U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

DESENVOLVIMENTO DE UM PROGRAMA DE ELEMENTOS FINITOS VERSÁTIL

ANDRÉ FILIPE MARTINS DE SOUSA

Dissertação submetida para satisfação parcial dos requisitos do grau de

MESTRE EM ENGENHARIA CIVIL — ESPECIALIZAÇÃO EM ESTRUTURAS

Orientador: Professor Doutor Álvaro Ferreira Marques Azevedo

OUTUBRO DE 2014

MESTRADO INTEGRADO EM ENGENHARIA CIVIL 2013/2014

DEPARTAMENTO DE ENGENHARIA CIVIL

Tel. +351-22-508 1901

Fax +351-22-508 1446

✉ miec@fe.up.pt

Editado por

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Rua Dr. Roberto Frias

4200-465 PORTO

Portugal

Tel. +351-22-508 1400

Fax +351-22-508 1440

✉ feup@fe.up.pt

🌍 http://www.fe.up.pt

Reproduções parciais deste documento serão autorizadas na condição que seja mencionado o Autor e feita referência a *Mestrado Integrado em Engenharia Civil - 2013/2014 - Departamento de Engenharia Civil, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal, 2014*.

As opiniões e informações incluídas neste documento representam unicamente o ponto de vista do respetivo Autor, não podendo o Editor aceitar qualquer responsabilidade legal ou outra em relação a erros ou omissões que possam existir.

Este documento foi produzido a partir de versão eletrónica fornecida pelo respetivo Autor.

"O único lugar onde sucesso
vem antes do trabalho é no dicionário."
Albert Einstein

RESUMO

A importância da simulação numérica justifica a realização de um trabalho que faça uso das possibilidades computacionais para desenvolver uma solução informática que implemente o Método do Elementos Finitos na resolução de diversos problemas na área da engenharia estrutural.

Nesta dissertação são descritas diversas teorias e formulações de elementos finitos usadas correntemente na análise de problemas estruturais. Das diversas formulações e tipos de elementos finitos disponíveis são focados os elementos finitos unidimensionais, bidimensionais, tridimensionais, de viga e de laje. São apresentados elementos finitos de viga formulados pela teoria de Euler-Bernoulli e pela teoria de Timoshenko. Os elementos finitos de laje apresentados abrangem a teoria de Kirchhoff e a teoria de Reissner-Mindlin.

De modo a implementar computacionalmente o Método dos Elementos Finitos são discutidos os fundamentos do método e o seu âmbito de aplicabilidade. É alvo de pormenorização a obtenção das funções de forma, das matrizes de rigidez elementares e dos vetores de forças nodais equivalentes. É focada a construção da equação de equilíbrio global e a sua resolução. A integração numérica e a geração de malhas é uma das partes constituintes da análise estrutural pelo Método dos Elementos Finitos. Assim, são abordadas algumas técnicas de geração de malhas e discutidos alguns requisitos que as malhas devem cumprir.

São abordados alguns conceitos sobre desenvolvimento de programas informáticos, discutido o modo como o programa desenvolvido foi construído e apresentados alguns códigos exemplificativos de algumas das tarefas realizadas.

É apresentado o programa de modelação e análise estrutural pelo Método dos Elementos Finitos desenvolvido no âmbito desta dissertação. São referidas e apresentadas algumas classes e métodos do programa. É apresentada a interface gráfica do utilizador e as funcionalidades disponíveis. De modo a exemplificar o funcionamento do programa são detalhadas todas as etapas desde a modelação da estrutura até à visualização gráfica dos resultados.

PALAVRAS-CHAVE: Método dos Elementos Finitos, Elementos Unidimensionais, Elementos Bidimensionais, Elementos Tridimensionais, Vigas, Lajes, Desenvolvimento de *Software*, Java, FEM for Students.

ABSTRACT

The importance of numerical simulation justifies a work that makes use of computational possibilities to develop a software solution that implements the Finite Element Method to solve various problems in the area of structural engineering.

This dissertation describes several theories and formulations currently used in finite element structural analysis problems. From the different formulations and types of finite elements available, the focus is on one-dimensional, two-dimensional, three-dimensional finite element, beams and slabs. Finite beam elements formulated by the Euler-Bernoulli theory and the Timoshenko theory are presented. The finite element slab presented covers the Kirchhoff theory and Reissner-Mindlin theory.

In order to implement computationally the Finite Element Method, the fundamentals of the method and its scope of applicability are discussed. The acquisition of shape functions, the elementary stiffness matrix and equivalent nodal forces vector are detailed. The construction of the global equilibrium equation and its resolution is focused on. The numerical integration and mesh generation is one of the constituent parts of the structural analysis by Finite Element Method. Thus, some techniques for mesh generation are addressed and requirements that the meshes must meet are discussed.

Concepts on the development of computer programs are referred to, how the software was built is discussed and illustrative codes of some of the tasks performed are presented.

Program modelling and structural analysis by Finite Element Method developed in the context of this dissertation is presented. Classes and methods of the program are referred to and presented. The graphical user interface and functionalities are addressed. In order to demonstrate the operation of the program, all the steps from the structure modelling to the graphic display of results are detailed.

KEYWORDS: Finite Element Method, One-dimensional Elements, Two-dimensional Elements, Tree-dimensional Elements, Beams, Slabs, Software Development, Java, FEM for Students.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1
INTRODUÇÃO

1.1. ÂMBITO E OBJETIVOS DO TRABALHO

A simulação numérica é fundamental para a competitividade do projeto de estruturas. O projeto de estruturas necessita de ferramentas informáticas que auxiliem o seu desenvolvimento. A dificuldade ou a impossibilidade de construir modelos físicos para validar os pressupostos do projeto impulsionou o desenvolvimento de ferramentas de simulação numérica. O Método dos Elementos Finitos é uma destas ferramentas ao alcance dos projetistas.

O Método dos Elementos Finitos é um método matemático de análise e resolução de problemas científicos e de engenharia. De uma forma geral, o Método dos Elementos Finitos é utilizado na busca de soluções para problemas complexos, para os quais não se conhece uma solução exata que possa ser expressa de forma analítica. A grande quantidade de cálculo associada à aplicação deste método requer a utilização de computadores capazes de dar resposta ao elevado número de cálculos efetuados. Surge, deste modo, a necessidade de transformar a formulação matemática do Método dos Elementos Finitos num conjunto de instruções que possam ser interpretadas pelos computadores.

Antes de implementar computacionalmente o Método dos Elementos Finitos é necessário fazer um estudo aprofundado da sua formulação. Assim, são apresentadas diversas formulações de elementos finitos, nomeadamente, as formulações de elementos finitos unidimensionais, bidimensionais, tridimensionais, de viga e de laje. É considerado que os materiais apresentam comportamento linear elástico e que as estruturas e cargas estruturais são estáticas. Esta simplificação reflete a forma de muitos dos problemas estruturais encontrados na prática de engenharia. O uso da integração numérica é também abordado na medida do uso recorrente que tem no âmbito do Método dos Elementos Finitos. Como a análise de um problema requer a decomposição do domínio em estudo numa malha de elementos finitos, são abordadas algumas técnicas para geração de malhas de elementos finitos e discutida a sua qualidade.

O desenvolvimento de ferramentas de simulação numérica orientadas para uso profissional tem renegado a componente académica para segundo plano. A componente prática desta dissertação consiste no desenvolvimento de uma ferramenta orientada para o ensino do Método dos Elementos Finitos. Esta ferramenta disponibiliza a experiência de modelar com elementos finitos e face à multiplicidade de plataformas e à velocidade com que evoluem, o programa desenvolvido foi construído utilizando a linguagem de programação Java.

Relativamente ao Método dos Elementos Finitos são discutidos alguns dos seus fundamentos e apresentadas as metodologias para obtenção das funções de forma, cálculo das matrizes de rigidez elementares e vetores de forças nodais equivalentes. É abordada a geração de malhas de elementos finitos, a resolução do sistema de equações global e o cálculo dos esforços e tensões ao nível de cada

elemento finito. Do ponto de vista de programação são apresentados alguns conceitos sobre o projeto de programas informáticos. Como referido, a componente prática desta dissertação consiste no desenvolvimento de um programa de modelação e análise estrutural por elementos finitos. Assim, é discutido o modo como o Método dos Elementos Finitos está implementado computacionalmente no programa desenvolvido.

Finalmente, o programa desenvolvido é apresentado pelas óticas do programador e utilizador. Do ponto de vista do programador é discutido o modo como o programa foi construído e exemplifica-se com a apresentação de excertos de algumas das classes e métodos do projeto. Esta exemplificação consiste na apresentação de alguns dos códigos construídos para realizar operações de cálculo associadas ao Método dos Elementos Finitos e outras relativas à interface gráfica do utilizador. Na ótica de utilização do programa é descrita a sua interface gráfica com especificação do modo como as etapas associadas a uma análise por elementos finitos são realizadas.

1.2. ESTRUTURAÇÃO DA DISSERTAÇÃO

A presente dissertação está organizada em dez capítulos. Esta divisão é resultante da distribuição de assuntos abordados.

O segundo capítulo apresenta os fundamentos do Método dos Elementos Finitos necessários para a compreensão dos assuntos abordados nos capítulos seguintes.

No terceiro e quarto capítulo são apresentados os mais comuns tipos de elementos finitos e teorias associadas. Estes capítulos são dedicados à apresentação das formulações de elementos finitos unidimensionais, bidimensionais, tridimensionais, de viga e de laje. Pretende-se apresentar a informação necessária à compreensão das diversas formulações de elementos finitos.

O quinto e o sexto capítulo são dedicados à integração numérica e à geração de malhas de elementos finitos, respetivamente. Procura-se completar a informação apresentada nos capítulos anteriores sobre o Método dos Elementos Finitos.

No sétimo capítulo são referidos alguns conceitos fundamentais para as fases de planeamento e desenvolvimento de programas informáticos. São transmitidos alguns conhecimentos específicos sobre programação e informações necessárias à correta tomada de decisões ao longo do projeto de *software*.

O oitavo capítulo é dedicado à implementação computacional. Neste capítulo é feita a apresentação do programa na ótica do programador, ou seja, é apresentado o modo como o programa está construído com a inclusão de alguns códigos para exemplificação.

O nono capítulo refere-se à apresentação da interface gráfica do programa de modelação e análise estrutural por elementos finitos desenvolvido no âmbito desta dissertação.

Por fim, são apresentadas algumas conclusões e sugestões para desenvolvimentos futuros.

2

FUNDAMENTOS DO MÉTODO DOS ELEMENTOS FINITOS

2.1. DESCRIÇÃO DO MÉTODO DOS ELEMENTOS FINITOS

O Método dos Elementos Finitos é uma excelente ferramenta numérica de resolução de problemas do meio contínuo. Hoje em dia, é impensável projetar estruturas inovadoras e/ou arrojadas sem se recorrer a este método de análise estrutural.

A formulação do Método dos Elementos Finitos apresentada neste trabalho é uma extensão da formulação matricial do Método dos Deslocamentos. Desta forma, muitos dos conceitos associados ao Método dos Deslocamentos têm correspondência com os da formulação do Método dos Elementos Finitos [2]. Por exemplo, destacam-se as noções de grau de liberdade, deslocamento generalizado, força generalizada, matriz de rigidez elementar, vetor de forças nodais equivalentes, assemblagem, introdução das condições de apoio, equação de equilíbrio, etc.

No âmbito da Engenharia de Estruturas, o Método dos Elementos Finitos tem como finalidade a determinação do estado de tensão e de deformação de um elemento ou estrutura sujeita a ações exteriores. É, assim, útil recorrer a uma sucessão de análises e modificações das suas características, com o objetivo de se alcançar uma solução otimizada, quer em termos económicos, quer na verificação dos requisitos funcionais e regulamentares [2].

O meio contínuo de carácter estrutural é, em geral, muito complexo para ser analisado de forma exata, por isso, são adotadas hipóteses simplificadoras para criar um modelo matemático aproximado em relação ao sistema físico original. O comportamento estático, por exemplo, pressupõe que as forças são aplicas de forma suficientemente lenta de maneira a se poder desprezar as forças de inércia e de amortecimento. A linearidade material corresponde à adoção de uma relação linear entre tensões e deformações. A linearidade geométrica implica que os deslocamentos nas estruturas sejam suficientemente pequenos de forma que as equações de equilíbrio possam ser escritas na configuração indeformada. Deste modo, a escolha do tipo de análise é parte integrante do procedimento de modelação.

A modelação de um problema genérico que envolve meios contínuos, através da análise de partes discretas desses meios, em detrimento do todo dá-se o nome de discretização. Cada elemento finito e as leis que regem o seu comportamento contribuem para o conhecimento e a análise do problema global [21]. A passagem da escala de análise ao nível de cada elemento finito para a análise do todo dá-se o nome de assemblagem. Assim, na ótica do utilizador, a resolução de um problema complexo, ou sem solução analítica conhecida, pelo Método dos Elementos Finitos passa pela resolução sequencial e estruturada de vários problemas mais simples com solução matemática conhecida que, quando agrupados, conduzem a uma solução do problema global inicial.

O Método dos Elementos Finitos é um método aproximado e por este motivo a realização de uma simulação numérica é sempre um modo aproximado para resolver um problema complexo. Assim, a sua utilização deve ser limitada a problemas para os quais não existam abordagens ou soluções analíticas para a sua resolução. Quando se recorre ao Método dos Elementos Finitos para resolver problemas de engenharia é importante identificar as possíveis fontes de erro e estimar a sua magnitude. A qualidade e o rigor que é empregue na modelação do problema estrutural vai influenciar os resultados obtidos da análise.

2.2. ANÁLISE DE MEIOS DISCRETOS E MEIOS CONTÍNUOS

Correntemente a análise de estruturas é feita com recurso a modelos de sistemas discretos. Este tipo de análise carateriza-se por fazer uso de um conjunto de elementos que modelam o domínio em estudo e que se encontram unidos pelas suas extremidades. Estes elementos podem estar sujeitos a um conjunto diversificado de ações exteriores. São exemplo de sistemas discretos as estruturas articuladas, as estruturas reticuladas e as grelhas.

As estruturas articuladas caraterizam-se pelo facto dos seus membros transmitirem, unicamente, esforços axiais. A ligação dos membros das estruturas articuladas é feita por meio de articulações daí a designação de estrutura articulada. Este facto leva a que haja liberdade de rotação nas articulações. Uma estrutura reticulada caracteriza-se pelo facto dos nós das suas barras estarem ligados de forma rígida ao contrário das estruturas articuladas em que estes estão ligados por meio de articulações. Os membros de uma estrutura reticulada podem ser submetidos a forças axiais, como os membros das estruturas articuladas, bem como a esforços transversos e a momentos fletores, como os elementos de vigas. As relações de rigidez para os elementos de uma estrutura reticulada podem ser convenientemente obtidas pela combinação das relações de rigidez da estrutura articulada com os membros de viga. Uma grelha é uma estrutura onde as forças são aplicadas perpendicularmente ao seu plano, ao contrário do que acontece com os pórticos planos onde as forças são aplicadas no seu plano.

A utilização do Método dos Elementos Finitos não se cinge à análise de sistemas discretos como os referidos anteriormente. De facto, a sua generalização permite cobrir a análise do comportamento de meios contínuos. A solução matemática da maioria dos problemas na Engenharia de Estruturas passa pela determinação de um conjunto de incógnitas que representam variáveis físicas [21]. Estas variáveis são os graus de liberdade do problema.

Na sua essência, a análise de uma estrutura pelo Método dos Elementos Finitos é uma aplicação do Método dos Deslocamentos. Em pórticos, treliças e grelhas, os elementos são barras ligadas pelos nós. Estes elementos de barra são considerados como sendo unidimensionais. Elementos finitos bidimensionais ou tridimensionais são utilizados na análise de paredes, lajes, reservatórios e estruturas de massa. Sendo a solução obtida pelo Método dos Elementos Finitos aproximada, a convergência para a solução exata é conseguida quando o número de parâmetros desconhecidos é aumentado. Estes parâmetros desconhecidos correspondem aos deslocamentos nodais da malha de elementos finitos. Por outras palavras, quando é utiliza uma malha de elementos finitos mais refinada, mais deslocamentos nodais desconhecidos são envolvidos sendo, deste modo, alcançada uma maior precisão.

A formulação matemática do Método do Elementos Finitos aplicada a meios contínuos consiste em substituir o integral sobre um domínio complexo por um somatório de integrais estendidos a subdomínios de geometria mais simples [2]. Se for possível calcular todos os integrais estendidos aos subdomínios, no final, basta efetuar o somatório para se obter o valor do integral estendido a todo o domínio.

2.3. METODOLOGIA DE CÁLCULO

A aplicação do Método dos Elementos Finitos na análise estrutural consiste na criação de um modelo que aproxime as propriedades geométricas e mecânicas da estrutura real com um número infinito de graus de liberdade por um modelo com um número finito de graus de liberdade.

Simplificadamente, os passos necessários para realizar uma análise pelo Método dos Elementos Finitos são os seguintes:

1. Seleção do modelo estrutural adequado à análise pretendida e definição das propriedades dos materiais;
2. Divisão do domínio da estrutura recorrendo a linhas, superfícies ou volumes, gerando uma malha de elementos finitos;
3. Cálculo das matrizes de rigidez elementares e dos vetores de forças nodais equivalentes para cada elemento finito;
4. Assemblagem das contribuições de cada elemento finito na matriz de rigidez e vetor de forças globais da estrutura;
5. Introdução das condições de fronteira e resolução do sistema de equações;
6. Cálculo das extensões e tensões para cada elemento finito a partir do conhecimento dos deslocamentos nodais;
7. Interpretação e apresentação dos resultados graficamente;
8. Avaliação da necessidade de considerar alterações ao modelo ou aumento do refinamento da malha de elementos finitos de modo a precisar as zonas onde existam variações de tensões acentuadas.

A sequência de passos apresentada pretende mostrar o encadeamento usado numa análise por elementos finitos. Ao longo dos capítulos 2 a 6 são pormenorizados cada um destes passos de modo a no final estarem reunidas as condições necessárias à implementação computacional do método. Importa destacar a importância das decisões que se tomam ao longo do cumprimento destes passos, pois, os resultados obtidos da análise vêm afetados por essas decisões.

2.4. ABORDAGEM DE ENGENHARIA

2.4.1. SELEÇÃO DO MODELO ESTRUTURAL

O primeiro passo para resolução de um problema é a identificação do próprio problema. Para se poder analisar corretamente um problema é necessário identifica-lo corretamente. A correta identificação de todos os fenómenos físicos envolvidos que influenciam o comportamento da estrutura, a natureza estática ou dinâmica, as propriedades dos materiais, entre outros, são fundamentais para a escolha adequada do modelo estrutural. Quando se pretende modelar um problema por elementos finitos é necessário escolher, por exemplo, se a análise vai ser bidimensional ou tridimensional. A escolha tem implicações na quantidade de cálculos realizados.

Os modelos, no âmbito de uma análise por elementos finitos, costumam ser classificados como conceptuais, estruturais ou computacionais [13].

Os modelos computacionais são aplicados aos modelos conceptuais de um problema real e não ao problema real em si [13]. Um modelo conceptual pode ser desenvolvido no seguimento da compreensão da natureza física de um problema. Um modelo conceptual deve excluir os detalhes supérfluos e incluir todas as características relevantes do problema em análise de modo a descrever a realidade com precisão adequada.

Um modelo conceptual para o estudo de uma estrutura deve incluir todos os dados necessários para a sua representação e análise. Depois de se selecionar um modelo conceptual adequado para uma estrutura, o passo seguinte para o seu estudo é a definição de um modelo estrutural. Um modelo estrutural deve incluir a descrição geométrica da estrutura por meio dos seus componentes geométricos, a expressão matemática das leis físicas básicas que regem o comportamento da estrutura e a especificação das propriedades dos materiais e das cargas que atuam sobre a estrutura [13]. É perfeitamente possível que o mesmo modelo conceptual de uma estrutura possa ser analisado utilizando diferentes modelos estruturais, dependendo do rigor e/ou simplicidade procurada na análise.

O passo seguinte na sequência da análise estrutural é a escolha de um método numérico. A aplicação do Método dos Elementos Finitos é, normalmente, feita com a sua implementação num programa de computador. No Capítulo 9 é apresentado o programa desenvolvido no âmbito desta dissertação que possibilita a modelação e a análise estrutural pelo Método dos Elementos Finitos.

2.4.2. DISCRETIZAÇÃO DO MODELO

A discretização de um modelo em elementos finitos constitui uma das primeiras tarefas na análise por elementos finitos. A figura 2.1 mostra a discretização de uma viga de espessura *h* com recurso a elementos finitos retangulares.

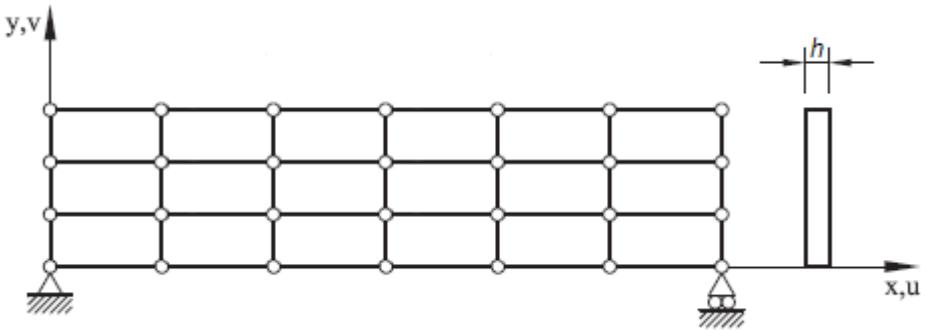


Fig.2.1 – Discretização de uma viga com elementos finitos retangulares.

Todo o conhecimento sobre problemas físicos, elementos finitos e algoritmos de resolução contribui para a experiência de modelação. A principal dificuldade enfrentada nesta etapa consiste em não entender a ação física e as condições fronteira da estrutura real, bem como as limitações da teoria aplicável, para criar um modelo adequado. Outra dificuldade é de não compreender o comportamento dos vários elementos finitos para se selecionar os adequados à resolução do problema. O resultado pode ser uma má pormenorização do problema. Um modelo que não incorpore as características importantes do problema físico, uma discretização imprópria dos carregamentos ou uma introdução de condições de apoio inadequadas conduz a que os resultados obtidos não tenham correspondência com os observados na realidade.

Os utilizadores de programas de modelação por elementos finitos estão habituados à utilização de ferramentas que geram automaticamente as malhas para o domínio em estudo. Contudo, a qualidade da malha depende da qualidade dos algoritmos programados para a gerar. Logo, estão dependentes do conhecimento que o programador tem sobre a modelação com elementos finitos. Portanto, o procedimento de modelação não deverá nunca ser menosprezado independentemente da sua dimensão ou importância. Face à importância que a modelação por elementos finitos apresenta esta costuma ser considerada uma arte [3].

É adequado que uma malha de elementos finitos seja composta por elementos finitos pouco distorcidos, com dimensões semelhantes e refinada o suficiente para a obtenção de resultados com qualidade aceitável. A melhor precisão está associada à modelação com muitos elementos finitos e de ordem de interpolação mais elevada. Igual precisão pode ser obtida com elementos de baixa ordem recorrendo a um elevado refinamento da malha. A escolha do elemento finito é também dependente do problema em análise. Um elemento ou malha que funcione bem numa situação pode funcionar mal noutra. A escolha está dependente do conhecimento sobre cada elemento finito e da compreensão física do problema.

O principal problema associado ao custo computacional de uma análise é que as capacidades computacionais são limitadas. Isto pode não ser evidente na análise de pequenos problemas mas torna-se relevante nos grandes problemas. Deve-se procurar usar modelos simples de modo a que os possíveis erros sejam mais facilmente detetáveis. É uma boa prática usar dois modelos na análise. Um modelo mais simples que fornece resultados aproximados que depois pode ser usado para orientar a construção de um novo modelo mais refinado e validar os seus resultados.

Existem algumas orientações a seguir no processo de modelação de uma estrutura com elementos finitos, nomeadamente:

- Procurar incluir toda a estrutura real no modelo;
- Modelar adequadamente os limites curvos do domínio;
- Usar elementos que modelam o campo de deslocamentos real;
- Evitar o uso de elementos desproporcionados ou distorcidos;
- Usar malhas regulares e sem variações bruscas de refinamento;
- Modelar adequadamente as cargas;
- Usar condições de apoio que simulem as condições reais;
- Aumentar o refinamento nas zonas onde se prevejam grandes variações nos gradientes de tensões.

A escolha dos elementos a usar tem influência na precisão dos resultados obtidos. Há elementos que não são capazes de modelar alguns modos de deformação ou distorções [6]. Elementos diferentes têm diferentes sensibilidades para modelar distorções. Deve-se procurar que a seleção dos elementos a usar esteja em conformidade com o tipo de deformação esperada. Os elementos finitos vizinhos do mesmo tipo devem ter uma geometria semelhante e as transições entre elementos diferentes devem ser graduais.

As condições de apoio do modelo são tão relevantes como a própria malha de elementos finitos. Não adianta recorrer a refinamentos elevados se os apoios usados no modelo não modelem as condições de apoio reais da estrutura. Normalmente, em teoria, os apoios estruturais são idealizados como completamente rígidos ou como articulados. Os apoios reais, em geral, situam-se entre um apoio rígido e um apoio articulado. Esta diferença pode levar a que exista uma alteração significativa da distribuição de esforços na estrutura. No final de uma análise deve-se procurar verificar a consistência dos resultados obtidos.

2.4.3. Sistema de Equações Elementares

Assumindo que o material de que é constituído o modelo de elementos finitos é homogéneo e tem um comportamento linear elástico, as forças nodais dependem de forma direta e proporcional dos deslocamentos dos nós associados ao desenvolvimento de deformações.

A relação entre o vetor de forças nodais equivalentes e o vetor de deslocamentos nodais, que se assume ser de proporcionalidade direta, pode ser expressa através da relação matricial

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} & \cdots & k_{1n} \\ k_{21} & k_{22} & k_{23} & \cdots & k_{2n} \\ k_{31} & k_{32} & k_{33} & \cdots & k_{3n} \\ \vdots & \vdots & \vdots & & \vdots \\ k_{n1} & k_{n2} & k_{n3} & \cdots & k_{nn} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{Bmatrix} \tag{2.1}$$

onde *n* representa o número total de graus de liberdade do elemento finito. Colocando (2.1) numa forma mais compacta,

$$\boldsymbol{k}\,\boldsymbol{a} = \boldsymbol{f} \tag{2.2}$$

em que $\boldsymbol{k}$ designa a matriz de rigidez elementar, $\boldsymbol{a}$ o vetor de deslocamentos e $\boldsymbol{f}$ o vetor de forças nodais equivalentes. Determinadas todas as matrizes de rigidez elementares para o problema em estudo, torna-se necessário agrupá-las de forma a construir a matriz de rigidez global do problema. Esta operação é frequentemente designada por assemblagem. Uma operação semelhante tem de ser efetuada com os vetores de forças nodais equivalentes dos elementos finitos.

2.4.4. ASSEMBLAGEM

Uma análise por elementos finitos de um problema genérico que envolve meios contínuos, através da análise de partes discretas desses meios, implica que se proceda à junção dos contributos de cada uma das partes de modo a se conhecer o comportamento do problema global. A passagem da análise ao nível de cada elemento finito para a análise do todo dá-se o nome de assemblagem. Com a operação de assemblagem é construída a matriz de rigidez e o vetor de forças nodais globais para todo o domínio do problema. Note-se que na essência desta operação está o facto de que uma qualquer força externa aplicada num determinado nó da malha é partilhada por todos os elementos finitos que têm esse nó em comum.

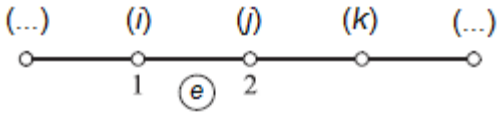


Fig.2.2 – Malha constituída por elementos finitos unidimensionais.

Um vez construídas as matrizes de rigidez elementares e os vetores de forças nodais equivalentes associados às cargas aplicadas nos elementos finitos procede-se à assemblagem das suas contribuições na matriz de rigidez e no vetor de forças nodais globais. Para realizar a assemblagem é necessário definir a conetividade da malha de elementos finitos, isto é, a relação existente entre a numeração dos graus de liberdade dos elementos finitos e a numeração dos graus de liberdade da malha. Para isso é construída, para cada elemento finito, uma matriz de assemblagem do tipo

$$\left[\begin{array}{c|c} 1 & i \\ 2 & j \\ 3 & k \\ \vdots & \vdots \\ n & r \end{array}\right] \tag{2.3}$$

onde a primeira coluna contém a numeração local dos *n* graus de liberdade do elemento finito e a segunda coluna a correspondência destes com a numeração global dos graus de liberdade da malha de elementos finitos.

O procedimento de assemblagem não é mais do que o espalhamento dos termos da matriz de rigidez elementar e do vetor de forças nodais equivalentes do elemento finito pelas posições globais que estes

ocupam na matriz de rigidez e vetor de forças nodais globais, respetivamente. A assemblagem de elementos finitos é em tudo idêntico à assemblagem das contribuições das barras no Método dos Deslocamentos. De acordo com o exposto e atendendo à figura 2.2, anteriormente apresentada, o número total de graus de liberdade de uma malha de elementos finitos é menor ou igual que o número total de graus de liberdade dos elementos finitos. Note-se que no momento da assemblagem as contribuições dos vários elementos finitos que partilham o mesmo nó são somadas.

Depois de realizada a operação de assemblagem obtém-se a representação física do comportamento de uma estrutura na forma de um sistema global de equações. Este sistema de N equações, onde N representa o número total de graus de liberdade da malha de elementos finitos, pode ser escrito na forma genérica

$$\begin{array}{c} \\ 1 \\ \vdots \\ i \\ j \\ \vdots \\ N \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & \cdots & i & j & \cdots & N \end{array} \\ \begin{bmatrix} & & \vdots & \vdots & & \\ \cdots & & k_{11}^e & k_{12}^e & \cdots & \\ \cdots & & k_{21}^e & k_{22}^e & \cdots & \\ & & \vdots & \vdots & & \end{bmatrix} \end{array} \begin{Bmatrix} \vdots \\ d_i \\ d_j \\ \vdots \end{Bmatrix} = \begin{Bmatrix} \vdots \\ f_1^e \\ f_2^e \\ \vdots \end{Bmatrix} \tag{2.4}$$

onde k_{pq}^e e f_p^e, $\forall_{p,q=1,\cdots,n}$, correspondem aos termos da matriz de rigidez elementar e do vetor de forças nodais equivalentes, respetivamente, do elemento finito e. A matriz de rigidez global é uma matriz quadrada com dimensão correspondente ao número total de graus de liberdade do problema, enquanto, os vetores de deslocamentos e forças nodais têm N linhas. Colocando o sistema de equações (2.4) numa forma mais compacta,

$$\boldsymbol{K}\,\boldsymbol{d} = \boldsymbol{F}\,. \tag{2.5}$$

2.4.5. Condições de Fronteira e Carregamentos

Na forma como o sistema de equações (2.4) é apresentado não são considerados os apoios e fixações do problema e nem as condições particulares de carregamento.

O vetor $\boldsymbol{F}$ é constituído pelos carregamentos que atuam no interior e/ou na fronteira de cada elemento finito com os carregamentos atuantes nos nós da malha de elemento finitos. Os carregamentos concentrados e distribuídos atuantes em cada elemento finito são considerados no cálculo dos vetores de forças nodais equivalentes. Os carregamentos aplicados nos nós da malha são adicionados diretamente ao vetor de forças nodais globais.

Atendendo a que os graus de liberdade do sistema são as componentes de deslocamento dos nós da estrutura, sem a substituição de um número mínimo de deslocamentos prescritos, para prevenir os movimentos de corpo rígido da estrutura, não se pode resolver o sistema de equações. Isto advém do facto dos deslocamentos não poderem ser unicamente determinados pelo vetor $\boldsymbol{F}$ se a este não corresponder uma situação de equilíbrio estático. Matematicamente é interpretado com a singularidade da matriz $\boldsymbol{K}$. Assim, para que o sistema de equações tenha uma solução admissível é necessário definir deslocamentos prescritos correspondentes à fixação da estrutura.

2.4.6. Resolução do Sistema de Equações

Após a introdução das condições de apoio e a prescrição dos deslocamentos, o sistema de equações (2.5) pode ser escrito na forma

$$\boldsymbol{K}\,\boldsymbol{d} = \boldsymbol{F} + \boldsymbol{R} \tag{2.6}$$

em que o vetor $\boldsymbol{R}$ armazena a informação relativa às condições de apoio da estrutura. Para facilitar a sua resolução do ponto de vista computacional é conveniente organizá-lo de modo a separar os termos associados aos graus de liberdade não prescritos dos graus de liberdade prescritos, isto é, colocando-o na forma

$$\begin{bmatrix} \boldsymbol{K_{LL}} & \boldsymbol{K_{LF}} \\ \boldsymbol{K_{FL}} & \boldsymbol{K_{FF}} \end{bmatrix} \begin{Bmatrix} \boldsymbol{d_L} \\ \boldsymbol{d_F} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{F_L} \\ \boldsymbol{F_F} \end{Bmatrix} + \begin{Bmatrix} \boldsymbol{0} \\ \boldsymbol{r} \end{Bmatrix} \tag{2.7}$$

onde o índice *L* se refere aos graus de liberdade não prescritos ou livres e o índice *F* aos graus de liberdade prescritos ou fixos. O vetor $\boldsymbol{r}$ corresponde às reações de apoio da estrutura que até esta fase são uma incógnita do problema como, também, os deslocamentos nodais associados aos nós não prescritos. Este novo sistema de equações pode ser resolvido do seguinte modo.

$$\boldsymbol{d_L} = \boldsymbol{K_{LL}}^{-1}(\boldsymbol{F_L} - \boldsymbol{K_{LF}}\,\boldsymbol{d_F}) \tag{2.8a}$$

$$\boldsymbol{r} = \boldsymbol{K_{FL}}\,\boldsymbol{d_L} + \boldsymbol{K_{FF}}\,\boldsymbol{d_F} - \boldsymbol{F_F} \tag{2.8b}$$

Este procedimento genérico permite determinar o valor dos deslocamentos nodais associados aos nós não prescritos e sucessivamente o valor das reações de apoio na estrutura. Se não existirem assentamentos de apoio na estrutura, o vetor $\boldsymbol{d_F}$ é nulo, pelo que as expressões (2.8) são simplificadas nas seguintes expressões.

$$\boldsymbol{d_L} = \boldsymbol{K_{LL}}^{-1}\boldsymbol{F_L} \tag{2.9a}$$

$$\boldsymbol{r} = \boldsymbol{K_{FL}}\,\boldsymbol{d_L} - \boldsymbol{F_F} \tag{2.9b}$$

2.4.7. Verificação e Validação dos Resultados

De forma a se ter confiança nos resultados obtidos é conveniente efetuar a sua validação. A validação assenta fundamentalmente em duas partes distintas que são realizadas em fases diferentes e normalmente por pessoas diferentes. A primeira parte consiste em verificar o código de modo a estabelecer a confiança que tudo foi corretamente programado. A segunda parte consiste em verificar os cálculos de modo a estabelecer a confiança no modelo criado para a análise.

Como a discretização introduz uma aproximação [13], existem duas fontes de erro desde o início, ou seja, o erro de modelação e o erro de discretização. O primeiro pode ser reduzido, melhorando os modelos conceptuais e estruturais que descrevem o comportamento real da estrutura. O erro de discretização pode ser reduzido pelo uso de uma malha mais refinada ou usando elementos finitos de ordens superiores. Os computadores também introduzem erros numéricos associados à sua limitação para representar os dados com elevada precisão. O erro numérico apesar de pequeno é somado aos erros associados à modelação e à discretização.

2.5. Princípio dos Trabalhos Virtuais

Define-se como virtual algo que não é real, portanto, imaginário. Um deslocamento virtual ou uma força virtual são, respetivamente, um deslocamento imaginário ou uma força imaginária, arbitrariamente impostos sobre um sistema estrutural.

Considere-se o corpo representado na figura 2.3 sujeito a um conjunto de forças de volume, de superfície, de linha e concentradas que lhe provocam deformação. Com base no seu estado de equilíbrio

estático, a configuração do corpo é modificada por um conjunto de deslocamentos muito pequenos e compatíveis com as condições de fronteira.

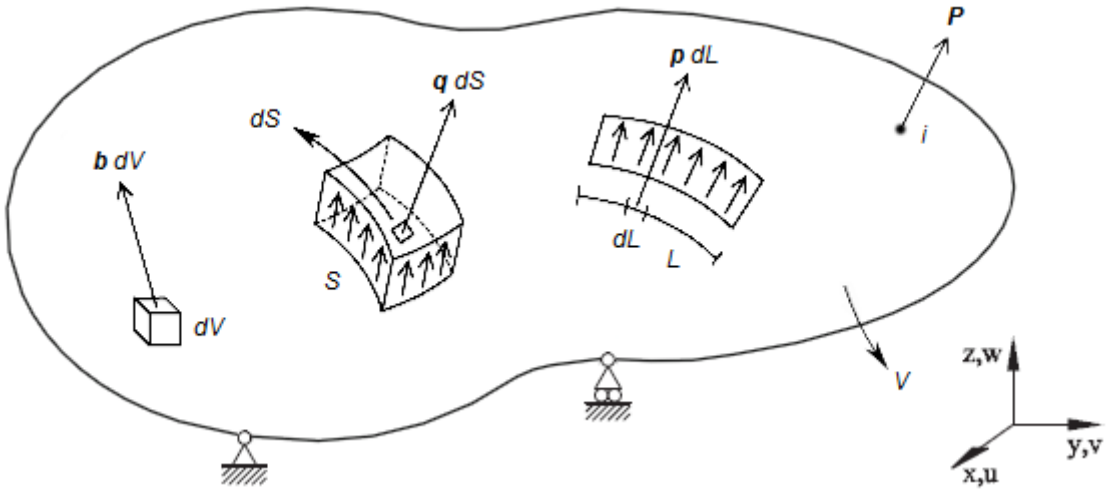


Fig.2.3 – Corpo sujeito a diversos tipos de ações exteriores (Adaptado de [2]).

O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{2.10}$$

Para um corpo tridimensional contínuo representado pela figura 2.3, o Princípio dos trabalhos Virtuais pode ser escrito na forma

$$\int_V \delta\boldsymbol{\varepsilon}^T \boldsymbol{\sigma}\, dV = \int_V \delta\mathbf{u}^T \boldsymbol{b}\, dV + \int_S \delta\mathbf{u}^T \boldsymbol{q}\, dS + \int_L \delta\mathbf{u}^T \boldsymbol{p}\, dL + \delta\mathbf{u}^T \boldsymbol{P} \tag{2.11}$$

onde $\delta\boldsymbol{\varepsilon}$ indica as deformações virtuais, $\boldsymbol{\sigma}$ as tensões reais aproximadas e $\delta\mathbf{u}$ os deslocamentos virtuais. Na formulação do Método dos Elementos Finitos, o campo de deformações é interpolado a partir dos deslocamentos nodais com a expressão

$$\boldsymbol{\varepsilon} = \boldsymbol{B}\, \boldsymbol{a} \tag{2.12}$$

onde $\boldsymbol{B}$ é a matriz de deformação e $\boldsymbol{a}$ o vetor de deslocamentos nodais. Quando esta equação se refere aos deslocamentos virtuais e correspondentes deformações virtuais, tem-se

$$\delta\boldsymbol{\varepsilon} = \boldsymbol{B}\, \delta\boldsymbol{a} \tag{2.13}$$

que é equivalente a

$$\delta\boldsymbol{\varepsilon}^T = \delta\boldsymbol{a}^T \boldsymbol{B}^T. \tag{2.14}$$

As tensões, para um material isotrópico com comportamento linear elástico, podem ser obtidas através da relação constitutiva

$$\boldsymbol{\sigma} = \boldsymbol{D}\, \boldsymbol{\varepsilon} \tag{2.15}$$

onde a matriz $\boldsymbol{D}$ é designada por matriz de elasticidade e na isotropia é função das propriedades elásticas do material, isto é, do módulo de elasticidade, E, e do coeficiente de Poisson, v. Substituindo (2.12) em (2.15) obtém-se

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{a}\,. \tag{2.16}$$

Na formulação do Método dos Elementos Finitos considera-se que a interpolação do campo de deslocamentos a partir dos deslocamentos nodais é efetuada com a expressão

$$\mathbf{u} = \boldsymbol{N}\,\boldsymbol{a} \tag{2.17}$$

em que $\boldsymbol{N}$ agrupa as funções de forma que dependem do elemento finito considerado e $\boldsymbol{a}$ contém os deslocamentos nodais cujo número depende do número total de graus de liberdade que o elemento finito apresenta. Quando esta equação se refere aos deslocamentos virtuais, tem-se

$$\delta\mathbf{u} = \boldsymbol{N}\,\delta\boldsymbol{a} \tag{2.18}$$

que é equivalente a

$$\delta\mathbf{u}^T = \delta\boldsymbol{a}^T\boldsymbol{N}^T. \tag{2.19}$$

Substituindo em (2.11) as equações (2.14), (2.16) e (2.19), o Princípio dos Trabalhos Virtuais é expresso na forma

$$\int_V \delta\boldsymbol{a}^T\boldsymbol{B}^T\boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{a}\,dV = \int_V \delta\boldsymbol{a}^T\boldsymbol{N}^T\boldsymbol{b}\,dV + \int_S \delta\boldsymbol{a}^T\boldsymbol{N}^T\boldsymbol{q}\,dS + \int_L \delta\boldsymbol{a}^T\boldsymbol{N}^T\boldsymbol{p}\,dL + \delta\boldsymbol{a}^T\boldsymbol{N}^T\boldsymbol{P}\,. \tag{2.20}$$

Uma vez que os vetores de deslocamentos nodais virtuais e de deslocamentos nodais são constantes, podem passar-se para fora do integral. Assim, após alguma manipulação algébrica, obtém-se

$$\delta\boldsymbol{a}^T\int_V \boldsymbol{B}^T\boldsymbol{D}\,\boldsymbol{B}\,dV\,\boldsymbol{a} = \delta\boldsymbol{a}^T\left(\int_V \boldsymbol{N}^T\boldsymbol{b}\,dV + \int_S \boldsymbol{N}^T\boldsymbol{q}\,dS + \int_L \boldsymbol{N}^T\boldsymbol{p}\,dL + \boldsymbol{N}^T\boldsymbol{P}\right). \tag{2.21}$$

Considerando que, para além destes serem constantes, o campo de deslocamentos nodais virtuais é sempre não-nulo nos domínios considerados, então da equação anterior pode concluir-se que

$$\int_V \boldsymbol{B}^T\boldsymbol{D}\,\boldsymbol{B}\,dV\,\boldsymbol{a} = \int_V \boldsymbol{N}^T\boldsymbol{b}\,dV + \int_S \boldsymbol{N}^T\boldsymbol{q}\,dS + \int_L \boldsymbol{N}^T\boldsymbol{p}\,dL + \boldsymbol{N}^T\boldsymbol{P}\,. \tag{2.22}$$

Comparando esta equação com a relação de rigidez que é utilizada no Método dos Deslocamentos, conclui-se que

$$\boldsymbol{k}\,\boldsymbol{a} = \boldsymbol{f}\,. \tag{2.23}$$

Este procedimento genérico pode facilmente ser utilizado para determinar os sistemas de equações elementares de todos os elementos finitos.

2.6. Requisitos para a Convergência da Solução

2.6.1. Condição de Continuidade

Os deslocamentos devem ser contínuos dentro de cada elemento finito. Esta condição é automaticamente satisfeita usando aproximações polinomiais para o campo de deslocamentos. É necessário também que exista continuidade do campo de deslocamentos na fronteira de cada um dos elementos que lhe são adjacentes na malha de elementos finitos.

Elementos que satisfazem a condição de continuidade do campo de deslocamentos são denominados elementos finitos conformes. No entanto, em alguns casos particulares, tais como, em alguns elementos à flexão baseados na teoria de Kirchhoff, esta condição não é satisfeita. Estes elementos, em que a condição de continuidade não é satisfeita, são denominados elementos finitos não conformes. Elementos

não conformes podem convergir para a solução exata se passarem no *patch test*. Em algumas ocasiões estes elementos finitos fornecem boas soluções com malhas relativamente grosseiras.

2.6.2. Condição de Derivabilidade

As derivadas das funções de forma devem existir até às derivadas que aparecem nos elementos da matriz de deformação $\boldsymbol{B}$. Por exemplo, para uma matriz que contenha derivadas de primeira ordem, as funções forma devem ser, pelo menos, de primeira ordem.

Uma função contínua com a primeira derivada descontínua é designada por função com continuidade C^0. Por outro lado, uma função contínua com primeira derivada contínua e segunda derivada descontínua possui continuidade C^1. Desta forma, uma função possui continuidade C^n quando todas as suas derivadas até à ordem n são contínuas, mas a sua derivada de ordem $n + 1$ é descontínua.

2.6.3. Condição de Integrabilidade

No caso de uma função ter continuidade C^0, a sua primeira derivada apresentará descontinuidades pontuais, correspondendo a uma função de continuidade C^{-1}. Contudo, tais descontinuidades não impedem que esta derivada seja integrável em todo o seu domínio.

A integrabilidade de funções de continuidade C^{-1} torna admissível a utilização de funções com continuidade C^0. Nesta situação, tal significa que o campo de deslocamentos tem continuidade C^0, pelo que o campo de deformação, definido com base na primeira derivada do deslocamento, possui continuidade C^{-1}. Devido à simplicidade das funções de continuidade C^0, este tipo de funções é usualmente utilizado na resolução de problemas pelo Método dos Elementos Finitos [21].

2.6.4. Condição de Corpo Rígido

As funções de forma devem ser capazes de representar um movimento de translação sem deformação, isto é, um movimento de corpo rígido. Isto significa que todos os pontos pertencentes ao elemento finito têm um deslocamento igual ao dos nós. Esta condição física é satisfeita para um único elemento finito, se a soma das suas funções de forma avaliadas em qualquer ponto do elemento finito for igual à unidade, isto é, se

$$N_1(x) + N_2(x) + N_3(x) + \cdots + N_n(x) = 1 \tag{2.24}$$

onde $N_i(x)$ é a função de forma do elemento finito para o nó i.

2.6.5. Condição de Deformação Constante

Quando se impõem deslocamentos nodais correspondentes a um estado de deformação constante, o campo de deslocamentos deve originar um campo de deformações constantes no interior do elemento finito. O critério de deformação constante incorpora o requisito de corpo rígido. A função de deslocamento tem que ser de tal modo que se os deslocamentos nodais forem compatíveis com um campo de deformação constante, o estado de deformação obtido no interior do elemento finito deve ser também constante.

3
FORMULAÇÃO DE ELEMENTOS FINITOS

3.1. ELEMENTOS FINITOS UNIDIMENSIONAIS

3.1.1. TENSÃO

A generalidade dos elementos estruturais aplicados nas estruturas de edifícios são do tipo barra e muitos destes estão submetidos a cargas axiais aplicadas nas extremidades. O estudo dos elementos finitos unidimensionais prende-se com a necessidade de apresentar uma formulação que permita o estudo destes elementos estruturais.

De modo a se poder calcular a distribuição de tensão que atua na secção transversal da barra são adotadas as seguintes hipóteses simplificadoras [4]:

- A barra permanece reta tanto antes como depois da aplicação da carga, e, além disso, a secção transversal permanece plana durante a deformação;
- A carga é aplicada segundo o eixo longitudinal da barra e o material é homogéneo e isotrópico.

Considere-se uma barra reta de secção transversal constante e sujeita a um carregamento axial centrado nas suas extremidades. Como a barra sofre uma variação de comprimento após aplicação da carga, a deformação axial é calculada recorrendo à expressão

$$\varepsilon = \frac{\Delta L}{L} \tag{3.1}$$

em que ΔL designa a variação de comprimento e L o comprimento indeformado da barra. Pela definição de tensão,

$$\sigma = \frac{P}{A} \tag{3.2}$$

em que P corresponde à carga axial e A à área da sua secção transversal da barra. A partir do valor da deformação é possível calcular o valor da tensão em qualquer secção transversal da barra. Admitindo que o material da barra apresenta um comportamento linear elástico é possível definir a relação entre tensões e deformações

$$\sigma = E\,\varepsilon \tag{3.3}$$

em que E é designado por módulo de elasticidade do material. Através da formulação de elementos finitos é possível usar esta relação nas situações em que as extensões variam ao longo da barra.

3.1.2. PRINCÍPIO DOS TRABALHOS VIRTUAIS

O estudo dos elementos finitos unidimensionais justifica-se com a necessidade de introduzir as técnicas usadas para formular os diversos elementos finitos, nomeadamente, os elementos finitos bidimensionais e tridimensionais.

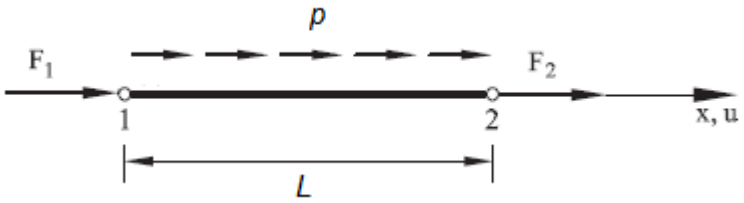


Fig.3.1 – Elemento finito unidimensional sujeito a uma carga axial.

O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{3.4}$$

Para o caso de um elemento finito unidimensional definido segundo o eixo *Ox*, de comprimento *L* e sujeito a uma carga axial *p* uniformemente distribuída ao longo do seu comprimento, pelo Princípio dos Trabalhos Virtuais, tem-se

$$\int_V \delta\varepsilon^T \sigma \, dV = \int_L \delta \mathrm{u}^T p \, dL \, . \tag{3.5}$$

Atendendo ao apresentado no Capítulo 2 sobre o Princípio dos Trabalhos Virtuais, a relação (3.5), após alguma manipulação algébrica, é equivalente a

$$\int_{-L/2}^{+L/2} \boldsymbol{B}^T E \, \boldsymbol{B} \, A \, dx \, \boldsymbol{a} = \int_{-L/2}^{+L/2} \boldsymbol{N}^T p \, dx \tag{3.6}$$

onde $\boldsymbol{B}$ é a matriz de deformação, $\boldsymbol{a}$ o vetor de deslocamentos nodais e $\boldsymbol{N}$ agrupa as funções de forma que dependem do elemento finito considerado. Colocando (3.6) numa forma mais compacta,

$$\boldsymbol{k}\,\boldsymbol{a} = \boldsymbol{f} \, . \tag{3.7}$$

3.1.3. FUNÇÕES DE FORMA

As funções de forma do tipo polinomial para um elemento finito de *n* nós podem ser determinadas recorrendo ao polinómio de Lagrange. Conhecidas as *n* coordenadas dos *n* nós do elemento finito unidimensional, a expressão genérica do polinómio de Lagrange de grau *n* - 1, associado ao nó *i*, é

$$N_i(x) = \prod_{j=1\ (j\neq i)}^{n} \frac{(x - x_j)}{(x_i - x_j)} \tag{3.8}$$

em que x_i e x_j correspondem à coordenada associada ao nó *i* e ao nó *j*, respetivamente. Os parâmetros $N_\mathrm{i}(x)$ designam-se por funções de forma para o nó *i* e interpolam dentro de cada elemento finito o deslocamento do nó *i*. Uma das características das funções de forma é de valerem um no nó *i* e zero em todos os outros nós do elemento finito. Note-se que recorrendo a um polinómio de grau *n* o respetivo elemento finito tem *n* + 1 nós.

3.1.4. CAMPO DE DESLOCAMENTOS

Considere-se o elemento finito unidimensional com *n* nós representado na figura 3.2, de comprimento *L*, área da secção transversal *A*, módulo de elasticidade *E* e sujeito a um carregamento axial *p* uniformemente distribuído ao longo de todo o seu comprimento.

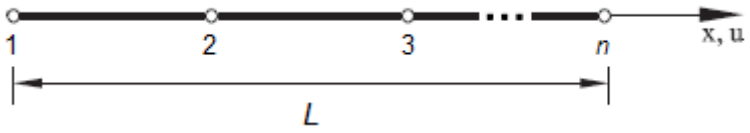


Fig.3.2 – Elemento finito unidimensional de *n* nós.

Sendo apenas considerado o eixo *Ox*, todos os deslocamentos ocorrem paralelamente a ele. Se se conhecer os deslocamentos nodais do elemento finito, o campo de deslocamentos $\mathrm{u}(x)$ do elemento finito pode ser interpolado a partir desses mesmos deslocamentos, ou seja,

$$\mathrm{u}(x) = N_1(x)a_1 + N_2(x)a_2 + \cdots + N_n(x)a_n \tag{3.9}$$

ou, de forma mais compacta,

$$\mathrm{u}(x) = \sum_{i=1}^{n} N_{\mathrm{i}}(x)a_i \tag{3.10}$$

onde os parâmetros a_i correspondem ao valor do deslocamento nodal no nó *i* do elemento finito.

3.1.5. CAMPO DE DEFORMAÇÕES

Sendo o campo de deslocamentos aproximado com recurso a funções interpoladoras, conhecido o campo de deslocamentos para o elemento finito é suficiente derivar as suas funções de forma e multiplicá-las pelos respetivos deslocamentos nodais.

$$\varepsilon = \frac{d\mathrm{u}}{dx} \tag{3.11}$$

Na forma matricial, derivando as *n* funções de forma do elemento finito com *n* nós, o campo de deformações é obtido pela multiplicação da matriz de deformação pelo vetor de deslocamentos nodais do elemento finito, isto é,

$$\varepsilon = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \cdots & \frac{\partial N_n}{\partial x} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{Bmatrix} \tag{3.12}$$

ou, de forma mais compacta,

$$\varepsilon = \boldsymbol{B}\,\boldsymbol{a} \tag{3.13}$$

sendo a matriz $\boldsymbol{B}$ designada por matriz de deformação ou, correntemente designada, matriz das derivadas das funções de forma.

3.1.6. CAMPO DE TENSÕES

As tensões, para um material isotrópico com comportamento linear elástico, podem ser obtidas através da relação constitutiva (3.3) que após a substituição de (3.13) resulta em

$$\sigma = E\,\boldsymbol{B}\,\boldsymbol{a} \tag{3.14}$$

sendo E designado por módulo de elasticidade.

Note-se que o cálculo da tensão num dado ponto do elemento finito está dependente do conhecimento dos deslocamentos nodais e da avaliação da matriz $\boldsymbol{B}$ nesse ponto. Como o campo de deslocamentos do elemento finito é determinado a partir dos deslocamentos nodais com recurso a um polinómio interpolador, verifica-se que existem pontos onde os resultados obtidos se aproximam da solução teórica esperada e pontos onde o resultado obtido se afasta.

3.1.7. Esforços Internos

Conhecida a tensão instalada num dado ponto do elemento finito, o esforço axial é obtido através da multiplicação do valor da tensão pela área da secção transversal do elemento finito. Se o esforço axial obtido é positivo, a secção transversal está tracionada, caso contrário, a secção transversal está comprimida.

3.1.8. Transformação de Coordenadas

Na forma como foi apresentada esta formulação de elementos finitos, estes só podem ser aplicados ao cálculo de estruturas em que todos os seus elementos estejam alinhados, ou seja, na situação em que todos os graus de liberdade dos elementos da estrutura sejam colineares.

Uma vez que os membros da estrutura podem ter diferentes direções, é necessário definir um sistema de eixos global para a estrutura. Como consequência desta definição de sistema de eixos, cada grau de liberdade de um elemento de barra é decomposto nos respetivos graus liberdade dependentes para cada direção do novo referencial. Esta decomposição dos graus de liberdade do elemento finito nas duas direções ortogonais do sistema de eixos global é realizada com recurso a uma matriz de transformação coordenadas.

A transformação da matriz de rigidez elementar do referencial local para o referencial global é feita com recurso à expressão

$$\boldsymbol{k_G} = \boldsymbol{T^T k\, T} \tag{3.15}$$

onde $\boldsymbol{T}$ é a matriz de transformação de coordenadas. Para o vetor de forças nodais equivalentes a mudança de referencial é realizada com a expressão seguinte.

$$\boldsymbol{f_G} = \boldsymbol{T^T f} \tag{3.16}$$

Determinadas as matrizes de rigidez elementares e os vetores de forças nodais equivalentes dos vários elementos finitos no referencial global procede-se ao seu espalhamento de forma a formar o sistema de equações globais. Resolvido o sistema de equações globais, para determinar os deslocamentos nodais no referencial local de cada elemento finito é necessário realizar um procedimento oposto ao apresentado para os passar do referencial local para o global.

Conhecidos os deslocamentos no sistema de eixos global da estrutura, os deslocamentos nodais para cada elemento finito no respetivo sistema de eixos local são calculados do seguinte modo.

$$\boldsymbol{a} = \boldsymbol{T}\,\boldsymbol{a_G} \tag{3.17}$$

A metodologia de transformação de coordenadas apresentada pode ser aplicada a qualquer quantidade e é válida para meios discretos e meios contínuos.

3.2. ELEMENTOS FINITOS BIDIMENSIONAIS

3.2.1. ESTADO PLANO DE TENSÃO/DEFORMAÇÃO

Na análise bidimensional de estruturas em estado plano de tensão é admitido que a componente de tensão ao longo da direção *Oz* e as tensões de corte nessa mesma direção são nulas. No caso de um estado plano de deformação, a componente de deformação e as distorções associadas à direção perpendicular a esse plano são consideradas nulas. Assim, é possível realizar uma análise modelando e discretizando apenas a secção representativa do problema com recurso, tipicamente, a elementos finitos bidimensionais de geometria triangular e/ou retangular.

Considera-se que um corpo está em estado plano de tensão quando [9]:

- Todas as cargas aplicadas atuam no plano médio do corpo, sendo ainda simétricas relativamente a este;
- A espessura do corpo é desprezável em relação às outras dimensões;
- Todas as condições de apoio são simétricas relativamente ao plano médio;
- As tensões normal e de corte segundo a direção *z* podem ser desprezadas.

Tipicamente, um corpo sujeito a um estado plano de deformação apresenta uma das dimensões significativamente superior às restantes, isto é, a espessura deixa de ser desprezável, bem como a componente de tensão σ_z. Aplicando o conceito de deformação infinitesimal da Mecânica dos Sólidos, as componentes cartesianas da extensão e da distorção são dadas por

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{3.18}$$

em que se considera $\gamma_{xz} = \gamma_{yz} = 0$. A componente de deformação segundo a direção perpendicular ao plano *Oxy*, isto é, a componente de deformação ε_z pode ser nula. Este caso caracteriza-se por representar um estado plano de deformação. Alternativamente, esta componente de deformação pode ser diferente de zero, representando um estado plano de tensão.

O campo de tensões associado ao campo de deformações pode ser escrito pelas seguintes componentes cartesianas.

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} \tag{3.19}$$

Para materiais que apresentem comportamento linear elástico, homogeneidade e isotropia, as tensões podem ser obtidas através da relação entre tensões e deformações

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{\varepsilon} \tag{3.20}$$

onde a matriz $\boldsymbol{D}$ é designada por matriz de elasticidade e é função das propriedades elásticas do material, isto é, do módulo de elasticidade, *E*, e do coeficiente de Poisson, ν. Num estado plano de tensão, para um material isotrópico com comportamento linear elástico,

$$\boldsymbol{D} = \begin{bmatrix} \dfrac{E}{1-\nu^2} & \dfrac{E\nu}{1-\nu^2} & 0 \\ \dfrac{E\nu}{1-\nu^2} & \dfrac{E}{1-\nu^2} & 0 \\ 0 & 0 & \dfrac{E}{2(1+\nu)} \end{bmatrix} \tag{3.21}$$

enquanto que, para um estado plano de deformação

$$\boldsymbol{D} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \dfrac{1-2\nu}{2} \end{bmatrix}. \tag{3.22}$$

3.2.2. PRINCÍPIO DOS TRABALHOS VIRTUAIS

Na figura 3.3 encontra-se representado um elemento finito retangular de quatro nós, de comprimento *a*, altura *b* e espessura *h*. Do ponto de vista de implementação computacional é conveniente numerar os nós do elemento finito no sentido anti-horário, numerando, em primeiro lugar, os nós que se encontram na sua fronteira e depois, se existirem, os nós interiores.

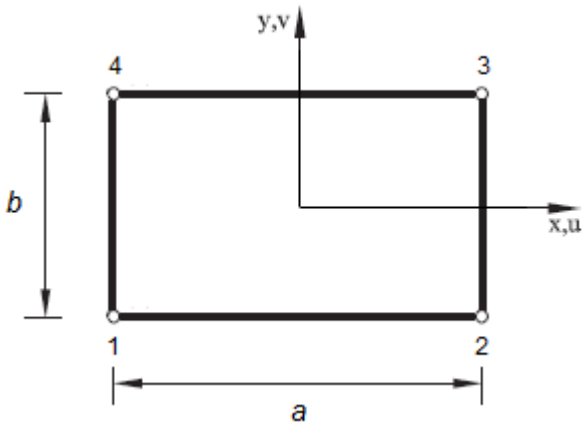


Fig.3.3 – Elemento finito retangular de quatro nós.

O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{3.23}$$

No caso geral, a expressão anterior que relaciona o trabalho interno com o trabalho externo pode ser expressa na forma

$$\int_V \delta\boldsymbol{\varepsilon}^T \boldsymbol{\sigma}\, dV = \int_V \delta\mathbf{u}^T \boldsymbol{b}\, dV \tag{3.24}$$

onde $\delta\boldsymbol{\varepsilon}$ indica as deformações virtuais, $\boldsymbol{\sigma}$ as tensões reais aproximadas e $\delta\mathbf{u}$ os deslocamentos virtuais no elemento finito. Atendendo ao apresentado no Capítulo 2 sobre o Princípio dos Trabalhos Virtuais, para um elemento finito bidimensional de espessura *h* e sujeito a um carregamento específico $\boldsymbol{q}$ por unidade de superfície no plano *Oxy*, a relação expressa em (3.24), após alguma manipulação algébrica, é equivalente a

$$\int_S \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, h\, dS\, \boldsymbol{a} = \int_S \boldsymbol{N}^T \boldsymbol{q}\, h\, dS\,. \tag{3.25}$$

Atendendo às dimensões do elemento finito representado na figura 3.3, os integrais de superfície, expressos anteriormente, para um elemento finito retangular sujeito a um carregamento por unidade de superfície no plano *Oxy*, são equivalentes a

$$\int_{-b/2}^{+b/2}\int_{-a/2}^{+a/2} \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, h\, dx\, dy\, \boldsymbol{a} = \int_{-b/2}^{+b/2}\int_{-a/2}^{+a/2} \boldsymbol{N}^T \boldsymbol{q}\, h\, dx\, dy\,. \tag{3.26}$$

Na situação em que o elemento finito bidimensional está sujeito a carregamentos distribuídos ao longo da sua fronteira, o segundo membro de (3.26) reduz-se a um integral de linha. Colocando (3.26) numa forma mais compacta,

$$\boldsymbol{k}\,\boldsymbol{a} = \boldsymbol{f}\,. \tag{3.27}$$

Note-se que a expressão apresentada em (3.26) apenas permite o cálculo da matriz de rigidez elementar e do vetor de forças nodais equivalentes para elementos finitos retangulares.

3.2.3. Funções de Forma

As funções de forma para os elementos finitos bidimensionais de geometria e distribuição de nós qualquer, em que x_i e y_i são as coordenadas do nó *i* e *n* representa o número total de nós, são determinadas através da expressão matricial

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ \vdots \end{Bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ \vdots & \vdots & & \vdots \end{bmatrix}^{-1} \begin{Bmatrix} 1 \\ x \\ y \\ \vdots \end{Bmatrix} \tag{3.28}$$

ou, de forma mais compacta,

$$\boldsymbol{N_V} = \boldsymbol{Q}^{-1}\boldsymbol{V}\,. \tag{3.29}$$

Os elementos finitos bidimensionais costumam ser divididos em duas famílias, isto é, em elementos da família lagrangeana e em elementos da família serendipity.

Os elementos finitos da família lagrangeana são quadriláteros que têm em comum o facto das suas funções de forma poderem ser obtidas através do polinómio de Lagrange e de possuírem o mesmo número de nós segundo cada uma das duas direções do sistema de coordenadas. A função de forma $N_i(x, y)$ para o nó *i* é determinada pela multiplicação dos polinómios interpoladores obtidos para cada uma das duas direções do sistema de coordenadas. Os elementos finitos bidimensionais da família lagrangeana têm p^2 nós, sendo *p* o número de nós de um bordo.

Os elementos finitos da família serendipity são caraterizados por apenas possuírem nós sobre a sua fronteira. O número de nós de cada elemento finito bidimensional quadrilátero da família serendipity é 4(*p* - 1), sendo *p* o número de nós de um bordo. Estes elementos finitos apresentam um bom compromisso entre o número de nós e a qualidade dos resultados obtidos.

3.2.4. Campo de Deslocamentos

A análise de problemas estruturais em duas dimensões baseia-se na descrição cinemática do movimento através de um campo de deslocamentos bidimensional. Este campo de deslocamentos pode ser escrito na forma seguinte.

$$\mathbf{u} = \begin{Bmatrix} u(x,y) \\ v(x,y) \end{Bmatrix} \tag{3.30}$$

O polinómio que descreve a aproximação elementar de uma determinada componente do campo de deslocamentos é definido por um número de parcelas igual ao número de nós do elemento finito. A

seleção dos termos que compõem o polinómio é feita com recurso ao triângulo de Pascal. Se se conhecer os *n* deslocamentos nodais para cada uma das direções, cada componente de $\mathbf{u}(x,y)$ é interpolada separadamente com base nas funções de forma $N_i(x,y)$ e nos deslocamentos nodais.

$$u(x,y) = N_1(x,y)u_1 + N_2(x,y)u_2 + \cdots + N_n(x,y)u_n \tag{3.31a}$$

$$v(x,y) = N_1(x,y)v_1 + N_2(x,y)v_2 + \cdots + N_n(x,y)v_n \tag{3.31b}$$

Na forma matricial, (3.31) é equivalente a

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & \cdots & N_n & 0 \\ 0 & N_1 & 0 & N_2 & \cdots & 0 & N_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} \tag{3.32}$$

ou, de forma mais compacta,

$$\mathbf{u} = \boldsymbol{N}\,\boldsymbol{a}\,. \tag{3.33}$$

Como referido anteriormente, a matriz $\boldsymbol{N}$ agrupa as funções de forma e o vetor $\boldsymbol{a}$ os deslocamentos nodais que dependem do elemento finito considerado.

3.2.5. Campo de Deformações

Conhecidas as funções de forma, o campo de deslocamentos num qualquer ponto do elemento finito bidimensional pode ser determinado a partir das suas funções de forma e dos valores dos seus deslocamentos nodais. Sendo o campo de deformações do elemento finito obtido por derivação do campo de deslocamentos, então, o campo de deformações é definido da seguinte forma.

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix} \tag{3.34}$$

Como o campo de deslocamentos é aproximado com recurso a funções interpoladoras, conhecido o campo de deslocamentos é suficiente derivar as funções de forma e multiplicá-las pelos respetivos deslocamentos nodais. Assim,

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \dfrac{\partial N_1}{\partial x} & 0 & \dfrac{\partial N_2}{\partial x} & 0 & \cdots & \dfrac{\partial N_n}{\partial x} & 0 \\ 0 & \dfrac{\partial N_1}{\partial y} & 0 & \dfrac{\partial N_2}{\partial y} & \cdots & 0 & \dfrac{\partial N_n}{\partial y} \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial y} & \dfrac{\partial N_2}{\partial x} & \cdots & \dfrac{\partial N_n}{\partial y} & \dfrac{\partial N_n}{\partial x} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_n \\ v_n \end{Bmatrix} \tag{3.35}$$

ou, de forma mais compacta,

$$\boldsymbol{\varepsilon} = \boldsymbol{B}\,\boldsymbol{a}\,. \tag{3.36}$$

A matriz $\boldsymbol{B}$ é designada por matriz de deformação e é obtida apenas com recurso à derivação das funções de forma do elemento finito associadas a cada nó.

3.2.6. CAMPO DE TENSÕES

As tensões, para um material isotrópico com comportamento linear elástico, podem ser obtidas através da relação constitutiva (3.20) que após a substituição de (3.36) resulta em

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{a} \tag{3.37}$$

onde a matriz $\boldsymbol{D}$ é designada por matriz de elasticidade. Como se pode constatar, o cálculo das tensões no elemento finito está dependente do conhecimento dos deslocamentos nodais e da avaliação da matriz $\boldsymbol{B}$ no ponto onde se pretende conhecer o estado de tensão.

3.2.7. TENSÕES E DIREÇÕES PRINCIPAIS

O estado plano de tensões num determinado ponto de uma estrutura em estudo é representado pelo elemento da figura 3.4, onde também estão definidas as tensões principais e as suas direções.

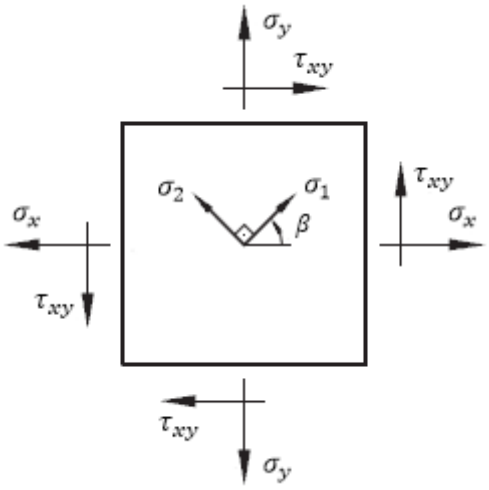


Fig.3.4 – Definição de tensões, tensões principais e respetivas direções.

Na prática da engenharia é importante determinar a orientação dos planos que fazem a tensão normal atingir o máximo e o mínimo, bem como a orientação dos planos que fazem a tensão de corte chegar ao máximo.

$$\sigma_{1,2} = \frac{\sigma_x + \sigma_y}{2} \pm \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + {\tau_{xy}}^2} \tag{3.38}$$

Dependendo do sinal escolhido obtém-se o valor para as tensões σ_1 e σ_2. As tensões principais σ_1 e σ_2 representam, respetivamente, a tensão normal máxima e a mínima no ponto. Quando o estado de tensão é representado pelas tensões principais, nenhuma tensão de corte atua sobre o elemento. O ângulo formado pela direção da tensão principal σ_1 com a horizontal é definido por

$$tan\,2\beta = \frac{2\tau_{xy}}{\sigma_x - \sigma_y}. \tag{3.39}$$

Da análise de (3.38) pode-se referir que as duas parcelas que permitem obter o valor das tensões principais correspondem, respetivamente, às tensões normais médias e à tensão de corte máxima. O elemento que representa a tensão de corte máxima está orientado a 45º da posição do elemento que representa as tensões principais.

3.3. ELEMENTOS FINITOS TRIDIMENSIONAIS

3.3.1. ESTADO GERAL DE TENSÃO

A generalidade dos problemas encontrados na prática da engenharia são tridimensionais. Correntemente recorre-se a simplificações que permitem analisar o meio tridimensional através de uma abordagem bidimensional. Contudo, em muitos casos não é possível ou desejável adotar as simplificações do estado plano de tensão ou outros.

A utilização de elementos finitos tridimensionais proporciona uma maior abrangência de possibilidades de modelação de meios contínuos que não são fornecidas pelos elementos finitos bidimensionais. As necessidades que levam à exigência de se utilizar elementos tridimensionais podem ser [21], por exemplo:

- A geometria do meio em estudo ser tridimensional;
- Aplicação de cargas em três planos ortogonais;
- Necessidade de conhecer o comportamento do meio em três direções ortogonais;
- Utilização de materiais com propriedades distintas em cada direção ortogonal.

No estudo dos meios contínuos, independentemente do comportamento mecânico, pretende-se conhecer as tensões e extensões num qualquer sólido contínuo. No caso tridimensional, a tensão resultante num dado ponto pode ser decomposta nas tensões que atuam nas faces do elemento de volume representado na figura 3.5 e que está orientado segundo o sistema de eixos ortogonal *Oxyz*.

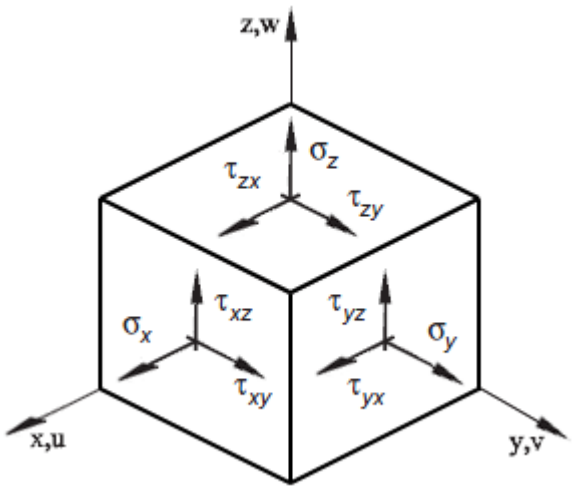


Fig.3.5 – Sólido sujeito a um estado tridimensional de tensão.

O elemento representado tem lados com dimensões infinitesimais *dx*, *dy* e *dz*, encontrando-se em equilíbrio de tensões. Este elemento representa o estado de tensão num dado ponto considerando desprezável a variação das tensões ao longo das suas faces. Para haver equilíbrio, o momento de todas as forças deve ser nulo [4].

Deste modo, considerando individualmente cada um dos eixos e calculando os momentos em relação a esse eixo conclui-se o seguinte.

$$\tau_{xy} = \tau_{yx} \tag{3.40}$$

$$\tau_{xz} = \tau_{zx} \tag{3.41}$$

$$\tau_{yz} = \tau_{zy} \tag{3.42}$$

Deste modo, no caso tridimensional, obtêm-se seis parâmetros independentes de tensão, ou seja, três tensões normais e três tensões tangenciais. Assim, as componentes cartesianas da extensão e da distorção, para o estado geral de tensão, são as seguintes.

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{Bmatrix} \tag{3.43}$$

O campo de tensões associado ao campo de deformações pode ser escrito pelas seguintes componentes cartesianas.

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{Bmatrix} \tag{3.44}$$

Para materiais que apresentem comportamento linear elástico, homogeneidade e isotropia, as tensões podem ser obtidas através da relação constitutiva

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{\varepsilon} \tag{3.45}$$

onde a matriz $\boldsymbol{D}$ é designada por matriz de elasticidade e é função das propriedades elásticas do material, isto é, do módulo de elasticidade, E, e do coeficiente de Poisson, ν. Para um material isotrópico com comportamento linear elástico,

$$\boldsymbol{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}. \tag{3.46}$$

3.3.2. Princípio dos Trabalhos Virtuais

O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{3.47}$$

Para um elemento finito tridimensional sujeito a um carregamento de volume, o Princípio dos trabalhos Virtuais pode ser escrito na forma

$$\int_V \delta\boldsymbol{\varepsilon}^T \boldsymbol{\sigma}\, dV = \int_V \delta\mathbf{u}^T \boldsymbol{b}\, dV \tag{3.48}$$

em que $\delta\boldsymbol{\varepsilon}$ indica as deformações virtuais, $\boldsymbol{\sigma}$ as tensões reais aproximadas e $\delta\mathbf{u}$ os deslocamentos virtuais. De acordo com o apresentado no Capítulo 2, para um elemento finito tridimensional sujeito a um carregamento de volume caraterizado através do vetor $\boldsymbol{b}$, tem-se

$$\int_V \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, dV\, \boldsymbol{a} = \int_V \boldsymbol{N}^T \boldsymbol{b}\, dV \tag{3.49}$$

onde $\boldsymbol{B}$ é a matriz de deformação, $\boldsymbol{D}$ a matriz de elasticidade, $\boldsymbol{a}$ o vetor de deslocamentos nodais, $\boldsymbol{N}$ a matriz que agrupa as funções de forma do elemento finito e $\boldsymbol{b}$ o vetor caraterizador da força de volume que atua no elemento finito.

Para o caso de um elemento finito tridimensional paralelepipédico de largura a, comprimento b e altura c tem-se

$$\int_{-c/2}^{+c/2}\int_{-b/2}^{+b/2}\int_{-a/2}^{+a/2} \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, dx\, dy\, dz\, \boldsymbol{a} = \int_{-c/2}^{+c/2}\int_{-b/2}^{+b/2}\int_{-a/2}^{+a/2} \boldsymbol{N}^T \boldsymbol{b}\, dx\, dy\, dz \tag{3.50}$$

ou, de forma mais compacta,

$$\boldsymbol{k}\, \boldsymbol{a} = \boldsymbol{f}\,. \tag{3.51}$$

Note-se que (3.50) apenas permite o cálculo da matriz de rigidez elementar e do vetor de forças nodais equivalentes para elementos finitos hexaédricos de geometria regular. Para elementos finitos distorcidos é habitual recorrer-se a elementos finitos isoparamétricos e integração numérica.

3.3.3. Funções de Forma

Na figura 3.6 encontra-se representado o elemento finito paralelepipédico com oito nós, largura a, comprimento b e altura c.

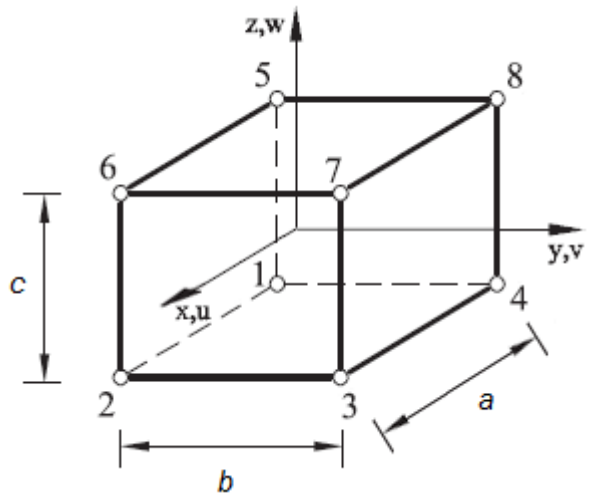


Fig.3.6 – Elemento finito paralelepipédico de oito nós.

As funções de forma para os elementos finitos tridimensionais, em que x_i, y_i e z_i são as coordenadas do nó i e n representa o número total de nós, são determinadas através da expressão matricial

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ \vdots \end{Bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ z_1 & z_2 & \cdots & z_n \\ \vdots & \vdots & & \vdots \end{bmatrix}^{-1} \begin{Bmatrix} 1 \\ x \\ y \\ z \\ \vdots \end{Bmatrix} \tag{3.52}$$

ou, de forma mais compacta,

$$\boldsymbol{N_V} = \boldsymbol{Q}^{-1}\boldsymbol{V} \tag{3.53}$$

em que $\boldsymbol{Q}$ é uma matriz quadrada, que se supõe não singular, que após a substituição das respetivas coordenadas dos nós do elemento finito permite calcular as funções de forma.

3.3.4. Campo de Deslocamentos

A análise de problemas estruturais em três dimensões baseia-se na descrição cinemática do movimento através de um campo de deslocamentos tridimensional. Este campo de deslocamentos pode ser escrito na forma seguinte.

$$\mathbf{u} = \begin{Bmatrix} u(x,y,z) \\ v(x,y,z) \\ w(x,y,z) \end{Bmatrix} \tag{3.54}$$

Se se conhecer os *n* deslocamentos nodais para cada direção, cada uma das componentes de $\mathbf{u}(x,y,z)$ é interpolada separadamente com base nas funções de forma $N_i(x,y,z)$ e nos deslocamentos nodais. Na forma matricial é equivalente a

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & 0 & N_2 & 0 & 0 & \cdots & N_n & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & \cdots & 0 & N_n & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & \cdots & 0 & 0 & N_n \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ u_2 \\ v_2 \\ w_2 \\ \vdots \\ u_n \\ v_n \\ w_n \end{Bmatrix} \tag{3.55}$$

ou, de forma mais compacta,

$$\mathbf{u} = \boldsymbol{N}\,\boldsymbol{a}\,. \tag{3.56}$$

6.3.5. Campo de Deformações

O campo de deformações do elemento finito é obtido por derivação do campo de deslocamentos. Deste modo, o campo de deformações num estado tridimensional de tensão é definido derivando cada uma das componentes do campo de deslocamentos.

$$\boldsymbol{\varepsilon} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial w}{\partial z} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \\ \dfrac{\partial u}{\partial z} + \dfrac{\partial w}{\partial x} \\ \dfrac{\partial v}{\partial z} + \dfrac{\partial w}{\partial y} \end{Bmatrix} \tag{3.57}$$

Sendo o campo de deslocamentos aproximado com recurso a funções interpoladoras, conhecido o campo de deslocamentos para o elemento finito é suficiente derivar as suas funções de forma e multiplicá-las pelos respetivos deslocamentos nodais. Assim, para um elemento finito tridimensional de *n* nós,

$$\boldsymbol{\varepsilon} = \left[\begin{array}{ccc|ccc|c|ccc} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & & \frac{\partial N_n}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & & 0 & \frac{\partial N_n}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & & 0 & 0 & \frac{\partial N_n}{\partial z} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \cdots & \frac{\partial N_n}{\partial y} & \frac{\partial N_n}{\partial x} & 0 \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & & \frac{\partial N_n}{\partial z} & 0 & \frac{\partial N_n}{\partial x} \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & & 0 & \frac{\partial N_n}{\partial z} & \frac{\partial N_n}{\partial y} \end{array}\right] \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ u_2 \\ v_2 \\ w_2 \\ \vdots \\ u_n \\ v_n \\ w_n \end{Bmatrix} \tag{3.58}$$

ou, de forma mais compacta,

$$\boldsymbol{\varepsilon} = \boldsymbol{B}\,\boldsymbol{a}\,. \tag{3.59}$$

Como já referido, a matriz $\boldsymbol{B}$ é designada por matriz de deformação e é obtida apenas com recurso às derivadas das funções de forma do elemento finito associadas a cada nó.

3.3.6. Campo de Tensões

As tensões, para um material isotrópico com comportamento linear elástico, podem ser obtidas através da relação constitutiva (3.45) que após a substituição de (3.59) conduz a

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{a} \tag{3.60}$$

onde a matriz $\boldsymbol{D}$ é designada por matriz de elasticidade. Como se pode constatar, o cálculo das tensões no elemento finito está dependente do conhecimento dos deslocamentos nodais e da avaliação da matriz $\boldsymbol{B}$ no ponto onde se pretende conhecer os seus valores. A dificuldade desta metodologia reside em escolher o ponto adequado para avaliar a matriz $\boldsymbol{B}$ de modo a se obter resultados precisos.

3.3.7. Famílias de Elementos Finitos

Do mesmo modo que os elementos finitos bidimensionais, os elementos finitos tridimensionais também costumam ser divididos em duas famílias, isto é, em elementos da família lagrangeana e em elementos da família serendipity.

Os elementos finitos hexaédricos da família lagrangeana são obtidos pela extensão dos elementos finitos bidimensionais quadrilaterais ao caso tridimensional. As funções de forma dos elementos desta família podem ser obtidas de duas maneiras. A primeira metodologia para a obtenção das funções de forma consiste em recorrer à expressão geral (3.53). A outra metodologia resulta da combinação dos polinómios de Lagrange deduzidos para cada direção ortogonal.

Os elementos finitos hexaédricos da família serendipity possuem apenas nós sobre a sua fronteira. As funções de forma para os elementos finitos desta família, em que *n* representa o número total de nós, são determinadas através da expressão geral (3.53).

3.4. ELEMENTOS FINITOS DE VIGA

3.4.1. VIGAS PELA TEORIA DE EULER-BERNOULLI

3.4.1.1. Breve Descrição da Teoria

A teoria Euler-Bernoulli considera que as secções transversais se mantêm planas e normais ao eixo da barra após a deformação, pelo que, as deformações por corte são desprezadas. Considera-se somente a existência de cargas transversais ao eixo da viga e são desprezadas as cargas axiais.

Considere-se uma viga de comprimento L, momento de inércia I_y, sob ação de cargas verticais e momentos que atuam no plano xz e que o eixo x da viga coincide com a linha que une o centro de gravidade das secções transversais. A teoria de Euler-Bernoulli para vigas estabelece as seguintes hipóteses [14]:

- O deslocamento vertical, w, dos pontos contidos numa secção transversal são iguais ao valor de deflexão da viga;
- O deslocamento lateral v ao longo do eixo y é nulo;
- As secções transversais normais ao eixo da viga permanecem planas e ortogonais ao eixo da viga após deformação.

3.4.1.2. Princípio dos Trabalhos Virtuais

O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{3.61}$$

Para dedução da equação de equilíbrio considere-se a figura 3.7 relativa a um elemento finito de viga sujeito a um carregamento transversal distribuído ao longo do seu comprimento que origina forças nodais concentradas e momentos fletores.

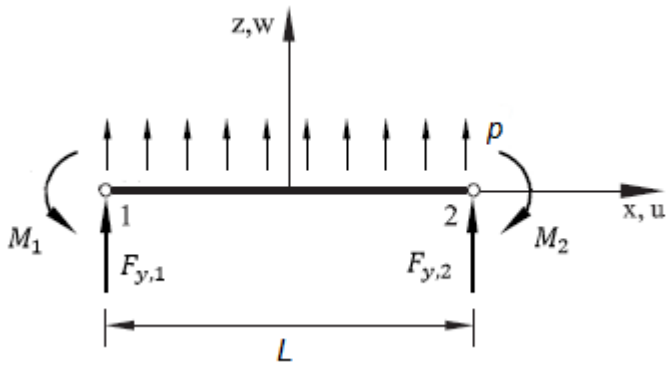


Fig.3.7 – Elemento finito de viga sujeito a um carregamento transversal.

As cargas verticais são consideradas positivas se atuarem na sentido do eixo vertical z. Os momentos fletores concentrados são tomados como positivos se atuarem no sentido anti-horário, em coerência com a rotação θ. Para o caso do elemento finito representado na figura 3.7, de comprimento L e sujeito a uma carga transversal p uniformemente distribuída ao longo do seu comprimento,

$$\int_V \delta\varepsilon^T \sigma \, dV = \int_L \delta w^T p \, dL \, . \tag{3.62}$$

De acordo com o Capítulo 2, o Princípio dos Trabalhos Virtuais, após as devidas manipulações algébricas, pode-se escrever na forma

$$\int_{-L/2}^{+L/2} \boldsymbol{B}^T E\, \boldsymbol{B}\, I_y\, dx\, \boldsymbol{a} = \int_{-L/2}^{+L/2} \boldsymbol{N}^T p\, dx \tag{3.63}$$

em que I_y representa o momento de inércia em relação ao eixo y. Colocando (3.63) numa forma mais compacta,

$$\boldsymbol{k}\,\boldsymbol{a} = \boldsymbol{f}\,. \tag{3.64}$$

3.4.1.3. Funções de Forma

Considere-se o elemento finito de viga com n nós representado na figura 3.8 de comprimento L, módulo de elasticidade E e momento de inércia I_y.

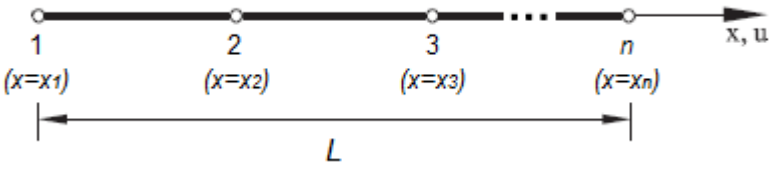


Fig.3.8 – Elemento finito de viga com n nós.

Os elementos finitos de viga possuem dois graus de liberdade por nó, isto é, um deslocamento segundo z e uma rotação no plano xz. Num elemento finito com n nós existem $2n$ graus de liberdade. Quanto à numeração dos nós, para efeitos de implementação computacional, é conveniente que seja feita da esquerda para a direita.

A obtenção das funções de forma para o elemento finito de viga de Euler-Bernoulli é feita com recurso à interpolação hermitiana. Simplificadamente, a obtenção das funções de forma é realizada através da expressão matricial

$$\begin{Bmatrix} N_1 \\ N_2 \\ \vdots \\ N_{2n} \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & \cdots & 1 & 0 \\ x_1 & 1 & x_2 & 1 & \cdots & x_n & 1 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_1^{\alpha} & \alpha x_1^{\beta} & x_2^{\alpha} & \alpha x_2^{\beta} & \cdots & x_n^{\alpha} & \alpha x_n^{\beta} \end{bmatrix}^{-1} \begin{Bmatrix} 1 \\ x \\ \vdots \\ x^{\alpha} \end{Bmatrix} \tag{3.65}$$

com $\alpha = 2n-1$ e $\beta = 2n-2$ ou, de forma mais compacta,

$$\boldsymbol{N_V} = \boldsymbol{Q}^{-1}\boldsymbol{V}\,. \tag{3.66}$$

3.4.1.4. Campo de Deslocamentos

Um dado ponto que se encontre à distância z do eixo da viga no estado indeformado, permanece à mesma distância após deformação da viga. No estado deformado, a secção transversal da viga sobre apenas uma rotação

$$\theta = \frac{dw}{dx}\,. \tag{3.67}$$

Desta forma, o campo de deslocamentos num ponto qualquer do elemento finito de viga pode ser determinado através das igualdades seguintes.

$$u(x,y,z) = -z\,\theta \tag{3.68}$$

$$v(x,y,z) = 0 \tag{3.69}$$

$$w(x,y,z) = w(x) \tag{3.70}$$

Nas expressões anteriores, *u*, *v* e *w* são os deslocamentos nas direções *x*, *y* e *z*, respetivamente. Assim, encontrado o valor de *w*, o campo de deslocamentos do problema fica determinado, pois, a rotação é obtida a partir da derivada de *w*. Conhecidos os deslocamentos verticais e as rotações nodais nos nós do elemento finito, o campo de deslocamentos verticais pode ser interpolado a partir desses mesmos deslocamentos, ou seja,

$$w(x) = N_1(x)a_1 + N_2(x)a_2 + \cdots + N_{2n}(x)a_{2n} \tag{3.71}$$

em que $N_i(x)$ são as funções de forma do elemento finito para o grau de liberdade *i*. Colocando (3.71) em notação matricial,

$$w = [N_1 \quad N_2 \quad \cdots \quad N_{2n}] \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{2n} \end{Bmatrix} \tag{3.72}$$

ou, de forma mais compacta,

$$w = \boldsymbol{N}\,\boldsymbol{a}\,. \tag{3.73}$$

3.4.1.5. Campo de Deformações

A partir do campo de deslocamentos descrito, conclui-se que só existe deformação segundo a direção horizontal paralela ao eixo longitudinal *x*. Sendo o campo de deformações do elemento finito obtido pela derivação do campo de deslocamentos, então,

$$\varepsilon = \frac{d\mathrm{u}}{dx} = -z\frac{d^2w}{dx^2}\,. \tag{3.74}$$

Devido à presença de segundas derivadas em (3.74) é necessário considerar elementos finitos de continuidade C^1, ou seja, com continuidade da função e das suas primeiras derivadas. Como se pode verificar, para caracterizar de forma completa o estado de deformação numa viga é apenas necessário conhecer uma curvatura de flexão.

De acordo com a expressão (3.74), o campo de deformações para um elemento finito com *n* nós na forma matricial obtém-se pela expressão

$$\varepsilon = -z\begin{bmatrix} \frac{d^2N_1}{dx^2} & \frac{d^2N_2}{dx^2} & \cdots & \frac{d^2N_{2n}}{dx^2} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{2n} \end{Bmatrix} \tag{3.75}$$

ou, de forma mais compacta,

$$\varepsilon = -z\,\boldsymbol{B}\,\boldsymbol{a}\,. \tag{3.76}$$

3.4.1.6. Campo de Tensões

As tensões, para um material isotrópico com comportamento linear elástico, são obtidas através da relação constitutiva

$$\sigma = E\,\varepsilon \tag{3.77}$$

em que E é o módulo de elasticidade do material. Após a substituição de (3.76) chega-se a

$$\sigma = -\mathrm{z}\,E\,\boldsymbol{B}\,\boldsymbol{a}\,. \tag{3.78}$$

3.4.1.7. Esforços Internos

Para vigas à flexão, as tensões variam linearmente ao longo da altura e os valores máximos ocorrem nos planos limite da viga. Assim, o momento fletor é obtido a partir da integração da tensão ao longo da secção transversal da viga.

$$M = \int_S \sigma\, z\, dS \tag{3.79}$$

Após as devidas substituições em (3.79) e resolvendo o integral, o momento fletor na viga é obtido através da expressão

$$M = -E\,I_y\,\boldsymbol{B}\,\boldsymbol{a} \tag{3.80}$$

onde a matriz $\boldsymbol{B}$ deve ser avaliada no ponto onde se pretende calcular o momento fletor. Note-se que esta formulação de elementos finitos não permite considerar esforços axiais. Esta limitação é ultrapassa com a combinação desta formulação com a do elemento finito unidimensional.

3.4.2. VIGAS PELA TEORIA DE TIMOSHENKO

3.4.2.1. Breve Descrição da Teoria

A teoria de Timoshenko considera que as secções se mantêm planas após deformação mas, ao contrário da formulação de Euler-Bernoulli, estas não se mantêm normais ao eixo da barra. A formulação de Timoshenko considera o contributo das deformações por corte. Considera-se somente a existência de cargas transversais ao eixo da viga aplicadas segundo a direção z e que as cargas axiais são desprezadas.

A teoria de Timoshenko para vigas estabelece as seguintes hipóteses [14]:

- O deslocamento vertical, w, dos pontos contidos numa secção transversal são iguais ao valor de deflexão da viga;
- O deslocamento lateral v ao longo do eixo y é nulo;
- As secções transversais normais ao eixo da viga permanecem planas mas não necessariamente ortogonais ao eixo da viga após deformação.

3.4.2.2. Princípio dos Trabalhos Virtuais

Considere-se um elemento finito de comprimento L, com uma área de secção transversal A, sujeito a um carregamento distribuído ao logo do seu comprimento no plano xz e que o eixo x do elemento finito coincide com a linha que une o centro de gravidade de todas as secções transversais.

O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{3.81}$$

Para o elemento finito descrito, a relação entre trabalho virtual interno e externo é equivalente a

$$\int_V \delta\varepsilon^T \sigma \, dV + \int_V \delta\gamma^T \tau \, dV = \int_L \delta \mathbf{u}^T \boldsymbol{p} \, dL \, . \tag{3.82}$$

Atendendo ao comprimento do elemento finito e de acordo com o que já foi expresso para os elementos finitos de viga pela teoria Euler-Bernoulli, após alguma manipulação algébrica, chega-se a

$$\int_{-L/2}^{+L/2} \boldsymbol{B_b^T} E \, \boldsymbol{B_b} \, I_y \, dx \, \boldsymbol{a} + \int_{-L/2}^{+L/2} \boldsymbol{B_s^T} G \, \boldsymbol{B_s} \, A^* \, dx \, \boldsymbol{a} = \int_{-L/2}^{+L/2} \boldsymbol{N}^T \boldsymbol{p} \, dx \tag{3.83}$$

ou, de forma mais compacta,

$$\boldsymbol{k} \, \boldsymbol{a} = \boldsymbol{f} \, . \tag{3.84}$$

3.4.2.3. Funções de Forma

Considere-se o elemento finito de viga com *n* nós representado na figura 3.9, de comprimento L, área da secção transversal A, momento de inércia I_y, módulo de elasticidade E e módulo de distorção G.

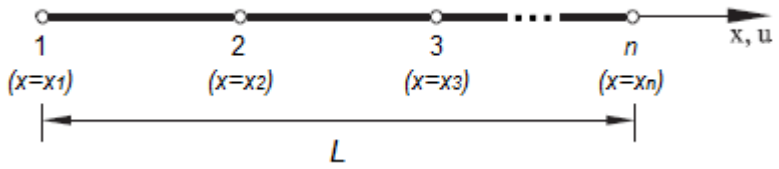


Fig.3.9 – Elemento finito de viga com *n* nós.

Como o elemento finito de viga possui dois graus de liberdade por nó, isto é, um deslocamento segundo z e uma rotação no plano xz, então, no elemento finito com n nós existem $2n$ graus de liberdade. As funções de forma para o elemento finito de viga de Timoshenko são obtidas do mesmo modo que as funções de forma para os elementos finitos unidimensionais.

3.4.2.4. Campo de Deslocamentos

Um dado ponto que se encontra à distância z do eixo da viga no estado indeformado, permanecerá à mesma distância após deformação da viga mas não necessariamente ortogonal. No estado deformado, a secção transversal da viga sobre apenas uma rotação dada por

$$\theta = \frac{dw}{dx} + \phi \, . \tag{3.85}$$

Pela análise de (3.85), a rotação é obtida pela soma da inclinação do eixo da viga com uma rotação adicional ϕ. Note-se que o valor da rotação não coincide com a inclinação da viga como acontece na teoria de Euler-Bernoulli. O campo de deslocamentos num qualquer ponto do elemento de viga de Timoshenko pode ser determinado através das igualdades seguintes.

$$u(x, y, z) = -z \, \theta \tag{3.86}$$

$$v(x, y, z) = 0 \tag{3.87}$$

$$w(x, y, z) = w(x) \tag{3.88}$$

Nas expressões anteriores, *u*, *v* e *w* são os deslocamentos nas direções *x*, *y* e *z*, respetivamente. A interpolação do deslocamento vertical *w* e da rotação θ é efetuado separadamente para cada uma destas variáveis. Uma vez que *w* e θ apresentam *n* valores nodais cada é utilizada a seguinte interpolação unidimensional com *n* nós.

$$w(x) = N_1(x)w_1 + N_2(x)w_2 + \cdots + N_n(x)w_n \tag{3.89a}$$

$$\theta(x) = N_1(x)\theta_1 + N_2(x)\theta_2 + \cdots + N_n(x)\theta_n \tag{3.89b}$$

Onde em (3.89) $N_i(x)$ é a função de forma do elemento finito para o nó *i*. Colocando (3.89) em notação matricial, chega-se a

$$\begin{Bmatrix} w \\ \theta \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & \cdots & N_n & 0 \\ 0 & N_1 & 0 & N_2 & \cdots & 0 & N_n \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \\ \vdots \\ w_n \\ \theta_n \end{Bmatrix} \tag{3.90}$$

ou, de forma mais compacta,

$$\mathbf{u} = \boldsymbol{N}\,\boldsymbol{a}\,. \tag{3.91}$$

3.4.2.5. Campo de Deformações

O campo de deformações do elemento finito é dado por derivação do campo de deslocamentos, ou seja,

$$\varepsilon = \frac{du}{dx} = -z\frac{d\theta}{dx} \tag{3.92}$$

e

$$\gamma = \frac{dw}{dx} + \frac{du}{dz} = \frac{dw}{dx} - \theta\,. \tag{3.93}$$

Da análise de (3.92) conclui-se que os elementos finitos pela teoria de Timoshenko exigem apenas continuidade C^0 para a deflexão e campo de rotações. Por conseguinte, são mais simples do que os elementos de viga de Euler-Bernoulli. Infelizmente, sofrem geralmente do chamado *locking*, defeito que produz excesso de rigidez em vigas esbeltas.

Na forma matricial, as extensões para um elemento finito com *n* nós são obtidas pela expressão

$$\varepsilon = -z\begin{bmatrix} 0 & \dfrac{\partial N_1}{\partial x} & 0 & \dfrac{\partial N_2}{\partial x} & \cdots & 0 & \dfrac{\partial N_n}{\partial x} \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \\ \vdots \\ w_n \\ \theta_n \end{Bmatrix} \tag{3.94}$$

ou, de forma mais compacta,

$$\varepsilon = -z\,\boldsymbol{B_b}\,\boldsymbol{a}\,. \tag{3.95}$$

De igual modo, para um elemento finito com *n* nós, o cálculo das distorções é efetuado com recurso à expressão

$$\gamma = \begin{bmatrix} \frac{\partial N_1}{\partial x} & -N_1 & \frac{\partial N_2}{\partial x} & -N_2 & \cdots & \frac{\partial N_n}{\partial x} & -N_n \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_1 \\ w_2 \\ \theta_2 \\ \vdots \\ w_n \\ \theta_n \end{Bmatrix} \tag{3.96}$$

ou, de forma mais compacta,

$$\gamma = \boldsymbol{B_s}\, \boldsymbol{a}\,. \tag{3.97}$$

3.4.2.6. Campo de Tensões

As tensões axiais e de corte, para um material isotrópico com comportamento linear elástico, são obtidas através das relações

$$\sigma = E\, \varepsilon \tag{3.98}$$

e

$$\tau = G\, \gamma \tag{3.99}$$

em que E é o módulo de elasticidade e G o de distorção. Assim, após as correspondentes substituições, chega-se a

$$\sigma = -\mathrm{z}\, E\, \boldsymbol{B_b}\, \boldsymbol{a} \tag{3.100}$$

e

$$\tau = G\, \boldsymbol{B_s}\, \boldsymbol{a}\,. \tag{3.101}$$

3.4.2.7. Esforços Internos

Para vigas à flexão, as tensões axiais variam linearmente ao longo da altura e os valores máximos ocorrem nos planos limite da viga. O momento fletor é obtido a partir da integração da tensão ao longo da secção transversal da viga.

$$M = \int_S \sigma\, z\, dS = -E\, I_y\, \boldsymbol{B_b}\, \boldsymbol{a} \tag{3.102}$$

As tensões de corte pela teoria de Timoshenko permanecem constantes ao longo da espessura, pelo que, o esforço transverso atuante é calculado pela expressão

$$V = \int_S \tau\, dS = G\, A^* \boldsymbol{B_s}\, \boldsymbol{a} \tag{3.103}$$

em que A^* corresponde à área reduzida da secção transversal. Para secções maciças de forma retangular $A^* = (5/6)A$ e para circulares $A^* = (6/7)A$.

As matrizes $\boldsymbol{B_b}$ e $\boldsymbol{B_s}$ devem ser avaliadas nos pontos onde se pretende calcular o momento fletor e o esforço transverso, respetivamente. Importa referir que, quer o momento fletor, quer o esforço transverso, calculados pelas expressões (3.102) e (3.103) só apresentam valores aceitáveis em determinados pontos do elemento finito. Normalmente, estes pontos são coincidentes com os utilizados no procedimento de integração numérica pela quadratura de Gauss-Legendre.

3.5. ELEMENTOS FINITOS DE LAJE

3.5.1. LAJES PELA TEORIA DE KIRCHHOFF

3.5.1.1. Breve Descrição da Teoria

A teoria de Kirchhoff, à semelhança dos elementos de viga pela teoria Euler-Bernoulli, não permite considerar o efeito da deformabilidade por esforço transverso. A teoria de Kirchhoff é adequada para o estudo de lajes onde a razão entre a espessura e a menor dimensão da superfície média é inferior a 1/10, ou seja, é adequada ao estudo de lajes finas.

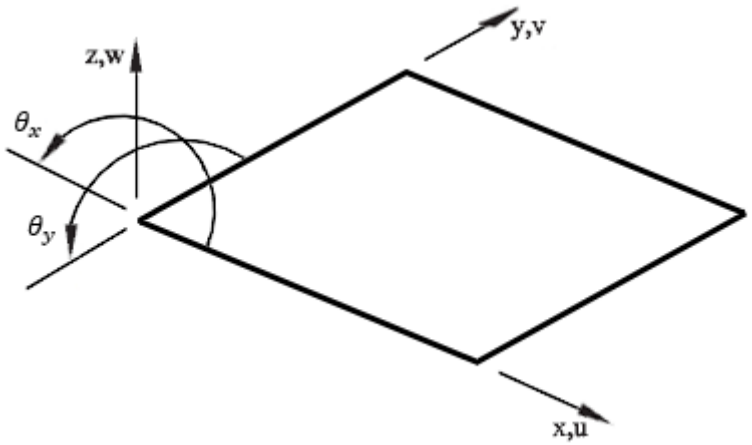


Fig.3.10 – Graus de liberdade e orientações das rotações.

No estudo de elementos finitos de laje pela teoria de Kirchhoff, além da condição da espessura ser muito inferior às outras dimensões, é considerado que as cargas são aplicadas transversalmente ao plano médio da laje. Esta teoria estabelece as seguintes hipóteses simplificadoras [17]:

- Nos pontos do plano médio os deslocamentos *u* e *v* são nulos;
- Todos os pontos normais ao plano médio possuem o mesmo deslocamento vertical;
- A tensão σ_z é desprezável;
- As retas normais ao plano médio indeformado da laje permanecem retas e normais ao plano médio após deformação.

Na teoria de Kirchhoff para lajes, o polinómio interpolador para o deslocamento vertical do plano médio, $w(x, y)$, é definido em função do número de graus de liberdade existentes em cada nó e da quantidade de nós do elemento finito. Uma das particularidades na formulação dos elementos finitos pela teoria de Kirchhoff prende-se com a continuidade das funções de forma. Se as funções de forma do elemento finito garantem continuidade das rotações entre elementos, estes designam-se por elementos finitos conformes. Caso não exista continuidade das rotações entre elementos finitos, estes designam-se por elementos finitos não conformes.

3.5.1.2. Princípio dos Trabalhos Virtuais

O domínio de uma laje pode ser descrito na forma

$$\Omega = \left\{(x,y,z) \in R^3 : z \in \left[-\frac{h}{2}, +\frac{h}{2}\right], (x,y) \in \alpha \subset R^2\right\} \tag{3.104}$$

onde α e h representam o plano médio e a espessura da laje, respetivamente. O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{3.105}$$

Considerando um elemento finito submetido a um carregamento q distribuído no seu plano médio atuando na direção z, a relação existente entre trabalho virtual interno e o trabalho virtual externo pode ser escrita na forma

$$\int_V \delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma}\, dV = \int_S \delta \mathbf{u}^T q\, dS\,. \tag{3.106}$$

Atendendo ao apresentado no Capítulo 2 sobre o Princípio dos Trabalhos Virtuais, a relação (3.106), após alguma manipulação algébrica, é equivalente a

$$\frac{h^3}{12}\int_{-b/2}^{+b/2}\int_{-a/2}^{+a/2} \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, dx\, dy\, \boldsymbol{a} = \int_{-b/2}^{+b/2}\int_{-a/2}^{+a/2} \boldsymbol{N}^T q\, dx\, dy \tag{3.107}$$

onde $\boldsymbol{B}$ é a matriz de deformação, $\boldsymbol{a}$ o vetor de deslocamentos nodais e $\boldsymbol{N}$ agrupa as funções de forma que dependem do elemento finito considerado. Colocando a relação (3.107) numa forma mais compacta,

$$\boldsymbol{k}\,\boldsymbol{a} = \boldsymbol{f}\,. \tag{3.108}$$

3.5.1.3. Funções de Forma

As funções de forma em elementos finitos de laje pela teoria de Kirchhoff são definidas em função do número de graus de liberdade dos nós do elemento finito. As diversas formulações de elementos finitos desta teoria podem apresentar três ou quatro graus de liberdade por nó. O significado físico deste quarto grau de liberdade corresponde ao valor da curvatura de torção no nó em causa [5].

Na figura 3.11 encontra-se representado um elemento finito retangular de comprimento a, largura b e espessura h.

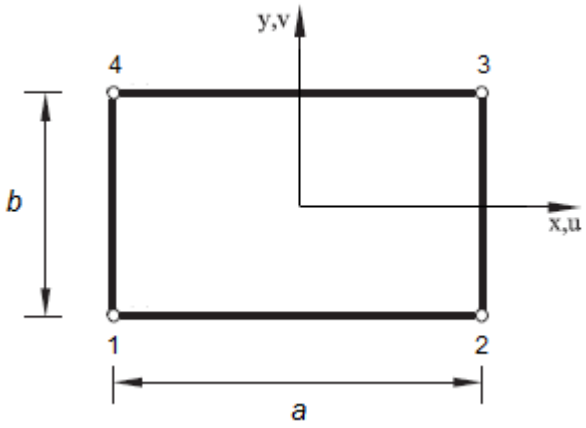


Fig.3.11 – Elemento finito retangular de quatro nós.

O polinómio interpolador que descreve a aproximação elementar para o deslocamento vertical do plano médio, $w(x, y)$, é definido por um número de termos igual ao número de graus de liberdade do elemento finito.

$$w(x,y) = c_1 + c_2x + c_3y + c_4x^2 + c_5xy + c_6y^2 + c_7x^3 + c_8x^2y + c_9xy^2 + c_{10}y^3 \cdots \quad (3.109)$$

Para determinar o valor das constantes c_i (i = 1, 2, … n), com n a representar o número de graus de liberdade do elemento finito, é necessário reescrever estas mesmas equações para cada uma das componentes de deslocamento dos nós.

3.5.1.4. Campo de Deslocamentos

Na figura 3.12 é mostrado o posicionamento de um qualquer ponto nos estados indeformado e deformado da laje. A figura representada também permite percecionar que as secções se mantêm planas e normais ao plano médio da laje após deformação.

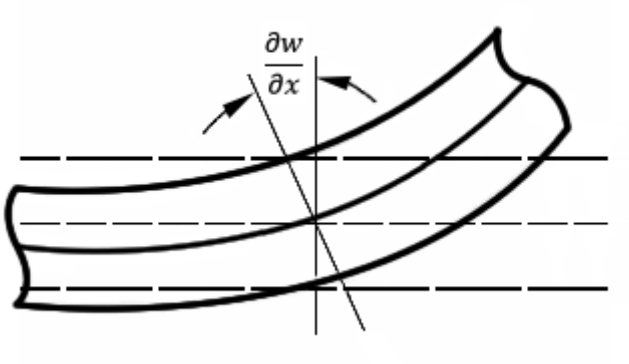


Fig.3.12 – Representação do deslocamento de um ponto no plano *xz*.

Segundo as hipóteses simplificadoras adotadas, um dado ponto que se encontre à distância z do plano médio no estado indeformado, permanece à mesma distância após deformação da laje. No estado deformado, qualquer segmento reto perpendicular ao plano médio sofre apenas rotação. Assim, analisando a figura 3.12, para o plano xz

$$\theta_x = \frac{\partial w}{\partial x} \quad (3.110)$$

e de forma análoga para o plano yz

$$\theta_y = \frac{\partial w}{\partial y}. \quad (3.111)$$

Desta forma, o campo de deslocamentos num ponto qualquer do elemento de laje pode ser determinado através das seguintes igualdades.

$$u(x,y,z) = -z\,\theta_x(x,y) \quad (3.112a)$$

$$v(x,y,z) = -z\,\theta_y(x,y) \quad (3.112b)$$

$$w(x,y,z) = w(x,y) \quad (3.112c)$$

Nas expressões (3.112), u, v e w são os deslocamentos nas direções x, y e z, respetivamente. Atendendo a que no plano médio os deslocamentos u e v são nulos, conclui-se que, encontrado o valor de w, o campo de deslocamentos do problema está determinado, pois, as rotações são obtidas a partir das derivadas de w em relação a x ou y. Como os campos $\theta_x(x,y)$ e $\theta_y(x,y)$ não são independentes do campo de deslocamentos transversais $w(x,y)$, então, quando se formulam elementos finitos pela teoria de Kirchhoff é apenas necessário definir uma aproximação conveniente para o campo de deslocamentos

transversais. O deslocamento transversal para elementos finitos da teoria de Kirchhoff pode ser interpolado a partir dos seus deslocamentos nodais, isto é,

$$w = [N_1 \quad N_2 \quad N_3 \quad N_4 \quad \cdots \quad N_{n-3} \quad N_{n-2} \quad N_{n-1} \quad N_n] \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_{n-3} \\ a_{n-2} \\ a_{n-1} \\ a_n \end{Bmatrix} \tag{3.113}$$

ou, de forma mais compacta,

$$w = \boldsymbol{N}\,\boldsymbol{a}\,. \tag{3.114}$$

3.5.1.5. Campo de Deformações

A partir do campo de deslocamentos descrito, a teoria da elasticidade no domínio dos pequenos deslocamentos e deformações estabelece as seguintes relações entre deformação e deslocamento.

$$\varepsilon_x = -z\frac{\partial^2 w}{\partial x^2} \tag{3.115a}$$

$$\varepsilon_y = -z\frac{\partial^2 w}{\partial y^2} \tag{3.115b}$$

$$\gamma_{xy} = -2z\frac{\partial^2 w}{\partial x \partial y} \tag{3.115c}$$

Devido à presença de segundas derivadas em (3.115) é necessário considerar elementos finitos de continuidade C^1, ou seja, com continuidade da função e das suas primeiras derivadas. Para caracterizar de forma completa o estado de deformação numa laje fina são necessárias duas curvaturas de flexão e uma curvatura de torção.

Na forma matricial, o campo de deformações do elemento finito de laje com *n* graus de liberdade é obtido pela relação matricial

$$\boldsymbol{\varepsilon} = -z\begin{bmatrix} \dfrac{\partial^2 N_1}{\partial x^2} & \dfrac{\partial^2 N_2}{\partial x^2} & \cdots & \dfrac{\partial^2 N_n}{\partial x^2} \\ \dfrac{\partial^2 N_1}{\partial y^2} & \dfrac{\partial^2 N_2}{\partial y^2} & \cdots & \dfrac{\partial^2 N_n}{\partial y^2} \\ 2\dfrac{\partial^2 N_1}{\partial x \partial y} & 2\dfrac{\partial^2 N_2}{\partial x \partial y} & \cdots & 2\dfrac{\partial^2 N_n}{\partial x \partial y} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{\mathrm{n}} \end{Bmatrix} \tag{3.116}$$

ou, de forma mais compacta,

$$\boldsymbol{\varepsilon} = -z\,\boldsymbol{B}\,\boldsymbol{a}\,. \tag{3.117}$$

3.5.1.6. Campo de Tensões

Uma vez que a laje está livre para se deformar na direção *z* e que a tensão σ_z é desprezável, pode-se admitir que a relação entre tensão e deformação é análoga à dos problemas de Estado Plano de Tensão.

Assim, para materiais isotrópicos e com comportamento linear elástico, estabelece-se a seguinte a relação constitutiva entre o campo de tensões e o campo de deformações.

$$\boldsymbol{\sigma} = \boldsymbol{D}\,\boldsymbol{\varepsilon} \tag{3.118}$$

Após as devidas substituições, (3.118) é equivalente a

$$\boldsymbol{\sigma} = -z\,\boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{a} \tag{3.119}$$

onde matriz $\boldsymbol{D}$ é designada por matriz de elasticidade e é função das propriedades elásticas do material, isto é, do módulo de elasticidade, *E*, e do coeficiente de Poisson, ν. Em problemas de lajes pela teoria de Kirchhoff,

$$\boldsymbol{D} = \begin{bmatrix} \dfrac{E}{1-\nu^2} & \dfrac{E\nu}{1-\nu^2} & 0 \\ \dfrac{E\nu}{1-\nu^2} & \dfrac{E}{1-\nu^2} & 0 \\ 0 & 0 & \dfrac{E}{2(1+\nu)} \end{bmatrix}. \tag{3.120}$$

O campo de tensões associado ao campo de deformações pode ser escrito pelas seguintes componentes cartesianas.

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} \tag{3.121}$$

3.5.1.7. Esforços Internos

Para lajes sujeitas à flexão, as tensões variam linearmente ao longo da espessura e os valores máximos ocorrem nos planos limite da laje. Os esforços internos são, assim, obtidos a partir da integração das tensões ao longo da espessura da laje.

$$\begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \int_{-h/2}^{+h/2} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} z\,dz \tag{3.122}$$

Em (3.122), M_x, M_y e M_{xy} são, respetivamente, os momentos fletores por unidade de comprimento nas direções *x* e *y* e o momento de torção. Numa forma mais compacta,

$$\boldsymbol{M} = -\frac{h^3}{12}\boldsymbol{D}\,\boldsymbol{B}\,\boldsymbol{a}\,. \tag{3.123}$$

3.5.2. Lajes pela Teoria de Reissner-Mindlin

3.5.2.1. Breve Descrição da Teoria

A teoria de Reissner-Mindlin, à semelhança dos elementos de viga pela teoria de Timoshenko, permite considerar o efeito da deformabilidade devido ao esforço transverso. Nesta teoria é considerado que as retas perpendiculares ao plano médio se mantêm retas após deformação mas não necessariamente normais ao plano médio da laje. Esta teoria é adequada ao estudo de lajes espessas, mas, dada a simplicidade e facilidade de utilização dos elementos finitos formulados por esta teoria também se utilizam em situações em que se deixa de considerar a laje como espessa.

O estudo do elemento finito de laje pela teoria de Reissner-Mindlin pressupõe que a espessura é muito inferior às outras dimensões e que as cargas são aplicadas transversalmente ao seu plano médio. Assim como na teoria de Kirchhoff, a teoria de Reissner-Mindlin também está fundamentada em hipóteses simplificadoras [17], nomeadamente:

- Nos pontos do plano médio os deslocamentos *u* e *v* são nulos;
- Todos os pontos normais ao plano médio possuem o mesmo deslocamento vertical;
- A tensão σ_z é desprezável;
- As retas normais ao plano médio indeformado da laje permanecem retas mas não necessariamente normais ao plano médio após deformação.

Como as deformações por corte não são nulas, deixa de se poder calcular diretamente o campo de rotações a partir do campo de deslocamentos transversais. Desta forma, para se caracterizar o campo de deslocamentos numa laje espessa é necessário determinar três campos independentes, isto é, um campo de deslocamentos transversais e dois campos de rotações.

3.5.2.2. Princípio dos Trabalhos Virtuais

O domínio de uma laje pode ser descrito na forma

$$\Omega = \left\{(x, y, z) \in R^3 : z \in \left[-\frac{h}{2}, +\frac{h}{2}\right], (x, y) \in \alpha \ \subset R^2\right\} \tag{3.124}$$

onde α e *h* representam o plano médio e a espessura da laje, respetivamente. O Princípio dos Trabalhos Virtuais estabelece que o trabalho realizado pelas tensões internas na deformação virtual do corpo é igual ao trabalho realizado pelas forças exteriores nos deslocamentos virtuais dos seus pontos de aplicação.

$$\delta W_I = \delta W_E \tag{3.125}$$

Considerando um elemento finito submetido a um carregamento distribuído no seu plano médio atuando na direção *z*, a relação existente entre trabalho virtual interno e trabalho virtual externo pode ser escrita da seguinte forma.

$$\int_V \delta \boldsymbol{\varepsilon}^T \boldsymbol{\sigma}\, dV = \int_S \delta \mathbf{u}^T \boldsymbol{q}\, dS \tag{3.126}$$

De acordo com o apresentado no Capítulo 2, a relação (3.126), após alguma manipulação algébrica, é equivalente a

$$\int_{-b/2}^{+b/2} \int_{-a/2}^{+a/2} \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, dx\, dy\, \boldsymbol{a} = \int_{-b/2}^{+b/2} \int_{-a/2}^{+a/2} \boldsymbol{N}^T \boldsymbol{q}\, dx\, dy \tag{3.127}$$

em que

$$\boldsymbol{D} = \begin{bmatrix} \frac{h^3}{12}\boldsymbol{D_b} & \mathbf{0} \\ \mathbf{0} & \alpha\, h\, \boldsymbol{D_s} \end{bmatrix}. \tag{3.128}$$

Colocando (3.127) numa forma mais compacta,

$$\boldsymbol{k}\, \boldsymbol{a} = \boldsymbol{f}\,. \tag{3.129}$$

3.5.2.3. Funções de Forma

Na figura 3.13 encontra-se representado um elemento finito retangular de quatro nós com três graus de liberdade por nó, comprimento a, largura b e espessura h. Uma das principais características dos elementos finitos formulados pela teoria de Reissner-Mindlin advém do facto de se poder utilizar diretamente as funções de forma dos elementos finitos bidimensionais. Esta característica dos elementos finitos usados nas lajes pela teoria de Reissner-Mindlin permite, ainda, a utilização de elementos finitos isoparamétricos bidimensionais.

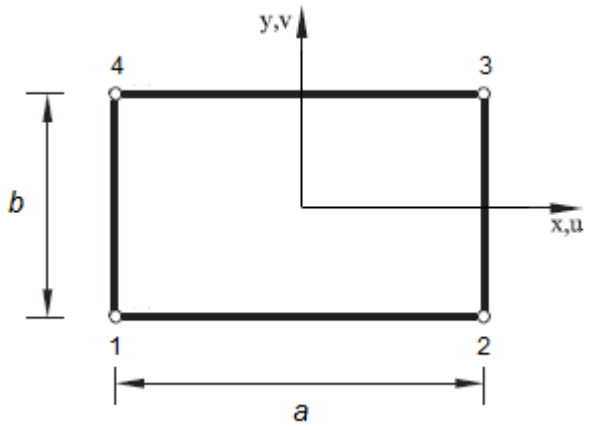


Fig.3.13 – Elemento finito retangular de quatro nós.

3.5.2.4. Campo de Deslocamentos

Da análise da figura 3.14 percebe-se que a rotação adicional da laje faz com que um dado segmento que seja perpendicular ao plano médio da laje no estado indeformado já não permaneça após a deformação da laje. Note-se que quanto mais espessa for a laje mais importante se revela a rotação adicional que este segmento reto sofre.

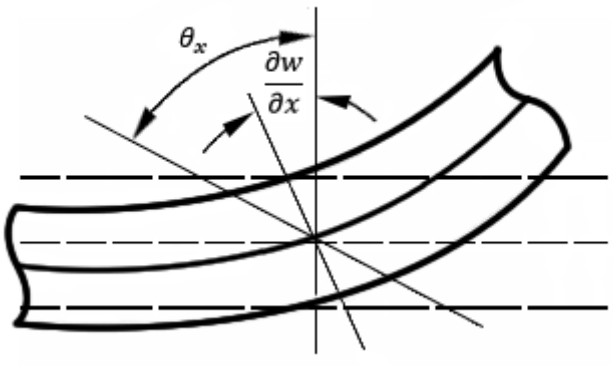


Fig.3.14 – Representação do deslocamento de um ponto no plano *xz*.

Como não existe ortogonalidade do segmento normal ao plano médio, o cálculo do campo de rotações difere do da teoria de Kirchhoff. Assim, para o plano *xz*

$$\theta_x = \frac{\partial w}{\partial x} + \phi_x \tag{3.130}$$

e de forma análoga para o plano *yz*

$$\theta_y = \frac{\partial w}{\partial y} + \phi_y \,. \tag{3.131}$$

Esta diferença garante à teoria de Reissner-Mindlin uma maior flexibilidade quanto à sua empregabilidade, tornando-a, hierarquicamente superior quando comparada com a teoria de Kirchhoff. O campo de deslocamentos num ponto qualquer do elemento de laje pode ser determinado através das igualdades seguintes.

$$u(x,y,z) = -z\,\theta_x(x,y) \tag{3.132a}$$

$$v(x,y,z) = -z\,\theta_y(x,y) \tag{3.132b}$$

$$w(x,y,z) = w(x,y) \tag{3.132c}$$

Nas expressões (3.132), *u*, *v* e *w* são os deslocamentos nas direções *x*, *y* e *z*, respetivamente. Na forma matricial, o campo de deslocamentos para o elemento finito pode ser interpolado a partir dos seus deslocamentos nodais. Assim, para um elemento finito de *n* nós,

$$\begin{Bmatrix} w \\ \theta_x \\ \theta_y \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & 0 & \cdots & N_n & 0 & 0 \\ 0 & N_1 & 0 & \cdots & 0 & N_n & 0 \\ 0 & 0 & N_1 & \cdots & 0 & 0 & N_n \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_{x,1} \\ \theta_{y,1} \\ \vdots \\ w_n \\ \theta_{x,n} \\ \theta_{y,n} \end{Bmatrix} \tag{3.133}$$

ou, de forma mais compacta,

$$\mathbf{u} = \boldsymbol{N}\,\boldsymbol{a} \tag{3.134}$$

onde a matriz $\boldsymbol{N}$ agrupa as funções de forma e o vetor $\boldsymbol{a}$ os deslocamentos nodais.

3.5.2.5. Campo de Deformações

A partir do campo de deslocamentos descrito, a teoria da elasticidade no domínio dos pequenos deslocamentos e deformações estabelece as seguintes relações entre deformação e deslocamento.

$$\varepsilon_x = -z\frac{\partial \theta_x}{\partial x} \tag{3.135a}$$

$$\varepsilon_y = -z\frac{\partial \theta_y}{\partial y} \tag{3.135b}$$

$$\gamma_{xy} = -z\left(\frac{\partial \theta_x}{\partial x} + \frac{\partial \theta_y}{\partial y}\right) \tag{3.135c}$$

$$\gamma_{xz} = -\phi_x \tag{3.135d}$$

$$\gamma_{yz} = -\phi_y \tag{3.135e}$$

Devido às rotações adicionais surgem deformações transversais que são independentes da coordenada *z*. Para caracterizar de forma completa o estado de deformação numa laje espessa, além das curvaturas utilizadas na caracterização do comportamento das lajes finas, passa a ser necessário conhecer o valor das deformações por corte.

Escrevendo matricialmente as componentes do campo de deformações para o elemento finito de *n* nós, chega-se a

$$\hat{\boldsymbol{\varepsilon}} = \begin{bmatrix} 0 & \frac{\partial N_1}{\partial x} & 0 & \cdots & 0 & \frac{\partial N_n}{\partial x} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial y} & \cdots & 0 & 0 & \frac{\partial N_n}{\partial y} \\ \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \cdots & \frac{\partial N_n}{\partial x} & \frac{\partial N_n}{\partial y} & \frac{\partial N_n}{\partial x} \end{bmatrix} \begin{Bmatrix} w_1 \\ \theta_{x,1} \\ \theta_{y,1} \\ \vdots \\ w_n \\ \theta_{x,n} \\ \theta_{y,n} \end{Bmatrix}, \quad \begin{matrix} -N_1 & 0 & \cdots & -N_n & 0 \\ 0 & -N_1 & \cdots & 0 & -N_n \end{matrix} \tag{3.136}$$

que é equivalente a

$$\hat{\boldsymbol{\varepsilon}} = \begin{bmatrix} \boldsymbol{B_b} \\ \boldsymbol{B_s} \end{bmatrix} \tag{3.137}$$

ou, de forma mais compacta,

$$\hat{\boldsymbol{\varepsilon}} = \boldsymbol{B}\,\boldsymbol{a}\,. \tag{3.138}$$

3.5.2.6. Campo de Tensões

Para materiais isotrópicos e com comportamento linear elástico, estabelece-se a seguinte relação constitutiva entre o campo de tensões e o campo de deformações.

$$\boldsymbol{\sigma} = \widehat{\boldsymbol{D}}\,\hat{\boldsymbol{\varepsilon}} \tag{3.139}$$

Após as devidas substituições, (3.139) é equivalente a

$$\boldsymbol{\sigma} = \widehat{\boldsymbol{D}}\,\boldsymbol{B}\,\boldsymbol{a} \tag{3.140}$$

onde a matriz $\widehat{\boldsymbol{D}}$ é designada por matriz de elasticidade e é função das propriedades elásticas do material, isto é, do módulo de elasticidade, *E*, e do coeficiente de Poisson, ν. Para um material isotrópico com comportamento linear elástico,

$$\widehat{\boldsymbol{D}} = \begin{bmatrix} -z\boldsymbol{D_b} & \boldsymbol{0} \\ \boldsymbol{0} & \alpha\boldsymbol{D_s} \end{bmatrix}. \tag{3.141}$$

O parâmetro α em (3.141) tem como finalidade corrigir o efeito da distribuição não uniforme das tensões tangenciais ao longo da espessura da laje. Como as deformações por corte são independentes da coordenada *z*, as tensões tangenciais transversais adquirem uma distribuição constante ao longo da espessura da laje.

De acordo com a teoria da elasticidade, a distribuição exata das tensões tangenciais transversais não é constante ao longo da espessura mas parabólica com valores nulos nas extremidades superior e inferior da laje. Para contornar este problema é utilizado um fator de correção com o objetivo de igualar o trabalho de deformação realizado entre as soluções aproximada (Teoria de Reissner-Mindlin) e exata (Teoria da Elasticidade) [14]. Para um elemento finito de laje com espessura constante adota-se o valor para α de 5/6.

3.5.2.7. Esforços Internos

Os esforços internos são obtidos a partir da integração das tensões ao longo da espessura da laje. Sabendo que as tensões de flexão variam linearmente ao longo da espessura e que as tensões de corte permanecem constantes, tem-se,

$$\begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \int_{-h/2}^{+h/2} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} z \, dz \tag{3.142}$$

e

$$\begin{Bmatrix} V_x \\ V_y \end{Bmatrix} = \int_{-h/2}^{+h/2} \begin{Bmatrix} \tau_{xz} \\ \tau_{yz} \end{Bmatrix} dz \, . \tag{3.143}$$

Em (3.142), M_x, M_y e M_{xy} são, respetivamente, os momentos fletores por unidade de comprimento nas direções *x* e *y* e o momento de torção, enquanto que em (3.143), V_x e V_y são os esforços transversos por unidade de comprimento nas direções *x* e *y*. Calculando os integrais (3.142) e (3.143),

$$\boldsymbol{M} = -\frac{h^3}{12} \boldsymbol{D_b} \, \boldsymbol{B_b} \, \boldsymbol{a} \tag{3.144}$$

e

$$\boldsymbol{V} = \alpha \, h \, \boldsymbol{D_s} \, \boldsymbol{B_s} \, \boldsymbol{a} \, . \tag{3.145}$$

Para um material isotrópico com comportamento linear elástico,

$$\boldsymbol{D_b} = \begin{bmatrix} \frac{E}{1-\nu^2} & \frac{E\nu}{1-\nu^2} & 0 \\ \frac{E\nu}{1-\nu^2} & \frac{E}{1-\nu^2} & 0 \\ 0 & 0 & \frac{E}{2(1+\nu)} \end{bmatrix} \tag{3.146}$$

e

$$\boldsymbol{D_s} = \begin{bmatrix} \frac{E}{2(1+\nu)} & 0 \\ 0 & \frac{E}{2(1+\nu)} \end{bmatrix}. \tag{3.147}$$

Dada a sua simplicidade e facilidade de utilização, os modelos de elementos finitos para lajes espessas são os mais utilizados, mesmo em situações para as quais se pode deixar de considerar a laje como espessa. Contudo, há situações em que a diminuição da espessura da laje coloca alguns problemas no que toca à utilização de elementos finitos baseados na teoria de Reissner-Mindlin.

Um dos fenómenos mais gravosos que podem surgir nesta circunstância é o chamado *locking*. Este fenómeno de *locking* está associado a uma rigidez excessiva do modelo numérico utilizado. A sua resolução passa pela subavaliação dos elementos das matrizes de rigidez elementares, ou seja, realizando uma integração numérica reduzida ou uma integração numérica seletiva.

A formulação apresentada não permite a inclusão de esforços normais atuantes no plano da laje. A combinação do elemento finito de laje pela teoria de Reissner-Mindlin com o elemento finito do Estado Plano de Tensão permite obter o designado elemento finito laminar. O elemento finito laminar possui cinco graus de liberdade sendo dois vindos do elemento finito bidimensional usado no Estado Plano de Tensão e três do elemento de laje pela teoria de Reissner-Mindlin. Devido às suas características, este elemento finito é empregue no estudo de lajes pré-esforçadas e de estruturas laminares compósitas. O elemento finito laminar, além do estudo de lajes, pode ser adaptado ao estudo de cascas.

4

ELEMENTOS FINITOS COM SUBSTITUIÇÃO DE VARIÁVEL

4.1. SUBSTITUIÇÃO DE VARIÁVEL EM UMA DIMENSÃO

De forma a normalizar o comprimento de elemento finito é adotado um sistema de coordenadas genérico. Este sistema de coordenadas, designado por *Or*, definido para normalizar o comprimento do elemento finito impõe um comprimento $L = 2$ e coordenadas dos nós extremos $r_1 = -1$ e $r_n = +1$.

A utilização de um referencial normalizado permite a automatização do cálculo numérico. Esta automatização consegue-se devido ao facto dos domínios de integração de cada elemento finito serem sempre os mesmos, independentemente do tamanho e do grau de distorção de cada elemento. Assim, as funções de forma a utilizar deixam de ser influenciadas pela geometria de cada elemento finito passando a depender apenas da representação dos elementos no sistema de coordenadas normalizado.

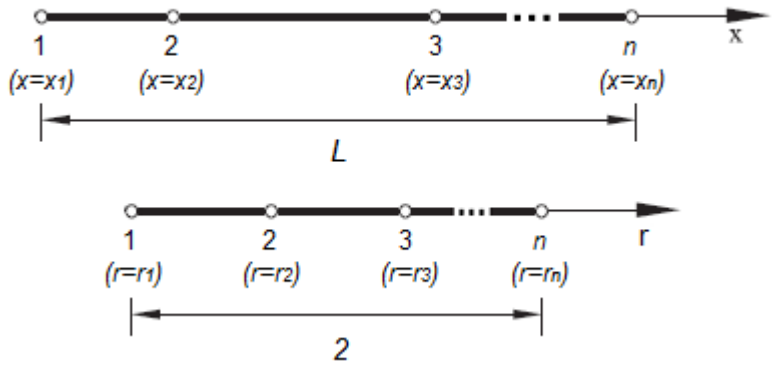


Fig.4.1 – Elemento finito unidimensional de *n* nós com substituição de variável.

Considere-se um elemento finito de comprimento *L*, área da secção transversal *A*, módulo de elasticidade *E* e sujeito a um carregamento axial *p* distribuído ao longo de todo o seu comprimento. As coordenadas do elemento finito real no referencial *Ox* são armazenadas no vetor $\overline{\boldsymbol{x}}$, sendo

$$\overline{\boldsymbol{x}} = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix} \tag{4.1}$$

e as coordenadas do elemento finito normalizado no referencial *Or* são armazenadas no vetor $\overline{\boldsymbol{r}}$. Para um elemento finito unidimensional de *n* nós

$$\bar{\boldsymbol{r}} = \begin{Bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{Bmatrix}. \tag{4.2}$$

De modo a automatizar o cálculo da matriz de rigidez elementar e do vetor de forças nodais equivalentes é conveniente realizar uma substituição de variável do seguinte tipo.

$$x = x(r) \tag{4.3}$$

As coordenadas dos nós do elemento finito são interpoladas com recurso à seguinte expressão geral.

$$x(r) = N_1(r)x_1 + N_2(r)x_2 + \cdots + N_n(r)x_n \tag{4.4}$$

As funções de forma para o elemento finito normalizado são deduzidas em função de *r* com as coordenadas armazenas no vetor $\bar{\boldsymbol{r}}$.

A matriz de rigidez de um elemento finito unidimensional com substituição de variável é obtida pelo integral

$$\boldsymbol{k} = \int_{-1}^{+1} \boldsymbol{B}^T E\, \boldsymbol{B}\, A\, J\, dr \tag{4.5}$$

em que *J* é o determinante jacobiano. O determinante jacobiano tem como função definir a relação existente para a dimensão do elemento finito entre os dois referenciais de modo a se poder usar os limites de integração considerados em (4.5). Se os parâmetros *E* e *A* não forem constantes, estes podem ser interpolados com as mesmas funções de forma com que foram interpoladas as coordenadas dos nós, ou seja,

$$E(r) = N_1(r)E_1 + N_2(r)E_2 + \ldots + N_n(r)E_n \tag{4.6}$$

e

$$A(r) = N_1(r)A_1 + N_2(r)A_2 + \ldots + N_n(r)A_n \tag{4.7}$$

onde E_i e A_i são os valores para o nó *i* do módulo de elasticidade e da área da secção transversal do elemento finito, respetivamente. O determinante jacobiano é obtido por derivação de (4.4).

$$J = \frac{\partial x}{\partial r} = \frac{\partial N_1}{\partial r} x_1 + \frac{\partial N_2}{\partial r} x_2 + \ldots + \frac{\partial N_n}{\partial r} x_n \tag{4.8}$$

Para avaliar o integral (4.5) é ainda necessário definir a matriz $\boldsymbol{B}$ em função de *r*. Desta forma, existe a necessidade de calcular as derivadas das funções de forma em ordem a *x* mas expressas em função de *r*. Recorrendo à regra de derivação em cadeia das funções de forma para um nó genérico *i*, chega-se à expressão geral

$$\frac{\partial N_i}{\partial r} = \frac{\partial x}{\partial r} \frac{\partial N_i}{\partial x}. \tag{4.9}$$

Assim, os termos da matriz $\boldsymbol{B}$ são determinados pela expressão geral

$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial r} J^{-1}. \tag{4.10}$$

Calculadas todas as derivadas das funções de forma procede-se à sua distribuição na matriz $\boldsymbol{B}$ de acordo com o formato apresentado no Capítulo 3. Apresentados todos os passos necessários para calcular a matriz de rigidez do elemento finito com recurso à substituição de variável do tipo (4.3) é pormenorizado o cálculo do vetor de forças nodais equivalentes.

Para um elemento finito unidimensional carregado axialmente ao longo do seu comprimento, o vetor de forças nodais equivalentes com substituição de variável é obtido pelo seguinte integral.

$$\boldsymbol{f} = \int_{-1}^{+1} \boldsymbol{N}^T p \, J \, dr \tag{4.11}$$

Em suma, relativamente à discretização com elementos finitos unidimensionais, quando as propriedades do material, a área da secção transversal e/ou o carregamento axial variam ao longo do seu desenvolvimento é preferível fazer uma discretização que atenda a estas variações em vez de se recorrer a elementos finitos de ordens superiores.

4.2. SUBSTITUIÇÃO DE VARIÁVEL EM DUAS DIMENSÕES

A utilização de elementos finitos de ordem superior com lados curvos ou elementos finitos distorcidos dificulta o cálculo das matrizes de rigidez elementares e dos vetores de forças nodais equivalentes. Estas dificuldades são ultrapassadas através do uso de elementos finitos isoparamétricos e de integração numérica.

Na figura 4.2 é mostrada a transformação entre as coordenadas cartesianas (x, y) e as coordenadas naturais (r, s) no plano de referência para elementos finitos quadriláteros. Esta transformação é frequentemente designada por mapeamento isoparamétrico. A espessura do elemento finito é designada por h e também pode ser função das coordenadas x e y.

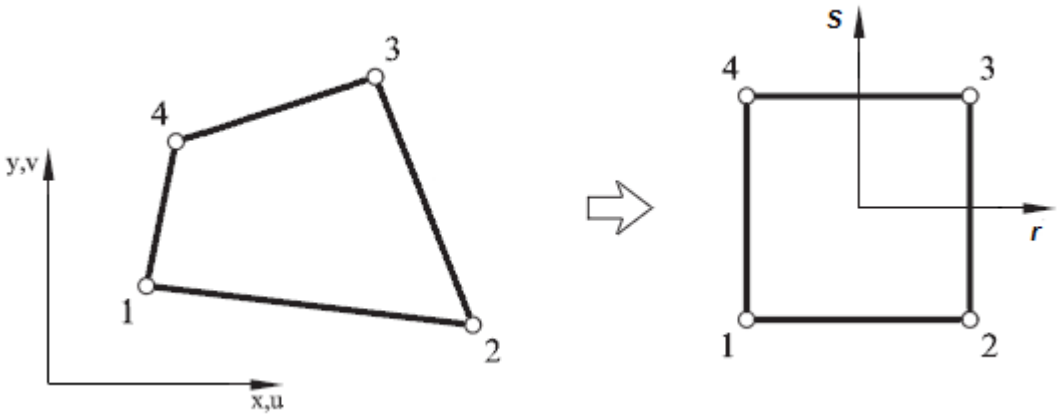


Fig.4.2 – Transformação de um elemento real para um elemento normalizado (Adaptado de [13]).

As coordenadas do elemento finito real bidimensional no referencial *Oxy* são armazenadas na matriz $\overline{\boldsymbol{x}}$, em que

$$\overline{\boldsymbol{x}} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \tag{4.12}$$

e as coordenadas do elemento finito normalizado bidimensional no referencial *Ors* são armazenadas na matriz $\overline{\boldsymbol{r}}$, sendo

$$\overline{\boldsymbol{r}} = \begin{bmatrix} r_1 & s_1 \\ r_2 & s_2 \\ \vdots & \vdots \\ r_n & s_n \end{bmatrix}. \tag{4.13}$$

De forma a facilitar o cálculo da matriz de rigidez e do vetor de forças nodais equivalentes do elemento finito é efetuada uma substituição de variáveis do seguinte tipo.

$$\begin{cases} x = x(r,s) \\ y = y(r,s) \end{cases} \tag{4.14}$$

Esta substituição de variáveis permite o estudo de elementos finitos com geometria irregular com as mesmas funções de forma deduzidas para um elemento finito de geometria regular. Este elemento finito de geometria normalizada corresponde a um elemento finito bidimensional quadrado cuja dimensão dos lados vale dois.

A interpolação de coordenadas em duas dimensões baseia-se na seguinte aproximação para a geometria do elemento finito de *n* nós.

$$x = \sum_{i=1}^{n} N_i(r,s)x_i \tag{4.15a}$$

$$y = \sum_{i=1}^{n} N_i(r,s)y_i \tag{4.15b}$$

Relativamente ao sistema de eixos *Ors*, as coordenadas de qualquer ponto do elemento finito são definidas em $r \in [-1,+1]$ e $s \in [-1,+1]$, independentemente do grau de distorção do mesmo e dos seus lados serem ou não curvos. As componentes de deslocamento podem ser determinadas a partir da interpolação dos deslocamentos nodais da forma seguinte.

$$u(x.y) = \sum_{i=1}^{n} N_i(r,s)u_i \tag{4.16a}$$

$$v(x,y) = \sum_{i=1}^{n} N_i(r,s)v_i \tag{4.16b}$$

As funções de forma do elemento finito de geometria normalizada são deduzidas em função de *r* e *s* com as coordenadas armazenas na matriz $\bar{\boldsymbol{r}}$.

A matriz de rigidez de um elemento finito quadrilátero com substituição de variável é obtida pelo integral

$$\boldsymbol{k} = \int_{-1}^{+1}\int_{-1}^{+1} \boldsymbol{B}^T\boldsymbol{D}\,\boldsymbol{B}\,h\,J\,dr\,ds \tag{4.17}$$

em que *J* é o determinante da matriz jacobiana. Para se calcular o integral (4.17) é necessário que todos os termos da função integranda dependam das variáveis *r* e *s*. Caso os parâmetros *E* e ν da matriz $\boldsymbol{D}$ e *h* não sejam constantes, os seus valores podem ser interpolados com recurso às mesmas funções de forma com que são interpoladas as coordenadas dos nós.

$$E(r,s) = N_1(r,s)E_1 + N_2(r,s)E_2 + \ldots + N_n(r,s)E_n \tag{4.18}$$

$$\nu(r,s) = N_1(r,s)\nu_1 + N_2(r,s)\nu_2 + \ldots + N_n(r,s)\nu_n \tag{4.19}$$

$$h(r,s) = N_1(r,s)h_1 + N_2(r,s)h_2 + \ldots + N_n(r,s)h_n \tag{4.20}$$

Note-se que pode ser preferível adotar uma discretização que atenda a esta variação de geometria e propriedades do material em vez de interpolar os seus valores.

A matriz jacobiana correspondente à transformação definida em (4.14) é definida por

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial x}{\partial r} & \dfrac{\partial x}{\partial s} \\ \dfrac{\partial y}{\partial r} & \dfrac{\partial y}{\partial s} \end{bmatrix}. \tag{4.21}$$

A obtenção dos termos da matriz jacobiana é realizada através da multiplicação das coordenadas dos nós do elemento finito real pelas derivadas das funções de forma do elemento finito normalizado. Na forma matricial, a obtenção da matriz jacobiana é realizada com recurso à expressão geral

$$\begin{bmatrix} \dfrac{\partial x}{\partial r} & \dfrac{\partial x}{\partial s} \\ \dfrac{\partial y}{\partial r} & \dfrac{\partial y}{\partial s} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \end{bmatrix} \begin{bmatrix} \dfrac{\partial N_1}{\partial r} & \dfrac{\partial N_1}{\partial s} \\ \dfrac{\partial N_2}{\partial r} & \dfrac{\partial N_2}{\partial s} \\ \vdots & \vdots \\ \dfrac{\partial N_n}{\partial r} & \dfrac{\partial N_n}{\partial s} \end{bmatrix} \tag{4.22}$$

ou, de forma mais compacta,

$$\boldsymbol{J} = \overline{\boldsymbol{x}}^T \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{r}}. \tag{4.23}$$

A matriz $\boldsymbol{B}$ também é dependente das variáveis *r* e *s*. De modo a ser possível calcular o integral (4.17) é necessário obter as derivadas das funções de forma que compõem a matriz $\boldsymbol{B}$ em ordem a *x* e *y* mas expressas em função de *r* e *s*. Desta forma, recorrendo à regra de derivação em cadeia das funções de forma para um nó genérico *i*, chega-se às expressões seguintes.

$$\frac{\partial N_i}{\partial r} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial r} \tag{4.24a}$$

$$\frac{\partial N_i}{\partial s} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial s} \tag{4.24b}$$

Estendendo aos *n* nós do elemento finito e colocando na forma matricial, a regra de derivação em cadeia fica

$$\begin{bmatrix} \dfrac{\partial N_1}{\partial r} & \dfrac{\partial N_1}{\partial s} \\ \dfrac{\partial N_2}{\partial r} & \dfrac{\partial N_2}{\partial s} \\ \vdots & \vdots \\ \dfrac{\partial N_n}{\partial r} & \dfrac{\partial N_n}{\partial s} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_1}{\partial y} \\ \dfrac{\partial N_2}{\partial x} & \dfrac{\partial N_2}{\partial y} \\ \vdots & \vdots \\ \dfrac{\partial N_n}{\partial x} & \dfrac{\partial N_n}{\partial y} \end{bmatrix} \begin{bmatrix} \dfrac{\partial x}{\partial r} & \dfrac{\partial x}{\partial s} \\ \dfrac{\partial y}{\partial r} & \dfrac{\partial y}{\partial s} \end{bmatrix} \tag{4.25}$$

ou, de forma mais compacta,

$$\frac{\partial \boldsymbol{N}}{\partial \boldsymbol{r}} = \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{x}} \boldsymbol{J}. \tag{4.26}$$

Assumindo que a matriz $\boldsymbol{J}$ tem inversa, pode-se determinar que

$$\frac{\partial \boldsymbol{N}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{r}} \boldsymbol{J}^{-1}. \tag{4.27}$$

É possível garantir que a inversa da matriz jacobiana existe desde que os elementos finitos reais não apresentem grandes distorções. No entanto, esta correspondência deixa de se verificar em situações em que dois dos lados de um elemento finito com lados retos compartilham um ângulo interno maior que π

radianos [21]. Calculadas todas as derivadas das funções de forma procede-se à sua distribuição na matriz $\boldsymbol{B}$ de acordo com o formato apresentado no Capítulo 3.

O vetor de forças nodais equivalentes de um elemento finito bidimensional com substituição de variável, sujeito a um carregamento por unidade de superfície no plano *Ors*, é obtido pelo seguinte integral.

$$\boldsymbol{f} = \int_{-1}^{+1} \int_{-1}^{+1} \boldsymbol{N}^T \boldsymbol{q}\, h\, J\, dr\, ds \tag{4.28}$$

Na situação em que o elemento finito, em estudo, está sujeito a carregamentos distribuídos nos seus bordos, a expressão para o cálculo do vetor de forças nodais equivalentes reduz-se a um integral de linha. Para determinar o valor das forças nodais equivalentes para um carregamento deste tipo, é necessário relacionar o comprimento de um segmento infinitesimal definido no referencial isoparamétrico com o correspondente referencial cartesiano. Isto é feito de forma análoga ao apresentado para os elementos finitos unidimensionais.

4.3. SUBSTITUIÇÃO DE VARIÁVEL EM TRÊS DIMENSÕES

A modelação de problemas com geometrias complexas exige a utilização de elementos finitos distorcidos. Atendendo a isto, a utilização de uma formulação isoparamétrica torna a análise por elementos finitos mais flexível. No sistema de eixos *Orst*, as coordenadas de qualquer ponto do elemento finito são definidas em r, s, $t \in [-1,+1]$, independentemente do grau de distorção do mesmo e dos seus lados serem ou não curvos.

Na figura 4.3 é mostrada a transformação entre as coordenadas cartesianas (x, y, z) e as coordenadas naturais (r, s, t) para elementos finitos hexaédricos.

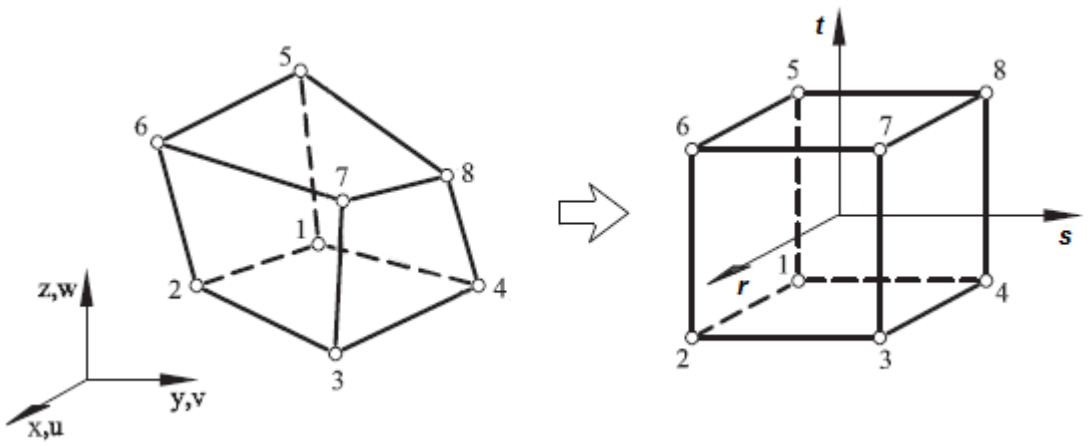


Fig.4.3 – Transformação de um elemento real para um elemento normalizado (Adaptado de [13]).

De modo a automatizar o cálculo da matriz de rigidez elementar e do vetor de forças nodais equivalentes é conveniente realizar uma substituição de variável do seguinte tipo.

$$\begin{cases} x = x(r,s,t) \\ y = y(r,s,t) \\ z = z(r,s,t) \end{cases} \tag{4.29}$$

As coordenadas do elemento finito real com n nós no referencial *Oxyz* são armazenadas na matriz $\overline{\boldsymbol{x}}$, em que

$$\overline{\boldsymbol{x}} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \tag{4.30}$$

e as coordenadas do elemento finito normalizado tridimensional no referencial *Orst* são armazenadas na matriz $\overline{\boldsymbol{r}}$, sendo

$$\overline{\boldsymbol{r}} = \begin{bmatrix} r_1 & s_1 & t_1 \\ r_2 & s_2 & t_2 \\ \vdots & \vdots & \vdots \\ r_n & s_n & t_n \end{bmatrix}. \tag{4.31}$$

As funções de forma do elemento finito de geometria normalizada são deduzidas em função de *r*, *s* e *t* com as coordenadas armazenas na matriz $\overline{\boldsymbol{r}}$.

No caso dos elementos finitos hexaédricos de ordem superior ao apresentado na figura 4.3, são necessárias funções de forma de ordem superior. Estes elementos finitos têm a vantagem de permitir modelar geometrias curvas recorrendo a discretizações com um menor número de elementos finitos, em comparação com a utilização de elementos finitos de baixa ordem. Contudo, dada a utilização de um maior número de nós e um maior grau das funções de forma, estes elementos finitos tornam-se mais dispendiosos computacionalmente.

A interpolação de coordenadas em três dimensões baseia-se na seguinte aproximação para a geometria do elemento finito de *n* nós.

$$x = \sum_{i=1}^{n} N_i(r,s,t)x_i \tag{4.32a}$$

$$y = \sum_{i=1}^{n} N_i(r,s,t)y_i \tag{4.32b}$$

$$z = \sum_{i=1}^{n} N_i(r,s,t)z_i \tag{4.32c}$$

A matriz de rigidez de um elemento finito hexaédrico com substituição de variável é obtida pelo integral

$$\boldsymbol{k} = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, J\, dr\, ds\, dt \tag{4.33}$$

em que J é o determinante da matriz jacobiana. Para se calcular o integral (4.33) é necessário que todos os termos da função integranda dependam das variáveis *r*, *s* e *t*.

A matriz jacobiana correspondente à transformação definida em (4.29) é definida da seguinte forma.

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial x}{\partial r} & \dfrac{\partial x}{\partial s} & \dfrac{\partial x}{\partial t} \\ \dfrac{\partial y}{\partial r} & \dfrac{\partial y}{\partial s} & \dfrac{\partial y}{\partial t} \\ \dfrac{\partial z}{\partial r} & \dfrac{\partial z}{\partial s} & \dfrac{\partial z}{\partial t} \end{bmatrix} \tag{4.34}$$

A obtenção dos termos da matriz jacobiana é realizada através da multiplicação das coordenadas dos nós do elemento finito real pelas derivadas das funções de forma do elemento finito normalizado. Na forma matricial, a obtenção da matriz jacobiana é realizada com recurso à expressão geral

$$\begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial s} & \frac{\partial z}{\partial t} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ y_1 & y_2 & \cdots & y_n \\ z_1 & z_2 & \cdots & z_n \end{bmatrix} \begin{bmatrix} \frac{\partial N_1}{\partial r} & \frac{\partial N_1}{\partial s} & \frac{\partial N_1}{\partial t} \\ \frac{\partial N_2}{\partial r} & \frac{\partial N_2}{\partial s} & \frac{\partial N_2}{\partial t} \\ \vdots & \vdots & \vdots \\ \frac{\partial N_n}{\partial r} & \frac{\partial N_n}{\partial s} & \frac{\partial N_n}{\partial t} \end{bmatrix} \tag{4.35}$$

ou, de forma mais compacta,

$$\boldsymbol{J} = \overline{\boldsymbol{x}}^T \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{r}} . \tag{4.36}$$

De modo a ser possível calcular o integral (4.33) é necessário obter as derivadas das funções de forma que compõem a matriz $\boldsymbol{B}$ em ordem a *x*, *y* e *z* mas expressas em função de *r*, *s* e *t*. Assim, recorrendo à regra de derivação em cadeia das funções de forma para um nó genérico *i*, chega-se às expressões seguintes.

$$\frac{\partial N_i}{\partial r} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial r} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial r} \tag{4.37a}$$

$$\frac{\partial N_i}{\partial s} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial s} \tag{4.37b}$$

$$\frac{\partial N_i}{\partial t} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial t} \tag{4.37c}$$

Na forma forma matricial, a regra de derivação em cadeia para um elemento finito de *n* nós fica

$$\begin{bmatrix} \frac{\partial N_1}{\partial r} & \frac{\partial N_1}{\partial s} & \frac{\partial N_1}{\partial t} \\ \frac{\partial N_2}{\partial r} & \frac{\partial N_2}{\partial s} & \frac{\partial N_2}{\partial t} \\ \vdots & \vdots & \vdots \\ \frac{\partial N_n}{\partial r} & \frac{\partial N_n}{\partial s} & \frac{\partial N_n}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial z} \\ \frac{\partial N_2}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial z} \\ \vdots & \vdots & \vdots \\ \frac{\partial N_n}{\partial x} & \frac{\partial N_n}{\partial y} & \frac{\partial N_n}{\partial z} \end{bmatrix} \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \\ \frac{\partial z}{\partial r} & \frac{\partial z}{\partial s} & \frac{\partial z}{\partial t} \end{bmatrix} \tag{4.38}$$

ou, de forma mais compacta,

$$\frac{\partial \boldsymbol{N}}{\partial \boldsymbol{r}} = \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{x}} \boldsymbol{J} . \tag{4.39}$$

Assumindo que a matriz $\boldsymbol{J}$ tem inversa, então

$$\frac{\partial \boldsymbol{N}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{N}}{\partial \boldsymbol{r}} \boldsymbol{J}^{-1} . \tag{4.40}$$

Calculadas todas as derivadas das funções de forma com recurso a (4.40) procede-se à sua distribuição na matriz $\boldsymbol{B}$ de acordo com a sua estrutura. Uma vez que todos os componentes da função integranda se encontram definidos em função de *r*, *s* e *t* é agora possível proceder ao cálculo da matriz de rigidez do elemento finito utilizando (4.33).

O vetor de forças nodais equivalentes de um elemento finito hexaédrico com substituição de variável sujeito a um carregamento de volume é obtido pelo integral seguinte.

$$\boldsymbol{f} = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \boldsymbol{N}^T \boldsymbol{b}\, J\, dr\, ds\, dt \tag{4.41}$$

5
INTEGRAÇÃO NUMÉRICA

5.1. SELEÇÃO DO MÉTODO DE INTEGRAÇÃO NUMÉRICA

A resolução de problemas pelo Método dos Elementos Finitos implica, muitas vezes, o cálculo de integrais relativamente complexos. Assim, é útil recorrer a métodos aproximados, mais eficientes e de fácil implementação computacional, para se calcular esses integrais.

A dificuldade que se coloca na utilização da integração numérica consiste na escolha do método de integração. A matemática fornece inúmeros métodos para realizar integração numérica. O que distingue esses métodos, além da sua complexidade, é a velocidade de convergência, ou seja, o número de iterações que é necessário realizar até se obter o valor exato do integral. Do ponto de vista de implementação computacional, o melhor método é aquele que, para além de fácil implementação, permite conhecer o valor exato, ou com precisão suficiente, realizando computacionalmente poucos cálculos.

Um dos métodos que reúne estes requisitos é a quadratura de Gauss-Legendre. Além destes motivos, outros associados à escolha dos pontos para se calcular as tensões e os esforços internos, fazem da quadratura de Gauss-Legendre a mais utilizada e divulgada no âmbito do Método dos Elementos Finitos.

5.2. QUADRATURA DE GAUSS-LEGENDRE

5.2.1. FUNDAMENTOS DA QUADRATURA DE GAUSS-LEGENDRE

A quadratura de Gauss-Legendre é um dos métodos mais expeditos para realizar a tarefa de integração numérica. Segundo este método, de um modo geral, pode-se determinar o integral de uma função genérica através de

$$\int_a^b f(x)\,dx = \sum_{i=1}^{\infty} w_i f(x_i) \approx \sum_{i=1}^{n} w_i f(x_i) \tag{5.1}$$

onde n corresponde ao número de pontos de integração e w_i ao peso de integração associado ao ponto de integração x_i. De modo a agilizar o procedimento de integração numérica é habitual definir o integral de uma função genérica para o intervalo [-1, +1]. A adaptação dos limites de integração de qualquer intervalo [a, b] para o intervalo [-1, +1], usado na quadratura de Gauss-Legendre, é realizada através da seguinte manipulação.

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{+1} f\left(\frac{b-a}{2}x_i + \frac{b+a}{2}\right)dx \tag{5.2}$$

Atendendo a (5.1),

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{b+a}{2}\right). \tag{5.3}$$

Com n pontos de integração obtém-se o valor exato do integral de um polinómio de grau $p = 2n - 1$, ou inferior. Assim, quando se pretende conhecer a solução exata de um polinómio de grau p, o número de pontos de integração a utilizar é $n = (p + 1) / 2$, ou superior. Caso o valor de p seja par, deve-se substituir o seu valor pelo número ímpar imediatamente superior.

5.2.2. Integração Numérica em Uma Dimensão

O objetivo da integração numérica de funções em uma dimensão é transformar a formulação integral contínua sobre o domínio físico do elemento finito em somatórios avaliados num conjunto discreto de pontos sobre esse elemento finito. Matematicamente é equivalente a calcular o integral de uma função genérica definida em termos de coordenadas naturais Or através da aproximação

$$\int_{-1}^{+1} f(r)\,dr \approx \sum_{i=1}^{n} w_i f(r_i) \tag{5.4}$$

onde n corresponde ao número de pontos de integração e w_i é o peso de integração associado ao ponto de integração r_i. Na tabela 5.1 são mostrados os pares de valores coordenada-peso de integração para as cinco primeiras ordens de integração definidas para o intervalo [-1, +1].

Tabela 5.1 – Posições dos pontos de amostragem e respetivos pesos.

n	w_i	r_i
1	2.0	0.0
2	1.0000000000	-0.5773502692
	1.0000000000	0.5773502692
3	0.5555555556	-0.7745966692
	0.8888888888	0.0000000000
	0.5555555556	0.7745966692
4	0.3478548451	-0.8611363116
	0.6521451549	-0.3399810436
	0.6521451549	0.3399810436
	0.3478548451	0.8611363116
5	0.2369268851	-0.9061798459
	0.4786286705	-0.5384693101
	0.5688888889	0.0000000000
	0.4786286705	0.5384693101
	0.2369268851	0.9061798459

5.2.3. INTEGRAÇÃO NUMÉRICA EM DUAS DIMENSÕES

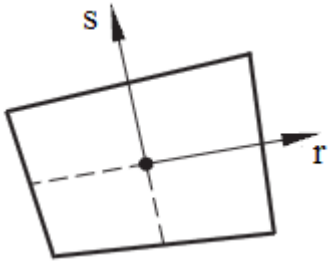
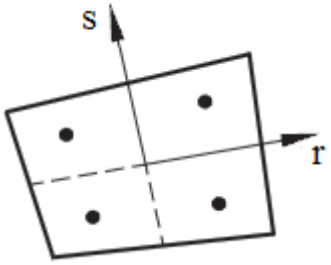
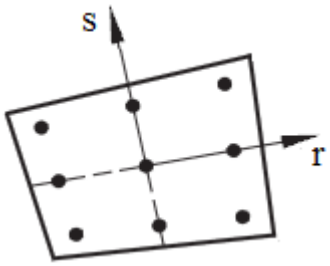
O conceito de integração numérica é agora estendido ao domínio bidimensional. O objetivo é transformar o integral duplo de uma função genérica definida em termos de coordenadas naturais *Ors* num duplo somatório.

$$\int_{-1}^{+1}\int_{-1}^{+1} f(r,s)\,dr\,ds \approx \sum_{i=1}^{n_r}\sum_{j=1}^{n_s} w_i\,w_j f(r_i,s_j) \tag{5.5}$$

Em (5.5) n_r e n_s correspondem ao número de pontos de integração a adotar ao longo das direções *Or* e *Os*, respetivamente. Por outro lado, r_i e s_j são as coordenadas no referencial natural do ponto de amostragem (i, j). Finalmente, w_i e w_j são os pesos associados a cada ponto de integração.

Na tabela 5.2 encontram-se as posições e os pesos correspondentes aos pontos de integração que foram obtidos do modo análogo ao caso da integração numérica em uma dimensão em que se aplicou o procedimento para cada uma das duas direções. A expressão (5.5) permite realizar a integração numérica de funções bidimensionais definidas para os elementos finitos quadriláteros.

Tabela 5.2 – Pesos e posições dos pontos de amostragem para quadriláteros.

n	w_i	w_j	r_i	s_j
	2.0	2.0	0.0	0.0
	1.0000000000	1.0000000000	-0.5773502692	-0.5773502692
	1.0000000000	1.0000000000	0.5773502692	-0.5773502692
	1.0000000000	1.0000000000	0.5773502692	0.5773502692
	1.0000000000	1.0000000000	-0.5773502692	0.5773502692
	0.5555555556	0.5555555556	-0.7745966692	-0.7745966692
	0.8888888888	0.5555555556	0.0000000000	-0.7745966692
	0.5555555556	0.5555555556	0.7745966692	-0.7745966692
	0.5555555556	0.8888888888	0.7745966692	0.0000000000
	0.5555555556	0.5555555556	0.7745966692	0.7745966692
	0.8888888888	0.5555555556	0.0000000000	0.7745966692
	0.5555555556	0.5555555556	-0.7745966692	0.7745966692
	0.5555555556	0.8888888888	-0.7745966692	0.0000000000
	0.8888888888	0.8888888888	0.0000000000	0.0000000000

Apesar de se procurar obter o valor exato da função integranda, em algumas situações é conveniente integrar as funções com um número de pontos inferior ao mínimo necessário para se obter o valor exato do integral. A escolha do número de pontos de integração, também designados por pontos de Gauss, influencia diretamente o custo computacional da análise por elementos finitos. Em teoria, subir a ordem de integração conduz a resultados sucessivamente mais corretos. Da mesma forma que a utilização de poucos pontos de integração, no limite apenas um, pode conduzir a resultados incorretos ou mesmo tornar as matrizes de rigidez elementares singulares.

A integração numérica designa-se completa quando se utiliza um número mínimo de pontos de modo a obter o valor exato do integral. A integração diz-se reduzida quando se considera um número de pontos de integração inferior ao mínimo necessário para efetuar a integração completa. A integração reduzida é muitas vezes usada propositadamente para reduzir o excesso de rigidez relativamente à solução física correta obtida quando se realiza integração completa. Assim, a integração reduzida é uma das primeiras técnicas utilizadas para reduzir a rigidez dos elementos finitos.

Alternativamente ao uso da integração reduzida, em algumas formulações de elementos finitos, é habitual realizar uma integração numérica seletiva. Em elementos finitos de laje formulados pela teoria de Reissner-Mindlin o uso de integração numérica seletiva corresponde a usar integração completa para integrar a parcela de flexão e integração reduzida para integrar a parcela de corte.

5.2.4. Integração Numérica em Três Dimensões

À semelhança do exposto nos pontos anteriores, os integrais em domínios tridimensionais podem ser aproximados efetuando um somatório de termos constituídos pelos valores da função integranda num certo número discreto de pontos predefinidos, multiplicados por pesos convenientes. Assim, para o caso de elementos hexaédricos obtém-se

$$\int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} f(r,s,t)\,dr\,ds\,dt \approx \sum_{i=1}^{n_r}\sum_{j=1}^{n_s}\sum_{k=1}^{n_t} w_i\,w_j\,w_k f(r_i,s_j,t_k) \tag{5.6}$$

em que n_r, n_s e n_t correspondem ao número de pontos de integração a adotar ao longo das direções *Or*, *Os* e *Ot*, respetivamente. Por outro lado, r_i, s_j e t_k são as coordenadas no referencial natural do ponto de amostragem (i, j, k). Finalmente, w_i, w_j e w_k são os pesos a considerar em função dos pontos de integração escolhidos.

O número de pontos de integração a usar está associado ao grau máximo do polinómio a integrar, ou seja, elementos finitos com ordens de integração superiores exigem mais pontos de integração. No contexto das formulações tridimensionais, a utilização de técnicas de integração reduzida tem por objetivo melhorar o desempenho dos elementos finitos, nomeadamente, através da redução da rigidez.

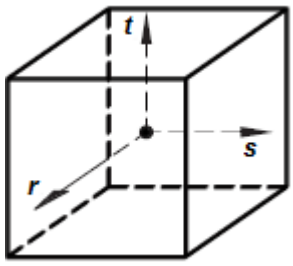
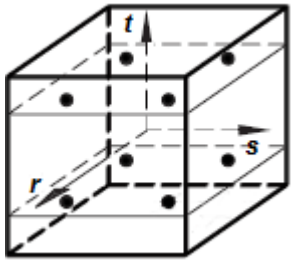
Na tabela 5.3 são mostrados os pesos e as coordenadas dos pontos de integração definidos para as duas primeiras ordens de integração de elementos finitos hexaédricos.

5.3. Outras Quadraturas

Qualquer procedimento de integração é perfeitamente válido. Contudo, como já anteriormente referido, a escolha do procedimento de integração deve passar por encontrar a metodologia que conduz à redução do número de cálculos computacionais. A quadratura de Gauss-Legendre é um dos poucos métodos que reúne simplicidade e eficiência.

Como os programadores procuram implementar métodos que conduzem à redução do número de operações computacionais, quadraturas semelhantes à quadratura de Gauss-Legendre mas que exigem que a função integranda seja avaliada em mais pontos, normalmente, não são preferíveis. Contudo, em contextos de integração reduzida, estas quadraturas podem fornecer melhores resultados que a quadratura de Gauss-Legendre.

Tabela 5.3 – Pesos e posições dos pontos de amostragem para hexaedros.

n	$w_i = w_j = w_k$	r_i	s_j	t_k
	2.0	0.0	0.0	0.0
	1.0000000000	-0.5773502692	-0.5773502692	-0.5773502692
	1.0000000000	0.5773502692	-0.5773502692	-0.5773502692
	1.0000000000	0.5773502692	0.5773502692	-0.5773502692
	1.0000000000	-0.5773502692	0.5773502692	-0.5773502692
	1.0000000000	-0.5773502692	-0.5773502692	0.5773502692
	1.0000000000	0.5773502692	-0.5773502692	0.5773502692
	1.0000000000	0.5773502692	0.5773502692	0.5773502692
	1.0000000000	-0.5773502692	0.5773502692	0.5773502692

5.4. AVALIAÇÃO DA MATRIZ DE DEFORMAÇÃO

Em muitas situações o valor do integral pode já ser previamente conhecido, dispensando o uso de integração numérica. Apesar da informação transmitida neste capítulo relativamente ao uso da quadratura de Gauss-Legendre como procedimento de integração numérica não parecer relevante nestas situações, isto não é necessariamente assim. Após a resolução do sistema de equações, no âmbito da resolução de um problema pelo Método dos Elementos Finitos, é necessário obter um conjunto de resultados ao nível dos elementos finitos. Para obtenção desses resultados é necessário avaliar a matriz $\boldsymbol{B}$ do elemento finito.

O uso de polinómios interpoladores evidencia que os resultados obtidos vão oscilar face à solução teórica exata consoante o ponto que se escolha para avaliar a matriz de deformação. Este facto leva à adoção de um critério de seleção dos pontos para avaliar a matriz $\boldsymbol{B}$ do elemento finito. O uso da quadratura de Gauss-Legendre revela-se novamente útil, pois, existe uma boa relação entre os pontos adequados para avaliar a matriz $\boldsymbol{B}$ e os pontos de integração da quadratura de Gauss-Legendre. Assim, no âmbito do Método dos Elementos Finitos, as coordenadas dos pontos de integração da quadratura de Gauss-Legendre usados para avaliar a função integranda podem, também, ser usados para avaliar cada uma das matrizes elementares $\boldsymbol{B}$.

Em suma, neste capítulo foi apresentada a metodologia de integração numérica mais utilizada no âmbito do Método dos Elementos Finitos. Assim, justifica-se a importância da quadratura de Gauss-Legendre no âmbito do Método dos Elementos Finitos.

6

GERAÇÃO DE MALHAS DE ELEMENTOS FINITOS

6.1. MALHAS DE ELEMENTOS FINITOS

A aplicação do Método dos Elementos Finitos requer a geração de uma malha que modele o domínio do problema em estudo. O modo como esta malha de elementos finitos é gerada vai influenciar o ritmo com que a solução numérica converge para os valores teóricos expectáveis. Assim, surge a necessidade de desenvolver estratégias de refinamento de malhas de elementos finitos que assegurem a convergência de solução.

As técnicas de geração de malhas de elementos finitos apresentadas neste capítulo são relativas a domínios bidimensionais. Relativamente a domínios unidimensionais, dada a sua simplicidade, não há propriamente muitos conselhos a fornecer, além de se procurar modelar adequadamente a geometria do problema em estudo. Quanto aos domínios tridimensionais, algumas das técnicas referidas para os domínios bidimensionais podem ser estendidas aos domínios tridimensionais. Normalmente, a geração de malhas em domínios bidimensionais consiste em discretizar o domínio em estudo em triângulos e/ou em quadriláteros. Para domínios tridimensionais a discretização é feita com recurso a elementos finitos tetraédricos, pentaédricos e/ou hexaédricos.

Os métodos de geração de malhas em domínios bidimensionais podem ser classificados em diretos ou algébricos e indiretos ou de equações diferenciais. Os métodos diretos são assim designados porque geram uma malha de elementos finitos sobre o domínio, baseado num dado algoritmo concreto. Os métodos indiretos podem ser divididos em geração de malhas em domínios elementares, geração de malhas por transformação de coordenadas, mapeamentos conformes, mapeamentos isoparamétricos, mapeamentos transfinitos e decomposição do domínio [10].

Com a apresentação das seguintes técnicas de geração de malhas de elementos finitos pretende-se transmitir a informação necessária para a escolha da metodologia de geração de malhas e, também, para o desenvolvimento de novas técnicas.

6.2. TÉCNICAS DE GERAÇÃO DE MALHAS

6.2.1. GERAÇÃO POR SOBREPOSIÇÃO DE GRELHA

Os elementos finitos da grelha podem ser triangulares ou retangulares. A grelha é sobreposta sobre o objeto a modelar e os elementos da grelha que estão fora do objeto são removidos. Os elementos da grelha que cruzam a fronteira do objeto são ajustados ou aparados para que se encaixem no objeto. A principal dificuldade na aplicação desta técnica consiste em ajustar os elementos da grelha à fronteira

do objeto. A vantagem da sua aplicação é que se obtém uma malha de elementos finitos bem discretizada no interior do objeto. A figura 6.1 mostra a aplicação da técnica de sobreposição de uma grelha.

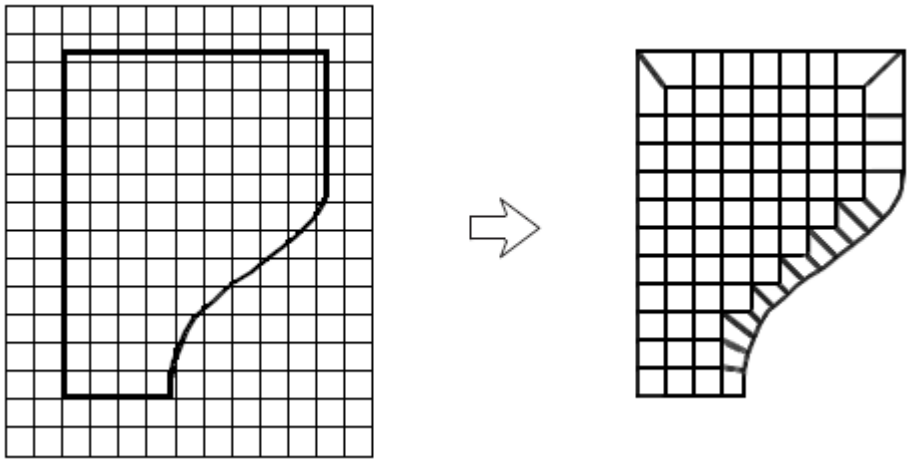


Fig.6.1 – Geração de malhas por sobreposição de grelha.

6.2.2. Geração por Transformação de Coordenadas

A geração de malhas por transformação de coordenadas é o tipo mais simples de mapeamento e consiste em usar uma transformação de coordenadas que transforma o domínio elementar no domínio real. A desvantagem deste método reside no facto de ser necessário determinar a transformação de coordenadas que faça a passagem do domínio elementar para o domínio real [10]. A figura 6.2 mostra a aplicação da técnica de transformação de coordenadas.

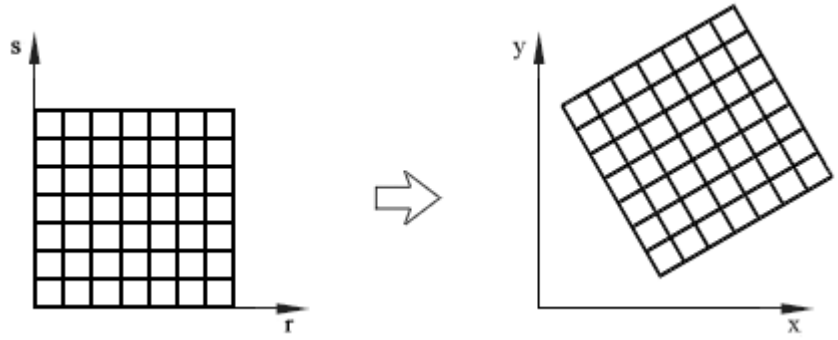


Fig.6.2 – Geração de malhas por transformação de coordenadas.

6.2.3. Geração por Mapeamento Conforme

O mapeamento conforme é toda a projeção cuja escala, em cada ponto, é independente da direção considerada. Em consequência, os ângulos em torno desse ponto são conservados, bem como a forma dos pequenos objetos. Na prática, o mapeamento conforme consiste em associar os pontos dos domínios elementar e real a números complexos. Entre as desvantagens deste método está o facto de que as características da malha são definidas automaticamente dificultando a produção de uma malha adequada ao domínio [10]. Este método também exige que se determine a transformação de coordenadas que faça o mapeamento do domínio elementar no domínio real.

6.2.4. GERAÇÃO POR MAPEAMENTO ISOPARAMÉTRICO

Esta técnica consiste em obter os valores das coordenadas dos pontos do domínio a partir de valores especificados no contorno através do uso de funções de interpolação. Nesse processo o contorno do domínio não é especificado, ele é aproximado por funções de interpolação que passam por pontos especificados. Neste tipo de mapeamento é ainda possível controlar a densidade da malha de elementos finitos. A figura 6.3 mostra a geração de uma malha de elementos finitos através da aplicação desta técnica.

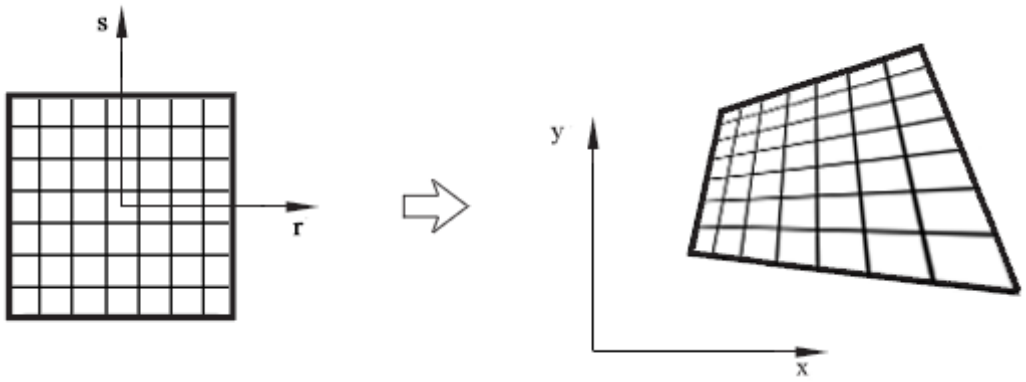


Fig.6.3 – Geração de malhas por mapeamento isoparamétrico.

6.2.5. GERAÇÃO POR MAPEAMENTO TRANSFINITO

O mapeamento transfinito estabelece um sistema de coordenadas curvilíneas definidas pelo contorno de domínios arbitrários [10]. Este método descreve uma superfície aproximada que coincide com a superfície real ou idealizada em um número não enumerável de pontos, propriedade que deu origem ao nome do método. A geometria do modelo é representada através de sub-regiões definidas por curvas do contorno que, por sua vez, são constituídas por um conjunto de primitivas (pontos, segmentos de reta, curvas quadráticas e cúbicas) [10].

6.2.6. GERAÇÃO POR TRIANGULAÇÃO DE DELAUNAY

Uma triangulação consiste em encontrar segmentos de reta que conectam um conjunto de pontos pertencentes ao domínio em estudo. Nenhum desses segmentos se pode cruzar com outro e cada ponto é vértice de pelo menos um triângulo. A Triangulação de Delaunay é um método de decomposição do domínio e assenta na determinação de um círculo cujo interior não contém nenhum outro ponto do conjunto de pontos que formam o triângulo. A Triangulação de Delaunay maximiza o menor ângulo de todos os triângulos na triangulação pelo que tende a evitar triângulos com ângulos internos muito pequenos.

6.2.7. GERAÇÃO POR QUADTREES

A *quatree* é uma técnica de decomposição do domínio que pode ser usada para implementar um algoritmo de decomposição recursiva do espaço. O processo consiste em dividir um domínio em domínios de forma similar sendo repetido o número de vezes necessário. Os algoritmos de decomposição recursiva do espaço trabalham com um número pequeno de formas. Normalmente, as

formas disponíveis em domínios bidimensionais são triângulos ou quadriláteros. A figura 6.4 mostra a aplicação desta técnica onde cada domínio elementar é dividido em quatro novas formas similares.

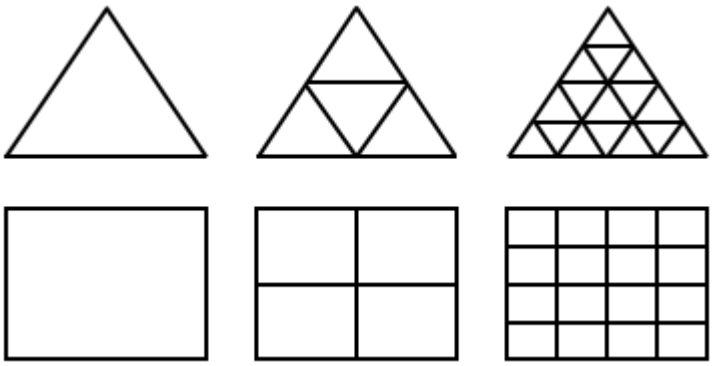


Fig.6.4 – Geração de malhas pela técnica de *quadtree*.

6.3. TIPOS DE REFINAMENTO

Existem vários procedimentos para melhorar os resultados fornecidos pelos elementos finitos. Em geral, estes procedimentos podem ser classificados da seguinte forma:

- Refinamento *h*;
- Refinamento *p*.

O refinamento *h* consiste em aumentar o nível de discretização da malha de elementos finitos, ou seja, em aumentar o número de elementos finitos presentes numa dada área onde se pretende melhorar a qualidade dos resultados obtidos. O refinamento *p* consiste em aumentar a ordem do elemento finito através do aumento do grau do polinómio interpolador. Um maior grau do polinómio interpolador pode ser conseguido com o aumento do número de nós no elemento finito.

Tipicamente, o refinamento *h* pode ser dividido em três subtipos. Uma das primeiras metodologias consiste em dividir os elementos finitos obtendo uma malha mais apertada. A segunda metodologia consiste em reformular a malha de elementos finitos. Por último, a terceira metodologia consiste em mover a posição dos nós dos elementos finitos alterando a configuração da malha.

O refinamento *p* também pode ser dividido em subtipos, nomeadamente dois. A primeira metodologia consiste em aumentar o grau do polinómio das funções de forma do elemento finito. A segunda em aumentar o número de nós do elemento finito. A forma de realizar o refinamento *p* está evidenciada no Capítulo 3 nos pontos relativos à obtenção das funções de forma dos elementos finitos. A junção dos dois tipos de refinamento é designada por refinamento *hp*.

6.4. REFINAMENTO DA MALHA DE ELEMENTOS FINITOS

As técnicas de geração de malhas, anteriormente apresentadas, só conduzem a malhas adequadas se o domínio a discretizar estiver corretamente definido.

Normalmente, antes de usar um método automático de geração de malhas, é adequado recorrer a uma prévia decomposição do domínio de forma a garantir que a malha gerada não apresente muitas irregularidades. A realização de um refinamento local exige que seja feita uma transição entre a zona refinada e a zona corrente. Esta transição é feita através de uma variação gradual do refinamento da

malha. A figura 6.5 mostra uma proposta de transição entre zonas com elementos finitos de dimensões diferentes.

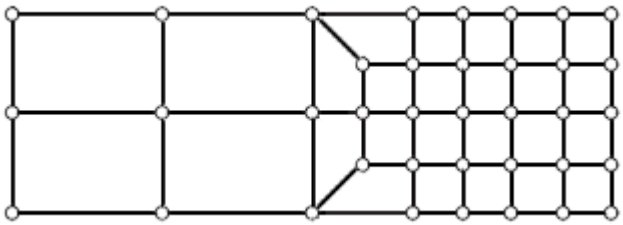


Fig.6.5 – Transição entre elementos finitos de dimensões diferentes.

Regiões de introdução de cargas concentradas ou zonas de apoios, regiões com descontinuidades geométricas ou ligações com diferentes elementos estruturais são exemplo de zonas que devem ser alvo de um refinamento individualizado, isto é, devem possuir uma malha mais apertada.

Uma das dificuldades na modelação com elementos finitos reside na modelação de meios muito irregulares ou com geometrias curvas, pois, os programas de cálculo automático são limitados quanto à disponibilidade de formas de elementos finitos. A utilização de elementos finitos triangulares em meios bidimensionais e os seus equivalentes tetraédricos para meios tridimensionais são a solução, muitas vezes adotada, para discretizar meios complexos ou realizar refinamentos locais em zonas com elevados gradientes de tensão e/ou deformação.

A grande flexibilidade dos elementos finitos triangulares e tetraédricos torna-os indispensáveis em qualquer programa de geração de malhas ou de cálculo automático pelo Método dos Elementos Finitos. No entanto, a maior desvantagem da discretização com recurso a elementos finitos triangulares de três nós e tetraédricos de quatro nós, reside na baixa acuidade dos resultados obtidos quando comparados, respetivamente, com os quadriláteros de quatro nós e os hexaedros de oito nós.

Sendo o Método dos Elementos Finitos um método aproximado, a qualidade dos resultados está sempre dependente do nível de refinamento da malha de elementos finitos. Assim, deve-se procurar usar malhas com um nível de refinamento ajustado à precisão dos resultados pretendidos.

6.5. QUALIDADE DAS MALHAS DE ELEMENTOS FINITOS

Normalmente, a primeira incógnita a determinar numa análise pelo Método dos Elementos Finitos é o campo de deslocamentos de um número finito de nós. Estes nós da malha encontram-se distribuídos ao longo da fronteira e no interior de cada elemento finito.

Os nós pertencentes à fronteira de elementos finitos adjacentes devem de ser comuns a todos os elementos que aí se encontram, pois, os elementos finitos devem estar ligados através dos seus nós. A figura 6.6 mostra alguns exemplos de ligações não conformes entre elementos finitos. É também recomendável que os elementos finitos vizinhos do mesmo tipo apresentem uma geometria semelhante e as transições entre elementos finitos diferentes sejam graduais.

Existem algumas orientações a seguir no processo de geração das malhas de elementos finitos [6], nomeadamente:

- Modelar adequadamente os limites curvos do domínio;
- Evitar o uso de elementos finitos desproporcionados ou distorcidos;
- Ligar os elementos finitos adequadamente através dos seus nós;

- Usar malhas regulares e sem variações bruscas de refinamento;
- Aumentar o refinamento nas zonas com grandes variações dos gradientes de tensões;
- Usar elementos que modelam o campo de deslocamentos real.

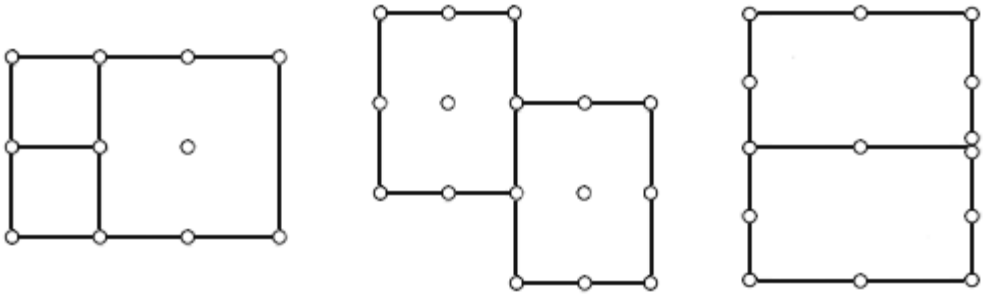


Fig.6.6 – Ligações não conformes entre elementos finitos.

A escolha dos elementos a usar tem influência na precisão dos resultados. Há elementos que não são capazes de modelar alguns modos de deformação ou distorções. Elementos diferentes têm diferentes sensibilidades para modelar distorções. Deve-se procurar que a seleção dos elementos finitos esteja em conformidade com o tipo de deformação esperada. A figura 6.7 mostra exemplos de elementos finitos cuja geometria é desproporcionada ou distorcida.

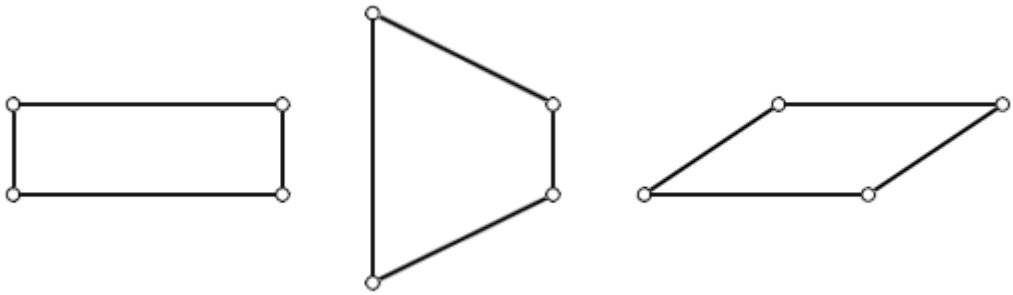


Fig.6.7 – Elementos finitos de geometria desproporcionada ou distorcida.

As condições de apoio do modelo são tão relevantes como a própria malha de elementos finitos. Não adianta usar refinamentos elevados se os apoios usados não apresentem correspondência com as condições de apoio reais da estrutura. Normalmente, em teoria, os apoios estruturais são idealizados como completamente rígidos ou como articulados. Os apoios reais estão normalmente entre um rígido e um apoio articulado. Esta diferença pode levar a que exista uma alteração significativa da distribuição de esforços na estrutura.

No que se refere à avaliação da qualidade de uma malha de elementos finitos, esta pode ser determinada com base nos seguintes fatores:

- Taxa de convergência;
- Precisão da solução;
- Tempo de cálculo.

Quanto maior for a taxa de convergência, melhor será a qualidade da malha. Isto significa que a solução teórica expectável para o problema é alcançada mais rapidamente. A melhor qualidade de uma malha garante uma solução mais precisa. Para melhorar a qualidade da malha é necessário refinar as zonas onde sejam expectáveis grandes gradientes de tensões. Relativamente ao tempo de cálculo, as malhas que necessitam de mais tempo para serem calculadas, em geral, fornecem melhores resultados que as

malhas que são rapidamente calculadas. Para uma malha altamente refinada, em que o número de elementos finitos por unidade de área é máximo, o tempo de cálculo é relativamente grande.

A ponderação e o conhecimento real sobre o efetivo comportamento das peças e das estruturas constitui um meio útil para a modelação com elementos finitos. Por melhor que sejam as ferramentas de cálculo, a sensibilidade do utilizador é fundamental para o processo de modelação com elementos finitos.

7

NOÇÕES SOBRE DESENVOLVIMENTO DE SOFTWARE

7.1. PROJETO DE SOFTWARE

Os comandos que os processadores dos computadores oferecem são extremamente básicos, normalmente orientados para realizar operações aritméticas como soma, subtração, divisão e multiplicação. Qualquer tarefa mais complexa deve ser resolvida através de uma sequência destes comandos básicos. Uma sequência de comandos básicos de processador que resolve uma determinada tarefa recebe o nome de programa. Os programas são armazenados em ficheiros comumente chamados de executáveis.

A construção de um programa requer uma definição prévia de objetivos e meios. Não há bons projetos sem uma correta definição dos objetivos e meios necessários para os concretizar.

A escolha da linguagem de programação é uma das primeiras tarefas do projeto. As linguagens de programação são normalmente orientadas para um contexto específico. Existem linguagens independentes de plataformas e outras associadas a uma dada plataforma ou tecnologia. Se a escolha da linguagem de programação não for a adequada, compromete-se a distribuição do programa. O mesmo se aplica a programas desenvolvidos para fins pessoais, pois, se a plataforma para o qual é desenvolvido o programa se tornar obsoleta a curto prazo, compromete-se todo o trabalho desenvolvido. Deve-se procurar obter a portabilidade da ferramenta informática desenvolvida. Ao se desenvolver programas que permitam a sua utilização em diferentes plataformas está-se a rentabilizar o seu desenvolvimento conseguindo, desta forma, tirar o máximo partido deles.

O desenvolvimento de *software* com qualidade é um objetivo importante. A qualidade pode ser definida como o grau de satisfação de requisitos dados por um conjunto de características intrínsecas. No contexto do desenvolvimento de *software*, a qualidade pode ser definida como um processo eficaz aplicado de uma maneira que cria um produto útil e que fornece valor mensurável para quem o produz e para quem o utiliza.

A programação é um processo de escolhas e todas elas têm implicações no desenvolvimento da solução informática. A previsão e a antecipação dos processos de decisão futuros é fundamental para que as escolhas do presente não comprometam o desenvolvimento e a manutenção futura do programa. Apesar do futuro ser uma incerteza, o que se pretende salientar é que antes do início da programação, deve-se começar pela preparação do processo de desenvolvimento, nomeadamente com a divisão das tarefas necessárias ao desenvolvimento da solução global. Este trabalho é importantíssimo no processo de

desenvolvimento na medida que previne o aparecimento de eventuais conflitos entre o trabalho já desenvolvido e o trabalho a desenvolver.

Em suma, salienta-se que no processo de desenvolvimento, abordagens diferentes conduzem a programas diferentes.

7.2. LINGUAGENS DE PROGRAMAÇÃO

7.2.1. CRITÉRIOS DE AVALIAÇÃO DA LINGUAGEM

A apresentação dos critérios de avaliação de uma linguagem de programação prende-se com o objetivo de fornecer os meios necessários à escolha da linguagem de programação mais adequada ao desenvolvimento e posterior manutenção de um determinado programa informático. Os critérios frequentemente usados para avaliar uma linguagem são [19]:

- Legibilidade;
- Capacidade de escrita;
- Confiabilidade;
- Custo.

A legibilidade é um dos critérios mais importantes para avaliar uma linguagem de programação, nomeadamente, porque tem implicações na facilidade com que os programas podem ser lidos e entendidos. Uma vez que a facilidade de manutenção é condicionada pela legibilidade dos programas, esta tornou-se uma forma de avaliar a qualidade dos programas e das linguagens. Uma linguagem com um grande número de componentes é difícil de aprender e de ler em comparação com uma linguagem mais pequena.

A capacidade de escrita é uma medida de quão facilmente uma linguagem pode ser utilizada para criar programas para um domínio de problema escolhido. A maioria das características da linguagem que afeta a legibilidade também afeta a capacidade de escrita. Se uma linguagem tiver um grande número de construções alguns programadores podem não estar familiarizados com todas elas.

A confiabilidade de uma linguagem significa que os programas desenvolvidos se comportam conforme especificado. A verificação de tipos e a manipulação de exceções são exemplos de mecanismos que reforçam a confiabilidade de uma linguagem de programação.

O custo final de uma linguagem de programação é função de muitas das suas características. Os custos associados a uma linguagem de programação [19] são, genericamente, os seguintes:

- Custos de formação dos programadores;
- Custo de escrita dos programas na linguagem;
- Custo de compilação dos programas;
- Custo de execução dos programas;
- Custo do sistema de implementação da linguagem;
- Custo da má confiabilidade;
- Custo de manutenção dos programas.

De todos os fatores que contribuem para os custos de uma linguagem, destacam-se o tempo de escrita do programa, a manutenção e a confiabilidade como sendo os mais importantes.

Evidentemente, existem inúmeros critérios para avaliar as linguagens de programação. Critérios como a portabilidade ou a facilidade com que os programas podem ser mudados de uma implementação para

outra são atualmente critérios importantíssimos, face ao número de tipos de dispositivos existentes. A portabilidade de uma linguagem está associada ao grau de padronização desta.

7.2.2. CATEGORIAS DE LINGUAGENS DE PROGRAMAÇÃO

A linguagem máquina é a única linguagem que os computadores maioritariamente entendem. As linguagens de programação podem ser classificadas como sendo de baixo nível ou de alto nível, consoante estão mais próximas ou afastadas da linguagem máquina. As linguagens de baixo nível são interpretadas diretamente pelo computador, contudo, são bastante incómodas para se programar com elas. De modo a agilizar o processo de programação foram criadas as linguagens de programação de alto nível.

As linguagens de programação, habitualmente, são categorizadas em quatro tipos [19]:

- Linguagens imperativas;
- Linguagens funcionais;
- Linguagens lógicas;
- Linguagens orientadas a objetos.

As linguagens imperativas são orientadas a ações, onde a computação é vista como uma sequência de instruções que manipulam valores de variáveis. O fundamento da programação imperativa é o conceito de Máquina de Turing, que nada mais é que uma abstração matemática que corresponde ao conjunto de funções computáveis. Os operandos das expressões são passados da memória para o processador e o resultado da expressão é passado de volta para a memória.

As linguagens funcionais fundamentam-se no paradigma de programação funcional que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste com a programação imperativa, que enfatiza mudanças no estado do programa. Uma função, neste sentido, pode ou não receber parâmetros e tem um simples valor de retorno.

Uma linguagem de programação lógica é um exemplo de linguagem baseada em regras. A programação lógica é um paradigma de programação que faz uso da lógica matemática. O sentido da programação lógica é trazer o estilo da lógica matemática à programação de computadores. A abordagem ao desenvolvimento é radicalmente diferente do usado com os outros três tipos de linguagens.

As linguagens orientadas a objetos fundamentam-se no paradigma da orientação a objetos em que cada entidade do sistema corresponde, durante a execução do programa, a um objeto, com atributos e ações próprias que definem o modo de funcionamento desse programa. A maioria das linguagens de programação orientadas a objetos usa o conceito de classe, para descrição de grupos de objetos semelhantes. A programação orientada a objetos foi criada com o objetivo de aproximar o mundo real ao mundo virtual.

7.2.3. MÉTODOS DE IMPLEMENTAÇÃO

7.2.3.1. Compilação

A compilação corresponde à tradução do código fonte do programa para linguagem máquina. Uma vez traduzido, o programa pode ser diretamente executado no computador. Este método tem a vantagem de uma execução de programa muito rápida, assim que o processo de tradução for concluído. Alguns compiladores permitem a otimização dos programas, tornando-os menores e mais rápidos. O ficheiro

gerado pelo compilador é designado de executável e é específico para um determinado sistema operativo e uma determinada arquitetura de processador.

7.2.3.2. Interpretação

Os programas podem ser diretamente interpretados por um outro programa sem recurso a nenhuma tradução. O programa interpretador comporta-se como um simulador de uma máquina cujo ciclo de funcionamento lida com as instruções do programa numa linguagem de alto nível em vez de instruções de máquina. Esta simulação é conseguida com recurso a uma máquina virtual para a linguagem. A interpretação tem a vantagem de permitir uma fácil implementação de muitas operações de depuração do código fonte, pois, todas as mensagens de erro em tempo de execução podem ser referidas a unidades do código. A interpretação apresenta algumas desvantagens relativamente à compilação, nomeadamente, a execução mais lenta e o maior consumo de memória.

7.2.3.3. Sistemas de Implementação Híbridos

Alguns sistemas de implementação de linguagem são um meio-termo entre os compiladores e os interpretadores. Os sistemas de implementação híbridos traduzem os programas em linguagem de alto nível para uma linguagem intermediária que foi projetada para permitir uma fácil interpretação. Este método é mais rápido do que a interpretação porque as instruções da linguagem fonte são decodificadas somente uma vez. Na prática, em vez de traduzir o código da linguagem intermediária para código de máquina, o código intermediário é diretamente interpretado.

7.3. PRINCÍPIOS DE DESENVOLVIMENTO

Como a componente prática desta dissertação consistiu na construção de um programa, é útil fornecer um conjunto de dicas que auxiliem o desenvolvimento de *software*. Os princípios propostos por David Hooker [19] são uma sugestão de como se deve desencadear o desenvolvimento de *software*.

Os princípios de desenvolvimento propostos são os seguintes:

1. A razão pela qual tudo existe;
2. Mantenha as coisas simples;
3. Mantenha a visão;
4. O que produz, os outros vão consumir;
5. Esteja aberto para o futuro;
6. Planeie com antecedência para a reutilização;
7. Pense.

Um programa existe para fornecer valor aos seus utilizadores. Todas as tomadas de decisão devem ser feitas com este princípio em mente. Assim, antes de fazer o que quer que seja deve-se avaliar se vai ser acrescentado valor. Caso contrário não deverá ser feito.

O desenvolvimento de programas informáticos deve orientar-se pela simplificação. A facilidade de um sistema facilita a sua compreensão e manutenção. A simplificação é um processo árduo, que é o oposto de rápido e desorganizado. Um programa melhor organizado é menos propício a erros.

Manter uma visão clara do que se pretende é fundamental para manter o rumo de qualquer projeto. Sem a definição clara do seu conceito, um sistema informático pode vir a tornar-se um conjunto de projetos

incompatíveis. Ter alguém que garanta a visão do desenvolvimento é fundamental para se alcançar os objetivos de forma bem-sucedida.

Relativamente ao quarto princípio, raramente um programa fica apenas limitado ao domínio privado. Atualmente é muito fácil que qualquer aplicação tenha algum tipo de distribuição. Sempre que se projeta um programa, deve-se ter em mente que um programa é para ser utilizado por pessoas. Assim, um programa deve ser desenvolvido de forma que permita a fácil manutenção, expansão e utilização. Quanto mais focado nas pessoas maior é a possibilidade de se obter o sucesso.

Um programa que tenha um grande período de vida tem mais valor. Atualmente, as plataformas informáticas ficam rapidamente obsoletas pelo que é frequente medir o tempo de vida dos programas em meses e não em anos. Para se garantir um tempo de vida elevado deve-se fornecer flexibilidade para que estes programas se possam adaptar às mudanças. Os programas que não possuem flexibilidade suficiente rapidamente têm de ser substituídos por outros.

O sexto princípio salienta que o planeamento está na base de qualquer projeto. Um bom planeamento permite a reutilização do trabalho já desenvolvido, economizando tempo e esforço. Permitir a reutilização de todo o conteúdo já desenvolvido é sem dúvida difícil de alcançar. Planear com antecedência para a reutilização reduz o custo e aumenta o valor de ambos os componentes reutilizáveis e dos programas em que são incorporados.

Colocar o pensamento antes da ação quase sempre permite obter melhores resultados. Ao se pensar na resolução de um problema está-se a aumentar as possibilidades de o resolver direito. Ao se pensar também se está a reconhecer que não se sabe tudo. Deste modo, a pesquisa por novas soluções permite acrescentar valor ao programa.

A aplicação de todos estes princípios requer um pensamento constante. Não se pode dar mais valor aos princípios relativamente àquilo que eles valem. O desenvolvimento requer uma ponderação sobre a importância de cada um destes princípios no projeto em causa.

7.4. DESENVOLVIMENTO DE SOFTWARE

7.4.1. PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Um processo é um conjunto de atividades, ações e tarefas que são executadas para criar algum produto. No contexto do desenvolvimento de *software*, um processo não é uma prescrição rígida de como construir programas de computador. Pelo contrário, é uma abordagem flexível que permite que as pessoas que fazem o trabalho possam escolher o conjunto de ações adequadas ao trabalho e às tarefas [15]. A intenção é sempre de entregar o programa em tempo útil e com qualidade suficiente para satisfazer aqueles que desencadearam a sua criação e aqueles que o vão utilizar.

A estrutura do processo genérico do desenvolvimento de *software* abrange as seguintes atividades [15]:

- Comunicação;
- Planeamento;
- Modelação;
- Construção;
- Implantação.

A comunicação em qualquer trabalho técnico é extremamente importante. A intenção é compreender os objetivos das partes interessadas para o projeto e para reunir os requisitos que ajudam a definir os recursos do programa e as suas funções. Na fase de planeamento são descritas as tarefas técnicas a

realizar, os riscos que podem ocorrer, os recursos que são necessários, os produtos a produzir e o período de trabalho. Na fase de modelação, o programador faz uma criação de modelos para melhor entender os requisitos de programa e do *design* pretendido. A atividade de construção combina a escrita de código e os testes que são necessários para descobrir os erros existentes. Por fim, a implantação corresponde à entrega ao cliente do produto desenvolvido.

Estas cinco atividades genéricas podem ser usadas durante o desenvolvimento de um qualquer programa, independentemente da sua dimensão, complexidade ou especificidade. Na base de um qualquer projeto está a gestão de projetos. Estas atividades constituem, de um modo genérico, as atividades a pormenorizar no âmbito do projeto de *software*.

7.4.2. Princípios Fundamentais

O desenvolvimento de *software* é orientado por um conjunto de princípios básicos. Em [15] são apresentados vários princípios para o processo e para a prática. Estes princípios genéricos pretendem ser orientadores e abranger a generalidade das situações do desenvolvimento de *software*.

Ao nível do processo estabelecem-se os seguintes princípios básicos numa base filosófica que oriente a equipa de desenvolvimento:

- Seja flexível;
- Foque o trabalho na qualidade;
- Esteja pronto para se adaptar;
- Construa uma equipa eficaz;
- Estabeleça mecanismos de comunicação e coordenação;
- Administre a mudança;
- Avalie o risco;
- Crie produtos que agreguem valor para outros.

Ao nível da prática estabelecem-se um conjunto de valores e regras que servem como um guia para análise de um problema, formulação de uma solução, implementação, teste da solução e, finalmente, implantação do programa. Assim, os princípios básicos que guiam a prática são:

- Divida os problemas nas suas componentes;
- Compreenda o uso da abstração;
- Mantenha a consistência;
- Foque-se na transferência de informações;
- Construa programas que exibam modularidade;
- Procure padrões;
- Procure representar o problema e a solução em diferentes perspetivas;
- Lembre-se que alguém vai manter o programa.

7.4.3. Conceitos de Projeto

7.4.3.1. Abstração

Abstrair significa decompor um sistema complicado nas suas partes fundamentais e descrevê-las numa linguagem de programação de forma simples e precisa. A descrição das partes de um sistema implica atribuir-lhes um nome e descrever as suas funcionalidades.

O uso da abstração ao longo do desenvolvimento é talvez dos conceitos mais importantes. O programador ao se abstrair consegue encontrar as componentes elementares que constituem o problema a resolver. Ao se dividir um problema nas suas partes mais simples, está-se a diminuir a complexidade de resolução do problema. Deste modo, ao se programar as partes elementares é suficiente, posteriormente, chamar cada uma das partes elementares pela ordem com que são necessárias para resolver o problema.

Este modo de programação com conceitos gerais e elementares propicia a reutilização do código desenvolvido, pois, problemas diferentes têm muitas vezes aspetos comuns de resolução. Deste modo evita-se a duplicação de blocos ao longo do código fonte do programa. Outra vantagem da abstração é de permitir a implementação em qualquer projeto. Quanto mais abstratos forem os códigos mais facilmente podem ser inseridos em novos contextos.

7.4.3.2. Arquitetura

A arquitetura de *software* faz referência à estrutura geral do programa e às formas como essa estrutura fornece integridade ao programa. Na sua forma mais simples, a arquitetura é a estrutura ou organização dos componentes do programa. Num sentido mais amplo, os componentes podem ser generalizados para representar os principais elementos do sistema e as suas interações. Dada a especificação destas propriedades, a arquitetura pode ser representada usando um ou mais de um número de modelos diferentes. Os modelos estruturais representam a arquitetura como uma coleção organizada de componentes do programa.

7.4.3.3. Padrões

Um padrão de projeto descreve uma solução geral reutilizável para um problema recorrente no desenvolvimento de *software*. Não é um código final mas uma descrição ou modelo de como resolver o problema de modo a ser implementado em diferentes situações. Os padrões são usados para representar, registar e reutilizar elementos do projeto, bem como a experiência acumulada ao longo do desenvolvimento.

Os padrões, normalmente, definem as relações e interações entre as classes ou objetos, sem especificar os detalhes das classes ou objetos envolvidos, ou seja, recorrem a um nível de abstração elevado. Os padrões complementam as abordagens correntes e constituem uma base de experiência reutilizável para a fase de desenvolvimento. Os padrões fornecem um conjunto de vocabulário comum para os programadores comunicarem, permitindo um desenvolvimento mais harmonioso. A sua implementação permite a reutilização de blocos de código ou a distribuição de bibliotecas sem a necessidade de conhecer o seu funcionamento aprofundadamente. São a forma de reduzir a complexidade de um sistema.

Contrariamente aos padrões, os anti-padrões indicam precisamente o contrário. O foco dos anti-padrões são os erros comuns dos programadores e as falhas que levam ao fracasso do desenvolvimento. O seu conhecimento evita a tomada de decisões que futuramente comprometam a evolução e a manutenção dos programas.

7.4.3.4. Modularidade

Os programas são compostos por diferentes componentes que devem interagir corretamente, fazendo com que o sistema como um todo funcione de forma adequada. Para se manter estas interações é

necessário que os diversos componentes estejam bem organizados. A modularidade refere-se a uma estrutura de organização na qual os diferentes componentes de um de programa são divididos em unidades funcionais separadas. A estrutura imposta pela modularidade fomenta a reutilização, isto é, se os módulos forem escritos de uma forma abstrata para resolver problemas genéricos, então estes módulos podem ser reutilizados quando o mesmo problema surgir noutro contexto.

7.4.3.5. Encapsulamento

O conceito de encapsulamento estabelece que os diferentes componentes de um programa não devem revelar detalhes internos das suas respetivas implementações. Por ouras palavras, os módulos devem ser especificados e concebidos de modo a que informação contida dentro de cada um seja inacessível para outros módulos que não têm necessidade de tais informações.

O encapsulamento define e impõe restrições de acesso dentro de um módulo e/ou qualquer estrutura de dados local usada pelo módulo. O uso de encapsulamento como um critério de projeto para sistemas modulares oferece grandes benefícios quando são necessárias alterações durante o desenvolvimento e manutenção do programa. Esta vantagem resulta do facto dos dados e detalhes processuais estarem escondidos das outras partes do programa. Assim, erros involuntários introduzidos durante a modificação são menos propensos a se propagarem para outros locais dentro do programa. O encapsulamento permite que não seja necessário conhecer o funcionamento interno de um módulo para o poder implementar. Isto permite libertar o programador da necessidade de compreender integralmente tudo o que implementa ao longo das diversas fases do desenvolvimento.

7.4.3.6. Independência Funcional

Deve-se projetar de modo que cada módulo aborde um subconjunto específico de requisitos e tenha uma interface simples quando visto a partir de outras partes da estrutura do programa. *Software* com modularidade eficaz, ou seja, módulos independentes, é mais fácil de desenvolver. Módulos independentes são também mais fáceis de manter porque os efeitos secundários causados pela conceção ou modificação do código estão limitados, logo, a propagação de erros é reduzida e os módulos são mais facilmente reutilizáveis.

7.4.3.7. Refinamento

O refinamento passo a passo é uma estratégia de projeto *top-down*. Na prática, começa-se com uma declaração de uma função que é definida num alto nível de abstração, ou seja, a função é descrita conceptualmente mas sem nenhuma informação sobre o seu funcionamento interno. Posteriormente, a declaração original é reescrita proporcionando mais e mais detalhes. A abstração e o refinamento são conceitos complementares. A abstração permite a especificação de procedimentos e de dados internamente suprimindo a necessidade de ter conhecimento dos detalhes de baixo nível. O refinamento ajuda a revelar detalhes de baixo nível com a evolução do projeto.

7.4.3.8. Refatoração

A refatoração é uma técnica de reorganização que simplifica o projeto de um componente sem alterar a sua função ou comportamento. Quando o *software* é reformulado, o projeto existente é examinado quanto à redundância, elementos não utilizados, algoritmos ineficientes ou desnecessários, mal

construídos ou estruturas de dados inadequadas, ou qualquer outra falha é corrigida para produzir um projeto melhor. O uso desta técnica aprimora a conceção de um *software* e evita a deterioração durante o ciclo de vida de um código. Esta deterioração é geralmente causada por mudanças com objetivos de curto prazo ou por alterações realizadas sem a clara compreensão da conceção do sistema.

7.4.3.9. Projeto Orientado a Objetos

No paradigma de programação orientada a objetos, a construção de um programa para implementação de um determinado sistema baseia-se numa correspondência natural e intuitiva entre esse sistema e a simulação do comportamento do mesmo. A programação orientada a objetos foi criada com o objetivo de aproximar o mundo real ao mundo virtual. A ideia fundamental é tentar simular o mundo real dentro do computador. Assim, a cada entidade do sistema corresponde, durante a execução do programa, um objeto, com atributos e ações próprias que definem o modo de funcionamento desse programa.

7.4.3.10. Projeto de Classes

A maioria das linguagens de programação orientadas a objetos usa o conceito de classe para descrição de grupos de objetos semelhantes. Uma classe é o projeto do objeto. A classe representa o tipo de objeto que é construído a partir dela. Estes objetos são compostos por atributos e ações. Uma classe é formada, essencialmente, por construtores de objetos dessa classe, variáveis e métodos. A criação de um objeto dessa classe consiste na criação de cada uma das variáveis do objeto, especificadas na classe. Os valores armazenados nessas variáveis determinam o estado do objeto, isto é, os seus atributos. Os objetos podem trocar mensagens entre si, sendo uma mensagem basicamente uma chamada a um método específico de um objeto, que realiza uma determinada operação. A execução de um método do objeto pode modificar o estado desse objeto e/ou retornar um resultado.

7.4.4. Ambientes de Desenvolvimento

Um ambiente de desenvolvimento ou programação é o conjunto de ferramentas usadas no desenvolvimento de programas. Esse conjunto pode consistir num sistema de ficheiros, editor de texto, ligador e um compilador. Pode, também, incluir uma coleção de ferramentas integradas, cada uma das quais acessível por meio de uma interface gráfica. A utilização de ambientes que disponibilizem uma interface gráfica de desenvolvimento com uma coleção de ferramentas são preferenciais na medida que melhoram significativamente o processo de desenvolvimento. Existem para a generalidade das linguagens de programação ambientes de desenvolvimento que facilitam ou geram o código para muitas das tarefas rotineiras associadas ao processo de desenvolvimento. Hoje em dia, uma das tarefas importantes no início da construção de um programa informático consiste na escolha do ambiente de programação. Uma escolha acertada vai impulsionar significativamente a sua codificação.

7.5. Interface Gráfica do Utilizador

Uma interface é a fronteira que define a forma de comunicação entre duas entidades. Pode ser entendida como uma abstração que estabelece a forma de interação entre uma máquina e o mundo exterior. Sendo o foco de um programa os utilizadores, deve-se procurar que os programas disponibilizem um meio que facilite a comunicação entre o utilizador e o computador. As interfaces gráficas são o meio disponível para a comunicação entre as duas partes.

No desenvolvimento de *software* deve-se, desde o início, definir o modo como se vai estabelecer a comunicação entre o utilizador e o computador. A interface gráfica não se pode comportar como um elemento neutro mas de um modo que os utilizadores possam interagir com ela. A interface gráfica deve ser desenvolvida a pensar no utilizador e implementar metodologias que facilitem a utilização do programa. Uma interface deve cumprir com um conjunto de requisitos próprios que se distinguem dos demais componentes constituintes de um programa. Estes requisitos centram-se na comunicabilidade, acessibilidade, usabilidade e aplicabilidade.

No desenvolvimento de uma interface gráfica deve-se procurar identificar padrões com os quais os futuros utilizadores estejam familiarizados. O uso de elementos ou representações que não sejam do conhecimento geral devem ser evitados. Interfaces pouco organizadas ou com excesso de informação tornam-se confusas e pouco práticas. As interfaces devem ser limpas e claras. Todos os elementos que não sejam essenciais devem estar ocultos e só devem ser mostrados nos momentos adequados. Como os programas são desenvolvidos com o intuito de realizaram uma dada operação ou apresentarem informações, estas áreas devem estar destacadas e sem elementos perturbadores.

O desenvolvimento de uma interface gráfica é uma tarefa criativa. Não é fácil definir critérios objetivos para avaliar uma interface gráfica. Contudo, as interfaces gráficas mal construídas fazem com que os utilizadores se deparem, por exemplo, com menus excessivamente complexos, terminologia incompreensível e caminhos de navegação caóticos. Uma forma de trazer qualidade e eficácia a um programa com aumento da utilização efetiva, menor curva de aprendizagem e menor resistência, consiste em melhorar os processos de construção de interfaces gráficas do utilizador.

8
IMPLEMENTAÇÃO COMPUTACIONAL

8.1. FOCO E FUNCIONALIDADES DO PROGRAMA

O desenvolvimento de uma ferramenta informática começa pela definição do seu alvo e funcionalidades a construir. Existem diversas ferramentas comerciais que possibilitam a análise estrutural com recurso a elementos finitos. Contudo, estas ferramentas estão focadas na prática profissional da engenharia, não disponibilizando ferramentas que permitam aos estudantes ter uma noção adequada sobre a modelação e análise estrutural com elementos finitos. De modo a fornecer essa experiência aos alunos de um curso sobre elementos finitos, era necessário o desenvolvimento de uma ferramenta informática com essas capacidades.

No âmbito desta dissertação foi desenvolvido um programa com a finalidade de colmatar a falta de ferramentas informáticas voltadas para o ensino do Método dos Elementos Finitos. Assim, o programa desenvolvido abrange a generalidade dos problemas estruturais encontrados na prática da engenharia e que são alvo de estudo em qualquer curso universitário de Engenharia Civil. Das diversas formulações e tipos de elementos finitos disponíveis foram focados os elementos finitos unidimensionais, bidimensionais, tridimensionais, de viga e de laje.

Além da possibilidade de testar diversas formulações, a ferramenta permite intervir ao nível das malhas e do número de nós dos elementos finitos. O programa disponibiliza flexibilidade suficiente aos seus utilizadores de modo a que estes possam realizar um conjunto de alterações sobre o modelo em estudo e visualizar os seus resultados da forma rápida e simples. Uma outra necessidade consistia em fornecer uma solução que pudesse ser utilizada em diferentes plataformas informáticas. Deste modo, ao se utilizar a linguagem de programação Java assegura-se a possibilidade de distribuição pelos diferentes dispositivos informáticos utilizados pelos estudantes. Por fim, o programa também oferece suporte para a persistência de dados, isto é, o programa permite que os utilizadores guardem e recuperem os projetos que criaram.

8.2. PROGRAMAÇÃO EM JAVA

O programa construído no âmbito desta dissertação foi escrito na linguagem de programação Java e com recurso ao ambiente de desenvolvimento integrado NetBeans IDE 8.0 [24].

A escolha da linguagem de programação recaiu sobre a linguagem de programação Java de modo a possibilitar que o programa funcionasse em diferentes plataformas informáticas. A linguagem de programação Java é uma linguagem de programação orientada a objetos cujo desenvolvimento começou

na década de 90 na empresa Sun Microsystems. A linguagem Java é compilada para um *bytecode* que é executado por uma máquina virtual denominada Java Virtual Machine.

Para um programa codificado em Java executar é necessário definir o primeiro método a ser chamado quando o programa for inicializado. Este método precisa de ser público, estático, sem retorno e receber um vetor de *strings* como argumento. Cada projeto tem somente um método main() cuja estrutura é apresentada no código abaixo.

```
/**
 * @param args são os argumentos da linha de comandos
 */
public static void main(String[] args) {}
```

Como o método main() é o primeiro a ser chamado, logo, é a partir dele que todo o encadeamento lógico se desencadeia. Assim, neste método devem estar definidas as instruções que iniciem o funcionamento do programa. Como os métodos são os mecanismos que desencadeiam ações dentro de uma classe é necessário definir o método main() dentro de uma classe. O código abaixo mostra a implementação do método main() dentro da classe FEMforStudents.

```
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

/**
 *
 * @author André de Sousa
 */
public class FEMforStudents {

    /**
     * @param args são os argumentos da linha de comandos
     */
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException | InstantiationException | IllegalAccessException |
UnsupportedLookAndFeelException ex) {
            Logger.getLogger(UserInterface.class.getName()).log(Level.SEVERE, null, ex);
        }

        new UserInterface().setVisible(true);
    }
}
```

O código inserido dentro do método main() executa duas tarefas distintas. A primeira consiste em importar o tema da interface gráfica de utilizador do sistema operativo para alguns dos componentes gráficos da interface gráfica do programa. A segunda tarefa consiste em criar um objeto da classe UserInterface e passar o parâmetro de visibilidade. A classe UserInterface é responsável, entre muitas tarefas, por criar a janela do programa.

O código fonte Java deve ser colocado em arquivos com a extensão *.java*. O próximo passo é compilar o código fonte para gerar um executável que possa ser processado pela máquina virtual do Java. O compilador padrão da plataforma Java pode ser utilizado para compilar os ficheiros do projeto do programa. O programa de desenvolvimento utilizado também permite realizar a compilação do projeto.

8.3. MÉTODO DOS ELEMENTOS FINITOS

8.3.1. CRIAÇÃO DOS ELEMENTOS FINITOS

A programação do Método dos Elementos Finitos consistiu na criação de um conjunto de classes que fornecessem todas as propriedades e elementos necessários a uma análise por elementos finitos. Deste modo, a implementação do método consistiu, em geral, pela criação do pacote finiteelement e da classe FiniteElement dentro do pacote calculations.

O pacote finiteelement é constituído por duas dezenas de classes das quais se destacam as classes MatrixB, MatrixD, MatrixJ, MatrixK e VectorF. A classe FiniteElement permite criar qualquer tipo de elemento finito necessitando somente de receber a informação que descreva o elemento finito a ser construído. Construído o elemento finito, a classe disponibiliza vários métodos que permitem obter, por exemplo, a matriz de rigidez do elemento finito e o vetor de forças nodais equivalentes. A importância da classe FiniteElement não se resume à criação dos elementos finitos. Toda a informação associada a cada elemento finito é disponibilizada por esta classe. A implementação desta classe no projeto consiste em criar uma lista de elementos finitos. Esta lista é definida através de um objeto da classe ArrayList que permite armazenar os elementos finitos criados.

As listas são um tipo de coleção que a linguagem de programação Java suporta e a criação do objeto para armazenar os elementos finitos é feita da seguinte forma.

```
ArrayList<FiniteElement> listOfFiniteElements = new ArrayList();
```

Este objeto que armazena os elementos finitos do modelo construído pelo utilizador é designado por listOfFiniteElements. A classe FiniteElement é composta por dezenas de variáveis e métodos na maioria de âmbito privado. Os métodos públicos são utilizados para definir atributos, desencadear ações internas ou obter os elementos necessários, por exemplo, à construção do sistema de equações globais.

Para se criar um objeto é necessário chamar o método construtor desse objeto que se encontra definido dentro da classe do objeto. O método construtor é sempre criado para uma classe, mesmo que não seja explicitamente definido. Neste caso, o método construtor é um método público com o nome da classe, sem retorno e que não recebe parâmetros.

```
/**
 *
 * @author André de Sousa
 */
public class FiniteElement {

    /**
     * Método construtor da classe FiniteElement
     */
    public FiniteElement(int ID, String shape, String type, String theory, int degreeOfFreedom,
            double[][] coordinates) {
        this.ID = ID;
        this.shape = shape;
        this.type = type;
        this.theory = theory;
        this.degreeOfFreedom = degreeOfFreedom;
        this.shapeCoordinates = coordinates;
    }
}
```

O método construtor da classe FiniteElement recebe alguns parâmetros que permitem que a partir do momento da criação do elemento finito, as variáveis que armazenam a informação relativa ao seu tipo,

geometria ou às suas coordenadas fiquem automaticamente definidas. Deste modo, todas as ações desencadeadas posteriormente já têm em conta o tipo de elemento finito criado o que assegura que aquando da chamada dos seus métodos as ações desencadeadas tenham em conta as propriedades desse elemento finito. De preferência, estes atributos não devem estar acessíveis publicamente de modo a impedir eventuais alterações realizadas externamente.

Os parâmetros recebidos pelo método construtor da classe FiniteElement não permitem definir por completo o elemento finito de modo a se calcular, por exemplo, a matriz de rigidez elementar. Assim, é necessário definir o número de nós do elemento finito, a secção e o material. Para o efeito, existem na classe os métodos apresentados abaixo.

```
/**
 * Método para atribuir o número de nós ao elemento finito
*/
public void setNumberOfNodes(int nodes) {
    this.nodes = nodes;
    this.nodesCoordinates = nodesCoordinates(shapeCoordinates, nodes);
}

/**
 * Método para atribuir a secção ao elemento finito
 */
public void setSection(double inertia, double area, double thickness) {
    this.inertia = inertia;
    this.area = area;
    this.thickness = thickness;
}

/**
 * Método para atribuir o material ao elemento finito
*/
public void setMaterial(String material, double elasticity, double shear, double poisson) {
    this.material = material;
    this.elasticity = elasticity;
    this.shear = shear;
    this.poisson = poisson;
}
```

O método que define o número de nós do elemento finito também atribui as coordenadas nodais pela chamada do método nodesCoordinates() que recebe como parâmetros as coordenadas da forma geométrica do elemento finito e o número de nós. Os outros dois métodos públicos e sem retorno limitam-se a atribuir o valor recebido às variáveis da classe FiniteElement. A classe possui ainda outros métodos para atribuição de informações de entre os quais se destaca o método setLoads() para atribuição das cargas aplicadas no elemento finito.

Os métodos anteriormente apresentados e outros que a classe possui necessários à definição dos atributos dos elementos finitos, no geral, limitam-se a atribuir às variáveis das classes um determinado valor recebido por parâmetro pelo que não realizam cálculos. Esta característica é comum a todo o projeto e tem como objetivo separar a lógica de cálculo da lógica de comunicação entre os diferentes objetos. Deste modo garante-se uma maior estabilidade no funcionamento dos métodos públicos.

Após a criação dos elementos finitos pode-se chamar os métodos para obter, por exemplo, a matriz de rigidez ou o vetor de forças nodais equivalentes. Estes métodos, que abaixo se apresentam, estão definidos como públicos e não recebem parâmetros. No geral, o funcionamento destes métodos é

assegurado por outros métodos que estão definidos na classe como privados e calculam os elementos que estes dois retornam.

```
/**
 * Método que retorna a matriz de rigidez elementar do elemento finito
*/
public double[][] getStiffnessMatrix() {
   double[][] stiffnessMatrix;

   if ("Gauss-Legendre".equals(quadrature) || "Gauss-Lobatto".equals(quadrature)) {
      stiffnessMatrix = numericalIntegration();
   } else {
      MatrixK matrixK = new MatrixK(elasticity, shear, poisson, inertia, area, thickness, type);
      stiffnessMatrix = matrixK.getStiffnessMatrix(L, a, b, c, nodes, theory);
   }

   return stiffnessMatrix;
}

/**
 * Método que retorna o vetor de forças nodais equivalentes do elemento finito
*/
public double[][] getLoadVector() {
   return loadVector;
}
```

O método getStiffnessMatrix() retorna a matriz de rigidez do elemento finito que pode ser calculada de duas maneiras. A primeira possibilidade corresponde à necessidade ou à exigência de se utilizar integração numérica para o cálculo da matriz de rigidez. No caso geral, a matriz de rigidez do elemento finito é obtida através da criação de um objeto da classe MatrixK. Criado este objeto, é apenas necessário chamar o método getStiffnessMatrix() para obter a matriz de rigidez. No caso do vetor de forças nodais equivalentes, este é definido no momento de atribuição das cargas aplicadas no elemento finito. Se não forem atribuídas cargas, o método retorna um vetor de zeros cuja dimensão depende do número de nós e dos graus de liberdade por nó do elemento finito.

A classe MatrixK permite obter a matriz de rigidez elementar para todos os tipos e teorias de elementos finitos com geometria regular. Os elementos finitos distorcidos são calculados com recurso a substituição de variável e integração numérica através da seleção de uma das duas quadraturas disponíveis. Para um elemento finito regular, o cálculo da matriz de rigidez elementar passa somente pela criação dentro da classe FiniteElement de um objeto da classe MatrixK. Assim, toda a lógica necessária ao cálculo da matriz de rigidez elementar fica escondida dentro da classe MatrixK. Esta característica evidencia-se em todas as classes do pacote finiteelement. Deste modo, as outras partes do programa não necessitam do conhecimento do funcionamento destas classes. Esta característica, além de simplificar a codificação, fomenta a reutilização das classes, melhora a manutenção e a expansão do código e reduz a propagação de erros.

Para os elementos finitos com geometria regular, a matriz de rigidez elementar foi deduzida num formato adimensional de modo a receber os parâmetros necessários ao seu cálculo sem a necessidade de se especificar qualquer tipo de unidades. O funcionamento da classe MatrixK consiste em fornecer a matriz de rigidez para o tipo de elemento finito criado. Para isso, a estrutura organizacional da classe chama o método interno que calcula a matriz de rigidez requerida. De modo a exemplificar o que a classe MatrixK executa internamente apresenta-se o seguinte método estático, definido dentro da classe MatrixK_1D, que permite calcular a matriz de rigidez de um elemento finito unidimensional.

```
/**
 * Método para calcular a matriz de rigidez de um elemento finito
 */
public static double[][] matrixK_1D(double L, int nodes) {
   double[][] matrixK;

    if (nodes == 2) {
      matrixK = new double[2][2];

      matrixK[0][0] = 1 / L;
      matrixK[0][1] = -1 / L;
      matrixK[1][0] = -1 / L;
      matrixK[1][1] = 1 / L;
   } else {
      matrixK = null;
   }

   return matrixK;
}
```

A matriz fornecida pelo método anterior deve posteriormente ser multiplicada pelo módulo de elasticidade e pela área da secção transversal do elemento finito antes de ser retornada pelo método getStiffnessMatrix() da classe MatrixK. A estrutura adotada para o cálculo da matriz de rigidez do elemento finito é semelhante para os outros tipos de elementos finitos e elementos finitos do mesmo tipo com diferente número de nós. A única tarefa lógica que é executada dentro da classe MatrixK corresponde à chamada do método específico para calcular a matriz de rigidez elementar para o elemento finito requisitado. Essa informação é obtida logo no momento da chamada do método construtor da classe e completada pelos parâmetros do método getStiffnessMatrix().

O cálculo do vetor de forças nodais equivalentes também é feito com recurso a uma estrutura análoga à apresentada para a matriz de rigidez elementar. Para o caso unidimensional, o vetor de forças nodais equivalentes é calculado pelo método seguinte.

```
/**
 * Método para calcular o vetor de forças nodais equivalentes
*/
public static double[][] vectorF_1D(double L, int nodes) {
   double[][] vectorF;

   if (nodes == 2) {
      vectorF = new double[2][1];
      vectorF[0][0] = L / 2;
      vectorF[1][0] = L / 2;
   } else {
      vectorF = null;
   }

   return vectorF;
}
```

O vetor fornecido pelo método anterior deve posteriormente ser multiplicado pelo valor da carga distribuída ao longo do elemento finito. Para a obtenção do vetor de forças nodais equivalentes é necessário criar dentro da classe FiniteElement um objeto da classe VectorF e, posteriormente, chamar o método getLoadVector().

Nas situações em que a matriz de rigidez é calculada com recurso a integração numérica, a classe FiniteElement possui um método privado que desencadeia o seu cálculo. Este método, consoante o tipo

de elemento finito, chama o método específico para calcular a matriz de rigidez elementar. O método seguinte calcula a matriz de rigidez de um elemento finito bidimensional com substituição de variável e recorrendo a integração numérica.

```
/**
 * Método para calcular a matriz de rigidez elementar
*/
private double[][] stiffnessMatrix_2D() {
   double[][] globalStiffnessMatrix = new double[nodes * dof][nodes * dof];

   double[] dimensions = {2.0, 2.0, 2.0, 2.0};
   double[][] matrixD = matrixD_2D(elasticity, poisson);
   double[][] weightsAndCoordinates = getWeightsAndCoordinates(2.0, 2.0, 2.0, 2.0);

   //Ciclo para todos os pontos de integração da matriz de rigidez
   int n = 0;
   while (n < points) {
      double[][] matrixJ, inverseJ, derivedMatrix, derivedMatrixB, matrixB, matrixK;
      double[] pointsCoordinates = weightsAndCoordinates[n];

      //Cálculo da matriz jacobiana e da sua inversa no ponto de integração (r, s)
      MatrixJ jacobianMatrix = new MatrixJ(type, nodesCoordinates, pointsCoordinates);
      matrixJ = jacobianMatrix.getJacobianMatrix(nodes);
      inverseJ = inverse(matrixJ);

      //Cálculo da matriz das derivadas das funções de forma
      derivedMatrix = derivedMatrix(type, pointsCoordinates, dimensions, nodes);
      derivedMatrixB = multiply(derivedMatrix, inverseJ);
      matrixB = matrixB(type, derivedMatrixB, nodes);

      double weightX = weightsAndCoordinates[n][0];
      double weightY = weightsAndCoordinates[n][2];
      double determinantJ = determinant(matrixJ);

      //Cálculo de uma matriz de rigidez avaliada no ponto de integração
      matrixK = multiply(multiply(transpose(matrixB), matrixD), matrixB);
      matrixK = multiply((weightX * weightY) * determinantJ, matrixK);

      //Cálculo da matriz de rigidez do elemento finito
      globalStiffnessMatrix = sum(globalStiffnessMatrix, matrixK);
      n++;
   }

   globalStiffnessMatrix = multiply(thickness, globalStiffnessMatrix);

   return globalStiffnessMatrix;
}
```

A utilização do método stiffnessMatrix_2D() necessita que a sua classe forneça as variáveis e os métodos que são exigidos durante o seu funcionamento. De um modo geral, o método stiffnessMatrix_2D() começa por inicializar a matriz de rigidez do elemento finito com zeros. As dimensões da matriz de rigidez elementar são definidas pela multiplicação do número de nós pelo número de graus de liberdade por nó.

Com as dimensões do elemento finito isoparamétrico, a matriz de elasticidade bidimensional, os pesos e as coordenadas dos pontos de integração está-se em condições de entrar no ciclo e calcular as sucessivas matrizes de rigidez auxiliares, avaliadas no respetivo ponto de integração, que vão sendo

somadas à matriz de rigidez elementar inicialmente nula. No final do ciclo e após a multiplicação da matriz obtida pela espessura chega-se a matriz de rigidez elementar do elemento finito.

Como já anteriormente referido, o método stiffnessMatrix_2D() necessita de vários parâmetros e métodos para o seu funcionamento. Alguns desses parâmetros são, por exemplo, a matriz de elasticidade e a matriz jacobiana. A matriz de elasticidade é importada da classe MatrixD que contém vários métodos que retornam a matriz de elasticidade para cada tipo de elemento finito.

```
/**
 * Método que retorna a matriz de elasticidade para um estado plano de tensão
 */
public static double[][] matrixD_2D(double E, double v) {
    double a = E / (1.0 - (v * v));
    double b = (E * v) / (1.0 - (v * v));
    double c = E / (2.0 * (1.0 + v));

    return new double[][] {{a, b, 0.0}, {b, a, 0.0}, {0.0, 0.0, c}};
}
```

Os pesos e as coordenadas dos pontos de integração variam consoante a quadratura, o número de pontos utilizados e o número de dimensões. Para um elemento finito bidimensional quadrilátero, os pesos e as coordenadas pela quadratura de Gauss-Legendre são obtidas pelo seguinte método estático.

```
/**
 * Método que retorna uma matriz com os pesos e respetivas coordenadas
 */
public static double[][] weightsCoordinates_2D(double a, double b, int points) {
    double[][] matrixWC;

    double weightX = a / 2.0;
    double weightY = b / 2.0;
    double pointX = a / 2.0;
    double pointY = b / 2.0;

    if (points == 1) {
        matrixWC = new double[1][4];

        matrixWC[0][0] = weightX * 2.0;
        matrixWC[0][1] = pointX * 0.0;
        matrixWC[0][2] = weightY * 2.0;
        matrixWC[0][3] = pointY * 0.0;
    } else {
        matrixWC = null;
    }

    return matrixWC;
}
```

Por conveniência, são apenas mostrados os pesos e as coordenadas para um ponto de integração no elemento finito. Este método recebe como parâmetros as dimensões do elemento finito bidimensional e o número de pontos que se pretende utilizar.

Para o cálculo da matriz de rigidez elementar avaliada num dado ponto é necessário definir a matriz jacobiana, calcular a sua inversa e o seu determinante. No cálculo da matriz jacobiana é necessário avaliar as funções de forma nos pontos obtidos pelo método anterior. De modo a ser possível calcular a matriz de rigidez avaliada nestes pontos é necessário obter as derivadas das funções de forma que compõem a matriz de deformação em função de r e s. O seguinte método fornece uma matriz com as

derivadas das funções de forma do elemento finito avaliadas num dado ponto de integração necessária para o cálculo dos termos da matriz de deformação.

```
/**
 * Método que retorna as derivadas das funções de forma
 */
public static double[][] derivedMatrix_2D(double r, double s, double a, double b, int nodes) {
   double[][]derivedMatrix;

   if (nodes == 4) {
      derivedMatrix = new double[4][2];

      derivedMatrix[0][0] = -1.0 / (2.0 * a) + s / (a * b);
      derivedMatrix[0][1] = -1.0 / (2.0 * b) + r / (a * b);
      derivedMatrix[1][0] = 1.0 / (2.0 * a) - s / (a * b);
      derivedMatrix[1][1] = -1.0 / (2.0 * b) - r / (a * b);
      derivedMatrix[2][0] = 1.0 / (2.0 * a) + s / (a * b);
      derivedMatrix[2][1] = 1.0 / (2.0 * b) + r / (a * b);
      derivedMatrix[3][0] = -1.0 / (2.0 * a) - s / (a * b);
      derivedMatrix[3][1] = 1.0 / (2.0 * b) - r / (a * b);
   } else {
      derivedMatrix = null;
   }

   return derivedMatrix;
}
```

Para obter os termos da matriz de deformação usada no produto matricial do cálculo da matriz de rigidez elementar avaliada em cada um dos pontos selecionados é necessário multiplicar a matriz fornecida pelo método estático derivedMatrix_2D() com a inversa da matriz jacobiana. Como apresentado no método stiffnessMatrix_2D(), a matriz de rigidez avaliada em cada um dos pontos selecionados é obtida pelo produto matricial entre as matrizes de deformação e a matriz de elasticidade. Após a multiplicação pelo determinante da matriz jacobiana e pelos pesos de integração, esta matriz de rigidez avaliada nos pontos de integração é somada à matriz de rigidez definida no início do método que após o fim do ciclo e multiplicada pela espessura corresponde à matriz de rigidez elementar do elemento finito.

8.3.2. Assemblagem e Resolução do Sistema de Equações

A classe FiniteElement contém as instruções para construir cada elemento finito. Assim, é necessário que, num dado momento e local, os elementos finitos sejam construídos. Para esse efeito foi criada uma classe designada por Processor.

Esta classe é responsável por construir todos os elementos finitos necessários para caracterização da malha de elementos finitos, fazer a assemblagem das suas contribuições no sistema global de equações e resolver esse mesmo sistema. Este modelo organizacional permite que sempre que se pretenda analisar uma nova estrutura seja apenas necessário criar um novo objeto da classe Processor. Deste modo, resolvido o sistema de equações, basta apenas obter a lista de elementos finitos construída pelos métodos desta classe para se ter acesso a todos os resultados ao nível de cada elemento finito. Esta classe também disponibiliza para apresentação na interface gráfica do utilizador o sistema global de equações.

Para a construção dos elementos finitos, a classe Processor tem de receber as informações relativas ao tipo de problema em análise, à geometria e ao material dos elementos finitos. Necessita também de receber as cargas aplicadas na estrutura, isto é, nos elementos finitos e nos nós da malha de elementos finitos, e os apoios estruturais. De modo a assegurar a integridade da classe, o método construtor recebe

dois parâmetros necessários à definição do tipo de problema para a construção dos elementos finitos e resolução do sistema global de equações.

```
/**
 *
 * @author André de Sousa
 */
public class Processor {

    public ArrayList<FiniteElement> listOfFiniteElements;

    /**
     * Método construtor da classe Processor
     */
    public Processor(String type, String theory) {
        this.type = type;
        this.theory = theory;
        listOfFiniteElements = new ArrayList();
    }
}
```

Para receber todos os atributos necessários ao cálculo da estrutura, a classe Processor possui métodos específicos para receber estas informações. Alguns destes métodos são apresentados abaixo e consistem na atribuição dos valores recebidos por parâmetro às variáveis definidas na classe.

```
/**
 * Método para atribuir as figuras geométricas desenhadas
 */
public void setDrawnFigures(ArrayList<DrawLine> arrayListLines, ArrayList<DrawPolygon>
arrayListPolygons) {
    this.arrayListLines = arrayListLines;
    this.arrayListPolygons = arrayListPolygons;
}

/**
 * Método para atribuir os apoios estruturais
 */
public void setDrawnSupports(ArrayList<DrawSupports> arrayListSupports) {
    this.arrayListSupports = arrayListSupports;
}

/**
 * Método para atribuir as diferentes cargas estruturais
 */
public void setDrawnLoads(ArrayList<DrawLoads> arrayListLoads) {
    this.arrayListLoads = arrayListLoads;
}
```

Importa referir que a ordem para a criação de um objeto da classe Processor parte do utilizador que pressiona um botão na interface gráfica do programa que aciona o cálculo da estrutura. Assim, na lógica desse botão estão somente as instruções para criar um novo objeto desta classe sempre que seja pressionado. Esta implementação permite libertar as áreas de tratamento dos eventos que ocorrem na interface gráfica do programa da parte lógica usada na análise estrutural pelo Método dos Elementos Finitos. Logo, os métodos criados para o tratamento desses eventos estão apenas focados no desencadeamento das operações e não na sua realização.

A primeira tarefa especificada na classe Processor após receber toda a informação necessária para o seu funcionamento interno consiste em construir a lista de elementos finitos. A informação necessária para construir os elementos finitos já foi evidenciada aquando da apresentação da classe FiniteElement. Para atribuir as cargas aplicadas em cada elemento finito, a classe possui um método que ao percorrer a lista das cargas estruturais faz a atribuição ao respetivo elemento finito. Note-se que para realizar esta operação é necessário conhecer as coordenadas dos elementos finitos e dos locais na malha onde as cargas estruturais estão aplicadas.

Para construir o sistema de equações globais da estrutura é necessário obter de cada elemento finito as suas contribuições e distribuí-las no sistema de equações globais de acordo com a informação da matriz de assemblagem individual. Esta operação é realizada dentro do método assembly() que é apresentado abaixo.

```
/**
 * Este método realiza a operação de assemblagem
 */
private void assembly() {
   for (FiniteElement finiteElement : listOfFiniteElements) {
      double[][] vectorF = finiteElement.getLoadVector();
      double[][] matrixK = finiteElement.getStiffnessMatrix();

      int[][] assemblyMatrix = finiteElement.getAssemblyMatrix();
      int length = assemblyMatrix.length;

      for (int i = 0; i < length; i++) {
         int column = assemblyMatrix[i][1] - 1;

         //Prenchimento da matriz de rigidez global
         for (int j = 0; j < length; j++) {
            int line = assemblyMatrix[j][1] - 1;
            stiffnessMatrix[line][column] = stiffnessMatrix[line][column] + matrixK[j][i];
         }

         //Preenchimento do vetor de solicitação global
         int line = assemblyMatrix[i][1] - 1;
         loadVector[line][0] = loadVector[line][0] + vectorF[i][0];
      }
   }
}
```

Salienta-se o facto da matriz de assemblagem ter sido previamente construída a partir da relação existente entre os nós dos elementos finitos e os nós da malha de elementos finitos. Construído o sistema de equações globais e introduzidas as condições de fronteira estão reunidas as condições para resolver o sistema de equações globais. Do mesmo modo que foram atribuídas as cargas aplicadas sobre os elementos finitos e consideradas as cargas concentradas aplicadas nos nós da malha de elementos finitos, para considerar os apoios estruturais é necessário comparar as coordenadas dos nós com as coordenadas dos apoios estruturais. Esta operação permite saber em que nós da malha de elementos finitos estão aplicados os apoios da estrutura.

Para realizar todas as operações matriciais necessárias à implementação computacional do Método dos Elementos Finitos foi criado um pacote com um conjunto de classes para disponibilizar os métodos necessários à realização destas operações algébricas. Este pacote foi designado por matrices e contém as classes Determinant, Inverse, Multiply, Subtract, Sum e Transpose. Para exemplificar o tipo de

métodos que estas classes disponibilizam, o método sum() da classe Sum realiza a operação de soma de duas matrizes recebidas por parâmetro.

```
/**
 * Método para realizar a operação de soma de duas matrizes
 */
public static double[][] sum(double[][] matrixA, double[][] matrixB) {
    double[][] sumMatrix;

    int linesA = matrixA.length;
    int columnsA = matrixA[0].length;
    int linesB = matrixB.length;
    int columnsB = matrixB[0].length;

    if (linesA == linesB && columnsA == columnsB) {
        sumMatrix = new double[linesA][columnsA];
        for (int i = 0; i < linesA; i++) {
            for (int j = 0; j < columnsA; j++) {
                sumMatrix[i][j] = matrixA[i][j] + matrixB[i][j];
            }
        }
    } else {
        sumMatrix = null;
    }

    return sumMatrix;
}
```

Após a resolução do sistema de equações são atribuídos aos nós dos elementos finitos o valor do seu deslocamento de acordo com a relação que estes possuem com os nós da malha de elementos finitos.

8.3.3. Extensões, Tensões e Esforços Internos

Resolvido o sistema de equações e atribuídos os valores dos deslocamentos nodais aos nós dos elementos finitos está-se em condições de obter todos os resultados de uma análise por elementos finitos ao nível de cada elemento finito.

As tensões nos elementos finitos podem ser obtidas através da relação entre tensões e deformações. O seguinte método, pertencente à classe FiniteElement, permite obter o valor das tensões nodais do elemento finito. Este método para retornar o valor das tensões nodais necessita de criar um objeto da classe NodalResults.

```
/**
 * Método qua retorna a lista de tensões nodais do elemento finito
 */
public ArrayList<double[][]> getNodalStresses() {
    NodalResults nodalResults = new NodalResults(type, theory, shape, nodes);
    nodalResults.setMaterial(elasticity, shear, poisson);
    nodalResults.setSection(inertia, area, thickness);

    ArrayList<double[][]> nodalStresses = nodalResults.getNodalStresses(L, a, b, c,
displacementVector);

    return nodalStresses;
}
```

Como demostrado no Capítulo 3, para se conhecer as tensões num dado ponto do elemento finito é necessário avaliar a matriz de deformação nesse ponto. Normalmente, os pontos que fornecem melhores resultados coincidem com os utilizados na quadratura de Gauss-Legendre. Para se obter a matriz de deformação, a classe NodalResults necessita de criar um objeto da classe MatrixB. O método getMatrixB() exemplifica o modo como este fornece a matriz correta para o tipo de elemento finito considerado.

```
/**
 * Método para obter a matriz B de um elemento finito
 */
public double[][] getMatrixB(double x, double y, double z, int nodes, String theory) {
   double[][] matrixB;

   switch (type) {
      case "1D":
         matrixB = matrixB_1D(x, L, nodes);
         break;
      case "2D":
         matrixB = matrixB_2D(x, y, a, b, nodes);
         break;
      case "3D":
         matrixB = matrixB_3D(x, y, z, a, b, c, nodes);
         break;
      case "Beams":
         matrixB = matrixB_Beams(x, L, nodes, theory);
         break;
      case "Slabs":
         matrixB = matrixB_Slabs(x, y, a, b, nodes, theory);
         break;
      default:
         matrixB = null;
         break;
   }

   return matrixB;
}
```

As matrizes de deformação para cada tipo de elemento finito estão programas com recurso a métodos que requerem os pontos onde devem ser avaliadas, as dimensões e o número de nós do elemento finito considerado. Deste modo, os métodos podem fornecer as matrizes de deformação para qualquer tipo de elemento finito.

A partir do valor dos deslocamentos nodais, da matriz de deformação e da matriz de elasticidade, as tensões nos pontos usados na quadratura de Gauss-Legendre num elemento finito tridimensional são calculadas através do método listOfStresses_3D(). Este método requer as dimensões do elemento finito e o vetor de deslocamentos nodais. A partir dos atributos da classe definidos aquando da criação do objeto da classe NodalResults, o método avalia a matriz de deformação nos pontos selecionados da quadratura de Gauss-Legendre onde devem ser calculadas as tensões. Obtidas todas as matrizes necessárias ao cálculo das tensões, o método realiza o produto matricial entre estas matrizes e adiciona o resultado a uma variável da classe ArrayList.

O cálculo dos esforços internos nos elementos finitos é outro dos resultados requeridos numa análise por elementos finitos. A estrutura dos métodos para calcular os esforços internos é bastante semelhante à dos métodos usados para calcular as tensões nos elementos finitos. O método listOfForces_Beams()

retorna uma lista com o valor do momento fletor atuante em vigas formuladas pela teoria de Euler-Bernoulli calculado nos pontos selecionados da quadratura de Gauss-Legendre.

```
/**
 * Método para calcular as tensões no caso tridimensional
 */
private ArrayList<double[][]> listOfStresses_3D(double a, double b, double c, double[][]
displacementVector) {
   ArrayList<double[][]> listOfStresses = new ArrayList();

   double[] dimensions = {0.0, a, b, c};
   double[][] matrixD = matrixD_3D(elasticity, poisson);

   MatrixB matrix = new MatrixB(type, 0.0, a, b , c);
   GaussLegendre gaussLegendre = new GaussLegendre(type, theory, points, dimensions);
   double[][] weightsCoordinates = gaussLegendre.getWeightsCoordinates();

   //Ciclo para todos os pontos no elemento finito
   int n = 0;
   while (n < points) {
      double[] pointsCoordinates = weightsCoordinates[n];

      double pointX = pointsCoordinates[1];
      double pointY = pointsCoordinates[3];
      double pointZ = pointsCoordinates[5];

      double[][] matrixB = matrix.getMatrixB(pointX, pointY, pointZ, nodes, theory);

      double[][] stresses = multiply(multiply(matrixD, matrixB), displacementVector);
      listOfStresses.add(stresses);
      n++;
   }

   return listOfStresses;
}
```

Atendendo ao facto de as vigas formuladas pela teoria Timoshenko e as lajes formuladas pela teoria de Reissner-Mindlin considerarem o efeito da deformabilidade por esforço transverso, os métodos implementados que retornam o valor das forças num dado ponto do elemento finito também calculam o valor do esforço transverso nesse ponto.

A implementação de todas as formulações de elementos finitos abordadas nesta dissertação seguiu os passos anteriormente apresentados. Assim, a programação de uma dada formulação de elementos finitos ou modelo estrutural consiste na devida implementação da informação disponibilizada.

Pretende-se com esta apresentação e exposição de alguns métodos evidenciar o modo como o Método do Elementos Finitos foi implementado na prática. Existem outros métodos e classes que não foram evidenciados. As funcionalidades existentes e o modo como são programas conduzem à necessidade de criar classes e métodos específicos. A informação apresentada só evidencia a fase de cálculo da estrutura. Note-se ainda que a implementação anteriormente apresentada se referiu apenas a alguns dos cálculos necessários a uma análise por elementos finitos.

Existem diferentes possibilidades de codificação de cada uma das tarefas necessárias a uma análise por elementos finitos. Toda a estrutura organizacional sobre o modo como foi implementada a análise por elementos finitos consiste numa proposta do autor. Outros modelos organizacionais podem

evidentemente ser explorados. O que importa salientar é que todas as abordagens têm as suas vantagens e desvantagens.

É da responsabilidade do programador decidir quais as abordagens que deve seguir de modo a obter a melhor implementação possível do Método dos Elementos Finitos.

```
/**
 * Método para calcular os esforços em vigas
 */
public ArrayList<double[][]> listOfForces_Beams(double L, double[][] displacementVector) {
   ArrayList<double[][]> nodalForces = new ArrayList();

   double[] dimensions = {L, 0.0, 0.0, 0.0};

   MatrixB matrix = new MatrixB(type, L, 0.0, 0.0, 0.0);
   GaussLegendre gaussLegendre = new GaussLegendre(type, theory, points, dimensions);
   double[][]weightsCoordinates = gaussLegendre.getWeightsCoordinates();

   //Ciclo para todos os pontos no elemento finito
   int n = 0;
   while (n < points) {
      double[][] matrixB, forces, M;
      double[] pointsCoordinates = weightsCoordinates[n];
      double point = pointsCoordinates[1];

      matrixB = matrix.getMatrixB(point, 0.0, 0.0, nodes, theory);
      M = multiply(multiply(-elasticity * inertia, matrixB), displacementVector);
      nodalForces.add(M);

      n++;
   }

   return nodalForces;
}
```

8.4. INTERFACE GRÁFICA DO UTILIZADOR

8.4.1. ELEMENTOS DA INTERFACE DO UTILIZADOR

Sendo o foco dos programas os utilizadores, deve-se procurar que os programas disponibilizem um meio que facilite a comunicação entre o utilizador e o computador. Até este momento apenas foram descritas as tarefas de cálculo referentes ao Método dos Elementos Finitos.

Para armazenar todas as classes que se relacionam com a interface do utilizador foram criados os pacotes frontend e backend. O pacote frontend contém todas as classes associadas à criação dos elementos gráficos da interface gráfica do utilizador enquanto o pacote backend contém as classes para a criação de objetos associados ao Método dos Elementos Finitos que são usados na interface gráfica. Apenas para exemplificar, o pacote frontend contém classes como a UserInterface, JPanelWelcome e ReactionsSupportTable enquanto o pacote backend contém classes como a DrawingPanel, ResultsPane e OpenSave.

Atendendo aos requisitos da modelação com elementos finitos e da apresentação gráfica dos resultados, no desenvolvimento da interface gráfica do utilizador houve a necessidade do desenvolvimento de ferramentas para desenho e funcionalidades para apresentação dos resultados. Das ferramentas de desenho construídas destacam-se as relativas ao desenho dos elementos finitos, as ferramentas para

anular ou repetir a introdução de objetos, ferramentas para selecionar, mover, cortar, copiar e colar, entre outras. Para visualização dos resultados era indispensável a visualização da deformada da malha de elementos finitos. Sendo o programa direcionado para o ensino do Método dos Elementos Finitos era também indispensável que o programa permitisse a visualização de elementos como a matriz de rigidez elementar ou o vetor de forças nodais equivalentes de cada elemento finito. Outras funcionalidades como a visualização dos mapas de tensões ou as tensões e direções principais para os problemas bidimensionais também foram desenvolvidas.

Para a construção da janela foi criada a classe UserInterface que permite criar a janela com a interface gráfica do programa desenvolvido. Apenas para exemplificar a criação desta janela, o código abaixo foi extraído da classe UserInterface e mostra a codificação que permite criar a janela que contém a interface gráfica do programa.

```
import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JFrame;

/**
 *
 * @author André de Sousa
 */
public class UserInterface extends JFrame {

   /**
    * Método construtor da classe UserInterface
    */
   public UserInterface() {
      Dimension screen = Toolkit.getDefaultToolkit().getScreenSize();
      setBounds((screen.width / 2) - (900 / 2), (screen.height / 2) - (600 / 2), 0, 0);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setTitle("FEM for Students");
      setSize(900, 600);
   }
}
```

Para visualizar a janela é criado um objeto desta classe dentro do método main() implementado na classe FEMforStudents. A construção de todos os elementos gráficos que compõem a interface do utilizador são chamados dentro do método construtor da classe. Qualquer interface gráfica é composta por painéis, botões, rótulos, campos de texto, tabelas, entre outros elementos gráficos. A utilização do NetBeans como ferramenta de desenvolvimento permitiu criar muitos destes elementos que compõem a interface gráfica do programa de uma forma simples e ágil.

Não devendo ser a interface gráfica um elemento estático é necessário definir os meios que permitam receber os eventos que ocorrem sobre ela. Alguns destes eventos são, por exemplo, eventos de *click* do rato ou ações de pressionar as teclas do teclado do computador. De modo a poder capturar estes eventos é necessário implementar interfaces ouvintes que capturem estas ações. A implementação destas interfaces exige a criação nas classes de um conjunto de métodos que recebem os eventos ocorridos na interface gráfica do utilizador. Assim, é apenas necessário definir toda a lógica associada a um dado evento ocorrido dentro do método que o recebe. Deste modo, todas as ações desencadeadas no programa partem de dentro dos métodos que recebem os diferentes eventos ocorridos na interface gráfica. Como facilmente se perceciona, o programador deve apenas associar a cada um dos eventos que está a capturar a respetiva função como, por exemplo, o cálculo da estrutura.

Para assegurar o funcionamento de qualquer classe e de modo a usufruir do encapsulamento é necessário construir vários métodos privados que executem pequenas operações elementares que quando devidamente encadeadas permitem realizar as tarefas para as quais foram definidos os métodos públicos da classe. É na classe que cria a interface gráfica do utilizador que reside muita da lógica que desencadeia o funcionamento de todas as operações descritas. A classe UserInterface é sem dúvida a classe mais complexa de todo o projeto, pois, tem de lidar com todos os elementos visuais com os quais o utilizador interage.

Para exemplificar a forma como estes eventos são recebidos, apresenta-se a adição ao painel de desenho dos métodos para receber os eventos de movimento e *click* dos botões do rato quando o ponteiro se encontra posicionado no seu interior. Estes métodos encontram-se todos definidos dentro da classe UserInterface e são o meio utilizado para que as ações desencadeadas pelo utilizador possam ser tratadas pelo objeto onde ocorreram.

```
jDrawingPanel.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent evt) {
        jDrawingPanel_MouseClicked(evt);
    }

    @Override
    public void mouseEntered(MouseEvent evt) {
        jDrawingPanel_MouseEntered(evt);
    }

    @Override
    public void mouseExited(MouseEvent evt) {
        jDrawingPanel_MouseExited(evt);
    }

    @Override
    public void mousePressed(MouseEvent evt) {
        jDrawingPanel_MousePressed(evt);
    }

    @Override
    public void mouseReleased(MouseEvent evt) {
        jDrawingPanel_MouseReleased(evt);
    }
});

jDrawingPanel.addMouseMotionListener(new MouseMotionAdapter() {
    @Override
    public void mouseDragged(MouseEvent evt) {
        jDrawingPanel_MouseDragged(evt);
    }

    @Override
    public void mouseMoved(MouseEvent evt) {
        jDrawingPanel_MouseMoved(evt);
    }
});
```

As interfaces gráficas do utilizador são orientadas a eventos. Consoante o tipo de evento ocorrido num determinado objeto deve-se procurar implementar o método adequado. Os componentes da interface gráfica do utilizador geram diferentes eventos e para diferentes eventos existem diferentes métodos para

os receber. Assim, toda a estrutura lógica associada a uma determinada ação é definida no interior desses métodos.

A construção da interface gráfica do utilizador consiste em posicionar no interior da janela do programa diferentes componentes visuais e em capturar o devido evento para desencadear a execução de uma dada tarefa. A informação aqui disponibilizada pretende mostrar que a construção de um programa não é mais que a execução sequencial de um conjunto elementar de tarefas.

Dos vários elementos gráficos que foram construídos destaca-se o painel de desenho que é definido pela classe DrawingPanel. Esta classe possui um diversificado conjunto de métodos que aquando da ocorrência dos respetivos eventos permitem executar todas as tarefas necessárias ao desenho da malha de elementos finitos e de todos os outros elementos requeridos na modelação do problema.

No código abaixo é apresentado um excerto da classe DrawingPanel, com o seu método construtor, que estende da classe JPanel herdando todos os seus atributos e ações.

```
import javax.swing.JPanel;

/**
 *
 * @author André de Sousa
 */
public class DrawingPanel extends JPanel {

    /**
     * Método construtor da classe DrawingPanel
     */
    public DrawingPanel() {
        setBackground(new java.awt.Color(255, 255, 255));
        setMaximumSize(new java.awt.Dimension(2500, 2500));
        setMinimumSize(new java.awt.Dimension(500, 500));
        setPreferredSize(new java.awt.Dimension(2500, 2500));
    }
}
```

Para desenhar as figuras no painel é necessário reescrever o método paintComponent() que recebe como parâmetro um objeto do tipo Graphics. Este método permite desenhar as figuras definidas pelo utilizador sempre que for invocado o método repaint() que ordenada o redesenho do conteúdo do painel.

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    Graphics2D shape = (Graphics2D) g.create();
}
```

8.4.2. Desenho dos Elementos do Modelo Estrutural

Numa análise por elementos finitos é necessário construir um modelo estrutural. Todos os elementos que compõem o modelo estrutural são desenhados no painel de desenho do programa construído a partir da classe DrawingPanel. Como referido anteriormente, esta classe contém as instruções necessárias para desenhar todos os elementos requeridos na modelação do problema.

Para desenhar os elementos finitos foram criadas diversas classes que são utilizadas dentro da classe DrawingPanel e que permitem desenhar diversas figuras geométricas a partir de um conjunto de

coordenadas. As coordenadas referidas são recebidas pelos métodos criados para o efeito que, a partir delas, criam uma nova figura geométrica. Apresentam-se, de seguida, alguns destes métodos de modo a exemplificar o seu funcionamento.

```
/**
 * Método para criar e mandar desenhar uma linha definitiva
 */
private void drawNewLine(int x1, int y1, int x2, int y2) {
    arrayListLines.add(new DrawLine(x1, y1, x2, y2, "Line"));
    repaint();
}

/**
 * Método para criar e mandar desenhar um retângulo definitivo
 */
private void drawNewRectangle(int[] xPoints, int[] yPoints) {
    arrayListPolygons.add(new DrawPolygon(xPoints, yPoints, 4, "Rectangle"));
    repaint();
}
```

Os métodos anteriormente apresentados criam uma nova figura geométrica a partir das coordenadas recebidas e adicionam-na a uma variável que a armazena. Para criar estas figuras geométricas que representam os elementos finitos foram criadas classes com métodos para as construir e desenhar. Algumas destas classes são, por exemplo, a classe DrawLine e a classe DrawPolygon. Para exemplificar, apresenta-se parte da classe DrawPolygon com o seu método construtor que permite construir qualquer figura geométrica do tipo polígono.

```
import java.awt.Polygon;
import java.io.Serializable;

/**
 *
 * @author André de Sousa
 */
public class DrawPolygon implements Serializable {

    public String shape;
    public Polygon polygon;
    public boolean selected;

    /**
     * Método construtor da classe DrawPolygon
     */
    public DrawPolygon(int[] xPoints, int[] yPoints, int nPoints, String name) {
        polygon = new Polygon(xPoints, yPoints, nPoints);
        selected = false;
        shape = name;
    }
}
```

Além das classes associadas ao desenho dos elementos finitos foram criadas classes, por exemplo, para desenhar os apoios estruturais e as cargas que podem ser aplicadas aos elementos finitos e aos nós da malha de elementos finitos. Cada uma destas classes possui métodos que desenham cada um destes elementos visuais.

De modo a permitir a manipulação dos objetos desenhados no painel de desenho, as classes desses objetos têm definidos métodos que possibilitam uma fácil interação com eles. Alguns destes métodos,

por exemplo, permitem selecionar o objeto ou obter as suas coordenadas. Estas ações permitem que o utilizador possa selecionar os elementos desenhados e realizar operações como mover, cortar, copiar ou colar. Se o programa não disponibilizasse estas operações, a sua qualidade ficaria substancialmente afetada.

O método mousePressedSelect(), pertencente à classe DrawingPanel, mostra como é feita a mudança de estado de um polígono. Para realizar esta operação é necessário saber se o polígono contém as coordenadas do ponto onde ocorreu o *click* do botão do rato. A classe DrawPolygon disponibiliza o método contains() que faz esta verificação e retorna verdadeiro ou falso. Deste modo, se a resposta for afirmativa, o estado do polígono muda para selecionado.

```
/**
 * Método para selecionar os objetos desenhados
 */
private void mousePressedSelect(int xPoint, int yPoint, boolean ctrlDown) {
    Point point = new Point(xPoint, yPoint);

    for (DrawPolygon polygon : arrayListPolygons) {
        if (polygon.contains(point)) {
            polygon.select(true);
        } else if (!ctrlDown) {
            polygon.select(false);
        }
    }

    repaint();
}
```

Para remover da lista de objetos todos os objetos selecionados, o método mousePressedCut() da classe DrawingPanel percorre a lista de objetos da classe DrawPolygon e elimina aqueles que estão selecionados.

```
/**
 * Método para eliminar os objetos selecionados
 */
private void mousePressedCut() {
    int i = 0;
    int j = arrayListPolygons.size();
    while (i < j) {
        DrawPolygon polygon = arrayListPolygons.get(i);
        if (polygon.selected) {
            arrayListPolygons.remove(i);
            j--;
            i--;
        }
        i++;
    }

    repaint();
}
```

Os métodos movePolygon() e getCoordinates(), pertencentes à classe DrawPolygon, mostram como é realizada a operação de translação de um polígono e a obtenção da matriz de coordenadas do polígono. O conhecimento das coordenadas e da forma é também importante para a construção dos elementos finitos. Todas as classes associadas ao desenho de objetos no painel de desenho possuem métodos equivalentes aos apresentados para a classe DrawPolygon.

O conhecimento da posição dos objetos é das informações mais valiosas, pois, a generalidade das operações descritas necessitam do seu conhecimento. Por exemplo, somente com o conhecimento das coordenadas dos objetos desenhados é possível selecioná-los ou movê-los. Muita da simplicidade das operações descritas advém da forma como foram criados estes métodos.

```
/**
 * Este método aplica uma translação ao polígono
 */
public void movePolygon(int deltaX, int deltaY) {
   polygon.translate(deltaX, deltaY);
}

/**
 * Este método retorna uma matriz com as coordenadas do polígono
 */
public int[][] getCoordinates() {
   int[] xPoints = polygon.xpoints;
   int[] yPoints = polygon.ypoints;
   int nPoints = polygon.npoints;

   int[][] coordinates = new int[nPoints][2];

   for (int i = 0; i < nPoints; i++) {
      coordinates[i][0] = xPoints[i];
      coordinates[i][1] = yPoints[i];
   }

   return coordinates;
}
```

Como anteriormente referido, o programa, além dos elementos finitos, também tem de desenhar as cargas e os apoios estruturais. No método seguinte, pertencente à classe DrawLoads, é apresentada a parte referente ao desenho de uma carga concentrada vertical descendente. Para executar esta operação, o método necessita de conhecer as coordenadas do ponto de origem de onde deve começar a desenhar a representação gráfica da carga. Essa informação é fornecida no momento da construção do objeto e fica armazenada na variável pointA.

```
/**
 * Método para desenhar uma carga concentrada
 */
private void drawConcentratedLoad(Graphics2D load) {
   int x = pointA.x;
   int y = pointA.y;

   Line2D drawLineA, drawLineB, drawLineC;

   //Carga concentrada vertical descendente
   if ("Vertical Positive".equals(this.verticalSign)) {
      drawLineA = new Line2D.Double(x, y, x, y - 25);
      drawLineB = new Line2D.Double(x, y, x - 5, y - 5);
      drawLineC = new Line2D.Double(x, y, x + 5, y - 5);

      load.draw(drawLineA);
      load.draw(drawLineB);
      load.draw(drawLineC);
   }
}
```

Uma das dificuldades enfrentadas para desenhar os diferentes elementos com o devido rigor técnico consiste em fornecer ao utilizador a possibilidade de desenhar os elementos com as corretas dimensões e exatamente no sítio pretendido. Atendendo ao exposto, houve a necessidade do desenvolvimento de funcionalidades para auxiliar o utilizador na etapa de modelação com elementos finitos. Estas funcionalidades executam tarefas elementares como mover a posição das coordenadas do rato, por exemplo, para o vértice mais próximo de um polígono quando o ponteiro do rato se encontra nas proximidades desse vértice. Sem estas funcionalidades o utilizador dificilmente conseguiria desenhar no sítio certo o que havia planeado.

Uma funcionalidade importante no momento da modelação da estrutura é a apresentação gráfica das dimensões dos elementos finitos. No momento em que o utilizador está a desenhar os elementos finitos, a apresentação desta informação permite que o utilizador possa controlar a suas dimensões.

A funcionalidade da grelha de pontos e o *snap* são ferramentas muito úteis na etapa de modelação da estrutura. Apesar da sua grande importância, porventura, são das funcionalidades mais simples de se construir. A grelha de pontos consiste em desenhar um ponto numa localização pré-estabelecida. Como a linguagem Java não disponibiliza nenhum método para desenhar pontos é necessário recorrer ao método para desenhar linhas. Assim, para se obter um ponto atribui-se as mesmas coordenadas para o ponto inicial e final da linha. O *snap* consiste em alterar as coordenadas do ponteiro do rato para um ponto existente na área de desenho. Quando o *snap* se encontra associado à grelha de pontos, o resultado consiste em alterar as coordenadas do ponteiro do rato para o ponto da grelha mais próximo.

Durante o processo de modelação, o utilizador necessita de poder definir as propriedades dos elementos finitos ou das cargas, mudar os apoios ou refinar a malha de elementos finitos. Para exemplificar, refere-se a funcionalidade de refinamento da malha de elementos finitos. O método meshRefinementLines() permite refinar uma malha de elementos finitos unidimensionais. Para executar esta tarefa ele recebe a linha que será dividida em duas novas linhas.

```
/**
 * Método de refinamento que retorna uma lista de linhas
 */
public static ArrayList<DrawLine> meshRefinementLines(DrawLine line) {
   ArrayList<DrawLine> listOfLines = new ArrayList();

   int[][] coordinates = line.getCoordinates();

   int x1 = coordinates[0][0];
   int y1 = coordinates[0][1];
   int x3 = coordinates[1][0];
   int y3 = coordinates[1][1];

   Point2D.Double point = midPoint(x1, y1, x3, y3);
   int x2 = (int) (Math.round(point.x));
   int y2 = (int) (Math.round(point.y));

   listOfLines.add(new DrawLine(x1, y1, x2, y2, line.shape, line.selected));
   listOfLines.add(new DrawLine(x2, y2, x3, y3, line.shape, line.selected));

   return listOfLines;
}
```

Quando uma malha é refinada, a tarefa interna que é executada corresponde à criação de uma nova malha de elementos finitos a partir da informação da malha inicial. Se existirem cargas distribuídas aplicadas nos elementos finitos da malha inicial é necessário que estas cargas fiquem sujeitas ao mesmo

padrão de refinamento. Caso contrário, as cargas que inicialmente estavam referenciadas a determinados elementos finitos deixam de o estar, pois, nessas posições, esses elementos finitos já não lá estão porque foram substituídos por novos elementos com geometria distinta.

Para que todas estas tarefas sejam executadas é necessário construir na interface gráfica elementos que permitam ao utilizador escolher a tarefa que pretende realizar, criar métodos para receber esses eventos e métodos para executar essas ordens. Todas estas tarefas são implementadas em níveis diferentes que exigem que os diferentes objetos possuam métodos específicos para a comunicação entre eles.

8.4.3. Apresentação Gráfica dos Resultados

Do ponto de vista de apresentação dos resultados, o programa apresenta as matrizes de rigidez elementares, os vetores de forças nodais e de deslocamentos para cada elemento finito, o sistema de equações globais e todos os resultados obtidos da análise, nomeadamente, as reações de apoio, a deformada da estrutura, as tensões e os esforços internos. A apresentação destes resultados passou pela construção de tabelas e de um painel para representar graficamente alguns destes resultados.

As tabelas foram construídas para apresentar o sistema de equações globais e todos os resultados numéricos ao nível dos elementos finitos, nomeadamente, tensões nodais e esforços nodais. Construídas as tabelas, sempre que o utilizador ordene a exibição do seu conteúdo, o programa executa a substituição do seu conteúdo através da mudança de modelo. Esta operação é consideravelmente mais eficiente que a operação de manipulação do modelo existente. O método printMatrix() mostra a construção de um modelo com o conteúdo de uma matriz quadrada.

```
/**
 * Método para construir a tabela com o conteúdo de uma matriz
 */
public static DefaultTableModel printMatrix(double[][] stiffnessMatrix) {
   DefaultTableModel table = new DefaultTableModel();
   int length = stiffnessMatrix.length;

   //Construção das colunas da tabela
   table.addColumn("");
   for (int i = 1; i <= length; i++) {
      table.addColumn(String.valueOf(i));
   }

   for (int i = 0; i < length; i++) {
      String[] lineResults = new String[length + 1];

      //Construção do conteúdo de uma linha da tabela
      lineResults[0] = String.valueOf(i + 1);
      for (int j = 0; j < length; j++) {
         lineResults[j + 1] = String.valueOf(stiffnessMatrix[i][j]);
      }

      table.addRow(lineResults);
   }

   return table;
}
```

O método printMatrix() além de imprimir o conteúdo de uma matriz quadrada ainda adiciona a numeração às linhas e colunas da tabela.

A apresentação de todos os elementos usados no cálculo e os resultados obtidos foi um dos principais focos do programa. A apresentação gráfica dos resultados foi desenvolvida com a construção de classes que permitem a visualização, por exemplo, da deformada, dos mapas de tensões ou das tensões e direções principais. Estas funcionalidades são muito úteis ao permitir, por exemplo, visualizar a forma da deformada da estrutura ou a distribuição de tensões nas peças. Para os elementos do tipo barra está disponível a visualização dos diagramas de esforços.

Para visualização dos resultados de forma gráfica, à semelhança do realizado para desenhar os vários elementos, foi construída uma classe designada por ResultsPane que, a partir da informação obtida de cada elemento finito, representa os diferentes resultados. Após a criação do objeto desta classe é apenas necessário chamar o método relativo ao tipo de resultado que se pretende visualizar. Esta classe cria objetos de muitas das classes que já foram referidas aquando da apresentação da classe DrawingPanel.

O desenho da deformada da estrutura consiste em somar à posição dos nós da estrutura indeformada o valor do deslocamento nodal obtido do cálculo segundo cada direção ortogonal. Contudo, como o valor do deslocamento é normalmente pequeno, é necessário adotar um fator de escala que permita visualizar convenientemente a deformada da estrutura. Esse fator fica ao critério do programador que o deve definir consoante a dimensão que pretende obter para a deformada da estrutura. O método seguinte exemplifica o modo como são definidas as posições dos nós da deformada da estrutura para o caso das lajes.

```
/**
 * Método para definir a posição dos nós da estrutura deformada
 */
private void nodes_deformedStructure(String type, String theory, int[][] nodesCoordinates,
        double[][] displacements) {
    int[][] coordinates = new int[nodesCoordinates.length][2];

    switch (type) {
        case "Slabs":
            int[][] perspectiveSlabs = createPerspective(nodesCoordinates);

            for (int i = 0; i < nodesCoordinates.length; i++) {
                coordinates[i][0] = perspectiveSlabs[i][0];
                coordinates[i][1] = perspectiveSlabs[i][1]
                        + (int) Math.round(displacements[i * 3][0] * deformedFactor);
            }
            break;
    }

    for (int[] nodeCoordinates : coordinates) {
        listNodesDeformed.add(new DrawNode(nodeCoordinates[0], nodeCoordinates[1],
defaultColor));
    }
}
```

Uma ferramenta bastante útil para o utilizador visualizar a deformada da estrutura consiste na possibilidade de ajuste da sua dimensão. Este ajuste da dimensão da deformada é conseguido através da adição à interface gráfica de um *slider*. Assim, a dimensão da deformada é ajustada em conformidade a partir do valor de posicionamento fornecido por este componente visual.

Para desenhar as linhas de isovalores e os mapas de tensões foi criada a classe IsolinesAndMaps. O desenho das linhas de isovalores e dos mapas de tensões é realizado através da analogia com a representação de curvas de nível. Basicamente, para desenhar estas duas representações gráficas começa-se por idealizar uma malha de triângulos definidos a partir das coordenadas dos nós dos elementos finitos. Definida a escala de cores, para cada triângulo, unem-se os pontos de igual tensão

correspondentes à divisão efetuada na escala de cores. No caso dos mapas de tensão, estes pontos são unidos de modo a formar um polígono cujo interior é pintado com a cor atribuída no momento da sua construção.

8.5. PERSISTÊNCIA DE DADOS

A necessidade de armazenar o trabalho criado pelo utilizador no programa exige o armazenamento dessa informação na memória permanente do computador de modo a que esta possa ser recuperada futuramente após o fecho do programa.

A solução adotada para garantir a persistência dos dados foi de recorrer a ficheiros de bytes. Esta escolha revela-se mais interessante na medida que se pode recorrer aos recursos disponibilizados na biblioteca do Java para manipulação de ficheiros. De modo a automatizar a gravação e a recuperação de ficheiros foi criada a classe OpenSave.

```
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
import javax.swing.JOptionPane;

/**
 *
 * @author André de Sousa
 */
public class OpenSave {

   /**
    * Construtor da classe para gravar os objetos do projeto
    */
   public OpenSave(String absolutePath, ArrayList<DrawPolygon> arrayListPolygons) {
      saveAllObjects(absolutePath, arrayListPolygons);
   }

   /**
    * Método para gravar as listas de objetos do projeto
    */
   private void saveAllObjects(String absolutePath, ArrayList<DrawPolygon>
arrayListPolygons) {
      try (FileOutputStream stream = new FileOutputStream(absolutePath)) {
         try (ObjectOutputStream outputStream = new ObjectOutputStream(stream)) {
            outputStream.writeObject(arrayListPolygons);
         } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Error! Unable to save the project.");
         }
      } catch (Exception e) {
         JOptionPane.showMessageDialog(null, "Error! Unable to save the project.");
      }
   }
}
```

De modo a compactar a apresentação da classe é apenas apresentada a gravação da lista de polígonos desenhados. Para recuperar a informação gravada é realizada uma operação inversa à apresentada. O mais importante a reter sobre a gravação e recuperação de ficheiros consiste no tipo de objetos que podem ser gravados e na obtenção do caminho do local de realização desta operação.

A metodologia de gravação e recuperação de ficheiros apresentada só é aplicável a objetos serializáveis, isto é, objetos que implementem a interface Serializable. A classe ArrayList e a classe DrawPolygon têm implementada esta interface. Desta forma, a gravação de ficheiros e a sua recuperação em Java é uma operação bastante simples.

Importa ainda salientar que qualquer conexão tem de ser aberta e no final de realizadas todas as operações encerrada. A estrutura lógica apresentada realiza estas tarefas mesmo nas situações em que possam ocorrer erros inesperados.

9

APRESENTAÇÃO DO PROGRAMA FEM FOR STUDENTS

9.1. DESCRIÇÃO DO PROGRAMA

O objetivo desta dissertação consistiu em construir integralmente um programa de modelação e análise estrutural pelo Método dos Elementos Finitos. O programa foi construído para utilização em computador usando a linguagem de programação Java e foi batizado com o nome FEM for Students.

O foco do programa é a modelação e análise estrutural pelo Método dos Elementos Finitos. Assim, o programa disponibiliza várias ferramentas para a etapa de modelação e funcionalidades para visualização dos resultados. De entre as diversas ferramentas disponíveis, o programa permite o desenho de diversos tipos de elementos finitos, com diferente número de nós e formulados por diferentes teorias. Além destas funcionalidades, o programa possibilita a inserção de diversos tipos de carregamentos e apoios estruturais. O uso da integração numérica é uma das opções disponíveis no momento que antecede o cálculo. Deste modo, o utilizador ao escolher um método de integração deve também definir o número de pontos para avaliar os integrais. Do ponto de vista de apresentação dos resultados obtidos, a par de uma visualização gráfica, o programa também mostra os resultados numericamente, isto é, toda a informação relevante é apresentada em tabelas.

Algumas das preocupações durante o desenvolvimento do programa consistiram em fornecer uma interface apelativa, ferramentas que otimizassem a etapa de desenho dos modelos e funcionalidades para visualização dos resultados da análise estrutural. Sendo a usabilidade uma das principais exigências houve a preocupação de desenvolver um conjunto de ferramentas que agilizassem todo o processo de modelação e análise por elementos finitos.

Como anteriormente salientado, o programa possibilita o uso de diferentes formulações de elementos finitos. Além disto, o programa abrange a generalidade dos modelos estruturais usados na prática de engenharia. Do ponto de vista de simulação numérica o programa permite que sejam feitas alterações ao modelo de forma bastante intuitiva e visualizá-las rapidamente de modo a compreender o seu impacto no comportamento da estrutura.

De modo a facilitar os desenvolvimentos futuros houve a necessidade de organizar convenientemente o projeto do programa. Isto passou pela criação de classes genéricas que facilmente permitem a criação de novos objetos sem ser necessário entender pormenorizadamente o seu funcionamento. Assim, por exemplo, a adição de novos elementos finitos faz-se sem recurso à alteração do funcionamento das outas classes. Isto é possível porque os métodos desenvolvidos funcionam independentemente do tipo de elemento finito selecionado.

9.2. Painel Inicial e Separadores

Após a inicialização, o programa apresenta um painel inicial que permite ao utilizador escolher o tipo de modelo estrutural que pretende para o seu projeto. A figura 9.1 mostra a interface gráfica do programa após a inicialização.

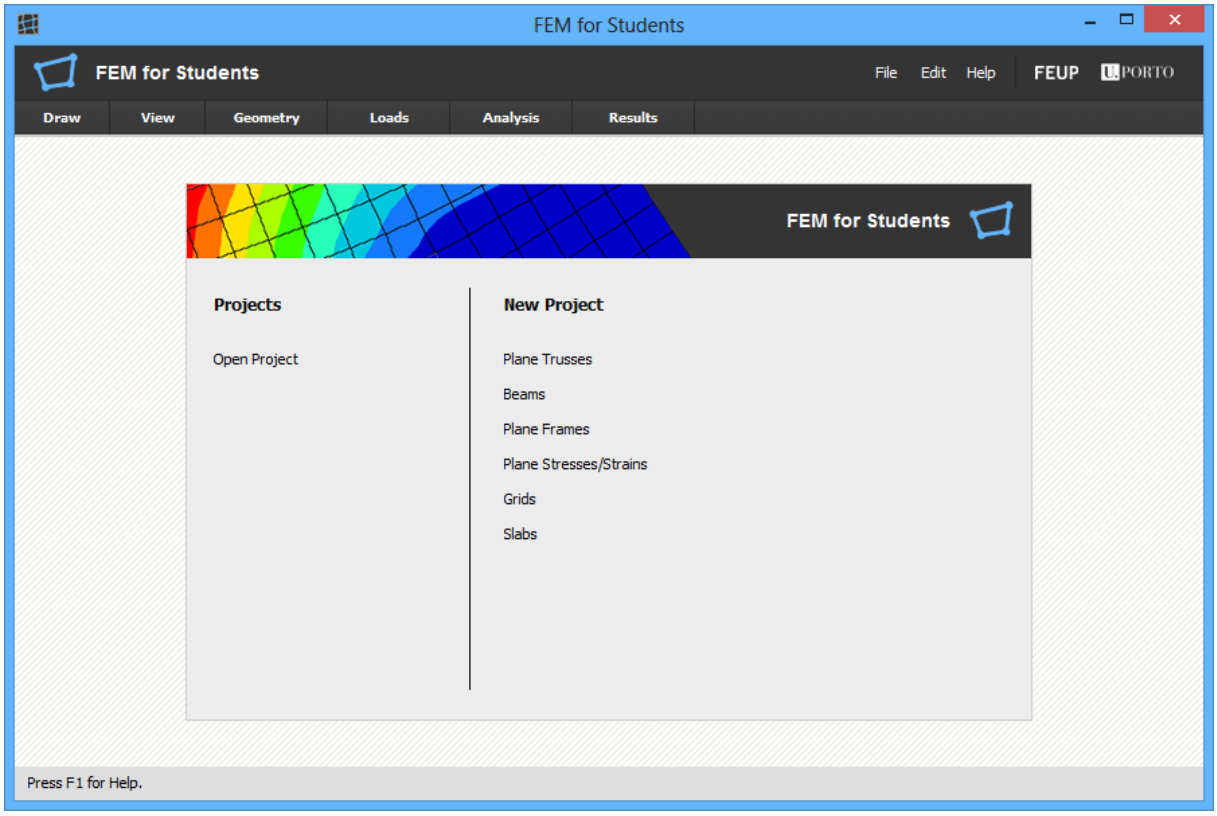


Fig.9.1 – Visualização da janela e do painel inicial do programa.

Atendendo às disposições de construção de interfaces, os elementos que aparecem após a inicialização do programa estão dispostos de modo a que o utilizador apenas se foque na escolha do modelo estrutural para o seu projeto. O painel que aparece ao centro da janela permite que o utilizador escolha um tipo de modelo estrutural ou um projeto existente. Selecionado o tipo de modelo para o projeto a desenvolver entra-se na área de modelação do programa e passam a estar disponíveis todas as ferramentas para desenho dos elementos finitos e todos os outros elementos necessários à modelação.

De modo a tornar o programa intuitivo, as ferramentas estão dispostas por um conjunto de separadores organizados sequencialmente pelas etapas da modelação, análise estrutural e visualização dos resultados da análise. Assim, a primeira tarefa consiste em desenhar a malha de elementos finitos. De seguida, são definidos os restantes detalhes como, por exemplo, o número de nós dos elementos finitos, as propriedades dos materiais ou as condições de apoio. Na próxima etapa aplicam-se os carregamentos à estrutura. Após todas as características do modelo estarem definidas é realizado o seu cálculo. Por fim, são visualizados os resultados da análise obtidos para o modelo construído.

Atendendo a estas etapas, as ferramentas e funcionalidade do programa estão organizadas nos seguintes seis separadores:

- Draw;
- View;
- Geometry;

- Loads;
- Analysis;
- Results.

O separador Draw contém as ferramentas necessárias ao desenho dos diferentes elementos finitos disponíveis no programa. Possui algumas funcionalidades de produtividade, nomeadamente, seleção dos elementos finitos, mover, cortar, copiar e colar. Permite também a inserção de pontos de referência através de coordenadas introduzidas pelo teclado para auxiliar a construção da malha de elementos finitos.

O separador View contém ferramentas de visualização, nomeadamente, disponibiliza a possibilidade de mover o painel, fazer *zoom*, mostrar uma malha de pontos, e outras funcionalidades que facilitam o desenho dos elementos finitos.

O separador Geometry contém as funcionalidades para definir o número de nós dos elementos finitos, as restantes propriedades das secções dos elementos finitos, as propriedades do material, refinamento das malhas de elementos finitos e adição dos apoios estruturais. Algumas destas funcionalidades abrem um painel lateral onde aparecem os elementos necessários à realização das respetivas operações.

O separador Loads contém as ferramentas para atribuição dos carregamentos ao modelo estrutural construído. Os botões deste separador abrem um painel lateral onde é possível definir as propriedades da carga que se pretende adicionar ao modelo. Consoante o modelo estrutural é possível adicionar carregamentos como cargas concentradas, momentos fletores, cargas lineares distribuídas, cargas axiais distribuías e/ou carregamentos de superfície.

Relativamente ao separador Analysis, este disponibiliza, por exemplo, para os elementos finitos de viga e de laje a seleção da teoria de formulação. Possibilita ainda escolher se a análise vai ser analítica ou numérica e o botão para efetuar o cálculo da estrutura. Do ponto de vista de uma análise numérica, o programa permite usar a quadratura de Gauss-Legendre ou a quadratura de Gauss-Lobatto para realizar a integração numérica.

O separador Results, e último, contém os botões para escolher o tipo de resultado obtido da análise estrutural a visualizar. Consoante o modelo selecionado é possível ver, por exemplo, o sistema de equações globais, todos os resultados ao nível de cada elemento finito, a deformada da estrutura, os diagramas de esforços, os mapas de tensões e/ou as tensões e direções principais.

9.3. Etapa de Modelação

Escolhido o modelo estrutural para o projeto, o programa apresenta o painel de desenho e passam a estar disponíveis todas as ferramentas para modelação. Como referido anteriormente, as funções do programa encontram-se organizadas por um conjunto de separadores. Muitos dos botões aí apresentados abrem painéis laterais onde são disponibilizadas as áreas com as funcionalidades a que o utilizador pretende aceder. Se a construção do modelo respeitar todas as etapas, estas funcionalidades vão sendo acedidas sequencialmente. Deste modo, o utilizador não necessita de saltar de separador em separador ou painel em painel para realizar um determinada tarefa.

Na figura 9.2 é mostrado o conteúdo do separador Draw que apresenta todas as ferramentas para modelar qualquer problema bidimensional. Pela análise da figura observa-se no primeiro grupo as ferramentas de desenho de várias formas geométricas como linhas, triângulos ou quadriláteros. No segundo grupo estão as ferramentas que permitem retroceder ou avançar na introdução de elementos. No terceiro e último grupo estão algumas ferramentas de produtividade como, por exemplo, as ferramentas para

cortar, copiar ou colar. Destaca-se que para mover um objeto ou copiá-lo é necessário, em primeiro lugar selecioná-lo. Só depois é possível movê-lo ou copiá-lo.

Para selecionar um dos objetos desenhados é necessário, em primeiro lugar, escolher a ferramenta *select* do separador Draw. Depois, basta clicar no interior do objeto ou, no caso de uma linha, nas suas proximidades. Para selecionar vários objetos é necessário manter pressionada a tecla *control* ou usar o retângulo de seleção. Para cancelar o desenho de um objeto pode-se pressionar a tecla *escape*.

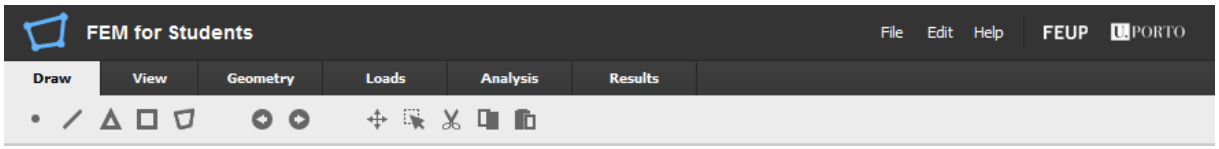


Fig.9.2 – Ferramentas do separador Draw.

Algumas funcionalidades como a possibilidade de ampliação ou redução da dimensão do desenho estão disponíveis no separador View. A figura 9.3 mostra o conteúdo do separador View. As ferramentas deste separador podem ser usadas tanto na etapa de modelação como na visualização gráfica dos resultados. Por exemplo, para ativar a malha de pontos e o *snap* é necessário, em primeiro lugar, ativar a *grid* e depois o *snap*.

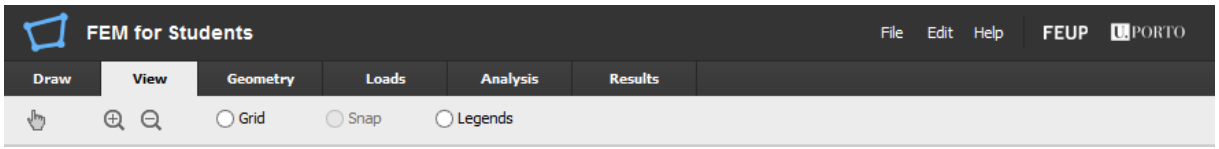


Fig.9.3 – Ferramentas do separador View.

Desenhada a malha de elementos finitos, a etapa seguinte da modelação consiste em definir todas as propriedades dos elementos finitos e em colocar os apoios estruturais. No separador Geometry, mostrado na figura 9.4, são disponibilizadas as ferramentas necessárias para realizar estas tarefas.

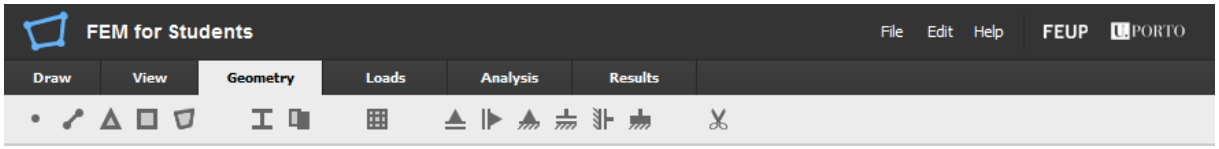


Fig.9.4 – Ferramentas do separador Geometry.

No primeiro grupo encontram-se as funções para atribuição do número de nós aos elementos finitos. No segundo grupo encontram-se as funções para a definição das restantes dimensões dos elementos finitos e atribuição do material. Todas estas funcionalidades estão associadas a painéis laterais. A ferramenta seguinte permite o refinamento automático da malha de elementos finitos. O último grupo de funcionalidades corresponde aos botões para selecionar o tipo de apoio estrutural a adicionar ao modelo.

Como referido, algumas das ferramentas do painel Geometry abrem painéis laterais. Para definir o número de nós dos elementos finitos existem os painéis laterais Bars, Triangles e Quadrilaterals. Para definir as restantes dimensões dos elementos finitos existe o painel lateral Sections. Este painel, consoante o tipo de elemento finito, permite especificar a espessura, a área, a inércia e/ou a constante de torção. Estas propriedades podem ser especificadas individualmente para cada um dos elementos finitos da malha. Para isso, o utilizador deve previamente criar a secção e posteriormente selecionar o respetivo elemento finito. Selecionado o elemento finito, o utilizador deve pressionar o botão direito do rato e escolher a opção *Finit Element* que abre uma nova janela onde são detalhadas as características do elemento finito.

Quanto às propriedades do material dos elementos finitos, o painel Materials permite ao utilizador especificar o valor para os módulos de elasticidade e de distorção e o valor do coeficiente de Poisson. A figura 9.5 mostra o conteúdo painel Materials que permite especificar os valores das propriedades do material selecionado. Relativamente ao refinamento da malha, o painel lateral Mesh permite definir o nível de refinamento a aplicar à malha de elementos finitos.

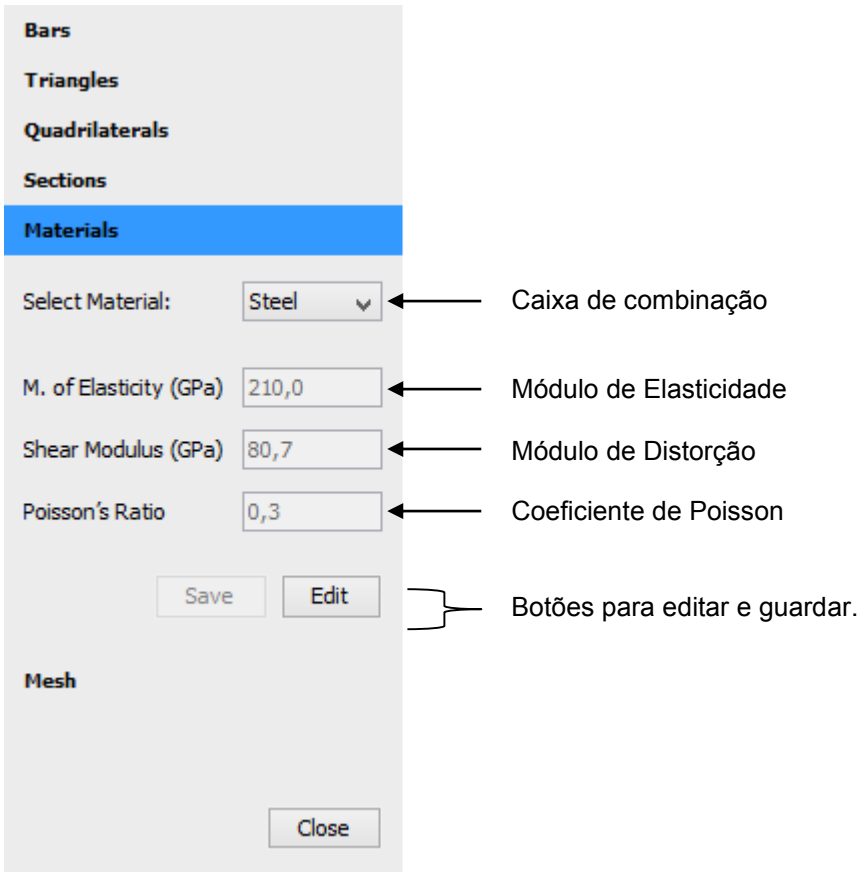


Fig.9.5 – Conteúdo do painel lateral Materials.

Para adicionar os apoios estruturais ao modelo é necessário, em primeiro lugar, selecionar no separador Geometry o tipo de apoio estrutural. Depois, deve-se clicar sobre o nó da malha de elementos finitos em que se pretende aplicar o apoio estrutural. Note-se que os botões relativos aos apoios estruturais ficam habilitados consoante o modelo estrutural selecionado.

O passo seguinte na etapa de modelação consiste em colocar as cargas no modelo. Na figura 9.6 é mostrado o conteúdo do separador Loads. Neste separador, consoante o tipo de modelo estrutural, ficam habilitados diferentes tipos de carregamentos para adicionar ao modelo. Consoante o tipo de modelo, é possível adicionar cargas concentradas, momentos fletores, cargas lineares distribuídas, cargas axiais distribuídas e/ou carregamentos de superfície.

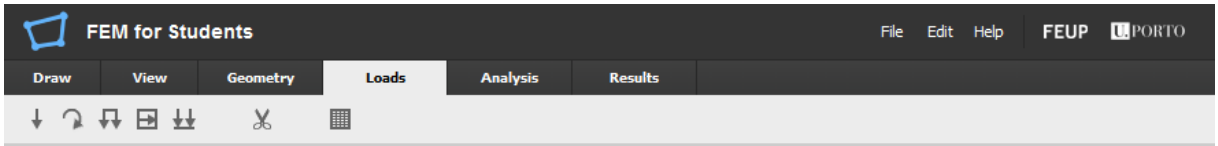


Fig.9.6 – Ferramentas do separador Loads.

A figura 9.7 mostra o conteúdo do painel lateral Uniformly Distributed Loads que permite criar uma carga linear uniformemente distribuída. O programa só permite a adição de cargas ao modelo após a sua

criação no painel lateral. Para criar uma carga é necessário selecionar a opção *New* na caixa de combinação. Criada a carga, é necessário preencher os campos de texto com o seu valor e no final carregar no botão para guardar. Posteriormente, é apenas necessário clicar com o botão do rato nos locais onde se pretende adicionar a carga. Note-se que para adicionar cargas concentradas é apenas necessário definir um único nó. Já as cargas distribuídas, à exceção do carregamento de superfície, exigem que se selecione dois nós do elemento finito.

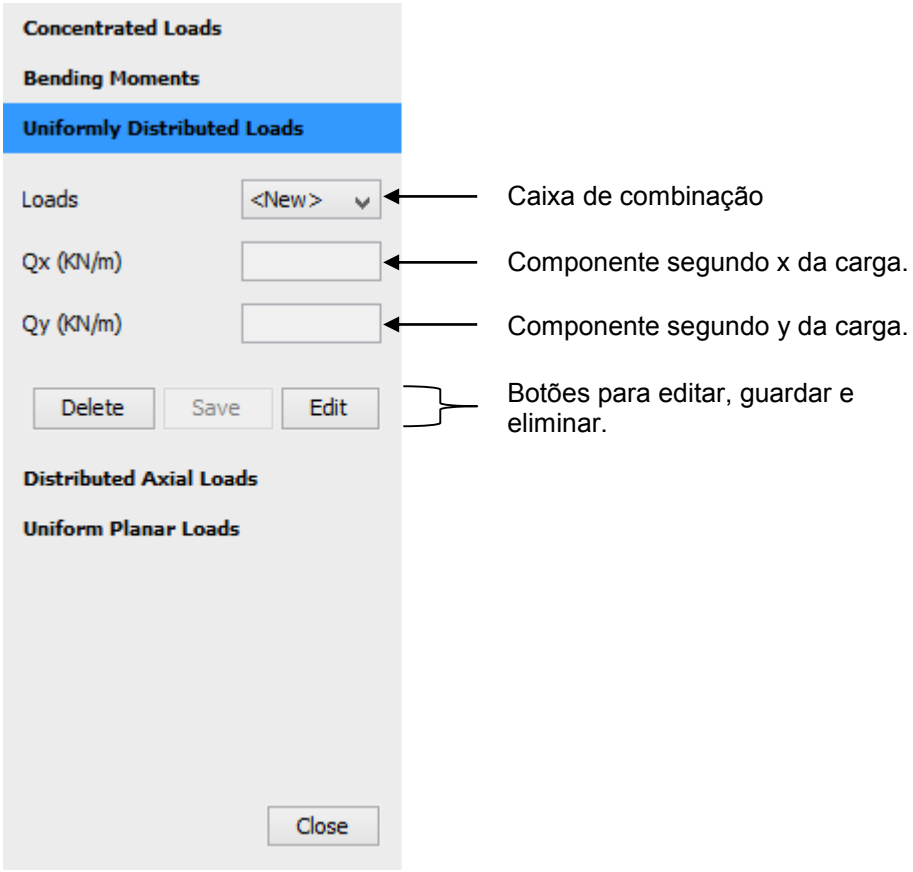


Fig.9.7 – Conteúdo do painel lateral Uniformly Distributed Loads.

Posteriormente, é sempre possível editar o valor da carga ou removê-la do modelo. Para remover uma carga do modelo é necessário selecionar a ferramenta para remover as cargas disponível no separador Loads e, em seguida, clicar com o rato sobre o nó ou um dos nós em que a carga está aplicada. É também possível visualizar a lista com todos os carregamentos estruturais aplicados ao modelo.

9.4. Tipos de Análise

Construída a malha de elementos finitos, definidas as suas propriedades e introduzidas as condições de fronteira, o modelo fica em condições de ser calculado. Nesta etapa, são construídos os elementos finitos e o sistema de equações globais que, após a sua resolução, permite caracterizar o comportamento da estrutura. O separador Analysis, representado na figura 9.8, disponibiliza o botões para escolher a teoria de formulação dos elementos finitos, quando aplicável, e se o cálculo dos elementos finitos vai ser analítico ou numérico. Disponibiliza ainda o botão para calcular a estrutura.

A escolha da teoria de formulação dos elementos finitos é aplicável somente às formulações de elementos finitos de viga e de laje. Para as vigas, o utilizador pode escolher entre a teoria de Euler-

Bernoulli e a teoria de Timoshenko. No caso das lajes pode escolher entre a teoria de Kirchhoff e a teoria de Reissner-Mindlin. De modo a poder escolher a teoria aplicável ou definir os dados necessários para realizar a análise numérica, o programa quando solicitado abre painéis laterais onde estas opções são disponibilizadas. É também nesta etapa que, quando aplicável, o utilizador deve escolher entre realizar uma análise em estado plano de tensão ou deformação.

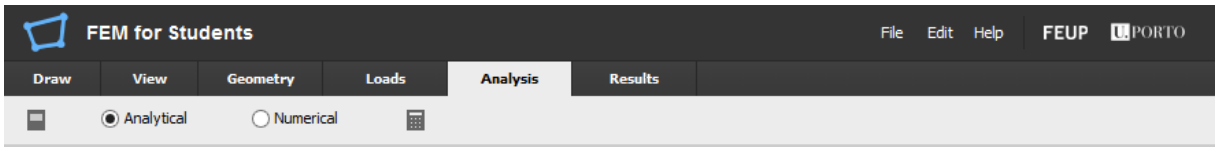


Fig.9.8 – Funcionalidades do separador Analysis.

Relativamente à possibilidade de calcular os elementos finitos com recurso a integração numérica, estão disponíveis duas quadraturas. Assim, o utilizador pode escolher entre a quadratura de Gauss-Legendre e a quadratura de Gauss-Lobatto. Independentemente da quadratura escolhida é necessário definir o número de pontos de integração para cada tipo de elemento finito do modelo. A figura 9.9 mostra o conteúdo do painel lateral Numerical Analysis para selecionar o tipo de quadratura e o número de pontos usados para avaliar as funções.

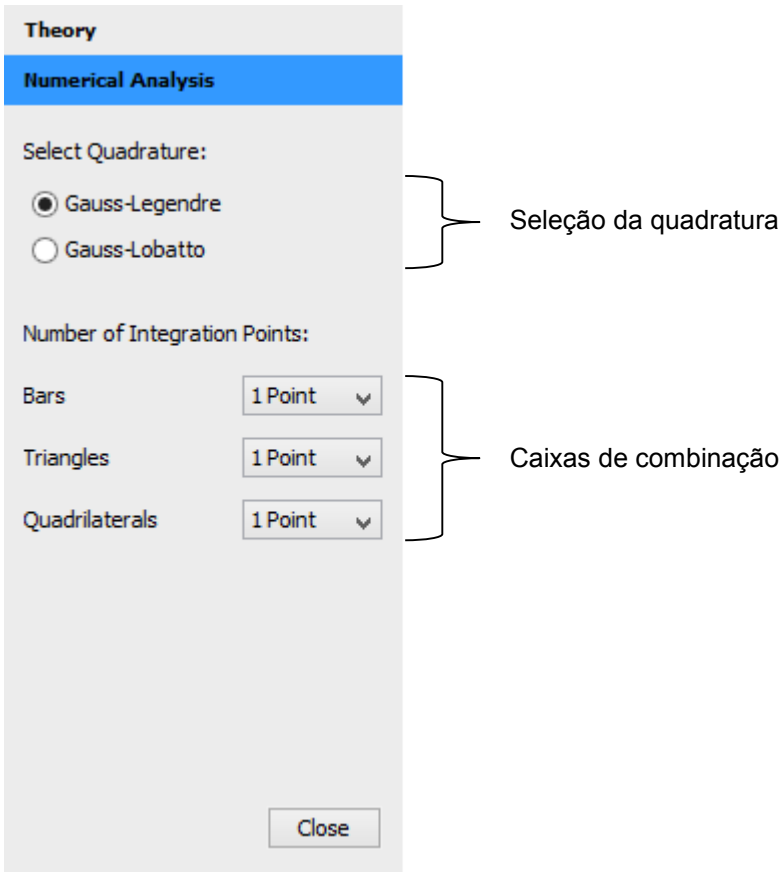


Fig.9.9 – Conteúdo do painel lateral Numerical Analysis.

A principal diferença entre escolher uma análise analítica ou numérica está no facto de o utilizador não ter qualquer poder de controlo sobre o modo como são calculados os elementos finitos sempre que selecione a opção de análise analítica.

Construído o modelo e definidos todos os elementos necessários para o calcular, o utilizador deve clicar no botão Calculate do separador Analysis para efetuar o cálculo.

9.5. APRESENTAÇÃO DOS RESULTADOS

Realizada a análise, entra-se na etapa de visualização dos resultados. No separador Results, apresentado na figura 9.10, estão disponíveis as opções para escolher o tipo de resultado para visualização. No primeiro grupo de botões estão disponíveis as opções relativas à visualização do sistema global de equações. No segundo grupo estão disponíveis as opções para visualização dos resultados em tabelas ao nível dos elementos finitos. Assim, além do sistema global de equações, é possível visualizar a equação de equilíbrio para cada elemento finito, as reações de apoio, as forças nodais e as tensões nodais.

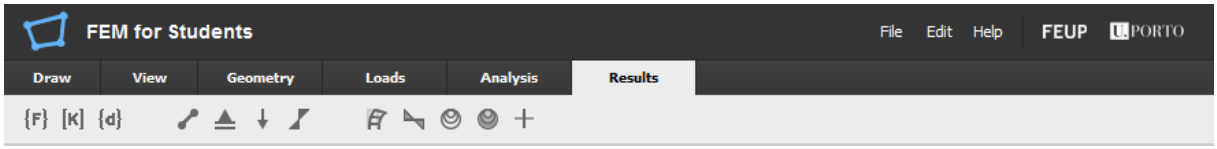


Fig.9.10 – Funcionalidades do separador Results.

No último grupo estão concentradas as funcionalidades para apresentação gráfica dos resultados. Para todos os modelos é possível visualizar a deformada da estrutura. A figura 9.11 mostra o conteúdo do painel lateral Displacements. Como se pode observar pela imagem do painel lateral Displacements, o utilizador pode controlar a dimensão do desenho da deformada da estrutura, visualizar os nós da malha de elementos finitos e a sua numeração.

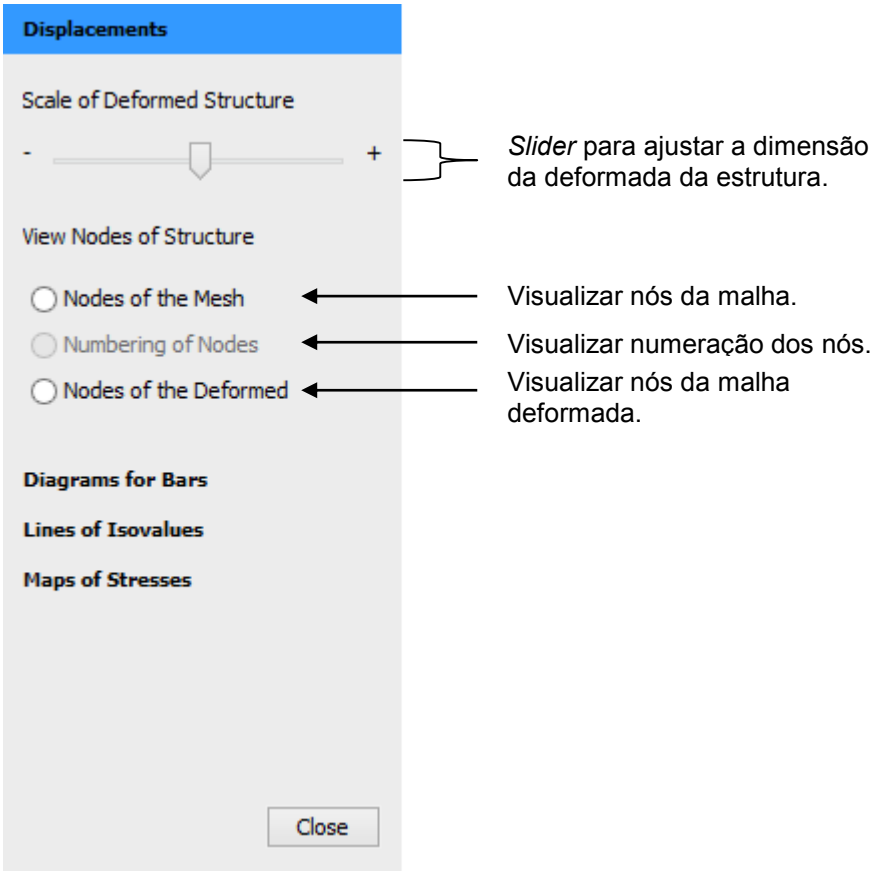


Fig.9.11 – Conteúdo do painel lateral Displacements.

Além disto, para os modelos de barras é possível visualizar os diagramas de esforços e para os modelos bidimensionais as linhas de isovalores, os mapas de tensões e as tensões e direções principais. Relativamente à visualização das linhas de isovalores e dos mapas de tensões, o utilizador pode escolher qual a tensão que pretende ver representada.

9.6. OUTRAS FUNCIONALIDADES

A descrição anteriormente apresentada do programa salientou as ferramentas relativas à modelação por elementos finitos e às funcionalidades de visualização de resultados. Existem outros elementos na interface gráfica do programa que são úteis ao utilizador.

O programa apresenta, por exemplo, a informação relativa ao posicionamento do rato e dicas sobre o funcionamento das ferramentas selecionadas nos separadores. Estas dicas são apresentadas sempre que o utilizador seleciona determinada ferramenta do programa e têm como objetivo informá-lo sofre o modo de funcionamento desta. Estes elementos estão dispostos no painel inferior da interface do programa.

Outras funcionalidades do programa estão concentradas nos menus File, Edit e Help. A funcionalidade que mais se destaca é relativa à persistência de dados. De modo a que os utilizadores possam guardar os projetos criados e posteriormente recuperá-los, o programa permite a gravação dos projetos na memória permanente do computador. A informação gravada corresponde a todos os elementos desenhados, valores das diversas propriedades dos elementos finitos e carregamentos.

Por fim, analisando globalmente o programa, este está focado apenas na modelação e análise estrutural por elementos finitos. A componente de dimensionamento não é abordada. O desenvolvimento de ferramentas orientadas ao projeto deve fazer parte dos desenvolvimentos futuros. A importância que os meios informáticos têm no contexto profissional não deve ser ignorada pelo meio académico. Pretende-se com isto salientar a importância de os alunos serem capazes de desenvolver as suas próprias ferramentas, pois, elas são um meio para o aumento da competitividade e redução de custos.

10
CONCLUSÃO

10.1. CONSIDERAÇÕES FINAIS

Esta dissertação teve como objetivo central o desenvolvimento de um programa que permitisse a modelação e a análise estrutural pelo Método dos Elementos Finitos. Assim, ao longo dos capítulos foram abordados os elementos necessários à implementação computacional do método e à construção da interface gráfica do programa.

O programa desenvolvido permite modelar problemas simples com diversas formulações de elementos finitos. Contudo, nem todas as formulações de elementos finitos foram abordadas. Os modelos de sistemas discretos disponíveis permitem o estudo de estruturas articuladas, estruturas reticuladas, vigas e grelhas. Relativamente aos meios contínuos, o utilizador pode escolher entre um estado plano de tensão ou deformação e o estudo de lajes pelas duas teorias abordadas nesta dissertação. Existem grandes melhorias a implementar na análise de meios discretos e meios contínuos. A extensão a contextos dinâmicos e a realização de análises não lineares é fundamental. A resolução deste tipo de problemas também privilegia a sua implementação computacional porque requerem uma grande quantidade de cálculos.

Escrever e validar o código fonte de um programa com quarenta e cinco mil linhas foi um processo árduo e complexo. Além da dificuldade de codificação e o estudo das diferentes formulações e teorias de elementos finitos acresceu-se ainda a necessidade de aprender a linguagem de programação Java. De modo a facilitar a sua aceitação, a construção do programa obedeceu a um conjunto de critérios de modo se obter uma solução versátil e apelativa. Isto exigiu a programação de diferentes modelos e teorias de elementos finitos e a construção de uma interface gráfica moderna. Uma das principais vantagens do programa é a sua portabilidade. Assim, o programa funciona em diferentes dispositivos informáticos desde que os utilizadores tenham instalada a plataforma Java.

O desenvolvimento de ferramentas informáticas que otimizem os tempos de cálculo e aumentem a capacidade de modelação são e devem continuar a ser motivo de grande foco no contexto da Engenharia de Estruturas. É indispensável o uso destas ferramentas em ambiente de projeto. Devido à sua enorme importância, o seu desenvolvimento deve também ser alvo de atenção pela comunidade académica. É fundamental impulsionar o desenvolvimento de novas soluções que aumentem a capacidade de simulação numérica. A importância do uso destas ferramentas é conhecida pela comunidade académica e profissional mas tem sido pouco impulsionada pelos membros que a compõem. O seu desenvolvimento deverá ser impulsionado por aqueles que a elas recorrem. Em nada contribui para o seu desenvolvimento esperar que outras áreas venham colmatar a falta de investimento daqueles que delas necessitam. É importante que este trabalho comece logo na fase de formação de modo a consciencializar os futuros profissionais da sua importância.

A simulação computacional é um meio impulsionador no desenvolvimento futuro da Engenharia de Estruturas. A possibilidade de se poder experimentar diferentes soluções sem os custos e inconvenientes de se recorrer a modelos físicos é uma das muitas possibilidades que o desenvolvimento e a programação de métodos de análise tem vindo a proporcionar.

A solução desenvolvida nesta dissertação vem colmatar algumas das falhas evidenciadas. Contudo, o programa construído constitui uma solução simples, precisando de ser melhorada e expandida. Devido a ter sido desenvolvido muito do trabalho de base para a expansão, esta tarefa fica bastante simplificada para os desenvolvimentos futuros. Por fim, salienta-se que as possibilidades de desenvolvimento de novos métodos e ferramentas são infinitas.

10.2. SUGESTÕES PARA DESENVOLVIMENTOS FUTUROS

A inovação está na base do sucesso de qualquer setor. Esta dissertação pretendeu contribuir para o desenvolvimento de novas soluções que ajudem à aprendizagem e à divulgação do Método dos Elementos Finitos.

Sendo o programa desenvolvido limitado, é importante referir algumas áreas por onde começar a expandi-lo. Como já anteriormente referido, o programa foca apenas a modelação e a análise de estruturas. O dimensionamento automático é talvez dos assuntos mais relevantes pelo qual se pode começar a expandir a versão do programa desenvolvido no âmbito desta dissertação. É um tema bastante desafiante e perfeitamente executável que permitiria ao programa abranger todas as fases do projeto de estruturas.

Não faltarão certamente aos interessados ideias para expandir a versão do programa desenvolvido ou ideias para construir outras soluções. O aparecimento de novas soluções exige a motivação necessária para as procurar desenvolver. Assim, que este trabalho seja um impulsionador da procura e desenvolvimento de novas soluções.

BIBLIOGRAFIA

[1] Arnold, K., Gosling, J., Holmes, D. *A linguagem de programação Java*. Artmed Editora S.A., Porto Alegre, 2006.

[2] Azevedo, A. *Método dos Elementos Finitos*. Faculdade de Engenharia da Universidade do Porto, Porto, 2003.

[3] Barros, R. *Introdução ao Método dos Elementos Finitos*. Apontamentos da disciplina de Teoria das Estruturas 2, Faculdade de Engenharia da Universidade do Porto, Porto, 2006.

[4] Branco, C. *Mecânica dos Materiais*. Fundação Calouste Gulbenkian, Lisboa, 2011.

[5] Castro, L. *Elementos Finitos para a Análise Elástica de Lajes*. Apontamentos da disciplina de Análise de Estruturas II, Instituto Superior Técnico, Lisboa, 2007.

[6] Cook, R., Malkus, D., Plesha, M. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons., Madison, 1989.

[7] Delgado, R. *Método dos Elementos Finitos*. Texto de apoio às aulas de Método dos Elementos Finitos, Faculdade de Engenharia da Universidade do Porto, Porto, 1990.

[8] Eck, D. *Introduction to Programming Using Java*. Department of Mathematics and Computer Science, Hobart and William Smith Colleges, Estados Unidos da América, 2011.

[9] Ferreira, A. *Problemas de Elementos Finitos em MATLAB*. Fundação Calouste Gulbenkian, Lisboa, 2010.

[10] Gonçalves, M. *Geração de Malhas Bidimensionais de Elementos Finitos Baseada em Mapeamentos Transfinitos*. Dissertação de Mestrado, Universidade Federal de Minas Gerais, 2004.

[11] Kassimali, A. *Matrix Analysis of Structures*. Cengage Learning, Carbondale, 2012.

[12] Moreira, R. *Sistema Gráfico Interativo para Ensino de Análise Estrutural Através do Método dos Elementos Finitos*. Dissertação de Mestrado, Universidade Federal de Minas Gerais, 2006.

[13] Oñate, E. *Structural Analysis with the Finite Element Method. Linear Statics. Volume 1. Basis and Solids*. Springer, Barcelona, 2009.

[14] Oñate, E. *Structural Analysis with the Finite Element Method. Linear Statics. Volume 2. Beams, Plates and Shells*. Springer, Barcelona, 2013.

[15] Pressman, R. *Software Engineering. A Practitioner's Approach*. McGraw-Hill, Estados Unidos da América, 2010.

[16] Radeş, M. *Finite Element Analysis*. Editura Printech, Bucareste, 2006.

[17] Saliba, S. *Implementação Computacional e Análise Crítica de Elementos Finitos de Placas*. Dissertação de Mestrado, Universidade Federal de Minas Gerais, 2007.

[18] Sanches, P. *Elementos finitos triangulares compatíveis na análise estrutural de lajes finas*. Dissertação de Mestrado, Instituto Superior Técnico, 2011.

[19] Sebesta, R. *Concepts of programming languages*. Pearson, Colorado, 2012.

[20] Soriano, H. *Método de Elementos Finitos em Análise de Estruturas*. Editora da Universidade de São Paulo, São Paulo, 2003.

[21] Teixeira-Dias, F., Pinho-da-Cruz, J., Valente, R., Sousa, R. *Método dos Elementos Finitos. Técnicas de Simulação Numérica em Engenharia*. ETEP, Aveiro, 2010.

[22] Vlissides, J., Helm, R., Johnson, R., Gamma, E. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Estados Unidos da América, 1994.

[23] Zienkiewicz, O.C., Taylor, R.L., Zhu, J.Z. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, Reino Unido, 2013.

[24] http://www.netbeans.org/. Consultado em 01/10/2014.

ANEXOS

A. PACOTES E CLASSES DO PROJETO DO FEM FOR STUDENTS

A.1. PACOTES DO PROJETO

O projeto do programa FEM for Students encontra-se organizado por pacotes em que cada pacote contém um conjunto de classes que definem o nome do pacote. A tabela A.1 mostra os pacotes do projeto e o número de classes de cada pacote.

Tabela A.1 – Pacotes e número de classes em cada pacote.

Pacotes do projeto	*Número de classes*
backend	20
calculations	6
finiteelement	27
frontend	15
gausslegendre	8
gausslobatto	8
images	0
matrices	6
shapefunctions	8
variablesubstitution	16

A classe que contém o método main() é designada por FEMforStudents e não se encontra dentro de nenhum pacote.

A.2. CLASSES DE CADA PACOTE

Nas tabelas seguintes apresentam-se as classes de cada pacote e o número de linhas de cada uma. Cada uma das tabelas apresenta somente as classes públicas contidas no respetivo pacote.

Tabela A.2 – Descrição das classes do pacote backend.

Classes	*Número de linhas*
Command	272
DiagramsForBars	833
DrawEllipse	162
DrawLine	256
DrawLoads	902
DrawNode	136
DrawPolygon	276
DrawRectangle	105

DrawSupports	405
DrawingMethods	1366
DrawingPanel	2894
Geometry	136
IsolinesAndMaps	903
LoadTable	261
Loads	172
OpenSave	300
PrincipalStresses	142
ResultsPane	1248
ResultsTables	708
VerticesCoordinates	329

Tabela A.3 – Descrição das classes do pacote calculations.

Classes	*Número de linhas*
AnalyticGeometry	500
FiniteElement	1193
FormatResults	885
NodalResults	828
NodesCoordinates	214
Processor	1453

Tabela A.4 – Descrição das classes do pacote finiteelement.

Classes	*Número de linhas*
MatrixB	178
MatrixB_1D	59
MatrixB_2D	152
MatrixB_3D	107
MatrixB_Beams	187
MatrixB_Frames	53
MatrixB_Grids	53
MatrixB_Slabs	513

MatrixD	158
MatrixJ	235
MatrixK	348
MatrixK_1D	77
MatrixK_2D	938
MatrixK_3D	805
MatrixK_Beams	498
MatrixK_Frames	82
MatrixK_Grids	82
MatrixK_Slabs	3731
MatrixT	67
VectorF	341
VectorF_1D	52
VectorF_2D	267
VectorF_3D	100
VectorF_Beams	111
VectorF_Frames	47
VectorF_Grids	45
VectorF_Slabs	224

Tabela A.5 – Descrição das classes do pacote frontend.

Classes	*Número de linhas*
IndividualProperties	170
JLateralPanelAnalysis	689
JLateralPanelGeometry	1508
JLateralPanelLoads	1446
JLateralPanelResults	1001
JPanelBottom	100
JPanelHelp	150
JPanelWelcome	585
LegendsForPanels	654
MatrixTableResults	66

NodalForcesTable	96
NodalStressesTable	464
ReactionsSupportTable	94
UserInterface	8842
VectorTableResults	88

Tabela A.6 – Descrição das classes do pacote gausslegendre.

Classes	*Número de linhas*
GaussLegendre	141
WeightsCoordinates_1D	77
WeightsCoordinates_2D	282
WeightsCoordinates_3D	114
WeightsCoordinates_Beams	139
WeightsCoordinates_Frames	77
WeightsCoordinates_Grids	77
WeightsCoordinates_Slabs	311

Tabela A.7 – Descrição das classes do pacote gausslobatto.

Classes	*Número de linhas*
GaussLobatto	141
WeightsCoordinates_1D	77
WeightsCoordinates_2D	282
WeightsCoordinates_3D	114
WeightsCoordinates_Beams	139
WeightsCoordinates_Frames	77
WeightsCoordinates_Grids	77
WeightsCoordinates_Slabs	309

Tabela A.8 – Descrição das classes do pacote matrices.

Classes	*Número de linhas*
Determinant	104
Inverse	120

Multiply	220
Subtract	117
Sum	117
Transpose	102

Tabela A.9 – Descrição das classes do pacote shapefunctions.

Classes	*Número de linhas*
MatrixNv	122
MatrixNv_1D	59
MatrixNv_2D	84
MatrixNv_3D	59
MatrixNv_Beams	124
MatrixNv_Frames	71
MatrixNv_Grids	71
MatrixNv_Slabs	208

Tabela A.10 – Descrição das classes do pacote variablesubstitution.

Classes	*Número de linhas*
DerivedMatrix	81
DerivedMatrix_1D	63
DerivedMatrix_2D	167
DerivedMatrix_3D	107
DerivedMatrix_Beams	186
DerivedMatrix_Frames	39
DerivedMatrix_Grids	39
DerivedMatrix_Slabs	155
MatrixB	89
MatrixB_1D	54
MatrixB_2D	149
MatrixB_3D	103
MatrixB_Beams	155
MatrixB_Frames	38

MatrixB_Grids	38
MatrixB_Slabs	153

B. MODELAÇÃO E ANÁLISE DE UM PÓRTICO

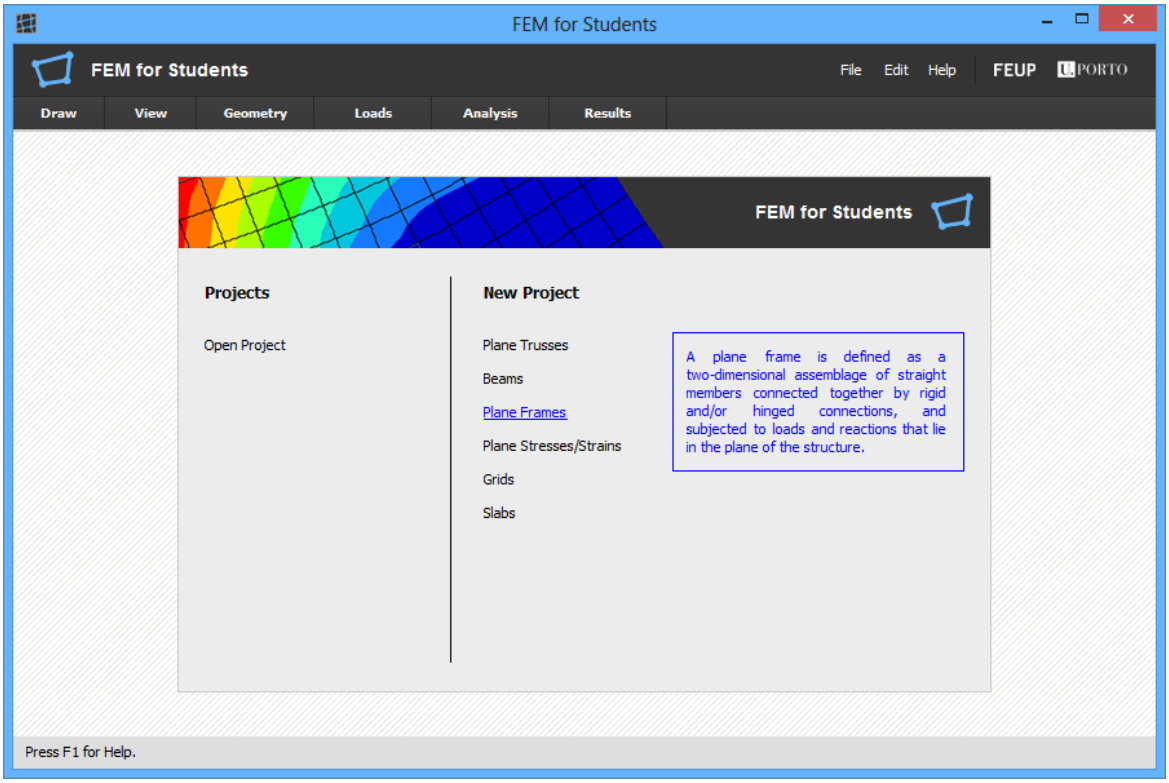


Fig.B.1 – Visualização do painel inicial do programa.

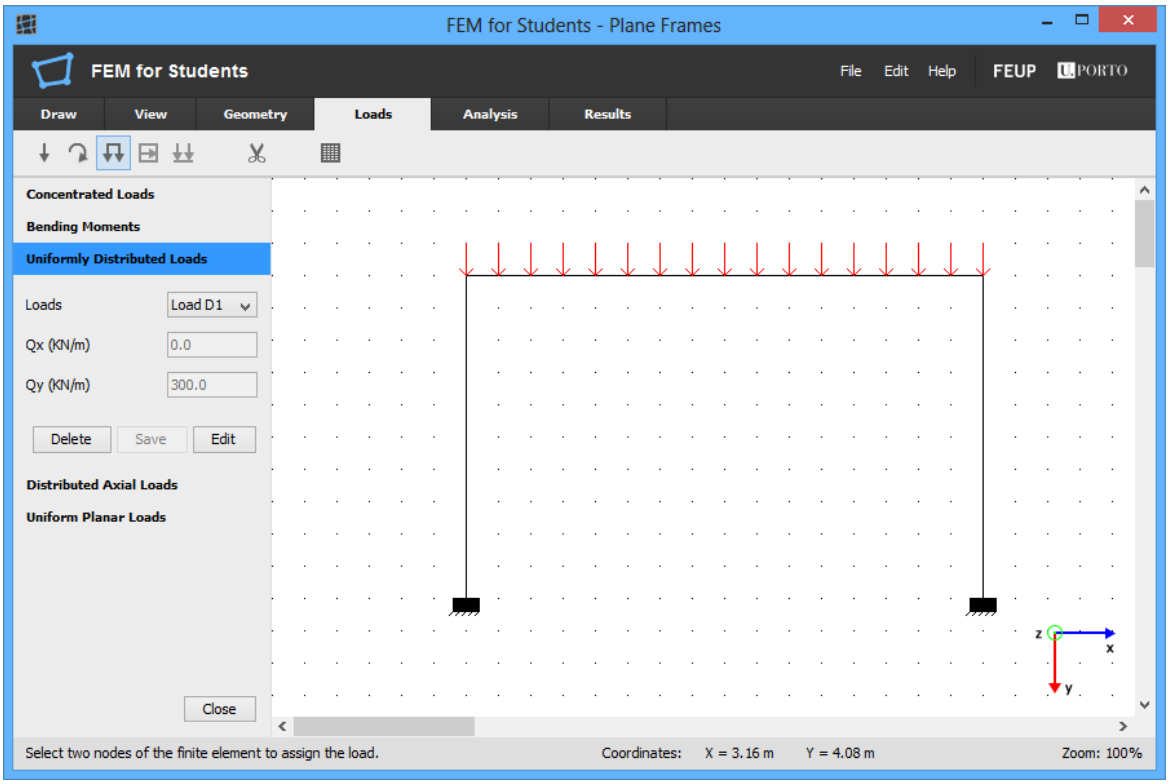
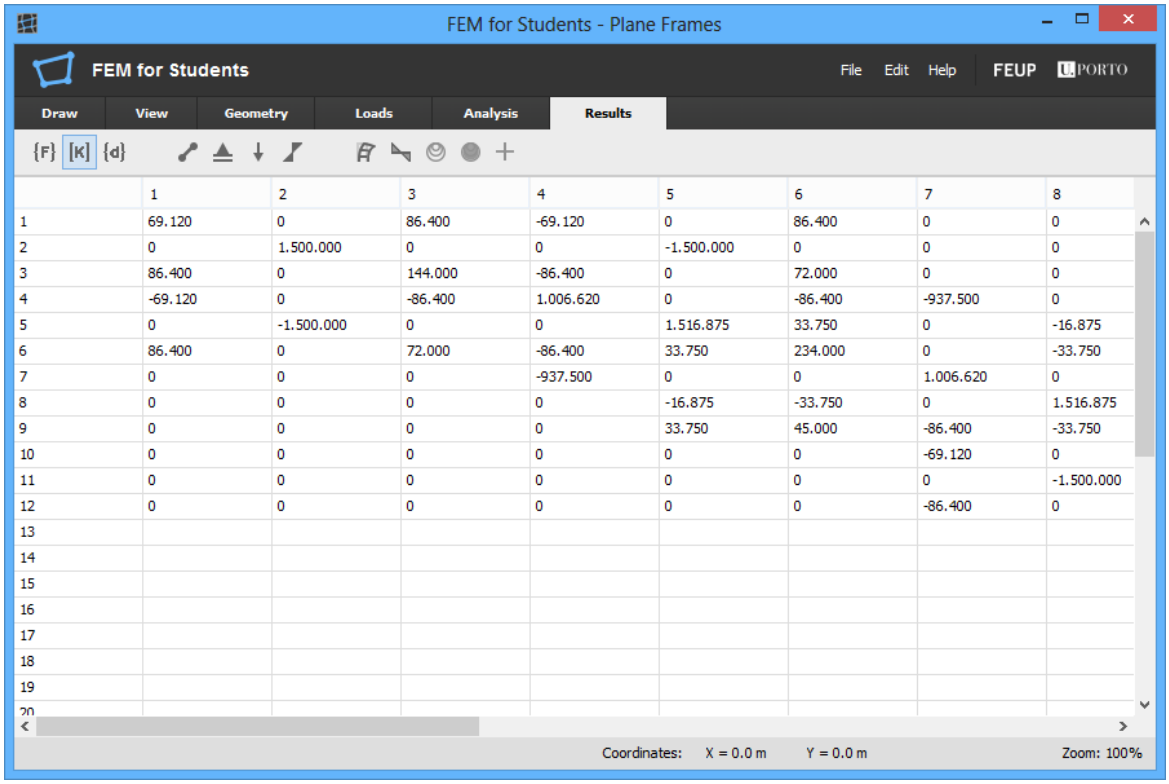


Fig.B.2 – Representação do pórtico em análise.



	1	2	3	4	5	6	7	8
1	69.120	0	86.400	-69.120	0	86.400	0	0
2	0	1.500.000	0	0	-1.500.000	0	0	0
3	86.400	0	144.000	-86.400	0	72.000	0	0
4	-69.120	0	-86.400	1.006.620	0	-86.400	-937.500	0
5	0	-1.500.000	0	0	1.516.875	33.750	0	-16.875
6	86.400	0	72.000	-86.400	33.750	234.000	0	-33.750
7	0	0	0	-937.500	0	0	1.006.620	0
8	0	0	0	0	-16.875	-33.750	0	1.516.875
9	0	0	0	0	33.750	45.000	-86.400	-33.750
10	0	0	0	0	0	0	-69.120	0
11	0	0	0	0	0	0	0	-1.500.000
12	0	0	0	0	0	0	-86.400	0
13								
14								
15								
16								
17								
18								
19								
20								

Fig.B.3 – Visualização da matriz de rigidez da estrutura.

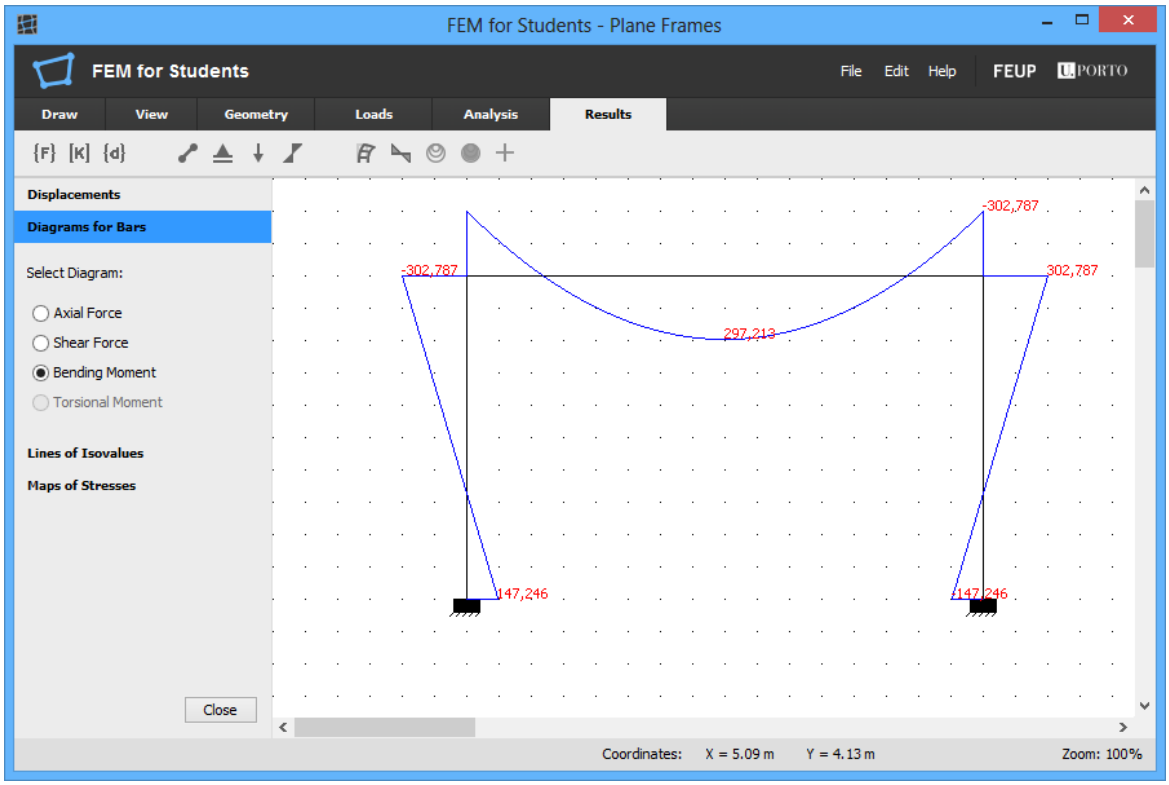


Fig.B.4 – Visualização do diagrama de momentos fletores.

C. MODELAÇÃO E ANÁLISE DE UMA PAREDE

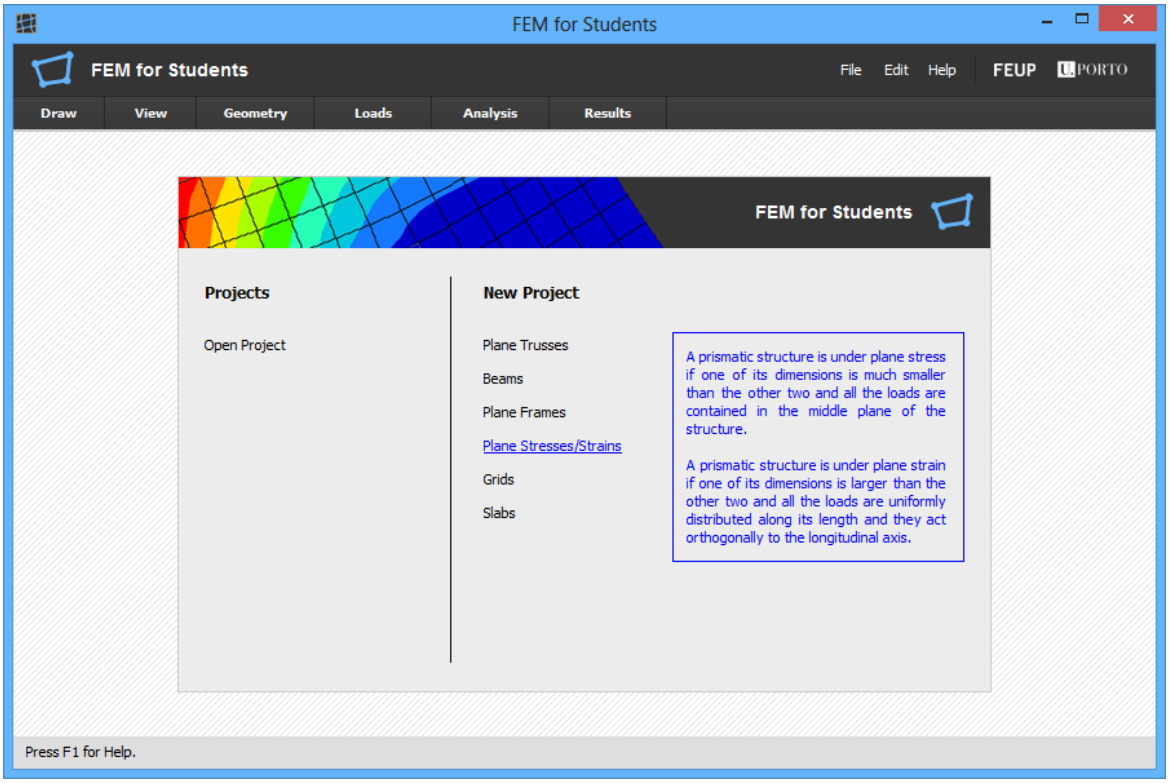


Fig.C.1 – Visualização do painel inicial do programa.

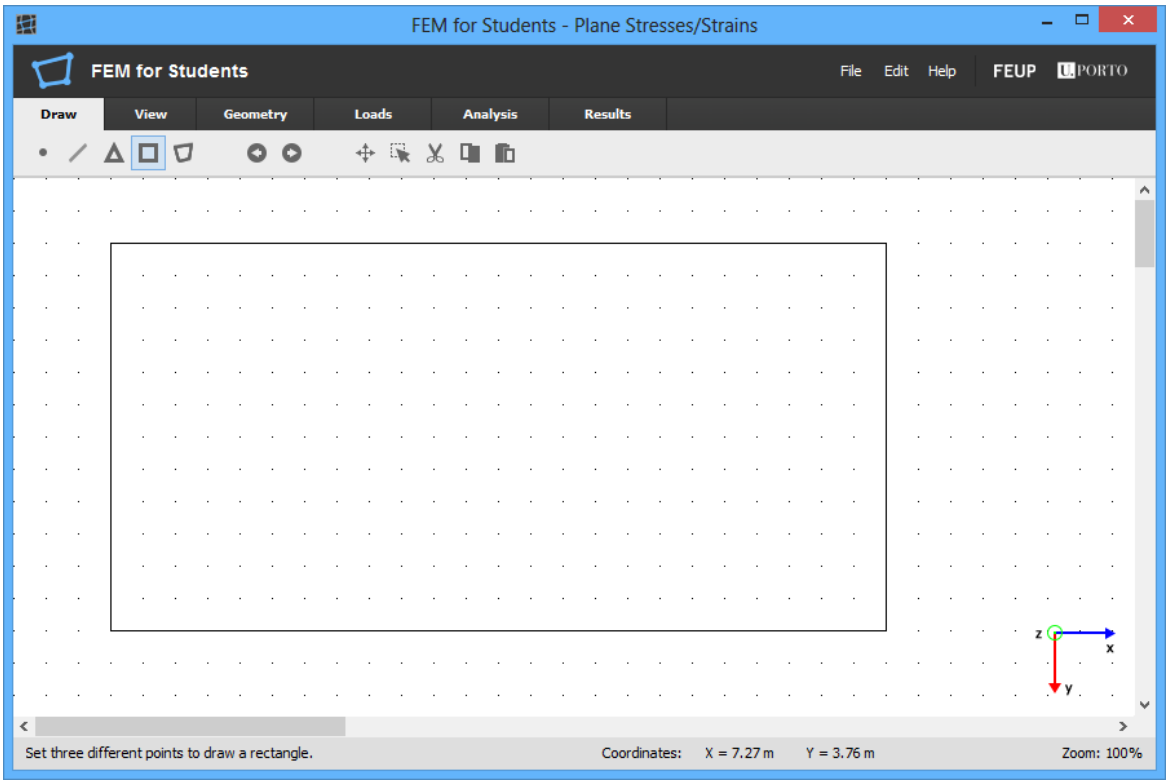


Fig.C.2 – Desenho de um retângulo para representar a parede.

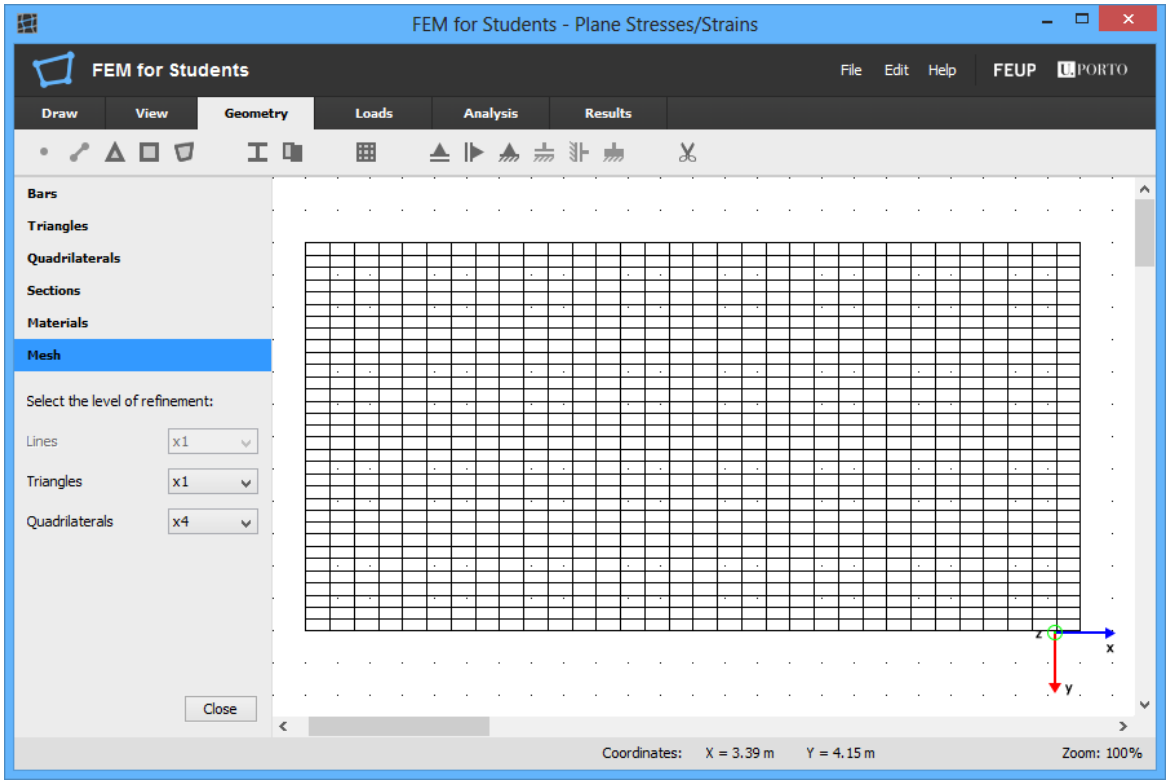


Fig.C.3 – Refinamento da malha de elementos finitos.

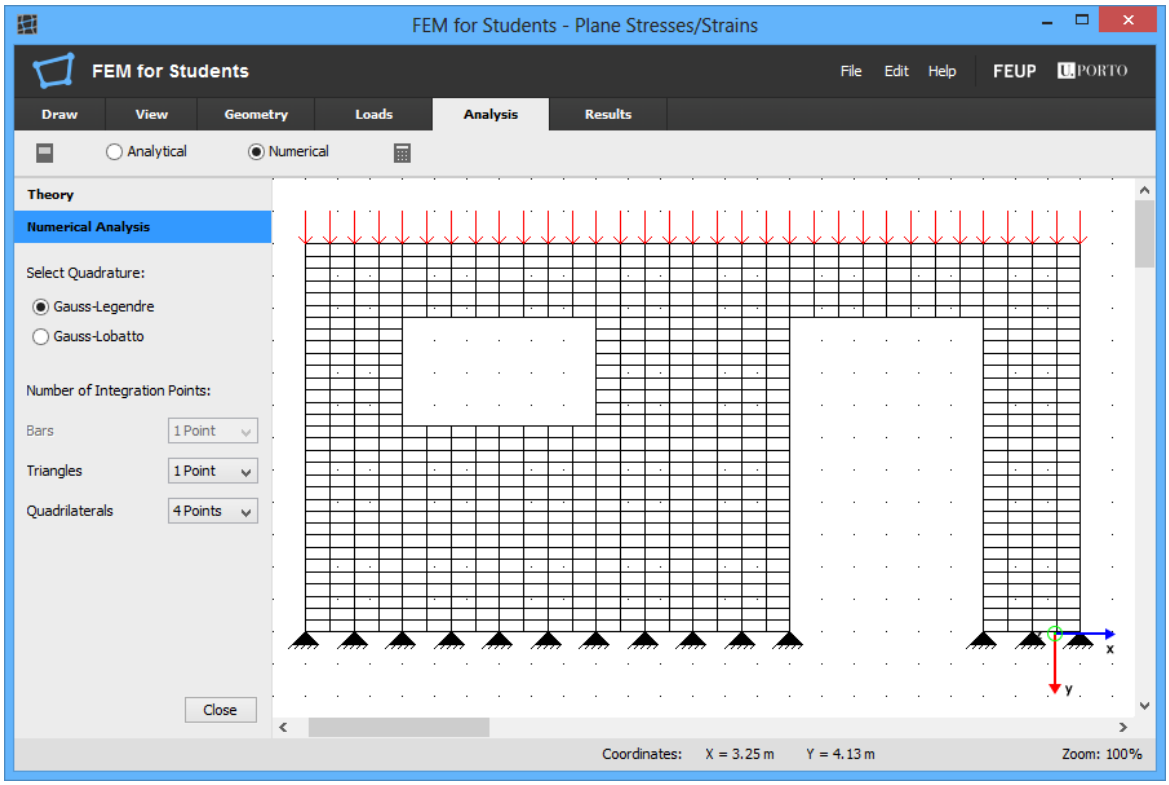


Fig.C.4 – Escolha da quadratura e do número de pontos de integração.

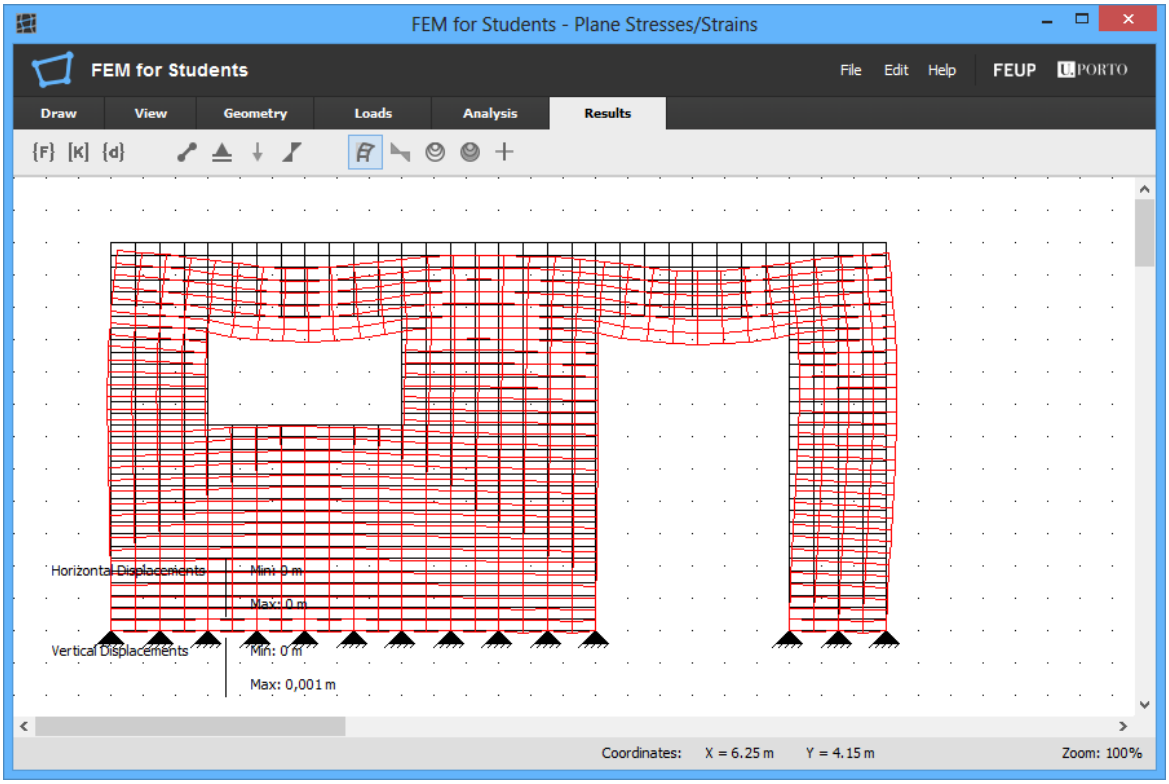


Fig.C.5 – Visualização da deformada da parede.

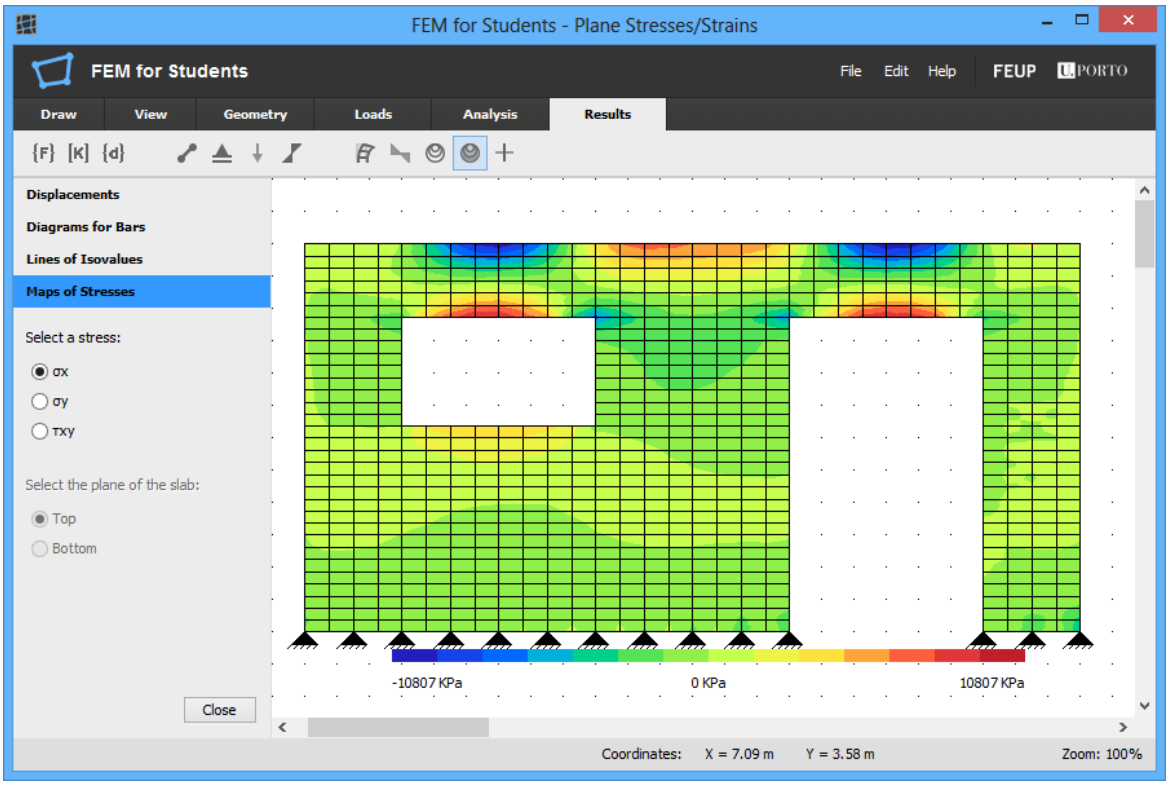


Fig.C.6 – Visualização do mapa de tensões na parede.