



INSTITUTO SUPERIOR TÉCNICO

PROJETO INTEGRADOR DE 1º CICLO

2º SEMESTRE - 2021/2022

LICENCIATURA EM MATEMÁTICA APLICADA E COMPUTAÇÃO

Método dos Elementos Finitos

Projeto realizado por:
Nuno MARQUES - 95758

Orientador:
Juha VIDEMAN

13 de setembro de 2022

Conteúdo

1	Introdução	2
2	Conceitos Teóricos	2
2.1	Problema de Poisson em 1D	2
2.2	Método dos elementos finitos em 1D	3
2.3	Análise do erro em 1D	4
2.4	Problema de Poisson em 2D	5
2.5	Método dos elementos finitos em 2D	5
3	Implementação do M.E.F em 1D	6
3.1	Assemblagem do sistema em 1D	6
3.2	Análise do erro empírico em 1D	10
4	Implementação do M.E.F em 2D	16
4.1	Assemblagem do sistema linear em 2D	16
4.2	Análise do erro empírico em 2D	20
5	Comparaçāo com o software <i>FreeFEM</i>	23
6	Conclusão	27
7	Apêndice Matemático	27
7.1	Espaços de Sobolev	27
7.2	Problema variacional e existência de solução única	28
7.3	Resultados sobre interpolação	29
8	Bibliografia	29

1 Introdução

Uma equação diferencial parcial (EDP) é uma equação que impõe uma relação entre as derivadas parcias de uma função multivariada, sendo característica na sua capacidade de modelação de certos fenómenos físicos tais como: o comportamento de fluídos, condutividade térmica e crescimento celular. No entanto, as soluções explícitas destas equações ora não existem ora são inalcançáveis ora são pouco eficientes de utilizar do ponto de vista computacional. Desta forma, o âmbito deste trabalho computacional é a aproximação numérica de soluções de equações diferenciais parcias através de um método chamado Método dos Elementos Finitos, que se baseia na discretização do espaço (e do tempo se a equação diferencial não for estacionária) em estudo em partes mais simples denominadas de elementos finitos. A fim de ilustrar este processo, é escolhida a equação de Poisson, com condição de fronteira de Dirichlet nula, como sendo o problema em que o método vai ser aplicado, expondo-se uma análise teórica e uma análise prática. No módulo teórico desenvolve-se a teoria necessária para a compreensão do relatório e é feita uma análise das estimativas de erro do método. O módulo prático foca-se na assemblagem em código, seguindo o desenvolvimento indicado no módulo teórico, e no estudo dos erros empíricos, observando-se a diferença entre uma solução conhecida e uma solução aproximada. Para encontrar a solução numérica de outras EDP's, é somente necessário retraçar os passos seguidos para encontrar a solução deste problema modelo.

Adicionalmente, a parte prática deste projeto foi composta utilizando a linguagem de programação e software *MATLAB*, usufruindo de uma biblioteca própria para análise de E.D.P's chamada *pde toolbox*. Os pedaços de código mais relevantes para o desenvolvimento do trabalho encontram-se inseridos neste documento assim como num ficheiro anexado, no qual se encontram outros *scripts* e funções auxiliares ao código. Para além disso, foi também adicionada uma secção na qual se compararam alguns resultados obtidos no desenvolver deste relatório com soluções aproximadas utilizando um software próprio para o uso deste método numérico, o *FreeFEM*, que é grátis e está disponível online.

2 Conceitos Teóricos

2.1 Problema de Poisson em 1D

Começa-se por constatar o problema de Poisson em **1D** e as suas respetivas formulações (forma fraca e forma forte). Efetivamente, quer-se encontrar uma solução $u \in C^2(0, 1) \cap C([0, 1])$ tal que:

$$\begin{aligned} -\frac{d^2u(x)}{dx^2} &= f \text{ em } (0, 1) \\ u(0) &= u(1) = 0 \end{aligned}$$

Esta é a chamada formulação *forte* do problema de Poisson, visto que exige uma solução que seja duas vezes diferenciável, pelo que o problema não tem solução para algumas funções relevantes. Desta forma, é desejável relaxar as condições do problema de modo a alargar o conjunto de funções para os quais se tem um problema bem definido. Neste sentido, multiplica-se ambos os lados por uma função teste φ , que é suficientemente regular e que se anula em $x=0$ e $x=1$.

$$-\frac{d^2u(x)}{dx^2} \cdot \varphi = f \cdot \varphi$$

Seguidamente, integra-se ambos os lados no intervalo:

$$\int_0^1 -\frac{d^2u(x)}{dx^2} \cdot \varphi \, dx = \int_0^1 f \cdot \varphi \, dx$$

Desenvolvendo através do Teorema Fundamental do Cálculo e integração por partes tem-se que:

$$\int_0^1 \frac{du(x)}{dx} \cdot \frac{d\varphi(x)}{dx} dx + \frac{du(1)}{dx} \cdot \varphi(1) - \frac{du(0)}{dx} \cdot \varphi(0) = \int_0^1 f \cdot \varphi dx$$

Dada as condições na fronteira, vai se ter então, finalmente, a forma *fraca* expressa por:

$$\int_0^1 \frac{du(x)}{dx} \cdot \frac{d\varphi(x)}{dx} dx = \int_0^1 f \cdot \varphi dx, \quad \forall \varphi \in V$$

É de notar que qualquer solução da forma *forte* é solução da forma *fraca*. Como espaço de funções V escolhe-se o espaço de *Sobolev* definido como:

$$H_0^1 := \{\varphi \in L^2(0, 1) \mid \int_0^1 \left(\frac{d\varphi(x)}{dx}\right)^2 dx < \infty \text{ e } \varphi(0) = \varphi(1) = 0\}$$

2.2 Método dos elementos finitos em 1D

Agora olha-se para o método dos elementos finitos em **1D**, que será usado para resolver numericamente a forma *fraca* do problema de Poisson limitado a um subespaço finito $V_h \subset V$. Neste sentido, o problema é transformado num sistema linear $A\beta = b$ em que $A \in \mathbb{R}^{n \times n}$ e $b \in \mathbb{R}^n$ com entradas $A_{ij} = \int_0^1 \frac{d\varphi_j}{dx} \cdot \frac{d\varphi_i}{dx} dx$ e $b_i = \int_0^1 f \cdot \varphi_i dx$ para um escolha de base apropriada $\{\varphi_1, \dots, \varphi_n\}$ de V_h . Para simplificação, considera-se, inicialmente, uma partição do espaço $(0, 1)$ em sub-intervalos (elementos) I_i tal que $I_i = (x_i, x_{i+1})$ e $0 = x_1 < x_2 < \dots < x_n = 1$, sendo que para construção da base se considera funções seccionalmente lineares, com as condições habituais de fronteira; formalmente é definido o espaço $V_h := \{u \in C[0, 1] \mid u(0) = u(1), u|I_i \in P^1(I_i) \text{ para } i = 1, \dots, (n-1)\}$. Repare-se que as condições impostas no primeiro e último nó complicam a assemblagem do sistema, pelo que se alarga o subespaço a $\hat{V}_h := \{u \in C[0, 1] \mid u|I_i \in P^1(I_i) \text{ para } i = 1, \dots, (n-1)\}$. Assim sendo, vão se ter as funções base $\{\hat{\varphi}_j\}_{j=1}^n$ definidas como:

$$\hat{\varphi}_j \in \hat{V}_h \quad \text{e} \quad \hat{\varphi}_j(x_p) = \begin{cases} 1, & \text{se } j = p \\ 0, & \text{caso contrário} \end{cases}$$

Portanto, uma base para o espaço V_h será dada por $V_h = \{\hat{\varphi}_2, \dots, \hat{\varphi}_{n-1}\}$, com funções de expressão:

$$\hat{\varphi}_j(x) = \begin{cases} \frac{x-x_{j-1}}{x_j-x_{j-1}}, & \text{se } x \in (x_{j-1}, x_j] \\ \frac{x_{j+1}-x}{x_{j+1}-x_j}, & \text{se } x \in (x_j, x_{j+1}) \\ 0, & \text{caso contrário} \end{cases}$$

Desta forma, está construído o espaço de polinómios seccionalmente lineares V_h , e, portanto, resta apenas computar as entradas da matrix \hat{A} e o vetor \hat{b} . Novamente, tem-se:

$$\hat{A}_{ij} = \int_0^1 \frac{d\hat{\varphi}_j}{dx} \cdot \frac{d\hat{\varphi}_i}{dx} dx = \sum_{k=1}^{N-1} \int_{I_k} \frac{d\hat{\varphi}_j}{dx} \cdot \frac{d\hat{\varphi}_i}{dx} dx$$

Ou seja, a matriz \hat{A} é computada através de um *loop* que vai sumando os valores dos integrais ao longo das respetivas partições. Repare-se que, num dado intervalo I_j apenas as funções base $\hat{\varphi}_j$ e $\hat{\varphi}_{j+1}$ têm valores diferentes de zero, pelo que apenas são avaliados os integrais $\int_{I_k} \frac{d\hat{\varphi}_j}{dx} \cdot \frac{d\hat{\varphi}_j}{dx} dx$, $\int_{I_k} \frac{d\hat{\varphi}_j}{dx} \cdot \frac{d\hat{\varphi}_{j+1}}{dx} dx$ e $\int_{I_k} \frac{d\hat{\varphi}_{j+1}}{dx} \cdot \frac{d\hat{\varphi}_{j+1}}{dx} dx$. Seguidamente os resultados são adicionados às entradas $(j, j)(j+1, j)(j, j+1)$ e $(j+1, j+1)$ de \hat{A} , tal que $\hat{A}_{lk} = \hat{A}_{lk} + \int_{I_k} \frac{d\hat{\varphi}_l}{dx} \cdot \frac{d\hat{\varphi}_k}{dx} dx$ para $l, k \in \{j, j+1\}$.

É de notar que se vai ter que $\hat{\varphi}_j|I_j = \frac{x_{j+1}-x}{x_{j+1}-x_j}$ e $\hat{\varphi}_{j+1}|I_j = \frac{x_j-x}{x_j-x_{j+1}}$. Por sua vez, para o vetor \hat{b} , em I_j , avalia-se os integrais $\int_{I_j} f \cdot \hat{\varphi}_j dx$ e $\int_{I_j} f \cdot \hat{\varphi}_{j+1} dx$, logo $\hat{b}_l = \hat{b}_l + \int_{I_j} f \cdot \hat{\varphi}_l dx$ para $l, k \in \{j, j+1\}$. O processo de computação de \hat{b} vai ser dado por qualquer quadratura numérica para aproximação de integrais, o que será explorado nas seguintes secções.

2.3 Análise do erro em 1D

A função erro é definida pela expressão $e = |u - u_h|$, em que u é a solução exata e u_h é a solução aproximada obtida por M.E.F. Consideram-se as normas $L^2(0, 1)$, norma de energia $\|\cdot\|_E$ e norma $H^1(0, 1)$ definidas como $\|e\|_{L^2(0,1)} := (\int_0^1 e^2)^{1/2}$, $\|e\|_{H^1(0,1)} := (\int_0^1 (\frac{de}{dx})^2 + e^2)^{1/2}$, $\|e\|_E := (\int_0^1 (\frac{de}{dx})^2)^{1/2}$. Olha-se, agora, para a forma variacional no caso contínuo e discreto:

$$\text{Encontrar } u \in V \text{ tal que } a(u, v) = L(v) \quad \forall v \in V, \text{ no caso contínuo (1)}$$

$$\text{Encontrar } u_h \in V_h \text{ tal que } a(u_h, v_h) = L(v_h) \quad \forall v_h \in V_h, \text{ no caso discreto (2)}$$

Desta forma, pode-se fazer uma tradução do problema de Poisson contínuo e discreto em formulação variacional, considerando-se, ainda, os resultados do [apêndice matemático](#), isto é:

$$V, \quad a(u, v) = \int_0^1 \frac{du}{dx} \frac{dv}{dx} dx \quad e \quad L(v) = \int_0^1 fv dx, \text{ no caso contínuo (3)}$$

$$V_h, \quad a(u_h, v_h) = \int_0^1 \frac{du_h}{dx} \frac{dv_h}{dx} dx \quad e \quad L(v_h) = \int_0^1 fv_h dx, \text{ no caso discreto (4)}$$

Em que V e V_h são os espaços definidos acima.

Antes de enunciar o Lema de Céa, que é crucial para obter as estimativas de erro, é necessário enunciar um resultado conhecido por **Ortogonalidade de Galerkin**.

Teorema: Seja $u \in V$ a solução do problema (1) (solução fraca) e $u_h \in V_h$ a solução do problema (2) (aproximação de elementos finitos). Então, tem-se: $a((u - u_h), v_h) = 0 \quad \forall v_h \in V_h$.

Dem. Seja $v_h \in V_h$ arbitrário. Tem-se:

$$a((u - u_h), v_h) = a(u, v_h) - a(u_h, v_h) \quad (1) = L(v_h) - L(v_h) = 0 \quad (2)$$

(1) tem-se por bilinearidade e (2) tem-se porque $V_h \subset V$ e então $a(u, v_h) = L(v_h) \quad \forall v_h \in V_h$.

Lema (Lema de Céa): Seja $u \in V$ solução do problema contínuo (1) e seja $u_h \in V_h$ solução para o problema discreto (2). Então, tem-se a seguinte estimativa de quasi-optimalidade: $\|u - u_h\|_E \leq \inf_{v_h \in V_h} \|u - v_h\|_E$.

Dem:

Segue-se, para todo o $v_h \in V_h$, que:

$$\|u - u_h\|_E^2 = a(u - u_h, u - v_h + v_h - u_h) = a(u - u_h, u - v_h) + a(u - u_h, v_h - u_h) = a(u - u_h, u - v_h) \quad (1)$$

$$a(u - u_h, u - v_h) \leq \|u - u_h\|_E \|u - v_h\|_E \quad (2)$$

(1) tem-se por ortogonalidade de Galerkin e (2) vem da desigualdade de Cauchy-Schwartz.

Dividindo ambos os lados por $\|u - u_h\|_E$ obtém-se o resultado pretendido.

Repare-se que o lema anterior indica que o erro é comparável à melhor aproximação de u por um elemento de V_h . Usando este lema em conjunto com os resultados sobre interpolação da [secção 7.5](#) é possível produzir o seguinte teorema: Seja $\{x_j\}_{i=1}^N$ uma partição de I e $V_h \subset V$ o

espaço de elementos finitos lineares associados à partição $\{x_j\}_{j=1}^{\hat{n}}$. Então dada a solução fraca u tal que $u'' \in L^2(0, 1)$ e sendo u_h a solução do problema discreto então $\|u - u_h\|_E \leq Ch\|u''\|_E$. Isto é, temos ordem de convergência linear para a norma de energia. Para a norma $L^2(0, 1)$ vem $\|u - u_h\|_{L^2} \leq Ch^2\|u''\|_{L^2}$.

2.4 Problema de Poisson em 2D

Como foi feito anteriormente, é constatado o problema de Poisson em **2D** e as suas respectivas formulações fraca e forte. O problema de Poisson é definido fortemente como $\Delta u = f$ em Ω com a condição de fronteira $u = 0$ em $\partial\Omega$ em que $\Omega \subset \mathbb{R}^2$ e u é tal que $u \in C^2(\Omega) \cap C(\bar{\Omega})$. Assume-se que a fronteira $\partial\Omega$ é um polígono e que Ω é um domínio aberto, limitado e simplesmente conexo. Define-se o espaço de Sobolev $H_0^1(\Omega)$ de funções em $H^1(\Omega)$ que se anulam na fronteira no sentido do traço, i.e: $H_0^1(\Omega) := \{u \in H^1(\Omega) \mid Tu = 0\}$ em que T é chamado de operador de traço e é definido como $Tv = v|_{\partial\Omega}$. É necessária a introdução deste operador dado que uma função que pertença a $H^1(\Omega)$ não garante que ela seja contínua. A formula fraca do problema de Poisson é encontrar $u \in H_0^1(\Omega)$ tal que: $\int_{\Omega} \nabla u \cdot \nabla v = \int_{\Omega} fv$ para todo o $v \in H_0^1(\Omega)$.

2.5 Método dos elementos finitos em 2D

Tal como no caso **1D**, a assemblagem é feita através de um sistema de equações lineares $A\beta = b$, em que as funções base $\{\hat{\varphi}_j\}_{j=1}^{\hat{n}}$ são obtidas considerando-se uma partição triangular T_h de Ω (também se poderia considerar uma partição em quadriláteros). Define-se como triangulação uma partição de um domínio Ω em um conjunto de triângulos $T = \{T_1, \dots, T_M\}$ tal que:

- $\bar{\Omega} = \cup_{i=1}^M T_i$
- A intersecção de dois triângulos ou é o vazio ou um vértice ou uma aresta

O espaço de funções seccionalmente lineares e contínuas (mais geralmente polinomias e descontínuas consoante o problema) sobre T_h dado por $\hat{V}_h := \{v \in C(\bar{\Omega}) \mid v|_T \in P^1(T) \text{ para todo } T \in T_h\}$, incluindo a condição de fronteira dada por $V_h := \{v \in \hat{V}_h \mid v = 0 \text{ em } \partial\Omega\}$. Assim, definindo $\{n_i\}_{i=1}^{\hat{n}}$ como sendo os vértices da triangulação vamos ter:

$$\hat{\varphi}_j(n_i) = \begin{cases} 1, & \text{se } j = i \\ 0, & \text{caso contrário} \end{cases}$$

Colocando o problema na formulação variacional, então $\hat{A}_{ij} = a(\hat{\varphi}_j, \hat{\varphi}_i)$ e $\hat{b}_i = L(\hat{\varphi}_i)$. Desta forma, vamos ter $\hat{A}_{ij} := \int_{\Omega} \nabla \hat{\varphi}_j \cdot \nabla \hat{\varphi}_i$ e $\hat{b}_i = \int_{\Omega} f \hat{\varphi}_i$. Para facilitar a avaliação do integral de um triângulo $T \sim [n_1 \ n_2 \ n_3]$, usa-se uma transformação afim $F|_T$ de um triângulo de referência $\hat{T} \sim [\hat{n}_1 \ \hat{n}_2 \ \hat{n}_3] = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ para este mesmo. Esta aplicação é dada por $F_T(x) = A_T x + b_T$, em que a matriz $A_T = [a_1 \ a_2]$ e o vetor b_T são tais que $b_T = n_1$, $a_1 = n_2 - n_1$ $a_2 = n_3 - n_1$ e, portanto, $\int_T f(x) dA$ pode ser avaliado como $\int_{\hat{T}} f(F_T(\hat{x})) |\det A_T| d\hat{A}$.

Resta saber como indexar as funções base. Efetivamente, dado um triângulo $T \sim [n_{i_1} \ n_{i_2} \ n_{i_3}]$ define-se a função $\sigma : T_h \times 1, 2, 3 \rightarrow 1, \dots, \hat{n}$ que devolve o índice global do vértice dado um triângulo e um índice do vértice nesse triângulo. Assim, vamos ter que a contribuição de um triângulo para as entradas \hat{A} e \hat{b} são respetivamente dadas por: $\hat{A}_{\sigma(T,l)\sigma(T,k)} = \hat{A}_{\sigma(T,l)\sigma(T,k)} + \int_T \nabla \hat{\varphi}_{\sigma(T,l)} \cdot \nabla \hat{\varphi}_{\sigma(T,k)}$, $\hat{b}_{\sigma(T,l)} = \hat{b}_{\sigma(T,l)} + \int_T f \hat{\varphi}_{\sigma(T,l)}$. Estes integrais vão sofrer a dita transformação afim definida no parágrafo anterior. Definindo $\psi_l(\hat{x}) := \hat{\varphi}_{\sigma(T,l)}(F_T(\hat{x}))$ então é necessário que:

$$\hat{\psi}_l(\hat{n}_k) = \begin{cases} 1, & \text{se } l = k \\ 0, & \text{caso contrário} \end{cases}$$

Resolvendo o sistema acima:

$$\psi_1(\hat{x}) = 1 - \hat{x}_1 - \hat{x}_2, \quad \psi_2(\hat{x}) = \hat{x}_1, \quad \psi_3(\hat{x}) = \hat{x}_2$$

Assim sendo, e usando uma regra de quadratura com pesos w_i e pontos t_i , o integral de \hat{b} passa a ter a forma $\sum_{i=1}^N f(F_T(t_i))\psi_l(t_i)|\det A_T|w_i$ e o integral de \hat{A} , tendo em conta que os gradientes ψ_l e as matrizes A_T são constantes e a área do triângulo de referência \hat{T} é $\frac{1}{2}$, pode ser avaliado como: $\frac{1}{2}\hat{\nabla}\psi_l^T A_T^{-1} A_T^{-T} \hat{\nabla}\psi_k |\det A_T|$.

Seja $h = \max_{T \in T_h} h_T$, em que h_T se trata do diâmetro da menor circunferência em qual o triângulo T se encontra inscrito. Então, para a norma $L^2(\Omega)$ e a norma de energia $E(\Omega)$, tem-se as seguintes estimativas de erro: $\|u - u_h\|_E \leq Ch|u|_{H^2}$ e $\|u - u_h\|_{L^2} \leq Ch^2|u|_{H^2}$, onde u é a solução exata e u_h é a solução aproximada obtida pelo método dos elementos finitos lineares e em que as constantes $C > 0$ não dependem de h , ver [1].

3 Implementação do M.E.F em 1D

3.1 Assemblagem do sistema em 1D

Esta secção é usada como um estudo prático do método dos elementos finitos em **1D**, ou seja, é introduzido um código *MATLAB* para a assemblagem do sistema linear, seguindo os passos introduzidos na secção dos conceitos teóricos, que servirá como ponto de partida para futuras manipulações do problema original. Como foi mencionado anteriormente é necessário considerar-se uma quadratura numérica para aproximação de integrais. Desta forma, escolhe-se uma regra de quadratura de Gauss de dois pontos, com mudança de intervalo de $[-1, 1]$ para $[0, 1]$, tal que $\int_0^1 f(x) dx \approx \sum_{i=1}^2 w_i f\left(\frac{1}{2}t_i + \frac{1}{2}\right)$, em que t é o vetor de pontos de integração e w é o vetor de pesos (a função auxiliar *gaussint* devolve os pontos de samplear já transformados). É de realçar que esta quadratura numérica é exata para polinómios de graus inferior ou iguais a 3, dado que uma regra de quadratura de Gauss de n -pontos é exata para todos os polinómios de grau menor ou igual a $2n - 1$.

Temos então o seguinte código base para o método dos elementos finitos em 1D:

```

1 % Codigo adaptado do Professor Antti Hannukainen da Universidade de
2 % Aalto [1]
3 % Codigo adaptado do Professor Antti Hannukainen da Universidade de
4 % Aalto [1]
5
6 clear all
7 clf
8 % Criacao da particao no intervalo [0 ,1]
9 N = 10;
10 x = linspace(0 ,1 ,N);
11
12 % Nesta expressao podemos fazer a manipulacao de f%
13 f = @(x)(1+0*x);
14 uf= @(x)(1/2*(1-x)*x);
15
16 % Foi usada a funcao sparse porque Ahat tem muitos valores nulos
17 Ahat = sparse(N,N);

```

```

16 bhat = zeros(N,1);
17
18 % Assemblagem
19
20 for k=1:(N-1)
21
22     % inicio e fim do intervalo
23     x1 = x(k);
24     x2 = x(k+1);
25
26
27     % termos
28     phi{:,1} = @(x) (x-x2)./(x1-x2);
29     phi{:,2} = @(x) (x-x1)./(x2-x1);
30
31
32     % valores da derivada de phi. Obtido logo por derivacao de phi em x
33     dphi(1) = 1/(x1-x2);
34     dphi(2) = 1/(x2-x1);
35
36     enum =[k k+1];
37     [X,W]=gaussint(2,x(k),x(k+1),1);
38     for i=1:2
39         for l=1:2
40             bhat(enum(i))=bhat(enum(i)) + W(l)*f(X(l))*phi{i}(X(l));
41         end
42         for j=1:2
43             int=0;
44             for l=1:2
45                 int=int + W(l)*dphi(i)*dphi(j);
46             end
47             Ahat(enum(i),enum(j)) = Ahat(enum(i),enum(j)) + int;
48         end
49     end
50 end
51
52 % Sao removidos o primeiro e o ultimo ponto do intervalo
53 A=Ahat(2:(N-1),2:(N-1));
54 b=bhat(2:(N-1),1);
55
56 % Construcao da solucao u
57 u(1,1)=0;
58 u(N,1)=0;
59 u(2:(N-1),1)=A\b;
60
61 % Figura da solucao
62 figure(1)
63 plot(x,u,'r*-')
64 figure(2)
65 fplot(uf,[0,1]);
66 ylim([0 0.14])

```

Para a função constante escolhida $f(x) = 1$ com dez partições, tem-se o seguinte gráfico solução da aproximada u_h e da solução exata u .

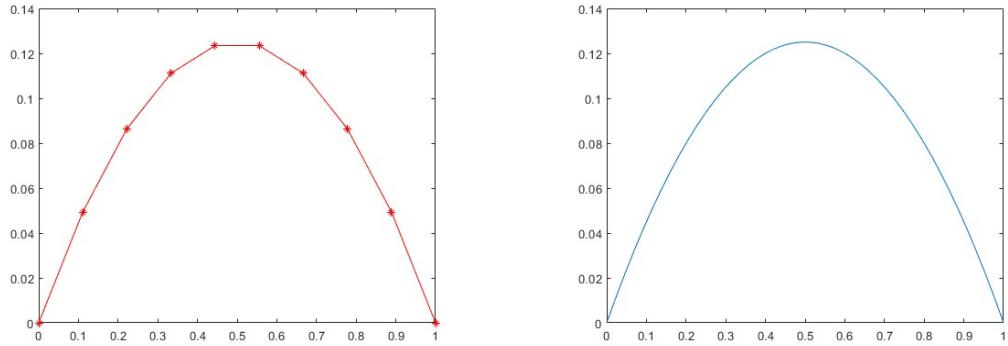


Figura 1: Solução aproximada u_h obtida por M.E.F quando $f(x) = 1$ e $n=10$ (gráfico à esquerda) em comparação com a solução exata $u(x) = \frac{1}{2}(1-x)x$ (gráfico à direita).

Como foi referido anteriormente, este código serve de base para o método dos elementos finitos em **1D**, no sentido em que é possível reproduzir soluções para outros problemas manipulando algumas linhas de código deste *script*. Por exemplo: alterar o número de elementos variando o parâmetro N ; definir outra função fonte alterando o conteúdo da linha $f = @(x) ()$; modificar as condições de fronteira, obtendo outro problema, mas cuja resolução numérica segue os passos da forma fraca do problema de Poisson ou; escolher outro espaço de elementos finitos, ou seja, modificar as funções teste ϕ_i e $d\phi_i$. Neste sentido, e para efeitos ilustrativos, escolhe-se o espaço de elementos finitos de segundo grau V_h^2 , que sem considerar condições de fronteira é definido da seguinte forma: $\hat{V}_h^2 := \{u \in C[0,1] \mid u|_{I_i} \in P^2(I_i) \text{ para } i = 1, \dots, (N-1)\}$. Escolhemos então uma base em $P^2(I_j)$ que é obtida adicionando à base construída anteriormente para o espaço V_h de funções lineares uma função bolha dada por:

$$\hat{\phi}_j(x) = \begin{cases} (x - x_j)(x_{j+1} - x), & \text{se } x \in I_j \\ 0, & \text{caso contrário} \end{cases}$$

Assim, e usando a quadratura numérica anterior, o *script* passa a ter a seguinte forma:

```

1 clear all
2 clf
3
4 % Criacao da particao [0 ,1]
5
6 N = 10;
7 dim=2*N-1;
8 p = linspace (0 ,1 ,N);
9
10 % Nesta expressao podemos fazer a manipulacao de f%
11 f = @(x) ( sin(x) );

```

```

12
13 Ahat = sparse(dim,dim);
14 bhat = zeros(dim,1);
15
16 for k=1:(N-1)
17   x1 = p(k);
18   x2 = p(k+1);
19   %function handle porque vamos estar sempre a avaliar
20   phi{:,1} = @(x) (x-x2)./(x1-x2);
21   phi{:,2} = @(x) (x-x1)./(x2-x1);
22   phi{:,3} = @(x) (x-x1).* (x2-x); %funcao bolha%
23
24   dphi{:,1} = @(x) 1/(x1-x2);
25   dphi{:,2} = @(x) 1/(x2-x1);
26   dphi{:,3} = @(x) -2*x+x2+x1;
27
28   enum = [k k+1 N+k];
29   %Regra de quadratura de Gauss
30   [X,W]=gaussint(2,p(k),p(k+1),1);
31
32   for i=1:3
33     for l=1:2
34       bhat(enum(i))=bhat(enum(i))+W(l)*f(X(l))*phi{i}(X(l));
35     end
36     for j=1:3
37       for l=1:2
38         Ahat(enum(i),enum(j))= Ahat(enum(i),enum(j))+W(l)*dphi{i}(X(l));
39           *dphi{j}(X(l));
40       end
41     end
42   end
43
44 idof = setdiff(1:(2*N-1),[1 N]);
45 A = Ahat(idof,idof);
46 b = bhat(idof,1);
47 u(idof) = A\b;
48
49
50 figure(1)
51 hold on;
52
53 % Plot da solucao com bolha
54 for k=1:N-1
55   x1 = p(k);
56   x2 = p(k+1);
57   phi{:,1} = @(x) (x-x2)./(x1-x2);
58   phi{:,2} = @(x) (x-x1)./(x2-x1);
59   phi{:,3} = @(x) (x-x1).* (x2-x);
60   t = linspace(p(k),p(k+1));
61   plot(t,phi{1}(t)*u(k)+phi{2}(t)*u(k+1)+phi{3}(t)*u(k+N));

```

```

62 end
63 hold off;
64
65 figure(2)
66 hold on;
67
68 % Plot da solucao sem bolha
69 for k=1:N-1
70 x1 = p(k);
71 x2 = p(k+1);
72 phi{:,1} = @(x) (x-x2)./(x1-x2);
73 phi{:,2} = @(x) (x-x1)./(x2-x1);
74 t = linspace(p(k),p(k+1));
75 plot(t,phi{1}(t)*u(k)+phi{2}(t)*u(k+1));
76 end
77 ylim([0 0.07]);
78 hold off;

```

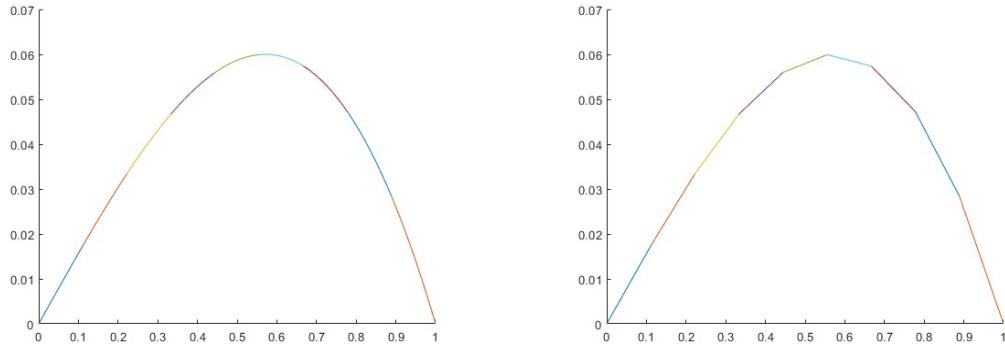


Figura 2: Soluções aproximadas obtidas em V_h^2 (gráfico à esquerda) e em V_h (gráfico à direita). As cores representam os diferentes elementos.

Como é possível observar, a utilização de elementos quadráticos ”suaviza” a curva, o que remete para o facto de a inclusão da função bolha levar a menores erros. Esta questão será aprofundada na secção seguinte.

3.2 Análise do erro empírico em 1D

É agora feita uma análise empírica do erro, isto é, estudamos o erro de solução aproximada comparando-a com a solução explícita conhecida. Definindo uma partição uniforme do intervalo, usa-se o seguinte código *MATLAB* para observar a influência do tamanho do passo h no erro. É de notar que foi criada uma função auxiliar `solver1D`, que corresponde ao código anterior para a assemblagem linear mas adaptado ao formato `function`.

```

1 % Codigo adaptado do Professor Antti Hannukainen da Universidade de
2 Aalto [1]

```

```

3 | clear all
4 | clf
5 |
6 | % Lista das particoes
7 | N_list = [5 10 20 40];
8 |
9 | f=@(x)(1+0*x);
10 |
11 | t=linspace(0,1,100);
12 |
13 | figure(1);
14 | % O comando interp1 faz a interpolacao do vetor u da solucao aproximada
15 | for i=1:length(N_list)
16 |     x = linspace(0,1,N_list(i));
17 |
18 |     u = solver1D(f,x);
19 |
20 |     uh_t=interp1(x,u,t);
21 |
22 |     u_t=(1/2)*t.* (1-t);
23 |
24 |     plot(t,u_t-uh_t); hold on;
25 |
26 | legend('N=5','N=10','N=20','N=40');

```

Executando este *script* são obtidas as seguintes comparações entre as soluções aproximadas e a solução exata da função constante $f(x) = 1$ para um número de elementos progressivamente maior:

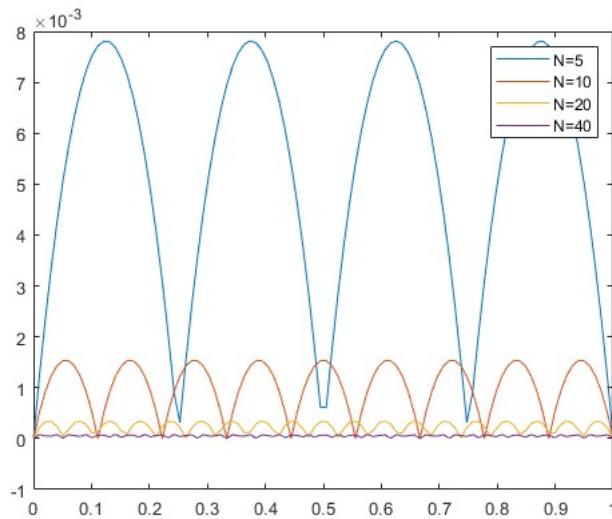


Figura 3: Função erro e versus o número de partições N .

Observando a Figura 3 conclui-se que o aumento do número de elementos leva a uma diminuição substancial do erro.

É possível também quantificar os erros pela sua norma associada. Para tal, é necessário reparar que para a restrição $e|I_j = u|I_j - u_h|I_j$ se tem $u_h|I_j = \hat{\beta}_j \hat{\varphi}_j|I_j + \hat{\beta}_{j+1} \hat{\varphi}_{j+1}|I_j$ em que $\hat{\beta}_j$ e $\hat{\beta}_{j+1}$ são, respectivamente, a j -ésima e $\{j+1\}$ -ésima coordenadas do vetor:

$$\hat{\beta} = \begin{bmatrix} 0 \\ \beta \\ 0 \end{bmatrix}$$

Considerando as normas $L^2(0, 1)$, $H^1(0, 1)$ e a norma de energia $E(0, 1)$, então, usando, novamente, a regra de quadratura de Gauss de dois pontos tal que $\|e\| \approx \sum_{k=1}^2 e(t_k) \cdot w_k$, em que t é o vetor de pontos de integração e w o vetor de pesos, temos o seguinte código *MATLAB*:

```

1 function [ L2error , Error , Herror ] = erro_fem1D_linear (x , u , ufun , dufun )
2 % Devolve as respectivas normas
3 % Valores para as normas
4 val_L2=0;
5 val_E=0;
6 val_H=0;
7 for i=1:(length(x)-1)
8     % Regra de Quadratura de Gauss
9     [t ,w] = gaussint (2 ,x(i) ,x(i+1));
10    % Pontos avaliados
11    uh_tk = zeros (1 ,length(t));
12    duh_tk = zeros (1 ,length(t));
13    u_tk = zeros (1 ,length(t));
14    du_tk = zeros (1 ,length(t));
15    for k=1:length(t)
16        uh_tk(k) = u(i)*( x(i+1)-t(k))/(x(i+1)-x(i));
17        uh_tk(k) = uh_tk(k) + u(i+1)*( x(i)-t(k))/(x(i)-x(i+1))
18        ;
19        duh_tk(k) = -u(i)/(x(i+1)-x(i));
20        duh_tk(k) = duh_tk(k) - u(i+1)/(x(i)-x(i+1));
21        u_tk(k) = ufun(t(k));
22        du_tk(k) = dufun(t(k));
23    end
24    val_L2 = val_L2 + (u_tk - uh_tk).^2*w(:);
25    val_E = val_E + (du_tk - duh_tk).^2*w(:);
26    val_H = val_H + (u_tk - uh_tk).^2*w(:) + (du_tk - duh_tk).^2*w
27    (:);
28 end
29 L2error = sqrt(val_L2);
30 Error = sqrt(val_E);
31 Herror = sqrt(val_H);
32 end

```

Escolhendo agora uma função definida por ramos f tal que:

$$f(x) = \begin{cases} -1, & \text{se } x \in [0, \frac{1}{2}] \\ 1, & \text{se } x \in (\frac{1}{2}, 1] \end{cases}$$

Neste caso a solução explícita é dada por:

$$u(x) = \begin{cases} \frac{1}{2}x^2 - \frac{1}{4}x, & \text{se } x \in [0, \frac{1}{2}] \\ -\frac{1}{2}(x-1)^2 - \frac{1}{4}(x-1), & \text{se } x \in (\frac{1}{2}, 1] \end{cases}$$

e a sua primeira derivada é:

$$u'(x) = \begin{cases} x - \frac{1}{4}, & \text{se } x \in [0, \frac{1}{2}] \\ -x + \frac{3}{4}, & \text{se } x \in (\frac{1}{2}, 1] \end{cases}$$

É importante realçar que a primeira derivada de u é contínua mas a segunda não é, pelo que u não satisfaz a formulação forte do problema de Poisson mas satisfaz a formulação variacional (fraca), isto é, $u \in H_0^1(0, 1)$.

Calcula-se, então, o erro da solução aproximada por elementos finitos em função do tamanho de h através do seguinte *script*:

```

1 clear all;
2 N_list = [10 20 40 80 160 320];
3 f=@(x) (-1).* (x<=1/2)+(1).* (x>1/2);
4 e=@(x) ((1/2)*x^2-(1/4)*x).* (x<=1/2)+(-(1/2)*(x-1)^2-(1/4)*(x-1)).*(x
    >1/2);
5 de=@(x) (x-(1/4)).*(x<=1/2)+(-(x)+1-(1/4)).*(x>1/2);
6 for i = 1:length(N_list)
7     x = linspace(0,1, N_list(i));
8     u = solver1D(f,x);
9     [L2error(i), Error(i), Herror(i)] = erro_fem1D_linear(x,u,e,de);
10 end
11
12 figure(1)
13 loglog(1./(N_list-1),L2error,'k:o');
14 hold on;
15 loglog(1./(N_list-1),1./(N_list-1).^2,'—or');
16 legend({'Erro na norma L2','Taxa de h^2'},'Location','northwest')
17 xlabel('Tamanho da Particao')
18 hold off;
19
20 figure(2)
21 loglog(1./(N_list-1),Error,'k:o');
22 hold on;
23 loglog(1./(N_list-1),1./(N_list-1),'—or');
24 legend({'Erro na norma E','Taxa de h'},'Location','northwest')
25 xlabel('Tamanho da Particao')
26 hold off;
27
28 figure(3)
29 loglog(1./(N_list-1),Herror,'k:o');
30 hold on;
31 loglog(1./(N_list-1),1./(N_list-1),'—or');
32 legend({'Erro na norma H^1','Taxa de h'},'Location','northwest')
33 xlabel('Tamanho da Particao')
```

34 | **hold off;**

Os resultados em diferentes normas são apresentados nas Figuras 4, 5 e 6.

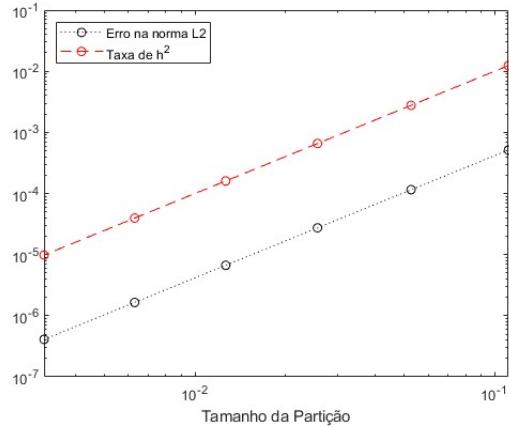


Figura 4: Gráfico **loglog** da norma $L^2(0,1)$ versus h .

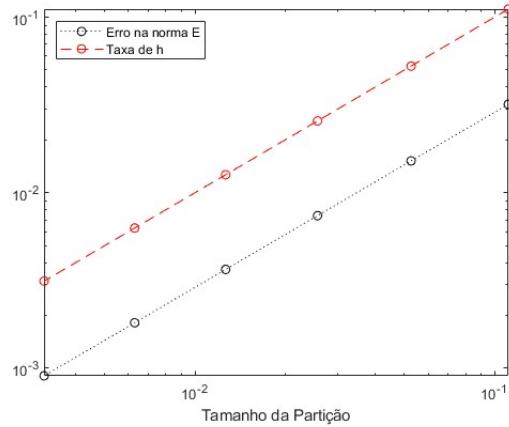
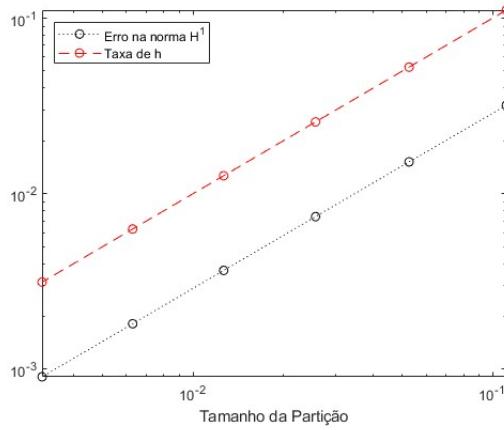
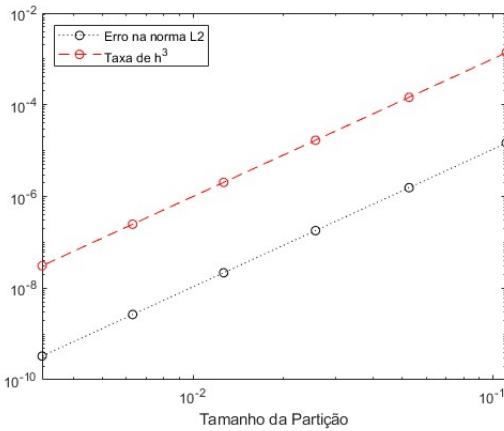


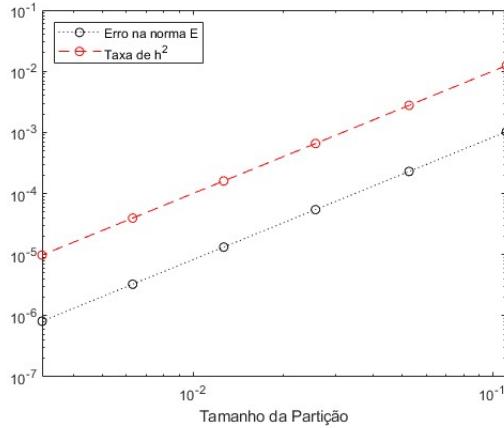
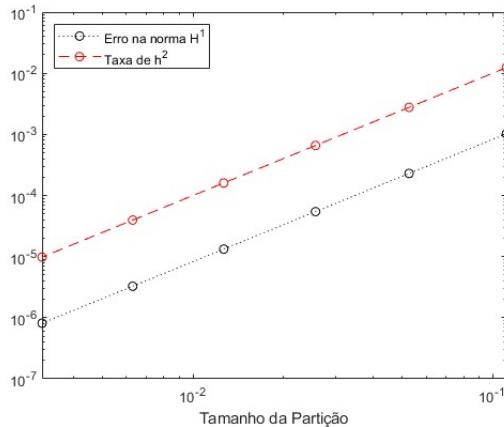
Figura 5: Gráfico **loglog** da norma de energia $E(0,1)$ versus h .

Figura 6: Gráfico loglog da norma $H^1(0, 1)$ versus h .

Desta forma, confirmam-se os resultados teóricos da secção 2.3. Efetivamente, relembrando que a transformação **loglog** satisfaz $\log(y) = k\log(x) + b$ tal que $y = 10^{k\log(x)+b} = Cx^k$, é possível estabelecer uma relação direta entre o declive da reta no gráfico **loglog** e a convergência $O(h^k)$ das normas. Assim sendo, olhando para a Figura 4, conclui-se que a norma $L^2(0, 1)$ se comporta como Ch^2 para uma dada constante C . Observando a Figura 5 e a Figura 6, é possível deduzir que as normas $E(0, 1)$ e $H^1(0, 1)$ se comportam como Ch .

No caso dos elementos finitos da segunda ordem, modificando tanto o *script* anterior como a função *solver1D* para acomodar a inclusão da função bolha, escolhendo a função $f(x) = \sin(\pi x)$ com solução exata $u(x) = \frac{\sin(\pi x)}{(\pi^2)}$ e derivada $u'(x) = \frac{\cos(\pi x)}{\pi}$ obtém-se as imagens subsequentes:

Figura 7: Gráfico loglog da norma $L^2(0, 1)$ em V_h^2 versus h .

Figura 8: Gráfico loglog da norma de energia $E(0,1)$ em V_h^2 versus h .Figura 9: Gráfico loglog da norma $H^1(0,1)$ em V_h^2 versus h .

Usando o raciocínio do parágrafo anterior, conclui-se que para este escolha de espaço de funções base a norma $L^2(0,1)$ se comporta como Ch^3 e as normas E -norm e $H^1(0,1)$ se comportam como Ch^2 , aumentando, portanto, um grau na ordem de convergência. Assim sendo, a estipulação prévia de que o uso de um espaço de segundo grau leva a menores erros é confirmada.

4 Implementação do M.E.F em 2D

4.1 Assemblagem do sistema linear em 2D

O primeiro passo para obter a solução aproximada, com condição de Dirichlet nula, num domínio $\Omega \subset \mathbb{R}^2$ é criar uma malha triangular (ou quadrilateral) definida pela matriz $p \in \mathbb{R}^{2 \times N_p}$, que contém os p nós (ou nodos) da triangulação, e pela matriz $t \in \mathbb{N}^{3 \times N_t}$, que contém os índices dos vértices dos N_t triângulos. Estas matrizes podem ser introduzidas de forma manual ou, a fim de obter triangulações mais complexas, usando funções auxiliares disponibilizadas pela biblioteca `pde toolbox`. Por curiosidade foram escolhidas três malhas: uma quadrada; outra rectangular; e uma em forma L. Seguidamente foi aplicada a função `solver2D` que calcula a solução aproximada para uma dada função fonte f , neste exemplo $f(x, y) = \sin(\pi x)\sin(\pi y)$, que também recebe como argumentos a malha e o parâmetro de refinamento N . As funções responsáveis pela geração da

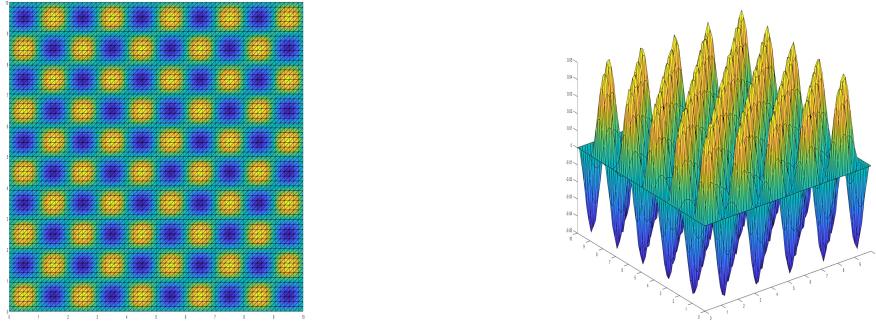
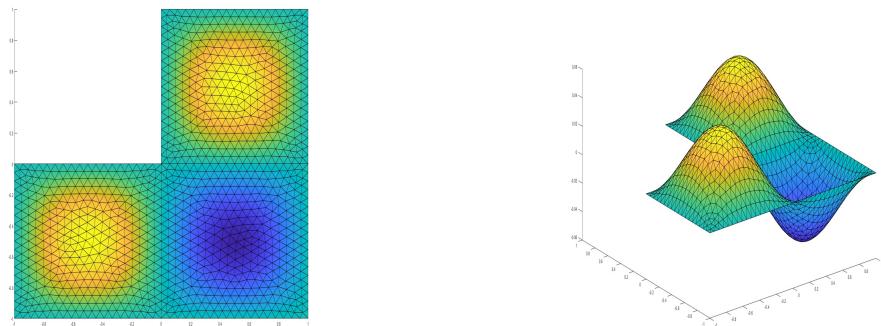
malha, `inittri(p,t)`, pelo refinamento, `refine_tri(mesh)`, e pela regra de quadratura triangular, `inttri(p)`, foram obtidas das notas do Professor Antti Hannukainen da Universidade de Aalto [1].

```

1 clear all
2 % funcao source
3 % Os valores de N indicam o numero de vezes a refinar a malha
4 f = @(x,y)( sin(pi.*x).*sin(pi.*y));
5
6 % Criar triangulacao em mesh retangular
7 N=6;
8 p = [ 0 10 10 0 ; 0 0 10 10];
9 t = [ 1 2 3; 1 4 3] ';
10 mesh1=make_mesh(N,p,t);
11
12 solver2D(f,N,mesh1);
13
14 % Criar triangulacao em mesh exemplo
15 N=5;
16 p =[ 0 1 2 0 1 2 ; 0 0 0 1 1 1];
17 t = [ 1 2 2 3 ; 2 4 3 5 ; 4 5 5 6 ];
18 mesh2=make_mesh(N,p,t);
19
20 solver2D(f,N,mesh2);
21
22 % Criar triangulacao em mesh L
23 N=1;
24 model = createpde(1);
25 geometryFromEdges(model,@lshapeg);
26 lshape=generateMesh(model);
27 mesh3=make_mesh(N,lshape.Nodes,lshape.Elements);
28
29 solver2D(f,N,mesh3);
30
31 % Criar triangulacao em mesh Test
32 N=1;
33 model = createpde(1);
34 g = geometryFromEdges(model,@cardg);
35 testshape=generateMesh(model);
36 mesh4=make_mesh(N,testshape.Nodes,testshape.Elements);
37
38 solver2D(f,N,mesh4);

```

Cada uma das seguintes Figuras (10-13) contém: uma imagem da solução aproximada na própria triangulação, em que o gradiente de cores ilustra o afastamento do plano $z = 0$, e; uma visualização da solução em três dimensões.

Figura 10: Malha quadrada $[0, 10] \times [0, 10]$ com seis refinamentos e solução aproximada.Figura 11: Malha quadrada rectangular $[0, 2] \times [0, 1]$ com cinco refinamentos e solução aproximada.Figura 12: Malha em forma L no quadrado $[0, 1] \times [0, 1]$ e solução aproximada.

O código da função *solver2D* segue os passos indicados na secção 2.5 para assemblagem da solução aproximada em **2D**.

```

1 | function [u] = solver2D(f,N,mesh)
2 | T=mesh.t;

```

```

3 P=mesh.p;
4 Nt=size(T,2);
5
6 bhat=sparse(size(P,2),1);
7 Ahat=sparse(size(P,2),size(P,2));
8
9
10 for i=1:Nt
11
12 % X - Pontos a serem avaliados
13 % W - Pesos
14 [X,W] = inttri(N);
15
16 % Matriz At, vetor bt
17
18 At(1,1)=(P(1,T(2,i))-P(1,T(1,i)));
19 At(1,2)=(P(1,T(3,i))-P(1,T(1,i)));
20 At(2,1)=(P(2,T(2,i))-P(2,T(1,i)));
21 At(2,2)=(P(2,T(3,i))-P(2,T(1,i)));
22
23 bt(1,1)=P(1,T(1,i));
24 bt(2,1)=P(2,T(1,i));
25
26 % Calculo det
27 detAt=det(At);
28
29 % Derivadas
30 Nip=size(X,2);
31 dL{1} = [ -ones(1,1); -ones(1,1) ];
32 dL{2} = [ ones(1,1); zeros(1,1) ];
33 dL{3} = [ zeros(1,1); ones(1,1) ];
34
35 for l=1:3
36 for j=1:Nip
37 % Avaliacao das bases
38 phi(:,1) = 1-X(1,j)-X(2,j);
39 phi(:,2) = X(1,j);
40 phi(:,3) = X(2,j);
41
42 % Avaliacao de Ft
43 Xhat(1,1)=X(1,j);
44 Xhat(2,1)=X(2,j);
45
46 Ft=(At*Xhat)+bt;
47
48 bhat(T(1,i))=bhat(T(1,i))+f(Ft(1),Ft(2))*phi(1)*abs(
49 detAt)*W(j);
50 end
51 for k=1:3
52 Ahat(T(1,i),T(k,i))=Ahat(T(1,i),T(k,i))+0.5*transpose(
53 dL{1})*inv(At)*transpose(inv(At))*dL{k}*abs(detAt);

```

```

52         end
53     end
54 end

55

56
57 % Pontos de interior e fronteira
58 be = find( mesh.e2t(2,:) == 0 );
59 bind = mesh.edges(:,be);
60 bind = unique(bind(:));
61 iind = setdiff(1:size(mesh.p,2),bind);

62
63 u=zeros( size(mesh.p,2),1);
64 u(iind) = Ahat(iind,iind)\bhat(iind);
65 X = mesh.p(1,:);
66 Y = mesh.p(2,:);
67 t = mesh.t;

68
69 figure
70 plot_2Dtri_mesh(mesh)

71
72 figure
73 patch(X(t),Y(t),u(t),u(t));
74 view(3);
75 end

```

Para efeitos de visualização da Figura 14 mostra-se a diferença entre a solução obtida por M.E.F e a solução exata em $\Omega = (0, 1)^2$ da função $f(x, y) = 2y(1 - y) + 2x(1 - x)$.

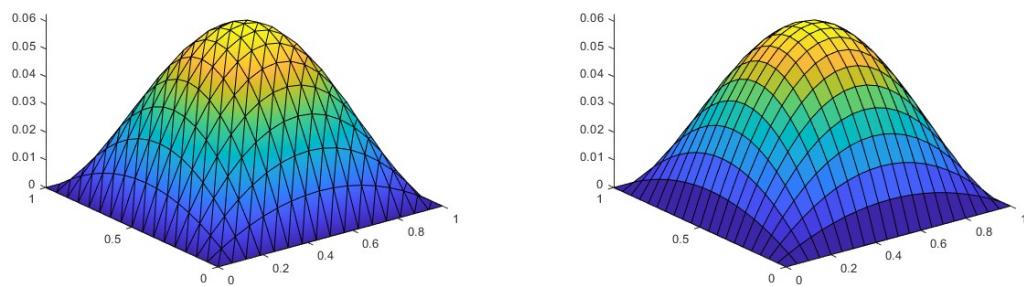


Figura 13: Soluções obtida por M.E.F e representação da solução exata $u = x(1 - x)y(1 - y)$.

4.2 Análise do erro empírico em 2D

O estudo dos erros empíricos em **2D** segue as mesmas ideias introduzidas na secção irmã 3.2. Contudo, dada à dificuldade de arranjar soluções exatas para estas equações, é necessário arranjar outra estratégia: em vez de se considerar a solução exata considera-se uma solução aproximada já com um valor de h muito reduzido. Desta forma, considera-se a seguinte função `erro_FEM2D`:

```

1 | function [L2error, Error] = erro_fem2D(mesh, f, uf, u)

```

```

2 % Devolve as respetivas normas
3 % Valores para as normas
4 val_L2=0;
5 val_E=0;
6 % Inicializacao das matrizes
7 P=mesh.p;
8 T=mesh.t;
9 Nt=size(T,2);

10
11 % Bases referencia
12 phi{:,1} = @(x,y) 1-x-y;
13 phi{:,2} = @(x,y) x;
14 phi{:,3} = @(x,y) y;

15
16
17
18 for i=1:Nt
19 % Funcao erro e derivada do erro
20 err = @(x,y) (uf(T(1,i))*phi{1}(x,y)+uf(T(2,i))*phi{2}(x,y)+uf(
21 T(3,i))*phi{3}(x,y))-(u(T(1,i))*phi{1}(x,y)+u(T(2,i))*phi
22 {2}(x,y)+u(T(3,i))*phi{3}(x,y));
23 derr = @(x,y) f(x,y)*((uf(T(1,i))*phi{1}(x,y)+uf(T(2,i))*phi
24 {2}(x,y)+uf(T(3,i))*phi{3}(x,y))-(u(T(1,i))*phi{1}(x,y)+u(T
25 (2,i))*phi{2}(x,y)+u(T(3,i))*phi{3}(x,y)));
26 % Pesos e pontos
27 [t,w] = inttri(20);

28
29 % Matriz At, vetor bt e det
30 At(1,1)=(P(1,T(2,i))-P(1,T(1,i)));
31 At(1,2)=(P(1,T(3,i))-P(1,T(1,i)));
32 At(2,1)=(P(2,T(2,i))-P(2,T(1,i)));
33 At(2,2)=(P(2,T(3,i))-P(2,T(1,i)));

34 bt(1,1)=P(1,T(1,i));
35 bt(2,1)=P(2,T(1,i));

36 detAt=det(At);

37
38 % Loop de regra de quadratura
39 for j=1:length(t)
40 Xhat(1,1)=t(1,j);
41 Xhat(2,1)=t(2,j);
42 Ft=(At*Xhat)+bt;
43 val_L2=val_L2+(err(Ft(1),Ft(2)))^2*abs(detAt)*w(j);
44 val_E=val_E+(abs(derr(Ft(1),Ft(2)))*abs(detAt)*w(j));
45 end
46 end
47 % Avaliacao das normas
48 L2error = sqrt(val_L2);
49 Error = sqrt(val_E);
50 end

```

Esclarece-se ainda que para a análise do erro na norma de energia se utilizou a igualdade:

$$\int_{\Omega} |\nabla u - \nabla u_h|^2 = \int_{\Omega} f(u - 2u_h) + \int_{\Omega} fu_h$$

Considerando-se, novamente, a função $f(x, y) = \sin(\pi x)\sin(\pi y)$ é possível calcular o erro para diferentes valores de h_T através do seguinte script:

```

1 clear all;
2 % Script que produz a norma associada ao erro para varias normas e
3 % particoes
4 N_list = [ 3 4 5 6 7];
5 f = @(x,y)( sin( pi*x)*sin( pi*y) );
6
7 p = [ 0 1 1 0 ; 0 0 1 1];
8 t = [ 1 2 3; 1 4 3] ';
9
10 % Criacao da malha proxima da solucao
11 meshy=make_mesh(8,p,t);
12 uf=solver2D(f,7,meshy);
13
14 % Loop de comparacao
15 for i = 1:length(N_list)
16     mesh=make_mesh(N_list(i),p,t);
17     h(i)=give_h(mesh);
18     u=solver2D(f,N_list(i),mesh);
19     [ L2error(i), Eerror(i) ] = erro_fem2D(mesh,f,uf,u);
20 end
21
22 figure(1)
23 loglog(h,L2error,'k:o');
24 hold on;
25 loglog(h,h.^2,'--or');
26 legend({ 'Erro na norma L2' , 'Taxa de h^2' },'Location','northwest')
27 xlabel('Tamanho de h')
28 hold off;
29
30 figure(2)
31 loglog(h,Eerror,'k:o');
32 hold on;
33 loglog(h,h,'--or');
34 legend({ 'Erro na norma E' , 'Taxa de h' },'Location','northwest')
35 xlabel('Tamanho de h')
36 hold off;
```

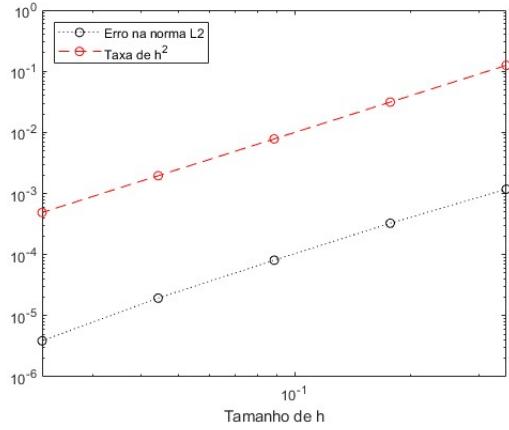


Figura 14: Gráfico loglog da norma L^2 na triangulação versus o tamanho de h .

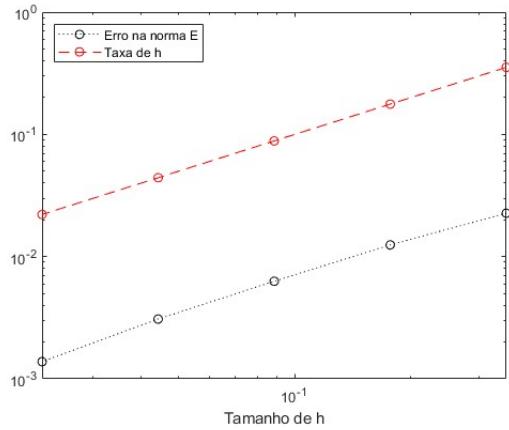


Figura 15: Gráfico loglog da norma E na triangulação versus o tamanho de h .

Desta forma, dado que, novamente, as retas apresentam o mesmo declive, é possível concluir que a norma L^2 se comporta como Ch^2 e que a norma de energia se comporta como Ch , o que vai de encontro aos resultados teóricos indicados nas secções 2.6 e 2.3. É também importante mencionar que a função auxiliar `give_h(mesh)` foi retirada das notas do Professor Antti Hannukainen da Universidade de Aalto [1].

5 Comparação com o software *FreeFEM*

Por motivos de curiosidade, compara-se as soluções obtidas do problema de Poisson em **2D** com as soluções obtidas no software grátis para análise do método dos elementos finitos chamado *FreeFEM*. Para utilização desta ferramenta, usa-se o editor de texto disponível [online](#) e o compilador disponibilizado para [download](#). O código usado neste projeto é apresentado tanto em formato .edp, para visualização através da aplicação *FreeFEM*, e em formato .txt, para modificações futuras na IDE do *website*.

Começa-se por discretizar o domínio Ω , de formato L, dentro do quadrado $[0, 1] \times [0, 1]$, e a função fonte $f(x, y) = x^2$. Para tal, em línguagem *FreeFEM*, é necessário definir as fronteiras usando o comando `border` e, subsequentemente, utilizar o comando `buildmesh` para construir a malha. A solução é calculada através do *built-in solver* do software. Para se traçarem as fronteiras

recorre-se ao uso de condições, isto é, definir intervalos para os valores x e y com uma dada orientação (é necessário que a orientação seja tal que é possível percorrer o perímetro). Efetivamente, é somente necessário escrever `border` seguido de um nome e um intervalo de valores para t . Subsequentemente, iguala-se ora x ora y a t (consoante o valor que se pretende alterar) e o restante a uma constante. Neste caso, constrói-se uma fronteira em formato de L. É de realçar que o sentido é ditado da esquerda para a direita. O seguinte código ilustra o processo.

```

1  real error = 0.1;
2
3  func f = (x^2);
4  func g = 0.;
5
6  int NAdapt = 10;
7
8 // Malha
9 border ba(t=-1, 1){x=t; y=-1; label=1;}
10 border bb(t=-1, 1){x=1; y=t; label=1;}
11 border bc(t=1, 0){x=t; y=1; label=1;}
12
13 border bd(t=1, 0){x=0; y=t; label=1;}
14 border be(t=0, -1){x=t; y=0; label=1;}
15 border bf(t=0, -1){x=-1; y=t; label=1;}
16
17
18 plot(ba(4) + bb(4) + bc(4) + bd(4) + be(4) + bf(4));
19
20 mesh Th = buildmesh(ba(6) + bb(4) + bc(4) + bd(4) + be(4) + bf(6));
21
22 plot(Th);
23
24 fespace Vh(Th, P1);
25 Vh u, v;
26
27
28 // Problema
29 problem Poisson(u, v, solver=CG, eps=1.e-6)
30   = int2d(Th)(
31     dx(u)*dx(v)
32     + dy(u)*dy(v)
33   )
34   - int2d(Th)(
35     f*v
36   )
37   + on(1, u=0);
38
39
40
41 // Adaptmesh loop
42 for (int i = 0; i < 4; i++){
43   Poisson;

```

```

44 Th = adaptmesh(Th, u, err=error);
45 error = error/2;
46 }
47
48 // Plot
49 plot(u);

```

Usando os comandos de visualização do *FreeFEM* é possível obter comparações com os gráficos obtidos por *MATLAB*:

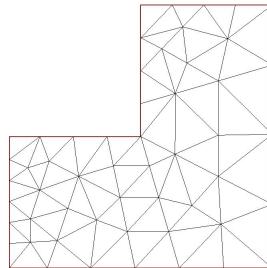


Figura 16: O domínio Ω e a sua respetiva triangulação no *FreeFem*.

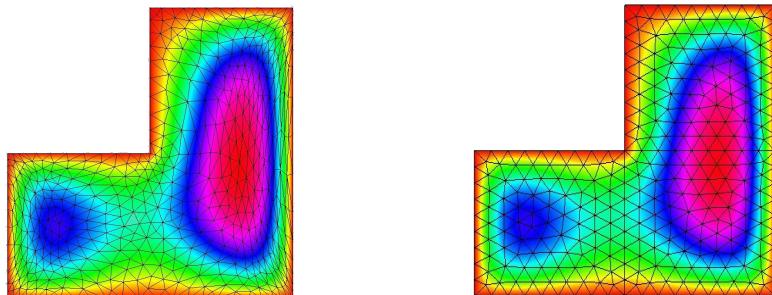


Figura 17: Solução MEF usando o software *FreeFEM* (à esquerda) versus solução *MATLAB* (à direita).

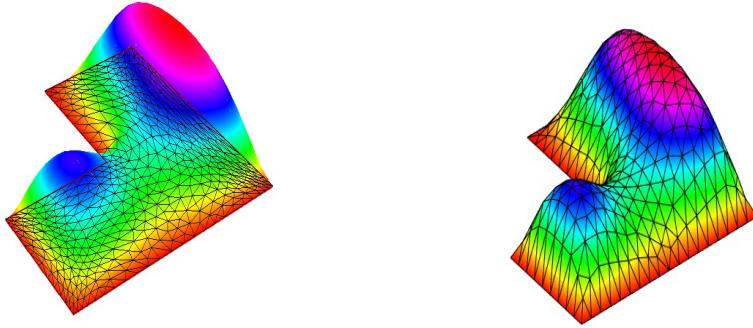


Figura 18: Visualização 3D MEF (à esquerda) usando o software *FreeFEM* versus visualização *MATLAB* (à direita).

Como se pode observar nas Figuras 18 e 19, as soluções obtidas são bastante semelhantes. No entanto, devido ao facto de ser baseada na linguagem de programação *C++*, o software *FreeFEM* é mais rápido a compilar.

Escolhendo ainda outro domínio $\Omega := [0, 1] \times [0, 1]$, é possível obter comparações para a função fonte $f(x, y) = 2y(1 - y) + 2x(1 - x)$. Para tal basta alterar no *script* anterior as fronteiras do domínio. Desta forma, temos simplesmente:

```

1 border ba(t=0, 1){x=t; y=0; label=1;}
2 border bb(t=0, 1){x=1; y=t; label=1;}
3 border bc(t=1, 0){x=t; y=1; label=1;}
4 border bd(t=1, 0){x=0; y=t; label=1;}
```

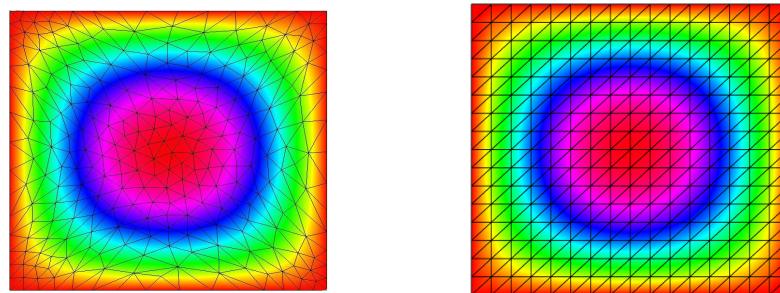


Figura 19: Solução MEF usando o software *FreeFEM* (à esquerda) versus solução *MATLAB* (à direita).

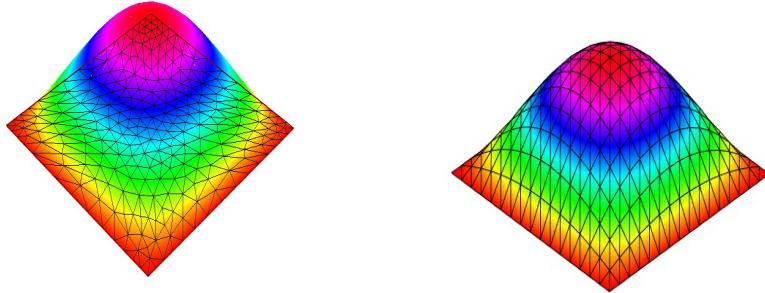


Figura 20: Visualização 3D MEF usando o software *FreeFEM* (à esquerda) versus visualização *MATLAB* (à direita).

Olhando para as Figuras 20 e 21, conclui-se que as soluções são praticamente idênticas, as pequenas diferenças resultam de malhas algo diferentes. Outro benefício da utilização do *FreeFEM* é a possibilidade de utilizar elementos de ordem superior a um. Para tal, basta alterar o parâmetro $P1$ na função *fespace*.

6 Conclusão

A aproximação da solução da equação de Poisson, tanto no caso **1D** como em **2D**, necessitou, primeiramente, da sua formulação em forma fraca, traduzindo-se o problema em forma variacional. Seguidamente, foi feita uma discretização do domínio Ω e escolha de bases seccionalmente polinomiais (lineares ou quadráticos), permitindo a tradução do problema num sistema de equações lineares. Finalmente, e após solução do sistema, foram observados os erros empíricos e a sua respectiva convergência em função de h , tamanho da malha, confirmando-se as estipulações teóricas. Adicionalmente, somente para o caso **1D**, foi analisado o elemento de segunda ordem, construído através da introdução de uma função bolha, verificando o seu efeito na diminuição do erro entre a solução aproximada e a solução exata. Paralelamente, foi ainda possível estabelecer uma comparação entre o solver *MATLAB* desenvolvido ao longo do trabalho e as soluções obtidas através do uso de um *software* próprio para a aplicação do método dos elementos finitos, o *FreeFEM*.

Em suma, este projeto serve como ilustração da aplicação do método dos elementos finitos a uma equação diferencial parcial, a equação de Poisson, analisando-se discretizações tanto no caso unidimensional como no caso bidimensional. Desta forma, o objetivo do projeto foi concluído, visto que foi possível a modelação da equação de Poisson e a construção de um ponto de partida para a resolução de outras E.D.P's, demonstrando-se a sua utilidade futura.

7 Apêndice Matemático

Nesta secção são introduzidas noções teóricas extra que servem de base para a secção dos conceitos teóricos no início do relatório. Foi decidida criar esta secção não só para o interessado como para não interromper a fluidez do desenvolvimento teórico e, sobretudo, porque estes resultados não estão diretamente associados à implementação do método.

7.1 Espaços de Sobolev

Antes de definir formalmente este tipo de espaços, é necessário introduzir a ideia de derivada

fraca, isto é, procura-se extender a definição de derivada para funções que não são diferenciáveis num sentido clássico. Para tal, observa-se que o valor do integral não depende do comportamento da função integrada num conjunto finito de pontos, dado que um ponto tem medida nula. Relembrando a fórmula de integração por partes e definindo $L_{loc}^1(I)$ como sendo o espaço de funções que são Lebesgue integráveis em todo o subconjunto compacto de I ; diz-se que $u \in L_{loc}^1(I)$ é fracamente diferenciável se existe $v \in L_{loc}^1(I)$ tal que $\int_I v \cdot \varphi \, dx = -\int_I u \cdot \varphi' \, dx$ para todo o $\varphi \in C_0^\infty(I)$. Assim é generalizado o conceito de derivada para funções que são somente Lebesgue integráveis, podendo não ser diferenciáveis, ou seja, a derivada fraca é definida em quase todo o I . Adicionalmente, uma função u é n -vezes fracamente diferenciável se a sua $(n-1)$ -ésima derivada fraca for fracamente diferenciável.

Um espaço de Sobolev é obtido definindo o espaço $H^m(I) \subset L^2(I)$ como sendo aquele cujos elementos são as funções m -vezes fracamente diferenciáveis munidas de produto interno dado por $(u, v)_{H^m(I)} = \sum_{k=0}^m (\frac{d^k u}{dx^k}, \frac{d^k v}{dx^k})_{L^2(I)}$ e norma $\|u\| = \sqrt{(u, v)_{H^m(I)}}$. É também importante definir a semi-norma de $H^m(I)$ dada por $|u|_{H^m(I)} := \left\| \frac{d^{(m)} u}{dx^{(m)}} \right\|_{L^2}$. Em **1D** as funções no espaço $H^1(I)$ são contínuas e portanto as fronteiras estão bem definidas. Desta forma, definimos o nosso primeiro espaço de Sobolev $H_0^1 := \{u \in H^1(I) \mid u(a) = u(b) = 0\}$. É de notar que $|u|_{H^1(I)}$ define uma norma em H_0^1 e que o espaço $H^1(I)$ é composto de funções quadrado integráveis cuja primeira derivada é também de quadrado integrável.

7.2 Problema variacional e existência de solução única

Considere o seguinte problema variacional: encontrar $u \in V$ que satisfaz $a(u, v) = L(v) \quad \forall v \in V$, em que V é um espaço de Hilbert com norma $\|\cdot\|_V$, a é uma aplicação $V \times V \rightarrow \mathbb{R}$ bilinear e L é uma aplicação $V \rightarrow \mathbb{R}$ linear. Supõe-se ainda que a forma bilinear a e a forma linear L satisfazem as seguintes propriedades:

- Continuidade de a : $a(u, v) \leq C \|u\|_V \|v\|_V$
- Elipticidade de a : $a(u, u) \geq \alpha \|u\|_V^2$
- Continuidade de L : $|L(V)| \leq C_L \|v\|_V$

em que C, α e C_L são constantes positivas independentes de u e v .

Agora define-se o teorema principal desta secção e que é fundamental para a prova da existência de solução única para o problema de Poisson.

Teorema (Teorema de Lax-Milgram): Seja a uma aplicação bilinear dotada de continuidade e elipticidade e L uma aplicação limitada. Então existe um e um só $u \in V$ tal que $a(u, v) = L(v) \quad \forall v \in V$, isto é, é solução do problema variacional.

Corolário: O problema de Poisson tem solução única.

Dem:

A demonstração faz-se por prova sucessiva das propriedades. Apresenta-se a prova apenas no caso **1D**.

Continuidade de a : Para mostrar a continuidade do problema de Poisson em **1D** tem que se provar que:

$$a(u, v) \leq \|u\|_{H^1} \|v\|_{H^1}$$

Efetivamente, pela desigualdade de Cauchy-Schwarz vem:

$$|a(u, v)| = \left| \int_0^1 \frac{du}{dx} \frac{dv}{dx} \right| = |(u', v')_{L^2(I)}| \leq \|u'\|_{L^2(I)} \|v'\|_{L^2(I)}$$

Adicionalmente, repare-se que $\|w\|_{H^1(0,1)} := \left(\|w\|_{L^2(I)}^2 + \|w'\|_{L^2(I)}^2 \right)^{1/2}$. Como $\|w\|_{L^2(I)} \geq 0$ então é claro que $\|w'\|_{L^2(I)} \leq \|w\|_{H^1(0,1)}$. Portanto, temos o resultado pretendido.

Elipticidade de a :

A elipticidade requer um resultado adicional chamado desigualdade de Poincaré-Friedrichs. Efetivamente dado $I = (a, b)$ e definindo $s = b - a$ então existe uma constante $C_P > 0$ tal que:

$$\|v\|_{L^2(I)} \leq C_P \|v'\|_{L^2(I)} \quad \forall v \in H_0^1(I)$$

Por definição:

$$a(u, u) = \int_0^1 (u')^2 dx = \|u'\|_{L^2(I)}$$

E também é verdade que $\|u'\|_{L^2(I)} = \frac{1}{2} \|u'\|_{L^2(I)} + \frac{1}{2} \|u'\|_{L^2(I)}$. Usando a desigualdade apresentada acima então $\frac{1}{2} \|u'\|_{L^2(I)} \geq \frac{1}{2C_P} \|u\|_{L^2(I)}$. Logo:

$$\|u'\|_{L^2(I)}^2 \geq \frac{1}{2} \|u'\|_{L^2(I)}^2 + \frac{1}{2C_P} \|u\|_{L^2(I)}^2 \geq \min \left\{ \frac{1}{2}, \frac{1}{2C} \right\} \left(\|u'\|_{L^2(I)}^2 + \|u\|_{L^2(I)}^2 \right)$$

E então a condição de elipticidade é cumprida para $\alpha = \min \left\{ \frac{1}{2}, \frac{1}{2C} \right\}$.

Continuidade de L :

Usando novamente a desigualdade de Cauchy-Schwarz:

$$|L(v)| = \left| \int_I f v \, dx \right| = |(f, v)_{L^2(I)}| \leq \|f\|_{L^2(I)} \|v\|_{L^2(I)} \quad \forall v \in H_0^1(I)$$

Como $\|v\|_{L^2(I)} \leq \|v\|_{H^1}$ a condição é satisfeita.

7.3 Resultados sobre interpolação

Considerando uma função contínua w e uma função πw que interpola essa mesma função nos nós $\{x_1, \dots, x_n\}$, contínua e linear em cada intervalo de partição $I_j = [x_j, x_{j+1}]$ tal que $\pi w(x) = \sum_{j=1}^N w(x_j) \hat{\varphi}_j(x)$ então é possível estabelecer as seguintes desigualdades;

$$\|w - \pi w\|_{L^2(I)} \leq h^2 \|w''\|_{L^2(I)},$$

$$\|(w - \pi w)'\|_{L^2(I)} \leq h \|w''\|_{L^2(I)}.$$

8 Bibliografia

- [1] Hannukainen, A., Finite Element Method, Lecture Notes, Universidade Aalto, 2021.
- [2] Larson, M.G. e F. Bengzon, The Finite Element Method: Theory, Implementation, and Applications, Springer, 2013.
- [3] Videman, J., Matemática Computacional, Aula XX, Departamento de Matemática, IST, 2021
- [4] Link para IDE: <https://freefem.org/tryit>
- [5] Link para compilador: <https://github.com/FreeFem/FreeFem-sources/releases>