UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIA
DEPARTAMENTO DE ENGENHARIA CIVIL E AMBIENTAL
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA CIVIL

ELIEL TAVARES DOS REIS

INTEGRAÇÃO PYTHON-SALOME PARA SOLUÇÃO DE PROBLEMAS DE DINÂMICA ESTRUTURAL 2D E 3D COM O MEF

Recife
2018

ELIEL TAVARES DOS REIS

INTEGRAÇÃO PYTHON-SALOME PARA SOLUÇÃO DE PROBLEMAS DE DINÂMICA ESTRUTURAL 2D E 3D COM O MEF

Dissertação submetida ao Departamento de Engenharia Civil do Centro de Tecnologia e Geociência da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Engenharia Civil.

Área de concentração: Estruturas.

Orientador: Prof. Dr. Paulo Marcelo Vieira Ribeiro.

Recife
2018

Catalogação na fonte
Bibliotecária Margareth Malta, CRB-4 / 1198

R375i Reis, Eliel Tavares dos.
Integração python-salome para solução de problemas de dinâmica estrutural 2D e 3D com o MEF / Eliel Tavares dos Reis. – 2018.
96 folhas, il., gráfs., tabs.

Orientador: Prof. Dr. Paulo Marcelo Vieira Ribeiro.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CTG. Programa de Pós-Graduação em Engenharia Civil, 2018.
Inclui Referências e Apêndice.

1. Engenharia Civil. 2. Elementos finitos. 3. Python. 4. Salome. 5. Dinâmica estrutural. 6. Integração. I. Ribeiro, Paulo Marcelo Vieira. (Orientador). II. Título.

UFPE

624 CDD (22. ed.) BCTG/2019-231

ELIEL TAVARES DOS REIS

INTEGRAÇÃO PYTHON-SALOME PARA SOLUÇÃO DE PROBLEMAS DE DINÂMICA ESTRUTURAL 2D E 3D COM O MEF

Dissertação submetida ao Departamento de Engenharia Civil do Centro de Tecnologia e Geociência da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Engenharia Civil.

Área de concentração: Estruturas.

Aprovada em: 31 de agosto de 2018.

BANCA EXAMINADORA

Prof. Dr. Paulo Marcelo Vieira Ribeiro (Orientador)
Universidade Federal de Pernambuco

Prof. Dr. Marcus Vinícius Girão de Morais (Examinador Externo)
Universidade de Brasília

Prof. Dr. Roque Luiz da Silva Pitangueira (Examinador Externo)
Universidade Federal de Minas Gerais

Dedicado à Vanessa, minha esposa,
presente em todas as etapas
desta dissertação.

AGRADECIMENTOS

Agradeço, inicialmente, a Deus que através da força do Seu Espírito fez que superasse as dificuldades encontradas no caminho e chegasse até aqui.

Ao Professor Paulo Marcelo, pela oportunidade e pelos sábios ensinamentos transmitidos, e também pela sua dedicação, compreensão, paciência e incentivo como professor e orientador.

A minha família, e principalmente, aos meus pais por sempre estarem ao meu lado, me apoiando e ajudando nos momentos da minha vida.

A minha querida esposa Vanessa, pela sua presença incansável, pela sua dedicação e auxílio ao decorrer da elaboração deste trabalho, sempre do meu lado sendo meu amparo e dando ânimo e motivação para prosseguir.

Aos meus amigos e colegas pela disponibilidade e constantemente dispostos a me ajudar partilhando valiosos conhecimentos.

Todas as pessoas do grupo de esquisa MAMNE que nunca desistiram do nosso objetivo e sempre em busca do sucesso promissor.

Á todos aqueles que de alguma forma contribuíram ou torceram pela realização e concretização desse trabalho que é de suma importância para minha vida acadêmica e profissional.

RESUMO

Neste trabalho apresentam-se rotinas para análise em vibração livre de estruturas, formuladas com o Método dos Elementos Finitos, desenvolvidas em linguagem Python e integradas à plataforma Salome por meio de plugins. Salome é um programa livre, lançado sob a licença GNU/LGPL, que apresenta uma multiplataforma genérica para pré e pós-processamento em vários domínios científicos. Pode-se estender ainda mais o software através de módulos e plugins adicionais. Com isso, foram desenvolvidos cinco segmentos de plugins, com menus autoexplicativos e de fácil utilização, permitindo análises com diferentes tipos de elementos finitos. Em termos específicos foram desenvolvidos códigos independentes para análise a análise modal de membranas, placas, cascas, sólidos e uma combinação casca-sólido. Este último foi aplicado a análise modal de turbinas eólicas e representa uma contribuição prática do trabalho. O elemento de membrana é baseado na formulação do triângulo de deformação constante CST ("Constant Strain Triangle"). O elemento de placa foi desenvolvido com a formulação DKT ("Discrete Kirchhoff Triangle"). Já o elemento de casca plana é definido por meio da soma dos elementos de membrana e placa. O elemento sólido surge com a formulação linear do tetraedro de quatro nós. Finalmente, a expansão dos graus de liberdade do elemento sólido permite o acoplamento com elementos de casca e a solução de uma nova categoria de problema (dinâmica de aerogeradores). Os desenvolvimentos foram realizados com foco em baixo custo computacional e enfatizando boas práticas de programação. As análises modais foram desenvolvidas com auxílio do pacote ARPACK, que contém rotinas otimizadas para solução de problemas de autovalores e autovetores em problemas de larga escala. Adicionalmente, foram empregadas bibliotecas de compilação eficiente e dedicadas a alta performance em problemas de álgebra linear, tais como: Numba, NumPy e SciPy. As rotinas, em sua totalidade, são apresentadas em um link do repositório GitHub disponível no apêndice. O desempenho e a precisão das rotinas foram avaliados por meio de uma comparação com análises realizadas no software ANSYS. Os resultados estão em excelente concordância e demonstram que os códigos desenvolvidos podem ser aplicados de forma eficiente na solução dos problemas propostos.

Palavras-chave: Elementos finitos. Python. Salome. Dinâmica estrutural. Integração.

ABSTRACT

This work presents routines for free vibration analysis of structures, formulated with the Finite Element Method, developed in Python language and integrated to the Salome platform using Graphical User Interface (GUI) plugins. Salome is an open source software, released under the GNU / LGPL license, which presents a generic cross-platform for pre and post processing in various scientific fields. This software can be extended through additional modules and plugins, providing user-defined routines. As a result, plugins segments were developed with self-explanatory and easy-to-use menus, allowing analysis with different types of finite elements. In specific terms, five independent codes were developed for modal analysis of membranes, plates, shells, solids, and a shell-solid combination. The latter was applied to the eigenvalue and eigenvector analysis of wind turbines and represents a practical contribution of this work. The membrane element is based on the Constant Strain Triangle (CST) formulation. The plate-bending element was developed using the Discrete Kirchhoff Triangle (DKT) theory. The flat shell element is defined by the sum of the membrane and plate elements. The solid element arises with the linear four-node tetrahedron formulation. Finally, the expansion of the degrees of freedom of the solid element allows the coupling with shell elements and the solution of a new class of problem (dynamics of wind turbines). Developments were made with a focus on low computational cost and emphasizing good programming practices. The modal analyzes were developed using the ARPACK package, which contains optimized routines for the large-scale solution of eigenvalues and eigenvectors. Additionally, specific libraries for efficient code compilation and dedicated to high-performance linear algebra were used, such as Numba, NumPy, and SciPy. These routines are available in a link from the GitHub repository, provided in the appendix. The performance and precision of the proposed routines were evaluated through a comparison with analyzes carried out using ANSYS software. The results are in excellent agreement and showcase that the developed codes can be applied efficiently in the solution of the proposed class of problems.

Keywords: Finite elements. Python. Salome. Structural dynamics. Integration.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

nnos	Número de pontos nodais da malha
ngl	Número de graus de liberdade do sistema global
nnosel	Número de nós por elemento
nglel	Número de graus de liberdade por elemento
MEF	Método dos Elementos Finitos
CST	Constant Strain Triangle
DKT	Discrete Kirchhoff Triangle
LGPL	Library General Public License
NREL	National Renewable Energy Laboratory
EDP	Equação Diferencial Parcial
MRP	Métodos dos Resíduos Ponderados
T4	Elemento tetraedro linear
T3	Elemento triangular linear (CST)
CEA	Commissariat à l'énergie atomique et aux énergies alternatives
EDF	Électricité de France
GUI	Graphical User Interface
CAD	Computer-aided design
GPU	Graphics Processing Unit
API	Application Programming Interface
HAWT	Horizontal-axis wind turbines
DOE	United States Department of Energy

LISTA DE SÍMBOLOS

σ_x	Tensão normal
τ_{xy}	Tensão de cisalhamento
x,y,z	Coordenadas locais
X,Y,Z	Coordenadas globais
f	Forças de corpo
u	Função deslocamento em x
v	Função deslocamento em y
w	Função deslocamento em z
E	Modulo de elasticidade longitudinal ou módulo de Young
ε	Deformação
ν	Coeficiente de Poisson
γ	Distorção angular
G	Módulo cisalhamento ou módulo de elasticidade transversal
$\mathbf{D}$	Matriz constitutiva do material
$\mathbf{D_m}$	Matriz constitutiva do material (membrana)
$\mathbf{D_b}$	Matriz constitutiva do material (placas)
W_p	Função peso
H	Função de forma
ξ, η	Coordenadas de área
A	Área
V	Volume
t	Espessura
ρ	Densidade

$\mathbf{U}$	Vetor deslocamentos
$\mathbf{K}$	Matriz de rigidez
$\mathbf{T}$	Matriz de transformação
κ	Curvatura da placa
$\beta_x, \beta_y, \beta_z$	Rotações na direção dos, respectivos, eixos x, y e z
$\theta_x, \theta_y, \theta_z$	Rotações em torno dos, respectivos, eixos x, y e z
P	Força de excitação
F_k	Força elástica linear
F_c	Força de amortecimento
ω	Frequência natural da estrutura
ϕ	Defasagem angular
$\mathbf{M}$	Matriz rigidez de massa
diag	Diagonal de uma matriz
g	Número de graus de liberdade por nó

SUMÁRIO

1 INTRODUÇÃO

Na Engenharia Civil, observam-se estruturas cada vez mais suscetíveis a vibrações e com reduzida capacidade de dissipação de energia. A análise dinâmica torna-se indispensável nessas situações, pois essas estruturas estão sujeitas a ações dependentes do tempo (quando submetido a impactos, explosões, ações cíclicas, ventos ou sismos), sendo de considerável importância prática o estudo em regime transitório, dado que nestes problemas a parcela transitória da resposta da estrutura é importante em relação à parcela estacionária da mesma, permitindo assim predizer o comportamento nas condições mais desfavoráveis.

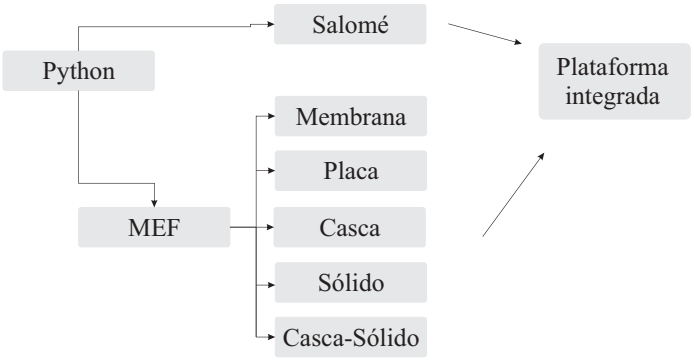
1.1 Generalidades e Objetivos

Desde o início, foi decidido que o projeto descrito nesta dissertação não seria apenas uma rotina de programação regular de elementos finitos que limitasse o seu uso em aplicações simples por não ter viabilidade em problemas complexos. O grupo de Matemática Aplicada e Métodos Numéricos em Engenharia da UFPE (MAMNE) sempre teve a ambição de desenvolver um ambiente prático com interface gráfica que pudesse integrar todos os códigos por ele desenvolvidos. Sabe-se que, construir um software dessa magnitude, com todos os elementos disponíveis, demanda tempo considerável e expertise que vai além dos conhecimentos da mecânica computacional.

Através dessa concepção, este trabalho tem o intuito de desenvolver um ambiente computacional específico e prático, elaborado com plugins escritos em Python e integrados à plataforma Salome, com a funcionalidade inicial para análise modal de modelos com diversos elementos finitos acoplados. Objetivando futuramente a construção de um software para o auxílio na análise dinâmica das estruturas, englobando recursos e ferramentas para facilitar o desenvolvimento do trabalho.

Salome é um software livre, lançado sob a licença GNU/LGPL, que disponibiliza uma multiplataforma genérica de pré e pós-processamento para simulação numérica. Pode-se estender ainda mais o software através da criação de módulos adicionais ou plugins, tais como: algoritmos e solucionadores específicos, Fig. 1. O objetivo é tornar a plataforma adequada a um dado campo de aplicação.

Figura 1 – Esquema geral de integração Python-Salome desta dissertação.



Fonte: O Autor (2018).

A linguagem Python foi lançada por Guido van Rossum em 1991. Atualmente é aberta e gerenciada pela organização sem fins lucrativos Python Software Foundation. Python prioriza a legibilidade do código, sendo concisa, clara e com recursos poderosos de sua biblioteca padrão e dos módulos e frameworks desenvolvidos por terceiros.

Por ser uma linguagem interpretada, ela pode ser executada em qualquer sistema operacional que possui interpretador Python. Quanto à aceitação da linguagem no mercado, pode-se dizer que possui uma imensa comunidade ao redor do globo, que desenvolvem bibliotecas complexas e funcionais aos problemas em questão.

Com o objetivo de elevar a precisão dos resultados e para formular cada elemento, foi necessário realizar uma revisão bibliográfica abordando as principais teorias acerca do tema, para melhor aperfeiçoamento das técnicas dos algoritmos dos códigos desenvolvidos. Através dessas pesquisas e estudos, foram formulados códigos computacionais em Python para as implementações dos algoritmos que estão voltados para análise modal.

Para a comparação de modelos, foram utilizados o software comercial ANSYS versão 18.2 e os códigos desenvolvidos para as obtenções dos resultados. Verifica-se uma eficiência dos códigos aplicados a problemas simples e complexos como, por exemplo, o modelo numérico de uma turbina eólica, obtendo resultados satisfatórios e validando a técnica de acoplamento entre elementos de cascas e sólidos. No final, é fornecido ao usuário (no repositório GIT) o código fonte dos plugins desenvolvidos, permitindo o acesso ao conjunto de funcionalidades desta dissertação.

1.2 Organização do Trabalho

Esta dissertação está organizada em seis capítulos.

No capítulo dois, abordam-se todas as teorias necessárias para os desenvolvimentos dos algoritmos, como a teoria de elasticidade bidimensional para placas finas, formulação e acoplamento do elemento de casca ao de volume e a teoria clássica da análise modal em

Dinâmicas das Estruturas. Todas as teorias são apresentadas com auxílio de referências da literatura.

No capítulo três, o conteúdo sobre as ferramentas computacionais adotadas é apresentado abordando o software Salome, a linguagem de programação Python e a integração entre eles. Ainda no mesmo capítulo, são discutidos os "plugins" desenvolvidos.

O capítulo quatro é dedicado aos estudos básicos de modelos simples discretizados por um único tipo de elemento finito, com o intuito de validar os códigos implementados. Em seguida os resultados são comparados com os obtidos pelo ANSYS.

Já no capítulo cinco, buscou-se realizar um estudo de caso com um modelo numérico complexo envolvendo elementos de cascas e sólidos acoplados. Novamente os resultados foram validados com os obtidos pelo ANSYS.

Por fim, são apresentadas as considerações finais e as perspectivas para desenvolvimentos futuros.

2 REVISÃO BIBLIOGRÁFICA

A teoria da elasticidade linear estuda as tensões, deformações e deslocamentos de um corpo, considerado elástico, causadas pela ação de forças externas (TIMOSHENKO; GERE, 1982).

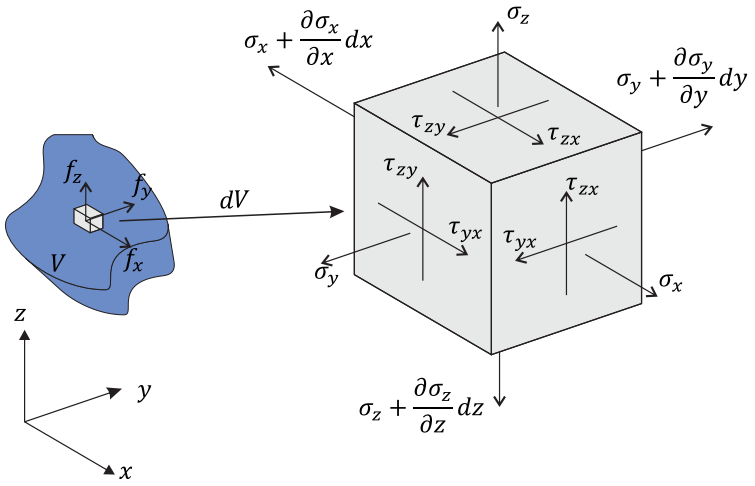
2.1 Teoria da Elasticidade

Segundo Logan (2006), para uma solução exata de um problema de mecânica estrutural, devem ser satisfeitas três conjuntos de equações incluídas na clássica teoria da elasticidade: (i) Sistema de equações diferenciais de equilíbrio formuladas em termos de tensões atuando sobre uma parte infinitesimal do corpo; (ii) As relações diferenciais cinemáticas ou relação entre deformações e deslocamentos; e (iii) As leis tensões-deformações ou relações constitutivas.

2.1.1 Equações diferenciais de equilíbrio

As equações diferencias parciais – EDP podem ser representadas por um sistema de equações. Para isso, considera-se o elemento infinitesimal tridimensional em coordenadas cartesianas com dimensões (dx, dy, dz) e tensão normal e de cisalhamento, Fig. 2, em equilíbrio:

Figura 2 – Equilíbrio de um elemento infinitesimal de um corpo contínuo.



Fonte: O Autor (2018).

Onde, σ_x, σ_y e σ_z são as tensões normais em cada plano; τ_{yx}, τ_{zx} e τ_{zy} são as tensões de cisalhamento; e f_x, f_y e f_z são as forças de corpo.

Para o equilíbrio dos esforços na direção x, têm-se:

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} + f_x = 0 \tag{2.1}$$

De forma similar nas direções y e z, respectivamente:

$$\frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} + \frac{\partial \tau_{xy}}{\partial x} + f_y = 0 \tag{2.2}$$

$$\frac{\partial \sigma_z}{\partial z} + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + f_z = 0 \tag{2.3}$$

Como as três equações devem ser satisfeitas simultaneamente, então o sistema de equações é definido pelas Eqs. (2.1), (2.2) e (2.3).

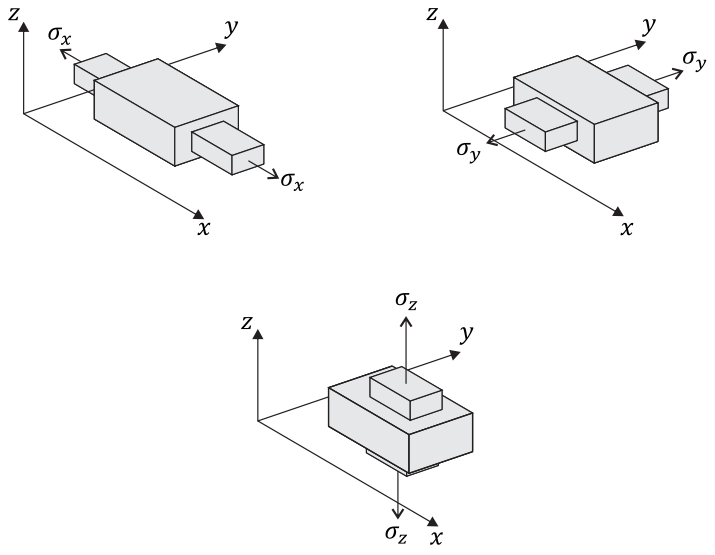
2.1.2 Relações constitutivas

As relações entre tensões e deformações em materiais isotrópicos têm como base a lei de Hooke:

$$\sigma = E\varepsilon \tag{2.4}$$

onde, σ, E e ε são, respectivamente, a tensão normal, módulo de elasticidade longitudinal e a deformação de alongamento (ou encurtamento) na direção da tensão normal.

Figura 3 – Elemento diferencial sujeito a tensões normais atuando em três direções perpendiculares



Fonte: O Autor (2018).

A Fig. 3 ilustra um corpo diferencial submetido a tensões normais às faces: σ_x, σ_y e σ_z. Pode-se observar que a peça sofreu alongamentos nas direções das tensões normais,

sendo perceptível encurtamentos nas laterais. Esse fenômeno é definido como efeito de Poisson. Na direção x, por exemplo, calcula-se:

$$\varepsilon_{yx} = -\nu\frac{\sigma_y}{E} \tag{2.5}$$

onde, ν é a razão de Poisson e ε_{yx} é o encurtamento provocado pela tensão σ_y na direção de x.

De forma análoga, pode-se determinar as equações das deformações longitudinais em cada eixo:

$$\varepsilon_x = \frac{\sigma_x}{E} - \nu\frac{\sigma_y}{E} - \nu\frac{\sigma_z}{E} \tag{2.6a}$$
$$\varepsilon_y = -\nu\frac{\sigma_x}{E} + \frac{\sigma_y}{E} - \nu\frac{\sigma_z}{E} \tag{2.6b}$$
$$\varepsilon_z = -\nu\frac{\sigma_x}{E} - \nu\frac{\sigma_y}{E} + \frac{\sigma_z}{E} \tag{2.6c}$$

O material pode sofrer distorções angulares γ em decorrência das tensões de cisalhamento τ, novamente se baseando na lei de Hooke para o cisalhamento:

$$\gamma_{xy} = \frac{\tau_{xy}}{G}, \qquad \gamma_{yz} = \frac{\tau_{yz}}{G}, \qquad \gamma_{zx} = \frac{\tau_{zx}}{G} \tag{2.7}$$

onde, G é o módulo de cisalhamento ou módulo de elasticidade transversal dado pela expressão

$$G = \frac{E}{2(1+\nu)}. \tag{2.8}$$

Manipulando as Eqs. (2.6) e (2.7) com intuito de deixar somente no lado esquerdo as tensões e organizando em forma matricial, chega-se a Eq. (2.9):

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{Bmatrix} = \mathbf{D} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix} \tag{2.9}$$

onde $\mathbf{D}$ define a matriz constitutiva do material:

$$\mathbf{D} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{2.10}$$

Para modelos bidimensionais, existem dois tipos de problemas em condições diferentes: Estado plano de tensões e Estado plano de deformações.

O estado plano de tensões é caracterizado por $\sigma_z = \tau_{xz} = \tau_{yz} = 0$ e componentes com espessura muito pequena. Portanto, desprezam-se as tensões na direção da espessura. Neste caso, a Eq. (2.9) assume a forma abaixo:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \mathbf{D_m} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{2.11}$$

onde $\mathbf{D_m}$ é a matriz constitutiva no Estado Plano de Tensões (EPT).

$$\mathbf{D_m} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{2.12}$$

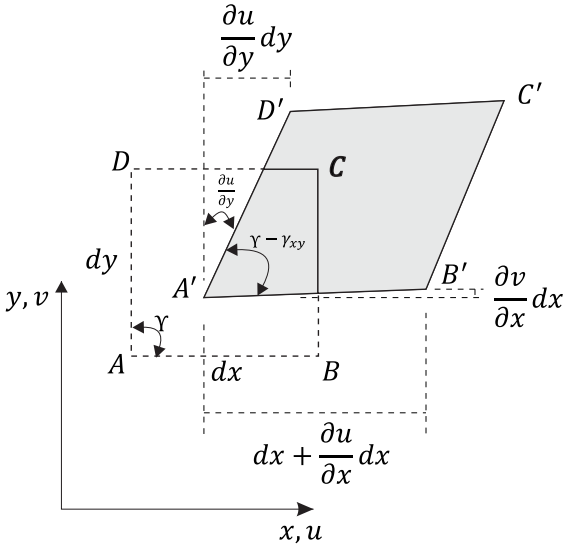
Já o Estado Plano de Deformações (EPD) é obtido com a hipótese onde $\varepsilon_z = \gamma_{yz} = \gamma_{xz} = 0$, desprezando as deformações na direção da espessura e reduzindo a Eq. (2.9) a:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{2.13}$$

2.1.3 Relações cinemáticas

A Fig. 4 esquematiza de forma ilustrativa as relações cinemáticas para o elemento bidimensional deformável:

Figura 4 – Elemento diferencial antes e depois da deformação



Fonte: O Autor (2018).

Na Fig. 4 o elemento em estado indeformável está em linhas tracejadas, e os deslocamentos nas direções x e y são representados respectivamente por u e v.

Por definição de engenharia, a deformação normal é dada pela mudança de comprimento dividido pelo comprimento original. Considerando a linha AB e sua deformada $A^{'}B^{'}$ do elemento da Fig. 4, e tendo em vista o negligenciamento de pequenos deslocamentos representados pelos termos $\frac{\partial u}{\partial y}dy$ e $\frac{\partial v}{\partial x}dx$, surgem as seguintes aproximações válidas:

$$\varepsilon_x = \frac{A^{'}B^{'} - AB}{AB} \tag{2.14a}$$

$$AB = dx \tag{2.14b}$$

$$A^{'}B^{'} = dx(1 + \frac{\partial u}{\partial x}) \tag{2.14c}$$

Com base nas Eqs. (2.14), define-se:

$$\varepsilon_x = \frac{\partial u}{\partial x} \tag{2.15}$$

De forma similar, considerando a linha AD na direção y:

$$\varepsilon_y = \frac{\partial v}{\partial y} \tag{2.16}$$

A distorção angular γ_{xy} é definida como alteração no ângulo entre duas linhas, como AB e AD, que originalmente formavam um ângulo reto. Observa-se na Fig. 4 que a distorção angular γ_{xy} é dada por:

$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{2.17}$$

As Eqs. (2.15), (2.16) e (2.17) representam as relações cinemáticas no plano.

Em situação tridimensional, o problema passará a ter deslocamento na direção z representado por w. Consequentemente, de forma análoga ao procedimento bidimensional, são obtidas relações cinemáticas adicionais com as Eqs. (2.18):

$$\varepsilon_z = \frac{\partial w}{\partial z} \tag{2.18a}$$

$$\gamma_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \tag{2.18b}$$

$$\gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \tag{2.18c}$$

2.2 Método dos Elementos Finitos em Elasticidade

O Método dos Elementos Finitos é um procedimento numérico que permite o cálculo de diversos problemas complexos por meios de aproximações. Modelos de geometrias complicadas são discretizados em elementos geométricos simples.

A solução do sistema de Equações Diferenciais Parciais (EDP), formado pelas Eqs. (2.1), (2.2) e (2.3),

$$\begin{cases} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} + f_x = 0 \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} + f_y = 0 \\ \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + f_z = 0 \end{cases} \tag{2.19}$$

pode ser obtida pelo Método dos Resíduos Ponderados (MRP), que permite a elaboração de declarações integrais que constituem a base do Método dos Elementos Finitos. As demonstrações a seguir foram adaptadas de Kwon e Bang (1996).

A aplicação da forma fraca do MRP ao sistema de Eqs. (2.19) produz:

$$\int_V \begin{Bmatrix} W_1(\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}) \\ W_2(\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}) \\ W_3(\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z}) \end{Bmatrix} dV + \int_V \begin{Bmatrix} W_1 f_x \\ W_2 f_y \\ W_3 f_z \end{Bmatrix} dV = 0 \tag{2.20}$$

onde, W_1, W_2 e W_3 são funções de ponderação e V representa o contínuo na Fig. 2.

O problema da Eq. (2.20) pode ser simplificado com a transformação do domínio V em integrais triplas x, y e z. Aplicando integração por partes ao primeiro termo da Eq. (2.20), obtém-se:

$$\begin{aligned} \iiint \begin{Bmatrix} W_1(\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}) \\ W_2(\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}) \\ W_3(\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z}) \end{Bmatrix} dxdydz = -\iiint \begin{Bmatrix} \frac{\partial W_1}{\partial x}\sigma_x + \frac{\partial W_1}{\partial y}\tau_{xy} + \frac{\partial W_1}{\partial z}\tau_{xz} \\ \frac{\partial W_2}{\partial x}\tau_{xy} + \frac{\partial W_2}{\partial y}\sigma_y + \frac{\partial W_2}{\partial z}\tau_{yz} \\ \frac{\partial W_3}{\partial x}\tau_{xz} + \frac{\partial W_3}{\partial y}\tau_{yz} + \frac{\partial W_3}{\partial z}\sigma_z \end{Bmatrix} dxdydz + \\ \iint \begin{Bmatrix} W_1\sigma_x \\ W_2\tau_{xy} \\ W_3\tau_{xz} \end{Bmatrix} dydz + \iint \begin{Bmatrix} W_1\tau_{xy} \\ W_2\sigma_y \\ W_3\tau_{yz} \end{Bmatrix} dxdz + \iint \begin{Bmatrix} W_1\tau_{xz} \\ W_2\tau_{yz} \\ W_3\sigma_z \end{Bmatrix} dxdy \end{aligned} \tag{2.21}$$

Simplificando os termos da Eq. (2.21) e substituindo na Eq. (2.20):

$$\int_V \begin{Bmatrix} \frac{\partial W_1}{\partial x}\sigma_x + \frac{\partial W_1}{\partial y}\tau_{xy} + \frac{\partial W_1}{\partial z}\tau_{xz} \\ \frac{\partial W_2}{\partial x}\tau_{xy} + \frac{\partial W_2}{\partial y}\sigma_y + \frac{\partial W_2}{\partial z}\tau_{yz} \\ \frac{\partial W_3}{\partial x}\tau_{xz} + \frac{\partial W_3}{\partial y}\tau_{yz} + \frac{\partial W_3}{\partial z}\sigma_z \end{Bmatrix} dV - \int_A \begin{Bmatrix} W_1 q_x \\ W_2 q_y \\ W_3 q_z \end{Bmatrix} dA - \int_V \begin{Bmatrix} W_1 f_x \\ W_2 f_y \\ W_3 f_z \end{Bmatrix} dV = 0 \tag{2.22}$$

A partir do primeiro termo da Eq. (2.22) é obtida a matriz de rigidez, no segundo e terceiro termos são apresentadas as forças nodais equivalentes (q_x, q_y e q_z) e as forças de corpo (f_x, f_y e f_z).

De acordo com as relações constitutivas e cinemáticas, o primeiro termo da Eq. (2.22) pode ser formulado da seguinte maneira:

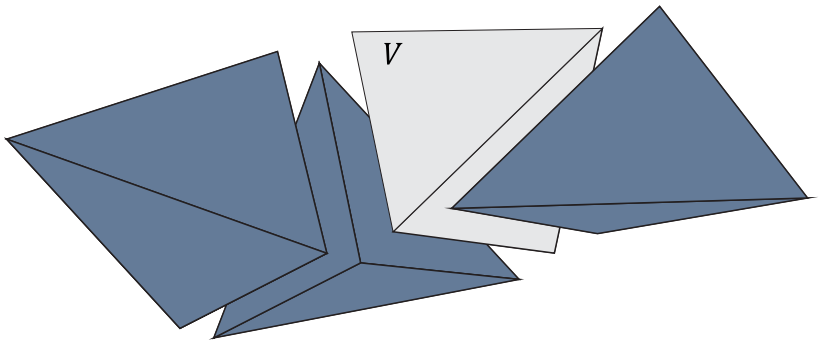
$$\int_V \mathbf{LD} \begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \end{Bmatrix} dV \tag{2.23}$$

onde,

$$\mathbf{L} = \begin{bmatrix} \frac{\partial W_1}{\partial x} & 0 & 0 & \frac{\partial W_1}{\partial y} & 0 & \frac{\partial W_1}{\partial z} \\ 0 & \frac{\partial W_2}{\partial y} & 0 & \frac{\partial W_2}{\partial x} & \frac{\partial W_2}{\partial z} & 0 \\ 0 & 0 & \frac{\partial W_3}{\partial z} & 0 & \frac{\partial W_3}{\partial y} & \frac{\partial W_3}{\partial x} \end{bmatrix} \tag{2.24}$$

Considera-se agora que o domínio V será tomado como um elemento finito de forma arbitrária. Assim o contínuo da Fig. 2 será substituído por um conjunto de elementos finitos (Fig. 5).

Figura 5 – Substituição do contínuo por uma malha de elementos finitos



Fonte: O Autor (2018).

No domínio de cada elemento, os deslocamentos u, v, e w são definidos pelas funções de interpolação abaixo:

$$u(x,y,z) = \sum_{i=1}^{n} H_i(x,y,z)u_i \tag{2.25}$$

$$v(x,y,z) = \sum_{i=1}^{n} H_i(x,y,z)v_i \tag{2.26}$$

$$w(x,y,z) = \sum_{i=1}^{n} H_i(x,y,z)w_i \tag{2.27}$$

onde por conveniência são escolhidas as mesmas funções de forma $H_i(x,y,z)$ para as três equações. Os termos u_i, v_i e w_i indicam valores nodais nos vértices do elemento (graus de liberdade).

Com as relações cinemáticas e as Eqs. (2.25), (2.26) e (2.27), sabe-se que:

$$\begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial z}+\frac{\partial w}{\partial y} \\ \frac{\partial u}{\partial z}+\frac{\partial w}{\partial x} \end{Bmatrix} = \begin{bmatrix} \frac{\partial H_1}{\partial x} & 0 & 0 & \frac{\partial H_2}{\partial x} & 0 & 0 & \cdots & \frac{\partial H_n}{\partial x} & 0 & 0 \\ 0 & \frac{\partial H_1}{\partial y} & 0 & 0 & \frac{\partial H_2}{\partial y} & 0 & \cdots & 0 & \frac{\partial H_n}{\partial y} & 0 \\ 0 & 0 & \frac{\partial H_1}{\partial z} & 0 & 0 & \frac{\partial H_2}{\partial z} & \cdots & 0 & 0 & \frac{\partial H_n}{\partial z} \\ \frac{\partial H_1}{\partial y} & \frac{\partial H_1}{\partial x} & 0 & \frac{\partial H_2}{\partial y} & \frac{\partial H_2}{\partial x} & 0 & \cdots & \frac{\partial H_n}{\partial y} & \frac{\partial H_n}{\partial x} & 0 \\ 0 & \frac{\partial H_1}{\partial z} & \frac{\partial H_1}{\partial y} & 0 & \frac{\partial H_2}{\partial z} & \frac{\partial H_2}{\partial y} & \cdots & 0 & \frac{\partial H_n}{\partial z} & \frac{\partial H_n}{\partial y} \\ \frac{\partial H_1}{\partial z} & 0 & \frac{\partial H_1}{\partial x} & \frac{\partial H_2}{\partial z} & 0 & \frac{\partial H_2}{\partial x} & \cdots & \frac{\partial H_n}{\partial z} & 0 & \frac{\partial H_n}{\partial x} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ u_2 \\ v_2 \\ w_2 \\ \vdots \\ \vdots \\ \vdots \\ u_n \\ v_n \\ w_n \end{Bmatrix} \tag{2.28}$$

onde n indica o número de nós do elemento.

A matriz e o vetor do lado direito da Eq. (2.28) são representados, respectivamente por $\mathbf{B}$ e $\mathbf{U}$. Em uma representação compacta, escreve-se:

$$\begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial w}{\partial z} \\ \frac{\partial u}{\partial y}+\frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial z}+\frac{\partial w}{\partial y} \\ \frac{\partial u}{\partial z}+\frac{\partial w}{\partial x} \end{Bmatrix} = \mathbf{BU} \tag{2.29}$$

No método de Galerkin as funções de ponderação W_1, W_2 e W_3 são obtidas por meio da derivada das funções de interpolações de u, v e w em relação aos graus de liberdade do elemento u_i, v_i e w_i. Por exemplo:

$$W_1 = \begin{cases} \dfrac{\partial u}{\partial u_1} = H_1 \\ \dfrac{\partial u}{\partial u_2} = H_2 \\ \vdots \\ \dfrac{\partial u}{\partial u_n} = H_n \end{cases} \quad , \quad W_2 = \begin{cases} \dfrac{\partial v}{\partial v_1} = H_1 \\ \dfrac{\partial v}{\partial v_2} = H_2 \\ \vdots \\ \dfrac{\partial v}{\partial v_n} = H_n \end{cases} \quad , \quad W_3 = \begin{cases} \dfrac{\partial w}{\partial w_1} = H_1 \\ \dfrac{\partial w}{\partial w_2} = H_2 \\ \vdots \\ \dfrac{\partial w}{\partial w_n} = H_n \end{cases} \tag{2.30}$$

Assim, aplicando as Eqs. (2.30) em (2.24) e lembrando que cada função de ponderação está associada a n graus de liberdade:

$$\mathbf{L} = \mathbf{B^T} \tag{2.31}$$

Finalmente, com a substituição das Eqs. (2.29) e (2.31) na Eq. (2.23), obtém-se:

$$\int_V \mathbf{B^T D B} dV \mathbf{U} \tag{2.32}$$

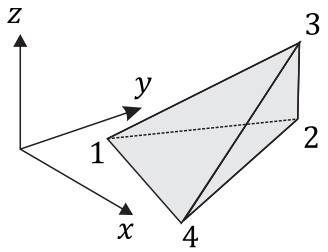
onde a matriz de rigidez do elemento é dada por:

$$\mathbf{K^e} = \int_V \mathbf{B^T D B} dV \tag{2.33}$$

2.2.1 Elemento Sólido – Tetraedro Linear (T4)

Neste tópico será abordada a formulação do tetraedro T4 por ser o elemento finito sólido de comportamento linear (ou de quatro nós) e de fácil entendimento.

Figura 6 – Elemento tetraedro



Fonte: O Autor (2018).

O tetraedro é um sólido composto por quatro faces triangulares, seis arestas e quatro vértices. De acordo com Logan (2006) os nós do elemento devem ser numerados de forma que, quando vistos pelo quarto nó, os três primeiros sejam ordenados no sentido anti-horário, como na Fig. 6. Essa ordenação evita o cálculo de volumes negativos.

Na formulação do elemento finito sólido, os graus de liberdade para o cálculo das deformações ε e tensões σ são referenciados às funções de deslocamento u, v e w. Ou seja, os graus de liberdade do sólido são apenas translações u_i, v_i e w_i.

$$\mathbf{U} = \begin{bmatrix} u_1 & v_1 & w_1 & u_2 & v_2 & w_2 & u_3 & v_3 & w_3 & u_4 & v_4 & w_4 \end{bmatrix}^T \tag{2.34}$$

Portanto, adotam-se funções de interpolações lineares:

$$\begin{cases} u(x,y,z) = \sum_{i=1}^{4} H_i u_i \\ v(x,y,z) = \sum_{i=1}^{4} H_i v_i \\ w(x,y,z) = \sum_{i=1}^{4} H_i w_i \end{cases} \tag{2.35}$$

onde,

$$H_1 = \frac{\alpha_1 + \beta_1 x + \gamma_1 y + \delta_1 z}{6V} \tag{2.36a}$$

$$H_2 = \frac{\alpha_2 + \beta_2 x + \gamma_2 y + \delta_2 z}{6V} \tag{2.36b}$$

$$H_3 = \frac{\alpha_3 + \beta_3 x + \gamma_3 y + \delta_3 z}{6V} \tag{2.36c}$$

$$H_4 = \frac{\alpha_4 + \beta_4 x + \gamma_4 y + \delta_4 z}{6V} \tag{2.36d}$$

$$\begin{aligned}
\alpha_1 &= \det\begin{vmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{vmatrix} & \beta_1 &= -\det\begin{vmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} \\
\gamma_1 &= \det\begin{vmatrix} 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \\ 1 & x_4 & z_4 \end{vmatrix} & \delta_1 &= -\det\begin{vmatrix} 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \end{vmatrix}
\end{aligned} \tag{2.37}$$

$$\begin{aligned}
\alpha_2 &= -\det\begin{vmatrix} x_1 & y_1 & z_1 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{vmatrix} & \beta_2 &= \det\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} \\
\gamma_2 &= -\det\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_3 & z_3 \\ 1 & x_4 & z_4 \end{vmatrix} & \delta_2 &= \det\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \end{vmatrix}
\end{aligned} \tag{2.38}$$

$$\begin{aligned}
\alpha_3 &= \det\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_4 & y_4 & z_4 \end{vmatrix} & \beta_3 &= -\det\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_4 & z_4 \end{vmatrix} \\
\gamma_3 &= \det\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x_4 & z_4 \end{vmatrix} & \delta_3 &= -\det\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_4 & y_4 \end{vmatrix}
\end{aligned} \tag{2.39}$$

$$\begin{aligned}
\alpha_4 &= -\det\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} & \beta_4 &= \det\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \\
\gamma_4 &= -\det\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \end{vmatrix} & \delta_4 &= \det\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}
\end{aligned} \tag{2.40}$$

e o termo V representa o volume do elemento.

De acordo com as Eqs. (2.28), (2.29) e (2.36), a matriz **B** para o elemento tetraedro TE4 pode ser definida pela expressão abaixo:

$$\mathbf{B_{tetra}} = \frac{1}{6V}\begin{bmatrix} \beta_1 & 0 & 0 & \beta_2 & 0 & 0 & \beta_3 & 0 & 0 & \beta_4 & 0 & 0 \\ 0 & \gamma_1 & 0 & 0 & \gamma_2 & 0 & 0 & \gamma_3 & 0 & 0 & \gamma_4 & 0 \\ 0 & 0 & \delta_1 & 0 & 0 & \delta_2 & 0 & 0 & \delta_3 & 0 & 0 & \delta_4 \\ \gamma_1 & \beta_1 & 0 & \gamma_2 & \beta_2 & 0 & \gamma_3 & \beta_3 & 0 & \gamma_4 & \beta_4 & 0 \\ 0 & \delta_1 & \gamma_1 & 0 & \delta_2 & \gamma_2 & 0 & \delta_3 & \gamma_3 & 0 & \delta_4 & \gamma_4 \\ \delta_1 & 0 & \beta_1 & \delta_2 & 0 & \beta_2 & \delta_3 & 0 & \beta_3 & \delta_4 & 0 & \beta_4 \end{bmatrix} \tag{2.41}$$

Aplicando a Eq. (2.33), obtem-se a matriz de rigidez do elemento:

$$\mathbf{K^e_{tetra}} = \mathbf{B^T_{tetra}} \mathbf{D} \mathbf{B_{tetra}} V \tag{2.42}$$

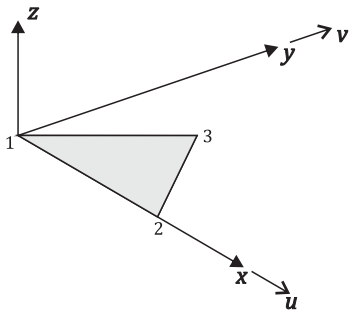
Importante destacar que o integrando é constante. O que torna sua implementação de baixo custo computacional.

2.2.2 Elemento de Membrana Linear (T3)

O elemento T3, também conhecido como triângulo de deformação constante (CST), é considerado um elemento de membrana no plano xy (Fig. 7). Os graus de liberdade para o cálculo das deformações ε e tensões σ são referenciados às funções de deslocamentos no plano u e v. Os graus de liberdade para o elemento CST são:

$$\mathbf{U_m} = \begin{bmatrix} u_1 & v_1 & u_2 & v_2 & u_3 & v_3 \end{bmatrix}^T \tag{2.43}$$

Figura 7 – Elemento CST



Fonte: O Autor (2018).

Assume-se que as variações dos deslocamentos são lineares:

$$\begin{cases} u(x,y) = \sum_{i=1}^{3} H_i u_i \\ v(x,y) = \sum_{i=1}^{3} H_i v_i \end{cases} \tag{2.44}$$

onde,

$$H_1 = \frac{(y_3 - y_2)(x - x_2) - (x_3 - x_2)(y - y_2)}{2A} \tag{2.45a}$$

$$H_2 = \frac{-(y_3 - y_1)(x - x_3) + (x_3 - x_1)(y - y_3)}{2A} \tag{2.45b}$$

$$H_3 = \frac{(y_2 - y_1)(x - x_1) - (x_2 - x_1)(y - y_1)}{2A} \tag{2.45c}$$

Como a temática desse trabalho envolve o estado plano de tensões, de forma análoga às Eqs. (2.28), (2.29), (2.44) e adotando a relação constitutiva, chega-se nas matrizes $\mathbf{B}$ e $\mathbf{K}$ para o elemento de membrana CST:

$$\mathbf{B_m} = \frac{1}{2A}\begin{bmatrix} (y_3 - y_2) & 0 & -(y_3 - y_1) & 0 & (y_2 - y_1) & 0 \\ 0 & -(x_3 - x_2) & 0 & (x_3 - x_1) & 0 & -(x_2 - x_1) \\ -(x_3 - x_2) & (y_3 - y_2) & (x_3 - x_1) & -(y_3 - y_1) & -(x_2 - x_1) & (y_2 - y_1) \end{bmatrix} \tag{2.46}$$

$$\mathbf{K_m^e} = At\mathbf{B_m^T}\mathbf{D_m}\mathbf{B_m} \tag{2.47}$$

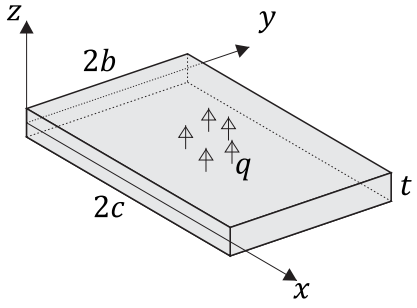
onde, t e A são, respectivamente, espessura e área do elemento.

Mais uma vez destaca-se a vantagem do integrando constante do elemento CST nas implementações computacionais.

2.3 Estruturas em Cascas e Placas Finas

Placas e cascas são estruturas laminares, ou seja, estruturas sólidas limitadas por duas superfícies geralmente equidistantes de espessura t sendo esta dimensão muito menor que as dimensões das limítrofes. As placas possuem superfície média plana e as cascas, superfície média não plana. Alguns exemplos de estruturas laminares são dados por: tabuleiros de pontes, chaminés e coberturas.

Figura 8 – Dimensões de uma placa fina básica com carregamento transversal



Fonte: O Autor (2018).

Para as derivações das equações básicas, considera-se que a placa está no plano xy e a espessura t na direção z, Fig. 8. Assume-se também que o plano da superfície média está em $z = 0$. Segundo Oñate (2009), pode-se chamar de placa fina quando a espessura t for menor que 5% das suas outras dimensões, caso t seja maior que 10% da extensão da placa, então a deformação transversal de cisalhamento γ deve ser considerada, e a placa é dita espessa.

Logan (2006) reforça que para se obter resultados reais, é necessário ter problemas com deflexão w menor que a espessura t.

Neste capítulo serão abordadas as teorias clássicas de flexão de placas de Kirchhoff e de Reissner-Mindlin, pois são fundamentais nas formulações dos elementos de flexão de placas.

Em seguida, são apresentados os elementos de flexão da placa triangular (DKT) e de casca triangular.

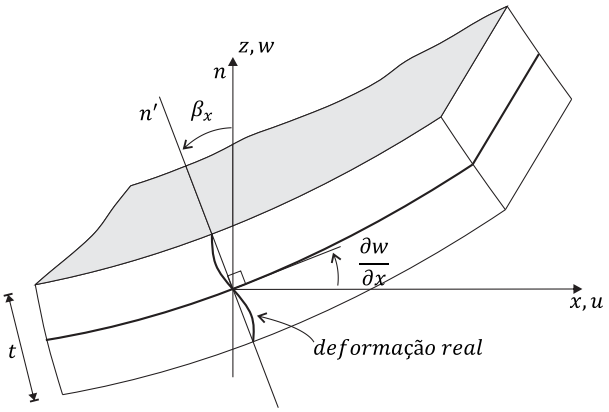
2.3.1 Teoria de Kirchhoff

Os pressupostos básicos para a teoria clássica de flexão de placas de Kirchhoff (1850) são muito semelhantes aos da teoria de vigas de Euler-Bernoulli. Uma das suposições mais importantes para ambas as teorias é o princípio da ortogonalidade, em que uma linha reta normal ao plano médio da placa antes da deformação permanece normal a esse plano após a deformação.

São adotadas as seguintes hipóteses:

1. Os pontos pertencentes ao plano médio só se movem na direção do eixo z;
2. Todos os pontos contidos em uma normal ao plano médio possuem o mesmo deslocamento vertical: $\varepsilon_z = 0$;
3. As tensões normais $(\sigma_z, \tau_{xz}, \tau_{yx})$ ao plano médio da placa são desprezíveis;
4. Admite-se a condição de ortogonalidade ilustrada na Fig. 9, por consequência, conforme as Eqs. (2.48), (2.18b) e (2.18c), as distorções angulares transversais são nulas $\gamma_{xz} = \gamma_{yz} = 0$.

Figura 9 – Condição de ortogonalidade na placa flexionada de Kirchhoff (Procedimento similar na direção do eixo y)



Fonte: O Autor (2018).

Na Fig. 9, n e $n^{'}$ são os eixos normais aos planos médios indeformado e deformado (assume-se uma reta), respectivamente, e $\beta_x = \frac{\partial w}{\partial x}$ (ou $\beta_y = \frac{\partial w}{\partial y}$).

Portanto, como mostrado na Fig. 9 e nas hipóteses, os deslocamentos u, v e w podem ser expressos como:

$$u = -z\frac{\partial w(x,y)}{\partial x} \tag{2.48a}$$

$$v = -z\frac{\partial w(x,y)}{\partial y} \tag{2.48b}$$

$$w = w(x,y) \tag{2.48c}$$

Substituindo as Eqs. (2.48) nas Eqs. (2.15), (2.16) e (2.17) pode-se escrever as deformações do problema na forma matricial:

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = -z \begin{Bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{Bmatrix} \tag{2.49}$$

Onde κ indica a curvatura:

$$\kappa_{\mathbf{Kirch.}} = \begin{Bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial^2 w}{\partial x^2} \\ \dfrac{\partial^2 w}{\partial y^2} \\ -2z\dfrac{\partial^2 w}{\partial x \partial y} \end{Bmatrix} \tag{2.50}$$

Conforme a hipótese 3 da teoria de Kirchhoff, aplicam-se as Eqs. (2.49) e (2.50) na Eq. (2.11), resultando na relação constitutiva do problema de flexão de placas de Kirchhoff:

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = -z\mathbf{D_m}\kappa_{\mathbf{Kirch.}} \tag{2.51}$$

Os momentos M_x, M_y e M_{xy} são definidos pela seguinte integral:

$$\mathbf{M} = \begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \int_{-\frac{t}{2}}^{\frac{t}{2}} -z \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} dz \tag{2.52}$$

Resolvendo a integral da Eq. (2.52), chega-se a relação entre os momentos $\mathbf{M}$ e curvaturas κ:

$$\mathbf{M} = \mathbf{D_b}\kappa_{\mathbf{Kirch.}} \tag{2.53}$$

onde,

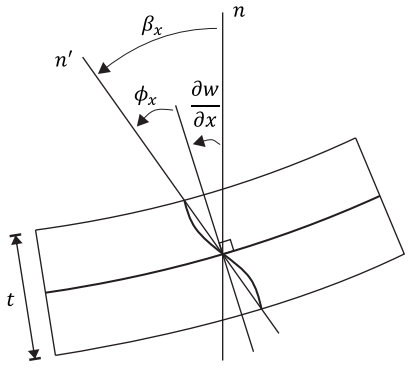
$$\mathbf{D_b} = \frac{t^3}{12}\mathbf{D_m} \tag{2.54}$$

2.3.2 Teoria de Reissner-Mindlin

A teoria de flexão de placas de Mindlin (1951) surge como solução de placas que não podem ser consideradas finas para as quais as distorções angulares transversais γ_{xz} e γ_{yx} são significativas. Portanto, a teoria de flexão de Reissner-Mindin compartilha os três primeiros pressupostos da teoria de flexão de Kirchhoff, e distingue-se no pressuposto 4 (MESQUITA, 1998), que fica na forma:

4. Não se admite a condição de ortogonalidade, pois a linha reta normal ao plano médio indeformado continua sendo reta mas não necessariamente ortogonal ao plano médio deformado (Fig. 10).

Figura 10 – Não ortogonalidade na placa flexionada de Reissner-Mindlin (Procedimento similar na direção do eixo y)



Fonte: O Autor (2018).

Na Fig. 10, n indica o eixo normal ao plano médio antes da deformação, $n^{'}$ indica a nova orientação de n que passa a ser não ortogonal ao plano médio deformado, e $\beta_x = \phi_x + \frac{\partial w}{\partial x}$ (ou $\beta_y = \phi_y + \frac{\partial w}{\partial y}$).

Assim, como mostrado na Fig. 10 e nas hipóteses, os deslocamentos u, v e w podem ser expressos como:

$$u = -z\beta_x(x,y) \tag{2.55a}$$

$$v = -z\beta_y(x,y) \tag{2.55b}$$

$$w = w(x,y) \tag{2.55c}$$

De acordo com as relações cinemáticas, pode-se escrever as relações dos deslocamentos, da Eq. (2.55), com as deformações na seguinte forma matricial:

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{xz} \end{Bmatrix} = \begin{Bmatrix} -z\dfrac{\partial \beta_x}{\partial x} \\ -z\dfrac{\partial \beta_y}{\partial y} \\ -z(\dfrac{\partial \beta_x}{\partial y} + \dfrac{\partial \beta_y}{\partial x}) \\ -\phi_y \\ -\phi_x \end{Bmatrix} \tag{2.56}$$

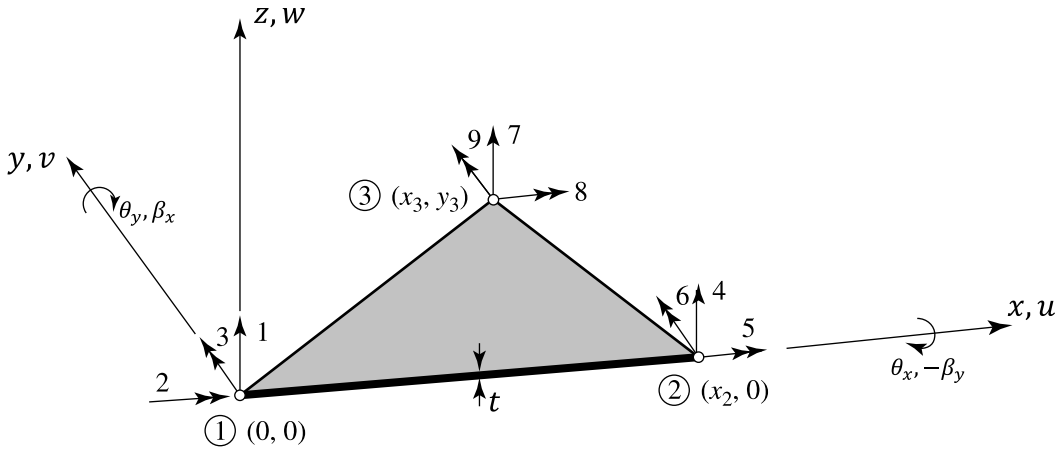
Observando-se a Eq. (2.56), conclui-se que a hipótese da não ortogonalidade da normal afeta o campo de deformação, de tal forma, qua as distorções angulares transversais γ_{yz} e γ_{xz} são, exatamente, os ângulos ϕ (MESQUITA, 1998).

2.3.3 Formulação do Elemento Triangular de Flexão de Placas Finas – DKT

A abordagem do elemento Triangular Discreto de Kirchhoff – DKT surgiu nos anos 70 com as publicações de Dhatt (1969) e Kikuchi (1975), mas sua aceitação mais ampla aconteceu na década de 80 com a publicação de Batoz et al. (1980). Após inúmeros testes de convergência e a comprovação da sua eficiência, o elemento DKT continua sendo um dos melhores elementos triangulares para flexão de placas finas.

Embora a formulação do elemento DKT se baseie na teoria de Mindlin (para placas moderadamente grossas), a teoria de Kirchhoff é aplicada de forma discreta em pontos específicos, onde as distorções angulares transversais (ou deformações de cisalhamento) γ_{yz} e γ_{xz} devem ser nulas.

Figura 11 – Elemento DKT explicitamente formulado.

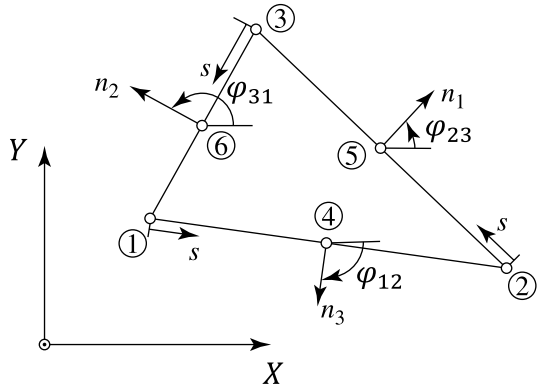


Fonte: Adaptado de Szilard (2004).

Por isso, apenas as contribuições dos termos de flexão β_x e β_y são levadas em conta para o cálculo da matriz de rigidez do elemento, entretanto, a compatibilidade com o termo de deflexão w tem que existir. Conforme a Fig. 11, os graus de liberdade do elemento DKT são:

$$\mathbf{U_b} = \begin{bmatrix} w_1 & \theta_{x1} & \theta_{y1} & w_2 & \theta_{x2} & \theta_{y2} & w_3 & \theta_{x3} & \theta_{y3} \end{bmatrix}^T \tag{2.57}$$

Figura 12 – Elemento inicial (triângulo de Reissner-Mindlin).



Fonte: Adaptado de Szilard (2004).

De acordo com Batoz et al. (1980), o ponto de partida para formulação do elemento DKT é o triângulo de Reissner-Mindlin de 6 pontos da Fig. 12, sob as seguintes restrições:

1. As funções de aproximação das rotações β_x e β_y tem variações quadráticas sobre o elemento, doze graus de liberdade nos seis nós:

$$\beta_x = \sum_{i=1}^{6} N_i \beta_{xi} \tag{2.58a}$$

$$\beta_y = \sum_{i=1}^{6} N_i \beta_{yi} \tag{2.58b}$$

onde, N_i são polinômios quadráticos para funções de forma expressas nas coordenadas de área ξ e η, como mostrado na Fig. 13.

Essas funções de forma são dadas por:

$$N_1 = 2(1-\xi-\eta)(\frac{1}{2}-\xi-\eta) \tag{2.59a}$$

$$N_2 = \xi(2\xi-1) \tag{2.59b}$$

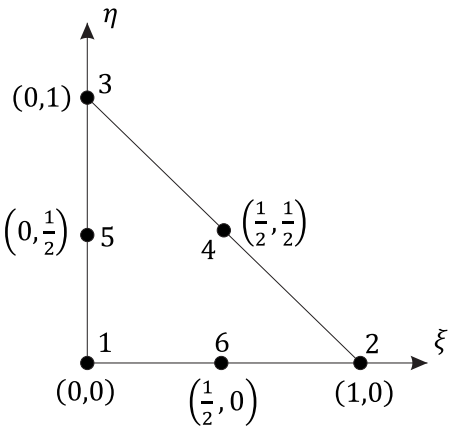
$$N_3 = \eta(2\eta-1) \tag{2.59c}$$

$$N_4 = 4\xi\eta \tag{2.59d}$$

$$N_5 = 4\eta(1-\xi-\eta) \tag{2.59e}$$

$$N_6 = 4\xi(1-\xi-\eta) \tag{2.59f}$$

Figura 13 – Coordenadas de área: ξ e η



Fonte: O Autor (2018).

2. A função deflexão $w(s)$ tem variações cúbicas ao longo de cada lado ij com os termos w_i, $(\frac{\partial w}{\partial s})_i$, w_j e $(\frac{\partial w}{\partial s})_j$, onde $ij = (23),(31),(12)$. Sendo assim, ao longo das bordas, a rotação $\frac{\partial w}{\partial s}$ no ponto médio k é

$$\left.\frac{\partial w}{\partial s}\right|_k = -\frac{3}{2l_{ij}}w_i - \frac{1}{4}\frac{\partial w_i}{\partial s} + \frac{3}{2l_{ij}}w_j - \frac{1}{4}\frac{\partial w_j}{\partial s} \qquad para\ k = 4{,}5{,}6, \tag{2.60}$$

onde l_{ij} indica o comprimento do lado ij com os vértices i e j.

3. A hipótese de Kirchhoff é imposta discretamente nos seguintes pontos do elemento da Fig. 12:

   a) Pontos nodais de vértice, $i = 1{,}2{,}3$:

$$\gamma_{xz} = \left.\frac{\partial w}{\partial x}\right|_i - \beta_{xi} = 0 \tag{2.61a}$$

$$\gamma_{yz} = \left.\frac{\partial w}{\partial y}\right|_i - \beta_{yi} = 0 \tag{2.61b}$$

b) Pontos médios nodais, k = 4,5,6:

$$\left.\frac{\partial w}{\partial s}\right|_k - \beta_{sk} = 0 \tag{2.62}$$

4. Uma variação linear de β_n é imposta ao longo dos lados ij, ou seja:

$$\beta_{nk} = \frac{\beta_{ni} + \beta_{nj}}{2} \tag{2.63}$$

5. Para adquirir uma relação entre os 12 graus de liberdade de rotação do item 1 com os 9 graus de liberdade $\mathbf{U_b}$ da Eq. (2.57), a seguinte transformação é necessária para cada lado ij:

$$\begin{Bmatrix}\beta_x\\ \beta_y\end{Bmatrix} = \begin{bmatrix}\cos\varphi_{ij} & -\sin\varphi_{ij}\\ \sin\varphi_{ij} & \cos\varphi_{ij}\end{bmatrix}\begin{Bmatrix}\beta_n\\ \beta_s\end{Bmatrix}, \quad \begin{Bmatrix}\dfrac{\partial w}{\partial s}\\ \dfrac{\partial w}{\partial n}\end{Bmatrix} = \begin{bmatrix}\cos\varphi_{ij} & \sin\varphi_{ij}\\ \sin\varphi_{ij} & -\cos\varphi_{ij}\end{bmatrix}\begin{Bmatrix}\theta_x\\ \theta_y\end{Bmatrix} \tag{2.64}$$

onde φ_{ij} é o ângulo entre o eixo X e a normal do lado ij no plano XY, conforme a Fig. 12.

Com as condições 3, 4 e 5, o campo de rotação é finalmente expresso em termos dos nove graus de liberdade padrão – $\mathbf{U_b}$:

$$\begin{Bmatrix}\beta_x\\ \beta_y\end{Bmatrix} = \begin{bmatrix}N_{x1} & N_{x2} & N_{x3} & N_{x4} & N_{x5} & N_{x6} & N_{x7} & N_{x8} & N_{x9}\\ N_{y1} & N_{y2} & N_{y3} & N_{y4} & N_{y5} & N_{y6} & N_{y7} & N_{y8} & N_{y9}\end{bmatrix}\begin{Bmatrix}w_1\\ \theta_{x1}\\ \theta_{y1}\\ w_2\\ \theta_{x2}\\ \theta_{y2}\\ w_3\\ \theta_{x3}\\ \theta_{y3}\end{Bmatrix} \tag{2.65}$$

onde as funções de forma N_{xi} e N_{yi} são mostradas na Fig. 14.

Figura 14 – Funções de forma do elemento DKT(N_i são as funções de forma da Eq. (2.59)).

$$\begin{array}{lll}
N_{x_1} = 1.5(a_6N_6 - a_5N_5) \quad ; & N_{x_2} = b_5N_5 + b_6N_6 & N_{x_3} = N_1 - c_5N_5 - c_6N_6\\
N_{x_4} = 1.5(a_4N_4 - a_6N_6) \quad ; & N_{x_5} = b_6N_6 + b_4N_4 & N_{x_6} = N_2 - c_4N_4 - c_6N_6\\
N_{x_7} = 1.5(a_5N_5 - a_4N_4) \quad ; & N_{x_8} = b_4N_4 + b_4N_5 & N_{x_9} = N_3 - c_4N_4 - c_5N_5\\
N_{y_1} = 1.5(d_6N_6 - d_5N_5) \quad ; & N_{y_2} = -N_1 + e_5N_5 + e_6N_6 \quad ; & N_{y_3} = -N_{x_2}\\
N_{y_4} = 1.5(d_4N_4 - d_6N_6) \quad ; & N_{y_5} = -N_2 + e_4N_4 + e_6N_6 \quad ; & N_{y_6} = -N_{x_5}\\
N_{y_7} = 1.5(d_5N_5 - d_4N_4) \quad ; & N_{y_8} = -N_3 + e_4N_4 + e_5N_5 \quad ; & N_{y_9} = -N_{x_8}
\end{array}$$

$$a_k = -\frac{x_{ij}}{l_{ij}^2} \quad ; \quad b_k = \frac{3}{4l_{ij}^2}x_{ij}y_{ij} \qquad c_k = \left(\frac{1}{4}x_{ij}^2 - \frac{1}{2}y_{ij}^2\right)/l_{ij}^2$$

$$d_k = -\frac{y_{ij}}{l_{ij}^2} \quad ; \quad e_k = \left(\frac{1}{4}y_{ij}^2 - \frac{1}{2}x_{ij}^2\right)/l_{ij}^2 \quad ; \quad l_{ij}^2 = (x_{ij}^2 + y_{ij}^2)$$

$x_{ij} = x_i - x_j$, $y_{ij} = y_i - y_j$, $k = 4, 5, 6$ *para os lados* $ij = 23, 31, 12$.

Fonte: Adaptado de Oñate (2009).

Sabe-se que as deformações ε_x, ε_y e γ_{xy} podem ser expressas por:

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = -z\kappa_{DKT} \tag{2.66}$$

onde,

$$\kappa_{DKT} = \begin{Bmatrix} \dfrac{\partial \beta_x}{\partial x} \\ \dfrac{\partial \beta_y}{\partial y} \\ \dfrac{\partial \beta_x}{\partial y} + \dfrac{\partial \beta_y}{\partial x} \end{Bmatrix} = \mathbf{B_b U_b} \tag{2.67}$$

Calculando a matriz $\mathbf{B_b}$ em coordenadas de área ξ e η conforme a Fig. 13, chega-se:

$$\mathbf{B_b} = \frac{1}{2A} \begin{bmatrix} y_{31}\dfrac{\partial \mathbf{H_x}}{\partial \xi} + y_{12}\dfrac{\partial \mathbf{H_x}}{\partial \eta} \\ -x_{31}\dfrac{\partial \mathbf{H_y}}{\partial \xi} - x_{12}\dfrac{\partial \mathbf{H_y}}{\partial \eta} \\ -x_{31}\dfrac{\partial \mathbf{H_x}}{\partial \xi} - x_{12}\dfrac{\partial \mathbf{H_x}}{\partial \eta} + y_{31}\dfrac{\partial \mathbf{H_y}}{\partial \xi} + y_{12}\dfrac{\partial \mathbf{H_y}}{\partial \eta} \end{bmatrix} \tag{2.68}$$

onde,

$$\begin{aligned} \mathbf{H_x} &= \begin{bmatrix} N_{x1} & N_{x2} & N_{x3} & N_{x4} & N_{x5} & N_{x6} & N_{x7} & N_{x8} & N_{x9} \end{bmatrix} \\ \mathbf{H_y} &= \begin{bmatrix} N_{y1} & N_{y2} & N_{y3} & N_{y4} & N_{y5} & N_{y6} & N_{y7} & N_{y8} & N_{y9} \end{bmatrix} \end{aligned} \tag{2.69}$$

A matriz de rigidez do elemento DKT é, agora, expressa pela forma padrão

$$\mathbf{K_b^e} = 2A \int_0^1 \int_0^{1-\eta} \mathbf{B_b^T D_b B_b} d\xi d\eta \tag{2.70}$$

para o qual Batoz (1982) introduziu a forma explícita obtida por integração exata em coordenadas de área

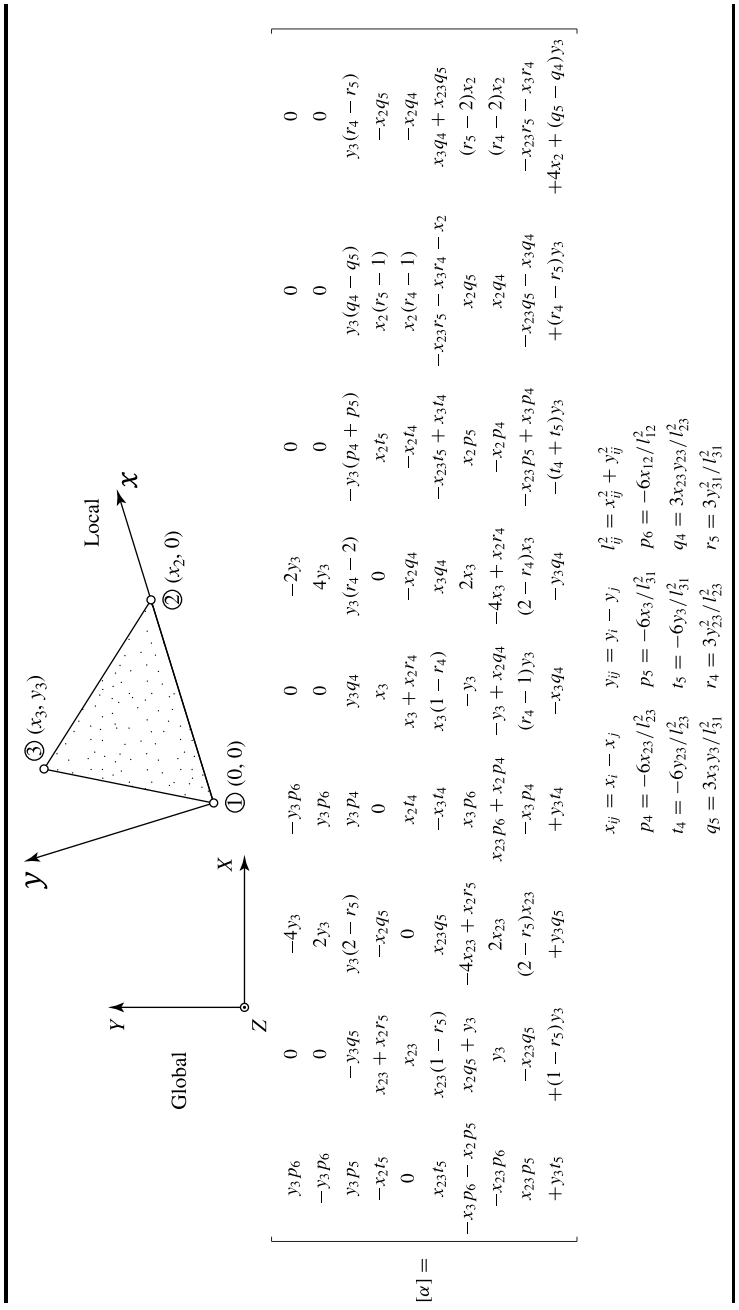
$$\mathbf{K_b^e} = \frac{1}{2A} \alpha^{\mathbf{T}} \mathbf{E} \alpha \tag{2.71}$$

onde, o termo A representa a área do elemento triangular, $\mathbf{E}$ indica a matriz de material elástico isotrópico homogêneo dado por

$$\mathbf{E} = \frac{Et^3}{288(1-\nu^2)} \begin{bmatrix} \mathbf{R} & \nu\mathbf{R} & \mathbf{0} \\ \nu\mathbf{R} & \mathbf{R} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dfrac{1-\nu}{2}\mathbf{R} \end{bmatrix}, \qquad \mathbf{R} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{2.72}$$

e a Fig. 15 fornece a expressão explícita da matriz α.

Figura 15 – Expressão explícita da matriz α.



$$[\alpha] = \begin{bmatrix}
y_3p_6 & 0 & -4y_3 & -y_3p_6 & 0 & -2y_3 & 0 & 0 & 0 \\
-y_3p_6 & 0 & 2y_3 & y_3p_6 & 0 & 4y_3 & 0 & 0 & 0 \\
y_3p_5 & -y_3q_5 & y_3(2-r_5) & y_3p_4 & y_3q_4 & y_3(r_4-2) & -y_3(p_4+p_5) & y_3(q_4-q_5) & y_3(r_4-r_5) \\
-x_2t_5 & x_{23}+x_2r_5 & -x_2q_5 & 0 & x_3 & 0 & x_2t_5 & x_2(r_5-1) & -x_2q_5 \\
0 & x_{23} & 0 & x_2t_4 & x_3+x_2r_4 & -x_2q_4 & -x_2t_4 & x_2(r_4-1) & -x_2q_4 \\
x_{23}t_5 & x_{23}(1-r_5) & x_{23}q_5 & -x_3t_4 & x_3(1-r_4) & x_3q_4 & -x_{23}t_5+x_3t_4 & -x_{23}r_5-x_3r_4-x_2 & x_3q_4+x_{23}q_5 \\
-x_3p_6-x_2p_5 & x_2q_5+y_3 & -4x_{23}+x_2r_5 & x_3p_6 & -y_3 & 2x_3 & x_2p_5 & x_2q_5 & (r_5-2)x_2 \\
-x_{23}p_6 & y_3 & 2x_{23} & x_{23}p_6+x_2p_4 & -y_3+x_2q_4 & -4x_3+x_2r_4 & -x_2p_4 & x_2q_4 & (r_4-2)x_2 \\
x_{23}p_5 & -x_{23}q_5 & (2-r_5)x_{23} & -x_3p_4 & (r_4-1)y_3 & (2-r_4)x_3 & -x_{23}p_5+x_3p_4 & -x_{23}q_5-x_3q_4 & -x_{23}r_5-x_3r_4 \\
+y_3t_5 & +(1-r_5)y_3 & +y_3q_5 & +y_3t_4 & -x_3q_4 & -y_3q_4 & -(t_4+t_5)y_3 & +(r_4-r_5)y_3 & +4x_2+(q_5-q_4)y_3
\end{bmatrix}$$

$$x_{ij} = x_i - x_j \qquad y_{ij} = y_i - y_j \qquad l_{ij}^2 = x_{ij}^2 + y_{ij}^2$$
$$p_4 = -6x_{23}/l_{23}^2 \quad p_5 = -6x_3/l_{31}^2 \quad p_6 = -6x_{12}/l_{12}^2$$
$$t_4 = -6y_{23}/l_{23}^2 \quad t_5 = -6y_3/l_{31}^2 \quad q_4 = 3x_{23}y_{23}/l_{23}^2$$
$$q_5 = 3x_3y_3/l_{31}^2 \quad r_4 = 3y_{23}^2/l_{23}^2 \quad r_5 = 3y_{31}^2/l_{31}^2$$

Fonte: Adaptado de Szilard (2004).

Batoz (1982) também deixa explícita a matriz $\mathbf{B_b}$:

$$\mathbf{B_b} = \frac{1}{2A}\begin{bmatrix} 1 & -\xi & -\xi\eta & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -\xi & -\xi\eta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -\xi & -\xi\eta \end{bmatrix} \alpha \tag{2.73}$$

2.3.4 Formulação do Elemento de Casca Triangular

Segundo Cook et al. (2002), as formulações de elementos finitos para cascas são complexas e difíceis de implementar. De modo simplificado, Mesquita (1998) define que o

comportamento estrutural de cascas é resultante do estado de membrana trabalhando em conjunto com o estado de flexão. O estado de membrana é caracterizado como deformações de alongamento ou encurtamento e distorção da estrutura com distribuições de tensões através da espessura. Já o estado de flexão é consequência de deformações proveniente de momentos que tendem a fletir e torcer a estrutura e de esforços cortantes que tentam cisalhar a mesma.

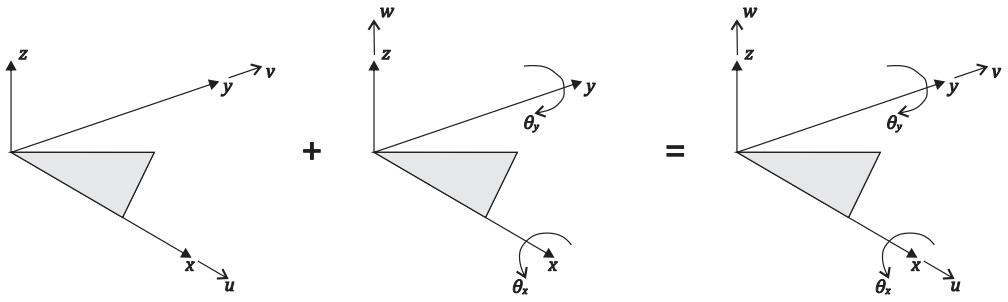
Deste modo, o elemento de casca (plano, sem curvatura) pode ser obtido combinando os elementos de flexão da placa e de membrana. Como apresentado na Fig. 16, o elemento de flexão da placa possui uma deflexão transversal w e rotações de flexão θ_x e θ_y como graus de liberdade em cada nó. Por outro lado, o elemento de membrana possui dois graus de liberdade u e v referentes aos deslocamentos no plano. Todos juntos, a casca passa a ter cinco graus de liberdade por nó, três deslocamentos e duas rotações.

A matriz de rigidez de um elemento de casca é dada abaixo:

$$\begin{bmatrix} \mathbf{K_b} & \mathbf{0} \\ \mathbf{0} & \mathbf{K_m} \end{bmatrix} \begin{Bmatrix} \mathbf{d_b} \\ \mathbf{d_m} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F_b} \\ \mathbf{F_m} \end{Bmatrix} \tag{2.74}$$

onde, $\mathbf{K}, \mathbf{d}$ e $\mathbf{F}$ indicam matrizes de rigidez, vetor deslocamentos/rotações nodais e vetor momentos/forças nodais, respectivamente. As matrizes e vetores consistem em duas partes, um referente a flexão da placa – **b** e outro ao alongamento (ou encurtamento) da placa (efeito de membrana) – **m**

Figura 16 – Elemento de membrana, elemento de flexão da placa e elemento de casca.



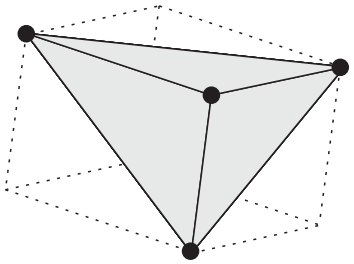
Fonte: O Autor (2018).

O grau de liberdade de rotação em torno do eixo normal (eixo -z) ao plano da casca, também denominado na literatura global de “drilling”, é inexistente no elemento de casca. Entretanto, quando os elementos são orientados de maneira diferente, como por exemplo, o encontro de três elementos triangulares em planos perpendiculares, como pode ser visto na Fig. 17, a rotação de flexão em um dos elementos torna-se o “drilling” em outro elemento. Sendo assim, Kwon e Bang (2000) recomendam garantir a posição do grau de liberdade “drilling” em cada ponto nodal do elemento de casca, caracterizando a

liberdade rotacional em torno do eixo z. Então, a Eq. (2.74) é reescrita como:

$$\begin{bmatrix} \mathbf{K_b} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{K_m} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{d_b} \\ \mathbf{d_m} \\ \theta_\mathbf{z} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F_b} \\ \mathbf{F_m} \\ \mathbf{0} \end{Bmatrix} \tag{2.75}$$

Figura 17 – Três elementos de casca perpendiculares entre si.



Fonte: O Autor (2018).

Em casos que a casca é realmente plana, a matriz de rigidez torna-se singular devido aos termos singulares associados aos graus de liberdade "drilling". A fim de evitar esse problema, um pequeno número pode ser adicionado ao termo diagonal da matriz na Eq. (2.75), associado ao grau de liberdade "drilling". Este número não deve ser tão pequeno para que a matriz modificada possa ser numericamente singular ou quase singular. Por outro lado, o número não deve ser tão grande para afetar a precisão da solução, porque é um acréscimo arbitrário.

$$\begin{bmatrix} \mathbf{K_b} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{K_m} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{K}_{\theta_\mathbf{z}} \end{bmatrix} \begin{Bmatrix} \mathbf{d_b} \\ \mathbf{d_m} \\ \theta_\mathbf{z} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F_b} \\ \mathbf{F_m} \\ \mathbf{0} \end{Bmatrix} \tag{2.76}$$

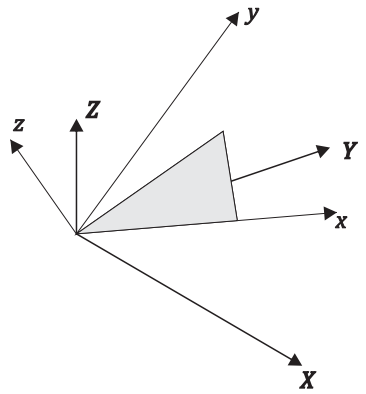
Outra opção é remover as linhas e colunas referentes ao "drilling" em cascas (placas) 100% planas.

A matriz e os vetores da Eq. (2.75) são representados em termos do sistema de coordenada local que tem o eixo-x e -y ao longo do plano médio de cada elemento de casca e eixo-z normal ao mesmo plano, Fig 18. Para montar essas matrizes e vetores dos elementos na matriz e vetor do sistema, os graus de liberdade nodais em termos de cada eixo local devem ser transformados nos graus de liberdade nodais correspondentes em termos dos eixos de coordenas globais comuns a todos os elementos. Se a matriz de transformação é $\mathbf{T}$, então pode-se escrever:

$$\mathbf{d_{local}} = \mathbf{T}\mathbf{d_{global}} \tag{2.77}$$

A matriz de transformação $\mathbf{T}$ consiste em cossenos de direção entre os sistemas de coordenadas globais e locais.

Figura 18 – Eixos globais – (X,Y,Z), eixos locais – (x,y,z)



Fonte: O Autor (2018).

Em cada nó, a relação entre os graus locais e globais de liberdade é expressada pela seguinte forma:

$$\begin{Bmatrix} u_{local} \\ v_{loval} \\ w_{local} \\ \theta_{x_{local}} \\ \theta_{y_{local}} \\ \theta_{z_{local}} \end{Bmatrix} = \begin{bmatrix} \cos xX & \cos xY & \cos xZ & 0 & 0 & 0 \\ \cos yX & \cos yY & \cos yZ & 0 & 0 & 0 \\ \cos zX & \cos zY & \cos zZ & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos xX & \cos xY & \cos xZ \\ 0 & 0 & 0 & \cos yX & \cos yY & \cos yZ \\ 0 & 0 & 0 & \cos zX & \cos zY & \cos zZ \end{bmatrix} \begin{Bmatrix} u_{global} \\ v_{global} \\ w_{global} \\ \theta_{x_{global}} \\ \theta_{y_{global}} \\ \theta_{z_{global}} \end{Bmatrix} \tag{2.78}$$

onde a matriz da Eq. (2.78) é chamada $\mathbf{T_d}$.

Portanto, para um elemento de três nós, a matriz de transformação $\mathbf{T}$ torna-se:

$$\mathbf{T} = \begin{bmatrix} \mathbf{T_d} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{T_d} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{T_d} \end{bmatrix} \tag{2.79}$$

Usando a matriz de transformação, a matriz de rigidez transformada e o vetor de cargas são dados abaixo:

$$\mathbf{K_{global}} = \mathbf{T^T K_{local} T} \tag{2.80}$$

$$\mathbf{F_{global}} = \mathbf{T^T F_{local}} \tag{2.81}$$

2.3.5 Conexão entre elementos sólidos e de casca

A importância do desenvolvimento de modelos com diferentes tipos de elementos numa mesma análise colabora tanto para convergência do problema com malhas menos refinadas como também possibilita uma melhor representação de certas simplificações no modelo numérico.

A dificuldade na combinação dos elementos de cascas com os de sólidos é que eles tem diferentes graus de liberdade em cada nó. Como por exemplo o elemento tetraedro e o de casca triangular de três nós:

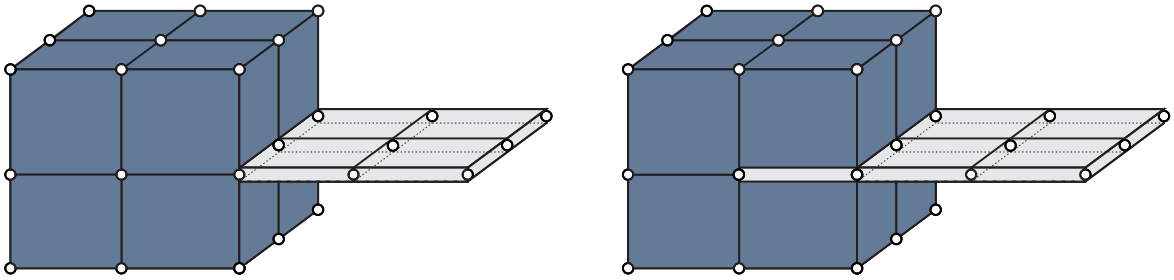
- Elementos sólidos: u, v e w.
- Elementos de casca: u, v, w, θ_x, θ_y e θ_z.

Logan (2006) mostra que se o submodelo casca estiver conectado a apenas uma borda do submodelo sólido, o resultado é articulações mecânica (Fig. 19(a)). Ainda de acordo com Logan, esse problema pode ser corrigido estendendo o submodelo de casca dentro do submodelo sólido, de modo que as malhas dos submodelos fiquem sobrepostas, compartilhando os mesmos pontos nodais na área de conexão (Fig. 19(b)).

Figura 19 – Conexão solido-casca.

(a) Submodelo casca articulado no submodelo sólido.

(b) Submodelo casca engastado no submodelo sólido.



Fonte: Adaptado de Logan (2006)

Isso significa que os graus de liberdade de translação de ambos são sincronizados e as rotações são relacionadas somente aos elementos de cascas. Entretanto, Logan (2006) afirma que as tensões próximas à borda de contato não serão precisas.

Para montagem das matrizes de rigidez e de massa global do sistema, uma estratégia distinta, com as expansões das matrizes do elemento T4, Eq. (2.42), adicionando os graus de rotação θ_x, θ_y e θ_z em cada ponto nodal do elemento. Ressalta-se que são desconsideradas as massas dos elementos de casca sobrepostos.

Agora, o T4 passa a ter 24 graus de liberdade:

$$\mathbf{K^e_{tetra}} = \begin{bmatrix} \mathbf{K^e_{1,1}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{K^e_{1,2}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{K^e_{1,3}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{K^e_{1,4}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} \\ & \mathbf{K}_\theta & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} \\ & & \mathbf{K^e_{2,2}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{K^e_{2,3}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{K^e_{2,4}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} \\ & & & \mathbf{K}_\theta & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} \\ & & & & \mathbf{K^e_{3,3}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{K^e_{3,4}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} \\ & & & & & \mathbf{K}_\theta & \mathbf{0}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} \\ & & & & & & \mathbf{K^e_{4,4}}_{\mathbf{[3x3]}} & \mathbf{0}_{\mathbf{[3x3]}} \\ & & & & & & & \mathbf{K}_\theta \end{bmatrix} \tag{2.82}$$

onde, $\mathbf{K_{i,j}}$ são submatrizes da Eq. (2.42), e

$$\mathbf{K}_\theta = \begin{bmatrix} k_{\theta_x} & & \\ & k_{\theta_y} & \\ & & k_{\theta_z} \end{bmatrix} \tag{2.83}$$

em que os coeficientes k_θ são pequenos o suficiente para não singularidade.

2.4 Dinâmica das Estruturas

No decorrer desta dissertação, foram abordados problemas de cargas estáticas. Entretanto, na prática, os problemas de engenharia envolvem perturbações dinâmicas produzidas por forças externas ou deslocamentos variáveis do tempo: rajadas de vento, perturbação sísmicas, máquinas, impactos de ondas marítimas, veículo em movimento, etc.

A dinâmica das estruturas lida com movimentos dependentes do tempo, e analisa os esforços internos associados a eles. Assim o objetivo final é analisar o desempenho da estrutura com o efeito das vibrações, Szilard (2004).

O movimento alternado de um corpo sólido em relação a uma posição de referência, é definido como vibração, Collacott (1979).

Os problemas com as vibrações em estruturas eólicas são específicos devido ao fato de integrarem, no seu topo, várias toneladas de peso e estarem sujeitas a paradas súbitas, que provocam vibrações na estrutura, Sequeira (2012).

Os desenvolvimentos a seguir contemplam aspectos associados a dinâmica estrutural e fornecem subsídios ao produto final da dissertação, que é um módulo Python-Salome para análise modal de aerogeradores.

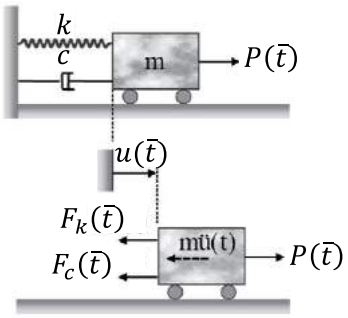
2.4.1 Equações Diferenciais de Movimento

Os conceitos fundamentais da análise dinâmica serão apresentados através de um modelo de um grau de liberdade. A essência do comportamento dinâmica elástico desse sistema é simples, e a mesma pode ser aplicada a problemas complexos.

De acordo com a segunda lei de Newton, a força resultante $F_r(\bar{t})$ em um corpo é igual ao produto da massa m e sua aceleração resultante $\ddot{u}(\bar{t})$:

$$F_r(\bar{t}) = m\ddot{u}(\bar{t}) \tag{2.84}$$

Figura 20 – Sistema $k - c - m$ excitado e o diagrama de corpo livre.



Fonte: Adaptado de Soriano (2014).

Observando a Fig. 20 e com base na segunda lei de Newton, chega-se a equação de equilíbrio dinâmico:

$$P(\bar{t}) - F_k(\bar{t}) - F_c(\bar{t}) = m\ddot{u}(\bar{t}) \tag{2.85}$$

onde, P, F_k e F_c são, respectivamente, força de excitação, força elástica e força de amortecimento. Assume-se:

$$F_k = ku(\bar{t}) \tag{2.86a}$$

$$F_c = c\dot{u}(\bar{t}) \tag{2.86b}$$

onde c e k são os coeficientes de amortecimento e rigidez, respectivamente.

Dessa maneira, equação diferencial do movimento pode ser formulada reescrevendo a Eq. (2.85) da seguinte maneira:

$$m\ddot{u}(\bar{t}) + c\dot{u}(\bar{t}) + ku(\bar{t}) = P(\bar{t}) \tag{2.87}$$

2.4.2 Vibração Livre Não Amortecida

Quando o sistema não possui amortecimento e nem carga externa aplicada, o problema é definido como vibração livre não amortecida. Neste cenário específico, a

equação diferencial do movimento, Eq. (2.87), simplifica-se para forma homogênea:

$$m\ddot{u}(\bar{t}) + ku(\bar{t}) = 0 \tag{2.88}$$

A solução geral da Eq. (2.88) é dada por:

$$u(\bar{t}) = \alpha_1 \cos \omega_n \bar{t} + \alpha_2 \sin \omega_n \bar{t} \tag{2.89}$$

onde o termo ω_n é a frequência natural da estrutura, os coeficientes α são as constantes a serem obtidas.

De acordo com Soriano (2014), existem duas maneiras de encontrar os coeficientes α da Eq. (2.89):

1. Aplicando a condição de contorno no instante inicial $\bar{t}_0 = 0$, resulta-se em $\alpha_1 = u_0$ e $\alpha_2 = \frac{\dot{u}_0}{\omega_n}$, então:

$$u(\bar{t}) = u_0 \cos \omega_n \bar{t} + \frac{\dot{u}_0}{\omega_n} \sin \omega_n \bar{t} \tag{2.90}$$

   onde, u_0 e $\dot{u}_0$ são, respectivamente, deslocamento inicial e velocidade inicial.

2. Entretanto, as posição e a velocidade iniciais dependem da amplitude $\bar{\alpha}$ e do ângulo de fase $\bar{\phi}$, ou seja, $\alpha_1 = \bar{\alpha} \cos \bar{\phi}$ e $\alpha_2 = \bar{\alpha} \sin \bar{\phi}$, então:

$$u(\bar{t}) = \bar{\alpha} \cos (\omega_n \bar{t} - \bar{\phi}) \tag{2.91}$$

   Com base nas Eqs. (2.90) e (2.91), chega-se as seguintes expressões:

$$\bar{\alpha} = \sqrt{u_0^2 + \frac{\dot{u}_0^2}{\omega_n^2}} \tag{2.92a}$$

$$\bar{\phi} = \tan^{-1} \frac{\dot{u}_0}{u_0 \omega_n} \tag{2.92b}$$

   A função da aceleração pode ser obtida pela segunda derivada da Eq. (2.90):

$$\ddot{u}(\bar{t}) = -\omega_n^2 u(\bar{t}) \tag{2.93}$$

   Substituindo a Eq. (2.93) na (2.88), define-se a frequência natural ω_n:

$$\omega_n = \sqrt{\frac{k}{m}} \tag{2.94}$$

   Reescrevendo a Eq. (2.88) com base na Eq. (2.94):

$$\ddot{u}(\bar{t}) + \omega_n^2 u(\bar{t}) = 0 \tag{2.95}$$

2.4.3 Frequências e Modos Naturais de Vibração

Na análise dinâmica estrutural, as frequências e modos naturais de vibração têm grande importância nas tomadas de decisões e formulações. São características dinâmicas da estrutura.

Em estruturas de múltiplos graus de liberdades, as frequências e modos de vibração só dependem das matrizes restringidas de rigidez e massas $\mathbf{K}$ e $\mathbf{M}$ respectivamente (SORIANO, 2014). Reescrevendo a Eq. (2.88) em forma matricial, tem-se:

$$\mathbf{M}\ddot{\mathbf{U}}(\bar{t}) + \mathbf{K}\mathbf{U}(\bar{t}) = 0 \tag{2.96}$$

Com o pressuposto de que seja dada uma pertubação inicial ao modelo, de forma semelhante a Eq. (2.91), chega-se a solução harmônica de deslocamentos nodais:

$$\mathbf{U}(\bar{t}) = \begin{Bmatrix} \bar{\alpha}_{1j} \\ \bar{\alpha}_{2j} \\ \vdots \\ \bar{\alpha}_{nj} \end{Bmatrix} \cos(\omega_j \bar{t} - \bar{\phi}_j) \tag{2.97}$$

de forma compacta,

$$\mathbf{U}(\bar{t}) = \hat{\alpha}_{\mathbf{j}} \cos(\omega_j \bar{t} - \bar{\phi}_j) \tag{2.98}$$

onde, $\hat{\alpha}_j$ é o j-ésimo modo natural de vibração, ω_j é a correspondente frequência natural de vibração livre e $\bar{\phi}_j$ é o correspondente ângulo de fase.

A substituição da Eq. (2.98) na (2.96), conduz ao sistema de n-equações algébricas homogêneas:

$$(\mathbf{K} - \mathbf{M}\omega_j^2)\hat{\alpha}_{\mathbf{j}} \cos(\omega_j \bar{t} - \bar{\phi}_j) = \mathbf{0} \tag{2.99}$$

Essa função só é satisfeita para qualquer valor de $\bar{t}$, se:

$$\mathbf{K}\hat{\alpha}_{\mathbf{j}} = \omega_j^2 \mathbf{M}\hat{\alpha}_{\mathbf{j}} \tag{2.100}$$

A Eq. (2.100) representa um problema de autovalor generalizado de n-soluções não triviais, em que ω_j^2 é o autovalor e $\hat{\alpha}_{\mathbf{j}}$ é o autovetor.

2.4.4 Matriz de Massa em Elementos Finitos

Uma matriz de massa pode ser discreta ou consistente. É qualificada como discreta quando se adota procedimento simplificado de consideração de massa concentrada nos

pontos nodais, o que tem a vantagem de fornecer matriz diagonal, simplificando o tratamento numérico. É qualificada como consistente quando obtida com a formulação MEF, (SORIANO, 2014, p. 181).

De acordo com Sydenstricker et al. (1995), a matriz de massa concentrada do elementos de casca é:

$$\mathbf{M_L^e} = \frac{\rho A t}{3}\mathbf{diag}\begin{bmatrix}1 & 1 & 1 & 0 & 0 & 0 & | & 1 & 1 & 1 & 0 & 0 & 0 & | & 1 & 1 & 1 & 0 & 0 & 0\end{bmatrix} \tag{2.101}$$

onde as massas são concentradas nos graus de liberdade de translação u_i, v_i e w_i.

De forma similar, a matriz massa concentrada para o elemento tetraedro é:

$$\mathbf{M_L^e} = \frac{\rho V}{4}\mathbf{diag}\begin{bmatrix}1 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 1 & | & 1 & 1 & 1\end{bmatrix} \tag{2.102}$$

Para o cálculo da matriz de massa consistente, é preciso compreender o conceito de vetor de força de corpo. São forças originadas do próprio peso real do corpo (forças gravitacionais), velocidade angular (força centrífuga), forças inerciais na dinâmica ou forças magnéticas. A formulação completa da expressão vetor de força de corpo, de forma compacta, é apresentada abaixo:

$$\mathbf{f} = \int_V \mathbf{H^T} f_e dV \tag{2.103}$$

onde, $\mathbf{H}$ representa as funções de forma do elemento e f_e é a força efetiva de corpo.

Aplicando o princípio de D' Alembert, para o cálculo da matriz de massa consistente, adota-se que a força efetiva é igual a força peso do corpo, ou seja:

$$f_e = \rho \ddot{u} \tag{2.104}$$

Sabe-se que:

$$\ddot{u} = \mathbf{H}\ddot{\mathbf{U}} \tag{2.105}$$

Aplicando as Eqs. (2.104) e (2.105) na (2.103), tem-se:

$$\mathbf{f} = \int_V \rho \mathbf{H^T H}\ddot{\mathbf{U}} dV \tag{2.106}$$

De acordo com a Segunda Lei de Newton expressa na Eq. (2.84), sabe-se:

$$\mathbf{f} = \mathbf{M}\ddot{\mathbf{U}} \tag{2.107}$$

Portanto, a matriz de massa consistente é dada por:

$$\mathbf{M_c^e} = \int_V \rho \mathbf{H^T H} dV \tag{2.108}$$

Nesta dissertação foram implementadas nos diversos códigos, as matrizes de massas do tipo discreta.

3 ASPECTOS COMPUTACIONAIS

Desde o início, foi decidido que o projeto descrito nesta dissertação não seria apenas um script regular de elementos finitos que limitasse o seu uso em aplicações simples por não ter viabilidade em problemas complexos. O grupo MAMNE sempre teve a ambição de desenvolver um ambiente prático com interface gráfica que pudesse integrar todos os códigos por ele desenvolvidos. Sabe-se que para construir um software dessa magnitude, com todos os elementos disponíveis, requer tempo excessivo, então a ênfase foi colocada em desenvolver plugins escritos em Python e integrados à plataforma Salome (2017). Neste caso em uma versão inicial para análise modal de modelos com diferentes tipos de elementos finitos acoplados.

Neste capítulo são abordadas as ferramentas computacionais utilizadas que se dividem em três etapas: pré-processamento, processamento e pós-processamento, integração Python/Salome, os plugins desenvolvidos e denominados de Code_Mamne e os principais algoritmos.

A etapa referente ao pré e pós-processamento é realizada com o programa Salome, onde é feita desde a modelagem geométrica à geração de malha. Já os solucionadores são executados pelos os plugins (desenvolvidos na dissertação).

3.1 Visão Geral da Plataforma Salome

Salome é um software livre, lançado sob a licença GNU/LGPL, que disponibiliza uma multiplataforma genérica para realizar simulações numérica de pré e pós-processamento em vários domínios científicos.

A plataforma surgiu no ano 2000 a partir das necessidades de encontrar uma solução multifísica e facilitar a integração de soluções de cálculo específicas dentro do software. Atualmente, a Commissariat d'Èngies Atomique (CEA) e a Électricité de France (EDF), ambas da França, utilizam o Salome para realizar uma ampla gama de simulações, que são tipicamente relacionadas a equipamentos industriais em usinas de energia nucleares, eólicas, barragem e outras.

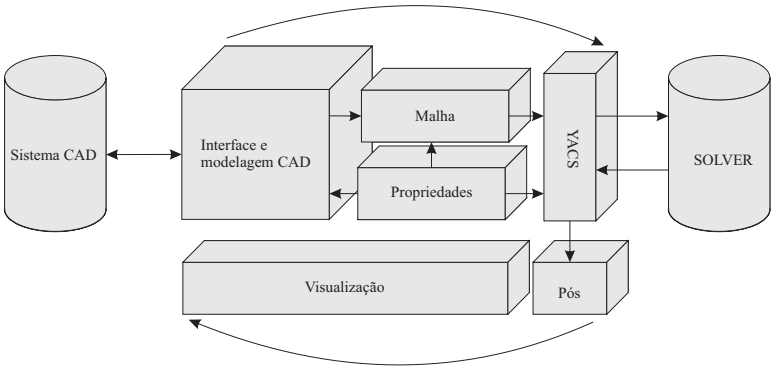
A arquitetura do Salome é dividida em duas partes: Kernel e vários módulos. A plataforma utiliza dois idiomas: Python e C++. Todas as funcionalidades do Kernel e dos módulos precisam ser exportadas para o Python. Assim, a plataforma pode ser usada através de "scripts", através da interface gráfica do usuário (GUI) ou através de uma combinação de ambos. Esta regra permite que Salome seja usada por diferentes categorias de usuários ou em diferentes situações (RIBES; CAREMOLI, 2007).

Os módulos padrões são:

- GUI: interface de usuário homogênea para Salome, cada módulo pode integrar sua própria interface.
- Geometry: funcionalidades em criar, importar e corrigir qualquer modelo geométrico computacional (IGES, STEP, BRP).
- Mesh: possui compatibilidade com modelo CAD e disponibiliza algoritmos de geração de malhas padrões ou externos (como plugin). Pode-se importar e exportar nos formatos MED, UNV, STL, CGNS, DAT, GMF e SAUVE.
- Med: descrição e gerenciamento de propriedades físicas e condições de contorno.
- Yacs: uma maneira de descrever um esquema computacional envolvendo o acoplamento multi-solver.
- Paraview: Um aplicativo de visualização e análise de dados. Usa o VTK como o mecanismo de processamento e renderização de dados.

Pode-se estender ainda mais o software através da criação de módulos adicionais ou plugins, tais como: algoritmos, solvers padrão ou específico, etc. O objetivo é tornar a plataforma adequada a um dado campo específico de pesquisa, como cálculos nucleares, mecânica, etc. A Fig. 21 mostra a interação entre os diferentes módulos e a integração de códigos de simulações numéricas:

Figura 21 – Visão geral da plataforma Salome.



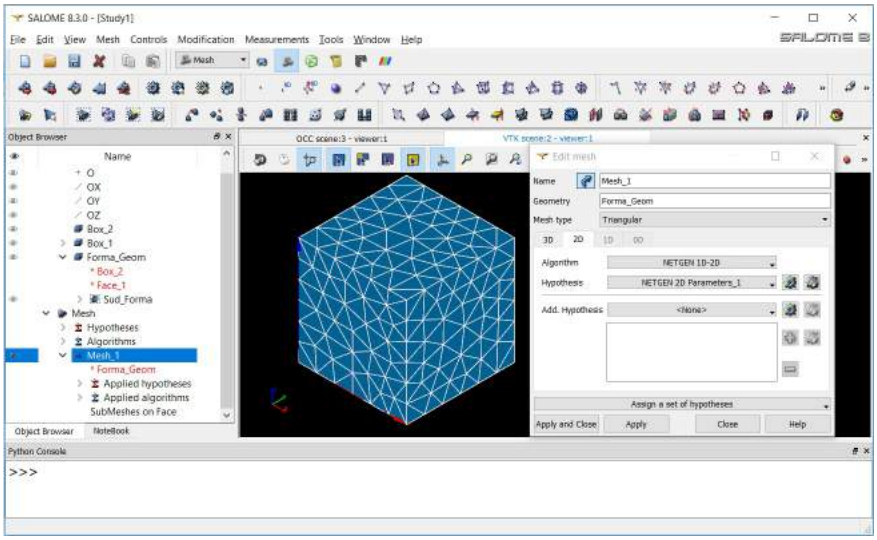
Fonte: Adaptado de Ribes e Caremoli (2007).

3.1.1 Geração de Malhas com o Salome

O módulo Mesh do Salome fornece várias maneiras de criar malhas. A principal é a baseada na forma geométrica produzida no módulo Geometry. Desta forma, implica a seleção de um objeto geométrico e parâmetros de malha através de algoritmos e hipóteses (Fig. 22).

Os algoritmos citados representam implementações de técnicas diversas de geração de malhas, como a discretização por elementos triangulares ou tetraedros. Já as hipóteses representam condições de contorno que são consideradas pelos algoritmos de geração de malha, como por exemplo, gerenciar o nível de refinamento da malha resultante.

Figura 22 – Geração de malha triangular uniforme sobre a superfície do cubo com o módulo Mesh.

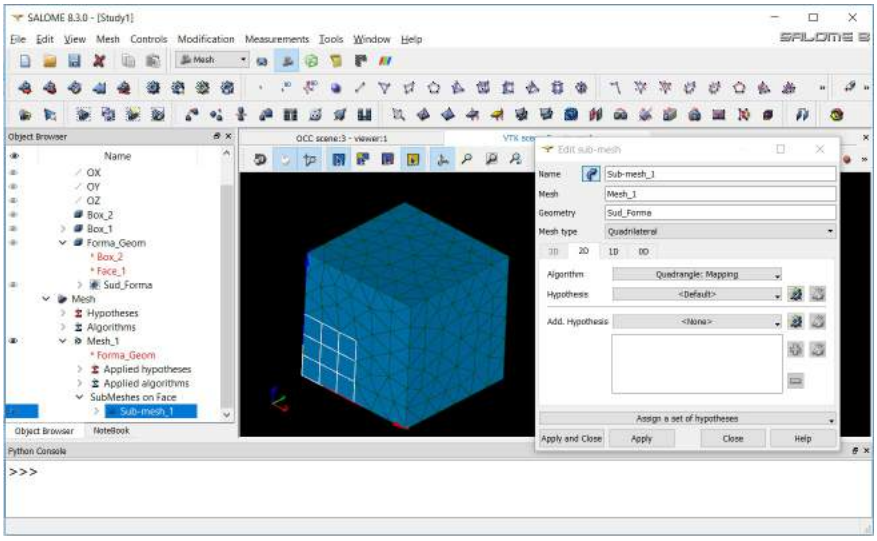


Fonte: O Autor (2018).

Dentre desses algoritmos, pode-se gerar malhas tridimensionais através de malhas bidimensionais, construção de malha composta de outras malhas, entre outros.

O módulo Mesh também disponibiliza a opção de criar sub-malhas, permitindo discretizar algumas sub-formas da forma principal geométrica usando os parâmetros de malha que diferem daqueles usados para outras sub-formas (Fig. 23). Os parâmetros referenciados à malha são chamadas de globais e aqueles referenciados às sub-malhas são considerados locais.

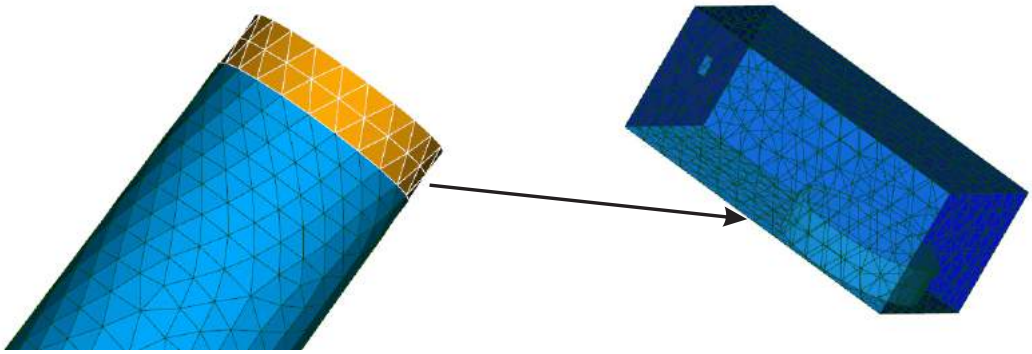
Figura 23 – Geração de uma sub-malha com elementos quadrilaterais em malha com elementos triangulares.



Fonte: O Autor (2018).

Outro algoritmo útil para o desenvolvimento de diferentes malhas com certas regiões sobrepostas por exemplo, é a técnica de importar elementos unidimensional ou bidimensional de outra malha permitindo definir a malha de um objeto geométrico importando elementos adequadamente localizados de outra malha (Fig. 24). Os elementos da malha de referência importados para outra malha devem estar contidos em grupos, ou seja, os elementos de origem devem cobrir totalmente a forma ou sub-forma geométrica em malha.

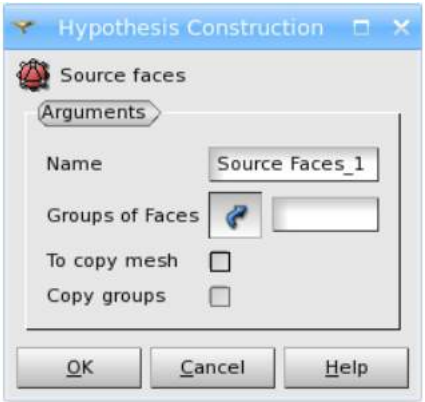
Figura 24 – Importando os elementos triangulares da malha à esquerda como faces dos elementos TE4 da malha de volume à direita.



Fonte: O Autor (2018).

Para aplicar este algoritmo, seleciona-se a face geométrica a ser gerada (indicada no campo do Geometry da caixa de diálogo Create Mesh ou Sub-mesh), em seguida, seleciona-se a opção **Import 1D-2D Elements from Another Mesh** na lista de algoritmos e clica-se no botão Adicionar Hipótese, a seguinte caixa de diálogo será exibida, Fig. 25, onde serão informados quais elementos são importados de outra malha.

Figura 25 – Caixa de diálogo de definição do tipo e grupo de elementos a serem importados.



Fonte: O Autor (2018).

3.2 Python

A linguagem Python foi criada por Guido van Rossum em 1991. Atualmente é aberta e gerenciada pela organização sem fins lucrativos Python Software Foundation.

Python prioriza a legibilidade do código e possui bibliotecas, módulos e frameworks padrões e desenvolvidos por terceiros.

Por ser uma linguagem interpretada, ela pode ser executada em qualquer sistema operacional que possui interpretador Python. Quanto à aceitação da linguagem no mercado, possui uma imensa comunidade ao redor do globo. Portanto dispõe de bibliotecas complexas e funcionais aos problemas em questão.

Problemas de elementos finitos envolvendo milhões de graus de liberdade requerem operações matriciais gigantescas, necessitando de matrizes esparsas e compilações visando otimizar a performance dos processamentos. Estes recursos estão disponíveis nas bibliotecas NumPy, SciPy e Numba.

NumPy é o pacote fundamental para a computação científica com Python e licenciado sob a licença (BSD). Possui ferramentas úteis em operações matriciais. Assim, suporta arrays e matrizes multidimensionais, e possui um enorme repertório de funções matemáticas para operar nesses conjuntos.

Já o SciPy é uma biblioteca de código aberto, também sob a licença (BSD), que trabalha em conjunto com o NumPy, diferenciando-se por possuir um repertório de funções bem mais complexos e variados, como: otimização, álgebra linear, integração, interpolação, solucionadores de equações diferenciais e outras tarefas comuns em ciência e engenharia.

O SciPy, apesar de escrito em uma linguagem interpretada como Python, apresenta uma performance de compilação rápida. Na verdade, muito do SciPy é uma fina camada de código em cima de rotinas científicas que estão disponíveis gratuitamente em um repositório chamado NetLib, com algoritmos científicos incrivelmente valiosos e robustos, em sua maioria deles escritos em C e Fortran.

Processar grandes quantidades de dados utilizando loops em Python pode ser 300 vezes mais lento que o mesmo código escrito em C, C++ e Fortran. Uma das alternativas para este problema é a biblioteca Numba, uma fonte aberta que permite acelerar aplicativos com funções de alto desempenho escritas diretamente em Python, principalmente com o NumPy, pois possuem uma integração perfeita. Pode ser executado nativamente pelo CPU ou GPU em tempo de execução. Isto acontece por decoração-funções do Python, que permite aos usuários criar funções nativas para diferentes tipos de entrada, obtendo resultados de desempenhos similares aos das linguagens C, C ++ e Fortran.

Abaixo segue uma comparação entres os códigos Python escritos em NumPy e Numba para resolução da seguinte equação:

$$\mathbf{X} = \mathbf{X} + 2\mathbf{Y} \tag{3.1}$$

onde, $\mathbf{X}$ e $\mathbf{Y}$ são vetores colunas com milhares de elementos.

Foram processados dez vezes cada código desenvolvido e registrado o tempo médio

de processamento para matrizes com dimensões 10^6, 10^7 e 10^8 (Fig. 26). A versão do Python utilizado é a 2.7.15 para o sistema operacional Windows 10 e a máquina utilizada é um Intel(R) Corel(TM) i7-7500U CPU @ 2.70GHz com 16 GB de RAM

Tabela 1 – Código NumPy

```
import numpy as np
import time as time

n = np.array( int(1e8))

X = np.ones(n, dtype=np.float)
Y = np.ones(n, dtype=np.float)
t0 = time.time()
X += Y; X += Y;
t1 = time.time()
run_time = t1 - t0
```

Fonte: O Autor (2018).

Tabela 2 – Código NumPy + Numba

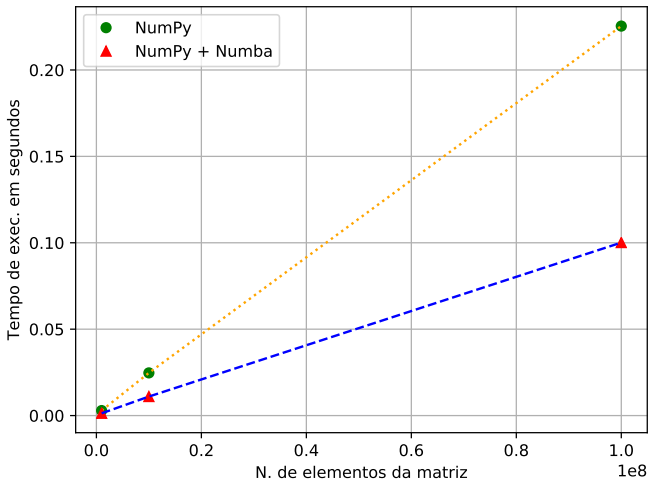
```
import numpy as np
import numba
import time as time

@numba.vectorize(["float64(float64,float64)"],
        nopython=True,target='parallel')
def add2_par(x, y):
        return x + 2 * y
n = np.array( int(1e8))

X = np.ones(n, dtype=np.float)
Y = np.ones(n, dtype=np.float)
t0 = time.time()
add2_par(X, Y, out=X)
t1 = time.time()
run_time = t1 - t0
```

Fonte: O Autor (2018).

Figura 26 – Gráfico: tempo de execução em segundos x número de elementos da matriz.

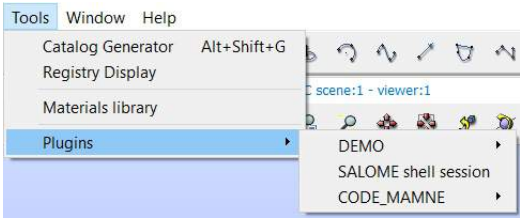


Fonte: O Autor (2018).

3.2.1 Integração Python-Salome

O Gerenciador de Plugins Python do Salome permite ao usuário final que consiga estender a plataforma com funções personalizadas escritas com bibliotecas Python. O menu padrão para os plugins ficam em Tools -> Plugins (Fig. 27):

Figura 27 – Menu padrão dos plugins na plataforma Salome.



Fonte: O Autor (2018).

A importância da interface gráfica (ou GUI de Graphical User Interface) está relacionado na interação usuário com a máquina por meio de janelas, ícones, menus e outros. Diferentemente de linhas de comando, onde o usuário precisa digitar cada passo desejado em um terminal, o GUI oferece várias opções que são visíveis e que podem ser apontadas e selecionadas pelo usuário.

Existem várias ferramentas computacionais que facilitam o desenvolvimento de interfaces gráficas. Dentre as ferramentas mais populares e genéricas pode-se citar o Qt5, um framework escrito em C++ de multiplataforma que implementam Interface de Programação de Aplicações (APIs) de alto nível para o desenvolvimento de aplicações desktop. A escolha da ferramenta Qt5 dá-se pelo fato do mesmo ser distribuído sob licença aberta GNU/LGPL, ser bem documentada, ter boa legibilidade, possuir ferramentas

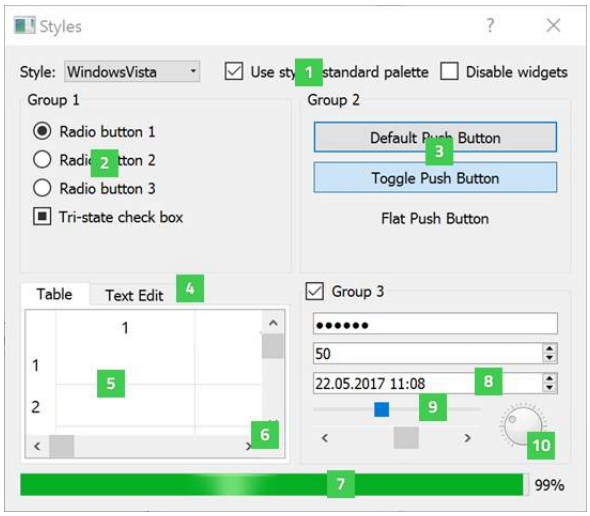
robustas para qualquer tipo de aplicativo e por possuí conjunto de extensões do Qt5 para Python (PyQt5).

Figura 28 – Conjunto de extensões do Qt5 para Python (PyQt5).

Fonte: O Autor (2018).

Para se desenvolver uma interface gráfica é necessário definir o conjunto de elementos gráficos que comporão a interface final, como esses elementos devem ser colocados (posição de cada elemento na janela), que eventos cada elemento deve responder e as rotinas para tratar cada evento.

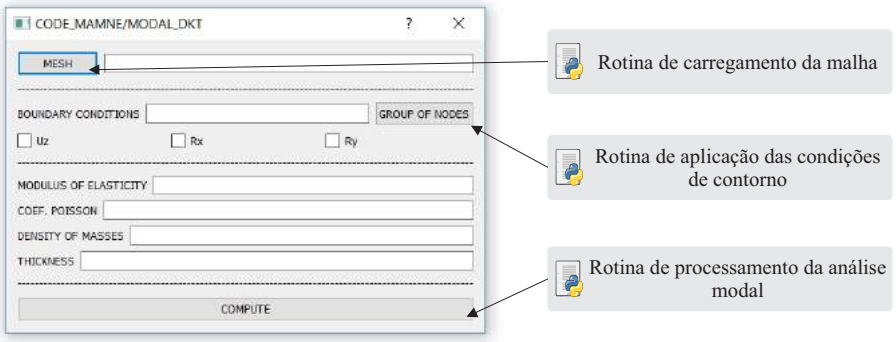
Figura 29 – Exemplo de conjuntos variados de elementos posicionados no layout da janela de um aplicativo aleatório.



Fonte: O Autor (2018).

Portanto, como os plugins desta dissertação se dão em janelas de diálogo (onde o usuário informa o modelo de análise e os parâmetros), toda a interface foi elaborada usando QDialog, a classe base de janelas de diálogo do PyQt5. As rotinas de processamento e pós-processamento de análises modais são implementadas por estruturas orientadas a objetos através de eventos definidos por clicks de botões devidamente posicionado nos layouts das janelas das interfaces desenvolvidas.

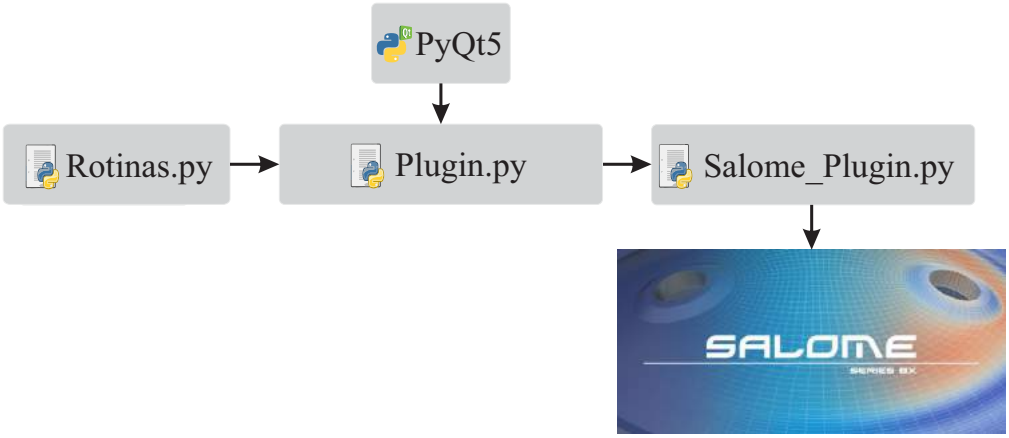
Figura 30 – Esquema de implementação das rotinas à interface gráfica do plugin Modal_DKT.



Fonte: O Autor (2018).

Embora todas as construções das interfaces gráficas nessa dissertação se deram por linhas de comandos, a comunidade Qt disponibiliza a ferramenta Qt Designer, própria para projetar e construir GUIs. Ele permite que você crie widgets, diálogos ou janelas principais usando formulários na tela e uma interface simples de arrastar e soltar. Permite ao usuário visualizar seus protótipos antes mesmo de escrever qualquer código.

Figura 31 – Esquema geral de integração Salome-Python



Fonte: O Autor (2018).

O arquivo Salome_Plugin.py (Fig. 31) tem a função de declarar o plugin desenvolvido à plataforma Salome. A forma geral do arquivo Salome_Plugin.py é:

Tabela 3 – Estrutura padrão do arquivo do tipo Salome_Plugin.py

```
import salome_pluginsmanager

def meuplugin1(context):
        ...
        # Plugin 01
        ...

def meuplugin2(context):
        ...
        # Plugin 02
        ...

# Declarando os plugins para o Gerenciador
salome_pluginsmanager.AddFunction(
        'Plugin 01',
        'Executa o plugin 01',
        meuplugin1)
salome_pluginsmanager.AddFunction(
        'Plugin 02',
        'Executa o plugin 02',
        meuplugin2)
...
```

Fonte: O Autor (2018).

O programador define uma função que implemente o plugin e através da estrutura padrão (Tab. 3), gerencie-o no próprio Salome. A forma de implementação pode ser considerada variável, entretanto, é aconselhável usar o script da Tab. 3 como modelo.

É importante ressaltar alguns objetos usados no script (Tab. 3), o objeto **context** é automaticamente instanciado pelo gerenciador quando o plugin é solicitado na plataforma. O mesmo fornece pelo menos os atributos **context.salome.study** e **context.salome.sg**, atributos relacionados, respectivamente, ao módulo Kernel e GUI do Salome (citados na sessão 3.1). Eles permitem, por exemplo, selecionar uma malha e exibir algumas informações relacionadas a ela.

Alguns comandos úteis e importantes no desenvolvimento do plugin, são:

1. **context.salome.sg.getAllSelected()**: para obter a lista de objetos selecionados no navegador de objetos.

2. **objId = context.salome.sg.getSelected(0)**: para obter número de identificação do primeiro objeto selecionado no navegador de objetos.

3. **salomeObj = context.salome.study.FindObjectID(objId).GetObject()**: para recuperar o objeto com o seu número de identificação no Salome. Pode ser um objeto do tipo Geometry, Mesh ou qualquer outro objeto de módulo.

Uma vez escrito, este script Salome_Plugin.py deve ser movido para um lugar específico no seu sistema de arquivos, onde Salome é programado para procurar por plugins. Os diretórios possíveis são (na ordem de pesquisa):

1. Quando o plugin é desenvolvido na estrutura de um módulo específico do Salome, o arquivo do tipo Salome_Plugin.py tem que ser movido para o diretório <ROOT_DIR> -> share -> salome -> plugins -> <module_name>, onde <ROOT_DIR> é o diretório de instalação raiz do módulo e <module_name> é o nome do módulo em letras minúsculas.

Figura 32 – Caminho do diretório do módulo Smesh

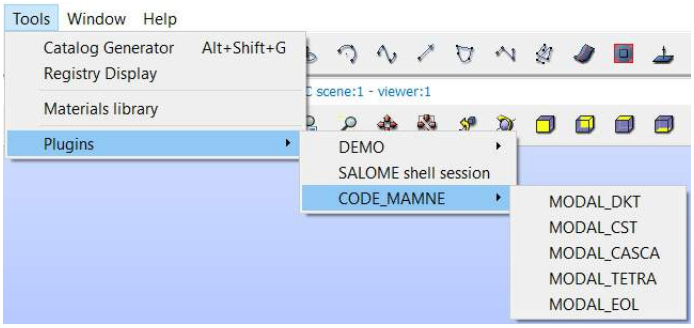
Fonte: O Autor (2018).

2. Quando o intuito do desenvolvimento é pessoal, o arquivo Salome_Plugin.py poder ser colocado no diretório /.config/salome/Plugins.

3.3 Plugins Code_Mamne

Foram desenvolvidos cinco segmentos de plugins com menus bastantes autoexplicativos e fáceis de usar permitindo análises com diferentes elementos finitos atendendo ao modelo de interesse do usuário (Fig. 33):

Figura 33 – Menu padrão dos plugins Code_Mamne.



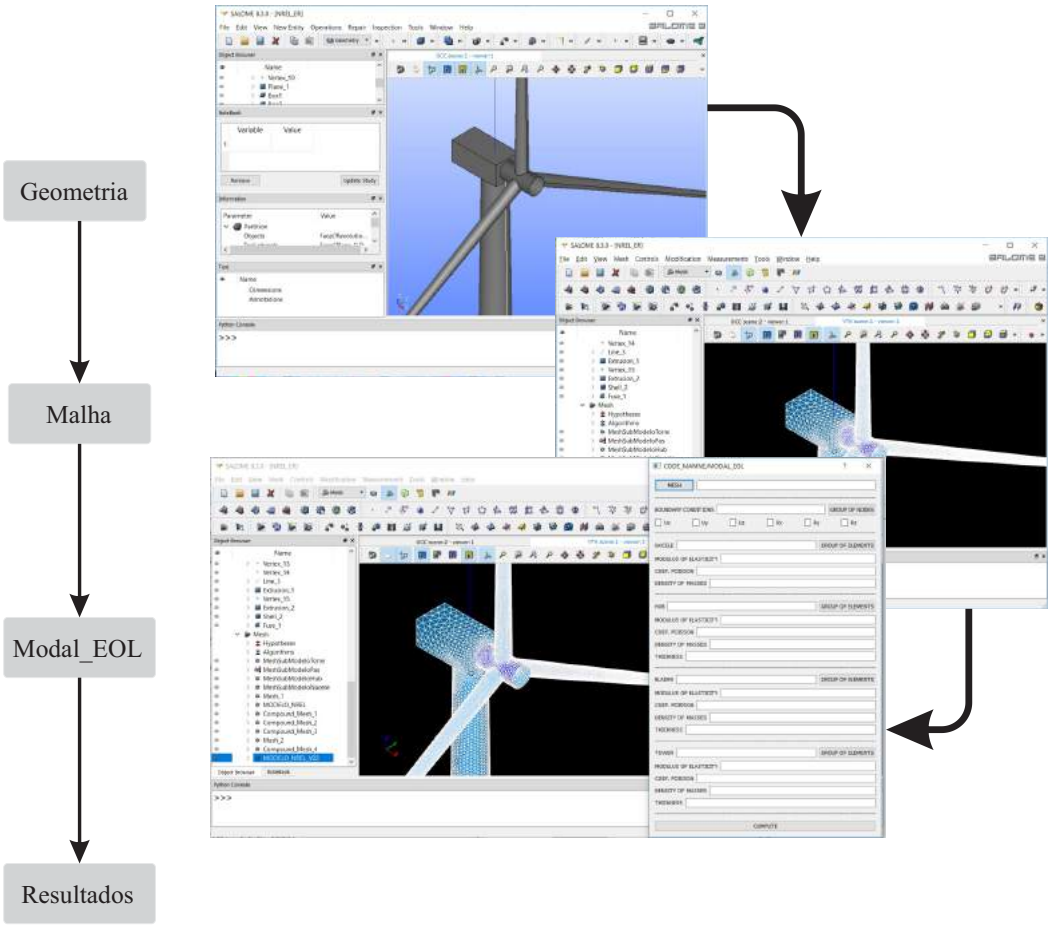
Fonte: O Autor (2018).

- Code_Mamne/Modal_CST: para análise modal com elementos finitos CST.
- Code_Mamne/Modal_DKT: para análise modal com elementos finitos DKT.
- Code_Mamne/Modal_CASCA: para análise modal com elementos finitos lineares de casca triangular.

- Code_Mamne/Modal_TETRA: para análise modal com elementos finitos tetraédricos.
- Code_Mamne/Modal_EOL: projetado para cálculo específico de turbinas eólicas em que o modelo envolve os submodelos Nacele (sólido), Hub (superfície), Blades ou Pás (superfície) e Torre (superfície).

Toda a parte da modelagem geométrica e da malha são feitas pelo usuário final com o Salome através dos módulos Geometry e Mesh, cabe ao mesmo verificar se o modelo está válido, como por exemplo a conformidade da malha. Uma vez alimentado o plugin de interesse com os dados do modelo de análise, dá-se início ao processamento e, em seguida, obtêm-se os resultados das frequências e modos de vibrações em dois arquivos nos formatos: .CSV e .VTK respectivamente (Fig. 34).

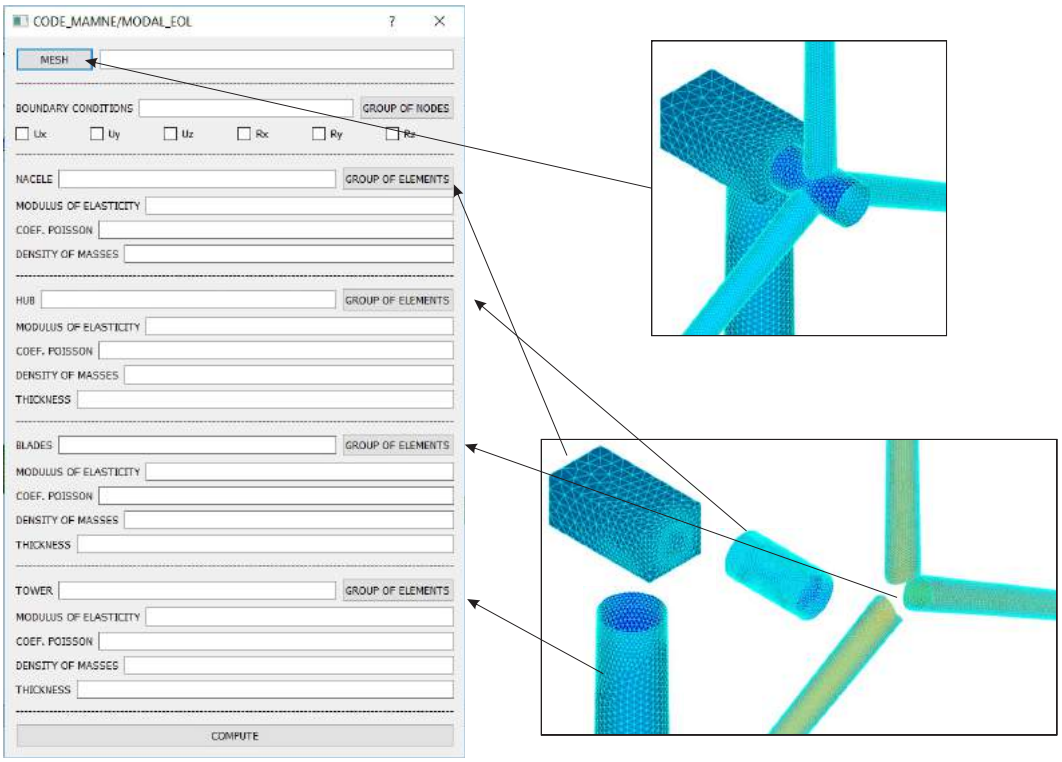
Figura 34 – Fluxograma do processo de análise com o plugin dentro do Salome.



Fonte: O Autor (2018).

Enfatizando o plugin Modal_EOL, como exemplo para entradas de dados, o usuário informa o modelo eólico através da malha com os submodelos acoplados, em seguida, são criados grupos específicos de pontos nodais para as condições de contorno e grupos de elementos de cada componente da turbina (Fig. 35).

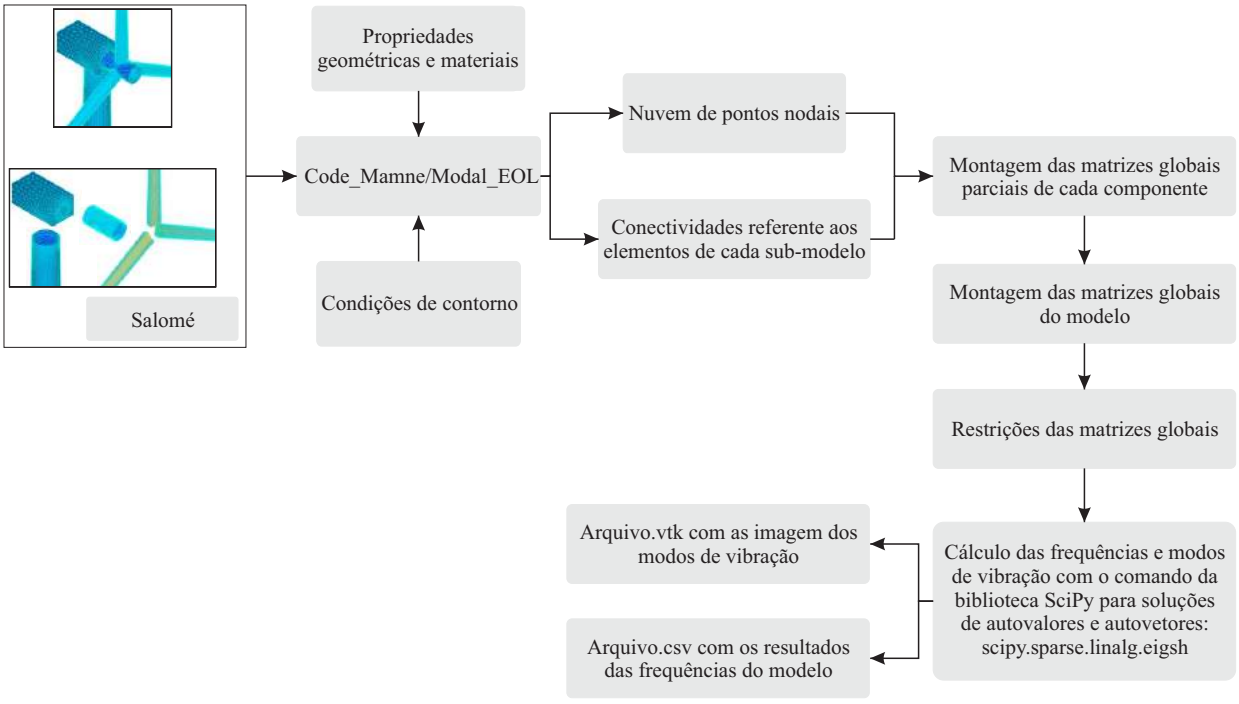
Figura 35 – Modal_EOL



Fonte: O Autor (2018).

Os parâmetros geométricos e materiais são informados nos seus específicos campos pelo usuário. Um esquema geral é apresentado na Fig. 36 de como se dá o processamento de um modelo eólico na análise modal.

Figura 36 – Esquema geral do Modal_EOL



Fonte: O Autor (2018).

3.4 Principais Algoritmos do Código

Os algoritmos clássicos de montagem das matrizes globais $\mathbf{K^g}$ e $\mathbf{M^g}$ são menos eficientes por apresentarem estruturas de repetições nas suas implementações. Na busca de melhorar a eficiência, foi escolhido a implementação baseada em vetorização na montagem das matrizes globais.

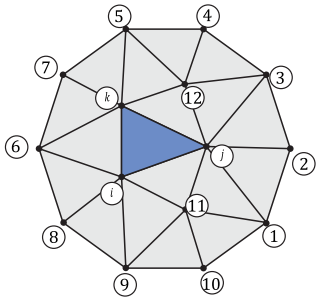
3.4.1 Montagem das Matrizes Globais de Elementos Finitos

A técnica adotada foi adaptada do artigo publicado por Cuvelier et al. (2013), que consiste em eliminar os loops.

Com a intenção de fornecer uma explicação bem clara e objetiva, foi escolhido um problema de elasticidade em membrana usando o elemento CST para montagem da matriz de rigidez $\mathbf{K^g}$.

Seja um domínio bidimensional discretizado por elementos triangulares (Fig. 37).

Figura 37 – Descrição da malha

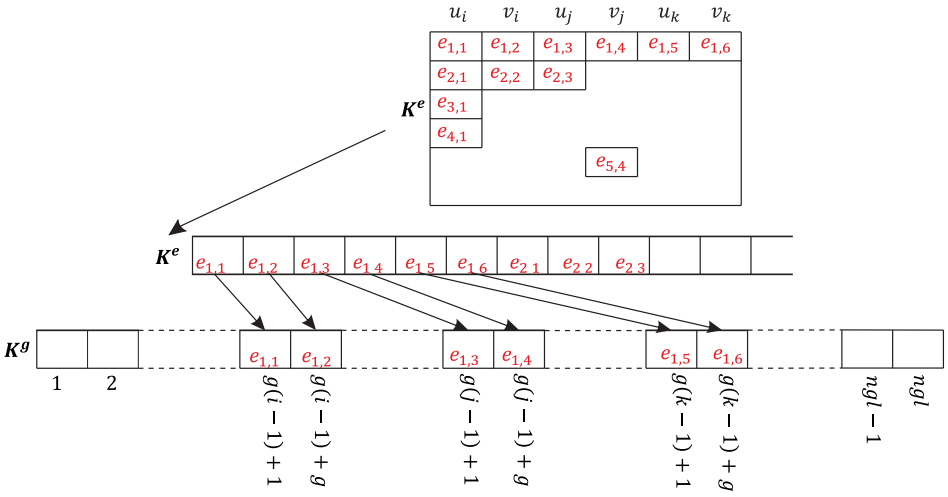


Fonte: O Autor (2018).

Na Fig. 37, os termos i, j e k representam os vértices do elemento e.

A ideia consiste em transformar a matriz de rigidez de cada elemento em vetor linha e realocar na matriz de rigidez global, também em vetor linha, com as posições previamente calculadas (Fig. 38).

Figura 38 – Inserção da matriz de rigidez de um elemento na matriz global.

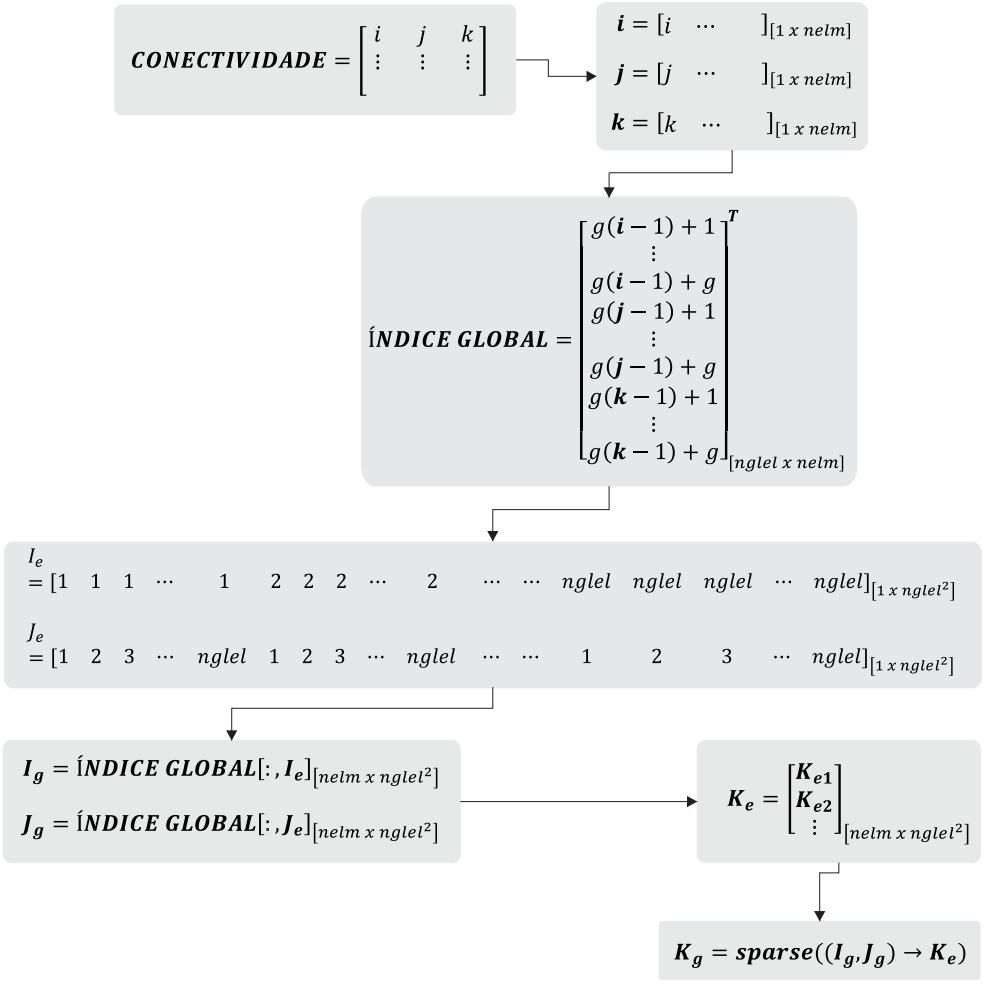


Fonte: O Autor (2018).

Na Fig. 38, os termos g e ngl representam, respectivamente, número de graus de liberdade por nó e número de graus de liberdade do sistema global.

A partir disso, segue na Fig. 39 o fluxograma do algoritmo de montagem da matriz de rigidez global de elementos CST, podendo ser ajustado para qualquer tipo de elemento, inclusive elementos tridimensionais:

Figura 39 – Algoritmo eficiente para montagem das matrizes globais



Fonte: O Autor (2018).

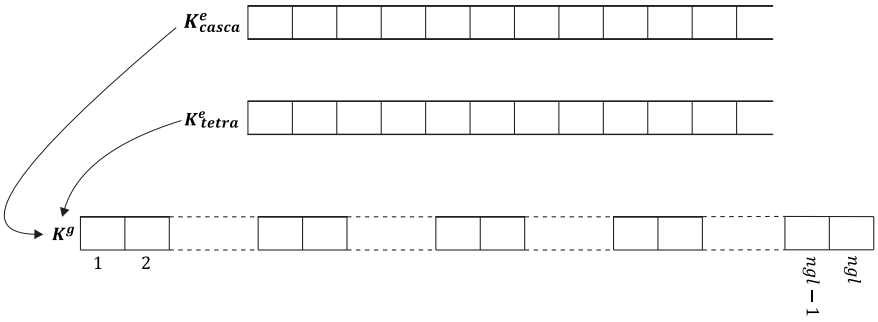
Na Fig. 39, os termos *nelem* e *ngle* são, respectivamente, número de elementos e de graus de liberdade por elemento.

3.4.2 Conexão entre Casca e Sólido

A técnica aplicada para a montagem das matrizes do sistema global com elementos de diferentes graus de liberdade foi em tratar especificadamente cada submodelo (ou tipo de elemento) com suas conectividades dependentes da mesma nuvem de pontos nodais.

Conforme foi mostrado na Eq. (2.82), pode ser observado a técnica de sincronização dos graus de liberdade entre os submodelos estabelecidos. Sendo assim, a montagem das matrizes do sistema global se dá pela as inserções de forma simultânea das matrizes de rigidez de todos os tipos de elementos envolvidos (Fig. 40).

Figura 40 – Inserções simultâneas das matrizes dos diferentes tipos de elementos no sistema global.



Fonte: O Autor (2018).

3.4.3 Código para Análise Modal

Conforme a Eq. (2.100), o cálculo das frequências naturais e modos de vibração expressa um problema de autovalores e autovetores:

$$\mathbf{K}\hat{\alpha}_\mathbf{j} = \omega_j^2 \mathbf{M}\hat{\alpha}_\mathbf{j} \tag{3.2}$$

onde ω_j^2 é o autovalor e $\hat{\alpha}_\mathbf{j}$ é o autovetor. As matrizes globais $\mathbf{K}$ e $\mathbf{M}$ são reduzidas por eliminação de linhas e colunas referentes aos graus de liberdade restringidos.

A solução de todos os autovalores e autovetores exige um esforço computacional considerável, principalmente em modelos com malhas bem refinadas, tendo como consequência uma enorme quantidade de graus de liberdade. Diante deste problema, foram adotadas funções otimizadas, que pudessem resolver a quantidade máxima de modos de vibração definida pelo usuário.

Esse problema foi resolvido com a implementação do ARPACK, Lehoucq et al. (1997), um pacote Fortran que fornece rotinas para encontrar rapidamente alguns autovalores e autovetores de grandes matrizes esparsas, simétricas ou não-simétricas. O pacote é projetado para calcular autovalores com recursos especificados pelo usuário, como aqueles da menor parte real ou menor magnitude.

Todas as funcionalidades fornecidas no ARPACK estão contidas em duas interfaces de alto nível na biblioteca SciPy, **scipy.sparse.linalg.eigs** e **scipy.sparse.linalg.eigsh**. A interface **eigs** fornece funções para encontrar os autovalores e autovetores de matrizes quadradas não simétricas reais ou complexas, enquanto a **eigsh** fornece funções para matrizes simétricas reais ou complexas hermitianas.

Como nesta dissertação as matrizes envolvidas são simétricas, foi utilizada a interface **eigsh** nos códigos desenvolvidos.

.

4 APLICAÇÕES BÁSICAS

São realizadas três aplicações básicas com o objetivo de validar os códigos desenvolvidos. A Tab. 4, apresenta o resumo das aplicações:

Tabela 4 – Resumo das Aplicações.

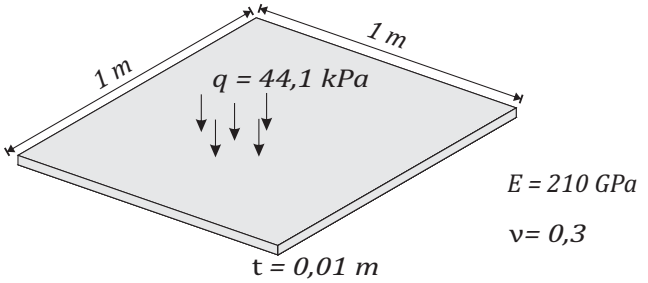
Aplicação	Análise	Elemento
Placa	Estática	DKT
Superfície Cônica	Modal	Casca de 3 Nós
Pórtico Espacial	Modal	Tetraedro de 4 Nós

Fonte: O Autor (2018).

4.1 Placa

Este exemplo foi retirado do trabalho de Fraga (2015), representa uma placa quadrada e é usado para estudar apenas a ação de flexão de placas com a discretização por elementos DKT. A placa está simplesmente apoiada nas bordas e submetida a uma carga de 44,1 kPa, Fig. 41.

Figura 41 – Propriedades geométricas e materiais do modelo numérico da placa.



Fonte: O Autor (2018).

4.1.1 Análise Estática Linear

Para análise estática linear do modelo numérico, foi gerada uma malha do tipo conforme no Salome com:

- 5716 elementos DKT,
- e um total de 2959 pontos nodais em todo o domínio da malha.

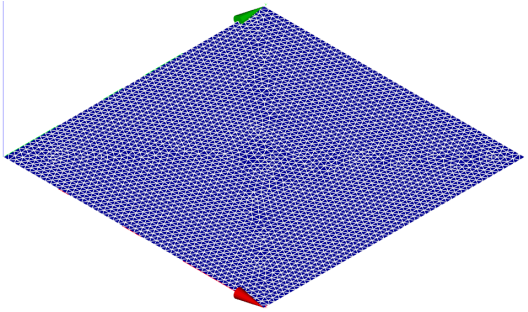
Paralelamente, o mesmo modelo foi analisado no software comercial ANSYS, onde foi criada uma malha do tipo conforme com:

- 5686 elementos SHELL63,

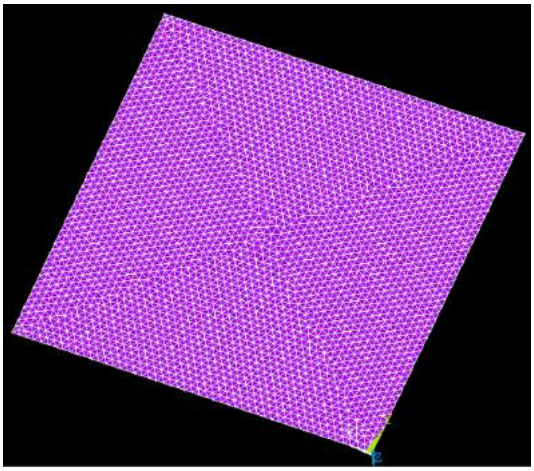
- e 2944 pontos nodais em todo o domínio da malha.

Figura 42 – Malhas do modelo numérico da placa.

(a) Malha conforme gerada no Salome



(b) Malha conforme gerada no ANSYS



Fonte: O Autor (2018).

A seguir, a Tab. 5 apresenta os resultados da deflexão e rotações máximas obtidos pelo código com o elemento DKT implementado e pelo ANSYS.

Tabela 5 – Resultados para o exemplo da placa quadrada, dados por ANSYS e o código em Python.

Deslocamentos	Código	ANSYS	ERRO (%)
Uz	$-0{,}009595m$	$-0{,}009579m$	0,17361
Rx	$0{,}03192rad$	$0{,}03187rad$	0,182
Ry	$0{,}03192rad$	$0{,}031871rad$	0,182

Fonte: O Autor (2018).

4.2 Superfície Cônica

Nesta aplicação, foi escolhido uma torre de turbina eólica com 87 metros de altura. A estrutura é de superfície cônica de paredes finas com espessura em 23 milímetros. Possui diâmetros de base e topo, respectivamente, 6 e 4 metros. Tipicamente de aço com o módulo de elasticidade de 210 GPa, coeficiente de Poisson de 0,3 e densidade de massa 7800 $\frac{kg}{m^3}$.

Figura 43 – Modelo numérico da torre.

Fonte: O Autor (2018).

4.2.1 Frequências e Modos Naturais de Vibração

Para análise modal do modelo numérico, foi gerada uma malha do tipo conforme no Salome com:

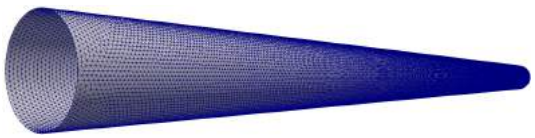
- 51161 elementos de casca triangular,
- e um total de 25643 pontos nodais em todo o domínio da malha.

Paralelamente, o mesmo modelo foi analisado no software comercial ANSYS, onde foi criada uma malha do tipo conforme com:

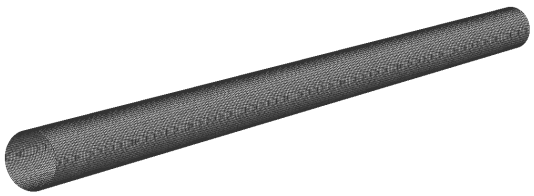
- 22048 elementos SHELL63,
- e 22086 pontos nodais em todo o domínio da malha.

Figura 44 – Malhas do modelo numérico da torre.

(a) Malha conforme gerada no Salome



(b) Malha conforme gerada no ANSYS



Fonte: O Autor (2018).

A seguir, serão apresentados os resultados das frequências naturais de translação obtidas pelo Modal_CASCA e software ANSYS, com os seus modos de vibração através dos arquivos .VTK gerado pelo plugin.

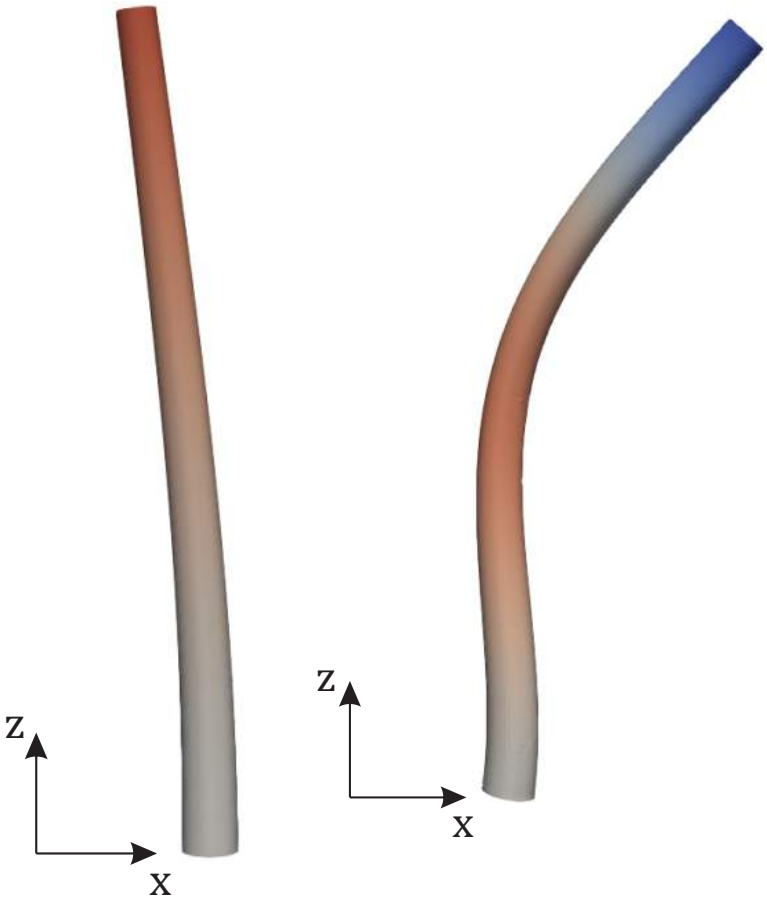
Tabela 6 – Frequências naturais do modelo numérico.

$\mathbf{f_n}$	Modal_CASCA (**Hz**)	ANSYS (**Hz**)
1 (Translação em x)	0,841	0,830
2 (Translação em y)	0,841	0,833
7 (Translação em x)	4,282	4,232
8 (Translação em y)	4,282	4,243

Fonte: O Autor (2018).

Figura 45 – Modos de vibração do modelo numérico.

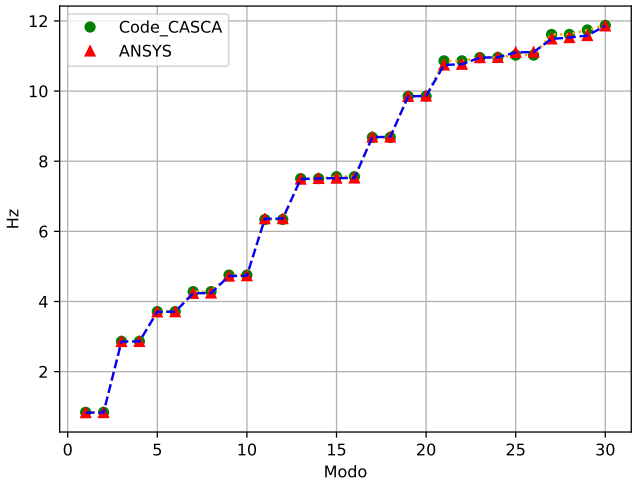
(a) $1^{ž}$ modo (b) $7^{ž}$ modo



Fonte: O Autor (2018).

Na Fig. 46 segue um gráfico com os 30 primeiros modos naturais de ambos os softwares com o intuito de comparação dos resultados.

Figura 46 – Os 30 primeiros modos de vibração: comparação de resultados entre o Modal_CASCA e ANSYS.

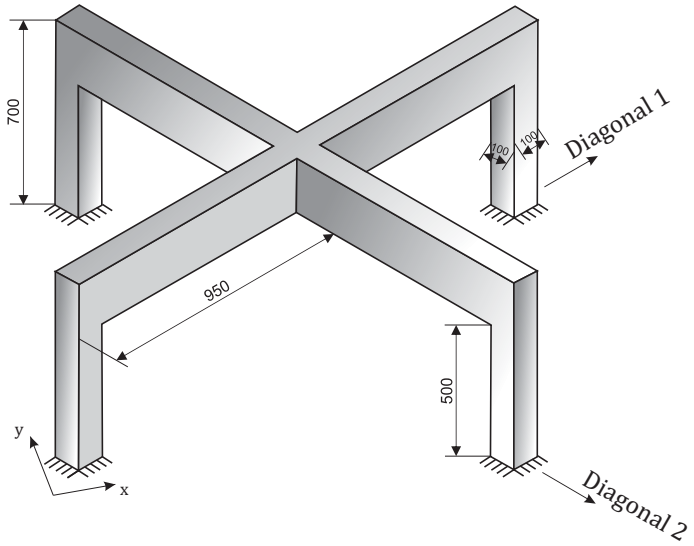


Fonte: O Autor (2018).

4.3 Pórtico Espacial

A estrutura em análise foi retirada do trabalho publicado por Barros (2016), e consiste em um pórtico espacial simétrico com vigas de seção transversal de um metro de largura por dois metros de altura e pilares com seção quadrada de um metro, tendo suas bases engastadas. A estrutura possui módulo de elasticidade de 20 GPa, coeficiente de Poisson de 0,2 e densidade de massa 2500 $\frac{kg}{m^3}$.

Figura 47 – Modelo numérico do pórtico espacial.



Fonte: Adaptado de Barros (2016).

4.3.1 Frequências e Modos Naturais de Vibração

Para análise modal do modelo numérico, foi gerada uma malha do tipo conforme no Salome com:

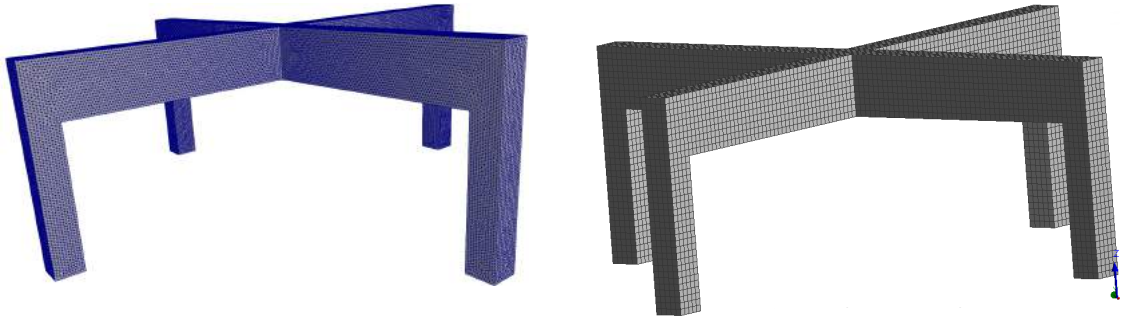
- 382147 elementos tetraedros de 4 nós,
- e um total de 83973 pontos nodais em todo o domínio da malha.

Paralelamente, o mesmo modelo foi analisado no software comercial ANSYS, onde foi criada uma malha do tipo conforme com:

- 12447 elementos SOLID186,
- e 16666 pontos nodais em todo o domínio da malha.

Figura 48 – Malhas do modelo numérico pórtico espacial.

(a) Malha conforme gerada no Salome (b) Malha conforme gerada no ANSYS



Fonte: Adaptado de Barros (2016).

A seguir, serão apresentados os resultados das frequências naturais de translação obtidos pelo Modal_TETRA e software ANSYS, com os seus modos de vibração através dos arquivos .VTK gerado pelo plugin.

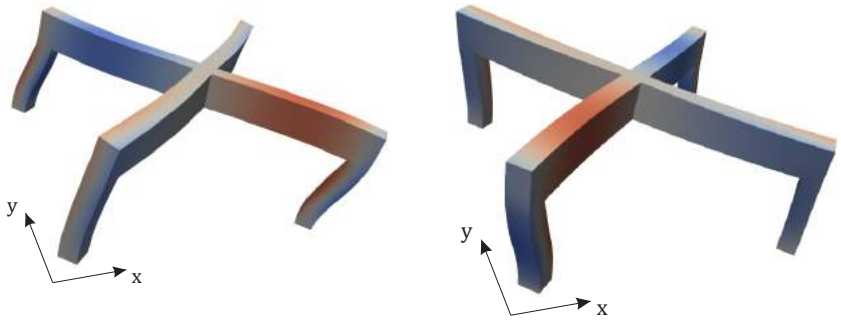
Tabela 7 – Frequências naturais do modelo numérico do pórtico.

$\mathbf{f_n}$	Modal_TETRA (**Hz**)	ANSYS (**Hz**)
1 (Translação na diagonal 2 – Fig. 47)	5,704	5,627
2 (Translação na diagonal 1 – Fig. 47)	5,705	5,628
3 (Torsional)	6,879	6,769

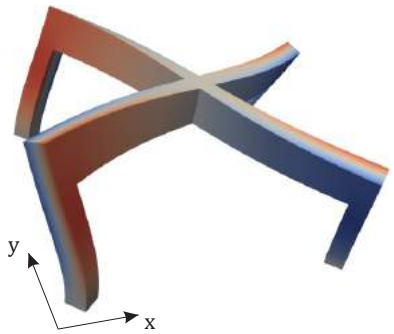
Fonte: O Autor (2018).

Figura 49 – Modos de vibração do modelo numérico.

(a) 1^{z} modo ($5{,}704Hz$) (b) 2^{z} modo ($5{,}705Hz$)



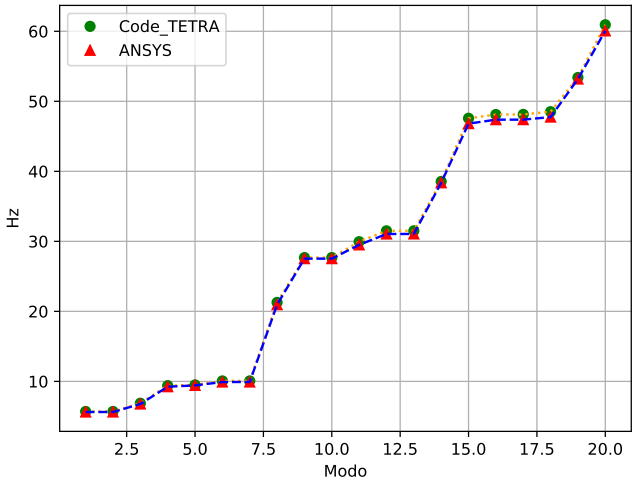
(c) 3^{z} modo ($6{,}879Hz$)



Fonte: O Autor (2018).

Na Fig. 50 segue um gráfico com os 20 primeiros modos naturais de ambos os softwares com o intuito de comparação dos resultados.

Figura 50 – Os 20 primeiros modos de vibração: comparação de resultados entre o Modal_TETRA e ANSYS.



Fonte: O Autor (2018).

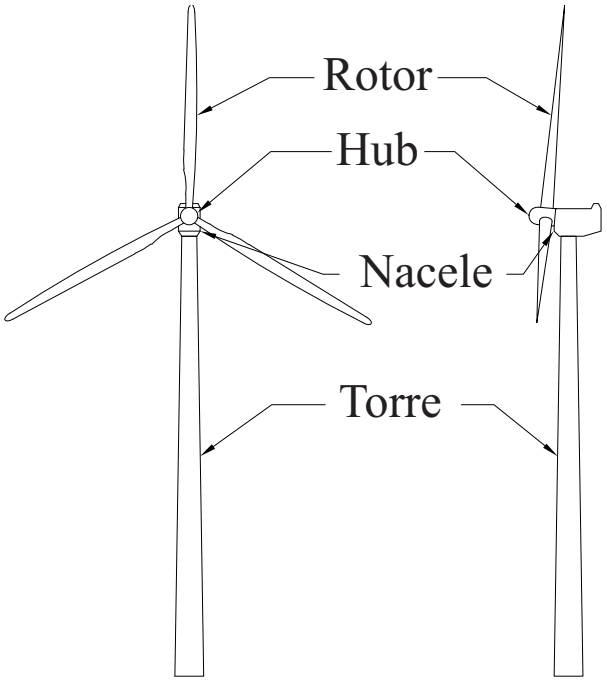
5 TURBINA EÓLICA NREL 5 MW

Após as revisões bibliográficas, fundamentos teóricos e métodos para a implementação do código de análise modal envolvendo elementos finitos de casca e sólido, buscou-se neste capítulo apresentar um modelo de turbina eólica de referência para o desenvolvimento na prática.

As especificações base da turbina de referência para o modelo numérica se encontram no relatório de Jonkman et al. (2009), baseadas no aerogerador de linha base NREL de classe 5 MW. NREL é o Laboratório Nacional de Energia Renovável do Departamento de Energia dos EUA (DOE).

O modelo de referência se enquadra nas turbinas eólicas de eixo horizontal de três pás (HAWTs), composto pelo Rotor, Nacele e Torre. A Torre é tipicamente de aço com paredes finas, enquanto as Pás do Rotor são geralmente feitas de algum material de fibra de carbono. A Nacele contém a caixa de câmbio, gerador, eixos e outros componentes (ANDRE, 2013).

Figura 51 – Componentes de uma moderna turbina eólica.



Fonte: Adaptado de Prowell (2011)

A turbina NREL de 5 MW foi desenvolvida apenas para estudos, sendo apenas um conceito. Abaixo, segue a Tab. 8 com as propriedades geométricas e materiais da versão terrestre especificada por Jonkman et al. (2009).

Tabela 8 – Propriedades da turbina eólica de 5 MW.

Propriedades	**Valores**
Altura da Torre	87,6 m
Diâmetro da seção inferior da Torre	6 m
Diâmetro da seção superior da Torre	3,87 m
Espessura da parede da Torre	$27 - 19$ mm
Diâmetro do Rotor	126 m
Diâmetro do Hub	3 m
Comprimento longitudinal das Pás	61,5 m
Localização do CM das Pás	20,475 m
Massa da Torre	347460 kg
Massa da Nacele	240000 kg
Coordenada do CM Nacele	$(-1{,}9; 0; 89{,}35)$ m
I_{Nacele} sobre o eixo z	2607890 $kg \cdot m^2$
Massa do Hub	56780 kg
Coordenada do CM do Hub	$(5{,}01910; 0; 90)$ m
I_{Hub} sobre o eixo da Nacele paralelo ao eixo x	115926 $kg \cdot m^2$
Massa da Pá	17740 kg
Massa do Rotor	350000 kg
Massa total da Turbina	697460 kg

Fonte: Jonkman et al. (2009).

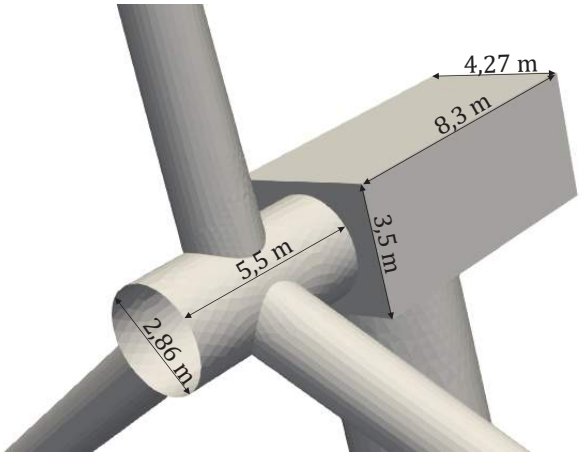
A Torre e as Pás foram modeladas como cascas cilíndricas de espessura uniforme, já a Nacele, como um hexaedro sólido de massa e momentos de inércias próximos aos especificados na Tab. 8. O Hub foi discretizado por elementos de cascas formando um cilindro, também com massa e momentos de inércias próximos aos especificados na Tab. 8.

As figuras 52 e 53, a seguir, mostram o modelo numérico da turbina eólica de 5 MW desenvolvida com base nas informações da Tab. 8.

Figura 52 – Modelo numérico do NREL 5 MW: vista frontal e lateral.

Fonte: O Autor (2018).

Figura 53 – Modelo numérico do NREL 5 MW: dimensões da Nacele e Hub.



Fonte: O Autor (2018).

Para compatibilização do modelo proposto com os dados da Tab. 8, as densidades das massas dos componentes tiveram que ser um pouco modificadas, conforme a Tab. 9.

Tabela 9 – Dados dos componentes do modelo NREL 5 MW usado no código desenvolvido.

Comp.	Dens. de massa	Mód. de Young	Coef. Poisson	Espessura
Torre	11128,513 $kg \cdot m^{-3}$	210 GPa	0,3	0,023 m
Nacele	1934,8 $kg \cdot m^{-3}$	210 GPa	0,3	–
Hub	49956 $kg \cdot m^{-3}$	210 GPa	0,3	0,023 m
Pá	3510,7857 $kg \cdot m^{-3}$	10 GPa	0,3	0,023 m

Fonte: O Autor (2018).

5.1 Frequências e Modos Naturais de Vibração do Modelo Numérico

Para análise modal do modelo numérico, foi gerada uma malha do tipo conforme no Salome com:

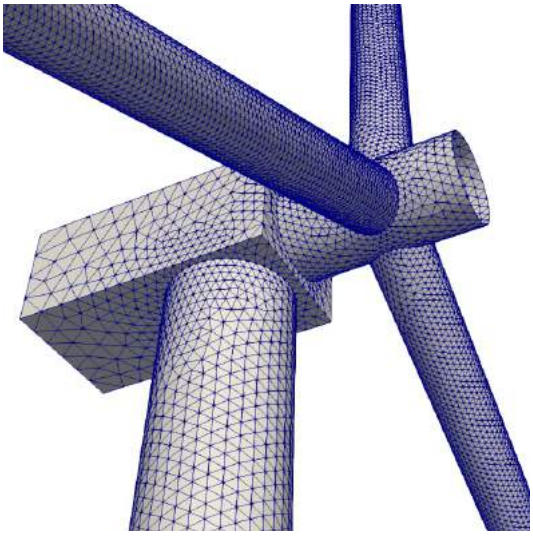
- 120618 elementos de cascas triangulares de 3 nós,
- 17959 elementos tetraedros de 4 nós,
- e um total de 63572 pontos nodais em todo domínio da malha.

Paralelamente, o mesmo modelo foi analisado no software comercial ANSYS, onde foi criada uma malha do tipo não conforme com:

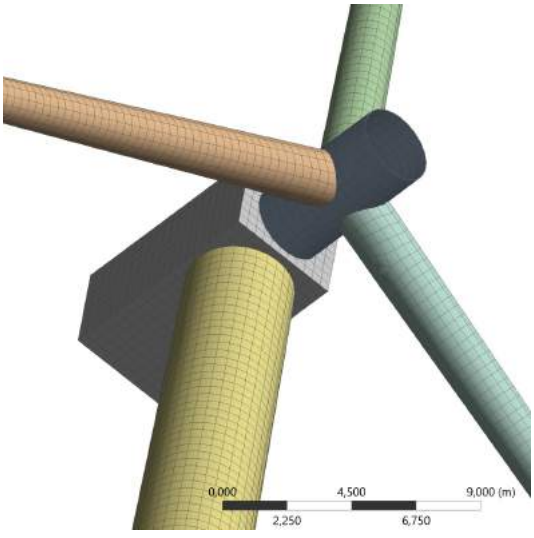
- 19033 elementos SHELL181 para as superfícies da Torre, Hub e Pás,
- 2079 elementos SOLID186 para a Nacele,
- 652 elementos de contato entre as Pás, Hub, Torre e Nacele,
- e 29073 pontos nodais em todo o domínio da malha.

Figura 54 – Malhas do modelo NREL 5MW

(a) Malha conforme gerada no Salome



(b) Malha não conforme gerada no ANSYS



Fonte: O Autor (2018).

A seguir, Tab. 10, serão apresentados os resultados das frequências naturais de translação obtidos pelo Modal_EOL e software ANSYS, com os seus respectivos modos de vibração através dos arquivos .VTK gerados pelo plugin.

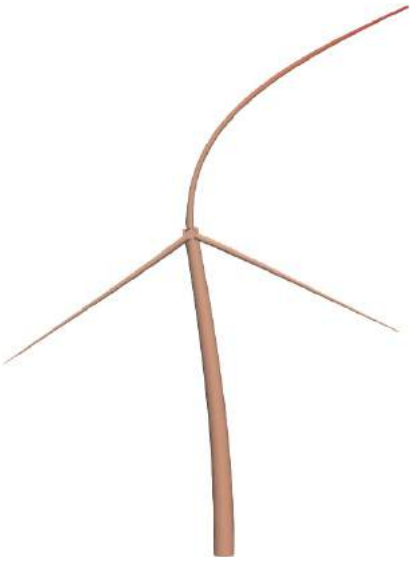
Tabela 10 – Frequências naturais de translação do modelo numérico da turbina NREL 5 MW.

$\mathbf{f_n}$	Modal_EOL (**Hz**)	ANSYS (**Hz**)
7 (Translação em y)	0,2872	0,2879
8 (Translação em x)	0,2943	0,2944
21 (Translação em x)	2,3744	2,3721
24 (Translação em y)	2,4698	2,4658

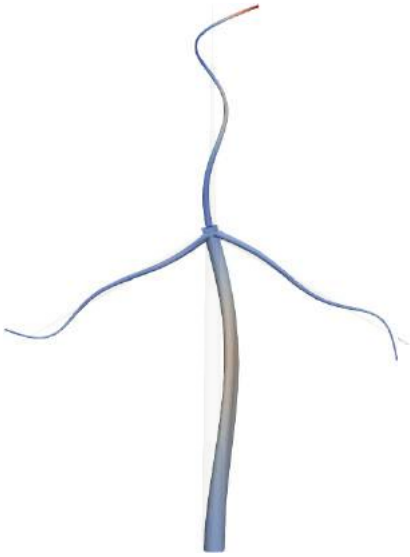
Fonte: O Autor (2018).

Figura 55 – Modos de vibração do modelo numérico (Modal_EOL).

(a) $7^{ž}$ modo ($0{,}287Hz$)



(b) $24^{ž}$ modo ($2{,}47Hz$)



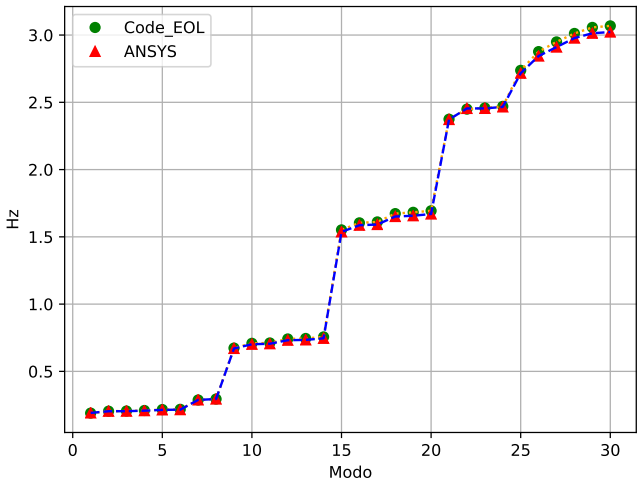
(c) $8^{ž}$ modo ($0{,}294Hz$)

(d) $21^{ž}$ modo ($2{,}374Hz$)

Fonte: O Autor (2018).

Na Fig. 56 segue um gráfico com os 30 primeiros modos naturais de ambos os softwares com o intuito de comparação dos resultados.

Figura 56 – Os 30 primeiros modos de vibração: comparação de resultados entre o Modal_EOL e ANSYS.



Fonte: O Autor (2018).

O plugin Modal_EOL conseguiu simular o caso envolvendo modelo com diferentes elementos acoplados com bastante precisão, obtendo erros menores que 5 %, como se pode observar no gráfico acima. Observa-se, também, uma concordância de resultados com os obtidos pelo ANSYS, validando o código fonte.

6 CONSIDERAÇÕES FINAIS

O principal objetivo desta dissertação foi o desenvolvimento de um ambiente computacional específico incluindo códigos Python integrados ao software SALOME, com a finalidade de análise modais pelo Método dos Elementos Finitos. Ainda existem uma gama de funcionalidades e ajustes operacionais que podem ser adicionados no futuro. Este trabalho destaca-se como um projeto inicial sobre quais evoluções promissoras possam agregar implementações de algoritmos com praticidade na comunidade acadêmica.

Os códigos do Code_Mamne foram projetados para aproveitar os recursos do Python, seguindo estruturas orientadas a objetos. A modelagem desde a definição da forma geométrica e geração de malhas se dão pela plataforma Salome, permitindo ao usuário liberdade de usar as ferramentas disponíveis no software. Para testar a precisão dos recursos dos plugins, uma série de exemplos de testes foram aplicados e os resultados foram comparados com os obtidos com o software comercial ANSYS.

Os resultados das aplicações básicas obtidos com os elementos DKT, os de casca triangular de 3 nós e tetraedros de 4 nós, encontraram uma concordância perfeita com o ANSYS. A análise modal da turbina eólica NREL 5 MW com elementos finitos de cascas e tetraedros acoplados mostrou que a técnica de acoplamento é válida para modelos especiais e complexos, erros inferiores a 5%.

Em conclusão, todos os elementos desenvolvidos neste trabalho se mostraram adequados, em análises estáticas e modais, podendo-se concluir que os procedimentos envolvidos na montagem e na solução das equações funcionam para todos os tipos de elementos.

São propostos os seguintes tópicos para contribuição em trabalhos futuros:

- Utilização de outros tipos de elementos finitos (de alta ordem);
- emprego de outros métodos numéricos;
- Realização de análises não-lineares;
- Implementação em Cython. Uma linguagem de programação que combina Python com C e C++. Smith (2015) afirma que o Cython é um compilador que traduz o código-fonte do Cython em código-fonte C ou C++ de forma eficiente.

REFERÊNCIAS

ANDRE, R. *Seismic Response of Wind Turbines*. Dissertação (Mestrado) — Institutt for konstruksjonsteknikk, 2013.

BARROS, W. M. *Matrizes explícitas em elementos finitos de alta ordem aplicadas a problemas de elasticidade 2D e 3D.* 2016. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2016.

BATOZ, J.-L. An explicit formulation for an efficient triangular plate-bending element. *International Journal for Numerical Methods in Engineering*, v. 18, n. 7, p. 1077–1089, 1982.

BATOZ, J.-L.; BATHE, K.-J.; HO, L.-W. A study of three-node triangular plate bending elements. *International Journal for Numerical Methods in Engineering*, v. 15, p. 1771–1812, dec 1980.

COLLACOTT, R. A. *Vibration monitoring and diagnosis: Techniques for cost-effective plant maintenance*. New York: Wiley, 1979.

COOK, R.; MALKUS, D.; PLESHA, M.; WITT, R. *Concepts and applications of finite element analysis*. 4. ed. New York: Wiley, 2002.

CUVELIER, F.; JAPHET, C.; SCARELLA, G. An efficient way to perform the assembly of finite element matrices in matlab and octave. *CoRR*, abs/1305.3122, 2013.

DHATT, G. Numerical analysis of thin shells by curved triangular elements based on discrete kirchhoff hypothesis. **in** PROCEEDINGS OF THE ASCE SYMPOSIUM ON APPLICATION OF FEN IN CIVIL ENGINEERING, p. 255–278, 1969.

FRAGA, P. T. *EFFECT – Efficient finite element code.* 2015. Dissertação (Mestrado) — Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa, 2015.

JONKMAN, J.; BUTTERFIELD, S.; MUSIAL, W.; SCOTT, G. Definition of a 5-mw reference wind turbine for offshore system development. *National Renewable Energy Lab. (NREL)*, 2009.

KIKUCHI, F. On a finite element scheme based on the discrete kirchhoff assumption. *Numerische Mathematik*, v. 24, n. 3, p. 211–231, Jun 1975.

KIRCHHOFF, G. Über das gleichgewicht und die bewegung einer elastischen scheibe. *Journal für die reine und angewandte Mathematik*, v. 40, p. 51–88, 1850.

KWON, Y. W.; BANG, K. *The Finite Element Method using Matlab*. 1. ed. Flórida: CRC Press, 1996.

LEHOUCQ, R. B.; SORENSEN, D. C.; YANG, C. *ARPACK Users Guide: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods.* 1997.

LOGAN, D. L. *A First Course in the Finite Element Method*. 4. ed. Platteville: University of Wisconsin, 2006.

MESQUITA, A. D. *Uma formulação do método dos elementos finitos aplicada à análise elastoplástica de cascas.* 1998. Dissertação (Mestrado) — Universidade de São Paulo, 1998.

MINDLIN, R. D. Influence of rotary inertia and shear on flexural motions of isotropic, elastic plates. *J. of Appl.Mech.*, v. 18, p. 31–38, 1951.

OñATE, E. *Structural Analysis with the Finite Element Method. Linear Statics: Volume 1: Basis and Solids*. 1. ed. Barcelona: Springer Science & Business Media, 2009.

PROWELL, I. *An Experimental and Numerical Study of Wind Turbine Seismic Behavior*. [S.l.]: University of California, San Diego, 2011. ISBN 9781124576923.

RIBES, A.; CAREMOLI, C. Salome platform component model for numerical simulation. In: *31st Annual International Computer Software and Applications Conference (COMPSAC 2007)*. [S.l.: s.n.], 2007. v. 2, p. 553–564. ISSN 0730-3157.

SALOME. *The Open Source Integration Platform for Numerical Simulation*. versão 8.3. [S.l.]: http://www.salome-platform.org., 2017.

SEQUEIRA, C. D. *A análise de vibrações como ferramenta para a melhoria da manutenção em aerogeradores*. Dissertação (Mestrado) — Universidade Nova de Lisboa – Faculdade de Ciências e Tecnologia, 2012.

SMITH, K. W. *Cython - A guide for Python programmers*. [S.l.]: O'Reilly, 2015. ISBN 9781491901557.

SORIANO, H. *Introduçao A Dinamica Das Estruturas*. [S.l.]: ELSEVIER EDITORA, 2014. ISBN 9788535251531.

SYDENSTRICKER, R. M.; COUTINHO, A. L. G. A.; LANDAU, L.; MARQUES, O. A. Pseudoconsistent load vector and mass matrix for the discrete kirchhoff triangle and the discrete shear triangle elements. *Communications in Numerical Methods in Engineering*, v. 11, n. 4, p. 317–330, 1995.

SZILARD, R. Theories and applications of plate analysis: Classical, numerical and engineering methods. v. 57, 01 2004.

TIMOSHENKO, S. P.; GERE, J. E. *Mecânica dos Sólidos*. 1. ed. Rio de Janeiro: Livos Técnicos e Científicos, 1982.

APÊNDICE A – CODE_MAMNE

URL – Repositório GIT

https://github.com/elielreis/Projeto_Mamne.git

MODAL_DKT.py

```
from numpy import (shape, zeros, sqrt, pi)
from scipy.sparse.linalg import eigsh
from FUNCTIONS_DKT import Index_Global_DKT
from KG_DKT import KG_DKT
from MG_DKT import MG_DKT


class MODAL_DKT:

    """
        SOLVER = MODAL_DKT(coord, connec_face, nodesr,
                 E, nu, t, rho, gr)

        PARAMETERS: coord:  ndarray((N,3), dtype(float64))
                            Global Cartesian coordinates in
                            3d Euclidean space
                            (N = number of nodal points).
                    connec_face:    ndarray((N,3), dtype(int64))
                                Matrix of connectivity of the
                                elements with the global coordinates.
                                (N = number of elements).
                    nodesr: ndarray1d.dtype(int64)
                            The one-dimensional array with the
                            indices of the restricted nodes in
                            the global coordinate array.
                    E:  float64
                        Modulus of linear elasticity.
                    nu:  float64
                        Poisson Coefficient.
                    t:  float64
                        Thickness
                    rho:    float64
```

```
                        The effective density.
                gr:     [True, False, False] -> Simply supported.
                       [True, True, True] -> Fixed support.
                        Restricted degree of freedom

    RETURNS:    self.kg:    ndarray((N,N), float64)
                            Global stiffness matrix
                            (N = Number of degree of freedoms).
                self.kgr:   Restricted global stiffness matrix.
                self.mg:    ndarray((N,N), float64)
                self.mgr:   Restricted global matrix
                self.w:     Frequency in radians
                self.hz:    Frequency in Hz
  """

def __init__(self, coord, connec_face, nodesr,
  E, nu, t, rho, gr):
      self.g = 3
      self.coord = coord
      self.connec_face = connec_face
      self.nodesr = nodesr
      self.E = E
      self.nu = nu
      self.t = t
      self.rho = rho
      self.nelem, self.nnosel = shape(self.connec_face)
      self.nnos = len(self.coord)
      self.ngl = self.g * self.nnos
      self.gr = gr
      self.dirr = zeros((self.nnos, self.g), int)
      for i in range(len(self.gr)):
        if gr[i]:
          self.dirr[self.nodesr, i] = 1
      self.nglel = self.g * self.nnosel

      self.I, self.J = Index_Global_DKT(self.connec_face)
      self.kg = KG_DKT(
                      self.E,
                      self.nu,
                      self.t,
                      self.ngl,
                      self.coord,
```

```
                    self.connec_face,
                    self.I,
                    self.J
                    )
        self.mg = MG_DKT(
                    self.rho,
                    self.t,
                    self.ngl,
                    self.coord,
                    self.connec_face,
                    self.I,
                    self.J
                    )

        self.COMPUTE()

        def COMPUTE(self):
          self.r = (self.dirr.reshape(self.ngl) - 1)**2
          self.r = self.r.astype(bool)
          self.kgr = self.kg[self.r]
          self.kgr = self.kgr[:, self.r]
          self.mgr = self.mg[self.r]
          self.mgr = self.mgr[:, self.r]

          self.w2, self.modo = eigsh(
          self.kgr, k=6, M=self.mgr, sigma=0, which='LM')
          self.w = sqrt(self.w2)
          self.hz = (1. / (2. * pi)) * self.w
```

FUNCTIONS_DKT.py

```
def Index_Global_DKT(connec_face):
  from numpy import array

  g = 3
  nglel = 9

  no1 = g * connec_face[:, 0]
  no2 = g * connec_face[:, 1]
  no3 = g * connec_face[:, 2]

  m = array([no1, no1 + 1, no1 + 2,
```

```
            no2, no2 + 1, no2 + 2,
            no3, no3 + 1, no3 + 2])
  m = m.T

  j = array(list(range(nglel)) * nglel).reshape((nglel, nglel))
  i = j.T
  j = j.reshape(j.size)
  i = i.reshape(i.size)

  J = m[:, j]
  J = J.reshape(J.size)
  I = m[:, i]
  I = I.reshape(I.size)

  return I, J
```

KG_DKT.py

```
from numba import guvectorize
from numpy import shape

def KE_DKT(xg1, xg2, xg3, yg1, yg2, yg3, E, nu, t):

  from numpy import (array, sqrt, zeros)

  l12 = sqrt((xg2 - xg1)**2. + (yg2 - yg1)**2.)

  cosxX = (xg2 - xg1) / l12
  cosxY = (yg2 - yg1) / l12

  cosyX = - cosxY
  cosyY = cosxX

  lamb = array([[1., 0., 0.],
                [0., cosxX, cosyX],
                [0., cosxY, cosyY]])

  coord_global = array([[0., 0., 0.],
                        [xg1, xg2, xg3],
                        [yg1, yg2, yg3]])
  coord_local = lamb.T.dot(coord_global)
```

```
# Translating point 1 to the local coordinate 0 (0, 0, 0).
dx = coord_local[1, 0]
dy = coord_local[2, 0]
coord_local[1, :] -= dx
coord_local[2, :] -= dy
coord_local[1, 0] = 0.
coord_local[2, 0] = 0.
coord_local[2, 1] = 0.

x1, x2, x3 = coord_local[1]
y1, y2, y3 = coord_local[2]

x23 = x2 - x3
y23 = y2 - y3

x12 = x1 - x2
y12 = y1 - y2

x31 = x3 - x1
y31 = y3 - y1

A = abs(x31 * y12 - x12 * y31) * 0.5

l2_23 = x23**2. + y23**2.
l2_12 = x12**2. + y12**2.
l2_31 = x31**2. + y31**2.

p4 = - 6. * x23 / l2_23
p5 = - 6. * x31 / l2_31
p6 = - 6. * x12 / l2_12

q4 = 3. * x23 * y23 / l2_23
q5 = 3. * x31 * y31 / l2_31

r4 = 3. * (y23**2.) / l2_23
r5 = 3. * (y31**2.) / l2_31

t4 = - 6. * y23 / l2_23
t5 = - 6. * y31 / l2_31

alfa = array([[y3 * p6, 0., -4. * y3, - y3 * p6, 0.,
               -2. * y3, 0., 0., 0],
```

```
        [-y3 * p6, 0., 2. * y3, y3 * p6, 0.,
         4. * y3, 0., 0., 0.],
        [y3 * p5, -y3 * q5, y3 * (2. - r5), y3 * p4, y3 * q4,
        y3 * (r4 - 2.), -y3 * (p4 + p5), y3 * (q4 - q5),
        y3 * (r4 - r5)],
        [-x2 * t5, x23 + x2 * r5, -x2 * q5, 0., x3, 0., x2 * t5,
        x2 * (r5 - 1.), -x2 * q5],
        [0., x23, 0., x2 * t4, x3 + x2 * r4, -x2 * q4, -x2 * t4,
        x2 * (r4 - 1.), -x2 * q4],
        [x23 * t5, x23 * (1. - r5), x23 * q5, -x3 * t4,
        x3 * (1. - r4), x3 * q4, -x23 * t5 + x3 * t4,
        -x23 * r5 - x3 * r4 - x2, x3 * q4 + x23 * q5],
        [-x3 * p6 - x2 * p5, x2 * q5 + y3, -4. * x23 + x2 * r5,
        x3 * p6, -y3, 2. * x3, x2 * p5, x2 * q5, (r5 - 2.) * x2],
        [-x23 * p6, y3, 2. * x23, x23 * p6 + x2 * p4, -y3 + x2 * q4,
        -4. * x3 + x2 * r4, -x2 * p4, x2 * q4, (r4 - 2.) * x2],
        [x23 * p5 + y3 * t5, -x23 * q5 + (1. - r5) * y3,
        (2. - r5) * x23 + y3 * q5, -x3 * p4 + y3 * t4,
        (r4 - 1.) * y3 - x3 * q4, (2. - r4) * x3 - y3 * q4,
        -x23 * p5 + x3 * p4 - (t4 + t5) * y3,
        -x23 * q5 - x3 * q4 + (r4 - r5) * y3,
        -x23 * r5 - x3 * r4 + 4. * x2 + (q5 - q4) * y3]])

R = array([[2., 1., 1.], [1., 2., 1.], [1., 1., 2.]])
D = (E * t**3.) / (12. * (1. - nu**2.))

DR = D * R

D_ = zeros((9, 9), float)
D_[:3, :3] = (1. / 24.) * DR
D_[3:6, :3] = (1. / 24.) * nu * DR
D_[:3, 3:6] = (1. / 24.) * nu * DR
D_[3:6, 3:6] = (1. / 24.) * DR
D_[6:, 6:] = (1. / 24.) * (1. - nu) * 0.5 * DR

ke_local = (1. / (2. * A)) * alfa.T.dot(D_).dot(alfa)

T = zeros((9, 9), float)
T[:3, :3] = lamb.T
T[3:6, 3:6] = lamb.T
T[6:, 6:] = lamb.T
```

```
  ke_global = T.T.dot(ke_local).dot(T)

  return ke_global


def VKG_DKT(E, nu, t, coord, connec_face, arrayfalse, out):
  nglel = 9

  p, q = shape(coord)
  pp, qq = shape(connec_face)
  k = len(arrayfalse)

  nelem = pp

  tt = int(nglel**2)
  for i in range(nelem):
    x1, x2, x3 = coord[connec_face[i], 0]
    y1, y2, y3 = coord[connec_face[i], 1]

    ke = KE_DKT(x1, x2, x3, y1, y2, y3, E, nu, t)

    ve = ke.reshape(ke.size)

    start = i * tt
    end = (i + 1) * tt
    out[start:end] = ve


GU_VKG_DKT = guvectorize(
             ['float32, float32, float32, float32[:,:],\
               int32[:,:], int32[:], float32[:]',
              'float64, float64, float64, float64[:,:],\
               int64[:,:], int64[:], float64[:]'],
              '(),(),(),(p,q),(pp,qq),(k)->(k)')(VKG_DKT)


def KG_DKT(E, nu, t, ngl, coord, connec_face, I, J):
  from numpy import empty
  from scipy.sparse import csr_matrix

  arrayfalse = empty(I.size, int)
```

```
  vkg = GU_VKG_DKT(E, nu, t, coord, connec_face, arrayfalse)

  kg = csr_matrix((vkg, (I, J)), shape=(ngl, ngl))

  return kg
```

MG_DKT.py

```
from numba import guvectorize
from numpy import shape


def ME_DKT(xg1, xg2, xg3, yg1, yg2, yg3, rho, t):
  from numpy import eye

  nglel = 9

  xg12 = xg1 - xg2
  xg31 = xg3 - xg1
  yg12 = yg1 - yg2
  yg31 = yg3 - yg1

  A = abs(xg31 * yg12 - xg12 * yg31) * 0.5

  me_lumped = rho * (1. / 3.) * A * t * eye(nglel)
  me_lumped[1, 1] = 0.
  me_lumped[2, 2] = 0.
  me_lumped[4, 4] = 0.
  me_lumped[5, 5] = 0.
  me_lumped[7, 7] = 0.
  me_lumped[8, 8] = 0.

  return me_lumped


def VMG_DKT(rho, t, coord, connec_face, arrayfalse, out):

  nglel = 9

  p, q = shape(coord)
  pp, qq = shape(connec_face)
  k = len(arrayfalse)
```

```
  nelem = pp

  tt = int(nglel**2)
  for i in range(nelem):
    x1, x2, x3 = coord[connec_face[i], 0]
    y1, y2, y3 = coord[connec_face[i], 1]

    me = ME_DKT(x1, x2, x3, y1, y2, y3, rho, t)

    ve = me.reshape(me.size)

    start = i * tt
    end = (i + 1) * tt
    out[start:end] = ve


GU_VMG_DKT = guvectorize(
             ['float32, float32, float32[:,:],\
              int32[:,:], int32[:], float32[:]',
             'float64, float64, float64[:,:],\
              int64[:,:], int64[:], float64[:]'],
             '(),(),(p,q),(pp,qq),(k)->(k)')(VMG_DKT)


def MG_DKT(rho, t, ngl, coord, connec_face, I, J):
  from numpy import empty
  from scipy.sparse import csr_matrix

  arrayfalse = empty(I.size, int)

  vmg = GU_VMG_DKT(rho, t, coord, connec_face, arrayfalse)
  mg = csr_matrix((vmg, (I, J)), shape=(ngl, ngl))

  return mg
```

GUI_DKT.py

```
def GUI_DKT(context):

  '''
  This file contains the function of generating the
```

```
MAMNE/MODAL_DKT plugin with the graphical user
interface (GUI).

Initially it is necessary to import all the libraries
required for the MAMNE/MODAL_DKT plugin to work.
'''

import salome
from salome.smesh import smeshBuilder
import SMESH
from PyQt5.QtWidgets import QDialog
from PyQt5.QtWidgets import (
  QPushButton,
  QLabel,
  QCheckBox,
  QHBoxLayout,
  QVBoxLayout,
  QLineEdit)
from numpy import array, zeros
from MODAL_DKT import MODAL_DKT
import pandas as pd

study = context.study
studyId = context.studyId
sg = context.sg
smesh = smeshBuilder.New(salome.myStudy)

'''
The plugin takes place in dialog windows (where the
user informs the analysis model and the parameters),
the whole interface was elaborated using QDialog,
the base class of PyQt5 dialog windows. The modal
processing and post-processing routines are
implemented by object-oriented structures through
events defined by button clicks that are properly
positioned in the window layouts of the developed
interfaces.
'''

class APP_MODAL_DKT(QDialog):
  def __init__(self, parent=None):
    super(APP_MODAL_DKT, self).__init__()
```

```
        '''
        Generates and makes visible to the user
        the final interface.
        '''
        self.init_ui()
        self.show()

        '''
        Events defined by clicks of the respective
        buttons positioned on the final interface.
        '''
        self.mesh_b.clicked.connect(self.mesh_b_click)
        self.nodesCc_b.clicked.connect(self.nodesCc_b_click)
        self.compute.clicked.connect(
          lambda: self.compute_click(
            self.z.isChecked(),
            self.rx.isChecked(),
            self.ry.isChecked(),
            self.E_l.text(),
            self.nu_l.text(),
            self.rho_l.text(),
            self.t_l.text()))

    def init_ui(self):

        '''
        The layout of the dialog box is defined by the init_ui()
        function. In the development of a graphical interface it
        is necessary to define the set of graphic elements that
        will make up the final interface. Each element must be
        positioned in the window, and which events and routines
        should be associated.

        The first set of graphical elements are related to the
        selection of the mesh of the analysis model.
        '''
        self.mesh_b = QPushButton('MESH')
        self.mesh_l = QLineEdit()

        h_boxMesh = QHBoxLayout()
        h_boxMesh.addWidget(self.mesh_b)
```

```
h_boxMesh.addWidget(self.mesh_l)

self.div = QLabel(
  '-----------------------------------------'+
  '-----------------------------------------'+
  '-----------------')

'''
The second set of graphical elements are related to
the application of the contour conditions of
the analysis model.
'''
self.nodesCc_lb = QLabel('BOUNDARY CONDITIONS')
self.nodesCc_l = QLineEdit()
self.nodesCc_b = QPushButton('GROUP OF NODES')
self.z = QCheckBox('Uz')
self.rx = QCheckBox('Rx')
self.ry = QCheckBox('Ry')

h_boxNodesCc = QHBoxLayout()
h_boxNodesCc.addWidget(self.nodesCc_lb)
h_boxNodesCc.addWidget(self.nodesCc_l)
h_boxNodesCc.addWidget(self.nodesCc_b)

h_boxCheck = QHBoxLayout()
h_boxCheck.addWidget(self.z)
h_boxCheck.addWidget(self.rx)
h_boxCheck.addWidget(self.ry)

v_boxNodesCc = QVBoxLayout()
v_boxNodesCc.addLayout(h_boxNodesCc)
v_boxNodesCc.addLayout(h_boxCheck)

self.div2 = QLabel(
  '-----------------------------------------'+
  '-----------------------------------------'+
  '-----------------')

'''
The third set of graphical elements are related
to the applications of the parameters of
the materials and geometric properties.
```

```
'''
self.E_lb = QLabel('MODULUS OF ELASTICITY')
self.E_l = QLineEdit()
self.nu_lb = QLabel('COEF. POISSON')
self.nu_l = QLineEdit()
self.rho_lb = QLabel('DENSITY OF MASSES')
self.rho_l = QLineEdit()
self.t_lb = QLabel('THICKNESS')
self.t_l = QLineEdit()

h_box_E = QHBoxLayout()
h_box_E.addWidget(self.E_lb)
h_box_E.addWidget(self.E_l)

h_box_nu = QHBoxLayout()
h_box_nu.addWidget(self.nu_lb)
h_box_nu.addWidget(self.nu_l)

h_box_rho = QHBoxLayout()
h_box_rho.addWidget(self.rho_lb)
h_box_rho.addWidget(self.rho_l)

h_box_t = QHBoxLayout()
h_box_t.addWidget(self.t_lb)
h_box_t.addWidget(self.t_l)

self.div3 = QLabel(
  '-----------------------------------------------'+
  '-----------------------------------------------'+
  '----------------')

'''
The fourth set of graphics is related to
the modal analysis processing.
'''
self.compute = QPushButton('COMPUTE')

'''
Final interface.
'''
v_box = QVBoxLayout()
v_box.addLayout(h_boxMesh)
```

```
    v_box.addWidget(self.div)
    v_box.addLayout(v_boxNodesCc)
    v_box.addWidget(self.div2)
    v_box.addLayout(h_box_E)
    v_box.addLayout(h_box_nu)
    v_box.addLayout(h_box_rho)
    v_box.addLayout(h_box_t)
    v_box.addWidget(self.div3)
    v_box.addWidget(self.compute)

    self.setLayout(v_box)
    self.setWindowTitle('CODE_MAMNE/MODAL_DKT')

  '''
  Routines to be defined by clicks on buttons specified
  initially.
  '''
  def mesh_b_click(self):
    self.mesh = None
    objId = salome.sg.getSelected(0)
    if objId:
      mm = study.FindObjectID(objId).GetObject()
      mesh = None
      try:
        mm.Load()
        mesh = mm
      except BaseException:
        mesh = None
        self.mesh_l.setText('Select a valid mesh')
        pass
      if mesh:
        name = smeshBuilder.GetName(mm)
        self.mesh_l.setText(name)
        self.mesh = mm
        nodes_id = list(self.mesh.GetElementsByType(SMESH.NODE))
        self.coord = array([self.mesh.GetNodeXYZ(i)
                  for i in nodes_id], float)
        face_id = list(self.mesh.GetElementsByType(SMESH.FACE))
        self.connec_face = array(
          [self.mesh.GetElemNodes(i) for i in face_id], int) - 1

  def nodesCc_b_click(self):
```

```
        self.cc = None
        objId = salome.sg.getSelected(0)
        if objId:
          cc = study.FindObjectID(objId).GetObject()
          Cc = None
          try:
            cc.GetNodeIDs()
            Cc = cc
          except BaseException:
            Cc = None
            self.nodesCc_l.setText('Select a valid group')
            pass
          if Cc:
            name = cc.GetName()
            self.nodesCc_l.setText(name)
            self.nodesr = array(cc.GetNodeIDs(), int) - 1

    def compute_click(self, Uz, Rx, Ry, E, nu, rho, t):

        self.gr = [Uz, Rx, Ry]
        self.E = float(E)
        self.nu = float(nu)
        self.rho = float(rho)
        self.t = float(t)

        self.MODOS = MODAL_DKT(self.coord, self.connec_face,
                               self.nodesr, self.E, self.nu, self.t,
                               self.rho, self.gr)

        raw_data = {'NATURAL FREQUENCIES': self.MODOS.hz}
        df = pd.DataFrame(raw_data, columns= ['NATURAL FREQUENCIES'])
        df.to_csv('MODAL_DKT.csv')

'''
Run the plugin.
'''
app = APP_MODAL_DKT()
app.exec_()
```