \neq m \\ \dfrac{\pi}{2} & n = m \neq 0 \end{cases}$$

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 Analí
    tica", 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")

```

```
84 M_numeric = np.vstack((vetx,u_n(vetx))).T
85 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 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).

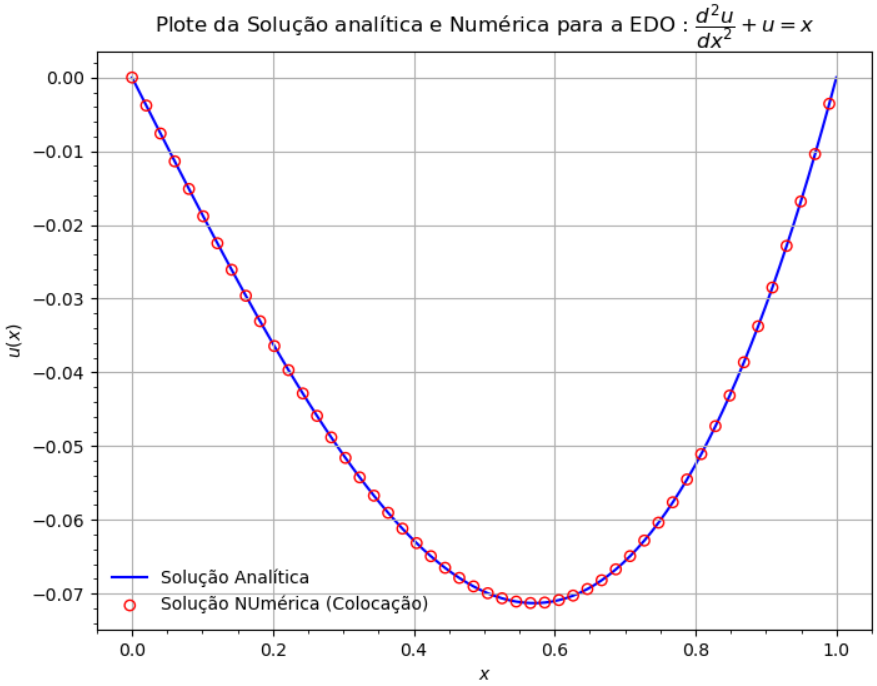

Figura 1: Solução Numérica e Analítica

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.

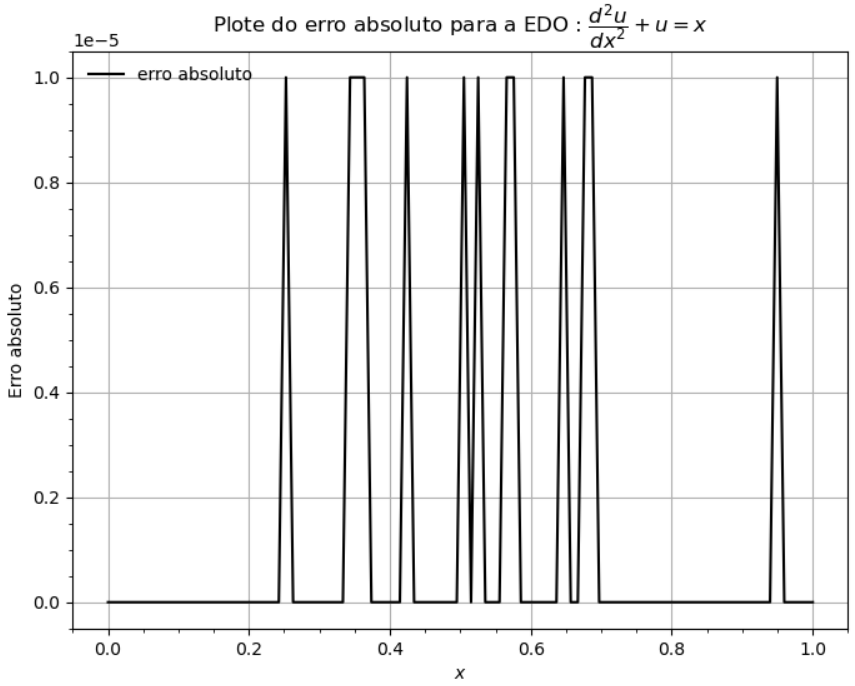

Figura 2: Erro Absoluto

2.2 Método de Mínimos Quadrados (MMQ)

Outro Método bastante importante para resolver equações diferencias Ordinária e Parciais é o **Método de Mínimos Quadrados**. Diferente do método da colocação visto anteriormente, o método MMQ minimiza a integral usando o *quadrado do resíduo* sobre o intervalo do domínio do problema diferecial. Para entendermos como fuciona esse método bem importante, consideramos a seguinte equação diferencial oridnária de 2 ordem genérica :

$$a(x)\frac{d^2u}{dx^2} + b(x)\frac{du}{dx} + c(x)u(x) = h(x) \quad \text{com} \quad u(a) = u_0, u(b) = u_1 \qquad (2.12)$$

Aqui supomos que aproximamos a solução $u(x)$ por um conjunto de funções de base linearmente independentes, que satisfazem as condições de contorno do problema, como fizemos no método da colocação. Assim teremos

um resíduo $R(x)$. No método de minimizamos o quadrado do resíduo, que é equivalente a usar a função peso $w(x) = R(x)$, ou seja :

$$I = \int_a^b R^2(x)dx = 0 \tag{2.13}$$

Que é equivalente a minimizar a norma L^2 do resíduo :

$$\min_{c_j} \int_a^b \left|\left|R(x)\right|\right|^2 dx \tag{2.14}$$

Isso é matematicamente equivalente À resolver o seguinte problema diferencial :

$$\frac{\partial}{\partial c_j}\left(\int_a^b \left|R(x)\right|^2 dx\right) = 0 \tag{2.15}$$

Substituindo a solução aproximada $u_n(x) = \sum_{j=1}^n c_j\phi_j(x)$ no resíduo da equação (2.15) e usando a condição de minimizção , teremos :

$$\int_a^b L(u_n(x) - h(x))L\left(\phi_k(x)\right)dx = 0 \quad \text{para} \quad k = 1, 2, \ldots, n. \tag{2.16}$$

Isso gera um sistema linear da seguinte forma :

$$\sum_{j=1}^n c_j \int_a^b L\left(\phi_j(x)\right) L\left(\phi_k(x)\right) dx = \int_a^b L\left(\phi_k(x)\right) h(x)dx \tag{2.17}$$

A equação (2.17) acima pode ser escrita mna forma matricial :

$$\mathbf{Ac} = \mathbf{b} \tag{2.18}$$

onde :

- **A** é a matriz do sistema, que possui a característica de ser simétrica e positiva definida.
- **c** é o vetor de incógnitas, que são os coeficientes c_j.
- **b** é o vetor de termos independentes, que são os produtos internos entre as funções de base e a função $h(x)$.

Mediante a essas características, podemos resolver o sistema linear usando a fatoração de Cholesky, que é uma técnica eficiente para resolver sistemas lineares com matrizes simétricas e positivas definidas. Essa fatoração decompõe a matriz **A** em um produto de uma matriz triangular inferior por sua transposta e resolve o sistema em duas etapas:

- Primeiro, resolvemos o sistema triangular inferior $\mathbf{Ly} = \mathbf{b}$.
- Em seguida, resolvemos o sistema triangular superior $\mathbf{L}^T\mathbf{c} = \mathbf{y}$.

Essa abordagem é mais eficiente do que usar métodos diretos, como a eliminação de Gauss, especialmente para matrizes grandes e esparsas. A seguir, apresentamos um exemplo de aplicação do método de mínimos quadrados para resolver uma equação diferencial de 2ª ordem. O código a seguir implementa o método de mínimos quadrados para resolver a equação diferencial dada por:

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

A seguri será mostrado o código em Python que implementa o método de Mínimos Quadrados para obter a solução aproximada do PVC. Como os códigos estão ficando cada ve maiores, vou dividir em partes o código completo, ficando apenas para de forma completa no pasta do método colocado no repositório do Github (repositório).

Módulos Python

```
"""
============================================================
descrição : Solução da Equação Diferencial usando o
método de minimos quadrados
data : 10/01/25
programador : Marcelo L. Almeida
============================================================
"""
import numpy as np
import matplotlib.pyplot as plt
import sympy as smp
from sympy import (symbols, diff, integrate, lambdify,
    symbols,
                   Eq, solve, dsolve, Function, Matrix, sin,
                       cos, exp,
                   init_printing)
import numpy.linalg as la
init_printing(use_latex=True)
```

Funções criadas Python

```

# criando a função para o operador L de segunda ordem

def L(phi) :
    """
```

```
    funlção que calcular o operador L sobre
    cada função de aproximação phi.
    """
    a_x = 1 # termo da segunda derivada
    b_x = 0  # termo da primeira derivada
    c_x = 1  # termo da função

    L_un = a_x*diff(phi,x,2) + b_x *diff(phi,x,1) + c_x * phi
    return L_un


# criando a função para verificar se
# uma matriz é simetrica
def is_symmetric(A):
    """
    Verifica se a matriz A é simétrica.
    """
    return np.allclose(A, A.T)

# criando uma função para
# verificar se é positiva definida

def is_positive_definite(A):
    """
    verific se a matriz A é definda possitiva
    """

    autovalores = la.eigvals(A)

    # teste de verificação

    if (autovalores > 0).all() :

        return True
    else :

        return False
```

Código Principal Python

```
## ========== função principal ================ ##

if __name__ == "__main__" :

    x = symbols('x', real = True)

    ## solução analitica :
    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 a nossa funçções de base
    n = 3         # quantidade de termos usados
    a , b = 0, 1 # intervalo de integração
    lista_phi_j = [x**(i)*(1 - x)  for i in range(1,n+1)] #
       lista de funções phi
    h_x = x       # função fonte
    # criando o sistema de equações :
    A = np.empty((n,n), dtype= np.float64)
    b = np.empty((n,1), dtype=np.float64)

    # criando a matriz
    for i in range(n) :

        for j in range(n) :

            phi_i = L(lista_phi_j[i])  # operador L{phi_i(x)}
            phi_j = L(lista_phi_j[j])  # operador L{phi_j(x)}
            integrando = integrate(phi_i*phi_j, (x, 0, 1)).
               evalf()
            A[i,j] = np.float64(integrando)


    # criando o vetor b :
    for i in range(n) :

        phi_i = L(lista_phi_j[i])
        term = integrate(phi_i*h_x,(x,0,1))
        b[i] = np.float64(term)


    # vericando se a matriz é simétrica e positiva definida
    if is_symmetric(A) and is_positive_definite(A):
        print("A matriz A é simétrica e positiva definida.")

        # resolve usando a decomposição de cholesky
        L = la.cholesky(A)
        y = la.solve(L,b)
        ci = la.solve(L.T,y)


```

```
58
59      else:
60          print("A matriz A não é simétrica ou não é positiva
                definida.")
61          # resolve usando a decomposição LU
62          P, L, U = la.lu(A)
63          y = la.solve(L, b)
64          ci = la.solve(U, y)
65
66      # criando a função de aproximação
67      phi = sum(ci[i] * lista_phi_j[i] for i in range(n))
68
69
70      # ploteando a função de aproximação
71      x_vals = np.linspace(0, 1, 100)
72      sol_analitic = lambdify(x,sol_analitic.rhs,"numpy")
73      u_analtic = sol_analitic(x_vals)
74      phi_func = lambdify(x, phi[0], 'numpy')
75      y_vals = phi_func(x_vals)
76      plt.plot(x_vals,u_analtic, "-b", label = "Solução Exata",
77               zorder = 1)
78      plt.scatter(x_vals[::2], y_vals[::2], facecolors = "none"
            , edgecolors = "r",label='Aproximação- MMQ',
79                  s=20, zorder = 2)
80      plt.title('Função de Aproximação')
81      plt.xlabel('x')
82      plt.ylabel('phi(x)')
83      plt.legend(frameon = False , loc = 0)
84      plt.grid()
85      plt.show()
```

Após executaro código, temos a figura da solução via MMQ :

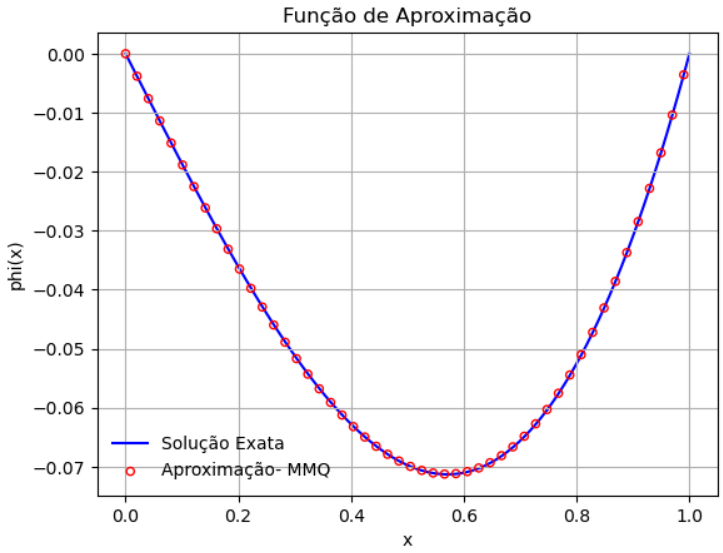

Figura 3: Solução Numérica e Analítica

2.3 Método de Galerkin

Nesse seção exploraremos o último e talvez o mais famoso método pertencente ao Resíduos Ponderados, que é o método de Galerkin. Esse método é a base para entender como métodos númericos mais sofisticados de resolução equações diferenciais são implementados, a exemplo, o *Método de Elementos Finitos* ou o *Método de Elementos de Contorno* [12].

Assim como nas seções anteriores, desenvolveremos a forma fraca do problema para esse método, e aqui, vale todo o esforço para se familiarizar com os passos fornecidos, pois, diferentes dos outros métodos apresentados, o método de Galerkin possui algumas especificidades bem importantes :

- As funções de ponderação nesse método *são as próprias funções de aproximação* $(\phi_i((x)))$
- Na dedução da forma fraca, um passo importante é o uso da *integração por partes*.[5] Esse passo é essencial no método de Galerkin, pois está ligado diretamente com as condições de continuidade e diferenciabilidade das funções de aproximação.

[5] A integração por partes é uma técnica do cálculo integral usada para resolver integrais definidas e indefinidas quando há um produto de funções de natureza distintas. Sua fórmula é

$$\int_a^b udv = uv\Big|_a^b - \int_a^b vdu$$

Desenvolveremos a forma fraca para o método de Galerkin considerando a seguinte EDO :

$$\frac{d^2u}{dx^2} + p(x)\left(\frac{du}{dx}\right) + q(x)u(x) = h(x) \tag{2.19}$$

com $x \in [0, L]$ e com as seguintes condições de contorno : $u(0) = u(L) = 0$.

Aproximando $u(x)$ como fizemos em seções anteriores :

$$u(x) \approx u_n(x) = \sum_{j=1}^{n} c_j \phi_j(x) \tag{2.20}$$

Então afirmamos que o *Resíduo* é *ortogonal* às funções de peso $w(x)$:

$$\int_0^L \left[\frac{d^2u_n}{dx^2} + p(x)\left(\frac{du_n}{dx}\right) + q(x)u_n(x)\right] w(x)dx = \int_0^L h(x)w(x)dx \tag{2.21}$$

Abrindo a equação acima para separar em várias intgerais, teremos :

$$\int_0^L \frac{d^2u_n}{dx^2}w(x)dx + \int_0^L p(x)\frac{du_n}{dx}w(x)dx + \int_0^L q(x)u_n(x)w(x)dx = \int_0^L h(x)w(x)dx \tag{2.22}$$

Agora um passo importante na derivação da forma fraca no método de Galerkin, que consiste na aplicação da regra de integração por partes na primeira integral do membro esquerdo acima, sendo assim, teremos que :

$$\int_0^L \frac{d^2u_n}{dx^2}w(x)dx = \left(w(x)\frac{du_n}{dx}\right)\Big|_0^L - \int_0^L \frac{du_n}{dx}\frac{dw}{dx}dx \tag{2.23}$$

$$\int_0^L \frac{d^2u_n}{dx^2}w(x)dx = \left(w(L)\frac{du_n}{dx}(L) - w(0)\frac{du_n}{dx}(0)\right) - \int_0^L \frac{du_n}{dx}\frac{dw}{dx}dx \tag{2.24}$$

Como por definição as funções de peso são nulas nas condições de contorno essenciais, portanto :

$$\int_0^L \frac{d^2u_n}{dx^2}w(x)dx = -\int_0^L \frac{du_n}{dx}\frac{dw}{dx}dx \tag{2.25}$$

Substituindo na equação original :

$$-\int_0^L \frac{du_n}{dx}\frac{dw}{dx}dx + \int_0^L p(x)\frac{du_n}{dx}w(x)dx + \int_0^L q(x)u_n w(x)dx = \int_0^L h(x)w(x)dx \tag{2.26}$$

Agora susbstituindo as funções de aproximação na equação acima e fazendo as funções de peso serem iguais as funções de teste, temos :

$$-\int_0^L \sum_{j=1}^{n} c_j \frac{d\phi_j(x)}{dx} \sum_{i=1}^{n} c_i \frac{d\phi_i(x)}{dx} dx + \int_0^L p(x) \sum_{j=1}^{n} c_j \frac{d\phi_j(x)}{dx} \sum_{i=1}^{n} c_i \phi_i(x) dx + \int_0^L q(x) \sum_{j=1}^{n} c_j \phi_j(x) \sum_{i=1}^{n} c_i \phi_i(x) dx = \int_0^L h(x) \sum_{i=1}^{n} c_i \phi_i(x) \quad (2.27)$$

para $i, j = 1, 2, \ldots, n$. A equação 2.27 é a forma fraca via método de Galerkin e pode ser representada por um sistema linear de equações da forma:

$$\boldsymbol{Kc} = \boldsymbol{b} \quad (2.28)$$

Onde :

$$k_{ij} = -\int_0^L \frac{d\phi_j(x)}{dx} \frac{d\phi_i(x)}{dx} dx + \int_0^L p(x) \frac{d\phi_j(x)}{dx} \phi_i(x) dx + \int_0^L q(x) \phi_j(x) \phi_i(x) dx \quad (2.29)$$

$$\boldsymbol{c} = [c_1 \ c_2 \ c_3 \ \ldots \ c_n]^T \quad (2.30)$$

$$b_i = \int_0^L h(x) \phi_i(x) \quad (2.31)$$

A utilição da integração por parte no método faz com que o sistema linear resultante seja do tipo *tridiagonal*[6], onde podemos usar apenas as três diagonais não nulas para resolver o sistema de forma mais eficiente computacionalmente.

[6]Um sistema matricial do tipo $\boldsymbol{Ax} = \boldsymbol{b}$, onde $\boldsymbol{A}$ é uma matriz tridiagonal é representado por :

$$A = \begin{bmatrix} d_1 & u_1 & 0 & \cdots & 0 & 0 \\ \ell_2 & d_2 & u_2 & \cdots & 0 & 0 \\ 0 & \ell_3 & d_3 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & u_{n-2} & 0 \\ 0 & \cdots & 0 & \ell_{n-1} & d_{n-1} & u_{n-1} \\ 0 & \cdots & 0 & 0 & \ell_n & d_n \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Referências

[1] Bassalo, J. M. F., and Cattani, M. S. D. Elementos de física matemática. *São Paulo: Editora Livraria da Física 2* (2011).

[2] Boyce, W. E., and DiPrima, R. C. *Equações Diferenciais Elementares e Problemas de Valores de Contorno*, vol. 10. LTC Rio de Janeiro, 2010.

[3] Chapra, S. C., and Canale, R. P. *Numerical Methods for Engineers*, vol. 1221. McGraw-Hill, 2011.

[4] Farlow, S. J. *Partial Differential Equations for Scientists and Engineers*. Courier Corporation, 1993.

[5] Johansson, R., and John, S. *Numerical Python*, vol. 1. Springer, 2019.

[6] Kreyszig, E. *Matemática Superior para Engenharia*. Livros Técnicos e Científicos, 2009.

[7] Matthes, E. *Python Crash Course: A Hands-On, Project-Based Introduction to Programming*. No Starch Press, 2023.

[8] Nakamura, J. *Applied Numerical Methods with Software*. Prentice Hall PTR, 1990.

[9] Press, W. H. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[10] Sweigart, A. *Automate the Boring Stuff with Python*. No Starch Press, 2025.

[11] Zienkiewicz, O. C., and Morgan, K. *Finite Elements and Approximation*. Courier Corporation, 2006.

[12] Zienkiewicz, O. C., Taylor, R. L., and Zhu, J. Z. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, 2005.

[13] Zill, D. G. *Equações Diferenciais com Aplicações em Modelagem*. Cengage Learning, 2016.