MÉTODOS NUMÉRICOS PARA O CÁLCULO DE FORÇAS DE TENSÃO SUPERFICIAL E MODELAGEM DA INTERFACE EM ESCOAMENTOS BIFÁSICOS

Leonardo Voltarelli Regini de Andrade

Projeto de Graduação apresentado ao Curso de Engenharia Mecânica da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Gustavo Rabello dos Anjos

Rio de Janeiro

Novembro de 2021

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Departamento de Engenharia Mecânica
DEM/POLI/UFRJ

MÉTODOS NUMÉRICOS PARA O CÁLCULO DE FORÇAS DE TENSÃO SUPERFICIAL E MODELAGEM DA INTERFACE EM ESCOAMENTOS BIFÁSICOS

Leonardo Voltarelli Regini de Andrade

PROJETO FINAL SUBMETIDO AO CORPO DOCENTE DO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO MECÂNICO.

Aprovado por:

Prof. Gustavo Rabello dos Anjos, Ph.D.

Prof. Fernando Alves Rochinha, D.Sc.

Prof. Daniel Onofre de Almeida, D.Sc.

Daniel Barbedo Vasconcelos Santos, M.Sc.

RIO DE JANEIRO, RJ - BRASIL
NOVEMBRO DE 2021

Voltarelli Regini de Andrade, Leonardo

Métodos Numéricos para o Cálculo de Forças de Tensão Superficial e Modelagem da Interface em Escoamentos Bifásicos/ Leonardo Voltarelli Regini de Andrade. – Rio de Janeiro: UFRJ/Escola Politécnica, 2021.

XVI, 176 p.: il.; 29, 7cm.

Orientador: Gustavo Rabello dos Anjos

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia Mecânica, 2021.

Referências Bibliográficas: p. 120 – 126.

1. Escoamentos Bifásicos. 2. Tensão Superficial. 3. Elementos Finitos. 4. Curvatura. 5. Métodos Numéricos. I. Rabello dos Anjos, Gustavo. II. Universidade Federal do Rio de Janeiro, UFRJ, Curso de Engenharia Mecânica. III. Métodos Numéricos para o Cálculo de Forças de Tensão Superficial e Modelagem da Interface em Escoamentos Bifásicos.

"It's the dawn chorus"

Thom Yorke

Agradecimentos

Escrever um agradecimento nunca é fácil para mim. É uma vontade muito grande de incluir cada um que teve importância que faz com que eu me enrole, me emocione, e dure horas pensando e repensando a escolha correta de palavras. Talvez seja tudo devido à um sentimentalismo extremo da minha parte; mas no fim vale a pena. Como vale a pena passar anos se preparando para esse momento que, para mim, é coroado com esse texto. Então vamos lá.

Gostaria, primeiramente, de agradecer aos meus pais, Sueli e Dorival, pelo apoio constante e incondicional. Sei que se cheguei até aqui, foi por muita dedicação e amor de vocês; tenho muito orgulho e sorte de tê-los comigo. Em uma frase, hoje famosa, Newton disse que se enxergou tão longe, foi por se apoiar em ombros de gigantes; e os meus gigantes são vocês.

No mesmo sentido, gostaria de agradecer à minha irmã, Maria Vittoria, por todo carinho e apoio nos momentos de dificuldade e incerteza. Tenho muita sorte de ter encontrado em você uma grande amiga e alguém com quem estarei junto para sempre; saiba que muito da força que foi colocada nesse projeto foi por apoio seu.

Gostaria, também, de agradecer à UFRJ, como uma instituição de excelência no estudo e na pesquisa científica de alto nível. Assim, também gostaria de agradecer aos diversos, mas únicos, professores que tive a oportunidade de conhecer e que muito me ensinaram. Em particular, agradeço aos professores do departamento de Engenharia Mecânica pelas experiências transmitidas.

Agradeço, em especial, ao professor Gustavo Rabello pela oportunidade de fazer o presente trabalho e, além disso, pela notável e constante dedicação no decorrer de sua realização. Ademais, agradeço pela ajuda incomparável que me deu; se hoje consigo me graduar, muito é devido à sua boa vontade e apoio. Em um momento de dificuldade pessoal, foi com seu auxílio que consegui encontrar meios de concluir

o curso.

Muitas experiências na UFRJ também me marcaram e merecem seu espaço aqui. Agradeço à Fluxo Consultoria, por ter me dado amigos e me apresentado pessoas maravilhosas. Também agradeço à todos os colegas e professores do Núcleo Interdisciplinar de Dinâmica dos Fluidos, em particular a professora Juliana Loureiro, pela oportunidade de trabalhar em projetos instigantes e diferentes.

Agradeço à minha amiga e professora, Luisa, pelos ensinamentos sobre o que é ser professor e toda a paciência e empatia que vem com isso. Toda a minha admiração e todo carinho são poucos.

Agradeço à minha namorada e melhor amiga, Yasmin Saade, pelo seu inigualável apoio e carinho nos momentos bons, mas, principalmente, naqueles em que mais precisei. Tenho muita sorte de te ter nesse caminho tortuoso e, sem dúvida, não o faria da mesma forma se não fosse por você. Obrigado por ser minha companheira.

Por fim, dedico esses últimos parágrafos aos meus amigos. Agradeço ao meu grande amigo e fiel escudeiro Marcelo Velloso - que me ajudou a enfrentar muito mais que assustadores moinhos de vento - pelas conversas reconfortantes e pela amizade incondicional. Agradeço, também, ao meu colega de anos passados, Lucas Campos, pelo carinho e pela duradoura amizade.

Em especial, agradeço aos maravilhosos colegas e incríveis amigos que criei nesses cinco anos de faculdade. Todas as nossas conversas, todos os momentos árduos e todas as celebrações que passamos serão eternamente guardadas comigo. Em particular, agradeço aos amigos Lucas Coelho, Luiz Pirrone e Rodrigo Vojta pelas discussões inacreditáveis e pelos campeonatos de xadrez frustrados, além de todos os almoços juntos, mesmo que, as vezes, em restaurantes duvidosos. Agradeço, por fim, ao Matheus Lucchesi e à Camila Borges pela longa amizade e pelos momentos engraçados e reconfortantes.

Tudo que se segue a partir dessa página tem um toque dado por cada um de vocês. Obrigado.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Mecânico

MÉTODOS NUMÉRICOS PARA O CÁLCULO DE FORÇAS DE TENSÃO SUPERFICIAL E MODELAGEM DA INTERFACE EM ESCOAMENTOS BIFÁSICOS

Leonardo Voltarelli Regini de Andrade

Novembro/2021

Orientador: Gustavo Rabello dos Anjos

Programa: Engenharia Mecânica

As forças de tensão superficial estão presentes em diversos fenômenos físicos e químicos, sendo de particular interesse para aplicações em fluidodinâmica. No estudo de escoamento bifásicos, por exemplo, o termo de forças de tensão superficial na interface dos fluidos deve ser levado em conta na resolução adequada da equação de Navier-Stokes, em algumas situações. Nesse sentido, o presente trabalho se pauta no modelo de *Continuum Surface Force* (CSF) para descrever e calcular tal grandeza. São desenvolvidas abordagens baseadas nos métodos numéricos de Elementos Finitos e Volumes Finitos, majoritariamente, para o cálculo de curvatura em curvas e superfícies. A partir disso, as forças de tensão superficial são computadas segundo dois esquemas propostos: um de interpretação Lagrangiana, com malha da interface acoplada, e outro Euleriana, com malha desacoplada. Assim, uma comparação entre as diferentes metodologias supracitadas é apresentada.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Mechanical Engineer

NUMERICAL METHODS FOR THE COMPUTATION OF SURFACE TENSION FORCES AND INTERFACE MODELING IN TWO-PHASE FLOWS

Leonardo Voltarelli Regini de Andrade

November/2021

Advisor: Gustavo Rabello dos Anjos

Department: Mechanical Engineering

Surface tension forces are present in many different physical and chemical phenomena, being of particular interest in the context of fluid dynamics. In the study of two-phase flows, for example, the term due to surface tension forces on the interface should be considered for an adequate solution of the Navier-Stokes equation in some cases. In that sense, the present work is based on the Continuum Surface Force (CSF) model in order to compute such values. Different approaches based on numerical methods such as Finite Elements and Finite Volumes, mostly, are developed for the computation of curvature on curves and surfaces. With that, the surface tension forces are calculated following two different schemes: one based on a Lagrangian description, with coupled interface mesh, and other in a Eulerian one, with uncoupled mesh. Therefore, a comparison between the different aforementioned methodologies is presented.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

As forças de tensão superficial são um tipo especial de força que atua em fluidos devido à interação molecular, que ocorre de forma diferente em sua superfície e no interior do volume. Para moléculas que não estão nas bordas de um volume de fluido, as forças de coesão existem mutuamente e se equilibram, enquanto que, na superfície, tal comportamento não se verifica. Nesse caso, pode-se dizer que as moléculas estão em um estado desfavorável no que tange sua energia, uma vez que possuem menos moléculas vizinhas [11] [12]. Surge, assim, a presença de forças de tensão superficial [Figura 1.1].

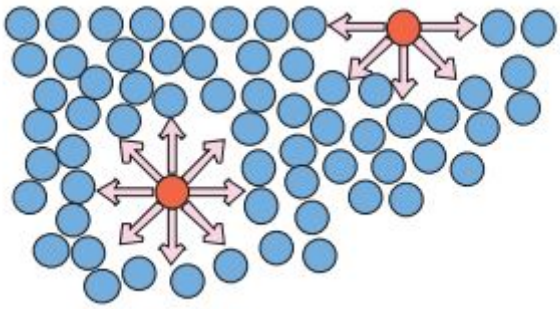


Figura 1.1: Representação ilustrativa das moléculas e forças de coesão em um fluido [1]

Tais grandezas estão ligadas, portanto, às tensões que atuam em uma escala pequena mas que, ainda assim, possuem efeitos macroscópicos. Alguns dos fenômenos relacionados podem ser observados no quotidiano; é o caso do formato que a água assume ao sair de uma torneira e a lâmina de água que se forma em uma piscina,

por exemplo. Não obstante, fenômenos físicos e químicos mais complexos estão intimamente atrelados à essas forças. Por isso, encontra-se uma larga quantidade de pesquisas que contempla modelos de tensão superficial e suas aplicações em processos químicos, biologia e medicina, geofísica, escoamentos multifásicos, entre diversos outros.

Como exemplos de aplicação em biologia e medicina, a pesquisa encontrada em [13] relaciona a formação de tecidos com forças de tensão superficial, como também ocorre em [14] e [15]. Similarmente, essas forças são importantes em processos como separação celular [16]. De fato, por sua origem microscópica, é natural que a tensão superficial tenha um importante papel nas iterações entre componentes biológicos elementares, como células e tecidos. Em [12], algumas aplicações são exploradas mais a fundo, como no comportamento de lipídios e proteínas.

O artigo [17], na área de ciência dos materiais, é focado na nucleação cristalina em líquidos e em materiais cerâmicos como o vidro. Tal processo é descrito com mais detalhes em [18]. Já na engenharia química, vários trabalhos dedicam-se ao estudo de tensões no contexto de compostos específicos e seu comportamento na formação de gotas.

No presente projeto, será dado destaque ao cálculo de forças de tensão superficial no contexto de escoamentos bifásicos sob uma ótica de dinâmica dos fluidos. O estudo de tais escoamentos é fundamental no desenvolvimento de sistemas de produção e exploração de petróleo, em que há a presença de água com óleos, gases e possivelmente partículas sólidas [19] [2] [20] [21].

Outras aplicações são encontradas em processos de controle térmico, uma vez que a presença da interface pode ser vantajosa em termos da troca de calor. Escoamentos bifásicos com esse intuito são utilizados em diversos contextos; desde centrais de geração de energia nuclear [22] ao resfriamento de componentes eletrônicos [8]. Também, escoamento de sangue e até de fluidos geológicos como magma são estudados a partir de tal descrição.

Nesses casos, assim como ocorre na superfície, na interface entre os componentes há a forte influência da força de tensão superficial, que atua diretamente no escoamento sendo, portanto, chamadas também de forças interfaciais. A modelagem correta de tal domínio físico é, assim, fundamental no estudo de tais tipos de

escoamento.

Nestes - e particularmente quando há um escoamento líquido-gás - estão presentes diferentes padrões que caracterizam sua distribuição topográfica. Devido às disparidades entre as propriedades, assim como aspectos ligados ao próprio sistema em que o escoamento está inserido, existem diversas formas com que as fases se comportam. Há, por exemplo, formação de pequenas bolhas dispersas, assim como escoamento anular e bolhas em golfadas ("slug") [Figura 1.2].

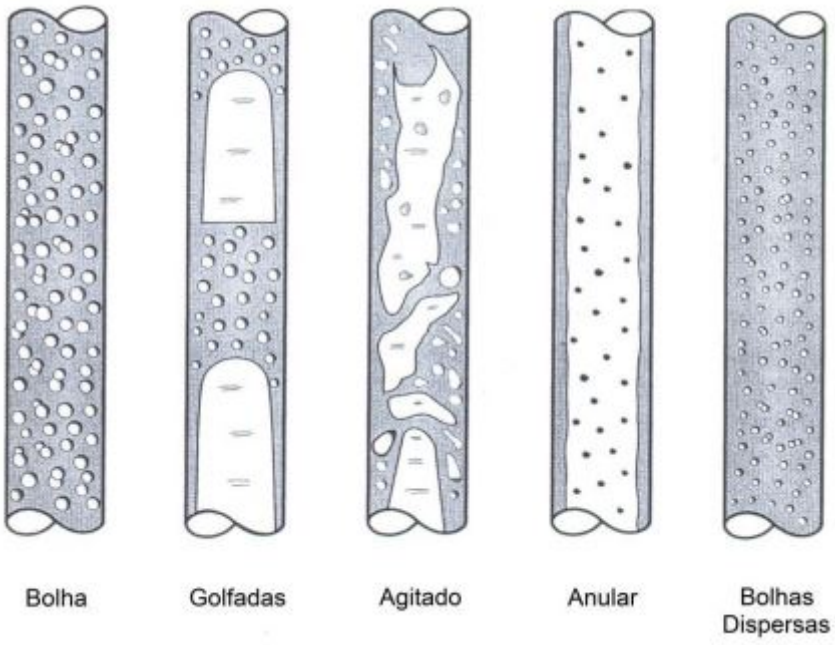


Figura 1.2: Padrões de escoamento líquido-gás [2]

Não obstante, uma descrição precisa que consiga capturar os padrões acima destacados, principalmente em relação ao movimento e mudanças na interface, coloca-se como um desafio no contexto da fluidodinâmica computacional. Como será explorado no decorrer do projeto, diferentes abordagens foram propostas com o objetivo de contornar problemas numéricos e outras perturbações nas soluções encontradas. Dessa forma, tal matéria mostra-se como um campo de frutífera atuação e especial interesse para a Engenharia Mecânica.

1.2 Objetivo

Os modelos encontrados para o cálculo das forças de tensão superficial, em geral, expressam sua dependência de grandezas geométricas, particularmente com a cur-

vatura da forma que os fluidos assumem. Assim sendo, para corretamente calcular o valor dessa força, é necessário antes aproximar tal propriedade geométrica.

O presente trabalho tem como objetivo inicial comparar diferentes métodos numéricos para a obtenção da curvatura em curvas e superfícies, de tal forma a indicar a metodologia mais adequada a partir dos resultados obtidos. Nesse sentido, são apresentadas as técnicas mais usuais, assim como propostas soluções alternativas, combinando resultados de múltiplas fontes e abordagens diferentes para um mesmo método.

Serão explorados os métodos de Diferenças Finitas (MDF), Elementos Finitos (MEF) e Volumes Finitos (MVF). Baseando-se no primeiro, será proposta uma discretização das equações de Frenet - que descreve curvas bidimensionais - com a qual é possível obter uma aproximação da curvatura.

Quanto ao Método de Elementos Finitos, este será usado para se obter uma discretização do operador Laplace-Beltrami que, aplicado nas coordenadas de uma curva ou superfície, fornece a curvatura média. Já o terceiro método numérico será empregado similarmente para o cálculo de tal propriedade em um contexto tridimensional.

A partir disso, como objetivo final, serão apresentadas duas abordagens para a obtenção de um valor para a força de tensão superficial e modelagem da interface; uma seguindo um modelo Euleriano, com malha desacoplada, e outra baseada em um esquema Lagrangiano, com malha acoplada.

Para tal, será utilizada da descrição das forças de tensão superficial seguindo o modelo de *Continuum Surface Force* (CSF), em que estas são descritas como forças de volume que atuam sobre a interface dos dois fluidos. Novamente, sua equação será discretizada utilizando o procedimento do MEF.

O projeto contempla, assim, o desenvolvimento de códigos, em linguagem de programação *Python*, para a realização dos cálculos supracitados. Com esse intuito, foi utilizado o *software* livre *Gmsh* para geração de malhas, que foram lidas utilizando bibliotecas instaladas em *Python*, em particular a biblioteca *meshio*. Os códigos disponíveis se encontram no Apêndice A.

1.3 Organização da tese

A presente tese está dividida em cinco capítulos, sendo esses:

- Capítulo 1: Introdução, motivação e apresentação geral do trabalho;
- Capítulo 2: Revisão Bibliográfica contemplando as noções básicas para desenvolvimento dos modelos propostos, dando destaque aos resultados relevantes de Geometria Diferencial, aos métodos numéricos empregados e ao embasamento teórico das forças de tensão superficial e das modelagens da interface entre os fluidos;
- Capítulo 3: Apresentação das metodologias utilizadas nos cálculos e sua implementação, incluindo comentários relativo ao código utilizado para a análise computacional;
- Capítulo 4: Resultados e aplicação dos modelos; comparação com valores conhecidos da literatura e discussão dos dados obtidos;
- Capítulo 5: Conclusão do trabalho e apresentação de propostas para projetos futuros.

Capítulo 2

Revisão Bibliográfica

2.1 Geometria Diferencial

A Geometria Diferencial é dividida em dois aspectos, como destacado por Manfredo P. do Carmo em [3], sendo eles a geometria diferencial clássica e a geometria diferencial global.

O primeiro está intimamente relacionado com o Cálculo Diferencial desenvolvido por Leibniz e, principalmente, por Newton - que foi quem primeiro definiu a noção de curvatura [23], tendo como foco propriedades locais das curvas e superfícies dependentes, em um dado ponto, somente de sua vizinhança. Mais tarde, com as pesquisas de Euler e Monge, os fundamentos da geometria diferencial tomaram forma [24]. Porém, Gauss é considerado seu maior contribuidor, em grande parte devido ao texto *Disquisitiones Circa Superficies Curvas* [24] [25], mas também por ter iniciado o estudo da geometria diferencial global.

Este segundo aspecto, por sua vez, refere-se às propriedades gerais dos objetos geométricos que dependem da geometria como um todo, e como são influenciados por grandezas locais. Nesse âmbito, inspirados pelo trabalho de Gauss, Bolyai e Lobachevsky romperam com a geometria euclidiana [23], ao defender a hiperbólica, o que, posteriormente, daria início ao estudo moderno da geometria com os trabalhos de Bernhard Riemann [25] e Henri Poincaré.

No contexto do presente trabalho, as propriedades locais serão de maior interesse. Assim, abaixo estão explicitados os conceitos básicos que serão aplicados posteriormente; deduções mais aprofundadas podem ser encontradas em diversas fontes,

como [3], [26] e [27].

2.1.1 Curvas em $\mathbb{R}^2$

As curvas no plano são as entidades mais triviais de estudo na Geometria Diferencial, mas de relevante importância para a compreensão de generalizações mais avançadas. Como será visto mais a frente, apesar de sua simplicidade, tais objetos serão utilizados como referências em testes do modelo proposto e simplificações em 2D.

Definição de Curva Plana: Uma curva plana dita diferenciável parametrizada é uma aplicação diferenciável $\alpha : I \to \mathbb{R}^2$ onde I é um intervalo aberto de $\mathbb{R}$

Ou seja, a aplicação α corresponde a cada ponto $t \in I$ um ponto $\alpha(t) = (x(t), y(t))$ em $\mathbb{R}^2$, sendo $x(t)$ e $y(t)$ funções diferenciáveis e t designado como o parâmetro da curva.

Define-se, então, o comprimento de arco da curva como a função:

$$s(t) = \int_{t_0}^{t} \|\alpha'(\xi)\| \, d\xi = \int_{t_0}^{t} \sqrt{(x'(\xi))^2 + (y'(\xi))^2} d\xi \tag{2.1}$$

Em que t_0 é um ponto arbitrário de I.

Sendo α como apresentada anteriormente, temos que $\|\alpha'(t)\|$ é contínua. Pelo Teorema Fundamental do Cálculo:

$$s'(t) = \|\alpha'(t)\| \tag{2.2}$$

Uma curva é dita parametrizada pelo comprimento de arco se o parâmetro t difere de $s(t)$ por uma constante:

$$s(t) = t + C \tag{2.3}$$

O que equivale a:

$$\|\alpha'(t)\| = 1 \tag{2.4}$$

A partir disso, são definidos:

$$\mathbf{t}(t) = \alpha'(t) = (x'(t), y'(t)) \tag{2.5}$$

$$\mathbf{n}(t) = (-y'(t), x'(t)) \tag{2.6}$$

O referencial $\{\mathbf{t}(t), \mathbf{n}(t)\}$ é dito o **Refencial de Frenet** da curva α. Nota-se que tal definição constitui uma base ortogonal do $\mathbb{R}^2$. De fato, $\mathbf{t}(t)$ representa o vetor tangente à curva no ponto t, enquanto $\mathbf{n}(t)$ é o vetor normal. Fica evidente, então, que a exigência de diferenciabilidade é necessária para garantir que tal referencial exista em todos os pontos da curva. Além disso, uma curva é dita regular se sua deriva não se anula, de forma que os vetores definidos acima nunca são identicamente nulos nesse caso.

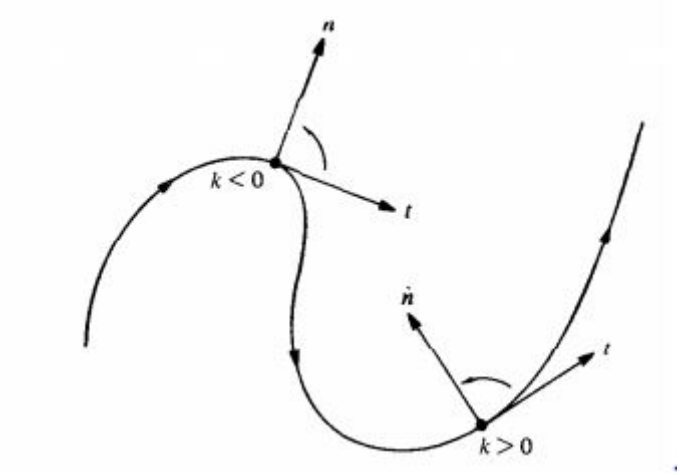


Figura 2.1: Curva no plano [3]

Sendo a curva em questão parametrizada pelo comprimento de arco, temos que o vetor $\mathbf{t}(s)$ é unitário e, assim, perpendicular à sua derivada. Logo, temos que $\mathbf{t}'(s)$ e $\mathbf{n}(s)$ são paralelos, portanto sendo possível escrever:

$$\mathbf{t}'(s) = k(s)\mathbf{n}(s) \tag{2.7}$$

onde $k(s)$ é uma função real definida no intervalo I chamada de **curvatura**. A imagem acima [Figura 2.1] fornece uma intuição do significado da curvatura, sendo relacionada com o quão afastada a curva está de seus vetores tangente e normal em um dado ponto. Além disso, possui sinal positivo quando a curva está localmente

contida no semiplano determinado pelos dois vetores e negativo caso contrário.

t e **n**, por sua vez, satisfazem o seguinte sistema de equações, conhecidas como **Equações de Frenet** para uma curva plana:

$$\begin{cases} \mathbf{t}'(s) = k(s)\mathbf{n}(s) \\ \mathbf{n}'(s) = -k(s)\mathbf{t}(s) \end{cases} \tag{2.8}$$

Vale notar que, no caso de uma curva parametrizada pelo comprimento de arco, o Referencial de Frenet constitui uma base ortonormal local do $\mathbb{R}^2$.

2.1.2 Curvas em $\mathbb{R}^3$

A definição de curva no $\mathbb{R}^3$ é similar à anterior, substituindo somente o contradomínio da função α. Tem-se:

$$\alpha(t) = (x(t), y(t), z(t)) \tag{2.9}$$

onde cada coordenada é dada por uma função diferenciável pelos mesmos motivos do caso anterior.

Da mesma forma, temos o comprimento de arco dado por:

$$s(t) = \int_{t_0}^{t} \|\alpha'(\xi)\| \, d\xi = \int_{t_0}^{t} \sqrt{(x'(\xi))^2 + (y'(\xi))^2 + (z'(\xi))^2} d\xi \tag{2.10}$$

as condições para que a curva seja parametrizada pelo comprimento de arco são as mesmas.

A definição de **curvatura**, contudo, é dada por:

$$k(s) = \|\alpha''(s)\| \tag{2.11}$$

ao passo que a curvatura representa, então, a velocidade com que a curva se afasta, em uma vizinhança de s da reta tangente no mesmo ponto.

É possível notar que, sendo a curva parametrizada pelo comprimento de arco e o vetor tangente a curva unitário, $\alpha'(s)$ e $\alpha''(s)$ são perpendiculares e, por isso, temos:

$$\alpha''(s) = k(s)\mathbf{n}(s) \tag{2.12}$$

onde, novamente, $\mathbf{n}(s)$ representa o vetor normal.

Similarmente, sendo $\mathbf{t}(s)$ o vetor tangente à curva, define-se o vetor binormal:

$$\mathbf{b}(s) = \mathbf{t}(s) \wedge \mathbf{n}(s) \tag{2.13}$$

Semelhantemente à definição de curvatura, é possível definir a **torção** τ da curva como:

$$\mathbf{b}'(s) = \tau(s)\mathbf{n}(s) \tag{2.14}$$

No caso tridimensional, as **Fórmulas de Frenet** são, portanto, dadas por:

$$\begin{cases} \mathbf{t}'(s) = k(s)\mathbf{n}(s) \\ \mathbf{n}'(s) = -k(s)\mathbf{t}(s) - \tau(s)\mathbf{b}(s) \\ \mathbf{b}'(s) = \tau(s)\mathbf{n}(s) \end{cases} \tag{2.15}$$

com o referencial local $\{\mathbf{t}(s), \mathbf{n}(s), \mathbf{b}(s)\}$ conhecido como **Triedro de Frenet**. Assim como no caso anterior, quando tratamos de curvas parametrizadas pelo comprimento de arco, tal referencial constitui uma base ortonormal, agora do $\mathbb{R}^3$.

Todas as curvas referidas no decorrer do presente trabalho serão consideradas como parametrizadas por seu comprimento de arco, uma vez que isso garante a validade das equações deduzidas.

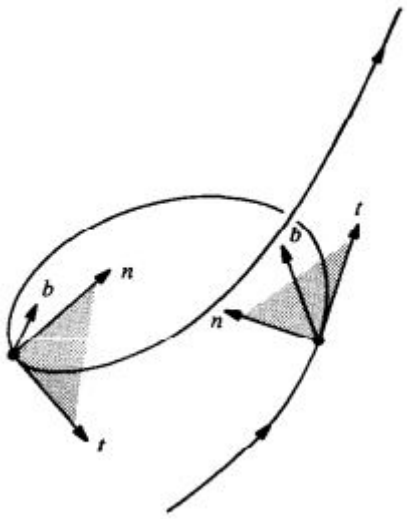


Figura 2.2: Curva no espaço [3]

2.1.3 Superfícies

As superfícies no espaço constituem grande parte do foco de estudo da Geometria Diferencial. Por ser um objeto mais complexo que os anteriores, sua teoria exige um embasamento em Cálculo Diferencial mais aprofundado e, por muitas vezes, requer conhecimento dos resultados obtidos para curvas.

A definição usual de superfície é dada a partir de um conjunto - e não como uma função. Do Carmo define uma superfície regular como:

Definição de Superfície: Chamamos um subconjunto S do $\mathbb{R}^3$ de superfície regular se, para todo ponto p em S, existe uma vizinhança V desse ponto no espaço e uma aplicação $\mathbf{x} : U \to V \cap S$, onde U é um conjunto aberto de $\mathbb{R}^2$.

$$\mathbf{x}(u, v) = (x(u, v), y(u, v), z(u, v)) \tag{2.16}$$

Além disso, são necessárias as seguinte propriedades:

- $\mathbf{x}$ é diferenciável;
- $\mathbf{x}$ é um homeomorfismo, ou seja, é uma aplicação contínua com inversa contínua;
- Para todo ponto $q \in U$, $d\mathbf{x}_q$ é injetiva.

Tais adições à definição anterior indicam a regularidade da superfície, assim garantindo algumas propriedades relevantes no âmbito da Geometria Diferencial. A injetividade de $\mathbf{x}$ garante, por exemplo, que não existe auto-interseção na superfície, o que, junto ao terceiro ponto, possibilita a definição unívoca de um plano tangente em qualquer ponto.

Sem as restrições acima, a superfície obtida não é dita regular. O cone é um exemplo simples de exceção à tal definição, o que pode ser verificado pela impossibilidade de definir um plano tangente único em seu vértice.

Para descrever o equivalente aos vetores tangente e normal na superfície, inicialmente observa-se que existe uma quantidade infinita de vetores que são tangentes em um dado ponto. De fato, são os vetores tangentes às curvas, como definido anteriormente, que pertencem à superfície e contêm o ponto em questão. Contudo,

com a forma com que foi apresentada a definição de superfície, pode-se obter um plano tangente único a partir de tais vetores, o que leva a um único vetor normal - o próprio vetor diretor do plano. Tem-se:

$$\mathbf{n}(p) = \frac{\frac{\partial \mathbf{x}}{\partial u} \wedge \frac{\partial \mathbf{x}}{\partial v}}{\left\| \frac{\partial \mathbf{x}}{\partial u} \wedge \frac{\partial \mathbf{x}}{\partial v} \right\|}(q) \tag{2.17}$$

em que $p = \mathbf{x}(q)$.

Assim como no caso das curvas, a definição de curvatura é uma importante propriedade das superfícies. Esta se baseia fortemente naquela dada para curvas no espaço e parte da definição de **curvatura normal**:

Definição de Curvatura Normal: Dada uma curva regular C que pertence a superfície S e passa pelo ponto p em S, a curvatura normal de C nesse ponto é dada por $k_n = k \cos\theta$, onde k é a curvatura de C em p e θ representa o ângulo entre o vetor normal à curva e o vetor normal da superfície [Figura 2.3]

Um resultado necessário para a definição das demais propriedades refentes à curvatura é o seguinte:

Teorema de Meusnier: Em uma superfície, todas as curvas que tem mesma reta tangente em um dado ponto possuem mesma curvatura normal nesse ponto.

Os valores máximo e mínimo da curvatura normal em um ponto da superfície, comumente designados k_1 e k_2 respectivamente, são ditos as curvaturas principais em dado ponto. À luz do Teorema de Meusnier, podemos definir $\mathbf{e_1}$ e $\mathbf{e_2}$ como as direções principais associadas a tais curvaturas. Assim, duas das mais importantes grandezas no estudo das superfícies são derivadas, sendo essas a Curvatura Gaussiana (K) e a Curvatura Média (H):

$$K = k_1 k_2 \tag{2.18}$$

$$H = \frac{k_1 + k_2}{2} \tag{2.19}$$

No presente projeto, será explorado o cálculo da curvatura média por sua relação com o modelo de forças de tensão superficial. Destaca-se que, comumente, em aplicação de fluidodinâmica, tal gradeza é definida sem o fator de 0.5, pela decorrente simplificação nos cálculos. Nos desenvolvimentos que se seguem, esta será a forma utilizada. Além disso, na literatura de geometria diferencial é mais comum utilizar-se a letra H, porém, em concordância com a literatura de fluidodinâmica tal grandeza será referida pela letra k, como feito para a curvatura 2D (também é comum encontrá-la como κ). Tem-se:

$$k = k_1 + k_2 \tag{2.20}$$

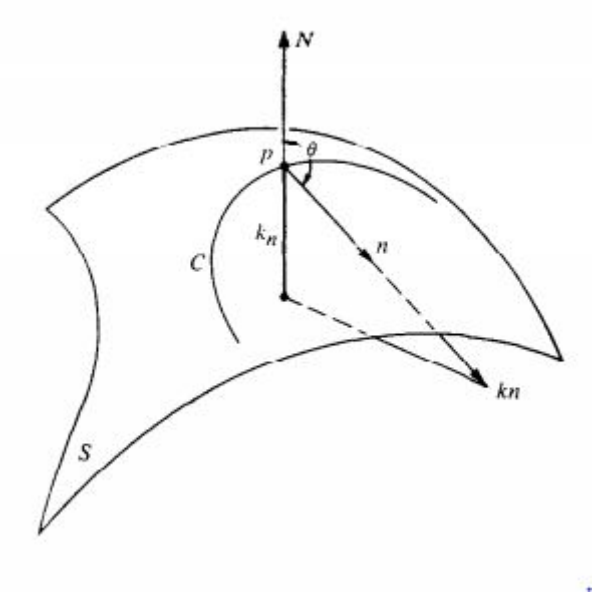


Figura 2.3: Curvatura Normal k_n [3]

2.1.4 Operador Laplace-Beltrami

Relativamente ao cálculo da curvatura de uma superfície, um operador, conhecido como operador Laplace-Beltrami, é normalmente utilizado por sua relação direta com tal propriedade e o vetor normal em um ponto.

De fato, esse operador é uma generalização do operador Laplaciano usual para diferentes geometrias contidas no espaço euclidiano. Ainda mais, pode ser estendido para qualquer variedade Riemanniana, isto é, uma variedade diferenciável à qual é possível associar uma métrica Riemanniana [28].

O Laplaciano usual, para uma função definida em um subespaço do espaço euclidiano, é dado por:

$$\Delta f \equiv \nabla^2 f = \nabla \cdot \nabla f \equiv div(grad\, f) \tag{2.21}$$

Ou ainda, em coordenadas cartesianas:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \tag{2.22}$$

Ou seja, é o mesmo que tomar o divergente do gradiente de uma função. Tal operador é, portanto, equivalente à noção de segunda derivada para uma função de várias variáveis, o que indica sua relação com a curvatura de uma dada geometria.

No caso do operador Laplace-Beltrami, é necessário definir os operadores de gradiente e divergente em uma variedade qualquer. Usualmente, utiliza-se a seguinte notação:

$$\Delta_s f = div_s \nabla_s f \tag{2.23}$$

A derivação de uma forma explicita do operador demanda uma abordagem mais extensa de geometria diferencial e, por isso, não será desenvolvida. Não obstante, o operador Laplace-Beltrami depende da métrica utilizada, uma vez que essas são a base da formulação dos operadores diferenciais em sua definição. De fato, uma forma explícita, dada em função da primeira forma fundamental da superfície, pode ser encontrada em textos de geometria diferencial mais avançados [29] [30].

O resultado relevante que será utilizado para calculo da curvatura, porém, é a seguinte relação:

$$\Delta_s \mathbf{x} = -k\mathbf{n} \tag{2.24}$$

Onde $\mathbf{x}$ é a parametrização da superfície, como apresentado na definição de superfície, ou seja, a aplicação que retorna suas coordenadas. k e $\mathbf{n}$, por sua vez, são a curvatura média e o vetor normal como definidos na seção anterior.

Devido à tal equação, o operador Laplace-Beltrami é amplamente utilizado em aplicações que buscam calcular a curvatura, estando presente sob diferentes formas

discretas. De fato, tal operador é empregado em diferentes contextos relativos à geometria computacional e computação gráfica, assim como em processamento de sinais e no estudo de EDPs [31].

Em [32], é proposta uma discretização do operador que garante convergência pontual em malhas de superfícies. Uma extensa prova teórica de tal convergência é exposta, assim como resultados favoráveis para testes em diferentes superfícies com malhas arbitrárias, principalmente em casos com malhas não-estruturadas - isto é, em que os elementos têm tamanhos diferentes e estão dispostos de maneira irregular.

[33] apresenta uma forma discreta aplicada à suavização de superfícies baseada em processamento de sinais. Assim, "sinais de superfícies discretas"são analisados em malhas poliédricas genéricas de forma à obter um algoritmo de otimização adequado.

Já o artigo [34], utiliza uma aproximação da superfície em um ponto a partir de uma superfície quadrática, que é encontrada minimizando uma função de energia. Dessa forma, o operador Laplace-Beltrami é aproximado na superfície discretizada calculando as quantidades diferenciais - diga-se primeira forma fundamental - da superfície quadrática. Resultados de testes quanto à velocidade, precisão e robustez do método são apresentados.

Em [31], por outro lado, são apresentados diversos esquemas de discretizações simples. O objetivo de tal trabalho é fazer uma comparação entre os esquemas propostos sob uma perspectiva do erro cometido no calculo da curvatura. Para tal são comparados os resultados em quatro superfícies diferentes.

Uma importante discretização do operador é proposta em [10]. Como será citado na próxima seção, o artigo propõe o cálculo de diferentes propriedades de geometria diferencial discretas em malhas triangulares, também apresentando diferentes testes de validação. A formulação proposta é conhecida como "fórmula da cotangente", devido à presença de tal função trigonométrica na expressão. Esta será explorada mais a frente, e é expressa pela equação [3.6].

2.1.5 Geometria Diferencial Discreta

Até então, os resultados apresentados concernem objetos geométricos em que existe certa suavidade, no sentido de que é possível definir derivadas e aplicar operadores

diferenciais diretamente. De fato, a definição de superfície regular foi produzida de uma forma que tais propriedades fossem verificadas.

Para aplicação dos métodos computacionais, contudo, não é viável a utilização de objetos contínuos dessa forma; é necessário que os domínios em questão sejam discretizados para calcular as quantidades requeridas em um número finito de pontos ou volumes.

Os métodos em si serão aprofundados na seção seguinte, no entanto, ainda no que tange a geometria diferencial, algumas aproximações em termos de uma discretização dos objetos estudados serão apresentadas de antemão.

Discretização de curvas e superfícies

A divisão de uma curva em pedaços menores é o caso mais simples de um objeto no contexto da geometria diferencial discreta. Esse processo é baseado em definir pontos na curva que serão interpolados por reta, fazendo com que a curva possa, agora, ser entendida como uma aplicação com partes diferenciáveis.

O processo de discretização baseia-se, portanto, na escolha de uma quantidade finita de pontos sobre a curva. No contexto do Método de Elementos Finitos, tais pontos são denominados nós e as retas obtidas são os elementos [Figura 2.4].

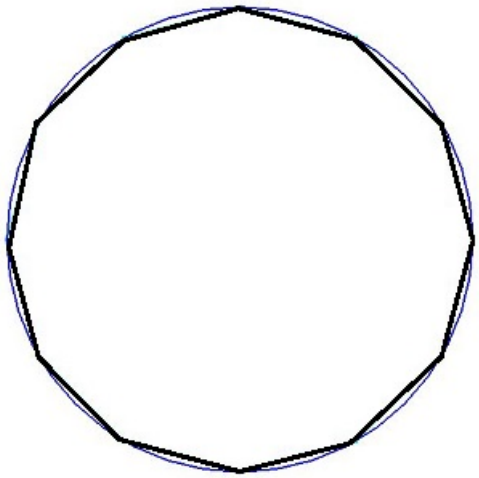


Figura 2.4: Circunferência discretizada

No caso das superfícies, porém, a divisão se dá por meio de elementos tridimensionais, criando uma malha que se aproxima do formato do objeto original. Em geral, tal malha é feita a partir da definição de pontos que serão interpolados por retas seguindo um esquema de triangulação, como a triangulação de Delaunay. Apesar

de essa ser a prática mais comum, é possível que a superfície seja aproximada por outros polígonos tridimensional.

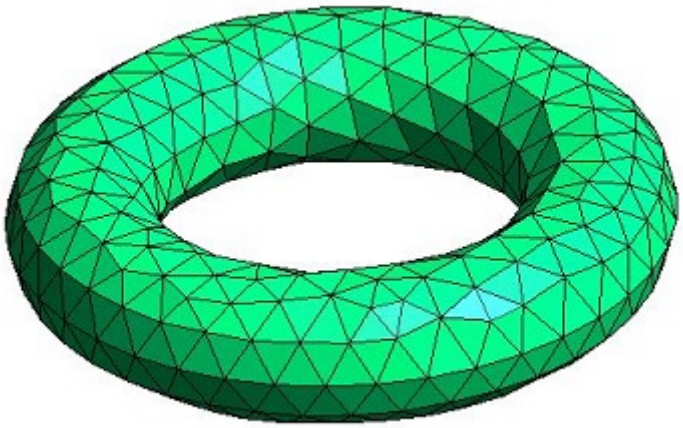


Figura 2.5: Superfície discretizada por elementos triangulares

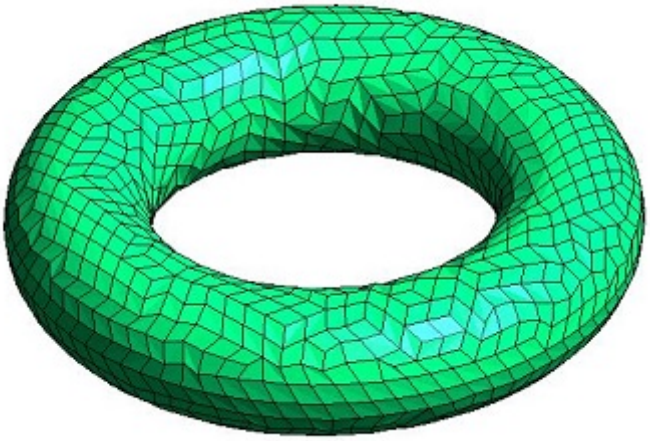


Figura 2.6: Superfície discretizada por quadriláteros

Como já citado, tais definições da geometria diferencial discreta estão fortemente alinhadas como os métodos numéricos que serão utilizados, como o método de Diferenças Finitas e Elementos Finitos. Além disso, nota-se que quando mais nós existem, mais aproxima-se do objeto original, o que indica, em geral, resultados melhores. Todavia, a demanda computacional também aumenta.

Vetor normal discreto

Para o cálculo do vetor normal, por outro lado, recorre-se, usualmente, à sua ortogonalidade com os vetores tangentes.

No caso bidimensional, tais vetores podem ser facilmente encontrados aplicando uma rotação no plano. Assim, uma aproximação pode ser obtida considerando que, em cada ponto, tal vetor é o resultado da soma das contribuições das duas arestas [Figura 2.7] a que o ponto pertence [8].

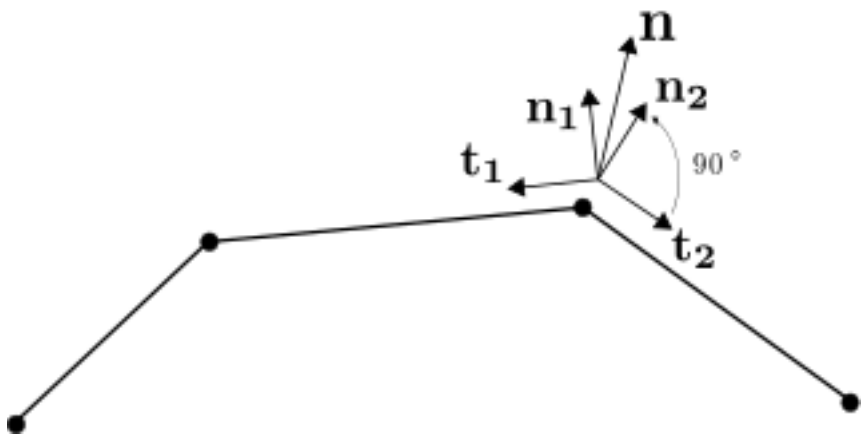


Figura 2.7: Obtenção dos vetores normais em uma curva discreta

No caso 3D, o vetor normal pode ser ser obtido de forma similar, calculando a soma dos vetores normais a cada face em que um dado nó está contido. Ou seja, para cada elementos toma-se o produto vetorial dos vetores tangentes que contêm o ponto em que deseja-se calcular a normal e, então, os vetores obtidos são somados [8] [Figura 2.8].

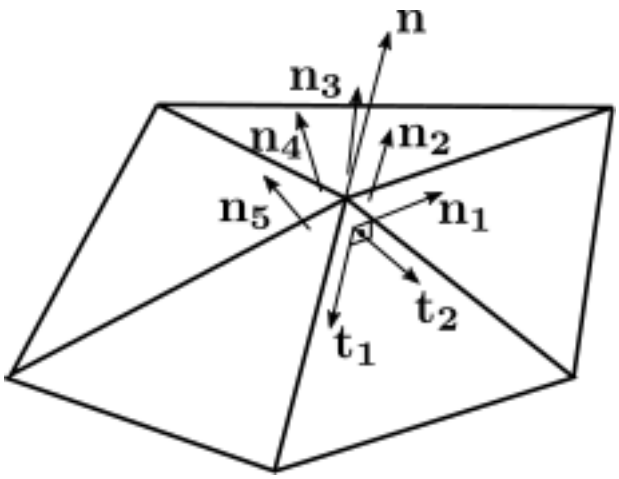


Figura 2.8: Obtenção do vetor normal em uma superfície discreta

Outras grandezas

Há diversas outras formas de se obter aproximações para propriedades geométricas discretas, que encontram aplicações em áreas além da fluidodinâmica.

Como citado, em [10] são apresenta algumas ideias que serão exploradas no próximo capítulo. No artigo estão explicitadas formas de obter aproximações para as curvaturas média e gaussiana, assim como para as direções principais em uma superfície.

Os outros trabalhos mencionados sobre formas discretas do operador Laplace-Beltrami também apresentam formas de calcular a curvatura discreta. Não obstante, no presente projeto, tal grandeza será calculada à partir de uma discretização do operador baseada no método de Elementos Finitos e Volumes Finitos (a serem apresentados a seguir), assim como nas Equações de Frenet. O cálculo da curvatura em si será destacado no Capítulo 3.

2.2 Métodos Numéricos

A utilização de métodos numéricos para resolução de problemas é uma ferramenta de vasta aplicação, devido à sua capacidade de aproximar soluções em casos complexos e quando não há uma solução analítica. Nesse sentido, diversos problemas de engenharia e matemática são resolvidos com o auxílio de técnicas de computação que utilizam tais métodos. No contexto do presente projeto, são de particular interesse métodos empregados na resolução de equações diferenciais e que utilizam a discretização de domínios contínuos, como será explorado a seguir.

2.2.1 Diferenças Finitas

O Método de Diferenças Finitas (MDF) é baseado na discretização do domínio de uma certa função com o intuito de aproximá-la e, assim, encontrar uma possível solução para uma equação diferencial. Em sua forma original, proposta por Euler na segunda metade do século XVIII [35], é considerado um dos mais simples método numéricos.

Em princípio, pode ser pensado a partir da própria definição de derivada em um ponto como um limite:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.25}$$

O que pode ser aproximado para:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{2.26}$$

Tal aproximação também pode ser vista como um truncamento na Série de Taylor da função, de forma que temos:

$$f'(x) = \frac{f(x+h) - f(x)}{h} + o(h) \tag{2.27}$$

Assim, o domínio em que a função está definida é dividido em pedaços de tamanho h, chamado de passo do método, e, iterativamente, seus valores são aproximados em cada ponto assim definido. Ou seja, define-se uma sequência finita de pontos (x_n) igualmente espaçados em que a função será calculada, o que produz uma sequência (y_n) de pontos da função buscada.

Seja o problema dado por:

$$y' = F(x, y) \quad y(x_0) = y_0 \tag{2.28}$$

Segundo a abordagem das diferenças finitas, tem-se:

$$y_{n+1} = y_n + F(x_n, y_n)(x_{n+1} - x_n) \tag{2.29}$$

Fica clara, a partir da equação acima, a necessidade de que a equação diferencial constitua um problema de valor inicial, de forma que o método possa ser iniciado. Além disso, para garantir estabilidade, existência e unicidade de solução, são fundamentais algumas restrições. Como destacado em [36], é suficiente que F seja Lipschitziana, o que implica continuidade em $\mathbb{R}^n$ [37].

Essa formulação é a mais simples do método de diferenças finitas, conhecida como Método de Euler ou Método da Tangente [Figura 2.9], por sua interpretação geométrica [36] [35].

Apesar disso, existem formas mais abrangentes - e mais relevantes computacionalmente - do MDF. Como desenvolvido acima, temos um método dito progressivo,

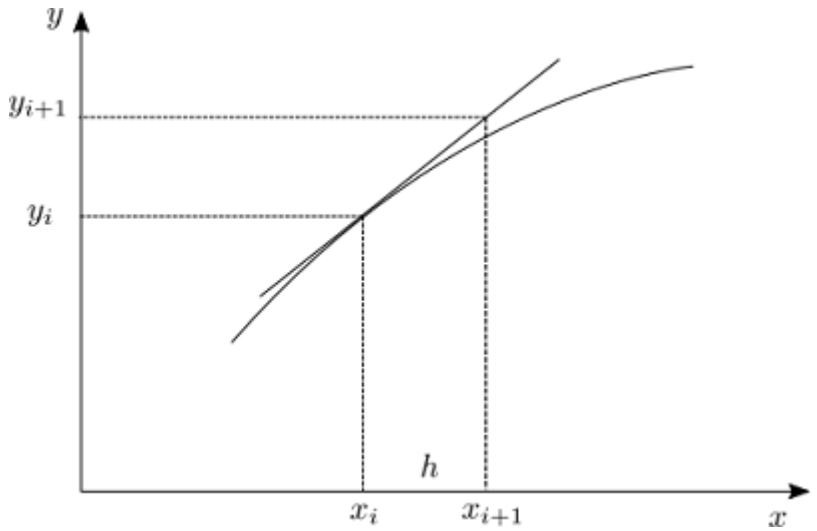


Figura 2.9: Interpretação Geométrica do Método de Euler

mas é comum também encontrar a forma regressiva:

$$f'(x) = \frac{f(x) - f(x-h)}{h} + o(h) \tag{2.30}$$

e a formulação centrada:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + o(h^2) \tag{2.31}$$

Destaca-se que, assim como uma aproximação para a primeira derivada pode ser obtida a partir da Série de Taylor, derivadas de ordem maior também podem ser encontradas da mesma maneira.

Na prática, como mencionado, o método de Euler só é empregado em equações básicas, sendo mais comum a utilização de método mais gerais, conhecidos como métodos de Runge-Kutta. Também, variações do método podem ser empregadas na resolução de sistemas de equações diferenciais, assim como em equações diferenciais parciais. Outra possibilidade é a utilização de passos variáveis.

O método de Runge-Kutta de quarta ordem, por exemplo, tem a seguinte formulação [35]:

$$y_{n+1} = y_n + h\left(\frac{k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}}{6}\right) \tag{2.32}$$

$$
\begin{aligned}
k_{n1} &= F(x_n, y_n) \\
k_{n2} &= F(x_n + 0.5h, y_n + 0.5hk_{n1}) \\
k_{n3} &= F(x_n + 0.5h, y_n + 0.5hk_{n2}) \\
k_{n4} &= F(x_n + h, y_n + hk_{n3})
\end{aligned}
\tag{2.33}
$$

A classe de métodos de Runge-Kutta é, assim, definida para múltiplas ordem, que equivalem ao número parâmetros k como na formulação acima. Quarta ordem é o tratamento mais comum, enquanto que se nota que o método de primeira ordem seria equivalente ao Método de Euler.

2.2.2 Elementos Finitos

O Método de Elementos Finitos (MEF), assim como o método anterior, tem ampla empregabilidade em problemas relacionados à equações diferenciais, tanto ordinárias quanto parciais. Sua formulação é mais elaborada que o caso das Diferenças Finitas - ainda que também baseada na discretização de um domínio. Nesse sentido, o MEF pode ser utilizado em geometrias complexas que não são facilmente capturadas por outros métodos numéricos, o que assegura sua versatilidade frente à análises de engenharia.

Como já referido anteriormente, o método consiste em dividir o domínio do problema em subdomínios chamadas elementos, enquanto que são escolhidos pontos, ditos nós, em que serão aproximadas as grandezas e funções de interesse. A junção desses define uma malha de elementos finitos [Figura 2.10].

Os nós, assim como os elementos, são numerados em ordem crescente seguindo alguma lógica para que seja possível então referenciá-los. No caso de elementos bidimensionais, por exemplo, numera-se os nós de um elemento sempre seguindo a mesma orientação, seja no sentido horário ou anti-horário. Pode-se então, colocar uma numeração tanto local dos nós em cada elemento, quanto outra global que dispõe de todos os nós.

A origem do método é mais recente que a do anterior, sendo normalmente colocada na metade do século XX. Há certa dificuldade em traçar sua origem exata,

sendo creditada a R. Courant [4] por sua aplicação de elementos triangulares para a aproximação variacional em um problema de vibrações [38], no ano de 1943, ou a Hrennikoff, por um trabalho realizado em 1941 [39]. Por volta de 1950, contudo, trabalhos na indústria aerospacial deram os primeiros passos na consolidação das principais ideias do método, que, então, seriam desenvolvidas nos próximos anos com o forte crescimento de pesquisas em suas aplicações e formalização [38].

Mesmo que inicialmente tenha sido empregado para análises de mecânica estrutural, sua utilização em problemas de transferência de calor e em escoamentos de fluidos foi desenvolvida à partir de abordagens como os Métodos de Galerkin. Assim, o MEF é de grande importância para a área conhecida como Computacional Fluid Dynamics (CFD), em que emprega-se métodos computacionais na solução de problemas complexos no âmbito da modelagem de fluidos. O método também ganhou notoriedade em aplicações não-lineares, de particular interesse em modelos de bioengenharia [39], por exemplo.

O sucesso do MEF, portanto, é decorrência do alinhamento de um embasamento teórico e do desenvolvimento de tecnologias de processamento de informação e armazenamento de dados; sendo possível, hoje, sua aplicação em computadores pessoais e em problemas de alta demanda computacional.

Já no final dos anos 60, viu-se o surgimento de programas dedicados exclusivamente à seu emprego em situações específicas [38]. Atualmente, diversos *softwares*, como ANSYS e ABAQUS, são capazes de realizar cálculos de Elementos Finitos em um contexto multifísico, em que análises de diferentes campos da engenharia e física são utilizadas em conjunto para um entendimento completo do problema. Ademais,

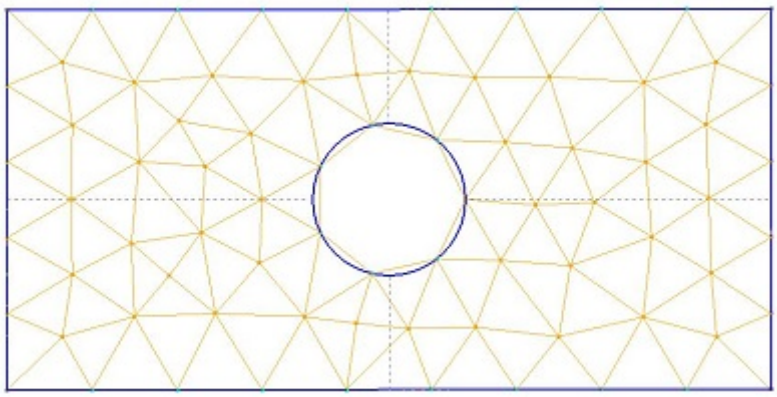


Figura 2.10: Malha bidimensional de Elementos Finitos

softwares de licença privada e uso exclusivo empresarial são empregados em diversas indústrias.

Sua aplicação segue os seguintes passos:

1. Obtenção da equação diferencial que modela o fenômeno a ser estudado em sua "forma forte";
2. Formulação da "forma fraca"do problema a partir da integração por partes, utilizando uma função peso;
3. Definição de uma aproximação da função de interesse, assim como da função peso;
4. Aplicação das discretizações à "forma fraca"e obtenção de uma equação matricial;
5. Cálculo das matrizes elementares locais;
6. Criação das matrizes globais, por meio do *Assembling* das matrizes locais;
7. Imposição das condições de contorno e derivação da solução por meio do sistema linear obtido.

Forma Forte

A obtenção da forma forte de um problema provém diretamente de sua interpretação física e aplicação de leis, como a conservação de massa, energia e momento. Nesse sentido, quando há a dedução de uma equação diferencial, o que é obtido, em geral, é a formulação forte, em que as condições de contorno do problema devem ser satisfeitas completamente e deseja-se uma solução exata.

Um exemplo de forma forte é como comumente se escreve a equação de Navier-Stokes, assim como diversas outras equações diferenciais parciais. Também encontram-se muitas aplicações no contexto da mecânica dos sólidos, em que fenômenos associados à flexão e flambagem, por exemplo, são descritos por equações diferenciais de segunda ordem.

Forma Fraca

A partir da forma forte, pode-se obter uma nova formulação baseada em aproximar a solução exata. Nesse sentido, busca-se encontrar uma função tentativa - normalmente chamada u - que seja solução ao menos em um sentido mais fraco, não necessariamente cumprindo com todas as restrições da forma original e assumindo os valores exatos em todos os pontos. Assim, tal descrição também recebe o nome de forma variacional.

Para a função tentativa, é necessária a seguinte propriedade em todo o domínio Ω do problema:

$$\int_{\Omega} \|\nabla u\|^2 \, d\Omega < \infty \tag{2.34}$$

Ou seja, a função u deve ter o quadrado da norma de seu gradiente integrável, além de que seja necessariamente derivável. Também exige-se que a função em si tenha seu quadrado integrável. Nota-se, portanto, já uma flexibilização da forma fraca: a solução buscada mesmo para um problema de segunda ordem não necessariamente é de classe C^2. Diz-se, assim, que a função tentativa pertence à um espaço de Sobolev, isto é, um espaço vetorial de funções munido de uma norma que garante a regularidade de seus elementos, assim como de suas derivadas até a requerida ordem.

Para obtenção da forma fraca, então, escolhe-se uma função w que deve pertencer ao mesmo espaço de funções que u, além de obedecer restrições adicionais quanto às condições do contorno. Para exemplificar o procedimento, consideremos o seguinte problema, em sua forma forte:

$$\begin{aligned} \Delta u + \mathbf{v} \cdot \nabla u &= f \ \text{ em } \ \Omega \\ u &= g \ \text{ em } \ \Gamma_D \\ \nabla u \cdot \mathbf{n} &= h \ \text{ em } \ \Gamma_N \end{aligned} \tag{2.35}$$

onde f, g e h são funções escalares diferenciáveis em cada variável e $\mathbf{v}$ um vetor constante.

Γ_D e Γ_N representam partes do contorno do domínio em questão, isto é $\partial\Omega = \overline{\Gamma_D \cup \Gamma_N}$. Sua diferença está no tipo de condição de contorno designada. No primeiro

há a chamada condição de contorno de Dirichlet, em que se prescreve um valor para a função em si, enquanto que no segundo a condição é dita de Neumann, em que se dá um valor para a derivada da função. No primeiro caso, a condição é dita essencial, ao passo que a outra diz-se natural.

Tal diferença de nomenclatura é bem clara no que tange a forma fraca de um problema. De fato, exige-se que a função tentativa obedeça as condições essenciais, mesmo que não se verifiquem as naturais. Existem ainda outros tipos de condição de contorno, como a de Robin, em que há uma combinação da função com sua derivada. Apesar disso, Dirichlet e Neumann são, consideravelmente, as mais comuns.

A função peso w, além de também pertencem ao espaço de Sobolev da função tentativa, deve se anular no contorno essencial, isto é:

$$w = 0 \ \text{ em } \ \Gamma_D \tag{2.36}$$

Para, então, obter a forma fraca, considera-se a equação diferencial original multiplicada por w e integrada no domínio. Isto equivale à uma ponderação da equação, de forma que fica clara a ideia de utilizar w como um peso para a solução. Assim:

$$\int_\Omega w(\Delta u + \mathbf{v} \cdot \nabla u - f)\, d\Omega = 0 \tag{2.37}$$

Ou ainda:

$$\int_\Omega w\Delta u\, d\Omega + \int_\Omega w\, \mathbf{v} \cdot \nabla u\, d\Omega = \int_\Omega wf\, d\Omega \tag{2.38}$$

Visando flexibilizar às restrições sobre as funções, assim como chegar em um problema simétrico após a discretização (o que será explicado mais a frente), utiliza-se integrações por parte para igualar o grau das derivadas de u e w. Por meio da aplicação de uma das Identidades de Green:

$$\int_\Omega \nabla w \cdot \nabla u d\Omega + \int_\Omega wf d\Omega = \int_{\Gamma_D} w\nabla u \cdot \mathbf{n} d\Gamma_D + \int_{\Gamma_N} w\nabla u \cdot \mathbf{n} d\Gamma_N + \int_\Omega w\mathbf{v} \cdot \nabla u d\Omega \tag{2.39}$$

Com a identidade 2.36, tem-se:

$$\int_{\Omega} \nabla w \cdot \nabla u \, d\Omega + \int_{\Omega} w f \, d\Omega = \int_{\Gamma_N} w h \, d\Gamma_N + \int_{\Omega} w \, \mathbf{v} \cdot \nabla u \, d\Omega \tag{2.40}$$

Assim, o problema em sua forma fraca é definido como sendo encontrar uma função u no espaço de funções referido e que satisfaça as condições de contorno essenciais e a equação acima. Isso dado que w é uma função qualquer que verifica as condições acima mencionadas.

Discretização do domínio e funções de forma

Para proceder com o Método de Elementos Finitos propriamente, é necessário então discretizar o domínio de u e w, fazendo assim a aproximação e empregando o método variacional. Existem diferentes esquemas que podem ser utilizados para tal definição. Dentre eles se destaca o método de Galerkin, em que as funções de forma para u e w são iguais. Tal método possui vasta aplicação em problemas de dinâmica dos fluidos, mas existem também variações que são utilizadas em situações mais específicas. Exemplos são os métodos de Ritz-Galerkin, Petrov-Galerkin, Taylor-Galerkin e Bubnov-Galerkin.

Inicialmente aplica-se a divisão do domínio em elementos, definindo assim a malha a ser utilizada, como o exemplo em [Figura 2.10]. A partir disso, são definidas as seguintes somas que aproximam os valores das funções em cada elemento:

$$u^e = \sum_{\text{nós}} N_i u_i \tag{2.41}$$

$$w^e = \sum_{\text{nós}} N_j w_j \tag{2.42}$$

$$f^e = \sum_{\text{nós}} N_k f_k \tag{2.43}$$

em que N_i, N_j e N_k são chamadas de funções de forma (ou de interpolação) e u_i e w_j são incógnitas constantes do problema. A soma acima é tomada em todos os pontos do elemento em questão; por exemplo, no caso unidimensional somam-se os valores nos dois extremos da reta [Figura 2.11]. Os sobrescritos indicam o elemento a que a função pertence, enquanto os subscritos estão indicando a qual nó

está associada no sistema local do elemento.

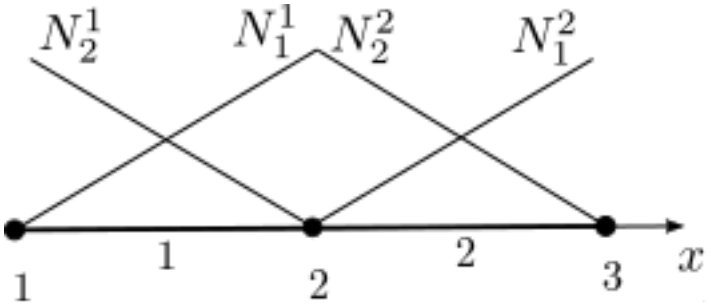


Figura 2.11: Funções de forma lineares em elementos unidimensionais

Substituindo as somas na equação 2.40:

$$\int_\Omega \sum_{i,j} \nabla N_j \cdot \nabla N_i u_i w_j d\Omega + \int_\Omega \sum_{j,k} N_j w_j N_k f_k d\Omega = \int_{\Gamma_N} \sum_j N_j w_j h d\Gamma_N + \int_\Omega \sum_{i,j} N_j w_j \mathbf{v} \cdot \nabla N_i u_i d\Omega \tag{2.44}$$

Dividindo por w_j e utilizando a linearidade da integral e do produto escalar de vetores, escreve-se:

$$\mathbf{K}^e\mathbf{u}^e + \mathbf{M}^e\mathbf{f}^e = \text{c.c.} + \mathbf{v} \cdot \mathbf{G}^e\mathbf{u}^e \tag{2.45}$$

Em que $\mathbf{K}^e$, $\mathbf{M}^e$ e $\mathbf{G}^e$ são as matrizes elementares que tem as seguintes entradas:

$$k_{ij} = \int_\Omega \nabla N_j \cdot \nabla N_i \, d\Omega \tag{2.46}$$

$$m_{jk} = \int_\Omega N_j N_k \, d\Omega \tag{2.47}$$

$$\mathbf{g_{ij}} = \int_\Omega N_j \nabla N_i \, d\Omega \tag{2.48}$$

Nota-se que $\mathbf{g_{ij}}$ são vetores, de forma que será conveniente dividir a matriz em cada uma das coordenadas do problema posteriormente. O termo "c.c."representa as condições de contorno que não foram anuladas na forma fraca. A matriz $\mathbf{M}$ é comumente chamada de matriz de massa, enquanto $\mathbf{K}$ é a matriz de rigidez. Tal nomenclatura está associada a modelagem de mecânica dos sólidos pelo papel que essas matrizes empregam na equação acima, comparativamente ao da massa e

rigidez na cinemática de um corpo. A matriz $\mathbf{G}$ é dita matriz gradiente, por ser uma discretização desse operador diferencial.

O método de Galerkin consiste em definir as funções de forma como iguais, isto é, $N_i = N_j = N_k$ em cada nó da malha. Dessa forma, garante-se que as matrizes $\mathbf{M}$ e $\mathbf{K}$ serão simétricas. Assim, tomando funções de forma usuais, como expressões polinomiais, é fácil obter os valores para os termos acima; em certo casos estes já estão disponíveis na literatura em formas de simples implementação.

Somando em cada elemento da malha, constrói-se um sistema linear com as entradas das matrizes dadas pelas expressões anteriores:

$$\mathbf{Ku} + \mathbf{Mf} = \mathbf{c.c.} + \mathbf{v} \cdot \mathbf{Gu} \tag{2.49}$$

Destaca-se que, na equação acima, $\mathbf{u}$, $\mathbf{f}$ e $\mathbf{c.c.}$ são vetores que contêm os valores nodais das funções, de forma que o sistema acima retorna os valores de u nos pontos desejados. Rearranjando e considerando que a matriz obtida é inversível:

$$\mathbf{u} = (\mathbf{K} - \mathbf{v} \cdot \mathbf{G})^{-1}(\mathbf{c.c.} - \mathbf{Mf}) \tag{2.50}$$

Matrizes Elementares

A partir do método de Galerkin, tomando as funções de forma como iguais, as matrizes elementares podem ser calculadas, como comentado. Além disso, elas serão simétricas, o que facilita o cálculo numérico.

No caso de um elemento unidimensional, as funções de forma dependem somente da única coordenada presente. Podem ser escolhidas as seguintes interpolações, em que x_1 e x_2 representam as coordenadas dos respectivos nós no elemento:

$$N_1 = 1 - \frac{x - x_1}{h} \tag{2.51}$$

$$N_2 = \frac{x - x_2}{h} \tag{2.52}$$

De fato, são interpolações como mostrado em [Figura 2.11]. Essas proporcionam

as seguintes matrizes:

$$\mathbf{k}^e = \frac{1}{h}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2.53}$$

$$\mathbf{m}^e = \frac{h}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \tag{2.54}$$

onde h representa o comprimento do elemento.

Para o caso bidimensional, há uma gama de elementos que podem ser empregados, em que utilizam-se quantidades diferentes de nós. Será escolhido para o presente projeto o elemento de triângulo linear por sua simplicidade e eficiência frente à problemas numéricos [Figura 2.12].

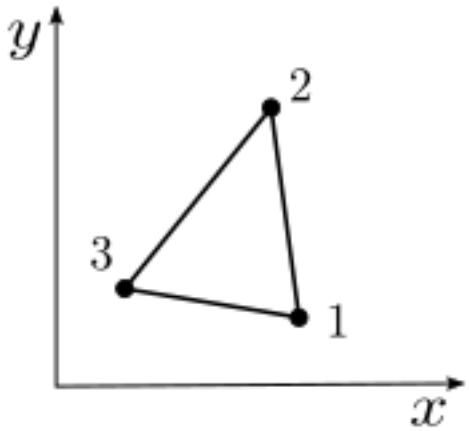


Figura 2.12: Elemento Triangular Linear

Escolhendo a base de funções de forma como:

$$N_1 = \frac{1}{2A}(a_1 + b_1 x_1 + c_1 y_1) \tag{2.55}$$

$$N_2 = \frac{1}{2A}(a_2 + b_2 x_2 + c_2 y_2) \tag{2.56}$$

$$N_3 = \frac{1}{2A}(a_3 + b_3 x_3 + c_3 y_3) \tag{2.57}$$

Em que a área do elemento pode ser calculada como:

$$A = \frac{1}{2} \det \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix} \tag{2.58}$$

Os coeficientes do polinômio interpolador são dados por:

$$\begin{aligned} a_1 &= x_2 y_3 - x_3 y_2 \qquad & b_1 &= y_2 - y_3 \qquad & c_1 &= x_3 - x_2 \\ a_2 &= x_3 y_1 - x_1 y_3 \qquad & b_2 &= y_3 - y_1 \qquad & c_2 &= x_1 - x_3 \\ a_3 &= x_1 y_2 - x_2 y_1 \qquad & b_1 &= y_1 - y_2 \qquad & c_3 &= x_2 - x_1 \end{aligned} \tag{2.59}$$

De forma que é obtida a seguinte $\mathbf{k}^e$:

$$\mathbf{k}^e = A\mathbf{B}^T\mathbf{D}\mathbf{B} \tag{2.60}$$

Em que as matriz $\mathbf{B}$ e $\mathbf{D}$ são dadas por:

$$\mathbf{B} = \frac{1}{2A} \begin{bmatrix} b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \tag{2.61}$$

$$\mathbf{D} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \tag{2.62}$$

Considerando $k_x = k_y = 1$, $\mathbf{D}$ torna-se a matriz identidade.

As demais matrizes elementares são:

$$\mathbf{m}^e = \frac{A}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \tag{2.63}$$

$$\mathbf{g_x}^e = \frac{1}{6} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \tag{2.64}$$

$$\mathbf{g_y}^e = \frac{1}{6}\begin{bmatrix} c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \tag{2.65}$$

Por fim, no caso tridimensional, utilizando elementos tetraédricos lineares [Figura 2.13] de forma similar ao caso bidimensional, tem-se a seguinte base:

$$N_1 = \frac{1}{6V}(a_1 + b_1x_1 + c_1y_1 + d_1z_1) \tag{2.66}$$

$$N_2 = \frac{1}{6V}(a_2 + b_2x_2 + c_2y_2 + d_2z_2) \tag{2.67}$$

$$N_3 = \frac{1}{6V}(a_3 + b_3x_3 + c_3y_3 + d_3z_3) \tag{2.68}$$

$$N_4 = \frac{1}{6V}(a_4 + b_4x_4 + c_4y_4 + d_4z_4) \tag{2.69}$$

E o volume V pode ser calculado por:

$$V = \frac{1}{6}\det\begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{bmatrix} \tag{2.70}$$

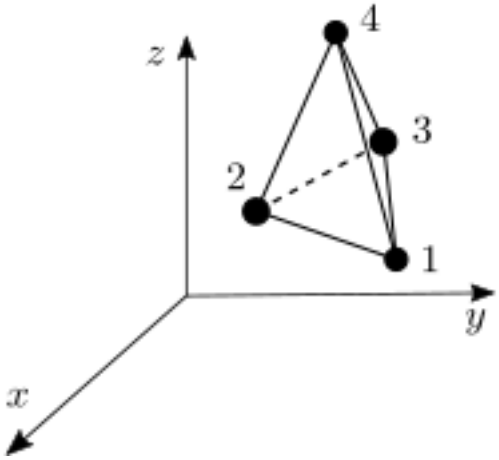


Figura 2.13: Elemento Tetraédricos Linear

Por sua vez, as matrizes elementares são dadas por:

$$\mathbf{k}^e = A\mathbf{B}^T\mathbf{D}\mathbf{B} \tag{2.71}$$

Em que as matriz **B** e **D** são:

$$\mathbf{B} = \frac{1}{6V}\begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \tag{2.72}$$

$$\mathbf{D} = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} \tag{2.73}$$

Novamente, será considerado $k_x = k_y = k_z = 1$. As demais matrizes são:

$$\mathbf{m}^e = \frac{V}{20}\begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \tag{2.74}$$

$$\mathbf{g_x}^e = \frac{1}{24}\begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ b_1 & b_2 & b_3 & b_4 \\ b_1 & b_2 & b_3 & b_4 \\ b_1 & b_2 & b_3 & b_4 \end{bmatrix} \tag{2.75}$$

$$\mathbf{g_y}^e = \frac{1}{24}\begin{bmatrix} c_1 & c_2 & c_3 & c_4 \\ c_1 & c_2 & c_3 & c_4 \\ c_1 & c_2 & c_3 & c_4 \\ c_1 & c_2 & c_3 & c_4 \end{bmatrix} \tag{2.76}$$

$$\mathbf{g_z}^e = \frac{1}{24}\begin{bmatrix} d_1 & d_2 & d_3 & d_4 \\ d_1 & d_2 & d_3 & d_4 \\ d_1 & d_2 & d_3 & d_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \tag{2.77}$$

As entradas das matrizes que equivalem aos coeficientes dos polinômios interpo-

ladores, no caso tridimensional para a base escolhida, são dados por:

$$\begin{aligned} b_1 &= (y_2 - y_4)(z_3 - z_4) - (y_3 - y_4)(z_2 - z_4) \\ b_2 &= (y_3 - y_4)(z_1 - z_4) - (y_1 - y_4)(z_3 - z_4) \\ b_3 &= (y_1 - y_4)(z_1 - z_4) - (y_2 - y_4)(z_1 - z_4) \\ b_4 &= -(b_1 + b_2 + b_3) \end{aligned} \tag{2.78}$$

$$\begin{aligned} c_1 &= (z_2 - z_4)(x_3 - x_4) - (z_3 - z_4)(x_2 - x_4) \\ c_2 &= (z_3 - z_4)(x_1 - x_4) - (z_1 - z_4)(x_3 - x_4) \\ c_3 &= (z_1 - z_4)(x_1 - x_4) - (z_2 - z_4)(x_1 - x_4) \\ c_4 &= -(c_1 + c_2 + c_3) \end{aligned} \tag{2.79}$$

$$\begin{aligned} d_1 &= (x_2 - x_4)(y_3 - y_4) - (x_3 - x_4)(y_2 - y_4) \\ d_2 &= (x_3 - x_4)(y_1 - y_4) - (x_1 - x_4)(y_3 - y_4) \\ d_3 &= (x_1 - x_4)(y_1 - y_4) - (x_2 - x_4)(y_1 - y_4) \\ d_4 &= -(d_1 + d_2 + d_3) \end{aligned} \tag{2.80}$$

Montagem das matrizes - *Assembling*

Após obter as matrizes elementares, passa-se então para a fase de acoplamento global dos valores obtidos, em um processo conhecido como *Assembling* das matrizes para formação de suas equivalentes globais.

Nesse momento, utiliza-se a numeração dada aos elementos e nós para que seja possível identificar a posição relativa ao sistema global em que cada matriz elementar deve se encontrar [Figura 2.14]. Para tal, o método utiliza a chamada Matriz de Conectividade ou de Incidência (IEN).

Tal matriz é constituída de forma a conter, em cada linha, os nós que estão presentes em cada elemento, seguindo o esquema de numeração global pré-definido. Ou seja, sua dimensão é dada pela quantidade de elementos e número de nós em cada elemento. Uma matriz de uma malha de cem elementos quadriláteros, por

exemplo, possui quatro colunas e cem linhas.

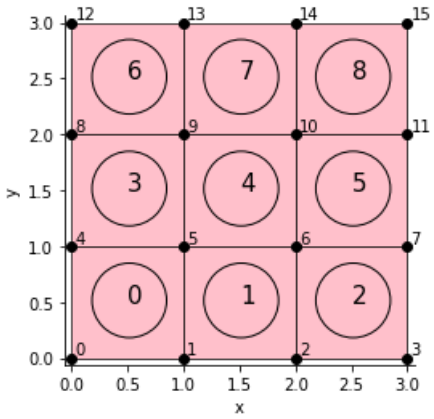


Figura 2.14: Exemplo de malha bidimensional numerada

Na malha acima, tem-se a seguinte IEN:

$$\text{IEN} = \begin{bmatrix} 0 & 1 & 5 & 4 \\ 1 & 2 & 6 & 5 \\ 2 & 3 & 7 & 6 \\ 4 & 5 & 9 & 8 \\ 5 & 6 & 10 & 9 \\ 6 & 7 & 11 & 10 \\ 8 & 9 & 13 & 12 \\ 9 & 10 & 14 & 13 \\ 10 & 11 & 15 & 14 \end{bmatrix} \tag{2.81}$$

Assim, nota-se no exemplo acima que, para o primeiro elemento, seus nós são equivalentes aos números 0, 1, 5 e 4; onde o elemento foi percorrido no sentido anti-horário. Portanto, a primeira linha da matriz elementar equivale a posição 0 na matriz global: sua primeira entrada está associada à posição 00; a segunda à posição 01; a terceiro à 05 e assim por diante, percorrendo cada elemento - e equivalentemente cada linha da Matriz de Conectividade.

Conclusão do Método

Após o *Assembling*, as matrizes globais estão formadas e é possível resolver o sistema linear encontrado, como o caso presente na equação 2.49.

O método concluí-se, então, com a análise dos resultados obtidos e sua inter-

pretação física. Sua aplicação é feita de forma iterativa, de forma que após gerada uma malha e observados os resultados, podem ser buscadas melhorias por meio de refino da malha. Nesse sentido, busca-se uma convergência dos valores encontrados quando há aumento do número de elementos.

Além disso, técnicas de pós-processamento e otimização da qualidade da malha e do modelo geométrico podem ser empregadas. O procedimento geral da solução de um modelo físico a partir de um método numérico, em especial no caso do Método de Elementos Finitos, é resumido por Bathe, em [4], pelo seguinte fluxograma em [Figura 2.15].

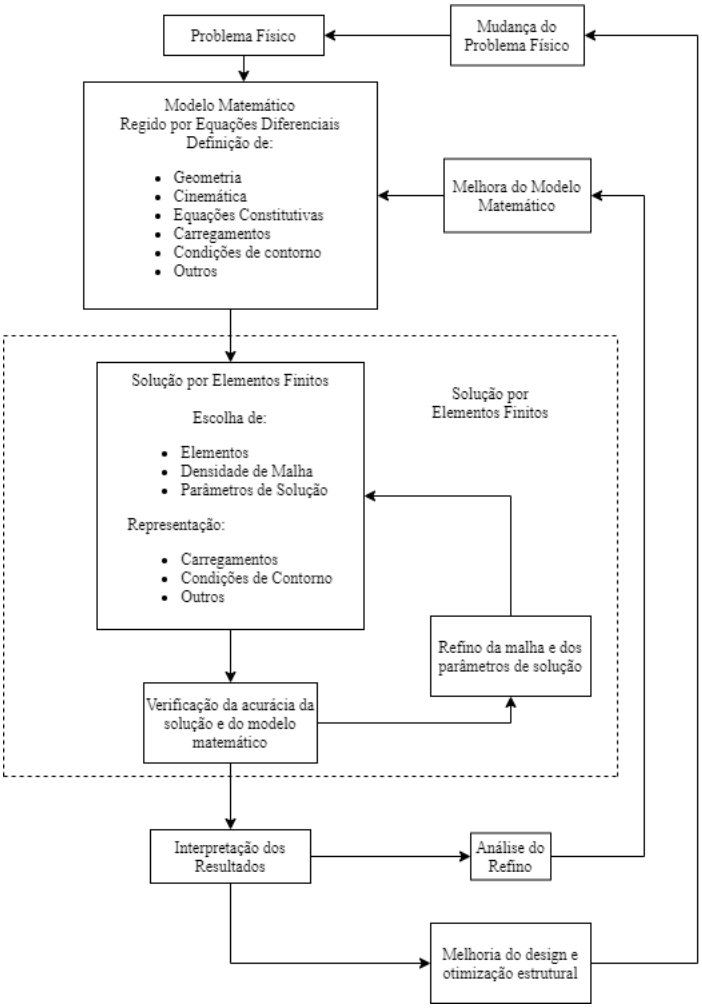


Figura 2.15: Fluxograma do MEF - Adaptado de [4]

2.2.3 Volumes Finitos

Um terceiro método numérico a ser apresentado é o Método de Volumes Finitos (MVF). Este é similar aos anteriores em sua essência, no sentido que demanda uma

discretização do domínio de interesse. Porém, esta é feita de forma a separá-lo em pedaços de volume (diga-se área, no caso 2D) e avaliar o valor médio da função que deseja-se obter.

Nesse sentido, a equação diferencial a ser solucionada é integrada nos volumes discretos. O MVF é dito, então, um método conservativo, em que se garante que as leis de conservação serão verificadas [40].

Usualmente, define-se uma malha de elementos, com vértices representando os nós, assim como é feito no MEF e representado em [Figura 2.16]; nesse caso, os volumes e elementos podem coincidir e toma-se o centroide do volume como sendo igual ao do elementos. No presente projeto, contudo, é de particular interesse definir volumes com centro nos nós e seus limites sendo dados conectando os centroides dos elementos originais. Um exemplo de tal discretização pode ser visto na imagem [Figura 2.17].

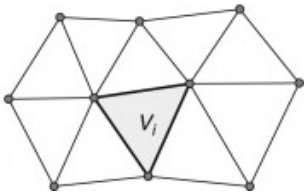


Figura 2.16: Volumes Finitos coincidentes com os elementos [5]

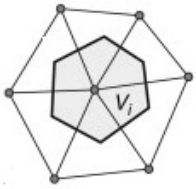


Figura 2.17: Volumes Finitos com centro nos nós [5]

Em alguns casos, tal procedimento é descrito como uma combinação entre o método de Elementos Finitos e Volumes Finitos, por empregar uma discretização muito similar ao primeiro, ainda que haja presença de um volume diferente em que as propriedades de interesse são avaliadas [10]. Também encontra-se o nome Método de Volumes Finitos Baseado em Elementos (EbFVM - *Element-based Finite Volume Method*) [41] [42].

2.3 Forças de Tensão Superficial

2.3.1 Histórico e a equação de Young-Laplace

O cálculo das forças de Tensão Superficial teve seu início séculos atrás. Na nona edição da Enciclopédia Britânica, James Clark Maxwell - imortalizado por seus trabalhos em eletromagnetismo - escreveu um texto referente à história do estudo de tais forças e dos fenômenos de capilaridade. Na décima edição, o texto foi revisado por outro célebre cientista: John W. Strutt - o Lorde Rayleigh. Em [43], o autor faz uma revisão crítica e um apanhamento histórico da segunda versão, em que destaca o desenvolvimento do estudo da tensão superficial e seus principais contribuidores.

Maxwell aponta Leonardo Da Vinci como a primeira pessoa a observar os efeitos de forças moleculares em fluidos e traduzi-los como uma forma de tensão devido à coesão entre as moléculas. Apesar disso, ainda seriam necessários alguns séculos para que uma teoria sólida sobre tais grandezas fosse estabelecida. São creditados nesse sentido Thomas Young e Pierre Simon Laplace por seus trabalhos, conduzidos de forma independente, no início do século XIX.

Pomeau destaca em [43], indo ao encontro do defendido em [44], que, apesar de terem obtido resultados similares, as publicações dos dois cientistas diferiam bastante no que tange seu conteúdo. Young apresentou, em seus artigos, uma abordagem majoritariamente teórica, em que fazia pouco uso de formulações matemáticas ou explicações diretas de suas ideias. Já Laplace seguia uma apresentação mais similar à artigos modernos, disponibilizando uma dedução extensa do equacionamento obtido.

Não obstante, destaca-se que ambos chegaram ao importante resultado que relaciona a tensão superficial com a curvatura média da superfície do fluido. Uma dedução seguindo os passos dados por Laplace está disponível em [43], contudo, atualmente há diversas modificações que simplificam o desenvolvimento. Essas em geral seguem duas abordagens: uma baseada no equilíbrio de forças e outra a partir da conservação de energia, como é usual em deduções de física mecânica.

Um exemplo pode ser encontrado em [6], em que a imagem abaixo é considerada [Figura 2.18]. Sendo **P** um ponto em uma superfície, traça-se uma curva que dista ρ do ponto e nela identifica-se os arcos **AB** e **CD**, parte das curvas na superfície

que correspondem às direções principais $\mathbf{e_1}$ e $\mathbf{e_2}$.

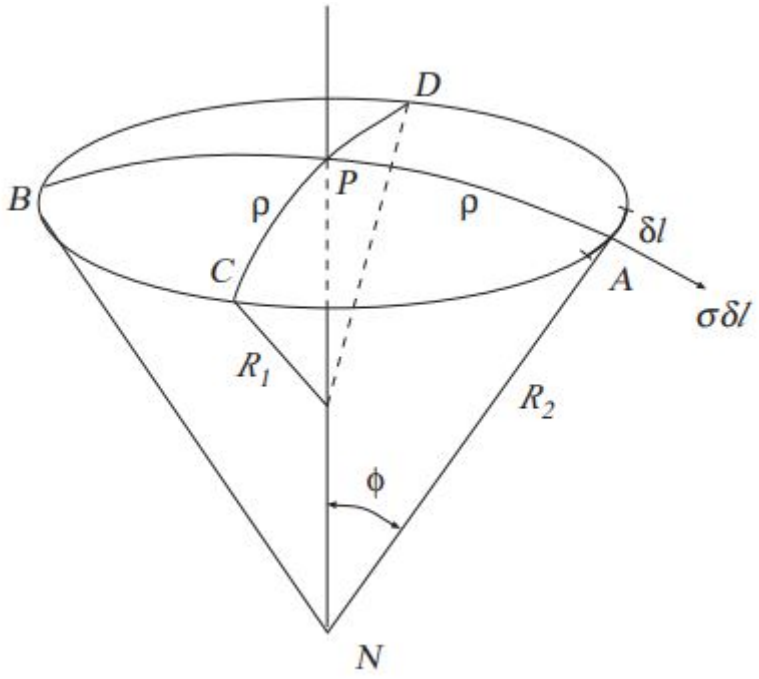


Figura 2.18: Representação esquemática da decomposição do campo de tensões em um elemento superficial infinitesimal [6]

N indica o encontro das direções normais das curvas com a normal da superfície. Dessa forma, pode-se obter as curvaturas normais a partir do ângulo ϕ, aqui sendo o complementar do ângulo dado na definição de Curvatura Normal [Figura 2.3]. Em se tratando das direções principais, os raios de curvatura representados R_1 e R_2 são justamente os inversos das curvaturas principais k_1 e k_2.

Assim, admitindo uma aproximação para pequenos ângulos, a força em um elemento δl na periferia em **A**, sob a ação de uma tensão superficial σ, é dada por:

$$\delta l \sigma \sin\phi = \delta l \sigma \frac{\rho}{R_1} \approx \delta l \sigma \phi \tag{2.82}$$

Considerando a força em elementos equivalentes em **A**, **B**, **C** e **D** e somando-os:

$$\delta F = 2\rho\sigma\delta l \left(\frac{1}{R_1} + \frac{1}{R_2}\right) \tag{2.83}$$

Sendo a curva obtida uma circunferência, pode-se integrar em um quarto de seu

perímetro para obter:

$$F = 2\rho\sigma\delta l \left(\frac{1}{R_1} + \frac{1}{R_2}\right) \times \frac{\pi\rho}{2} = \pi\rho^2\sigma \left(\frac{1}{R_1} + \frac{1}{R_2}\right) \tag{2.84}$$

Por outro lado, se existe uma diferença de pressão entre o interior e o exterior da superfície, a força equivalente será dada por seu produto com a área:

$$F \approx \Delta p \pi \rho^2 \tag{2.85}$$

O que implica na equação de Young-Laplace em sua forma tradicional, em literatura nacional também chamada Lei de Laplace:

$$\Delta p = \sigma \left(\frac{1}{R_1} + \frac{1}{R_2}\right) \tag{2.86}$$

Ou ainda:

$$\Delta p = \sigma \left(k_1 + k_2\right) = \sigma k \tag{2.87}$$

Outras deduções se apoiam na mudança de área equivalente à um aumento de energia na superfície, em que é explorada a seguinte equação, em que W representa a energia e A a área:

$$\delta W = \sigma \delta A \tag{2.88}$$

Detalhes são apresentados em [45], assim como deduções alternativas estão presentes em [6] e [46]. Como mencionado, a dedução como feita por Laplace está presente em [43].

A equação de Young-Laplace é a mais comum modelagem de tensão superficial, indicando sua relação com a diferença de pressão e a curvatura. Apesar disso, em aplicações mais modernas, outras equações são utilizadas para representar tal fenômeno, como é o caso da expressão de Kirkwood-Buff, que utiliza princípios de mecânica estatística. Outro exemplo é a equação de Triezenberg-Zwanzig [47].

2.3.2 *Continuum Surface Force* (CSF)

No contexto do presente projeto, contudo, é de interesse utilizar uma equação que seja mais adequada para o cálculo da tensão superficial especificamente em escoa-

mentos bifásicos, na interface entre os fluidos. O modelo mais utilizado nesse sentido foi proposto por Brackbill, Kother e Zemach em [48], chamado de *Continuum Surface Force* (CSF).

Baseando-se fortemente na equação de Young-Laplace, o modelo proposto busca adequar-se a diferentes métodos de descrição da interface no escoamento. Nesse sentido, a equação do CSF é aplicável em geometrias complexas, além de poder ser calculada em uma interface cujo formato varia com o tempo. Tem-se:

$$\mathbf{f} = \sigma k \, \mathbf{n} \, \delta_h \tag{2.89}$$

em que $\mathbf{f}$ representa a força de tensão superficial em um ponto da interface, σ é a tensão em si - como presente na equação [2.87] -, k é a curvatura média e $\mathbf{n}$ é o vetor normal no dado ponto. O termo δ_h representa uma função Delta de Dirac, que nesse caso assume valor 1 na interface e 0 fora, a qual possui espessura h. De fato, se fazemos com que h tenda a zero, nota-se a semelhança com a equação de Young-Laplace.

Destaca-se, portanto, que a força de tensão superficial não é interpretada como uma condição de contorno do escoamento; no contexto do CSF, esta assume a caracterização de uma força de volume que age na interface. Nesse sentido, no caso de uma bolha estática por exemplo, pela equação de Navier-Stokes, teremos $\nabla p = f$, em que ∇p é o gradiente de pressão [8], uma vez incluído o termo de tal força no equilíbrio de momento.

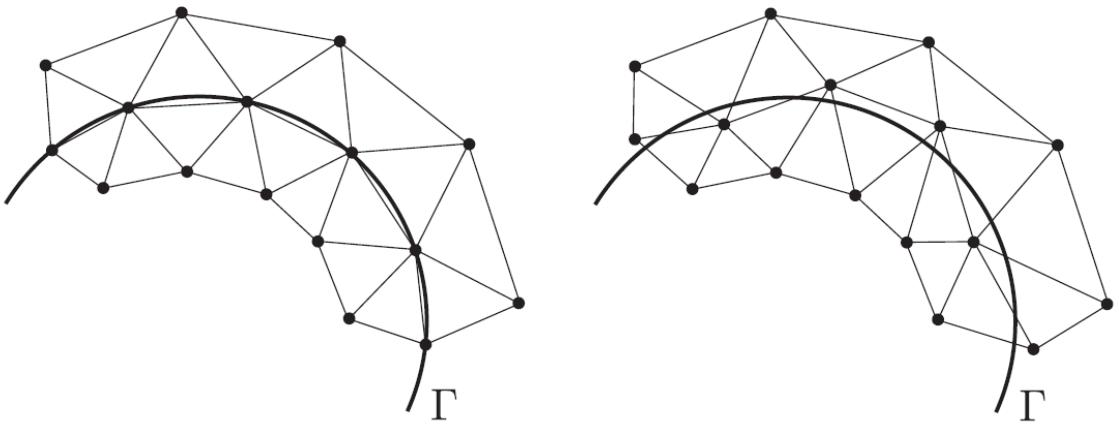
2.3.3 Modelagem da Interface em Escoamentos Bifásicos

Modelos que consigam capturar de forma satisfatória o comportamento de escoamentos bifásicos apresentam diversos desafios computacionais. Entre esses destacam-se a dificuldade de forçar a conservação de massa, momento e energia no sistema, de forma à obter uma solução robusta e realista [49]. Além disso, deve-se atentar à presença de erros numéricos devido aos métodos utilizados, assim como o aparecimento de correntes espúrias. Estas referem-se à velocidades de origem não-física que são encontradas em esquemas de discretização da equação de Navier-Stokes, como descrito em [50].

Nesse sentido, diversos métodos que tentam representar a interface de uma forma coerente foram propostos. Em geral, estes se dividem em dois grupos: *Interface Capturing* e *Interface Tracking*, ou ainda abordagem Euleriana e Lagrangiana, respectivamente.

De fato, tal nomenclatura está ligada com as usuais interpretações de um escoamento segundo cada abordagem. No caso Euleriano, a interface não faz parte da malha computacional e deve ser calculada a partir de uma função que a represente (*color function* [8]), de forma que é necessário uma equação adicional para a cinemática da interface. No caso Lagrangiano, contudo, a interface é descrita por elementos geométricos (que podem fazer parte da malha do fluido ou não) e deve verificar as mesmas equações que o escoamento como um todo. Nesse sentido, a abordagem Euleriana também recebe o nome de "desacoplada"ou de interface "difusa"e a Lagrangiana de "acoplada", com interface "nítida"[51].

Destaca-se que o primeiro caso é mais eficiente na descrição de alguns fenômenos como quebra e coalescência de bolhas [8] [52], porém demanda mais atenção no sentido computacional, uma vez que é necessária uma maior qualidade de malha para capturar a interface corretamente. Além disso, a função utilizada para descrevê-la pode ser uma fonte de erros numéricos consideráveis e de problemas na conservação de massa no sistema, o que não acontece na segunda abordagem. Não obstante, a interface difusa consegue modelar com mais facilidade topologias complexas.



(a) Malha Acoplada (b) Malha Desacoplada

Figura 2.19: Malhas das diferentes abordagens para modelagem da interface [7]

No caso da interpretação Euleriana, então, destacam-se os métodos de *Volume of Fluid* (VOF) e *Level-Set* (LS), apesar de existirem outros como o "*Phase-Field*"e

métodos híbridos. Em [49] é realizada uma completa revisão desses tipos de método, indicando o desafio de verificar as leis de conservação - particularmente do momento e massa - e, em conclusão, colocando o primeiro tipo como o mais promissor.

O VOF consiste em associar a cada elemento de malha um número entre zero e um que expressa o quanto de cada fluido está presente no volume do elemento [Figura 2.20]. Para tal, inicia-se com uma função H, como definida em [2.90], e calcula-se a posição da interface por meio de uma equação hiperbólica [2.91], integrando-a no tempo. Esta metodologia foi apresentada pela primeira vez em [53] e desde então recebeu diferentes tratamentos e aplicações.

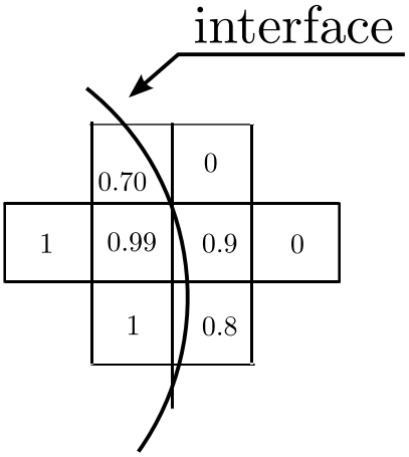


Figura 2.20: Malha no método VOF [8]

Em [54], por exemplo, é desenvolvido um método VOF para cálculos em duas e três dimensões, com resultados satisfatórios quando comparados com dados experimentais no movimento de bolhas e em fluidos estagnados. Já em [52], é proposta uma metodologia que busca diminuir a presença de correntes espúrias em escoamentos interfaciais, realizando uma comparação entre diferentes modelagens da tensão superficial - indicando uma incompatibilidade com o CSF.

Ainda assim, como acontece em [55], é proposto um método similar que se adéqua ao modelo de CSF, também verificando bons resultados com testes experimentais em coalescência de bolhas. Ademais, em [56], o mesmo modelo de tensão superficial é utilizado para propor um esquema de interpolação da pressão entre os fluidos a partir de uma representação da interface de forma difusa. Não obstante, deve-se destacar que o VOF apresenta problemas principalmente no cálculo de propriedades geométricas, como normal e curvatura.

$$H(\mathbf{x},t) = \begin{cases} 1 & \text{se } \mathbf{x} \text{ está no fluido interior à interface} \\ 0 & \text{se } \mathbf{x} \text{ está no fluido exterior à interface} \end{cases} \tag{2.90}$$

Um exemplo de equação para o cálculo da interface é dada em [49]:

$$V\frac{\partial C_k(t)}{\partial t} + \int_{\partial\Omega} (\mathbf{u}\cdot\mathbf{n})H(\mathbf{x},t)dS = \int_{\Omega} H\nabla\cdot\mathbf{u}dV \tag{2.91}$$

em que integra-se no volume Ω, para cada elemento de volume V. $\mathbf{u}$ é o vetor de velocidade do escoamento e $\mathbf{n}$ o vetor normal calculado na interface. C_k, por sua vez, representa a média da função de Heaviside no volume. Nota-se que, em um escoamento incompressível, pode-se anular o lado direito da equação.

Já no caso de métodos do tipo LS, sua origem é tradicionalmente devido à estudos em computação gráfica - como [57] -, sendo empregados pela primeira vez em modelagens de escoamentos bidimensionais em [58]. Estes baseiam-se na definição de uma função de distância com sinal $\phi(\mathbf{x},t)$ que é calculada a partir de uma equação de advecção, como em [2.92] [8].

$$\frac{\partial\phi}{\partial t} + \mathbf{u}\cdot\mathbf{n}\,|\nabla\phi| = 0 \tag{2.92}$$

A função ϕ é, geralmente, definida como a menor distância entre um ponto da malha dos fluidos e um ponto da interface; de forma que a interface é obtida quando esta se anula. Além disso, a tal distância é dado um sinal (positivo ou negativo) que indica a posição do nó em relação a interface - seguindo a mesma ideia da equação [2.90]. Deve-se observar, portanto, que é necessário reinicializar a interface em cada instante (ou pelo menos em uma quantidade significativa) para que a interface seja recalculada a partir da equação [2.92].

Partindo de [58], diversos trabalhos foram propostos que utilizam tal método, como é o caso de [59], que o apresente no caso de uma superfície livre em três dimensões, realizando diferentes testes de validação com resultados experimentais conhecidos.

Em [60], utiliza-se um método de elementos finitos à partir da discretização de GLS (*Galerkin Least Squares*) em um escoamento bifásico. Discute-se, em particu-

lar, a importância da reinicialização da interface a partir da equação de advecção para garantir estabilidade no método. Outro exemplo relevante é dado em [61], em que o modelo de CSF é utilizado com um esquema de Petrov-Galerkin para resolução da equação de Navier-Stokes, fornecendo comparações com resultados como a ascenção de bolhas.

Destaca-se que, em tal formulação, especial atenção deve ser dada à conservação de massa por possíveis problemas na resolução de [2.92] [58]. Não obstante, o cálculo de propriedades geométricas pode ser feito diretamente a partir da função ϕ, utilizando as relações abaixo. Nesse sentido, alguns métodos híbridos que utilizam tanto o LS quanto VOF foram propostos.

$$\mathbf{n} = \frac{\nabla \phi}{|\nabla \phi|} \tag{2.93}$$

$$k = \nabla \cdot \mathbf{n} \tag{2.94}$$

Já na abordagem Lagrangiana, destacam-se os métodos de "volume tracking"e "front tracking"[8], ainda que diversos outros existam. Em [62] é feito um apanhado das diferentes metodologias de abordagem Lagrangiana e são resumidos métodos usuais.

O "volume tracking"é um método, primeiramente proposto em [63], baseado na definição de marcadores e células no domínio dos fluidos que terão seu movimento estudado para definição da interface. Por esse motivo, também recebe o nome de MAC (Marker-and-Cell). Um exemplo está presente em [64], em que estuda-se o método em escoamentos multifásicos, apresentando diferentes resultados do método numérico.

Já os métodos de "front tracking"utilizam a definição de uma interface dada por elementos conectados (como uma malha de nós e retas, por exemplo). Estes são inseridos em um escoamento interpretado por uma abordagem Euleriana, como destacado em [8] e [65].

O método foi inicialmente proposto em [66], em que foi estudado o movimento de ondas. Em [67], foi utilizado o método de Galerkin, alinhado à função de distância definida como no método LS. Dessa forma, foram utilizadas as equações anteriores

([2.93], [2.94]) para calcular as propriedades geométricas a partir de uma função ϕ. Contudo, nesse caso não houve necessidade de reconstruir a interface utilizando uma equação de advecção, sendo isto feito pela interpretação Euleriana do movimento dos fluidos.

Além disso, foi utilizado o modelo de CSF, com uma modificação na função Delta de Dirac, que foi substituída pelo gradiente de uma função de Heaviside, similar àquela dada em [2.90]. Tem-se:

$$H_1(\mathbf{x}) = \begin{cases} 1 & \text{se } \mathbf{x} \text{ está no fluido interior à interface} \\ 0.5 & \text{se } \mathbf{x} \text{ está na interface} \\ 0 & \text{se } \mathbf{x} \text{ está no fluido exterior à interface} \end{cases} \tag{2.95}$$

Assim, também pode-se utilizar a função acima para calcular a distribuição de propriedades, como a viscosidade [8]:

$$\Phi = \Phi_1 H + \Phi_0(1 - H) \tag{2.96}$$

em que Φ_1 e Φ_0 são as propriedades em cada fase do escoamento.

Outro exemplo de utilização da abordagem Lagrangiana para a interface está presente em [8], em que utilizam-se métodos numéricos baseados nos conceitos mencionados de geometria diferencial para o cálculo de curvatura e vetor normal, os quais serão explorados no próximo capítulo. A partir disso, uma descrição Euleriana-Lagrangiana do escoamento é utilizada (ALE - *Arbitrary Lagrangian Eulerian*), sendo apresentadas diversas validações do código numérico com resultados conhecidos.

Nesse sentido, destaca-se que no presente projeto não serão apresentadas soluções da equação de Navier-Stokes e assim aplicação das diferentes interpretações no que tange o movimento dos fluidos e sua interface. Serão explorados os métodos acima descritos exclusivamente no cálculo do termo de força de tensão superficial e representação da interface. A seguir são apresentadas, com mais profundidade, as metodologias utilizadas em tal propósito.

Capítulo 3

Metodologia

3.1 Cálculo da Curvatura Média

Como explicitado nas equações Young-Laplace e do CSF, as forças de tensão superficial estão intimamente ligadas à propriedade de curvatura média da forma geométrica do fluido em questão.

Nesse sentido, o primeiro passo para o cálculo das forças será o desenvolvimento de um método capaz de obter a curvatura média, tanto em casos bidimensional como em situações tridimensionais.

Serão propostas, assim, duas formas de obter a propriedade em questão em cada caso: uma metodologia baseada na ideia do Método de Diferenças Finitas e nas equações de Frenet [2.15] e um esquema baseado no Método de Elementos Finitos e na discretização do operador Laplace-Beltrami.

3.1.1 Curvatura 2D - Equações de Frenet

Iniciando pela configuração bidimensional, ao tomar a primeira igualdade do sistema de Equações de Frenet, podemos aplicar a ideia do método de Euler para obter a seguinte formulação:

$$k_{i+1}\mathbf{n_{i+1}} = \frac{\mathbf{t_{i+1}} - \mathbf{t_i}}{\Delta s_{i+1}} \tag{3.1}$$

Destaca-se que na igualdade acima, há uma operação entre vetores, diferentemente do foi exemplificado para o método de Diferenças Finitas. Não obstante, a

equação diferencial foi discretizada seguindo o mesmo esquema.

Nesse sentido, curva é discretizada em uma quantidade finita de pontos x_i, com i variando entre 1 e o total de pontos, seguindo o discutido na seção de geometria diferencial discreta. Os vetores tangentes são obtidos, então, como sendo exatamente aqueles dados pela direção de cada segmento de reta que une os pontos considerados na discretização [Figura 3.1]. Dessa forma, uma vez que o vetor normal é unitário, a curvatura pode ser obtida como a norma euclidiana da expressão acima.

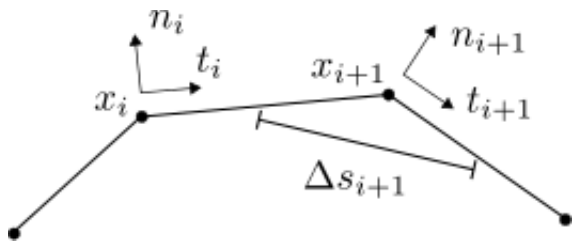


Figura 3.1: Vetores tangentes e normais na curva discretizada

Em relação ao valor de Δs_i na equação, deve-se considerar que este é relativo à distância entre dois pontos no domínio discreto, de forma que não fica tão evidente qual quantidade a ser utilizada se as distâncias não são uniformes. Nesse sentido, abordagens diferentes são propostas: utilizar o passo anterior ou o posterior; a média de cada desses valores; a distância entre os pontos médios de cada segmento ou aproximar os três pontos vizinhos por um arco de circunferência e usar seu comprimento. Como realizado em [8], será utilizado o esquema com os pontos médios [Figura 3.1].

Deve-se observar que no esquema apresentado não há diretamente uma condição de contorno como mencionado para resolução de equações diferenciais a partir do MDF. No entando, está sendo levado em consideração o fato de que o domínio é uma curva fechada e, portanto, deve existir um acoplamento entre o primeiro e último pontos considerados; o que é suficiente para fazer com que o problema e sua solução sejam bem postas no sentido de seja possível encontrar os vetores normal e tangente em cada ponto.

Para obter o sinal da curvatura, porém, ainda é necessário definir uma orientação na curva. Tal procedimento pode ser feito seguindo o explicado anteriormente e ilustrado em [Figura 2.7]. Assim, como previsto na discretização [3.1], a curvatura tem a direção calculada pela diferença dos vetores tangentes que, comparada com a

orientação definida, irá fornecer o sinal desejado.

3.1.2 Curvatura 3D - Volumes Finitos

No caso de superfícies tridimensionais, não se pode utilizar o esquema desenvolvido acima, uma vez que em cada ponto da superfície há uma quantidade infinita de curvas que o contêm. Ainda que discretizemos a superfície e obtenhamos uma quantidade finita de retas que passam por cada ponto, não é diretamente claro como definir as curvas a partir disso.

Portanto, para o cálculo de curvatura em três dimensões, inicialmente, será utilizado um método que surge da interpretação física da curvatura. Tal propriedade é relacionada com a pressão devido a uma força que age em cada elemento da superfície discreta, como sugerido em [8] e [68].

Consideremos, para tal, um nó i na superfície e seus nós vizinhos diretos ("1-ring neighbors"), assim como os elementos que os contêm. O conjunto desse elementos triangulares será referenciado como $\mathcal{N}_1^i$ [Figura 3.2]. Cada elemento contribui com uma força que está agindo sobre o nó principal, de forma que a curvatura é dada pela razão entre a soma das contribuições e uma área a ser definida como se segue.

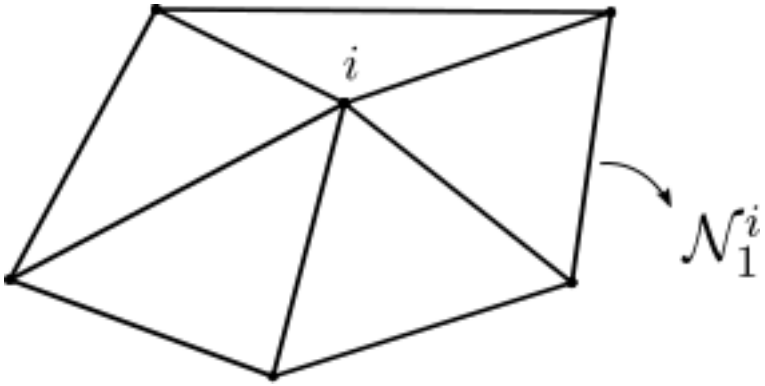


Figura 3.2: Vizinhos diretos de um nó

Cálculo da Área

Seja C o circuncentro do triângulo elementar - ou seja o encontro das mediatrizes de seus lados - e $\mathbf{t_1}$ e $\mathbf{t_2}$ os vetores unitários tangentes definidos pelos lados do elementos como mostrado em [Figura 3.3]. Ligando os pontos médios dos lados com C, define-se m_1 e m_2 e, assim, obtemos o que será chamado de área circuncêntrica

do nó no triângulo.

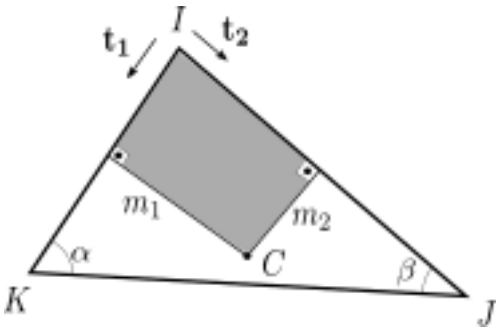


Figura 3.3: Vetores tangentes e área Circuncêntrica do triângulo

Uma equação para o valor de tal área é dada pela fórmula abaixo, que é utilizada em [10], e facilmente obtida pela expressão trigonométrica da tangente em um triângulo retângulo e notando que, pela definição do circuncentro, m_1 e m_2 são perpendiculares aos respectivos lados mostrados acima:

$$A_{circ} = \frac{1}{8}(\overline{IJ}\cot\alpha + \overline{IK}\cot\beta) \tag{3.2}$$

Não obstante, utilizando geometria analítica, pode-se obter as coordenadas do circuncentro em função dos lados do triângulo, fazendo com que o cálculo da área seja direto por meio de produtos vetoriais. Em particular, tal forma de proceder é mais conveniente quando tem-se as coordenadas dos vértices do triângulo:

$$C = \frac{(a^2(b^2+c^2-a^2)I + b^2(a^2+c^2-b^2)J + c^2(a^2+b^2-c^2)K}{(a^2(b^2+c^2-a^2) + b^2(a^2+c^2-b^2) + c^2(a^2+b^2-c^2)} \tag{3.3}$$

em que a, b e c são os lados do triângulo e I, J e K os respectivos vértices opostos em termos de suas coordenadas cartesianas.

Logo:

$$A_{circ} = \frac{1}{2}\left(\left\|\frac{\overline{IK}}{2}\mathbf{t_1} \wedge \mathbf{m_1}\right\| + \left\|\frac{\overline{IJ}}{2}\mathbf{t_2} \wedge \mathbf{m_2}\right\|\right) \tag{3.4}$$

Dessa forma, para cada nó, pode-se avaliar a área referente à cada elemento do conjunto $\mathcal{N}_1^i$ e, somando, define-se a região de Voronoi do nó:

$$\mathcal{A}_{Voronoi} = \sum_{e \in \mathcal{N}_1^i} A_{circ} \tag{3.5}$$

Tal nomenclatura é devido à analogia com o diagrama de Voronoi, em que, dado um conjunto de pontos em uma região, se calcula a separação necessária a fim de que cada polígono contenha um dos pontos e, além disso, as regiões formadas sejam aquelas mais próximas de cada ponto [Figura 3.4]. De fato, o circuncentro é o ponto equidistante a cada vértice do triângulo, de forma que as áreas circuncêntricas expressam justamente a região dos pontos mais próximos à seu respectivo vértice. Assim, também é comum a analogia com a triangulação de Delaunay - diz-se que o Diagrama de Voronoi é seu dual.

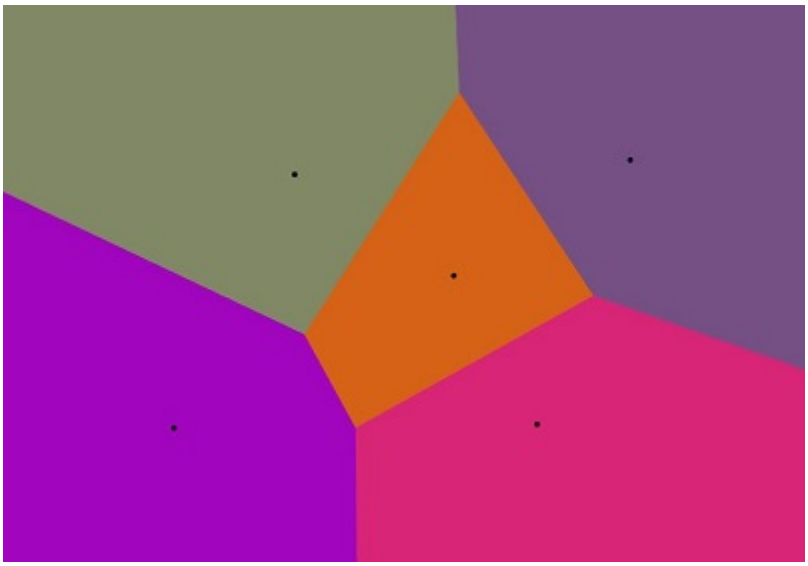


Figura 3.4: Diagrama de Voronoi gerado em [9]

Contudo, deve ser observado que nem todo triângulo possui um circuncentro interior, de forma que é necessária uma correção no caso em que o triângulo possui um ângulo obtuso. Em [10] sugere-se utilizar um quarto da área total nos ângulos agudos e metade no ângulo obtuso no casos desses triângulos. A região assim calculada é uma modificação da região de Voronoi, que em [10] recebe o nome de $\mathcal{A}_{mixed}$ [Algoritmo 1].

No presente trabalho, a região $\mathcal{A}_{mixed}$ será comparada com outras áreas propostas para o cálculo da curvatura. [8] sugere o uso da área baricêntrica, isto é, substitui-se o circuncentro na formulação acima por este outro ponto notável, de forma que a área equivale a um terço da área total do triângulo. Além disso, serão explorados os resultados sem a modificação de $\mathcal{A}_{Voronoi}$ para $\mathcal{A}_{mixed}$.

Algoritmo 1 Cálculo de A_{mixed} [10]

$A_{mixed} \leftarrow 0$
for e in $\mathcal{N}_i^1$ **do**
  A_{circ} ▷ Área do elemento e
  if e é triângulo acutângulo **then**
    $A_{mixed} \leftarrow A_{mixed} + A_{circ}$
  else
    if e é obtuso em i **then**
      $A_{mixed} \leftarrow A_{mixed} + A/2$
    else e não é obtuso em i
      $A_{mixed} \leftarrow A_{mixed} + A/4$
    end if
  end if
end for

Cálculo da Força de curvatura

Em [10], Meyer et al. utilizam-se da seguinte equação para o cálculo da curvatura:

$$k\mathbf{n} = \frac{1}{2\mathcal{A}_{mixed}} \sum_{e \in \mathcal{N}_1^i} (cot\alpha_{ij} + cot\beta_{ij}) \left\| (\mathbf{x}_i - \mathbf{x}_j) \right\|^2 \tag{3.6}$$

em que os ângulos referidos são mostrados na figura a seguir [Figura 3.5], assim como os nós $\mathbf{x_i}$ e $\mathbf{x_j}$.

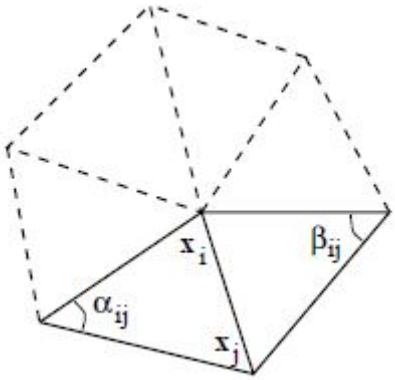


Figura 3.5: Ângulos e vértices na fórmula da cotangente [10]

Dessa forma, mesmo que a área acima referida seja calculada, a derivação da força que se segue não é utilizada.

No presente trabalho, porém, será seguido o proposto em [8]. Para tal, basta calcular o produto entre $\mathbf{t_1}$ e m_1, assim como entre $\mathbf{t_2}$ e m_2 e somá-los [Figura 3.3]. Isso é equivalente à computar duas força distribuídas, de forma que a interpretação

sugerida é que cada vetor tangente exerce uma força no vértice que deve ser integrada nos segmentos mencionados.

Uma forma mais simples, contudo, pode ser obtida observando que o resultado deve ser o mesmo quando toma-se o produto entre a base média do triângulo d e o vetor unitário $\mathbf{t_n}$ que é perpendicular a ela [Figura 3.6], em decorrência do Teorema de Stokes. Tal vetor pode ser obtido ortogonalizando um dos vetores tangentes.

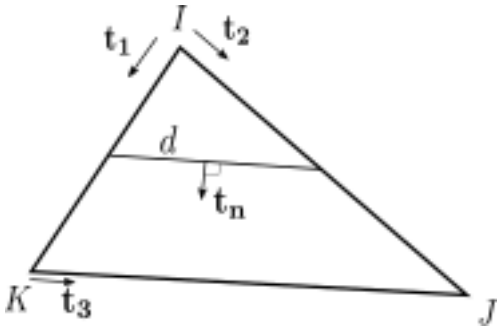


Figura 3.6: Vetor $\mathbf{t_n}$ em 3D

Ou seja:

$$\mathbf{F} = \mathbf{t_1} m_1 + \mathbf{t_2} m_2 = \mathbf{t_n} d \tag{3.7}$$

Em que, pode se calcular $\mathbf{t_n}$ como:

$$\mathbf{t_n} = \frac{\mathbf{t_3} \wedge (\mathbf{t_1} \wedge \mathbf{t_3})}{\|\mathbf{t_3} \wedge (\mathbf{t_1} \wedge \mathbf{t_3})\|} \tag{3.8}$$

Obtenção da Curvatura

Por fim, a curvatura em um nó da superfície discreta é dada pela seguinte expressão:

$$k\mathbf{n} = \frac{\sum_{e \in \mathcal{N}_1^i} \mathbf{F}}{\mathcal{A}_{mixed}} \tag{3.9}$$

onde são somadas as contribuições de $\mathbf{t_n}$ para cada elemento em $\mathcal{N}_1^i$ e tomada a razão com a região definida anteriormente.

Ademais, assim como no caso em 2D, é necessário definir uma orientação adequada para que o sinal da curvatura seja obtido. O vetor normal em uma superfície discreta será obtido pela soma dos produtos vetoriais dos vetores tangentes em cada elemento, seguindo o esquema referido no Capítulo 2 e ilustrado em [Figura 2.8].

Comentário sobre a Implementação do Código Numérico

A forma com que foi desenvolvido o cálculo acima, indica que é necessário conhecer os vizinhos diretos em cada nó, para que seja possível computar a contribuição de cada um. Além disso, seguir a fórmula da cotangente [3.6], faz com que seja necessário conhecer como os nós vizinhos se ordenam, de forma que seja possível sempre utilizar os ângulos corretamente. Dessa forma, pode-se calcular a curvatura pelo seguinte loop [Algoritmo 2]:

Algoritmo 2 Cálculo da curvatura com laço nos nós

```
for i in nós do
    K                                   ▷ Vetor com a curvatura em cada nó
    A ← 0
    F ← 0
    Obter os nós vizinhos ordenados a partir da IEN
    for j in nós vizinhos do
        Calcular contribuição de área A_j
        Calcular contribuição de força F_j
        A ← A + A_j
        A ← F + F_j
    end for
    K[i] ← F/A
end for
```

Contudo, para tal é necessário percorrer a matriz de incidência (IEN) da malha da superfície e encontrar os nós vizinhos, assim como ordená-los, o que pode ser custoso computacionalmente. Nesse sentido, será apresentado também o seguinte loop, baseado naquele empregado usualmente no Método de Elementos Finitos [Algoritmo 3]. Neste é necessário criar matrizes de Área e Força que tem o mesmo formato da IEN e, assim, calcular a curvatura somando a contribuição de cada nó em cada elemento.

3.1.3 Curvatura 2D e 3D - Elementos Finitos

As metodologias acima descritas utilizam de alguma forma a ideia de divisão de domínio contínuo em elementos finitos. Apesar disso, recorrem a outras ferramentas como uma aproximação por diferenças e/ou à utilização de vetores tangentes para indicar a ação de uma força. O método que se segue, por outro lado, emprega diretamente os passos do MEF, realizando a transformação do operador Laplace-

Algoritmo 3 Cálculo da curvatura com laço nos elementos

K ▷ Vetor com a curvatura em cada nó
A ▷ Matriz com a contribuição de área de cada nó por elemento - formato da IEN
F ▷ Matriz com a contribuição de força de cada nó por elemento - formato da IEN
for e in range(elementos) **do**
    Calcular contribuição de área devido a cada vértice do elemento $[A_i, A_j, A_k]$
    Calcular contribuição de força devido a cada vértice do elemento $[F_i, F_j, F_k]$
    for j_{local} in $[1, 2, 3]$ **do**
        $A[e, j_{local}] \leftarrow [A_i, A_j, A_k]$
        $F[e, j_{local}] \leftarrow [F_i, F_j, F_k]$
    end for
end for
for a in IEN **do**
    [i,j] ▷ Posição do elemento a na IEN
    A_f ▷ Vetor com área referente à cada nó
    F_f ▷ Vetor com força referente à cada nó
    $A_f[a] \leftarrow A_f[a] + A[i, j]$
    $F_f[a] \leftarrow F_f[a] + F[i, j]$
end for
F_f ▷ Vetor com curvatura em cada nó
$K_f \leftarrow A_f / F_f$ ▷ Divisão elemento por elemento

Beltrami ao que seria equivalente à sua "forma fraca".

Segundo a equação [2.24], pode-se aplicar o operador nas coordenadas dos pontos da superfície para, assim, obter a curvatura em cada um. Nesse sentido, como deseja-se trabalhar com um domínio discreto, será empregada uma forma similar que advém da interpretação do operador segundo a forma fraca explorada no MEF. Com isso, será possível escrevê-lo em uma forma matricial - que funciona como uma matriz de rigidez - e aplicá-la nos vetores de coordenadas.

Para tal, seja $\mathbf{w}$ a função peso vetorial. Multiplicando e integrando na superfície:

$$\int_\Gamma \nabla_s^2 \mathbf{x} \cdot \mathbf{w} \, d\Gamma = -\int_\Gamma \nabla_s \mathbf{x} : \nabla_s \mathbf{w}^T \, d\Gamma + \int_{\partial\Gamma} (\nabla_s \mathbf{x} \cdot \mathbf{w}) \cdot \mathbf{n} \, dC \tag{3.10}$$

em que foi considerada a possibilidade de existir um contorno. Para o contexto do presente trabalho, tal configuração não será admitida. Uma derivação completa da forma fraca, para um campo escalar, em que pode existir um contorno genérico é descrita em [69]. Logo, considerando uma discretização da função peso:

$$\int_\Gamma \nabla_s^2 \mathbf{x} \cdot \mathbf{w_j} \, d\Gamma = -\int_\Gamma \nabla_s \mathbf{x} : \nabla_s \mathbf{w_j}^T \, d\Gamma = -\mathbf{K_\Gamma} \tag{3.11}$$

Fica, dessa forma, clara a semelhança entre o que foi desenvolvido e a matriz de rigidez utilizada no método de Elementos Finitos, como mencionado. Por isso, para obter o valor da curvatura, serão utilizadas as mesmas matrizes elementares que foram deduzidas na seção referente ao método, com atenção somente no referencial que está sendo empregado. Isto é, no contexto tridimensional será utilizado um elemento triangular linear e a base de funções teste seguirá similarmente.

Dessa forma, escreve-se:

$$\mathbf{K_\Gamma x} = -\mathbf{F} \tag{3.12}$$

$\mathbf{F}$ acima é equivalente à força calculada a partir dos vetores tangentes nos métodos anteriores, de forma que basta dividir pela mesma área para que se obtenha o valor de uma tensão. A tensão assim deduzida é exatamente a curvatura em um dado ponto. Vale destacar que tal área é necessária pois trata-se de uma superfície discreta e, assim, aplicar o operador diretamente na função coordenada como previsto na equação [2.24] retornaria um resultado incorreto. Portanto:

$$k\mathbf{n} = \frac{-\mathbf{F}}{\mathcal{A}_{mixed}} \tag{3.13}$$

Assim como foi explicitado para o caso 3D acima, a mesma metodologia pode ser empregada em duas dimensões em uma curva discretizada por segmentos de reta [Figura 2.4] e ao invés da área ser utilizado o valor ΔS. Ainda no caso tridimensional, contudo, deve-se levar em conta que os coeficientes matrizes elementares apresentadas em [2.59] são aplicáveis no plano, mas devem ser levemente modificadas para elementos em $\mathbb{R}^3$. Assim, é necessário definir uma base local e empregar uma transformação como se segue [Figura 3.7]:

$$\xi = \frac{\mathbf{x_j} - \mathbf{x_k}}{\|\mathbf{x_j} - \mathbf{x_k}\|} \tag{3.14}$$

$$\eta = \frac{[(\mathbf{x_j} - \mathbf{x_k}) \wedge (\mathbf{x_i} - \mathbf{x_k})] \wedge (\mathbf{x_j} - \mathbf{x_k})}{\|[(\mathbf{x_j} - \mathbf{x_k}) \wedge (\mathbf{x_i} - \mathbf{x_k})] \wedge (\mathbf{x_j} - \mathbf{x_k})\|} \tag{3.15}$$

em que $\mathbf{x_i}$, $\mathbf{x_j}$ e $\mathbf{x_z}$ são as coordenadas cartesianas dos vértices do triângulo, como ilustrado. A base assim definida constitui um base ortonormal local do elemento.

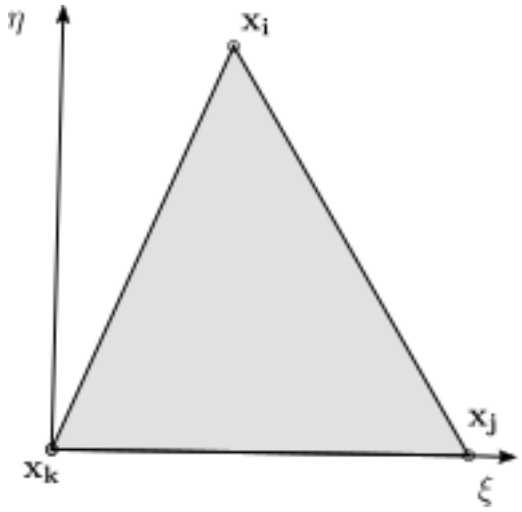


Figura 3.7: Base local no elemento triangular tridimensional

Portanto, os coeficientes destacados na equação 2.59, no caso de um elemento de superfície no espaço 3D, devem ser calculados como:

$$\begin{aligned} b_i &= \text{proj}_\eta(\mathbf{x_i} - \mathbf{x_j}) = \eta \cdot (\mathbf{x_i} - \mathbf{x_j}) \\ b_j &= \text{proj}_\eta(\mathbf{x_i} - \mathbf{x_j}) = \eta \cdot (\mathbf{x_k} - \mathbf{x_i}) \\ b_k &= \text{proj}_\eta(\mathbf{x_i} - \mathbf{x_j}) = \eta \cdot (\mathbf{x_j} - \mathbf{x_k}) \end{aligned} \tag{3.16}$$

$$\begin{aligned} c_i &= \text{proj}_\xi(\mathbf{x_i} - \mathbf{x_j}) = \xi \cdot (\mathbf{x_i} - \mathbf{x_j}) \\ c_j &= \text{proj}_\xi(\mathbf{x_i} - \mathbf{x_j}) = \xi \cdot (\mathbf{x_k} - \mathbf{x_i}) \\ c_k &= \text{proj}_\xi(\mathbf{x_i} - \mathbf{x_j}) = \xi \cdot (\mathbf{x_j} - \mathbf{x_k}) \end{aligned} \tag{3.17}$$

Assim, a matriz $\mathbf{K}_\Gamma$ pode ser obtida como explicitado na igualdade 2.60, utilizando as equações acima. No caso dos elementos de reta no plano, não há necessidade de modificar a base utilizada, podendo ser utilizada diretamente a equação 2.53.

Portanto, no presente projeto, a curvatura também será calculada seguindo os passos acima explicados. De fato, a formulação aqui desenvolvida é equivalente às anteriores, em que a força que se obtinha é igual a calcular a forma discretizada do operador Laplace-Beltrami aplicado na parametrização $\mathbf{x}$. No Capítulo 4 serão comparados os métodos propostos, mas destaca-se que a diferença existente entre o apresentado nessa seção e nas anteriores é puramente computacional, sendo equiva-

lentes de um ponto de vista matemático.

3.2 Cálculo das Forças de Tensão Superficial

Como explicado, diferentes abordagens são utilizadas para descrever a interface entre os fluidos em um escoamento bifásico. Serão propostas então, a seguir, duas metodologias distintas baseadas nas principais abordagens descritas. Em ambos os casos a força de tensão superficial será calculada a partir do modelo de CSF [2.89].

3.2.1 Abordagem Lagrangiana

Para aplicação de uma abordagem Lagrangiana, será considerada uma disposição em que existem duas malha do domínio físico: a malha exterior, de um dos fluidos e a malha interior, referente ao outro. Assim, o contato entre as duas malhas define a interface, portanto formada por objetos geométricos bem definidos e de espessura nula [Figura 3.8].

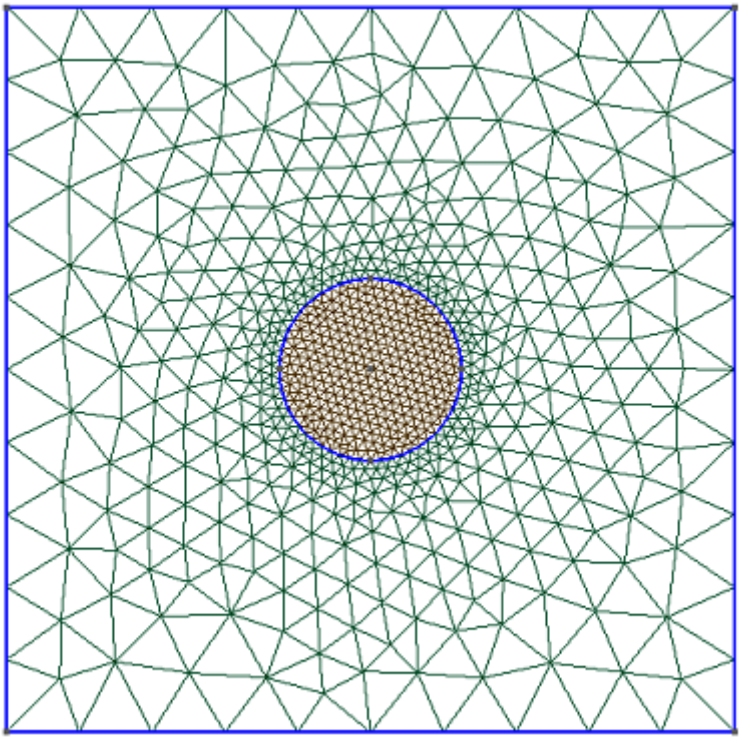


Figura 3.8: Malha Acoplada 2D

Logo, a malha gerada pelo Gmsh pode ser importada no código computacional em uma matriz de conectividade única, mas sendo possível distinguir à que região

pertence cada nó. Para obtenção da tensão superficial a partir do CSF, é necessário calcular a curvatura nos nós, assim como definir corretamente a função Delta de Dirac a ser utilizada.

Nesse sentido, a curvatura na interface pode ser obtida pelos métodos mencionados, bastando, então, extrapolá-la para os demais nós. Isso será feito designando o valor da curvatura como sendo igual ao valor no ponto mais próximo da interface, calculado pela distância euclidiana.

Para a função Delta será utilizada, contudo, sua expressão como o gradiente de uma função de Heaviside, como definida em [2.95]. Portanto, é possível obter uma formulação fraca para a equação do CSF [2.89]. Fazendo um paralelo com u no exemplo dado para o MEF, sendo f a função de interesse, tem-se:

$$\mathbf{Mf} = \Sigma\, \mathbf{G}H \tag{3.18}$$

em que $\mathbf{M}$ e $\mathbf{G}$ são a matriz de massa e matriz gradiente como explicitadas na seção do Método de Elementos Finitos, atentando-se ao tipo de elemento utilizado e sua dimensão. Σ é uma matriz diagonal que em cada entrada possui o produto σk referente à cada nó.

3.2.2 Abordagem Euleriana

Na segunda abordagem, do tipo Euleriana, é criada uma malha única que representa ambos os fluidos. Além disso, há também a malha inicial da interface, que possui uma dimensão a menos [Figura 3.9] e serve como condição inicial nas equações diferenciais da abordagem Euleriana. Nesse caso, então, são utilizadas duas matrizes de conectividade (IEN) com número de colunas diferentes, uma para cada uma das malhas citadas.

Em [Figura 3.9], esta representada uma malha desacoplada bidimensional, em que as linhas cheias em azul escuro representam o contorno do domínio e as de tom mais claro, a interface. Como é possível notar, esta atravessa elementos da malha triangular do domínio.

Assim como na abordagem Lagrangiana, a curvatura em cada ponto da malha será obtida primeiro computando-a na interface inicial, utilizando os métodos de-

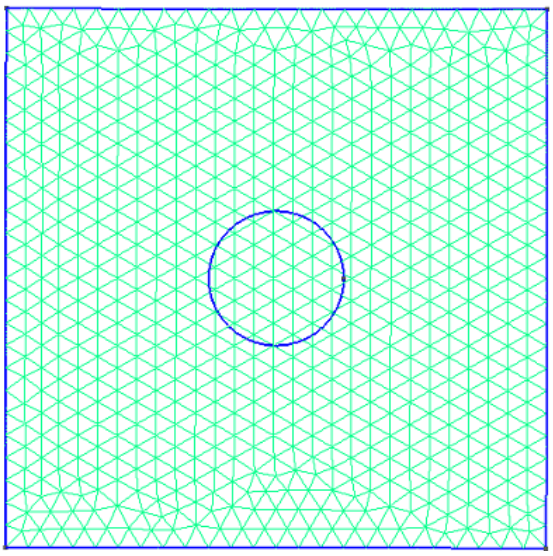


Figura 3.9: Malha Desacoplada bidimensional

senvolvidos anteriormente. Assim, será designado o valor de curvatura para cada nó como sendo a curvatura do nó mais próximo dessa interface. Similarmente, a força de tensão superficial será discretizada como expresso em [3.18].

No caso da abordagem Euleriana, contudo, recomenda-se a utilização de modificações na função de Heaviside utilizada para que haja mais suavidade na interface obtida a cada momento. Em [8] são apontadas duas possibilidade para a suavização:

$$H_1^\epsilon(\phi) = \begin{cases} 0 & \text{se } \phi < -\epsilon \\ \frac{1}{2}\left[1 + \frac{\phi}{\epsilon} + \frac{1}{\pi}\sin(\pi\phi/\epsilon)\right] & \text{se } -\epsilon \leq \phi \leq \epsilon \\ 1 & \text{se } \phi > \epsilon \end{cases} \tag{3.19}$$

$$H_2^\epsilon(\phi) = \begin{cases} 0 & \text{se } \phi < -\epsilon \\ \frac{1}{2} + \frac{1}{32}\left(45(\phi/\epsilon) - 50(\phi/\epsilon)^3 + 21(\phi/\epsilon)^5\right) & \text{se } -\epsilon \leq \phi \leq \epsilon \\ 1 & \text{se } \phi > \epsilon \end{cases} \tag{3.20}$$

Em que ϕ representa uma função de distância com sinal, como mencionado no capítulo anterior e ϵ é uma tolerância que define a espessura da interface. Seu valor é definido relativamente à malha utilizada, sendo, em geral, entre o tamanho do elemento de malha e o dobro deste número. Lembrando, também, que a função ϕ é dependente do tempo e deve-se resolver a equação [2.92] para obter uma nova

interface a cada instante. Para condição incial da equação diferencial, pode ser utilizada uma malha como mostrado em [Figura 3.9].

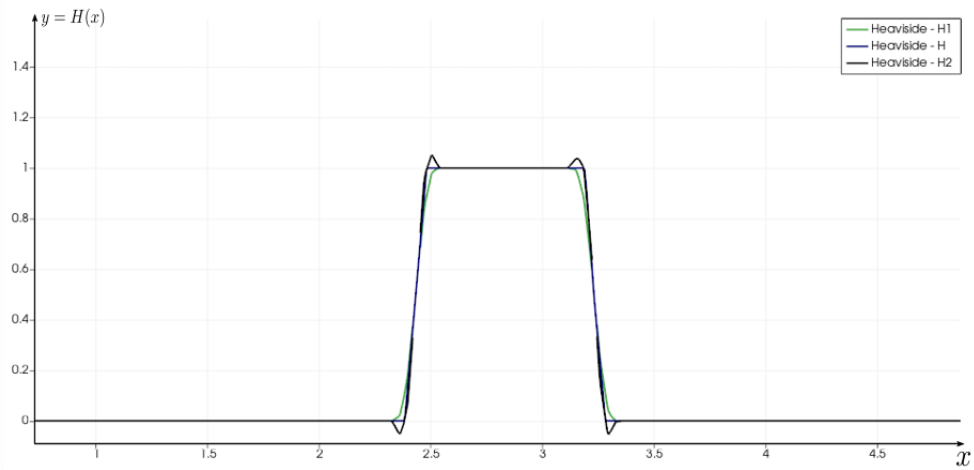


Figura 3.10: Comparação entre funções de Heaviside

Deve-se observar, entretanto, que nesse caso não há a informação direta do *software* de geração de malha da posição de cada nó em relação à interface, como só existe uma malha dos fluidos. Assim, para definir corretamente a função distância com sinal, pode-se utilizar um esquema para identificar se o nó em questão é interior ou exterior à interface a partir dos vetores normais da orientação nela definida.

Tal procedimento é feito calculando o vetor distância que está relacionado a distância mínima entre o nó da malha dos fluidos e o nó da interface - o que pode ser feito simultaneamente ao cálculo de tal distância. Assim, toma-se o produto escalar entre esse vetor e a normal no nó da interface; o sinal do número obtido define a posição desejada. Evidentemente, se o valor encontrado é zero, o nó dos fluidos também pertence à interface. Destaca-se, além disso, que os requeridos vetores normais já foram previamente calculados para obtenção da curvatura. O algoritmo utilizado está destacado no [Algoritmo 4].

Algoritmo 4 Definição da curvatura nos fluidos e da distância com sinal

```
N                                  ▷ Matriz com vetor normal em cada nó da interface
Kf_interface                       ▷ Vetor com curvatura em cada nó da interface
Kf_total                           ▷ Vetor com a curvatura em cada nó dos fluidos
sinal                              ▷ Vetor do sinal do produto interno com a normal
distancia                          ▷ Vetor da distância mínima para cada nó
for i in range(npoints) do
    pi ← [xi, yi, zi]              ▷ Coordenadas do nó dos fluidos
    j_min ← 0                      ▷ Número do nó da interface com distância mínima
    vector_min ← [0, 0, 0]         ▷ Vetor associado à distância mínima
    mini ← infinity                ▷ Distância mínima
    for j in range(npoints_interface) do
        pj ← [xj, yj, zj]          ▷ Coordenadas do nó da interface
        dist_vector ← pi − pj
        dist ← norm(dist_vector)
        if dist < mini then
            j_min ← j
            mini ← dist
            vector_min ← dist_vector
        end if
    end for
end for
Kf_total[i] ← Kf[j_min]
sinal[i] ← sinal(< N[j_min], vector_min >)
distancia[i] ← sinal[i] * mini
```

3.3 Resumo

Em suma, as metodologias acima apresentadas serão divididas em duas partes: obtenção da curvatura e modelagem da interface; em que a segunda é dependente da primeira.

No caso 2D, serão empregados dois métodos númericos para obtenção da curvatura. Um é baseado no sistema de equações de Frente para curvas 2D e uma adaptação do MDF enquanto o outro utiliza o MEF. No caso 3D, será utilizado tanto um método que emprega os cálculos com os vetores tangentes quanto outro que é derivado do MEF e do operador Laplace-Beltrami discreto.

No caso do cálculo em 3D, é de particular interesse verificar o efeito da área utilizada para avaliar a curvatura, em que foram apresentadas as áreas $\mathcal{A}_{voronoi}$ e $\mathcal{A}_{mixed}$, além da área utilizando o baricêntro no lugar do circuncêntro - esta será chamada $\mathcal{A}_{bari}$.

Com a curvatura obtida, utilizando o CSF discretizado, testes para ambas as abordagens Euleriana e Lagrangiana serão apresentados - tanto no caso bidimensional quanto tridimensional. Os resultados seguem no próximo capítulo.

Capítulo 4

Resultados e Discussão

A seguir serão apresentados os resultados obtidos pelos códigos numéricos, separados no caso 2D e 3D. Todos os dados aqui apresentados foram gerados utilizando um notebook Dell com processador *Intel Core i7-7500U* e 8GB de memória RAM. Os códigos foram rodados em *Python 3*, no *software Spyder* a partir do ambiente Anaconda.

Os elementos utilizados em 2D são todos lineares, assim como, em 3D, utilizou-se elementos de triângulo linear. Para cada caso será dado o número de elementos (n_e) e número de nós (n_p) da malha utilizada. Todas as malhas empregadas foram geradas no *software Gmsh* a partir da definição da geometria. Mais detalhes sobre o arquivo de malha gerado pelo *software* estão no Apêndice A.

4.1 Modelo 2D

No caso bidimensional, serão apresentados os resultados para os dois métodos propostos, que serão chamados no presente capítulo de "Frenet"e "MEF". Nesse sentido, será colocados, primeiramente, o que foi obtido para o cálculo da curvatura, mostrando a convergência da malha e tempo necessário para execução do código. Após, um dos dois métodos mencionados será escolhido e os resultados para a modelagem da interface serão mostrados. Destaca-se que esses números referem-se aos valores totais, da malha completa.

4.1.1 Curvatura 2D

No caso bidimensional, a obtenção da curvatura é direta a partir da resolução das Equações de Frenet, de forma que existem geometrias com uma equação analítica para tal grandeza. Assim, os métodos para cálculo de curvatura serão validados segundo expressões conhecidas.

Nesse sentido, serão estudados os valores obtidos variando o fator de tamanho do elemento de malha ("Element Size Factor"), chamado de h, e apresentando diferentes métricas para avaliação dos resultados. Para cada caso, serão apresentados os valores máximo e mínimo de curvatura obtidos, assim como um valor de erro, calculado como desvio dos valores teóricos. Este será dado por:

$$Erro = \sqrt{\frac{\sum (k_i - k_a)^2}{\sum k_i^2}} \tag{4.1}$$

Em que k_i é o valor calculado para a curvatura no nó e k_a seu valor teórico, como sugerido em [8]. Assim, o erro avalia o desvio em relação ao valor esperado e calcula uma forma de variância, somando-o em cada nó.

A curvatura de uma circunferência de raio R, por exemplo, é dada por:

$$k = \frac{1}{R} \tag{4.2}$$

Utilizando uma circunferência de raio 0.4, com o método de **Frenet**, obteve-se:

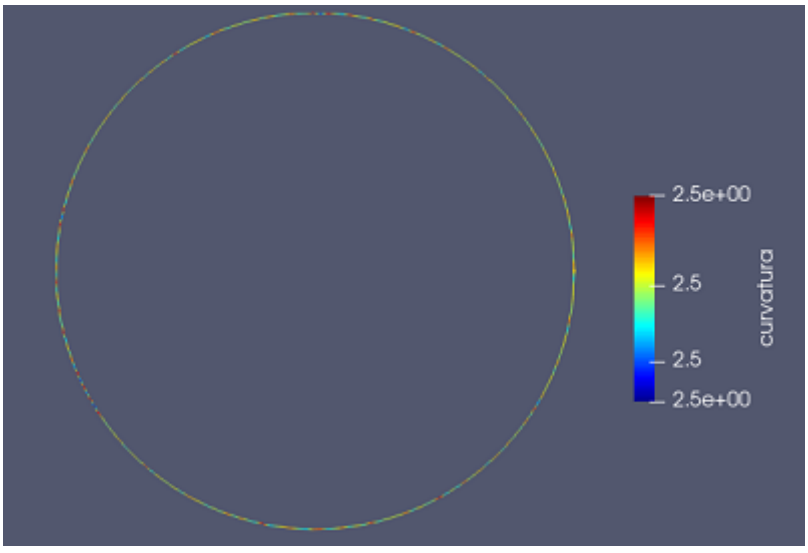


Figura 4.1: Curvatura da circunferência - $R = 0.4$ - Frenet

Enquanto o método de **Elementos Finitos** resultou em:

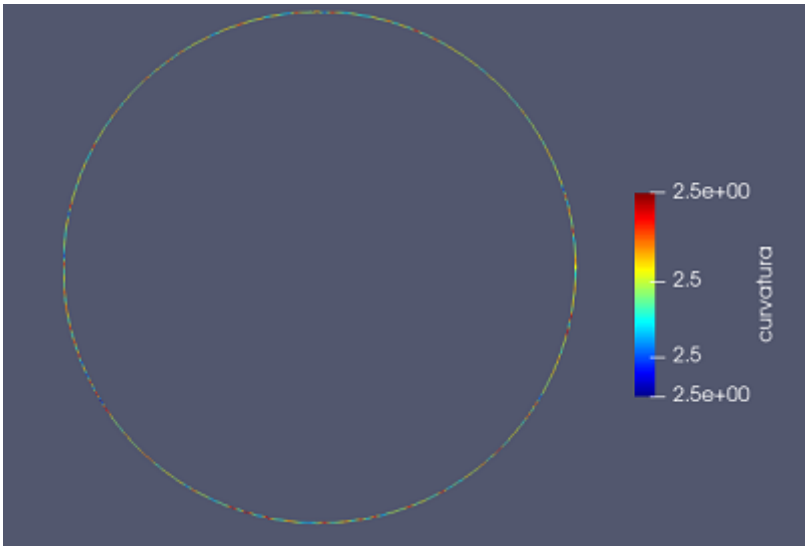


Figura 4.2: Curvatura da circunferência - $R = 0.4$ - MEF

Nos resultados acima, foi utilizado $h = 0.5$; $n_p = 46$ e $n_e = 45$. Observa-se que o resultado verifica o espera para ambos os casos, tendo o valor da curvatura em todos os pontos igual ao inverso do raio. De fato, mesmo que apareça alguma diferença na cor da curva, observa-se que a legenda não varia seu valor, sendo portanto somente devido à visualização no *Paraview*.

De forma mais completa, para a mesma geometria, foram realizados os seguintes testes da influência do tamanho do elemento:

h	Máx(k)	Mín(k)	Erro	Tempo [s]
5	2.500	2.500	2.014×10^{-16}	0.005
1	2.500	2.500	4.284×10^{-14}	0.005
0.5	2.500	2.500	1.340×10^{-14}	0.010
0.1	2.500	2.500	3.666×10^{-13}	0.040
0.05	2.500	2.500	1.216×10^{-12}	0.060

Tabela 4.1: Resultados para o método de Frenet com diferentes h - Circunferência, R = 0.4

Observa-se, portanto, que mesmo com um elemento bastante grosseiro os métodos já são capazes de obter o valor desejado para a curvatura em cada ponto. De fato, observa-se que o aumento de h implica no crescimento do valor do erro, assim como no tempo necessário para o cálculo.

De fato, com mais nós na malha, o código deve realizar mais cálculos e comete, portanto, uma quantidade maior de erros numéricos, o que explica tal aumento.

Contudo, este é desprezível em relação a sua ordem de grandeza, corroborando a aplicabilidade do método. No que tange o tempo, o MEF apresenta uma pequena vantagem em relação ao outro, não obstante os dois tenham se mostrado rápidos mesmo com um valor de h baixo.

A fim de se comparar com outra circunferência, foram gerados os mesmo gráficos para um novo valor de R, também utilizando $h = 0.5$, $n_p = 46$ e $n_e = 45$:

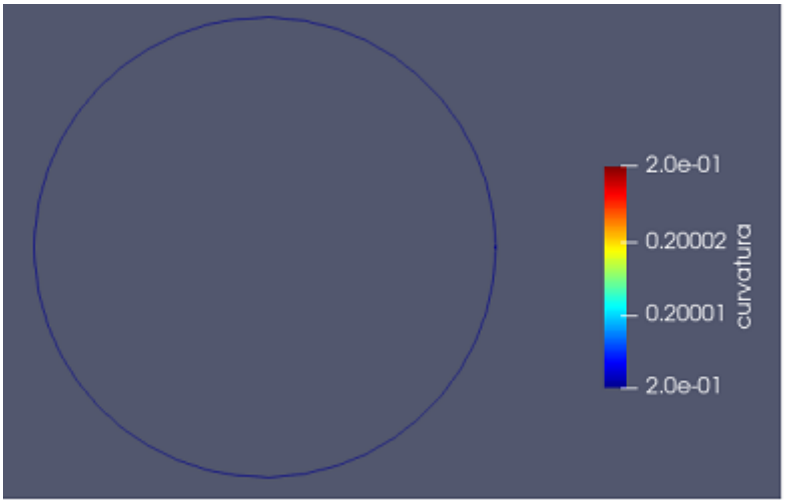


Figura 4.3: Curvatura da Circunferência - $R = 5$ - Frenet

Um caso mais genérico é a curvatura de uma elipse qualquer. Sendo a elipse de equação:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{4.3}$$

Sua curvatura em um ponto qualquer é dada por:

$$k = \frac{ab}{(a^2 \sin^2 \theta + b^2 \cos^2 \theta)^{3/2}} \tag{4.4}$$

em que θ representa o ângulo com o eixo das abscissas, variando entre 0 e 2π,

h	Máx(k)	Mín(k)	Erro	Tempo [s]
5	2.500	2.500	1.343×10^{-15}	0.004
1	2.500	2.500	5.003×10^{-15}	0.005
0.5	2.500	2.500	1.960×10^{-14}	0.009
0.1	2.500	2.500	4.493×10^{-13}	0.010
0.05	2.500	2.500	1.590×10^{-12}	0.040

Tabela 4.2: Resultados para o MEF com diferentes h - Circunferência, R = 0.4

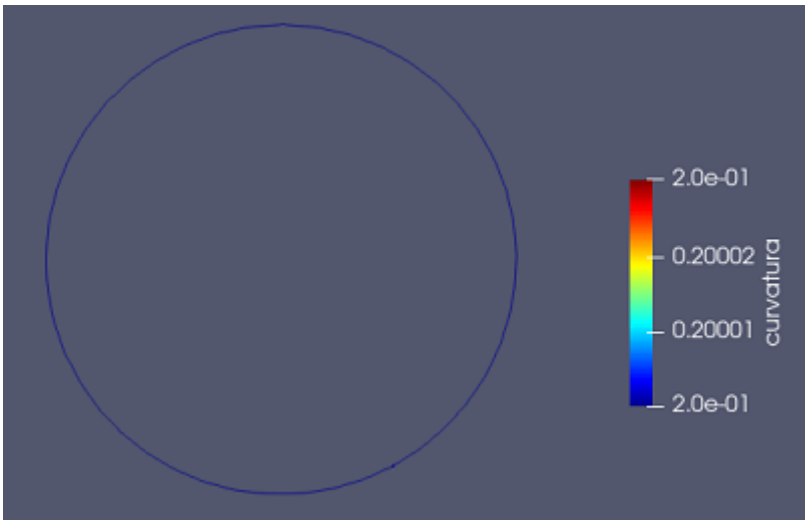


Figura 4.4: Curvatura da circunferência - $R = 5$ - MEF

sendo empregada a parametrização usual da elipse, $(x, y) = (a\cos\theta, b\sin\theta)$.

h	Máx(k)	Mín(k)	Erro	Tempo [s]
5	2.671	0.469	0.891	0.004
1	4.939	0.410	0.220	0.007
0.5	5.764	0.401	0.048	0.008
0.1	6.226	0.400	0.002	0.020
0.05	6.244	0.400	0.001	0.040

Tabela 4.3: Resultados para o método de Frenet com diferentes h - Elipse, a = 0.4, b = 1

h	Máx(k)	Mín(k)	Erro	Tempo [s]
5	2.671	0.469	0.887	0.005
1	4.939	0.410	0.220	0.005
0.5	5.764	0.401	0.049	0.006
0.1	6.226	0.400	0.002	0.015
0.05	6.244	0.400	0.001	0.030

Tabela 4.4: Resultados para o MEF com diferentes h - Elipse, a = 0.4, b = 1

Como era de se esperar, os dois métodos apontaram valores muito próximos em todos os dados medidos, com pequenas diferenças somente no erro e no tempo de execução. De fato, como mencionado, os métodos são equivalentes de um ponto de vista matemático. Por outro lado, agora nota-se que há convergência dos valores com melhora da malha - o que não se observava na circunferência pois essa possui curvatura constante, que os métodos conseguiam capturar mesmo com poucos nós.

Nota-se que tal convergência está ocorrendo para os valores esperados, diga-se

um máximo de 6.25 e mínimo de 0.4, indicando a validade dos métodos propostos, o que também é expresso pela diminuição no valor do erro. O resultado gráfico para a malha com $h = 0.05$ ($n_p = 428$ e $n_e = 429$) é apresentado a seguir.

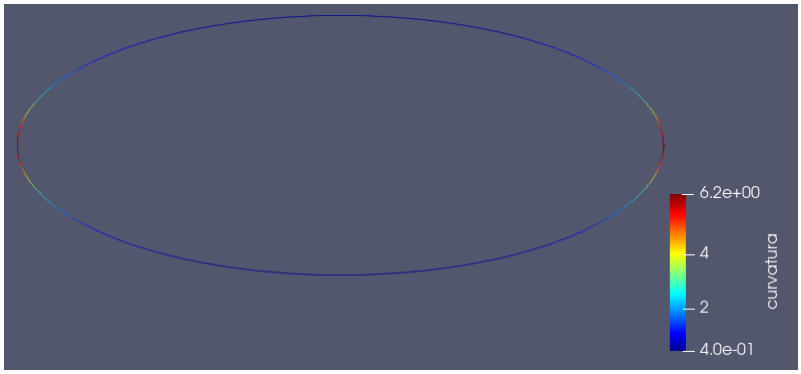


Figura 4.5: Curvatura da elipse - $a = 1, b = 0.4$ - Frenet

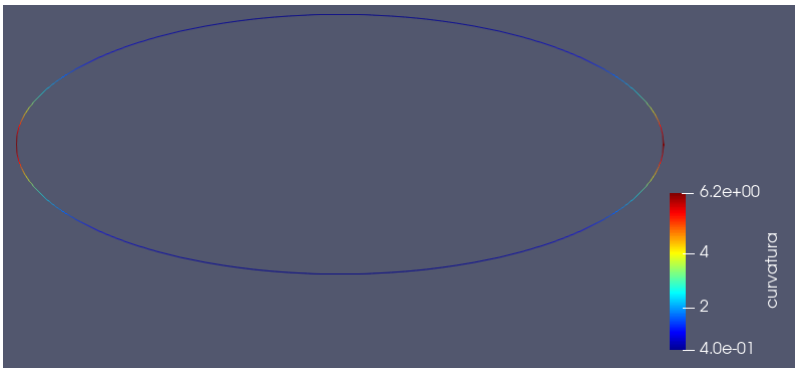


Figura 4.6: Curvatura da elipse - $a = 1, b = 0.4$ - MEF

Outro caso de interesse é um retângulo, em que, como todo os lados são segmentos de reta, a curvatura deve ser nula. Abaixo está os resultados obtidos para um quadrado de lado l igual a 1.

h	Máx(k)	Mín(k)	Erro	Tempo [s]
5	2.828	0.000	1.000	0.006
1	11.314	0.000	1.000	0.008
0.5	21.213	0.000	1.000	0.010
0.1	100.409	0.000	1.000	0.020
0.05	200.818	0.000	1.000	0.030

Tabela 4.5: Resultados para o método de Frenet com diferentes h - Quadrado, l = 1

Nesse caso, o valor nos vértices do quadrado crescem com o elemento de malha cada vez menor, isso pois a curvatura não pode ser definida nesses pontos, causando o valor de 1 encontrado no Erro. Diminuir o tamanho da reta elementar faz com

h	Máx(k)	Mín(k)	Erro	Tempo [s]
5	2.828	0.000	1.000	0.006
1	11.314	0.000	1.000	0.008
0.5	21.213	0.000	1.000	0.010
0.1	100.409	0.000	1.000	0.020
0.05	200.818	0.000	1.000	0.030

Tabela 4.6: Resultados para MEF com diferentes h - Quadrado, l = 1

que nos vértices calcule-se uma curvatura equivalente à de uma circunferência com raio cada vez maior.

Não obstante, os gráficos gerados no *Paraview* indicam o comportamento esperado. Nas imagens $h = 0.05$, $n_p = 568$ e $n_e = 572$.

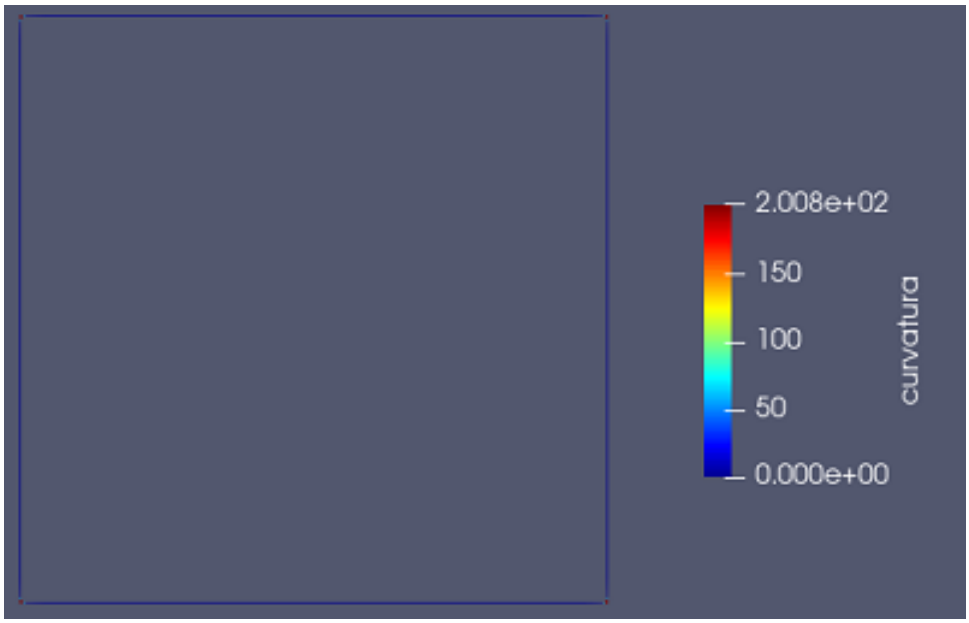


Figura 4.7: Curvatura do quadrado - $l = 1$ - Frenet

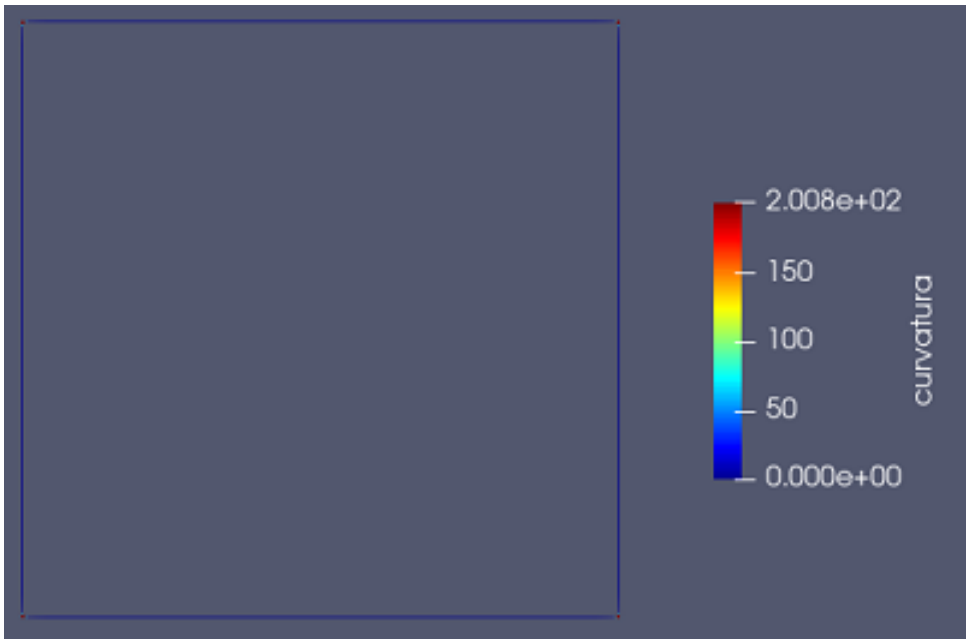


Figura 4.8: Curvatura do quadrado - $l = 1$ - MEF

Uma última figura de interesse para validação do cálculo da curvatura 2D será formada por arcos de circunferência. A geometria gerada encontra-se na [Figura 4.9].

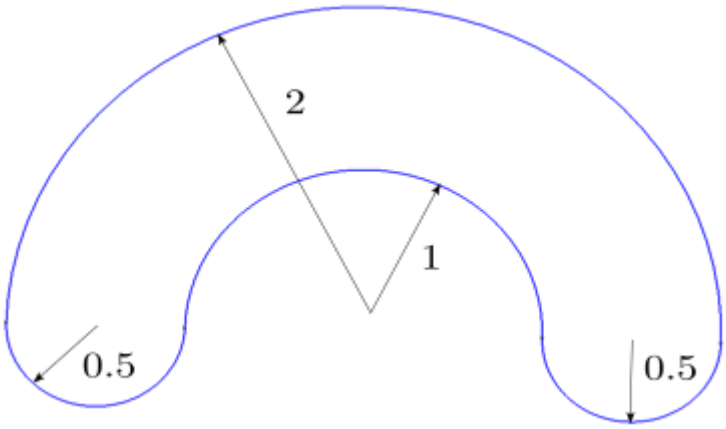


Figura 4.9: Figura genérica 2D - Geometria

Foram obtidos os resultados:

h	Máx(k)	Mín(k)	Tempo [s]
5	2.000	-1.000	0.006
1	2.000	-1.000	0.008
0.5	2.000	-1.000	0.008
0.1	2.000	-1.000	0.020
0.05	2.000	-1.000	0.020

Tabela 4.7: Resultados para MEF com uma figura genérica

Que foram iguais pela discretização da equação de Frenet, como esperado. Os gráficos gerados, portanto, também foram iguais - no caso com $h = 0.05$ ($n_e = 253$ e $n_p = 257$):

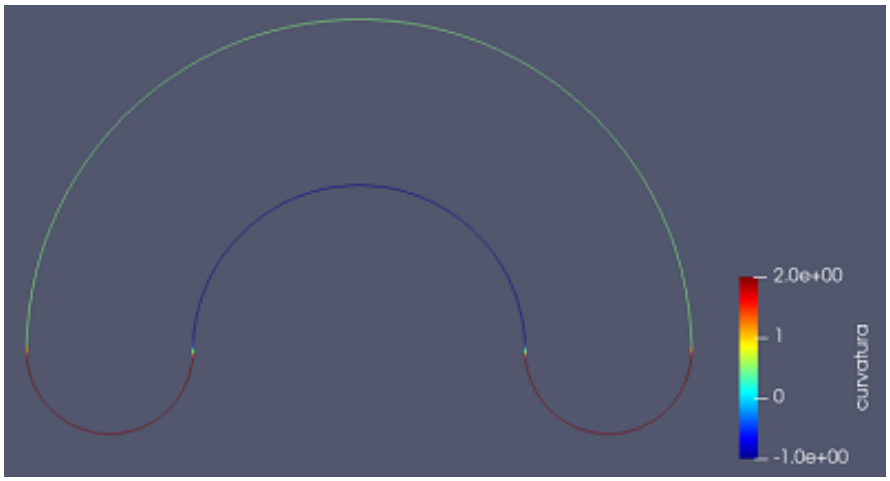


Figura 4.10: Curvatura de figura genérica - MDF

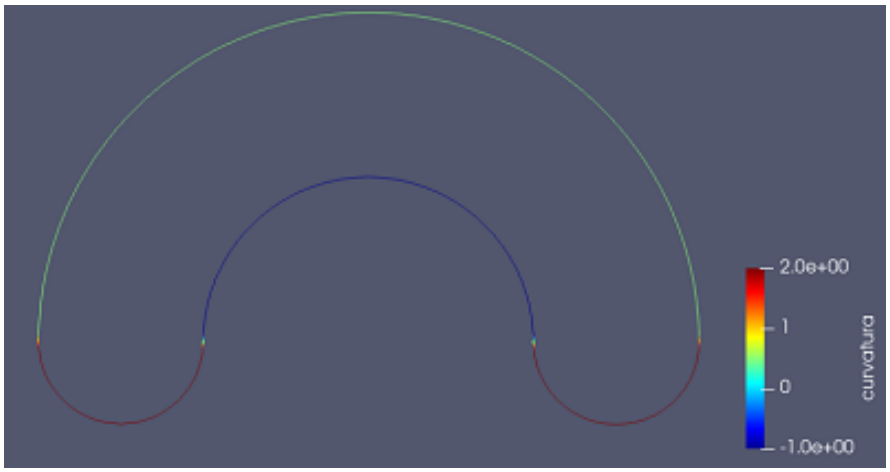


Figura 4.11: Curvatura de figura genérica - MEF

Observa-se que a distribuição da curvatura reproduz o esperado, no sentido que na regiões dos arcos de circunferência esta assume justamente o inverso do raio. Além disso, o sinal da curvatura se mostra condizente com o dado na definição na seção de Geometria Diferencial, sendo negativo no arco interno que torna a figura não-convexa. Nota-se, também, que, nos pontos que ligam este arco aos dois menores, a curvatura é próxima de zero, uma vez que há uma inflexão da curva e esta se aproxima à uma reta.

4.1.2 Força de Tensão Superficial 2D

A partir dos resultados acima, como era já previsto, nota-se que não há muita diferença entre as duas metodologias apresentadas para obtenção da curvatura. Para o cálculo das forças de tensão superficial no modelo 2D, então, a curvatura será obtida pelo método de elementos finitos, puramente por uma questão de facilidade na implementação do código. Outro ponto é a pequena diferença de tempo de execução que favorece o MEF. Contudo, os resultados seriam os mesmos independentemente de qual o método utilizado.

Abordagem Lagrangiana

No primeiro teste realizado para a modelagem da interface com a abordagem lagrangiana foi utilizada a mesma circunferência de raio 0.4. Os resultados expostos são a malha completa utilizada, assim como a distribuição da função de Heaviside e das forças de tensão superficial com um foco dado na região da interface.

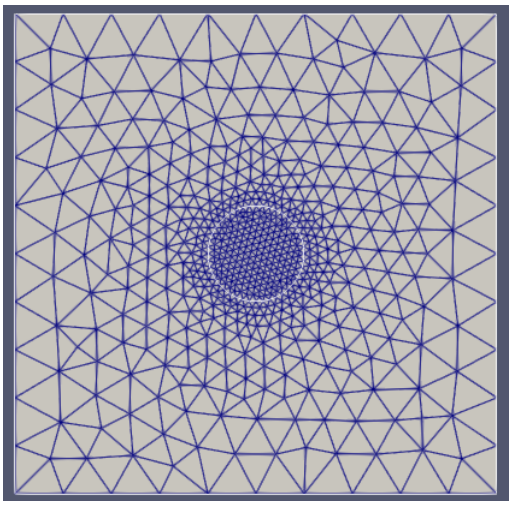


Figura 4.12: Malha acoplada 2D - Circunferência - $n_p = 624$ e $n_e = 1290$

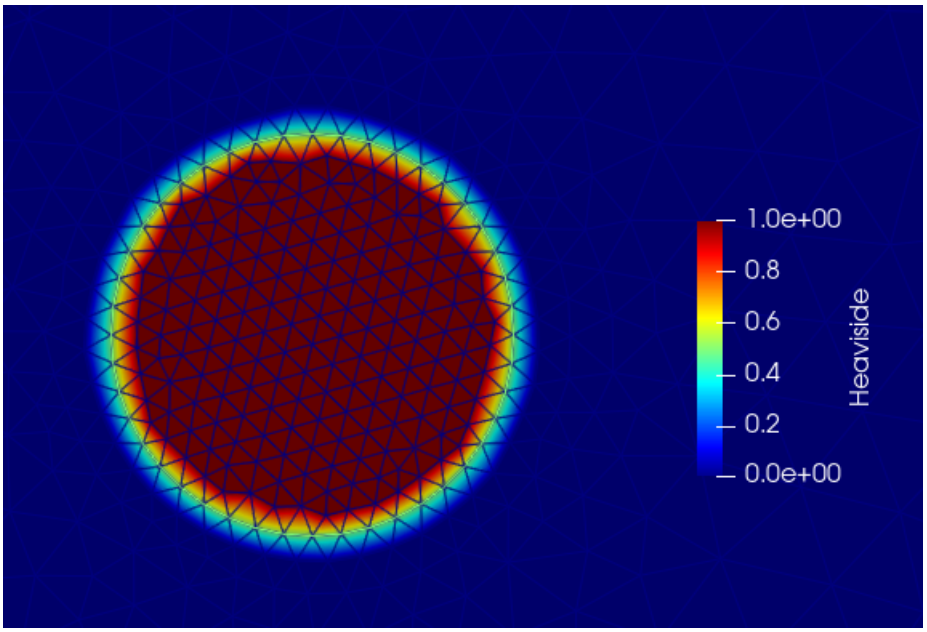


Figura 4.13: Função de Heaviside para malha acoplada 2D - Circunferência

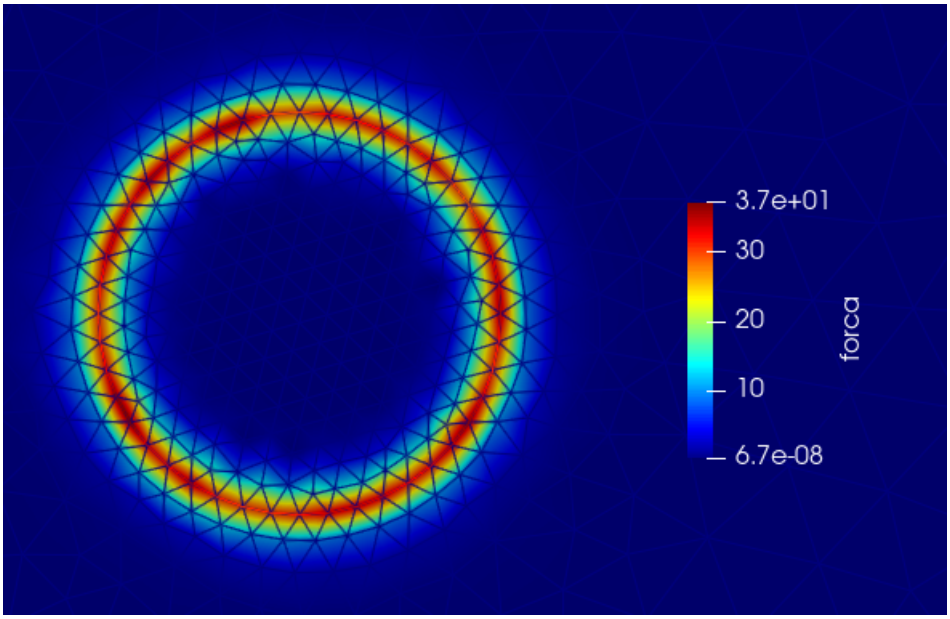


Figura 4.14: Força para malha acoplada 2D - Circunferência

As imagens mostram que a função de Heaviside foi prescrita como em 2.95 corretamente. Além disso, a distribuição de forças é razoável, uma vez que há certa regularidade na superfície e somente alguma variação em regiões - tal variação pode ser causada pela malha utilizada.

Para a elipse, obteve-se:

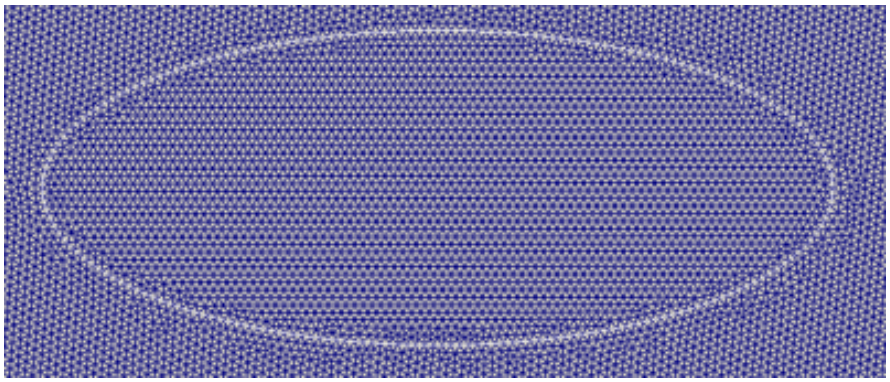


Figura 4.15: Malha acoplada 2D - Elipse - $n_p = 19039$ e $n_e = 38287$

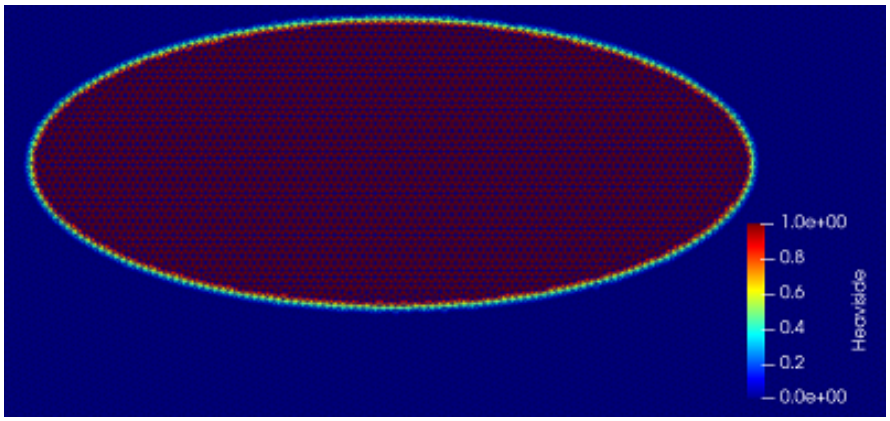


Figura 4.16: Função de Heaviside para malha acoplada 2D - Elipse

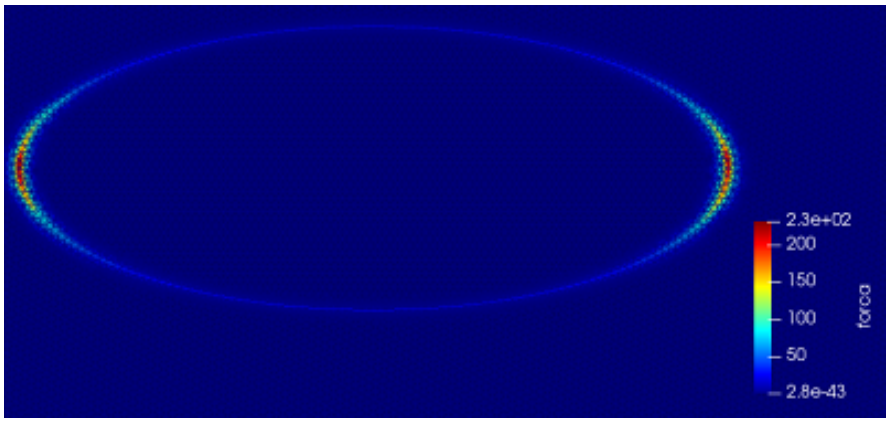


Figura 4.17: Força para malha acoplada 2D - Elipse

Agora com a figura formada por arcos de circunferência:

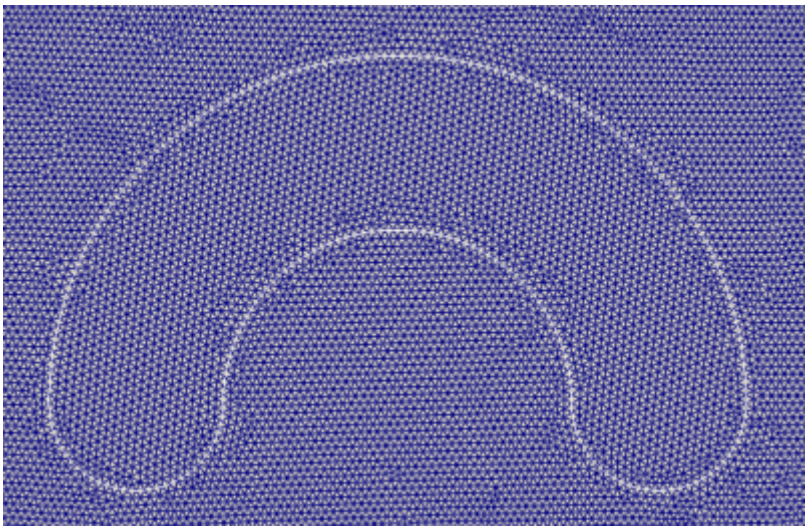


Figura 4.18: Malha acoplada 2D - Figura Genérica - $n_p = 17313$ e $n_e = 34339$

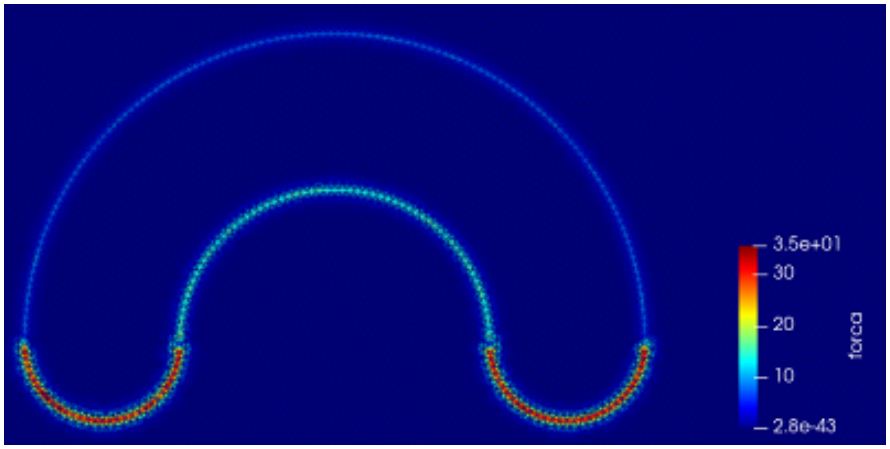


Figura 4.19: Força para malha acoplada 2D - Figura Genérica

Nota-se que a distribuição de forças indica a forte dependência com a curvatura da figura. Nas regiões de maior curvatura a força assume valores mais altos, fazendo com que exista certa descontinuidade, como acontece com o gráfico da curvatura.

Além das geometrias utilizadas para validação da curvatura, a força foi também calculada em outras que visam representar situações físicas em escoamentos bifásicos. Serão, portanto, apresentados quatro casos, em que são expostas a geometria e a malha, assim como a força calculada. O primeiro refere-se a presença de múltiplas bolhas, enquanto o segundo e terceiro à coalescência de bolhas. Já o último representa bolhas formadas no escoamento em um duto.

Com múltiplas circunferências representando as bolhas no escoamento, a força calculada teve a seguinte distribuição [Figura 4.22], em que utilizou-se a malha

apresentada em [Figura 4.21]. É interessante notar, nesse caso, como a diferença nos raios influencia o valor da força, comparativamente. De fato, com um raio menor - e, logo, curvatura maior - a força tem maior magnitude.

Deve-se observar que, nos resultados aqui apresentados, o valor absoluto da força mostrado foi calculado considerando a tensão superficial constante e unitária em todas as superfícies. Isto é, na equação [2.89] considerou-se $\sigma = 1$. Portanto, os valores absolutos da escala não é o ponto mais relevante a ser destacado nas imagens, mas sim a distribuição das cores e a ordem de grandeza.

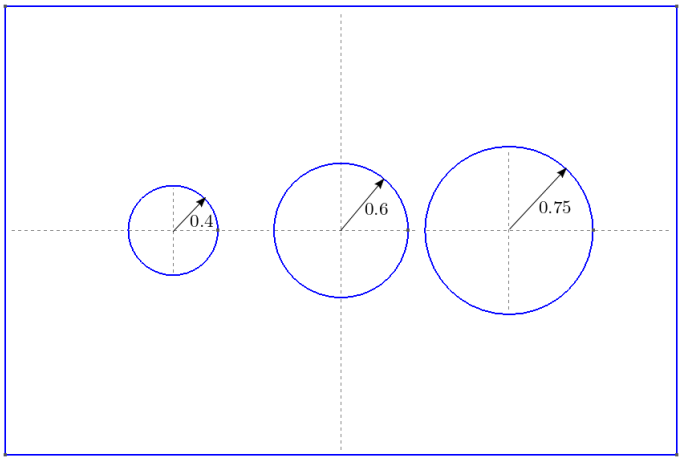


Figura 4.20: Geometria - Múltiplas Bolhas

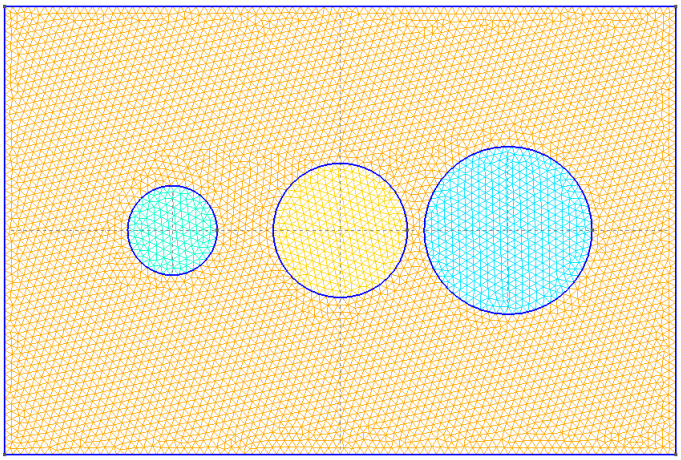


Figura 4.21: Malha acoplada 2D - Múltiplas Bolhas - $n_p = 5779$ e $n_e = 11409$

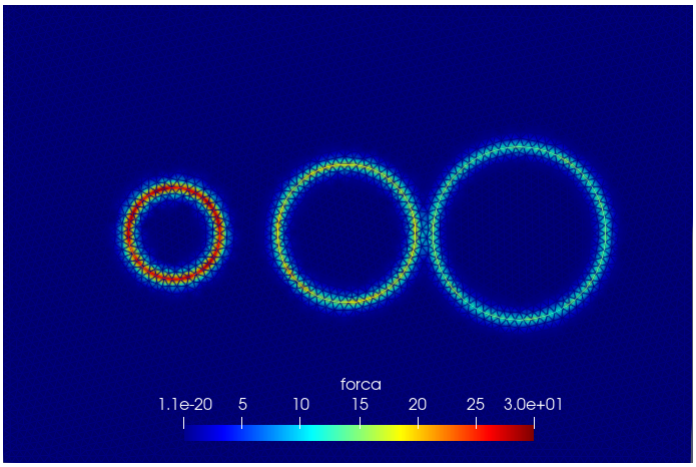


Figura 4.22: Força para malha acoplada 2D - Múltiplas Bolhas

Como mencionado, outro caso estudado foi a coalescência de bolhas. Abaixo são apresentados dois exemplos que reproduzem tal fenômeno. No primeiro foram representadas duas bolhas de mesmo raio; enquanto no segundo estas possuem raios diferentes e estão mais próximas. Para o primeiro caso:

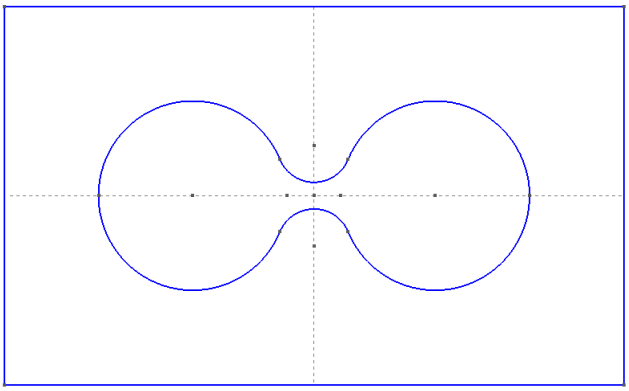


Figura 4.23: Geometria - Coalescência 1

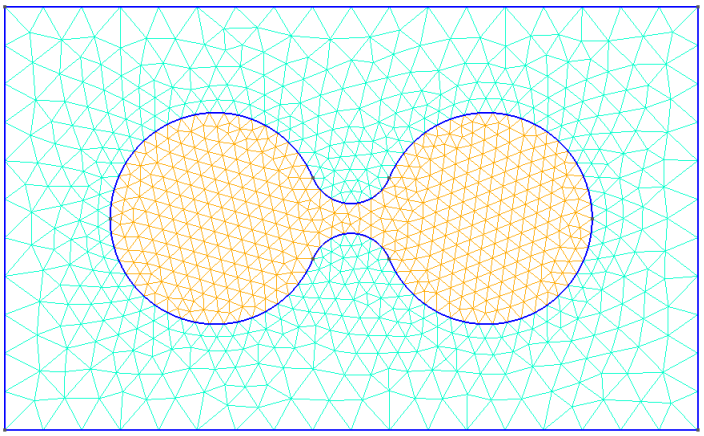


Figura 4.24: Malha acoplada 2D - Coalescência 1 - $n_p = 961$ e $n_e = 2022$

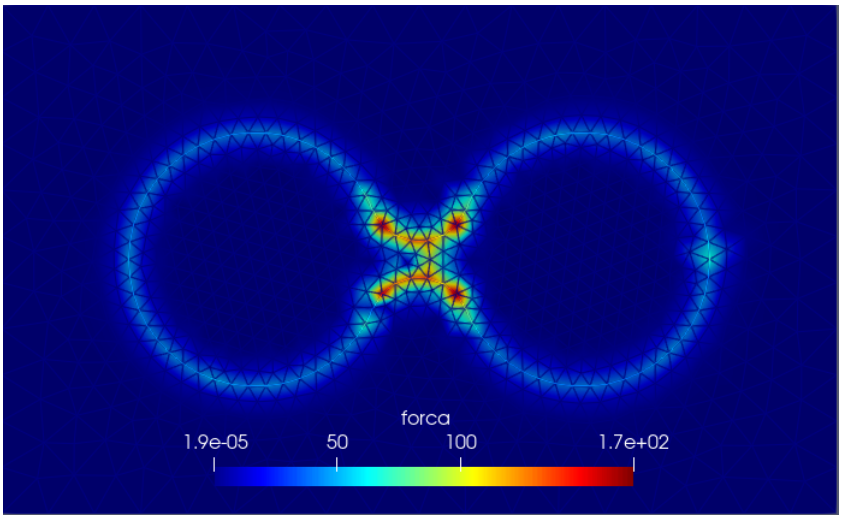


Figura 4.25: Força para malha acoplada 2D - Coalescência 1

Já o segundo:

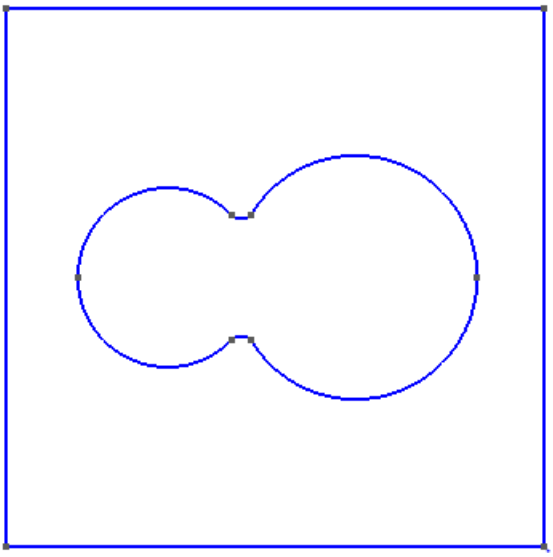


Figura 4.26: Geometria - Coalescência 2

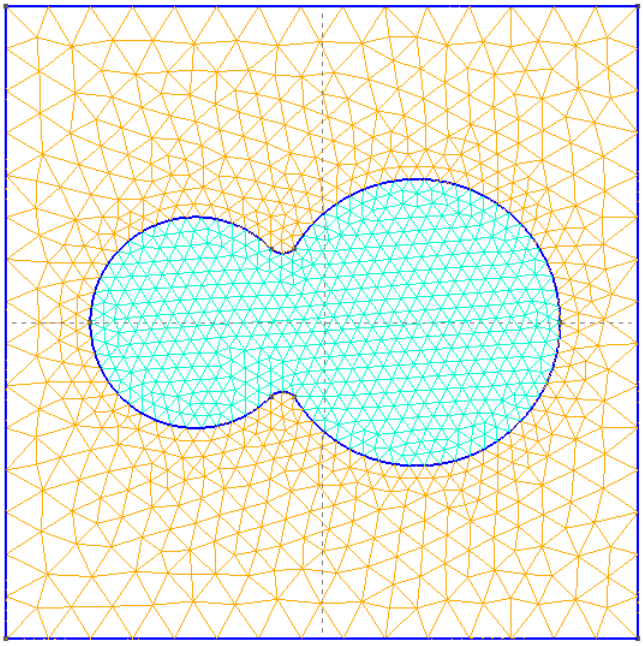


Figura 4.27: Malha acoplada 2D - Coalescência 2 - $n_p = 1059$ e $n_e = 2210$

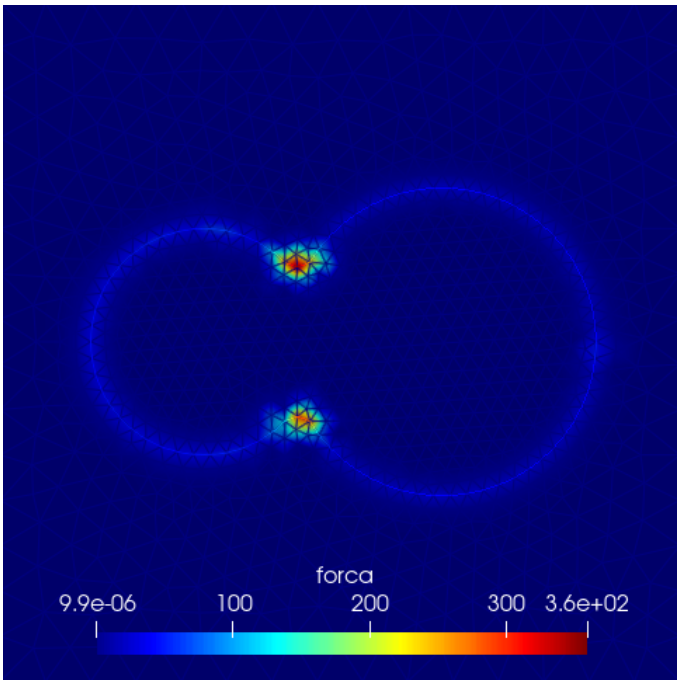


Figura 4.28: Força para malha acoplada 2D - Coalescência 2

Por fim, foi computada a distribuição da força em uma situação que simula a presença de bolhas em um tubo circular.

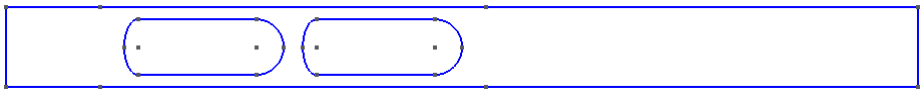


Figura 4.29: Geometria - Escoamento em tubo

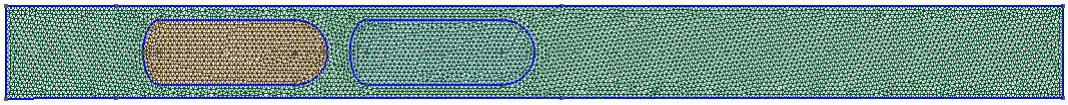


Figura 4.30: Malha acoplada 2D - Escoamento em tubo - $n_p = 5811$ e $n_e = 11316$

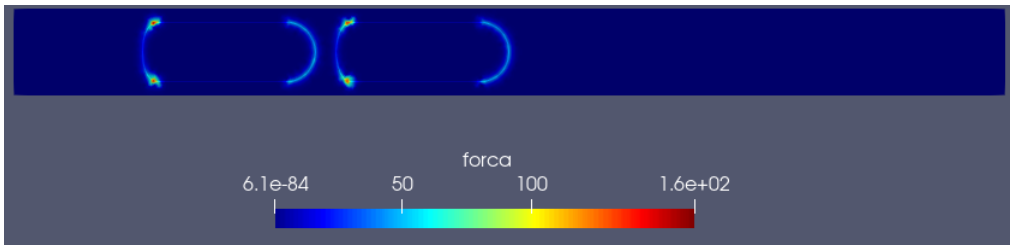


Figura 4.31: Força para malha acoplada 2D - Escoamento em tubo

Como esperado, nos exemplos anteriores observa-se que a força age com maior intensidade nas regiões de maior curvatura. No caso das coalescências, tal comportamento indica a presença nas regiões de transição das bolhas formada durante o fenômeno.

Abordagem Euleriana

No caso da abordagem euleriana, serão exploradas as mesmas figuras e representações de bolhas que as anteriores. Assim, serão mostrados os gráficos da força de tensão superficial, também evidenciando as malhas utilizadas.

O ponto importante a ser notado é a forma com que a interface se distingue nas imagens. Na abordagem anterior, por ter uma representação precisa ("sharp"), a interface é claramente perceptível em relação à malha. Nessa segunda abordagem, contudo, como comentado, a interface possui espessura não-nula.

O primeiro resultado observado foi o caso da circunferência, com a seguinte malha desacoplada, com $h = 0.1$:

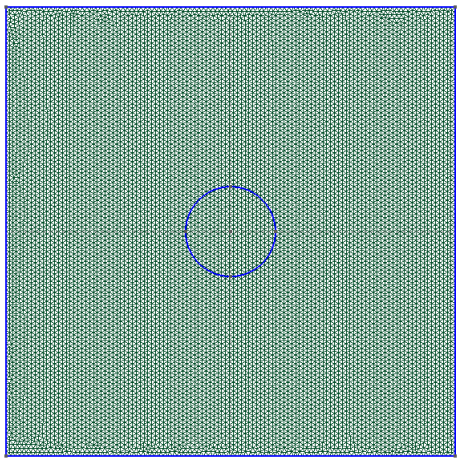


Figura 4.32: Malha desacoplada 2D - Circunferência - $n_p = 6047$ e $n_e = 12041$

Com a função de Heaviside [3.19], a distribuição foi a seguinte, em que foi utilizado $\epsilon = 0.1$:

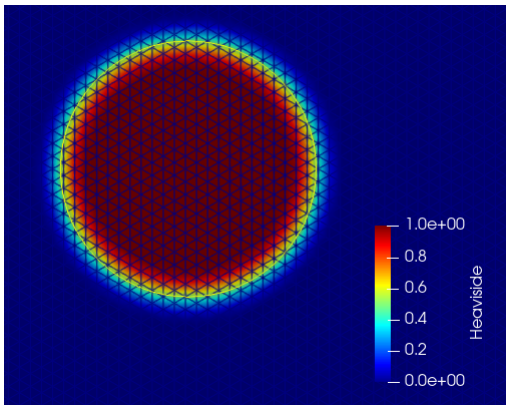


Figura 4.33: Função de Heaviside H_1^ϵ para desacoplada 2D - Circunferência

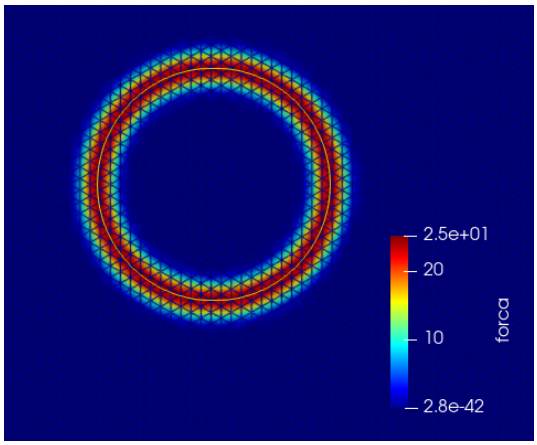


Figura 4.34: Força para malha desacoplada 2D - Circunferência - H_1^ϵ

Já com a equação [3.20], utilizando o mesmo valor de ϵ, obteve-se:

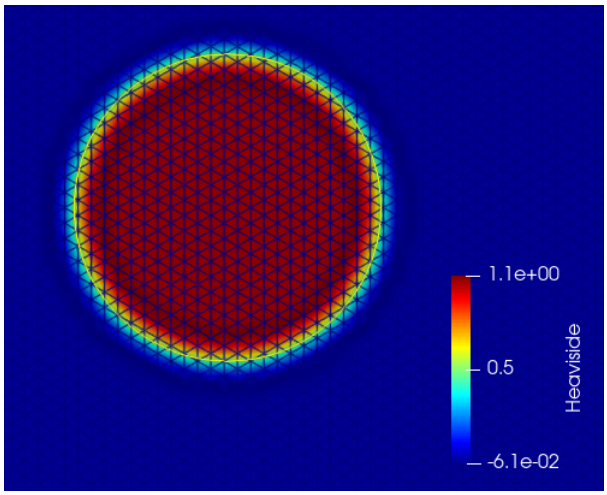


Figura 4.35: Função de Heaviside H_2^ϵ para desacoplada 2D - Circunferência

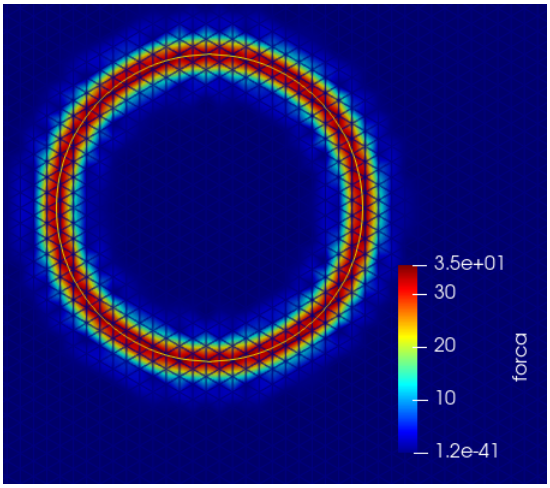


Figura 4.36: Força para malha desacoplada 2D - Circunferência - H_2^ϵ

Nota-se que não há uma diferença muito grande entre os dois resultados - apesar de o valor absoluto da força diferir as distribuições são similares. Não obstante, a função de H_2^ϵ parece apresentar mais flutuações no que tange os valores da própria função na malha, com valores ultrapassando o de 1, além de produzir uma interface levemente menos suave. Por esse motivo, a função H_1^ϵ será a utilizada para os testes que se seguem. Em todos foi $\epsilon = 0.1$.

Para a elipse:

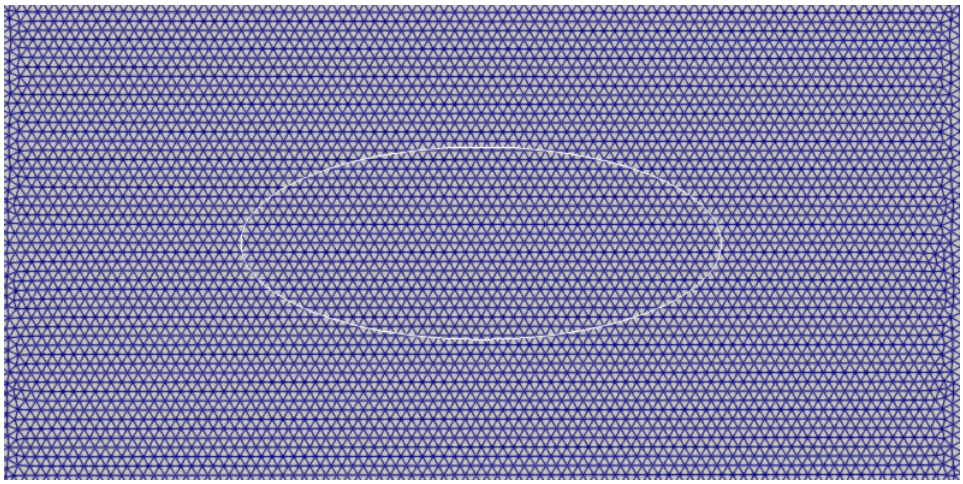


Figura 4.37: Malha desacoplada 2D - Elipse - $n_p = 4935$ e $n_e = 9495$

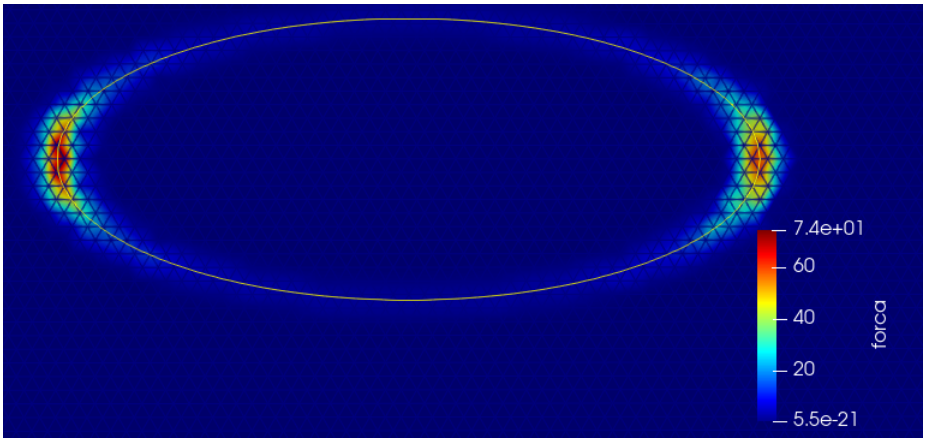


Figura 4.38: Força para malha desacoplada 2D - Elipse

Já para a figura genérica formada por arcos de circunferência:

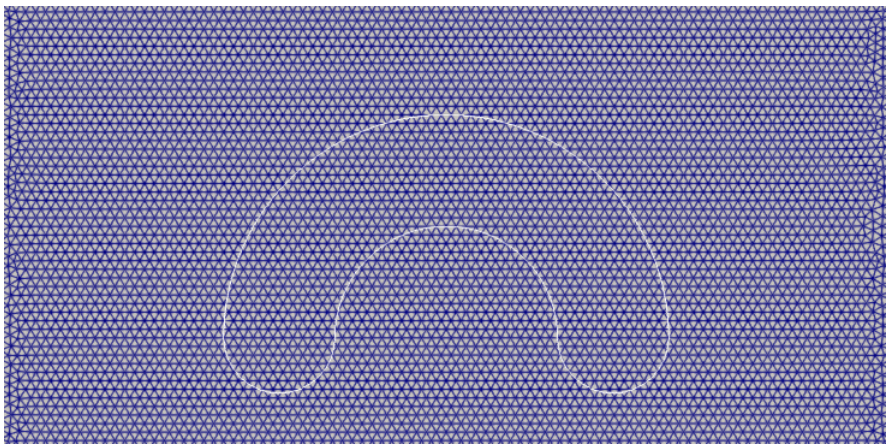


Figura 4.39: Malha desacoplada 2D - Figura Genérica - $n_p = 4960$ e $n_e = 9521$

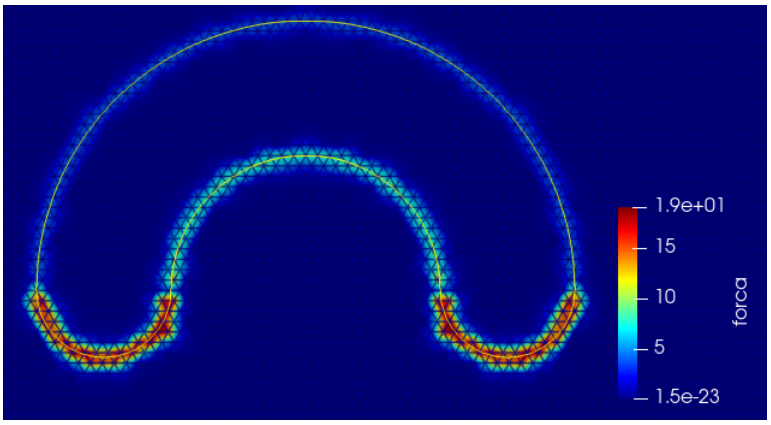


Figura 4.40: Força para malha desacoplada 2D - Figura Genérica

Com as múltiplas bolhas:

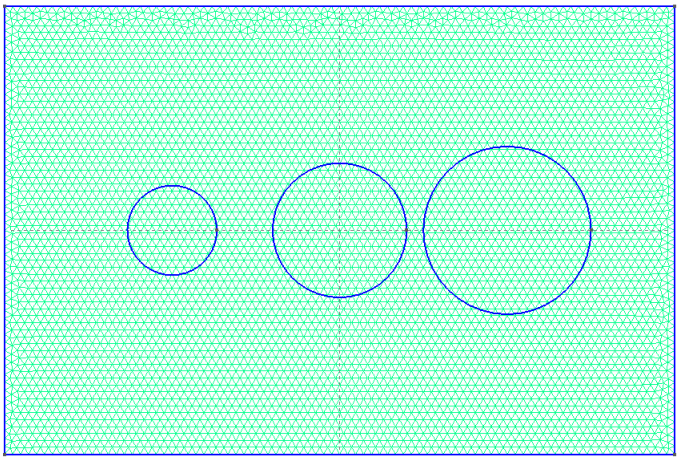


Figura 4.41: Malha desacoplada 2D - Múltiplas Bolhas - $n_p = 5779$ e $n_e = 11122$

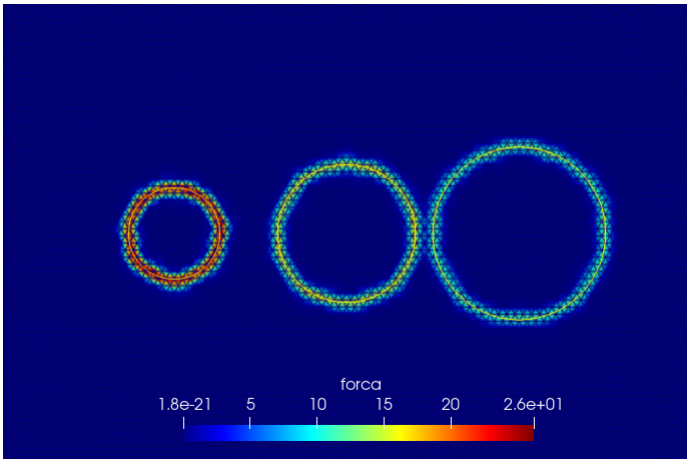


Figura 4.42: Força para malha desacoplada 2D - Múltiplas Bolhas

Para os casos de coalescência:

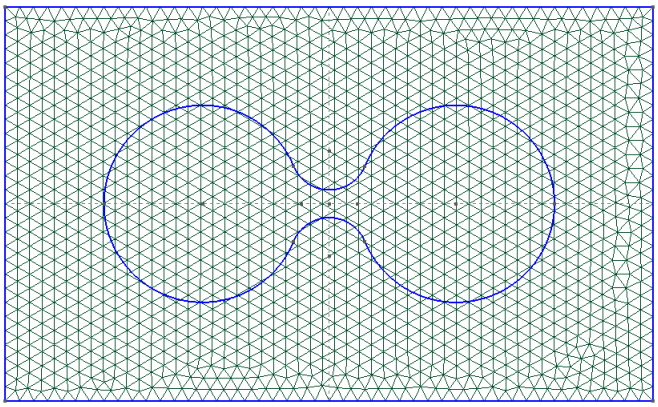


Figura 4.43: Malha desacoplada 2D - Coalescência 1 - $n_p = 1679$ e $n_e = 3265$

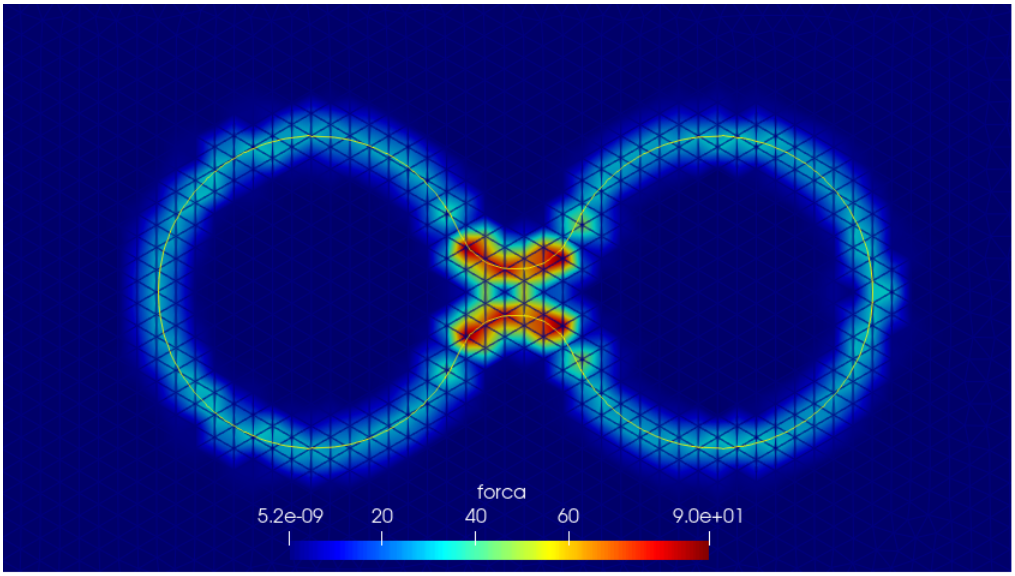


Figura 4.44: Força para malha desacoplada 2D - Coalescência 1

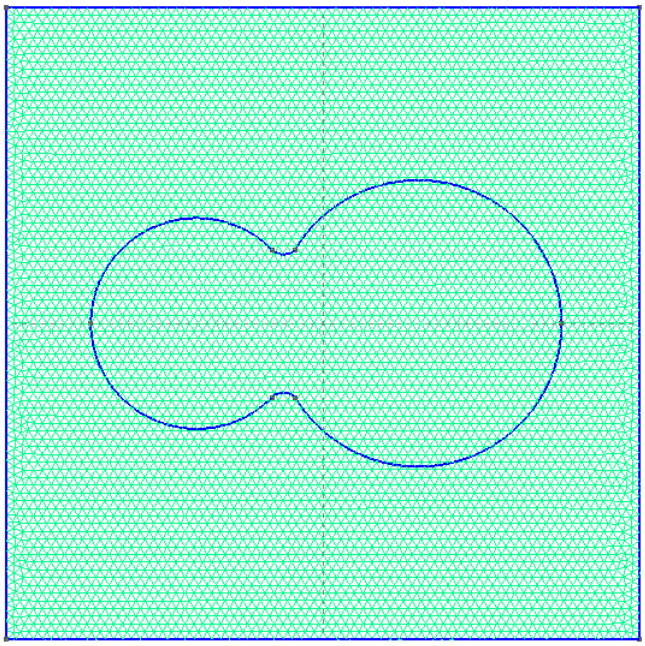


Figura 4.45: Malha desacoplada 2D - Coalescência 2 - $n_p = 6402$ e $n_e = 12400$

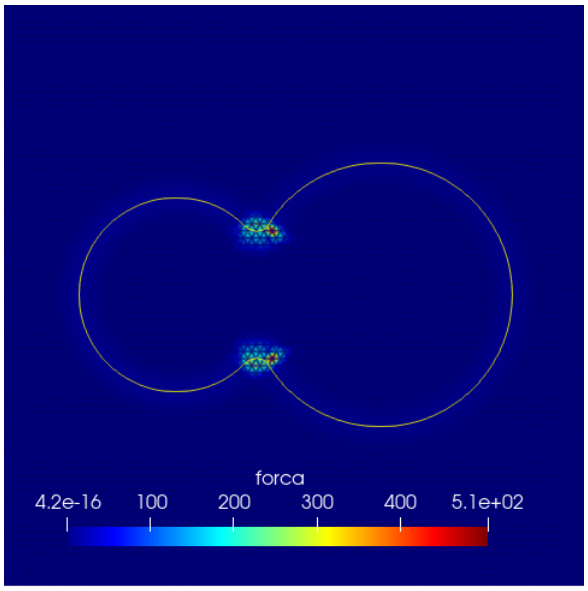


Figura 4.46: Força para malha desacoplada 2D - Coalescência 2

Por fim, no caso das bolhas formadas em um tubo:

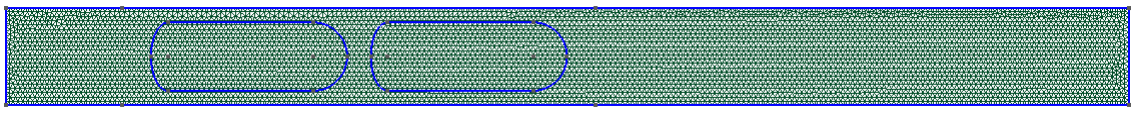


Figura 4.47: Malha desacoplada 2D - Escoamento em tubo - $n_p = 6039$ e $n_e = 11750$

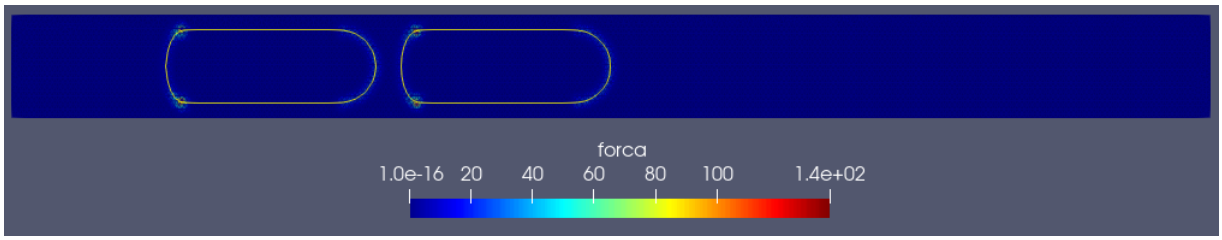


Figura 4.48: Força para malha desacoplada 2D - Escoamento em tubo

Fica evidente pelas imagens acima a diferença entre as duas abordagens. Para a figura elíptica, por exemplo, verifica-se uma região maior representando a interface entre os fluidos, que se destaca nas regiões de curvatura máxima.

Também é notável como a interface não é definida de maneira tão suave, mesmo utilizando uma função de Heaviside suavizada, como as apresentadas. Isto é particularmente perceptível na imagem referente à figura genérica [Figura 4.40].

Nos casos da coalescência e escoamento no tubo, nota-se que a abordagem com malha desacoplada parece atenuar a distribuição da força, de forma que os valores máximos são atingidos em menos pontos. Contudo, tais valores costumam ser maiores no caso com a malha acoplada.

Essa diferença entre os valores máximos da força em cada uma das abordagens pode estar relacionada com a função de Heaviside utilizada. No caso desacoplado, a suavização empregada pode ter causado o efeito de diminuir os valores extremos, uma vez que o gradiente da função será menor.

4.2 Modelo 3D

Assim como no caso 2D, os resultados apresentados a seguir para o modelo tridimensional serão divididos em duas partes. Primeiramente, será validado o cálculo da curvatura pelos dois métodos propostos - Volumes Finitos e Elementos Finitos. Nesse sentido, serão testados objetos geométricos com curvatura analítica conhecida, como esfera, elipsoide, toro e cubo.

Como mencionado, os métodos apresentados para tal cálculo são equivalentes, de forma que qualquer diferença que apareça é de caráter computacional. Serão apresentados os resultados para o MVF considerando os dois loops propostos e comparando o tempo de execução. Após serão comparadas as áreas (baricêntrica,

circuncêntrica e Voronoi) utilizando um dos métodos.

A partir disso, serão mostrados os resultados para as duas abordagens de modelagem da interface entre os fluidos, com a malha acoplada e desacoplada.

4.2.1 Curvatura 3D

Para o caso da esfera de raio R, a curvatura média é dada por:

$$k = \frac{2}{R} \tag{4.5}$$

Este caso será utilizado para comparar a performance dos métodos relativamente ao tempo. Assim, $Tempo_n$ representa o tempo utilizando o loop nos nós [Algoritmo 2] e $Tempo_e$ com o loop nos elementos [Algoritmo 3]. Obteve-se, utilizando $\mathcal{A}_{mixed}$:

h	Máx(k)	Mín(k)	Erro	$Tempo_n$ [s]	$Tempo_e$ [s]
3	6.939	4.289	0.082	0.134	0.057
1	6.163	4.322	0.031	0.510	0.361
0.5	5.298	4.867	0.003	1.964	1.575
0.1	5.947	4.215	0.004	92.593	33.155
0.05	7.595	3.873	0.003	> 600	164.148

Tabela 4.8: Resultados para o MVF com diferentes h - Esfera, $R = 0.4$

Para o MEF:

h	Máx(k)	Mín(k)	Erro	Tempo [s]
3	6.939	4.289	0.082	0.075
1	6.163	4.322	0.031	0.420
0.5	5.298	4.867	0.003	1.375
0.1	5.947	4.215	0.004	36.981
0.05	7.595	3.873	0.003	170.558

Tabela 4.9: Resultados para o MEF com diferentes h - Esfera, $R = 0.4$

Nota-se que, relativamente ao tempo, o método de volumes finitos utilizando um laço nos elementos mostrou-se como o mais vantajoso que com o laço nos nós. De fato, o cálculo dos elementos vizinhos é extremamente dispendioso quando há uma grande quantidade de elementos. Em relação ao MEF, os resultados estão próximos demais para que se chegue em uma conclusão decisiva, ainda que pareça existir uma pequena vantagem no MVF com loop nos elementos.

Além disso, novamente deve ser notado que os demais resultados foram exatamente iguais para os dois métodos. Não está mostrado nas tabelas acima, mas é evidente que os dois loops do MVF também produziram os mesmos resultados, com exceção do tempo.

No que tange os valores obtidos para a curvatura, observa-se que o refino da malha está gerando um afastamento do desejado ($k = 5$) em relação ao máximo e mínimo obtidos - ainda que o Erro esteja estabilizado. Isso deve estar relacionado à erros numéricos advindos do cálculo de área com elementos de tamanho muito pequeno.

Com uma esfera de raio $R = 5$:

h	Máx(k)	Mín(k)	Erro	$Tempo_n$ [s]	$Tempo_e$ [s]
3	0.495	0.344	0.070	0.082	0.057
1	0.503	0.345	0.028	0.551	0.361
0.5	0.513	0.329	0.020	1.766	1.575
0.1	0.784	0.265	0.010	79.821	29.466
0.05	0.481	0.328	0.002	> 600	150.282

Tabela 4.10: Resultados para o MVF com diferentes h - Esfera, $R = 5$

Para o MEF:

h	Máx(k)	Mín(k)	Erro	Tempo [s]
3	0.495	0.344	0.070	0.168
1	0.503	0.345	0.028	0.453
0.5	0.513	0.329	0.020	1.673
0.1	0.784	0.265	0.010	33.483
0.05	0.481	0.328	0.002	144.347

Tabela 4.11: Resultados para o MEF com diferentes h - Esfera, $R = 5$

Novamente observa-se o mesmo comportamento: apesar de o Erro estar estável ou diminuindo, o refino da malha produz valores de máximo e mínimo não necessariamente mais próximos do esperado ($k = 0.4$). Nesse sentido, os gráficos mostrados abaixo indicam o problema. A malha utilizada está está ilustrada em [Figura 4.51].

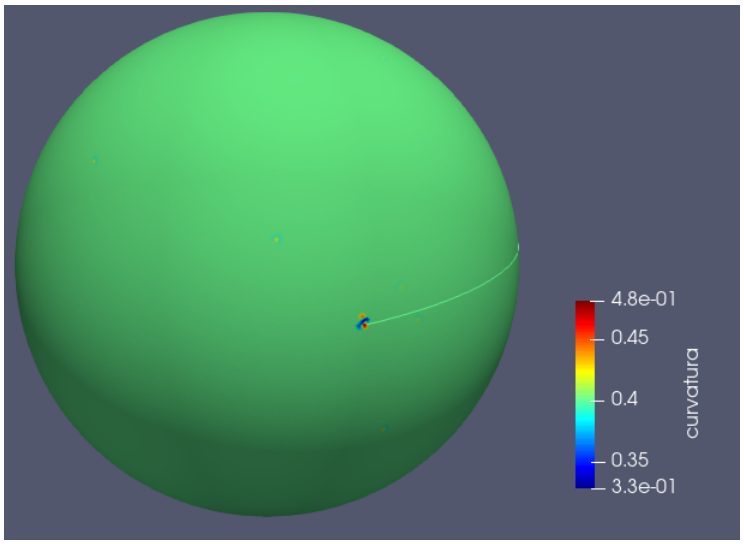


Figura 4.49: Curvatura da esfera - $R = 5$ - MVF

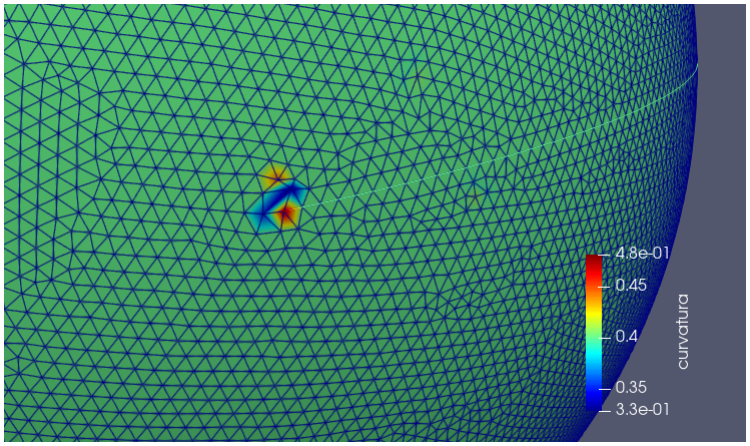


Figura 4.50: Curvatura - Foco na região de curvatura máxima - Esfera $R = 5$ - MVF

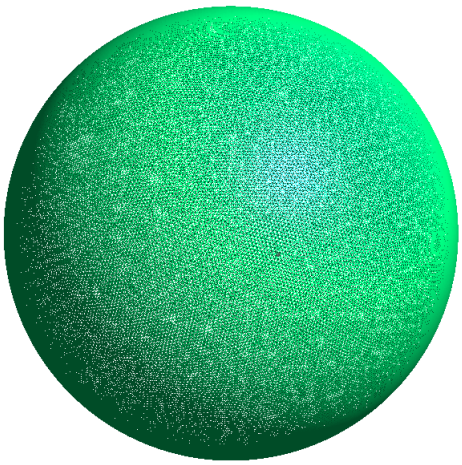


Figura 4.51: Malha 3D - Esfera $R = 5$ - $n_p = 50038$ e $n_e = 100256$

Nota-se que há um pico no valor da curvatura em um ponto específico da malha,

em uma região em que os triângulos que a compõem estão distantes de triângulos equiláteros. Assim, fica evidente o problema que o cálculo enfrenta com triângulos obtusângulos - de fato, o uso da área corrigida $\mathcal{A}_{mixed}$ é uma tentativa de contorná-lo. Não obstante a falta de convergência aparente nos valores máximo e mínimo, o Erro considerado indica que há aproximação do valor esperado na superfície.

Agora, com o intuito de avaliar o efeito utilizando as diferentes áreas, o código do MEF foi rodado com a mesma esfera:

h	Máx(k)	Mín(k)	Erro
3	0.602	0.282	0.154
1	0.631	0.297	0.088
0.5	0.598	0.265	0.069
0.1	0.955	0.241	0.041
0.05	0.617	0.280	0.034

Tabela 4.12: Resultados com $\mathcal{A}_{bari}$ - Esfera, $R = 5$

h	Máx(k)	Mín(k)	Erro
3	0.412	0.300	0.071
1	0.401	0.303	0.026
0.5	0.400	0.279	0.022
0.1	0.400	0.190	0.007
0.05	0.400	0.278	0.002

Tabela 4.13: Resultados com $\mathcal{A}_{voronoi}$ - Esfera, $R = 5$

Comparando com os resultados para $\mathcal{A}_{mixed}$, em [Tabela 4.11], nota-se que tal correção, em relação a $\mathcal{A}_{voronoi}$, apresenta aparentes desvantagens. A utilização da área sem correção é capaz de estabilizar o valor máximo da curvatura em seu valor real analítico. Todavia, $\mathcal{A}_{mixed}$ faz com que o valor mínimo fique mais próximo do esperado, enquanto que no outro caso esse é mais distante. No que tange o erro, não há grandes diferenças.

Quanto à área $\mathcal{A}_{bari}$, é evidente pelos resultados acima sua inferioridade para o calculo da curvatura. Esta apresenta valores de máximo e mínimo mais distantes do previsto, assim como erros maiores. Portanto, recomenda-se utilizar uma das duas opções: $\mathcal{A}_{mixed}$ ou $\mathcal{A}_{voronoi}$ dependendo do quão estruturada é a malha.

Os valores que sequem a partir desse pontos foram testados tanto para o MVF usando o loop nos elementos, assim como para o MEF. A área utilizada foi $\mathcal{A}_{mixed}$, seguindo o recomendado em [10].

Para um elipsoide genérico de equação dada por:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1 \tag{4.6}$$

A curvatura média em um ponto é dada por [70]:

$$k = \frac{|x^2 + y^2 + z^2 - a^2 - b^2 - c^2|}{\left(\frac{x^2}{a^4} + \frac{y^2}{b^4} + \frac{z^2}{c^4}\right)^{3/2} (abc)^2} \tag{4.7}$$

Para um elipsoide com $a = 1$, $b = 0.4$ e $c = 0.5$, foram obtidos os resultados:

h	Máx(k)	Mín(k)	Erro	Tempo [s]
3	4.819	2.117	0.241	0.056
1	8.302	1.932	0.110	0.147
0.5	9.163	1.990	0.088	0.450
0.1	10.202	2.000	0.022	10.851
0.05	10.267	1.915	0.015	47.229

Tabela 4.14: Resultados para o MVF com diferentes h - Elipsoide - $a = 1$, $b = 0.4$ e $c = 0.5$

h	Máx(k)	Mín(k)	Erro	Tempo [s]
3	4.819	2.117	0.241	0.113
1	8.302	1.932	0.110	0.201
0.5	9.163	1.990	0.088	0.655
0.1	10.202	2.000	0.022	12.857
0.05	10.267	1.915	0.015	51.791

Tabela 4.15: Resultados para o MEF com diferentes h - Elipsoide - $a = 1$, $b = 0.4$ e $c = 0.5$

Um exemplo das distribuições de curvatura na superfície, assim como a malha utilizada, estão mostradas nos gráficos:

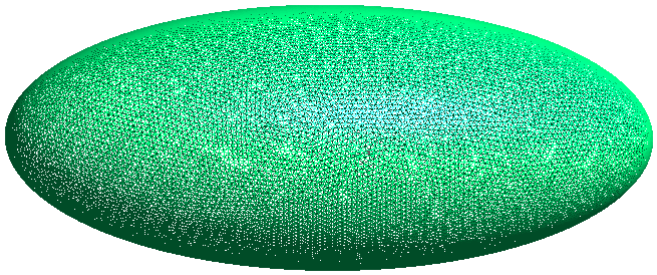


Figura 4.52: Malha 3D - Elipsoide - $n_p = 19248$ e $n_e = 38635$

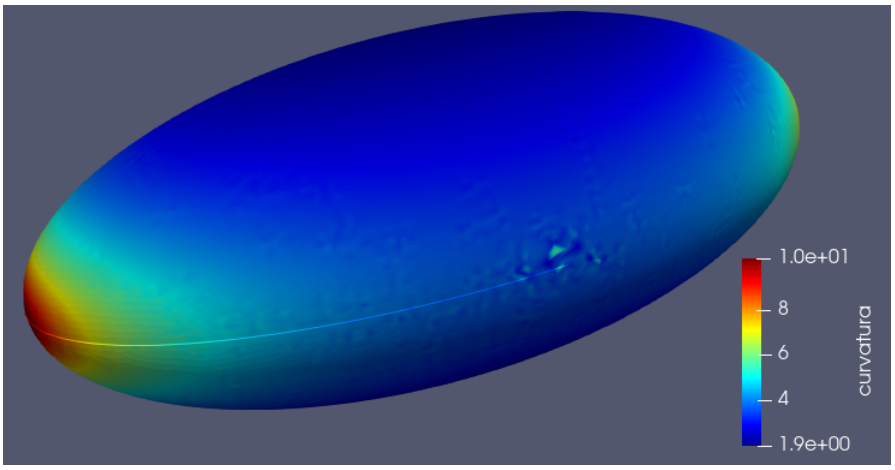


Figura 4.53: Curvatura do elipsoide - $a = 1$, $b = 0.4$ e $c = 0.5$ - MVF

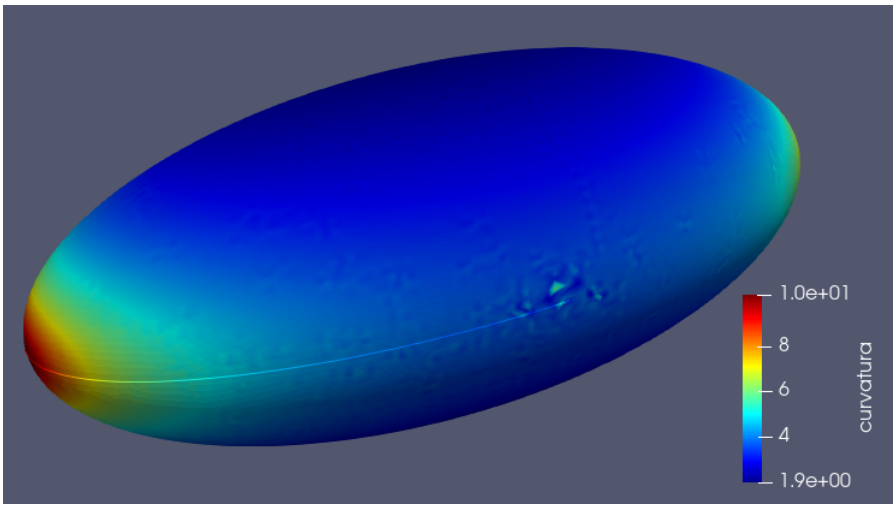


Figura 4.54: Curvatura do elipsoide - $a = 1$, $b = 0.4$ e $c = 0.5$ - MEF

Como esperado, as distribuições são exatamente iguais nos dois métodos. Além disso, é possível notar que a curvatura tem o comportamento previsto, variando com os valores dos comprimentos de seus eixos. De fato, a curvatura é maior nas direções em que a figura é mais alongada e menor quando esta se aproxima à um plano.

Outro caso com curvatura conhecida analiticamente é o toroide. Dada sua equação:

$$\left(c - \sqrt{x^2 + y^2}\right)^2 + z^2 = a^2 \tag{4.8}$$

Tem-se a curvatura em um ponto definida como [71]:

$$k = \frac{c + 2aA}{a(c + aA)} \tag{4.9}$$

em que:

$$A = \frac{x^2 + y^2 + z^2 - c^2 - a^2}{2ac} \tag{4.10}$$

Assim, utilizando $a = 0.4$ e $c = 0.75$:

h	Máx(k)	Mín(k)	Erro	Tempo [s]
3	1.310	-3.751	0.424	0.053
1	3.497	-0.450	0.061	0.319
0.5	3.773	-0.357	0.088	1.169
0.1	3.566	-0.811	0.022	28.289
0.05	3.557	-0.771	0.015	119.314

Tabela 4.16: Resultados para o MVF com diferentes h - Toroide - $a = 0.4$ e $c = 0.75$

h	Máx(k)	Mín(k)	Erro	Tempo [s]
3	1.310	-3.751	0.424	0.0125
1	3.497	-0.450	0.061	0.637
0.5	3.773	-0.357	0.088	2.246
0.1	3.566	-0.811	0.022	51.567
0.05	3.557	-0.771	0.015	125.120

Tabela 4.17: Resultados para o MEF com diferentes h - Toroide - $a = 0.4$ e $c = 0.75$

Os valores máximo e mínimo analíticos são respectivamente e 3.370 e -0.357, o que pode ser obtido observando que A varia entre -1 e 1. Abaixo são mostradas imagens da distribuição de curvatura na superfície:

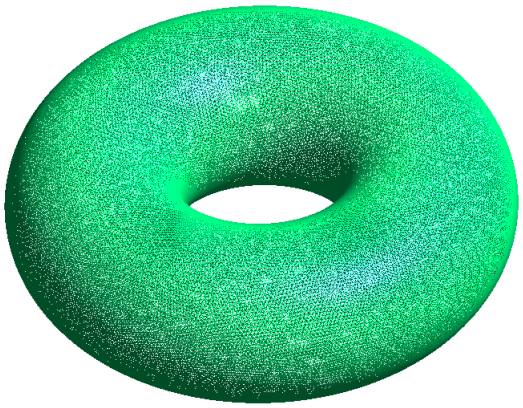


Figura 4.55: Malha 3D - Toroide - $n_p = 43376$ e $n_e = 87294$

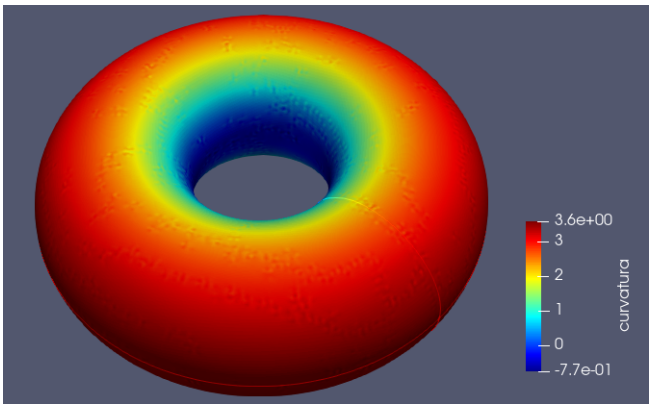


Figura 4.56: Curvatura do toroide - $a = 0.4$ e $c = 0.75$ - MVF

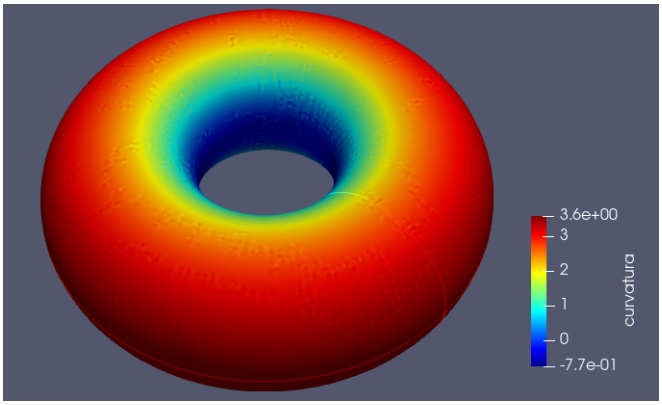


Figura 4.57: Curvatura do toroide - $a = 0.4$ e $c = 0.75$ - MEF

Outras três figuras de interesse são o hiperboloide, o cilindro e o cubo. Estas geometrias não são superfícies fechadas, tendo extensão infinita. Nesse caso, possuem uma forma de fronteira e não obedecem a definição dada de superfície regular; sendo indefinida a curvatura nessa região limite. Apesar disso, a propriedade pode ser calculada em seu interior, como será feito a seguir. Essas três formas, porém, não serão apresentadas na modelagem da interface.

No caso do cilindro, a curvatura na superfície lateral é igual a da circunferência na base $(1/r)$. Já nas faces do cubo a curvatura deve ser nula, uma vez que tratam-se de planos. A curvatura do hiperboloide, por outro lado, pode ser diretamente relacionada com a curva que é utilizada para formá-lo a partir de uma revolução.

Para um cilindro com raio da base $r = 0.4$ foram obtidos:

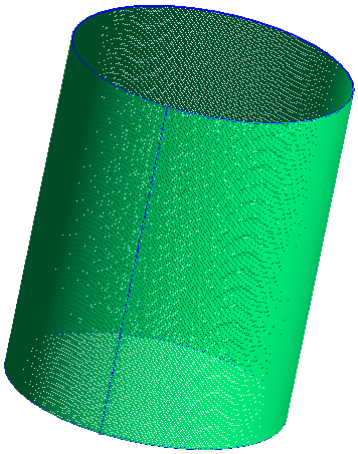


Figura 4.58: Malha 3D - Cilindro - $n_p = 43376$ e $n_e = 87294$

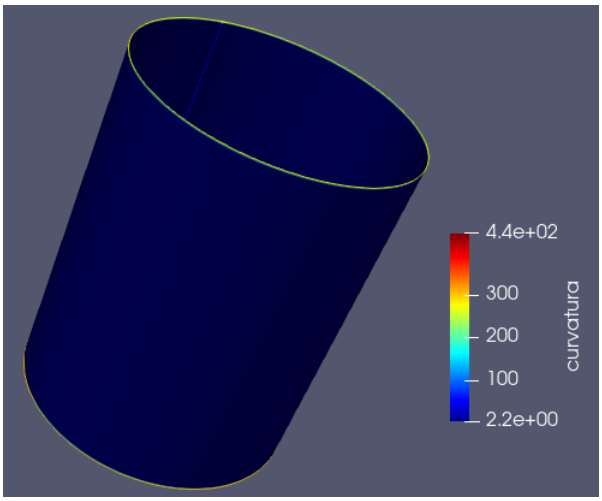


Figura 4.59: Curvatura do cilindro - $r = 0.4$ - MVF

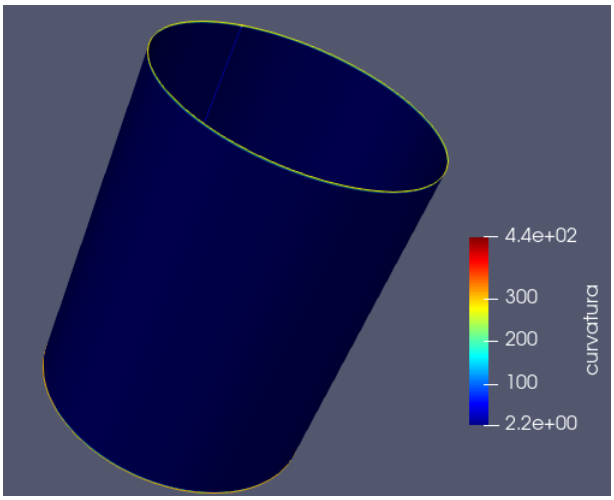


Figura 4.60: Curvatura do cilindro - $r = 0.4$ - MEF

Já para um cubo de lado $l = 1$:

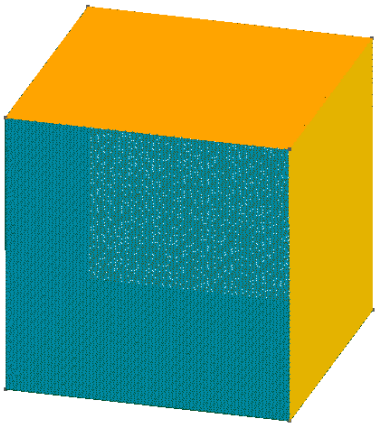


Figura 4.61: Malha 3D - Cubo - $n_p = 93461$ e $n_e = 188318$

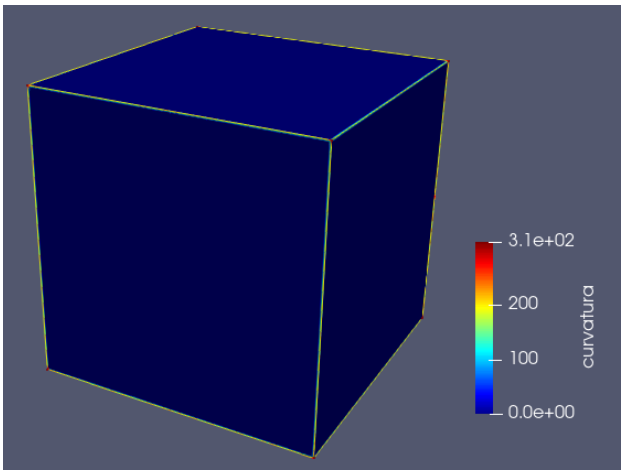


Figura 4.62: Curvatura do cubo - $l = 1$ - MVF

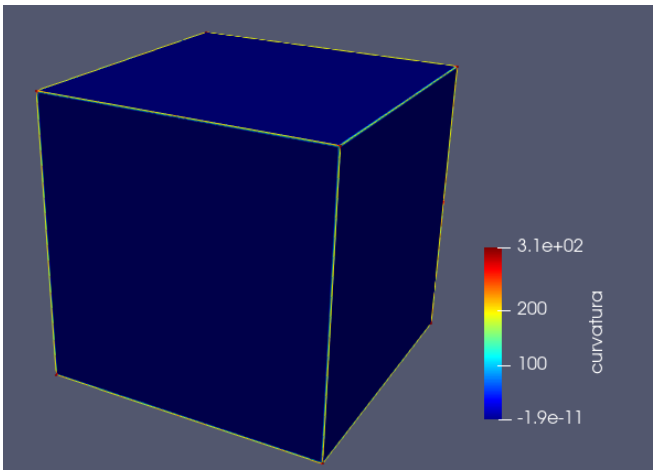


Figura 4.63: Curvatura do cubo - $l = 1$ - MEF

Por fim, foi utilizado um "pseudo hiperboloide", formado a partir da revolução de um arco de circunferência de raio $r = 0.5$ e de centro distante 0.8 do eixo de

revolução. Destaca-se que tal superfície não é de fato um hiperboloide, pois este deve ser formado a partir da revolução de uma hipérbole, que não é o caso. Por sua formação, a curvatura mostrada é equivalente a da parte interna de um toroide com $a = 0.5$ e $c = 0.8$.

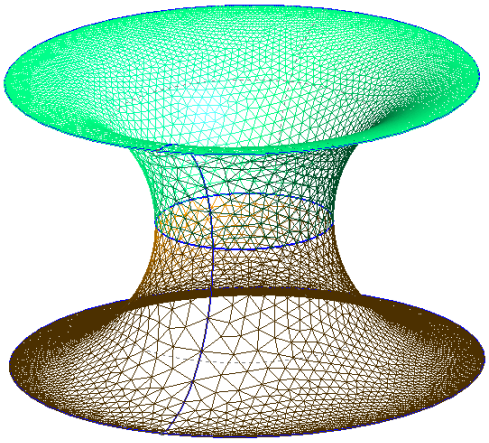


Figura 4.64: Malha 3D - Pseudo hiperboloide - $n_p = 10406$ e $n_e = 20062$

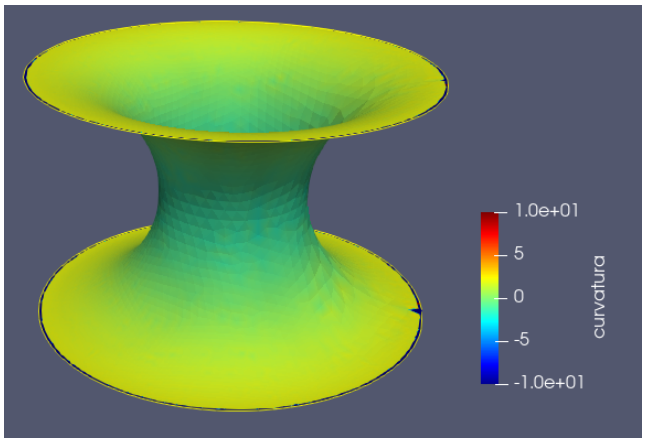


Figura 4.65: Curvatura do pseudo hiperboloide - MVF

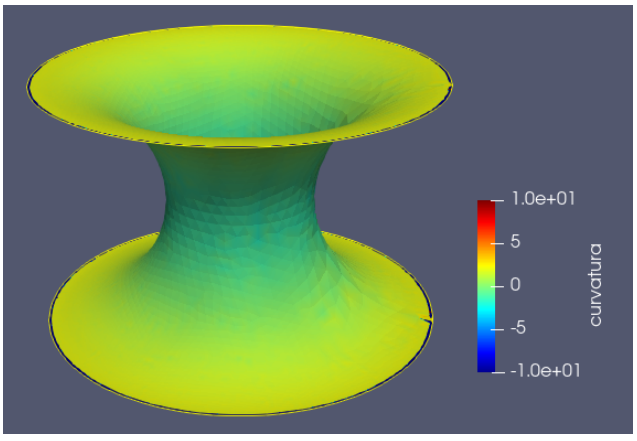


Figura 4.66: Curvatura do pseudo hiperboloide - MEF

As escalas nas duas figuras anteriores foram limitadas ao intervalo [-10,10]. De fato, evidencia-se que, em pontos de curvatura não definida, caso o código tente calculá-la, este acaba por obter um valor extremamente alto. Nesse sentido, é importante utilizar superfícies que sejam regulares ou que alguns pontos necessários sejam excluídos da malha gerada.

4.2.2 Forças de Tensão Superficial 3D

Abordagem Lagrangiana

A seguir, serão apresentados os resultados da modelagem da interface com a malha acoplada, segundo uma abordagem lagrangiana. Estes serão feitos utilizando algumas das geometrias apresentadas na seção anterior, assim como algumas outras que serão propostas.

Novamente, o valor da tensão σ foi considerado como unitário, de forma que o valor absoluto da força encontrado em si não possui tanta relevância física. Já sua distribuição, assim como sua ordem de grandeza, podem ser analisados propriamente.

No primeiro resultado produzido foi utilizada a esfera de raio 0.4, como segue nas próximas figuras. Para obter o efeito desejado de modelagem dos dois fluidos, a geometria foi inserida em um paralelepípedo.

Obteve-se, utilizando uma malha com $n_p = 21064$ e $n_e = 126124$:

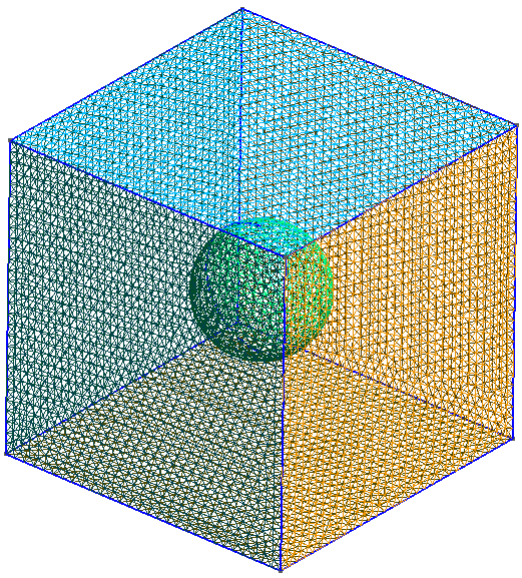


Figura 4.67: Malha acoplada - Arestas 2D - Esfera

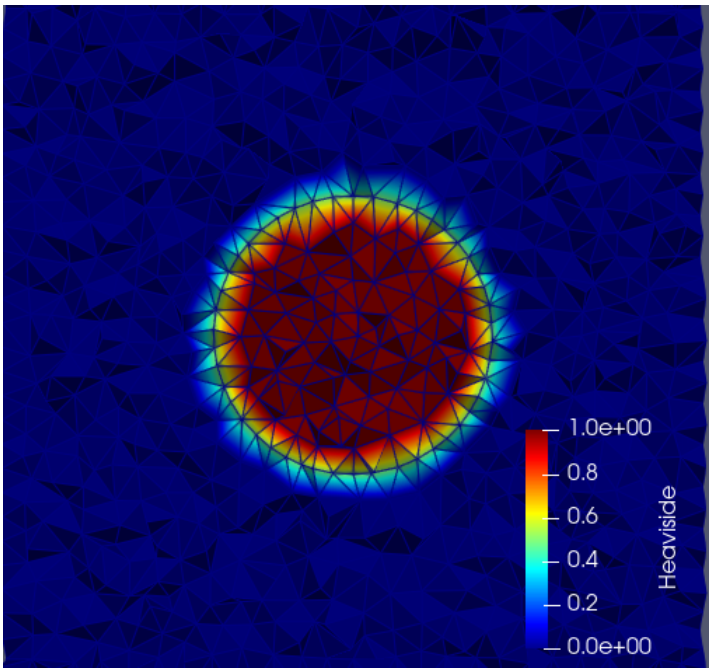


Figura 4.68: Função de Heaviside para malha acoplada 3D - Esfera

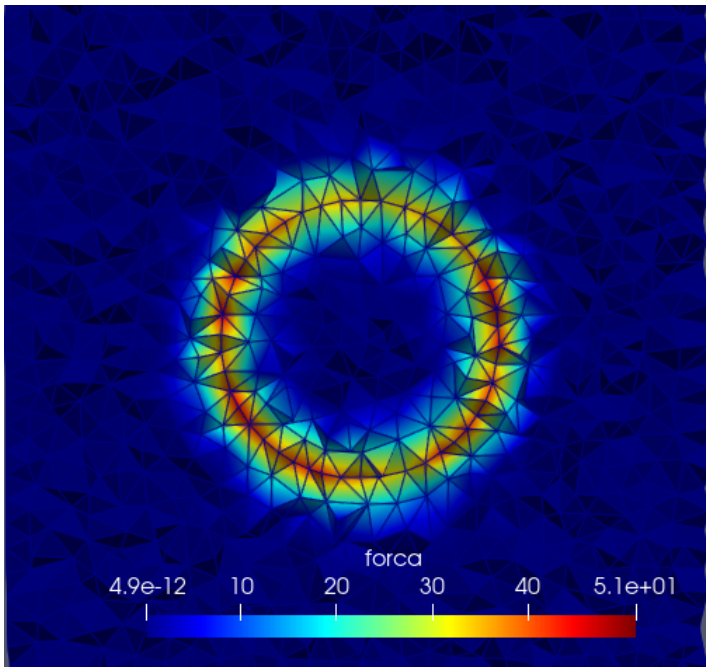


Figura 4.69: Força para malha acoplada 3D - Esfera

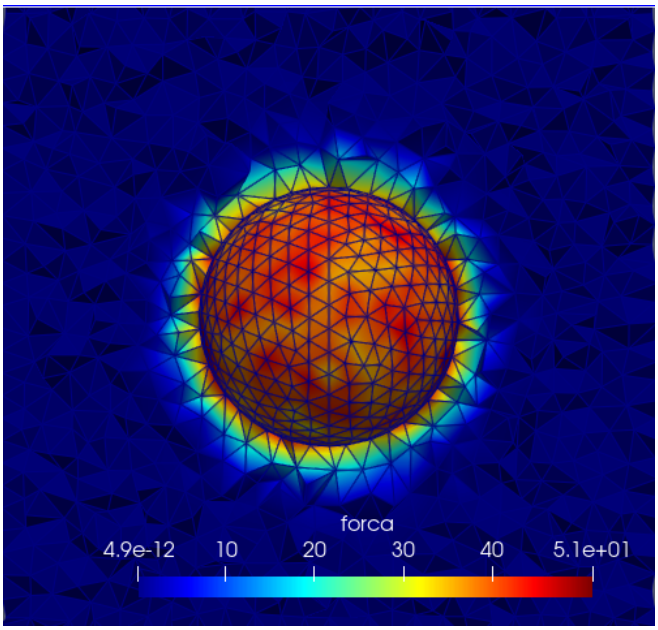


Figura 4.70: Força para malha acoplada 3D com superfície- Esfera

Na figura 4.68, pode-se verificar como foi adotada a função de Heaviside, segundo a equação 2.95. Além disso, as duas imagens para cálculo da força apresentadas diferem quanto à presença da superfície da esfera. Na primeira, sem a esfera, é possível ver o efeito da força no interior. Já na segunda imagem, é evidenciada a superfície a partir da qual foi calculada a curvatura, como explicado anteriormente.

Para o elipsoide, com uma malha de $n_p = 15186$ e $n_e = 90355$:

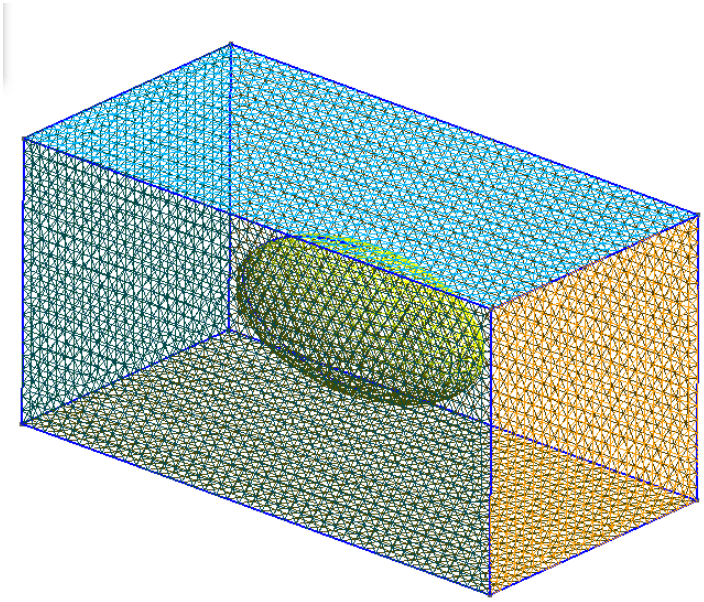


Figura 4.71: Malha acoplada - Arestas 2D - Elipsoide

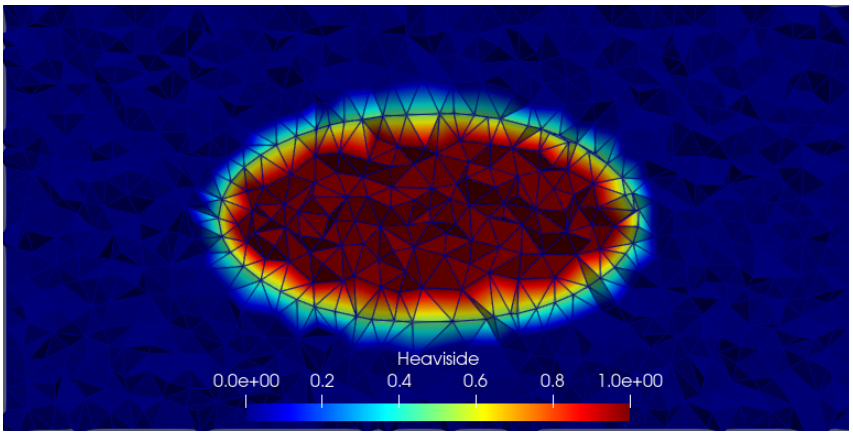


Figura 4.72: Função de Heaviside para malha acoplada 3D - Elipsoide

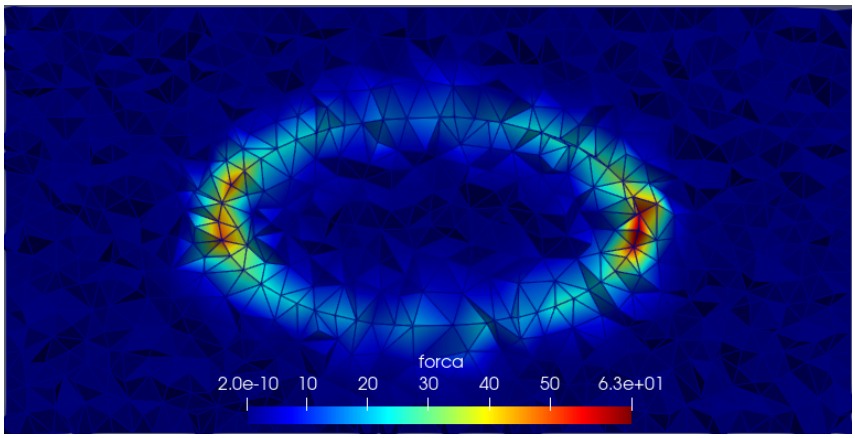


Figura 4.73: Força para malha acoplada 3D - Elipsoide

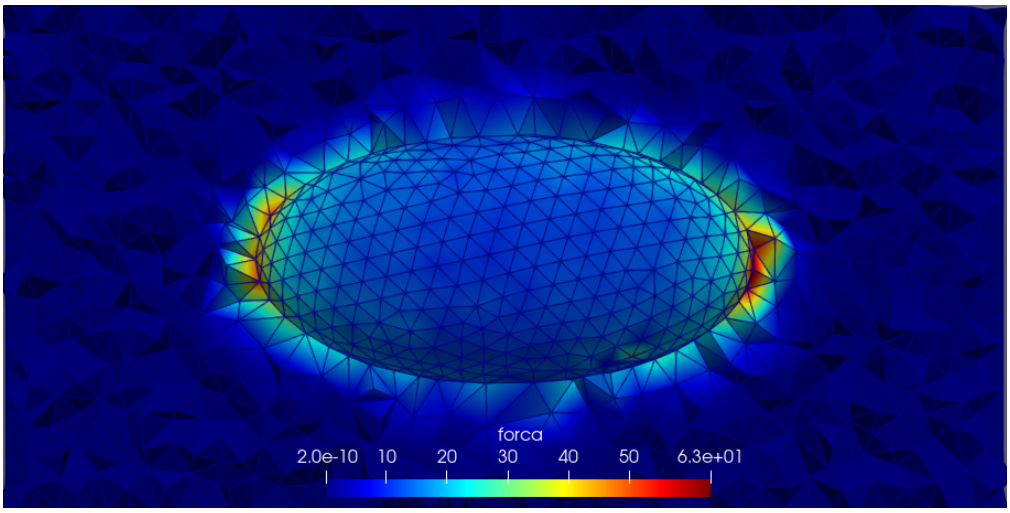


Figura 4.74: Força para malha acoplada 3D com superfície- Elipsoide

Já para o toroide, a seguinte malha foi gerada e o resultado para força obtido utilizando $n_p = 1759$ e $n_e = 3841$:

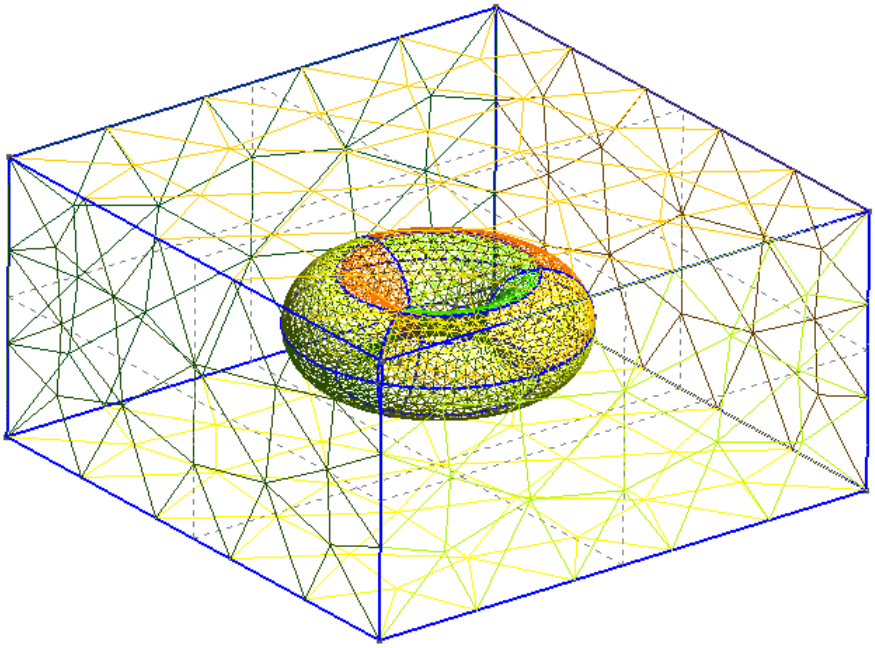


Figura 4.75: Malha acoplada - Arestas 2D - Toroide

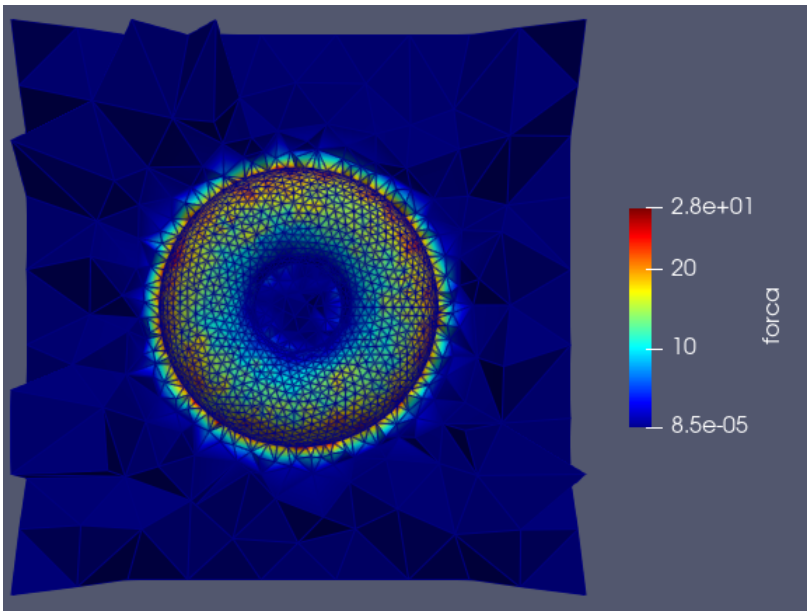


Figura 4.76: Força para malha acoplada 3D - Vista Superior - Toroide

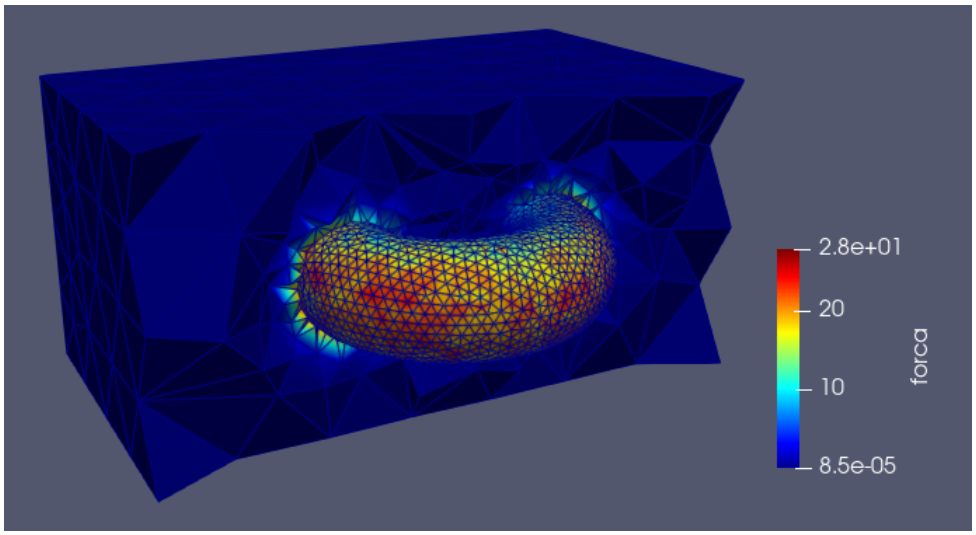


Figura 4.77: Força para malha acoplada 3D - Toroide

Optou-se por utilizar uma malha mais refinada na superfície do toroide e de pior qualidade nas paredes do paralelepípedo, de forma a dar mais atenção à região da interface sem exigir muito esforço computacional.

Outros exemplos testados foram o escoamento de uma bolha em uma tubo, em que utilizou-se um formato similar ao de uma bolha de Taylor. Também uma figura com múltiplas esferas, representando as bolhas, e outra com a coalescência de bolhas foram geradas.

Para o primeiro exemplo, utilizando uma malha de $n_p = 14152$ e $n_e = 77481$:

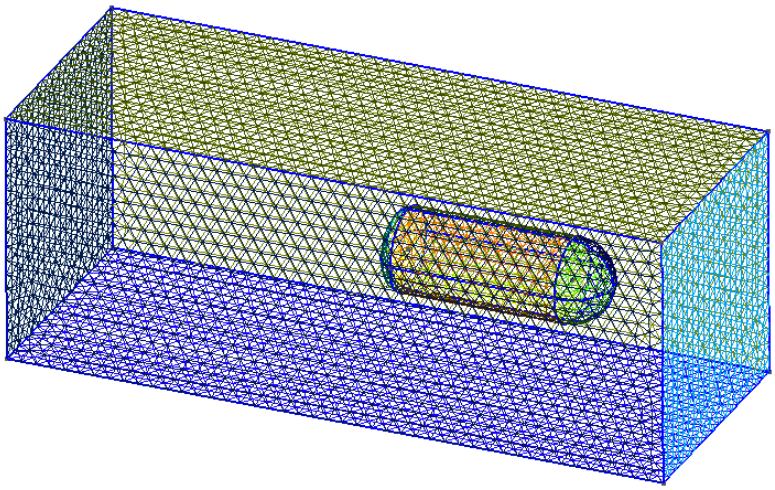


Figura 4.78: Malha acoplada - Bolha de Taylor

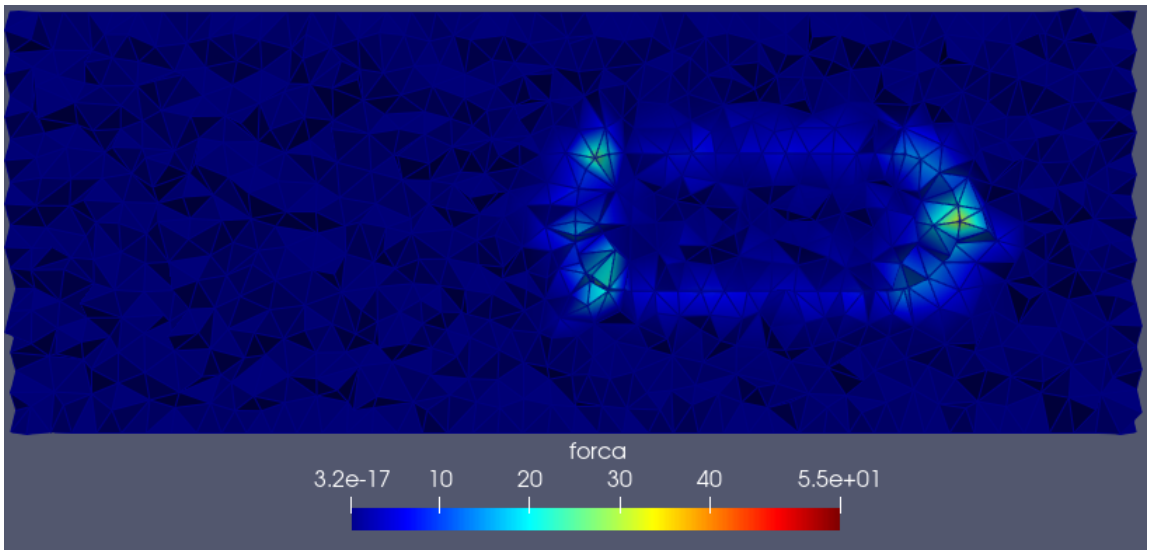


Figura 4.79: Força para malha acoplada 3D - Bolha de Taylor

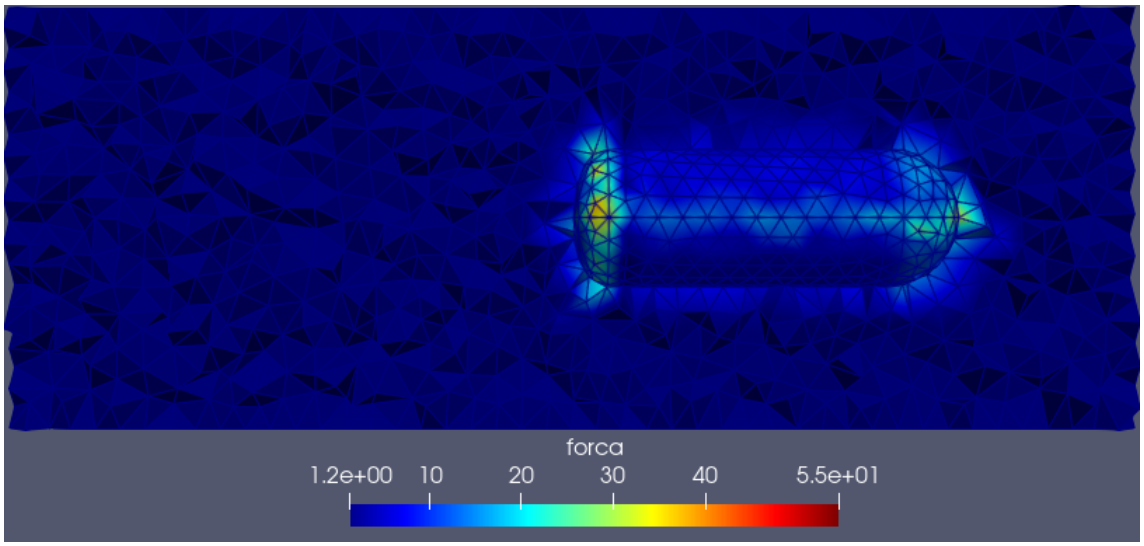


Figura 4.80: Força para malha acoplada 3D com superfície - Bolha de Taylor

Como esperado, a curvatura assume valores maiores nas regiões frontal e traseira da bolha. Em particular, a superfície é dada pela combinação de um cilindro com calotas esféricas/elipsoidais. Nesse caso, a parte traseira, por ter menor raio, faz com que esteja presente uma força maior.

Para o caso de múltiplas bolhas, foi criada uma malha com $n_p = 15241$ e $n_e = 90324$:

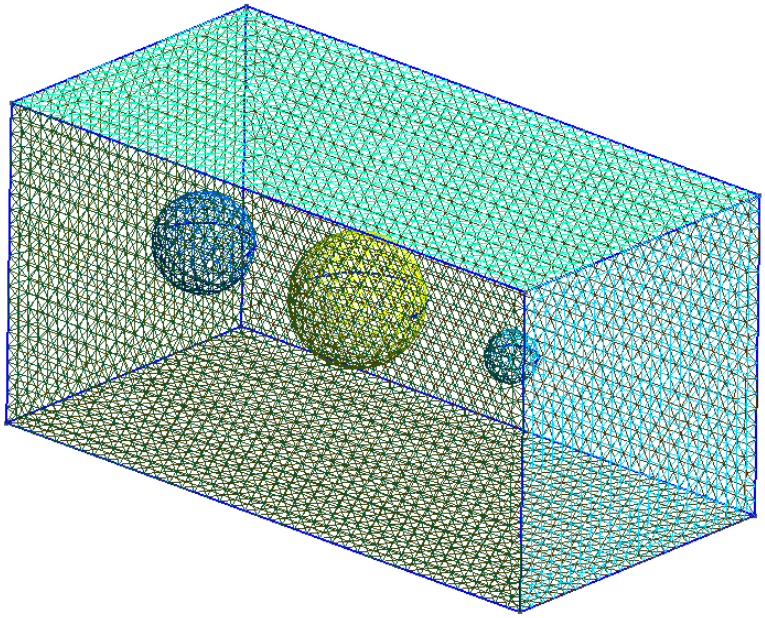


Figura 4.81: Malha acoplada - Múltiplas Bolhas

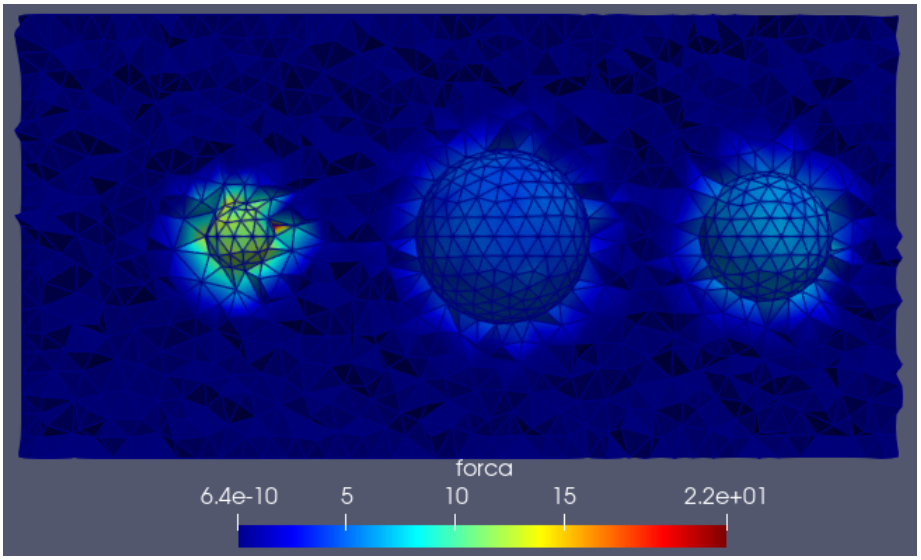


Figura 4.82: Força para malha acoplada 3D - Múltiplas Bolhas

Novamente como esperado, a força de tensão superficial, por sua proporcionalidade com a curvatura, assume valores maiores quanto menor o raio das bolhas.

Por fim, para a coalescência, com uma malha de $n_p = 32584$ e $n_e = 215792$:

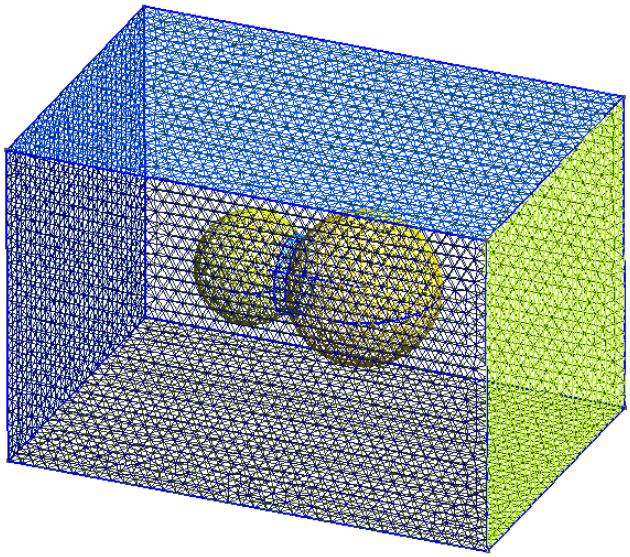


Figura 4.83: Malha acoplada - Coalescência

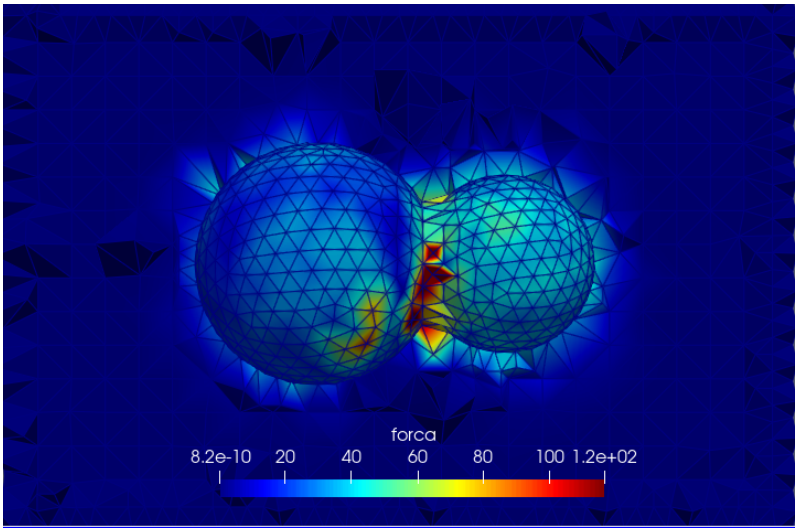


Figura 4.84: Força para malha acoplada 3D - Coalescência

Verifica-se que, geometricamente, a coalescência representada é uma união entre esferas e um "pseudo hiperboloide". Assim, a curvatura aumenta na região de junção das duas bolhas esféricas, devido à formação de um pequeno raio. Além disso, a força é menor na bolha com maior raio, como esperado.

Abordagem Euleriana

As mesma geometrias utilizadas para o caso da malha acoplada serão, agora, apresentadas com uma modelagem da malha desacoplada. As imagens mostram os resultados obtidos para a força, em que pode-se perceber como a interface é dada

com uma espessura maior - em todos os casos foi utilizado $\epsilon = 0.25$ e a função de Heaviside suavizada como em [3.19]. Estão ilustradas, também, as arestas dos elementos 2D da malha.

Optou-se por não apresentar a superfície em que foi calculada a curvatura nas imagens referentes à força, uma vez que tal grandeza não é calculada nessa região - que não faz parte da malha principal dos fluidos. Não obstante, a geometria da malha inicial da superfície pode ser vista nas primeiras imagens de cada caso.

Para a esfera, utilizando uma malha com $n_p = 21719$ e $n_e = 128161$:

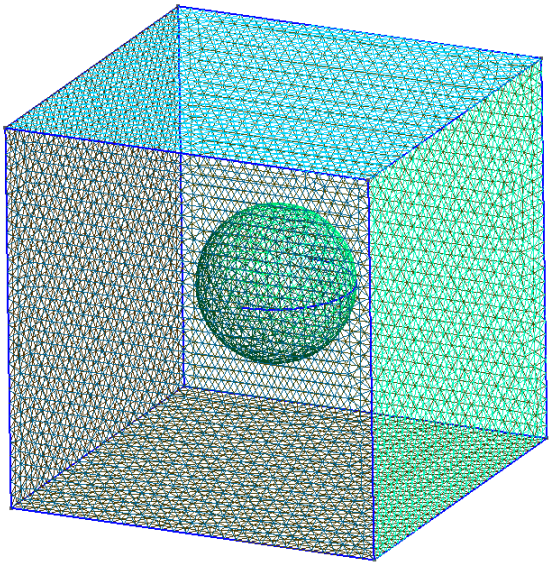


Figura 4.85: Malha desacoplada - Arestas 2D - Esfera

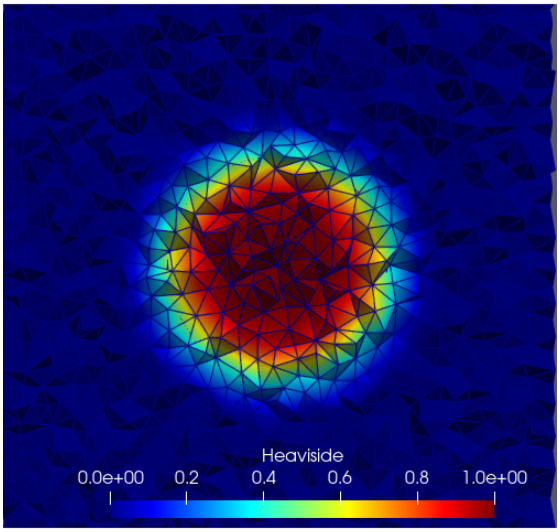


Figura 4.86: Função de Heaviside para malha desacoplada 3D - Esfera

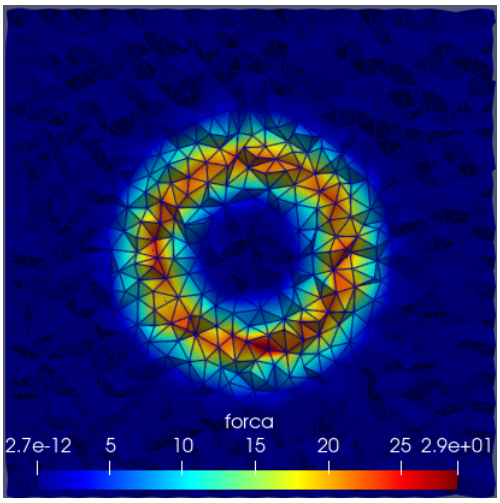


Figura 4.87: Força para malha desacoplada 3D - Esfera

Para o caso do elipsoide, em que foi utilizada uma malha com $n_p = 16383$ e $n_e = 95748$:

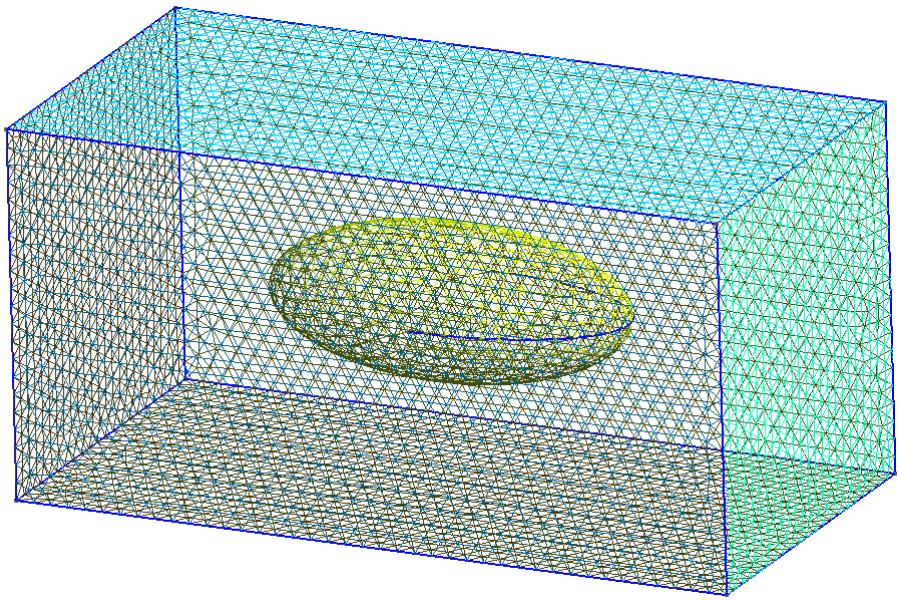


Figura 4.88: Malha desacoplada - Arestas 2D - Elipsoide

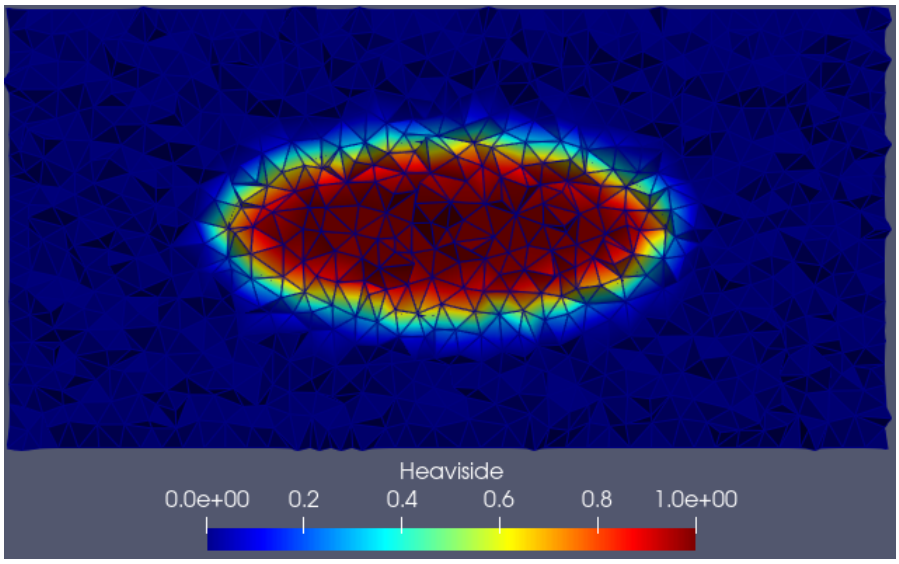


Figura 4.89: Função de Heaviside para malha desacoplada 3D - Elipsoide

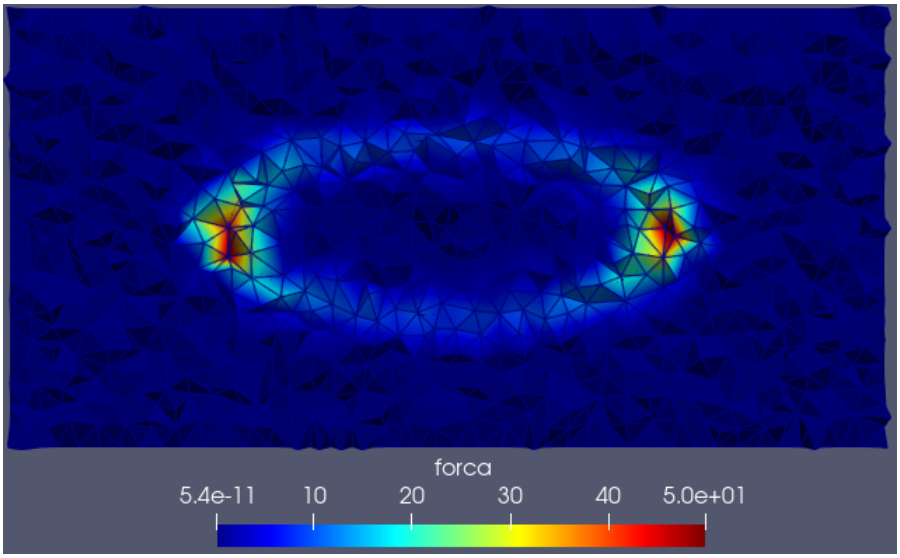


Figura 4.90: Força para malha desacoplada 3D - Elipsoide

Para o toroide, com $n_p = 18069$ e $n_e = 91985$:

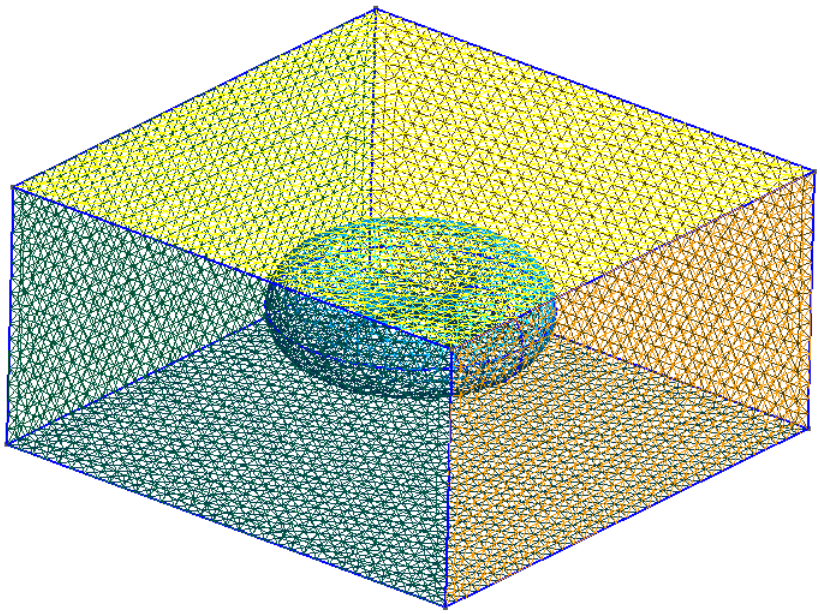


Figura 4.91: Malha desacoplada - Arestas 2D - Toroide

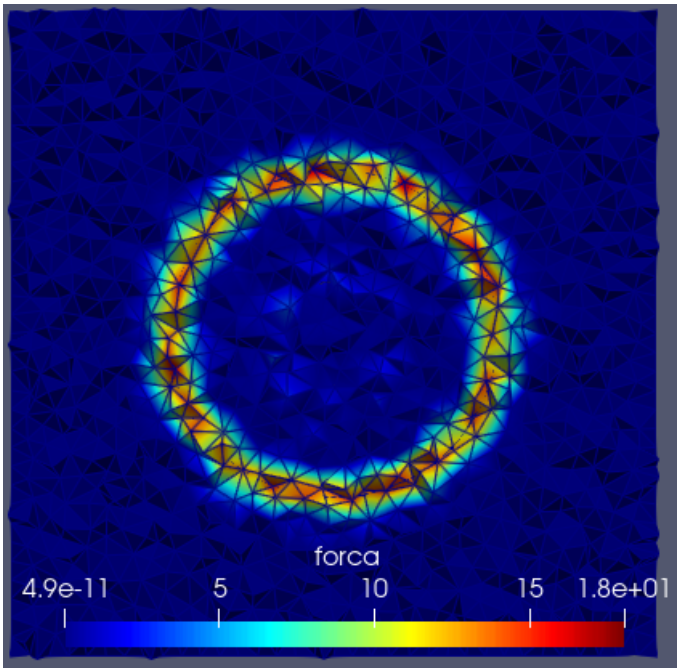


Figura 4.92: Força para malha desacoplada 3D - Vista Superior - Toroide

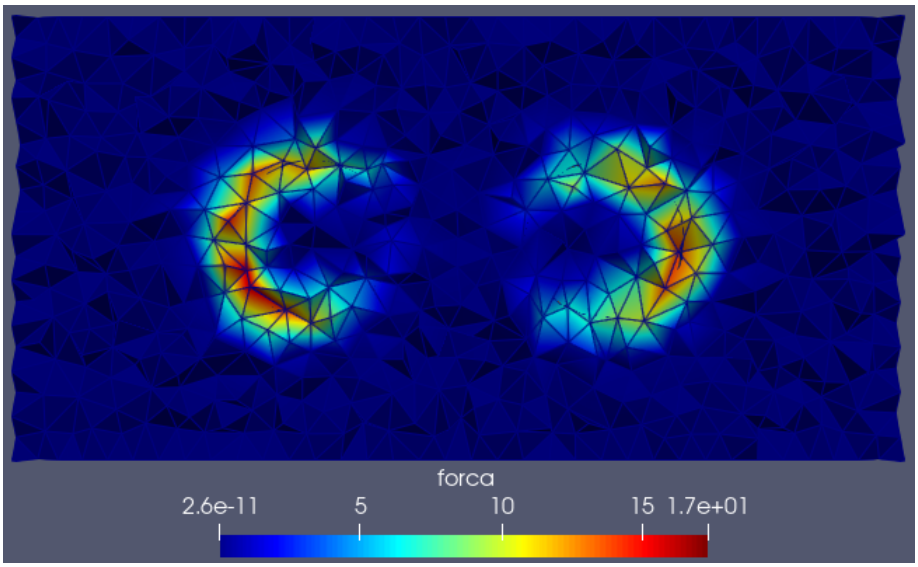


Figura 4.93: Força para malha desacoplada 3D - Toroide

Para o caso representando o escoamento de uma bolha de Taylor, utilizando uma malha de $n_p = 13093$ e $n_e = 71521$:

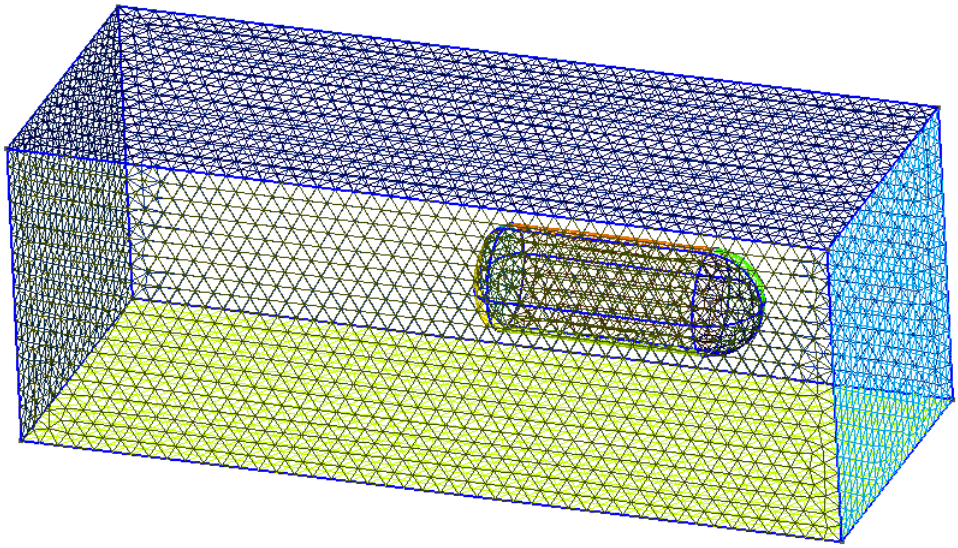


Figura 4.94: Malha desacoplada - Bolha de Taylor

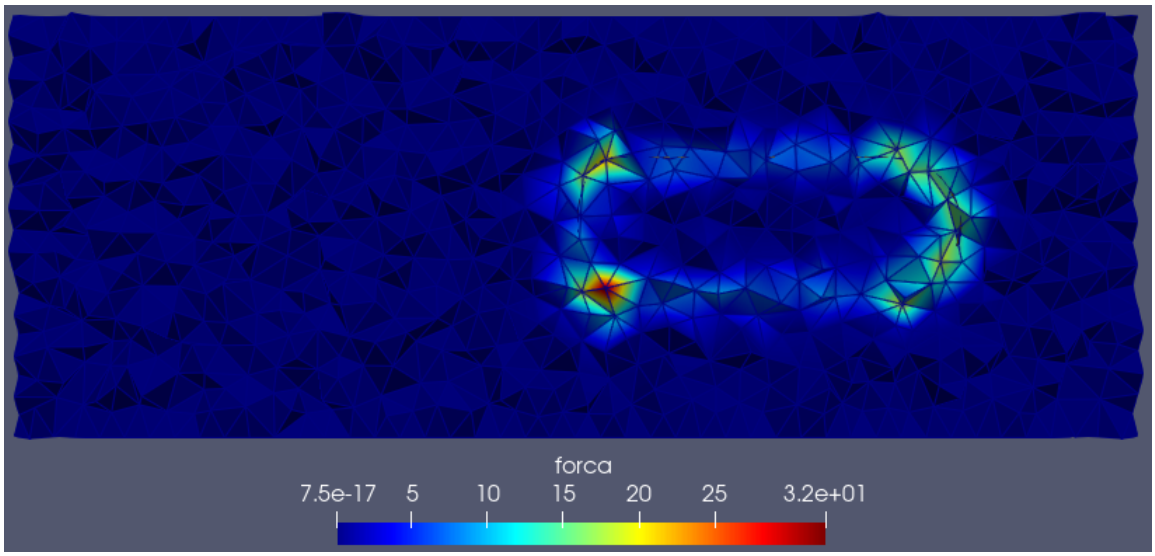


Figura 4.95: Força para malha desacoplada 3D - Bolha de Taylor

Agora, no caso com três bolhas de diâmetros diferentes $n_p = 16072$ e $n_e = 93936$:

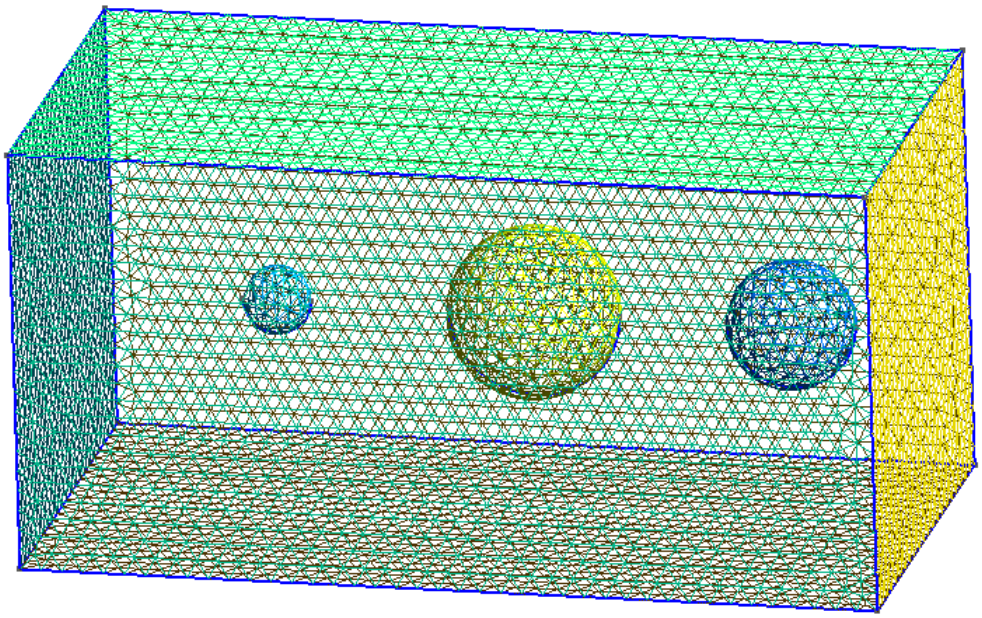


Figura 4.96: Malha desacoplada - Múltiplas Bolhas

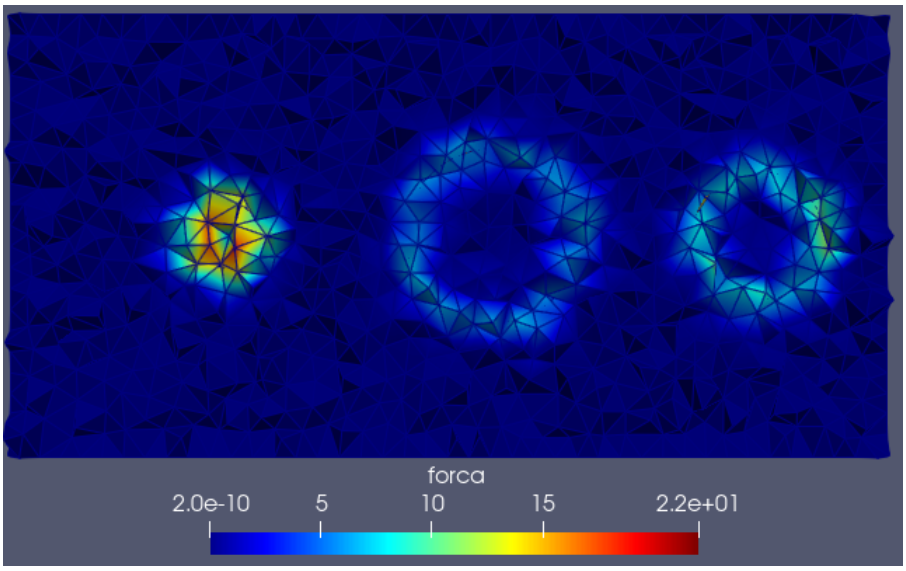


Figura 4.97: Força para malha desacoplada 3D - Múltiplas Bolhas

Já na simulação que representa a coalescência de bolhas, com uma malha de $n_p = 20948$ e $n_e = 123190$:

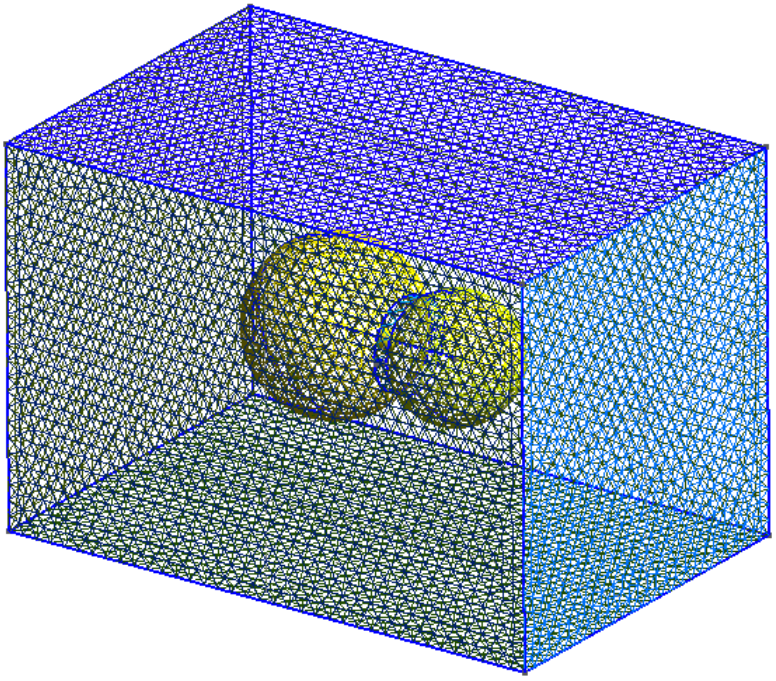


Figura 4.98: Malha desacoplada - Coalescência

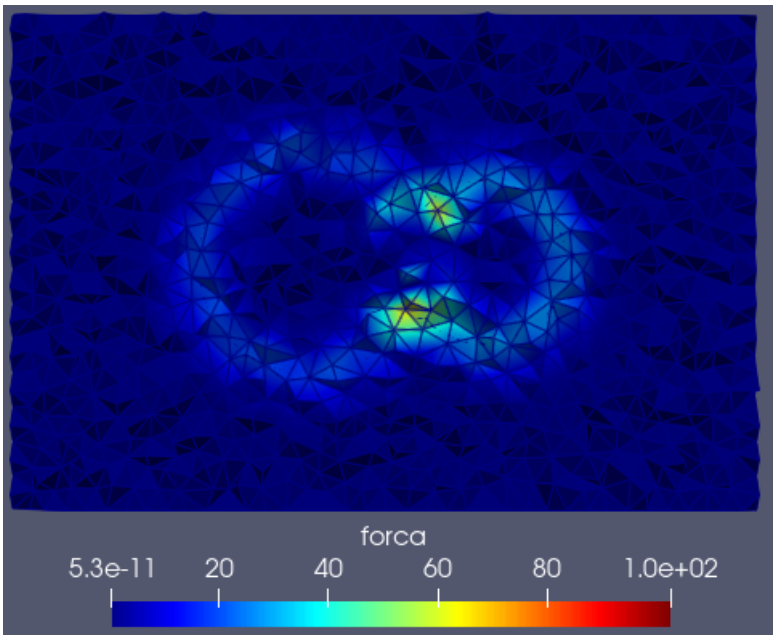


Figura 4.99: Força para malha desacoplada 3D - Coalescência

Independentemente da abordagem - Lagrangiana e Euleriana, com malha respectivamente acoplada e desacoplada - ambas foram capazes de representar bem a interface entre os fluidos. De fato, como no caso 2D, nota-se que esta segunda abordagem leva à uma difusividade da região em questão, que passa a ser definida por um conjunto de elementos. Como indicado, essa modelagem é mais adequada para fenômenos como quebra ou coalescência, uma vez que esta não exige uma malha tão precisa na interface, apesar de ser necessária uma quantidade maior de elementos 3D, em geral. Também nos exemplos tridimensionais, o valor máximo da força se mostrou maior no caso da malha acoplada, o que deve estar ligado à função de Heaviside.

Não obstante as diferenças presentes, ainda é possível notar os comportamentos esperados em cada caso, assim como destacados na abordagem anterior. Isto é, por sua relação com a curvatura, nas regiões em que há um raio menor - como no caso de bolhas pequenas ou na junção da coalescência - a força assume valores mais altos.

Além disso, nessa segunda abordagem, a utilização de uma função de Heaviside suavizada é importante; caso contrário, a representação da interface poderia ser muito pouco realista, com diversas descontinuidades. Da forma com que foram implementadas ambas as abordagens, os exemplos acima evidenciam a capacidade de modelagem dos fenômenos físicos em escoamentos bifásicos, como desejado.

Capítulo 5

Conclusão

No presente projeto, objetivou-se o desenvolvimento de um código em linguagem *Python* capaz de calcular o termo de forças de tensão superficial no contexto de escoamentos bifásicos.

Nesse sentido, foi apresentado o embasamento teórico relativo aos conceitos matemáticos fundamentais utilizados na metodologia apresentada. Assim, foram introduzidas as noções de curva e superfície baseadas em uma caracterização matemática rigorosa, dando destaque à definição de curvatura. A partir disso, foram apresentados os três métodos numéricos de interesse: Diferenças Finitas, Elementos Finitos e Volumes Finitos.

As forças de tensão superficial foram então discutidas, dando um breve destaque ao seu desenvolvimento histórico e à equação de Young-Laplace, que revela a dependência entre as forças em questão e a curvatura da forma dos fluidos. Assim, foi apresentado o modelo de CSF, como originalmente proposto por [48], como equação base para os modelos desenvolvidos. Além disso, foram discutidos os diferentes métodos de modelagem da interface em escoamentos bifásicos, destacando as abordagens Lagrangiana e Euleriana, de malha acoplada e desacoplada.

A metodologia apresentada para o cálculo das forças, portanto, foi considerada a partir de diferentes resultados já conhecidos de outros trabalhos, combinando-os. Para o caso 2D, foi utilizado um esquema baseado nas equações de Frenet para obtenção da curvatura. Similarmente, empregou-se o método de Volumes Finitos para o cálculo 3D. Ambos foram baseados no proposto em [8], mas utilizando modificações, seguindo o sugerido em [10]. Em particular, foi empregada uma área

diferente de $\mathcal{A}_{bari}$.

Além dos métodos acima, foi explicado o desenvolvimento de outra metodologia matematicamente equivalente, mas baseada na discretização do operador Laplace-Beltrami. Esta seguiu o método de Elementos Finitos e o Método de Galerkin para discretização e obtenção de uma forma fraca do operador, sendo aplicável tanto em problemas bidimensional quanto tridimensionais.

Por fim, ambas as abordagens com malha acoplada e desacoplada foram discutidas, apresentando modificações necessárias à equação do CSF, introduzindo uma função de Heaviside para representar a descontinuidade entre os fluidos, além de sua forma fraca baseada, como explorada no MEF.

Assim, foi apresentada uma vasta quantidade de testes para os diferentes modelos. No caso 2D, discutiu-se o cálculo da curvatura utilizando geometrias com curvatura analítica conhecida: circunferência, elipse e quadrado. Em todos, os resultados se mostraram satisfatórios. A partir disso, foram realizadas simulações para cálculo das forças de tensão superficial com ambas as modelagens da malha. Estas se mostraram corretas em capturar o efeito da força na interface, em que também foi evidenciada a diferença entre a interface "precisa"e a "difusa".

Similarmente, no caso 3D, foram exploradas figuras geométricas simples para validação do cálculo da curvatura: esfera, elipsoide e toroide, entre outros. Os resultados se mostraram adequados e foi apresentada a melhora proporcionada por utilizar a área $\mathcal{A}_{mixed}$ ao invés de $\mathcal{A}_{bari}$. No mesmo sentido, comentou-se quanto ao uso de $\mathcal{A}_{Voronoi}$, que garante estabilidade no valor máximo, mas uma piora nos valores mínimos. Ficando evidente, portanto, que a área utilizada nos cálculos influencia fortemente nos resultados obtidos.

Novamente, foram utilizadas as figuras geométricas para cálculo da curvatura, assim como novas geometrias que representam fenômenos físicos, para simulações quanto à representação da interface. Tanto a abordagem utilizando a malha acoplada, quanto a que utilizou a malha desacoplada, apresentaram resultados pertinentes.

Destacou-se, também, a devida atenção que deve ser dada à malha utilizada, sendo sua geração correta fundamental para aplicação dos métodos. Em geral, a abordagem Laplaciana apresenta maior dificuldade de representar geometrias com-

plexas por exigir uma malha mais fina na interface, o que não ocorre na abordagem Euleriana. Contudo, esta segunda pode requerer mais elementos na malha como um todo.

Um ponto de interesse para projetos futuros é a utilização de elementos mais genéricos para o cálculo da curvatura em modelos tridimensionais. Foi utilizado somente o elemento de triângulo linear, enquanto que o emprego de quadriláteros, por exemplo, pode ser interessante - ainda mais com uma malha não estruturada. O problema resta na questão da área discutida, que ainda não possui um equivalente para formas arbitrárias conhecido na literatura. Acredita-se que um esquema baseado na ideia do Diagrama de Voronoi ou na de pseudo-circuncentro seja adequado; investigações nesse sentido foram deixada para trabalhos futuros.

Como mencionado, o presente projeto não se ateve à utilização do código desenvolvido para a simulação de um escoamento. Em projetos futuros, tal aplicação pode ser realizada a partir dos códigos numéricos disponibilizados no Apêndice A. Não obstante, o presente trabalho serve como ferramenta de fácil empregabilidade em aplicação diversas que exigem o cálculo da curvatura média discreta. Ao mesmo tempo, é disponibilizado um aparato que consegue representar satisfatoriamente a interface entre as fases de um escoamento bifásico, sendo possível utilizar as duas abordagens propostas, ficando a cargo da aplicação em questão.

Referências Bibliográficas

[1] LOWER, S. K., "Liquids and their vapors", .

[2] ALVES, R. F., *Estudo experimental do escoamento bifásico líquido gás em golfadas com leve mudança de direção*, Master's Thesis, 2015.

[3] CARMO, M. P. D., *Geometria Diferencial de Curvas e Superfícies*. 6th ed. SBM - Sociedade Brasileira de Matemática, 2012.

[4] BATHE, K.-J., *Finite Element Procedures*. 2nd ed. Prentice Hall, Pearson Education, Inc., 2014.

[5] ALEKSENDRIĆ, D., CARLONE, P., *Soft Computing in the Design and Manufacturing of Composite Materials*. Woodhead Publishing, 2015.

[6] SIQVELAND, L. M., SKAEVELAND, S. M., "Derivations of the Young-Laplace Equation", 2013.

[7] ANJOS, G. R., "Moving mesh methods for two-phase flow systems: Assement, comparison and analysis", *Computer and Fluids 228*, 2021.

[8] DOS ANJOS, G. R., *A 3D ALE Finite Element Method for Two-Phase Flows with Phase Change*, Ph.D. Thesis, École Polytechnique Fedéréle de Lusanne, 2012.

[9] BEUTEL, A., "Interactive Voronoi Diagram Generator with WebGL", Disponível em http://alexbeutel.com/webgl/voronoi.html, acessado em 16/08/2021.

[10] MEYER, M., DESBRUN, M., SCHRÖDER, P., et al., "Discrete Differential-Geometry Operators for Triangulated 2-Manifolds", .

[11] BUSH, J. W. M., "18.357 Interfacial Phenomena, Fall 2010", 2013.

[12] ISRAELACHVILI, J. N., *Intermolecular and Surface Forces*. 3rd ed. Elsevier, 2011.

[13] EHRIG, S., SCHAMBERGER, B., BIDAN, C. M., et al., "Surface tension determines tissue shape and growth kinetics", 2019.

[14] LECUIT, T., LENNE, P.-F., "Cell surface mechanics and the control of cell shape, tissue patterns and morphogenesis", 2007.

[15] CALLENS, S. J. P., UYTTENDAELE, R. J. C., FRATILA-APACHITEI, L. E., et al., "Substrate Curvature as a cue to guide spatiotemporal cell and tissue", *Biomaterial 232*, 2020.

[16] BELTRÁN-HEREDIA, E., ALMENDRO-VEDIA, V. G., MONROY, F., et al., "Modeling the Mechanics of Cell Division: Influence of Spontaneous Membrane Curvature, Surface Tension, and Osmotic Pressure", 2017.

[17] SCHMELZER, J. W. P., ABYZOV, A. S., , et al., "Curvature dependence of the surface tension and crystal nucleation in liquids", *Int. J. Appl. Glass Sci.*, v. 10, pp. 57–68, 2019.

[18] ERDEMIR, D., LEE, A., MYERSON, A., "Crystal Nucleation", *Handbook of Industrial Crystallization*, pp. 76–114, 2019.

[19] SILVA, P. T. C., *Uma equação para o cálculo do atrito entre fases em escoamentos turbulentos estratificados em tubulações circulares*, Master's Thesis, 2017.

[20] ZHOU, J., *Flow Patterns in vertical air/water flow with and without srufactant*, Master's Thesis, 2013.

[21] JEREZ-CARRIZALES, M., JARAMILLO, J. E., FUENTES, D., "Prediction of Multiphase Flow in Pipelines: Literature Review", *Ingeniería y Ciencia*, 2015.

[22] FILHO, J. D. S. C., *Estudo Experimental de Escoamento Bifásico em Tubo Circular Inclinado Usando Técnicas Ultrasônicas e de Visualização*, Ph.D. Thesis, COPPE/UFRJ, 2010.

[23] STILLWELL, J., *Mathematics and Its History*. 3rd ed. Springer, 2010.

[24] MERZBACH, U. C., BOYER, C. B., *A History of Mathemathics*. 3rd ed. Wiley, 2010.

[25] MOL, R. S., *Uma Introdução à História da Matemática*. 1st ed. CEAD-UFMG, 2013.

[26] KREYSZIG, E., *Differential Geometry*. 1st ed. Dover, 1991.

[27] GRAY, A., *Modern Differential Geometry of Curves and Surfaces with Mathematica*. 1st ed. CRC Press, 1998.

[28] LEE, J. M., *Riemannian Manifolds - An Introduction to Curvature*. 1st ed. Springer, 1991.

[29] DIERKES, U., HILDEBRANDT, S., SAUVIGNY, F., *Minimal Surfaces*. 2nd ed. Springer, 2010.

[30] URAKAWA, H., "Geometry of Laplace-Beltrami Operator on a Complete Riemannian Manifold", *Advanced Studies in Pure Mathematics 22*, 1993.

[31] XU, G., "Discrete Laplace–Beltrami operators and their convergence", *Computer Aided Geometric Design 21*, 2004.

[32] BELKIN, M., SUN, J., WANG, Y., "Discrete Laplace Operator on Meshed Surfaces", 2008.

[33] TAUBIN, G., "A Signal Processing Approach To Fair Surface Design", .

[34] WANG, R., YANG, Z., LIU, L., et al., "Discretizing Laplace–Beltrami Operator from Differential Quantities", 2013.

[35] BOYCE, W. E., DIPRIMA, R. C., *Elementary Differential Equations and Boundary Value Problems*. 7th ed. John Wiley and Sons Inc., 2001.

[36] ATKINSON, K. E., *An Introduction to Numerical Analysis*. 2nd ed. John Wiley and Sons, 1989.

[37] ELON, L. L., *Curso de Análise Volume 2*. 5th ed. IMPA - Projeto Euclides, 1999.

[38] FISH, J., BELYTSCHKO, T., *Um Primeiro Curso em Elementos Finitos*. 1st ed. Wiley and Sons, 2007.

[39] LOGAN, D. L., *A First Course in the Finite Element Method*. 4th ed. Thomson, 2007.

[40] KOLDITZ, O., *Computational Methods in Environmental Fluid Mechanics*. Springer, 2002.

[41] MARCONDES, F., SEPEHRNOORI, K., "An element-based finite-volume method approach for heterogeneous and anisotropic compositional reservoir simulation", *Journal of Petroleum Sciences and Engineering 73*, 2010.

[42] YOON, J.-Y., KIM, J.-H., "Improvement of Hydrodynamic Performance of a Multiphase Pump Using Design of Experiment Techniques", *Journal Fluids Engineering 137*, 2015.

[43] POMEAU, Y., "Surface tension: from fundamental principles to applications in liquids and in solids", 2013.

[44] BIKERMAN, J. J., "Capillarity before Laplace: Clairaut, Segner, Monge, Young", 1978.

[45] LAUTRUP, B., *Physics of Continous Matter: Exotic and Everyday Phenomena in the Macroscopic World*. Institute of Physics Publishing, 2005.

[46] ADAMSON, A. W., GAST, A. P., *Physical Chemistry of Surfaces*. 6th ed. Wiley, 1997.

[47] BLOKHUIS, E. M., "Triezenberg-Zwanzig expression for the surface tension of a liquid drop", 2013.

[48] BRACKBILL, J. U., KOTHE, D. B., ZEMACH, C., "A Continuum Method for Modeling Surface Tension", 1991.

[49] MIRJALILI, S., JAIN, S. S., DODD, M. S., "Interface-capturing methods for two-phase flows: An overview and recent developments", *Center for Turbulence Research Annual Research Briefs 2017*, 2017.

[50] GANESAN, S., MATTHIES, G., TOBISKA, L., "On spurious velocities in incompressible flow problems with interfaces", .

[51] KIM, J., "A continuous surface tension force formulation for diffuse-interface models", *Journal of Computational Physics 204*, 2005.

[52] NADOOSHAN, A. A., SHIRANI, E., "Interface Pressure Model for Surface Tension Forces for VOF-Based Methods in Interfacial Flows", *Engineering Applications of Computational Fluid Mechanics Vol.2, No.4*, 2008.

[53] HIRT, C. W., NICHOLS, B. D., "Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries", *Journal of Computational Physics 39*, 1981.

[54] GINZBURG, I., WITTUM, G., "Two-Phase Flows on Interface Refined Grids Modeled with VOF, Staggered Finite Volumes, and Spline Interpolants", *Journal of Computational Physics 39*, 2001.

[55] CHEN, L., LI, Y., "A numerical method for two phase flows with an interface", *Enviromental Modeling and Software 13*, 1998.

[56] HEYNS, J. A., OXTOBY, O. F., "Modelling Surface Tension Dominated Multiphase Flows Using the VOF Approach", 2017.

[57] OSHER, S., SETHIAN, J. A., "Fronts Propagation with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations", *Journal of Computational Physics 79*, 1988.

[58] SUSSMAN, M., SMEREKA, P., OSHE, S., "A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow", *Journal of COmputational Physics 114*, 1994.

[59] SUSSMAN, M., SMEREKA, P., "Axisymmetric free boundary problems", *Journal of Fluid Mechanics 341*, 1997.

[60] KENNEDY, D., "Level Set Methods for Two-Phase Flows with FEM", 2014.

[61] NAGRATH, S., JANSEN, K., LAHEY, R. T., "Computation of incompressible bubble dynamics with a stabilized finite element level set method", *Computer Mechanics and Engineering 194*, 2005.

[62] STEPHEN, L., JOSE, A., "An Overview of Surface Tracking and Representation in Fluid Simulation", *International Journal of Advanced Computer Science and Applications Vol. 6, No. 11*, 2015.

[63] AMSDEN, A. A., HARLOW, F. H., "THE SMAC METHOD: A NUMERICAL TECHNIQUE FOR CALCULATING INCOMPRESSIBLE FLUID FLOWS", 1970.

[64] SANTOS, F. L. P., FERREIRA, V. G., TOMÉ, M. F., et al., "A marker-and-cell approach to free surface 2-D multiphase flows", *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS 70*, 2012.

[65] TORNBERG, A.-K., *Interface Tracking Methods with Application to Multiphase Flows*, Ph.D. Thesis, Kungl Tekniska Högskolan, 2000.

[66] GLIMM, J., GROOVE, J., LINDQUIST, B., et al., "The Bifurcation of Tracked Scalar Waves", 1988.

[67] SOUZA, F. S., MANGIAVACCHI, N., "A Lagrangian level-set approach for the simulation of incompressible two-fluid flows", *International Journal for Numerical Methods in Fluids 47*, 2005.

[68] DAS, S. K., CENANOVIC, M., ZHANG, J., "A Physics-Based Estimation of Mean Curvature Normal Vector for Triangulated Surfaces", *Procedings of the International Geometry Center Vol. 12*, 2019.

[69] KAMILIS, D., *Numerical Methods for PDEs in Curves and Surfaces*, Master's Thesis, 2013.

[70] POLAERT, D., SCHNIEWIND, J., JANSSENS, F., "Surface Area and Curvature of the General Ellipsoid", 2004.

[71] IRONS, M. L., "The Curvature and Geodesics of the Torus", Disponível em http://www.rdrop.com/ half/math/torus/torus.geodesics.pdf, acessado em 23/09/2021.

[72] SANTISO, E., FIROOZABADI, A., "Curvature Dependency of Surface Tension in Multicomponent Systems", *AIChE Journal*, v. 52, pp. No. 1, 2006.

[73] PRUSS, J., SIMONETT, G., "On the two-phase Navier-Stokes equations with surface tension", *Interfaces and Free Boundaries 12*, 2010.

[74] POPINET, S., "Numerical Models of Surface Tension", *Annual Review of Fluid Mechanics 2018*, 2018.

Apêndice A

Código Fonte

O código apresentado foi realizado em linguagem *Python*, rodado em *Python 3*.

Utilizou-se a versão 3.2.15 da biblioteca *meshio* - é **fundamental** para a utilização correta do código abaixo que essa seja a versão disponível no ambiente *Python*; não funcionará com lançamentos mais novos.

O código está dividido em casos 2D e 3D. O começo e o final são iguais para cada método utilizado, de forma que cada um difere em seu desenvolvimento central. No que tange os resultados apresentados no Capítulo 4, arquivos diferentes foram formados com a mesmas seção inicial e final, adequando o meio para cada caso. No final encontra-se um código referente à algumas funções criadas que foram utilizadas em outros arquivos.

Um comentário deve ser feito quanto aos arquivos de malha utilizados. Para que o código consiga lê-los corretamente, é necessário que estejam presente somente as informações corretas. Nesse sentido, para o caso de malha acoplada foram definidos três grupos físicos presente no arquivo .msh: ´´bubble"para a superfície da bolha, ´´in"para os elementos 3D da região interna da bolha e ´´out"para os elementos 3D externos à bolha. No caso com malha desacopladas foram definidos dois grupos físicos: novamente ´´bubble"para a superfície da bolha e ´´space"para todos os elementos tetraédricos.

O arquivos gerados pelo *Gmsh* possuem extensão .msh e dispõem das coordenadas de cada nó, assim como da Matriz de Conectividade (IEN) e informações sobre os domínios físicos criados. A leitura dos arquivos foi feita no código pela biblioteca *meshio*.

Modelo 2D

Cálculo da Curvatura

```
##################################################
Parte Inicial
##################################################

#importacao das bibliotecas
import numpy as np
import meshio as msh
import time

#inicializacao do tempo de execucao
start_time = time.time()

#importacao da malha
mesh = msh.read('arquivos.msh/2D/curvatura/elipse_005.msh')
#para cada arquivo de malha a linha acima deve ser modificada de
#acordo com sua localizacao e nome

#coordenadas dos nos
X = mesh.points[:,0]
Y = mesh.points[:,1]

npoints = len(X) #numero de pontos

IEN = mesh.cells['line'] #criacao da IEN

nelements = IEN.shape[0] #numero de elementos

##################################################
Curvatura 2D - Frenet
```

```
##########################################

Kf = np.zeros(npoints, dtype='double')
#vetor com curvatura em cada no

#loop nos pontos para calculo da curvatura
for i in range(npoints):
    pi = np.array([X[i],Y[i]]) #ponto atual

    i_prox = IEN[np.where(IEN[:,0]==i)][0][1]
    #numero do proximo ponto
    pi_prox = np.array([X[i_prox],Y[i_prox]])
    #coordenadas do proximo ponto

    i_ant = IEN[np.where(IEN[:,1]==i)][0][0]
    #numero do ponto anterior
    pi_ant = np.array([X[i_ant],Y[i_ant]])
    #coordenadas do ponto anterior

    #calculo de tn e do vetor normal no no
    t2 = pi_prox - pi
    t1 = pi - pi_ant
    h = (np.linalg.norm(t2)+np.linalg.norm(t1))/2

    t2 = t2/np.linalg.norm(t2)
    n2 = np.array([-t2[1],t2[0]])

    t1 = t1/np.linalg.norm(t1)
    n1 = np.array([-t1[1],t1[0]])

    t = (t2-t1)
    n = n1+n2
```

```
    sinal = np.sign(np.dot(n,t)) # sinal da curvatura

    Kf[i] = sinal*np.linalg.norm(t)/h


##############################################
Curvatura 3D - MEF
##############################################

#definicao das matrizes globais
K = np.zeros((npoints,npoints), dtype='double') #forca da curvatura
ht = np.zeros(npoints, dtype='double') #comprimento dos elementos
N = np.zeros((npoints,2), dtype='double') #vetor normal

#loop nos elementos
for e in range(0,nelements):
    #indices dos nos
    v1 = IEN[e,0]
    v2 = IEN[e,1]

    #coordenadas dos nos
    p1 = np.array([X[v1],Y[v1]])
    p2 = np.array([X[v2],Y[v2]])

    # "area", comprimento do elemento
    h = np.linalg.norm(p2-p1)



    #calculo do vetor normal
    t = p2 - p1
    n = np.array([-t[1],t[0]])
```

```
    #matriz de rigidez local
    k = 1/h * np.array([[1, -1], [-1, 1]])
    ht[e] = h

    #assembly
    for ilocal in range(0,2):
        iglobal = IEN[e,ilocal]
        N[iglobal] += n
        for jlocal in range(0,2):
            jglobal = IEN[e,jlocal]
            K[iglobal,jglobal] += k[ilocal,jlocal]

#calculo da forca
Fx = -K@X
Fy = -K@Y
F = np.sqrt(Fx**2+Fy**2)

#calculo do sinal da curvatura:
sinal = np.zeros(npoints, dtype='double') #vetor de sinal em cada no
for i in range(npoints):
    sinal[i] = np.sign(np.dot(N[i],np.array([Fx[i],Fy[i]])))

Kf = sinal * F/ht #curvatura em cada no

##################################################
Parte Final
##################################################

#calculo do erro

#circunferencia
```

```
# Ka = 0.2 * np.ones(npoints)
# Ka= 2.5 * np.ones(npoints)

#quadrado
#Ka =  np.zeros(npoints)

#elipse
a = 1
b = 0.4
t = np.arctan2(a*Y,b*X)
Ka = (a*b)/((a**2 * np.sin(t)**2 + b**2 * np.cos(t)**2)**(3/2))

num = np.sum(((Kf-Ka)**2))
den = np.sum(np.abs(Kf)**2)
Error = np.sqrt(num/den)

#gravacao da solucao
point_data = {'curvatura': Kf}
msh.write_points_cells('arquivos.vtk/2D/elipse_MVF_1.vtk',mesh.points,
                       mesh.cells,#file_format='vtk-ascii"
                       point_data=point_data,
                       )

#print dos dados de interesse
print(max(Kf))
print(min(Kf))
print(Error)
print(time.time() - start_time)
```

Abordagem Lagrangiana

```
#importacao das bibliotecas
import numpy as np
```

```
import meshio as msh
from scipy.sparse import lil_matrix, csc_matrix
from scipy.sparse.linalg import inv, spsolve
from functions import *

#importacao das propriedades da malha
mesh = msh.read('arquivos.msh/2D/acoplado/arcs_ac.msh')
#mesh = msh.read('plano.msh')

#coordenadas dos nos
X = mesh.points[:,0]
Y = mesh.points[:,1]

#numero total de pontos
npoints = len(X)

IEN_bubble = mesh.cells['line']
#IEN da superficie da bolha

IEN_space = mesh.cells['triangle']
#IEN dos elementos 3D

nelements_bubble = IEN_bubble.shape[0]
#numero de elementos na superficie
points_bubble = np.unique(IEN_bubble)
#numero dos pontos na superficie
npoints_bubble = len(points_bubble)
#quantidade total de pontos na superficie



#definicao dos nos dentro e fora da bolha
Names = list(mesh.field_data.keys())
```

```
IENTypeElem = list(mesh.cell_data['triangle']['gmsh:physical']
-(mesh.field_data['out'][0] - Names.index('out')))
IENElem = [Names[elem] for elem in IENTypeElem]
index_out = [i for i, x in enumerate(IENElem) if x == 'out']

IEN_out = IEN_space[index_out]
#IEN somente dos elementos fora da bolha


index_in = [i for i, x in enumerate(IENElem) if x == 'in']
IEN_in = IEN_space[index_in]
#IEN somente dos elementos dentro da bolha

nelements_in = IEN_in.shape[0]
#numero de elementos dentro da bolha
points_in = np.unique(IEN_in)
#numero dos elementos dentro da bolha

nelements_out = IEN_out.shape[0]
#numero de elementos fora da bolha
points_out = np.unique(IEN_out)
#numero dos elementos fora da bolha


IEN = np.concatenate((IEN_out, IEN_in))
#IEN de todos os elementos

##########################################
#calculo da curvatura na bolha - MEF
#definicao das matrizes global
K = lil_matrix((npoints_bubble,npoints_bubble), dtype='double')
#forca da curvatura
```

```
ht = np.zeros(npoints_bubble, dtype='double')
#matriz de
N = np.zeros((npoints_bubble,2), dtype='double')

#loop nos elementos
for e in range(0,nelements_bubble):
    #indices dos nos
    v1 = IEN_bubble[e,0]
    v2 = IEN_bubble[e,1]

    #coordenadas dos nos
    p1 = np.array([X[v1],Y[v1]])
    p2 = np.array([X[v2],Y[v2]])

    # "area", comprimento do elemento
    h = np.linalg.norm(p2-p1)

    #calculo do vetor normal
    t = p2 - p1
    n = np.array([-t[1],t[0]])

    #matriz de rigidez local
    k = 1/h * np.array([[1, -1], [-1, 1]])
    ht[e] = h

    #assembling
    for ilocal in range(0,2):
        iglobal = np.where(points_bubble ==
        IEN_bubble[e,ilocal])[0][0]
        N[iglobal] += n
        for jlocal in range(0,2):
            jglobal = np.where(points_bubble ==
```

```
                IEN_bubble[e,jlocal])[0][0]
                K[iglobal,jglobal] += k[ilocal,jlocal]

K = K.tocsc()

Fx = -K@X[points_bubble]
Fy = -K@Y[points_bubble]

#calculo do sinal da curvatura:
sinal = np.zeros(npoints_bubble, dtype='double')
for i in range(npoints_bubble):
    sinal[i] = np.sign(np.dot(N[i],np.array([Fx[i],Fy[i]])))

F = np.sqrt(Fx**2+Fy**2)
Kf = -sinal * F/ht

###################################

#calculo da curvatura de todos os pontos
#algoritmo para definicao da curvatura nos pontos
#que nao sao da superficie da bolha
Kf_total = np.zeros(npoints, dtype='double')
for i in range(npoints):
    pi = np.array([X[i],Y[i]])
    j_min = 0
    mini = np.inf
    for j in points_bubble:
        pj = np.array([X[j],Y[j]])
        dist = np.linalg.norm(pi-pj)
        if dist<mini:
            j_min = j
            mini = dist
```

```
        Kf_total[i] = Kf[np.where(points_bubble == j_min)[0][0]]

#calculo da forca de tensao superficial
#MEF nos elementos 2D
Gx = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em x global
Gy = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em y global
M = lil_matrix((npoints,npoints), dtype='double')
#matriz de massa global

for e in range(nelements_in + nelements_out):
    v1 = IEN[e,0]
    v2 = IEN[e,1]
    v3 = IEN[e,2]

    A = 0.5 * np.abs(np.linalg.det([[1,X[v1],Y[v1]],
                                    [1,X[v2],Y[v2]],
                                    [1,X[v3],Y[v3]]]))

    bi = Y[v2]-Y[v3]
    bj = Y[v3]-Y[v1]
    bk = Y[v1]-Y[v2]

    ci = X[v2]-X[v3]
    cj = X[v3]-X[v1]
    ck = X[v1]-X[v2]


    gx = 1/6 * np.array([[bi,bj,bk],
                         [bi,bj,bk],
                         [bi,bj,bk]])
```

```
        gy = 1/6 * np.array([[ci,cj,ck],
                             [ci,cj,ck],
                             [ci,cj,ck]])



        m = A/12 * np.array([[2,1,1],
                             [1,2,1],
                             [1,1,2]])
        #assembling
        for ilocal in range(0,3):
            iglobal = IEN[e,ilocal]
            for jlocal in range(0,3):
                jglobal = IEN[e,jlocal]
                Gx[iglobal,jglobal] += gx[ilocal,jlocal]
                Gy[iglobal,jglobal] += gy[ilocal,jlocal]
                M[iglobal,jglobal] += m[ilocal,jlocal]


#definicao da funcao de Heaviside
H = np.zeros(npoints, dtype='double')
for i in range(npoints):
    if i in points_out:
        H[i] = 0
    if i in points_in:
        H[i] = 1
    if i in points_bubble:
        H[i] = 0.5

M = M.tocsc()
Gx = Gx.tocsc()
```

```
Gy = Gy.tocsc()

fx = spsolve(M,(Kf_total*(Gx@H)))
#componente de forca em x
fy = spsolve(M,(Kf_total*(Gy@H)))
#componente de forca em y

f = np.sqrt(fx**2+fy**2)
#magnitude da forca

#gravacao da solucao
point_data = {'forca': f, 'curvatura': Kf_total, 'Heaviside': H}

msh.write_points_cells('arquivos.vtk/2D/acoplado/forca_arcs_2D_lag.vtk',m
                       point_data=point_data,
                       )
```

Abordagem Euleriana

```
#importacao das bibliotecas
import numpy as np
import meshio as msh
from scipy.sparse import lil_matrix, csc_matrix
from scipy.sparse.linalg import inv, spsolve
from functions import *

#importacao das propriedades da malha
mesh = msh.read('arquivos.msh/2D/desacoplado/Desacoplado2D_r04.msh')

IEN_bubble = mesh.cells['line']
#IEN da superficie da bolha
IEN_space = mesh.cells['triangle']
#IEN dos elementos triangulares
```

```
nelements_bubble = IEN_bubble.shape[0]
#numero de elementos na superficie
points_bubble = np.unique(IEN_bubble)
#numero dos pontos na superficie
npoints_bubble = len(points_bubble)
#numero de pontos na superficie

#coordenadas dos nos da superficie
X_bubble = mesh.points[points_bubble][:,0]
Y_bubble = mesh.points[points_bubble][:,1]

nelements_space = IEN_space.shape[0]
#numero de elementos
#que nao sao da superficie
points_space = np.unique(IEN_space)
#numero dos pontos
#fora da superficie
npoints_space = len(points_space)
#numero de pontos
#fora da superficie

#coordenadas dos pontos fora da superficie
X_space = mesh.points[points_space][:,0]
Y_space = mesh.points[points_space][:,1]
npoints = len(X_space)

##########################################
#calculo da curvatura na bolha - MEF
#definicao das matrizes globais
K = lil_matrix((npoints_bubble,npoints_bubble), dtype='double')
#forca da curvatura
```

```
ht = np.zeros(npoints_bubble, dtype='double')
#vetor de areas
N = np.zeros((npoints_bubble,2), dtype='double')
#vetor normal

#loop nos elementos
for e in range(0,nelements_bubble):
    #indices dos nos
    v1 = np.where(points_bubble == IEN_bubble[e,0])[0][0]
    v2 = np.where(points_bubble == IEN_bubble[e,1])[0][0]

    #coordenadas dos nos
    p1 = np.array([X_bubble[v1],Y_bubble[v1]])
    p2 = np.array([X_bubble[v2],Y_bubble[v2]])

    # "area", comprimento do elemento
    h = np.linalg.norm(p2-p1)

    #calculo do vetor normal
    t = p2 - p1
    n = np.array([-t[1],t[0]])

    #matriz de rigidez local
    k = 1/h * np.array([[1, -1], [-1, 1]])
    ht[e] = h

    #assembling
    for ilocal in range(0,2):
        iglobal = np.where(points_bubble ==
        IEN_bubble[e,ilocal])[0][0]
        N[iglobal] += n
        for jlocal in range(0,2):
```

```
                jglobal = np.where(points_bubble ==
                IEN_bubble[e,jlocal])[0][0]
                K[iglobal,jglobal] += k[ilocal,jlocal]

K = K.tocsc()

Fx = -K@X_bubble
Fy = -K@Y_bubble

#calculo do sinal da curvatura:
sinal = np.zeros(npoints_bubble, dtype='double')
for i in range(npoints_bubble):
    sinal[i] = np.sign(np.dot(N[i],np.array([Fx[i],Fy[i]])))

F = np.sqrt(Fx**2+Fy**2)
Kf = sinal * F/ht

##########################################

#calculo da curvatura de todos os pontos
#algoritmo para definicao da curvatura nos pontos
#que nao sao da superficie da bolha

Kf_total = np.zeros(npoints, dtype='double')
#curvatura de todos os pontos
sinal = np.zeros(npoints, dtype = 'int')
#sinal da curvatura
distancia = np.zeros(npoints, dtype='double')
#distancia minima de cada ponto a superficie
for i in range(npoints):
    pi = np.array([X_space[i],Y_space[i]])
    j_min = 0
```

```
        mini = np.inf
        vector_min = np.array([0,0])
        for j in range(npoints_bubble):
            pj = np.array([X_bubble[j],Y_bubble[j]])
            dist_vector = pi-pj
            dist = np.linalg.norm(dist_vector)
            if dist<mini:
                j_min = j
                mini = dist
                vector_min = dist_vector

        Kf_total[i] = Kf[j_min]
        sinal[i] = np.sign(np.dot(N[j_min],vector_min))
        distancia[i] = sinal[i] * mini


#calculo da forca de tensao superficial
#MEF nos elementos 2D
Gx = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em x global
Gy = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em y global
M = lil_matrix((npoints,npoints), dtype='double')
#matriz de massa global

for e in range(nelements_space):
    v1 = np.where(points_space == IEN_space[e,0])[0][0]
    v2 = np.where(points_space == IEN_space[e,1])[0][0]
    v3 = np.where(points_space == IEN_space[e,2])[0][0]

    A = 0.5 * np.abs(np.linalg.det([[1,X_space[v1],Y_space[v1]],
                                    [1,X_space[v2],Y_space[v2]],
```

```
                                    [1,X_space[v3],Y_space[v3]]]))

bi = Y_space[v2]-Y_space[v3]
bj = Y_space[v3]-Y_space[v1]
bk = Y_space[v1]-Y_space[v2]

ci = X_space[v2]-X_space[v3]
cj = X_space[v3]-X_space[v1]
ck = X_space[v1]-X_space[v2]

gx = 1/6 * np.array([[bi,bj,bk],
                     [bi,bj,bk],
                     [bi,bj,bk]])

gy = 1/6 * np.array([[ci,cj,ck],
                     [ci,cj,ck],
                     [ci,cj,ck]])

m = A/12 * np.array([[2,1,1],
                     [1,2,1],
                     [1,1,2]])
#assembling
for ilocal in range(0,3):
    iglobal = np.where(points_space ==
    IEN_space[e,ilocal])[0][0]
    for jlocal in range(0,3):
        jglobal = np.where(points_space ==
        IEN_space[e,jlocal])[0][0]
        Gx[iglobal,jglobal] += gx[ilocal,jlocal]
        Gy[iglobal,jglobal] += gy[ilocal,jlocal]
        M[iglobal,jglobal] += m[ilocal,jlocal]
```

```
#definicao da funcao de Heaviside
H = np.zeros(npoints, dtype='double')
for i in range(npoints):
    e = 0.1
    if distancia[i] < -e:
        H[i] = 0
    elif distancia[i] > e:
        H[i] = 1
    else:
        d = distancia[i]
        H[i] = 0.5 * (1 + d/e + 1/np.pi * np.sin(np.pi*d/e))
        #H[i] = 0.5 + 1/32 * (45*d/e - 50*(d/e)**3 + 21*(d/e)**5)

M = M.tocsc()
Gx = Gx.tocsc()
Gy = Gy.tocsc()

fx = spsolve(M,(Kf_total*(Gx@H)))
#componente de forca em x
fy = spsolve(M,(Kf_total*(Gy@H)))
#componente de forca em y

f = np.sqrt(fx**2+fy**2)
#magnitude da forca

#gravacao da solucao

p=npoints_bubble+npoints_space
ft = np.zeros(p)
Kt = np.zeros(p)
Ht = np.zeros(p)
```

```
#loop para que nao haja problema
#com a ausencia de valores para a
#superficie da bolha
for i in range(npoints_bubble+npoints_space):
    if i in points_space:
        ft[i] = f[np.where(points_space == i)[0][0]]
        Kt[i] = Kf_total[np.where(points_space == i)[0][0]]
        Ht[i] = H[np.where(points_space == i)[0][0]]
    else:
        ft[i] = None
        Kt[i] = None
        Ht[i] = None

point_data = {'forca': ft, 'curvatura': Kt, 'Heaviside': Ht}
msh.write_points_cells('arquivos.vtk/2D/desacoplado/
forca_bolha1_2D_eul_2.vtk', mesh.points,
                       #file_format= "vtk-ascii",
                       mesh.cells,
                       point_data=point_data,
                       )
```

Modelo 3D

Cálculo da Curvatura

```
##################################################
Parte Inicial
##################################################
#importacao das bibliotecas
import numpy as np
import meshio as msh
from functions import *
import time
```

```
from scipy.sparse import lil_matrix, csc_matrix
from scipy.sparse.linalg import inv, spsolve

#inicializacao do tempo de execucao
start_time = time.time()

#importacao da malha
mesh = msh.read('arquivos.msh/3D/curvatura/esfera_r5_005.msh')
#para cada arquivo de malha a linha acima deve ser modificada de
#acordo com sua localizacao e nome

#coordenadas dos nos
X = mesh.points[:,0]
Y = mesh.points[:,1]
Z = mesh.points[:,2]

npoints = len(X) #numero de pontos

IEN = mesh.cells['triangle'] #criacao da IEN

nelements = IEN.shape[0] #numero de elementos

##############################################
Curvatura 3D - MEF
##############################################

#definicao das matrizes globais
K = lil_matrix((npoints,npoints), dtype='double') #rigidez
A_global = np.zeros(npoints, dtype='double') #vetor de areas
A_bari = np.zeros(npoints, dtype='double')
#vetor de areas baricentricas
N = np.zeros((npoints,3), dtype='double') #vetor normal
```

```
#loop nos elementos - MEF
for e in range(0,nelements):
    #numeros dos nos
    v1 = IEN[e,0]
    v2 = IEN[e,1]
    v3 = IEN[e,2]

    #coordenadas do vertices do elemento triangular
    pi = np.array([X[v1],Y[v1],Z[v1]])
    pj = np.array([X[v2],Y[v2],Z[v2]])
    pk = np.array([X[v3],Y[v3],Z[v3]])

    #referencial local no triangulo e projecao
    xi = (pj - pi) / np.linalg.norm(pj-pi)
    aux = np.cross(np.cross(pj-pi,pi-pk),pj-pi)
    eta = aux/np.linalg.norm(aux)

    #calculo das areas
    A = 0.5 * np.linalg.norm(np.cross(pj-pi,pk-pi))

    #contribuicao de area de cada no
    Ai = voronoi(pi,pj,pk) #funcao no arquivo "functions"
    Aj = voronoi(pj,pk,pi)
    Ak = voronoi(pk,pi,pj)
    for i,j in zip([v1,v2,v3],[Ai,Aj,Ak]):
        A_global[i] += j

    #coeficientes da matriz de rigidez
    bi = (np.dot(pk-pj,eta))
    bj = (np.dot(pi-pk,eta))
    bk = (np.dot(pj-pi,eta))
```

```
        ci = (np.dot(pk-pj,xi))
        cj = (np.dot(pi-pk,xi))
        ck = (np.dot(pj-pi,xi))

        #matriz de rigidez
        B = 1.0/(2.0*A) * np.array([[bi,bj,bk],[ci,cj,ck]])
        k = A * np.matmul(np.transpose(B),B)

        #calculo do vetor normal de cada no
        ni = np.cross((pj-pi)/
        np.linalg.norm(pj-pi),(pk-pi)/np.linalg.norm(pk-pi))
        ni = ni/np.linalg.norm(ni)
        nj = np.cross((pk-pj)/
        np.linalg.norm(pk-pj),(pi-pj)/np.linalg.norm(pi-pj))
        nj = nj/np.linalg.norm(nj)
        nk = np.cross((pi-pk)/
        np.linalg.norm(pi-pk),(pj-pk)/np.linalg.norm(pj-pk))
        nk = nk/np.linalg.norm(nk)
        for i,j in zip([v1,v2,v3],[ni,nj,nk]):
            N[i] += j

        #assembling
        for ilocal in range(0,3):
            iglobal = IEN[e,ilocal]
            A_bari[iglobal] += 1/3 * A
            for jlocal in range(0,3):
                jglobal = IEN[e,jlocal]
                K[iglobal,jglobal] += k[ilocal,jlocal]

K = K.tocsc() #mudanca do tipo de matriz esparsa
- melhor para operacoes
```

```
#calculo da forca
Fx = -K@X
Fy = -K@Y
Fz = -K@Z
F = np.sqrt(Fx**2+Fy**2+Fz**2)

#calculo do sinal da curvatura
sinal = np.zeros(npoints) #vetor com sinal em cada no
Fvector = np.zeros((npoints,3), dtype='double') #vetor forca
for i in range(npoints):
    Fvector[i] = [Fx[i],Fy[i],Fz[i]]
    N[i] = N[i]/np.linalg.norm(N[i])
    sinal[i] = np.sign(np.dot(N[i],Fvector[i]))

Kf = -sinal*F/A_bari #vetor de curvatura em cada no

##############################################
Curvatura 3D - MVF - Loop nos elementos
##############################################

#matrizes globais
KK = np.zeros((nelements,3,3)) #forca da curvatura
AA = np.zeros(IEN.shape) #area
NN = np.zeros((nelements,3,3)) #vetor normal

#vetores locais
Ki = np.zeros((npoints,3)) #forca da curvatura em cada no
Ai = np.zeros(npoints) #area relativa a cada no
Ni = np.zeros((npoints,3)) #vetor normal em cada no

#loop na IEN para formacao das matrizes
for n, i in enumerate(IEN):
```

```
    #vertices do triangulo elementar
    p1 = np.array([X[i[0]],Y[i[0]],Z[i[0]]])
    p2 = np.array([X[i[1]],Y[i[1]],Z[i[1]]])
    p3 = np.array([X[i[2]],Y[i[2]],Z[i[2]]])

    #as funcoes utilizadas abaixos
    #sao chamadas do arquivo "functions"
    AA[n] = [voronoi(p1,p2,p3),voronoi(p2,p3,p1),
    voronoi(p3,p1,p2)]
    KK[n] = [tn(p1,p2,p3),tn(p2,p3,p1),tn(p3,p1,p2)]
    NN[n] = [normal(p1,p2,p3),
    normal(p2,p3,p1),normal(p3,p1,p2)]

#soma para cada no/'assembling'
for index, x in np.ndenumerate(IEN):
    Ki[x]+=KK[index[0]][index[1]]
    Ai[x]+=AA[index[0]][index[1]]
    Ni[x]+=NN[index[0]][index[1]]



#definicao do sinal da curvatura
sinal = np.zeros(npoints, dtype='int') #vetor de sinal em cada n
for i in range(npoints):
    sinal[i] = np.sign(np.dot(Ni[i],Ki[i]))

#calculo final da curvatura em cada no
Kf = np.zeros(npoints) #vetor de curvatura nodal
for i in range(npoints):
    Kf[i] = -sinal[i]*np.linalg.norm(Ki[i])/Ai[i]

##############################################
Curvatura 3D - MVF - Loop nos nos
```

```
##########################################
#matrizes globais
Tgeral = np.zeros((npoints,3), dtype = 'float') #vetor tangente
Ageral = np.zeros(npoints, dtype = 'float') #area
Ngeral = np.zeros((npoints,3), dtype = 'float') #vetor normal

#loop nos nos
for i in range(npoints):

    elementos_in = np.where(IEN == i)
    #posicoes da IEN em que i aparece
    valores = IEN[elementos_in[0]]
    #linhas da IEN com i

    # criacao de listas com os
    # numeros dos elementos e dos nos vizinhos
    elementos = []
    vizinhos = []

    #inicializacao do primeiro elemento
    e = elementos_in[0][0]
    elementos += [e]
    j = IEN[e][np.where(IEN[e] == i)[0][0]-1]
    k = IEN[e][np.where(IEN[e] == i)[0][0]-2]
    vizinhos += [j]
    vizinhos += [k]

    #loop para criacao das listas de vizinhos
    while len(elementos) != len(elementos_in[0]):
        for linha in valores:
            if k in linha and j not in linha:
                e = np.where((IEN == linha).all(axis=1))[0][0]
```

```
            elementos += [e]
            j = k
            k = linha[(linha !=i) & (linha !=j)][0]
            vizinhos += [k]
        else:
            next

vizinhos = np.delete(vizinhos,-1)
#o loop sempre cria o ultimo
#elemento igual ao primeiro

p1 = np.array([X[i],Y[i],Z[i]]) #coordenadas do no

#valores locais
Alocal = 0
Tlocal = np.array([0,0,0])
Nlocal = np.array([0,0,0])

#loop pelos elementos
for n in range(len(elementos)):
    #coordenadas dos pontos vizinhos locais
    v2 = vizinhos[n]
    v3 = vizinhos[n-1]
    p2 = np.array([X[v2],Y[v2],Z[v2]])
    p3 = np.array([X[v3],Y[v3],Z[v3]])

    #A = 1/3 * 0.5 *
    np.linalg.norm(np.cross(p2-p1,p3-p1))
    #area total
    A = voronoi(p1,p2,p3)
    Alocal += A
```

```
            #calculo do vetor normal
            n = np.cross((p2-p1)/
            np.linalg.norm(p2-p1),(p3-p1)/np.linalg.norm(p3-p1))
            n = n/np.linalg.norm(n)

            #vetor tangente
            t = tn(p1,p2,p3)
            Tlocal = Tlocal + t
            Nlocal = Nlocal + n

        Tgeral[i] = Tlocal
        Ageral[i] = Alocal
        Ngeral[i] = Nlocal

#calculo do sinal da curvatura
sinal = np.zeros(npoints, dtype='int')
#vetor com sinal da curvatura em cada no
for i in range(npoints):
    Ngeral[i] = Ngeral[i]/np.linalg.norm(Ngeral[i])
    sinal[i] = np.sign(np.dot(Ngeral[i],Tgeral[i]))

#calculo da curvatura
Kf = np.zeros(npoints) #vetor de curvatura em cada no
for i in range(npoints):
    Kf[i] = - sinal[i] * np.linalg.norm(Tgeral[i])/Ageral[i]

################################################
Parte Final
################################################

#calculo do erro
#esfera
```

```
Ka = 0.4 * np.ones(npoints)
#Ka = 5 * np.ones(npoints)

#elipsoide
# a = 1
# b = 0.4
# c = 0.5
# Ka = np.abs(X**2 + Y**2 + Z**2
- a**2 - b**2 - c**2)/
(((a*b*c)**2)*(X**2/a**4 + Y**2/b**4 + Z**2/c**4)**(3/2))

#toroide
# a = 0.4
# c = 0.75
# A = (X**2 + Y**2 + Z**2 - a**2 - c**2)/(2*a*c)
# Ka = (c+2*a*A)/(a*(c+a*A))'

#cubo
#Ka = np.zeros(npoints)

num = np.sum(((Kf-Ka)**2))
den = np.sum(np.abs(Kf)**2)
Error = np.sqrt(num/den)

#gravacao da solucao
point_data = {'curvatura': Kf}
msh.write_points_cells('arquivos.vtk/3D/curvatura/
curvatura_esfera_005_MEF_b.vtk',mesh.points,
                        mesh.cells,#file_format='vtk-ascii"
                        point_data=point_data,
                        )
```

```
#print dos dados de interesse
print(max(Kf))
print(min(Kf))
print(Error)
print(time.time() - start_time)
```

Abordagem Lagrangiana

```
#importacao das bibliotecas
import numpy as np
import meshio as msh
from scipy.sparse import lil_matrix, csc_matrix
from scipy.sparse.linalg import inv, spsolve
import matplotlib.pyplot as plt
from functions import *

#importacao das propriedades da malha
mesh = msh.read('arquivos.msh/3D/acoplado/hyperboloid.msh')
#mesh = msh.read('plano.msh')

#coordenadas dos nos
X = mesh.points[:,0]
Y = mesh.points[:,1]
Z = mesh.points[:,2]
npoints = len(X)

#numero total de pontos
IEN_bubble = mesh.cells['triangle']
#IEN da superficie da bolha
IEN_space = mesh.cells['tetra']
#IEN dos elementos 3D

nelements_bubble = IEN_bubble.shape[0]
```

```
#numero de elementos na superficie
points_bubble = np.unique(IEN_bubble)
#numero dos pontos na superficie
npoints_bubble = len(points_bubble)
#quantidade total de pontos na superficie

#definicao dos nos dentro e fora da bolha
Names = list(mesh.field_data.keys())
IENTypeElem = list(mesh.cell_data['tetra']['gmsh:physical']-
(mesh.field_data['out'][0] - Names.index('out')))
IENElem = [Names[elem] for elem in IENTypeElem]
index_out = [i for i, x in enumerate(IENElem) if x == 'out']

IEN_out = IEN_space[index_out]
#IEN somente dos elementos fora da bolha

index_in = [i for i, x in enumerate(IENElem) if x == 'in']
IEN_in = IEN_space[index_in]
#IEN somente dos elementos fora da bolha

nelements_in = IEN_in.shape[0]
#numero de elementos dentro da bolha
points_in = np.unique(IEN_in)
#numero dos pontos dentro da bolha

nelements_out = IEN_out.shape[0]
#numero de elementos fora da bolha
points_out = np.unique(IEN_out)
#numero dos pontos fora da bolha

IEN = np.concatenate((IEN_out, IEN_in))
#IEN de todos os elementos
```

```
########################################
#calculo da curvatura na bolha - MEF
#definicao das matrizes global
K = lil_matrix((npoints_bubble,npoints_bubble), dtype='double')
#matriz de rigidez global
A_global = np.zeros(npoints_bubble, dtype='double')
#vetor de areas global
N = np.zeros((npoints_bubble,3), dtype='double')
#vetor normal global

#loop nos elementos
for e in range(0,nelements_bubble):
    #indices dos nos
    v1 = IEN_bubble[e,0]
    v2 = IEN_bubble[e,1]
    v3 = IEN_bubble[e,2]

    #coordenadas dos nos
    pi = np.array([X[v1],Y[v1],Z[v1]])
    pj = np.array([X[v2],Y[v2],Z[v2]])
    pk = np.array([X[v3],Y[v3],Z[v3]])

    #referencial local no triangulo
    xi = (pj - pi) / np.linalg.norm(pj-pi)
    aux = np.cross(np.cross(pj-pi,pi-pk),pj-pi)
    eta = aux/np.linalg.norm(aux)

    #calculo das areas
    A = 0.5 * np.linalg.norm(np.cross(pj-pi,pk-pi))

    Ai = voronoi(pi,pj,pk)
```

```
Aj = voronoi(pj,pk,pi)
Ak = voronoi(pk,pi,pj)
for i,j in zip([v1,v2,v3],[Ai,Aj,Ak]):
    A_global[np.where(points_bubble == i)[0][0]] += j

#coeficientes da matriz de rigidez
bi = (np.dot(pk-pj,eta))
bj = (np.dot(pi-pk,eta))
bk = (np.dot(pj-pi,eta))
ci = (np.dot(pk-pj,xi))
cj = (np.dot(pi-pk,xi))
ck = (np.dot(pj-pi,xi))

#matriz de rigidez
B = 1.0/(2.0*A) * np.array([[bi,bj,bk],[ci,cj,ck]])
k = A * np.matmul(np.transpose(B),B)

#calculo do vetor normal
ni = np.cross((pj-pi)/np.linalg.norm(pj-pi),(pk-pi)/
np.linalg.norm(pk-pi))
ni = ni/np.linalg.norm(ni)
nj = np.cross((pk-pj)/np.linalg.norm(pk-pj),(pi-pj)/
np.linalg.norm(pi-pj))
nj = nj/np.linalg.norm(nj)
nk = np.cross((pi-pk)/np.linalg.norm(pi-pk),(pj-pk)/
np.linalg.norm(pj-pk))
nk = nk/np.linalg.norm(nk)
for i,j in zip([v1,v2,v3],[ni,nj,nk]):
    N[np.where(points_bubble == i)[0][0]] += j

#assembling
for ilocal in range(0,3):
```

```
            iglobal = np.where(points_bubble ==
            IEN_bubble[e,ilocal])[0][0]

            for jlocal in range(0,3):
                jglobal = np.where(points_bubble ==
                IEN_bubble[e,jlocal])[0][0]
                K[iglobal,jglobal] += k[ilocal,jlocal]

K = K.tocsc()

Fx = -K@X[points_bubble]
Fy = -K@Y[points_bubble]
Fz = -K@Z[points_bubble]
F = np.sqrt(Fx**2+Fy**2+Fz**2)

#calculo do sinal da curvatura
sinal = np.zeros(npoints_bubble,dtype='int')
Fvector = np.zeros((npoints_bubble,3), dtype='double') #vetor forca



for i in range(npoints_bubble):
    Fvector[i] = [Fx[i],Fy[i],Fz[i]]
    N[i] = N[i]/np.linalg.norm(N[i])
    sinal[i] = np.sign(np.dot(N[i],Fvector[i]))

Kf = -sinal*F/A_global

#######################################

#calculo da curvatura de todos os pontos
#algoritmo para definicao da curvatura nos pontos
#que nao sao da superficie da bolha
```

```
Kf_total = np.zeros(npoints, dtype='double')
for i in range(npoints):
    pi = np.array([X[i],Y[i],Z[i]])
    j_min = 0
    mini = np.inf
    for j in points_bubble:
        pj = np.array([X[j],Y[j],Z[j]])
        dist = np.linalg.norm(pi-pj)
        if dist<mini:
            j_min = j
            mini = dist
    Kf_total[i] = Kf[np.where(points_bubble == j_min)[0][0]]


#calculo da forca de tensao superficial
#MEF nos elementos 2D
Gx = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em x global
Gy = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em y global
Gz = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em z global
M = lil_matrix((npoints,npoints), dtype='double')
#matriz de massa global

for e in range(nelements_in + nelements_out):
    v1 = IEN[e,0]
    v2 = IEN[e,1]
    v3 = IEN[e,2]
    v4 = IEN[e,3]

    [xi,yi,zi]=[X[v1],Y[v1],Z[v1]]
```

```
[xj,yj,zj]=[X[v2],Y[v2],Z[v2]]
[xk,yk,zk]=[X[v3],Y[v3],Z[v3]]
[xl,yl,zl]=[X[v4],Y[v4],Z[v4]]



V = 1/6 * (np.linalg.det([[1,xi,yi,zi],
                          [1,xj,yj,zj],
                          [1,xk,yk,zk],
                          [1,xl,yl,zl]]))

bi = (yj-yl)*(zk-zl)-(yk-yl)*(zj-zl)
bj = (yk-yl)*(zi-zl)-(yi-yl)*(zk-zl)
bk = (yi-yl)*(zj-zl)-(yj-yl)*(zi-zl)
bl = -1*(bi+bj+bk)

ci = (zj-zl)*(xk-xl)-(zk-zl)*(xj-xl)
cj = (zk-zl)*(xi-xl)-(zi-zl)*(xk-xl)
ck = (zi-zl)*(xj-xl)-(zj-zl)*(xi-xl)
cl = -1*(ci+cj+ck)

di = (xj-xl)*(yk-yl)-(xk-xl)*(yj-yl)
dj = (xk-xl)*(yi-yl)-(xi-xl)*(yk-yl)
dk = (xi-xl)*(yj-yl)-(xj-xl)*(yi-yl)
dl = -1*(di+dj+dk)

gx = 1/24 * np.array([[bi,bj,bk,bl],
                      [bi,bj,bk,bl],
                      [bi,bj,bk,bl],
                      [bi,bj,bk,bl]])

gy = 1/24 * np.array([[ci,cj,ck,cl],
```

```
                                [ci,cj,ck,cl],
                                [ci,cj,ck,cl],
                                [ci,cj,ck,cl]])

        gz = 1/24 * np.array([[di,dj,dk,dl],
                                [di,dj,dk,dl],
                                [di,dj,dk,dl],
                                [di,dj,dk,dl]])



        m = V/20 * np.array([[2,1,1,1],
                                [1,2,1,1],
                                [1,1,2,1],
                                [1,1,1,2]])
        #assembling
        for ilocal in range(0,4):
            iglobal = IEN[e,ilocal]
            for jlocal in range(0,4):
                jglobal = IEN[e,jlocal]
                Gx[iglobal,jglobal] += gx[ilocal,jlocal]
                Gy[iglobal,jglobal] += gy[ilocal,jlocal]
                Gz[iglobal,jglobal] += gz[ilocal,jlocal]
                M[iglobal,jglobal] += m[ilocal,jlocal]


#definicao da funcao de Heaviside
H = np.zeros(npoints, dtype='double')
for i in range(npoints):
    if i in points_out:
        H[i] = 0
    if i in points_in:
        H[i] = 1
```

```
    if i in points_bubble:
        H[i] = 0.5

M = M.tocsc()
Gx = Gx.tocsc()
Gy = Gy.tocsc()
Gz = Gz.tocsc()

fx = spsolve(M, (Kf_total*(Gx@H)))
fy = spsolve(M, (Kf_total*(Gy@H)))
fz = spsolve(M, (Kf_total*(Gz@H)))

f = np.sqrt(fx**2+fy**2+fz**2)

#gravacao da solucao
point_data = {'forca': f, 'curvatura': Kf_total, 'Heaviside': H}


msh.write_points_cells('arquivos.vtk/3D/acoplado
/acoplado_hiperboloide.vtk',mesh.points,
                       mesh.cells,#file_format='vtk-ascii"
                       point_data=point_data,
                       )
```

Abordagem Euleriana

```
#importacao das bibliotecas
import numpy as np
import meshio as msh
from scipy.sparse import lil_matrix, csc_matrix
from scipy.sparse.linalg import inv, spsolve
import matplotlib.pyplot as plt
from functions import *
```

```
#importacao das propriedades da malha
mesh = msh.read('arquivos.msh/3D/desacoplado/hyperboloid.msh')

IEN_bubble = mesh.cells['triangle']
#IEN da superficie da bolha
IEN_space = mesh.cells['tetra']
#IEN dos elementos tetraedricos

nelements_bubble = IEN_bubble.shape[0]
#numero de elementos na superficie
points_bubble = np.unique(IEN_bubble)
#numero dos pontos na superficie
npoints_bubble = len(points_bubble)
#numero de pontos na superficie

X_bubble = mesh.points[points_bubble][:,0]
Y_bubble = mesh.points[points_bubble][:,1]
Z_bubble = mesh.points[points_bubble][:,2]
#coordenadas dos nos da superficie

nelements_space = IEN_space.shape[0]
#numero de elementos
#que nao sao da superficie
points_space = np.unique(IEN_space)
#numero dos pontos
#fora da superficie
npoints_space = len(points_space)
#numero de pontos
#fora da superficie

X_space = mesh.points[points_space][:,0]
```

```
Y_space = mesh.points[points_space][:,1]
Z_space = mesh.points[points_space][:,2]
#coordenadas dos pontos fora da superficie

npoints = len(X_space)


##########################################
#calculo da curvatura na bolha - MEF
#definicao das matrizes globai
K = lil_matrix((npoints_bubble,npoints_bubble), dtype='double')
#matriz de rigidez global
A_global = np.zeros(npoints_bubble, dtype='double')
#vetor de areas global
N = np.zeros((npoints_bubble,3), dtype='double')
#vetor normal global

#loop nos elementos
for e in range(0,nelements_bubble):
    #indices dos nos
    v1 = np.where(points_bubble == IEN_bubble[e,0])[0][0]
    v2 = np.where(points_bubble == IEN_bubble[e,1])[0][0]
    v3 = np.where(points_bubble == IEN_bubble[e,2])[0][0]

    #coordenadas dos nos
    pi = np.array([X_bubble[v1],Y_bubble[v1],Z_bubble[v1]])
    pj = np.array([X_bubble[v2],Y_bubble[v2],Z_bubble[v2]])
    pk = np.array([X_bubble[v3],Y_bubble[v3],Z_bubble[v3]])

    #referencial local no triangulo
    xi = (pj - pi) / np.linalg.norm(pj-pi)
    aux = np.cross(np.cross(pj-pi,pi-pk),pj-pi)
```

```
eta = aux/np.linalg.norm(aux)

#calculo das areas
A = 0.5 * np.linalg.norm(np.cross(pj-pi,pk-pi))

Ai = voronoi(pi,pj,pk)
Aj = voronoi(pj,pk,pi)
Ak = voronoi(pk,pi,pj)
for i,j in zip([v1,v2,v3],[Ai,Aj,Ak]):
    A_global[i] += j

#coeficientes da matriz de rigidez
bi = (np.dot(pk-pj,eta))
bj = (np.dot(pi-pk,eta))
bk = (np.dot(pj-pi,eta))
ci = (np.dot(pk-pj,xi))
cj = (np.dot(pi-pk,xi))
ck = (np.dot(pj-pi,xi))

#matriz de rigidez
B = 1.0/(2.0*A) * np.array([[bi,bj,bk],[ci,cj,ck]])
k = A * np.matmul(np.transpose(B),B)

#calculo do vetor normal
ni = np.cross((pj-pi),(pk-pi))
ni = ni/np.linalg.norm(ni)
nj = np.cross((pk-pj),(pi-pj))
nj = nj/np.linalg.norm(nj)
nk = np.cross((pi-pk),(pj-pk))
nk = nk/np.linalg.norm(nk)
for i,j in zip([v1,v2,v3],[ni,nj,nk]):
    N[i] += j
```

```
    #assembling
    for ilocal in range(0,3):
        iglobal = np.where(points_bubble ==
        IEN_bubble[e,ilocal])[0][0]

        for jlocal in range(0,3):
            jglobal = np.where(points_bubble ==
            IEN_bubble[e,jlocal])[0][0]
            K[iglobal,jglobal] += k[ilocal,jlocal]

K = K.tocsc()

Fx = -K@X_bubble
Fy = -K@Y_bubble
Fz = -K@Z_bubble
F = np.sqrt(Fx**2+Fy**2+Fz**2)

#calculo do sinal da curvatura
sinal = np.zeros(npoints_bubble,dtype='int')
Fvector = np.zeros((npoints_bubble,3), dtype='double') #vetor forca


for i in range(npoints_bubble):
    Fvector[i] = [Fx[i],Fy[i],Fz[i]]
    N[i] = N[i]/np.linalg.norm(N[i])
    sinal[i] = np.sign(np.dot(N[i],Fvector[i]))

Kf = -sinal*F/A_global

##########################################
```

```
#calculo da curvatura de todos os pontos
#algoritmo para definicao da curvatura nos pontos
#que nao sao da superficie da bolha
Kf_total = np.zeros(npoints, dtype='double')
#curvatura de todos os pontos
sinal = np.zeros(npoints, dtype = 'int')
#sinal da curvatura
distancia = np.zeros(npoints, dtype='double')
#distancia minima de cada ponto a superficie
for i in range(npoints):
    pi = np.array([X_space[i],Y_space[i],Z_space[i]])
    j_min = 0
    vector_min = np.array([0,0,0])
    mini = np.inf
    for j in range(npoints_bubble):
        pj = np.array([X_bubble[j],Y_bubble[j],Z_bubble[j]])
        dist_vector = pi-pj
        dist = np.linalg.norm(dist_vector)
        if dist<mini:
            j_min = j
            mini = dist
            vector_min = dist_vector
        Kf_total[i] = Kf[j_min]
        sinal[i] = np.sign(np.dot(N[j_min],vector_min))
        distancia[i] = -sinal[i] * mini

#calculo da forca de tensao superficial
#MEF nos elementos 2D
Gx = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em x global
Gy = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em y global
```

```
Gz = lil_matrix((npoints,npoints), dtype='double')
#matriz gradiente em z global
M = lil_matrix((npoints,npoints), dtype='double')
#matriz de massa global

for e in range(nelements_space):
    v1 = np.where(points_space == IEN_space[e,0])[0][0]
    v2 = np.where(points_space == IEN_space[e,1])[0][0]
    v3 = np.where(points_space == IEN_space[e,2])[0][0]
    v4 = np.where(points_space == IEN_space[e,3])[0][0]

    [xi,yi,zi]=[X_space[v1],Y_space[v1],Z_space[v1]]
    [xj,yj,zj]=[X_space[v2],Y_space[v2],Z_space[v2]]
    [xk,yk,zk]=[X_space[v3],Y_space[v3],Z_space[v3]]
    [xl,yl,zl]=[X_space[v4],Y_space[v4],Z_space[v4]]


    V = 1/6 * (np.linalg.det([[1,xi,yi,zi],
                              [1,xj,yj,zj],
                              [1,xk,yk,zk],
                              [1,xl,yl,zl]]))

    bi = (yj-yl)*(zk-zl)-(yk-yl)*(zj-zl)
    bj = (yk-yl)*(zi-zl)-(yi-yl)*(zk-zl)
    bk = (yi-yl)*(zj-zl)-(yj-yl)*(zi-zl)
    bl = -1*(bi+bj+bk)

    ci = (zj-zl)*(xk-xl)-(zk-zl)*(xj-xl)
    cj = (zk-zl)*(xi-xl)-(zi-zl)*(xk-xl)
    ck = (zi-zl)*(xj-xl)-(zj-zl)*(xi-xl)
    cl = -1*(ci+cj+ck)
```

```
di = (xj-xl)*(yk-yl)-(xk-xl)*(yj-yl)
dj = (xk-xl)*(yi-yl)-(xi-xl)*(yk-yl)
dk = (xi-xl)*(yj-yl)-(xj-xl)*(yi-yl)
dl = -1*(di+dj+dk)

gx = 1/24 * np.array([[bi,bj,bk,bl],
                      [bi,bj,bk,bl],
                      [bi,bj,bk,bl],
                      [bi,bj,bk,bl]])

gy = 1/24 * np.array([[ci,cj,ck,cl],
                      [ci,cj,ck,cl],
                      [ci,cj,ck,cl],
                      [ci,cj,ck,cl]])

gz = 1/24 * np.array([[di,dj,dk,dl],
                      [di,dj,dk,dl],
                      [di,dj,dk,dl],
                      [di,dj,dk,dl]])


m = V/20 * np.array([[2,1,1,1],
                     [1,2,1,1],
                     [1,1,2,1],
                     [1,1,1,2]])
#assembling
for ilocal in range(0,4):
    iglobal = np.where(points_space
    == IEN_space[e,ilocal])[0][0]
    for jlocal in range(0,4):
        jglobal = np.where(points_space ==
        IEN_space[e,jlocal])[0][0]
```

```
                Gx[iglobal,jglobal] += gx[ilocal,jlocal]
                Gy[iglobal,jglobal] += gy[ilocal,jlocal]
                Gz[iglobal,jglobal] += gz[ilocal,jlocal]
                M[iglobal,jglobal] += m[ilocal,jlocal]

#definicao da funcao de Heaviside
H = np.zeros(npoints, dtype='double')
for i in range(npoints):
    e = 0.25
    if distancia[i] < -e:
        H[i] = 0
    elif distancia[i] > e:
        H[i] = 1
    else:
        d = distancia[i]
        H[i] = 0.5 * (1 + d/e + 1/np.pi * np.sin(np.pi*d/e))

M = M.tocsc()
Gx = Gx.tocsc()
Gy = Gy.tocsc()
Gz = Gz.tocsc()

fx = spsolve(M, (Kf_total*(Gx@H)))
#componente de forca em x
fy = spsolve(M, (Kf_total*(Gy@H)))
#componente de forca em y
fz = spsolve(M, (Kf_total*(Gz@H)))
#componente de forca em z

f = np.sqrt(fx**2+fy**2+fz**2)
#magnitude da forca
```

```
#gravacao da solucao
p=npoints_bubble+npoints_space
ft = np.zeros(p, dtype='double')
Kt = np.zeros(p, dtype='double')
Ht = np.zeros(p, dtype='double')

#loop para que nao haja problema
#com a ausencia de valores para a
#superficie da bolha
for i in range(npoints_bubble+npoints_space):
    if i in points_space:
        ft[i] = f[np.where(points_space == i)[0][0]]
        Kt[i] = Kf_total[np.where(points_space == i)[0][0]]
        Ht[i] = H[np.where(points_space == i)[0][0]]
    else:
        ft[i] = None
        Kt[i] = None
        Ht[i] = None

point_data = {'forca': ft, 'curvatura': Kt, 'Heaviside': Ht}

msh.write_points_cells('arquivos.vtk/3D/
desacoplado/hiperboloide.vtk',mesh.points,
                       mesh.cells,#file_format='vtk-ascii"
                       point_data=point_data,
                       )
```

Funções

```
import numpy as np

# Codigo com funcoes utilizadas nos codigos globais
```

```
#funcao que define se o triangulo eh obtusangulo ou nao
#dadas as coordenadas de seus pontos
def ang(p1,p2,p3):

    l1 = p1-p2
    l2 = p1-p3
    l3 = p3-p2

    if np.dot(l3,l1) < 0 or np.dot(l2,-l3) < 0 or
    np.dot(-l1,-l2) < 0 :
        return 'Obtuso'


#funcao que calcula o vetor normal tn dados os pontos do triangulo
#vetor em relacao a p1
def tn(p1,p2,p3):
    #pontos medios
    m1 = (p1+p2)/2
    m3 = (p1+p3)/2

    #vetor ortogonal a base media
    v = np.cross(p3-p2,np.cross(p3-p2,p1-p2))
    tn = v/np.linalg.norm(v)

    #base media
    d = np.linalg.norm(m1-m3)

    t = tn * d
    return t


#funcao que calcula "Amixed" e "AVoronoi"
```

```
def voronoi(p1,p2,p3):
    #pontos medios
    m1 = (p1+p2)/2
    m3 = (p1+p3)/2

    #vetores tangentes
    t1 = (p2-p1)/np.linalg.norm(p2-p1)
    t2 = (p3-p1)/np.linalg.norm(p3-p1)

    # area 'circuncentrica'
    a = np.linalg.norm(p1-p2)
    b = np.linalg.norm(p1-p3)
    c = np.linalg.norm(p2-p3)
    alpha = a**2 * (b**2+c**2-a**2)
    betta = b**2 * (a**2+c**2-b**2)
    gamma= c**2 * (a**2+b**2-c**2)
    circ = (alpha*p3 + betta*p2 + gamma*p1)/(alpha+betta+gamma)
    a1 = 0.5 * np.linalg.norm(np.cross(m1-p1,m3-p1))
    a2 = 0.5 * np.linalg.norm(np.cross(circ-m1,circ-m3))



    #correcao para triangulos obtusangulos
    Atotal = 0.5 * np.linalg.norm(np.cross(p2-p1,p3-p1))
    if  ang(p1,p2,p3) == 'Obtuso':
        if np.dot(t2,t1) < 0:
            A = 0.5 * Atotal
        else:
            A = 0.25 * Atotal
    else:
        A = a1+a2

    #A = a1 + a2
```

```
    return A


#funcao que calcula o vetor normal dados os pontos do triangulo
#normal em relacao a p1
def normal(p1,p2,p3):
    n = np.cross((p2-p1)/np.linalg.norm(p2-p1),
    (p3-p1)/np.linalg.norm(p3-p1))
    n = n/np.linalg.norm(n)

    return n
```