INSTITUTO POLITÉCNICO DE LISBOA
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Licenciatura em Matemática Aplicada à Tecnologia e à Empresa

Projeto Final de Licenciatura

Estabilização do método do elementos finitos para a resolução de um problema de adveção-difusão

Número 43848. Nome Sérgio Calheiros Luzita[1]
Professor José Alberto de Sousa Rodrigues[2]

26 de Julho de 2020

[1]a43848@alunos.isel.pt
[2]jose.rodrigues@isel.pt

Conteúdo

1 Introdução **3**

2 Metodologia **4**

3 Formulação do problema 1D **5**
3.1 Equação do calor 5
3.1.1 Problema físico 5
3.2 Problema variacional 6
3.2.1 Introdução aos problemas variacionais 6
3.2.2 Formulação variacional para o problema do calor 6

4 Aplicação do método dos elementos finitos ao problema do calor **9**

5 Análise do número de condição **14**
5.1 Número de condição 14
5.2 Relação entre k(A) e o erro 17

6 Resultados numéricos **18**
6.1 Estudo do tempo de resolução 19
6.2 Estudo do erro 21
6.3 Majoração do erro 22

7 Problema de advecção-difusão **24**
7.1 Introdução de um método de estabilização 27
7.1.1 Cálculos da formulação fraca estabilizada de um novo problema 29
7.2 Comparação de resultados estabilizado vs não estabilizado 29
7.2.1 1º Problema 29
7.2.2 2º Problema 33

8 Análise do problema de difusão bidimensional **37**

9 Análise do problema de camada limite bidimensional **40**

10 Conclusão **44**

A Quadratura de Gauss-Legendre **46**

B Teoremas fundamentais da teoria do método de elementos finitos **48**
2.1 Alguma definições prévias 48

C Código Python **50**
3.1 1º Problema não Estabilizado 50
3.1.1 Resultados Obtidos 55
3.2 1º Problema Estabilizado 56
3.2.1 Resultados Obtidos 61
3.3 2º Problema não Estabilizado 62
3.3.1 Resultados Obtidos 68
3.4 2º Problema Estabilizado 69
3.4.1 Resultados Obtidos 74

D Código freeFEM++ **76**
4.1 Problema FreeFem bidimensional 76
4.1.1 Resultados Obtidos 77
4.2 Problema camada limite FreeFem bidimensional sem estabilização 77
4.2.1 Resultados Obtidos 78
4.3 Problema camada limite FreeFem bidimensional estabilizado 78
4.3.1 Resultados Obtidos 79

1. Introdução

Correntemente a simulação computacional é amplamente utilizada nas empresas para realizar análises de forma a melhorar a qualidade dos projetos. Algumas dessas análises são realizadas por meio de softwares que utilizam o Método dos Elementos Finitos (MEF), os quais possibilitam a obtenção de respostas para inúmeros problemas de concretos.

O MEF é um conceito de simulação desenvolvido a partir da base teórica estabelecida com a publicação de um conjunto de trabalhos científicos na década de 1940. Foi criado como técnicas numéricas para encontrar soluções aproximadas para problemas com condições na fronteira para equações diferenciais parciais, o MEF é baseado na subdivisão de um domínio de problema em partes mais simples - chamadas elementos finitos - e no cálculo de métodos variacionais para minimizar uma função de erro associada.

O MEF resulta diretamente da necessidade de resolver problemas complexos de elasticidade e análise estrutural em engenharia civil e aeronáutica. O primeiro desenvolvimento remonta ao trabalho de A. Hrennikoff em 1941 [1] e R. Courant em 1943 [2]. Embora esses pioneiros usassem perspectivas diferentes em suas abordagens de elementos finitos, cada um deles identificou uma característica comum e essencial: incluir a discretização de um domínio contínuo num conjunto de subdomínios discretos, geralmente chamados de elementos. Outra contribuição matemática fundamental para o MEF é representada pelo livro "Uma Análise do Método dos Elementos Finitos", de Gilbert Strang e George Fix, publicado pela primeira vez em 1973 [3]. Desde então, o MEF tem sido generalizado para a modelação computacional de sistemas físicos em muitas disciplinas de engenharia, incluindo eletromagnetismo, transferência de calor e dinâmica de fluidos.

Uma das preocupações quando se usa o MEF é o estudo das condições suficientes para a existência e unicidade da solução. Sendo um método que se traduz pela resolução de um sistema algébrico de grade dimensão, outra grande preocupação é a obtenção da solução numérica do sistema. Para tal concorrem as propriedades numéricas da matriz do sistema. Em qual nível, pretende-se que a solução obtida dependa continuamente dos dados.

Com este trabalho pretende-se a aquisição de competências ao nível da implementação do MEF e de métodos de estabilização na resolução numérica de um problema de advecção-difusão com forte advecção.

Serão abordadas questões Físicas, e da Análise Numérica associadas aos problemas a resolver.

2. Metodologia

Com vista a aplicação do Método de Elementos Finitos (de Lagrange) e à consciencialização das suas múltiplas etapas começamos por considerar o problema de difusão unidimensional e a sua implementação usando o software Python. Consideramos a formulação fisica do problema (dita formulação forte), aplicamos métodos variacionais com vista à obtenção da formulação fraca do problema e posterior implementação. Paralelamente determinaremos uma solução exata do problema e apresentaremos o estudo do erro absoluto cometido. Adicionalmente faremos a determinação das constantes de majoração do erro absoluto em função do parâmetro de discretização h. Posteriormente abordaremos o problema de difusão bidimensional e usaremos o software FreeFem++ para a obtenção da solução numérica e estudo do respetivo erro em função do grau dos polinómios utilizados. A última parte deste trabalho incide sobre o estudo de um método de estabilização aplicado ao MEF para a resolução de um problema de advecção-difusão com adveção dominante.

3. Formulação do problema 1D

3.1 Equação do calor

Interessa-nos estudar a resolução numérica de um problema de propagação de calor unidimensional (numa barra, por exemplo).

Vamos começar por deduzir as equações constitutivas.

3.1.1 Problema físico

Designamos por u a função da temperatura definida no domínio $\Omega = [0,1]$ e q o fluxo de calor neste intervalo, sujeita a uma fonte de calor de f. Assumindo que a temperatura é nula na fronteira do domínio $\Gamma = \{0,1\}$ obtemos um caso estacionário do fenómeno de propagação do calor numa barra localizada em Ω:

$$\begin{aligned} -q &= ku' && \text{(Lei de Fourier)} \\ q' &= f && \text{(Consevação de energia)} \\ u(0) &= u(1) = 0 && \text{(Condições de fronteira)} \end{aligned}$$

onde k é a condutividade térmica do material. Das equações (3.1.1) e (3.1.1) obtemos

$$ku''(x) = f(x) \qquad \text{para } 0 < x < 1 \tag{3.1}$$

sujeita às condições de fronteira (3.1.1), obtendo-se o problema

$$\begin{cases} -ku''(x) = f(x) & \text{para } x \in]0,1[\\ u(x) = 0 & \text{para } x = 0,\ x = 1 \end{cases} \tag{3.2}$$

Sendo f uma função contínua em $]0,1[$, $f \in C^0(0,1)$ [1], resulta que $u \in C^2(0,1) \cap C^0[0,1]$. Por esta razão o problema (3.2) designa-se por problema forte. Na secção seguinte iremos introduzir o designado por problema fraco. Esta designação de problema forte e problema fraco baseia-se na ordem de derivação presente.

Teorema 3.1.1. Existe uma solução $u \in C^2(0,1) \cap C^0[0,1]$ do problema (3.2).

[1] $C^0(0,1)$ designa o conjunto das funções contínuas em $]0,1[$.

Demonstração. Consideramos o primeiro problema: determinar a função $\bar{u}$ tal que

$$-k\bar{u}'' = f, \qquad \text{em }]0,1[, \tag{3.3}$$
$$\bar{u}(0) = 0 \tag{3.4}$$
$$k\bar{u}'(0) = 0 \tag{3.5}$$

Neste novo problema, as condições de fronteira são impostas no mesmo extremo do intervalo em $x = 0$ pelo que a teoria clássica das equações diferenciais lineares (teorema de Cauchy-Lipschitz), diz que existe uma única solução $\bar{u} \in C^2(0,1)$ do problema do Cauchy (3.2).

Por outro lado, verificamos que as funções

$$v_1 = 1 \quad \text{e} \quad v_2 = x$$

são soluções de $C^2(0,1)$, linearmente independentes do problema homogéneo $-kv'' = 0$. Consideramos agora a combinação linear $u = \bar{u} + \alpha v_1 + \beta v_2$, é facil encontrar α e β tal que $u(0) = \alpha = 0$ e $u(1) = \bar{u}(1) + \alpha + \beta = 0$. A função $u \in C^2(0,1) \cap C^0[0,1]$ é uma solução clássica (ou forte) do problema (3.2). □

Infelizmente a aplicação $D^2 : C^2(0,1) \mapsto C^0(0,1)$ não é invertível. Esses requisitos de continuidade são muito rígidos e não traduzem razoavelmente todos os fenómenos físicos (também ocorrem descontinuidades na vida real).

De seguida iremos ver uma interpretação alternativa da igualdade entre duas funções. Em vez de considerarmos a formulação baseada na avaliação pontual, vamos considerar a igualdade da integração pelo produto por funções ditas de teste. A integração é mais permissiva do que a avaliação pontual. Assim podemos ignorar o mau comportamento num conjunto de área ou comprimento nulo. Uma solução clássica satisfará a nossa nova formulação variacional, mas a formulação variacional será mais geral.

3.2 Problema variacional

3.2.1 Introdução aos problemas variacionais

A formulação variacional, também conhecida como formulação fraca, permite encontrar de maneira simples a solução numérica de problemas de equações diferenciais parciais (EDPs), uma vez que com a teoria clássica das EDPs é muito complexo encontrar uma solução que satisfaça todas as condições. Esta técnica é descrita como uma alternativa na qual os problemas com EDPs são escritos na forma integral, o que resulta em equações mais simples usando métodos de álgebra linear sobre um espaço vetorial de dimensão infinita ou espaço funcional. O sucesso do método dos elementos finitos decorre em grande parte das propriedades da formulação variacional.

3.2.2 Formulação variacional para o problema do calor

Nesta secção vamos introduzir uma formulação variacional para o problema (3.2). Começamos por considerar o espaço de funções V definido por:

$$V = \left\{ C^2(0,1) \cap C^0[0,1] : \quad v(0) = v(1) = 0 \right\} \tag{3.1}$$

Supomos $f :]0,1[\to \mathbb{R}$ com $f \in C^0(0,1)$. Para obter a formulação fraca de (2.1) , primeiro multiplicamos por os dois lados da primeira igualdade em (3.1) por $v \in V$, depois integramos os dois lados da equação e obtemos:

$$\begin{aligned} -\int_0^1 ku''(x)v(x)dx &= \int_0^1 ku'(x)v'(x)dx - \left[u'(1)v(1) - u'(1)v(1)\right] \qquad (3.2) \\ &= \int_0^1 ku'(x)v'(x)dx - [u'(1)v'(1) - u'(0)v'(0)] \\ &= \int_0^1 ku'(x)v'(x)dx \qquad (3.3) \end{aligned}$$

para $u \in V$.

O problema (3.2) pode ser escrito como, utilizando uma notação mais abstrata:

$$\begin{cases} \text{Determinar } u \in V \text{ tal que} \\ a(u,v) = b(v) \qquad\qquad \forall v \in V \end{cases} \qquad (3.4)$$

Com

$$a(u,v) = \int_0^1 ku'(x)v'(x)dx$$

e

$$b(v) = \int_0^1 f(x)v(x)dx$$

Teorema 3.2.1. Os problemas (3.2) e (3.4) são equivalentes.

Demonstração. Da sequência 3.2-3.3 podemos concluir que uma solução do problema (3.2) também é solução do problema (3.4).

Vejamos o sentido inverso:

$$\begin{aligned} \int_0^1 ku'(x)v'(x)dx = \int_0^1 f(x)v(x)dx &\Leftrightarrow \int_0^1 ku'(x)v'(x) - f(x)v(x)dx = 0 \\ &\Leftrightarrow \int_0^1 \left(ku''(x) - f(x)\right)v(x)dx = 0 \qquad (3.5) \end{aligned}$$

Uma vez que (3.5) se verifica para todo o $v \in V$ concluimos

$$ku'' - f = 0$$

e $ku'' - f \in C^0(0,1)$. As condições de fronteira resultam do espaço V escolhido. □

Tal como foi referido anteriormente a imposição $u \in C^2(0,1)$ é muito restritiva. Adicionalmente da análise da formulação variacional podemos escolher, informalmente, como espaço V o seguinte

$$V = \{a(v,v) < +\infty \text{ e } b(v) < \infty \text{ e } v(0) = v(1) = 0\} \tag{3.6}$$

Para implementação do da formulação variacional através do Método dos Elementos Finitos precisamos de introduzir um novo conceito de diferenciação e novos espaços de funções, nomeadamente os espaços de Sobolev (cf. [4]) garantir a resolubilidade do problema (3.4), razão pela qual introduzimos o novo espaço de funções

$$V = \left\{v \in H^1(0,1) \text{ e } v(0) = v(1) = 0\right\} \tag{3.7}$$

Teorema 3.2.2. O problema (3.4) admite uma só solução em V.

Demonstração. A demonstração do teorema pode ser visto em [5] ou [4]. Contudo, usando o teorema 3.2.2 e o teorema 3.1.1 também podemos concluir a existência e unicidade de solução para este caso particular. □

4. Aplicação do método dos elementos finitos ao problema do calor

Considerando o problema 3.4 formulado no espaço V, a aplicação do método dos elementos finitos baseia-se na utilização de uma formulação de Galerkin em que se pretende resolver um problema aproximado num subespaço de dimensão finita $V_h \subset V$:

$$\begin{cases} \text{Determinar } u_h \in V_h \text{ tal que} \\ \\ a(u_h, v_h) = b(v_h) \qquad\qquad \forall v_h \in V_h \end{cases}$$

Vamos ver resumidamente como construir o espaço V_h e o sistema linear associado:

Definição 4.0.1. **Unissolvência do espaço P em relação ao conjunto Σ**
Consideremos o conjunto $\Sigma = \{x_0, x_1, \ldots, x_n\}$ de pontos distintos de Ω e P um espaço vetorial de dimensão finita de funções reais definidas sobre $\mathcal{T}_h$, dizemos que Σ é P-unisolvente se para cada sequência de valores reais $a_0, a_1, \ldots, a_n$ existe apenas uma função $p \in P$ tal que

$$p(x_i) = a_i, \qquad i \in \{0, 1, \cdots, n\}$$

Ou seja, todas as funções do espaço P são definidas univocamente pelos valores que tomam sobre os pontos de Σ

Definição 4.0.2. Partição de um domínio
Sobre um domínio Ω definimos uma partição $\mathcal{T}_h$ de subdomínios (neste caso intervalos), compactos e conexos, que verificam as seguintes propriedades (para dimensão 1):

- $\Omega = \underset{K \in \mathcal{T}_h}{\cup} K$
- $K, L \in \mathcal{P}_h, \quad K \cap L = \emptyset,$ ou $K \cap L =$ um ponto

Definição 4.0.3. Diâmetro de uma partição
$h = \underset{K \in \mathcal{T}_h}{\max} |K|$, com $|K|$ a medida dos subdomínios (neste caso o comprimento dos intervalos).

Definição 4.0.4. Elementos Finitos de Lagrange
Um elemento finito de Lagrange é um tripleto $(\mathcal{T}_h, \Sigma, P)$ tais que :

- $\mathcal{T}_h$ é uma partição do domínio Ω
- $\Sigma = \{x_0, x_1, \ldots, x_n\}$ o conjunto de pontos distintos de Ω
- P é um espaço vetorial de dimensão finita de funções reais definidas sobre $\mathcal{T}_h$, tal que Σ é P-unisolvente.

Vamos agora construir um subespaço V_h de dimensão finita do espaço V definido anteriormente, constituido por funções contínuas em Ω polinomiais (neste caso de grau 1) por elementos.
Consideramos o conjunto $0 = x_0 < x_1 < ... < x_{n-1} < x_n = 1$,uma partição do intervalo $[0, 1]$ em subintervalos $I_j = [x_{j-1}, x_j]$, sobre esta partição vamos designar por

$$\{\varphi_j,\ 0 \leq j \leq n\} \tag{4.1}$$

a base de funções de V_h verificando a seguinte propriedade:

$$\varphi_j(x_i) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \quad 0 \leq i\,,\, j \leq n \tag{4.2}$$

como podemos observar na Figura 4.1.

Agora deixamos V_h ser o conjunto de funções v de tal modo que v é linear em cada subintervalo I_j , v é continua em $[0, 1]$ e $v(0) = v(1) = 0$.

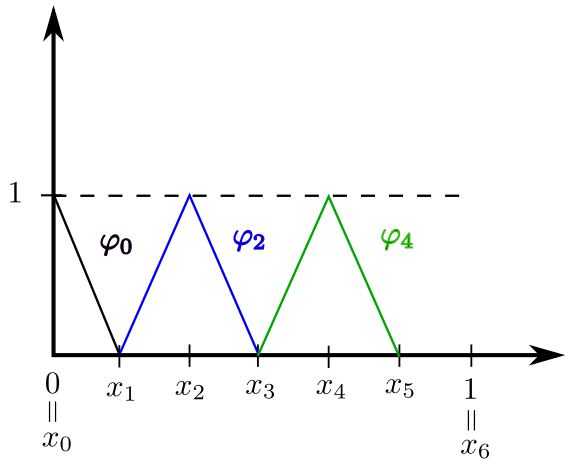


Figura 4.1: Exemplo de funções φ_0 , φ_2 , φ_4

Uma função $v \in V_h$ então tem a representação

$$v(x) = \sum_{i=0}^{n} \eta_i \varphi_i(x) \quad x \in [0, 1] \tag{4.3}$$

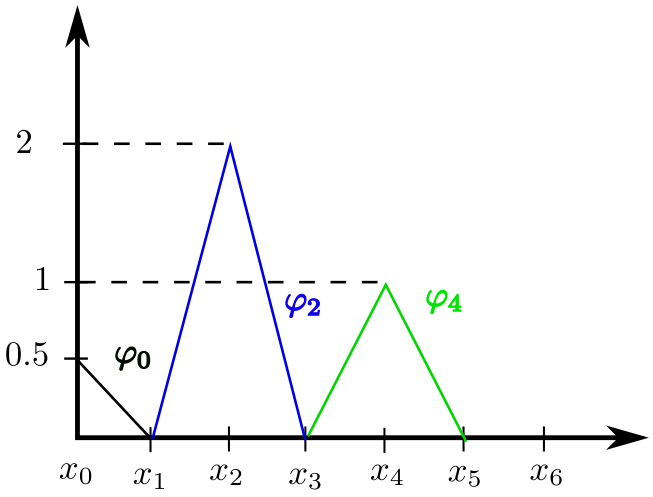


Figura 4.2: Exemplo do gráfico de uma função do espaço de dimensão finita V_h, $v(x) = \frac{1}{2}\varphi_0(x) + 2\varphi_2(x) + \varphi_4(x)$

com $\eta_i \in \mathbb{R}$, isto é , cada $v \in V_h$ pode ser escrito de uma forma única como uma combinação linear das funções base φ_i.

Vamos agora formular a discretização do problema (3.4) pelo método dos elementos finitos de Lagrange, com vista à obtenção de uma sua solução aproximada:

$$\begin{cases} \text{Determinar } u_h \in V_h \text{ tal que:} \\ a\left(u_h, v_h\right) = b\left(v_h\right) \qquad\qquad \forall v_h \in V_h \end{cases} \tag{4.4}$$

Onde a integração ser efetuada por quadratura de Gauss, com 3 pontos (ver Apendice A).

O método de elementos finitos para o problema de fronteira (4.4) pode ser formulado como um problema de minimização (ver Apendice 2.1):

$$\text{Determinar } u_h \in V_h \text{ onde } J(u_h) \leq J(v_h) \;\; \forall v_h \in V_h \tag{4.5}$$

Com $J(v) = \frac{1}{2}a(v,v) - b(v)$.

O problema (4.4) é usualmente referido como método de Galerkin, tal como inicialmente referimos, e (4.5) como o método de Ritz.

Vamos agora contruir o problema linear. Começamos primeiro por constatar que $u_h \in V_h$, ou seja $u_h(x) = \sum_{i=0}^{n} \xi_i \varphi_i(x)$, satisfaz

$$\begin{aligned} (u_h', \varphi_j') &= (f, \varphi_j) \qquad j = 0, ..., n \\ \left(\left[\sum_{i=0}^{n} \xi_i \varphi_i \right]', \varphi_j' \right) &= (f, \varphi_j) \qquad j = 0, ..., n \\ \sum_{i=0}^{n} \xi_i (\varphi_i', \varphi_j') &= (f, \varphi_j) \qquad j = 0, ..., n \end{aligned} \tag{4.6}$$

que é uma sistema linear de equações com $n+1$ e $\xi_0, ..., \xi_n$ incógnitas . Em forma matricial o sistema linear (4.6) pode ser escrito como

$$A\xi = b \tag{4.7}$$

onde $A = (a_{ij})$ é a matriz $(n+1) \times (n+1)$ com elementos $a_{ij} = (\varphi_i', \varphi_j')$ e onde $\xi = (\xi_0, ..., \xi_n)$ e $b = (b_0, ..., b_n)$ com $b_i = (f, \varphi_i)$ são vetores de dimensão $n+1$

$$A = \begin{bmatrix} a_{00} & \cdots & a_{0n} \\ \vdots & \ddots & \vdots \\ a_{n0} & \cdots & a_{nn} \end{bmatrix}, \xi = \begin{bmatrix} \xi_0 \\ \vdots \\ \xi_n \end{bmatrix}, b = \begin{bmatrix} b_0 \\ \vdots \\ b_n \end{bmatrix}$$

A matriz A é chamada a matriz de rigidez e b o vetor de peso, com terminologia das primeiras aplicações do método de elementos finitos em mecânica estrutural.
Os elementos $a_{ij} = (\varphi_i', \varphi_j')$ na matriz de rigidez A podem ser facilmente calculados. Primeiro observamos que $(\varphi_i', \varphi_j') = 0$ se $|i-j| > 1$ visto que neste caso para todos os $x \in [0,1]$ seja $\varphi_i(x)$ ou $\varphi_j(x)$ é igual a 0.Portanto a matriz A é tridiagonal e por isso, apenas os elementos na diagonal principal e as duas diagonais adjacentes poderão ser diferentes de zero. Temos para $j = 0, ..., n$

$$(\varphi_j', \varphi_j') = \int_{x_{j-1}}^{x_j} \frac{1}{h_j^2}\, dx + \int_{x_j}^{x_{j+1}} \frac{1}{h_{j+1}^2}\, dx = \frac{1}{h_j} + \frac{1}{h_{j+1}}$$

e para $j = 2, ..., n$

$$(\varphi_j', \varphi_{j-1}') = (\varphi_{j-1}', \varphi_j') = -\int_{x_{j-1}}^{x_j} \frac{1}{h_j^2}\, dx = \frac{1}{h_j}$$

Note também que a matriz A é simétrica e definida positiva visto que $(\varphi_i', \varphi_j') = (\varphi_j', \varphi_i')$ e com $v(x) = \sum_{j=1}^{M} \eta_j \varphi_j(x)$ temos

$$\sum_{i,j=0}^{n} \eta_i (\varphi_i', \varphi_j') \eta_j = \left(\sum_{i=0}^{n} \eta_i \varphi_i' \, , \, \sum_{j=0}^{n} \eta_j \varphi_j' \right) = (v', v') \geq 0$$

com igualdade só se $v' \equiv 0$, visto que como $v(0) = 0$ apenas se $v \equiv 0$, ou $\eta_j = 0$ para $j = 0, ..., n$. Podemos recordar que a matriz simétrica $S = (s_{ij})$ com tamanho $(n+1) \times (n+1)$, é dita ser definida positiva se

$$\eta \cdot S\eta = \sum_{i,j=0}^{n} \eta_i s_{ij} \eta_j > 0 \qquad \forall \eta \in \mathbb{R}^{n+1}, \eta \neq 0$$

Também podemos recordar que a matriz simétrica S é positiva definida se e só se os valores próprios de S forem estritamente positivos.
Visto que como uma matriz postiva definida não é singular segue-se que o sistema linear (4.7) tem uma solução única. Também podemos notar que A é esparsa, isto é, apenas poucos elementos de A são diferentes de zero(A é tridiagonal). Esta propriedade muito importante depende, como já vimos, do facto de que a função base φ_j ou V_h é diferente de zero apenas em alguns intervalos e dai irá interferir apenas com algumas outras funções base. O facto de que as funções base podem ser escolhidas desta forma é a característica distintiva importante do método de elementos finitos.

Neste caso especial de partição uniforme com $h_j = h = \dfrac{1}{n+1}$ o sistema (4.7) toma a forma algébrica:

$$-\frac{k}{h}\,u_{j-1} + \frac{2k}{h}\,u_j - \frac{k}{h}\,u_{j+1} = b_j \tag{4.8}$$

e com a seguinte forma matricial:

$$AX = B$$

$$\frac{k}{h}\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} \xi_0 \\ \vdots \\ \vdots \\ \vdots \\ \xi_n \end{bmatrix} = \begin{bmatrix} b_0 \\ \vdots \\ \vdots \\ \vdots \\ b_n \end{bmatrix} \tag{4.9}$$

com $b_i = \int_0^1 f(x)\varphi_i(x)dx, \quad i = 0, \ldots, n.$

A imposição das condições de fronteira faz-se por eliminação das váriaveis correspondentes aos graus de liberdade sobre a fronteira.

5. Análise do número de condição

5.1 Número de condição

O número de condição associado a um sistema linear $Ax = b$, é um número que estabelece uma estimativa da precisão que se pode obter para uma solução aproximada de x.

No entanto o numero de condição não tem em consideração os efeitos dos erros de arredondamento. A condição é uma propriedade da matriz e não do algoritmo ou da precisão de, por exemplo , o algoritmo do ponto flutuante utilizado pelo computador para resolver o sistema correspondente.

Definição 5.1.1. Número de condição
Designa-se por número de condição de uma matriz A relativamente à norma $\|.\|$, o valor:

$$k(A) = \|A\|\|A\|^{-1}$$

Diz-se que um problema com um número de condição baixo está bem condicionado, enquanto que um problema com um número de condição alto é considerado mal condicionado.

Em termos não matemáticos, um problema mal condicionado é um problema que para uma pequena variação nas variáveis independentes(ou o lado direto da equação), há uma grande variação na solução aproximada ou variável dependente. Isso significa que a solução correta para a equação se torna dificil de encontrar.

Definição 5.1.2. Norma vetorial
Dado um espaço vetorial X sobre o corpo K dos números reais ou complexos , uma função $\|\cdot\| : X \to \mathbb{R}^+$, é chamada norma se, e para quaisquer $x, y \in X$ e todo o $\alpha \in K$:

$$\|x\| = 0 \Leftrightarrow x = 0 \tag{5.1}$$
$$\|\alpha x\| = |\alpha|\|x\| \tag{5.2}$$
$$\|x + y\| \leq \|x\| + \|y\| \tag{5.3}$$

Definição 5.1.3. Norma euclideana

Quando uma matriz $A \in \mathbb{R}^{m\times n}$ é vista como a representação de uma aplicação linear entre os espaços euclideanos $\mathbb{R}^n$eR^m , a norma natural é dada pela norma operacional:

$$\|A\| = \sup_{x\in\mathbb{R}^n, x\neq 0} \frac{\|Ax\|}{\|x\|} \tag{5.4}$$

Esta norma tem as seguintes propriedades adicionais :

$\|AB\| \leq \|A\|\|B\|$, sempre que o produto está bem definido

$\|I\| = 1$, onde I é a matriz identidade

Definição 5.1.4. Norma matricial

Seja $\mathbb{R}^{m\times n}$ o espaço vetorial das matrizes $m\times n$ reais ou complexas. Uma norma $\|\cdot\|$ é uma função que associa a cada matriz um número real não negativo e satisfaz as propriedades:

$$\|A\| = 0 \Leftrightarrow A = 0 \tag{5.5}$$
$$\|\lambda A\| = |\lambda|\|A\| \tag{5.6}$$
$$\|A + B\| \leq \|A\| + \|B\| \tag{5.7}$$

Estas definições e propriedades podem ser vistas com maior detalhe no livro [6] .

Com vista a estudar a dificuldade de resolução númerica do sistema $Ax = b$ linear, que pretendemos resolver, vamos então considerar um sistema perturbado (perturbação δ) onde vamos observar o impacto que existe no sistema Ax, ao fazer pequenas alterações em b.

Assumindo um sistema de equação matricial linear:

$$Ax \quad = \quad b \tag{5.8}$$

Em seguida inserimos uma perturbação δ no sistema, transfomando-o num sistema perturbado:

$$A(x + \delta x) \quad = \quad b + \delta b \tag{5.9}$$

Começamos então por subtrair o sistema perturbado pelo sistema original:

$$A(x + \delta x) - Ax = (b + \delta b) - b$$

$$A\,\delta x = \delta b$$

e assim é possível verificar que a matriz do sistema é invertível. Utilizando o sistema da equação, vamos então encontrar uma propriedade importante para deduzir o numero de condição da matriz A, utilizando as propriedades anteriores 5.2 e 5.4 .

$$\begin{aligned} b = Ax \quad \Rightarrow \quad \|b\| = \|Ax\| \qquad &\leq \qquad \|A\|\,\|x\| \\ \|b\| \leq \|A\|\,\|x\| \quad &\underset{\substack{\|b\|\neq 0}}{\overset{\|x\|\neq 0}{\Longleftrightarrow}} \quad \frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|} \end{aligned}$$

Agora, vamos utilizar o sistema anteriormente calculado, para deduzir então o numero de condição:

$$\begin{aligned} \delta x \quad &= \quad A^{-1}\,\delta b \\ \|\delta x\| \quad &= \quad \|A^{-1}\,\delta b\| \leq \|A^{-1}\|\|\delta b\| \\ \frac{\|\delta x\|}{\|x\|} \quad &\leq \quad \|A^{-1}\|\frac{\|\delta b\|}{\|x\|} \leq \|A^{-1}\|\|\delta b\| \times \frac{1}{\|x\|} \\ &\leq \quad \|A^{-1}\|\|\delta b\|\frac{\|A\|}{\|b\|} = \|A^{-1}\|\|A\|\frac{\|\delta b\|}{\|b\|} \end{aligned}$$

Temos a assim, a seguinte inequação matricial:

$$\frac{\|\delta x\|}{\|x\|} \leq \|A^{-1}\|\|A\|\frac{\|\delta b\|}{\|b\|}$$

Onde $\dfrac{\|\delta x\|}{\|x\|}$ é o erro relativo na solução , $\dfrac{\|\delta b\|}{\|b\|}$ é o erro relativo no 2^{o} membro e
$\|A^{-1}\|\|A\| = k(A)$ é o número de condição de A, e mede a dificuldade da obtenção da solução x.

Propriedades do número de condição:

- $1 \leq \|A^{-1}\|\|A\| = k(A) \leq +\infty$

- $AA^{-1} = I$

- $I = AA^{-1} \Rightarrow \|I\| \leq \|A\|\|A^{-1}\| = k(A)$

Quando o número de condição da matriz é grande usamos uma matriz de précondicionamento P tal que PA^{-1} tenha um número de condição baixo.

Propriedades de uma matriz de précondicionamento:

- $(P^{-1}A)x = (P^{-1}b)$

- $(Ax = PP^{-1}b)$

5.2 Relação entre k(A) e o erro

É possível concluir que um número de condição elevado não nos permite estabelecer boas majorações para o erro relativo. Quanto maior for o número de condição, pior será a majoração do erro relativo obtida. Consequentemente para matrizes cujo número de condição seja elevado, um pequeno erro relativo no vetor b pode provocar um grande erro relativo na solução do sistema.

O número de condição $k(A) = \|A\|\|A\|^{-1}$ determina o nível de ampliação que o erro relativo do lado direito da equação toma.

O número da condição associado à equação linear $Ax = b$ fornece um limite da imprecisão da solução x após a aproximação. È importante notar que isso ocorre antes que os efeitos de arredondamento do erro sejam levados em consideração. O condicionamento é uma propriedade da matriz. Em particular, deve-se pensar no número da condição como sendo (muito aproximadamente) a taxa na qual a solução x mudará em relação a uma mudança em b. Portanto, se o número da condição for grande, mesmo um pequeno erro em b pode causar um grande erro em x. Por outro lado, se o número da condição for pequeno, o erro em x não será muito maior que o erro em b.

O número da condição é definido com maior precisão como sendo a proporção máxima do erro relativo em x para o erro relativo em b.

6. Resultados numéricos

Utilizando o método de elementos finitos implementado com o software Python [7], realizamos dois testes com $n+1=6$ e $n+1=100$ nós, para resolver o problema (4.4) que em termos matriciais corresponde ao sistema (4.9).

Para avaliar a qualidade da solução numérica obtida vamos estudar o correspondente erro.

Começamos por calcular o erro para um caso concreto e de seguida obteremos estimativas de erro gerais.

Determinamos a solução exata para um problema concreto e comparamos com a solução obtida pelo método dos elementos finitos. Na determinação da solução exata usamos a seguinte técnica:

i. Escolha da solução u_0 que verifica as condições de fronteira do problema (3.2):

$$u_0(x) = (x-1)^2 x^2 \tag{6.1}$$

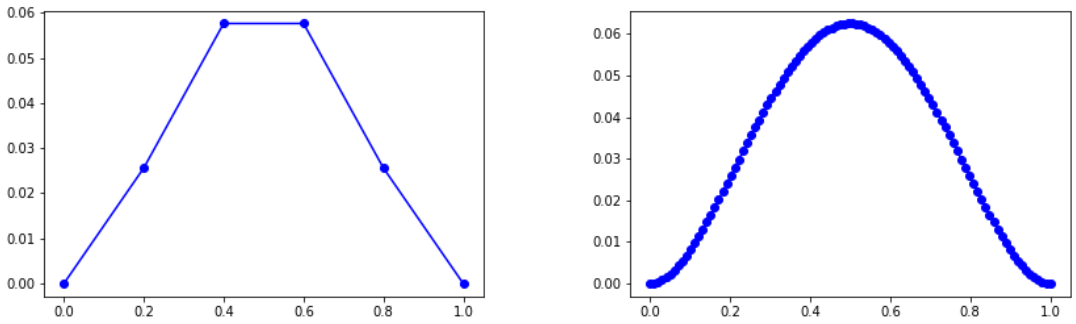
$u_0(0) = u_0(1) = 0$

ii. Determinação do 2^o membro f do problema (8.1) para o qual u_0 é solução exata:
$u'(x) = 2(x-1)(x^2) + 2x(x-1)^2 = 2x(x-1)(2x-1)$
$u''(x) = 2(x-1)(2x-1) + 2x(2x-1) + 4x(x-1) = 12x^2 - 12x + 2$

Quando $f(x) = -k(12x^2 - 12x + 2)$
u_0 é a solução do problema (3.2).

Posteriormente fazemos uma análise sobre o número de condição e o seu respetivo erro.

Para os testes realizados, o número de condição da matriz A, $k(A)$, varia entre 32.62 e 4746.29 para um número de nós igual a 6 e 100, respetivamente, tendo-se obtido os resultados da Figura 6.1 e na Figura 6.2 pode-se observar a evolução do número de condição com base no parâmetro h.



(a) Solução aproximada com 6 nós

(b) Solução aproximada com 100 nós

Figura 6.1: Solução Aproximada

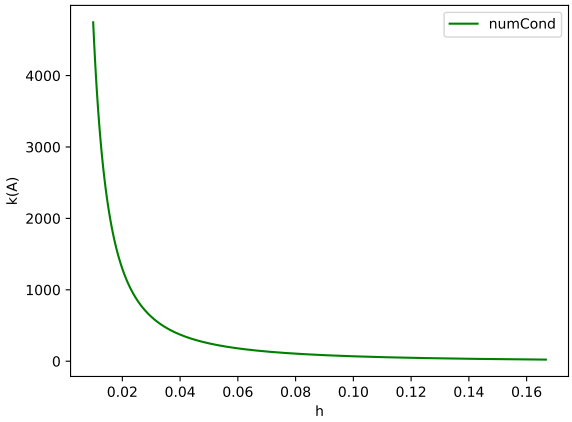


Figura 6.2: Numero de condição com base no parâmetro h

6.1 Estudo do tempo de resolução

Com vista o estudo da dificuldade de resolução do sistema $Ax = b$ de onde retiramos medidas como o erro relativo ou absoluto do problema em estudo, e o número de condição $k(A)$, vamos então estudar o tempo de resolução do sistema para tirar conclusões.

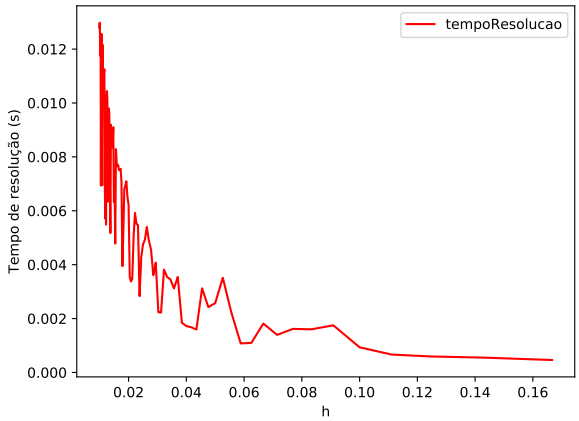


Figura 6.3: Tempo de resolução

Observando a Figura 6.3, é possivel verificar que à medida que o parâmetro h vai diminuindo o tempo de resolução do sistema vai aumentando, o que é expectável pois quanto menor for h, maior será a matriz A e por isso era de esperar que a resolução do sistema seja mais dificil.

Temos também na Figura 6.4 , um gráfico que demonstra a evolução do tempo de resolução consoante o número de condição e vice-versa. É de notar que a evolução dos dois, parece seguir a forma de uma função linear.

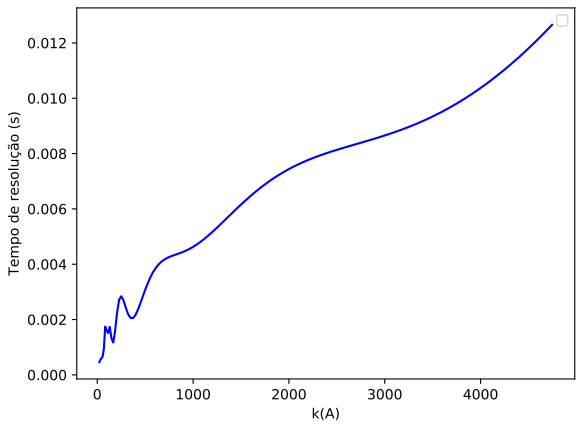


Figura 6.4: Tempo de resolução em função do número de condição h

6.2 Estudo do erro

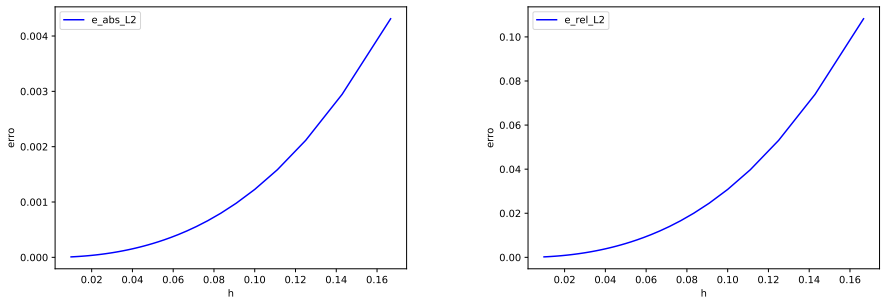
Definição 6.2.1. Sobre o espaço V definimos a seguinte norma (ver Anexo 2.1):

$$\|v\|^2 = \int_0^1 v(x)^2 \ dx \tag{6.1}$$

$$\text{Erro Absoluto} \qquad E_a(h) = \|u_0 - u_{0h}\| \tag{6.2}$$

$$\text{Erro Relativo} \qquad E_r(h) = \frac{\|u_0 - u_{0h}\|}{\|u_0\|} \tag{6.3}$$

Agora iremos calcular a evolução dos erros absoluto e relativo em função do parâmetro de discretização h, utilizando a implementação do método de elementos finitos na ferramenta Python como se pode observar nas Figuras 6.5(a) e 6.5(b).

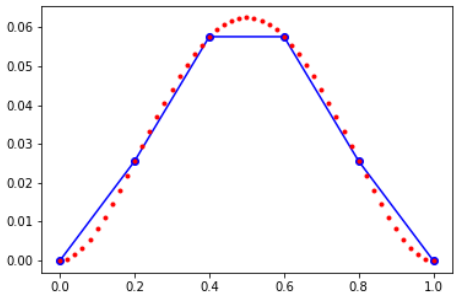


(a) Estudo do erro absoluto da função teste u_0

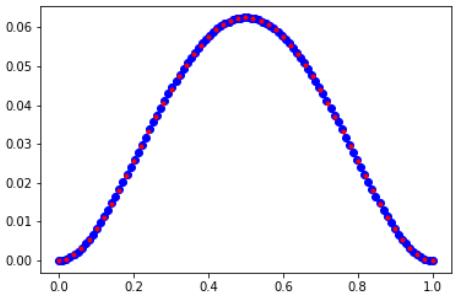
(b) Estudo do erro relativo da função teste u_0

Figura 6.5: Estudo do erro absoluto e erro relativo

Seguidamente apresentamos dois exemplos com a comparação entre a solução exata e a solução aproximada pelo método de elementos finitos, com base no número de nós utilizado:



(a) Solução aproximada com 6 nós



(b) Solução aproximada com 100 nós

Figura 6.6: Gráfico da solução aproximada (em azul) e gráfico da solução exata (em vermelho)

Observando as Figuras 6.5(a) e 6.5(b) e as soluções aproximadas em 6.6, é possível notar que à medida que o parâmetro h vai diminuindo, ou seja, o número de nós vai aumentando, a solução aproximada torna-se cada vez mais idêntica à solução exata(as aproximações feitas pelo método de elementos finitos aproximam bem a solução u) e pode-se confirmar esta conclusão observando os graficos dos erros.

6.3 Majoração do erro

Vamos agora fazer o estudo da majoração do erro absoluto e do erro relativo da aproximação do método de elementos finitos.

Para tal recorremos a um resultado de majoração de erro introduzido e demonstrado em [4]:

Teorema 6.3.1. Se $u \in V$ for solução do problema (3.2) e $u_h \in V_h$ solução do problema (4.4) temos que

$$\|u - u_h\| \leq C_1 h_1^k \tag{6.1}$$

$$\frac{\|u - u_h\|}{\|u\|} \leq C_2 h_2^k \tag{6.2}$$

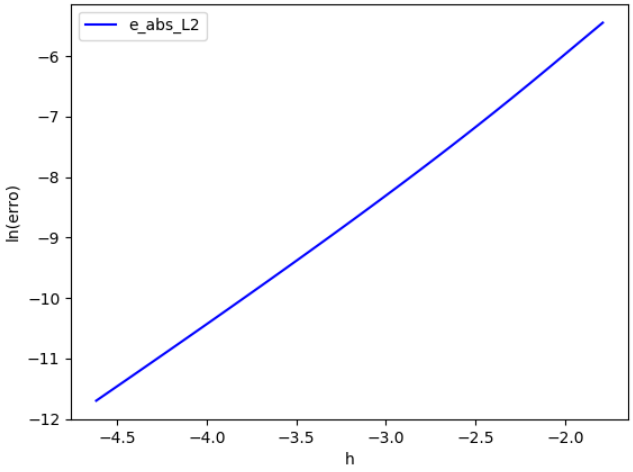
Sendo possível reconhecer que as constante C_1, C_2, k_1 e k_2 usadas aqui são independentes de h, mas dependem sim da solução e do domínio considerado. Esta dependência é detrimental quando são consideradas condições materiais incompressíveis porque aí, as constantes torna-se muito grandes e a ordem da convergência k resulta numa boa precisão, apenas com valores muito pequenos de h.

Vamos agora determinar as constantes C_1 e k_1 da proposição 6.3.1 e começamos por aplicar a função logaritmo nos dois membros:

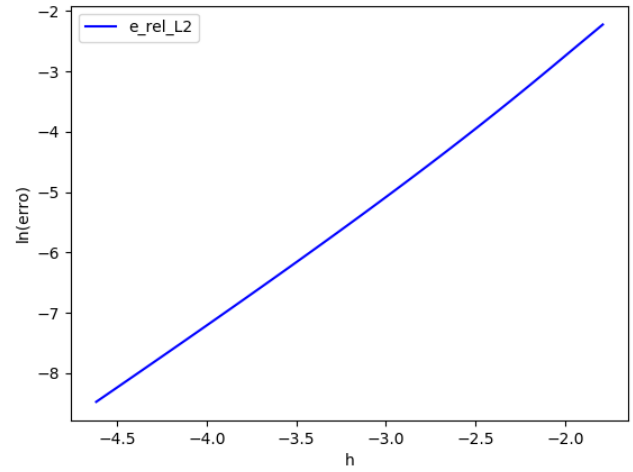
$$\begin{aligned}
&\ln(\|u - u_h\|) \leq \ln(C_1 h_1^k) \Leftrightarrow \\
&\ln(\|u - u_h\|) \leq \ln(C_1) + \ln(h_1^k) \Leftrightarrow \\
&\ln(\|u - u_h\|) \leq \ln(C_1) + k_1 \ln(h)
\end{aligned}$$

Assim, apresentamos a desigualdade da proposição 6.3.1 como uma desigualdade linear do tipo $y \leq b + mx$. Aplicando o logaritmo no erro ($\|u - u_h\|$) e calculando a regressão linear da inequação anterior é possivel obter os valores de C_1 e k_1. Para determinar as constantes C_2 e k_2 procedemos de igual modo.

Em seguida, apresentam-se os graficos das regressões lineares do erro absoluto e do erro relativo onde se apresentam os valores para as constantes da proposição 6.3.1:



(a) Regressão linear com erro absoluto ($k_1 = 2.1571$; $C_1 = 0.1673$)



(b) Regressão linear com erro relativo ($k_2 = 2.1571$; $C_2 = 4.1997$)

7. Problema de advecção-difusão

Continuamos com o caso 1D para introduzir o designado por problema de advecção difusão

$$\begin{cases} - \underbrace{ku''(x)}_{\text{termo de difusão}} + \underbrace{\alpha u'(x)}_{\text{termo de advecção}} = f(x) & \text{para } x \in]0,1[\\ u(x) = 0 & \text{para } x = 0,\ x = 1 \end{cases} \tag{7.1}$$

Primeiro iremos introduzir a noção do número de Péclet, uma medida relevante no estudo de fenómenos de transporte,neste caso, da transferência de calor.

Definição 7.0.1. Número de Péclet

O número de Péclet é definido como sendo a razão entre a taxa de advecção de uma grandeza física α, e a taxa de difusão k, da mesma grandeza. De forma simplificada vamos considerar:

$$Pe = \frac{\alpha}{k} \tag{7.2}$$

Seguindo o mesmo raciocínio do capítulo anterior, relativamente à formulação do problema e à sua discretização, vamos então fazer algumas simulações numéricas de resolução do problema (7.1) com diferentes valores para constantes de advecção e difusão α e k respetivamente, comparando então os valores das soluções aproximadas a azul com a solução exata u em vermelho utilizando o par solução e segundo membro como no Problema 6.1.

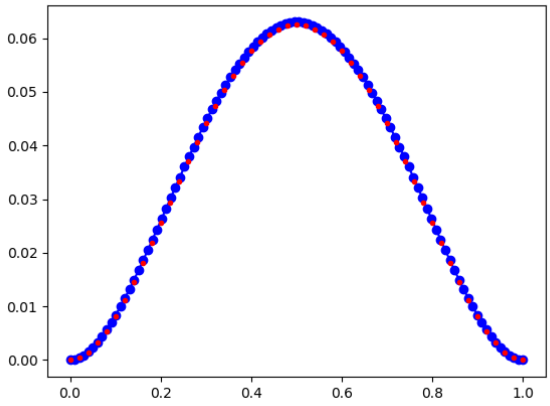


Figura 7.1: Teste A - k = 10 ; $\alpha = 0.1$; Pe = 0,01 ; h = 0,01

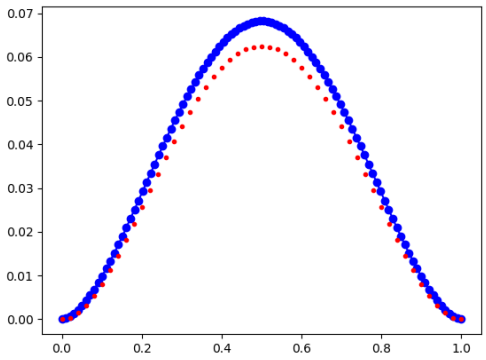


Figura 7.2: Teste B - k = 1 ; $\alpha = 1$; Pe = 1 ; h = 0.01

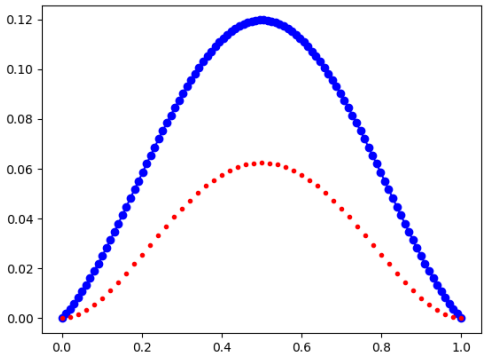


Figura 7.3: Teste C - k = 0.1 ; $\alpha = 1$; Pe = 10 ; h = 0.01

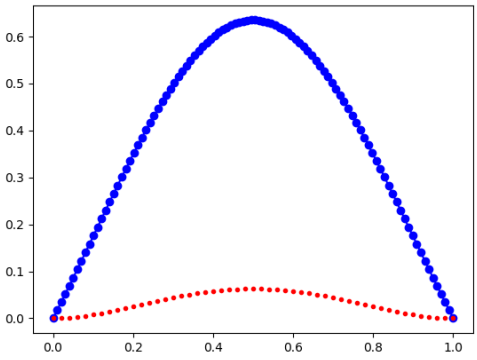


Figura 7.4: Teste D - k = 0.01 ; $\alpha = 1$; Pe = 100 ; h = 0.01

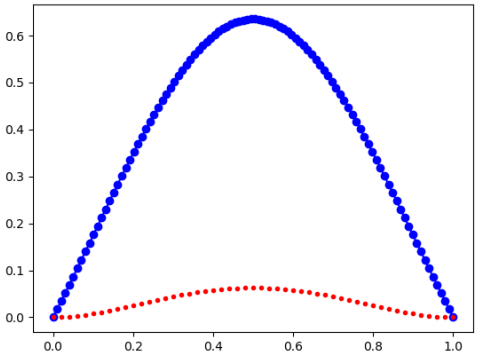


Figura 7.5: Teste E - k = 0.1 ; $\alpha = 10$; Pe = 100 ; h = 0.01

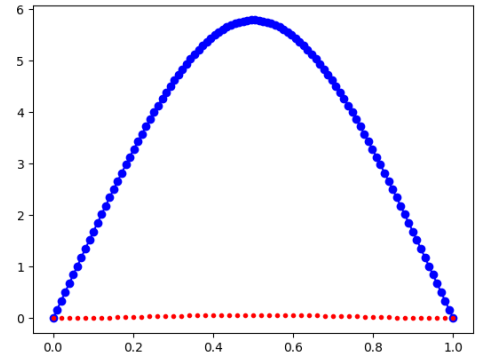


Figura 7.6: Teste F - k = 0.01 ; $\alpha = 10$; Pe = 1000 ; h = 0.01

Ao observar os gráficos dos resultados obtidos através dos vários testes feitos para diferentes constantes de advecção e difusão , α e k correspondentes, é possivel tirar as conclusões que quanto

menor for o número de Péclet, melhor serão as aproximações obtidas pelo método numérico, ou seja, o grau de precisão do método de elementos finitos aumenta consoante o crescimento da constante k ou do decréscimo da constante α. Também é possivel observar o facto que neste problema, a constante de difusão é muito dominante sobre as aproximações calculadas.

7.1 Introdução de um método de estabilização

Consideramos a descretização da formulação variacional correspondente, tal como descrita em 3.2.2 com um segundo membro nulo obtemos:

$$\frac{\alpha}{2}\Big(u_{j+1}-u_j\Big)+\frac{k}{h}\Big(-u_{j+1}+2u_j-u_{j-1}\Big)=0 \tag{7.1}$$

Em (7.1) quando k é muito pequeno ($k<h$) obtemos $u_{j+1}\simeq u_j$, $j=1,...,n$ com $u_j\in[0,1]$ os valores da solução aproximada u_h nos nós pelo que u_h é uma má aproximação de u. A melhor ideia será melhorar o método de elementos finitos implementado, adicionando desta vez um método de estabilização que permita boas aproximações sem usar um leque restrito de valores das constantes de adveção-difusão.

O método de estabilização que vai ser introduzido em junção com o método de elementos finitos será o Método "Streamline-diffusion method (SDM)"(com difusão ao longo das linhas de fluxo).

Seja v uma função de base,vamos substituir v por $(v+\frac{\alpha h}{2}v')$, a fim de estabilizar a equação de advecção-difusão com o seguinte aspecto:

$$-ku''+\alpha u'=f \tag{7.2}$$

ou seja, temos que:

$$\int_0^1(-ku''+\alpha u')\,v\;dx=\int_0^1 fv\;dx$$

Vamos então substituir a função de base pela sua versão estabilizada:

$$\int_0^1 \alpha u'\left(v+\frac{\alpha h}{2}v'\right)-ku''\left(v+\frac{\alpha h}{2}v'\right)dx=\int_0^1 f\left(v+\frac{\alpha h}{2}v'\right)dx \tag{7.3}$$

No caso de elementos finitos de Lagrange P1:

$$\int_0^1 u''v'dx=\sum_{j=1}^{n}\int_{I_j}u''v'dx=0$$

O 1º membro simplifica:

$$\alpha\int_0^1 u'vdx+\left(\frac{\alpha^2h}{2}+k\right)\int_0^1 u'v'dx-\frac{\alpha h}{2}\times 0 \tag{7.4}$$

o que corresponde à discretização:

$$\frac{\alpha}{2}\Big(u_{j+1}-u_j\Big)+\alpha^2\left(-\frac{1}{2}u_{j+1}+u_j-\frac{1}{2}u_{j-1}\right)+\frac{k}{h}\Big(-u_{j+1}+2u_j-u_{j-1}\Big)=$$

$$= \left(\frac{\alpha}{2} - \frac{\alpha^2}{2} - \frac{k}{h}\right) u_{j+1} + \left(\frac{2k}{h} + \alpha^2 - \frac{\alpha}{2}\right) u_j - \left(\frac{\alpha^2}{2} + \frac{k}{h}\right) u_{j-1}$$

O segundo membro simplifica:

$$\int_0^1 f\left(v + \frac{\alpha h}{2} v'\right) dx = \int_0^1 fv\, dx + \frac{\alpha h}{2} \int_0^1 fv' dx$$

o que corresponde à forma discreta:

$$\int_0^1 f\varphi_j\, dx + \frac{\alpha h}{2} \int_0^1 f\varphi_j'\, dx = \int_{x_{j-1}}^{x_{j+1}} f\varphi_j\, dx + \frac{\alpha h}{2} \int_{x_{j-1}}^{x_{j+1}} f\varphi_j'\, dx =$$

$$= \int_{x_{j-1}}^{x_j} f\hat{\varphi}_l\, dx + \int_{x_j}^{x_{j+1}} f\hat{\varphi}_r\, dx + \frac{\alpha h}{2} \int_{x_{j-1}}^{x_j} f\hat{\varphi}_l'\, dx + \frac{\alpha h}{2} \int_{x_j}^{x_{j+1}} f\hat{\varphi}_r'\, dx \tag{7.5}$$

para $0 < j < n$.

Temos em (4.8) a forma algébrica do método de elementos finitos:

$$-\frac{k}{h} u_{j-1} + \frac{2k}{h} u_j - \frac{k}{h} u_{j+1} = b_j \tag{7.6}$$

Vamos então comparar o método dos elementos finitos , com a sua adição do método de estabilização "Streamline-diffusion method (SDM)":

$$\left(\frac{\alpha}{2} - \frac{\alpha^2}{2} - \frac{k}{h}\right) u_{j+1} + \left(\frac{2k}{h} + \alpha^2 - \frac{\alpha}{2}\right) u_j - \left(\frac{\alpha^2}{2} + \frac{k}{h}\right) u_{j-1}$$

Vamos então separar a matriz A(à esquerda) da matriz de estabilização que iremos denominar por matriz S(à direita):

$$-\tfrac{k}{h} u_{j-1} + \tfrac{2k}{h} u_j - \tfrac{k}{h} u_{j+1} \quad + \quad \tfrac{\alpha}{2}\left[(1-\alpha) u_{j+1} + (2\alpha - 1) u_j - \alpha u_{j-1}\right]$$

Tendo assim o seguinte esquema matricial:

$$[A + S]\,[u_j] = [b + F]$$

$$A = \frac{k}{h} \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \tag{7.7}$$

$$S = \frac{\alpha}{2} \begin{bmatrix} 2\alpha - 1 & 1 - \alpha & 0 & \cdots & 0 \\ -\alpha & 2\alpha - 1 & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & 2\alpha - 1 & 1 - \alpha \\ 0 & \cdots & 0 & -\alpha & 2\alpha - 1 \end{bmatrix} \tag{7.8}$$

b como em (4.9) e F_j obtido por (7.5).

$[A + S]$ não é uma matriz simétrica como estávamos à espera e portanto temos de utilizar outro método, diferente do método dos métodos de resolução utilizados anteriormente.

7.1.1 Cálculos da formulação fraca estabilizada de um novo problema

Com vista a comprovar que de facto a estabilização ajuda o método de elementos finitos a ter melhores aproximações da solução exata, vamos considerar um novo problema, desta vez um problema de tipo camada limite.
Ao alterar as constantes de advecção e difusão, as soluções deste tipo de problema sofrem uma grande alteração, e sem haver a estabilização do método, as aproximações da solução vão ter um nivel baixo de precisão. Seguimos com a introdução do problema (7.1) considerando nulo o segundo membro:

Temos o objectivo de estabilizar a equação de advecção-difusão com o seguinte aspecto:

$$\begin{cases} -ku'' + \alpha u' = 0 & 0 < x < 1 \\ u(0) = 1\,,\, u(1) = 0 \end{cases} \tag{7.9}$$

com solução exata

$$u(x) = C(e^{\frac{1}{k}} - e^{\frac{x}{k}}) \qquad , \qquad C = \frac{1}{e^{\frac{1}{k}} - 1} \tag{7.10}$$

Consideramos a discretização da formulação fraca estabilizada como deduzida anteriormente:

$$\sum_j \left(k + \frac{\alpha^2 h}{2}\right) \int_{x_j}^{x_{j+1}} \varphi_j' \varphi_i' \, dx + \alpha \int_{x_j}^{x_{j+1}} \varphi_j' \varphi_i \, dx = 0$$

No caso concreto da função escolhida: $u = C\left(e^{\frac{1}{k}} - e^{\frac{x}{k}}\right)$, com $C = \frac{1}{e^{\frac{1}{k}}-1}$, a solução só é exata se $\alpha = 1$,então iremos substituir outra vez α por 1.

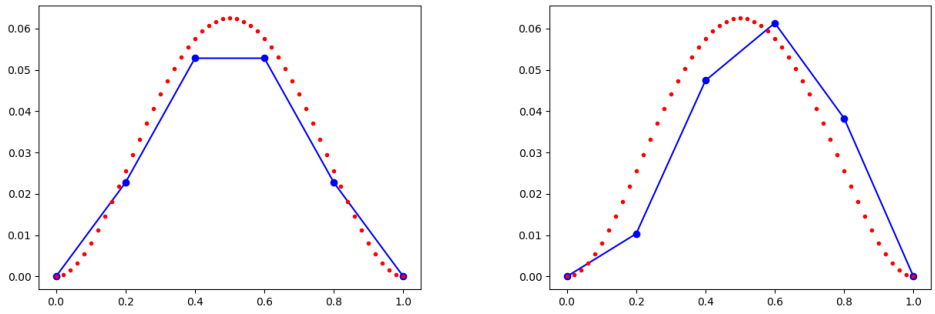
$$\sum_j \left(k + \frac{h}{2}\right) \int_{x_j}^{x_{j+1}} \varphi_j' \varphi_i' \, dx + \int_{x_j}^{x_{j+1}} \varphi_j' \varphi_i \, dx = 0$$

7.2 Comparação de resultados estabilizado vs não estabilizado

Com o objétivo de estudar o nível de precisão que o método estabilizado tem sobre o método não estabilizado, foram feitos vários testes para diferentes nós(n+1 = 6 , 11 , 21) e para diferentes valores da constante de difusão k(k = 1 , 0.1 , 0.01) e onde também se apresenta o e_r(erro relativo) para o problema 6.1 e para o novo problema 7.10 .De notar que a constate de advecção $\alpha = 1$, devido à restrição do problema de camada limite.

7.2.1 1º Problema

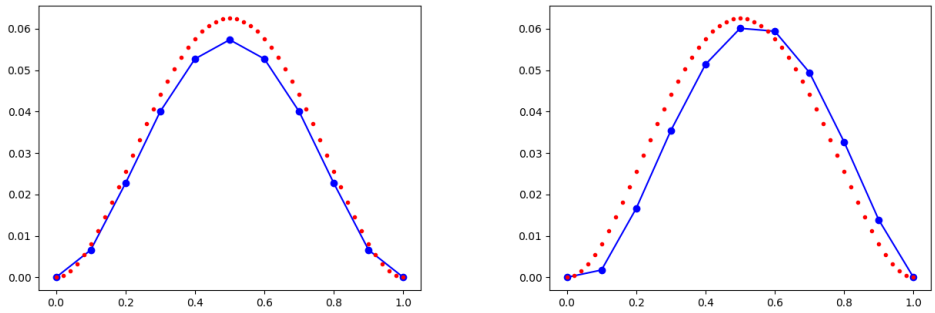
O seguinte problema de adveção-difusão $-ku'' + \alpha u' = f$ com solução exata $u = (x-1)^2 x^2$.



(a) Método elementos finitos não estabilizado com $e_r = 0{,}1336$

(b) Método elementos finitos estabilizado com $e_r = 0{,}2427$

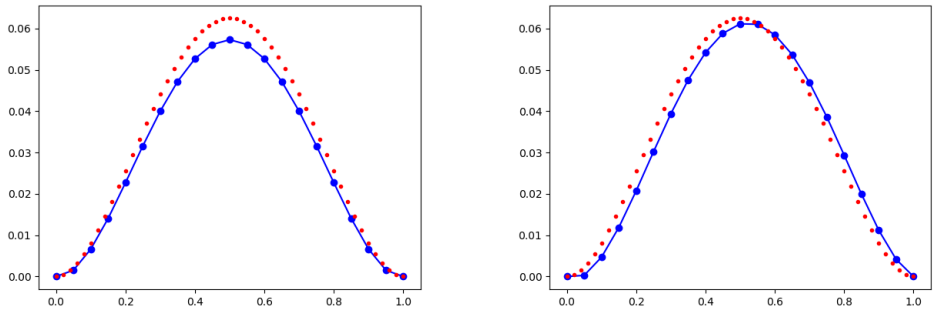
Figura 7.7: Método de elementos finitos com $n = 5$, $\alpha = 1$ e $k = 1$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}0993$

(b) Método elementos finitos estabilizado com $e_r = 0{,}1486$

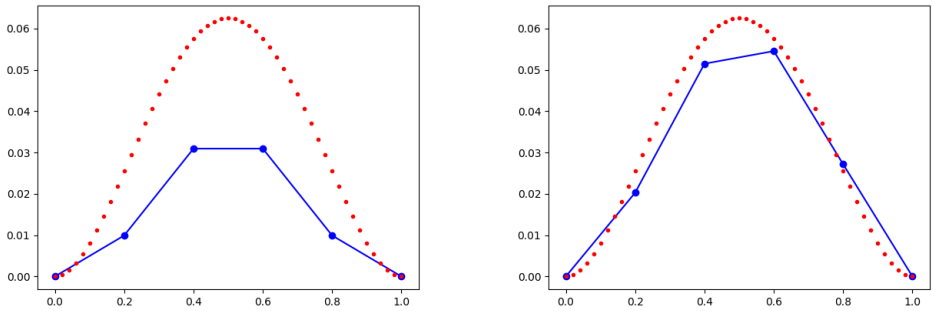
Figura 7.8: Método de elementos finitos com $n = 10$, $\alpha = 1$ e $k = 1$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}0928$

(b) Método elementos finitos estabilizado com $e_r = 0{,}0812$

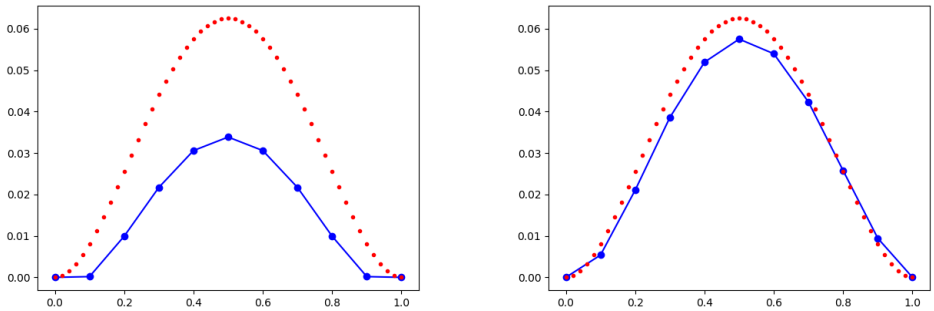
Figura 7.9: Método de elementos finitos com $n = 20$, $\alpha = 1$ e $k = 1$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}5093$

(b) Método elementos finitos estabilizado com $e_r = 0{,}138$

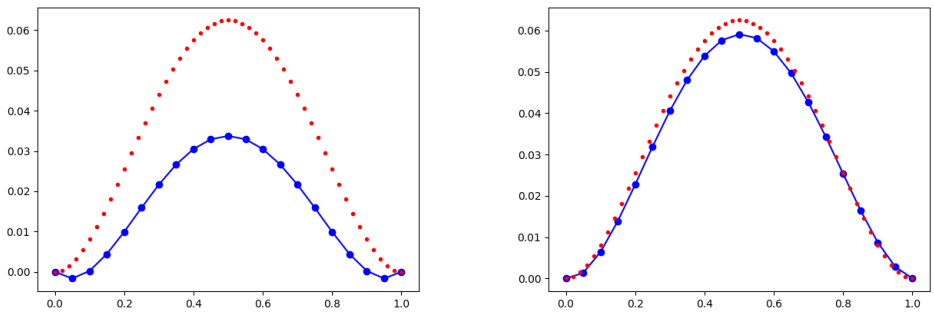
Figura 7.10: Método de elementos finitos com $n = 5$, $\alpha = 1$ e $k = 0.1$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}5002$

(b) Método elementos finitos estabilizado com $e_r = 0{,}1007$

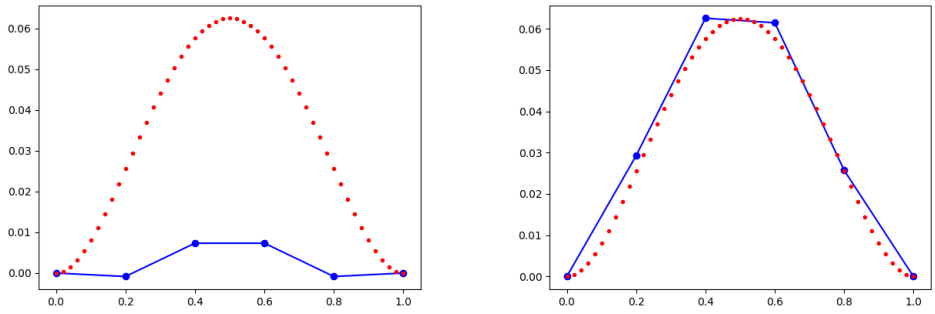
Figura 7.11: Método de elementos finitos com $n = 10$, $\alpha = 1$ e $k = 0.1$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}4983$

(b) Método elementos finitos estabilizado com $e_r = 0{,}0631$

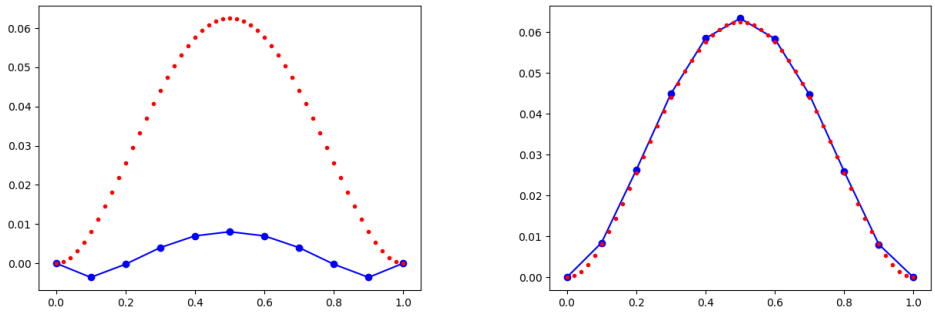
Figura 7.12: Método de elementos finitos com $n = 20$, $\alpha = 1$ e $k = 0.1$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}9055$

(b) Método elementos finitos estabilizado com $e_r = 0{,}0813$

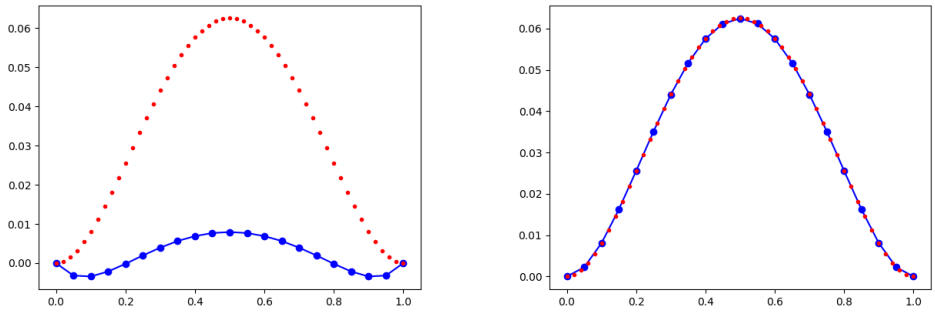
Figura 7.13: Método de elementos finitos com $n = 5$, $\alpha = 1$ e $k = 0.01$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}9035$

(b) Método elementos finitos estabilizado com $e_r = 0{,}0192$

Figura 7.14: Método de elementos finitos com $n = 10$, $\alpha = 1$ e $k = 0.01$



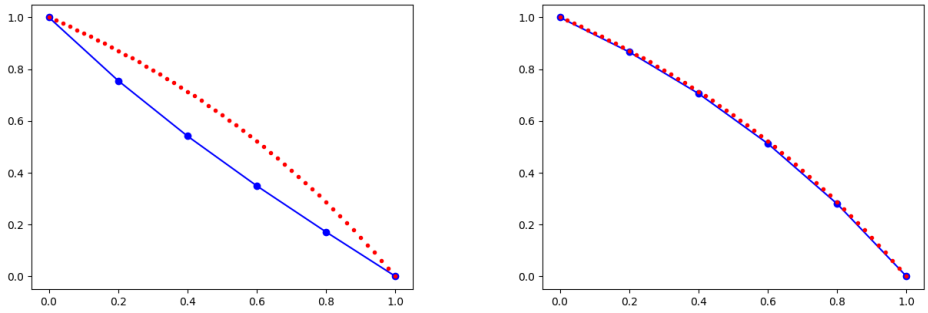
(a) Método elementos finitos não estabilizado com $e_r = 0{,}9032$

(b) Método elementos finitos estabilizado com $e_r = 0{,}0058$

Figura 7.15: Método de elementos finitos com $n = 20$, $\alpha = 1$ e $k = 0.01$

7.2.2 2º Problema

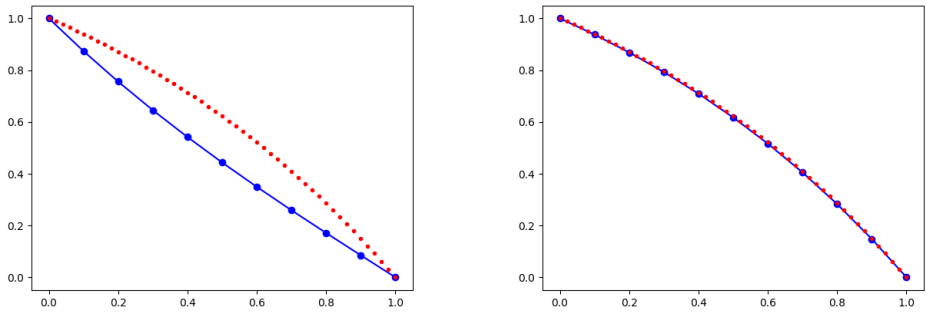
O seguinte problema de adveção-difusão estacionário $-ku'' + \alpha u' = 0$ com solução exata $u(x) = C(e^{\frac{1}{k}} - e^{\frac{x}{k}})$, $C = \dfrac{1}{e^{\frac{1}{k}} - 1}$.



(a) Método elementos finitos não estabilizado com $e_r = 0{,}2002$

(b) Método elementos finitos estabilizado com $e_r = 0{,}0148$

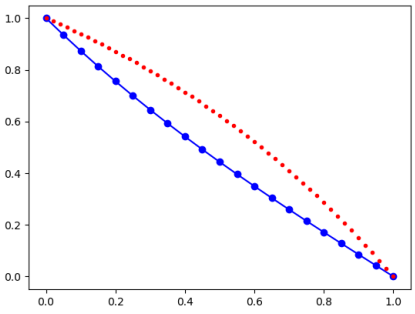
Figura 7.16: Método de elementos finitos com $n = 5$, $\alpha = 1$ e $k = 1$



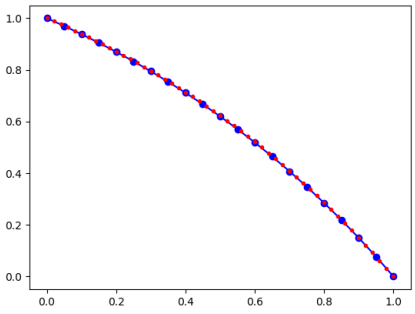
(a) Método elementos finitos não estabilizado com $e_r = 0{,}2017$

(b) Método elementos finitos estabilizado com $e_r = 0{,}0069$

Figura 7.17: Método de elementos finitos com $n = 10$, $\alpha = 1$ e $k = 1$

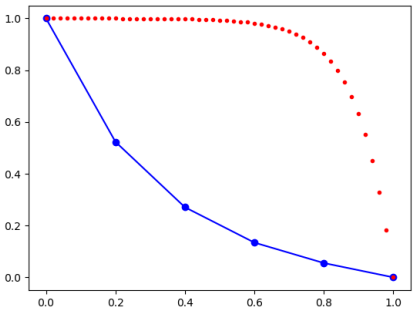


(a) Método elementos finitos não estabilizado com $e_r = 0{,}2021$

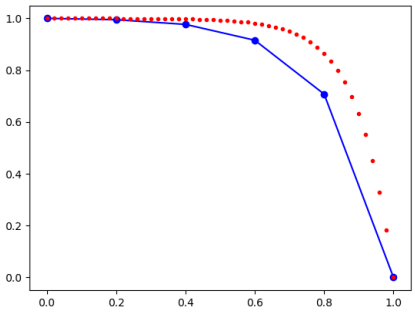


(b) Método elementos finitos estabilizado com $e_r = 0{,}0034$

Figura 7.18: Método de elementos finitos com $n = 20$, $\alpha = 1$ e $k = 1$

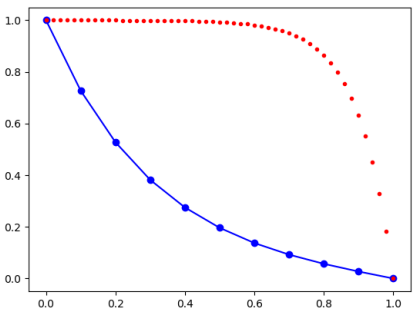


(a) Método elementos finitos não estabilizado com $e_r = 0{,}7076$

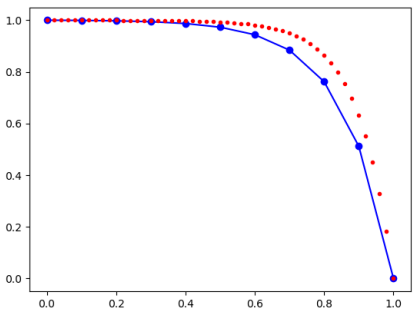


(b) Método elementos finitos estabilizado com $e_r = 0{,}1286$

Figura 7.19: Método de elementos finitos com $n = 5$, $\alpha = 1$ e $k = 0.1$

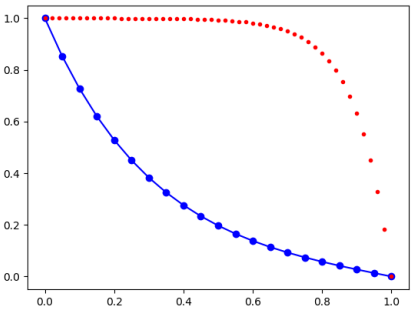


(a) Método elementos finitos não estabilizado com $e_r = 0{,}7103$

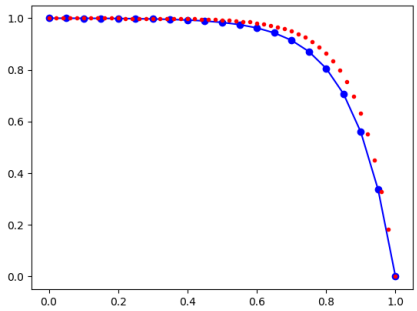


(b) Método elementos finitos estabilizado com $e_r = 0{,}0719$

Figura 7.20: Método de elementos finitos com $n = 10$, $\alpha = 1$ e $k = 0.1$

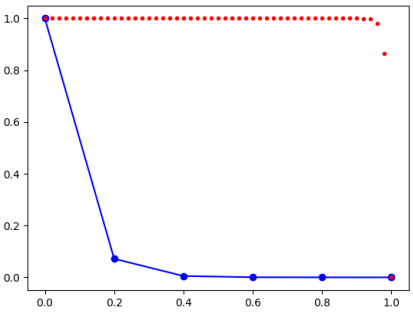


(a) Método elementos finitos não estabilizado com $e_r = 0{,}711$

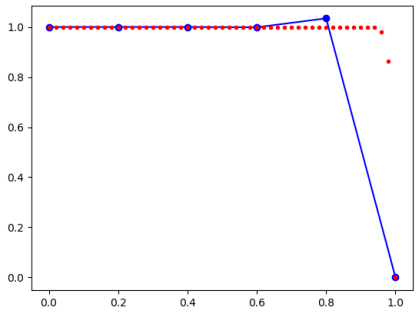


(b) Método elementos finitos estabilizado com $e_r = 0{,}0388$

Figura 7.21: Método de elementos finitos com $n = 20$, $\alpha = 1$ e $k = 0.1$

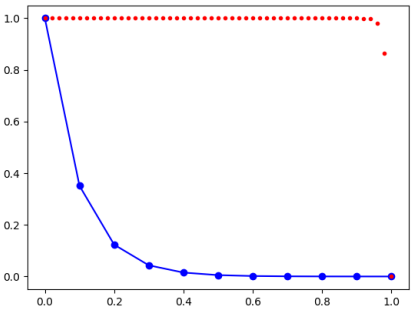


(a) Método elementos finitos não estabilizado com $e_r = 0{,}9162$

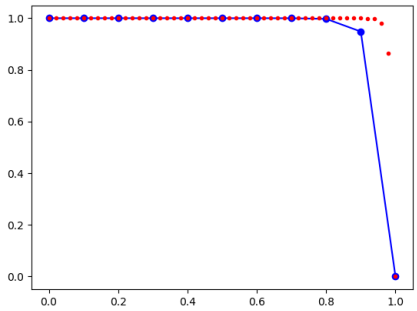


(b) Método elementos finitos estabilizado com $e_r = 0{,}2355$

Figura 7.22: Método de elementos finitos com $n = 5$, $\alpha = 1$ e $k = 0.01$

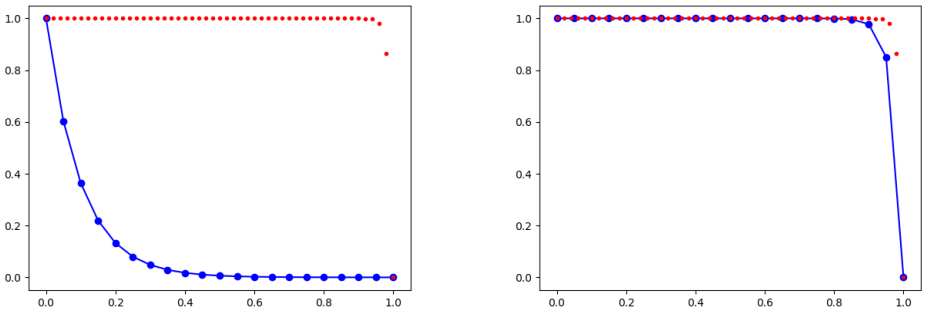


(a) Método elementos finitos não estabilizado com $e_r = 0{,}9195$



(b) Método elementos finitos estabilizado com $e_r = 0{,}1487$

Figura 7.23: Método de elementos finitos com $n = 10$, $\alpha = 1$ e $k = 0.01$



(a) Método elementos finitos não estabilizado com $e_r = 0{,}9204$

(b) Método elementos finitos estabilizado com $e_r = 0{,}091$

Figura 7.24: Método de elementos finitos com $n = 20$, $\alpha = 1$ e $k = 0.01$

8. Análise do problema de difusão bidimensional

Consideremos agora Ω, um conjunto aberto e limitado: $\Omega =]0,1[\times]0,1[$ de fronteira $\partial\Omega = \Gamma = \{0,1\} \times [0,1] \bigcup [0,1] \times \{0,1\}$, tal como representado na Figura 8.1.

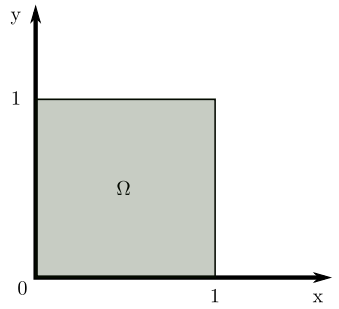


Figura 8.1: Representação da região Ω

Consideramos o seguinte problema de difusão, com vista a resolução numérica pelo método de elementos finitos:

$$\begin{cases} -k\Delta u &= f \quad \text{em } \Omega \\ u &= 0 \quad \text{sobre } \Gamma \end{cases} \tag{8.1}$$

Onde Δu representa o Laplaciano da função escalar u.

Para avaliarmos o erro cometido na resolução numérica vamos definir uma solução exata do problema 8.1 para um determinado segundo membro f:

Consideramos a seguinte função que verifica as condições de fronteira:

$$u(x,y) = (x-1)x(y-1)y \tag{8.2}$$

Vamos calcular o Laplaciano desta função para obter a função do segundo membro f:

$\frac{\partial u}{\partial x} = (x-1)(y-1)y + x(y-1)y = (2x-1)(y-1)y$

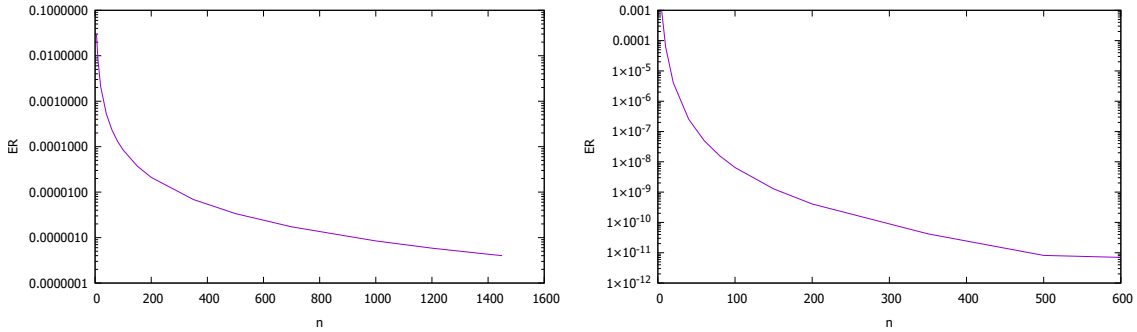
$\frac{\partial u}{\partial y} = (y-1)(x-1)x + y(x-1)x = (2y-1)(x-1)x$

$\frac{\partial^2 u}{\partial x^2} = 2(y-1)y$

$\frac{\partial^2 u}{\partial y^2} = 2(x-1)x$

$$f(x,y) = -\triangle u = -2\Big[(y-1)y + (x-1)x\Big] \tag{8.3}$$

Assim a função definida em 8.2 é solução do problema 8.1 quando tomamos a função f definida em (8.3).
Com o problema agora completo, utilizamos o software FreeFem++ [8] para calcular a solução exata e a solução aproximada, utilizando o método de elementos finitos. Depois de calculadas, utilizamos o software Gnuplot [9] para esboçar os gráficos do estudo dos erros entre a solução exata e a solução aproximada:



(a) Erro relativo com P_1

(b) Erro relativo com P_2

Figura 8.2: Variação logarítmica da interpolação linear do erro relativo em função do número de nós (graus de liberdade) para polinómios P_1 (a) e P_2 (b)

Aproveitamos também para colocar aqui dois gráficos adicionais, a figura 8.3 , onde estão representados os isovalores da função de calor com base na sua posição espacial (x, y) e a figura (8.4), onde está representado o gradiente da função de calor, com os vetores a partirem do centro(zona mais quente) para as zonas exteriores (percurso do calor vai da zona mais quente para a mais fria).

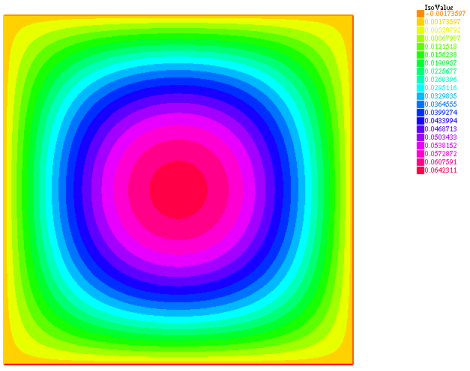


Figura 8.3: Função com os respetivos valores de calor por posição

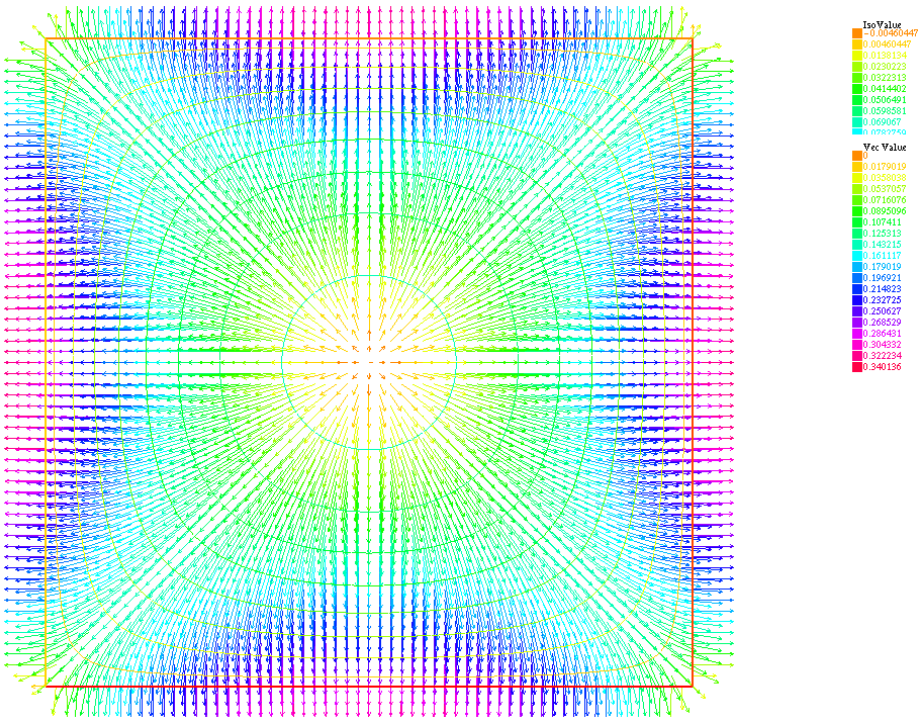


Figura 8.4: Função com o gradiente da função de calor

9. Análise do problema de camada limite bidimensional

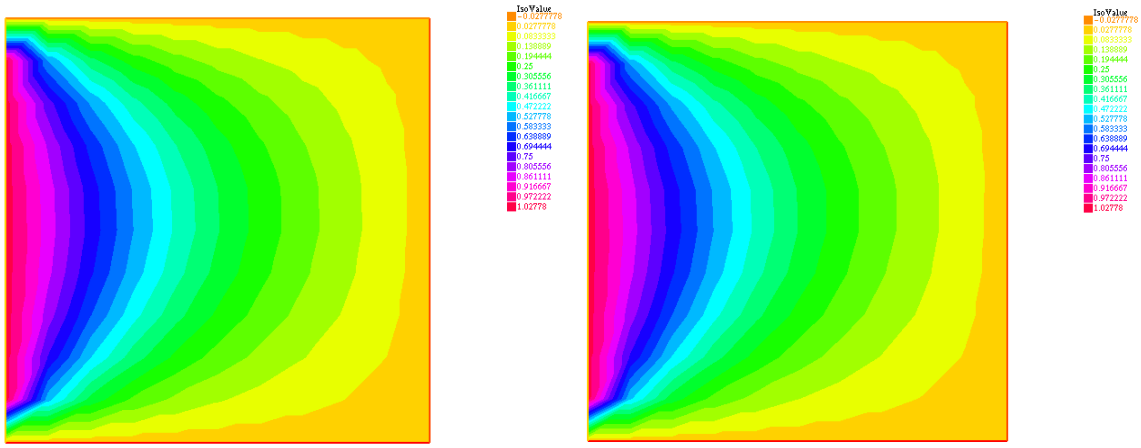
Após fazer os cálculos para o problema de camada limite unidimensional 7.10, iremos agora fazer os mesmos calculos utilizando o método de elementos finitos estabilizados para um problema de camada limite bidimensional.

Temos o objectivo de estabilizar a equação de advecção-difusão com o seguinte aspecto:

$$\begin{cases} -k\Delta u + \vec{a} \cdot \nabla u = 0 \qquad 0 < x < 1 \ \wedge \ 0 < y < 1 \\ u(0,y) = 1 \qquad 0 \leq y \leq 1 \\ u(1,y) = u(x,1) = u(x,0) = 0 \qquad 0 \leq x \leq 1 \ \vee \ 0 \leq y \leq 1 \end{cases} \tag{9.1}$$

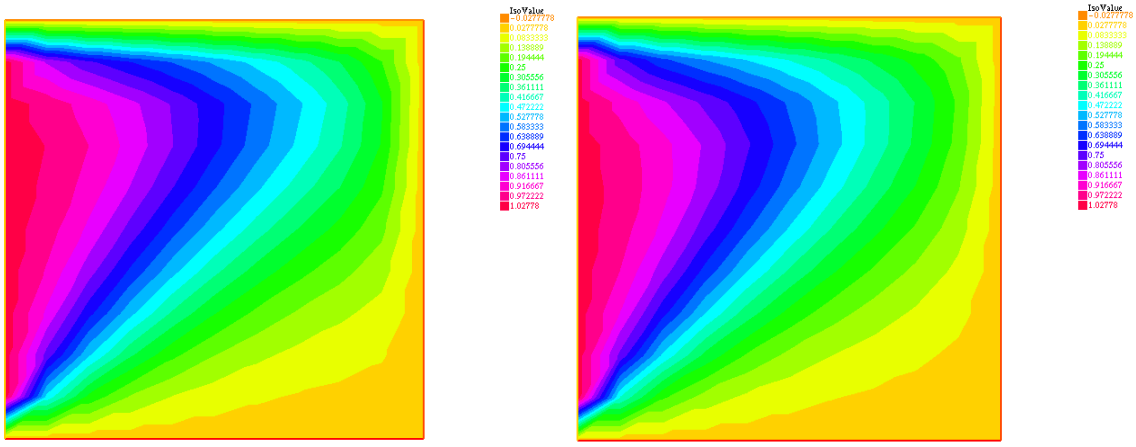
com $\vec{a} = (a_1, a_2)$.

De seguida utilizamos o software FreeFEM++ para calcular as aproximações da solução do problema (9.1) para constantes de adveção e difusão de valor $\vec{a} = (1,1)$ e $\vec{a} = (1,0)$, com $\alpha = \|\vec{a}\|$, e $k = 1 \vee k = 0.1 \vee k = 0.01$ para um número de nós $(n+1 = 21)$.



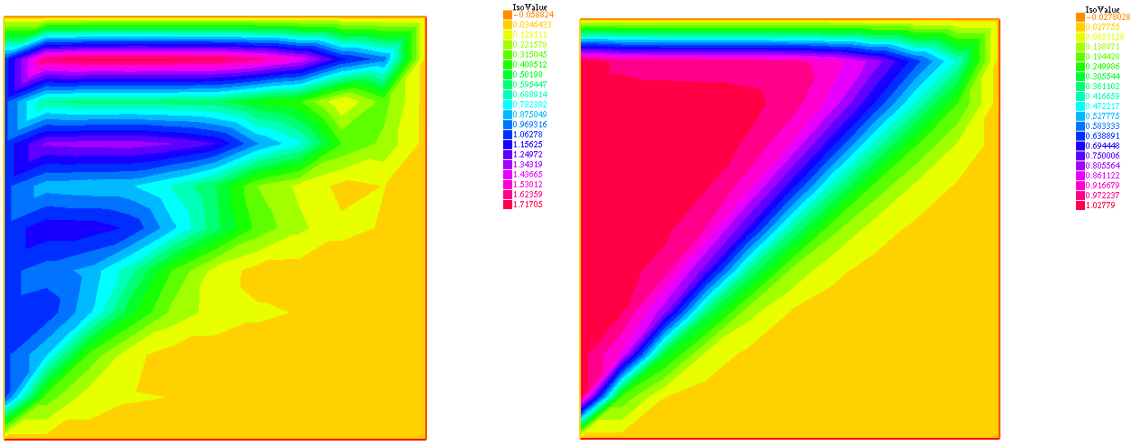
(a) Teste não estabilizado (b) Teste estabilizado

Figura 9.1: Teste A - k = 1 ; $\vec{a} = (1, 1)$; $n = 10$



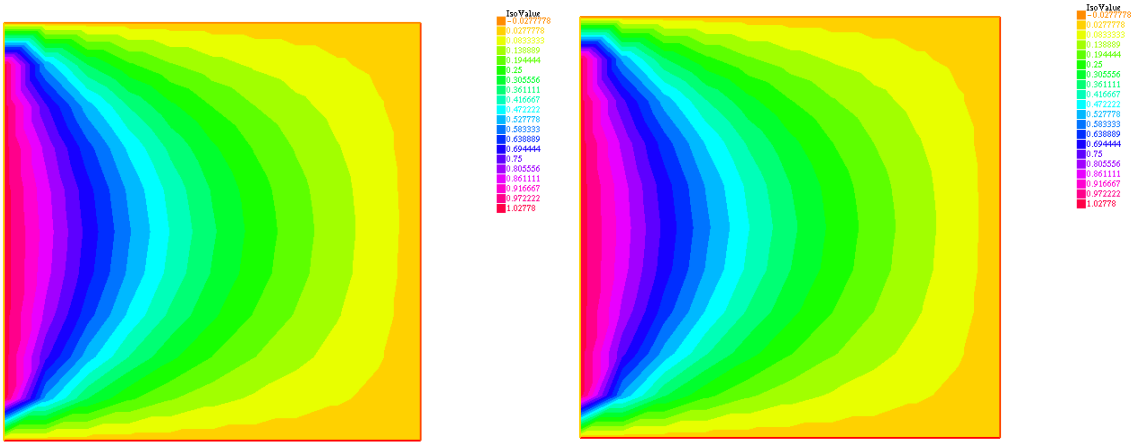
(a) Teste não estabilizado (b) Teste estabilizado

Figura 9.2: Teste B - k = 0.1 ; $\vec{a} = (1, 1)$; $n = 10$



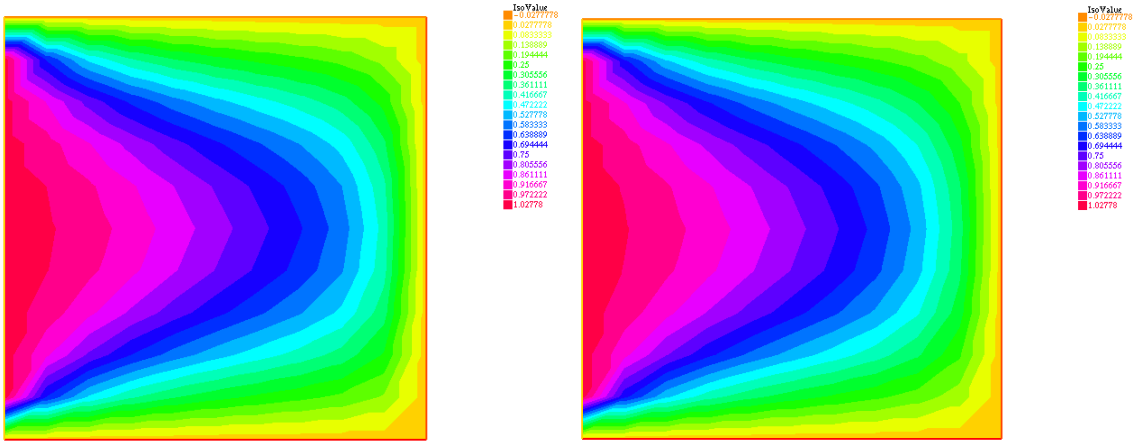
(a) Teste não estabilizado (b) Teste estabilizado

Figura 9.3: Teste C - k = 0.01 ; $\vec{a} = (1, 1)$; $n = 10$



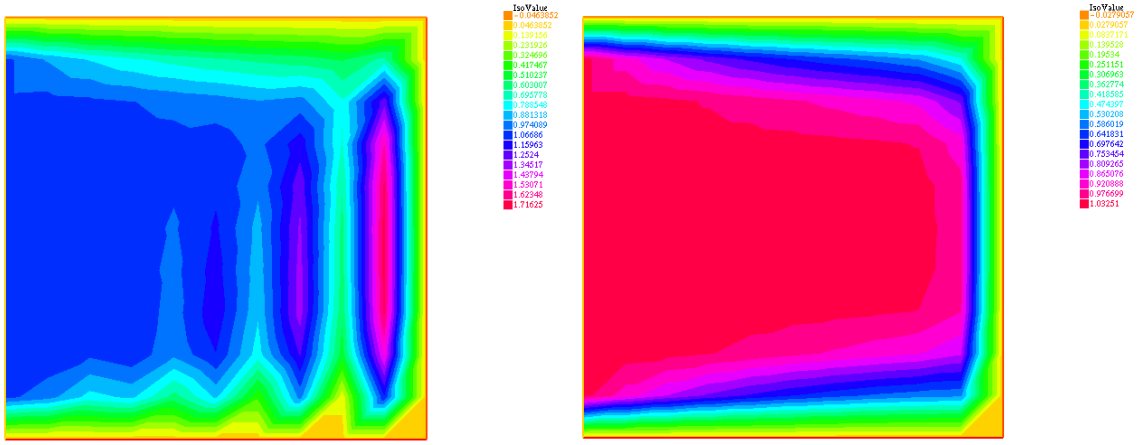
(a) Teste não estabilizado (b) Teste estabilizado

Figura 9.4: Teste D - k = 1 ; $\vec{a} = (1, 0)$; $n = 10$



(a) Teste não estabilizado (b) Teste estabilizado

Figura 9.5: Teste E - k = 0.1 ; $\vec{a} = (1, 0)$; $n = 10$



(a) Teste não estabilizado (b) Teste estabilizado

Figura 9.6: Teste F - k = 0.01 ; $\vec{a} = (1, 0)$; $n = 10$

Ao observar os resultados obtidos, é possível verificar que ao alterar a constante da difusão k, a solução muda de valores radicalmente, o que já se tinha observado anteriormente e que reforça a inclusão de um método de estabilização integrado no método de elementos finitos, para que exista um bom nível de aproximação para diferentes tipos de funções de advecção-difusão e também para diferentes valores das constantes de advecção e difusão.

10. Conclusão

Com este trabalho apresentei uma introdução ao Método dos Elementos Finitos, para a resolução de equações com derivada parciais. Utilizei conhecimentos teóricos adquiridos na licenciatura para ilustrar a sua aplicabilidade num contexto mais prático de aplicação à vida real. A introdução de novas formulações para os problemas evidenciou a necessidade da generalização dos conceitos adquiridos. Com os casos apresentados, obtive resultados numéricos ilustrativos da potencialidade do método de elementos finitos e da sua fiabilidade na aproximação de problemas com forte difusão. No entanto quando a constante de difusão k é pequena relativamente ao parâmetro de discretização h, ou relativamente à constante de advecção α, no caso dos problema de advecção-difusão com número de Peclet elevado, é necessário implementar técnicas de estabilização adicionais. O que torna bastante importante o estudo dos métodos de estabilização aplicados ao método de elementos finitos, obtendo-se boas aproximações com erros muito pequenos para diferentes soluções de problemas de advecção-difusão.

Bibliografia

[1] A. Hrennikoff. Solution of Problems of Elasticity by the Framework Method. *Journal of Applied Mechanics*, 1941.

[2] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Amer. Math. Soc.*, 1943.

[3] G. Strang and G.J. Fix. *An analysis of the finite element method*. Wellesley-Cambridge Press, 1988.

[4] C. Johnson. *Numerical Solution of Partial Differential Equations by the Finite Element Method*. Dover Books on Mathematics Series. Dover Publications, Incorporated, 2012.

[5] J. Galvis and H. Versieux. *Introdução à aproximação numérica de equações diferenciais parciais via o método de elementos finitos*. Publicações matemáticas. IMPA, 2011.

[6] B.N. Datta. *Numerical Linear Algebra and Applications, Second Edition*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2010.

[7] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[8] F. Hecht. New development in freefem++. *J. Numer. Math.*, 20(3-4):251–265, 2012.

[9] Thomas Williams, Colin Kelley, and many others. Gnuplot 5.2.7: an interactive plotting program. `http://gnuplot.sourceforge.net/`, May 2019.

[10] K.J. Bathe. *Finite Element Procedures*. Prentice Hall, 2006.

[11] Vincent Manet. Méthode des Éléments Finis: vulgarisation des aspects mathématiques et illustrationde la méthode. Lecture, December 2018.

A. Quadratura de Gauss-Legendre

A quadratura de Gauss-Legendre é um método de integração numérico. A quadratura de Gauss sobre um intervalo $[-1, 1]$ com uma função de pesos $W(x) = 1$. Os pontos de abcissa para uma quadratura de ordem n são dados pelas raizes dos polinomiais de Legendre $P_n(x)$, que ocorrem simétricamente sobre 0. Os pesos são dados por:

$$w_i = \frac{A_{n+1}\gamma_n}{A_n P'_n(x_i)P_{n+1}(x_i)} \tag{A.1}$$

$$= \frac{A_n}{A_{n-1}} \frac{\gamma_{n-1}}{P_{n-1}(x_i)P'_n(x_i)} \tag{A.2}$$

onde A_n é o coeficiente de x^n em $P_n(x)$. Para polinômios Legendre:

$$A_n = \frac{(2n)!}{2^n(n!)^2} \tag{A.3}$$

utilizando a teoria de Hildebrand 1956, temos que:

$$\frac{A_{n+1}}{A_n} = \frac{2n+1}{n+1} \tag{A.4}$$

Adicionalmente sabemos que:

$$\gamma_n = \int_{-1}^{1} [P_n(x)]^2 dx = \frac{2}{2n+1} \tag{A.5}$$

E portanto podemos concluir que:

$$w_i = \frac{2}{nP_{n-1}(x_i)P'_n(x_i)} \tag{A.6}$$

Usando a relação de recorrência:

$$(1-x^2)P'_n(x) = -nxP_n(x) + nP_{n-1}(x) = (n+1)xP_n(x) - (n+1)P_{n+1}(x) \tag{A.7}$$

Corrigindo o valor inicial de Hildebrand 1956:

$$w_i = \frac{2}{(1-x_i^2)[P'_n(x_i)]^2} \tag{A.8}$$

Os pesos w_i satisfazem:

$$\sum_{n=1}^{n} w_i = 2 \tag{A.9}$$

Como iremos usar 3 pontos, vamos ter os seguintes pesos e abcissas:

$$w_1 = \frac{5}{9} \quad \Leftrightarrow \quad x_1 = -\sqrt{\frac{3}{5}} \tag{A.10}$$

$$w_2 = \frac{8}{9} \quad \Leftrightarrow \quad x_2 = 0 \tag{A.11}$$

$$w_3 = \frac{5}{9} \quad \Leftrightarrow \quad x_3 = \sqrt{\frac{3}{5}} \tag{A.12}$$

B. Teoremas fundamentais da teoria do método de elementos finitos

2.1 Alguma definições prévias

Introduzem-se de seguida algumas noções e resultados fundamentais que se destacam no estudo do étodo dos Elementos Finitos [4]: Considera-se Ω um conjunto aberto em $\mathbb{R}^N, N \geq 1$

Definição 2.1.1. Espaço $L^2(\Omega)$

$$L^2(\Omega) = \left\{ v : \Omega \to \mathbb{R} : \int_\Omega v^2 \, d\Omega < +\infty \right\}$$

Definição 2.1.2. Em $L^2(\Omega)$ define-se o produto interno

$$(u, v) = \int_\Omega uv \, d\Omega$$

e a norma

$$\|u\| = \sqrt{(u, u)}$$

Definição 2.1.3. Forma bilinear contínua
Uma forma bilinear $a : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ diz-se contínua se

$$\exists c > 0, (u, v) \in \mathcal{H}^2, |a(u, v)| \leq c\|u\|_{\mathcal{H}}\|v\|_{\mathcal{H}}.$$

Definição 2.1.4. Forma bilinear coerciva
Uma forma bilinear $a(\cdot, \cdot)$ diz-se coerciva em $\mathcal{H}$ se

$$\exists \alpha > 0, \forall u \in \mathcal{H}, \quad a(u, u) \geq \alpha\|u\|^2_{\mathcal{H}}.$$

Vamos introduzir aqui uma ferramenta importante para garantir a existência e unicidade de soluções para a formulação variacional de problemas de contorno do tipo elíptico.

Teorema 2.1.1. Teorema de Lax-Milgram Sejam $\mathcal{H}$ um espaço de Hilbert , $a(\cdot,\cdot)$ uma forma bilinear contínua e coerciva, $L(\cdot)$ uma forma linear continua em $\mathcal{H}$, então existe um único u de $\mathcal{H}$ tal que a equação $a(u,v) = L(v)$ se verifica para todo o v de $\mathcal{H}$, ou seja:

$\exists^1 u \in \mathcal{H}: \quad a(u,v) = L(v), \quad \forall v \in \mathcal{H}$

Se a forma bilinear a for simétrica , então u é o único elemento de $\mathcal{H}$ que minimiza o funcional $J :\to \mathbb{R}$ definido por

$$J(v) = \frac{1}{2}a(v,v) - L(v)$$

para todo o v de $\mathcal{H}$, ou seja :

$\exists^1 u \in \mathcal{H}, J(u) = \min J(v)$, para $v \in \mathcal{H}$.

A demonstração deste teorema pode ser vista, por exemplo, em [4].

C. Código Python

3.1 1º Problema não Estabilizado

Apresenta-se scriptum em código Python para a resolução de um problema de difusão em dimensão 1.

```
def sol_e ( x ):

#*********************************************************************************80
#
## SOLUCAO EXATA.
#
 value = (x-1)*(x-1)*x*x


 return value


def rhs_f ( k,x,alpha ):

#*********************************************************************************80
#
## SEGUNDO MEMBRO.
#
 value = -k*(12*x*x-12*x+2)


 return value



#*********************************************************************************80
#    Para 0 < x < 1:
#      -k u'' + a u' = f(x)
#    condicoes de fronteira
#      u(0) = u(1) = 0.
#
```

```
f = open("e_rel_L2_SDM.txt",'a+')
fa = open("e_abs_L2_SDM.txt",'a+')
fc = open("numCond_SDM.txt",'a+')
fb = open("tempoResol_SDM.txt",'a+')

#
a = 0.0;   k= 0.1; alpha=1;
b = 1.0

#for n in range(5,100): # número de elementos =  número de nós -1 =
 # número de graus de liberdade -1 = número de funções de base -1

n=10

x = np.linspace ( a, b, n + 1 )
#
#  3 pontos  quadratura de Gauss [-1,1].
#
ng = 3

xg = np.array ( ( -np.sqrt(3./5.),  0.   , np.sqrt(3./5.) ) )

wg = np.array ( (       5./9.,  8./9.,      5./9. ) )
#
#  Calculo de A e RHS.
#
A = np.zeros ( ( n + 1, n + 1 ) )
rhs = np.zeros ( n + 1 )
#
#  Ciclo sobre os elementos E: (0, 1, 2, ..., N-1).
#
for e in range ( 0, n ):
  l = e
  r = e + 1

  xl = x[l]
  xr = x[r]
#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
#
  for q in range ( 0, ng ):
#
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
```

```
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
      phil  = ( xr - xq  ) / ( xr - xl )
      dphil =  -1. / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )
      dphir = 1. / ( xr - xl )
#
#
      A[l][l] = A[l][l] + wq * ( k * dphil * dphil + alpha * phil * phil )
      A[l][r] = A[l][r] + wq * ( k * dphil * dphir + alpha * phil * phir )

      rhs[l]  = rhs[l]  + wq * phil * rhs_f (k, xq ,alpha)

      A[r][l] = A[r][l] + wq * ( k * dphir * dphil + alpha * phir * phil )
      A[r][r] = A[r][r] + wq * ( k * dphir * dphir + alpha * phir * phir )

      rhs[r]  = rhs[r]  + wq * phir * rhs_f (k, xq ,alpha)
#
#  imposicao das condicoes de fronteira.
#
A[0,0] = 1.0
A[0,1:n+1] = 0.0
rhs[0] = 0.0    # valor da função em x=a
#
A[n,n] = 1.0
A[n,0:n] = 0.0
rhs[n] = 0.0    # valor da função em x=b
#
#  resolucao do sistema linear.
#



time1 = timeit.default_timer()
u = linalg.solve( A, rhs ,assume_a='sym' )
#u = cg(A, rhs,  maxiter=n)
time2 = timeit.default_timer() - time1

#u = u[0]
```

```
######print ( " \n  \n    tempo de execução = {} s".format(tempo2))

#  solucao exata.
#
er_abs = 0
er_rel = 0
w_er = np.zeros ( n + 1 )

uex = np.zeros ( n + 1 )
for i in range ( 0, n + 1):
    uex[i]  = sol_e ( x[i] )
    w_er[i] = abs( uex[i] - u[i]  )
    er_abs  = er_abs +  ( w_er[i] )**2
    er_rel  = er_rel +  ( uex[i] )**2
#
er_abs = np.sqrt(er_abs);  er_abs_m = max(w_er)
er_rel = er_abs/np.sqrt(er_rel)

npp = 51
xp = np.linspace ( a, b, npp )



up = np.zeros ( npp )
for i in range ( 0, npp ):
    up[i] = sol_e ( xp[i] )

#
#
plt.plot ( x, u, 'bo-', xp, up, 'r.' )
#plt.savefig ( 'fem1d_'+str(n)+'.png' )
plt.savefig ( '1probEstabK'+str(k)+'alpha'+str(alpha)+'N'+str(n)+'.png' )
plt.show ( )
#
#
print ( '' )
print ( '  Nós          Uh           Ue          Erro' )
print ( '' )
for i in range ( 0, n + 1 ):
    err = abs ( uex[i] - u[i] )
    print ( '  %4d  %14.6g  %14.6g  %14.6g' % ( i, u[i], uex[i], err ) )
#

#nc = np.linalg.cond(A, 'fro')
#nc = np.linalg.cond(A, 1) #norma da soma
nc = np.linalg.cond(A, 2)  #norma euclideana
```

```
evals, evecs = np.linalg.eig(A)

print(" \n  \n    valores próprios = {}".format(evals))

print  (" \n  \n    número de condição = {}".format(nc))

print ( " \n  \n    ndof = {} , h = {}".format(n+1,1./(n+1) ))
print ( " \n  \n    erro absoluto discreto = {} ".format(er_abs))
print  (" \n  \n    erro relativo discreto = {} ".format(er_rel))

## norma continua cálculo do erro para a NORMA L2

erro_L2 = 0. # erro absoluto
norma_L2 = 0. # norma da soluçao exata

for e in range ( 0, n ):
  l = e
  r = e + 1

  xl = x[l]
  xr = x[r]
#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
#
  for q in range ( 0, ng ):
#
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
      phil  = ( xr - xq  ) / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )

      erro_L2  = erro_L2 + wq*( u[l]*phil + u[r]*phir - sol_e( xq ) )**2
      norma_L2 = norma_L2 + wq*sol_e( xq )**2

erro_L2  = np.sqrt(erro_L2)
norma_L2 = np.sqrt(norma_L2)

#
print ( " \n  \n    erro absoluto L2 = {} ".format(erro_L2))
print  (" \n  \n    erro relativo L2 = {} ".format(erro_L2/norma_L2))
```

```
##

f.write("{},{} \n".format(n+1,erro_L2/norma_L2))

fa.write("{},{} \n".format(n+1,erro_L2))

fc.write("{},{} \n".format(n+1,nc))

fb.write("{},{} \n".format(n+1,time2))




f.close()
fa.close()
fb.close()
fc.close()
```

3.1.1 Resultados Obtidos

Nós	Uh	Ue	Erro
0	0	0	0
1	0.000172083	0.0081	0.00792792
2	0.00992099	0.0256	0.015679
3	0.0216958	0.0441	0.0224042
4	0.0305921	0.0576	0.0270079
5	0.0338538	0.0625	0.0286462
6	0.0305921	0.0576	0.0270079
7	0.0216958	0.0441	0.0224042
8	0.00992099	0.0256	0.015679
9	0.000172083	0.0081	0.00792792
10	0	0	0

```
   valores próprios = [0.19625552 0.47559991 0.910689   1.45893324 2.06666667 3.93707782
3.65773342 2.67440009 3.22264433 1.         1.        ]


   número de condição = 20.477792088888688


   ndof = 11 , h = 0.09090909090909091
```

```
    erro absoluto discreto = 0.06245576283943101


    erro relativo discreto = 0.4957223440995644


    erro absoluto L2 = 0.019927488295919747


    erro relativo L2 = 0.500176060546813
```

3.2 1º Problema Estabilizado

Apresenta-se scriptum em código Python para a resolução de um problema de difusão em dimensão 1 com método de estabilização.

```
#! /usr/bin/env python
#

def sol_e ( x ):

#*******************************************************************************80
#
## SOLUCAO EXATA.
#
 value = (x-1)*(x-1)*x*x


 return value


def rhs_f ( k,x,alpha ):

#*******************************************************************************80
#
## SEGUNDO MEMBRO.
#
 value = -k*(12*x*x-12*x+2) + alpha*2*x*(x-1)*(2*x-1)


 return value



#*******************************************************************************80
#    Para 0 < x < 1:
```

```
#      -k u'' + a u' = f(x)
#    condicoes de fronteira
#      u(0) = u(1) = 0.
#



f = open("e_rel_L2_SDM.txt",'a+')
fa = open("e_abs_L2_SDM.txt",'a+')
fc = open("numCond_SDM.txt",'a+')
fb = open("tempoResol_SDM.txt",'a+')

#
a = 0.0;   k=0.01; alpha=1;
b = 1.0

#for n in range(5,100): # número de elementos =  número de nós -1 =
 # número de graus de liberdade -1 = número de funções de base -1

n=20

h=1./(n+1)
x = np.linspace ( a, b, n + 1 )
#
#  3 pontos  quadratura de Gauss [-1,1].
#
ng = 3

xg = np.array ( ( -np.sqrt(3./5.),  0.   , np.sqrt(3./5.) ) )

wg = np.array ( (       5./9.,  8./9.,      5./9. ) )
#
#  Calculo de A e RHS.
#
A = np.zeros ( ( n + 1, n + 1 ) )
rhs = np.zeros ( n + 1 )
#
#  Ciclo sobre os elementos E: (0, 1, 2, ..., N-1).
#
for e in range ( 0, n ):
  l = e
  r = e + 1

  xl = x[l]
  xr = x[r]
#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
```

```
#
  for q in range ( 0, ng ):
#
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
      phil  = ( xr - xq  ) / ( xr - xl )
      dphil =  -1. / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )
      dphir = 1. / ( xr - xl )
#
#
      A[l][l] = A[l][l] + wq * ( (k+alpha*alpha*h*.5) * dphil * dphil -
                                      alpha * dphil * phil )

      A[l][r] = A[l][r] + wq * ( (k+alpha*alpha*h*.5) * dphil * dphir -
                                      alpha * dphil * phir )

      rhs[l]  = rhs[l]  + wq * phil * rhs_f (k, xq ,alpha)  + wq * dphil * rhs_f (k, xq,alpha )

      A[r][l] = A[r][l] + wq * ( (k+alpha*alpha*h*.5) * dphir * dphil -
                                      alpha * dphir * phil )

      A[r][r] = A[r][r] + wq * ( (k+alpha*alpha*h*.5) * dphir * dphir -
                                      alpha * dphir * phir )

      rhs[r]  = rhs[r]  + wq * phir * rhs_f (k, xq,alpha ) + wq * dphir * rhs_f (k, xq,alpha )
#
#  imposicao das condicoes de fronteira.
#
A[0,0] = 1.0
A[0,1:n+1] = 0.0
rhs[0] = 0.0    # valor da função em x=a
#
A[n,n] = 1.0
A[n,0:n] = 0.0
rhs[n] = 0.0    # valor da função em x=b
#
#  resolucao do sistema linear.
#
```

```
time1 = timeit.default_timer()
#u = cg(A, rhs, tol=1e-05, maxiter=n)
u = linalg.solve( A, rhs , assume_a='gen')

time2 = timeit.default_timer() - time1

#u = u[0]



######print ( " \n  \n    tempo de execução = {} s".format(tempo2))

#  solucao exata.
#
er_abs = 0
er_rel = 0
w_er = np.zeros ( n + 1 )

uex = np.zeros ( n + 1 )
for i in range ( 0, n + 1):
    uex[i]  = sol_e ( x[i] )
    w_er[i] = abs( uex[i] - u[i]  )
    er_abs  = er_abs +  ( w_er[i] )**2
    er_rel  = er_rel +  ( uex[i] )**2
#
er_abs = np.sqrt(er_abs);  er_abs_m = max(w_er)
er_rel = er_abs/np.sqrt(er_rel)

npp = 51
xp = np.linspace ( a, b, npp )



up = np.zeros ( npp )
for i in range ( 0, npp ):
    up[i] = sol_e ( xp[i] )

#
#
plt.plot ( x, u, 'bo-', xp, up, 'r.' )
#plt.savefig ( 'fem1d_'+str(n)+'.png' )
plt.savefig ( '1probEstabK'+str(k)+'alpha'+str(alpha)+'N'+str(n)+'.png' )
plt.show ( )
#
```

```
#
print ( '' )
print ( '  Nós          Uh           Ue          Erro' )
print ( '' )
for i in range ( 0, n + 1 ):
    err = abs ( uex[i] - u[i] )
    print ( '  %4d  %14.6g  %14.6g  %14.6g' % ( i, u[i], uex[i], err ) )
#

#nc = np.linalg.cond(A, 'fro')
#nc = np.linalg.cond(A, 1) #norma da soma
nc = np.linalg.cond(A, 2)  #norma euclideana

evals, evecs = np.linalg.eig(A)

print(" \n  \n    valores próprios = {}".format(evals))

print  (" \n  \n    número de condição = {}".format(nc))

print ( " \n  \n    ndof = {} , h = {}".format(n+1,1./(n+1) ))
print ( " \n  \n    erro absoluto discreto = {} ".format(er_abs))
print  (" \n  \n    erro relativo discreto = {} ".format(er_rel))

## norma continua cálculo do erro para a NORMA L2

erro_L2 = 0. # erro absoluto
norma_L2 = 0. # norma da soluçao exata

for e in range ( 0, n ):
  l = e
  r = e + 1

  xl = x[l]
  xr = x[r]
#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
#
  for q in range ( 0, ng ):
#
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
```

```
      phil  = ( xr - xq  ) / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )

      erro_L2  = erro_L2 + wq*( u[l]*phil + u[r]*phir - sol_e( xq ) )**2
      norma_L2 = norma_L2 + wq*sol_e( xq )**2

erro_L2  = np.sqrt(erro_L2)
norma_L2 = np.sqrt(norma_L2)

#
print ( " \n  \n    erro absoluto L2 = {} ".format(erro_L2))
print  (" \n  \n    erro relativo L2 = {} ".format(erro_L2/norma_L2))
##

f.write("{},{} \n".format(n+1,erro_L2/norma_L2))

fa.write("{},{} \n".format(n+1,erro_L2))

fc.write("{},{} \n".format(n+1,nc))

fb.write("{},{} \n".format(n+1,time2))



f.close()
fa.close()
fb.close()
fc.close()
```

3.2.1 Resultados Obtidos

Nós	Uh	Ue	Erro
0	-1.9542e-17	0	1.9542e-17
1	0.00224049	0.00225625	1.57568e-05
2	0.00807027	0.0081	2.97279e-05
3	0.0162143	0.0162563	4.19133e-05
4	0.0255477	0.0256	5.23129e-05
5	0.0350953	0.0351562	6.09269e-05
6	0.0440322	0.0441	6.77551e-05
7	0.0516835	0.0517562	7.27976e-05
8	0.0575239	0.0576	7.60544e-05

```
 9       0.0611787       0.0612563     7.75255e-05
10       0.0624228          0.0625     7.72109e-05
11       0.0611811       0.0612562     7.51105e-05
12       0.0575288          0.0576     7.12245e-05
13       0.0516907       0.0517562     6.55528e-05
14       0.0440419          0.0441     5.80955e-05
15       0.0351074       0.0351562     4.88539e-05
16       0.0255622          0.0256     3.78353e-05
17       0.0162312       0.0162562     2.50897e-05
18      0.00808905          0.0081     1.09501e-05
19      0.00225861      0.00225625     2.36033e-06
20               0               0               0


   valores próprios = [2.25162996 2.21827821 2.1636052  2.08895716 1.99617216 1.88753489
1.76572035 1.63372803 1.494808   1.35238095 1.2099539  0.45313195
0.48648369 0.5411567  0.61580475 0.70858974 1.07103388 0.81722702
0.93904155 1.         1.        ]


   número de condição = 34.310718131079696


   ndof = 21 , h = 0.047619047619047616


   erro absoluto discreto = 0.0002448339670160885


   erro relativo discreto = 0.0013741270176846493


   erro absoluto L2 = 0.0002311581369312179


   erro relativo L2 = 0.0058020232029031135
```

3.3 2º Problema não Estabilizado

Apresenta-se scriptum em código Python para a resolução de um problema de camada limite em dimensão 1.

```
def sol_e ( x ):

#*************************************************************************80
#
```

```
## SOLUCAO EXATA.
#
 constC = (1) / ( np.exp(1/k) - 1)
 value = constC * ( np.exp(1/k) - np.exp(x/k) )

 return value


def rhs_f ( k,x,alpha ):

#*******************************************************************************80
#
## SEGUNDO MEMBRO.
#
 constC = (1) / ( np.exp(1/k) - 1)
 value = -k*( -1 * constC/(k**2) * np.exp(x/k) ) + alpha * ( -1 * constC/k * np.exp(x/k) )

 return value



#*******************************************************************************80
#    Para 0 < x < 1:
#      -k u'' + a u' = 0
#    condicoes de fronteira
#      u(0) = 1 , u(1) = 0.
#



f = open("e_rel_L2_SDM.txt",'a+')
fa = open("e_abs_L2_SDM.txt",'a+')
fc = open("numCond_SDM.txt",'a+')
fb = open("tempoResol_SDM.txt",'a+')
fd = open("valPropios.txt",'a+')


a = 0.0;   k= 0.01; alpha=1;
b = 1.0

#for n in range(5,100): # número de elementos =  número de nós -1 =
 # número de graus de liberdade -1 = número de funções de base -1

n=20

x = np.linspace ( a, b, n + 1 )
#
```

```
#  3 pontos  quadratura de Gauss [-1,1].
#
ng = 3

xg = np.array ( ( -np.sqrt(3./5.),  0.   , np.sqrt(3./5.) ) )

wg = np.array ( (       5./9.,  8./9.,      5./9. ) )
#
#  Calculo de A e RHS.
#
A = np.zeros ( ( n + 1, n + 1 ) )
rhs = np.zeros ( n + 1 )
#
#  Ciclo sobre os elementos E: (0, 1, 2, ..., N-1).
#
for e in range ( 0, n ):
  l = e
  r = e + 1

  xl = x[l]
  xr = x[r]
#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
#
  for q in range ( 0, ng ):
#
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
      phil  = ( xr - xq  ) / ( xr - xl )
      dphil =  -1. / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )
      dphir = 1. / ( xr - xl )
#
#
      A[l][l] = A[l][l] + wq * ( k * dphil * dphil + alpha * phil * phil )
      A[l][r] = A[l][r] + wq * ( k * dphil * dphir + alpha * phil * phir )

      #rhs[l]  = rhs[l]  + wq * phil * rhs_f (k, xq )

      A[r][l] = A[r][l] + wq * ( k * dphir * dphil + alpha * phir * phil )
```

```
      A[r][r] = A[r][r] + wq * ( k * dphir * dphir + alpha * phir * phir )

      #rhs[r]  = rhs[r]  + wq * phir * rhs_f (k, xq )
#
#  imposicao das condicoes de fronteira.
#
A[0,0] = 1.0
A[0,1:n+1] = 0.0
rhs[0] = 1.0    # valor da função em x=a
#
A[n,n] = 1.0
A[n,0:n] = 0.0
rhs[n] = 0.0    # valor da função em x=b
#
#  resolucao do sistema linear.
#




time1 = timeit.default_timer()
#u = cg(A, rhs, tol=1e-05, maxiter=n)
u = linalg.solve( A, rhs ,assume_a='sym' )
time2 = timeit.default_timer() - time1

#u = u[0]



######print ( " \n  \n    tempo de execução = {} s".format(tempo2))

#  solucao exata.
#
er_abs = 0
er_rel = 0
w_er = np.zeros ( n + 1 )

uex = np.zeros ( n + 1 )
for i in range ( 0, n + 1):
    uex[i]  = sol_e ( x[i] )
    w_er[i] = abs( uex[i] - u[i]  )
    er_abs  = er_abs +  ( w_er[i] )**2
    er_rel  = er_rel +  ( uex[i] )**2
#
er_abs = np.sqrt(er_abs);  er_abs_m = max(w_er)
er_rel = er_abs/np.sqrt(er_rel)
```

```
npp = 51
xp = np.linspace ( a, b, npp )



up = np.zeros ( npp )
for i in range ( 0, npp ):
    up[i] = sol_e ( xp[i] )

#
#
plt.plot ( x, u, 'bo-', xp, up, 'r.' )
#plt.savefig ( 'fem1d_'+str(n)+'.png' )
plt.savefig ( '2probEstabK'+str(k)+'alpha'+str(alpha)+'N'+str(n)+'.png' )
plt.show ( )
#
#
print ( '' )
print ( '  Nós          Uh           Ue          Erro' )
print ( '' )
for i in range ( 0, n + 1 ):
    err = abs ( uex[i] - u[i] )
    print ( '  %4d  %14.6g  %14.6g  %14.6g' % ( i, u[i], uex[i], err ) )
#

#nc = np.linalg.cond(A, 'fro')
#nc = np.linalg.cond(A, 1) #norma da soma
nc = np.linalg.cond(A, 2)  #norma euclideana

evals, evecs = np.linalg.eig(A)

print(" \n  \n    valores próprios = {}".format(evals))

print  (" \n  \n    número de condição = {}".format(nc))

print ( " \n  \n    ndof = {} , h = {}".format(n+1,1./(n+1) ))
print ( " \n  \n    erro absoluto discreto = {} ".format(er_abs))
print  (" \n  \n    erro relativo discreto = {} ".format(er_rel))

## norma continua cálculo do erro para a NORMA L2

erro_L2 = 0. # erro absoluto
norma_L2 = 0. # norma da soluçao exata

for e in range ( 0, n ):
  l = e
  r = e + 1
```

```
  xl = x[l]
  xr = x[r]
#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
#
  for q in range ( 0, ng ):
#
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
      phil  = ( xr - xq  ) / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )

      erro_L2  = erro_L2 + wq*( u[l]*phil + u[r]*phir - sol_e( xq ) )**2
      norma_L2 = norma_L2 + wq*sol_e( xq )**2

erro_L2  = np.sqrt(erro_L2)
norma_L2 = np.sqrt(norma_L2)

#
print ( " \n  \n    erro absoluto L2 = {} ".format(erro_L2))
print  (" \n  \n    erro relativo L2 = {} ".format(erro_L2/norma_L2))
##

f.write("{},{} \n".format(n+1,erro_L2/norma_L2))

fa.write("{},{} \n".format(n+1,erro_L2))

fc.write("{},{} \n".format(n+1,nc))

fb.write("{},{} \n".format(n+1,time2))

fd.write("{},{} \n".format(n+1,evals))




f.close()
fa.close()
```

```
fb.close()
fc.close()
fd.close()
```

3.3.1 Resultados Obtidos

Nós	Uh	Ue	Erro
0	1	1	0
1	0.603289	1	0.396711
2	0.363957	1	0.636043
3	0.219571	1	0.780429
4	0.132465	1	0.867535
5	0.0799147	1	0.920085
6	0.0482116	1	0.951788
7	0.0290855	1	0.970915
8	0.0175469	1	0.982453
9	0.0105857	1	0.989414
10	0.0063861	1	0.993614
11	0.00385239	1	0.996148
12	0.00232365	1	0.997676
13	0.00140108	1	0.998599
14	0.000844003	1	0.999156
15	0.000507104	1	0.999493
16	0.000302494	1	0.999698
17	0.000176795	1	0.999823
18	9.72171e-05	0.999955	0.999857
19	4.29999e-05	0.993262	0.993219
20	0	0	0

```
   valores próprios = [0.8119472  0.797905   0.77488583 0.74345651 0.70439093 0.65865101
0.60736302 0.55178985 0.49329988 0.43333333 0.37336679 0.31487682
0.05471947 0.06876167 0.09178083 0.12321015 0.16227573 0.20801565
0.25930364 1.         1.        ]


   número de condição = 18.72171478838395


   ndof = 21 , h = 0.047619047619047616


   erro absoluto discreto = 4.064182367231877


   erro relativo discreto = 0.9090861710548648
```

```
    erro absoluto L2 = 0.913357779144238


    erro relativo L2 = 0.9204125141219555
```

3.4 2º Problema Estabilizado

Apresenta-se scriptum em código Python para a resolução de um problema de camada limite em dimensão 1 com método de estabilização.

```
def sol_e ( x ):

#*********************************************************************************80
#
## SOLUCAO EXATA.
#
 constC = (1) / ( np.exp(1/k) - 1)
 value = constC * ( np.exp(1/k) - np.exp(x/k) )

 return value


def rhs_f ( k,x ,alpha):

#*********************************************************************************80
#
## SEGUNDO MEMBRO.
#
 constC = (1) / ( np.exp(1/k) - 1)
 value = -k*( -1 * constC/(k**2) * np.exp(x/k) ) + alpha * ( -1 * constC/k * np.exp(x/k) )

 return value



#*********************************************************************************80
#    Para 0 < x < 1:
#      -k u'' + a u' = 0
#    condicoes de fronteira
#      u(0) = 1 , u(1) = 0.
#



f = open("e_rel_L2_SDM.txt",'a+')
```

```
fa = open("e_abs_L2_SDM.txt",'a+')
fc = open("numCond_SDM.txt",'a+')
fb = open("tempoResol_SDM.txt",'a+')
fd = open("valPropios.txt",'a+')

#
a = 0.0;   k=0.01; alpha=1;
b = 1.0


#for n in range(4,100): # número de elementos =  número de nós -1 =
 # número de graus de liberdade -1 = número de funções de base -1

n=5

h=1./(n+1)

x = np.linspace ( a, b, n + 1 )
#
#  3 pontos  quadratura de Gauss [-1,1].
#
ng = 3

xg = np.array ( ( -np.sqrt(3./5.),  0.   , np.sqrt(3./5.) ) )

wg = np.array ( (       5./9.,  8./9.,      5./9. ) )
#
#  Calculo de A e RHS.
#
A = np.zeros ( ( n + 1, n + 1 ) )
rhs = np.zeros ( n + 1 )
#
#  Ciclo sobre os elementos E: (0, 1, 2, ..., N-1).
#
for e in range ( 0, n ):
  l = e
  r = e + 1

  xl = x[l]
  xr = x[r]


#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
#
  for q in range ( 0, ng ):
#
```

```
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
      phil  = ( xr - xq  ) / ( xr - xl )
      dphil =  -1. / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )
      dphir = 1. / ( xr - xl )
#
#
      A[l][l] = A[l][l] + wq * ( (k+alpha*alpha*h*.5) * dphil * dphil -
                                  alpha * dphil * phil )

      A[l][r] = A[l][r] + wq * ( (k+alpha*alpha*h*.5) * dphil * dphir -
                                  alpha * dphil * phir )

#     rhs[l]  = rhs[l]  + wq * phil * rhs_f (k, xq ,alpha)  + wq * dphil * rhs_f (k, xq,alph

      A[r][l] = A[r][l] + wq * ( (k+alpha*alpha*h*.5) * dphir * dphil -
                                  alpha * dphir * phil )

      A[r][r] = A[r][r] + wq * ( (k+alpha*alpha*h*.5) * dphir * dphir -
                                  alpha * dphir * phir )

#     rhs[r]  = rhs[r]  + wq * phir * rhs_f (k, xq,alpha ) + wq * dphir * rhs_f (k, xq,alpha
#
#  imposicao das condicoes de fronteira.
#
A[0,0] = 1.0
A[0,1:n+1] = 0.0
rhs[0] = 1.0    # valor da função em x=a
#
A[n,n] = 1.0
A[n,0:n] = 0.0
rhs[n] = 0.0    # valor da função em x=b
#
#  resolucao do sistema linear.
#
```

```
time1 = timeit.default_timer()
#u = cg(A, rhs, x0=None, tol=1e-05, maxiter=None, M=None, callback=None, atol=None)
u = linalg.solve( A, rhs, assume_a='gen' )
time2 = timeit.default_timer() - time1

#u = u[0]



######print ( " \n  \n    tempo de execução = {} s".format(tempo2))

#  solucao exata.
#
er_abs = 0
er_rel = 0
w_er = np.zeros ( n + 1 )

uex = np.zeros ( n + 1 )
for i in range ( 0, n + 1):
    uex[i]  = sol_e ( x[i] )
    w_er[i] = abs( uex[i] - u[i]  )
    er_abs  = er_abs +  ( w_er[i] )**2
    er_rel  = er_rel +  ( uex[i] )**2
#
er_abs = np.sqrt(er_abs);  er_abs_m = max(w_er)
er_rel = er_abs/np.sqrt(er_rel)

npp = 51
xp = np.linspace ( a, b, npp )



up = np.zeros ( npp )
for i in range ( 0, npp ):
    up[i] = sol_e ( xp[i] )

#
#
plt.plot ( x, u, 'bo-', xp, up, 'r.' )
#plt.savefig ( 'fem1d_'+str(n)+'.png' )
#plt.savefig ( '2probEstabK'+str(k)+'alpha'+str(alpha)+'N'+str(n)+'.png' )
plt.show ( )
#
#
print ( '' )
print ( '  Nós          Uh           Ue          Erro' )
print ( '' )
```

```
for i in range ( 0, n + 1 ):
    err = abs ( uex[i] - u[i] )
    print ( '  %4d  %14.6g  %14.6g  %14.6g' % ( i, u[i], uex[i], err ) )
#

#nc = np.linalg.cond(A, 'fro')
#nc = np.linalg.cond(A, 1) #norma da soma
nc = np.linalg.cond(A, 2)  #norma euclideana

evals, evecs = np.linalg.eig(A)

print(" \n  \n    valores próprios = {}".format(evals))

print  (" \n  \n    número de condição = {}".format(nc))

print ( " \n  \n    ndof = {} , h = {}".format(n+1,1./(n+1) ))
print ( " \n  \n    erro absoluto discreto = {} ".format(er_abs))
print  (" \n  \n    erro relativo discreto = {} ".format(er_rel))

## norma continua cálculo do erro para a NORMA L2

erro_L2 = 0. # erro absoluto
norma_L2 = 0. # norma da soluçao exata

for e in range ( 0, n ):
  l = e
  r = e + 1

  xl = x[l]
  xr = x[r]
#
#  Considerar os pontos de quadratura  Q: (0, 1, 2 ) no  elemento E.
#
  for q in range ( 0, ng ):
#
#  aplicacao de [0,1] para
#      XQ e WQ para [XL,XR].
#
      xq = xl + .5*( xr - xl )*(xg[q] +1.)   # transformação dos pontos de gauss
      wq = wg[q] * ( xr - xl )*.5
#
#  Calculo das funcoes de base e derivadas para  XQ em XL e XR.
#
      phil  = ( xr - xq  ) / ( xr - xl )

      phir  = ( xq - xl ) / ( xr - xl )
```

```
      erro_L2  = erro_L2 + wq*( u[l]*phil + u[r]*phir - sol_e( xq ) )**2
      norma_L2 = norma_L2 + wq*sol_e( xq )**2

erro_L2  = np.sqrt(erro_L2)
norma_L2 = np.sqrt(norma_L2)

#
print ( " \n  \n    erro absoluto L2 = {} ".format(erro_L2))
print  (" \n  \n    erro relativo L2 = {} ".format(erro_L2/norma_L2))
##

f.write("{},{} \n".format(n+1,erro_L2/norma_L2))

fa.write("{},{} \n".format(n+1,erro_L2))

fc.write("{},{} \n".format(n+1,nc))

fb.write("{},{} \n".format(n+1,time2))

fd.write("{},{} \n".format(n+1,evals))




f.close()
fa.close()
fb.close()
fc.close()
fd.close()
```

3.4.1 Resultados Obtidos

Nós	Uh	Ue	Erro
0	1	1	0
1	0.999999	1	1.46262e-06
2	1.00004	1	4.09533e-05
3	0.998811	1	0.00118911
4	1.03448	1	0.0344827
5	0	0	0

```
   valores próprios = [0.93333333+0.29044599j 0.93333333-0.29044599j 0.93333333+0.1109405j
 0.93333333-0.1109405j  1.        +0.j         1.        +0.j        ]
```

```
número de condição = 6.464903415593965

ndof = 6 , h = 0.16666666666666666

erro absoluto discreto = 0.034503231244201085

erro relativo discreto = 0.015430314107446504

erro absoluto L2 = 0.23415402267039198

erro relativo L2 = 0.23545970569522182
```

D. Código freeFEM++

4.1 Problema FreeFem bidimensional

Apresenta-se scriptum em código freeFEM++ para a resolução de um problema de difusão bidimensional.

```
int nn = 100; //qualidade da malha

func f = -2*((y-1)*y+(x-1)*x); //força externa

func u = (x-1)*x*(y-1)*y;  // solução exata

// Malha
mesh Th = square(nn, nn, [x, y]);

//plot(Th);


//qualidade da malha
fespace Vh(Th, P1);
Vh uh,v;

// Macro
macro grad(u) [dx(u), dy(u)] //

// Problema
problem Laplacian (uh, v)
= int2d(Th)(
  grad(uh)' * grad(v)
)
- int2d(Th)(
  f * v
)
+ on(1, 2, 3, 4, uh=0)
;

// Resolução
```

```
Laplacian;

v=uh-u;

real erro = sqrt( int2d(Th)( v*v) );

cout<<" erro na regiao "<<erro<<endl;

// Plot
plot(uh,value=1, fill=true);

//plot(v, fill=true,  value=1);
```

4.1.1 Resultados Obtidos

```
  -- Square mesh : nb vertices  =10201 ,  nb triangles = 20000 ,  nb boundary edges 400
  -- Solve :
         min 9.79917e-65  max 0.0624951
 erro na regiao 2.75006e-06
```

4.2 Problema camada limite FreeFem bidimensional sem estabilização

Apresenta-se scriptum em código freeFEM++ para a resolução de um problema de camada limite bidimensional sem estabilização.

```
int nn = 10;
real h = 1./(nn+1);

real k=1.;

mesh Th = square(nn, nn, [x, y]);
fespace Vh(Th,P1);
Vh uh,vh;

func f = 1.;
func g = 0.;
func a1= 1.; func a2 =1.;

macro Grad(u)[dx(u),dy(u)]//
```

```
solve advdif(uh,vh) =   int2d(Th)(k*dx(uh)*dx(vh))
  + int2d(Th)(k*dy(uh)*dy(vh))
                      + int2d(Th)((a1*dx(uh)+a2*dy(uh))*vh)
                      + on (4,uh=f)
  + on(1,2,3,uh=g);

plot(uh,dim=2,fill=true,value=true);
```

4.2.1 Resultados Obtidos

```
  -- Square mesh : nb vertices  =121 ,  nb triangles = 200 ,  nb boundary edges 40
  -- Solve :
         min 1.37168e-62  max 1
times: compile 0.052s, execution 0.005s,  mpirank:0
 CodeAlloc : nb ptr  3456,  size :466304 mpirank: 0
```

4.3 Problema camada limite FreeFem bidimensional estabilizado

Apresenta-se scriptum em código freeFEM++ para a resolução de um problema de camada limite bidimensional estabilizado.

```
int nn = 10;
real h = 1./(nn+1);

real k=1.;

mesh Th = square(nn, nn, [x, y]);
fespace Vh(Th,P1);
Vh uh,vh;

func f = 1.;
func g = 0.;

func a1= 1.; func a2 =1.;

macro Grad(u)[dx(u),dy(u)]//

solve advdif(uh,vh) =   int2d(Th)(k*dx(uh)*dx(vh))
  + int2d(Th)(k*dy(uh)*dy(vh))
                      + int2d(Th)((a1*dx(uh)+a2*dy(uh))*(vh+.5*h*(a1*dx(vh)+a2*dy(vh))))
                      + on(4,uh=f)
  + on(1,2,3,uh=g);

plot(uh,dim=2,fill=true,value=true);
```

4.3.1 Resultados Obtidos

```
  -- Square mesh : nb vertices  =121 ,  nb triangles = 200 ,  nb boundary edges 40
  -- Solve :
         min 1.24112e-62  max 1
times: compile 0.65s, execution 0.146s,  mpirank:0
 CodeAlloc : nb ptr  3475,  size :467472 mpirank: 0
```