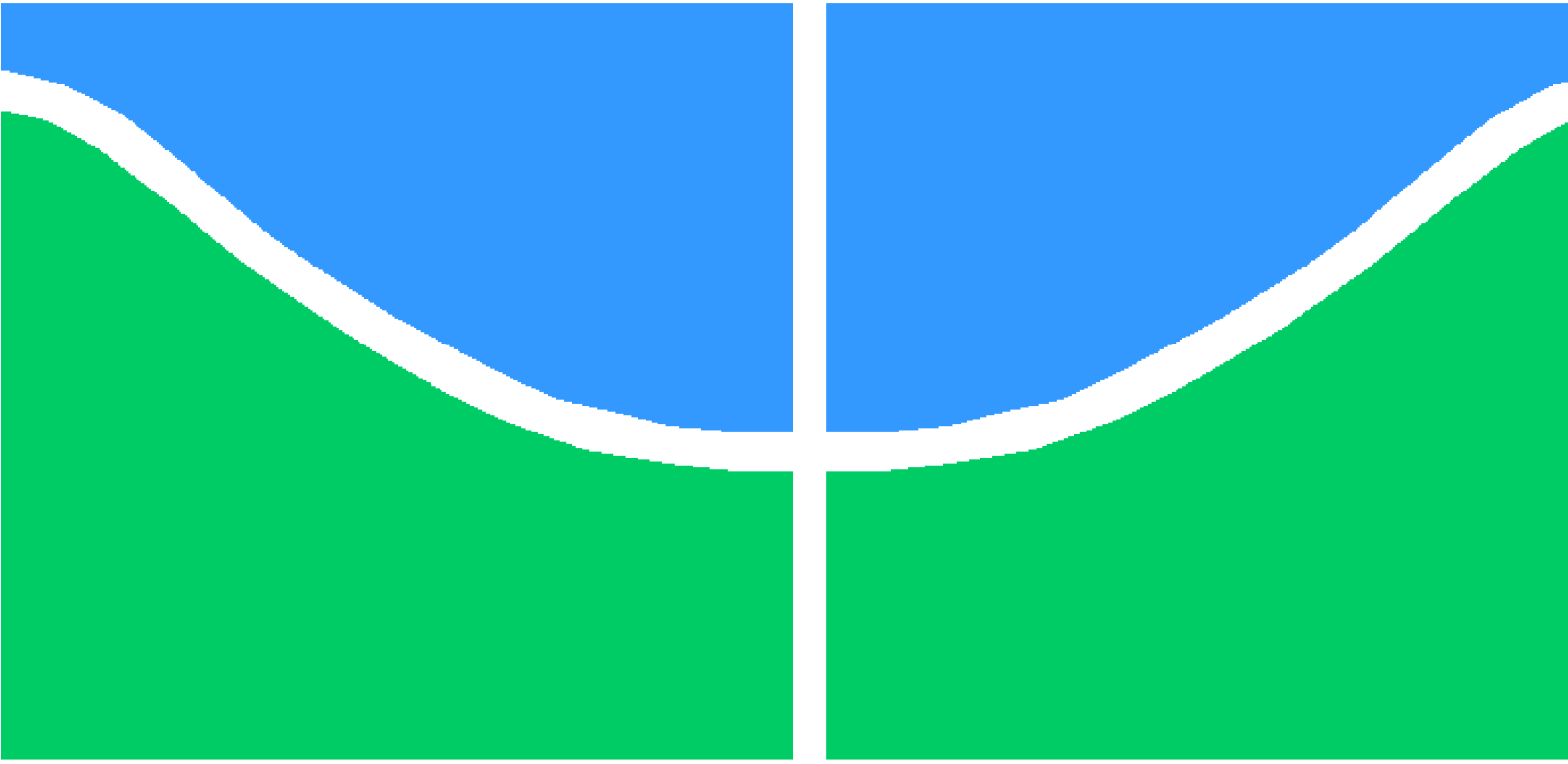


PROJETO DE GRADUAÇÃO

Análise de Materiais compósitos usando a Biblioteca deal.ii

Por,

Erik Silva Fujiyama

Brasília, 5 de dezembro de 2019

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECÂNICA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

PROJETO DE GRADUAÇÃO

Análise de Materiais compósitos usando a Biblioteca deal.ii

Por,

Erik Silva Fujiyama

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Mecânico

Banca Examinadora

Eder Lima de Albuquerque (Orientador) _____

Prof. Edgar Nobuo Mamiya _____

MsC. Gustavo Silva Vaz Gontijo _____

Brasília 5 de dezembro de 2019

Agradecimentos

Gostaria de agradecer primeiramente a meus pais, por todo o apoio e incentivo que me propiciaram.

Em seguida, agradeço ao Prof. Éder Lima de Albuquerque, por toda a atenção, paciência e ajuda ao longo deste trabalho.

Agradeço também aos meus amigos, de forma geral, por todos os momentos agradáveis e não tão agradáveis que passamos juntos.

Além disso, sinto-me obrigado a agradecer a todos aqueles, anônimos ou não, que se esforçam para ajudar os outros a ter acesso a informação sem esperar nenhum retorno.

.

.

Resumo

Este trabalho tem como objetivo fazer uma análise de problemas de elasticidade plana anisotrópica usando a biblioteca de elementos finitos deal.II. Primeiramente, fez-se uma revisão teórica sobre a teoria da elasticidade linear anisotrópica e sobre o método dos elementos finitos. A teoria de elasticidade anisotrópica foi voltada para os materiais compósitos laminados. Foi mostrada em detalhes a obtenção de matrizes de rigidez para este tipo de material e o cálculo de deslocamentos, tensões e deformações. Na análise de elementos finitos a formulação apresentada se restringiu a problemas bi-dimensionais em estado plano de tensão ou deformação consdierando materiais isotrópicos e anisotrópicos.

Palavras-chaves: Método dos Elementos Finitos. deal.II. Materiais Compósitos. Elasticidade Linear. Anisotropia.

Abstract

The objective of this undergraduation project is to analyse anisotropic elasticity problems with the finite element library deal.II. First, a theoretical review is done about anisotropic linear elasticity and the finite elements method is carried out. The anisotropic elasticity has laminated composite materials as focus. In the finite element analysis, the presented approach is restricted to bi-dimensional plane stress or plain strain problems, in isotropic and anisotropic materials.

Key-words: Finite Element Method. deal.II. Composite Materials. Linear Elasticity. Anisotropy

Lista de Figuras

Figura 1 – Utilização de Materiais compósitos nos EUA em 2004 (AGARWAL; BROUTMAN; CHANDRASHEKHARA, 2006) 2
Figura 2 – Principais classes do deal.II para simulações numéricas (BANGERTH; HARTMANN; KANSCHAT, 2007) 3
Figura 3 – Publicações conhecidas que usam o deal.II. Fonte: deal.org 4
Figura 4 – Tensões em um elemento cúbico 6
Figura 5 – Lâmina especialmente Ortotrópica (AGARWAL; BROUTMAN; CHANDRASHEKHARA, 2006) 8
Figura 6 – Variação de tensão e deformação em um laminado hipotético de 3 lâminas 9
Figura 7 – Mapeamento pelo domínio bi-unit para funções bilineares (FISH; BELYTSCHKO, 2007) com alterações 15
Figura 8 – Mapeamento pelo domínio bi-unit para funções quadráticas (FISH; BELYTSCHKO, 2007) com alterações 15
Figura 9 – Nós de um elemento 16
Figura 10 – Função de forma bilinear 17
Figura 11 – Função de forma quadrática 18
Figura 12 – Placa com furo circular no centro submetida a tensões simétricas nas laterais. 25
Figura 13 – Domínio discretizado 25
Figura 14 – Deslocamentos em X_1 26
Figura 15 – Deslocamentos em X_2 26
Figura 16 – Resultados Numéricos comparados com os analíticos 27
Figura 17 – Malha refinada 28
Figura 18 – Resultados numéricos $\times$ analíticos em malha mais refinada 28
Figura 19 – σ_1 29
Figura 20 – σ_2 29
Figura 21 – σ_{12} 30
Figura 22 – Comparação analítca numérica 31
Figura 23 – Placa quadrada com furo circular de material laminado simétrico . . . 33
Figura 24 – Malha para placa quadrada com furo circular 33

Figura 25 – Deslocamentos em X_1 para placa quadrada com furo circular 34
Figura 26 – Deslocamentos em X_2 para placa quadrada com furo circular 34
Figura 27 – Tensões σ_1 lâmina 0 35
Figura 28 – Tensões σ_2 lâmina 0 35
Figura 29 – Tensões σ_{12} na lâmina 0 36
Figura 30 – Tensões σ_1 na lâmina 90 36
Figura 31 – Tensões σ_2 na lâmina 90 37
Figura 32 – Tensões σ_{12} na lâmina 90 37
Figura 33 – Geometria e tensões aplicadas 38
Figura 34 – Malha utilizada no caso 2 39
Figura 35 – Deslocamentos em X_1 do caso 2 39
Figura 36 – Deslocamentos em Y_2 do caso 2 39
Figura 37 – Tensões σ_1 na lâmina de 20 40
Figura 38 – Tensões σ_2 na lâmina de 50 40
Figura 39 – Tensões σ_{12} na lâmina de 20 41
Figura 40 – Tensões σ_1 na lâmina de 50 41
Figura 41 – Tensões σ_2 na lâmina de 50 42
Figura 42 – Tensões σ_{12} na lâmina de 50 42

Lista de Tabelas

Tabela 1 – Propriedades de Materiais compósitos em comparação com materiais convencionais (fonte: (AGARWAL; BROUTMAN; CHANDRASHEKHARA, 2006)) 1
Tabela 2 – Notação de Voigt 6
Tabela 3 – Parâmetros para placa quadrada com furo circular 27
Tabela 4 – Parâmetros para placa quadrada com furo circular 28
Tabela 5 – Erros relativos a densidade de energia de deformação com funções de forma de 2ª ordem 30
Tabela 6 – Erros relativos a densidade de energia de deformação com funções de forma de 1ª ordem 30
Tabela 7 – Parâmetros de erro do cilindro 31
Tabela 8 – Propriedades da lâmina 31
Tabela 9 – Resultados para laminado [0°/90°/90°/0°] 32
Tabela 10 – Resultados para laminado [45°/135°/135°/45°] 32
Tabela 11 – Propriedades do material 32
Tabela 12 – Diferença RMS e deformação média 34
Tabela 13 – Propriedades mecânicas do material 38
Tabela 14 – Comparação de resultados 40

Lista de símbolos

Símbolos Latinos

C	Tensor Rigidez
E	Módulo de elasticidade
F	Forças de Campo
G	Módulo de Cisalhamento
N	Tensão integrada ao longo da espessura
S	Espaço de dimensão infinita onde se encontra a solução
S	Matriz de flexibilidade
S^h	Espaço de dimensão finita onde se encontra a solução aproximada
V	Espaço de dimensão infinita onde se encontram as funções peso
V^h	Espaço de dimensão finita onde se encontram as funções peso
c	Peso em determinado nó
d	Deslocamento em determinado nó
u	Deslocamento
w	Função peso

Símbolos Gregos

Γ	Contorno do domínio
ε	Tensor de deformações
ν	Coeficiente de Poisson

ξ	Variável do domínio bi-unitário
ρ	Densidade
σ	Tensor de Tensões
Ω	Domínio

Outros

$\varnothing$	Função de base

Sumário

1 INTRODUÇÃO 1

1.1 Motivação 1

1.1.1 Materiais Compósitos 1

1.1.2 O Método dos Elementos Finitos 2

1.2 Objetivos 4

1.3 Contribuição 4

2 TEORIA DE ELASTICIDADE ANISOTRÓPICA 5

2.1 Introdução 5

2.2 Equações de Elasticidade 5

2.3 Laminados Ortotrópicos 7

2.3.1 Introdução 7

2.3.2 Lâmina Ortotrópica 7

2.3.3 Materiais laminados em estado plano de tensão 9

3 ELEMENTOS FINITOS 11

3.1 Introdução 11

3.2 Condições de Contorno 11

3.3 Forma Forte e Forma Fraca do Problema 12

3.4 Forma de Dimensão Finita e Funções de Forma 13

3.4.1 Forma de dimensão finita 13

3.4.2 Funções de Base 14

3.4.3 Forma Matricial 19

4 IMPLEMENTAÇÃO NO DEAL.II 21

5 RESULTADOS 24

5.1 Introdução 24

5.2 Quarto de placa com Condições de Contorno de Neumann e Dirichlet 24

5.3 Cilindro Pressurizado 30

5.4 Placa simples de material compósito 31

5.5 Placa quadrada com furo circular 32
5.6 Placa com furo circular 38

6 CONCLUSÕES 43
6.1 Trabalhos futuros 44

REFERÊNCIAS 45

7 ANEXOS 46

ANEXOS 47
.1 Anexo 1 48

1 INTRODUÇÃO

1.1 Motivação

1.1.1 Materiais Compósitos

De acordo com Agarwal, Broutman e Chandrashekhara (2006) um material é considerado como compósito quando é constituído por 2 ou mais constituintes, e apresenta diferenças significativas entre as propriedades do todo e das partes que o formam. Eles são compostos de uma ou mais fases descontínuas, denominadas reforço, embutidas em uma fase contínua, denominada matriz. As propriedades do material são influenciadas pelas propriedades dos materiais que o constituem, sua interação e sua distribuição.

É comum o reforço estar na forma de fibras. Geralmente, estas são mais fortes e mais rígidas. Portanto, uma razão volumétrica composta por um porcentual maior de fibras no material tende a garantir maior resistência a carga, pois grande parte desta se transfere às fibras.

Materiais compósitos são normalmente fabricados para melhorar as propriedades mecânicas, como rigidez, tenacidade e dureza. Na Tab. 1 vê-se uma comparação de propriedades mecânicas de materiais compósitos com materiais convencionais.

É notável que a resistência e o módulo de elasticidade específicos dos materiais compósitos são significativamente maiores que os demais materiais.

A importância de materiais compósitos vem crescendo de maneira significativa

Tabela 1 – Propriedades de Materiais compósitos em comparação com materiais convencionais (fonte: (AGARWAL; BROUTMAN; CHANDRASHEKHARA, 2006))

Material	Fração Volumétrica de Fibra (V_f)(%)	Módulo de Elasticidade (E) (GPa)	Tensão de Escoamento (σ_u) (GPa)	Densidade (ρ) (g/cm^3)	Módulo Específico (E/rho)	Tensão Específica (σ_u/ρ)
Aço-carbono		210	0,45-0,83	7,8	26,9	0,058-0,106
Alumínio 2024-T4 6061-T6		 73 69	 0,41 0,26	 2,7 2,7	 27,0 25,5	 0,152 0,096
E-glass-epoxy	57	21,5	0,57	1,97	10,9	0,26
Kevlar 49-epoxy	60	40	0,65	1,40	29,0	0,46
Fibra de carbono epoxy	58	83	0,38	1,54	53,5	0,24
Boro-epoxy	60	106	0,38	2,0	53,0	0,19

nos últimos anos. Segundo Agarwal, Broutman e Chandrashekhara (2006), a indústria de materiais compósitos dos Estados Unidos da América cresceu em uma média de 6,5% ao ano desde 1960, o que é o dobro do crescimento da economia norte-americana. A utilização de compósitos cresceu em 16 vezes entre 1960 e 2004.

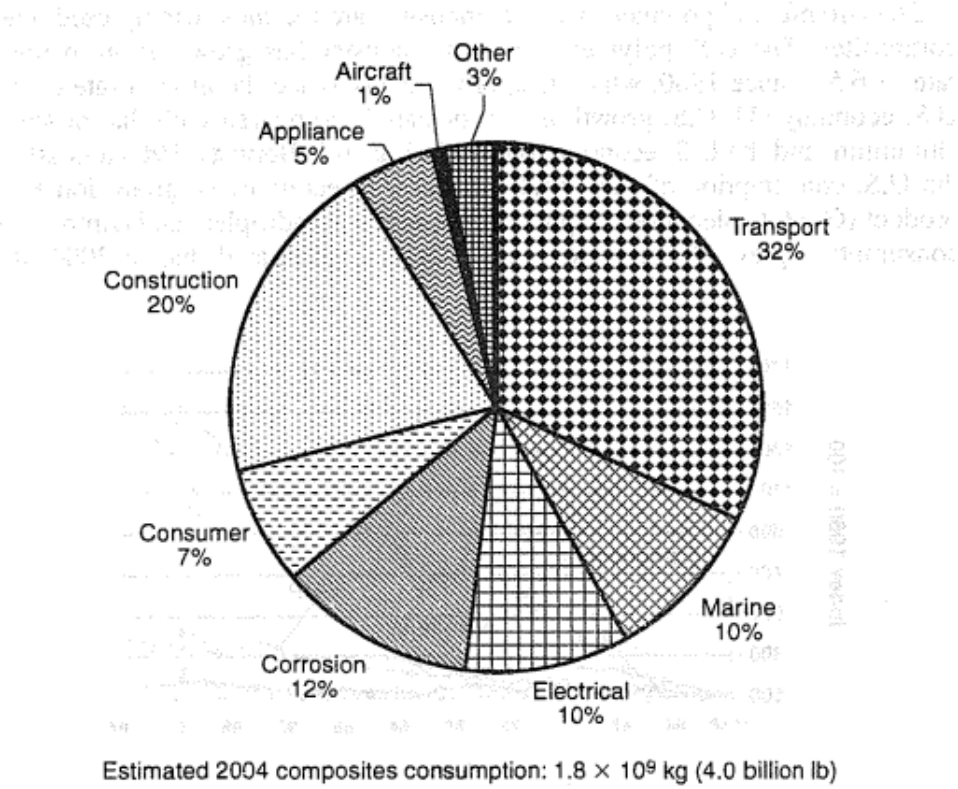


Figura 1 – Utilização de Materiais compósitos nos EUA em 2004 (AGARWAL; BROUTMAN; CHANDRASHEKHARA, 2006)

Embora os materiais compósitos sejam associados com tecnologias de ponta, eles possuem diversas outras aplicações. Na Fig. 1, seus principais usos são mostrados.

Uma característica de materiais compósitos é que estes são normalmente anisotrópicos. Esta característica dificulta significativamente a modelagem matemática. Soluções analíticas para estes são limitadas a geometrias e carregamentos simples. Em problemas com um pouco mais de complexidade, os métodos numéricos passam a ser a única alternativa para a análise. Dentre os métodos numéricos disponíveis, o método dos elementos finitos é o que tem sido usado na maior parte das aplicações de engenharia.

1.1.2 O Método dos Elementos Finitos

O método dos elementos finitos é um método numérico muito utilizado na resolução de problemas de valor inicial e de contorno. Ele é utilizado para diversas aplicações da engenharia como análise de tensões, fluidos, eletromagnetismo e transferência de calor. Segundo Fish e Belytschko (2007), só nos Estados Unidos, US$1 bilhão são gastos

anualmente em software e tempo de computação de elementos finitos, o que mostra a popularidade e relevância do método.

Dentro deste contexto, se encontra o deal.II (www.dealii.org). Deal.II é uma biblioteca que oferece todas as ferramentas para realizar simulações usando o método dos elementos finitos. Ele começou a ser desenvolvido em 1992 e está em código livre desde 2000. De acordo com Bangerth, Hartmann e Kanschat (2007), a biblioteca foi projetada não para resolver um problema específico, mas vários tipos de problemas diferentes. Uma de suas principais características é a extensiva documentação.

Sua documentação é dividida em 2 partes. A 1ª se apresenta na forma de tutorial, contendo mais de 60 programas e se impressa, ocupará mais de 400 páginas. O propósito desta é mostrar como as diferentes partes da biblioteca podem ser combinadas para resolver um problema. A 2ª é um manual descrevendo todas as classes e funções presentes. Em caso de erros de programação, o deal.II conta com uma série de verificações.

Existe uma infinidade de bibliotecas para o método dos elementos finitos. Entretanto, segundo Bangerth, Hartmann e Kanschat (2007) poucas são feitas de maneira a ser suficientemente gerais para vários problemas. Além disso, diminui-se a quantidade de bibliotecas que sejam bem documentadas permitindo-se o uso fora dos desenvolvedores. O deal.II é uma que é muito utilizada por diversos cientistas sem ligação com seus desenvolvedores e cresce desde 1997.

O deal.II é uma biblioteca em C++ orientada ao objeto que possui todas as funcionalidades para o método dos elementos finitos.

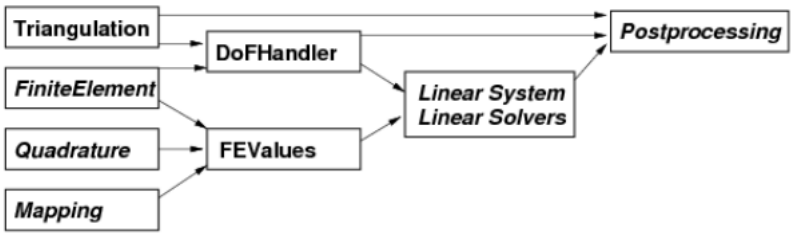


Figura 2 – Principais classes do deal.II para simulações numéricas (BANGERTH; HARTMANN; KANSCHAT, 2007)

A Fig. 2 mostra várias funcionalidades que o deal.II contém. Vê-se que todos os passos necessários para resolver um problema com o método dos elementos finitos estão disponíveis.

A fim de oferecer flexibilidade, os recursos do deal.II são organizados em diferentes classes que podem ser combinadas conforme a necessidade do usuário.

Segundo Bangerth, Hartmann e Kanschat (2007), o deal.II foi usado numa variedade de aplicações em pesquisa acadêmica, ciências aplicadas e projetos industriais. Há publicações em áreas diversas desde escoamentos compressíveis e incompressíveis, modelagem de células de combustível, simulação de crescimento de cristais e outros. Em Fig.

3 está uma lista de publicações conhecidas que utilizam o deal.II. As colunas em cinza, devido a serem mais recentes estão incompletas.

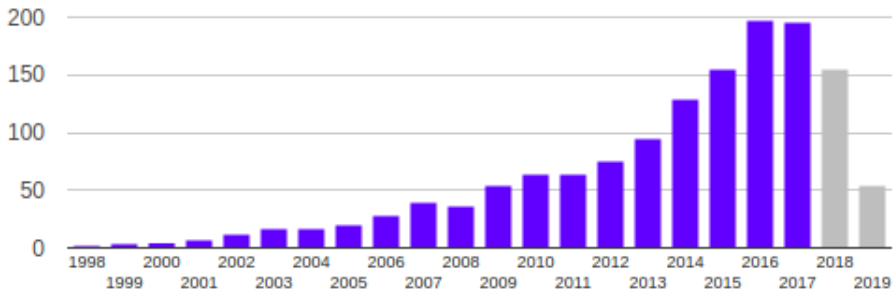


Figura 3 – Publicações conhecidas que usam o deal.II. Fonte: deal.org

1.2 Objetivos

O objetivo geral deste trabalho é o cálculo de tensões em materiais anisotrópicos com o programa deal.II e a verificação dos resultados.

Como objetivos específicos, têm-se:

- Estender a formulação elástica isotrópica para a formulação elástica anisotrópica através da implementação da relação constitutiva tensão-deformação para materiais anisotrópicos
- Cálculo de deslocamentos nodais em problemas de elasticidade isotrópica e anisotrópica
- Cálculo de tensões e deformações nos pontos de integração em problemas de elasticidade isotrópica e anisotrópica
- Comparação dos resultados obtidos neste trabalho com soluções analíticas e com resultados obtidos usando o método dos elementos de contorno para verificação da formulação implementada.

1.3 Contribuição

Após buscas em bases de dados de periódicos indexados na documentação da biblioteca deal.II não foi encontrado nenhum trabalho, no qual a biblioteca fosse usada para a análise de elasticidade linear anisotrópica. Esse fato mostra a importância do presente trabalho.

2 Teoria de elasticidade anisotrópica

2.1 Introdução

Neste capítulo são apresentadas a teoria e as hipóteses da elasticidade anisotrópica necessária para o entendimento do trabalho. Primeiramente são descritos os princípios físicos governantes gerais, baseados na 2ª Lei de Newton e na Lei da elasticidade de Hooke. Em seguida, analisa-se o caso de uma lâmina especialmente ortotrópica e quais são suas propriedades mecânicas. Enfim, estende-se a análise para materiais compósitos constituídos por várias lâminas. Nisso, explica-se as considerações feitas para a formulação do problema a ser resolvido.

2.2 Equações de Elasticidade

Para trabalhar um problema de elasticidade linear, começa-se pela 2ª lei de Newton. Esta diz que a somatória de forças é igual a variação de momentum. Em um ponto, ela é enunciada conforme a Eq. (2.1):

$$\nabla \cdot \boldsymbol{\sigma} + \boldsymbol{F} = \rho \ddot{\boldsymbol{u}} \tag{2.1}$$

onde $\nabla\cdot$ é o operador divergente, $\boldsymbol{\sigma}$ é o tensor de tensões de Cauchy, $\boldsymbol{F}$ são as forças de campo por unidade de volume, ρ a densidade e $\boldsymbol{u}$ o vetor deslocamento, $\ddot{\boldsymbol{u}}$ representa a segunda derivada do deslocamento em relação ao tempo, ou seja, a aceleração. Considerando-se um problema estático, $\ddot{\boldsymbol{u}}$ é nulo. Na Fig. 4 tem-se as componentes de tensão em um elemento cúbico tridimensional.

Uma grandeza importante de ser definida para problemas de elasticidade é a deformação, que é dada por:

$$\boldsymbol{\varepsilon} = \frac{1}{2}[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T] \tag{2.2}$$

Tabela 2 – Notação de Voigt

Notação tensorial	Notação de Voigt
11	1
22	2
33	3
23=32	4
13=31	5
12=21	6

onde $\boldsymbol{\varepsilon}$ é a deformação. Para se ter um problema bem posto, é necessário relacionar Eq. (2.1) e Eq. (2.2) através de uma relação constitutiva. Neste caso, é a Lei de Hooke, dada por:

$$\boldsymbol{\sigma} = \boldsymbol{C} : \boldsymbol{\varepsilon} \tag{2.3}$$

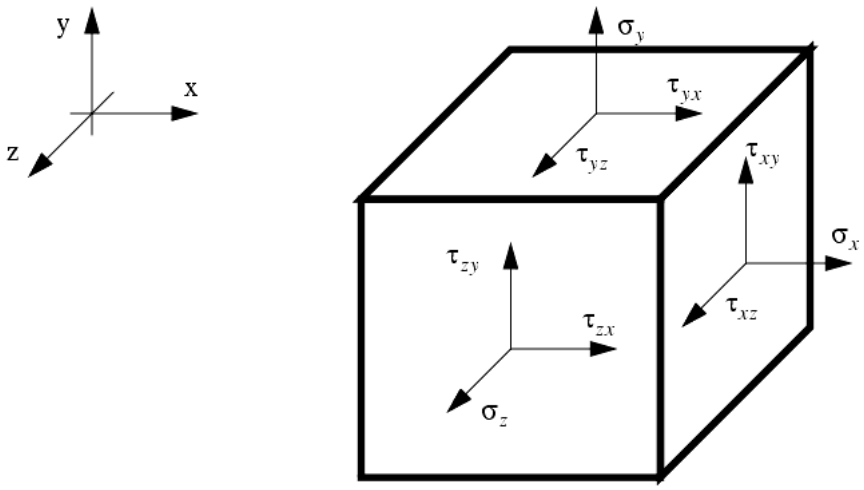


Figura 4 – Tensões em um elemento cúbico

No qual $\boldsymbol{C}$ representa o tensor rigidez, que é de 4ª ordem. Devido a propriedades de simetria do tensor de tensões de Cauchy e do tensor de deformações, além de considerações termodinâmicas baseadas na função deformação-energia, $\boldsymbol{C}$ apresenta as seguinte simetrias (em notação indicial).

$$C_{ijks} = C_{jiks} = C_{ksji} \tag{2.4}$$

Devido a estas simetrias, é possível então representar a relação constitutiva de uma maneira mais simples, com vetores e uma matriz de 2ª ordem. Para isso, usa-se a notação de Voigt. A relação entre os quatro índices do tensor de 4ª ordem $\boldsymbol{C}$ e a matriz de Voigt é dada pela Tabela 2.

Assim C_{ijks} se torna C_{ij}.

2.3 Laminados Ortotrópicos

2.3.1 Introdução

Devido às propriedades da matriz de rigidez, os materiais podem ser classificados em isotrópicos e anisotrópicos. Materiais isotrópicos, possuem propriedades mecânicas independentes da direção. Com isso, possuem uma série de simetrias no tensor de rigidez e todos os elementos deste podem ser escritos em função de 2 constantes independentes. Seu tensor de rigidez em notação de Voigt só possui 12 valores não nulos.

Os materiais que não são isotrópicos são chamados de anisotrópicos. Nestes, as propriedades mecânicas dependem da direção, fazendo sua análise mais complexa. Com isso, o número de constantes independentes necessárias para caracterizar as propriedades mecânicas aumenta. Segundo Agarwal, Broutman e Chandrashekhara (2006), os materiais compósitos são, em geral, anisotrópicos.

2.3.2 Lâmina Ortotrópica

Os materiais anisotrópicos podem ser classificados conforme simetrias que exibem em suas propriedades elásticas. Um tipo especial bastante estudado é o material ortotrópico, muito comum na forma de lâminas ortotrópicas.

De acordo com Agarwal, Broutman e Chandrashekhara (2006), uma lâmina é uma estrutura plana cuja espessura é muito menor que suas outras dimensões. Em materiais compósitos ela é feita de uma composição entre matriz e fibras. As propriedades mecânicas deste material são então intermediárias entre as dos componentes. Na Fig 5 , temos um desenho de uma lâmina ortotrópica.

A ortotropia é caracterizada por 3 planos de simetria. Nestes planos, as constantes do material não são alteradas por uma transformação ortogonal de reflexão. O tensor $\boldsymbol{C}$ então contém 9 constantes independentes para casos tridimensionais e 4 para casos bi-dimensionais. Define-se as direções e os planos principais como sendo as paralelas e as perpendiculares às fibras. Se houver coincidência entre as direções principais e as direções do sistema de coordenadas utilizado, o sistema fica muito mais simples, pois muitos termos se tornam iguais a zero. Com a notação de Voigt, a relação constitutiva é dada por:

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_{23} \\ \tau_{13} \\ \tau_{12} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ C_{12} & C_{22} & C_{23} & 0 & 0 & 0 \\ C_{31} & C_{32} & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & C_{66} \end{bmatrix} \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \gamma_{23} \\ \gamma_{13} \\ \gamma_{12} \end{bmatrix} \tag{2.5}$$

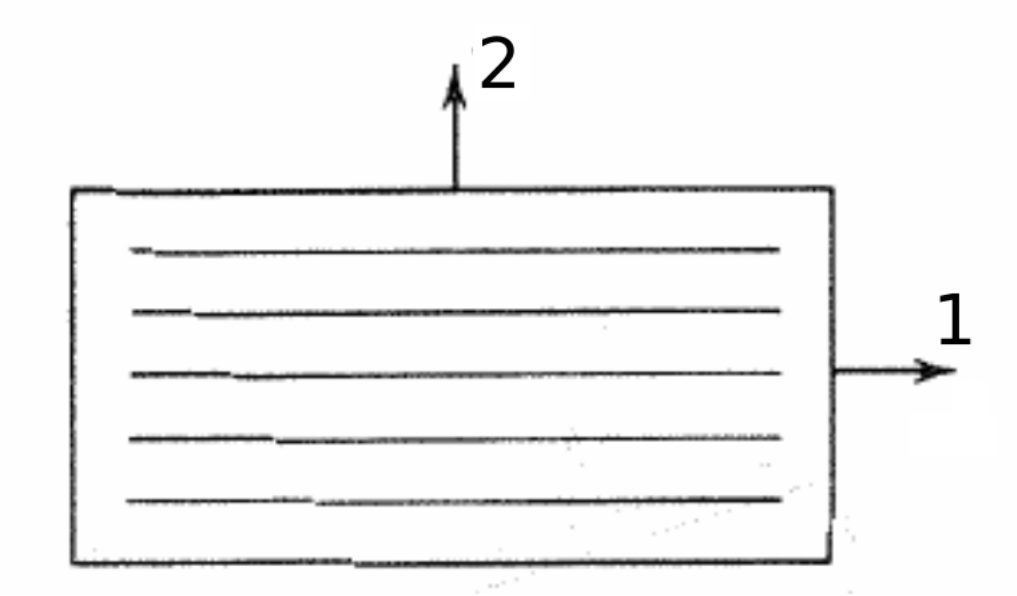


Figura 5 – Lâmina especialmente Ortotrópica (AGARWAL; BROUTMAN; CHANDRASHEKHARA, 2006)

e o material é chamado de especialmente ortotrópico.

As relações tensão deformação podem ser expressas de maneira inversa da seguinte forma:

$$\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \gamma_{23} \\ \gamma_{13} \\ \gamma_{12} \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} & S_{13} & 0 & 0 & 0 \\ S_{12} & S_{22} & S_{23} & 0 & 0 & 0 \\ S_{31} & S_{32} & S_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & S_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & S_{66} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_{23} \\ \tau_{13} \\ \tau_{12} \end{bmatrix} \tag{2.6}$$

A matriz $\boldsymbol{S}$ é chamada de matriz de flexibilidade. Segundo Lekhnitskii (1981), tem-se nas Eq. (2.7), Eq. (2.8), Eq. (2.9) e Eq. (2.10), as equações gerais de elasticidade para um corpo ortotrópico em função das constantes de engenharia:

$$\epsilon_1 = \frac{1}{E_1}\sigma_1 - \frac{\nu_{21}}{E_2}\sigma_2 - \frac{\nu_{31}}{E_3}\sigma_3 \tag{2.7}$$

$$\epsilon_2 = -\frac{\nu_{12}}{E_1}\sigma_1 + \frac{1}{E_2}\sigma_2 - \frac{\nu_{32}}{E_3}\sigma_3 \tag{2.8}$$

$$\epsilon_3 = -\frac{\nu_{13}}{E_1}\sigma_1 - \frac{\nu_{23}}{E_2}\sigma_2 + \frac{1}{E_3}\sigma_3 \tag{2.9}$$

$$\gamma_{23} = \frac{1}{G_{23}}\tau_{23}, \gamma_{13} = \frac{1}{G_{13}}\tau_{13}, \gamma_{12} = \frac{1}{G_{12}}\tau_{12} \tag{2.10}$$

onde E_i são os módulos de elasticidade, G_{ij} os módulos de cisalhamento e ν_{ij} os coeficientes de Poisson. Pode-se então obter os tensores em função das constantes de engenharia. A

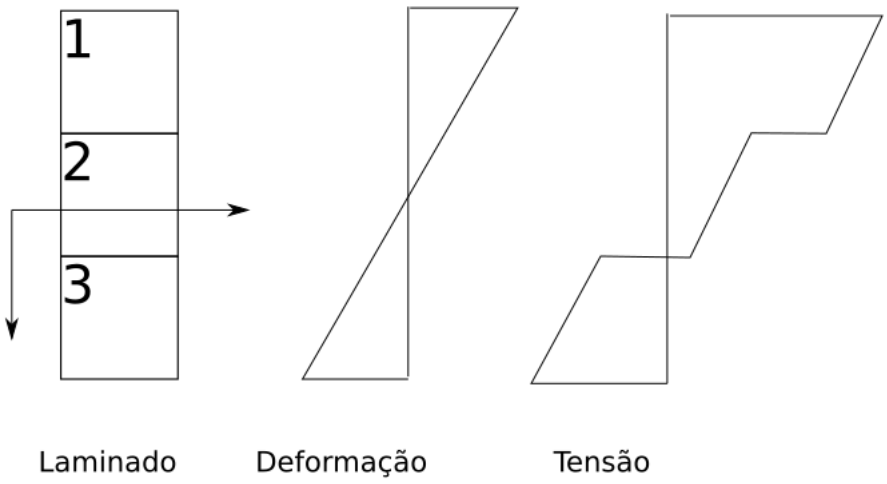


Figura 6 – Variação de tensão e deformação em um laminado hipotético de 3 lâminas

matriz de flexibilidade para o material ortotrópico então é:

$$\boldsymbol{S} = \begin{bmatrix} \frac{1}{E_1} & -\frac{\nu_{21}}{E_2} & -\frac{\nu_{31}}{E_3} & 0 & 0 & 0 \\ -\frac{\nu_{12}}{E_1} & \frac{1}{E_2} & -\frac{\nu_{32}}{E_3} & 0 & 0 & 0 \\ -\frac{\nu_{23}}{E_1} & -\frac{\nu_{23}}{E_2} & \frac{1}{E_3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{G_{23}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{G_{31}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{G_{12}} \end{bmatrix} \tag{2.11}$$

Para obter o tensor de rigidez, basta calcular o inverso do tensor de flexibilidade.

2.3.3 Materiais laminados em estado plano de tensão

Laminados são fabricados empilhando-se lâminas. Para o estudo dos mesmos, considera-se a hipótese de que a ligação entre as lâminas é perfeita. Ou seja, não ocorre deslizamento de lâminas e os deslocamentos são contínuos. Devido a isso, a variação da deformação e da tensão ao longo das lâminas de um laminado é linear. Porém, como as lâminas possuem, em geral, diferentes matrizes de rigidez, as tensões são em geral descontínuas ao longo do laminado, como pode ser visualizado em Fig. 6.

Para um problema laminado simétrico em estado plano de tensão, a tensão integrada ao longo da espessura do laminado é expressa por:

$$\begin{bmatrix} N_{11} \\ N_{22} \\ N_{12} \end{bmatrix} = \int_{\frac{-h}{2}}^{\frac{h}{2}} \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{12} \end{bmatrix} dZ \tag{2.12}$$

Neste caso, as demais componentes de tensão são nulas.

Considerando a relação entre tensões e deformações, pode-se escrever Eq. (2.12) em função das deformações do plano médio, da seguinte forma:

$$\begin{bmatrix} N_{11} \\ N_{22} \\ N_{12} \end{bmatrix} = \sum_{k=1}^{n} \left\{ \int_{-h_k}^{h_k} \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{bmatrix} \varepsilon_{11}^0 \\ \varepsilon_{22}^0 \\ \varepsilon_{12}^0 \end{bmatrix} dZ \right\} \tag{2.13}$$

Como as deformações do plano médio, neste caso (laminado simétrico sem aplicação de momentos), são constantes ao longo de toda a espessura, e as matrizes de rigidez não variam dentro de uma mesma lâmina, Eq. (2.13) pode ser simplificada para Eq. (2.14):

$$\begin{bmatrix} N_{11} \\ N_{22} \\ N_{12} \end{bmatrix} = \left\{ \sum_{k=1}^{n} \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \int_{-h_k}^{h_k} dZ \right\} \begin{bmatrix} \varepsilon_{11}^0 \\ \varepsilon_{22}^0 \\ \varepsilon_{12}^0 \end{bmatrix} \tag{2.14}$$

Define-se então a matriz A como sendo o resultado da somatória em Eq. (2.14) integrada ao longo da espessura, obtendo-se:

$$\begin{bmatrix} N_{11} \\ N_{22} \\ N_{12} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} \varepsilon_{11}^0 \\ \varepsilon_{22}^0 \\ \varepsilon_{12}^0 \end{bmatrix} \tag{2.15}$$

Caso não haja torções, as deformações são constantes ao longo da espessura.

3 O Método dos Elementos Finitos

3.1 Introdução

Muitos problemas de engenharia são descritos por equações diferenciais. Apesar de frequentemente ser possível resolvê-las analiticamente, em casos onde o domínio e as condições de contorno são levemente mais complicados, este trabalho torna-se árduo e algumas vezes impossível de ser realizado. Para isso, é bem comum recorrer-se ao Método dos Elementos Finitos, muito utilizado para problemas de elasticidade linear, transferência de calor e eletromagnetismo.

3.2 Condições de Contorno

Para resolver uma equação diferencial, é necessário conhecer as suas condições de contorno. Neste trabalho, vêm-se 2 tipos de condições de contorno:

- Condição de Dirichlet
- Condição de Neumann

Em um problema definido em um domínio Ω de contorno Γ, as condições de contorno de Dirichlet são definidas como: $u_\Gamma = f(x, y)$, enquanto as condições de Neumann são: $(\frac{\partial u}{\partial x})_\Gamma = g(x, y)$, sendo $f(x, y)$ e $g(x, y)$ duas funções quaisquer, contínuas ou não. Uma equação diferencial pode conter condições de Dirichlet em parte do seu contorno e condições de Neumann na outra parte. Neste caso, define-se então como Γ_u como sendo a região do contorno com condições de Dirichlet e Γ_t como sendo a com condições de Neumann. No caso de problemas elásticos, como o vetor deslocamento é uma grandeza vetorial, cada ponto do contorno possui d componenetes de deslocamento onde d é a dimensão ($d = 2$ para elasticidade plana e $d = 3$ para elasticidade tridimensional). Diz-se neste caso que um dado ponto específico do contorno possui d graus de liberdade. Por sua vez, as condições de contorno de Neumann e Dirichlet são definidas em cada um dos graus de liberdade de forma independente. Isto permite, por exemplo, em um problema de elasticidade plana, que uma certa região do contorno possa ter os deslocamentos conhecidos em um grau de liberdade e as forças de superíficie conhecidas em outro.

3.3 Forma Forte e Forma Fraca do Problema

Partindo-se das equações de elasticidade linear explicadas anteriormente, com Eq. (2.1), Eq. (2.2), Eq. (2.3), e com as condições de contorno, pode-se definir a forma forte do problema por (3.1)

$$\begin{aligned} \text{dado } \ \boldsymbol{u}^g, \boldsymbol{F}, \boldsymbol{t}, \boldsymbol{\sigma} = \boldsymbol{C} : \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon} = \frac{1}{2}[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T] \\ \text{encontrar } \boldsymbol{u} \text{ de maneira que} \\ \nabla \cdot \boldsymbol{\sigma} + \boldsymbol{F} = 0 \text{ em } \Omega \\ u_i = u_i^g \text{ em } \Gamma_{\boldsymbol{u}} \\ (\boldsymbol{\sigma n})_i = t_i \text{ em } \Gamma_{\boldsymbol{t}} \end{aligned} \tag{3.1}$$

Para aplicar o Método dos Elementos Finitos, usa-se a formulação integral, chamada de forma fraca. Esta formulação é equivalente, ou seja, se $\boldsymbol{u}(\boldsymbol{x})$ for uma solução para a forma fraca, também é uma solução para a forma forte.

As funções $\boldsymbol{u}(\boldsymbol{x})$ devem ter derivadas quadrado integráveis e estar contidas no espaço S onde

$$\boldsymbol{u} \in S = \{u | u(\Gamma_{u_i}) = f(x, y)\} \tag{3.2}$$

É necessário também definir as funções peso w. O espaço é similar a S, porém com a seguinte diferença

$$V = \{w | w(\Gamma_{u_i}) = 0\} \tag{3.3}$$

Sendo Ω o domínio, multiplica-se o caso estático da Eq. (2.1) por w_i e integra-se ao longo de Ω, obtendo a Eq. (3.4) (em notação indicial):

$$\int_\Omega w_i \sigma_{ij,j} d\Omega + \int_\Omega w_i f_i d\Omega = 0 \tag{3.4}$$

Com uma integral por partes no 1º termo, obtém-se:

$$\int_\Omega (w_i \sigma_{ij})_{,j} d\Omega - \int_\Omega w_{i,j} \sigma_{ij} d\Omega + \int_\Omega w_i f_i d\Omega = 0 \tag{3.5}$$

Com o teorema do divergente, tem-se:

$$\int_\Gamma w_i \sigma_{ij} n_j d\Gamma - \int_\Omega w_{i,j} \sigma_{ij} d\Omega + \int_\Omega w_i f_i d\Omega = 0 \tag{3.6}$$

Separando-se o contorno Γ em $\Gamma_{\bar{t}_i}$ e Γ_{u_i}, relativos às condições de contorno, obtém-se:

$$\int_\Omega w_{i,j}\sigma_{ij}d\Omega = \int_\Omega w_i f_i d\Omega + \int_{\Gamma_{u_i}} w_i\sigma_{ij}n_j d\Gamma + \int_{\Gamma_{\bar{t}_i}} w_i\sigma_{ij}n_j d\Gamma \tag{3.7}$$

Sabendo então que $w_i = 0$ em Γ_{u_i} e que $\bar{t}_i = \sigma_{ij}n_j$, com valor conhecido em $\Gamma_{\bar{t}_i}$, tem-se:

$$\int_\Omega w_{i,j}\sigma_{ij}d\Omega = \int_\Omega w_i f_i d\Omega + \int_{\Gamma_{\bar{t}_i}} w_i\bar{t}_i d\Gamma \tag{3.8}$$

A formulação fraca do problema é então dada por:

$$\begin{gathered}\text{dado } \; u_i^g, f_i, t_i, \sigma_{ij} = C_{ijkl} : \varepsilon_{kl} \text{ e } \varepsilon_{kl} = \frac{1}{2}[\frac{\partial u_k}{\partial x_l} + \frac{\partial u_l}{\partial x_k}] \\ \text{encontrar } u_i \text{ de maneira que} \\ \int_\Omega w_{i,j}\sigma_{ij}d\Omega = \int_\Omega w_i f_i d\Omega + \int_{\Gamma_{\bar{t}_i}} w_i\bar{t}_i d\Gamma \;\; \forall w \in V = \{w|w(\Gamma_{u_i}) = 0\}\end{gathered} \tag{3.9}$$

3.4 Forma de Dimensão Finita e Funções de Forma

3.4.1 Forma de dimensão finita

Tanto w_i quanto u_i pertencem a espaços de funções de dimensão infinitas. Porém, no método dos elementos finitos, busca-se uma solução aproximada. Para isso, deve-se trabalhar em um espaço de dimensão finita. Por isso o espaço de funções da solução e da função peso são restritos. A solução e a função peso aproximadas são denominadas por u_i^he w_i^h. Estas funções pertencem a espaços de soluções de dimensão finita e estes espaços estão contidos nos espaços de dimensão infinita, ou seja

$$u^h \in S^h = \{u^h|u^h(\Gamma_{u_i}) = f(x)\} \tag{3.10}$$

$$w^h \in V^h = \{w^h|w^h(\Gamma_{u_i}) = 0\} \tag{3.11}$$

e

$$S^h \subset S \tag{3.12}$$

$$V^h \subset V \tag{3.13}$$

Na Eq. (3.14), tem-se então a forma de dimensão finita.

$$\int_\Omega w_{i,j}^h\sigma_{ij}^h d\Omega = \int_\Omega w_i^h f_i d\Omega + \int_{\Gamma_{\bar{t}_i}} w_i^h\bar{t}_i d\Gamma \tag{3.14}$$

Divide-se o domínio então em várias sub-domínios Ω^e, chamados de elementos finitos, de maneira que $\Omega^{e_1} \cap \Omega^{e_2} = \emptyset$, obtendo-se a seguinte equação:

$$\sum_e \int_{\Omega^e} w^h_{i,j} \sigma^h_{ij} d\Omega = \sum_e \int_{\Omega^e} w^h_i f_i d\Omega + \sum_{e \in E_{N_i}} \int_{\Gamma^e_{\bar{t}_i}} w^h_i \bar{t}_i d\Gamma \tag{3.15}$$

No caso do deal.II, para problemas bi-dimensionais, os elementos são quadriláteros.

3.4.2 Funções de Base

Em cada elemento então, há um certo número de nós. Considera-se então que S^h e V^h consistam de combinações lineares de determinadas funções $N^A(\boldsymbol{x})$. Portanto o deslocamento e a função peso aproximados em um determinado elemento são:

$$u^h_e = \sum_A \varnothing^A(\boldsymbol{x}) d^A_e \tag{3.16}$$

$$w^h_e = \sum_A \varnothing^A(\boldsymbol{x}) c^A_e \tag{3.17}$$

onde A enumera os nós do elemento e d e c são os valores das funções no respectivo elemento. $\varnothing^A(\boldsymbol{x})$ é chamado de função de forma ou de base.

Mapeia-se o domínio físico, da variável $\boldsymbol{x}$ com um domínio chamado de bi-unitário, com a variável $\boldsymbol{\xi}$. Portanto pode-se definir $\boldsymbol{x}$ em função de $\boldsymbol{\xi}$. ξ_I varia entre -1 e 1. Portanto

$$\varnothing(\boldsymbol{x}) = \varnothing(\boldsymbol{x}(\boldsymbol{\xi})) \tag{3.18}$$

e

$$x_i(\xi) = \sum_A \varnothing^A(\xi_I) x^A_i \tag{3.19}$$

Em Fig. 7 e Fig. 8, isto fica mais claro

Para facilitar a notação, serão referidos então por $\varnothing^A(\boldsymbol{\xi})$. Uma forma muito utilizada, é definir $\varnothing^A$ como sendo polinômios de Lagrange, onde $\varnothing^A = 1$ nos nós de d^A_e correspondente e $\varnothing^A = 0$ nos demais nós. Neste trabalho, serão utilizados polinômios de 1ª ordem, (chamadas funções bilineares) e quadráticos. No caso bilinear, os elementos devem possuir então 4 nós, enquanto no caso quadrático 9, conforme Fig. 9a e Fig. 9b.

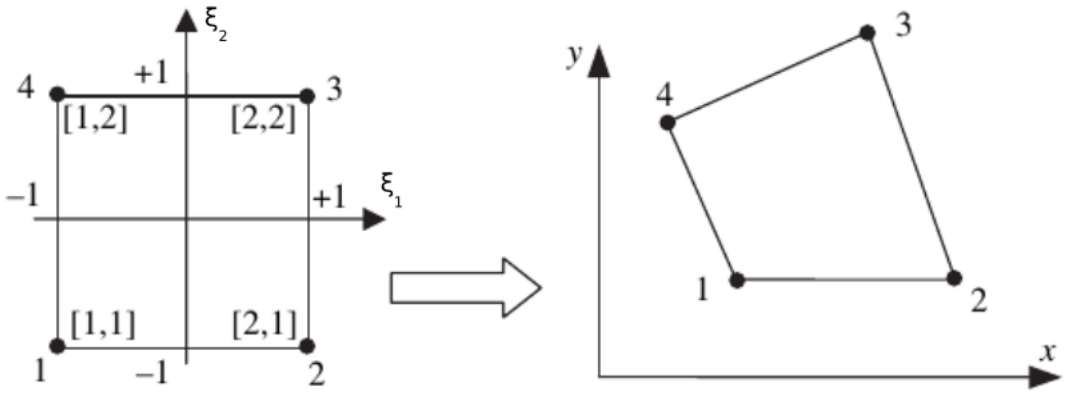


Figura 7 – Mapeamento pelo domínio bi-unit para funções bilineares (FISH; BELYTSCHKO, 2007) com alterações

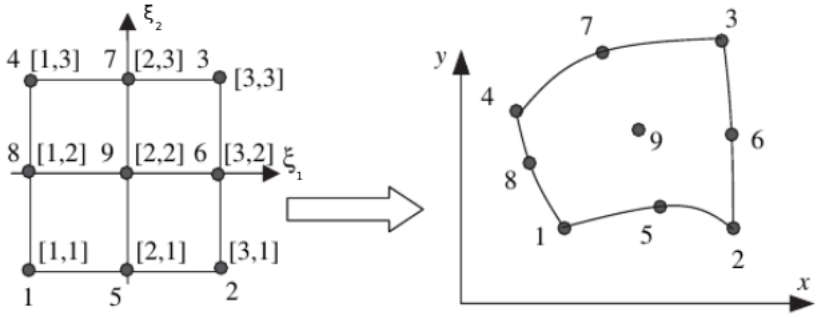
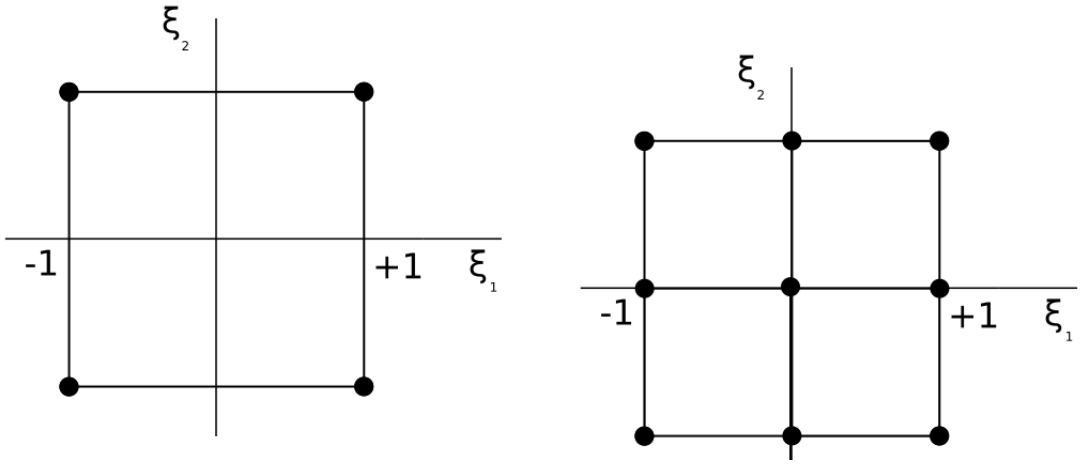


Figura 8 – Mapeamento pelo domínio bi-unit para funções quadráticas (FISH; BELYTSCHKO, 2007) com alterações

Para funções de forma bilineares, $\varnothing$ é dado então por Eq. (3.20)

$$\begin{aligned}
\varnothing^1 &= \frac{1}{4}(1-\xi_1)(1-\xi_2) \\
\varnothing^2 &= \frac{1}{4}(1+\xi_1)(1-\xi_2) \\
\varnothing^3 &= \frac{1}{4}(1+\xi_1)(1+\xi_2) \\
\varnothing^4 &= \frac{1}{4}(1-\xi_1)(1+\xi_2)
\end{aligned} \tag{3.20}$$



(a) Posições dos nós das funções de forma bilineares

(b) Posições dos nós das funções de forma quadráticas

Figura 9 – Nós de um elemento

Já funções de forma quadráticas, são dadas por Eq. (3.21)

$$
\begin{aligned}
\varnothing^1 &= \frac{1}{4}\xi_1(\xi_1-1)\xi_2(\xi_2-1)\\
\varnothing^2 &= \frac{1}{2}(1-\xi_1^2)\xi_2(\xi_2-1)\\
\varnothing^3 &= \frac{1}{4}\xi_1(1+\xi_1)\xi_2(\xi_2-1)\\
\varnothing^4 &= \frac{1}{2}\xi_1(1+\xi_1)(1-\xi_2^2)\\
\varnothing^5 &= \frac{1}{4}\xi_1(1+\xi_1)\xi_2(1+\xi_2)\\
\varnothing^6 &= \frac{1}{2}(1-\xi_1^2)\xi_2(1+\xi_2)\\
\varnothing^7 &= \frac{1}{4}\xi_1(\xi_1-1)\xi_2(1+\xi_2)\\
\varnothing^8 &= \frac{1}{2}\xi_1(\xi_1-1)(1-\xi_2^2)\\
\varnothing^9 &= \frac{1}{4}(1-\xi_1^2)(1-\xi_2^2)
\end{aligned}
\tag{3.21}
$$

Na Fig. 10 e na Fig. 11 estão representações gráficas das funções de forma bilineares e quadráticas, respectivamente.

Além disso, $\varnothing^A_{,j} = \varnothing_{,\xi_I}\frac{\partial \xi_I}{\partial x_j}$. Sabendo que o Jacobiano $J(\xi)$ é $J_{ij}(\xi) = x_{i,\xi_I}$, pode-se obter as derivadas de $\boldsymbol{\xi}$ em relação a $\boldsymbol{x}$, pois $J^{-1}_{ij}(\boldsymbol{\xi}) = \xi_{I,x_j}$. Será necessário usar os gradientes destas funções, que são definidos como:

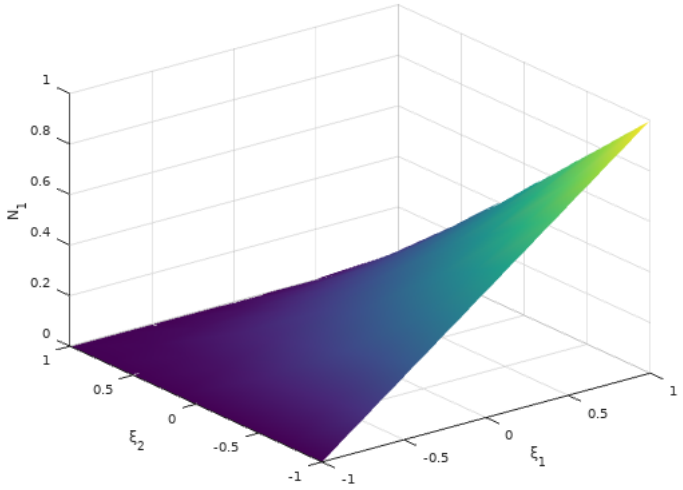


Figura 10 – Função de forma bilinear

$$u^h_{ei,j} = \sum_A \varnothing^A_{,j} d^A_{ie} \tag{3.22}$$

$$w^h_{ei,j} = \sum_A \varnothing^A_{,j} c^A_{i_e} \tag{3.23}$$

onde $\varnothing^A_{,j}$ é:

$$\varnothing^A_{,j} = \varnothing_{,\xi_I} \frac{\partial \xi_I}{\partial x_j} \tag{3.24}$$

Analisando-se o 1º termo em um elemento, do tensor de tensões, considerando-se (2.3), (2.2), (3.16), (3.17), (3.24) e (3.22) pode-se reescrevê-lo como:

$$\int_{\Omega^e} w^h_{i,j} \sigma^h_{ij} d\Omega = \int_{\Omega^e} w^h_{i,j} C_{ijkl} u^h_{k,l} d\Omega = \int_{\Omega^e} \sum_A \varnothing^A_{,j} c^A_i C_{ijkl} \sum_B \varnothing^B_{,l} d^B_k d\Omega \tag{3.25}$$

Como $\boldsymbol{d}$ e $\boldsymbol{c}$ são constantes, é possível escrevê-los fora da integral. Continuando o desenvolvimento então, obtém-se:

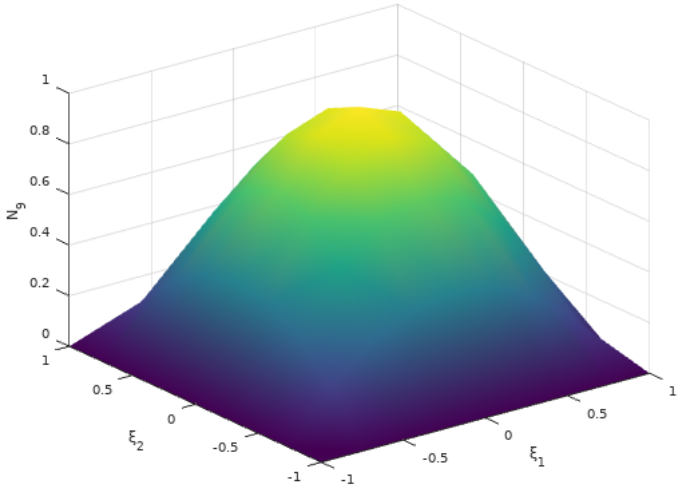
$$\sum_{A,B} c^A_i \left(\int_{\Omega^e} \varnothing^A_{,j} C_{ijkl} \varnothing^B_{,l} d\Omega\right) d^B_k = \sum_{A,B} c^A_i \left(\int_{\Omega^\xi} \varnothing^A_{,j} C_{ijkl} \varnothing^B_{,l} det[J(\xi)] d\Omega_\xi\right) d^B_k \tag{3.26}$$

Chamando o resultado da integral de K, obtém-se:

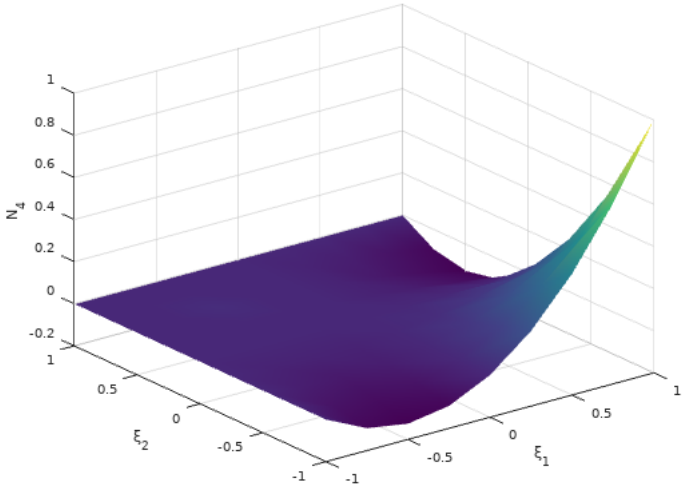
$$\sum_{A,B} c^A_i \left(\int_{\Omega^\xi} \varnothing^A_{,j} C_{ijkl} \varnothing^B_{,l} det[J(\xi)] d\Omega_\xi\right) d^B_k = \sum_{A,B} c^A_i K^{AB}_{ik} d^B_k \tag{3.27}$$

Aplicando-se um processo similar no 2º termo, o da força de campo, obtém-se:

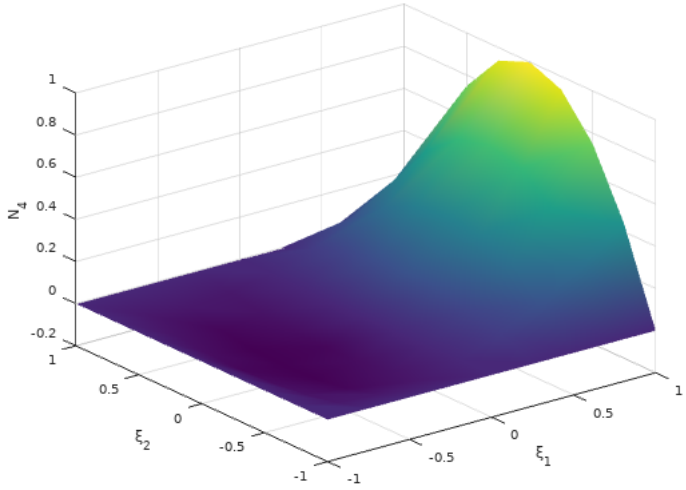
$$\int_{\Omega^e} w^h_i f_i d\Omega = \int_{\Omega^e} \left(\sum_A \varnothing^A c^A_i\right) f_i d\Omega = \sum_A c^A_i \int_{\Omega^\xi} \varnothing^A f_i det[J(\xi)] d\Omega_\xi \tag{3.28}$$



(a) Funções no nó central



(b) Função em um nó de canto



(c) Função em um nó no ponto médio da aresta

Figura 11 – Função de forma quadrática

Chamando o resultado da integral de F^{intA}, obtém-se:

$$\int_{\Omega^e} w_i^h f_i d\Omega = \sum_A c_i^A F_i^{intA} \tag{3.29}$$

Por fim, desenvolve-se o último termo, da força de superfície:

$$\int_{\Gamma^e_{\bar{t}_i}} w_i^h \bar{t}_i d\Gamma = \int_{\Gamma^e_{\bar{t}_i}} (\sum_A \varnothing^A c_{i_e}^A) \bar{t}_i d\Gamma = \sum_A c_{i_e}^A \int_{\Gamma^e_{\bar{t}_i}} \varnothing^A \bar{t}_i d\Gamma = \sum_A c_{i_e}^A \int_{\Gamma^e_{\bar{t}_i}}^{\xi} \varnothing^A \bar{t}_i det[J_s] d\Gamma_\xi \tag{3.30}$$

Chamando-se então o resultado da integral de $F^{\bar{t}A}$:

$$\int_{\Gamma^e_{\bar{t}_i}} w_i^h \bar{t}_i d\Gamma = \sum_A c_{i_e}^A F_i^{\bar{t}A} \tag{3.31}$$

Juntando-se todos os termos então, obtém-se a equação completa:

$$\sum_e \sum_{A,B} c_{i_e}^A K_{ik_e}^{AB} d_{k_e}^B = \sum_e \sum_A c_{i_e}^A F_{i_e}^{intA} + \sum_{e \in E_{N_i}} \sum_A c_{i_e}^A F_{i_e}^{\bar{t}A} \tag{3.32}$$

A equação está agora na forma discreta.

3.4.3 Forma Matricial

Escrevendo em forma matricial e resolvendo a somatória dentro do elemento, obtém-se para o lado esquerdo:

$$\sum_e \sum_{A,B} c_{i_e}^A K_{ik_e}^{AB} d_{k_e}^B = \sum_e \sum_{A,B} \boldsymbol{c_e^{AT} K_e^{AB} d_e^B} = \sum_e \boldsymbol{c_e^T K_e d_e} \tag{3.33}$$

onde $\boldsymbol{K_e}$ é uma matriz maior agora. Para o 1 º termo do lado direito, tem-se:

$$\sum_e \sum_A c_{i_e}^{AT} F_i^{intA} = \sum_e \sum_A \boldsymbol{c_e^{AT} F_e^{intA}} = \sum_e \boldsymbol{c_e^T F_e^{int}} \tag{3.34}$$

Juntando-se os lados direito e esquerdo, obtém-se a forma matricial da equação dada por:

$$\sum_e \boldsymbol{c_e^T K_e d_e} = \sum_e \boldsymbol{c_e^T F_e^{int}} + \sum_{e \in E_{N_i}} \sum_A c_{i_e}^A F_{i_e}^{\bar{t}A} \tag{3.35}$$

Agora é necessário fazer a montagem para uma equação global. De certa maneira, juntar os elementos finitos. Os vetores globais $\boldsymbol{c}$ e $\bar{\boldsymbol{d}}$ (chamado assim pois inclui todos os pontos) são da forma:

$$\boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_1^A \\ \boldsymbol{c}_2^A \\ \boldsymbol{c}_3^A \\ . \\ . \\ . \end{bmatrix} ; \bar{\boldsymbol{d}} = \begin{bmatrix} \boldsymbol{d}_1^A \\ \boldsymbol{d}_2^A \\ \boldsymbol{d}_3^A \\ . \\ . \\ . \end{bmatrix} \tag{3.36}$$

É importante deixar claro que as condições de Dirichlet eliminam alguns elementos no vetor $\boldsymbol{c}$. A representa o nó e o sub-índice representa a direção (se o grau de liberdade é no eixo x_1 ou x_2). Notar que vetor $\bar{\boldsymbol{d}}$ possui uma quantidade de elementos como sendo o número de nós vezes o número de dimensões espaciais do problema, enquanto o vetor $\boldsymbol{c}$ possui isso menos o número de graus de liberdade com condições de Dirichlet.

Define-se também $\bar{\boldsymbol{K}}$ da seguinte forma:

$$\bar{\boldsymbol{K}} = A_e \boldsymbol{K_e} \tag{3.37}$$

onde A_e representa um operador de montagem sobre cada matriz de rigidez. Nesta operação uma matriz global é montada, considerando inclusive elementos próximos um do outro fazendo uma soma dos termos. Assim então:

$$\sum_e \boldsymbol{c_e^T K_e d_e} = \boldsymbol{c^T \bar{K} \bar{d}} \tag{3.38}$$

Com processo análogo com $\boldsymbol{F_e^{int}}$ e $\boldsymbol{F_e^{t\bar{A}}}$, pode-se obter a forma matricial global que é dada por:

$$\boldsymbol{c^T \bar{K} \bar{d} = c^T F^{int} + c^T F^{\bar{t}}} \tag{3.39}$$

Por fim, para resolver o sistema, basta somar então os vetores $\boldsymbol{F^{int}}$ e $\boldsymbol{F^{\bar{t}}}$ e incluir as condições de Dirichlet, obtendo

$$\boldsymbol{c^T K \bar{d} = c^T (F^{int} + F^{\bar{t}} - \bar{d}_i^A \bar{K}^A)} \tag{3.40}$$

chamando o termo entre parênteses de $\boldsymbol{F}$:

$$\boldsymbol{c^T (Kd - F)} = 0 \tag{3.41}$$

Como $\boldsymbol{c^T}$ pode ser um vetor arbitrário, então é necessário que:

$$\boldsymbol{Kd - F} = 0 \tag{3.42}$$

Com um resolvedor de sistemas lineares, é possível então resolver o sistema linear dado por:

$$\boldsymbol{Kd = F} \tag{3.43}$$

Com $\boldsymbol{d}$ e as funções de forma é possível obter $\boldsymbol{u^h}$.

4 Implementação no deal.II

O código usado como modelo, foi baseado no template CA3b de Garikipati (2013), que originalmente possui implementação da formulação de elasticidade linear isotrópica. Para casos ortotrópicos no deal.II, é necessário alterar algumas coisas. No snippet 4.1 está o código da função que retorna os valores da matriz de rigidez do código originalm isotrópico.

```
template <int dim>
double FEM<dim>::C(unsigned int i,unsigned int j,unsigned int k,unsigned int l){

  //Define the material parameters of Young's modulus and Poisson's ratio
//  double E=2.0e11 ,  //EDIT
 //   nu=0.3 ; //EDIT
  double E=pow(10,5) ,  //EDIT
    nu=0.25 ; //EDIT
  double lambda=(E*nu)/((1.+nu)*(1.-2.*nu)),
    mu=E/(2.*(1.+nu));

  return lambda*(i==j)*(k==l) + mu*((i==k)*(j==l) + (i==l)*(j==k));

}
```

Listing 4.1 – Função que retorna valores da matriz de rigidez para casos isotrópicos

Nesta função, declara-se o módulo de elasticidade E e o coeficiente de Poisson (nu), e os elementos da matriz de rigidez são retornados. Para casos ortotrópicos, o código foi modificado conforme mostrado na listagem 4.2.

```
template <int dim>
double FEM<dim>::C(unsigned int i,unsigned int j,unsigned int k,unsigned int l){

  double C11,C12,C13,C16,C26,C22,C23,C33,C44,C55,C66,lambda,mu;
  C11=8.4867*pow(10,8);
  C12=3.0066*pow(10,8);
  C13=0;
  C16=3.6529*pow(10,8);
  C22=4.0700*pow(10,8);
  C23=0;
  C26=2.4145*pow(10,8);
  C33=0;
  C44=0;
  C55=0;
  C66=3.1118*pow(10,8);
  lambda=3;
  mu=4.3333;

  if (i==0 && j==0 && k==0 && l==0)
    {
      return (C11);
    }
  else if ((i==0 && j==0 && k==1 && l==1) || (i==1 && j==1 && k==0 && l==0))
    {
      return (C12);
```

```
    }
  else if ((i==0 && j==0 && k==0 && l==1) || (i==0 && j==0 && k==1 && l==0) || (i==0 && j
      ↪ ==1 && k==0 && l==0) || (i==1 && j==0 && k==0 && l==0))
    {
      return (C16);
    }
  else if ((i==1 && j==1 && k==0 && l==1) || (i==1 && j==1 && k==1 && l==0) || (i==0 && j
      ↪ ==1 && k==1 && l==1) || (i==1 && j==0 && k==1 && l==1))
    {
      return (C26);
    }
  else if ((i==0 && j==0 && k==2 && l==2) || (i==2 && j==2 && k==0 && l==0))
    {
      return (C13);
    }
  else if ((i==1 && j==1 && k==1 && l==1))
    {
      return (C22);
    }
  else if ((i==2 && j==2 && k==1 && l==1) || (i==1 && j==1 && k==2 && l==2))
    {
      return (C23);
    }
  else if ((i==2 && j==2 && k==2 && l==2))
    {
      return (C33);
    }
  else if ((i==1 && j==2 && k==1 && l==2) || (i==1 && j==2 && k==2 && l==1) || (i==2 && j
      ↪ ==1 && k==1 && l==2) || (i==2 && j==1 && k==2 && l==1))
    {
      return (C44);
    }
  else if ((i==0 && j==2 && k==0 && l==2) || (i==0 && j==2 && k==2 && l==0) || (i==2 && j
      ↪ ==0 && k==0 && l==2) || (i==2 && j==0 && k==2 && l==0))
    {
      return (C55);
    }
  else if ((i==0 && j==1 && k==0 && l==1) || (i==0 && j==1 && k==1 && l==0) || (i==1 && j
      ↪ ==0 && k==0 && l==1) || (i==1 && j==0 && k==1 && l==0))
    {
      return (C66);
    }
  else
    {
      return 0;
    }

}
```

Listing 4.2 – Função que retorna valores da matriz de rigidez para casos ortotrópicos.

A função assim retorna os elementos da matriz para casos ortotrópicos.

O resto do código, não exige modificações relativas à matriz de rigidez. No snippet 4.3, monta-se então as matrizes locais, para cada elemento.

```
    for (unsigned int q=0; q<num_quad_pts; ++q){
      //      double z = fe_values.quadrature_point(q)[2]; //y-coordinate at the current
          ↪ surface quad. point
      //      std::cout << "z " << z << std::endl;
      //evaluate elemental stiffness matrix, K^{AB}_{ik} = \integral N^A_{,j}*C_{ijkl}*N^
          ↪ B_{,l} dV
      for (unsigned int A=0; A<nodes_per_elem; A++) { //Loop over nodes
        for(unsigned int i=0; i<dim; i++){ //Loop over nodal dofs
          for (unsigned int B=0; B<nodes_per_elem; B++) {
            for(unsigned int k=0; k<dim; k++){
              for (unsigned int j = 0; j<dim; j++){
                for (unsigned int l = 0; l<dim; l++){
                  /*//EDIT - You need to define Klocal here. Note that the indices of
                      ↪ Klocal are the element dof numbers (0 through 23),
                    which you can caluclate from the element node numbers (0 through 8)
                        ↪ and the nodal dofs (0 through 2).
```

```
                    You'll need the following information:
                    basis gradient vector: fe_values.shape_grad(elementDOF,q), where
                        ↪ elementDOF is dim*A+i or dim*B+k
                    NOTE: this is the gradient with respect to the real domain (not the
                        ↪ bi-unit domain)
                    elasticity tensor: use the function C(i,j,k,l)
                    det(J) times the total quadrature weight: fe_values.JxW(q)*/
                        Klocal[dim*A+i][dim*B+k]+=fe_values.shape_grad(dim*A+i,q)[j]*C(i,
                            ↪ j,k,l)*
                         fe_values.shape_grad(dim*B+k,q)[l]*fe_values.JxW(q);
              }
            }
          }
        }
      }
    }
  }
```

Listing 4.3 – Montagem do Klocal

E em Snippet 4.4, ocorre a montagem da matriz global.

```
for(unsigned int i=0; i<dofs_per_elem; i++){
  //EDIT - Assemble F from Flocal (you can look at HW2)
  F[local_dof_indices[i]]+=Flocal[i];
  for(unsigned int j=0; j<dofs_per_elem; j++){
    //EDIT - Assemble K from Klocal (you can look at HW2)
    K.add(local_dof_indices[i],local_dof_indices[j],Klocal[i][j]);
  }
}
```

Listing 4.4 – Montagem do K global

Dessa maneira então, acrescentando-se o Snippet 4.2 ao código já existente, pode-se resolver problemas anisotrópicos.

5 Resultados

5.1 Introdução

A fim de verificar o entendimento do programa, o deal.II foi aplicado na análise de problemas bem conhecidos. As geometrias escolhidas foram a de uma placa com um furo central circular e um cilindro pressurizado, sendo todos estes problemas isotrópicos. Então, foi analisado uma placa de material compósito laminado sob tração. Estes problemas possuem soluções analíticas que permitem uma comparação com os resultados obtidos numericamente. Por fim, duas placas com furo de compósitos laminados sob tração multiaxial foram analisadas. Devido à ausência de soluções analíticas para estes casos, comparou-se os resultados com uma solução encontrada através do método dos elementos de contorno pelo programa utilizado em Albuquerque (2001). Para estes casos, utilizou-se sempre funções de forma de 2º grau.

5.2 Quarto de placa com Condições de Contorno de Neumann e Dirichlet

Este é o primeiro exemplo. As especificações são as seguintes: parte interna de uma placa com um furo circular no centro e tensões simétricas aplicadas na lateral, conforme indicado na Fig. 12.

Segundo Timoshenko (1951), a solução para o campo de tensões na parte interna é:

$$\begin{bmatrix}\sigma_{11}(r,\theta)\\ \sigma_{22}(r,\theta)\\ \sigma_{12}(r,\theta)\end{bmatrix} = \begin{bmatrix}1-\frac{a^2}{r^2}(\frac{3}{2}\cos(2\theta)+\cos(4\theta))+\frac{3a^4}{2r^4}\cos(4\theta)\\ -\frac{a^2}{r^2}(\frac{1}{2}\cos(2\theta)-\cos(4\theta))-\frac{3a^4}{2r^4}\cos(4\theta)\\ -\frac{a^2}{r^2}(\frac{1}{2}\sin(2\theta)+\sin(4\theta))+\frac{3a^4}{2r^4}\sin(4\theta)\end{bmatrix} \tag{5.1}$$

E segundo Fan, Coombs e Augarde (2018), a solução para o campo de deslocamentos é:

$$\begin{bmatrix}u_1(r,\theta)\\ u_2(r,\theta)\end{bmatrix} = \begin{bmatrix}\frac{10a}{8G}\{\frac{r}{a}(\kappa+1)\cos\theta+\frac{2a}{r}[(1+\kappa)\cos\theta+\cos(3\theta)]-\frac{2a^3}{r^3}\cos(3\theta)\}\\ \frac{10a}{8G}\{\frac{r}{a}(\kappa-3)\sin\theta+\frac{2a}{r}[(1-\kappa)\sin\theta+\sin(3\theta)]-\frac{2a^3}{r^3}\sin(3\theta)\}\end{bmatrix} \tag{5.2}$$

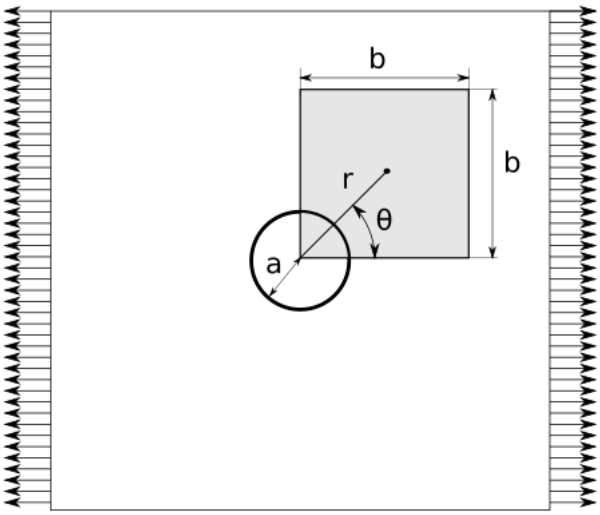


Figura 12 – Placa com furo circular no centro submetida a tensões simétricas nas laterais.

onde G é o módulo de cisalhamento e κ é a constante de Kolosov, definidos como

$$G = \frac{E}{2(1+\nu)} \tag{5.3}$$

$$\kappa = 3 - 4\nu \tag{5.4}$$

Foi analisada somente a seção destacada, devido às simetrias do problema. Por isso, na face inferior e na lateral esquerda serão aplicadas condições de restrição no deslocamento na direção x_2 e x_1 respectivamente. Já a face superior e a lateral direita serão carregadas com as tensões de superfície correspondentes às da solução exata.

Considerou-se $a = 1$, $b = 5$, $E = 10^5$ e $\nu = 0,25$.

O domínio foi discretizado conforme Fig. 13, com 32 elementos. Para funções de forma de 1º grau, contém então 45 nós, enquanto para funções de 2º grau e 153 nós.

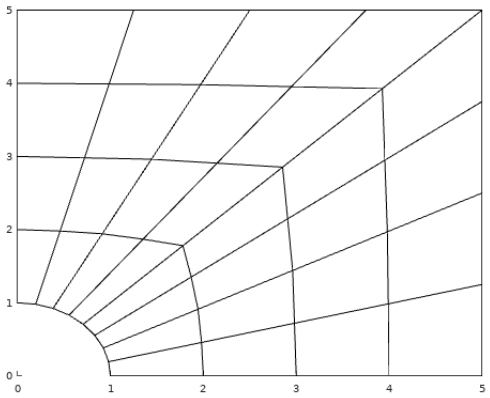


Figura 13 – Domínio discretizado

Obteve-se então os resultados mostrados na Fig. 14 e na Fig. 15

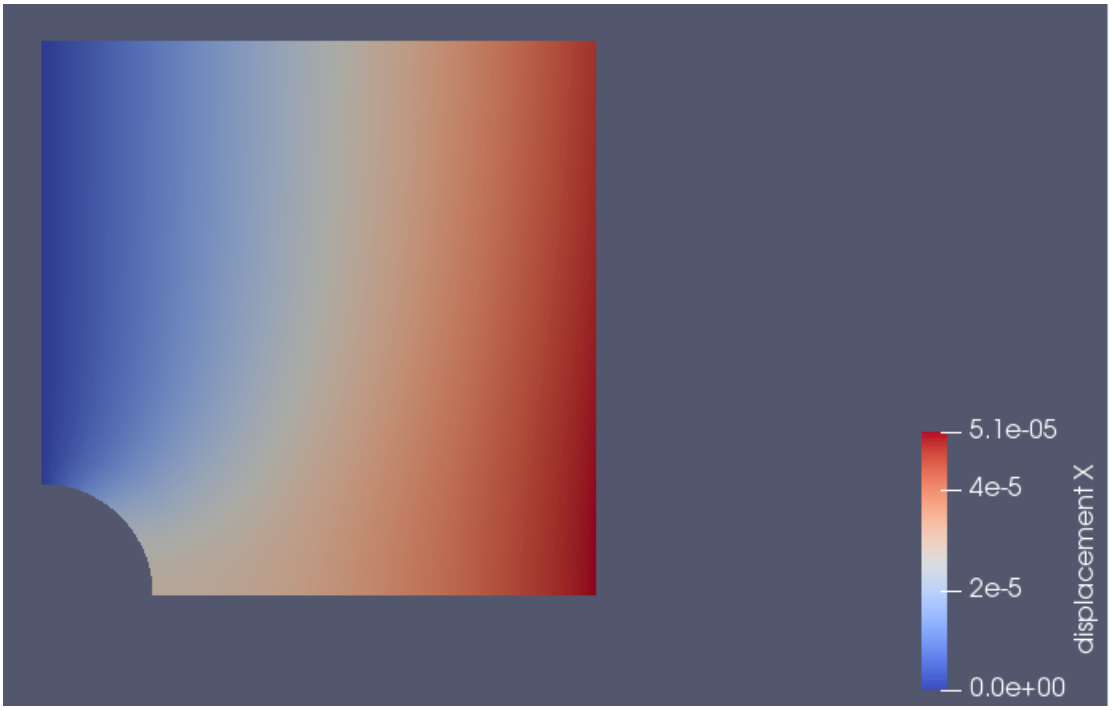


Figura 14 – Deslocamentos em X_1

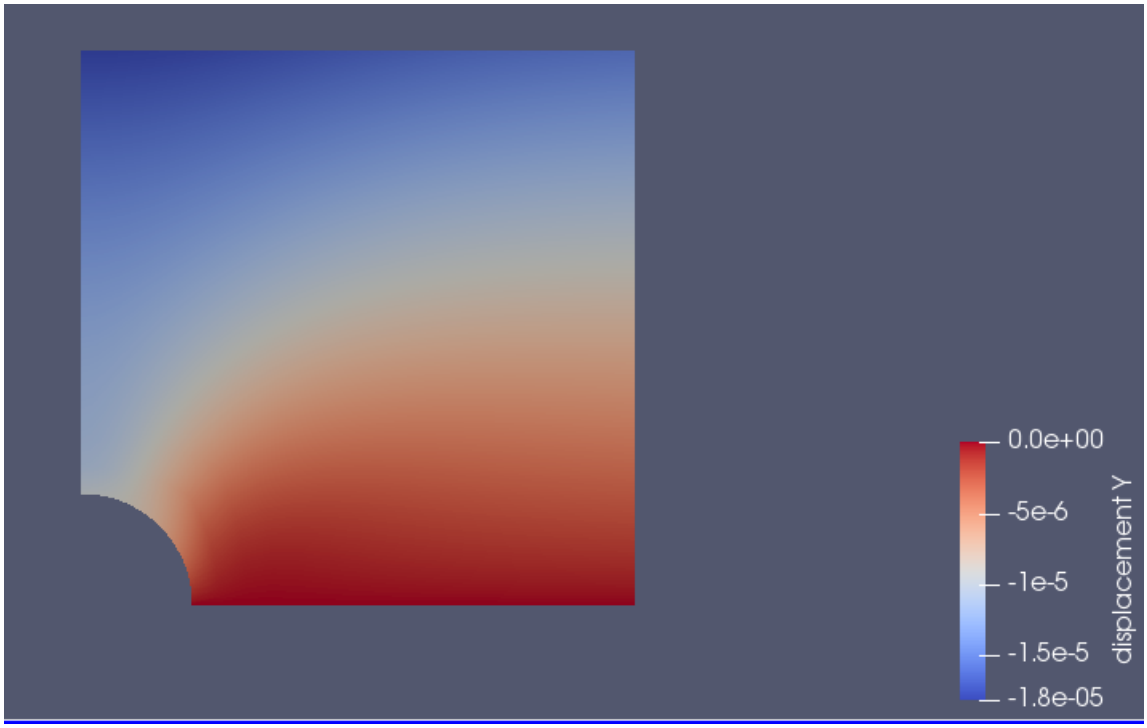


Figura 15 – Deslocamentos em X_2

Usando o programa Octave, comparou-se os resultados encontrados numericamente com os resultados fornecidos pela solução analítica, obtendo a Fig. 16.

O erro RMS foi calculado da seguinte forma

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}[u_{numérico} - u_{analítico}]^2} \tag{5.5}$$

E calculou-se a média dos valores absolutos de deslocamentos para ter-se uma ideia de grandeza.

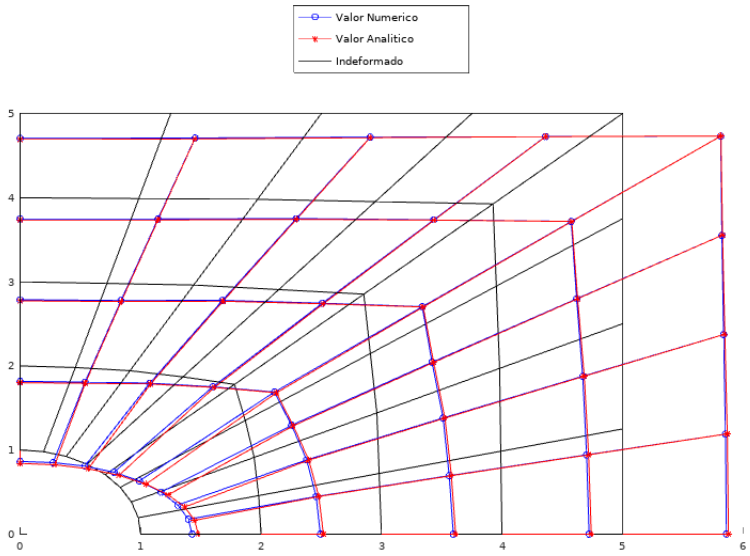


Figura 16 – Resultados Numéricos comparados com os analíticos

É perceptível uma boa concordância entre os valores obtidos pelo deal.II e os valores da solução.

Isto pode ser confirmado pelos seguintes parâmetros na Tab. 3

Tabela 3 – Parâmetros para placa quadrada com furo circular

	Funções de Forma de 1ª Ordem	Funções de Forma de 2ª Ordem
Erro RMS	7e-07	2e-07
Deslocamento médio	2,9e-05	2,9e-05
Erro RMS/Desloc. médio	2,4%	0,7%

Roda-se novamente o programa com uma malha refinada, representada na Fig 17, com 512 elementos, 1122 nós para funções de ordem de 1º grau e 2145 nós para funções de ordem de 2º grau.

E o resultado está na Fig 18

Percebe-se então uma diminuição do erro comparando com o analítico na Tabela 4.

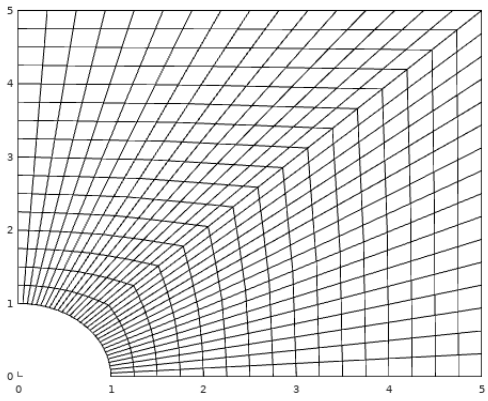


Figura 17 – Malha refinada

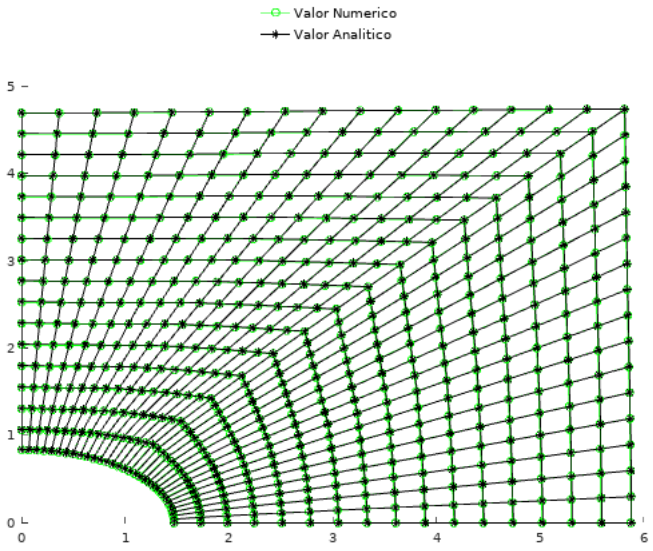


Figura 18 – Resultados numéricos × analíticos em malha mais refinada

Tabela 4 – Parâmetros para placa quadrada com furo circular

	1ª Ordem	2ª Ordem
Erro RMS	1,4e-7	9,4e-9
Deslocamento médio	2,8e-05	2,7e-05
Erro RMS/Deslocamento médio	0,5%	0,03%

Com os gradientes das funções de forma, é possível também calcular as tensões. Os resultados encontrados estão nas Fig. 19, Fig. 20 e Fig. 21

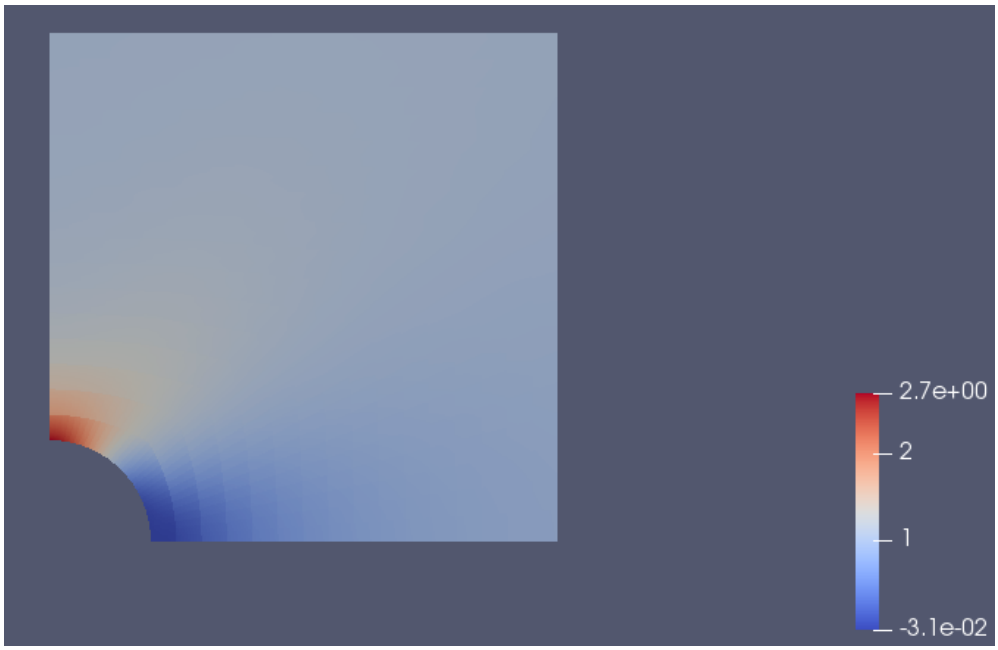


Figura 19 – σ_1

Para verificar os erros relativos às tensões, usou-se a densidade de energia de deformação, dada pela Eq.(5.6)

$$DED = \frac{1}{2E}(\sigma_{11}^2 + \sigma_{22}^2) - \frac{\nu}{E}(\sigma_{11}\sigma_{22}) + \frac{1}{2\mu}(\sigma_{xy}^2) \tag{5.6}$$

Os parâmetros de erros estão em Tab. 5 e Tab. 6

Figura 20 – σ_2

Figura 21 – σ_{12}

Tabela 5 – Erros relativos a densidade de energia de deformação com funções de forma de 2ª ordem

	1ª malha	2ª malha
Erro RMS	3,7e-07	1,8e-07
Valor médio	5e-06	5e-06
Erro RMS/Valor médio	7,4%	3,6%

Tabela 6 – Erros relativos a densidade de energia de deformação com funções de forma de 1ª ordem

	1ª malha	2ª malha
Erro RMS	7,4e-07	4,9e-07
Valor médio	5e-06	5e-06
Erro RMS/Valor médio	14%	9,8%

5.3 Cilindro Pressurizado

O 2º exemplo possui as seguintes especificações: um cilindro com um furo no centro e uma carga uniforme normal à sua superfície aplicada. A solução analítica para os deslocamentos radiais é dada por:

$$u_r = \frac{(1+\nu)pa^2}{E(b^2-a^2)}\left[(1-2\nu)r + \frac{b^2}{r}\right] \tag{5.7}$$

onde $\nu = 0,3$, a pressão $p = 100$, $b = 0,1$, $a = 0,05$ e $E = 1000$.

Devido à simetria do problema, é possível modelar somente um quarto do colindro. Pela Fig. 22, os resultados mostram boa convergência, o que é confirmado pelos dados relativos ao erro na Tab. 7

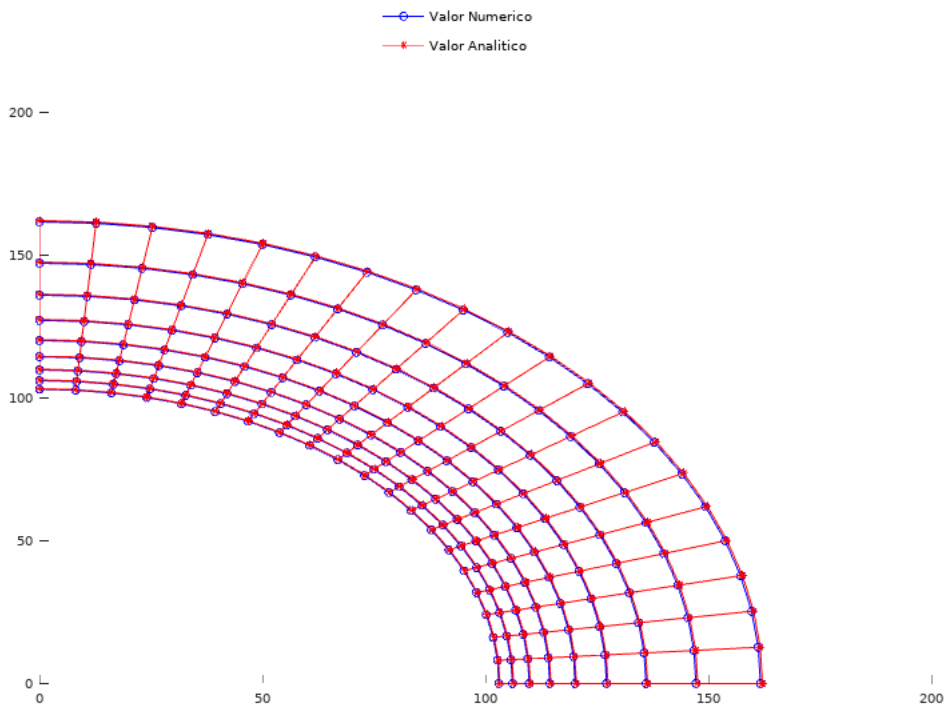


Figura 22 – Comparação analítca numérica

Tabela 7 – Parâmetros de erro do cilindro

	1ª Ordem	2ª Ordem
Erro RMS	1,5e-05	1,2e-06
Deslocamento médio	7e-3	7e-3
Erro RMS/Deslocamento médio	0,2%	0,017%

5.4 Placa simples de material compósito

Foi simulada uma placa simples de um material compósito laminado. Esta possui dimensões de 1×1. A lâmina considerada possui as propriedades, mostradas na Tab. 8, e foi aplicada uma força de 1000 na direção x. Devido à simplicidade do problema, os valores de deformação são constantes em todo o domínio.

Tabela 8 – Propriedades da lâmina

Propriedade	Valor
E_1	148e9
E_2	10.5e9
G_{12}	5,61e9
ν_{12x}	0,3
Espessura	0,002

Primeiramente, considerou-se um laminado $[0°/90°]_s$. Os resultados encontrados estão em Tabela 9.

Além disso, realizou-se as simulações para um laminado $[45°/135°]_s$. Os resultados

Tabela 9 – Resultados para laminado [0°/90°/90°/0°]

Grandezas	Valor Analítico	Valor Numérico	Erro percentual
Deformação em X	1,5697e-5	1,5697e-5	0
Deformação em Y	-6,23917e-7	-6,23916e-07	0,0002%
Deformação cisalhante	1,307e-22	2,16205e-19	
Tensão em X nas placas 0°	2,3361e6	2,336e6	0,004%
Tensão em Y nas placas 0°	43169,9	43170	0,0002%
Tensão cisalhante nas placas em 0°	7,33225e-13	6,065e-10	
Tensão em X nas placas 90°	1,63899e5	1,639e+05	0,0006%
Tensão em Y nas placas 90°	-43169,9	-43170	0,0002%
Tensão cisalhante nas placas em 90°	-7,33225e-13	6,065e-10	

encontrados estão na Tabela 10.

Tabela 10 – Resultados para laminado [45°/135°/135°/45°]

Grandezas	Valor Analítico	Valor Numérico	Erro percentual
Deformação em X	6,32406e-5	6,32463e-5	0,01%
Deformação em Y	-4,81676e-5	-4,81731e-05	0,01%
Deformação cisalhante	-2,35159e-20	2,71051e-20	
Tensão em X nas placas 45°	1,25e6	1,25e6	0%
Tensão em Y nas placas 45°	-1,66658e-10	-5,344e1	
Tensão cisalhante nas placas em 45°	5,21465e5	5,215e5	0,007%
Tensão em X nas placas 135°	1,25e6	1,25e6	0
Tensão em Y nas placas 135° °	-9,90288e-11	-5,344e1	
Tensão cisalhante nas placas em 135°	-5,21465e-5	-5,215e-5	0,007%

É importante notar que apesar dos erros serem muito grandes nas deformações e tensões cisalhantes de Tab. 9 e nas deformações cisalhantes e tensões em y de Tab. 10, isto se deve ao fato de que estes valores são muito pequenos em relação aos outros, ordem de grandeza inferior a 0,01%.

5.5 Placa quadrada com furo circular

Considere uma placa com furo circular de um laminado simétrico $[0/90]_s$,cujas propriedades das lâminas estão na Tabela 11. As cargas são uma tração de $1e5N/m$ longitudinal e uma compressão de $9e4N/m$ transversal, conforme Fig. 23

Tabela 11 – Propriedades do material

Propriedade	Valor
E_1(GPa)	14
E_2(GPa)	3,5
G_{12}(GPa)	4,2
ν_{12}	0,04286

A malha usada está em Fig. 24, com 512 elementos e 4352 nós.

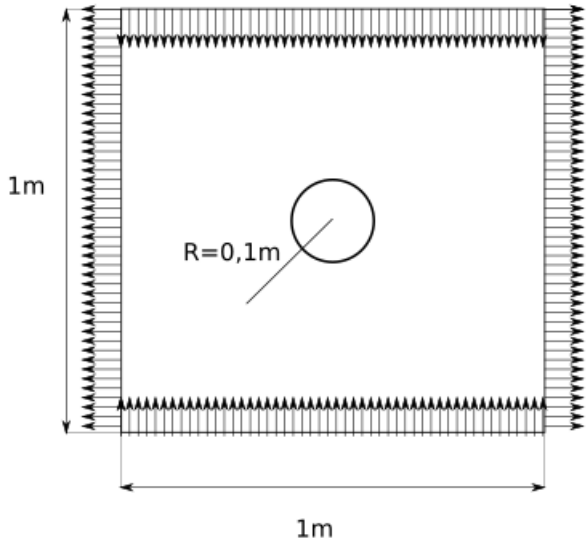


Figura 23 – Placa quadrada com furo circular de material laminado simétrico

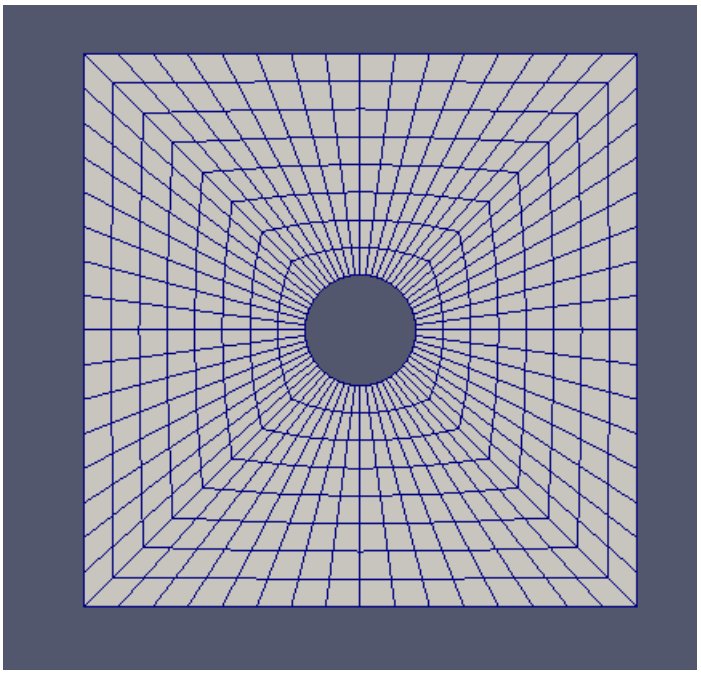


Figura 24 – Malha para placa quadrada com furo circular

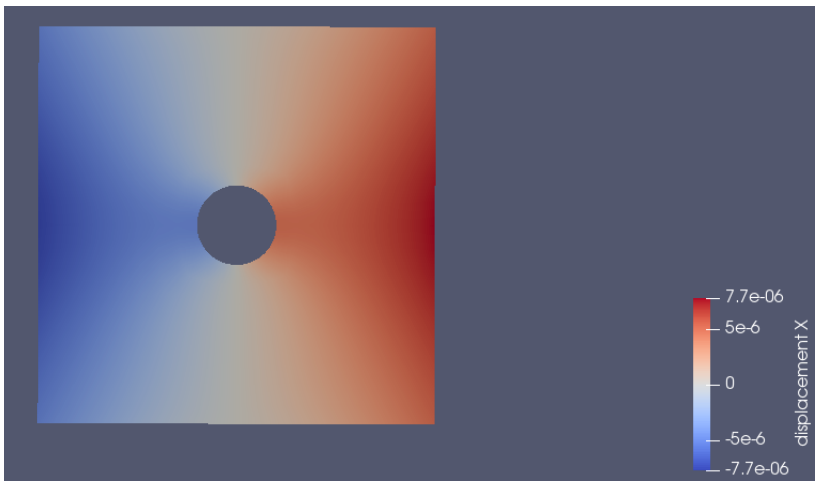


Figura 25 – Deslocamentos em X_1 para placa quadrada com furo circular

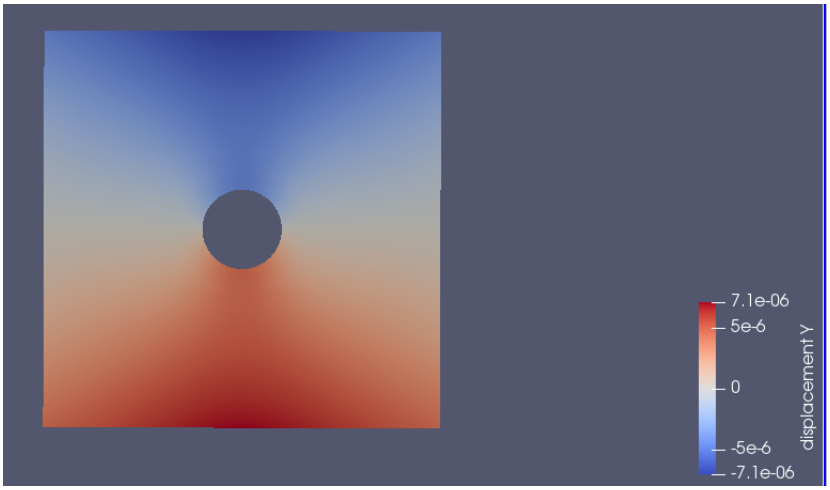


Figura 26 – Deslocamentos em X_2 para placa quadrada com furo circular

Os deslocamentos estão em Fig. 25 e Fig. 26

Com as deformações, é possível então calcular as tensões para cada lâmina, conforme mostrado em Fig. 27, Fig. 28, Fig. 29, Fig. 30, Fig. 31, e Fig. 32,

Comparando-se os deslocamentos e deformações encontradas nos pontos internos com os resultados obtidos pelo programa utilizado em (ALBUQUERQUE, 2001), obteve-se os valores em Tab. 12.

Tabela 12 – Diferença RMS e deformação média

	Valor
Diferença RMS de deslocamento	1,03e-7
Deslocamento médio	5,16e-6
Diferença RMS/Desloc. médio	2%
Diferença RMS de Deformação	2,8e-7
Deformação média	8,9e-6
Diferença RMS/Deform. média	3%

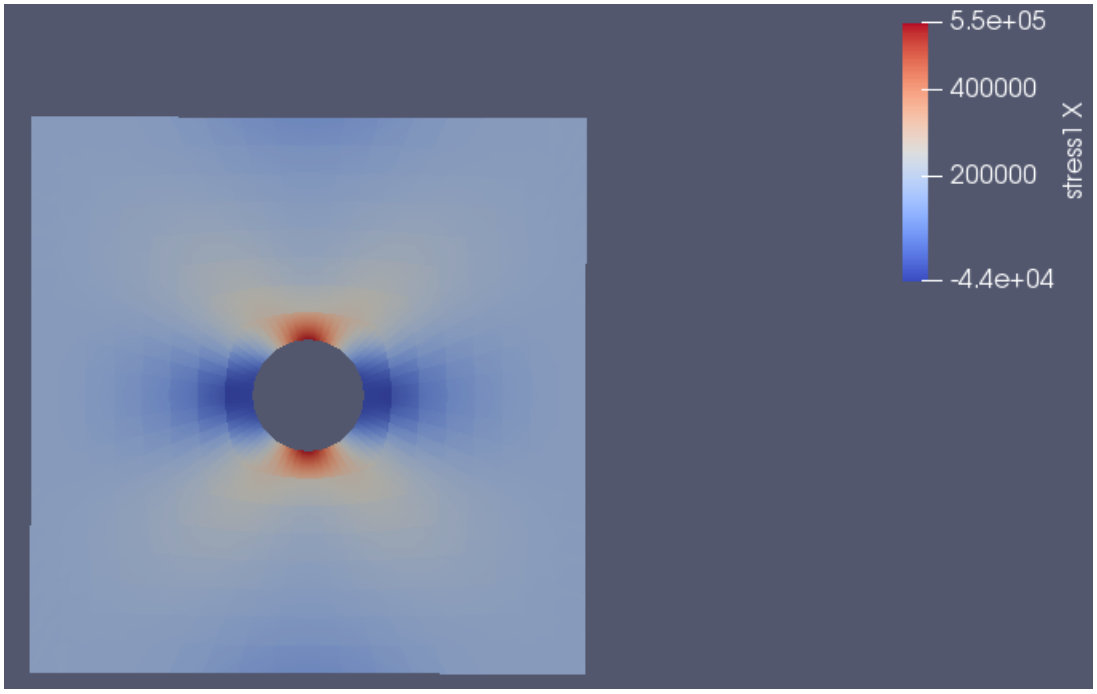


Figura 27 – Tensões σ_1 lâmina 0

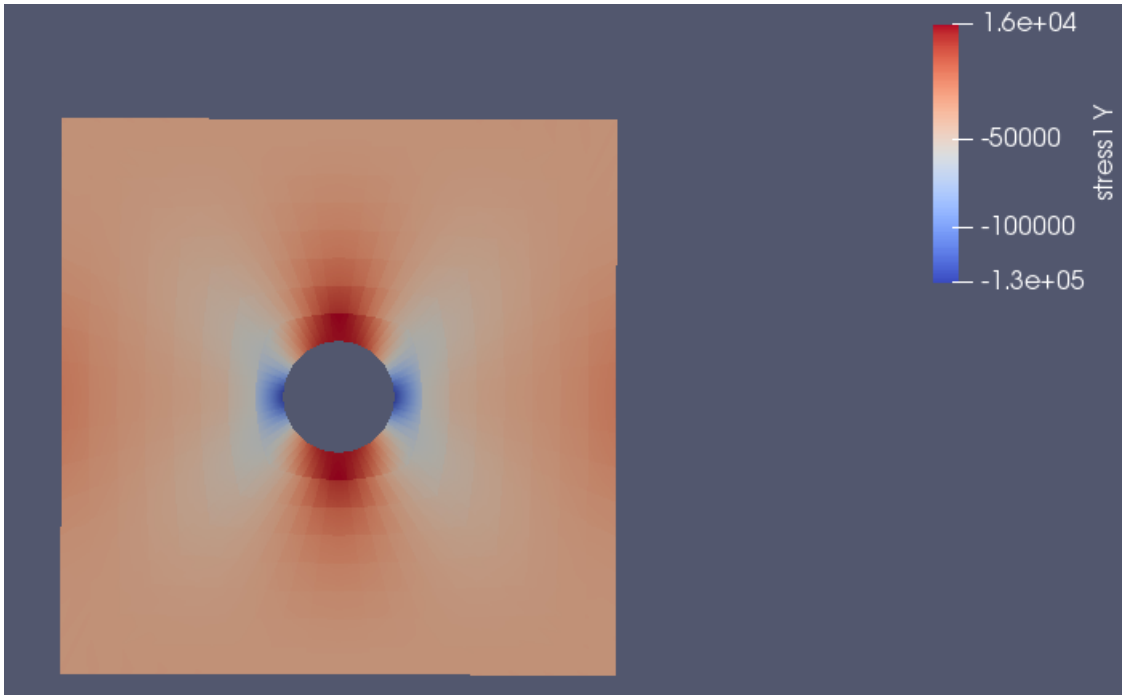


Figura 28 – Tensões σ_2 lâmina 0

x

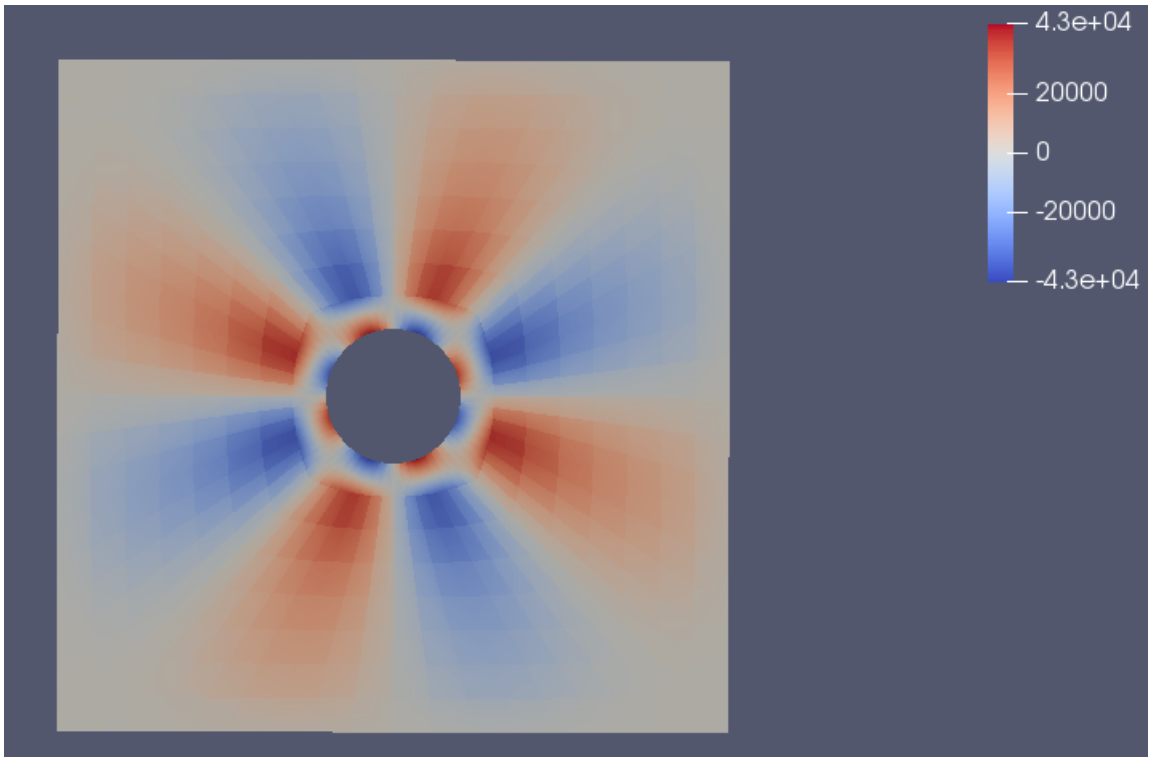


Figura 29 – Tensões σ_{12} na lâmina 0

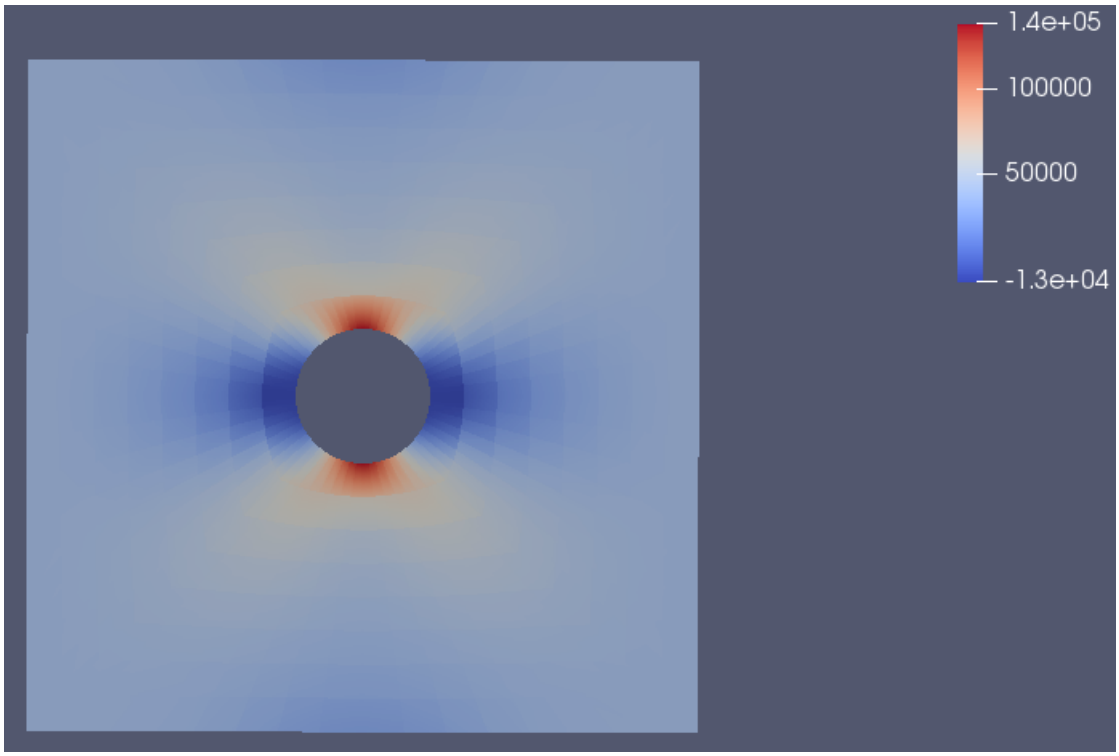


Figura 30 – Tensões σ_1 na lâmina 90

x

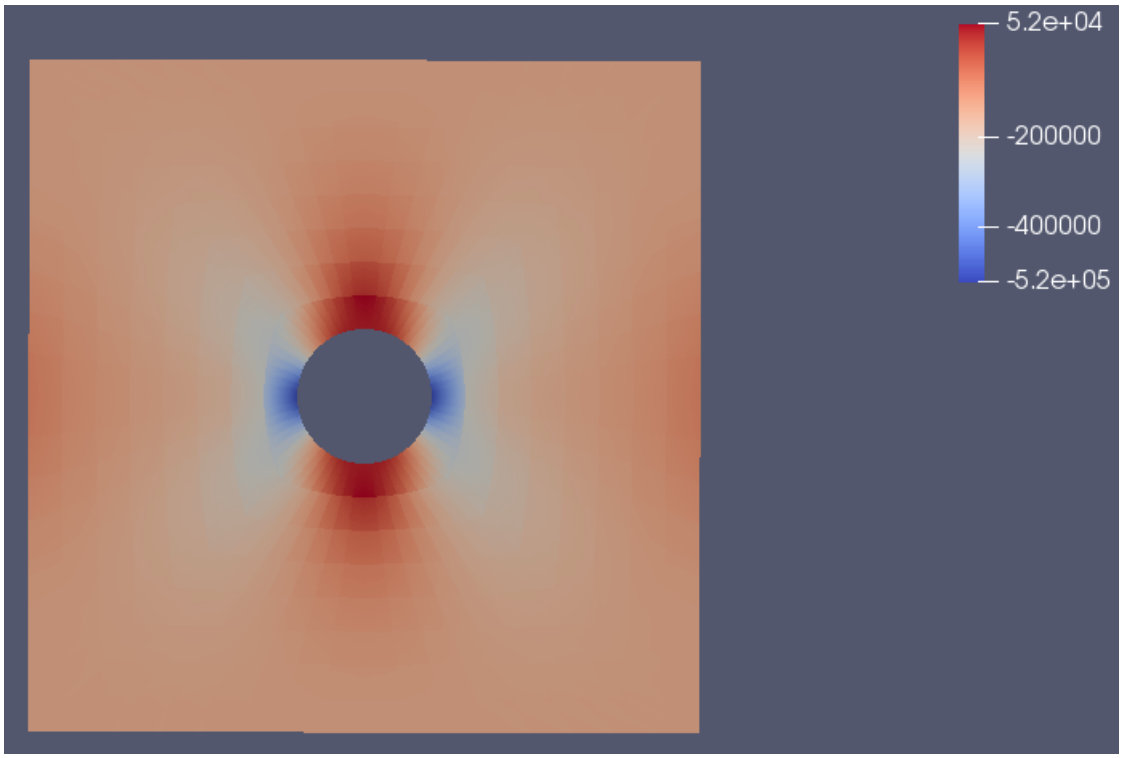


Figura 31 – Tensões σ_2 na lâmina 90

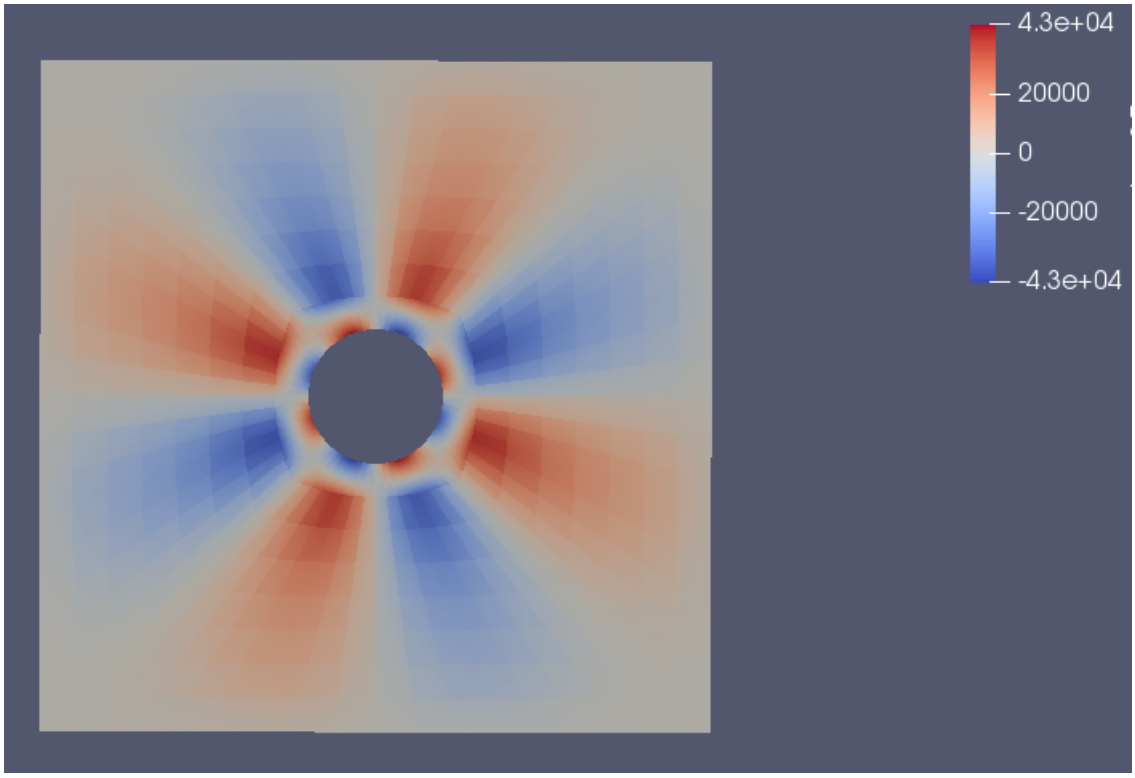


Figura 32 – Tensões σ_{12} na lâmina 90

5.6 Placa com furo circular

Neste exemplo a geometria não é mais um simples quadrado, mas possui a geometria e condições de contorno indicadas na Fig 33.

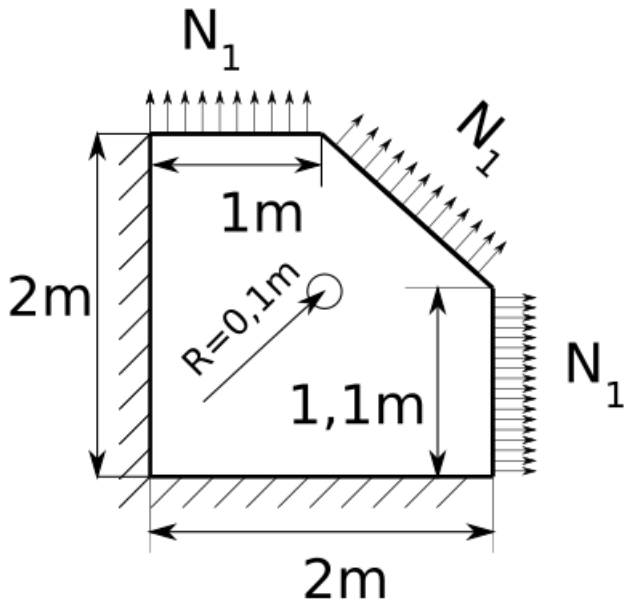


Figura 33 – Geometria e tensões aplicadas

Neste caso N_1 vale 100MN/m e N_2 vale 50MN/m

Além disso, considera-se um laminado simétrico $[20/50]_s$. Com lâminas de espessura 0,002 m e com propriedades em Tab. 13:

Tabela 13 – Propriedades mecânicas do material

Propriedade	Valor
c E_1(GPa)	204
E_2(GPa)	18,5
G_{12}(GPa)	5,59
ν_{12}	0,23

A malha usada está em Fig. 34, com 1472 elementos e 12000 nós.

Os deslocamentos encontrados estão em Fig. 35 e Fig. 36

A partir das deformações é então possível calcular as tensões em cada lâmina, mostradas em Fig. 37, Fig. 38, Fig. 39, Fig. 40, Fig. 41, Fig. 42

Comparando-se os deslocamentos e deformações encontradas nos pontos internos com os resultados obtidos pelo método dos elementos de contorno, obteve-se os valores em Tab. 14

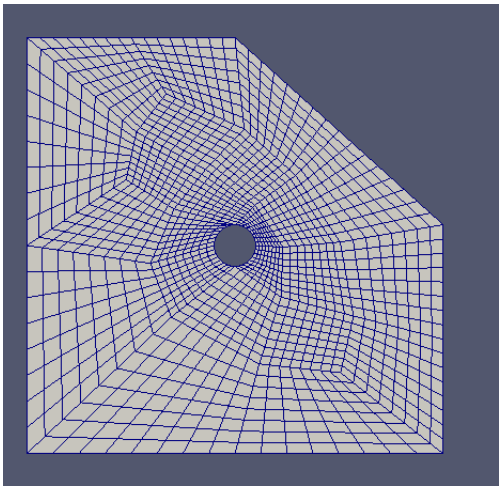


Figura 34 – Malha utilizada no caso 2

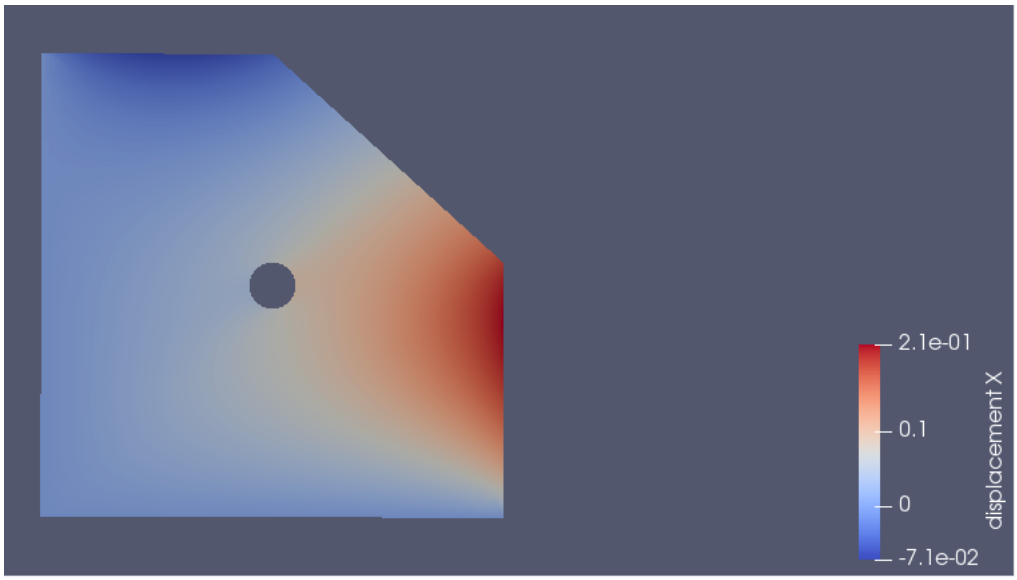


Figura 35 – Deslocamentos em X_1 do caso 2

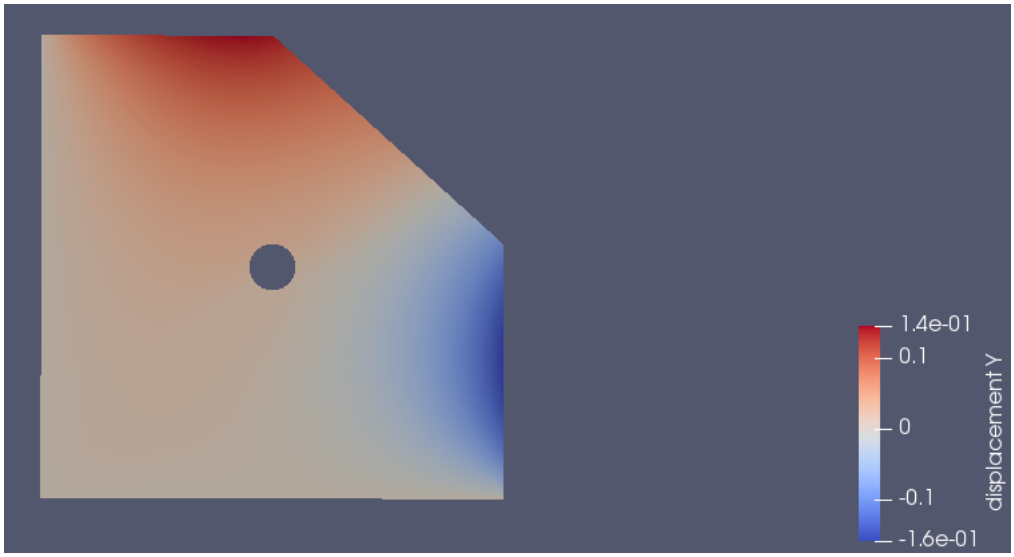


Figura 36 – Deslocamentos em Y_2 do caso 2

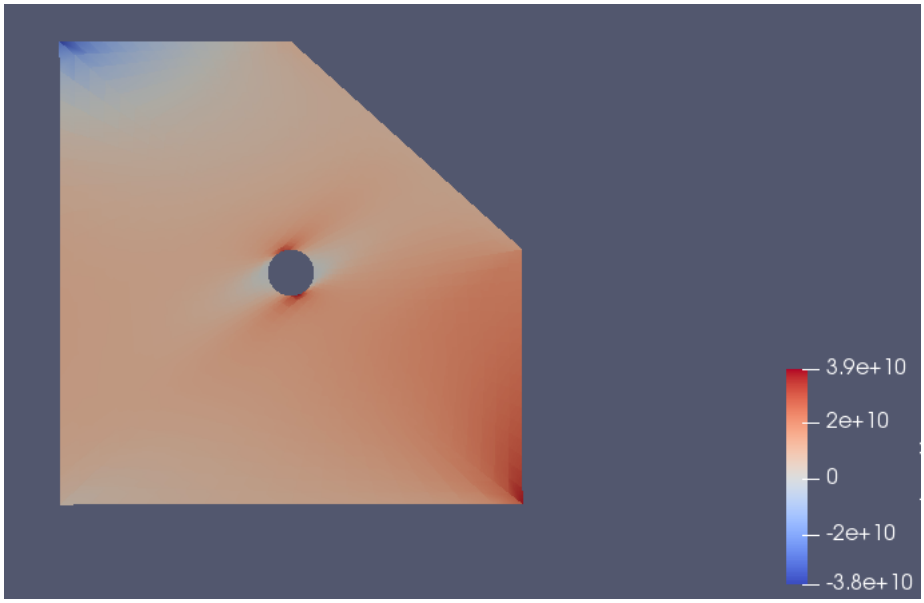


Figura 37 – Tensões σ_1 na lâmina de 20

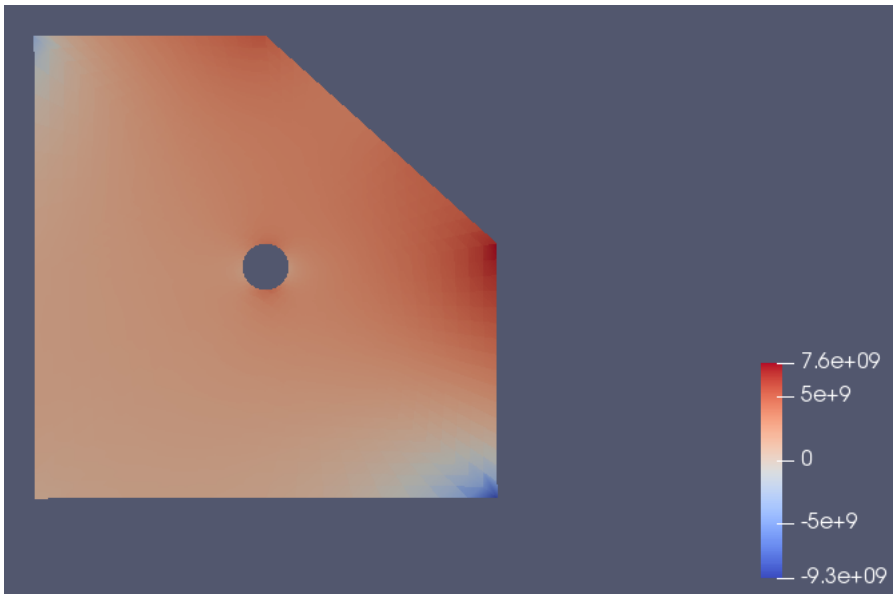


Figura 38 – Tensões σ_2 na lâmina de 50

Tabela 14 – Comparação de resultados

	Valor
Diferença dos RMS deslocamentos	0,0017
Deslocamento médio	0,044
Diferença RMS/Deslocamento médio	4%
Diferença RMS da deformação	0,0014
Deformação média	0,084
Diferença RMS/Deformação média	1,6%

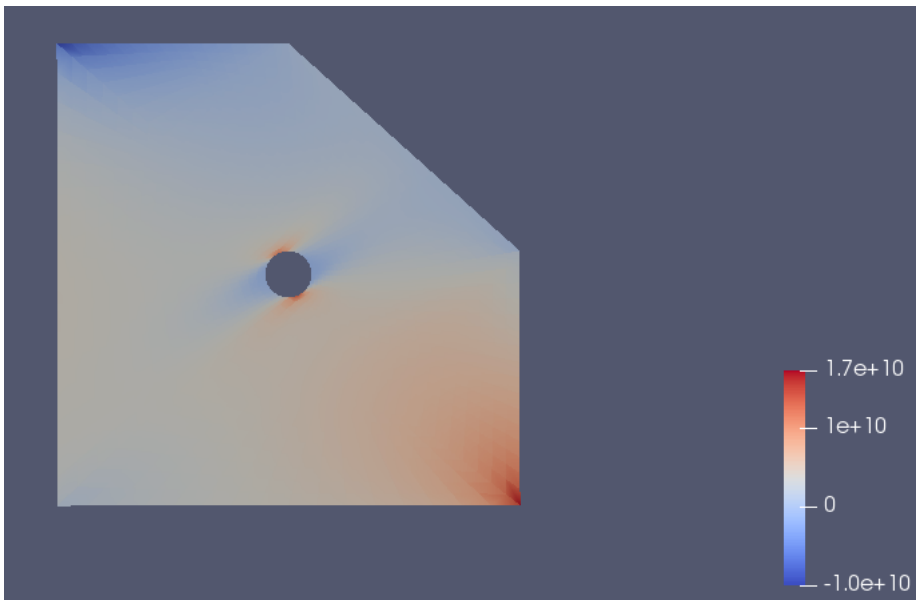


Figura 39 – Tensões σ_{12} na lâmina de 20

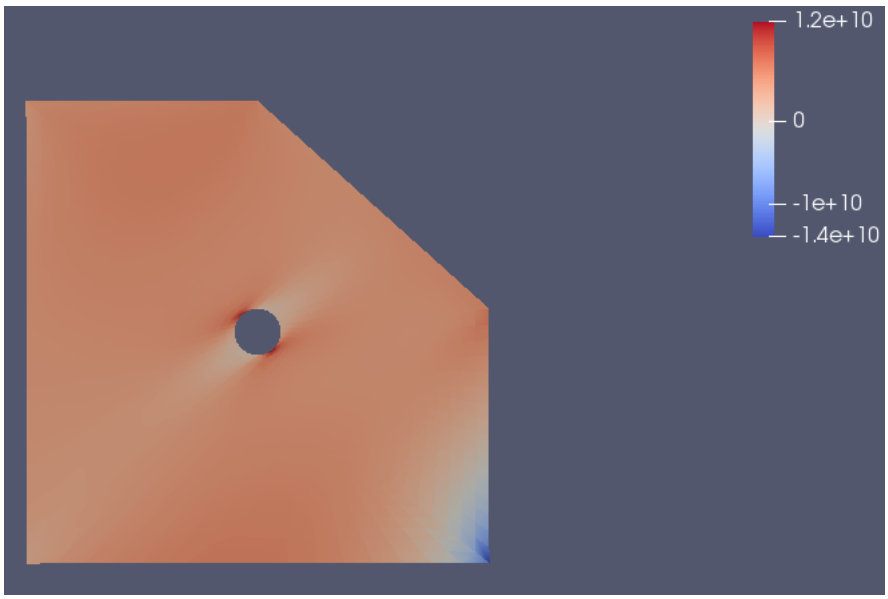


Figura 40 – Tensões σ_1 na lâmina de 50

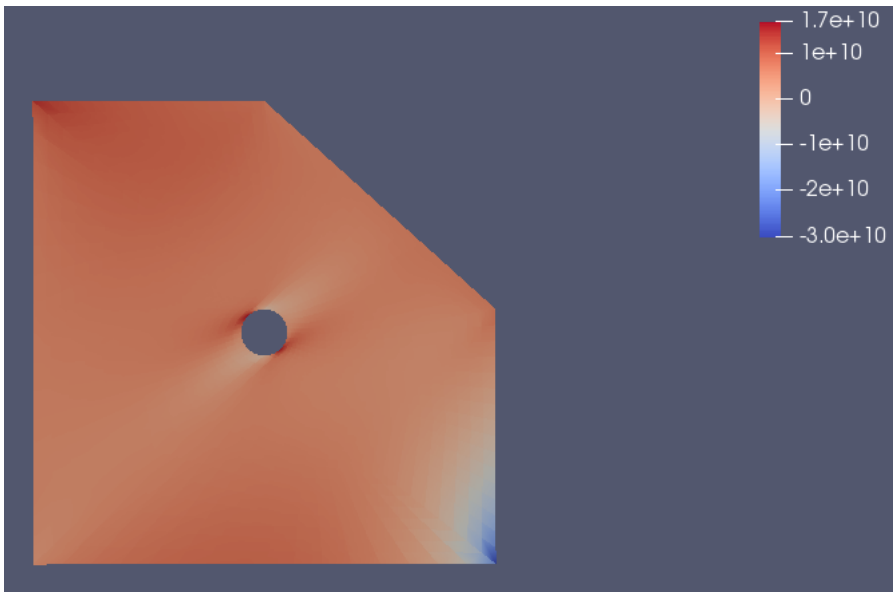


Figura 41 – Tensões σ_2 na lâmina de 50

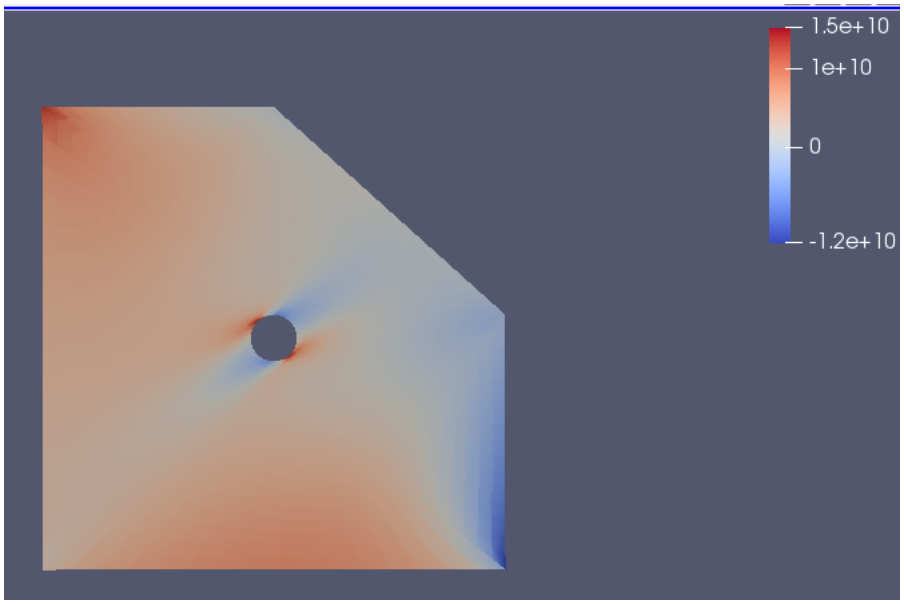


Figura 42 – Tensões σ_{12} na lâmina de 50

6 Conclusões

Neste trabalho foi feita uma implementação do método dos elementos finitos para a análise estrutural de materiais compósitos laminados usando o programa de código aberto deal.ii. O programa deal.ii trata-se de uma biblioteca de elementos finitos desenvolvida em linguagem C++ na qual encontra-se implementada as principais formulações do método dos elementos finitos para os mais variados problemas. Entretanto, não havia na biblioteca uma formulação para a análise de materiais anisotrópicos, como é o caso dos materiais compósitos. Neste sentido, partiu-se da implementação da formulação isotrópica e, através da troca da relação constitutiva tensão - deformação, foi implementada a matriz de rigidez para materiais compósitos laminados (materiais anisotrópicos). Para validar a implementação, foram avaliados vários problemas isotrópicos e anisotrópicos que possuem soluções analíticas e também problemas anisotrópicos que não possuem soluções analíticas. Nos casos dos problemas que não possuem soluções analíticas, deslocamentos e deformações foram comparados com soluções numéricas obtidas com o método dos elementos de contorno, caso contrário, as comparações foram feitas com as soluções analíticas. Os resultados obtidos neste trabalho mostraram boa concordância tanto com o método dos elementos de contorno como com as soluções analíticas.

Foram explorados alguns recursos presentes na biblioteca deal.ii como a leitura de malhas geradas pelo programa de código aberto GMSH, e o pós-processamento no Paraview, também um programa de código aberto. Este trabalhou atingiu seu objetivo principal que era fazer uma análise de problemas de materiais compósitos laminados usando o programa deal.ii. Apesar do programa deal.ii não ter uma interface gráfica e exigir um conhecimento intermediário do método dos elementos finitos e de programação em linguagem C++, ao contrário de vários outros programas de elementos finitos de código aberto ou mesmo comerciais que são mais focados na fácil utilização pelo usuário sem experiência com a formulação do método dos elementos finitos e com programação, o programa mostrou-se uma ferramenta muito útil para os desenvolvedores de métodos numéricos. Seus vários módulos possuem documentação farta, tanto na forma de texto como de vídeos, o que acelera bastante o aprendizado do usuário desenvolvedor. Além disso, existem fóruns online de discussões de problemas do programa, bastante ativos.

6.1 Trabalhos futuros

Este primeiro trabalho explorou apenas algumas das muitas características do programa deal.ii. Como trabalhos futuros, propõe-se:

- Explorar as ferramentas de malhas adaptativas do deal.ii na análise de problemas de materiais compósitos. Uma das principais características do programa é a facilidade do refinamento adaptativo das malhas.
- Análise de problemas de materiais compósitos de larga escala usando computação paralela em grupos (clusters) de computadores. Conforme descrito na documentação do programa deal.ii, já foram analisados problemas com mais de 1 bilhão de graus de liberdade em um computador com 1024 processadores.
- Estender a análise de problemas de materiais compósitos para formulações não lineares como problemas com grandes deformações ou de mecânica do dano.

Referências

AGARWAL, B. D.; BROUTMAN, L. J.; CHANDRASHEKHARA, K. *Analyzis and Performance of Fiber Composites*. [S.l.: s.n.], 2006. Citado 6 vezes nas páginas v, vi, 1, 2, 7 e 8.

ALBUQUERQUE, E. L. de. Análise de problemas dinâmicos em materiais anisotrópicos usando o método dos elementos de contorno. Universidade de Campinas, 2001. Citado 2 vezes nas páginas 24 e 34.

BANGERTH, W. *deal.org*. 2019. <deal.org>. Nenhuma citação no texto.

BANGERTH, W.; HARTMANN, R.; KANSCHAT, G. deal.II – a general purpose object oriented finite element library. Texas, United States, p. 27, 2007. Citado 2 vezes nas páginas v e 3.

FAN, L.; COOMBS, W.; AUGARDE, C. The point collocation method with a local maximum entropy approach. *Computers and Structures*, v. 201, p. 1–14, 02 2018. Citado na página 24.

FISH, J.; BELYTSCHKO, T. *A First Course in Finite Elements*. [S.l.: s.n.], 2007. Citado 3 vezes nas páginas v, 2 e 15.

GARIKIPATI, W. The finite element method for problems in physics. Michigan, United States, 2013. Citado na página 21.

LEKHNITSKII, S. G. *Theory of Elasticity of an Anisotropic Body*. [S.l.: s.n.], 1981. Citado na página 8.

TIMOSHENKO, S. *Theory of Elasticity*. [S.l.: s.n.], 1951. Citado na página 24.

7 Anexos

Anexos

.1 Anexo 1

Código utilizado no último exemplo da seção de resultados

Main

```
//Include files
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>

#include "FEM3.h"
//#include "writeSolutions.h"

using namespace dealii;

//The main program, using the FEM class
int main (){
  try{
    deallog.depth_console (0);

                const int dimension = 2;

                //NOTE: This is where you define the number of elements in the mesh
                std::vector<unsigned int> num_of_elems(dimension);
                num_of_elems[0] = 5;
                num_of_elems[1] = 5;
                //              num_of_elems[2] = 10; //For example, a 10 x 10 x 10
                    ↪ element mesh

          FEM<dimension> problemObject;
          problemObject.generate_mesh(num_of_elems);
          problemObject.setup_system();
          problemObject.assemble_system();
          problemObject.solve();
          problemObject.compute_stress();
          problemObject.output_results();

    //write solutions to h5 file
  //  char tag[21];
 //   sprintf(tag, "CA3");
//    writeSolutionsToFileCA3(problemObject.D, tag);

  }
  catch (std::exception &exc){
    std::cerr << std::endl << std::endl
              << "----------------------------------------------------"
              << std::endl;
    std::cerr << "Exception␣on␣processing:␣" << std::endl
              << exc.what() << std::endl
              << "Aborting!" << std::endl
              << "----------------------------------------------------"
              << std::endl;

    return 1;
  }
  catch (...){
    std::cerr << std::endl << std::endl
              << "----------------------------------------------------"
              << std::endl;
    std::cerr << "Unknown␣exception!" << std::endl
              << "Aborting!" << std::endl
              << "----------------------------------------------------"
              << std::endl;
    return 1;
  }

  return 0;
}
```

FEM.h

```
//Include files
//Data structures and solvers
#include <deal.II/base/quadrature_lib.h>
#include <deal.II/base/function.h>
#include <deal.II/base/logstream.h>
#include <deal.II/base/tensor_function.h>
#include <deal.II/lac/vector.h>
#include <deal.II/lac/full_matrix.h>
#include <deal.II/lac/sparse_matrix.h>
#include <deal.II/lac/sparse_direct.h>
#include <deal.II/numerics/vector_tools.h>
#include <deal.II/numerics/matrix_tools.h>
#include <deal.II/numerics/data_out.h>
#include <deal.II/base/symmetric_tensor.h>
//Mesh related classes
#include <deal.II/grid/tria.h>
#include <deal.II/grid/grid_generator.h>
#include <deal.II/grid/tria_accessor.h>
#include <deal.II/grid/tria_iterator.h>
#include <deal.II/grid/tria_boundary_lib.h>
#include <deal.II/grid/grid_tools.h>
#include <deal.II/dofs/dof_handler.h>
#include <deal.II/dofs/dof_accessor.h>
#include <deal.II/dofs/dof_tools.h>
#include <deal.II/grid/grid_in.h>
#include <deal.II/grid/manifold_lib.h>
#include <deal.II/grid/grid_out.h>
#include <deal.II/lac/constraint_matrix.h>
//Finite element implementation classes
#include <deal.II/fe/fe_system.h>
#include <deal.II/fe/fe_values.h>
#include <deal.II/fe/fe_q.h>

#include <deal.II/lac/solver_cg.h>
#include <deal.II/lac/precondition.h>


//Standard C++ libraries
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <math.h>
using namespace dealii;

//Define the order of the basis functions (Lagrange polynomials)
//and the order of the quadrature rule globally
const unsigned int order = 2;
const unsigned int quadRule = 2;

template <int dim>
class FEM
{
 public:
  //Class functions
  FEM (); // Class constructor
  ~FEM(); //Class destructor

  //Function to calculate components of the elasticity tensor
  double C(unsigned int i,unsigned int j,unsigned int k,unsigned int l);

  //Solution steps
  void generate_mesh(std::vector<unsigned int> numberOfElements);
  void define_boundary_conds();
  void setup_system();
  void assemble_system();
  void solve();
  void compute_stress();
  void output_results();
```

```
  //Class objects

  Triangulation<dim> triangulation; //mesh
  FESystem<dim>      fe;            //FE element
  DoFHandler<dim>    dof_handler;   //Connectivity matrices

  //NEW - deal.II quadrature
  QGauss<dim>   quadrature_formula;      //Quadrature
  QGauss<dim-1> face_quadrature_formula; //Face Quadrature

  //Data structures
  SparsityPattern      sparsity_pattern; //Sparse matrix pattern
  SparseMatrix<double> K;                //Global stiffness matrix - Sparse matrix - used
      ↪  in the solver
  ConstraintMatrix     hanging_node_constraints;
  Vector<double>       D, F;             //Global vectors - Solution vector (D) and
      ↪ Global force vector (F)

  Table<2,double>               dofLocation;     //Table of the coordinates of dofs by
      ↪ global dof number
  std::map<unsigned int,double> boundary_values; //Map of dirichlet boundary conditions

  //solution name array
  std::vector<std::string> nodal_solution_names;
  std::vector<DataComponentInterpretation::DataComponentInterpretation>
      ↪ nodal_data_component_interpretation;
};

template <int dim>
class StrainPostprocessor : public DataPostprocessorTensor<dim>
{
public:
  StrainPostprocessor ()
    :
    DataPostprocessorTensor<dim> ("strain",
                                  update_gradients)
  {}
  virtual
  void
  evaluate_vector_field(
    const DataPostprocessorInputs::Vector<dim> &input_data,
    std::vector<Vector<double> >               &computed_quantities) const
  {
    AssertDimension (input_data.solution_gradients.size(),
                     computed_quantities.size());
    for (unsigned int p=0; p<input_data.solution_gradients.size(); ++p)
      {
        AssertDimension (computed_quantities[p].size(),
                         (Tensor<2,dim>::n_independent_components));
        for (unsigned int d=0; d<dim; ++d)
          for (unsigned int e=0; e<dim; ++e)
            computed_quantities[p][Tensor<2,dim>::component_to_unrolled_index(
                ↪ TableIndices<2>(d,e))]
              = (input_data.solution_gradients[p][d][e]
                 +
                 input_data.solution_gradients[p][e][d]) / 2;
      }
  }
};

// Class constructor for a vector field
template <int dim>
FEM<dim>::FEM ()
:
fe (FE_Q<dim>(order), dim),
  dof_handler (triangulation),
  quadrature_formula(quadRule),
  face_quadrature_formula(quadRule)
{

  //Nodal Solution names - this is for writing the output file
  for (unsigned int i=0; i<dim; ++i){
    nodal_solution_names.push_back("u");
    nodal_data_component_interpretation.push_back(DataComponentInterpretation::
```

```
        ↪ component_is_part_of_vector);
  }
}

//Class destructor
template <int dim>
FEM<dim>::~FEM (){dof_handler.clear ();}

//Function to calculate the components of the 4th order elasticity tensor
template <int dim>
double FEM<dim>::C(unsigned int i,unsigned int j,unsigned int k,unsigned int l){

  double C11,C12,C13,C16,C26,C22,C23,C33,C44,C55,C66,lambda,mu;
  C11=8.4867*pow(10,8);
  C12=3.0066*pow(10,8);
  C13=0;
  C16=3.6529*pow(10,8);
  C22=4.0700*pow(10,8);
  C23=0;
  C26=2.4145*pow(10,8);
  C33=0;
  C44=0;
  C55=0;
  C66=3.1118*pow(10,8);
  lambda=3;
  mu=4.3333;

  if (i==0 && j==0 && k==0 && l==0)
    {
      return (C11);
    }
  else if ((i==0 && j==0 && k==1 && l==1) || (i==1 && j==1 && k==0 && l==0))
    {
      return (C12);
    }
  else if ((i==0 && j==0 && k==0 && l==1) || (i==0 && j==0 && k==1 && l==0) || (i==0 && j
      ↪ ==1 && k==0 && l==0) || (i==1 && j==0 && k==0 && l==0))
    {
      return (C16);
    }
  else if ((i==1 && j==1 && k==0 && l==1) || (i==1 && j==1 && k==1 && l==0) || (i==0 && j
      ↪ ==1 && k==1 && l==1) || (i==1 && j==0 && k==1 && l==1))
    {
      return (C26);
    }
  else if ((i==0 && j==0 && k==2 && l==2) || (i==2 && j==2 && k==0 && l==0))
    {
      return (C13);
    }
  else if ((i==1 && j==1 && k==1 && l==1))
    {
      return (C22);
    }
  else if ((i==2 && j==2 && k==1 && l==1) || (i==1 && j==1 && k==2 && l==2))
    {
      return (C23);
    }
  else if ((i==2 && j==2 && k==2 && l==2))
    {
      return (C33);
    }
  else if ((i==1 && j==2 && k==1 && l==2) || (i==1 && j==2 && k==2 && l==1) || (i==2 && j
      ↪ ==1 && k==1 && l==2) || (i==2 && j==1 && k==2 && l==1))
    {
      return (C44);
    }
  else if ((i==0 && j==2 && k==0 && l==2) || (i==0 && j==2 && k==2 && l==0) || (i==2 && j
      ↪ ==0 && k==0 && l==2) || (i==2 && j==0 && k==2 && l==0))
    {
      return (C55);
    }
  else if ((i==0 && j==1 && k==0 && l==1) || (i==0 && j==1 && k==1 && l==0) || (i==1 && j
      ↪ ==0 && k==0 && l==1) || (i==1 && j==0 && k==1 && l==0))
    {
```

```
      return (C66);
    }
  else
    {
      return 0;
    }

}

//Define the problem domain and generate the mesh
template <int dim>
void FEM<dim>::generate_mesh(std::vector<unsigned int> numberOfElements){

{

  GridIn<dim> gridin;
  gridin.attach_triangulation(triangulation);
  std::ifstream f("plate4");
  gridin.read_msh(f);
  triangulation.refine_global(0);
  const Point<2> center(1,1);

  // for (unsigned int step = 0; step < 2; ++step)
  //   {
  //     for (typename Triangulation<2>::active_cell_iterator
  //          cell = triangulation.begin_active();
  //        cell != triangulation.end(); ++cell)
  //    {
  //          for (unsigned int v = 0; v < GeometryInfo<2>::vertices_per_cell; ++v)
  //            {
  //              const double distance_from_center =
  //                center.distance(cell->vertex(v));
  //              if (std::fabs(distance_from_center) < 0.12)
  //                {
  //                  cell->set_refine_flag();
  //                  break;
  //                }
  //            }
  //        }
  //     triangulation.execute_coarsening_and_refinement();
  //   }


}

}

//Specify the Dirichlet boundary conditions
template <int dim>
void FEM<dim>::define_boundary_conds(){

  //EDIT - Define the Dirichlet boundary conditions.

  /*Note: this will be very similiar to the define_boundary_conds function
    in the HW2 template. You will loop over all degrees of freedom and use "dofLocation"
    to check if the dof is on the boundary with a Dirichlet condition.

    (Aside: Since we    now have more than 1 dof per node, it is possible to apply a
        ↪ different Dirichlet
    condition on each of the 3 dofs on the same node. Note that this is NOT the case for
        ↪ the current
    assignment. But if you wanted to do it, you would need to check
    the nodal dof (0, 1, or 2) as well as the location. For example, if you wanted to fix
        ↪  displacements
    only in the x-direction at x=0, you would have an condition such as this:
    (dofLocation[globalDOF][0] == 0 && nodalDOF == 0)
    Note that you can get the nodal dof from the global dof using the "modulo" operator,
    i.e nodalDOF = globalDOF%dim. Here "%" gives you the remainder when dividing
        ↪ globalDOF by dim.)

    Add the global dof number and the specified value (displacement in this problem)
    to the boundary values map, something like this:

    boundary_values[globalDOFIndex] = dirichletDisplacementValue
```

```
    Note that "dofLocation" is now a Table of degree of freedom locations, not just node
        ↪ locations,
    so, for example, dofs 0, 1, and 2 correspond to the same node, so that have the same
        ↪ coordinates.
    The row index is the global dof number; the column index refers to the x, y, or z
        ↪ component (0, 1, or 2 for 3D).
    e.g. dofLocation[7][2] is the z-coordinate of global dof 7*/

  const unsigned int totalDOFs = dof_handler.n_dofs(); //Total number of degrees of
      ↪ freedom

          for(unsigned int globalDOF=0; globalDOF<totalDOFs; globalDOF++){
                unsigned int nodalDOF = globalDOF%dim;

                if(dofLocation[globalDOF][0] == 0){
                  boundary_values[globalDOF] = 0;
                }

                if(dofLocation[globalDOF][1] == 0 ){
                  boundary_values[globalDOF] = 0;
                }


//              if(dofLocation[globalDOF][0] == 1 && nodalDOF==0){ //Aplica deslocamento
    ↪ prescrito na direcao x na face x = 1
        //        boundary_values[globalDOF] = 1;
        //      }
          }


}

//Setup data structures (sparse matrix, vectors)
template <int dim>
void FEM<dim>::setup_system(){

  //Let deal.II organize degrees of freedom
  dof_handler.distribute_dofs (fe);
  hanging_node_constraints.clear();
  DoFTools::make_hanging_node_constraints (dof_handler,
                                           hanging_node_constraints);
  hanging_node_constraints.close ();
  //Get a vector of global degree-of-freedom x-coordinates
  MappingQ1<dim,dim> mapping;
  std::vector< Point<dim,double> > dof_coords(dof_handler.n_dofs());
  dofLocation.reinit(dof_handler.n_dofs(),dim);
  DoFTools::map_dofs_to_support_points<dim,dim>(mapping,dof_handler,dof_coords);
  for(unsigned int i=0; i<dof_coords.size(); i++){
    for(unsigned int j=0; j<dim; j++){
      dofLocation[i][j] = dof_coords[i][j];
    }
  }

  //Specify boundary condtions (call the function)
  define_boundary_conds();

  //Define the size of the global matrices and vectors
  sparsity_pattern.reinit (dof_handler.n_dofs(),
                           dof_handler.n_dofs(),
                           dof_handler.max_couplings_between_dofs());
  DoFTools::make_sparsity_pattern (dof_handler, sparsity_pattern,hanging_node_constraints
      ↪ , true);
  sparsity_pattern.compress();
  K.reinit (sparsity_pattern);
  D.reinit(dof_handler.n_dofs());
  F.reinit(dof_handler.n_dofs());

  //Just some notes...
  std::cout << "␣␣␣Number␣of␣active␣elems:␣␣␣␣␣␣␣" << triangulation.n_active_cells() <<
      ↪ std::endl;
  std::cout << "␣␣␣Number␣of␣degrees␣of␣freedom:␣" << dof_handler.n_dofs() << std::endl;
}
```

```
//Form elmental vectors and matrices and assemble to the global vector (F) and matrix (K)
template <int dim>
void FEM<dim>::assemble_system(){

  /*NEW - deal.II basis functions, etc. The third input values (after fe and
      ↪ quadrature_formula)
    specify what information we want to be updated. For fe_values, we need the basis
        ↪ function values,
    basis function gradients,   and det(Jacobian) times the quadrature weights (JxW). For
        ↪  fe_face_values,
    we need the basis function values, the value of x at the quadrature points, and JxW.
        ↪ */

  //For volume integration/quadrature points


    FEValues<dim> fe_values(fe,
                            quadrature_formula,
                            update_values |
                            update_gradients |
                            update_quadrature_points |
                            update_JxW_values);

  //For surface integration/quadrature points
  FEFaceValues<dim> fe_face_values (fe,
                                    face_quadrature_formula,
                                    update_values |
                                    update_quadrature_points |
                                    update_JxW_values);

  K=0; F=0;

  const unsigned int dofs_per_elem = fe.dofs_per_cell;                    //This gives
      ↪ you dofs per element
  const unsigned int nodes_per_elem = fe.dofs_per_cell/dim;
  const unsigned int num_quad_pts = quadrature_formula.size();            //Total
      ↪ number of quad points in the element
  const unsigned int num_face_quad_pts = face_quadrature_formula.size();  //Total
      ↪ number of quad points in the face
  const unsigned int faces_per_elem = GeometryInfo<dim>::faces_per_cell;
  FullMatrix<double> Klocal (dofs_per_elem, dofs_per_elem);
  Vector<double>     Flocal (dofs_per_elem);

  std::vector<unsigned int> local_dof_indices (dofs_per_elem);            //This
      ↪ relates local dof numbering to global dof numbering

  //loop over elements
  typename DoFHandler<dim>::active_cell_iterator elem = dof_handler.begin_active(),
    endc = dof_handler.end();
  for (;elem!=endc; ++elem){
    //Update fe_values for the current element
    fe_values.reinit(elem);

    //Retrieve the effective "connectivity matrix" for this element
    elem->get_dof_indices (local_dof_indices);

    /*Global Assembly -
      Get the current Flocal and Klocal from the functions you wrote above
      and populate F_assembly and K_assembly using local_dof_indices to relate local and
          ↪ global DOFs.*/
    Klocal = 0.;

    //Loop over local DOFs and quadrature points to populate Klocal
    //Note that all quadrature points are included in this single loop
    for (unsigned int q=0; q<num_quad_pts; ++q){
      //      double z = fe_values.quadrature_point(q)[2]; //y-coordinate at the current
          ↪ surface quad. point
      //      std::cout << "z " << z << std::endl;
      //evaluate elemental stiffness matrix, K^{AB}_{ik} = \integral N^A_{,j}*C_{ijkl}*N^
          ↪ B_{,l} dV
      for (unsigned int A=0; A<nodes_per_elem; A++) { //Loop over nodes
        for(unsigned int i=0; i<dim; i++){ //Loop over nodal dofs
          for (unsigned int B=0; B<nodes_per_elem; B++) {
            for(unsigned int k=0; k<dim; k++){
```

```
            for (unsigned int j = 0; j<dim; j++){
              for (unsigned int l = 0; l<dim; l++){
                /*//EDIT - You need to define Klocal here. Note that the indices of
                    ↪ Klocal are the element dof numbers (0 through 23),
                  which you can caluclate from the element node numbers (0 through 8)
                    ↪ and the nodal dofs (0 through 2).
                  You'll need the following information:
                  basis gradient vector: fe_values.shape_grad(elementDOF,q), where
                    ↪ elementDOF is dim*A+i or dim*B+k
                  NOTE: this is the gradient with respect to the real domain (not the
                    ↪ bi-unit domain)
                  elasticity tensor: use the function C(i,j,k,l)
                  det(J) times the total quadrature weight: fe_values.JxW(q)*/
                    Klocal[dim*A+i][dim*B+k]+=fe_values.shape_grad(dim*A+i,q)[j]*C(i,
                      ↪ j,k,l)*
                     fe_values.shape_grad(dim*B+k,q)[l]*fe_values.JxW(q);
              }
            }
          }
        }
      }
    }

    Flocal = 0.;
    //Loop over faces (for Neumann BCs), local DOFs and quadrature points to populate
      ↪ Flocal.

    //If you had a forcing function, you would need to use FEValues here as well to
      ↪ integrate over the volume.

    //Add Neumann boundary conditions here in Flocal by integrating over the appropriate
      ↪ surface
    Vector<double> h(dim); h=0.;
    for (unsigned int f=0; f < faces_per_elem; f++){
      //Update fe_face_values from current element and face
      fe_face_values.reinit (elem, f);

      /*elem->face(f)->center() gives a position vector (in the real domain) of the
        ↪ center point on face f
        of the current element. We can use it to see if we are at the Neumann boundary,
          ↪ x_3 = 1.*/
      if(elem->face(f)->center()[0] >= 2){
//      if(elem->face(f)->center()[2] == 1){
        //To integrate over this face, loop over all face quadrature points with this
          ↪ single loop

        for (unsigned int q=0; q<num_face_quad_pts; ++q){


          //EDIT - define the value of the traction vector, h
          // h[2]=1.e9*x;
          h[0]=pow(10,8);

          //              h[0]=1.;
          for (unsigned int A=0; A<nodes_per_elem; A++){ //loop over all element nodes
            for(unsigned int i=0; i<dim; i++){ //loop over nodal dofs
              /*//EDIT - define Flocal. Again, the indices of Flocal are the element dof
                  ↪ numbers (0 through 23).
                Evaluate the basis functions using the elementDOF: fe_face_values.
                  ↪ shape_value(elementDOF,q)

                Note that we are looping over all element dofs, not just those on the
                  ↪ Neumann face. However,
                the face quadrature points are only on the Neumann face, so we are indeed
                  ↪  doing a surface integral.

                For det(J) times the total quadrature weight: fe_face_values.JxW(q)*/
               Flocal[dim*A+i]+=h[i]*fe_face_values.shape_value(dim*A+i,q)*fe_face_values
                  ↪ .JxW(q);

            }
          }
        }
```

```
}


      if(elem->face(f)->center()[1] == 2){
        //     if(elem->face(f)->center()[2] == 1){
        //To integrate over this face, loop over all face quadrature points with
            ↪ this single loop

        for (unsigned int q=0; q<num_face_quad_pts; ++q){


          //EDIT - define the value of the traction vector, h
          //   h[2]=1.e9*x;
          h[1]=5*pow(10,7);

          //        h[1]=1.;
          for (unsigned int A=0; A<nodes_per_elem; A++){ //loop over all element
              ↪ nodes
            for(unsigned int i=0; i<dim; i++){ //loop over nodal dofs
              /*//EDIT - define Flocal. Again, the indices of Flocal are the
                  ↪ element dof numbers (0 through 23).
                Evaluate the basis functions using the elementDOF: fe_face_values.
                    ↪ shape_value(elementDOF,q)

                Note that we are looping over all element dofs, not just those on
                    ↪ the Neumann face. However,
                the face quadrature points are only on the Neumann face, so we are
                    ↪ indeed doing a surface integral.

                For det(J) times the total quadrature weight: fe_face_values.JxW(q)
                    ↪ */
              Flocal[dim*A+i]+=h[i]*fe_face_values.shape_value(dim*A+i,q)*
                  ↪ fe_face_values.JxW(q);

            }
          }
        }
      }


      if(elem->face(f)->center()[1]+0.9*(elem->face(f)->center()[0]) >= 2.9){
        //     if(elem->face(f)->center()[2] == 1){
        //To integrate over this face, loop over all face quadrature points with
            ↪ this single loop

        for (unsigned int q=0; q<num_face_quad_pts; ++q){


          //EDIT - define the value of the traction vector, h
          //   h[2]=1.e9*x;
          double theta= atan(0.9);
          h[1]=5*pow(10,7)*sin(theta);
          h[0]=5*pow(10,7)*cos(theta);

          //        h[1]=1.;
          for (unsigned int A=0; A<nodes_per_elem; A++){ //loop over all element
              ↪ nodes
            for(unsigned int i=0; i<dim; i++){ //loop over nodal dofs
              /*//EDIT - define Flocal. Again, the indices of Flocal are the
                  ↪ element dof numbers (0 through 23).
                Evaluate the basis functions using the elementDOF: fe_face_values.
                    ↪ shape_value(elementDOF,q)

                Note that we are looping over all element dofs, not just those on
                    ↪ the Neumann face. However,
                the face quadrature points are only on the Neumann face, so we are
                    ↪ indeed doing a surface integral.

                For det(J) times the total quadrature weight: fe_face_values.JxW(q)
                    ↪ */
              Flocal[dim*A+i]+=h[i]*fe_face_values.shape_value(dim*A+i,q)*
                  ↪ fe_face_values.JxW(q);
```

```
                }
              }
            }
          }


    }
    //Assemble local K and F into global K and F
    for(unsigned int i=0; i<dofs_per_elem; i++){
      //EDIT - Assemble F from Flocal (you can look at HW2)
      F[local_dof_indices[i]]+=Flocal[i];
      for(unsigned int j=0; j<dofs_per_elem; j++){
        //EDIT - Assemble K from Klocal (you can look at HW2)
        K.add(local_dof_indices[i],local_dof_indices[j],Klocal[i][j]);
      }
    }

  }

  hanging_node_constraints.condense (K);
  hanging_node_constraints.condense (F);
  //Let deal.II apply Dirichlet conditions WITHOUT modifying the size of K and F global
  MatrixTools::apply_boundary_values (boundary_values, K, D, F, false);
}

//Solve for D in KD=F
template <int dim>
void FEM<dim>::solve(){

  //Solve for D
  SolverControl solver_control(1000000, 1e-30);
  SolverCG<> solver(solver_control);
  PreconditionSSOR<> preconditioner;
  preconditioner.initialize(K, 1.2);
  solver.solve(K, D, F, preconditioner);
  hanging_node_constraints.distribute (D);
//  std::cout << "D " << D << std::endl;
}

template <int dim>
void FEM<dim>::compute_stress(){

  std::ofstream strain;
  strain.open ("strain_fem.txt");

  //  QTrapez<dim>                vertex_quadrature;
  //  std::vector<Tensor<1, dim>> solution_gradients(vertex_quadrature.size());
  FEValues<dim> fe_values(fe,
                          quadrature_formula,
                          update_values |
                          update_gradients |
                          update_quadrature_points |
                          update_JxW_values);
  //  fe_values.get_function_gradients(D, solution_gradients);

  const unsigned int                          dofs_per_elem = fe.dofs_per_cell; //This
      ↪ gives you dofs per element
  std::vector<unsigned int> local_dof_indices (dofs_per_elem);
  const unsigned int                          num_quad_pts = quadrature_formula.size();
      ↪  //Total number of quad points in the element

  double  x, y,  strain1, strain2, strain3;
  int index=1;
  Vector<double> stress_vector(3);
  Vector<double> strain_vector(3);


  SymmetricTensor<2, 3> strain_stress;
  strain_stress[0][0] = 1.0608*pow(10,11);
  strain_stress[1][1] = 5.0875*pow(10,10);
  strain_stress[2][2] = 3.8897*pow(10,10);
  strain_stress[0][1] = 3.7582*pow(10,10);
  strain_stress[0][2] = 4.5662*pow(10,10);
  strain_stress[1][2] = 3.0182*pow(10,10);
```

```
  //  std::cout << "strain_stress " << strain_stress << std::endl;


  //loop over elements
  typename DoFHandler<dim>::active_cell_iterator elem = dof_handler.begin_active (),
    endc = dof_handler.end();
  for (;elem!=endc; ++elem){
    fe_values.reinit(elem);
    //Retrieve the effective "connectivity matrix" for this element
    elem->get_dof_indices (local_dof_indices);


    for(unsigned int q=0; q<num_quad_pts; q++){
      strain1=0;
      strain2=0;
      strain3=0;
      x = fe_values.quadrature_point(q)[0]; //x-coordinate at the current surface quad.
          ↪ point
      y = fe_values.quadrature_point(q)[1]; //y-coordinate at the current surface quad.
          ↪ point
      //      std::cout << "x " << x << std::endl;
      //      std::cout << "y " << y << std::endl;
      SymmetricTensor<2, dim> tmp;
      //Find the values of x and u_h (the finite element solution) at the quadrature
          ↪ points
      for(unsigned int B=0; B<dofs_per_elem; B++){
        for (unsigned int i = 0; i < dim; ++i)
          tmp[i][i] = fe_values.shape_grad_component(B, q, i)[i];
        for (unsigned int i = 0; i < dim; ++i)
          for (unsigned int j = i + 1; j < dim; ++j)
            tmp[i][j] =
              (fe_values.shape_grad_component(B, q, i)[j] +
               fe_values.shape_grad_component(B, q, j)[i]) /
              2;

        if (B%2==0){
          strain1 += D[local_dof_indices[B]]*tmp[0][0];
          strain3 += D[local_dof_indices[B]]*tmp[0][1];
        }
        else{
          strain2 += D[local_dof_indices[B]]*tmp[1][1];
          strain3 += D[local_dof_indices[B]]*tmp[0][1];
        }}
      // strain << index << "  ";
      index+=1;
      strain << x << "␣␣";
      strain << y << "␣␣";
      strain << strain1 << "␣␣";
      strain << strain2 << "␣␣";
      strain << strain3 << std::endl;

      strain_vector[0]=strain1;
      strain_vector[1]=strain2;
      strain_vector[2]=strain3;

      for (unsigned int i = 0; i < 3; ++i){
        for (unsigned int j = 0; j < 3; ++j){
          stress_vector[i]+=strain_stress[i][j]*strain_vector[j];
        }
      }
      // std::cout << x << "  " << y << std::endl;
      // std::cout << "stress_vector " << stress_vector << std::endl;

      // std::cout << "strain1 " << strain1 << std::endl;
      // std::cout << "strain2 " << strain2 << std::endl;
      // std::cout << "strain3 " << strain3 << std::endl;
    }
  }

  strain.close();
}

//Output results
template <int dim>
```

```
void FEM<dim>::output_results (){

  //Write results to VTK file
  std::ofstream output1 ("solution.vtk");
  StrainPostprocessor<dim> grad_u;
  DataOut<dim> data_out; data_out.attach_dof_handler (dof_handler);

  std::vector<DataComponentInterpretation::DataComponentInterpretation>
    data_component_interpretation
    (dim, DataComponentInterpretation::component_is_part_of_vector);
  data_out.add_data_vector (D,
                            std::vector<std::string>(dim,"displacement"),
                            DataOut<dim>::type_dof_data,
                            data_component_interpretation);
  data_out.add_data_vector (D, grad_u);
  data_out.build_patches ();
  data_out.write_vtk(output1);
  output1.close();
}
```