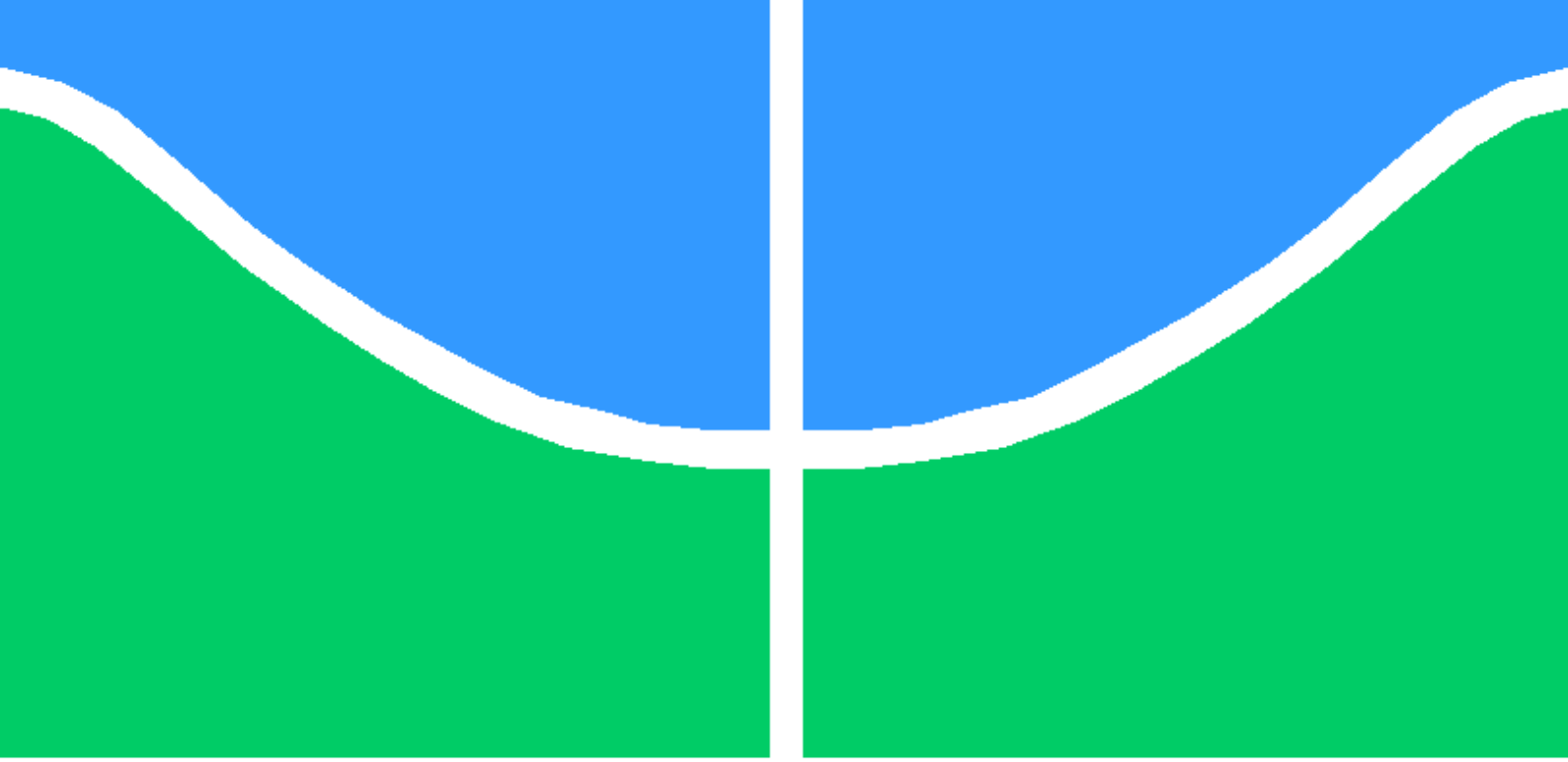

ANÁLISE DE PROBLEMAS TRIDIMENSIONAIS USANDO O MÉTODO DOS ELEMENTOS DE CONTORNO COM EXPANSÃO EM MULTIPOLOS

ARTHUR FROESE BUZOGANY

PROJETO DE GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

ANÁLISE DE PROBLEMAS TRIDIMENSIONAIS USANDO O MÉTODO DOS ELEMENTOS DE CONTORNO COM EXPANSÃO EM MULTIPOLOS

ARTHUR FROESE BUZOGANY

ORIENTADOR: PROF. DR. ÉDER LIMA DE ALBUQUERQUE, ENM/UNB

PROJETO DE GRADUAÇÃO EM ENGENHARIA MECÂNICA

PUBLICAÇÃO -
BRASÍLIA-DF, 30 DE NOVEMBRO DE 2017.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

ANÁLISE DE PROBLEMAS TRIDIMENSIONAIS USANDO O MÉTODO DOS ELEMENTOS DE CONTORNO COM EXPANSÃO EM MULTIPOLOS

ARTHUR FROESE BUZOGANY

PROJETO DE GRADUAÇÃO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO MECÂNICO EM ENGENHARIA MECÂNICA.

APROVADA POR:

Prof. Dr. Éder Lima de Albuquerque, ENM/UnB
Orientador

Prof. Dr. Marcus Vinicius Girão de Morais, ENM/UnB
Examinador interno

MSc. Álvaro Campos Ferreira, ENM/UnB
Examinador interno

BRASÍLIA, 30 DE NOVEMBRO DE 2017.

FICHA CATALOGRÁFICA
ARTHUR FROESE BUZOGANY
Análise de problemas tridimensionais usando o método dos elementos de contorno com expansão em multipolos
2017xv, 70p., 201x297 mm
(ENM/FT/UnB, Engenheiro Mecânico, Engenharia Mecânica, 2017)
Projeto de Graduação - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Mecânica

REFERÊNCIA BIBLIOGRÁFICA

ARTHUR FROESE BUZOGANY (2017) Análise de problemas tridimensionais usando o método dos elementos de contorno com expansão em multipolos. Projeto de Graduação em Engenharia Mecânica, Publicação , Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 70p.

CESSÃO DE DIREITOS

AUTOR: ARTHUR FROESE BUZOGANY
TÍTULO: Análise de problemas tridimensionais usando o método dos elementos de contorno com expansão em multipolos.
GRAU: Engenheiro Mecânico ANO: 2017

_____

ARTHUR FROESE BUZOGANY
Correio Eletrônico: arthurbuzogany@gmail.com

Agradecimentos

Agradeço primeiramente a Deus, pois todas as coisas são para Ele.

Agradeço a minha família, pois sempre me apoiaram e sem ela nada disso seria possível, em especial minha mãe, Mônica, meu pai, Carlos, e minha irmã, Raquel.

Ao meu orientador Éder, que sempre com a maior disposição e entusiasmo repassou seu conhecimento.

A minha namorada Juliana, me dando forças para continuar.

E por fim a todos meus amigos que tornaram esta uma jornada tão fantástica.

Arthur Froese Buzogany

ABSTRACT

This work presents a complete approach for tridimensional elastostatics analysis problems in solids using the boundary element method with multipole expansion. The analysis begins with the drawing of the solid in computer aided design programs (CAD programs). This drawing is then exported in the BREP (*Boundary Representation*) format which is imported in a mesh generator program, exported in the MSH (*Mesh*) format and then read in the Python developed program. The boundary conditions are then imposed, the unknown variables in fast multipole method are computed using the generalized minimum residual method (GMRES), provided that influence matrices are never fully assembled. Finally, the displacement field in the surface of the solid is shown on a heat map. Apart from the fast multipole external solver by Liu, all other programs used in this project are open source without any restriction of use, distribution or modification. The aim is to estimulate the use of this kind of programs in engineering projects, lowering their cost.

Keywords: Boundary Element Method with multipole expansion, Mesh generation, tridimensional elastostatics analysis

RESUMO

Este trabalho apresenta uma abordagem completa para a análise de problemas elásticos tridimensionais usando o método dos elementos de contorno com expansão em multipolos. A análise começa com o desenho do sólido em programas de desenho assistido por computador (programas CAD). Em seguida, estes desenhos são exportados em formato BREP (*Boundary Representation*, arquivos com extensão .brep) que então são importados em um programa gerador de malha, exportados em formato MSH (*Mesh*) e lidos no programa desenvolvido em Python. As condições de contorno do problema são então impostas, as variáveis desconhecidas no método dos elementos de contorno com expansão em multipolos são calculadas usando o método dos mínimos resíduos generalizados (GMRES) uma vez que as matrizes de influência nunca são montadas. Por fim, o deslocamento na superfície do sólido é mostrado em um mapa de calor. Com excessão do programa externo do método dos elementos de contorno com expansão em multipolos por Liu, todos os outros programas usados neste projeto são de código aberto, sem qualquer restrição de uso, distribuição ou modificação. Busca-se com isso estimular o uso deste tipo de programas em projetos de engenharia, reduzindo os custos dos mesmos.

Palavras-chave: Método dos elementos de contorno com expansão em multipolos, Geração de malhas, Análise elástica tridimensional

SUMÁRIO

LISTA DE FIGURAS

Capítulo 1

Introdução

A engenharia, em suas diferentes formas e apresentações, busca alcançar resultados satisfatoriamente precisos modelando problemas reais. A impossibilidade de desenvolver soluções analíticas, irrepreensíveis, para certos tipos de questões, devido à sua complexidade, demanda que sejam criadas simplificações nas equações que regem o problema, mas que, em contrapartida, também diminuem a precisão dos resultados encontrados.

A complexidade dos problemas demandou o desenvolvimento dos métodos numéricos que são capazes de obter soluções aproximadas da problemática real. O uso de métodos numéricos apresenta a vantagem de não impor muitas simplificações devido à crescente evolução dos recursos computacionais, alcançando assim análises mais realistas. Isso ocorre devido ao emprego de modelos discretos – ao contrário da análise analítica, que usa modelos contínuos – para resolver o problema.

Para a resolução de problemas de engenharia, destacam-se três métodos numéricos:

- Método das Diferenças Finitas (MDF);
- Método dos Elementos Finitos (MEF);
- Método dos Elementos de Contorno (MEC).

Nos últimos anos, entrou em ascensão o método dos elementos de contorno (MEC) estudado de forma mais detalhada neste trabalho. Este método tem como base equações integrais de contorno do problema. Neste método somente o contorno, ou seja, as superfícies dos problemas tridimensionais ou as curvas dos problemas bidimensionais têm que ser discretizadas, diminuindo assim a ordem do problema em uma dimensão. Quando se retira uma ordem de dimensão do problema, reduz-se também a quantidade de dados que precisam ser fornecidos para a solução do problema, afetando positivamente também o tempo necessário para o processamento e a memória necessária para o armazenamento das informações analisadas. O MEC apresenta maior precisão em relação aos outros métodos, especialmente em problemas de concentração de tensões como análise de fratura de estruturas e componentes.

A ascensão do uso do método de elementos de contorno se deu pela praticidade de geração e alteração da malha em questão, o que acelera bastante o processo de solução de problemas que apresentam malhas que variam muito ao longo da análise e têm que ser adaptadas para cada nova simulação, como no caso da propagação de trincas na Mecânica da Fratura ou nos problemas de escoamento turbulento com fronteiras móveis. A desvantagem do MEC está relacionado com a sua incapacidade de ser economicamente viável quando se trata de problemas de grande escala, por apresentar matrizes cheias e não simétricas em sua formulação. O seu alto custo computacional e amplo uso de memória são barreiras para a utilização nos problemas com mais de um milhão de graus de liberdade.

Tendo em vista essa inferioridade com relação a outros métodos, foi desenvolvido o método dos elementos de contorno rápido, que diminuiu o tempo de processamento e a quantidade de memória, especialmente em problemas de larga escala, (CAMPOS, 2016) e (LIU, 2009), tornando o MEC competitivo para a análise destes problemas. A ideia por trás dos métodos rápidos é, de alguma forma, não calcular todos os termos das matrizes do método dos elementos de contorno. Ao invés disso, faz-se uma divisão da matriz em blocos, sendo que vários destes blocos são aproximados por matrizes de baixo posto, que possuem menos elementos para se calcular.

1.1 Comparação entre MEF e MEC

Este trabalho terá foco no MEC, portanto, o MEF não será detalhado, mas é importante saber as diferenças entre eles. Estes dois métodos são os mais versáteis para análises de engenharia (GAUL; KöGL; WAGNER, 2003), sendo relevante então saber as vantagens e desvantagens de cada um.

Uma das maiores vantagens do MEC é a discretização apenas do contorno do problema. A Figura 1.1 ilustra um exemplo de como cada método aborda o problema. A complexidade do problema é determinante para a escolha do método que será utilizado, levando em conta a geometria, o tipo de carregamento e a criação da malha (GAUL; KöGL; WAGNER, 2003). Por outro lado, a maior desvantagem do MEC é que este produz matrizes cheias e não simétricas, enquanto o MEF produz matrizes esparsas e simétricas.

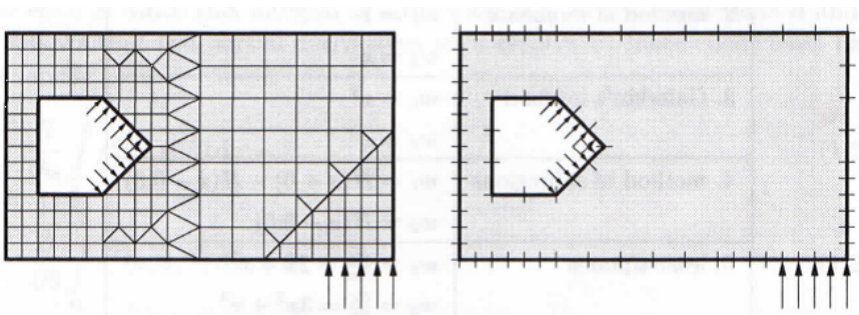

Figura 1.1: Discretização com MEF (esq.) e MEC (dir.)

1.2 Troca de arquivos de desenho entre programas

A compatibilidade entre os diversos programas utilizados na solução de um problema de engenharia torna-se, atualmente, possível através das diferentes possibilidades de arquivos que cada programa pode gerar e interpretar. No caso deste trabalho, foram criados arquivos em CAD analisados através de um programa em linguagem Python com a formulação do método dos elementos de contorno rápido,. Para que esta análise possa ser executada, foi necessário exportar o arquivo CAD no formato de Representação de Contorno ou *Boundary Representation* (BREP ou B-rep) com a intenção de poder importá-lo em um programa gerador de malha. Após a geração da malha, a mesma é exportada em um arquivo com a extensão .msh (*Mesh*), que finalmente é lido pelo programa de elementos de contorno.

1.3 Programas de código aberto

Um grande passo para os programas de código aberto se deu no ano de 1989 quando Richard Stallman publicou a primeira versão da licença *GNU General Public License* que era um agrupamento de diversas licenças menores restritas a *softwares* específicos, sendo seu objetivo unificar todas elas de forma que pudesse ser utilizada em qualquer projeto, possibilitando o compartilhamento de códigos. Atualmente, a terceira versão da licença já disseminou o conceito de liberdade; liberdade para distribuir cópias de *software* gratuito (e cobrar por elas se desejar), para ter acesso ao código fonte e a possibilidade de modificar o *software* ou utilizar pedaços deste em novos programas. É garantida a liberdade de compartilhamento, distribuição e modificação de qualquer versão de *software* sob essa licença.

Excluindo-se o programa do método dos elementos de contorno com expansão em multipolos, todos os outros *softwares* utilizados neste trabalho são *softwares* de código aberto, possibilitando acessibilidade a qualquer um que o desejar, sem custo, inclusive na implementação do conhecimento aqui gerado em outros projetos, pois estão sob a licença *GNU General Public License*. Para proteger os direitos dos usuários é necessário que todos os programas que façam uso de um programa com este tipo de licença também perpetuem o uso da mesma licença.

1.4 Objetivo e principais contribuições do trabalho

Este trabalho tem como objetivo apresentar a solução elasto-estática tridimensional pelo método dos elementos de contorno com expansão em multipolos, desde a construção de um modelo tridimensional em programas CAD, a exportação em formato de troca de arquivo de desenho, a geração da malha superficial triangular, a aplicação das condições de contorno, a análise do MEC e o pós-processamento.

A modelagem computacional da geometria pode ser feita em quaisquer *softwares* CAD. Entretanto, neste trabalho a opção foi pelo FreeCAD, que é um programa de código aberto. Estes modelos geométricos são exportados em arquivos em formato de Representação de Contorno (arquivos com extensão .brep). Os arquivos BREP produzidos pelos programas CAD são lidos no programa gerador de malha e, posteriormente, os arquivos GMSH Mesh (arquivos com extensão .msh) são analisados pelo programa desenvolvido neste trabalho, que faz uso de uma função externa.

A principal contribuição deste trabalho é a integração de diferentes ferramentas para alcançar uma análise completa de uma peça com o método dos elementos de contorno com expansão em multipolos. Foi utlizado um código externo para o cálculo dos deslocamentos e forças nodais pelo método dos elementos de contorno com expansão em multipolos desenvolvido por (LIU,2016). A rotina desenvolvida para a integração da malha, condições de contorno e o solucionador externo do método numérico foi desenvolvida em ambiente Python.

O programa foi aplicado a problemas elásticos com complexidade geométrica simples, modelados em CAD. O programa mostrou-se robusto e eficiente, podendo ser facilmente estendido para outras formulações.

Capítulo 2

Problemas Elásticos

Este capítulo faz uma breve descrição da formulação do método dos elementos de contorno aplicados a problemas elásticos tridimensionais. Primeiramente são listadas as principais equações da teoria da elasticidade linear sem deduzí-las. Então, a dedução da equação integral de contorno para problemas elásticos lineares é apresentada. Depois, estas equações são discretizadas utilizando elementos de contorno triangulares lineares contínuos, obtendo-se a formulação matricial do método dos elementos de contorno. Aplicando-se as condições de contorno, obtém-se o sistema linear, que então pode ser resolvido para calcular as variáveis desconhecidas. Por fim, são montados os vetores com deslocamentos e forças de superfície nos nós do contorno.

2.1 Formulação elástica linear

Considerando um elemento infinitesimal dentro de um domínio V,

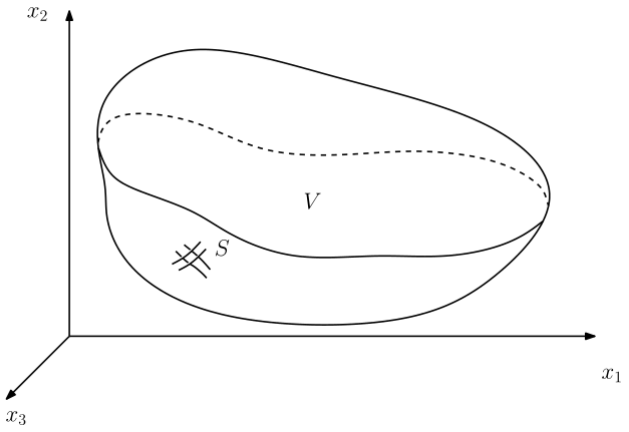

Figura 2.1: Domínio de um corpo.

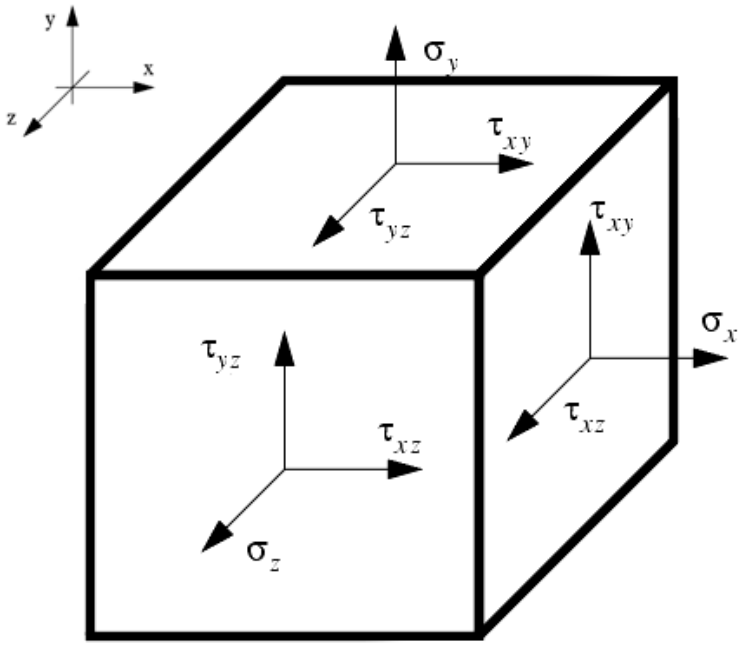

Figura 2.2: Forças em um elemento infinitesimal.

o equilíbrio de forças pode ser expresso por:

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

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

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

que, também pode ser escrito como:

$$\frac{\partial \sigma_{ij}}{\partial x_j} + b_i = 0 \tag{2.4}$$

ou ainda como

$$\sigma_{ij,j} + b_i = 0. \tag{2.5}$$

Por sua vez, o equilíbrio de momentos é expresso por

$$\sigma_{ij} = \sigma_{ji}, \tag{2.6}$$

em que σ_{ij} é o tensor de tensões e b_i é o vetor de forças de corpo.

O vetor de forças de superfície t_i em um ponto no contorno S de um domínio V é expresso na forma

$$t_i = \sigma_{ij} n_j, \tag{2.7}$$

em que n_j é o vetor normal do contorno S no ponto.

Em elasticidade linear, o vetor de deslocamentos e suas derivadas são assumidos como infinitesimais. O tensor de deformação, considerando deslocamentos infinitesimais, pode ser escrito como

$$\varepsilon_{kl} = \frac{1}{2}\left(u_{k,l} + u_{l,k}\right) \tag{2.8}$$

Para assegurar a unicidade dos deslocamentos, as componentes do tensor de deformações não podem ser designadas arbitrariamente, devendo satisfazer certas condições de compatibilidade e integrabilidade. A equação de compatibilidade é dada por:

$$\varepsilon_{ij,kl} + \varepsilon_{kl,ij} - \varepsilon_{ik,jl} - \varepsilon_{jl,ik} = 0 \tag{2.9}$$

No caso de material elástico linear, a relação entre o tensor de tensões com o tensor de deformações é escrita, na sua forma mais geral, como

$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ji} \tag{2.10}$$

em que

$$\mu = \frac{E}{(1+\nu)} \tag{2.11}$$

sendo μ o módulo de cisalhamento, E o módulo de elasticidade

$$\lambda = \frac{\nu E}{(1+\nu)}(1-2\nu) \tag{2.12}$$

e ν o coeficiente de Poisson.

2.2 Formulação integral

Assumindo-se uma função vetorial contínua $u_i^{\bullet}$, que representa o deslocamento de um estado elasto-estático definido sobre um domínio V, como sendo uma função peso residual

da equação de equilíbrio (2.5), tem-se:

$$\iiint_V \sigma_{ij,j} u_i^\bullet dV + \iiint_V b_i u_i^\bullet dV = 0 \tag{2.13}$$

Pela regra de derivação do produto de duas funções tem-se:

$$(\sigma_{ij} u_i^\bullet)_{,k} = \sigma_{ij,k} u_i^\bullet + \sigma_{ij} u_{i,k}^\bullet \tag{2.14}$$

Pode-se escrever $u_{i,j}^\bullet$ como a soma de um tensor simétrico e um anti-simétrico, da forma

$$u_{i,j}^\bullet = \frac{1}{2}(u_{i,j}^\bullet + u_{j,i}^\bullet) + \frac{1}{2}(u_{i,j}^\bullet - u_{j,i}^\bullet) = \varepsilon_{ij}^\bullet + V_{ij}^\bullet \tag{2.15}$$

sendo que $\varepsilon_{ij}^\bullet$ e $V_{ij}^\bullet$ representam os tensores deformação (simétrico) e rotação (anti-simétrico), respectivamente, do estado elástico $''\bullet''$.

Substituindo (2.15) em (2.14) tem-se

$$(\sigma_{ij} u_i^\bullet)_{,j} = \sigma_{ij,j} u_i^\bullet + \sigma_{ij} \varepsilon_{ij}^\bullet + \sigma_{ij} V_{ij}^\bullet \tag{2.16}$$

sendo σ_{ij} um tensor simétrico. O produto de um tensor simétrico por um anti-simétrico é nulo. Desta forma, a equação (2.16) torna-se

$$\sigma_{ij,j} u_i^\bullet = (\sigma_{ij} u_i^\bullet)_{,j} - \sigma_{ij} \varepsilon_{ij}^\bullet \tag{2.17}$$

Substituindo a equação (2.17) na equação (2.13) tem-se

$$-\iiint_V \sigma_{ij} \varepsilon_{ij}^\bullet dV + \iiint_V (\sigma_{ij} u_i^\bullet)_{,j} dV + \iiint_V b_i u_i^\bullet dV = 0 \tag{2.18}$$

Pelo teorema de Green tem-se:

$$\iiint_V (\sigma_{ij} u_i^\bullet)_{,j} dV = \iint_S (\sigma_{ij} u_i^\bullet) n_j dS = \iint_S t_i u_i^\bullet dS \tag{2.19}$$

em que

$$t_i = \sigma_{ij} n_j \tag{2.20}$$

Substituindo (2.19) em (2.18), tem-se

$$\iiint_V \sigma_{ij} \varepsilon_{ij}^\bullet dV = \iint_S t_i u_i^\bullet dS + \iiint_V b_i u_i^\bullet dV \tag{2.21}$$

Se partirmos da equação (2.5) como sendo a correspondente ao estado $u_i^\bullet$ e a função de

interpolação da equação (2.13) como sendo u_i, obtém-se, de forma análoga a anterior

$$\iiint_V \sigma_{ij}^{\bullet}\varepsilon_{ij}dV = \iint_S t_i^{\bullet}u_i dS + \iiint_V b_i^{\bullet}u_i dV \tag{2.22}$$

Pelo teorema recíproco dois estados de um mesmo material podem ser relacionados por $\sigma_{ij}^{\bullet}\varepsilon_{ij} = \sigma_{ij}\varepsilon_{ij}^{\bullet}$. Desta forma, igualando-se as equações (2.22) e (2.21), tem-se

$$\iint_S t_i u_i^{\bullet} dS + \iiint_V u_i^{\bullet} b_i dV = \iint_S t_i^{\bullet} u_i dS + \iiint_V u_i b_i^{\bullet} dV \tag{2.23}$$

A equação integral (2.23) relaciona dois estados quaisquer de tensões. Para que se possa tratar problemas de elasticidade em meio contínuo, será adotado que um destes estados é conhecido, e o outro se deseja determinar. No caso de elementos de contorno, o estado conhecido é o chamado estado fundamental, que corresponde à resposta de um corpo infinito a uma carga concentrada unitária em um ponto $\mathbf{x}'$. A representação matemática de uma carga concentrada unitária é dada pelo delta de Dirac que é definido como

$$\begin{cases} \delta(\mathbf{x}-\mathbf{x}') = \infty & \text{se } \mathbf{x} = \mathbf{x}' \\ \delta(\mathbf{x}-\mathbf{x}') = 0 & \text{se } \mathbf{x} \neq \mathbf{x}' \\ \int_{-\infty}^{\infty} \delta(\mathbf{x}-\mathbf{x}')dV = 1 \end{cases} \tag{2.24}$$

A razão da escolha do estado fundamental deve-se ao fato de a função delta de Dirac reduzir o número de integrais de domínio, pois esta possui a propriedade

$$\iiint_V f(\mathbf{x})\delta(\mathbf{x}-\mathbf{x}')dV = f(\mathbf{x}') \tag{2.25}$$

para um dado ponto $\mathbf{x}' \in V$.

Considerando o estado " $\bullet$ " como sendo o estado fundamental de um problema estático livre de forças de corpo ($b_i^{\bullet} = 0$), a equação (2.23) pode ser escrita como

$$\iint_S T_{ik}u_i dS + \iiint_V b_i U_{ik} dV = \iint_S t_i U_{ik} dS - \iiint_V \delta_{ik} u_i dV \tag{2.26}$$

em que U_{ik} e T_{ik} representam respectivamente deslocamentos e forças de superfície na direção k, num ponto $\mathbf{x}$, devido a uma força concentrada unitária aplicada de forma estática num ponto $\mathbf{x}'$ numa direção i. Por serem soluções do estado fundamental, U_{ik} e T_{ik} são chamadas soluções fundamentais de deslocamentos e forças de superfície, respectivamente.

Devido a propriedade (2.25), a equação (2.26) pode ser escrita como

$$u_k + \iint_S T_{ik}u_i dS = \iint_S U_{ik}t_i dS - \iiint_V b_i U_{ik} dV \tag{2.27}$$

Considerando que as forças de corpo b_i são nulas, pode-se escrever:

$$u_k + \iint_S T_{ik} u_i dS = \iint_S U_{ik} t_i dS \tag{2.28}$$

2.3 Soluções fundamentais

A equação 2.29 representa a solução fundamental elasto estática do deslocamento para um domínio homogêneo tridimensional e a equação 2.30 representa a solução fundamental das tensões e forças.

$$U_{ij}(x', x) = \frac{1}{16\pi\mu(1-\nu)r}[(3-4\nu)\delta_{ij} + r_{,i}r_{,j}] \tag{2.29}$$

$$T_{ij}(x', x) = \frac{-1}{8\pi(1-\nu)r^2}\left\{\frac{\partial r}{\partial n}[(1-2\nu)\delta_{ij} + 3r_{,i}r_{,j}] - (1-2\nu)(n_j r_{,i} - n_i r_{,j})\right\} \tag{2.30}$$

Nota-se que tanto a solução fundamental de deslocamentos quanto a de forças de superfície são singulares quando o ponto fonte tende ao ponto campo. No caso da solução fundamental de deslocamentos a singularidade é fraca $(1/r)$. Já no caso da solução fundamental de forças de superfície tem-se uma singularidade forte $(1/r^2)$. As formas como estas singularidades são tratadas foi descrita em (ALBUQUERQUE, 2017).

2.4 Formulação dos elementos de contorno discretizada

Para se obter a solução do problema elasto-estático, o contorno é dividido em elementos de contorno. Nesta etapa do trabalho, são utilizados apenas elementos triangulares lineares (3 nós por elementos) contínuos (elementos cujos nós da extremidades são compartilhados com os elementos vizinhos). Nesta formulação será mais conveniente trabalhar com vetores do que usar notação indicial.

As funções de interpolação no espaço utilizadas neste trabalho (funções de forma) são as funções de forma triangulares lineares. Estas funções de forma triangulares lineares só permitem o modelamento de elementos planos e são especialmente indicadas para problemas em que não se tem altos gradientes.

Os deslocamentos e as forças de superfícies são representados em um elemento triangular

linear padrão como:

$$\mathbf{u} = \left\{ \begin{array}{c} u_1 \\ u_2 \\ u_3 \end{array} \right\} = \left[\begin{array}{ccccccccc} N^{(1)} & 0 & 0 & N^{(2)} & 0 & 0 & N^{(3)} & 0 & 0 \\ 0 & N^{(1)} & 0 & 0 & N^{(2)} & 0 & 0 & N^{(3)} & 0 \\ 0 & 0 & N^{(1)} & 0 & 0 & N^{(2)} & 0 & 0 & N^{(3)} \end{array} \right] \left\{ \begin{array}{c} u_1^{(1)} \\ u_2^{(1)} \\ u_3^{(1)} \\ u_1^{(2)} \\ u_2^{(2)} \\ u_3^{(2)} \\ u_1^{(3)} \\ u_2^{(3)} \\ u_3^{(3)} \end{array} \right\} = \mathbf{N}\mathbf{u}^{(n)} \tag{2.31}$$

$$\mathbf{t} = \left\{ \begin{array}{c} t_1 \\ t_2 \\ t_3 \end{array} \right\} = \left[\begin{array}{ccccccccc} N^{(1)} & 0 & 0 & N^{(2)} & 0 & 0 & N^{(3)} & 0 & 0 \\ 0 & N^{(1)} & 0 & 0 & N^{(2)} & 0 & 0 & N^{(3)} & 0 \\ 0 & 0 & N^{(1)} & 0 & 0 & N^{(2)} & 0 & 0 & N^{(3)} \end{array} \right] \left\{ \begin{array}{c} t_1^{(1)} \\ t_2^{(1)} \\ t_3^{(1)} \\ t_1^{(2)} \\ t_2^{(2)} \\ t_3^{(2)} \\ t_1^{(3)} \\ t_2^{(3)} \\ t_3^{(3)} \end{array} \right\} = \mathbf{N}\mathbf{t}^{(n)} \tag{2.32}$$

em que $u_i^{(n)}$ e $t_i^{(n)}$ são os valores nodais de deslocamentos e forças de superfícies, respectivamente, e $N^{(i)}$ são as funções de forma definidas por:

$$N^{(1)} = \xi \tag{2.33}$$

$$N^{(2)} = \eta \tag{2.34}$$

$$N^{(3)} = 1 - \xi - \eta \tag{2.35}$$

em que ξ representa uma coordenada adimensional ao longo do elemento.

2.5 Elementos triangulares lineares contínuos

Neste trabalho, os elementos de superfície S_i são descritos por elementos triangulares planos pois são mais fáceis de gerar e possuem jacobiano constante ao longo do elemento, como será mostrado posteriormente.

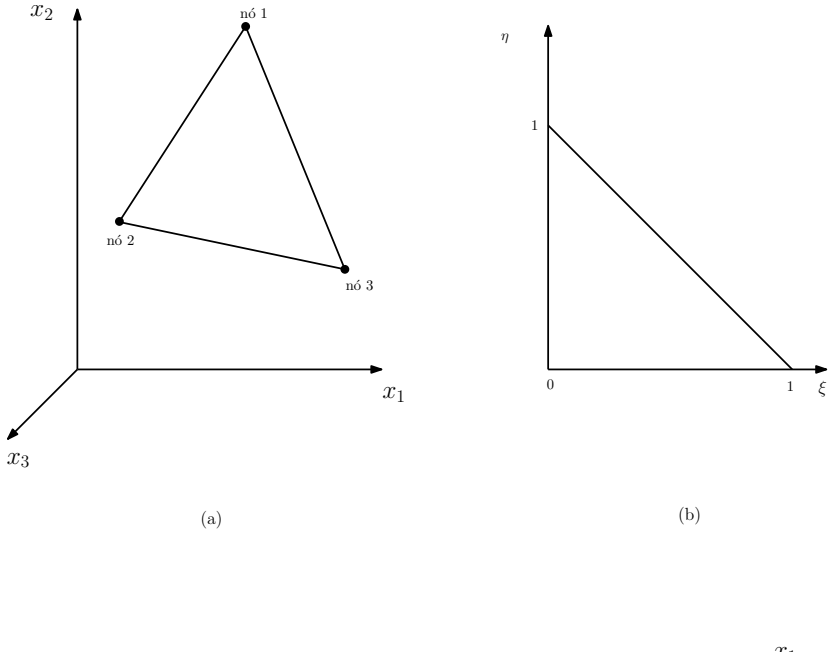

Figura 2.3: Elemento triangular linear: (a) no espaço real (x_1, x_2, x_3) e (b) no espaço paramétrico (ξ, η).

Uma vez que a integração de Gauss, por simplicidade, será realizada no intervalo $[0, 1]$, é coerente fazer o mapeamento do elemento S_i no intervalo $[0, 1]$.

Integrando o elemento triangular, obtemos a seguinte equação:

$$I_i = \iint_{S_i} f(x, y, z)dxdy = \int_0^1 \int_0^{1-\eta} f(x, y, z)Jd\xi d\eta \tag{2.36}$$

na qual *J* é o Jacobiano dado por:

$$J = \sqrt{q_1^2 + q_2^2 + q_3^2} \tag{2.37}$$

cujos valores de q_1, q_2 e q_3 são dados por:

$$\begin{cases} q_1 = \frac{dy}{d\xi}\frac{dz}{d\eta} - \frac{dy}{d\eta}\frac{dz}{d\xi} \\ q_2 = \frac{dz}{d\xi}\frac{dx}{d\eta} - \frac{dz}{d\eta}\frac{dx}{d\xi} \\ q_3 = \frac{dx}{d\xi}\frac{dy}{d\eta} - \frac{dx}{d\eta}\frac{dy}{d\xi} \end{cases} \tag{2.38}$$

Como:

$$\frac{\partial x}{\partial \xi} = \frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2 + \frac{\partial N_3}{\partial \xi} x_3 \tag{2.39}$$

e

$$\frac{\partial x}{\partial \eta} = \frac{\partial N_1}{\partial \eta} x_1 + \frac{\partial N_2}{\partial \eta} x_2 + \frac{\partial N_3}{\partial \eta} x_3 \tag{2.40}$$

então:

$$\begin{cases} \frac{\partial x}{\partial \xi} = x_1 - x_3 \\ \frac{\partial x}{\partial \eta} = x_2 - x_3 \end{cases} \tag{2.41}$$

Analogamente,

$$\begin{cases} \frac{\partial y}{\partial \xi} = y_1 - y_3 \\ \frac{\partial y}{\partial \eta} = y_2 - y_3 \end{cases} \tag{2.42}$$

e

$$\begin{cases} \frac{\partial z}{\partial \xi} = z_1 - z_3 \\ \frac{\partial z}{\partial \eta} = z_2 - z_3 \end{cases} \tag{2.43}$$

Daí, tem-se que:

$$\begin{cases} q_1 = (y_1 - y_3)(z_2 - z_3) - (y_2 - y_3)(z_1 - z_3) \\ q_2 = (z_1 - z_3)(x_2 - x_3) - (z_2 - z_3)(x_1 - x_3) \\ q_3 = (x_1 - x_3)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_3) \end{cases} \tag{2.44}$$

Uma mudança de variável, então, será requerida na equação (2.36) para que as duas integrais ocorram no intervalo de $[0, 1]$. Para isso, consideramos uma variável ξ' função de ξ de forma que:

$$\begin{cases} \xi'(\xi = 0) = 0 \\ \xi'(\xi = 1 - \eta) = 1 \end{cases} \tag{2.45}$$

Assim, considerando uma relação linear:

$$\xi' = a\xi + b \tag{2.46}$$

Obtemos dessa relação:

$$\begin{cases} \xi = (1-\eta)\xi' \\ \frac{d\xi}{d\xi'} = 1-\eta \end{cases} \tag{2.47}$$

Note que as derivadas das funções de forma dadas por:

$$\begin{cases} \frac{\partial N_1}{\partial \xi} = 1 \\ \frac{\partial N_1}{\partial \eta} = 0 \\ \frac{\partial N_2}{\partial \xi} = 0 \\ \frac{\partial N_2}{\partial \eta} = 1 \\ \frac{\partial N_3}{\partial \xi} = -1 \\ \frac{\partial N_3}{\partial \eta} = -1 \end{cases} \tag{2.48}$$

não dependem de ξ e η. Por isso, q_1, q_2 e q_3, o vetor normal **n** e o jacobiano J são constantes ao longo de todo o elemento. Nota-se que, neste caso, o jacobiano também pode ser dado pela equação (2.36) e é igual ao módulo vetorial

$$J = |\mathbf{v_2} \,\mathbf{x}\, \mathbf{v_1}| = \mathbf{2A_i} \tag{2.49}$$

no qual A_i = área do elemento i.

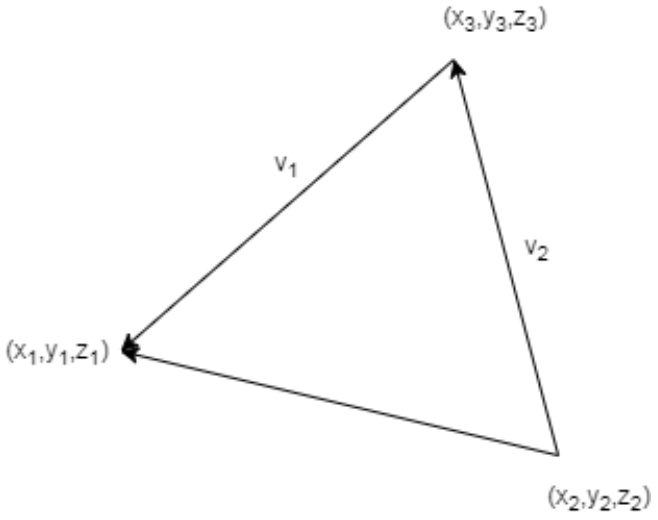

Figura 2.4: Vetores do elemento triangular linear.

Considere que o domínio tenha sido dividido em **NE** elementos de contorno. Substituindo as equações (2.31) e 2.32), tem-se

$$\mathbf{c}^l\mathbf{u}^l + \sum_{j=1}^{NE}\left\{\oint_{Sj}\mathbf{TNdS}\right\}\mathbf{u}^j = \sum_{j=1}^{NE}\left\{\iint_{S_j}\mathbf{UNdV}\right\}\mathbf{t}^j \tag{2.50}$$

Chamando

$$\iint_{S_j} \mathbf{UNdS} = \mathbf{G} \tag{2.51}$$

e

$$\iint_{S_j} \mathbf{TNdS} = \mathbf{H} \tag{2.52}$$

tem-se

$$\sum_{j=1}^{N} H^{lj} u^j = \sum_{j=1}^{N} G^{lj} t^j \tag{2.53}$$

ou, na forma matricial

$$\mathbf{Hu = Gt} \tag{2.54}$$

2.6 Integração no Espaço

Neste trabalho, os termos não singulares das matrizes $\mathbf{H}$ e $\mathbf{G}$ são integrados utilizando-se quadratura de Gauss padrão com 16 pontos de integração. Os termos singulares de $\mathbf{G}$ são do tipo $1/r$ sendo integrados usando o processo descrito por (KANE, 1994). Já os termos singulares de $\mathbf{H}$ são do tipo $1/r$ e precisam ser calculados no sentido do valor principal de Cauchy. Uma maneira bastante simples de se tratar esta singularidade é através de considerações de corpos rígidos. Assumindo que um corpo rígido tenha todos os seus pontos do contorno deslocados de um valor unitário e que não existam forças de corpo ($b_i = 0$) na direção de um dos eixos de coordenadas, as forças de superfície em qualquer ponto do contorno deste corpo deve ser zero. Desta forma, a equação (2.54) torna-se

$$\mathbf{Hv}^q = 0 \tag{2.55}$$

em que $\mathbf{v}^q$ é um vetor que, para todos os nós, tem deslocamentos unitários ao longo da direção q e zero na outra direção. Para satisfazer a equação (2.55) tem-se

$$H_{ii} = -\sum_{j=1}^{N} H_{ij} \qquad j \neq i \tag{2.56}$$

sendo j par ou ímpar.

O termo da diagonal da matriz $\mathbf{H}$ é igual à soma de todos os outros termos fora da diagonal correspondentes ao grau de liberdade em consideração.

2.7 Cálculo dos deslocamentos e tensões em pontos internos

O tensor de tensões para um ponto no interior do domínio V, obtido derivando-se a equação (2.28) neste ponto e aplicando-se a lei de Hooke, pode ser escrito como:

$$\sigma_{ik} + \iint_S S_{jik} u_j dS = \iint_S D_{jik} t_j dS \tag{2.57}$$

em que S_{kij} e D_{kij} são combinações lineares das derivadas de T_{ij} e U_{ij}, respectivamente.

Capítulo 3

Método dos elementos de contorno com expansão em multipolos

O método dos elementos de contorno com expansão em multipolos (FMM) é um modelo numérico, baseado na expansão de multipolos para diminuir o tempo de cálculo de fontes distantes. O conceito de multipolos aproxima um grupo de fontes em uma única fonte de forma que quanto mais as distâncias aumentam, grupos cada vez maiores são definidos pela decomposição hierárquica do espaço. O FMM tem aplicação em problemas que tenham produtos de matrizes muito densas por vetores, pois blocos destas matrizes podem ser aproximados por matrizes de baixo posto ($rank$) e o produto matriz vetor pode ser realizado de forma mais rápida e com reduzido uso da memória do computador. Este método não resolve exatamente o problema proposto, mas sim um problema muito similar, na qual a matriz para solução do problema é de menor complexidade. O processo de solução iterativa deste método também foi utilizado no método dos momentos aplicado a problemas computacionais eletromagnéticos, cálculos de densidade funcional em química quântica, dentre outros (CALLAHAN; KOSARAJU, 1995).

Este capítulo faz uma breve descrição dos conceitos no qual se baseia o método dos elementos de contorno com expansão em multipolos. Uma descrição mais detalhada está fora do escopo deste trabalho e pode ser encontrada em (LIU, 2009) e (BRAGA, 2012).

3.1 Princípio

O método com expansão em multipolos surgiu da física de partículas, nas quais se busca analisar a influência de **N** fontes em **M** pontos. Os algoritmos do MEC comuns costumam relacionar todas as fontes com os pontos a serem avaliados, um a um. Se o número de fontes **N** e o número de pontos a serem avaliados é **M**, o algoritmo roda um total de **N** x **M** operações. Enquanto isso, o algoritmo de Middleman em (GUMEROV, 2004), no qual o método dos elementos de contorno rápido é embasado, acopla todas as fontes em um único

grupo, fazendo assim um total de **N** + **M** operações como ilustrado na Figura 3.1.

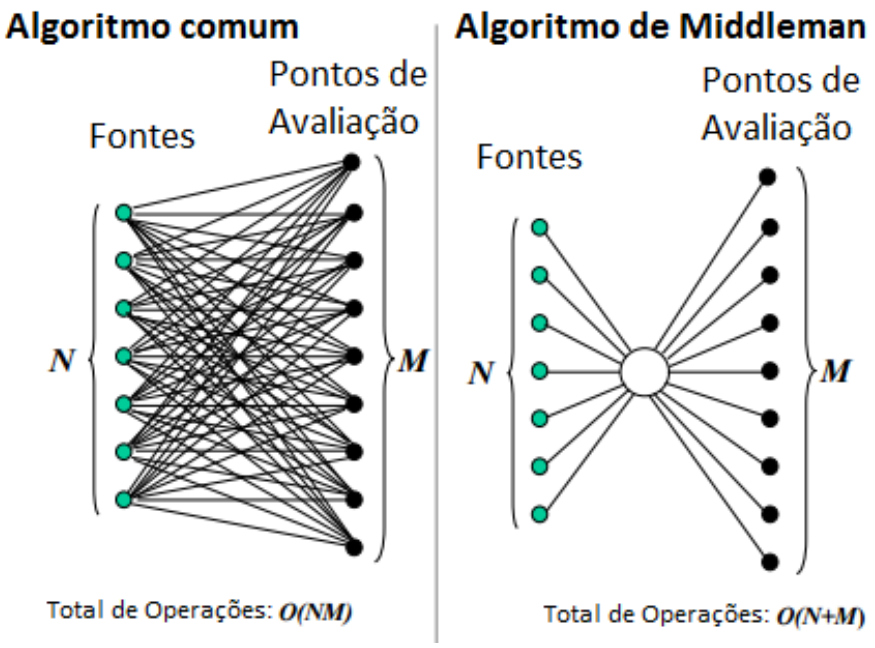

Figura 3.1: Algoritmo de Middleman.

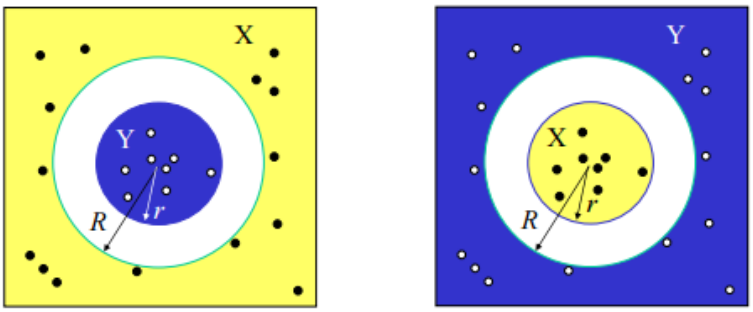

Figura 3.2: Conjuntos bem separados.

Conjuntos bem separados são definidos quando os pontos de um dos conjuntos das duas esferas concêntricas de raios **r** e **R**, sendo **r < R**, estão localizados dentro da esfera menor e os pontos do outro conjunto estão localizados externamente à esfera maior como representados na Figura 3.2.

O método dos elementos de contorno com expansão em multipolos utiliza-se do princípio do algoritmo de Middleman, considerando nós e elementos como fontes e pontos de avaliação, respectivamente, e excede-se de forma que se criam vários grupos intermediários, alocando assim mais fontes em cada grupo e diminuindo a quantidade necessária de operações como se vê na Figura 3.3.

Aumentando-se a complexidade da criação dos grupos e suas interações, diminui-se ainda mais a quantidade de operações necessárias para alcançar um mesmo resultado, como mostra a Figura 3.4, sendo que quanto maior a quantidade de grupos intermediários, menor será a quantidade de operações.

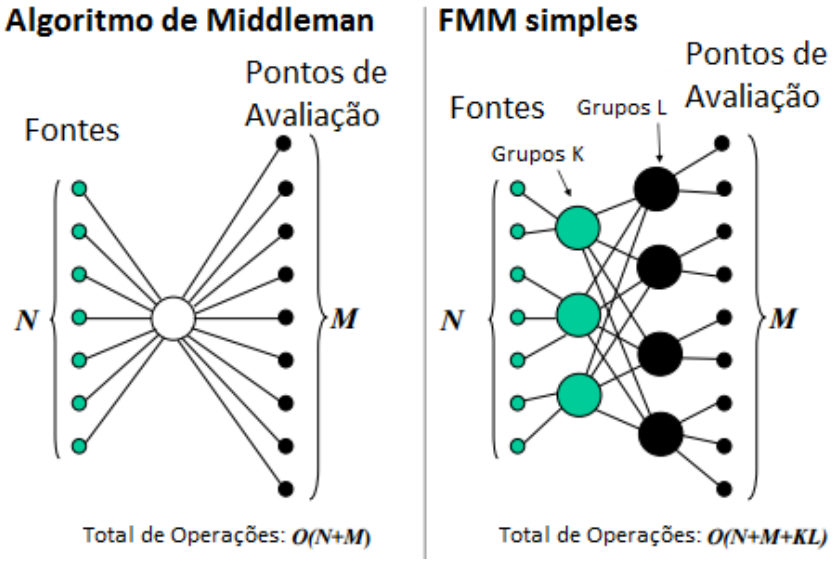

Figura 3.3: FMM de um nível.

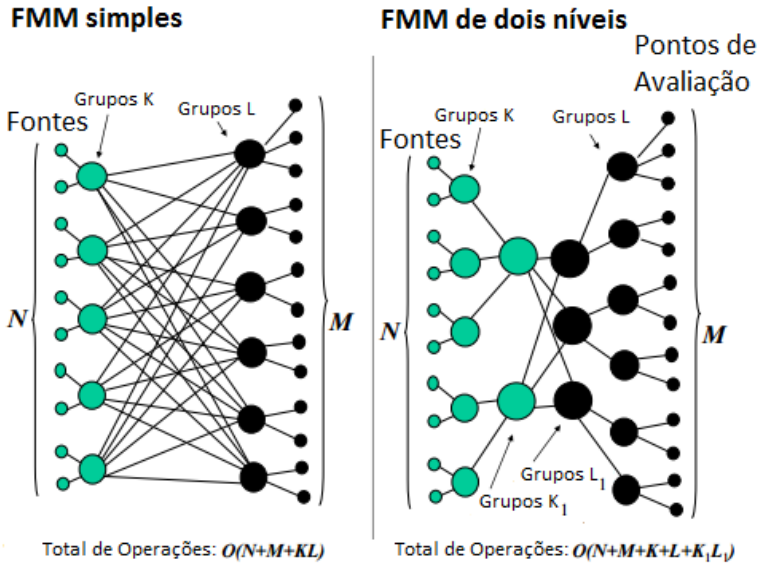

Figura 3.4: FMM de dois níveis.

Na Figura 3.5 pode-se observar um esquema de múltiplos níveis usando o método de elementos de contorno com expansão em multipolos.

Um sistema que funcione com o método dos elementos de contornos rápidos deve criar, agrupar e relacionar as fontes, grupos e resultados só uma vez, para que a partir desse ponto sejam somente feitas as diversas operações entre as variáveis (GUMEROV, 2004).

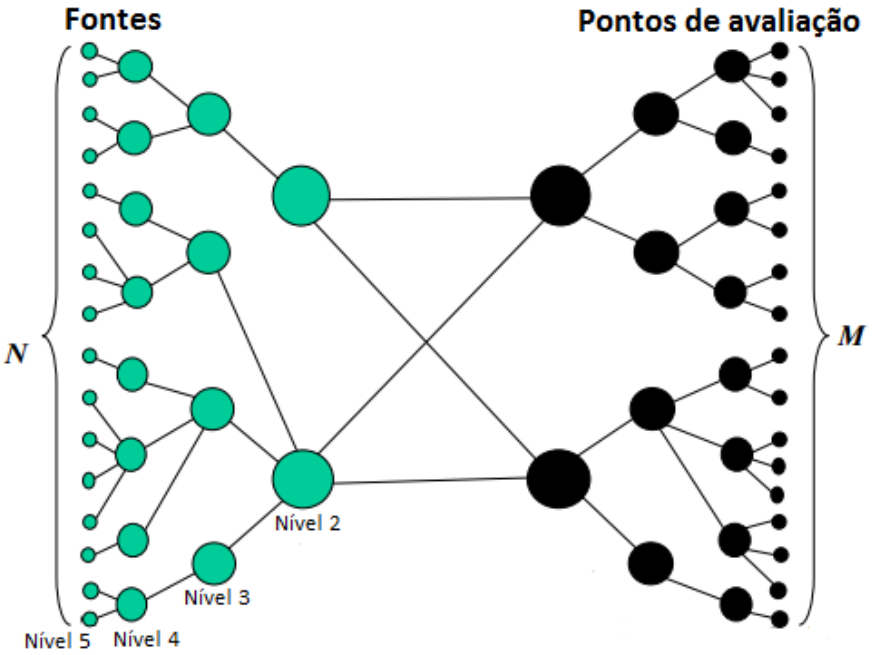

Figura 3.5: FMM de múltiplos níveis.

Capítulo 4

Programa

4.1 Rotina do programa

Este capítulo explana todas as funções utilizadas neste trabalho que foram implementados em linguagem Python, detalhando toda a execução do programa com os pacotes desenvolvidos para a formulação dos métodos dos elementos de contorno aplicada à análise de problemas elásticos em sólidos. O primeiro passo para abordar o problema consiste em modelar o sólido. Existem diferentes métodos para construção de sólidos, se forem de geometria simples, é possível manipular as curvas diretamente, mas para geometrias mais complexas o uso de *softwares* CAD dedicados torna-se necessário.

O fluxograma da Figura 4.1 exemplifica as etapas do trabalho começando pela criação do sólido com a ajuda de um programa CAD (FreeCad), a exportação do sólido em formato BREP para sua leitura no programa de geração de malha (GMSH), a subsequente geração de malha e a gravação do arquivo no formato MSH que pode ser lido pelo programa em Python.

A execução do programa desenvolvido no Python inicia com a leitura da malha a fim de poder analisar a geometria escolhida a partir de três coordenadas (x, y, z). Então, cria-se o arquivo de entrada de dados que contém os tipos das condições de contorno e as suas magnitudes, executa-se um programa desenvolvido por (LIU, 2016), que é responsável por todo o processamento pelo método dos elementos de contorno com expansão em multipolos, criação do arquivo de saída com os dados dos deslocamentos e forças em cada nó, e então, finalmente, é feito o pós-processamento no qual plota-se um gráfico do deslocamento ao qual está sujeito cada elemento da malha do sólido.

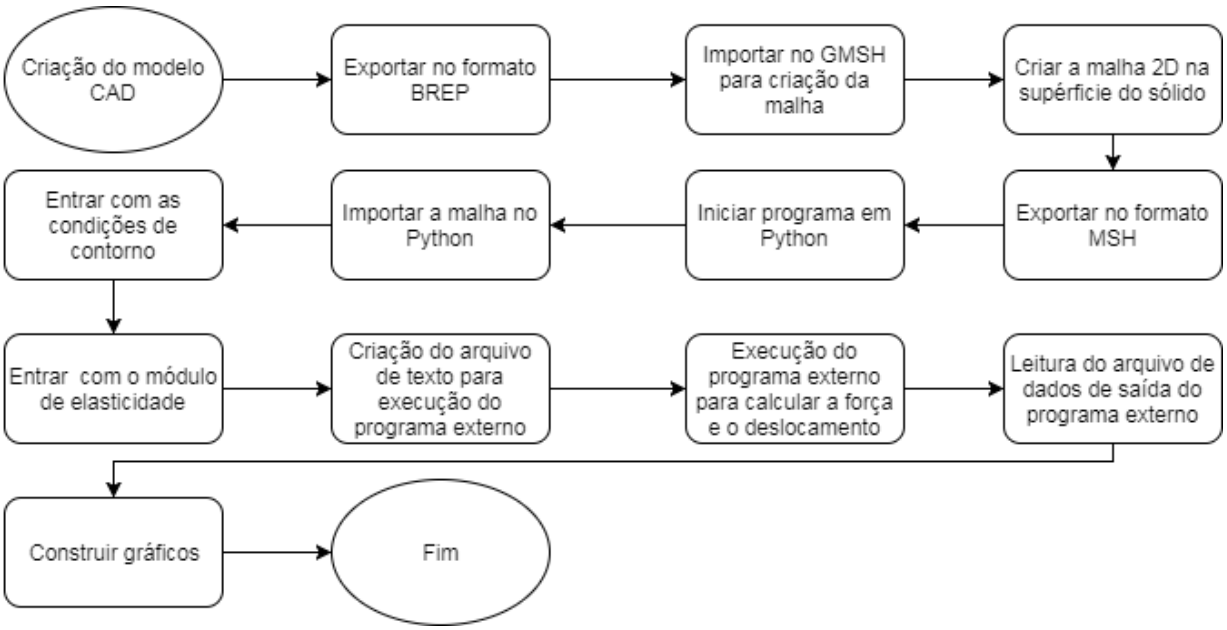

Figura 4.1: Fluxograma dos processos do trabalho.

4.2 Funções

Nesta seção é feita uma descrição de todas as funções que foram implementadas em Python. A listagem completa do programa encontra-se no anexo.

1. Função "Input data": essa função define de forma simples os parâmetros que o programa irá executar. Usando uma variável "file", que irá receber o nome do arquivo de malha criado no GMSH, e uma coleção do tipo dicionário nomeado como "surf.bc", no qual as chaves são definidas como as superfícies enumeradas no GMSH e sobre as quais as condições de contorno são aplicadas, que dão acesso à uma matriz com o tipo da condição de contorno (deslocamento ou força) em cada uma das direções e também sua magnitude. Esta função retorna ao programa principal as variáveis "file" e "surf.bc".

2. Função "Mymesh": essa função gratuita desenvolvida pela comunidade do Python que utiliza o GMSH foi retirada de https://github.com/lukeolson/python-mesh-scripts. Ela lê o arquivo de malha gerado no GMSH e retorna as variáveis "elem", definida como uma matriz com o número de linhas igual ao número de elementos e com cada linha tendo três termos que são os números dos nós que compõe cada elemento, e "surf" que é um vetor no qual são armazenados os índices das superfícies nas quais serão aplicados as condições de contorno definidos em "surf.bc" no item 1.

3. Função "Boundary": essa função recebe os argumentos "surf.bc", "surf" e "elem" e então cria uma matriz do tipo de condição de contorno "tipocdc" que apresenta duas colunas, sendo a primeira a identidade do elemento e a segunda 0, 1 ou 2 em respeito às direções x, y e z, respectivamente. Essa matriz é criada com a função "zeros" do

pacote "numpy", definindo todos os valores da matriz como inteiros e inicialmente iguais a zero, para alocar os valores mais rapidamente no futuro. É criada uma matriz similar para guardar os valores "valorcdc", ou seja, a magnitude das condições de contorno em cada direção, porém não sendo restrita a números inteiros.

Depois de definidas as variáveis, é iniciado um laço que percorre todo o dicionário "surf.bc". Com uma variável auxiliar "ix" que retorna a função "where" do pacote "numpy" quando "surf.bc" é igual a "surf", ou seja, retorna a identidade do elemento que faz parte da superfície de contorno, são definidos o tipo da condição de contorno no elemento (primeiro elemento da matriz após utilizar a chave, que é o número da superfície de contorno) e também a magnitude (segundo elemento da mesma matriz), sempre nas direções x, y e z.

A função retorna ao programa principal as variáveis "tipocdc" e "valorcdc".

4. Função "Cria arquivo de texto": essa função tem como objetivo preparar um arquivo de dados que será necessário na execução do programa externo criado por Liu (2016) e para isso recebe as variáveis "ELEM", "NOS", "file", "tipocdc" e "valorcdc". A função inicia definindo algumas variáveis, como "nelem", à qual é atribuído o comprimento da matriz "ELEM"; a variável "nnos" recebe o comprimento da matriz "NOS" e ainda o módulo de elasticidade do material "E", inserido pelo usuário.

   Usa-se a função "open" do Python para abrir um arquivo com o argumento "w" para escrever no arquivo. Então com a função "write" do pyhton, escreve-se no arquivo de dados o título do arquivo (neste caso o nome do arquivo de malha), as variáveis "nelem", "nnos" e "E".

   Cria-se um laço que percorre todos os nós ("i" de zero até "nnos"), no qual se atribui o valor de "x" como sendo a primeira coluna da matriz "NOS", "y" como sendo a segunda coluna em "NOS" e "z" como sendo a terceira coluna em "NOS"; para cada nó, escreve-se as componentes no arquivo de dados. Similarmente, depois, cria-se outro laço que percorre todos os elementos ("i" de zero até "nelem" menos um), no qual se atribui o valor de "no1" como sendo a primeira coluna da matriz "ELEM", "no2" como sendo a segunda coluna em "ELEM" e "no3"como sendo a terceira coluna em "ELEM". Então insere-se no arquivo as variáveis "no1", "no2", "no3", a matriz "tipocdc" nas três direções e a matriz "valorcdc", também nas três direções. Por fim, fecha-se o arquivo com a função "close" do Python.

5. Função "RunFMM": essa função é responsável por executar o programa externo do Liu (2016), sendo que para isso, define-se uma variável "cmd" que recebe o local do arquivo executável no computador. Fazendo uso da função "subprocess.Popen" com argumento "cmd" que inicia o executável e da função "getmtime" que recebe a data da última modificação do arquivo "output.dat" (se o arquivo não existir o valor é zero) que é criado pelo executável inicia-se um laço "while" que pausa o programa em Python até o momento no qual o arquivo "output.dat" é criado. Depois de criado o arquivo

"output.dat", executa-se a função "Lê resultados" e a função "proc.kill" encerra o processo iniciado pelo arquivo executável.

6. Função "Lê resultados": essa função, usando o comando "genfromtxt" do pacote numpy, recebe as linhas individuais do arquivo de texto "output.dat" criado em "RunFMM". A primeira coluna do arquivo diz respeito aos nós; as próximas três colunas são os valores do deslocamento nas direções x, y e z; as últimas três colunas representam a força atuante em cada uma das três direções. Depois de lido o arquivo e alocado cada uma das colunas com suas variáveis, é calculado o deslocamento total, "dt", como sendo a raiz quadrada da soma de cada direção do deslocamento ao quadrado. Por fim, a função retorna a variável "dt" à função "RunFMM" do item 5.

7. Função "Mostra resultados": essa função tem como parâmetros as variáveis "elem", "coord" e "dt" e gera um gráfico do tipo "Mapa de Calor" sobre a superfície da peça definida na variável "file".

Capítulo 5

Resultados

Este capítulo apresenta os resultados numéricos obtidos pelo programa implementado ao longo deste projeto. São mostrados exemplos com os valores do módulo de elasticidade igual a um, coeficiente de Poisson (ν) igual a zero e variação de complexidade geométrica.

5.1 Cubo

O cubo foi escolhido para uma análise simples do código, sabendo que quando uma força unitária é aplicada em um cubo de dimensões unitárias e o módulo de elasticidade do material também é unitário, observa-se um deslocamento unitário. Aplicou-se então uma força unitária na superfície dois na direção x e um deslocamento nulo na superfície um nas três direções. A numeração das superfícies é apresentada na Figura 5.1.

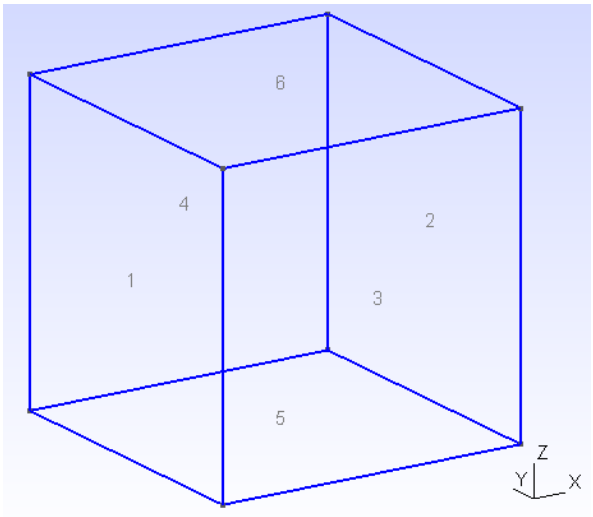

Figura 5.1: Enumeração das superfícies do cubo.

A entrada dos dados no programa em Python se deu a partir da variável dicionário "surf.bc", que recebeu as chaves zero e um representando as superfícies um e dois, pois a numeração do Python inicia-se no índice zero ao invés do um. Então, no valor de chave

igual a zero, da superfície um, como foi definido um deslocamento nulo, as três direções x, y e z recebem o tipo da condição de contorno igual a um e o seu valor igual a zero. No valor de chave igual a um, da superfície dois, como foi definida uma força unitária na direção x, o primeiro vetor referente a essa chave recebe a condição de contorno igual a dois e o valor igual a um, enquanto as outras duas direções recebem a condição de contorno igual a dois e o valor igual a zero.

As condições de contorno nas faces que não são listadas em "surf.bc" são as de superfície livre, ou seja, as forças iguais a zero em todas as direções.

```
def input_data():
  #name of the file
  file = 'Cubo_Unitario';
  #boundary condition 1:deslocamento 2:forca
  surf_bc={0:[[1,0],[1,0],[1,0]],1:[[2,1],[2,0],[2,0]]}
  return file,surf_bc
```

A Figura 5.2 apresenta o resultado esperado para a aplicação de uma força unitária.

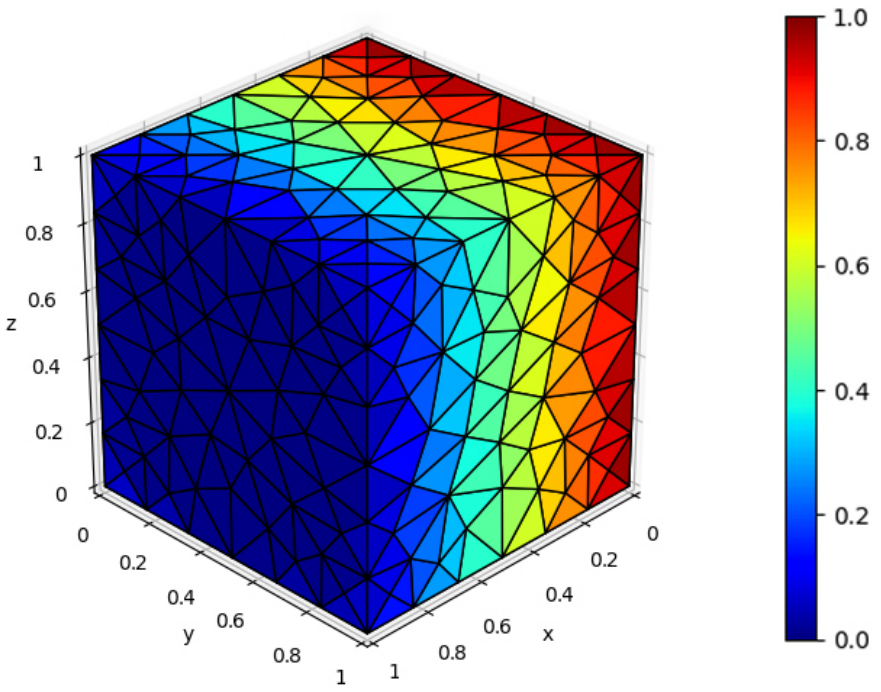

Figura 5.2: Deslocamento total plotado no cubo com força unitária aplicada.

O tempo de uso da CPU para solucionar o caso com 314 nós e 624 elementos foi de 88,550 segundos.

5.2 Placa com furo

O segundo exemplo trata-se de uma placa com furo, cujo de comprimento igual a cem, largura igual a cem, altura igual a dez e raio do furo igual a quinze. As condições contorno foram aplicadas nas superfícies dois e quatro, sendo as chaves um e três no Python vistas na Figura 5.3, respectivamente.

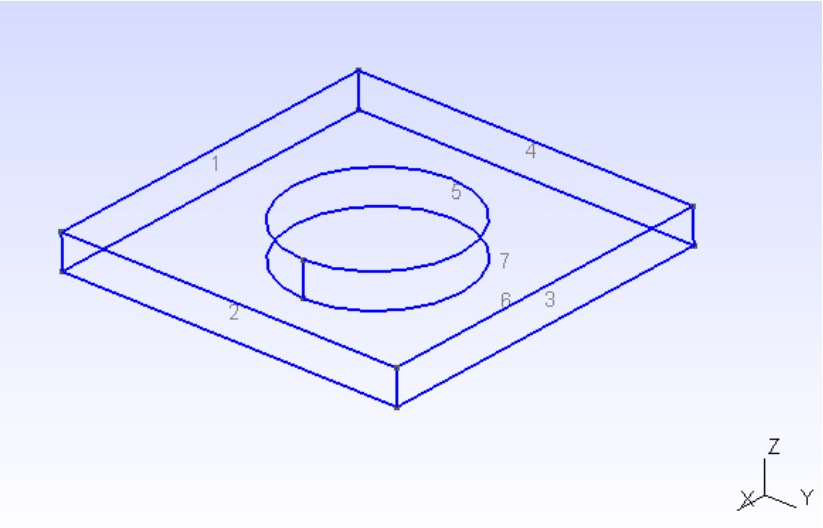

Figura 5.3: Enumeração das superfícies do sólido com furo.

Sobre a primeira superfície foi aplicado um deslocamento nulo nas três direções e sobre a segunda superfície, uma força unitária na direção x.

```
def input_data():
  #name of the file
  file = 'Placa_furo';
  #boundary condition 1:deslocamento 2:forca
  surf_bc={1:[[1,0],[1,0],[1,0]],3:[[2,1],[2,0],[2,0]]}
  return file,surf_bc
```

Na Figura 5.4 pode-se observar o mapa de calor com o módulo do vetor deslocamento nos nós. O resultado é o esperado, com deslocamento nulo na face engastada e máximo na face em que a força é aplicada.

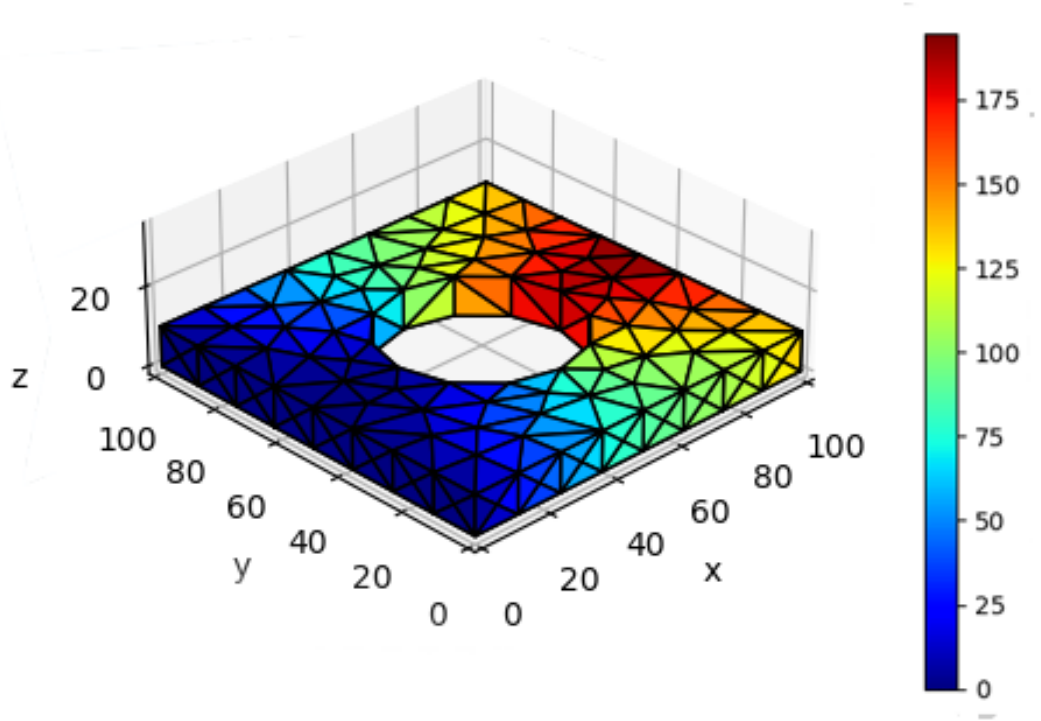

Figura 5.4: Deslocamento total plotado em um sólido com furo.

O tempo de uso da CPU para solucionar o caso com 172 nós e 344 elementos foi de 16,349 segundos.

5.3 Estrutura em L

A estrutura em "L" deve apresentar um comportamento diferente em relação aos anteriores, pelo fato de parte da estrutura estar sob flexão. A peça tem dimensões com comprimento e largura iguais a cem e altura igual a vinte e cinco.

5.3.1 Caso 1

As condições de contorno foram aplicadas sobre as superfícies quatro, com deslocamento nulo, e seis, com força unitária na direção x, da Figura 5.5, e as chaves do programa foram os valores três e cinco.

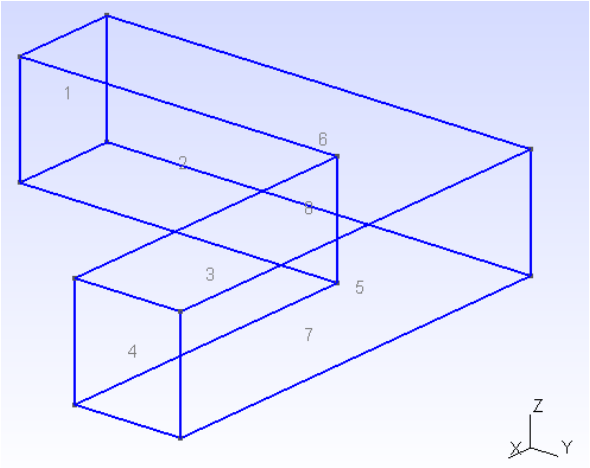

Figura 5.5: Enumeração das superfícies em uma estrutura com formato "L".

```
def input_data():
  #name of the file
  file = 'EstruturaEmL';
  #boundary condition 1:deslocamento 2:forca
  surf_bc={3:[[1,0],[1,0],[1,0]],5:[[2,1],[2,0],[2,0]]}
  return file,surf_bc
```

A força é aplicada de forma distribuída na superfície em todos os casos e, por esse motivo, o lado apoiado apresenta um deslocamento total menor do que o lado sem apoio, como mostra a Figura 5.6.

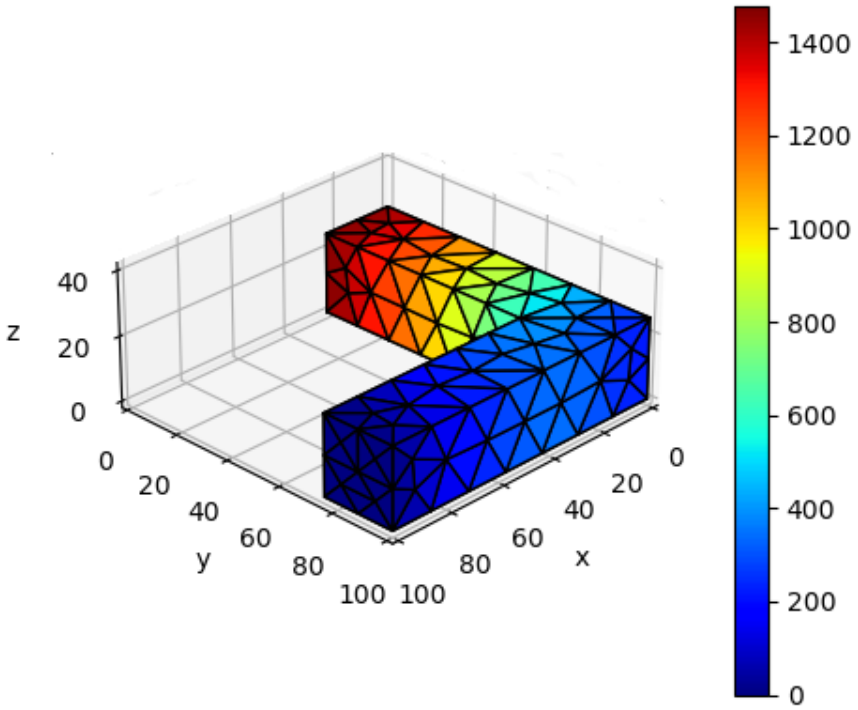

Figura 5.6: Deslocamento total plotado em um sólido com formato "L".

O tempo de uso da CPU para solucionar o caso com 146 nós e 288 elementos foi de 28,895 segundos.

5.3.2 Caso 2

Se as condições de contorno são aplicadas sobre a superfície quatro, um deslocamento nulo, e sobre a superfície um, uma força unitária na direção y, ocorre um maior deslocamento ao longo do eixo y, sendo o resultado dado pela Figura 5.7

```
def input_data():
  #name of the file
  file = 'EstruturaEmL';
  #boundary condition 1:deslocamento 2:forca
  surf_bc={3:[[1,0],[1,0],[1,0]],0:[[2,0],[2,1],[2,0]]}
  return file,surf_bc
```

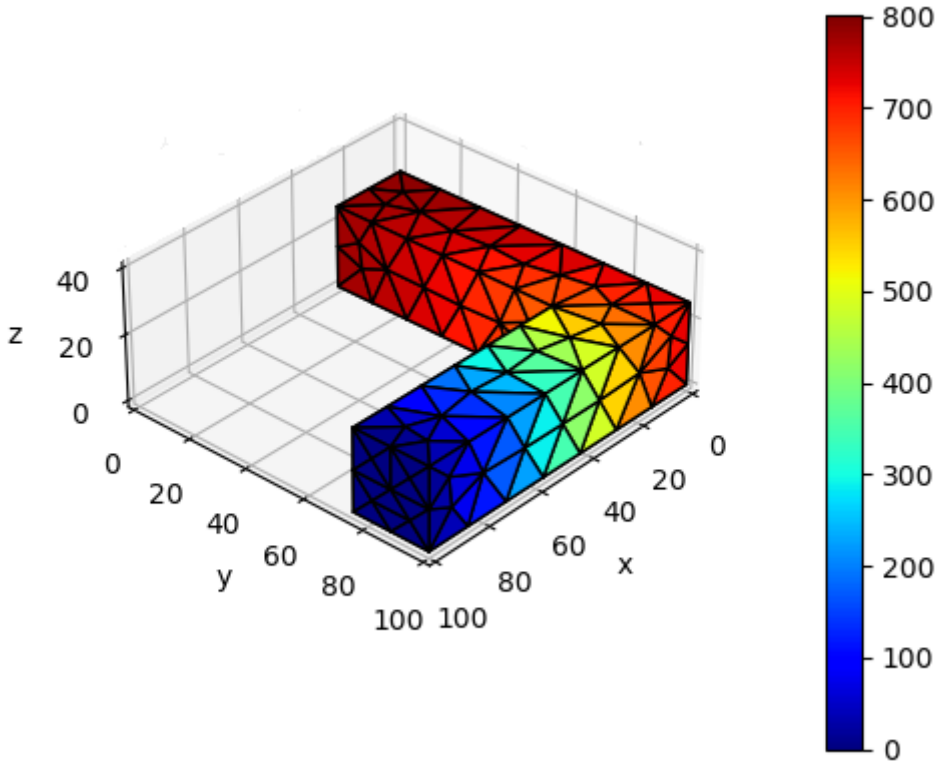

Figura 5.7: Deslocamento total plotado em um sólido com formato "L".

O tempo de uso da CPU para solucionar o caso com 146 nós e 288 elementos foi de 48,407 segundos.

5.4 Mancal

Para finalizar, ainda foi simulado um exemplo prático, geometricamente mais complexo, utilizando como peça um mancal. A peça teve as condições de contorno aplicadas nas superfícies seis (face mais à esquerda) e treze (face mais à direita), ilustrado na figura 5.8, força unitária na direção x e deslocamento nulo, respectivamente, sendo então observados os pontos críticos neste tipo de solicitação. As chaves do dicionário foram os valores cinco e doze.

```
def input_data():
#name of the file
file = 'Mancal';
#boundary condition 1:deslocamento 2:forca
surf_bc={5:[[2,1],[1,0],[1,0]],12:[[1,0],[1,0],[1,0]]}
return file,surf_bc
```

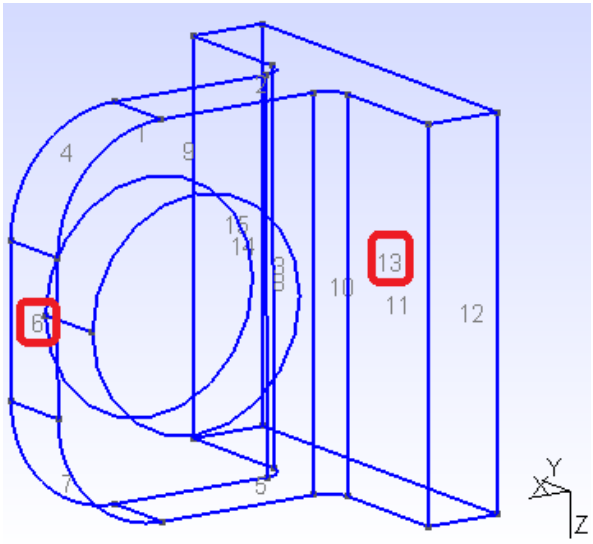

Figura 5.8: Enumeração das superfícies do mancal.

A Figura 5.9 mostra o mapa de calor com o módulo dos deslocamentos nodais. O resultado obtido foi o esperado com deslocamentos maiores na superfície carregada e zero na superfície engastada.

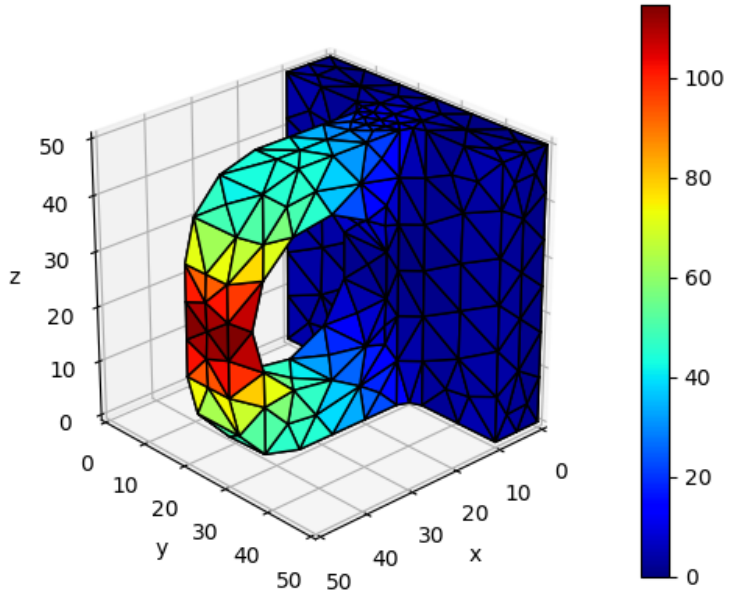

Figura 5.9: Deslocamento total plotado em um mancal.

O tempo de uso da CPU para solucionar o caso com 294 nós e 588 elementos foi de 25,194 segundos.

Capítulo 6

Conclusões e trabalhos futuros

6.1 Conclusões

Este trabalho apresentou uma abordagem na qual o método dos elementos de contorno foi utilizado na análise de problemas elásticos tridimensionais em sólidos. O foco principal foi na preparação do arquivo de entrada de dados para o processamento com método dos elementos de contorno com expansão de multipolos, usando *softwares* de código aberto, com exceção do programa de cálculo dos deslocamentos e forças por (LIU, 2016).

Os sólidos apresentados nos resultados foram desenvolvidos em ambiente Freecad. Depois foram geradas as malhas tridimensionais no GMSH. Em seguida, foram lidas no Python, sendo chamado então um programa do método dos elementos de contorno com expansão em multipolos, que calculou deslocamentos e forças em todos os nós do contorno. Por fim, um mapa do deslocamento foi gerado na superfície do sólido.

A maior contribuição deste trabalho foi a integração de ferramentas computacionais de código aberto para resolver problemas estruturais.

As rotinas foram desenvolvidas de forma a serem rápidas, práticas e intuitivas de usar. Esta formulação pode ser ainda desenvolvida para problemas diferentes do que aqui apresentado, como já feito com a temperatura por (LOYOLA, 2017), sendo aplicável também em problemas eletromagnéticos e de radiação acústica, entre outros.

6.2 Trabalhos Futuros

Apesar dos bons resultados na geração da malha superficial e na análise de problemas tridimensionais, este projeto encontra-se em um estágio inicial que ainda demanda muitas melhorias para tornar o código amigável ao usuário. Como trabalhos futuros, são sugeridos os seguintes tópicos:

1. Otimizar o programa em Python com o Cyhton.

2. Desenvolver um código de elementos de contorno para substituir o programa do método dos elementos de contorno com expansão em multipolos.

3. Desenvolver a formulação para outras análises como problemas eletromagnéticos, acústicos, dentre outros.

ANEXOS

I.1 Programa Principal - Python

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jul 29 22:29:08 2017
example of 3d bem program using the Liu solver
@author: Buzogany
require normals pointing outward.
"""

from mesh import Mesh
import graphics_problem
import input_data
import boundary
import runFMM
import cria_inputFMMPot

file,surf_bc,k=input_data.input_data4()

mymesh = Mesh() #criou objeto: instanciou a classe
#chamei o metodo do objeto
mymesh.read_msh(file + '.msh')
coord = mymesh.Verts
# get only the triangular elements [[physid],[nodes]] physid is the surf
                                    of the elem

elem = mymesh.Elmts[2][1]-1
surf = mymesh.Elmts[2][0]-1

elem_bc, tipocdc, valorcdc = boundary.elem(surf_bc, surf, elem)

cria_inputFMMPot.cria_inputFMM(elem,coord,elem_bc,file,tipocdc,valorcdc)

resultado,dados = runFMM.elastostatics()

print("[RIM3D] Dados de 1 a 3:\n",dados[:,1:4])

graphics_problem.show_results1(elem,coord,resultado)
```

I.2 Função "Input Data" - Python

```
"""
@author: Buzogany
"""
def input_data1():
    #name of the file
    file = 'viga10';
    #boundary condition 1:deslocamento 2:forca
    surf_bc={4:[[1,0],[1,0],[1,0]],5:[[1,0],[1,0],[2,1]]}
    k=1
    return file,surf_bc,k

def input_data2():
    #name of the file
    #file = 'PlacaFuro';
    file = 'Placa_furo';
    #boundary condition 1:deslocamento 2:forca
    surf_bc={1:[[1,0],[1,0],[1,0]],3:[[2,1],[1,0],[1,0]]}
    k=1
    return file,surf_bc,k

def input_data21():
    #name of the file
    #file = 'PlacaFuro';
    file = 'Placa_furo_q';
    #boundary condition 1:deslocamento 2:forca
    surf_bc={3:[[1,0],[1,0],[1,0]],1:[[2,1],[2,0],[2,0]]}
    k=1
    return file,surf_bc,k

def input_data3():
    #name of the file
    #file = 'PlacaFuro';
    file = 'LPadrao';
    #boundary condition 1:deslocamento 2:forca
    surf_bc={3:[[1,0],[1,0],[1,0]],5:[[2,1],[1,0],[1,0]]}
    k=1
    return file,surf_bc,k
```

I.3 Função "Mymesh" - Python

```
from __future__ import print_function
# gmsh reader
# neu writer
# * handles triangles (2d), tets(3d)
# available at https://github.com/lukeolson/python-mesh-scripts
import numpy
from scipy.sparse import coo_matrix, triu
import sys


class Mesh:
    """
    Store the verts and elements and physical data
    attributes
    ----------
    Verts : array
        array of 3d coordinates (npts x 3)
    Elmts : dict
        dictionary of tuples
        (rank 1 array of physical ids, rank 2 array of element to vertex
                                                ids
        (Nel x ppe)) each array in the tuple is of length nElmts Phys :
                                                dict
        keys and names
    methods
    -------
    read_msh:
        read a 2.0 ascii gmsh file
    write_neu:
        write a gambit neutral file. works for tets, tris in 3d and 2d
    write_vtu:
        write VTK file (calling vtk_writer.py)
    """
    def __init__(self):

        self.Verts = []
        self.Elmts = {}
        self.Phys = {}

        self.npts = 0
        self.nElmts = {}
        self.nprops = 0

        self._elm_types()    # sets elm_type

        self.meshname = ""

```

```
    def read_msh(self, mshfile):
        """Read a Gmsh .msh file.
        Reads Gmsh 2.0 mesh files
        """
        self.meshname = mshfile
        try:
            fid = open(mshfile, "r")
        except IOError:
            print("File '%s' not found." % mshfile)
            sys.exit()

        line = 'start'
        while line:
            line = fid.readline()

            if line.find('$MeshFormat') == 0:
                line = fid.readline()
                if line.split()[0][0] is not '2':
                    print("wrong gmsh version")
                    sys.exit()
                line = fid.readline()
                if line.find('$EndMeshFormat') != 0:
                    raise ValueError('expecting EndMeshFormat')

            if line.find('$PhysicalNames') == 0:
                line = fid.readline()
                self.nprops = int(line.split()[0])
                for i in range(0, self.nprops):
                    line = fid.readline()
                    newkey = int(line.split()[0])
                    qstart = line.find('"')+1
                    qend = line.find('"', -1, 0)-1
                    self.Phys[newkey] = line[qstart:qend]
                line = fid.readline()
                if line.find('$EndPhysicalNames') != 0:
                    raise ValueError('expecting EndPhysicalNames')

            if line.find('$ParametricNodes') == 0:
                line = fid.readline()
                self.npts = int(line.split()[0])
                self.Verts = numpy.zeros((self.npts, 3), dtype=float)
                for i in range(0, self.npts):
                    line = fid.readline()
                    data = line.split()
                    idx = int(data[0]) - 1  # fix gmsh 1-based indexing
                    if i != idx:
                        raise ValueError('problem with vertex ids')
                    self.Verts[idx, :] = list(map(float, data[1:4]))
                line = fid.readline()
```

```
95              if line.find('$EndParametricNodes') != 0:
96                  raise ValueError('expecting EndParametricNodes')
97
98          if line.find('$Nodes') == 0:
99              line = fid.readline()
100             self.npts = int(line.split()[0])
101             self.Verts = numpy.zeros((self.npts, 3), dtype=float)
102             for i in range(0, self.npts):
103                 line = fid.readline()
104                 data = line.split()
105                 idx = int(data[0]) - 1  # fix gmsh 1-based indexing
106                 if i != idx:
107                     raise ValueError('problem with vertex ids')
108                 self.Verts[idx, :] = list(map(float, data[1:4]))
109             line = fid.readline()
110             if line.find('$EndNodes') != 0:
111                 raise ValueError('expecting EndNodes')
112
113         if line.find('$Elements') == 0:
114             line = fid.readline()
115             self.nel = int(line.split()[0])
116             for i in range(0, self.nel):
117                 line = fid.readline()
118                 data = line.split()
119                 idx = int(data[0]) - 1  # fix gmsh 1-based indexing
120                 if i != idx:
121                     raise ValueError('problem with elements ids')
122                 etype = int(data[1])            # element type
123                 ntags = int(data[2])            # number of tags
                                                        following
124                 k = 3
125                 if ntags > 0:                    # set physical id
126                     physid = int(data[k+1])
127                     if physid not in self.Phys:
128                         self.Phys[physid] = 'Physical Entity %d' %
                                                                physid
129                         self.nprops += 1
130                     k += ntags
131
132                 verts = list(map(int, data[k:]))
133                 verts = numpy.array(verts)  # fixe gmsh 1-based
134
135                 if (etype not in self.Elmts) or\
136                         (len(self.Elmts[etype]) == 0):
137                     # initialize
138                     self.Elmts[etype] = (physid, verts)
139                     self.nElmts[etype] = 1
140                 else:
141                     # append
```

```
142                         self.Elmts[etype] = \
143                             (numpy.hstack((self.Elmts[etype][0], physid))
                                                                          ,
144                              numpy.vstack((self.Elmts[etype][1], verts)))
145                         self.nElmts[etype] += 1
146 #                   print(self.Elmts)
147 
148                 line = fid.readline()
149                 if line.find('$EndElements') != 0:
150                     raise ValueError('expecting EndElements')
151         fid.close()
152 
153     def _find_EF(self, vlist, E):
154         for i in range(0, E.shape[0]):
155             enodes = E[i, :]
156             if len(numpy.intersect1d_nu(vlist, enodes)) == len(vlist):
157                 # found element.  now the face
158                 missing_node = numpy.setdiff1d(enodes, vlist)
159                 loc = numpy.where(enodes == missing_node)[0][0]
160 
161                 # determine face from missing id
162                 if len(enodes) == 3:  # tri
163                     face_map = {0: 1, 1: 2, 2: 0}
164                 if len(enodes) == 4:  # tet
165                     face_map = {0: 2, 1: 3, 2: 1, 3: 0}
166 
167                 return i, face_map[loc]
168 
169     def write_vtu(self, fname=None):
170         if fname is None:
171             fname = self.meshname.split('.')[0] + '.vtu'
172 
173         from vtk_writer import write_vtu
174         vtk_id = {1: 3, 2: 5, 4: 10, 15: 1}
175         Cells = {}
176         cdata = {}
177         k = 0.0
178         for g_id, E in self.Elmts.iteritems():
179             k += 1.0
180             if g_id not in vtk_id:
181                 raise NotImplementedError('vtk ids not yet implemented')
182             Cells[vtk_id[g_id]] = E[1]
183             cdata[vtk_id[g_id]] = k*numpy.ones((E[1].shape[0],))
184 
185         write_vtu(Verts=self.Verts, Cells=Cells, cdata=cdata, fname=fname
                                                 )
186 
187     def write_neu(self, fname=None):
188         """ works for tets, tris in 3d and 2d"""
```

```
189
190         neu_id = {1: 3, 2: 3, 4: 6, 15: 1}
191         neu_pts = {1: 2, 2: 3, 4: 4, 15: 1}
192
193         if fname is None:
194             fname = self.meshname.split('.')[0] + '.neu'
195
196         if type(fname) is str:
197             try:
198                 fid = open(fname, 'w')
199             except IOError as e:
200                 (errno, strerror) = e.args
201                 print(".neu error (%s): %s" % (errno, strerror))
202         else:
203             raise ValueError('fname is assumed to be a string')
204
205         if 4 not in self.Elmts:
206             mesh_id = 4  # mesh elements are tets
207             bc_id = 2    # bdy face elements are tris
208             dim = 3
209             print('... (neu file) assuming 3d, using tetrahedra')
210         elif 2 not in self.Elmts:
211             mesh_id = 2  # mesh elements are tris
212             bc_id = 1    # bdy face elements are lines
213             dim = 2
214             print('... (neu file) assuming 2d, using triangles')
215         else:
216             raise ValueError('problem with finding elements for neu file'
                                                    )
217
218         E = self.Elmts[mesh_id][1]
219         nel = self.nElmts[mesh_id]
220         if E.shape[0] != nel:
221             raise ValueError('problem with element shape and nel')
222
223         Eb = self.Elmts[bc_id][1]
224         nelb = self.nElmts[bc_id]
225         if Eb.shape[0] != nelb:
226             raise ValueError('problem with element shape and nel')
227
228         bd_id_list = self.Elmts[bc_id][0]
229         bd_ids = numpy.unique(bd_id_list)
230         nbc = len(bd_ids)
231
232         # list of Elements on the bdy and corresponding face
233         EF = numpy.zeros((nelb, 2), dtype=int)
234         for i in range(0, nelb):
235             vlist = Eb[i, :]
236             el, face = self._find_EF(vlist, E)
```

```
            EF[i, :] = [el+1, face+1]
            #           ^^      ^^   neu is 1 based indexing

        fid.write('        CONTROL INFO 1.3.0\n')
        fid.write('** GAMBIT NEUTRAL FILE\n\n\n\n')
        fid.write('%10s%10s%10s%10s%10s%10s\n' %
                  ('NUMNP', 'NELEM', 'NGRPS', 'NBSETS', 'NDFCD', 'NDFVL')
                                                          )
        data = (self.npts, nel, 0, nbc, dim, dim)
        fid.write('%10d%10d%10d%10d%10d%10d\n' % data)
        fid.write('ENDOFSECTION\n')
        fid.write('   NODAL COORDINATES 1.3.0\n')
        for i in range(0, self.npts):
            if dim == 2:
                fid.write('%d  %e  %e\n' %
                          (i+1, self.Verts[i, 0], self.Verts[i, 1]))
            # ^^^ neu is 1-based indexing
            else:
                fid.write('%d  %e  %e  %e\n' %
                          (i+1, self.Verts[i, 0], self.Verts[i, 1],
                           self.Verts[i, 2]))
            #     ^^^ neu is 1-based indexing
        fid.write('ENDOFSECTION\n')

        fid.write('ELEMENTS/CELLS 1.3.0\n')
        for i in range(0, nel):
            data = [i+1, neu_id[mesh_id], neu_pts[mesh_id]]
            #   ^^^ neu is 1-based indexing
            data.extend((E[i, :]+1).tolist())
            #   ^^^ neu is 1-based indexing
            dstr = ''
            for d in data:
                dstr += ' %d' % d
            fid.write(dstr+'\n')
        fid.write('ENDOFSECTION\n')
        #    Write all the boundary condition blocks IMPORTANT, it is
                                                  assumed
        #    that the the mesh_condition_name stores the BC type and the
                                                  first
        #    number to follow the BC in the .neu file.  For circular
        #    boundaries, mesh_condition_name[i] = "circ  _radius-length_"
                                                   For
        #    standard dirichlet boundaries, mesh_condition_name[i] = "
                                                  diri 1".
        #    examples
        #    Wall 1
        #    Inflow 1
        #    Outflow 1
        #
```

```
        #    1 means cell
        #    0 means node
        for bcid in range(0, nbc):
            this_bdy = numpy.where(bd_id_list == bd_ids[bcid])[0]
            bnel = len(this_bdy)
            fid.write('   BOUNDARY CONDITIONS 1.3.0\n')
            fid.write('%10s   %d   %d   %d\n' %
                      (self.Phys[bd_ids[bcid]], bnel, 0, 0))
            for i in range(0, bnel):
                el = EF[this_bdy[i], 0]
                face = EF[this_bdy[i], 1]
                fid.write(' %d %d %d \n' % (el, neu_id[mesh_id], face))
            fid.write('ENDOFSECTION\n')

        fid.close()

    def _elm_types(self):
        elm_type = {}
        elm_type[1] = 2      # 2-node line
        elm_type[2] = 3      # 3-node triangle
        elm_type[3] = 4      # 4-node quadrangle
        elm_type[4] = 4      # 4-node tetrahedron
        elm_type[5] = 8      # 8-node hexahedron
        elm_type[6] = 6      # 6-node prism
        elm_type[7] = 5      # 5-node pyramid
        elm_type[8] = 3      # 3-node second order line
                             # (2 nodes at vertices and 1 with edge)
        elm_type[9] = 6      # 6-node second order triangle
                             # (3 nodes at vertices and 3 with edges)
        elm_type[10] = 9     # 9-node second order quadrangle
                             # (4 nodes at vertices,
                             #  4 with edges and 1 with face)
        elm_type[11] = 10    # 10-node second order tetrahedron
                             # (4 nodes at vertices and 6 with edges)
        elm_type[12] = 27    # 27-node second order hexahedron
                             # (8 nodes at vertices, 12 with edges,
                             #  6 with faces and 1 with volume)
        elm_type[13] = 18    # 18-node second order prism
                             # (6 nodes at vertices,
                             #  9 with edges and 3 with quadrangular faces
                                                                  )
        elm_type[14] = 14    # 14-node second order pyramid
                             # (5 nodes at vertices,
                             #  8 with edges and 1 with quadrangular face)
        elm_type[15] = 1     # 1-node point
        elm_type[16] = 8     # 8-node second order quadrangle
                             # (4 nodes at vertices and 4 with edges)
        elm_type[17] = 20    # 20-node second order hexahedron
                             # (8 nodes at vertices and 12 with edges)
```

```
        elm_type[18] = 15    # 15-node second order prism
                             # (6 nodes at vertices and 9 with edges)
        elm_type[19] = 13    # 13-node second order pyramid
                             # (5 nodes at vertices and 8 with edges)
        elm_type[20] = 9     # 9-node third order incomplete triangle
                             # (3 nodes at vertices, 6 with edges)
        elm_type[21] = 10    # 10-node third order triangle
                             # (3 nodes at vertices, 6 with edges, 1 with
                                                                    face)
        elm_type[22] = 12    # 12-node fourth order incomplete triangle
                             # (3 nodes at vertices, 9 with edges)
        elm_type[23] = 15    # 15-node fourth order triangle
                             # (3 nodes at vertices, 9 with edges, 3 with
                                                                    face)
        elm_type[24] = 15    # 15-node fifth order incomplete triangle
                             # (3 nodes at vertices, 12 with edges)
        elm_type[25] = 21    # 21-node fifth order complete triangle
                             # (3 nodes at vertices, 12 with edges, 6 with
                                                                     face)
        elm_type[26] = 4     # 4-node third order edge
                             # (2 nodes at vertices, 2 internal to edge)
        elm_type[27] = 5     # 5-node fourth order edge
                             # (2 nodes at vertices, 3 internal to edge)
        elm_type[28] = 6     # 6-node fifth order edge
                             # (2 nodes at vertices, 4 internal to edge)
        elm_type[29] = 20    # 20-node third order tetrahedron
                             # (4 nodes at vertices, 12 with edges,
                             #  4 with faces)
        elm_type[30] = 35    # 35-node fourth order tetrahedron
                             # (4 nodes at vertices, 18 with edges,
                             #  12 with faces, 1 in volume)
        elm_type[31] = 56    # 56-node fifth order tetrahedron
                             # (4 nodes at vertices, 24 with edges,
                             #  24 with faces, 4 in volume)
        self.elm_type = elm_type

    def refine2dtri(self, marked_elements=None):
        """
        marked_elements : array
            list of marked elements for refinement.  None means uniform.
        bdy_ids : array
            list of ids for boundary lists
        """
        E = self.Elmts[2][1]
        Nel = E.shape[0]
        Nv = self.Verts.shape[0]

        if marked_elements is None:
            marked_elements = numpy.arange(0, Nel)
```

```
375
376         marked_elements = numpy.ravel(marked_elements)
377         ############################################################
378         # construct vertex to vertex graph
379         col = E.ravel()
380         row = numpy.kron(numpy.arange(0, Nel), [1, 1, 1])
381         data = numpy.ones((Nel*3,))
382         V2V = coo_matrix((data, (row, col)), shape=(Nel, Nv))
383         V2V = V2V.T * V2V
384
385         # compute interior edges list
386         V2V.data = numpy.ones(V2V.data.shape)
387         V2Vupper = triu(V2V, 1).tocoo()
388
389         # construct EdgeList from V2V
390         Nedges = len(V2Vupper.data)
391         V2Vupper.data = numpy.arange(0, Nedges)
392         EdgeList = numpy.vstack((V2Vupper.row, V2Vupper.col)).T
393         self.EdgeList = EdgeList
394         Nedges = EdgeList.shape[0]
395
396         # elements to edge list
397         V2Vupper = V2Vupper.tocsr()
398         edges = numpy.vstack((E[:, [0, 1]],
399                               E[:, [1, 2]],
400                               E[:, [2, 0]]))
401         edges.sort(axis=1)
402         ElementToEdge = V2Vupper[edges[:, 0], edges[:, 1]].reshape((3,
                                                 Nel)).T
403         self.ElementToEdge = ElementToEdge
404
405         # mark edges as boundary
406         BE = self.Elmts[1][1]
407         BE.sort(axis=1)
408         BEdgeList = numpy.zeros((BE.shape[0],), dtype=int)
409         i = 0
410         for ed in BE:
411             ed.sort()
412             id0 = numpy.where(EdgeList[:, 0] == ed[0])[0]
413             id1 = numpy.where(EdgeList[:, 1] == ed[1])[0]
414             id = numpy.intersect1d(id0, id1)
415             if len(id) == 1:
416                 id = id[0]
417                 BEdgeList[i] = id
418                 i += 1
419         BEdgeFlag = numpy.zeros((Nedges,), dtype=bool)
420         BEdgeFlag[BEdgeList] = True
421         ############################################################
422
```

```
        marked_edges = numpy.zeros((Nedges,), dtype=bool)
        marked_edges[ElementToEdge[marked_elements, :].ravel()] = True

        # mark 3-2-1 triangles
        nsplit = len(numpy.where(marked_edges is True)[0])
        edge_num = marked_edges[ElementToEdge].sum(axis=1)
        edges3 = numpy.where(edge_num >= 2)[0]
        marked_edges[ElementToEdge[edges3, :]] = True  # marked 3rd edge
        nsplit = len(numpy.where(marked_edges is True)[0]) - nsplit

        edges1 = numpy.where(edge_num == 1)[0]
        # edges1 = edge_num[id]              # all 2 or 3 edge elements

        # new nodes (only edges3 elements)

        x_new = 0.5*(self.Verts[EdgeList[marked_edges, 0], 0]) \
            + 0.5*(self.Verts[EdgeList[marked_edges, 1], 0])
        y_new = 0.5*(self.Verts[EdgeList[marked_edges, 0], 1]) \
            + 0.5*(self.Verts[EdgeList[marked_edges, 1], 1])
        z_new = 0.5*(self.Verts[EdgeList[marked_edges, 0], 2]) \
            + 0.5*(self.Verts[EdgeList[marked_edges, 1], 2])

        Verts_new = numpy.vstack((x_new, y_new, z_new)).T
        self.Verts = numpy.vstack((self.Verts, Verts_new))
        # indices of the new nodes
        new_id = numpy.zeros((Nedges,), dtype=int)
        new_id[marked_edges] = Nv + numpy.arange(0, nsplit)
        # New tri's in the case of refining 3 edges
        # example, 1 element
        #                 n2
        #                / |
        #              /   |
        #            /     |
        #         n5-------n4
        #       / \       /|
        #     /    \     / |
        #   /       \  /  |
        # n0 --------n3-- n1
        ids = numpy.ones((Nel,), dtype=bool)
        ids[edges3] = False
        ids[edges1] = False

        E_new = numpy.delete(E, marked_elements, axis=0)  # E[id2, :]
        n0 = E[edges3, 0]
        n1 = E[edges3, 1]
        n2 = E[edges3, 2]
        n3 = new_id[ElementToEdge[edges3, 0]].ravel()
        n4 = new_id[ElementToEdge[edges3, 1]].ravel()
        n5 = new_id[ElementToEdge[edges3, 2]].ravel()
```

```
472
473         t1 = numpy.vstack((n0, n3, n5)).T
474         t2 = numpy.vstack((n3, n1, n4)).T
475         t3 = numpy.vstack((n4, n2, n5)).T
476         t4 = numpy.vstack((n3, n4, n5)).T
477
478         E_new = numpy.vstack((E_new, t1, t2, t3, t4))
479         self.Elmts[2] = (0, E_new)
480
481     def smooth2dtri(self, maxit=10, tol=0.01):
482         edge0 = self.Elmts[2][1][:, [0, 0, 1, 1, 2, 2]].ravel()
483         edge1 = self.Elmts[2][1][:, [1, 2, 0, 2, 0, 1]].ravel()
484         nedges = edge0.shape[0]
485         data = numpy.ones((nedges,), dtype=int)
486         # S = sparse(mesh.tri(:, [1, 1, 2, 2, 3, 3]),
487         # mesh.tri(:, [2, 3, 1, 3, 1, 2]), 1, mesh.n, mesh.n);
488         S = coo_matrix((data, (edge0, edge1)), shape=(self.Verts.shape[0]
                                                       ,
489                        self.Verts.shape[0])).tocsr().tocoo()
490         S0 = S.copy()
491         S.data = 0*S.data + 1
492
493         W = S.sum(axis=1).ravel()
494
495         L = (self.Verts[edge0, 0] - self.Verts[edge1, 0])**2 + \
496             (self.Verts[edge0, 1] - self.Verts[edge1, 1])**2
497
498         L_to_low = numpy.where(L < 1e-14)[0]
499         L[L_to_low] = 1e-14
500
501         # find the boundary nodes for this mesh (does not support a one-
                                                    element
502         # whole)
503         bid = numpy.where(S0.data == 1)[0]
504         bid = numpy.unique(S0.row[bid])
505         self.bid = bid
506
507         for iter in range(0, maxit):
508             x_new = numpy.array(S*self.Verts[:, 0] / W).ravel()
509             y_new = numpy.array(S*self.Verts[:, 1] / W).ravel()
510             x_new[bid] = self.Verts[bid, 0]
511             y_new[bid] = self.Verts[bid, 1]
512             self.Verts[:, 0] = x_new
513             self.Verts[:, 1] = y_new
514             L_new = (self.Verts[edge0, 0] - self.Verts[edge1, 0])**2 + \
515                     (self.Verts[edge0, 1] - self.Verts[edge1, 1])**2
516             L_to_low = numpy.where(L < 1e-14)[0]
517             L_new[L_to_low] = 1e-14
518
```

```
            move = max(abs((L_new-L) / L_new))   # inf norm

            if move < tol:
                return
            L = L_new

if __name__ == '__main__':

    # meshname = 'test.msh'
    meshname = 'bagel.msh'
    mesh = Mesh()
    mesh.read_msh(meshname)
    mesh.refine2dtri()
    mesh.refine2dtri()
    mesh.refine2dtri()
    mesh.smooth2dtri()
    print(mesh.Elmts[2][1].shape)

    import trimesh
    trimesh.trimesh(mesh.Verts[:, :2], mesh.Elmts[2][1])
    import matplotlib.pyplot as plt
    plt.plot(mesh.Verts[mesh.bid, 0], mesh.Verts[mesh.bid, 1], 'ro')
    if 1:
        import trimesh
        trimesh.trimesh(mesh.Verts[:, :2], mesh.Elmts[2][1])

        mesh.refine2dtri([0, 3, 5])
        trimesh.trimesh(mesh.Verts[:, :2], mesh.Elmts[2][1])

        mesh.refine2dtri()
        mesh.refine2dtri()
        mesh.refine2dtri()
        trimesh.trimesh(mesh.Verts[:, :2], mesh.Elmts[2][1])
        print(mesh.Elmts[2][1].shape)
```

I.4 Função "Boundary" - Python

```
"""
@author: Buzogany
"""

import numpy as np

def elem(surf_bc, surf, elem):

    elem_bc = np.zeros((len(elem),2))
    tipocdc = np.zeros((len(elem),3),dtype=int)
```

```
    tipocdc[:,:]=2
    valorcdc = np.zeros((len(elem),3))

    #all elements with forca 0
    elem_bc[:,0] = 1
    for bc_surf,  bc_value  in surf_bc.items():
        ix = np.where(surf == bc_surf)[0]
        print('[Boundary]Valor de ix: %d \n',ix)
        tipocdc[ix,0]=surf_bc[bc_surf][0][0]
        tipocdc[ix,1]=surf_bc[bc_surf][1][0]
        tipocdc[ix,2]=surf_bc[bc_surf][2][0]

        valorcdc[ix,0]=surf_bc[bc_surf][0][1]
        valorcdc[ix,1]=surf_bc[bc_surf][1][1]
        valorcdc[ix,2]=surf_bc[bc_surf][2][1]

    return elem_bc, tipocdc, valorcdc
```

I.5 Função "Cria arquivo de texto" - Python

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 29 17:48:18 2017

@author: Buzogany
"""

def cria_inputFMM(ELEM,NOS,cdc,titulo,tipocdc,valorcdc):
    nelem = len(ELEM)
    nnos = len(NOS)
    npint=0
    alfa=1
    beta=-1
    SolverType=1  # 1=FMM Solver, 2=ACA Solver, 3=Direct Solver
    N_s_loading=0 # Number of remote stress loading
    E=1
    Nu=0.0
    arq = open("input.dat", "w")
    arq.write(titulo)
    arq.write('\n    %s\n    %s   %s   %s \n    %s\n    %s  %d\n' % (
                                        SolverType,nelem,nnos,npint,
                                        N_s_loading,E,Nu))

    arq.write('%10.6e    %10.6e \n $ Nodes:\n' % (alfa,beta))

    for i in range(0,nnos):
        x=NOS[i,0]
```

```
        y=NOS[i,1]
        z=NOS[i,2]

    arq.write(" $ Elements and Boundary Conditions (Elem No., 3 Node
                                        Numbers, 3 BC Types (1=disp
                                        given; 2=traction given), and 3
                                        BC Values, in the x, y and z
                                        directions):\n")

    for i in range(0,nelem):
        no1=ELEM[i,0]
        no2=ELEM[i,1]
        no3=ELEM[i,2]


        arq.write(' %4d     %4d   %4d  %4d      %2d   %2d  %2d       %5.
                                            3e    %5.3e   %5.3e  \n' % (
                                            i, no1,no2,no3,tipocdc[i,0],
                                            tipocdc[i,1],tipocdc[i,2],
                                            valorcdc[i,0], valorcdc[i,1]
                                            ,valorcdc[i,2]))

    arq.close()
```

I.6 Função "RunFMM" - Python

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 18 13:57:37 2017

@author: Buzogany
"""

import le_resultados
import psutil
import os
import subprocess
import time

def elastostatics():

    cmd = r"C:\Users\Buzogany\OneDrive\UNB\PROJETO DE GRADUACAO 1\Python\
                                        bem3D\bem3D\RIM3D\
                                        FastBEM_Elastostatics_64.exe"
    file_path_output = r"C:\Users\Buzogany\OneDrive\UNB\PROJETO DE
                                        GRADUACAO 1\Python\bem3D\bem3D\
                                        RIM3D\output.dat"
```

```
    outdate=os.path.getmtime(file_path_output)

    subprocess.Popen(cmd)

    date=os.path.getmtime(file_path_output)

    while (date==outdate):
        time.sleep(1)
        date=os.path.getmtime(file_path_output)

    if os.path.isfile(file_path_output) and (date>outdate):
        # read file
        resultado,dados = le_resultados.output()
    else:
        raise ValueError("%s isn't a file!" % file_path_output)


    PROCNAME = "FastBEM_Elastostatics_64.exe"

    for proc in psutil.process_iter():
        # check whether the process name matches
        if proc.name() == PROCNAME:
            proc.kill()
            print("[runFMM] KILLING PROCESS")

    return resultado,dados
```

I.7 Função "Lê Resultados" - Python

```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 12 13:47:18 2017
@author: Buzogany
"""

import numpy as np

def output():

    #Copy Files to Keep records and output specific values for important
                                            point
    print("[le_resultados] Lendo os resultados\n")
    dados = np.genfromtxt('output.dat',dtype='float',skip_header=2)

    Node=dados[:,0]
    u=dados[:,1]
    v=dados[:,2]
```

```
    w=dados[:,3]

    t_x=dados[:,4]
    t_y=dados[:,5]
    t_z=dados[:,6]

    dt=np.sqrt(u*u+v*v+w*w)

    return dt,dados
```

I.8 Função "Mostra Resultados" - Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 30 15:17:11 2017
@author: eder
@modificated: Buzogany
"""
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

def show_results1(tri,XYZ,T):
    x=np.zeros(3)
    y=np.zeros(3)
    z=np.zeros(3)
    zc=np.zeros(len(tri))
    pc=[]
    for elem in range(len(tri)):
        no1=tri[elem,0]
        no2=tri[elem,1]
        no3=tri[elem,2]
        x[0]=XYZ[no1,0]
        y[0]=XYZ[no1,1]
        z[0]=XYZ[no1,2]
        x[1]=XYZ[no2,0]
        y[1]=XYZ[no2,1]
        z[1]=XYZ[no2,2]
        x[2]=XYZ[no3,0]
        y[2]=XYZ[no3,1]
        z[2]=XYZ[no3,2]
        pc.append([[x[0],y[0],z[0]],[x[1],y[1],z[1]],[x[2],y[2],z[2]]])
        zc[elem]=T[elem]
    fig = plt.figure()
```

```
    fig.set_size_inches
    ax = plt.axes(projection='3d')
    m = cm.ScalarMappable(cmap=cm.jet)
    b=m.to_rgba(zc)
    vetcor=[(i[0],i[1],i[2]) for i in b]
    m.set_array([min(zc),max(zc)])
    m.set_clim(vmin=min(zc),vmax=max(zc))
    q = Poly3DCollection(pc, linewidths=1, edgecolors="k")
    q.set_facecolor(vetcor)
    ax.add_collection3d(q)
    fig.colorbar(m)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    ax.set_xlim(min(XYZ[:,0]),max(XYZ[:,0]))
    ax.set_ylim(min(XYZ[:,1]),max(XYZ[:,1]))
    ax.set_zlim(min(XYZ[:,2]),max(XYZ[:,2]))
    ax.view_init(elev=18., azim=43.)
    ax.set_aspect('equal')
    ax.view_init(elev=48., azim=43.)
    plt.title('Deslocamento')
    plt.show()
```

I.9 FreeCAD

O FreeCad é modelador 3D de código aberto, direcionado principalmente para engenharia mecânica e para o desenvolvimento de produto. As ferramentas disponibilizadas no FreeCad são semelhantes às do Catia, SolidWorks e Creo, portanto pode ser definido como um modelador paramétrico com arquitetura modular o que permite adicionar funcionalidades sem modificar o sistema base.

Similarmente como diversos modeladores CAD, contém diversas funcionalidades relacionadas com componentes 2D para o desenho de formas 2D ou para extração de detalhes do modelo 3D para criação de desenhos 2D, no entanto o desenho em duas dimensões direto (como o AutoCAD LT) não é um objetivo deste software assim como a animação e a modelação de formas orgânicas (como Maya, 3ds Max, Blender ou Cinema 4D). Apesar de não ter essas funcionalidades como foco principal o FreeCad pode ser útil em tarefas desta natureza.

Outro objetivo deste software é fazer uso das diversas bibliotecas de código fonte aberto disponíveis na área da computação cientifica. Entre estas bibliotecas existem o OpenCascade, uma biblioteca de CAD bastante abrangente, o Coin3D, uma implementação do OpenInventor, a Qt, famosa arquitetura de interface gráfica, e o Python, uma das mais versáteis e completas linguagens de 'scripting' existentes. O FreeCad pode ser usado também em outros programas como biblioteca.

I.9.1 Troca de arquivos de desenho entre programas

A compatibilidade entre os diversos programas utilizados para alcançar uma análise confiável, torna-se, atualmente, possível através das diferentes possibilidades de arquivos que cada programa pode gerar e interpretar. No caso deste trabalho, foram criados arquivos em CAD que deveriam ser analisados através de um programa do método dos elementos de contorno rápido que é chamado em ambiente de linguagem Python. De forma que está análise possa ser executada, foi necessário exportar o arquivo CAD no formato de Representação de Contorno ou Boundary Representation (BREP ou B-rep) com a intenção de poder importá-lo em um programa gerador de malha para discretizar o modelo contínuo.

O formato de arquivo BREP (Boundary representation) é um método utilizado para representar superfícies, formas, usando seus limites. Um sólido pode ser comumente representado por uma coleção de elemtos de superfície conectados, ou seja, o meio termo entre o sólido e um não-sólido. Os modelos BREP são compostos por duas partes: a topologia – faces, arestas e vértices – e a geometria – superfícies, curvas e pontos – respectivamente, consoante (MäNTYLä, 1988). O conjunto de faces é denominado de casca, que é a representação que será utilizada neste trabalho. Os arquivos deste formato podem ser gerados por softwares comerciais como o Solidworks e softwares livres como o Freecad (HAVRE, 2017) (usado neste projeto). O formato BREP tem como fator determinante a necessidade de apresentar

um sólido de parte interior e exterior como na Figura 1 e não funciona com superfícies não orientáveis como a fita de Möbius da Figura 2.

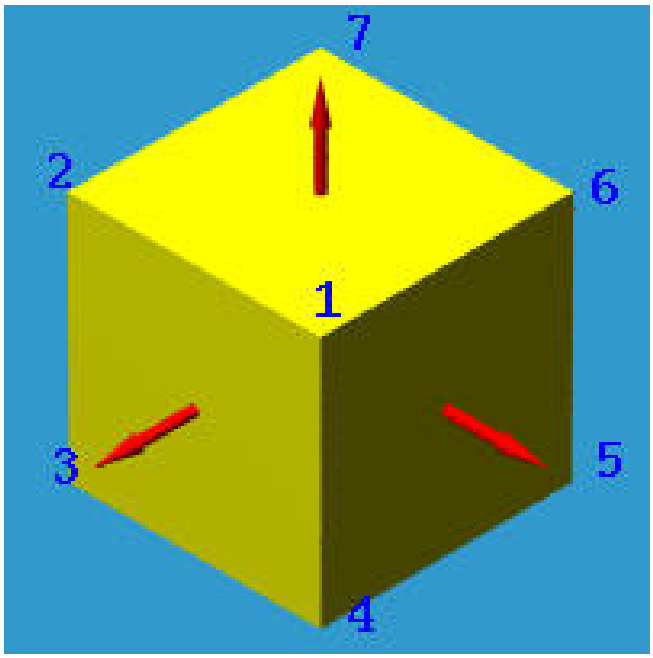

Figura 1: O cubo é um sólido orientável

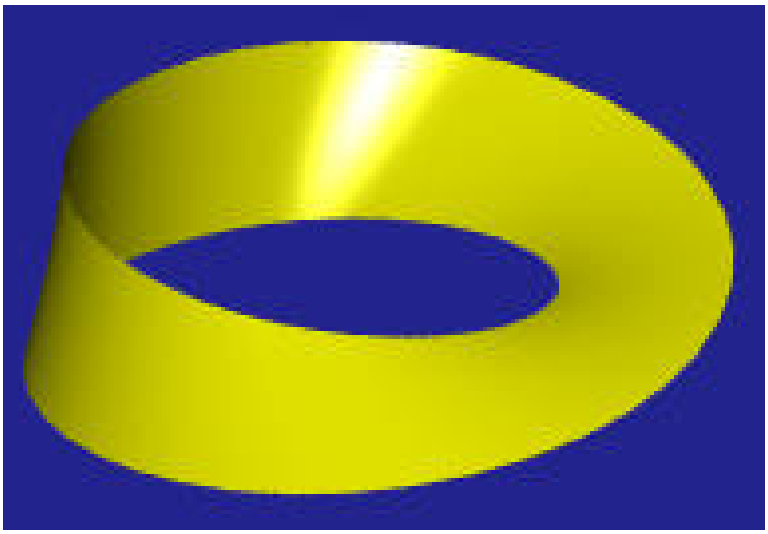

Figura 2: A fita de Möbius é um sólido não orientável

I.9.2 Tutorial

O ambiente de trabalho do FreeCad assemelha-se com o de outros programas como o AutoCAD e Solidworks, não contendo maiores dificuldades de utilização. Em seguida, será desenvolvido um breve tutorial para familiarização com o software, apresentando as funções básicas para modelagem de uma peça em 3D.

Para exemplificar como se constrói um sólido no GMSH, na sequência deste capítulo serão descritos os passos para se desenvolver um mancal (suporte de eixo).

1. Para começar o processo de desenvolvimento de uma nova peça, abra um novo arquivo, Figura 19;

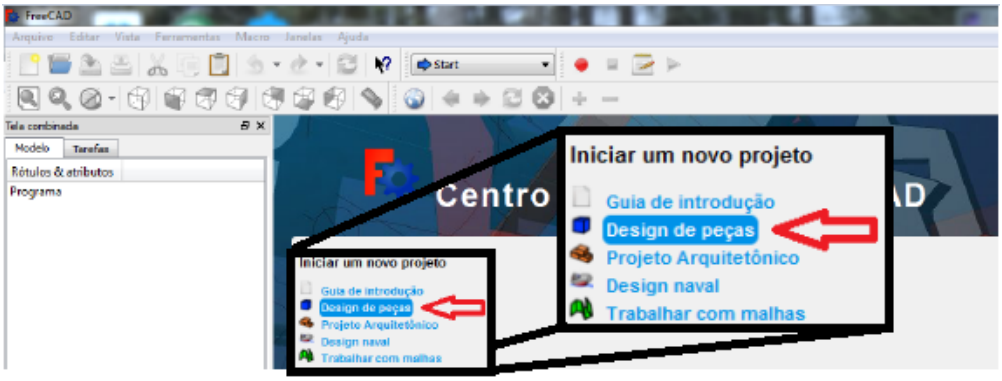

Figura 3: Criar um novo arquivo.

2. Crie um novo esboço para a peça em um plano, Figura 4;

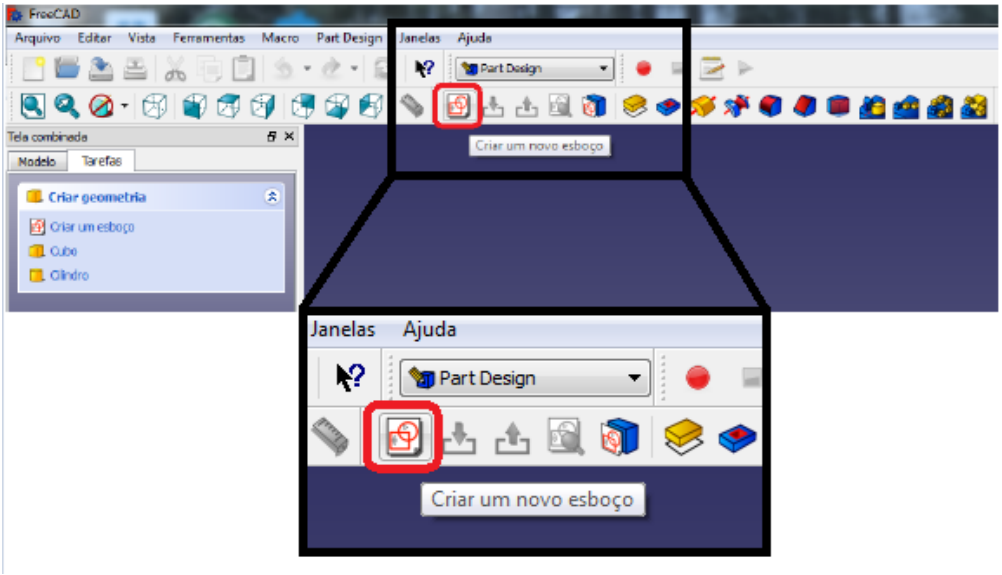

Figura 4: Criar um novo esboço.

3. A caixa vermelha demarca na Figura 5 as identidades para a criação e o desenvolvimento do esboço com as diferentes geometrias possíveis;

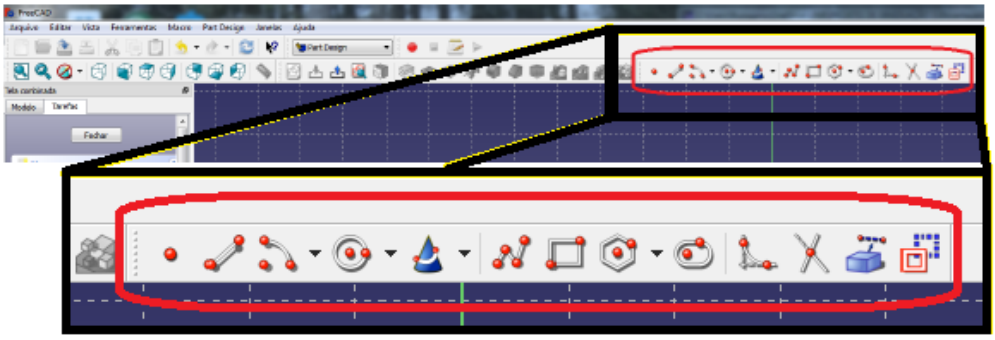

Figura 5: Identidades para criar o esboço.

4. Além das identidades para criação do esboço também usa-se comandos que aplicam restrições e relações entre os componentes com as opções marcadas na Figura 6;

Figura 6: Identidades para aplicar restrições.

5. Depois de desenvolvido um esboço, cria-se o modelo 3D com as diversas opções (extrudar, furar e chanfrar, por exemplo) demarcadas na Figura 7 para cada caso;

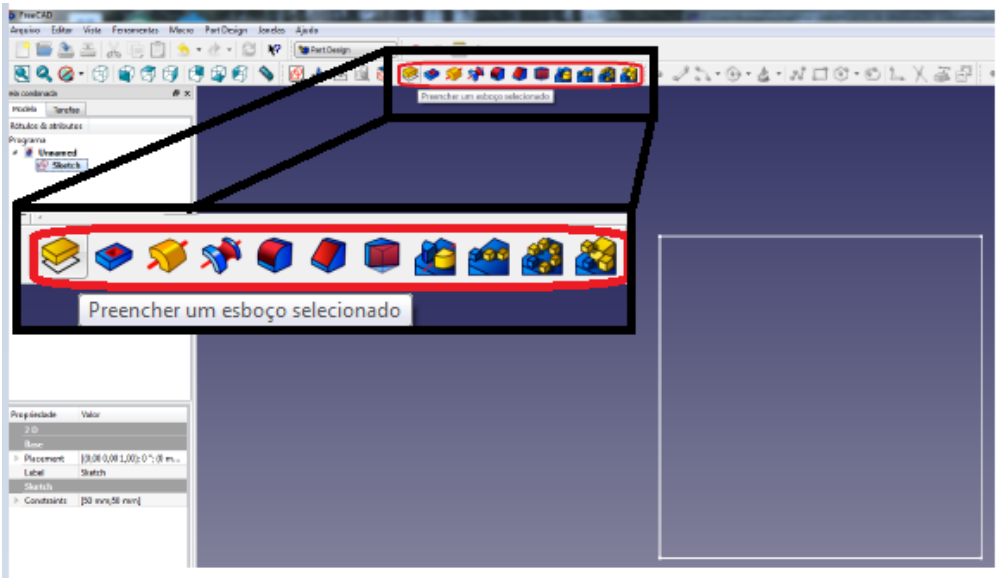

Figura 7: Funções para criar elementos no plano tridimensional.

6. Para esta peça será utilizado o método da extrusão selecionando o esboço e definindo a dimensão da largura como na Figura 8;

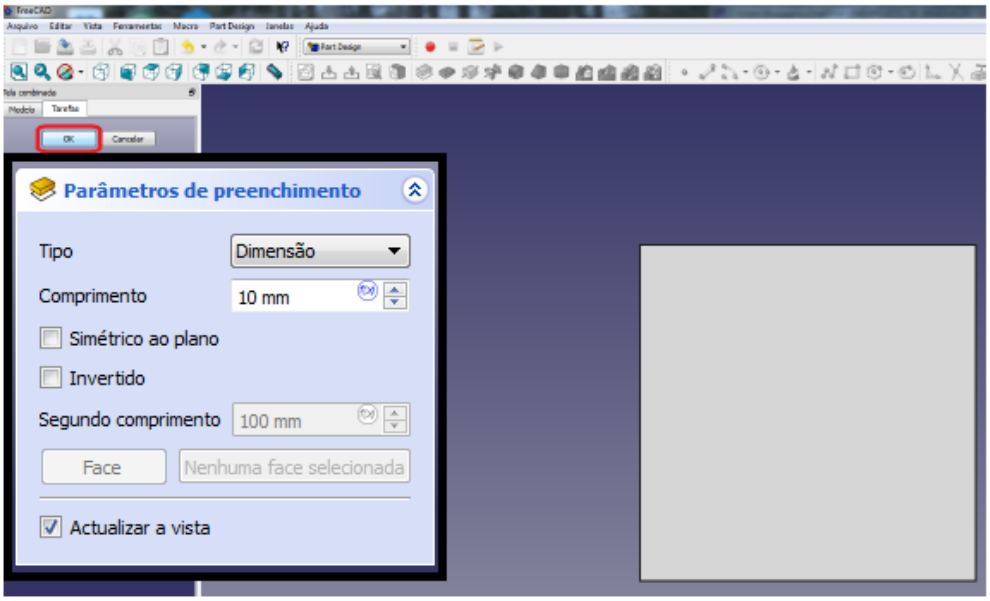

Figura 8: Criar extrusão a partir do esboço.

7. Os esboços não precisam ser desenhados somente sobre os planos iniciais, ou seja, os planos XY, XZ e YZ, sendo possível selecionar as faces sobre as quais serão adicionados os elementos como na Figura 9;

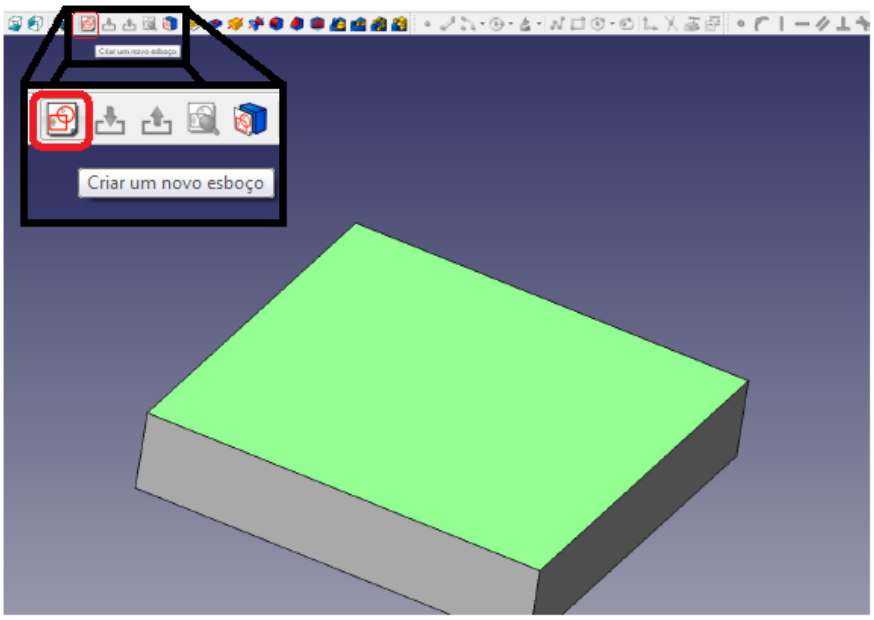

Figura 9: Selecione a face para desenvolver um novo esboço.

8. Segue-se os mesmos procedimentos anteriores para criar geometrias fechadas como na Figura 10;

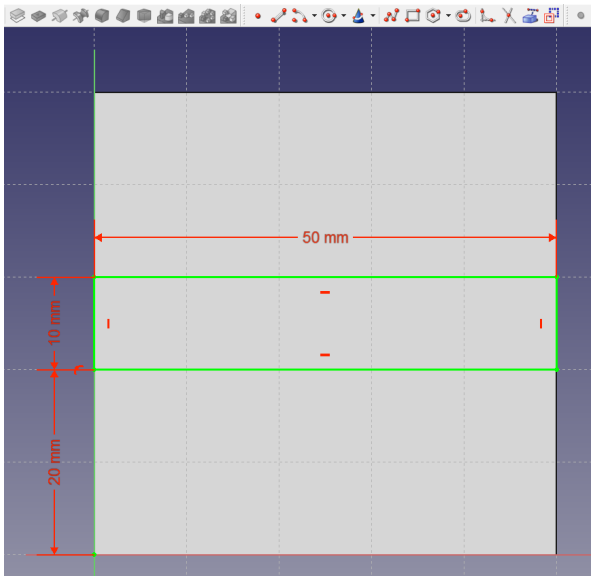

Figura 10: Progrida no esboço.

9. A criação de formas geométricas as vezes depende da selção diferenciada dos componentes, no caso dos chanfro e filetes é necessário selecionar a aresta que une as duas faces como na Figura 11 ou selecionar as duas faces;

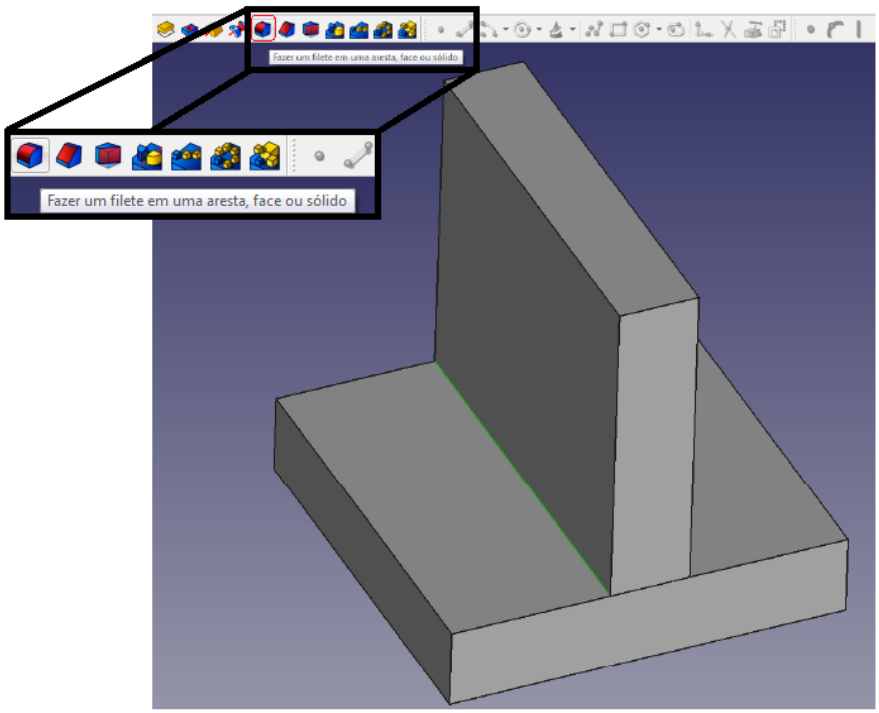

Figura 11: Para criação de chanfros/filetes selecione a aresta entre as faces.

10. Depois de selecionada a ferramenta, indica-se o valor do raio do filete como na Figura 12;

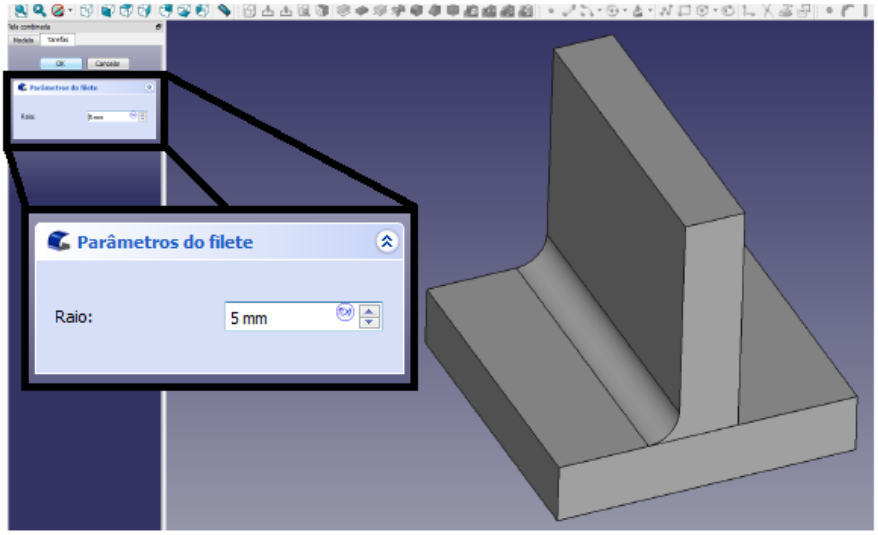

Figura 12: Indique o raio do filete.

11. Pode-se também selecionar várias arestas e aplicar o filete de mesmo raio em todas elas como na Figura 13;

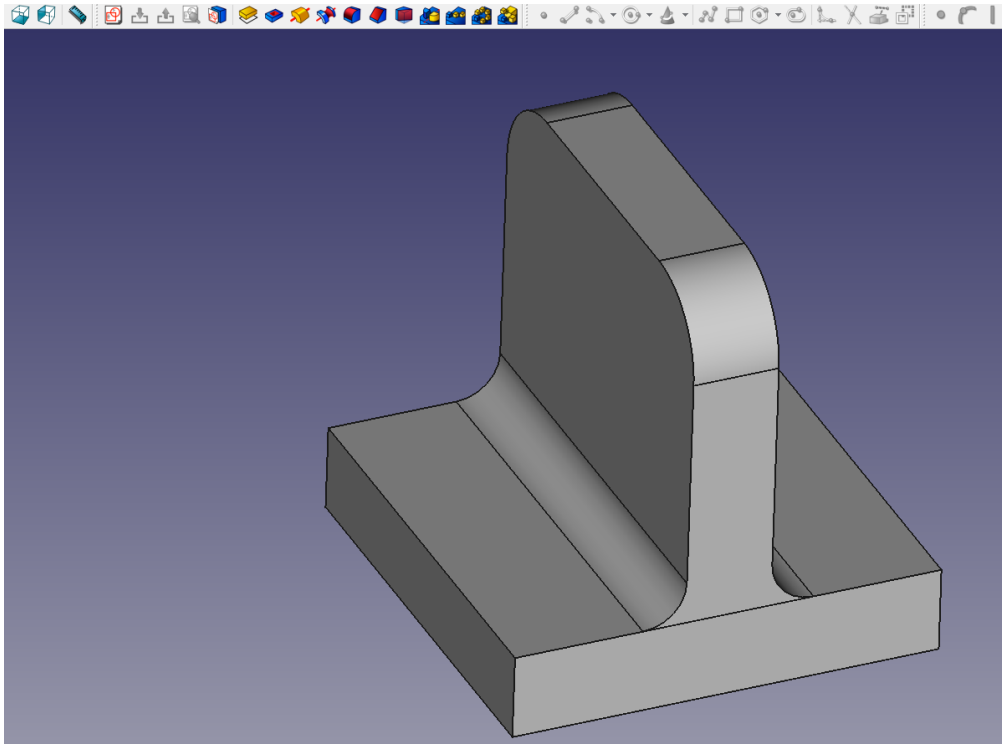

Figura 13: Repita o procedimento para cada aresta ou selecione todas simultaneamente.

12. Para criar um furo sobre uma face, é necessário selecioná-la e criar um novo esboço como na Figura 14;

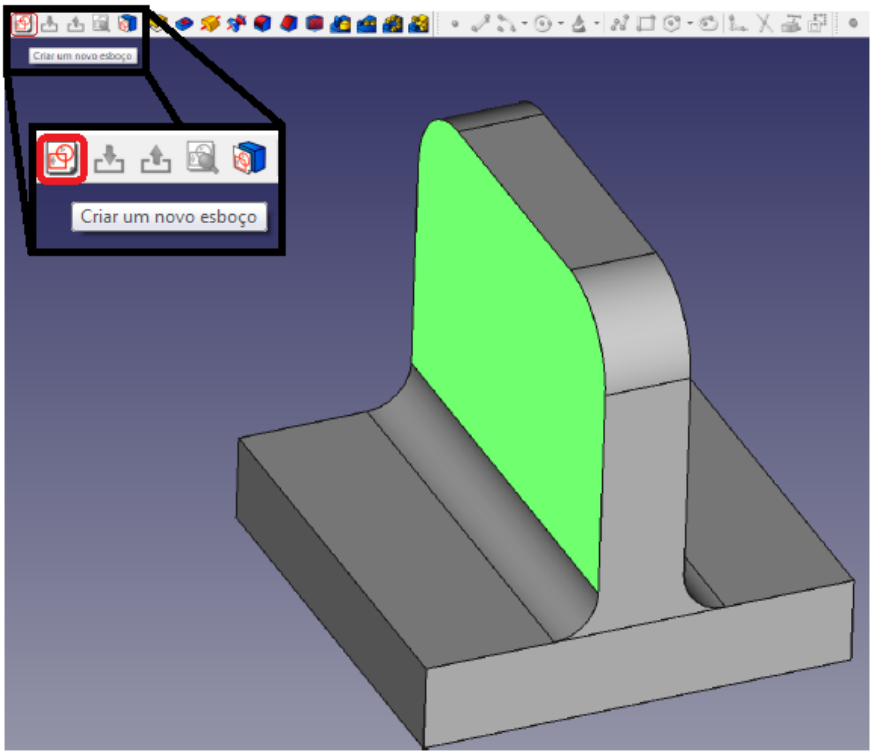

Figura 14: Selecione a face na qual deseja fazer um furo.

13. Designa-se as relações e limitações das geometrias do furo como na Figura 15;

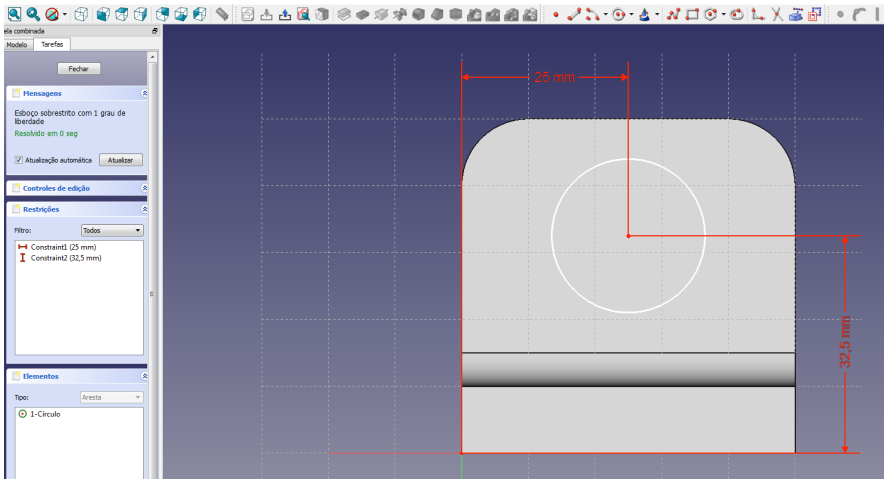

Figura 15: Desenvolva o esboço do furo.

14. Depois de finalizar o esboço, seleciona-se ele e escolhe-se a ferramenta para criação do furo como na Figura 16;

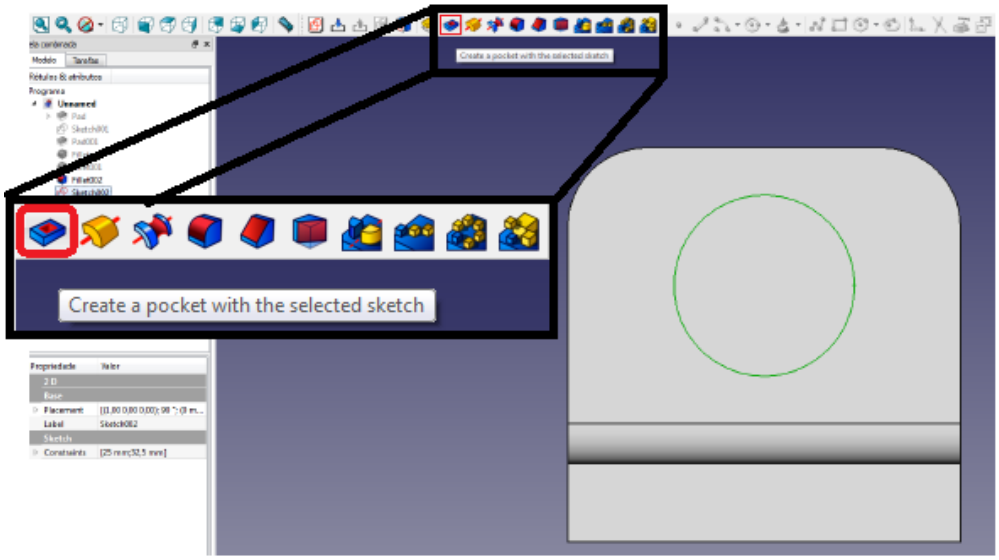

Figura 16: Selecione a ferramente para criar o furo.

15. Existem diversos tipos de furos, nos quais se especifica a direção e profundidade como na Figura 17;

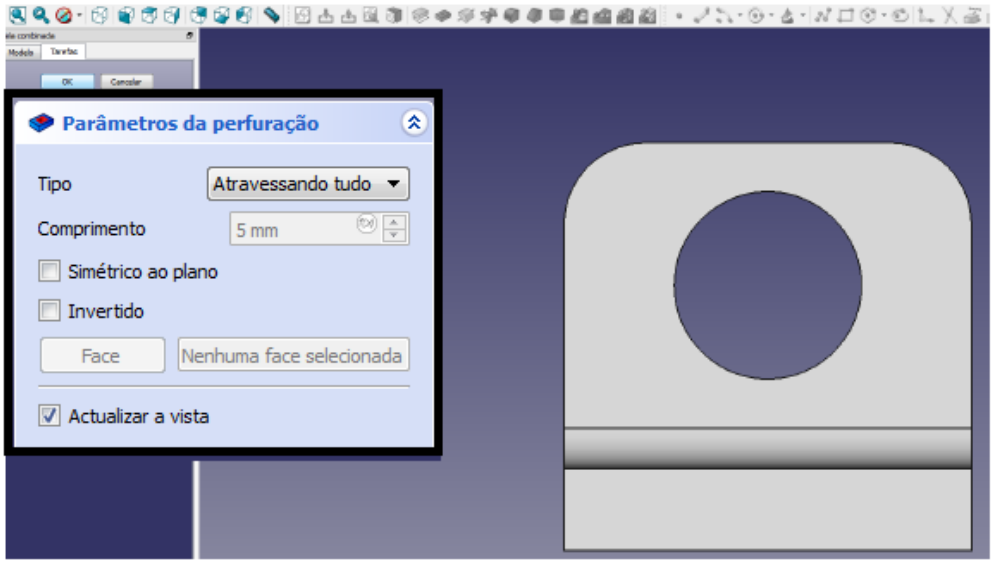

Figura 17: Especifique o tipo de furo e suas dimensões.

16. Assim alcança-se o resultado final da peça para os fins deste trabalho, apresentada na Figura 18;

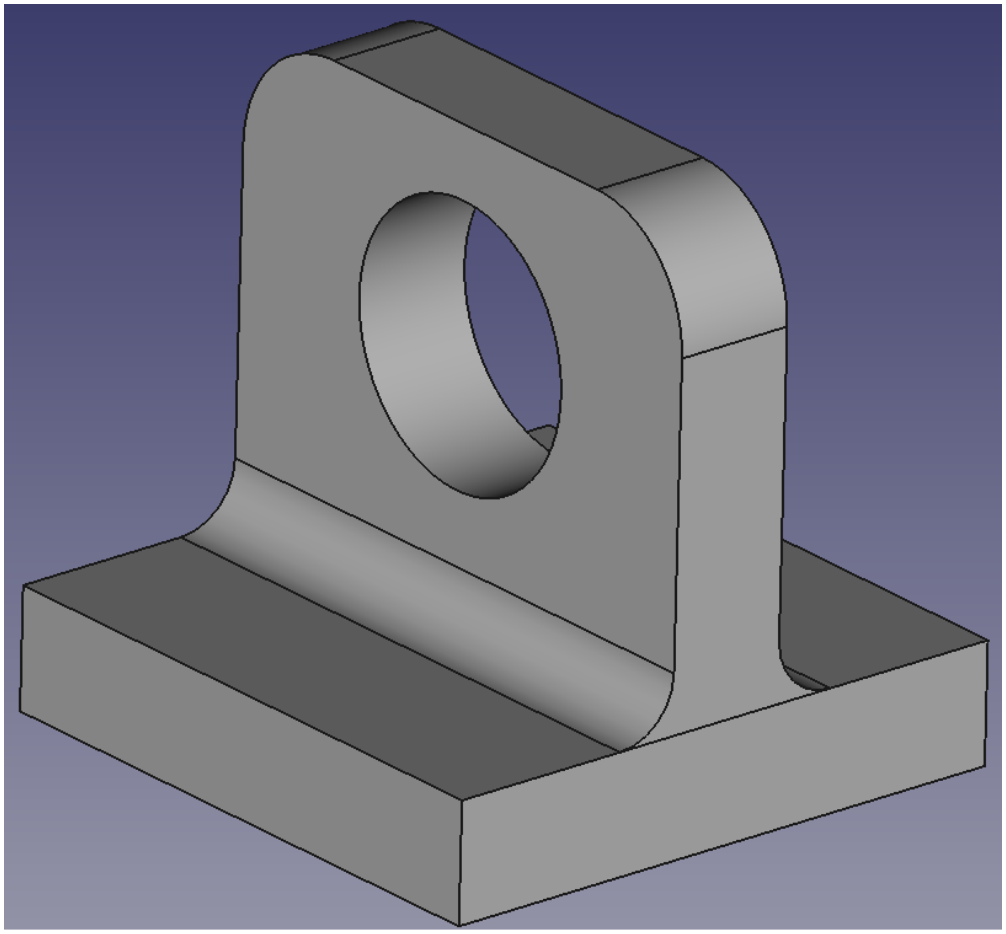

Figura 18: Peça final.

Este é somente um passo a passo simples com a finalidade de exemplificar as funções básicas do programa, mas pode-se abstrair a compreensibilidade da execução das tarefas do FreeCad, sendo, como mencionado anteriormente, bastante similar com outros softwares de modelagem 3D.

I.10 GMSH

O GMSH é um gerador de malhas de elementos finitos 3D com facilidades de pré e pós processamento integradas. O objetivo do software está em oferecer uma ferramenta geradora de malhas rápida, que demanda poucos recursos computacionais e intuitiva com entradas paramétricas e avançadas possibilidades de visualização. A arquitetura do GMSH foi desenvolvida em torno de quatro módulos principais: geometria, malha, processamento e pós-processamento. As entradas para análise podem ser feitas a partir da interface gráfica ou pelo padrão ASCII de texto com a linguagem de programação própria do GMSH.

I.10.1 Tutorial

A fim de exemplificar o funcionamento do GMSH, desenvolveu-se o seguinte tutorial ilustrativo.

1. Abra um novo documento/sólido no qual se quer gerar uma malha. Para isso, aperte em “File” e depois em “Open”, ou, simplesmente, use o atalho CTRL + O., Figura 19. O GMSH é um software bastante versátil, aceitando peças de diferentes formatos e extensões (.brep, .igs, .step, etc). As diferentes formas de extensões são interpretadas de formas especificas pelo programa e dependendo da geometria da peça é possível que algumas extensões tenham vantagens sobre as outras. Para fim deste tutorial, foi usada uma peça com extensão OpenCASCADE BRep (.brep);

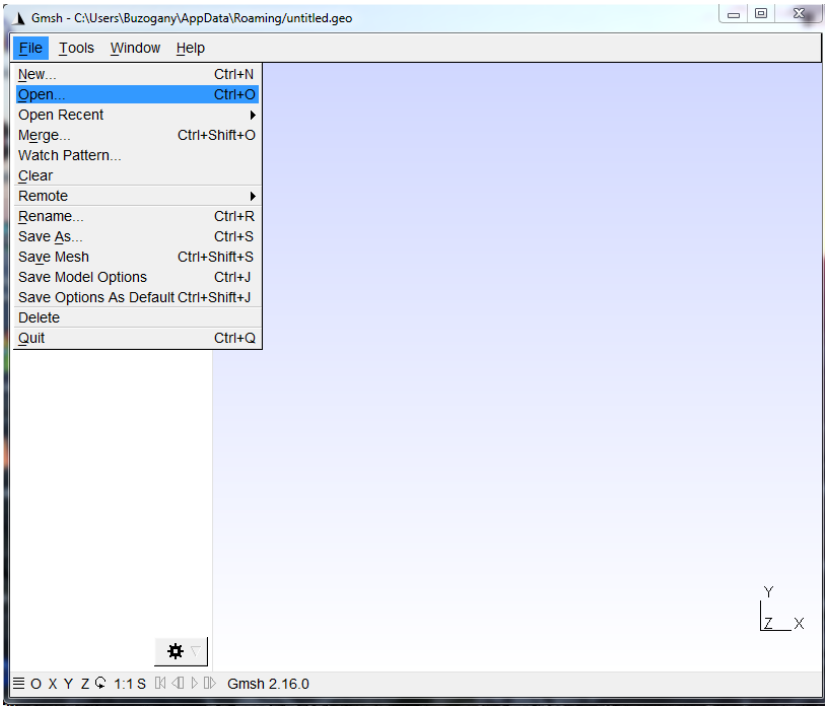

Figura 19: Abrir um novo arquivo.

2. Selecione um arquivo de extensão compatível e aperte em “Open”, Figura 20;

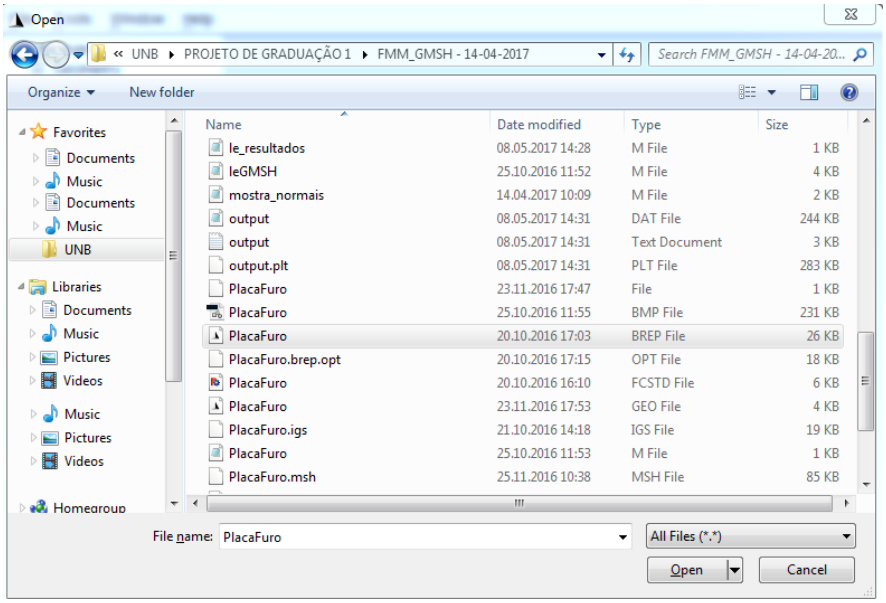

Figura 20: Selecionar o arquivo com a extensão suportada.

3. O programa irá carregar o arquivo escolhido e então aperte no símbolo ‘+’ , Figura 21;

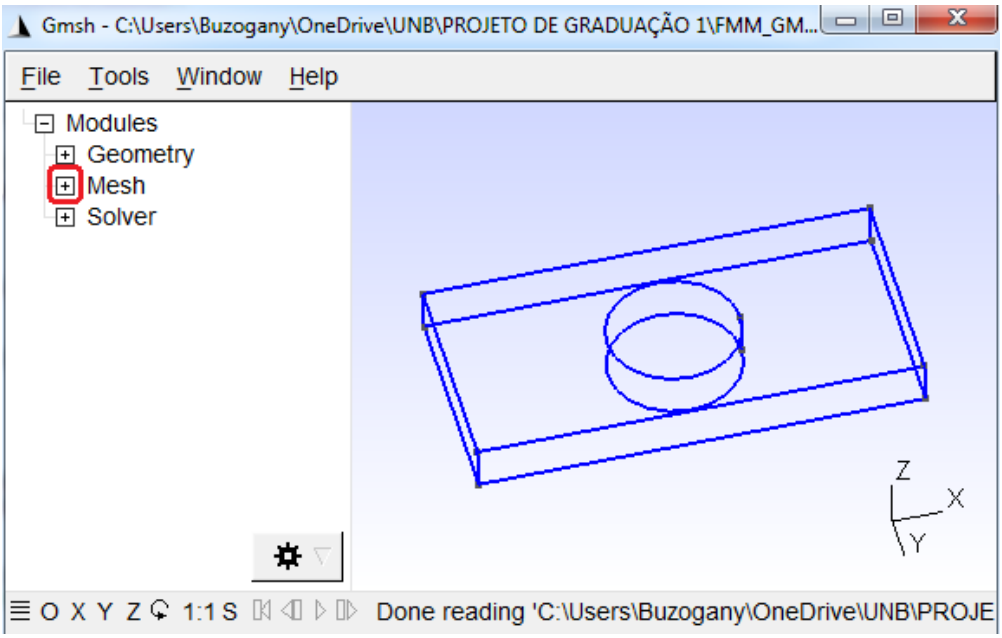

Figura 21: Menu de opções.

4. Dentre as varias opções para gerar malhas, escolheremos neste caso a geração de malha 2D na superfície do sólido. Basta um clique na opção desejada para que seja gerada a malha., Figura 22;

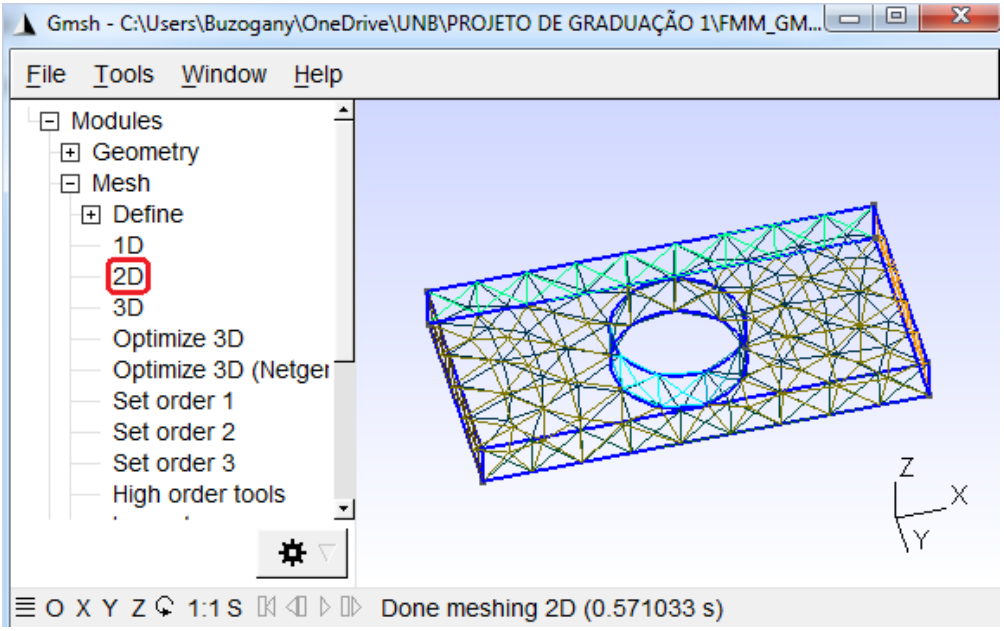

Figura 22: Gerar malha 2D sobre o sólido.

5. Para gerar uma malha que se molda melhor à superfície do sólido, as vezes é necessário refinar a malha inicial. Para isso, aperte sobre a opção "Refine by spliting", Figura 23;

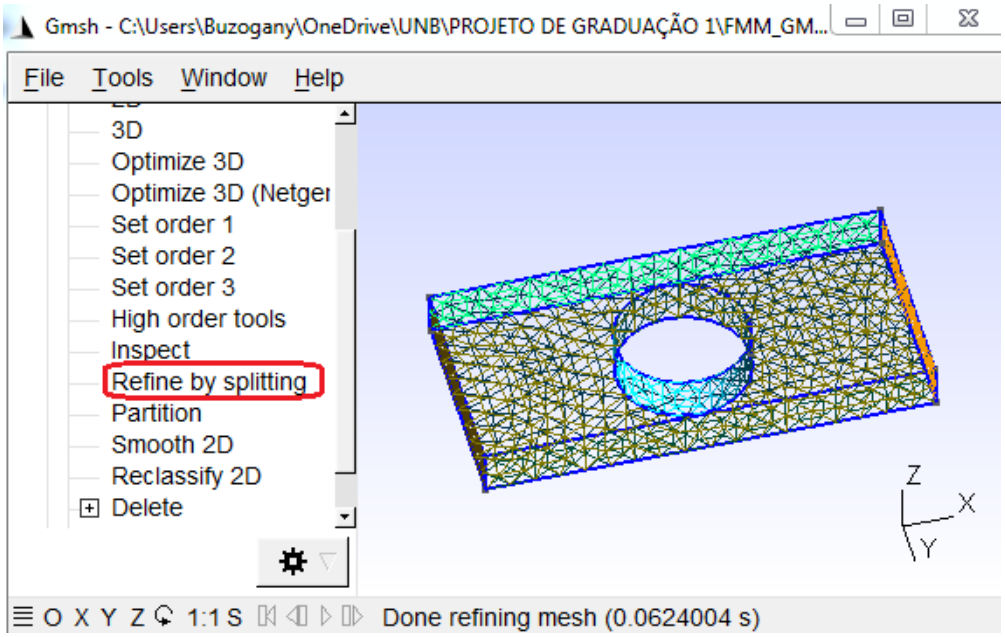

Figura 23: Refinar a malha.

6. Assim finaliza-se o procedimento de criação da malha, basta salvá-la como na Figura 24;

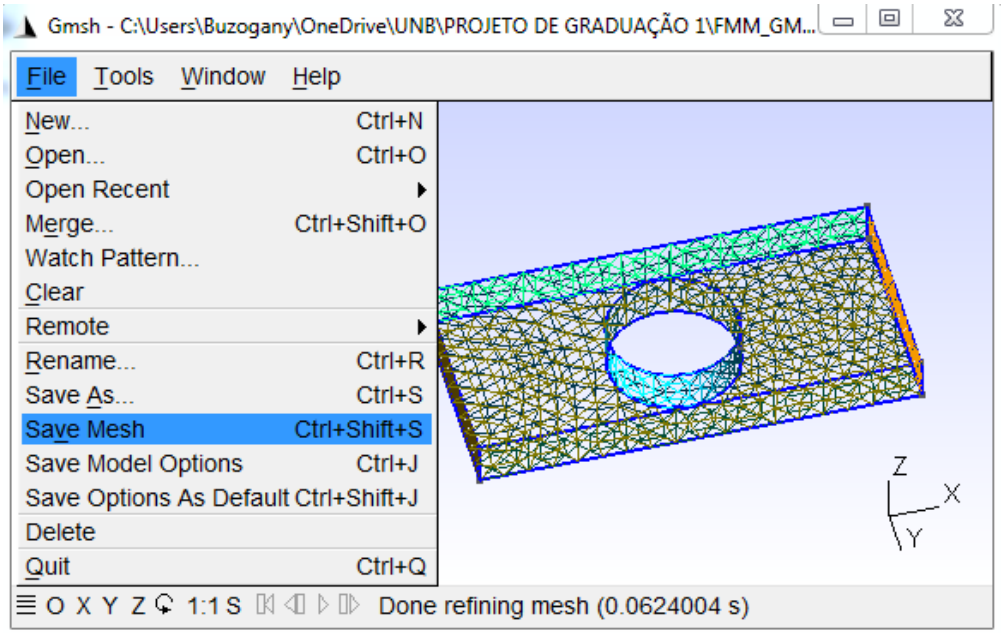

Figura 24: Salvar a malha.

7. O próximo passo será definir quais serão as superfícies sobre as quais aplicaremos a condição de contorno. Para visualizar a enumeração das superfícies do modelo, clica-se sobre a opção "Tools" e em seguida na opção "Options" visto na Figura 25.

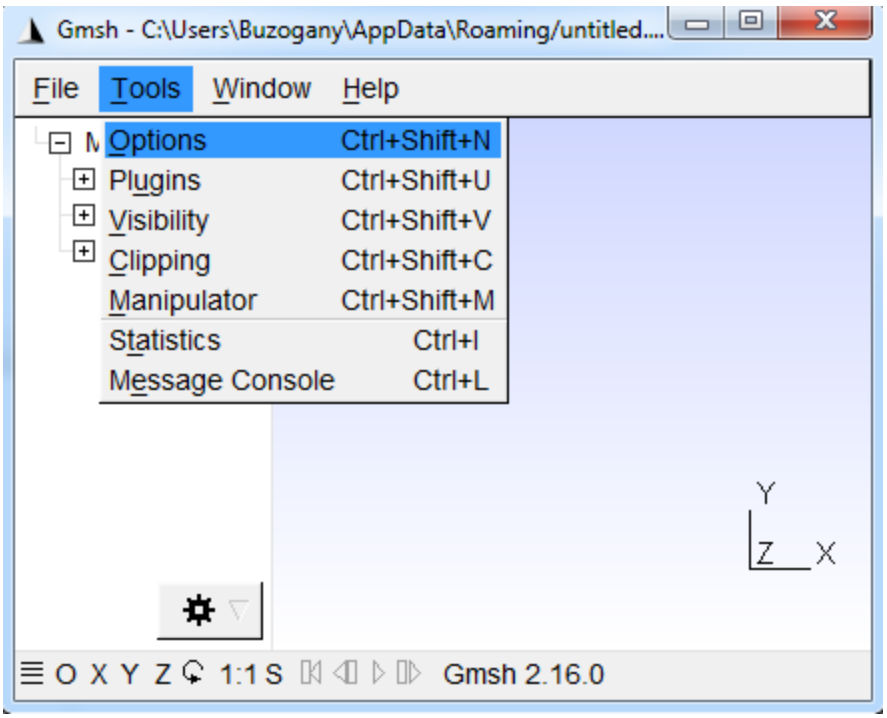

Figura 25: Visualizar enumeração das superfícies.

8. Então para ativar a visualização das superfícies clica-se em "Geometry" no menu esquerdo e então na aba "Visibility". Deve se marcar a caixa "Surface labels" com um "check" como na Figura 26.

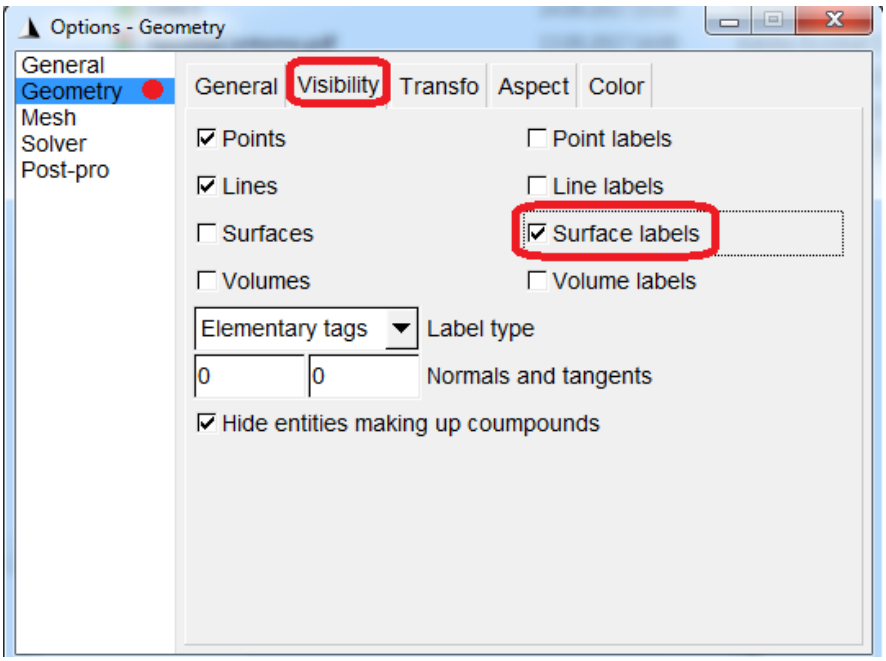

Figura 26: Ativar visualização.

9. Depois de fechado o menu de opções, nota se que a enumeração das faces agora está presente na Figura 27, sendo necessário rotacionar a peça para ver toda enumeração.

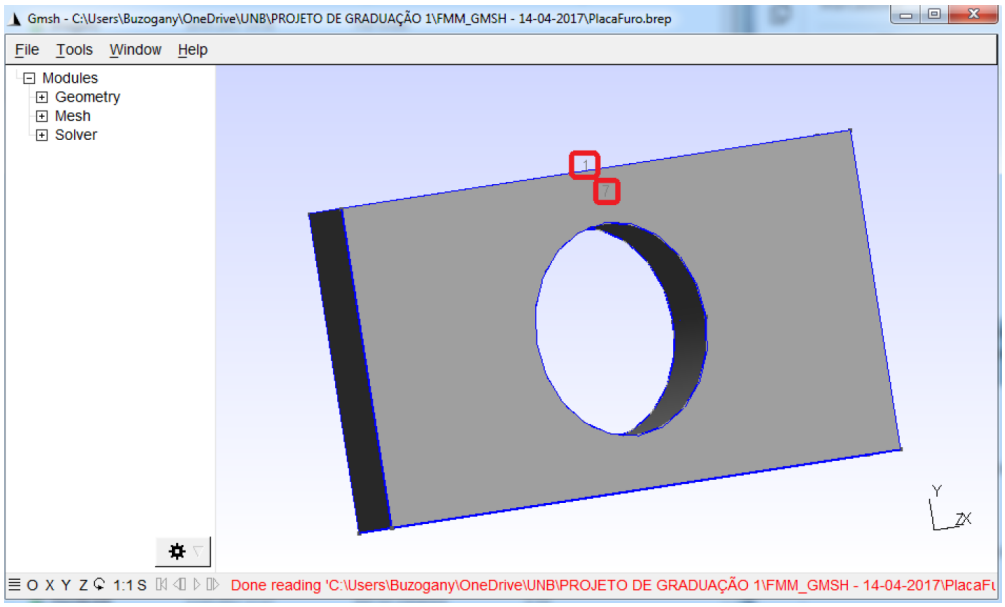

Figura 27: Enumeração das superfícies.

Referências Bibliográficas

ALBUQUERQUE Éder Lima de. *Introdução ao método dos elementos de contorno*. [S.l.], 2017.

BRAGA, L. M. *O Método dos Elementos de Contorno Rápido com Expansão em Multipólos Aplicado a Problemas de Condução de Calor*. [S.l.: s.n.], 2012.

CALLAHAN, P. B.; KOSARAJU, S. R. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, v. 42, p. 67–90, 1995.

CAMPOS, L. S. *Método dos Elementos de Contorno Isogeométricos Rápido*. Tese (Doutorado) — Universidade de Brasília, Brasília, 2016.

GAUL, L.; KöGL, M.; WAGNER, M. *Boundary Element Methods for Engineers and Scientists - An Introductory Course with Advanced Topics*. [S.l.]: Springer, 2003.

GUMEROV, R. D. N. A. *Fast Multipole Methods for the Helmholtz Equation in Three Dimensions*. [S.l.]: ELSEVIER, 2004.

HAVRE, Y. van. *FreeCad*. 2017. http://freecadweb.org/. [Online; acessado em 29-Janeiro-2017].

KANE, J. H. Boundary element analysis in engineering continuum mechanics. Prentice Hall, 1994.

LIU, Y. *Fast Multipole Boundary Element Method - Theory and applications in Engineering*. [S.l.]: Cambridge University Press, 2009.

LOYOLA, F. M. de. *Modelagem tridimensional de problemas potenciais usando o método dos elementos de contorno*. [S.l.], 2017.

MäNTYLä, M. *An Introduction to Solid Modeling*. [S.l.]: Computer Science Press, 1988.