KENNEDY LEANDRO DE SOUZA NEVES

Desenvolvimento de um Gêmeo Digital para previsão de falhas estruturais em casco de FPSO

São Paulo
2023

KENNEDY LEANDRO DE SOUZA NEVES

Desenvolvimento de um Gêmeo Digital para previsão de falhas estruturais em casco de FPSO

Versão Corrigida

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Ciências.

São Paulo
2023

KENNEDY LEANDRO DE SOUZA NEVES

Desenvolvimento de um Gêmeo Digital para previsão de falhas estruturais em casco de FPSO

Versão Corrigida

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo como parte dos requisitos para obtenção do título de Mestre em Ciências.

Área de concentração: Engenharia de Estruturas

Orientador:
Prof. Dr. Luís Antônio Guimarães Bitencourt Jr.
Coorientador:
Prof. Dr. Guilherme Rosa Franzini

São Paulo
2023

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 24 de julho de 2023

Assinatura do autor: Kennedy Leandro de Souza Neves

Assinatura do orientador: Marcos Pitanguy Jr.

Catalogação-na-publicação

Neves, Kennedy Leandro de Souza
Desenvolvimento de um Gêmeo Digital para previsão de falhas estruturais em casco de FPSO / K. L. S. Neves -- versão corr. -- São Paulo, 2023.
388 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Estruturas e Geotécnica.

1.Gêmeo digital 2.FPSO 3.Cascos 4.Elementos finitos 5.Extensão de vida I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Estruturas e Geotécnica II.t.

Aos meus amados pais.

Agradecimentos

O financiamento desta pesquisa pela Enauta Participações S.A. (contrato n.º TC-0219-02113-001) e a parceira entre a Escola Politécnica da Universidade de São Paulo e Technomar Engenharia Ltda. é gratamente reconhecido.

Gostaria de expressar meus sinceros agradecimentos ao meu orientador Professor Luís Bitencourt Jr. pelo conhecimento compartilhado, disponibilidade, compreensão, paciência e colaboração à pesquisa durante a realização deste trabalho.

Também sou grato ao meu coorientador Professor Guilherme Rosa Franzini, especialmente pelas contribuições e sugestões da perspectiva da Engenharia Naval.

Ao Professor Alfredo Gay Neto, pelas recomendações na utilização das ferramentas de análise estrutural e comentários que foram de grande importância no enriquecimento deste trabalho, meu reconhecimento e gratidão.

Aos colaboradores do Programa de Pós-Graduação em Engenharia Civil da Escola Politécnica da Universidade de São Paulo (USP), que sempre me trataram com muito respeito, eficiência e celeridade.

Aos professores que contribuíram com minha formação educacional ao longo da minha vida. Em especial, aos professores da Pontifícia Universidade Católica de Goiás, Sérgio Botassi dos Santos, Rodrigo Carvalho da Mata, Renata Machado Soares, Alberto Vilela Chaer e Robson Lopes Pereira e também ao professor da University of Missouri, Carlos Sun.

A Flávio Vasconcelos de Souza e Leandro Sales de Castro pela confiança a mim depositada e oportunidade de crescimento profissional.

Agradecimento especial ao meu amigo Flávio Mamede Pereira Gomes que me apresentou e despertou meu interesse na engenharia. Por meio de muitas aulas informais, incentivou meu entusiasmo e aprendizado ao longo dos anos. Sou extremamente grato ao meu "padrinho de engenharia".

Ao meu amigo Estevão Alencar Bandeira pela amizade, parceria, conversas e conselhos animadores que me fizeram não pensar em desistir.

Agradeço também ao meu irmão Kellington Fabrício de Souza Neves pelo estímulo desde muito cedo a ingressar-me no mestrado.

À minha amada esposa e amiga, Ivana Pereira Barbosa por entender e me ajudar a superar todos os momentos difíceis. Esse trabalho possui grande contribuição de seu amor, carinho, paciência, incentivo e compreensão no decorrer desta longa caminhada.

Por fim, aos meus pais, Jazon Alves das Neves e Maria Euza de Souza Neves, pela dedicação e esforço incondicional em sempre colocar a minha educação em primeiro lugar. Serei eternamente grato a vocês.

"Experiência é a moeda mais valiosa, viver não cabe em currículo algum."

O autor

Resumo

O propósito deste trabalho é desenvolver um Gêmeo Digital para previsão de falhas estruturais em casco de unidade flutuante de produção, armazenamento e transferência (FPSO - *Floating, Production, Storage and Offloading*) baseado em uma metodologia adaptativa. Esta metodologia, realiza múltiplas análises numéricas atualizando um modelo global de casco de FPSO em elementos finitos com base em sistemas acoplados (dados de sensores, análises numéricas, inspeções). O Gêmeo Digital adapta os dados de entrada para representar várias fases do ciclo de vida do ativo. Por meio de um sistema de apoio à decisão (DSS - *Decision Support System*), a avaliação de risco da estrutura é realizada com a finalidade da extensão da vida do FPSO na exploração no campo de petróleo.

Algoritmos foram elaborados para atualizar os modelos, submeter as análises numéricas para execução e verificar os resultados gerados. Para simplificar a aplicação destes algoritmos e manipulação dos dados, foi criada uma Guia de Interface do Usuário.

O primeiro estudo de caso utilizou dados de atualização dos níveis dos tanques e dados de onda irregular por uma série temporal para aplicação de carga hidrodinâmica no modelo em elementos finitos do casco do FPSO, reproduzindo um cenário de fase de operação. Foram identificados alguns membros estruturais atingindo níveis de alerta e sendo reportados pelo Gêmeo Digital.

O segundo estudo de caso utilizou dados de atualização para níveis dos tanques, dados de onda regular e modelos de evolução da corrosão para chapas e vigas. Foram analisadas quatro hipóteses por um período de 30 ano com níveis de alerta sendo reportados pelo Gêmeo Digital. Neste caso, um dos membros estruturais foi selecionado para uma avaliação de risco mais complexa, considerando a variabilidade do material. Os resultados da probabilidade de falha foram consistentes com a evolução da corrosão prevista por cada uma das hipóteses.

Palavras-chave: Gêmeo digital; FPSO; cascos; elementos finitos; extensão de vida.

Abstract

This work aims to develop a Digital Twin for predicting structural failures in FPSO (Floating Production, Storage and Offloading) hulls based on a proposed adaptive methodology. This methodology performs multiple numerical analyzes by updating a global finite element FPSO hull model based on coupled systems (sensor data, numerical analysis, inspections, etc.). The Digital Twin allows to adjust the input data in order to better adapt to represent different phases of the asset's life cycle. Employing a Decision Support System, the structural risk assessment is carried out with the purpose of extending the life of the FPSO in the oil field exploration.

Algorithms were designed to update the models, submit the numerical analyzes for execution and verify the generated results. A Guide User Interface was created to simplify the application of these algorithms and data manipulation.

The first case study used tank level update data and irregular wave data by a time series to apply hydrodynamic load to the finite element model of the FPSO hull, reproducing an operation phase scenario. Some structural members were identified as reaching alert levels and being reported by the Digital Twin.

The second case study used tank level update data, regular wave data and corrosion evolution models for plates and beams. Four hypotheses were analyzed over a period of 30 years with alert levels being reported by the Digital Twin. In this case, one of the structural members was selected for a more complex risk assessment, considering the variability of the material. The failure probability results were consistent with the corrosion evolution predicted by each hypotheses.

Keywords: Digital twin; FPSO; hull; finite elements, life extension.

Lista de Figuras

Lista de Tabelas

Notação

Símbolos

Letras gregas

α	Ângulo formado pela relação entre d_∞ e τ_t
ε	Vetor de incerteza do modelo
η	Elevação de onda
θ	Parâmetros de modelo
λ	Comprimento de onda
μ	Vetor de incerteza de medição
μ_R	Valor médio de R
μ_S	Valor médio de S
μ_Z	Valor médio de Z
ν	Coeficiente de Poisson
ρ	Densidade
τ_c	Tempo da vida útil do sistema de proteção anticorrosivo
σ_R^2	Variância de R
σ_S^2	Variância de S
σ_Z	Variância de Z

σ_y	Tensão limite de escoamento
σ_u	Tensão limite última
σ	Desvio padrão
σ_R	Desvio padrão de R
τ_t	Tempo de transição da evolução da corrosão
Φ	Função de distribuição cumulativa
ω	Frequência angular de onda

Letras romanas minúsculas

c	Velocidade de fase da onda
cg	Velocidade de grupo
d	Perda de espessura por corrosão
d_0	Espessura inicial
$\dot{d}$	Taxa de corrosão
d_∞	Perda de espessura por corrosão a longo prazo
$d_{was}\%$	Desgaste permitido por corrosão em %
f	Frequência de onda
f_R	Função de densidade de probabilidade de R
f_S	Função de densidade de probabilidade de S
k	Número de onda
t	Tempo
t_d	Tempo de vida útil de projeto
z	Resultados de saída

Letras romanas maiúsculas

A	Amplitude de onda
E	Módulo de elasticidade
F	Variável estocástica
F_R	Função de probabilidade acumulada de R
H	Altura de onda
H_s	Altura de onda significativa
M	Operador de modelo
$\tilde{M}$	Conjuntos de dados experimentais
$\mathcal{M}_m$	Classe de modelos
M_z	Momento de flexão em torno do eixo z
P	Probabilidade
P_o	Espaço de resposta de saída do modelo
P_m	Subconjunto de classe de modelos
R	Componente de resistência
S	Componente de tensão
T	Período de onda
T_i	Média dos períodos de onda T (entre zero-ascendentes)
T_z	Período de pico
Z	Variável aleatória

Abreviaturas e siglas

ABS	*American Bureau of Shipping*
ANP	Agência Nacional do Petróleo, Gás Natural e Biocombustíveis
CAD	*Computer-Aided Design*
CAE	*Computer-Aided Engineering*
CFD	*Computational Fluid Dynamics*
DM	*Digital Model*
DS	*Digital Shadow*
DSS	Sistema de apoio à decisão (*Decision Support System*)
DT	*Digital Twin*
DTA	*Digital Twin Aggregate*
DTI	*Digital Twin Instance*
DTP	*Digital Twin Prototype*
E&P	Exploração e Produção de petróleo e gás natural
FEAM	Método Alternativo de Elementos Finitos (*Finite Element Alternating Method*)
FEMU	*Finite Element Model Updating*
FPSO	*Floating, Production, Storage and Offloading*
FPU	Unidade flutuante de produção (*Floating Production Unit*)
GPS	*Global Positioning System*
GUI	*Guide User Interface*
HPC	*High Performance Computing*
HVAC	*Heating, Ventilation, and Air Conditioning*

IoT	*Internet of Things*
LOA	*Length overall*
MEF	Método dos Elementos Finitos
MATLAB®	*MATrix LABoratory*
MARPOL	*International Convention for the Prevention of Pollution from Ships*
NASA	*National Aeronautics and Space Administration*
NDCV	*Nominal Design Corrosion Values*
NRC	*National Research Council*
OPA	*Oil Pollution Act*
PDF	Função de densidade de probabilidade (*Probability Density Function)*
PLM	*Product Lifecycle Management*
PPC	Produção, Planejamento e Controle
SIF	Fator de intensidade da tensão (*Stress Intensity Factor*)
SRA	Análise de confiabilidade estrutural (*Structural Reliability Analysis*)
SSI	*Stress-Strength Interference*
TLP	*Tension-Leg Platforms*
USDm	*United States Dollar Million*
WAMIT®	*Wave Analysis MIT*

Sumário

1. Introdução

1.1. Aspectos gerais

O processo de exploração, extração e produção de petróleo e gás natural em todo o mundo cresce conforme a demanda por consumo de energia. Ao longo do tempo, junto ao aumento dessa necessidade, a indústria *offshore* de extração de petróleo e gás se expandiram, sendo crucial que desafios fossem superados para a exploração em águas marítimas cada vez mais profundas. A unidade flutuante de produção, armazenamento e transferência (FPSO - *Floating, Production, Storage and Offloading*) é resultado do esforço para atender tais exigências em regiões do oceano sem malhas de oleodutos para transporte do óleo produzido para a costa (Klare, 2009; Eide, 2008).

Qualquer ação de reparo nas instalações da plataforma de extração não apresenta possibilidade de deslocamento até uma doca e muitas vezes requer técnicas onerosas para isso. Assim, a estrutura, equipamentos e sistemas são projetados para atingir uma vida útil compatível com o tempo de concessão do campo de petróleo, o que representa cerca de 20 a 30 anos de operação. Dentro desse contexto, um FPSO pode ser projetado ou convertido a partir de cascos de antigas embarcações. A arquitetura naval consolida o arranjo geral da embarcação especialmente com consideração ao casco, o qual, corresponde à cerca de 20% a 30% do custo total de um FPSO (Gordo e Leal, 2018; Shetelig, 2013). O formato do casco é a parte mais notável dos navios de grande porte, podendo influenciar significativamente na disposição dos compartimentos, sistemas e equipamentos. Dentro do aspecto econômico, a otimização do casco reflete diretamente na redução de custos, capacidade de carga, resistência estrutural, equilíbrio, comportamento no mar, emissão de dióxido de carbono, segurança e quantidade limite de produção (Ang *et al.*, 2017).

Embora a construção de novos FPSOs exija grande planejamento, esforço e experiência em meio a um mercado extremamente volátil com oscilações econômicas muitas

vezes imprevisíveis, a E&P (Exploração e Produção de petróleo e gás natural), principalmente em águas profundas[1], se demonstra um mercado em expansão. Com a pandemia de COVID-19 e colapso do mercado de petróleo, por exemplo, o mercado de exploração *offshore* ainda tenta retomar o crescimento. A expectativa de boa recuperação do setor até o ano de 2025 em um cenário global (como apresentado na Figura 1.1) não tem se confirmado, apresentando um decréscimo de cerca de 24% (Rystad, 2022) na comparação entre o ano de 2022 com 2021. No entanto, observando o cenário nacional, o Brasil aparece como maior contribuinte no mundo em termos de blocos concedidos em 2022 (Rystad, 2022), demonstrando uma melhor perspectiva e se confirmando maior destino da maior parte dos investimentos para o setor de exploração *offshore* (Figura 1.2).

Segundo Boggs *et al.* (2020), a frota mundial de FPSOs é formada por 213 unidades em mais de 30 países, sendo 164 em operação, 26 disponíveis e 23 em construção. Deste total, o Brasil é responsável por 48 unidades em operação e 8 em desenvolvimento, se caracterizando como principal atuante no mercado global de FPSOs.

[1]No setor de E&P, a medida de profundidade no mar, também denominada lâmina d´água, é definida como a distância vertical entre a superfície e o solo do mar, e pode ser dividida em três níveis: águas rasas - até 300 metros; águas profundas - entre 300 a 1.500 metros; e águas ultraproprofundas - igual ou superior a 1.500 metros.

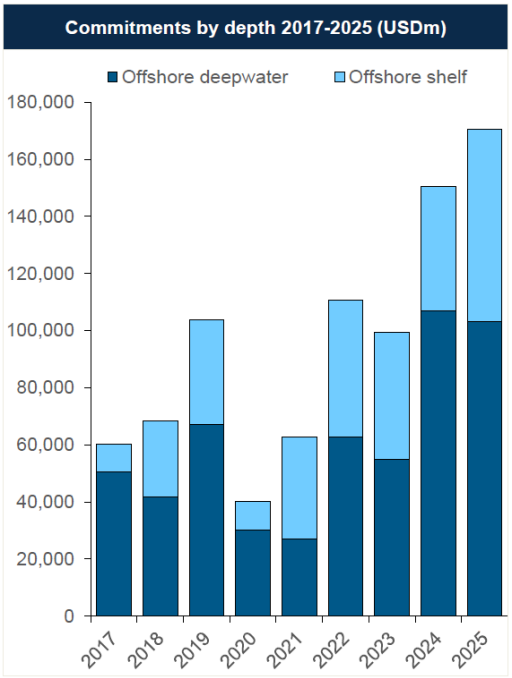


Figura 1.1.: Previsão de investimentos para exploração *offshore* em águas rasas e profundas entre 2017 e 2025 (Rystad, 2020).

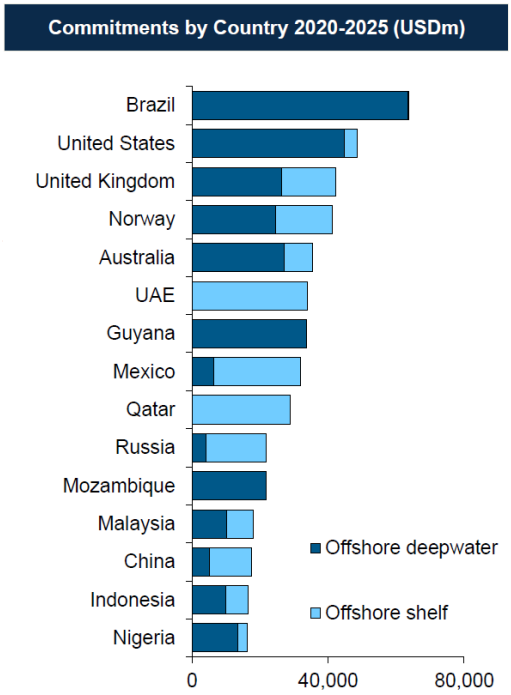


Figura 1.2.: Previsão de investimentos para exploração *offshore* em águas rasas e profundas por país entre 2020 e 2025 (Rystad, 2020).

Dentro do cenário nacional, 61% das plataformas em operação no Brasil são do tipo FPSO (ANP, 2021b), as quais correspondem a 73,4% da capacidade de processamento de petróleo e 98,6% da capacidade de processamento de gás natural do país (Tabela 1.1 e Tabela 1.2).

Tabela 1.1.: Capacidade de processamento de petróleo no Brasil das plataformas em operação por tipo de unidade (ANP, 2021b).

Tipo de unidade	Capacidade de processamento petróleo (bbl/dia [2])
FPSO	73,4%
Semi submersível	19,4%
Fixa	4,6%
FPU[3]	2,6%

Quando encerrado o tempo de concessão do campo de petróleo, deve ocorrer o descomissionamento das instalações de E&P ou a extensão de vida (ANP, 2019). A decisão entre essas duas opções é pautada por diferentes análises com objetivo de não

[2]Barris por dia.
[3]Unidade flutuante de produção (*Floating Production Unit*)

Tabela 1.2.: Capacidade de processamento de gás natural no Brasil das plataformas em operação por tipo de unidade (ANP, 2021b).

Tipo de unidade	Capacidade de processamento gás (mil m^3/dia)
FPSO	98,6%
Semi submersível	0,7%
Fixa	0,6%
FPU	0,1%

se realizar o descomissionamento precoce, e caso haja a viabilidade de adiamento do fim das operações, a concessão do campo de petróleo pode ser ampliada pela Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (ANP), após uma série de averiguações. Dentre essas, a prática de extensão de vida do campo de petróleo requer implementação de recomendações advindas da avaliação de riscos do estudo de extensão de vida útil, por vezes, é exigida a substituição de plataformas de operação, como aconteceu na renovação dos contratos de exploração dos campos Marlim e Voador, os quais deverão descomissionar as plataformas para a instalação de dois FPSOs novos (Petrobras, 2019). O bom estado de conservação da estrutura do casco do FPSO, é, na maioria das vezes, indispensável para exequibilidade da extensão de vida útil, visto que é necessário equilibrar investimentos e ganhos com a postergação do descomissionamento.

Nas últimas duas décadas, os Gêmeos Digitais (réplicas digitais de dispositivos físicos) vêm sendo estudados, desenvolvidos e aplicados com os mais variados objetivos, em embarcações de diferentes setores e plataformas de extração de petróleo e gás (Bolton *et al.*, 2018; Danielsen-Haces, 2018; Grange, 2018; Bole, Powell e Rousseau, 2017; Renzi *et al.*, 2017). A concepção mais geral de um Gêmeo Digital consiste em três partes distintas: o produto físico, o produto digital (ou virtual) e as conexões entre esses dois sistemas, que são dados de atualização, enviados e avaliados, primeiramente vindo do produto físico para o produto digital, sendo possível analisar e disponibilizar resultados por meio de um sistema de apoio à decisão (DSS - *Decision Support System*) para possibilitar ações de prevenção ou manutenção mais precisas nas operações do produto físico. Além disso, a organização e o registro das condições de operação ao longo da vida útil das estruturas *offshore* têm um papel crucial nas decisões estratégicas. Os dados armazenados oferecem informações essenciais para o planejamento de inspeções, a determinação da extensão da vida útil e a avaliação de riscos após incidentes, entre outros elementos relevantes. Essas informações de-

talhadas e atualizadas possibilitam uma abordagem mais precisa e fundamentada na gestão e manutenção das instalações.

1.2. Escopo da dissertação

Este trabalho é voltado para o desenvolvimento de um Gêmeo Digital que utiliza modelos numéricos via Método dos Elementos Finitos (MEF) para análises estruturais lineares e estáticas (sem consideração de efeitos térmicos e cargas ocasionais).

A metodologia idealizada é aplicada a modelos 3D globais em elementos finitos representativos com três tanques de carga de FPSOs. No entanto, cabe destacar que a metologia proposta é adaptativa (permite ajustar os dados de entrada fornecidos de forma a melhor se adequar para representar as diversas etapas do ciclo de vida do ativo) e que o processo adotado pode ser alterado para melhor atender às necessidades do problema em questão. É válido ressaltar, que nesse trabalho é empregado o *software* comercial Ansys Mechanical® para as análises numéricas via MEF.

O Gêmeo Digital é capaz de receber informações de sistemas acoplados (análises numéricas externas, dados de sensores, condições meteoceanográficas, dados de inspeção ou variações de parâmetros), alterar variáveis do modelo e analisar resultados. Os carregamentos podem ser oriundos do programa de elementos finitos utilizado e também provenientes de outras análises numéricas.

O DSS identifica regiões ou membros estruturais em níveis de alerta, os quais são baseados em valores predefinidos que podem ser comparados aos resultados de tensão, deformação ou deslocamento. Assim, evidenciando locais e elementos estruturais que extrapolem limites pré-definidos, assim auxiliando em diferentes cenários de atuação do FPSO (fase de projeto ou operação), e consequentemente, auxiliando na extensão da vida útil do FPSO. Após a primeira fase de identificação dos níveis de alerta, pode ser feita uma análise mais complexa para avaliação de risco das regiões ou membros estruturais.

Devido ao fato de ser uma tecnologia recente, sem muitas referências de trabalhos na literatura voltados para aplicações de FPSO, é conveniente e imprescindível que sejam compreendidas algumas considerações relacionadas ao desafios, cenários e necessidades do Gêmeo Digital desenvolvido.

1.3. Motivação e justificativa

Os primeiros estudos científicos sobre os Gêmeos Digitais surgiram recentemente. Em consequência da maior intensificação de casos de aplicação e uso na última década, houve também um crescente interesse da indústria de E&P pela tecnologia com o intuito de reduzir custos, aumentar a segurança e eficiência. Esse interesse não é por acaso, há uma grande carência por Gêmeos Digitais na indústria de petróleo e gás, e tal deficiência se tornou bastante evidente com o trabalho de Handscomb, Sharabura e Woxholth (2016), o qual apresentou o contexto de dificuldades técnicas enfrentadas nesse campo ao longo dos anos, destacando que esse setor é o único que perdeu eficiência nos últimos 100 anos quando comparado os demais setores industriais.

Existem várias vantagens na utilização de um FPSO na indústria *offshore*: a possibilidade de exploração em locais de águas profundas, a dispensa por infraestrutura de escoamento da produção, construção versátil diante da alternativa de conversão de casco por meio de uma embarcação existente, dentre outras. Essas facilidades fizeram com que grandes empresas de exploração investissem mais na construção desse tipo de ativo com o passar dos anos, diminuindo o uso de outros tipos de plataformas.

Em análise do cenário global, o Brasil se apresenta em posição de maior destaque, com a maior parte das operações atuais no mundo, correspondendo a 30% dos FPSOs da frota mundial em operação e 35% dos FPSOs em construção (Boggs *et al.*, 2020). Isto se deve à maior quantidade de investimentos em exploração de águas profundas devido ao pré-sal e maior abertura comercial do país, atraindo parceiros internacionais para o mercado interno brasileiro nos últimos anos. Além disso, as reservas provadas[4] e o consumo mundial de petróleo e gás natural continuam crescendo a cada ano (Figura 1.3 e Figura 1.4 com base nos dados da ANP (2022)).

[4] As reservas provadas são definidas como a quantidade de petróleo ou gás natural que é considerada comercialmente recuperável com uma certa margem de segurança, com base em análises de geociências e engenharia. Quando métodos probabilísticos são utilizados, espera-se que a probabilidade de recuperação da quantidade estimada seja de pelo menos 90%.

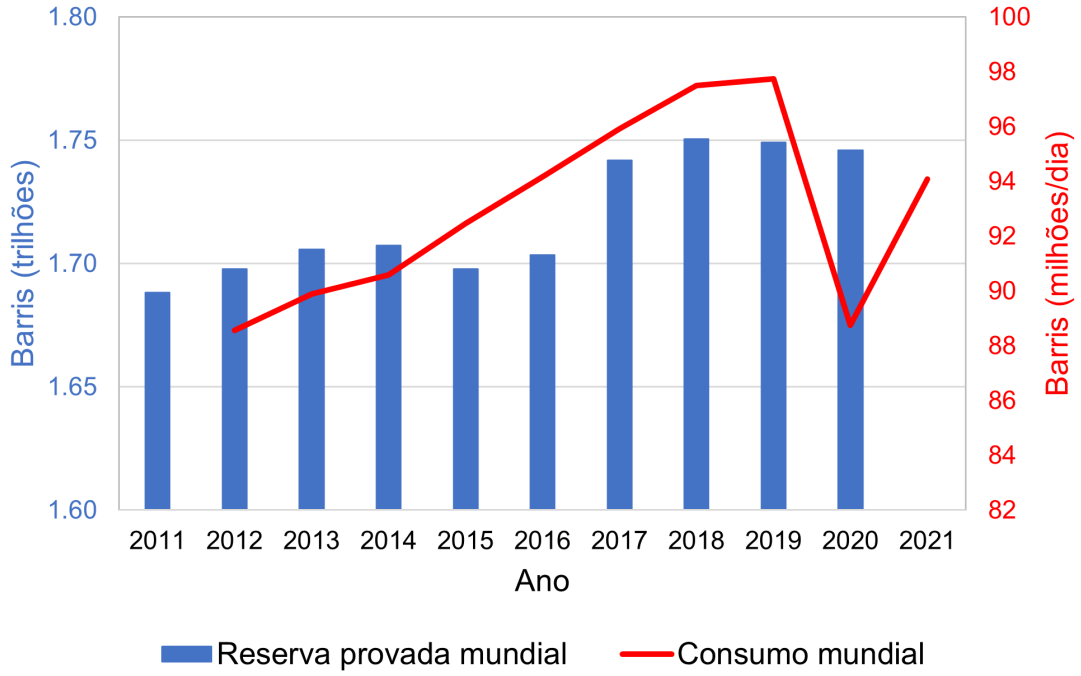


Figura 1.3.: Reservas provadas e consumo mundial de petróleo (ANP, 2022).

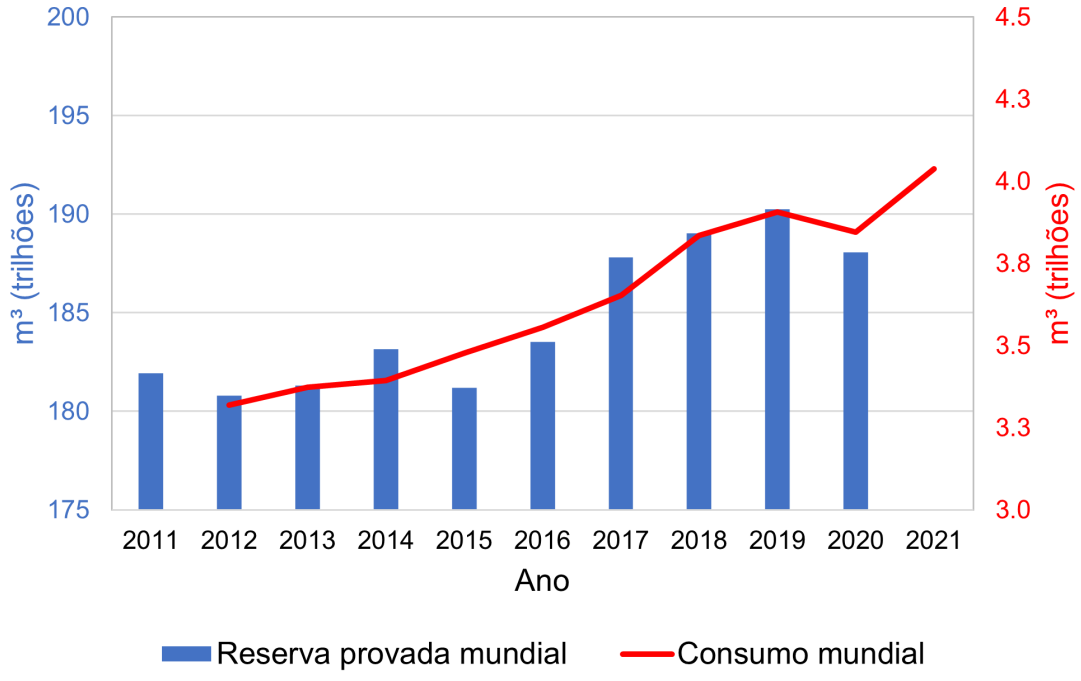


Figura 1.4.: Reservas provadas e consumo mundial de gás natural (ANP, 2022).

Outro importante fator que tem chamado a atenção mundial nas últimas décadas

são os desastres ambientais causados pelos acidentes ocorridos em instalações de petróleo e gás, com consequentes prejuízos ao meio ambiente que perduram por anos na natureza e multas impostas pelos órgãos competentes. No ano de 2019 por exemplo, um misterioso vazamento de petróleo, de autoria desconhecida, causou impacto em grande parte do litoral brasileiro, acarretando prejuízos para diversos setores da economia (Zacharias, Gama e Fornaro, 2021). Obviamente, a estrutura do casco é um agente fundamental perante os derramamentos de óleo originários de falhas estruturais. São diversas as causas que podem levar à rachaduras no casco: corrosão, fadiga, concentrações de tensões, falta de manutenção, dentre outras. As quais podem ser muitas vezes imprevisíveis e ainda afetadas por outros elementos: distribuição do peso da carga, forças estáticas e dinâmicas, vibração de maquinário, dentre outros (DeCola, 2009).

A extensão da vida útil do FPSO, sobretudo do casco, oferece uma vasta gama de benefícios. Esta estratégia é a mais apropriada quando os reservatórios do campo ainda dispõe de boa quantidade de petróleo e gás a ser explorado de maneira economicamente satisfatória. Segundo Ferreira *et al.* (2020), órgãos reguladores, mesmo em países mais avançados do setor de petróleo e gás ou que já passaram pelo processo de extensão, desenvolveram apenas material superficial sobre extensão de vida útil, sem definir uma metodologia estruturada para avaliação da possibilidade prolongação das atividades. No Brasil, segundo ANP (2021a), cerca de 55% das plataformas em operação têm mais de 20 anos e chegarão ao final do seu ciclo de vida em 2025. É previsto um investimento de mais de R$ 28 bilhões para descomissionamento de instalações de E&P até o ano de 2025 (Figura 1.5). Solicitações para a ampliação da fase de produção têm sido uma prática crescente dos operadores junto à ANP, por isso, a avaliação de extensão de vida útil se estabelece como um tema cada vez mais significativo nos últimos anos (ANP, 2020; ANP, 2016). Um Gêmeo Digital pode auxiliar no desafio de extensão de vida útil no setor, assegurando melhor integridade estrutural do casco. Além disso, o gêmeo digital possibilita a estruturação dos dados operacionais (calados, ações ambientais, corrosão, incidentes, etc.) necessários para a avaliação robusta da extensão de vida da unidade.

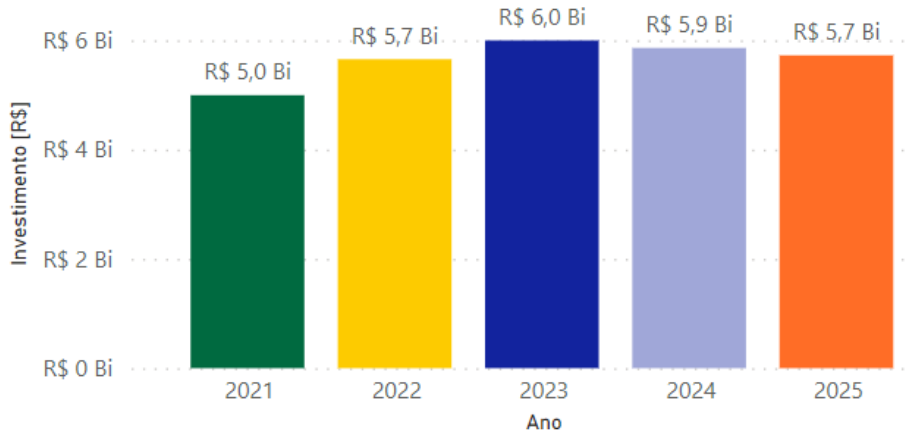


Figura 1.5.: Investimentos previstos por ano para o descomissionamento de instalações de E&P entre 2021 e 2025 (ANP, 2021c).

Gradativamente, pesquisas e experiências têm se tornado auxiliares do estudo de falhas estruturais e cada vez mais vinculados e dependentes. É válido ressaltar que, o uso de novas tecnologias associadas às principais ferramentas de simulação computacional da engenharia, já são essenciais para os modelos matemáticos, e regularmente são ajustados com a finalidade de buscar uma resposta aproximada que melhor corresponda com a realidade, amparando e contribuindo nas atividades diárias do setor industrial. O conhecimento e a utilização da análise numérica têm papel imprescindível na produção e tomada de decisão com o objetivo de reduzir custos, tempo e riscos. Além disso, muitas vezes, alguns procedimentos experimentais são inviáveis devido à complexidade e, ou escassez de tempo, deixando os projetos estruturais a cargo da experiência e discernimento do projetista.

Com apenas cerca de duas décadas de desenvolvimento, alguns trabalhos sobre Gêmeos Digitais foram publicados nos últimos anos com a finalidade de implementação da tecnologia em unidades de E&P (Danielsen-Haces, 2018; Grange, 2018; Bole, Powell e Rousseau, 2017). Porém, devido ao surgimento recente e também às dificuldades encontradas em se implementar a tecnologia ao longo dos últimos anos, ainda são raros de se encontrar trabalhos incluindo Gêmeos Digitais bem detalhados, especialmente que abordem a função estrutural do casco de FPSOs. É evidente que as sugestões das experiências até o presente agregam progressivamente o conhecimento aos projetos que virão a ser executados no futuro. Nota-se que há uma série de deficiências e imperfeições que podem ser melhoradas no desempenho de cascos de FPSO. Assim, identifica-se a ausência, oportunidade da exploração do tema e contribuição científica a ser feita com o trabalho proposto.

1.4. Objetivos

O propósito geral desse trabalho é desenvolver um Gêmeo Digital, baseado no MEF, que possa auxiliar na previsão de falhas estruturais em um casco de FPSO por meio de uma metologia adaptativa concebida por algoritmos criados em MATLAB®. Para atender a isso, os seguintes objetivos específicos foram estipulados:

- Criar uma metodologia reestruturando as etapas do MEF tradicional (pré-processamento, análise e pós-processamento) para que o Gêmeo Digital considere às necessidades de um FPSO em diferentes fases de operação;
- Receber dados, manipular informações, alterar variáveis e analisar resultados manipulando carregamentos de um modelo em elementos finitos alta hierarquia (realista) de um FPSO;
- Integrar cargas provenientes de análises hidrodinâmicas previamente executadas por outros programas com o modelo de elementos finitos;
- Criar um DSS para auxiliar na tomada de decisão e evidenciar regiões e elementos estruturais em condições críticas com base em um sistema com diferentes níveis de alerta;
- Ser capaz de utilizar um ou mais conjuntos de dados de sistemas acoplados (inspeções de avaliação, sensores, condições climáticas e análises numéricas externas) para análise estrutural e prover recomendações instantâneas no DSS periodicamente;
- Ser capaz de utilizar múltiplos conjuntos de dados para análises estruturais e prover recomendações no DSS.

1.5. Estrutura da dissertação

Esta dissertação está organizada em cinco capítulos e quatro apêndices. O Capítulo 1 se refere à introdução que contextualiza sobre o uso de FPSOs, fundamenta a importância da pesquisa e discute sobre aspectos gerais do objeto de estudo.

O Capítulo 2 apresenta uma revisão bibliográfica sobre os principais tópicos relacionados a Gêmeos Digitais e FPSOs. Uma vez que se trata de um tema ainda recente e pouco difundido na literatura, a abordagem histórica e conceitual se faz necessária para a melhor compreensão da metodologia criada. São apresentadas pesquisas anteriores que contribuíram para o estado da arte atual no que concerne a Gêmeos Digitais e seus mais diversos assuntos correlacionados (origem, tecnologias chave, tipos, categorias, classificação, importância, aplicações e eficácia). Também são abordados nessa seção os trabalhos que colaboraram com os temas intrínsecos, tais como as fases, cenários de atuação, desafios enfrentados e sugestões relacionadas.

O Capítulo 3 descreve a metodologia adaptativa proposta para o Gêmeo Digital baseado nas etapas do MEF tradicional para que as necessidades de um FPSO em diversas fases sejam atendidas. São detalhadas as subetapas desenvolvidas e o fluxo de trabalho criado com os algoritmos.

No Capítulo 4 são apresentados dois estudos de caso, demonstrando a utilização do Gêmeo Digital e com a finalidade de apresentar a capacidade e principais características da metodologia empregada.

O Capítulo 5 expõe as considerações finais da dissertação, evidenciando as conclusões e sugestões para trabalhos futuros.

O Apêndice A apresenta as diretrizes seguidas para criação dos modelos globais de elementos finitos utilizados neste trabalho.

O Apêndice B apresenta o manual de utilização com a guia de interface do usuário (GUI - *Guide User Interface*), variáveis, arquivos e diretórios utilizados pelos algoritmos desenvolvidos.

O Apêndice C apresenta todo o código dos algoritmos escritos em MATLAB®.

O Apêndice D apresenta os modelos de elementos finitos utilizados para os estudos de caso.

2. Estado da arte

2.1. Contexto histórico

Desde a primeira Revolução Industrial, no século XVIII, com os motores a vapor e mecanização das fábricas, ocorrem periodicamente, avanços que mudam radicalmente o modo como produtos ou serviços são concebidos em todo o mundo. Assim, a descoberta da eletricidade e produção em massa marcaram a Indústria 2.0, e a tecnologia da informação nos processos de fabricação, por meio dos computadores auxiliando a automação, estabeleceu a Indústria 3.0.

Esses avanços, herdados das revoluções passadas, e o desenvolvimento da tecnologia nas últimas décadas, possibilitaram que grandes inovações surgissem em diversas áreas. Esses novos conceitos, nos mais variados campos de estudo, promoveram um progresso significativo ao serem aplicados em conjunto na resolução dos problemas contemporâneos. Atualmente, a quarta Revolução Industrial ocorre especialmente por meio da associação de tecnologias da informação e comunicação empregadas na indústria de maneira a proporcionar o gerenciamento de processos, recursos, procedimentos e produção utilizando estratégias inteligentes. Nove tecnologias são consideradas por Kadir (2017) como pilares da Indústria 4.0: robô autônomo, Gêmeo Digital, computação em nuvem, impressão 3D, realidade aumentada, *Big Data*, IoT (*Internet of Things*), segurança cibernética e integração de sistemas.

Dentre tais tecnologias, algumas se destacam como cruciais para a Indústria 4.0, se sobressaindo como peças chave para o aprimoramento de procedimentos de diversos setores. Entre elas: a tecnologia *Big Data* e IoT contribuíram ao longo dos últimos tempos para que melhores decisões fossem tomadas nas quatro divisões fundamentais da indústria: produção, fabricação, logística e serviço.

Com seu uso de crescimento exponencial na última década, a tecnologia *Big Data*, pode ser definida como uma grande quantidade de dados não estruturados ou estruturados de uma variedade de fontes coletadas (Lu *et al.*, 2019). A captação e

armazenamento de dados monitorados no decorrer da execução das atividades ao longo do tempo possibilitam a análise de tendências e mudanças operacionais. Essas informações podem, muitas vezes, auxiliar na previsão de problemas, evitando-se ou precavendo-se diante de eventos indesejados na funcionalidade do sistema.

Atualmente muito utilizada, a IoT basicamente se estrutura em três componentes: dispositivos de sensoriamento instalados em um ambiente ou objeto, um meio de transferência de dados e uma central de controle. A infraestrutura de comunicação de dados, análises, aplicativos e pessoas é utilizada de maneira constante para se obter um maior controle em tempo real, oferecendo maior segurança, eficiência e economia na indústria (Khan *et al.*, 2017).

2.2. Conceito de Gêmeo Digital

As tecnologias descritas na Seção 2.1, aliadas ao uso de outras ferramentas amplamente difundidas para solucionar problemas de engenharia, como CAD (*Computer-Aided Design*), CAE (*Computer-Aided Engineering*), MEF, dentre outras, possibilitaram a criação de uma cópia virtual, de objetos ou processos do mundo real, criando-se um modelo digital. Nessa reprodução virtual, a estrutura de qualquer ativo físico ou não físico tem seu comportamento simulado devolvendo a resposta e otimizando o desempenho do ativo em tempo real. O chamado Gêmeo Digital engloba as características mais notórias da Indústria 4.0, interconectando componentes inteligentes, monitorando, analisando dados e aprimorando ativos para alcançar o mais alto nível de eficiência.

É bastante aceito e reconhecido na literatura que esta ideia de cópia virtual originou-se dos trabalhos de Michael Grieves (Grieves, 2019; Tao *et al.*, 2018; Mayani, Svendsen e Oedegaard, 2018; Grieves e Vickers, 2017), surgindo publicamente em 2002 na *University of Michigan* como uma ideia para Gerenciamento de Ciclo de Vida do Produto (PLM - *Product Lifecycle Management*), recebendo diferentes expressões ao longo do tempo como "*Mirrored Spaces Model*" em Grieves (2005b) e subsequentemente "*Information Mirroring Model*" em Grieves (2005a), até que foi então denominado como "*Digital Twin*" por John Vickers em Piascik *et al.* (2010), termo predominante adotado atualmente na literatura. Como apresentado na Figura 2.1, é possível observar que todos os elementos da idealização de Gêmeo Digital já existiam na ideia inicial de Michael Grieves, onde um espaço real alimenta um espaço

virtual (o qual tem subespaços virtuais), este por sua vez cria um ciclo, transmitindo a informação de volta ao espaço real após análises terem sido executadas com os dados recebidos.

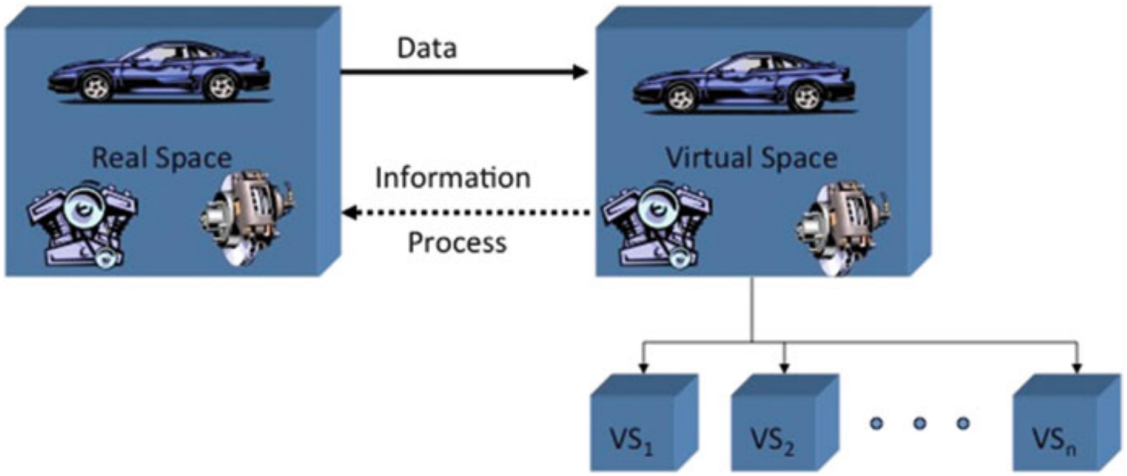


Figura 2.1.: Conceito ideal para um PLM (Grieves e Vickers, 2017).

Embora a origem do termo e ideia seja um consenso quase unânime na literatura, de maneira oposta, o conceito de Gêmeo Digital, no meio acadêmico e industrial, muitas vezes se confunde ao de tecnologias necessárias para o desenvolvimento do próprio Gêmeo Digital (IoT, *Big Data* e *Cloud Computing*, por exemplo). Assim, o seu conceito pode ser encontrado com diferentes perspectivas, muitas vezes sendo influenciado pela área de estudo, campo ou cenário de aplicação da tecnologia. Tuegel *et al.* (2011) utilizaram o conceito como proposta de reengenharia para previsão da vida estrutural de uma aeronave passando a explorar os avanços da computação de alto desempenho (HPC - *High Performance Computing*). Em seu procedimento, com um modelo de alta fidelidade para prever o comportamento estrutural de aeronaves, o autor utiliza um Gêmeo Digital para integrar o cálculo de deflexões estruturais e temperaturas em resposta às condições de voo, com danos locais resultantes da evolução do estado do material. A descrição apresentada por Glaessgen e Stargel (2012) é mais detalhada e amplamente utilizada na comunidade científica. Os autores enfatizam o pensamento de que existem diferenças no conceito adotado dependendo do contexto. Segundo os autores, um Gêmeo Digital é aceito comumente como uma simulação ultrarrealista, multifísica, multiescala e probabilística integrada de sistemas ou produtos que podem refletir a vida de seu gêmeo correspondente usando modelos físicos disponíveis, dados históricos e, ou dados em tempo real. Em Lee, Bagheri e Kao (2015), um Gêmeo Digital é caracterizado como um modelo acoplado

da máquina real que opera na plataforma de nuvem e simula a condição de saúde estrutural com um conhecimento integrado dos algoritmos analíticos orientados a dados e de outros conhecimentos físicos disponíveis. Rosen *et al.* (2015) elaboraram o conceito de Gêmeo Digital como um modelo que pode interagir entre os comportamentos do sistema autônomo e o ambiente no mundo físico. Ainda segundo os autores, existem muitas outras definições em campos específicos, mas eles preferem conceituar um Gêmeo Digital de forma geral como um mapeamento real de todos os componentes no ciclo de vida de um produto, usando dados físicos, dados virtuais e dados de interação entre eles. Grieves e Vickers (2017) definiram como um conjunto de construções de informações virtuais que descrevem completamente um produto físico manufaturado potencial ou real, desde o nível micro atômico até o nível macro geométrico. Na melhor das hipóteses, qualquer informação que possa ser obtida através da inspeção de um produto manufaturado físico pode ser obtido do seu Gêmeo Digital.

Apesar das variadas definições, alguns autores são mais breves, objetivos e generalizados no conceito. Bacchiega e Bondani (2018) descreve uma réplica digital em tempo real de um dispositivo físico. Söderberg *et al.* (2017) estabelece apenas como uma cópia digital do sistema físico para executar a otimização em tempo real. Tao *et al.* (2018) determina como um mapeamento real de todos os componentes no ciclo de vida do produto usando dados físicos, dados virtuais e dados de interação entre eles. E por fim, Bolton *et al.* (2018) resume como uma representação virtual dinâmica de um objeto ou sistema físico em todo o seu ciclo de vida, usando dados em tempo real para permitir entendimento, aprendizado e raciocínio. Em síntese, é importante destacar que os conceitos apresentados na maioria dos trabalhos, se baseia tecnicamente em três componentes (Lu *et al.*, 2020):

- Um modelo de informação que abstrai as especificações de um objeto físico;
- Um mecanismo de comunicação que transfere dados entre um Gêmeo Digital e sua contraparte física e;
- Um módulo de processamento de dados que pode extrair informações dos dados recebidos de várias fontes para construir a representação em tempo real de um objeto físico.

Além do funcionamento em conjunto desses três componentes, a sincronização e processamento dos dados em alta performance também é uma necessidade, pois os resultados ou resposta devem ser obtidos instantaneamente.

2.3. Classificação dos Gêmeos Digitais

Assim como o conceito, não há um consenso em relação à classificação dos tipos de Gêmeos Digitais, as formas de classificação dos tipos sofrem interferência da área de estudo ou setor industrial onde é aplicada a tecnologia. A maioria dos autores se fundamentam em itens como a organização, complexidade ou forma de associação para categorizar os sistemas de Gêmeos Digitais. Portanto, devido às mais diversas soluções existentes para caracterizarem Gêmeos Digitais na literatura, surgiram alguns trabalhos com o objetivo exclusivo de de estabelecer uma classificação.

Segundo Grieves (2016), há dois tipos ou estágios de Gêmeos Digitais: protótipo (DTP - *Digital Twin Prototype*) e instância (DTI - *Digital Twin Instance*). O DTP descreve o protótipo do ativo contendo o conjunto de informações necessárias para produzir uma versão física que copia a virtual. Esse conjunto inclui modelo 3D completo, com requisitos, listas de materiais, processos e serviços. O DTI descreve um produto físico correspondente específico ao qual um único Gêmeo Digital permanece vinculado ao longo da vida útil desse produto físico. Dependendo dos casos de uso necessários, esse tipo de Gêmeo Digital pode conter, mas não se limita aos seguintes conjuntos de informações: um modelo 3D completo com dimensionamento e tolerâncias gerais para representar a geometria da instância física e seus componentes, listas de materiais dos componentes diretamente e indiretamente relacionados, uma lista de processos das operações executadas na instância física, juntamente com os resultados de quaisquer medições e testes, registro de serviços relatando tarefas já executadas e componentes substituídos, os estados operacionais monitorados a partir de dados reais do sensor com dados atuais e anteriores e estudo de previsão dos dados futuros. O agrupamento de DTIs formam um agregado de Gêmeos Digitais (DTA – *Digital Twin Aggregate*), o qual se diferencia do DTI por ser um sistema dependente no que tange à estrutura de dados. Os Gêmeos Digitais são operados em um ambiente de Gêmeos Digitais (DTE - *Digital Twin Environment*), um espaço integrado de aplicação física de vários domínios para operar com uma variedade de propósitos. Tais propósitos também se dividem em duas principais categorias: preditivo ou interrogativo. O propósito preditivo tem o objetivo de prever o comportamento futuro e o desempenho do produto físico, sendo aplicado no DTP com o intuito de avaliar o comportamento do produto projetado com componentes que variam entre os limites de tolerâncias definidos, com a finalidade de verificar se o item atendeu aos requisitos propostos. No DTI o propósito preditivo trabalha na previsão de uma instância específica de um produto físico específico que incorpora compo-

nentes reais e histórico de componentes. Várias instâncias do produto podem ser agregadas para fornecer uma variedade de possíveis estados futuros. Já o propósito interrogativo se aplica em DTIs de um determinado DTA, onde instâncias gêmeas digitais correlacionadas podem ter seus históricos analisados quanto a um mesmo item, possibilitando a estimativa ou previsão de um estado futuro. Por exemplo, falhas subsequentes de um componente e as leituras precedentes de sensores ligados a tal componente devem gerar um alerta no futuro sempre que o mesmo padrão de leitura dos sensores se repetirem.

O trabalho de Kritzinger *et al.* (2018) tem como um de seus objetivos, apresentar uma metodologia de classificação para os sistemas baseada em alguns tópicos. Primeiramente é analisado o nível de integração de dados entre os objetos. O fluxo de dados pode ser manual ou automático, essa característica é utilizada para diferir inicialmente um Gêmeo Digital (DT - *Digital Twin*), um Modelo Digital (DM - *Digital Model*) e uma Sombra Digital (DS - *Digital Shadow*). Basicamente, nos três tipos de conjuntos, há no mínimo um objeto físico e um digital em cada, porém um DM não usa qualquer forma de integração automática de dados, enquanto em um DT todo o fluxo de dados é automático, ou seja, qualquer mudança no objeto físico implica em uma mudança de estado no objeto digital e vice-versa, já no DS o fluxo de dados automático ocorre somente do objeto físico para o objeto digital. A Figura 2.2 exemplifica de maneira simples a categorização dos sistemas quanto ao fluxo de dados. O autor destaca que por muitas vezes os trabalhos recentes utilizam o termo *Digital Twin*, quando na verdade o conjunto em questão é apenas um *Digital Model* ou um *Digital Shadow*.

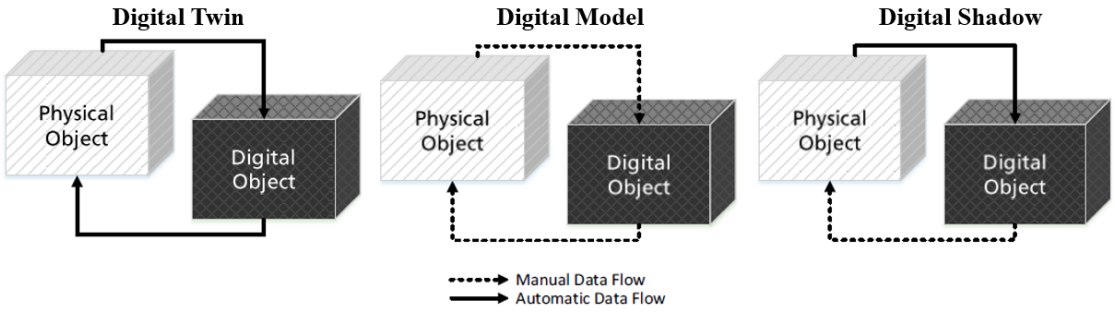


Figura 2.2.: Fluxo de dados nos diferentes sistemas (Kritzinger *et al.*, 2018).

Outro tópico de classificação citado por Kritzinger *et al.* (2018) é o foco de aplicação do Gêmeo Digital, onde o autor sugere as seguintes categorias: ciclo de vida do produto, fabricação em geral, Produção, Planejamento e Controle (PPC), processos

e manutenção. As formas e recursos como a tecnologia é empregada também podem originar em novas categorias de classificação, como por exemplo: o método de simulação dos sistemas (simulação contínua ou simulação de evento discreto), protocolos de comunicação e tecnologias utilizadas.

A função de um Gêmeo Digital é extremamente ampla, podendo servir como recurso de mapeamento dinâmico, nível de controle e otimização de um único equipamento ou peça, setores individuais ou mesmo sistemas complexos englobando numerosas variáveis. Lu *et al.* (2020), enfatizam que Gêmeos Digitais podem ser criados com aplicabilidades de aspectos completamente distintos, assim são apresentadas quatro finalidades para as quais a tecnologia pode ser designada: ativos, pessoas, fábricas ou indústrias e processos de produção.

2.4. Aplicações de Gêmeos Digitais

Com a evolução exponencial dos dispositivos de transmissão da informação, maior facilidade de acesso e uso de inovações, Gêmeos Digitais têm contribuído para o desempenho e eficiência em várias áreas nos últimos anos, como por exemplo: turbinas eólicas (Moghadam e Nejad, 2022), sistemas de climatização HVAC (*Heating, Ventilation, and Air Conditioning*) (Heems, 2018), edifícios (Lydon *et al.*, 2019), motores de aeronaves (Tuegel *et al.*, 2011), agricultura (Pylianidis, Osinga e Athanasiadis, 2021; Verdouw *et al.*, 2021), e até mesmo cidades inteiras (Cousins, 2017; Goh, 2015). Incontestavelmente, a tecnologia em questão demonstra enorme flexibilidade, e mesmo com alguns desafios e obstáculos enfrentados para sua implementação, tem ocorrido a sua adaptação aos mais diferentes contextos. Similarmente, surgiram aplicações na indústria de E&P (Lu *et al.*, 2020; Danielsen-Haces, 2018; Bole, Powell e Rousseau, 2017; Renzi *et al.*, 2017) com a finalidade de otimizar ou criar metodologias de produção em um mercado cada vez mais competitivo.

Além da possibilidade de ser implementado em diferentes tipos de plataformas *offshore*, um Gêmeo Digital é, na maioria das vezes, instaurado separadamente nos sistemas e subsistemas dos ativos (por exemplo: maquinário, equipamentos, estrutura, sistemas de controle e produção) (Poddar, 2018). Essa possibilidade de repartição de níveis é o principal agente responsável pela integração, organização, segurança e controle das repartições dos ativos trazendo amplas melhorias até mesmo antes da fase de operação.

2.5. Desafios na implementação de Gêmeos Digitais

Diante de toda a complexidade relacionada ao desenvolvimento de um Gêmeo Digital para um FPSO, por se tratar de uma diversidade de sistemas acoplados à embarcação e condições externas, é necessária uma abordagem com base na experiência prévia de trabalhos de implementação da tecnologia (Rasheed, San e Kvamsdal, 2019; Bolton *et al.*, 2018; Bole, Powell e Rousseau, 2017; Christodoulou, 2015). São mencionados desde simples obstáculos, como a falta de compartilhamento de experiência entre profissionais, até as falhas mais graves, como a falta de manutenção do Gêmeo Digital ao longo do tempo, tornando-o inútil. Os problemas mais notórios, são descritos nos tópicos a seguir:

2.5.1. Disparidade de conhecimento entre profissionais

O desequilíbrio entre experiência e adoção de novas tecnologias pelos colaboradores das empresas, muitas vezes representa uma dificuldade para a implementação de um Gêmeo Digital. É preciso a aproximação dos engenheiros com maior e menor experiência para o compartilhamento de conhecimento, não apenas conhecimento empírico, mas também aprendizado em tecnologia. Um estudo realizado por Dobson (2013) no mercado de trabalho da engenharia naval, relata que, quando se trata de adoção de novas tecnologias, existem 3 classes de grupos distintos: o primeiro formado por funcionários com maior experiência ocupando cargos de gerência, próximos da aposentadoria, que geralmente, podem não demonstrar entusiasmo em adotar novas ideias e tecnologias; o segundo grupo é formado por funcionários mais novos, que cresceram na era dos computadores, usam simulação computacional para resolução de problemas, aceitam e adotam novas ideias com facilidade e facilmente podem abandoná-las se forem de uso limitado, apesar da menor experiência entre os três grupos, eles sempre irão descobrir como algo funciona por meio de buscas na internet; o último grupo é o intermediário entre os dois citados anteriormente, foram expostos à tecnologia, porém nem todos optaram por utilizá-la em parte da carreira, eles enxergam os benefícios da tecnologia mas ficam frustrados pela resistência à adoção do primeiro grupo. Essa disparidade de conhecimento dos profissionais pode tornar a habilidade da operação de uma tecnologia, que ainda pode estar em processo de familiarização como o Gêmeo Digital, concentrada em poucos funcionários, fazendo com que não ocorra o gerenciamento apropriado do FPSO.

2.5.2. Informações de projeto

As informações geradas durante o processo de projeto e produção do FPSO é de extremo valor para a implementação do Gêmeo Digital. Dados relevantes são obtidos durante o processo de produção, instalação, comissionamento de componentes e sistemas, os quais devem ser organizados e fornecidos aos operadores das embarcações para auxílio durante a sua vida útil. Já durante a operação, essas informações precisam ser mantidas e atualizadas para quaisquer alterações feitas ao longo de toda a vida útil do navio. Todo esse conteúdo deve ser disponibilizado e exibido àqueles que operam e mantêm o FPSO, pois em caso de necessidade, são eles os responsáveis, em conjunto com o DSS, em agir de maneira efetiva para impedir, evitar a propagação de incidentes ou mitigar a falha em qualquer cenário inseguro ou não econômico com acontecimento de um imprevisto. Os estaleiros seriam as unidades a oferecer soluções de Gêmeos Digitais a seus clientes como um serviço adicional, para que a documentação mais eficaz fosse entregue antes de se tornar obsoleta ou desorganizada. No entanto, na maioria dos casos não é dada a devida importância a tais informações.

2.5.3. Manutenção do Gêmeo Digital

Na atualidade, há uma grande facilidade na coleta, transmissão e armazenamento dos dados sobre as operações dos FPSOs. No entanto, é preciso ter ciência de que o cruzamento e tratamento dessa vasta quantidade de informações precisa ser mantido constantemente para o correto funcionamento do Gêmeo Digital ao longo da vida útil do ativo. Isso implica em uma série de cuidados contínuos para que dados sejam organizados para servir de atualização para o desempenho do Gêmeo Digital:

- Conversão de dados de múltiplos subsistemas para torná-los compatíveis com o Gêmeo Digital;
- Versões diferentes ou atualizações de programas ou extensão de arquivos podem acarretar perda dos dados por incompatibilidade;
- Mudança no uso de programas por descontinuidade de desenvolvimento, término de licença ou quebra de contrato;
- Atenção com mudança nos procedimentos de operação ou monitoramento de dados;

- Treinamento adequado dos colaboradores;
- Manutenção de sensores e serviços de previsões climáticas;
- Manutenção de uma equipe permanente dedicada à manutenção do gêmeo digital (especialmente quando um dos propósitos do gêmeo digital é a avaliação de riscos durante ocorrências acidentais).

Eventualmente, a qualidade das informações também podem interferir no DSS do Gêmeo Digital, ocasionando alertas de sugestões para manutenções desnecessárias ou interrupção da operações do FPSO. Por isso, é desejada uma boa calibração dos sistemas acoplados que fornecem informações.

2.6. Cenários de atuação

Gêmeos Digitais podem ser aplicados em circunstâncias extremamente amplas. Por esse motivo, as considerações dos cenários ao qual o ativo irá exercer suas funções é de grande importância. A perspectiva utilizada durante o seu desenvolvimento, em acordo com os objetivos que se pretende alcançar, é o que determina o enfoque do produto final. Naturalmente, um Gêmeo Digital pode fracassar em suas deliberações se este for elaborado com sua concepção voltada para um contexto diferente da realidade. Perante os diferentes cenários de atuação do FPSO de acordo com as fases do ciclo de vida das instalações, devem ser analisadas adaptações necessárias para o desenvolvimento do Gêmeo Digital. A seguir são descritas e enumeradas as fases do ciclo de vida de instalações de acordo com ANP (2019):

1. Projeto, comissionamento e pré-operação;
2. Operação;
3. Cessão de direitos e obrigações;
4. Extensão de Vida;
5. Descomissionamento.

Basicamente, um Gêmeo Digital para previsão de falhas estruturais em casco de FPSO deve ter como propósito o auxílio das fases projeto, pré-operação e operação, sendo uma das principais ferramentas de assistência da fase de extensão de vida (caso seja aprovada pelo órgão regulador do setor). O Gêmeo Digital pode também

ser um instrumento de avaliação da unidade na fase descomissionamento, uma vez que já estão reunidos e atualizados todos os dados e histórico da embarcação.

A criação de um modelo global que seja realístico e integrado com vários subsistemas do FPSO, interpretando diversas variáveis da forma mais coerente em diferentes fases do ciclo de vida, se impõe como um dos maiores desafios. É notável que um fator que resulta numa maior complexidade é a escala de consideração dos subsistemas, visto que quanto menor a escala de consideração destes, maior a fidelidade do modelo e consequentemente maior a dificuldade de criação do Gêmeo Digital. Da mesma maneira, em muitos casos, deve-se evitar o erro de simplificar um modelo global de forma que detalhes de grande importância passem a ser negligenciados. No âmbito estrutural, um Gêmeo Digital requer uma solução rápida e precisa, pois é necessário solucionar modelos de alta hierarquia e prover resultados satisfatórios para tomada de decisão com velocidade e eficiência, por isso, equilíbrio entre tempo exequível e nível de precisão dos resultados é fundamental.

Com base nos desafios discutidos anteriormente, devem ser dispostos os fatores determinantes para que o Gêmeo Digital atenda às necessidades com foco nas fases de projeto, pré-operação e operação estabelecidas pelo ciclo de vida do ativo. Cada uma dessas fases demonstram certas particularidades essencialmente ligadas às definições de fatores determinantes feitas abaixo:

- Tipo das variáveis: não definidas (processo estocástico) ou definidas (processo determinístico);
- Frequência de execução das análises numéricas: discreta (uma única execução) ou contínua (execuções periódicas);
- Conjunto de dados: único (representação de apenas um instante ou condição) ou múltiplo (vários conjuntos únicos);
- Instante avaliado: real (simulação considerando o instante atual) ou previsto (tenta avaliar um evento futuro);
- Resposta do DSS: quase instantâneo (caso o instante avaliado seja em tempo real) ou não instantâneo (caso o instante avaliado seja uma hipótese de previsão).

Se observado o cenário de fase de operação, é esperado que sejam submetidas variáveis obtidas da instrumentação de monitoramento que representem um instante atual com uma certa periodicidade de tempo. Portanto, é aguardado um conjunto

único de dados para uma análise estrutural se repetindo regularmente, e, após cada análise, é requerida uma recomendação imediata do DSS para que qualquer ação possa ser tomada, se necessária. Contrariamente, no cenário de fase de projeto, podem ser submetidos múltiplos conjuntos de variáveis para testes, por uma ou mais vezes. Porém, nesta fase, o gerenciamento do FPSO não exige necessariamente uma resposta instantânea do DSS.

Em vista das observações dos desafios já enfrentados na implementação da tecnologia, foram fundamentados os fatores determinantes a serem atendidos para funcionamento do Gêmeo Digital de acordo com os cenários das fases do ciclo de vida do FPSO (Tabela 2.1).

Tabela 2.1.: Fatores determinantes a serem atendidos para funcionamento do Gêmeo Digital de acordo com os cenários das fases do ciclo de vida do FPSO.

	Fase do ciclo de vida	
Fator determinante	Projeto	Operação
Tipo das variáveis	Não definidas	Definidas
Frequência de execução	Discreta	Contínua
Conjunto de dados	Múltiplo	Único
Instante avaliado	Previsto	Real
Resposta do DSS	Não instantâneo	Quase instantâneo

É importante ressaltar que a definição dos fatores determinantes em alguns casos requer avaliação da ocasião e não apenas da fase ciclo de vida. Por exemplo, caso ocorra uma colisão com o casco do FPSO em fase operação, o tempo necessário para a aquisição de informações e a modelagem dos danos causados na estrutura não permitiriam a definição dos fatores determinantes para a fase de operação apresentados na Tabela 2.1. Portanto, é sempre necessário observar se o fator determinante se adequa à condição a qual o ativo é submetido. Um cuidado especial deve ainda ser tomado ao se estabelecer o fator determinante para a resposta do DSS, a qual pode ser bastante variável de acordo com a ocasião, quantidade de dados e tamanho do modelo de elementos finitos.

2.7. Exploração de petróleo e gás por FPSO

As necessidades e viabilidade econômica são o que ampara a escolha do tipo de plataforma mais adequada à exploração do campo de petróleo. As plataformas podem

ser de perfuração, de produção ou ter as duas funções. A Tabela 2.2 apresenta um comparativo entre os tipos de plataformas existentes, e a seguir, são descritas resumidamente as principais características de cada uma dessas plataformas Petrobras (2021):

- Fixa: estrutura rígida usada na perfuração de poços e produção de petróleo; apropriada para águas rasas; é fixada no fundo do mar por estacas cravadas para operações de longa duração. Se destaca por ser de fácil instalação e permitir que o controle dos poços seja feito na superfície.
- Autoelevável (ou autoelevatória ou *jack-up*): usada na perfuração de poços em águas rasas, tem pernas que alcançam e se firmam no solo do mar. Apresenta como principal vantagem a facilidade de locomoção, uma vez que suas pernas podem ser recolhidas. O comportamento de estrutura fixa permite que o controle dos poços seja feito na superfície.
- Semissubmersível: unidade flutuante empregada na perfuração de poços e produção de petróleo, tem grande mobilidade e é estabilizada por colunas. Pode ser instalada em águas profundas através de sistemas de ancoragem modernos dotada de sistema de posicionamento dinâmico, que mantém a posição da plataforma de forma automática. Especialmente projetada para ter pouco movimento.
- FPSO: plataforma flutuante capaz de produzir, armazenar e transferir petróleo. Pode ser convertida a partir de navios petroleiros e é ancorada no solo marinho. Apresenta grande vantagem na exploração de águas profundas, ultraprofundas e locais mais isolados por sua capacidade de armazenamento, permitindo operações em grandes distâncias da costa, onde a construção de oleodutos é inviável.
- FPSO Monocoluna: tem as mesmas características do FPSO, mas seu casco tem formato cilíndrico. Tem grande vantagem por apresentar movimentos menores em comparação ao FPSO tradicional, podendo operar em condições ambientais mais adversas.
- TLP (*Tension0Leg Platforms* ou plataforma de pernas atirantadas): plataforma flutuante usada na produção de petróleo. O sistema de ancoragem por cabos ou tendões de aço tracionados proporciona movimentos reduzidos, permitindo o controle de poços na superfície.
- Navio-Sonda: unidade flutuante usada para perfuração de poços em águas

ultraprofundas. Geralmente, possui tecnologias avançadas para assegurar boa estabilidade, assim trazendo maior autonomia para perfurar poços em grandes distâncias da costa.

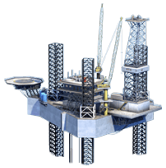
Os desafios superados para construção de estruturas *offshore* aptas à exploração e produção de petróleo em águas profundas representam um marco para a história do setor. Foram mais de 70 anos de desenvolvimento tecnológico para viabilização das atividades petrolíferas em tais condições de complexidade, tendo a Petrobras como pioneira mundial em progredir para atuação em profundidades cada vez maiores (Chakrabarti, Halkyard e Capanoglu, 2005).

O FPSO demonstra ser uma solução mais eficiente e conveniente à exploração de petróleo, sendo mais lucrativo e funcional para a indústria *offshore* ou, em muitos casos, a única alternativa viável. Devido a este fato, este tipo de plataforma se tornou mais popular na década de 1990 e teve sua demanda expandida (Chakrabarti, Halkyard e Capanoglu, 2005) a medida que as empresas se interessaram pela exploração de águas profundas e ultraprofundas.

Esse tipo de plataforma se desenvolveu num contexto da E&P onde havia a necessidade de uma unidade com grande capacidade de armazenamento que pudesse atender regiões mais afastadas da costa (onde a construção de oleodutos é inexequível e onerosa). Além disso, alguns FPSOs possuem a vantagem de permitir a desconexão do *turret* ou sistema de ancoragem em situações de risco, como a presença de *icebergs* ou furacões. Essa capacidade garante a segurança da embarcação ao se afastar de perigos iminentes.

Com todas essas funcionalidades, o setor de exploração *offshore* tem se voltado para a utilização de FPSOs pelo melhor custo-benefício. A importância dos FPSOs para o mercado de E&P e tendência de emprego deste tipo de plataforma no Brasil e no mundo foram apresentados na Seção 1.1 e Seção 1.3.

Tabela 2.2.: Comparativo entre os diferentes tipos de plataformas (adaptado de Petrobras (2021)).

							
Plataforma	Fixa	Autoelevável	Semissubmersível	FPSO	FPSO Monocoluna	TLP	Navio-Sonda
Lâmina d'água	$\leq 300m$	$\leq 150m$	$>$ 2000m	$>$ 2000m	$>$ 2000m	$\leq 1500m$	$>$ 2000m
Atividade de perfuração	Sim	Sim	Sim[1]	Não	Não	Sim[2]	Sim
Atividade de produção	Sim	Não	Sim[3]	Sim	Sim[4]	Sim	Não
Controle dos poços	Superfície	Superfície	Fundo do mar	Fundo do mar	Fundo do mar	Superfície	Fundo do mar
Armazenamento	Não	Não	Não	Sim	Sim	Não	Não
Escoamento da produção	Oleodutos	Não	Oleodutos[5]	Navios[6]	Navios[6]	FPSO[7]	Não

1. Algumas podem ser só de produção;
2. Só para a manutenção dos poços;
3. Algumas podem ser só de perfuração;
4. Geralmente são unidades de perfuração ou de produção;
5. Ou armazenamento em navios e posterior descarregamento nos terminais;
6. O óleo é exportado para navios petroleiros, que o descarregam nos terminais;
7. O óleo é escoado para uma plataforma de produção (FPSO), que realiza o processamento e o exporta através de navios.

2.8. O casco do FPSO

Toda embarcação tem a definição de seu casco desenvolvido conforme sua destinação, desempenhando um balanço entre custo e desempenho de modo que resista às condições críticas do ambiente agressivo que ele deve operar. A enorme quantidade de variáveis a serem consideradas durante o projeto do casco (por exemplo: propriedades dos materiais utilizados, custo, facilidade de manutenção e disposição dos compartimentos), tornou esse tema bastante pesquisado ao longo dos anos (Ang *et al.*, 2017; Yang e Huang, 2016; Tammer *et al.*, 2014; Caldwell, 2013; MacMillan, 2001; Terpstra, 2001; Bai, Bendiksen e Terndruppedersen, 1993).

Apesar da grande variedade disponível de formatos de cascos, a configuração da unidade estrutural típica dos navios são pensadas de maneira semelhante a uma viga. Esta estrutura é comumente conhecida como "viga navio" e deve suportar forças e momentos distribuídos ao longo do seu comprimento (Augusto, 2007).

Partindo da perspectiva de viga navio, a estrutura primária é a responsável por resistir aos efeitos globais dos esforços. É válido evidenciar que a distribuição das cargas e forças de flutuação são, sempre, desiguais no eixo longitudinal do navio e a viga navio é submetida a tensões de flexão e cisalhamento. O momento de flexão longitudinal (exemplificado como M_z na Figura 2.3) corresponde ao seu valor máximo a meia-nau[1] e zero nas extremidades da embarcação. Normalmente, o campo de tensões na seção mestra[2] equivale à metade do limite de escoamento do material.

Os efeitos locais dos esforços são resistidos pelas estruturas secundárias (reforçadores) e terciárias (chapas). O painel estrutural ou chapeamento reforçado (Figura 2.4) é responsável por garantir a estanqueidade do navio, sendo composto por perfis enrijecedores leves e pesados dispostos em direções ortogonais selecionados pelo projetista com base na região, eficiência estrutural, custos, continuidade estrutural e utilização do espaço (Augusto, 2007). O agrupamento desses painéis estruturais é o que molda a estrutura básica das embarcações tradicionais, constituídas por anteparas longitudinal, transversal, escoas e quilha, como mostrado na Figura 2.5.

Grandes catástrofes ambientais podem ser evitadas de acordo com o modelo de casco

[1] Linha média que divide uma embarcação em duas metades iguais, sendo equidistante da proa e popa.

[2] A maior das seções transversais de um casco. A seção mestra se situa coincidentemente com a seção a meia-nau ou muito próximo desta na maioria dos navios modernos qualquer que seja o tipo.

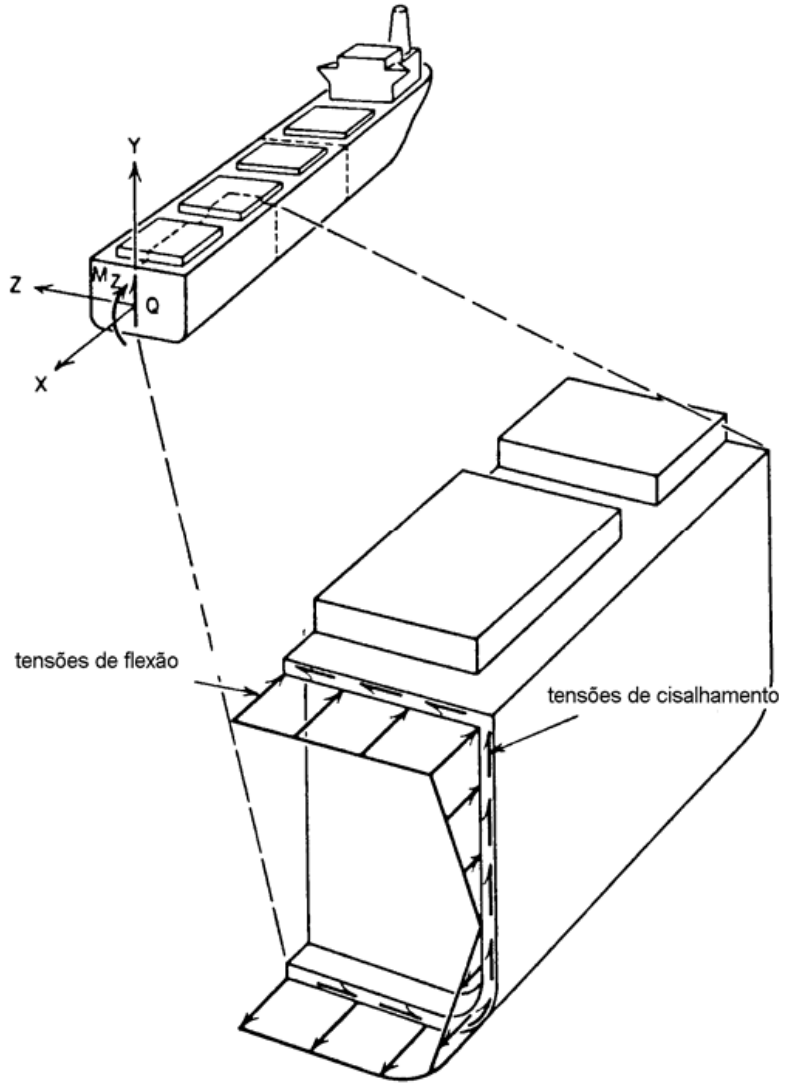


Figura 2.3.: Tensões primárias na viga navio (Hughes, 1983).

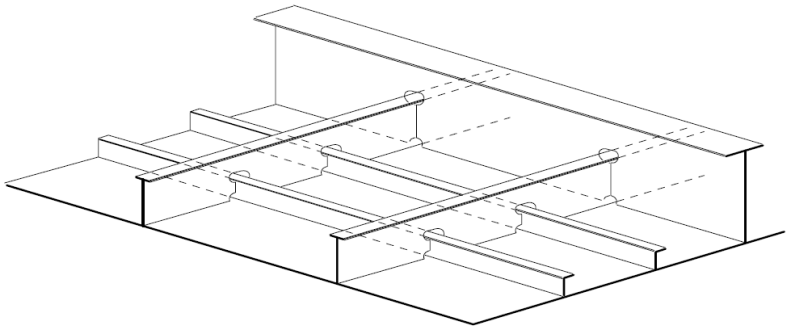


Figura 2.4.: Painel estrutural (Augusto, 2007).

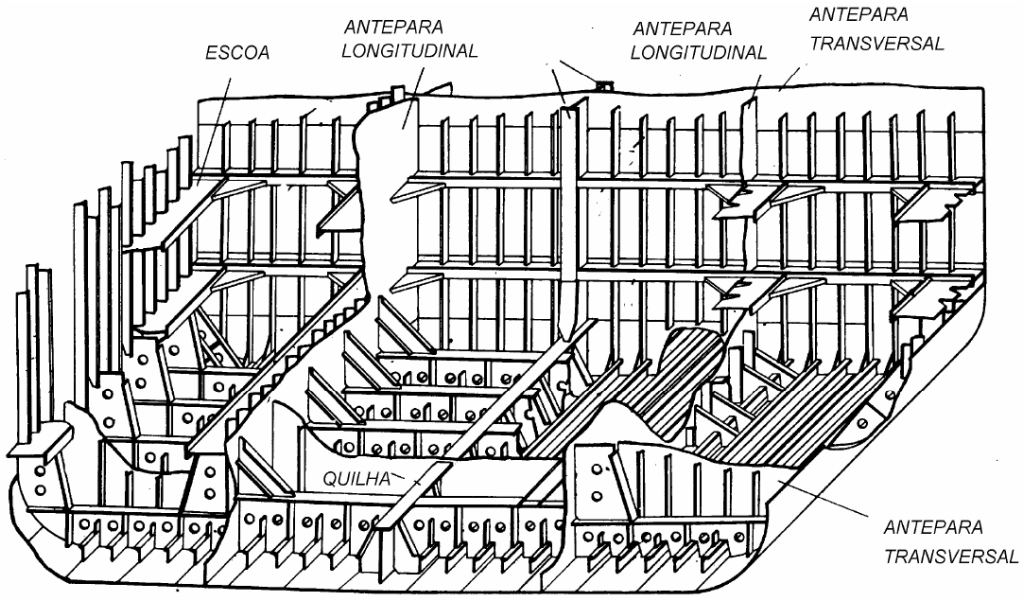
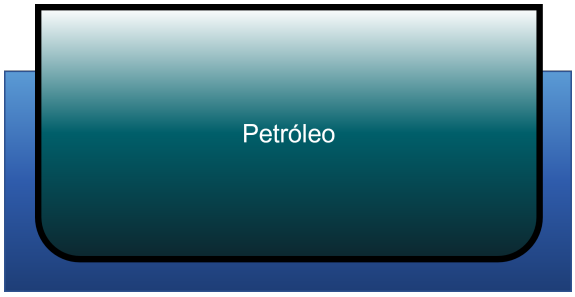


Figura 2.5.: Estrutura do fundo de um navio tanque de casco singelo (Augusto, 2007).

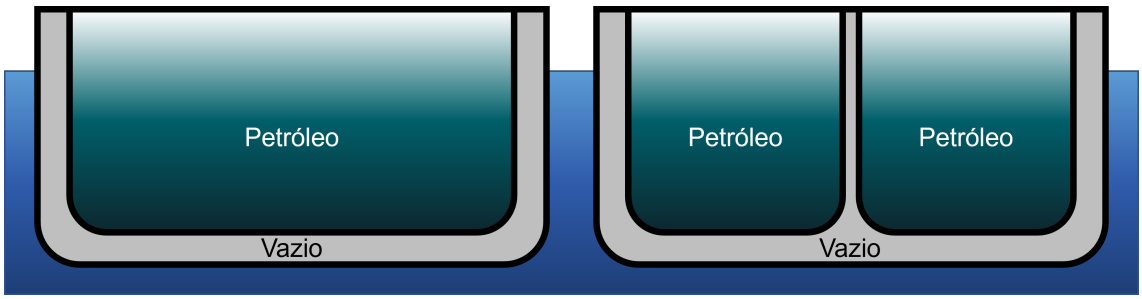
utilizado pelas embarcações petrolíferas e, por isso, foram instauradas recomendações para a construção de cascos de navios, primeiramente, pelo OPA (*Oil Pollution Act*) nos Estados Unidos da América em 1990, e posteriormente pelas emendas à convenção internacional para prevenção à poluição por navios (MARPOL - *International Convention for the Prevention of Pollution from Ships*) entre 1992 e 2003. A principal orientação por essas mudanças foi a utilização de casco duplo para navios petroleiros com mais de 5.000 toneladas brutas com escala na maioria dos principais portos internacionais (DeCola, 2009). Embarcações de casco duplo têm uma camada dupla de proteção estanque que percorre toda a extensão do navio, com camadas interna e externa presentes tanto na parte inferior quanto na lateral dos tanques. Já os navios que utilizam casco simples ou singelo, têm apenas uma camada externa à prova d'água e esta camada percorre toda a estrutura do navio (a Figura 2.6 ilustra a diferença entre o casco simples e o casco duplo).

Os FPSOs não são especificamente referidos por essas regulações. No entanto, como essas determinações foram originalmente planejadas para o comércio de navios-tanque, a maioria dos fabricantes atuais incluem o casco duplo no projeto (Sipkema, 2004).

A MARPOL fornece diretrizes mínimas para a construção de embarcações dispondo de casco duplo, cuja característica tem sido recomendada por muitos projetistas navais (Terpstra, 2001). Apesar de ser notável o potencial do casco duplo para redução



(a) Casco simples.



(b) Cascos duplos.

Figura 2.6.: Exemplos de configurações típicas de cascos.

dos vazamentos (Yip, Talley e Jin, 2011), precavendo-se contra derramamentos em caso de falha estrutural, existe uma enorme discussão sobre o seu emprego. A grande controvérsia sobre a utilização de cascos duplos nas embarcações aborda diversos pontos segundo alguns autores e representantes da indústria. Primeiramente, esse novo tipo de projeto não evita derramamento de petróleo em caso de colisões ou encalhes de alta energia (Devanney, 2006), contrariando seu principal propósito. Diversas dificuldades surgiram com os novos modelos de cascos (por exemplo: manutenção, corrosão estrutural, cobertura dos tanques, tensão e fadiga do casco) (DeCola, 2009; CEIDA, 2003). Um projeto de casco duplo demanda cerca de 20% a mais de aço quando comparado com seu equivalente de casco simples (Okumoto *et al.*, 2009), no entanto, o peso total de aço não deve ser superior aos projeto de casco simples, assim resultando em chapas geralmente menos espessas e resistentes. Por fim, existe ainda um maior risco de explosão em razão da possibilidade de vazamento de gases do casco interno para a bolsa de ar do casco duplo (TRB, 2001). Em uma pesquisa conduzida pelo *National Research Council* (NRC, 1998), poucas vantagens (estas relacionadas a velocidade de carregamento e limpeza dos tanques) foram observadas com o novo modelo de casco e foram mencionadas desvantagens relacionadas à segurança estrutural, estabilidade e principalmente o alto

custo de construção e dificuldade de manutenção. Assim, expondo a possibilidade das embarcações de casco duplo serem até mesmo mais problemáticas do que suas equivalentes de casco singelo.

2.9. Condições e carregamentos

Uma vez definida a estrutura do casco, ela necessita resistir aos diferentes carregamentos sob diferentes condições durante sua vida útil sem sofrer falhas ou deformações permanentes que ameacem a segurança humana, ambiental e operacional.

Devido ao ambiente agressivo que qualquer estrutura *offshore* é exposta, é fundamental que seja considerada a corrosão ao longo dos anos, principal componente na redução de espessura de chapas e reforçadores. Em atenção a este aspecto, algumas normas são publicadas e revisadas com periodicidade. A ABS (2022a) e DNV (2015) fornecem procedimentos de manutenção, segurança de projeto e estimativas de taxa de corrosão. Além da espessura necessária para resistência mínima de projeto, deve ser adicionada a proteção de revestimento para tanques de lastro[3] e valores de margem de corrosão (NDCV - *Nominal Design Corrosion Values*) de acordo com as características de cada região e elemento estrutural da embarcação. A Tabela 2.3 e a Figura 2.7 apresentam os NDCV por região adotados pela norma da ABS para projetos com vida útil de projeto de 20 anos ou menos. É importante ressaltar que a norma contempla algumas observações para regiões ou membros estruturais com alta taxa de corrosão (como locais de concentração de tensão) e recomendações para projetos com vida útil de projeto estimada em mais de 20 anos.

[3]Compartimento especial dos navios que tem seu nível controlado com água para equilibrar a embarcação.

Tabela 2.3.: NDCV para vida útil de projeto de 20 anos fornecidos pela ABS (2022a).

Structural Element/Location		Nominal Design Corrosion Values (mm)		
		Cargo Tank	Ballast Tank Effectively Coated	Void Space
Deck Plating		1.0	2.0	1.0
Side Shell Plating		NA	1.5	1.0
Bottom Plating		NA	1.0	1.0
Inner Bottom Plating		1.5		1.0
Longitudinal Bulkhead Plating	Between cargo tanks	1.0	NA	1.0
	Other Plating	1.5		1.0
Transverse Bulkhead Plating	Between cargo tanks	1.0	NA	1.0
	Other Plating	1.5		1.0
Transverse & Longitudinal Deck Supporting Members		1.5	2.0	1.0
Double Bottom Tanks Internals (Floors and Girders)		NA	2.0	1.0
Double Bottom Tanks Internals (Stiffeners)		NA	2.0	1.0
Vertical Stiffeners and Supporting Members Elsewhere		1.0	1.0	1.0
Non-vertical Longitudinals/Stiffeners and Supporting Members Elsewhere		1.5	2.0	1.0

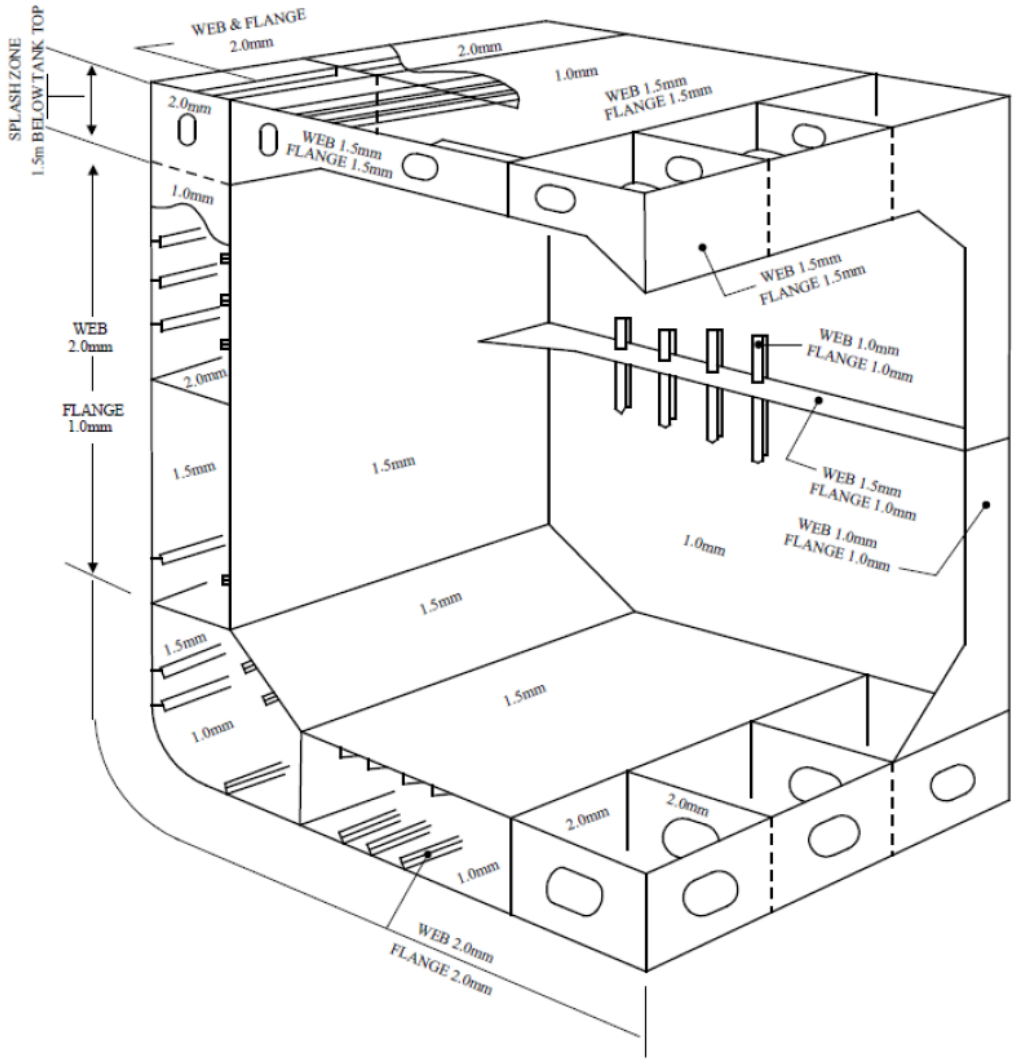


Figura 2.7.: NDCV por regiões, para vida útil de projeto de 20 anos (ABS, 2022a).

Uma das maiores dificuldades em qualquer modelo numérico é o entendimento do mundo físico, visto que há um enorme empenho para que os fenômenos reais sejam adequadamente capturados, compreendidos e simulados computacionalmente de maneira a representar o seu comportamento real. No caso de unidades flutuantes, esta tarefa é ainda mais difícil, pois existe a incisiva incerteza de cargas dependentes da movimentação da embarcação, condições climáticas e localização geográfica.

Diferente da maioria das embarcações, um FPSO pode ser visto como uma unidade estacionária, que por sua vez é um corpo rígido com seus movimentos influenciados pela oscilação das ondas e forças de tração dos sistemas de linhas e cabos conectados à estrutura do casco. Embarcações possuem seis graus de liberdade, dentre eles são três movimentos de translação e outros três de rotação (ver Figura 2.8):

- *Surge* (avanço): movimento de translação no eixo longitudinal (X);
- *Sway* (deriva): movimento de translação no eixo transversal (Y);
- *Heave* (afundamento): movimento de translação no eixo vertical (Z);
- *Roll* (jogo): movimento de rotação no eixo longitudinal (X);
- *Pitch* (arfagem): movimento de rotação no eixo transversal (Y);
- *Yaw* (guinada): movimento de rotação no eixo vertical (Z).

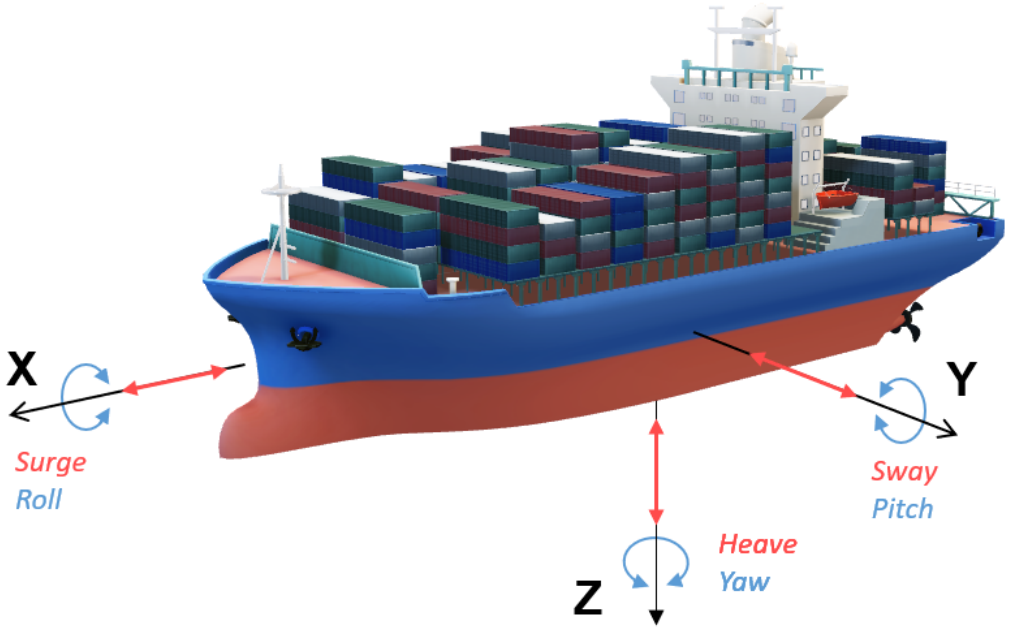


Figura 2.8.: Graus de liberdade de uma embarcação.

A numerosa quantidade de variáveis envolvidas, que não podem ser estimadas por processos determinísticos, faz com que os projetistas definam carregamentos que representem conjunturas hipotéticas para a concepção da estrutura da embarcação. Algumas normas (ABS, 2022a; DNV, 2014) abordam os carregamentos estruturais baseados em dados estatísticos e modelos matemáticos que descrevem a gama de variações às quais as embarcações são submetidas. Estes carregamentos podem ser divididos em três categorias:

1. Cargas estáticas: podem ser mais facilmente determinadas (apesar de haver variação dos componentes relacionados a essas cargas, como volume armazenado e temperatura). Alguns exemplos são citados abaixo:
    - Peso próprio (material, máquinas e equipamentos);
    - Carregamento operacional (induzido pelo volume armazenado);
    - Pressão hidrostática interna (induzida pelo volume armazenado);
    - Pressão hidrostática externa (induzida pelo mar);
    - Efeito térmico.
2. Cargas dinâmicas: geralmente são uma consequência das perturbações decorrentes das variações de clima e movimentação da embarcação. Alguns exemplos são citados abaixo:
    - Pressões no casco (causadas pelas ondas);
    - Pressões internas e externas (efeitos de *slamming*[4] e *sloshing*[5]);
    - Força do vento;
    - Cargas inerciais (devido à aceleração e velocidade originados da movimentação da embarcação);
    - Forças provenientes de outros sistemas (linhas de ancoragem, *risers*[6], *turret*[7]);
    - Forças periódicas do sistema de propulsão.

[4]Esforço dinâmico causado pela constante emersão e imersão da proa.
[5]Movimento de fluidos em tanques parcialmente preenchidos, resultando em carregamentos na parede.
[6]Dutos que fazem a ligação entre os poços de petróleo e plataformas.
[7]Estrutura composta por um corpo central cilíndrico, conectado internamente ou externamente ao casco de uma embarcação através de rolamentos e uniões rotativas, que permite a conexão radialmente das linhas de amarração e dos *risers* em um ponto central.

3. Cargas excepcionais ou acidentais: são causadas por ações esporádicas no tempo de vida operacional das embarcações (como encalhe, abalroamento e explosões), sendo mais comuns apenas para navios mais suscetíveis a impactos (como navios quebra-gelo[8] ou de caráter militar). Alguns exemplos são citados abaixo:
    - Cargas de gelo;
    - Cargas de impacto (por exemplo: mísseis e, ou explosões.);
    - Cargas severas (durante lançamento, docagem e atracamento);
    - Cargas acidentais (encalhe, choque e abalroamento).

É impraticável avaliar ou prever todos os carregamentos impostos com exatidão devido a vasta quantidade de variáveis associadas a condições ambientais, principalmente para cargas dinâmicas. No entanto, a maior parte das cargas tem relação com o comportamento das ondas. A tentativa de descrição da perturbação das superfície da água, denominada estado de mar, ocorre há alguns séculos e surgiu pela observação de marinheiros experientes, se tornando cada vez mais complexa e detalhada à proporção que instrumentos de monitoramento se tornaram mais precisos (Ardhuin *et al.*, 2019). Essa condição geral da agitação marinha é caracterizada, na maioria das vezes de maneira simplificada, por meio da informação da altura, período e direção da onda.

Apesar de ondas regulares não representarem de maneira realística as condições de estado de mar, elas são recomendadas (sob certos critérios para que a integridade estrutural seja corretamente avaliada) por algumas normas (ABS, 2022a; DNV, 2014). Uma onda regular pode ser descrita por simples parâmetros como altura e período, apesar de envolver uma série de outros parâmetros (Figura 2.9).

[8]Navio projetado com características especificas para navegação em águas cobertas por gelo.

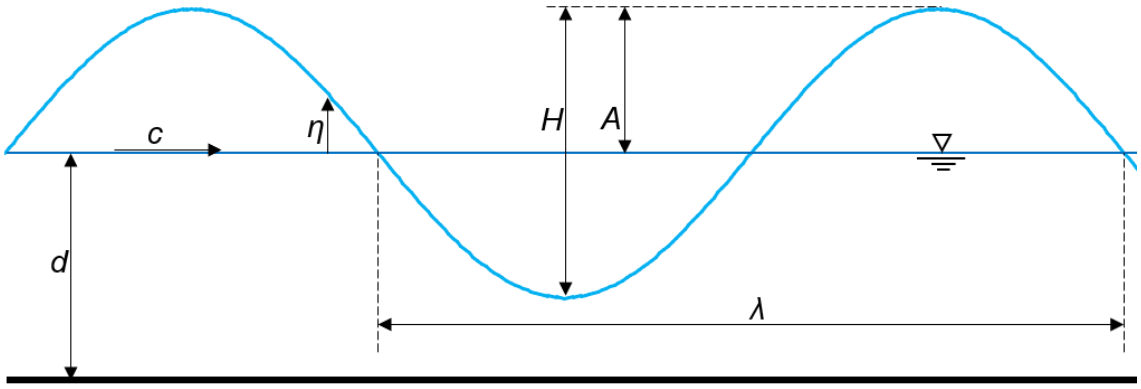


Figura 2.9.: Onda regular típica.

- Altura (H): distância vertical entre uma crista (máxima elevação) e o cavado (mínima elevação) adjacente;
- Período (T): intervalo de tempo entre a ocorrência de cristas (ou cavados) sucessivos, numa posição fixa;
- Amplitude (A): metade da altura de onda, $A = H/2$;
- Frequência (f): número de cristas (ou cavados) que ocorrem numa posição fixa por unidade de tempo, $f = 1/T$;
- Frequência angular (ω): relação $\omega = 2\pi/T$;
- Comprimento de onda (λ): distância horizontal entre cristas (ou cavados) consecutivas, na direção de propagação da onda;
- Velocidade de fase (c): velocidade na qual a onda viaja, $c = \lambda/T$;
- Número de onda (k): relação $k = 2\pi/\lambda$;
- Velocidade de grupo (cg): velocidade na qual a energia das ondas se propaga, $cg = d\omega/dk$;
- Elevação (η): distância vertical instantânea de um ponto da superfície a um nível representando a superfície sem distúrbio;
- Declividade: relação H/λ.

Modelos de ondas aleatórias e irregulares podem representear de forma mais realista o comportamento do mar, podendo ser a combinação dos efeitos de ondas lineares com diferentes parâmetros (ABS, 2022a; DNV, 2014). Segundo Riva (2004), em um período de 3 horas, as propriedades estatísticas do estado de mar podem ser consideradas constantes e o mar pode ser denominado estacionário (onde as propriedades estatísticas não variam com o tempo).

O estado de mar é, frequentemente, descrito por um espectro de densidade de energia, que pode ser demonstrado por um histórico de tempo de um processo estocástico, podendo ser decomposto em ondas regulares.

Ondas irregulares podem ser caracterizadas com os parâmetros de altura de onda significativa (H_s), que é a média do terço das ondas de maior altura (ver Figura 2.10), e período entre zeros-ascendentes, que é a média dos valores dos períodos da onda (T_i). Estes dois parâmetros podem ser obtidos dos registros de elevações do mar (exemplificado na Figura 2.11).

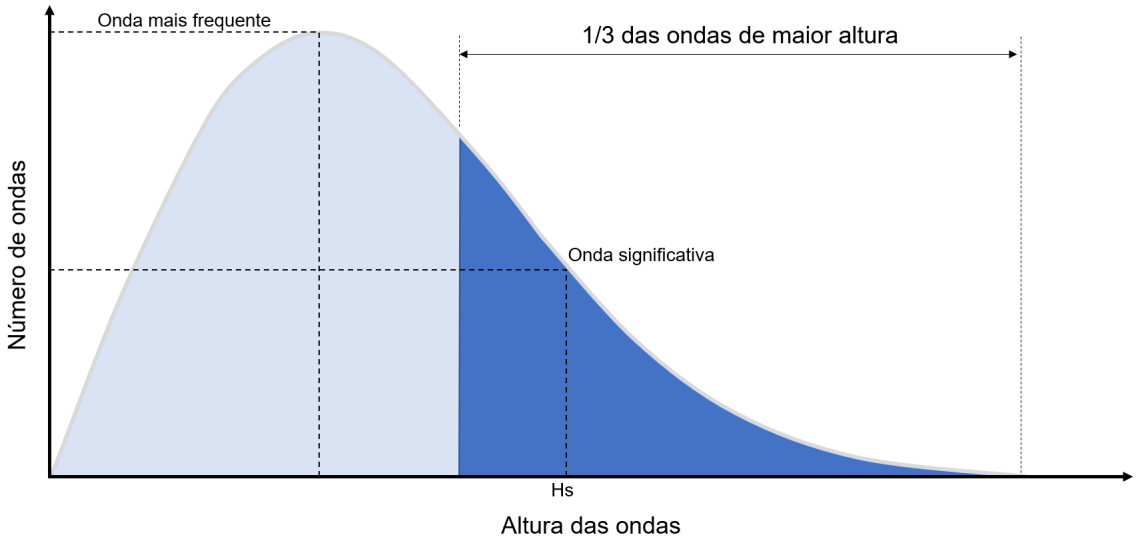


Figura 2.10.: Distribuição estatística da altura de ondas.

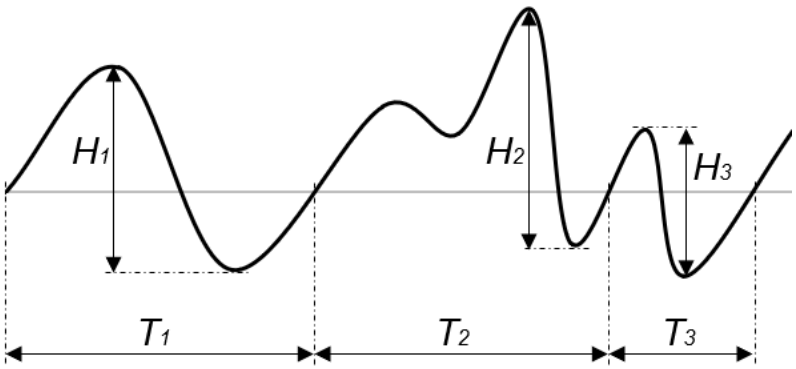


Figura 2.11.: Exemplo de onda irregular.

Segundo Young (1999), a altura de onda significativa também pode ser calculada a partir da densidade do espectro de energia da onda, como apresentado na Equação 2.1:

$$H_s = \int F(f)\, df = 4\sqrt{\sigma^2} \tag{2.1}$$

onde $F(f)$ é uma variável estocástica, e sendo, a elevação do mar esta variável, a altura de onda significativa pode ser representada por quatro vezes seu desvio padrão (σ).

Portanto, com base no monitoramento apropriado das condições meteoceanográficas podem ser obtidos os valores de altura de onda significativa (H_s) e períodos entre zero-ascendentes, os quais são usados para criação de um banco de dados que disponibilizam informações suficientes para diagramas com informações de ondas. Estes fornecem a probabilidade de ocorrência de cada estado de mar a longo prazo para uma região. Na Tabela 2.4 é apresentado o diagrama com alturas significativas e período de pico (T_z) de ondas mundial e na Tabela 2.5 o diagrama para ondas do Atlântico Norte.

Adicionalmente, é importante ressaltar que, a partir dos diagramas de dispersão da região de operação do FPSO (similares aos apresentados nas Tabela 2.4 e Tabela 2.5) são estimadas as condições extremas de onda. Essas condições extremas são caracterizadas por pares de H_s e T_z com período de retorno requeridos pelas regras das sociedades classificadoras, geralmente estabelecendo um período de retorno de 100 anos para condições extremas de projeto. Essas estimativas das condições ambientais extremas são essenciais para a determinação das respostas extremas, tais como tensões ou esforços internos, nas estruturas do casco do FPSO devido às condições ambientais previstas.

Tabela 2.4.: Diagrama de ondas mundial (adaptado de DNV (2014)).

$T_z(s)$	3,5	4,5	5,5	6,5	7,5	8,5	9,5	10,5	11,5	12,5	13,5	14,5	15,5	16,5	17,5	Soma
$H_s(m)$																
1,0	311	2734	6402	7132	5071	2711	1202	470	169	57	19	6	2	1		26287
2,0	20	764	4453	8841	9045	6020	3000	1225	435	140	42	12	3	1		34001
3,0		57	902	3474	5549	4973	3004	1377	518	169	50	14	4	1		20092
4,0		4	150	1007	2401	2881	2156	1154	485	171	53	15	4	1		10482
5,0			25	258	859	1338	1230	776	372	146	49	15	4	1		5073
6,0			4	63	277	540	597	440	240	105	39	13	4	1		2323
7,0			1	15	84	198	258	219	136	66	27	10	3	1		1018
8,0				4	25	69	103	99	69	37	17	6	2	1		432
9,0				1	7	23	39	42	32	19	9	4	1	1		178
10,0					2	7	14	16	14	9	5	2	1			70
11,0					1	2	5	6	6	4	2	1	1			28
12,0						1	2	2	2	2	1	1				11
13,0							1	1	1	1						4
14,0									1							1
Soma	331	3559	11937	20795	23321	18763	11611	5827	2480	926	313	99	29	9		100000

Tabela 2.5.: Diagrama de ondas para o Atlântico Norte (adaptado de ABS (2010)),

$H_s(m)$	$T_z(s)$											Soma (períodos)
	3,50	4,50	5,50	6,50	7,50	8,50	9,50	10,50	11,50	12,50	13,50	
0,5	8	260	1344	2149	1349	413	76	10	1			5610
1,5		55	1223	5349	7569	4788	1698	397	69	9	1	21158
2,5		9	406	3245	7844	7977	4305	1458	351	65	10	25670
3,5		2	113	1332	4599	6488	4716	2092	642	149	28	20161
4,5			30	469	2101	3779	3439	1876	696	192	43	12625
5,5			8	156	858	1867	2030	1307	564	180	46	7016
6,5			2	52	336	856	1077	795	390	140	40	3688
7,5			1	18	132	383	545	452	247	98	30	1906
8,5				6	53	172	272	250	150	65	22	990
9,5				2	22	78	136	137	90	42	15	522
10,5				1	9	37	70	76	53	26	10	282
11,5					4	18	36	42	32	17	7	156
12,5					2	9	19	24	19	11	4	88
13,5					1	4	10	14	12	7	3	51
>14,5					1	5	13	19	19	13	7	77
Soma (alturas)	8	326	3127	12779	24880	26874	18442	8949	3335	1014	266	100000

A ABS (2020) apresenta recomendações para o planejamento de projeto de monitoramento estrutural do casco. São apresentados métodos de monitoramento de cargas estruturais e respostas com o intuito de avaliar a integridade estrutural atual da embarcação, fazer uma previsão do estado futuro e informar a atividade de manutenção. Sensores instalados no gêmeo real (a estrutura real do FPSO) são responsáveis por monitorar as cargas e condições apresentadas. Esses sensores podem ser divididos

em três categorias: sensores de condições ambientais (vento, ondas e correntes marítimas); sensores de carga (para mensurar o nível de carregamento dos tanques); e sensores de atividades (por exemplo: acelerômetros, posicionamento via GPS - *Global Positioning System*, extensômetros). Por outro lado, o gêmeo virtual usa os dados coletados para alimentação das análises numéricas dos diferentes subsistemas (casco, sistema de amarração, linhas de ancoragem, *risers* e, ou *turret*). Numa análise acoplada, a análise estrutural do casco representa um subsistema dependente de todos esses outros subsistemas, uma vez que os esforços gerados nesses subsistemas precisam ser transferidos ao casco.

2.10. Gêmeos Digitais para evitar falhas estruturais

Os Gêmeos Digitais têm se estabelecido como um marco para o desenvolvimento de sistemas inteligentes capazes de coletar, processar e entender várias informações contextuais, com base nas quais decisões são tomadas com velocidade e eficiência. Há um grande potencial da utilização das tecnologias da informação em qualquer ramo de estudo na atualidade, por isso, existem muitos trabalhos publicados nos últimos anos com aplicações e projetos de Gêmeos Digitais. Porém, essa seção se limitará a explorar apenas casos de estudo de Gêmeos Digitais relacionados a falhas estruturais, focando especialmente naqueles que melhor detalham algum tipo de análise numérica ou procedimento de abordagem. É ressaltado por Negri, Fumagalli e Macchi (2017), que apesar da perspectiva da engenharia presumir que simulações computacionais em geral sejam um aspecto fundamental relacionado ao conceito de Gêmeo Digital, contrariando esse pensamento, na literatura alguns autores não mencionam o uso de qualquer tipo de simulação.

A maior parte dos trabalhos com simulações mecânicas computacionais se iniciaram no setor aeroespacial, reproduzindo o histórico de voos, gerando enormes bancos de dados para entender o que a aeronave experimentava e procurar prever futuras necessidades e intervenções de manutenção. Tudo isso, com grande contribuição dos mais renomados e conhecidos métodos numéricos usados em engenharia como o MEF e dinâmica de fluidos computacional (CFD - *Computational Fluid Dynamics*). O trabalho de Negri, Fumagalli e Macchi (2017) sintetiza de maneira objetiva a aplicação de Gêmeos Digitais utilizados no monitoramento de estruturas e planejamento de manutenção na literatura. De forma sucinta, o autor divide alguns trabalhos nas seguintes categorias:

(a) Monitorar anomalias, fadiga e prever fissuras de trincas no gêmeo físico;

(b) Monitorar a deformação geométrica e plástica no material do gêmeo físico;

(c) Avaliação da confiabilidade do modelo do sistema físico;

(d) Estudar o comportamento de longo prazo do sistema e prever seu desempenho, levando em consideração os diferentes efeitos sinérgicos das condições ambientais;

(e) Fornecer continuidade das informações ao longo das diferentes fases do ciclo de vida;

(f) Suporte ao comissionamento virtual do sistema;

(g) Gerenciar o ciclo de vida dos dispositivos de IoT;

(h) Otimização do comportamento do sistema durante a fase de projeto;

(i) Otimização do ciclo de vida do produto, conhecendo os estados passado e presente, possibilitando prever e otimizar os desempenhos futuros.

Um dos primeiros e mais completos trabalhos encontrados sobre a implementação de um Gêmeo Digital com a integração de múltiplas considerações de simulações computacionais para monitorar e prever falhas estruturais foi apresentado pela NASA (*National Aeronautics and Space Administration*). O projeto de Gêmeo Digital, denominado "*Virtual Digital Fleet Leader*", propôs um modelo de extrema complexidade e alta fidelidade de modelagem e simulação em tempo real de voo de modo contínuo para prover o estado de saúde estrutural, o tempo de vida restante e a probabilidade de sucesso das missões de veículos e sistemas espaciais. O trabalho de Piascik *et al.* (2010) se caracteriza pelo detalhamento e nível de integração que considera o comportamento dos materiais com simulações multiescala instantâneas, em escala microestrutural e atomística, abordando cinco categorias da área de produtos de tecnologia: materiais, estruturas, sistemas mecânicos, fabricação e tecnologias transversais. Além disso, a organização do plano de implementação do Gêmeo Digital proposta pela NASA chama a atenção não apenas pela prazo de duas décadas para sua total finalização, mas também o arranjo e definição de objetivos que precisam ser alcançados para o andamento do projeto. Dentre esses objetivos, é relatada a necessidade de implementação de modelos constitutivos capazes de simular computacionalmente e prever surgimento e propagação de fissuras, modelos de dano e falha em materiais pretendidos a serem produzidos ao longo do mesmo projeto. Estes itens são discutidos no tópico posterior que trata sobre desafios de implementação.

No mesmo contexto aeroespacial, Glaessgen e Stargel (2012) apresentam uma visão de longo prazo dos Gêmeos Digitais debatendo sobre deficiências das práticas atuais, gerenciamento, manutenção, requisitos, desenvolvimento e aplicação. Neste artigo, se destacam algumas críticas quanto à utilização de métodos apropriados e convencionais para os atuais processos de engenharia, mas que nem sempre são as melhores alternativas quando se trata de um Gêmeo Digital. Segundo o autor, os atuais fatores de segurança podem levar a produção de estruturas com sobrepeso e desempenho reduzido sem trazer qualquer melhoria. Mesmo as metodologias probabilísticas ou de confiabilidade, largamente empregadas atualmente, muitas vezes são inadequadas porque são baseadas na suposta similaridade entre circunstâncias

em que as estatísticas foram obtidas e o ambiente em que o ativo opera. A partir do momento que a similaridade é quebrada, o método probabilístico se torna impróprio para a avaliação. Um fato preocupante é que muitas vezes os engenheiros já estão tão acostumados à essas ocorrências que acabam aplicando a similaridade em casos adversos de maneira inconsciente. Um bom exemplo usado no artigo são os códigos comerciais baseados em MEF usados para prever falhas previamente observadas e implementadas em sua linguagem. Caso não haja ainda uma resposta para um novo fenômeno que ocorre, uma falha pode não ser detectada e o Gêmeo Digital acaba por não cumprir sua função. Uma outra crítica baseada nos métodos convencionais apresentada pelo autor é a aplicação do mesmo procedimento de manutenção para todas as unidades de um grupo ao se basear na pior resposta obtida da inspeção desse grupo ou parte dele sem considerar o estado atual de cada unidade e o histórico específico de cada uma. Ainda assim, é destacado que os dados estatísticos não devem ser descartados em qualquer situação, e não apenas continuam sendo de extrema importância para a abordagem geral, como passa a ser cada vez mais relevante para cada ativo conforme o tempo passa e mais problemas são solucionados e adicionados aos bancos de dados.

O setor aeroespacial é sem dúvida um dos que mais utilizam e contribuíram com o tema Gêmeos Digitais, por isso a maior parte dos primeiros estudos de caso ou projetos foram popularizados nessa área. Sustentando essa observação, um modelo de Gêmeo Digital foi publicado por Tuegel (2012), a duplicação digital que acompanha toda a vida útil da estrutura de aeronaves é constituída por submodelos de eletrônica, controles de voo, sistema de propulsão e outros subsistemas. Neste exemplo apresentado, a estrutura ultra-realista é explicitamente ligada aos materiais, especificações de fabricação, controles e processos usados não apenas para construir mas também para manter a aeronave. O autor ressalta que apesar de muitas vezes descrito como um único modelo, na verdade o Gêmeo Digital do trabalho é a integração de quatro submodelos: definição estrutural, dinâmica de voo, modelos estruturais e modelos de evolução do estado do material. Tais submodelos operam separadamente compartilhando as informações necessárias entre-si e trazendo um grande grau de integração, nível de fidelidade e quantificação de incerteza em todos os cálculos. Os estados de todos os materiais, quaisquer reparos, modificações e indefinições são monitoradas ao longo do ciclo de vida da aeronave. Da mesma maneira, qualquer resultado ou leitura contida em um submodelo, como tensão, temperatura, campo de deformações por exemplo, é constantemente disponível e acessível para qualquer outro submodelo.

No ano seguinte, Reifsnider e Majumdar (2013) publicaram um trabalho semelhante, no entanto, este trabalho se apresentava com um caráter muito mais científico e voltado para o comportamento de materiais compósitos, em especial as fases de a iniciação de falhas, acúmulo, interação e coalescência em microestruturas. Uma nova metodologia é introduzida, modelada e validada como um método experimental de detecção e interpretação do tipo, forma, densidade e interação dos defeitos na microestrutura à medida em que eles surgem nos materiais compósitos. Os estágios de início, acúmulo de danos e falha do material compósito foram medidos com a aplicação de um campo elétrico no espécime, o qual foi capaz de perceber com facilidade a mudança entre os estágios de dano. Os métodos empregados demonstraram alta sensibilidade à iniciação do dano, marco de definição entre fase de início, acúmulo e notória indicação do desenvolvimento do evento final da fratura. Isso possibilita a criação de parâmetros para alertas de risco, confiabilidade e desempenho do ativo. Essa habilidade é de extrema importância para a operação de um Gêmeo Digital, uma vez que é necessário inspeção dos elementos estruturais de forma frequente.

No contexto automotivo, Cerrone *et al.* (2014) publicou um artigo com enfoque na eficácia da modelagem e simulação de componentes individuais de frotas de veículos com objetivo de motivar e definir melhor um Gêmeo Digital. Apesar de não apresentar grande detalhamento do Gêmeo Digital em sua totalidade, o autor apresenta um caso de modelagem de danos por fratura dúctil para analisar uma peça de geometria não padronizada, mostrando que não basta um modelo constitutivo supostamente bem calibrado e uma boa avaliação de engenharia em análises numéricas, pois muitas vezes apenas pequenos desvios na geometria da peça em estudo na ordem de décimos de milímetros, o que seria presumivelmente insignificante, podem acarretar na mudança do caminho das fissuras. O fato de que nenhum veículo sofre condições de uso e ambiente equivalente ao longo de sua vida útil, faz com que a implementação de Gêmeos Digitais seja responsável por previsões de falhas, antes impossíveis no produto, assim propiciando a melhoria das estimativas de confiabilidade e garantia durante todo o ciclo de vida dos veículos.

Mais um vez no campo de estudo de falhas estruturais em veículos aeroespaciais, o trabalho de Wang *et al.* (2015) propõe uma abordagem para diagnóstico e prognóstico de estruturas de aeronaves danificadas, combinando a mecânica de fadiga de alto desempenho com as teorias de filtragem. É utilizado o Método Alternativo de Elementos Finitos (FEAM - *Finite Element Alternating Method*) para cálculo do Fator de Intensidade da Tensão (SIF - *Stress Intensity Factor*) em análises deter-

minísticas da propagação de fissuras por fadiga de maneira rápida e precisa. Com base nos resultados de tal diagnóstico, em seguida, os prognósticos das estruturas aeroespaciais são alcançados para estimar a distribuição probabilística da vida útil restante. Usando um exemplo simples de uma única fissura perto de um furo de fixação, é demonstrado o conceito e a eficácia da estrutura proposta. Também tendo base no SIF para cálculo de tolerância de dano, Karve *et al.* (2020) apresenta uma metodologia para planejamento de missões usando Gêmeo Digital baseada em 3 componentes para quantificação de incertezas: diagnóstico de dano, prognóstico de dano e otimização da missão. Esse grupo de fatores é utilizado em conjunto para otimização e estudo da propagação da fissura em uma ou várias missões com carregamentos repetitivos que implicam em falhas por fadiga.

Apesar de já serem numerosos os trabalhos relacionados a Gêmeos Digitais, são raros aqueles publicados com primorosa descrição dos detalhes. Um dos mais completos artigos com tópicos de grande clareza na apresentação e organização do trabalho foi publicado por Bole, Powell e Rousseau (2017). Neste trabalho são mostradas várias interfaces de Gêmeos Digitais que foram obtidas por dados monitorados de unidades marítimas em diferentes cenários e dispositivos, desde dispositivos pessoais portáteis até telas de grande formato para apoio à tomada de decisão da equipe. As interfaces de plataformas de Gêmeos Digitais apresentadas em seu trabalho, são predominantemente controladas via navegador de internet, podendo serem executadas em computador ou celular com a exibição da gama de informações disponibilizadas e visualização dos ativos e equipamentos em modelos 2D ou 3D. Os sistemas podem ser acessados por pesquisa de árvore do modelo, ajustados para exibir apenas partes da estrutura relativas a um certo usuário, baseando-se em sua função, departamento ou característica por exemplo. Além disso, todos os itens são ligados por palavras-chave, incluindo documentação, esquemas, desenhos e modelos 3D. O autor relata também que a experiência de usuário é muito importante para a eficiência da interface do Gêmeo Digital. Saber a maneira como os usuários irão pesquisar certas informações por exemplo, pode mudar completamente o funcionamento e benefício do Gêmeo Digital. É vital para o bom desempenho do Gêmeo Digital que a informação esteja disponível e facilmente acessível por diversos meios de procura ou filtros de busca.

O tópico integração dos dados, presente na maioria das seções de artigos publicados que tratam sobre as dificuldades de se implementar um Gêmeo Digital, é mencionado por Bole, Powell e Rousseau (2017). Segundo os autores, criar um Gêmeo Digital

a partir de documentos separados é, notavelmente, a abordagem mais desafiadora. O trabalho manual de identificação de fontes de pontos críticos pode ser necessário para a descrição de ocorrências pontuais e contextualização dos problemas. Aplicando esses conceitos, um Gêmeo Digital eletrônico implementado na FPSO Greater Plutonio, na costa da Angola, foi capaz de gerar (Bole, Powell e Rousseau, 2017):

- 25% de redução de horas de trabalho de engenharia;
- 5% de redução dos custos dos principais equipamentos por meio de documentação simplificada;
- 10% de redução em custos de comissionamento pelo aprimoramento do acesso à documentação;
- 10% de redução de custos em serviço de operação e manutenção pela diminuição de pesquisas locais.

É também relatado por Bole, Powell e Rousseau (2017) a dificuldade de implementar um Gêmeo Digital em um FPSO que já se encontra em operação. A dificuldade de sincronização de um navio em fase operação é cada vez maior após as fases de construção, comissionamento e entrega. É improvável que o uso obrigatório de um sistema sem detalhamento dos procedimentos e que precisa ser incorporado pelo dia a dia das atividades de operação seja bem-sucedido. Durante a fase de operação, os dados não podem ser complementados ao projeto por todos os usuários diariamente. Assim, o processo de integração precisa ser claramente capturado para garantir que a qualidade das informações permaneça alta e que os documentos sejam associados corretamente.

Um outro destaque ao trabalho de Bole, Powell e Rousseau (2017) é o fato de ser um dos poucos a discutir sobre a manutenção, não apenas do ativo, mas do Gêmeo Digital. Suprir o Gêmeo Digital com informações necessárias pode se tornar uma tarefa mais complicada caso haja alterações no FPSO. Apesar de que muitas ferramentas utilizadas para o funcionamento de Gêmeos Digitais admitam com facilidade a alteração de parâmetros, quando concluídas, cada mudança precisa ser incorporada pelo sistema e repassada para cada componente dependente diretamente ou indiretamente do novo conjunto de informações. Os autores relatam que sem a correta manutenção do Gêmeo Digital, alguns projetos pioneiros de Gêmeos Digitais foram encerrados por corte de custos. Portanto, o resultado de um Gêmeo Digital é estritamente ligado ao comprometimento de manutenção da atualização dos dados. Outro ponto levantado, é o de que nem toda embarcação é grande o suficiente para

demandar um sistema complexo e que muitas vezes um sistema de menor complexidade e custo de implementação pode atender às necessidades. Um sistema complexo demais pode se tornar difícil de ser alimentado com dados ou oneroso ao orçamento.

3. Proposta de Gêmeo Digital para FPSO

O MEF tradicional fundamenta-se em 3 etapas tradicionais: pré-processamento, análise e pós-processamento. A etapa tradicional de pré-processamento consiste na definição da geometria do modelo, tipo de análise, propriedades dos materiais, condições de contorno e discretização usual do domínio do problema utilizando elementos finitos. Em seguida, na etapa de análise (ou processamento), as informações definidas na primeira etapa são lidas pelo *solver* do programa utilizado para executar a análise numérica e gerar os resultados. Na última etapa, os resultados são processados e apresentados conforme solicitação do usuário.

No entanto, os programas computacionais baseados no MEF tradicional, por muitas vezes, não são convenientes para atender aos cenários e necessidades apresentados na Seção 2.1. Assim, foi elaborada uma metodologia adaptativa baseada nas etapas tradicionais do MEF para atender a todos os pré-requisitos necessários para um Gêmeo Digital de FPSOs. Algoritmos em MATLAB® (ver Apêndice C) foram desenvolvidos para ajustar as fases de pré-processamento e pós-processamento, inserindo subetapas capazes de receber dados externos de múltiplos sistemas acoplados, atualizar automaticamente um modelo de elementos finitos, realizar análise numérica, gerar e processar resultados para fornecer uma visão geral da integridade estrutural aos operadores e, em seguida, apresentar avaliação de risco com base em uma análise de confiabilidade estrutural (SRA - *Structural Reliability Analysis*) utilizando o DSS para regiões e membros estruturais de interesse.

Devido à grande quantidade de arquivos e diretórios manipulados pelos algoritmos, foi desenvolvida uma GUI com objetivo de fornecer aos usuários uma forma mais simples, compreensível e operacional de interagir com o código implementado para a metodologia adaptativa proposta pelo Gêmeo Digital.

Após o fim da fase de elaboração do modelo de elementos finitos utilizado no Gêmeo Digital (ver Apêndice A), não é necessária manipulação de arquivos ou algoritmos

de forma significante para a execução das análises numéricas, pois a GUI possibilita aos usuários gerir essas informações, configurar dados de entrada de acordo com os sistemas acoplados e resultados a serem verificados pelo DSS.

Uma visão simplificada da metodologia adaptativa desenvolvida para o Gêmeo Digital é apresentada na Figura 3.1 e as técnicas aplicadas no fluxo de trabalho são mais bem detalhadas nesta seção. Cabe ressaltar que os métodos e técnicas aplicadas em cada subetapa da metodologia apresentada podem ser modificados com a finalidade de se adaptar a diferentes problemas e resultados.

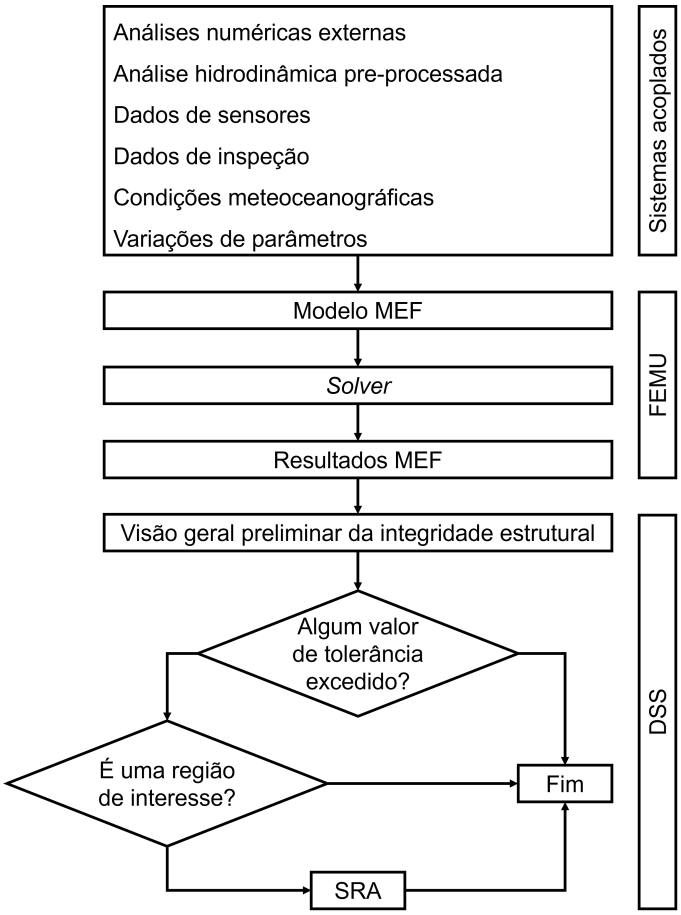


Figura 3.1.: Fluxograma simplificado da metodologia adaptativa em atendimento aos cenários e necessidades do Gêmeo Digital.

Com a organização proposta é possível consolidar o acoplamento de um modelo de elementos finitos capaz de se adaptar aos cenários hipotéticos de cargas (representando circunstâncias de condições críticas verificadas em fase de projeto), às situações reais com informações obtidas da instrumentação de monitoramento e, ou a dados externos de sistemas acoplados (durante a fase de operação). Da mesma

maneira, as subetapas de atualização do modelo de elementos finitos (FEMU - *Finite Element Model Updating*) e DSS, podem ser alterados para melhor atender às solicitações do operador.

3.1. Sistemas acoplados

O desafio inicial para desenvolver um modelo de Gêmeo Digital realista e integrado para uma FPSO está relacionado à coleta de dados por vários sistemas acoplados, levando à interpretação e uso de muitas variáveis em diferentes etapas do ciclo de vida da embarcação. Além disso, alguns dos dados de monitoramento dos sensores não podem ser aplicados diretamente aos modelos de Gêmeos Digitais, pois precisam de um tratamento prévio dos dados. Liu e Ren (2022) propuseram um método de correlação de tensão local, que pode ser usado para obter os dados de tensão estrutural adequados para a avaliação da força específica do modelo de Gêmeo Digital para a estrutura de monitoramento da embarcação a partir dos dados de monitoramento de tensão.

A precisão da resposta final fornecida pelo Gêmeo Digital depende de incertezas originadas de três fontes: parâmetros do modelo (por exemplo, propriedades do material, propriedades da seção, espessura da chapa, dimensões das seções transversais das vigas), modelo estrutural (por exemplo, simplificação da estrutura, condições de contorno, cargas, forma geométrica, comportamento estrutural) e o código do modelo (por exemplo, incertezas numéricas ou incertezas técnicas do modelo). O impacto dessas fontes de incerteza pode ser reduzido usando sistemas acoplados precisos e modelos de alta fidelidade, mas nunca pode ser eliminado (Ereiz, Duvnjak e Fernando Jiménez-Alonso, 2022).

Os sistemas acoplados são responsáveis pela maioria dos parâmetros do modelo estrutural utilizados no procedimento de FEMU do fluxo de trabalho do Gêmeo Digital. Suposições incorretas a partir de dados externos devido à imperfeição do equipamento de medição, falta de dados, ruído de medição, processamento de sinal ou calibração errada podem refletir em problemas de mal condicionamento. Inicialmente, um número limitado de parâmetros de atualização é recomendado para uma melhor compreensão da sensibilidade dos resultados. O grau de confiabilidade e a precisão do modelo numérico tendem a melhorar à medida que o nível de incerteza diminui. Consequentemente, são esperados mais parâmetros de atualização no Gêmeo Digital.

Eventualmente, a baixa qualidade das informações pode afetar a eficiência do Gêmeo Digital, causando alertas com sugestões para manutenções desnecessárias ou interrupções nas operações. Portanto, uma calibração adequada da base de dados fornecida pelos sistemas acoplados é fundamental para um desempenho satisfatório das respostas de probabilidade de falha do casco da embarcação. Atualmente, ferramentas e dispositivos para coletar, transmitir e armazenar dados são facilmente acessíveis para suportar as operações das plataformas *offshore*. No entanto, o tratamento correto da grande quantidade de dados é continuamente necessário durante a vida útil da embarcação para manter o Gêmeo Digital funcionando adequadamente e fornecendo resultados precisos.

FPSOs podem ser submetidas à diferentes condições de acordo com o cenário de operação. Por exemplo, pode-se obter um único conjunto de dados (homogêneo) para reproduzir o comportamento estrutural instantâneo usando dados coletados por sistemas de monitoramento, inspeções ou outros métodos numéricos. Por outro lado, são necessários múltiplos conjuntos de dados (heterogêneos) para representar as condições da estrutura da embarcação na fase de projeto, devem ser criadas condições de cargas hipotéticas devido à grande quantidade de variáveis envolvidas que não podem ser estimadas por processos determinísticos. Algumas práticas recomendadas e regras (ABS, 2022a; DNV, 2014) abordam exemplos de cargas estruturais baseadas em dados estatísticos e modelos matemáticos esperados durante o ciclo de vida da embarcação.

Os sistemas acoplados podem ser definidos no Gêmeo Digital por arquivos de entrada de dados e opções definidas (ver Seção B.2).

3.1.1. Atualização dos dados de ondas

As condições meteoceanográficas são geralmente usadas por análises numéricas em vários programas para calcular dados de série temporal da elevação do mar, aceleração da embarcação, localização do centro de massa, momento de inércia de massa de volume de cada tanque, resposta do *turret* e sistema de amarração. Além disso, a análise hidrodinâmica pode ser pré-calculada (ver Subseção B.2.1) para aplicar a pressão relativa na superfície do casco da embarcação usando um *software* externo (por exemplo, WAMIT®, Ansys Aqwa™). É importante ressaltar que como pré-requisito da metodologia desenvolvida nesse trabalho, a malha utilizada para análise hidrodinâmica do casco deve ser igual a malha do modelo de elementos fini-

tos.

Caso seja feita a opção de atualizar a carga hidrodinâmica do modelo, podem ser informadas séries temporais de elevação da superfície do mar, dados de ondas irregulares ou regulares. Caso seja feita a opção de atualizar a pressões hidrodinâmicas no casco do modelo utilizando dados de uma ou mais ondas regulares, componentes de aceleração da embarcação, componentes de velocidade da embarcação e componentes de força remota aplicado no modelo (ver Subsubseção B.2.1.1). Dados de onda irregular, assim como o de onda regular, podem ser utilizados para atualizar os valores de pressão hidrodinâmica, componentes de aceleração da embarcação, componentes de velocidade da embarcação e componentes de forças remotas aplicado no modelo. No entanto, os dados de estado de mar são utilizados para cálculo de H_s e T_z de acordo com um intervalo de tempo definido (ver Subsubseção B.2.1.2).

Os dados de onda (regular ou irregular) definem o estado de mar para o qual são extraídas as pressões hidrodinâmicas. É considerado o ângulo de fase onde ocorre a maior pressão hidrodinâmica na superfície do casco do modelo de elementos finitos, o ângulo de fase para a menor pressão hidrodinâmica, ou ambos (para maior e menor pressão hidrodinâmica).

São considerados o(s) ângulo(s) de fase onde ocorre(m) a maior e, ou a menor pressão hidrodinâmica na superfície do casco do modelo de elementos finitos.

3.1.2. Atualização dos dados do nível dos tanques

O nível de armazenamento dos tanques de carga é uma variável importante que afeta a distribuição de carga do FPSO. Portanto, é importante avaliar o efeito desse fator na integridade estrutural do ativo. Isso envolve a aplicação de cargas em diferentes níveis de armazenamento dos tanques de carga e a análise dos resultados para determinar a resistência estrutural nessas condições. Esse processo é uma etapa muito comum a ser realizada na fase projeto para verificação da embarcação, pois permite avaliar a capacidade para resistir a diferentes condições de operação. Com dados de sensores e uso do Gêmeo Digital, esse fator pode ser considerado não apenas na fase de projeto, mas também na fase de operação, melhor representando o carregamento dos tanques do FPSO.

Dados de carregamento dos tanques contendo informação da pressão hidrostática externa ao casco, e pressão hidrostática, massa, centro de massa e momento de inércia de cada tanque podem ser considerados no Gêmeo Digital (ver Subseção B.2.2).

3.1.3. Atualização dos dados de inspeção

Inspeções periódicas são realizadas para coletar dados, otimizar as intervenções de reparo e racionalizar os custos adicionais de manutenção. No entanto, na fase de projeto, apenas um modelo de previsão de corrosão pode estimar a perda de espessura esperada para preservar a integridade estrutural da estrutura ao longo dos anos.

Elementos de casca e viga têm como função representar as entidades geométricas usadas no modelo. No entanto, diferentes elementos utilizam uma certa quantidade de parâmetros para sua representação, que por sua vez, também devem ser alterados conforme o processo de corrosão age na estrutura modificando os valores de espessura ao longo dos anos.

De maneira análoga, defeitos de manufatura nos membros estruturais também podem refletir em significantes alterações nas dimensões e espessuras dos elementos estruturais. O que pode ser contabilizado por um modelo *as built*[1], o qual considera as diferenças entre modelo real e digital, reduzindo significativamente o nível de incertezas dos parâmetros de modelo do Gêmeo Digital.

Esses efeitos e seus impactos na estrutura podem ser considerados por meio da atualização de dimensões de chapas e seção transversal de vigas do modelo (ver Subseção 3.1.3).

É importante ressaltar que, após o recebimento das informações da degradação de corrosão real sofrida pela estrutura nos últimos anos e, ou de imperfeições nas dimensões dos membros estruturais, a metodologia proposta permite refazer uma análise retroativa do FPSO .

A atualização do modelo de elementos finitos para consideração da degradação de corrosão real feita por inspeções periódicas é feita com a identificação e relação de cada dado dimensional para cada um dos membros estruturais do FPSO.

3.2. Atualização do modelo de elementos finitos

De acordo com Friswell e Mottershead (1995) and Shahbaznia, Raissi Dehkordi e Mirzaee (2020), o FEMU é definido como um procedimento para atualizar o modelo numérico para melhor reproduzir a resposta medida da estrutura real.

[1]Consiste na verificação do projeto construído e posterior atualização da versão digital do modelo de projeto de acordo com o real.

O modelo de elementos finitos deve ser criado de acordo com os requisitos dos sistemas acoplados (ver Apêndice A). Por exemplo, cargas fictícias devem ser inseridas para representar qualquer condição que deva ser atualizada a partir dos dados dos sistemas acoplados (como a representação das massas dos volumes do conteúdo dos tanques ou quaisquer equipamentos da embarcação). Da mesma forma, se a atualização da espessura de placas e seções transversais de vigas for pretendida, esses membros estruturais devem ser modelados como tipos de elementos que permitam modificações a definição de tais propriedades (por exemplo, elementos de casca e viga).

A única exceção é feita para a carga hidrodinâmica na superfície do casco que, devido à grande quantidade de dados para diferentes combinações de estado de mar, precisa ser pré-calculada para ser aplicada no modelo de elementos finitos. Porém, para aplicação dessa carga, a metodologia não necessita de uma carga fictícia para modificação no modelo de elementos finitos.

Após a elaboração do modelo de elementos finitos criado com todas as condições de contorno e cargas necessárias para representar as informações monitoradas ou calculadas que são definidas pelos sistemas acoplados, inicia-se a etapa de atualização do modelo de elementos finitos e execução dos modelos (ver Seção B.3).

Um ou mais modelos únicos são criados de acordo com a opção feita para o conjunto de dados empregados (*data set*), definido nos sistemas acoplados (ver Seção B.2). Pode ser definido pelo *data set* que a variação dos dados de entrada ocorrerá pelos dados de onda, dados de nível dos tanques ou dados de inspeção. Dessa maneira, é possível a criação de estudos de caso em que um tipo de entrada de dados varia enquanto outros permanecem constantes, permitindo melhor avaliação dos dados que são variados. Essa forma de variar os dados é mais conveniente para a atuação do Gêmeo Digital em um cenário de fase de projeto. Por outro lado, se todos os tipos de entrada de dados variam, a definição do *data set* é indiferente, uma vez que todos os dados serão atualizados, criando um caso único de simulação. Essa segunda forma de atuação é mais adequada para a fase de operação.

Cada vez que um novo conjunto de dados é enviado por qualquer uma das fontes dos sistemas acoplados, o modelo de elementos finitos é utilizado para substituir os valores fictícios, montando e enviando uma ou mais análises numéricas para o *solver* de elementos finitos. Análises quase-estáticas são realizadas e o modelo atual pode ser utilizado com diferentes cargas cíclicas para análise de fadiga em estudos subsequentes. Como mencionado anteriormente, o fluxo de trabalho da metodologia

desenvolvida é adaptativo e o tipo de análise pode ser modificado para avaliar resultados específicos. Depois de gerar resultados para cada caso, este procedimento pode ser definido como uma função entrada-saída, representado matematicamente por Ereiz, Duvnjak e Fernando Jiménez-Alonso (2022) como:

$$z = M(\theta) \tag{3.1}$$

onde z são os resultados de saída (por exemplo, deslocamentos, tensões, deformações), M é o operador de modelo que descreve o comportamento de entrada-saída e θ são os parâmetros do modelo estrutural que representam uma classe de modelos $\mathcal{M}_m$ e variam em um subconjunto P_m. Portanto, o modelo na classe de modelo estrutural pode ser expresso como:

$$\mathcal{M}_m = \{\mathcal{M}_m(\theta) \mid \theta \in P_m\}. \tag{3.2}$$

O espaço de resposta de saída do modelo P_o está relacionado ao espaço de parâmetros do modelo P_m. Em caso de incertezas na modelagem, essas variáveis devem ser estimadas para levar em conta conjuntos de dados experimentais, definidos como:

$$\tilde{M} = M(\theta) + \varepsilon + \mu \tag{3.3}$$

onde $\tilde{M}$ são os conjuntos de dados experimentais, ε é o vetor de incerteza do modelo e μ é o vetor de incerteza de medição.

3.3. Sistema de apoio à decisão (DSS)

A fase de pós-processamento inicia com a verificação dos resultados gerados para cada caso, a fim de analisá-los e torná-los disponíveis através de um DSS. O objetivo principal desta ferramenta é fornecer uma interação intuitiva entre os operadores e o ativo físico com uma visão geral dos níveis de risco para regiões ou componentes estruturais.

Uma preparação adequada do DSS é essencial para exibir alertas de forma rápida para a tomada de decisão e evitar falhas na estrutura. Uma calibração incorreta do DSS pode resultar em resultados imprecisos e suspensões recorrentes das operações.

Na execução prática do Gêmeo Digital, grandes quantidades de dados podem ser produzidas pelo processo de FEMU. Como parte do Gêmeo Digital, o tratamento da informação é crucial para evitar processos computacionalmente pesados e onerosos. Além disso, para melhorar as capacidades de tomada de decisão e obter um fluxo de trabalho bem-sucedido e eficiente, são necessários resultados completos e organizados para interpretação e recomendação imediata de qualquer ação que deva ser tomada, se necessária.

O ativo completo ou apenas parte do ativo pode ser verificado pelo DSS (ver Seção B.4), permitindo se concentrar em uma região ou tanque de carga mais relevante para avaliar o risco dos componentes estruturais de interesse. Este procedimento é realizado com base em um processo de duas fases. Na primeira fase, o objetivo principal da ferramenta é identificar regiões ou elementos estruturais que são mais suscetíveis a falha. Um ou mais valores de tolerância predefinidos para resultados de saída (por exemplo, deslocamentos, tensões, deformações) são usados para realizar uma avaliação preliminar, detectando regiões que excedem esses valores e criando níveis de alerta. Em seguida, eles são categorizados de acordo com o resultado de saída e nível de risco para serem destacados e visualizados pelos operadores do FPSO. O processo simplificado da primeira fase muitas vezes não é capaz de proporcionar uma decisão razoável. Apesar de sua simplicidade, a aplicação desta fase é muito importante para reduzir a quantidade de dados e permitir que métodos mais complexos de confiabilidade estrutural sejam aplicados com consumo de tempo e memória minimizados para os componentes estruturais de interesse. A segunda fase é executada somente se um resultado de alerta durante a primeira fase. Os componentes identificados na primeira fase são usados pelo SRA para investigar melhor essas regiões com base nos dados de resistência do material, considerando as incertezas estatísticas. Esta fase (discutida no próximo capítulo) representa uma análise de confiabilidade mais alta do que o processo realizado na primeira fase do DSS.

4. Aplicações

Um modelo global que seja realista e integrado a vários subsistemas, interpretando muitas variáveis de forma coerente para as diferentes fases do ciclo de vida do ativo é um dos maiores desafios. É evidente que um modelo de alta fidelidade que considera múltiplos subsistemas aumenta a complexidade para interpretar os efeitos e o comportamento do modelo. Por outro lado, um modelo muito simplificado pode levar à negligência de detalhes importantes que devem ser considerados nas análises.

Os Gêmeos Digitais são ferramentas poderosas que combinam modelos digitais precisos com dados reais em tempo real para fornecer uma visão mais clara e precisa da situação atual de um sistema. No contexto de FPSOs, o uso de Gêmeos Digitais pode ser útil em vários aspectos, incluindo:

- Planejamento de manutenção: O Gêmeo Digital pode ser usado para simular diferentes cenários de manutenção e avaliar o impacto dessas operações na produção de petróleo e gás, além de fornecer informações para planejar as atividades de manutenção do FPSO de maneira mais eficiente e segura;
- Otimização da produção: O Gêmeo Digital pode ser usado para simular diferentes cenários de produção e avaliar o impacto dessas operações na produção de petróleo e gás, permitindo otimizar a produção e maximizar a eficiência da FPSO;
- Análise de segurança: O Gêmeo Digital pode ser usado para simular cenários de segurança e avaliar o impacto de possíveis falhas ou desastres no FPSO, permitindo identificar pontos fracos e tomar medidas para melhorar a segurança da plataforma;
- Treinamento de equipes: O Gêmeo Digital pode ser usado como ferramenta de treinamento para as equipes de operação e manutenção do FPSO, permitindo que os funcionários pratiquem diferentes cenários e situações sem correr riscos na vida real;

- Análise de desempenho: O Gêmeo Digital pode ser usado para monitorar e analisar o desempenho do FPSO ao longo do tempo, identificando pontos fracos e fornecendo informações para melhorar a eficiência e a operação da plataforma.

Em resumo, o Gêmeo Digital é uma ferramenta valiosa para o setor de petróleo e gás, permitindo simular, analisar e otimizar o desempenho da FPSO antes ou durante sua implementação no campo de exploração e extração resultando em uma operação mais segura, eficiente e rentável da plataforma.

A análise do modelo global em elementos finitos tem como principal propósito determinar a resposta global da estrutura. Nesse âmbito, é possível encontrar algumas normas com algumas orientações para a construção de modelos de embarcações em elementos finitos. É importante ressaltar que a metodologia desenvolvida nesse trabalho pode utilizar qualquer modelo criado com o MEF tradicional, no entanto, devido às particularidades de um modelo de para FPSOs que requer alta fidelidade, recomendações foram seguidas de maneira mais rigorosa para criação dos modelos FE. O Apêndice A apresenta de forma mais detalhada essas diretrizes, detalhes dos elementos da estrutura do casco e o modelo criado.

4.1. Estudo de caso 1

O primeiro estudo de caso apresenta um modelo de FPSO de alta fidelidade em cenário de operação com consideração de diferentes séries temporais produzidas por sistemas acoplados alimentados por condições meteoceanográficas. Informações da superfície livre (elevação do mar), aceleração da embarcação, tração no topo das linhas de amarração, nível de carregamento dos tanques e carga hidrodinâmica no casco são transferidas ao modelo de elementos finitos do casco para análise e possível identificação de estados de alerta.

Neste estudo de caso, o DSS é utilizado apenas para identificar membros estruturais que atingem níveis de alerta pré-definidos, sem qualquer investigação mais complexa feita pelo SRA.

4.1.1. Sistemas acoplados

Condições meteoceanográficas foram utilizadas por análises numéricas do *software* EDTools® (Malta, 2010) e Orcaflex® para calcular dados de 3 séries temporais de 20 minutos cada, fornecendo dados de elevação do mar (ver Figura 4.1) e aceleração da embarcação. Além disso, um sistema de linhas de amarração composto por nove linhas foi considerado. A carga de tensão de topo nessas linhas de amarração também é incluída pelos dados de séries temporais recebidos do Orcaflex®, sendo simplificada em uma força remota para transferir os efeitos para a estrutura do modelo.

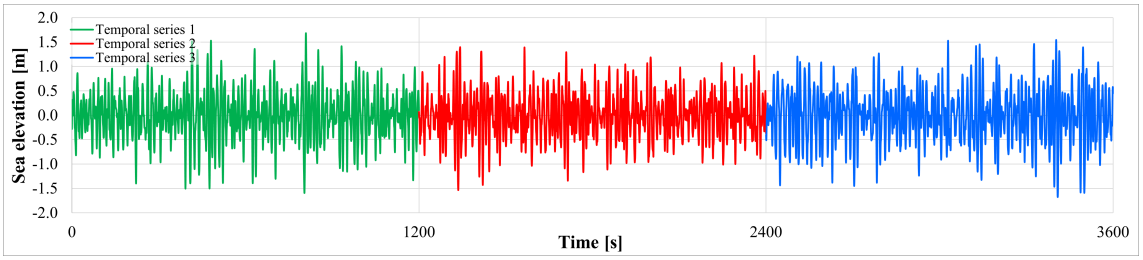


Figura 4.1.: Série temporal de elevação do mar considerada pelos sistemas acoplados.

Análises hidrodinâmicas foram previamente processadas utilizando o *software* WAMIT® (WAMIT, 2018) para fornecer a distribuição da pressão hidrodinâmica em elementos do casco do modelo utilizando 20.552 painéis planos[1].

Os problemas de radiação e difração foram resolvidos para períodos de onda na faixa de 4,5 segundos a 34 segundos (com discretização de 0,5 segundo) e direção de onda incidente na faixa de 0° a 180° (com discretização de 10°). Assim, qualquer uma das 1.140 soluções pré-calculadas pode ser usada para aplicação da carga hidrodinâmica para os nós da malha do modelo de elementos finitos. A Figura 4.2 mostra um exemplo de solução da análise de pressão hidrodinâmica para o casco do FPSO.

A altura de onda significativa (H_s) e os períodos entre zero-ascendentes são automaticamente calculados a partir dos dados de elevação do mar fornecidas ao Gêmeo Digital, sendo considerados para aplicação da pressão hidrodinâmica. Pode ser utilizado pelo Gêmeo Digital, o instante onde o ângulo de fase retorna o valor máximo e, ou mínimo de pressão na superfície do casco. Para outros valores, como acelerações e forças da embarcação, por exemplo, o vetor máximo de resultante da série

[1]Elemento utilizado para discretização da superfície do casco e subsequente formulação da análise hidrodinâmica no *software* WAMIT®.

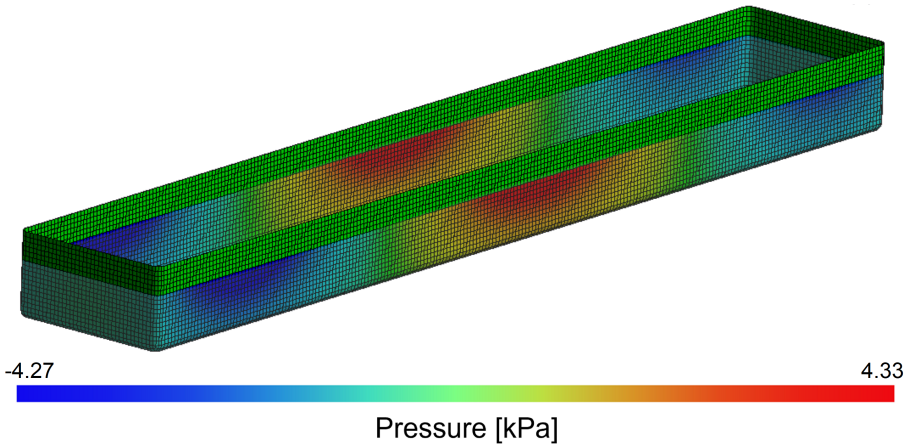


Figura 4.2.: Exemplo de pressões hidrodinâmicas pré-calculadas pelo *software* WAMIT® para a malha do modelo de elementos finitos.

temporal é utilizado de maneira a avaliar a situação de pior caso de carregamento da embarcação.

Os níveis de armazenamento dos tanques foram definidos como constantes em 70%. Além disso, foram consideradas a pressão hidrostática e o peso próprio da estrutura na análise estrutural do estudo de caso.

4.1.2. Atualização do modelo de elementos finitos

O modelo criado para o primeiro estudo de caso é composto por 135.424 nós em 136.117 elementos de casca e viga. A Tabela 4.1 e Tabela 4.2 apresentam os dados da embarcação e do modelo de elementos finitos (representando o modelo real). A Tabela 4.3 apresenta as propriedades mecânicas do material utilizado no modelo. A Figura 4.3 apresenta a vista isométrica do plano de seção longitudinal do modelo de elementos finitos (maiores detalhes do modelo são apresentados no Apêndice D).

Tabela 4.1.: Características da embarcação.

Definição	Dimensão (m)
Comprimento total (LOA[2])	215
Boca[3]	32
Pontal[4]	16

[2] *Length Overall* é comprimento total da embarcação.
[3] É a largura máxima da embarcação.
[4] É a distância vertical medida do convés até um plano horizontal que passa pela quilha da embarcação. É a soma da borda livre e do calado da embarcação.

Tabela 4.2.: Partes do modelo de elementos finitos e dimensões.

Parte do modelo	Dimensão (m)
Extensão de cavernas (ext. inicial)	10
Tanque de carga 1	25
Tanque de carga 2	25
Tanque de carga 3	25
Extensão de cavernas (ext. final)	10
modelo de elementos finitos completo	95

Tabela 4.3.: Propriedades mecânicas do material (estudo de caso 1).

Definição	Unidade	Valor
Módulo de elasticidade, E	GPa	200
Coeficiente de Poisson, ν	-	0,3
Densidade, ρ	kg/m^3	7.850

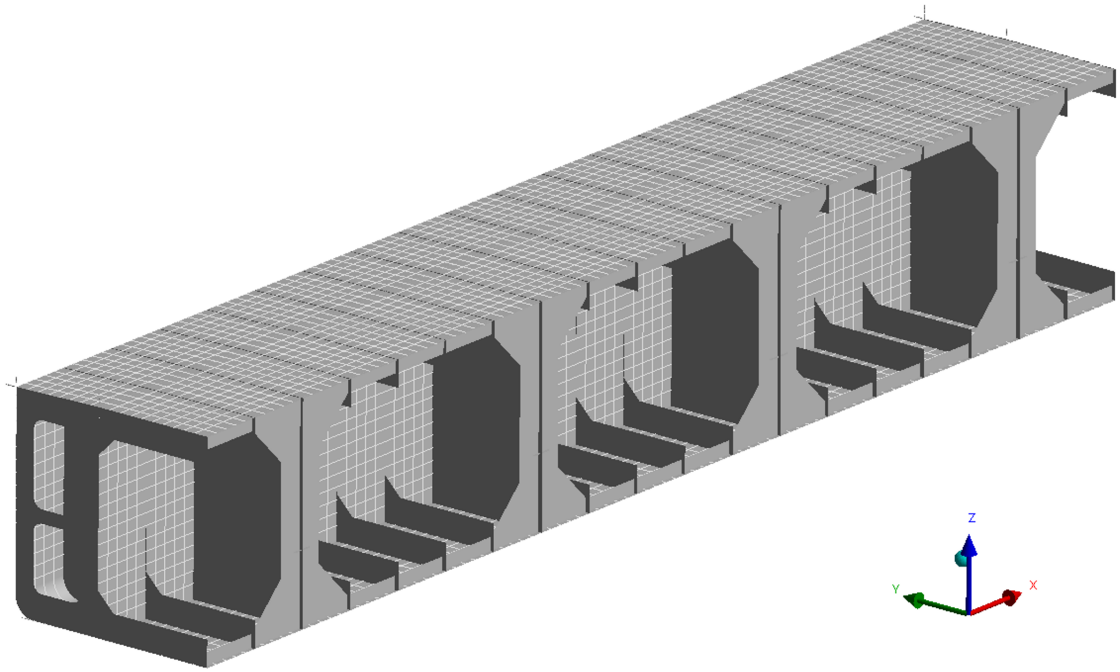


Figura 4.3.: Vista isométrica do plano de seção longitudinal do modelo de elementos finitos.

Para representar os dados recebidos dos sistemas acoplados, foram criadas cargas com valores fictícios para pressão hidrostática, centro de gravidade e momento de inércia de massa de volume de cada tanque; centro de gravidade, aceleração linear

e angular da embarcação. Também foi utilizada uma força remota para transferir para à estrutura do modelo os efeitos da carga de tração no topo das linhas de amarração.

4.1.3. Sistema de apoio à decisão (DSS)

Os valores de tolerância para o DSS foram criados baseados nos níveis nos valores de limite da tensão de escoamento e tensão última à tração do aço de acordo com a norma ABS (2022a) multiplicados por um coeficiente de segurança (conforme mostrado na Tabela 4.4).

Tabela 4.4.: Níveis de alerta e valores de tolerância criados para o DSS.

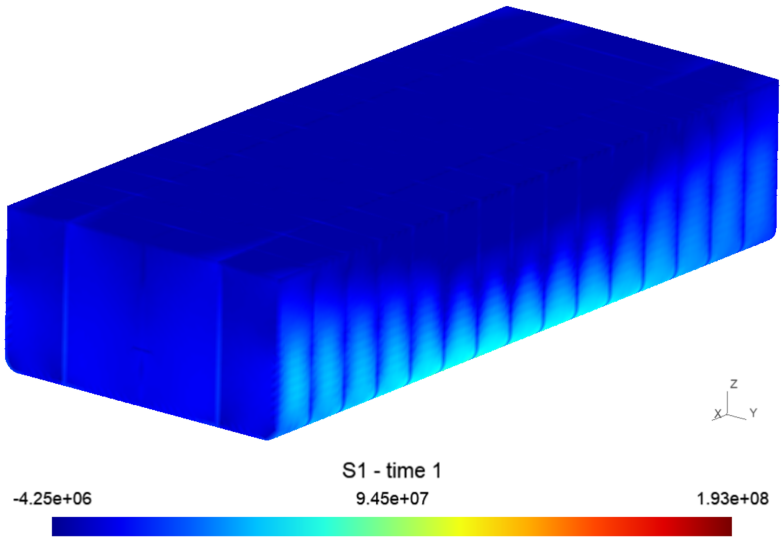
Propriedade	Valor [MPa]	Coeficiente	Nível de alerta	Limite
Tensão de escoamento (σ_y)	235	0,70	1	165
Tensão limite (σ_u)	400	0,45	2	180

Uma simulação numérica foi realizada após recebimento de cada uma das 3 séries temporais apresentadas em Figura 4.1. Os valores calculados de altura de onda significativa (H_s) e períodos entre zero-ascendentes a partir da elevação do mar são apresentados na Tabela 4.5, com tempo médio de computação de 2 minutos e 50 segundos usando 4 núcleos em um computador que possui um processador Intel® Xeon ® E5-1650 v3 @ 3,50 GHz, 64 Gb de RAM, sistema operacional Windows 10, versão 64 bits.

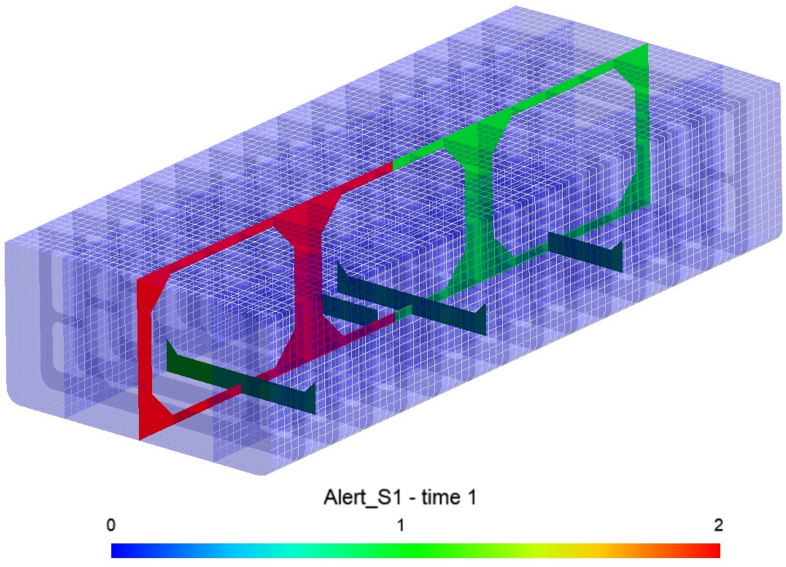
Tabela 4.5.: Altura significativa da onda e período de cruzamento zero calculado pelo Gêmeo Digital para cada série temporal.

Série temporal	1	2	3
H_s [m]	2,0	1,8	2,1
T_i [s]	9,7	9,9	10,9

O Gêmeo Digital permite a verificação de resultados comumente solicitados em análises de elementos finitos, por exemplo: tensão principal máxima (ver Figura 4.4(a)). Os níveis de alerta (Tabela 4.4) foram atingidos apenas na primeira série temporal, o DSS identificou 6 membros estruturais, um deles acima do nível de alerta 2 (exibido em vermelho na Figura 4.4(b)) e os outros acima do nível de alerta 1 (exibido em verde na Figura 4.4(b)). Além da visualização dos membros estruturais que atingiram qualquer nível de alerta, estes membros são salvos em um relatório (ver Figura 4.4(c)).



(a) Tensão principal máxima gerada no *software* Gmsh.



(b) Níveis de alerta para membros estruturais gerados no *software* Gmsh.

```
ALERT ID: 1
RESULT: Principal_Stress
COMPONENT: 1
ALERT LEVEL: 1
BODIES:
Quilha Surface Body.2
Cav Surface Body B.12
Cav Surface Body B.13
Cav Surface Body C.12
Cav Surface Body C.13

ALERT ID: 1
RESULT: Principal_Stress
COMPONENT: 1
ALERT LEVEL: 2
BODIES:
Quilha Surface Body.1
```

(c) Arquivo de texto informando membros estruturais.

Figura 4.4.: Resultados do DSS para a série temporal 1.

4.2. Estudo de caso 2

O segundo estudo de caso apresenta um modelo de casco de alta fidelidade desenvolvido para ser atualizado ao longo dos anos com desgaste por corrosão para cada membro estrutural, fornecendo índices de confiabilidade para regiões de interesse. Além disso, o movimento da embarcação, as condições meteoceanográficas e de armazenamento foram consideradas por análises numéricas de sistemas acoplados.

Um modelo de previsão de corrosão foi utilizado para criar diferentes hipóteses de degradação para chapas e reforçadores do fundo do casco (onde problemas de corrosão são mais comuns em FPSOs), enquanto dados fornecidos por sistemas acoplados são considerados para investigar o efeito da deterioração.

Após a verificação de resultados feita pelo DSS, componentes estruturais que atingiram qualquer nível de alerta são submetidos a um método de confiabilidade estrutural mais complexo para fornecer a probabilidade de falha destes considerando a distribuição estatística da resistência do material.

4.2.1. Sistemas acoplados

Condições meteoceanográficas foram utilizadas por análises numéricas do *software* EDTools® (Malta, 2010) e Orcaflex® as condições de equilíbrio e aceleração da embarcação. Além disso, um sistema de linhas de amarração composto por nove linhas foi considerado. A carga de tração de topo nessas linhas de amarração também é incluída pelos dados recebidos do Orcaflex® sendo simplificada em uma força remota para transferir os efeitos para a estrutura do modelo.

Análises hidrodinâmicas foram previamente processadas utilizando o *software* WAMIT® (WAMIT, 2018) para fornecer a distribuição da pressão hidrodinâmica em nós do casco do modelo utilizando 18.592 painéis planos (relativos apenas à região da embarcação representada no modelo de elementos finitos).

Os problemas de radiação e difração foram resolvidos para períodos de onda na faixa de 3,0 segundos a 17 segundos (com discretização de 0,5 segundo) e direção de onda incidente na faixa de 0° a 180° (com discretização de 10°). Assim, qualquer uma das 551 soluções pré-calculadas pode ser usada para aplicação da carga hidrodinâmica para os nós da malha do modelo de elementos finitos. A Figura 4.5 mostra um exemplo de solução da análise de pressão hidrodinâmica para o casco do FPSO.

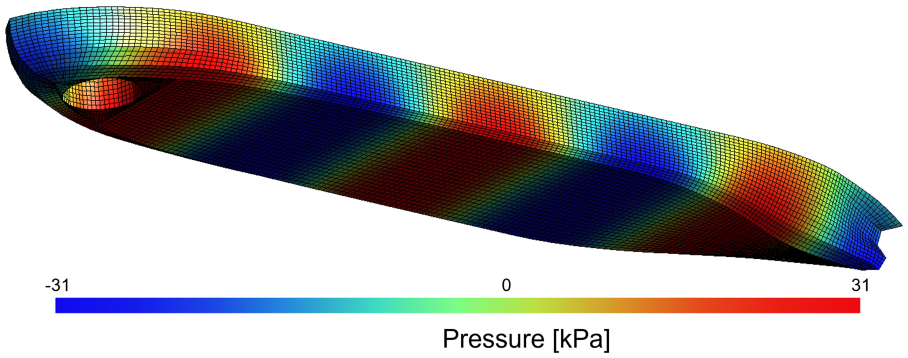


Figura 4.5.: Exemplo de pressões hidrodinâmicas pré-calculadas pelo *software* WAMIT® para a malha do modelo de elementos finitos.

4.2.1.1. Modelo não linear de previsão de corrosão

Como previamente mencionado, neste estudo de caso foi utilizado um modelo de corrosão para previsão. Muitas formulações foram propostas na literatura para prever a corrosão em embarcações (Southwell, Bultman e Hummer, 1979; Melchers, 1995; Soares e Garbatov, 1999). No entanto, a maioria desses modelos é calibrada com base em dados empíricos ou desenvolvida para representar um caso particular relacionado à uma região ou tipo de corrosão. No caso do comportamento corrosivo observado em embarcações, as estruturas são geralmente protegidas por revestimentos, pinturas ou sistemas catódicos que preservam as superfícies adiando o início do processo de corrosão. Com o passar dos anos, o processo de corrosão não linear passa a produzir material oxidado na superfície dos membros estruturais criando uma proteção natural que leva à estabilização da perda de espessura. Este comportamento é representado na Figura 4.6 e o modelo de previsão de corrosão não linear apresentado por Soares e Garbatov (1999) é descrito pela solução da equação diferencial de primeira ordem:

$$d_\infty \, \dot{d}(t) + d(t) = d_\infty \tag{4.1}$$

onde d_∞ é a perda de espessura pela corrosão a longo prazo, $d(t)$ é a perda de espessura no tempo t e $\dot{d}(t)$ é a taxa de corrosão.

A solução da Equação 4.1 leva à forma geral apresentada por:

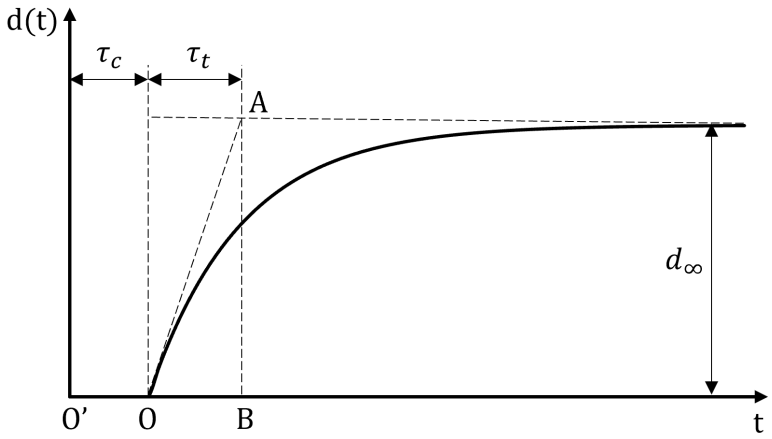


Figura 4.6.: Perda de espessura em função do tempo (adaptado de Soares e Garbatov (1999)).

$$d(t) = \begin{cases} 0, & t \leq \tau_c \\ \mathrm{d}_\infty \left[1 - exp\left(-\frac{t-\tau_c}{\tau_t}\right)\right], & t > \tau_c \end{cases} \tag{4.2}$$

onde τ_c é a vida útil do sistema de proteção anticorrosivo (período inicial sem corrosão devido aos sistemas anticorrosivos) e τ_t é o tempo de transição calculado como:

$$\tau_t = \frac{d_\infty}{tan(\alpha)} \tag{4.3}$$

onde α é o ângulo definido pelos segmentos OA e OB na Figura 4.6.

As três etapas do modelo de degradação para a corrosão dependente do tempo são ilustradas na Figura 4.6. O primeiro deles é o período inicial sem redução de espessura com intervalo de tempo igual a τ_c determinado pelo sistema de proteção das superfícies da embarcação. O segundo estágio, identificado experimentalmente (Farias e Netto, 2012), é caracterizado pelo processo de corrosão com taxa exponencial controlada pelo parâmetro α. Na última etapa, o próprio material produzido pela corrosão protege a superfície dos membros estruturais e a perda de espessura tende a atingir o valor máximo definido por d_∞ ao longo do tempo.

Variando os parâmetros da Equação 4.2, o modelo de corrosão proposto pode ser ajustado para diferentes casos. Além disso, casos hipotéticos podem ser criados

para prever a corrosão de espessuras e posteriormente calibrados usando dados de inspeção para ajustar as informações disponíveis à curva de corrosão não linear da equação. Essa estratégia foi adotada e apresentada por Farias e Netto (2012).

4.2.1.2. Hipóteses de evolução da corrosão

A degradação estrutural por corrosão em FPSOs depende de muitos fatores como salinidade, temperatura, nível de oxigênio, nível de pH e composição química da água que pode ser diferente de acordo com a localização e profundidade de extração do campo de petróleo. Geralmente, parte da água de formações geológicas com características corrosivas severas é removida do petróleo extraído e então se aloja no fundo dos tanques levando a um dos principais problemas de confiabilidade relacionados a danos por corrosão em navios-tanque (Farias e Netto, 2012). Da mesma forma, segundo Emi *et al.* (1994) a inibição do processo de corrosão pela duração do sistema de proteção pode variar, usualmente se iniciando em um discretização de 1,5 a 5,5 anos. A média desses valores é assumida para esse estudo caso, assim adotando a duração do sistema de proteção de 3,5 anos e uma vida útil de projeto de 30 anos para as quatro hipóteses de evolução da corrosão tratadas nesse estudo de caso.

A primeira hipótese considerou o desgaste por corrosão com base no NDCV e aplicado a todos os membros estruturais após o fim do tempo de proteção contra corrosão. Para esta hipóteses, o desgaste por corrosão pode ser definido pela equação:

$$d(t) = \begin{cases} 0, & t \leq \tau_c \\ \mathrm{d}_\infty \left(\frac{t-\tau_c}{t_d}\right), & t > \tau_c \end{cases} \tag{4.4}$$

onde τ_c é a vida útil da proteção contra corrosão, a corrosão de longo prazo d_∞ é baseada no desgaste total da corrosão do membro estrutural definido pelo NDCV e t_d é a vida útil do projeto estrutural.

Outras três hipóteses foram formuladas adotando a perda de espessura para os membros estruturais do fundo dos tanques. Nas hipóteses propostas, a corrosão de longa duração é dada por:

$$d_{\infty} = d_0 d_{was\%} \tag{4.5}$$

onde d_0 é a espessura inicial e $d_{was\%}$ é a porcentagem permitida de desgaste por corrosão, a qual tem valores limites recomendados de acordo com o membro estrutural analisado usando aço comum e de alta resistência para embarcações (ABS, 2022b).

Substituindo a Equação 4.5 na Equação 4.2 e Equação 4.3, a taxa de corrosão pode ser definida como:

$$d(t) = \begin{cases} 0, & t \leq \tau_c \\ d_0 d_{was\%} \left[1 - exp\left(-\frac{(t-\tau_c)tg(\alpha)}{d_0 d_{was\%}}\right)\right], & t > \tau_c \end{cases} . \tag{4.6}$$

Neste estudo de caso, para as hipóteses 2, 3 e 4, foram assumidos, respectivamente, valores de 15%, 20% e 25% para $d_{was\%}$, representando diferentes problemas para as chapas do fundo do FPSO, as quais que poderiam sofrer danos ao longo dos anos. Os parâmetros estabelecidos para cada hipótese são apresentados na Tabela 4.6. Uma representação genérica da perda de espessura dos membros estruturais para cada caso hipotético com base no $d_{was\%}$ é apresentada na Figura 4.7.

Tabela 4.6.: Parâmetros estabelecidos para cada hipótese de evolução da corrosão

Hipótese	Modelo de corrosão	$d_{was\%}$	Região afetada
1	ABS (2022a)	NDCV	Todas
2	Soares e Garbatov (1999)	15	Fundo
3	Soares e Garbatov (1999)	20	Fundo
4	Soares e Garbatov (1999)	25	Fundo

Valores de entrada com variações ao longo do tempo poderiam ser utilizados para atualizar as condições ao longo da vida útil do FPSO.

No entanto, cargas constantes e condições meteoceanográficas foram assumidas para todos os casos hipotéticos pelos sistemas acoplados neste estudo de caso para melhor investigar os efeitos da perda de espessura no fundo da estrutura. A direção da onda incidente foi assumida como 0° uma vez que a embarcação tende a se estabilizar em zero graus devido à presença do *turret* e vento (Zanganeh e Thiagarajan, 2018). A altura de onda significativa foi assumida com amplitude de 3 metros e período

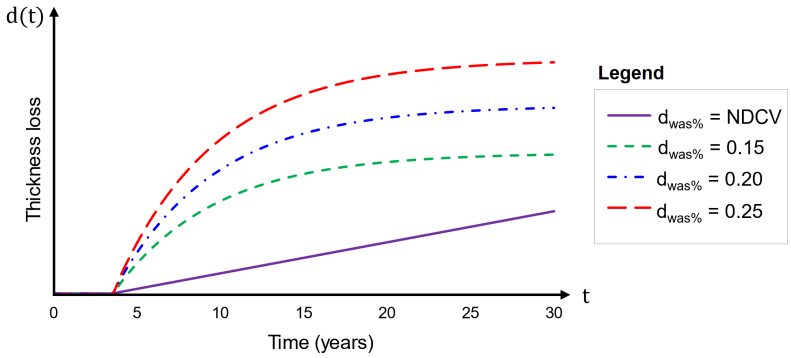


Figura 4.7.: Representação genérica da perda de espessura dos membros estruturais de acordo com cada hipótese proposta.

de zeros-ascendentes de 10 segundos. Os tanques de carga foram considerados pelos sistemas acoplados como parcialmente carregados com 70% da capacidade de carga total.

4.2.2. Atualização do modelo de elementos finitos

O modelo criado para o segundo estudo de caso é composto por 1.037.338 nós distribuídos em 821.411 elementos de casca e 133.571 elementos de viga, permitindo a atualização na espessura e seção transversal de qualquer membro estrutural. A Tabela 4.7 e Tabela 4.8 apresentam os dados da embarcação e do modelo de elementos finitos (representando o modelo real). A Figura 4.8 apresenta a vista isométrica do plano de seção longitudinal do modelo de elementos finitos com as espessuras dos elementos de casca. Todas as dimensões e espessuras das chapas e seções transversais das vigas foram modeladas de acordo com os dados reais de um FPSO, permitindo a realização de técnicas de reconstrução geométrica para produzir um modelo *as built*, obtendo informações de uma embarcação digitalizada realista (maiores detalhes do modelo são apresentados no Apêndice D).

Tabela 4.7.: Características da embarcação.

Definição	Dimensão (m)
Comprimento total (LOA)	322
Boca	56
Pontal	29

Tabela 4.8.: Partes do modelo de elementos finitos e dimensões.

Parte do modelo	Dimensão (m)
Extensão de cavernas (ext. inicial)	10,90
Tanque de carga 1	32,70
Tanque de carga 2	49,00
Tanque de carga 3	32,70
Extensão de cavernas (ext. final)	10,90
modelo de elementos finitos completo	136,20

Para reduzir o tempo de cálculo, a análise foi definida como quase estática realizada no *software* comercial Ansys Mechanical® (ANSYS, 2020) aplicando um modelo elástico linear de aço estrutural com propriedades e unidades do material apresentados na Tabela 4.9.

Tabela 4.9.: Propriedades mecânicas do material (estudo de caso 2).

Definição	Unidade	Valor
Módulo de elasticidade, E	GPa	200
Coeficiente de Poisson, ν	-	0,3
Densidade, ρ	kg/m^3	7,850

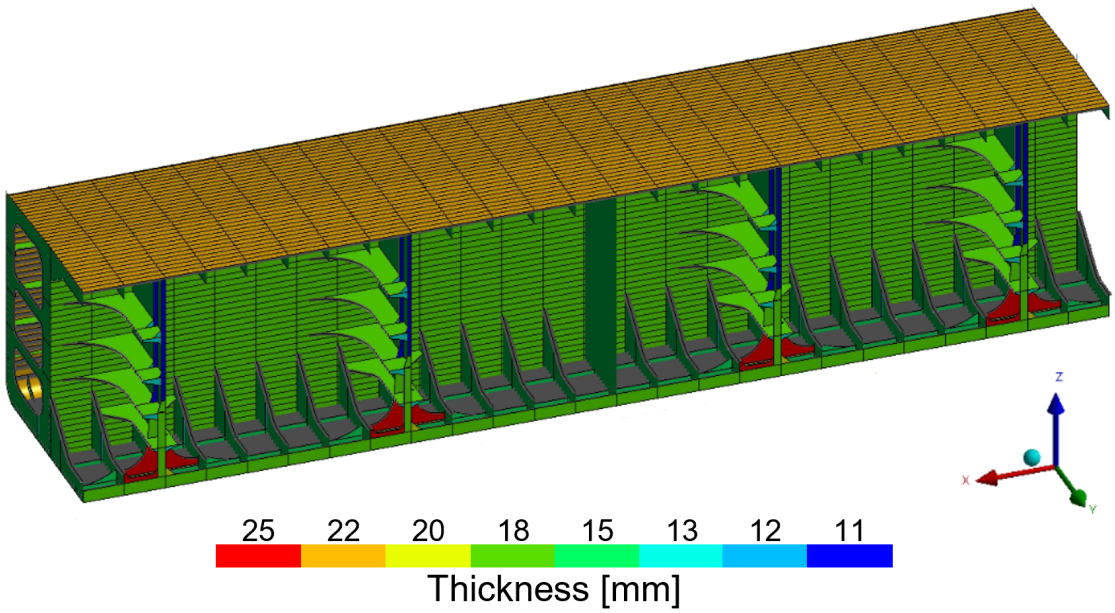


Figura 4.8.: Vista isométrica do plano de seção longitudinal do modelo de elementos finitos com informação da espessura.

Para representar os dados recebidos dos sistemas acoplados, foram criadas cargas

com valores fictícios para pressão hidrostática, centro de gravidade e momento de inércia de massa de volume de cada tanque; centro de gravidade, aceleração linear e angular da embarcação. Também foi utilizada uma força remota para transferir para à estrutura do modelo os efeitos da carga de tração no topo das linhas de amarração.

4.2.3. Sistema de apoio à decisão (DSS)

Os resultados para análise de avaliação de risco foram restritos apenas ao tanque 2 (central), onde resultados são mais precisos (devido aos efeitos das condições de contorno) e também são esperadas as maiores tensões.

O limite de escoamento à tensão do aço foi utilizado como valor de tolerância para verificar e criar o primeiro nível de alerta na primeira etapa do processo DSS. Normalmente, a variabilidade da resistência do material usada no SRA é baseada na distribuição estatística da resistência ao escoamento. Para o segundo nível de alerta, o limite último de resistência à tração foi usado como valor de tolerância. Este pode ser uma propriedade útil para fornecer avaliação de risco ao longo dos anos na visão geral do Gêmeo Digital. Além disso, os resultados experimentais para o limite de resistência à tração são menos sensíveis à taxa de deformação que os resultados do limite de escoamento (VanDerHorn e Wang, 2011), reduzindo o nível de incerteza do DSS.

Os resultados experimentais para a variabilidade do material podem ser significativamente afetados devido à falta de uniformidade nos métodos de teste. Assim, VanDerHorn e Wang (2011) apresentaram um extenso estudo estatístico considerando estudos anteriores da literatura e dados coletados de cinco diferentes fabricantes de aço para construção naval dos Estados Unidos e da Ásia.

As informações estatísticas e a distribuição para o limite de escoamento e limite último para aços de construção naval selecionados para análise DSS em todas as hipóteses deste estudo de caso são apresentadas na Tabela 4.10 e Tabela 4.11, e na Figura 4.9.

Na primeira etapa do DSS, o limite de escoamento e o limite último foram usados como valores de tolerância para os níveis de alerta 1 e 2 (Tabela 4.10 e Tabela 4.11), respectivamente. A Figura 4.10 mostra o corte transversal do modelo com os níveis de alerta previstos para o último ano do ciclo de vida do FPSO em cada hipótese (Tabela 4.6). O nível de alerta 2 (Tabela 4.11) não foi identificado em nenhuma das

Tabela 4.10.: Estatísticas para limite de escoamento do aço de construção naval (VanDerHorn e Wang, 2011).

Definição	Unidade	Valor
Limite de escoamento, σ_y	MPa	355,0
Valor médio, μ_R	MPa	421,7
Desvio padrão, σ_R	MPa	16,4

Tabela 4.11.: Estatísticas para limite último do aço de construção naval (VanDerHorn e Wang, 2011).

Definição	Unidade	Valor
Limite último, σ_u	MPa	490,0
Valor médio, μ_R	MPa	536,1
Desvio padrão, σ_R	MPa	15,7

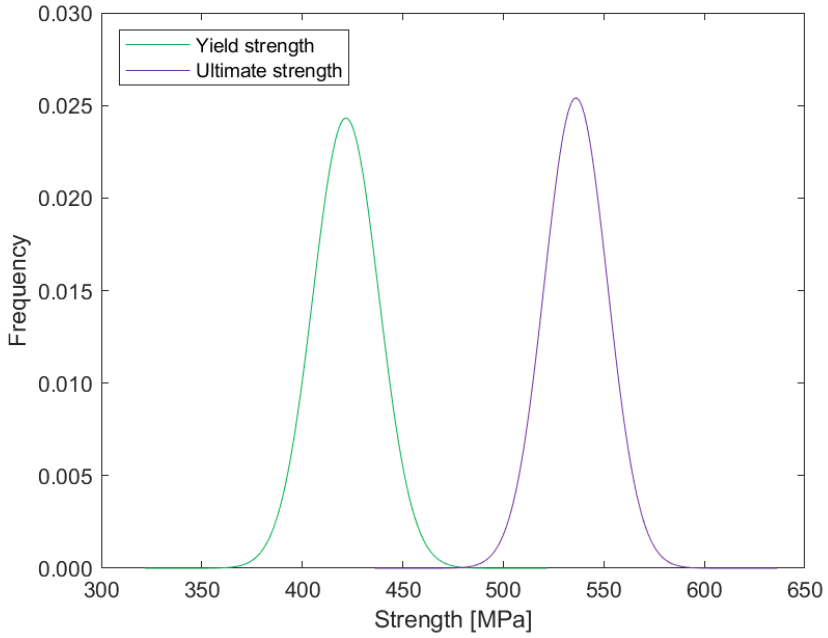
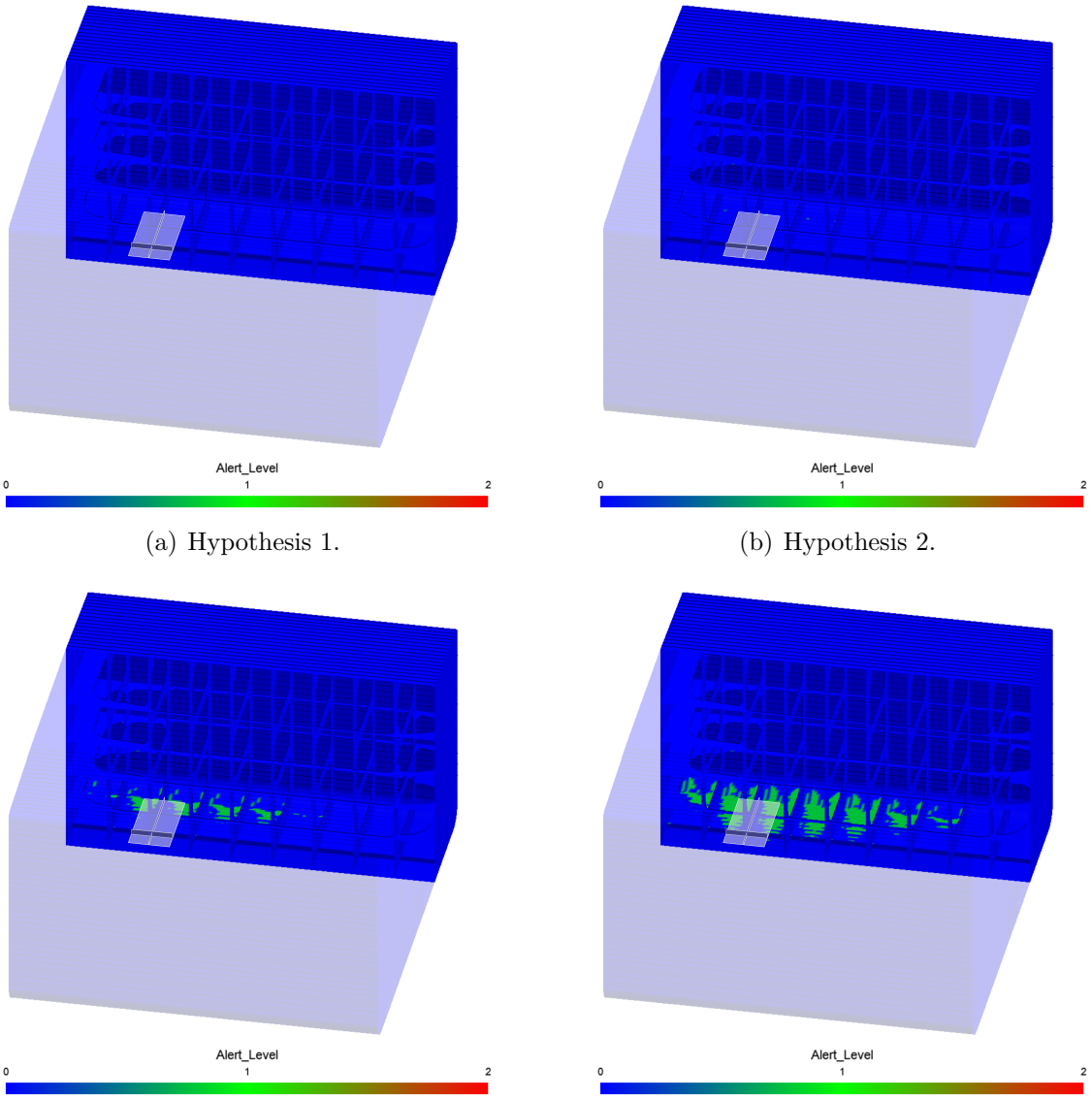


Figura 4.9.: Distribuição normal da resistência do aço (VanDerHorn e Wang, 2011).

hipóteses assumidas. No entanto, o nível de alerta 1 (Tabela 4.10) foi identificado nos painéis estruturais do fundo do casco nas hipóteses 2, 3 e 4, sendo mais significativo na hipótese 4 (Figura 4.10(d)) e diminuindo progressivamente as áreas destacadas do nível de alerta até a hipótese 2 (Figura 4.10(b)). Não houve identificação de nível de alerta 1 para a hipótese 1 em nenhum local do tanque.



(a) Hypothesis 1.

(b) Hypothesis 2.

(c) Hypothesis 3.

(d) Hypothesis 4.

Figura 4.10.: Corte transversal do modelo mostrando o nível de alerta previsto para o último ano de cada hipótese com a região de interesse destacada nas áreas sombreadas.

4.2.3.1. Análise de interferência (SSI)

Uma avaliação de risco mais precisa dos componentes estruturais de interesse encontrados na etapa anterior é realizada com uma análise de interferência de tensão-resistência (SSI - *Stress–Strength Interference*), levando em consideração a resistência do material e a distribuição de tensão para determinar a confiabilidade estrutural em regiões específicas.

A adoção da análise de confiabilidade selecionada pode ser justificada com base em diferentes fundamentos. Inicialmente, a utilização de métodos de análise de confiabilidade mais complexos possibilitaria considerar um maior número de incertezas inerentes ao modelo. No entanto, é imperativo empregar esses métodos com cautela, dado que as estimativas das probabilidades de falha estão sujeitas a diversas variáveis de incerteza no contexto da análise. Além disso, a abordagem de confiabilidade adotada neste exemplo é caracterizada por sua simplicidade, embora se concentre em áreas específicas da estrutura a delimitação permite uma focalização direcionada nos pontos críticos do sistema, otimizando a alocação de recursos empregados na análise. Outro aspecto relevante a ser enfatizado é a natureza adaptativa da metodologia apresentada, proporcionando ao usuário do gêmeo digital a capacidade de incorporar novos métodos de análise de confiabilidade de acordo com a necessidade identificada para o problema específico em questão. Essa flexibilidade permite uma abordagem personalizada e ajustável, considerando as particularidades e exigências do ativo sob análise.

Em geral, a variabilidade das propriedades mecânicas do material pode ser afetada por muitos fatores, como temperaturas de processamento e padrões de resfriamento durante o processo de fabricação. Esta fonte de incerteza está associada ao gêmeo real e sua quantificação probabilística pode ser estimada por uma função de densidade de probabilidade (PDF - *Probability Density Function*) a partir de dados fornecidos por testes experimentais e caracterização do material.

A análise SSI é baseada na interferência estatística de duas PDFs gaussianas, assumidas como $f_R(x)$ e $f_S(x)$ para representar, respectivamente, a componente com resistência R e tensão S. Assim, pode-se calcular a probabilidade de um componente com resistência R ser atingido quando submetido a uma tensão S.

A interação entre essas funções de probabilidade é exemplificada na Figura 4.11. Por definição, a área total sob a curva para qualquer PDF é sempre igual a 1. A ocorrência de falha é determinada pela probabilidade de S exceder R. Em termos

de uma função de densidade de probabilidade, a probabilidade de falha é definida como:

$$P\left(S > R\right) = \int_{-\infty}^{\infty} f_S(x) F_R(x) dx \tag{4.7}$$

onde $F_R(x)$ é a função de probabilidade acumulada da resistência do material.

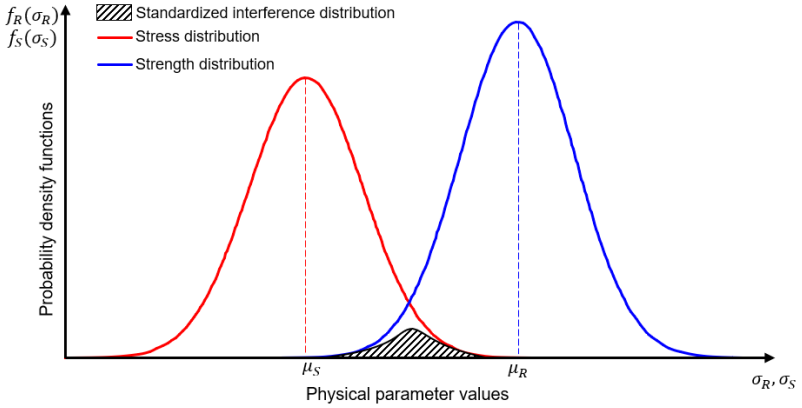


Figura 4.11.: Diagrama genérico para análise SSI.

Como R e S são assumidos como variáveis aleatórias normalmente distribuídas com valores médios μ_R e μ_S e variâncias σ_R^2 e σ_S^2. Consequentemente, a variável aleatória definida por $Z = R - S$ também é normalmente distribuída. A média de Z pode ser definida como $(\mu_R - \mu_S)$ e a variação como $(\sigma_R^2 - \sigma_S^2)$. Portanto, a área sob a curva de probabilidade normal de média μ_Z e variância σ_Z representa a probabilidade de falha e pode ser calculada pela função de distribuição cumulativa Φ da variável normal padronizada:

$$\begin{aligned} P(S > R) & \\ &= \int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} \sigma_z \left\{ exp\left[-\frac{1}{2} \left(\frac{x - \mu_z}{\sigma_z} \right)^2 \right] \right\} dx \\ &= \int_{-\infty}^{\frac{-\mu_z}{\sigma_z}} \frac{1}{\sqrt{2\pi}} exp\left(-\frac{1}{2} x^2 \right) dx \\ &= \Phi\left(-\frac{\mu_R - \mu_S}{\sqrt{\mu_R^2 + \mu_S^2}} \right). \end{aligned}$$

É importante observar que o método descrito anteriormente deve ser aplicado para avaliar componentes estruturais separadamente. A ampla variabilidade de tensão

em um grande modelo de elementos finitos pode afetar significativamente a curva de distribuição de tensão e subestimar a probabilidade de falha por reduzir a área de interferência.

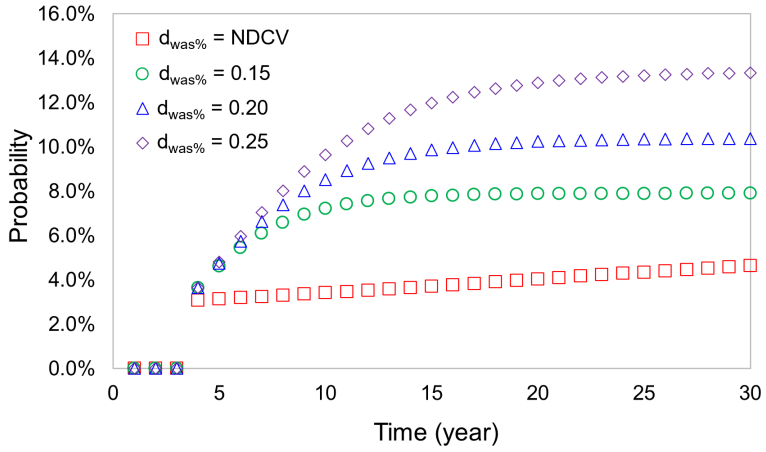
Não apenas o tratamento dos dados externos é uma preocupação para o DSS, mas também a seleção do componente estrutural para calcular a probabilidade de falha após a primeira etapa do DSS. A seleção desse componente deve ser feita de forma que os índices de confiabilidade não sejam superestimados como consequência de um componente extenso selecionado para a análise do SSI. No entanto, caso o nível de alerta da região não tenha sido identificado devido a uma singularidade da malha FE, este método tem a grande vantagem de desconsiderar os resultados para um componente estrutural, tendendo a probabilidade a zero com base no gradiente de tensões na maioria dos casos devido para a forma da curva de distribuição de tensão. O painel estrutural selecionado foi definido na região identificada com área a maior quantidade de nós no nível de alerta 1 para melhor resposta da avaliação de risco.

Após a seleção do componente estrutural de interesse (região destacada na área sombreada da Figura 4.10), a evolução da probabilidade de exceder limite de escoamento e limite último ao longo do tempo foi calculada para o ciclo de vida do FPSO com base na distribuição estatística do material (Tabela 4.10 e Tabela 4.11 e Figura 4.9) como mostrado na Figura 4.12.

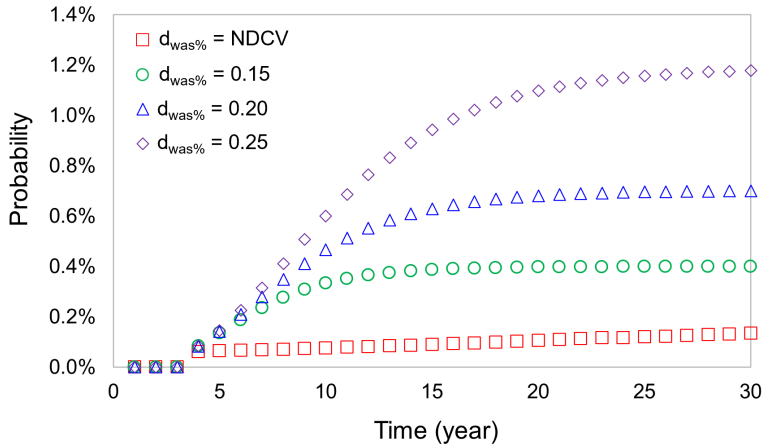
Para os resultados de probabilidade de falha da primeira hipótese (quando $d_{was\%}$ segue os valores NDCV), a probabilidade de exceder o limite de escoamento é muito baixa e varia levemente durante a vida útil, enquanto a probabilidade de exceder o limite último é menor que 0,2% para 30 anos. Isso era previsto, uma vez que as recomendações e diretrizes devem ser conservadoras. Além disso, como mostrado na Figura 4.7, a evolução de corrosão prevista pela hipótese 1 é linear e não fornece um comportamento realista para a perda de espessura em tanques de carga.

As probabilidades calculadas considerando as hipóteses 2, 3 e 4 (Tabela 4.6) representam os resultados mais significativos, pois são baseadas no modelo não linear de previsão de corrosão. Todas as três hipóteses começam com uma probabilidade de cerca de 4% de exceder o limite de escoamento. No entanto, a evolução das probabilidades é expressivamente diferente para cada caso, chegando a cerca de 8% para a hipótese 2; 10% para a hipótese 3; e 13% para a hipótese 4 no último ano da vida útil do FPSO. A curva da probabilidade de exceder o limite último ao longo dos anos ocorre de maneira análoga à do limite de escoamento. Assim, uma superfície de resposta interpolada usando os resultados dos casos hipotéticos 2, 3 e 4 foi criada

como apresentado na Figura 4.13.

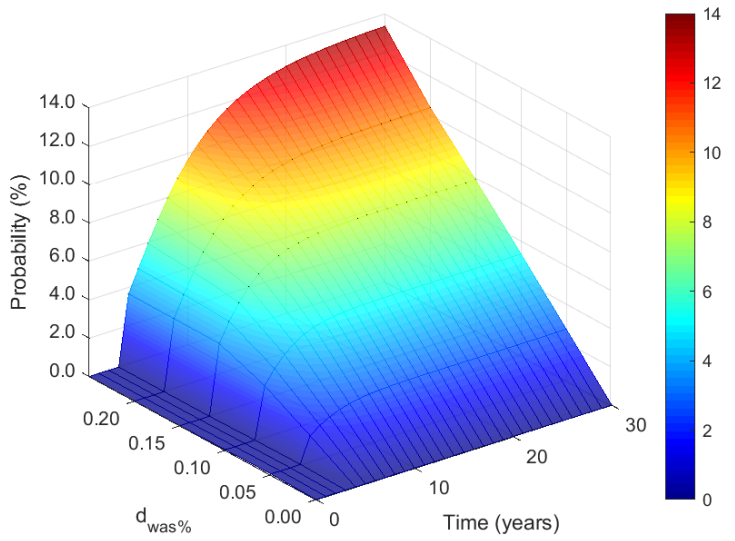


(a) Limite de escoamento.

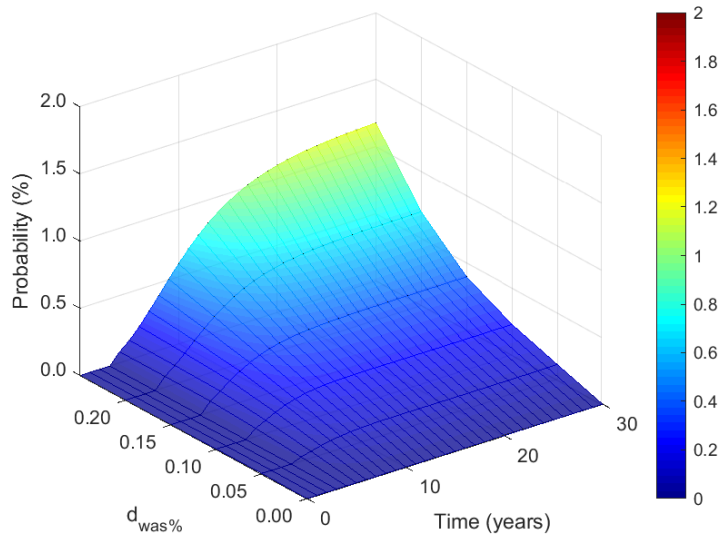


(b) Limite último.

Figura 4.12.: Probabilidade de exceder a distribuição estatística da resistência do material ao longo do tempo de vida útil para cada hipótese.



(a) Limite de escoamento.



(b) Limite último.

Figura 4.13.: Superfície de resposta interpolada para probabilidade de exceder a distribuição estatística de resistência do material de acordo com o tempo e $d_{was\%}$.

5. Considerações finais

Neste trabalho, foi proposta uma metodologia adaptativa para o desenvolvimento de um Gêmeo Digital para FPSO que realiza análises estruturais numéricas via MEF com base em sistemas acoplados. Utilizando modelos realistas, foram consideradas as exigências da embarcação com relação às diferentes fases do ciclo de vida, permitindo criar múltiplas análises numéricas.

Foram utilizados dados e cargas provenientes de outras análises numéricas, dentre elas a análise hidrodinâmica, para aplicação no modelo de elementos finitos do Gêmeo Digital.

Uma GUI foi desenvolvida e empregada para manipular os dados e algoritmos criados, executando as análises e gerando resultados de maneira mais simples pelo DSS, que possibilitou evidenciar regiões e elementos estruturais em condições críticas com base em um sistema com diferentes níveis de alerta.

O Gêmeo Digital proposto neste trabalho é capaz de retornar respostas automáticas de decisões decisões acerca da avaliação de risco, completando o ciclo de fluxo de dados citado por Kritzinger *et al.* (2018) para classificação de um Gêmeo Digital por meio do fluxo de dados (ver Figura 2.2). Para funcionamento aprimorado desta última fase, retornando dados ao objeto físico, seria necessário acesso às informações geradas durante o processo de projeto e produção do FPSO, mencionadas na Subseção 2.5.2.

O primeiro estudo de caso apresentado no trabalho, utilizou sistemas acoplados representando o cenário de fase de operação de um FPSO. Dados de ondas irregulares e níveis dos tanques foram utilizados para averiguar o desempenho do Gêmeo Digital. A rotina do fluxo de trabalho do Gêmeo Digital foi testada mostrando-se capaz de avaliar rapidamente, e de forma simples, a integridade estrutural da embarcação pelo DSS.

De maneira mais complexa, no segundo estudo, foi submetido um maior número de análises e uma avaliação de risco mais complexa da estrutura foi processada pelo

DSS. Assim, verificando as habilidades do Gêmeo Digital de modo mais rigoroso. Foram utilizados sistemas acoplados representando o cenário de fase de projeto de um FPSO, com dados de inspeção (utilizando um modelo não linear para evolução da corrosão), níveis dos tanques e onda regular. A avaliação de risco apontou probabilidade de falha das hipóteses levantadas de maneira consistente à evolução da corrosão.

A capacidade de usar vários dados de diferentes sistemas acoplados em um modelo de elementos finitos complexo para prever cenários hipotéticos ou simular condições reais pode impactar significativamente a tomada de decisões, melhorando a eficiência de operação do FPSO e prolongando o ciclo de vida da plataforma FPSO.

Por outro lado, a dificuldade em obter dados para criar o modelo, alimentar e comparar os resultados do Gêmeo Digital se demonstra com um grande desafio na implementação da metodologia proposta. O número de incertezas para construção e execução de um modelo global de FPSO demonstra ser um dos principais pontos críticos para desenvolvimento do Gêmeo Digital, pois as origens de incertezas são consideráveis (por exemplo: sistemas acoplados, parâmetros do modelo real, sensores).

5.1. Sugestão para futuras pesquisas

Devido à falta de referências na literatura, por se tratar de uma tecnologia recente, este trabalho deve ser considerado como uma contribuição para o desenvolvimento de Gêmeos Digitais na indústria de E&P com foco na integridade estrutural com base em análises numéricas via MEF. Assim, são destacadas sugestões para futuras pesquisas:

- Executar o Gêmeo Digital usando dados reais de sensores ou aplicando a pior condição de um local específico. À primeira vista, é necessário observar que o estado do mar e as condições de carregamento aplicadas ao modelo de elementos finitos neste trabalho foram selecionados apenas para exemplificar, e dados reais obtidos de sensores seriam mais úteis para prever a confiabilidade do FPSO;
- Obter manuais de orientação fornecidos pelo fabricante do FPSO para comparar a resposta estrutural. Se disponíveis, manuais de orientação fornecidos pelo

fabricante do FPSO poderiam facilitar o entendimento da resposta estrutural e a obtenção de um maior desempenho do Gêmeo Digital;

- Desenvolvimento de um Gêmeo Digital secundário usando um modelo de elementos finitos local (para componentes estruturais críticos) do modelo de elementos finitos. Assim, o modelo de elementos finitos primário (contendo modelo global) funcionaria como um sistema acoplado ao modelo de elementos finitos local secundário do componente estrutural, fornecendo os deslocamentos nodais adequados e permitindo uma avaliação de risco mais precisa para melhor investigar essas regiões;
- Utilizar um modelo *as built* considerando imperfeições geométricas do modelo real, observar diferenças consideráveis nos resultados e avaliar a importância da redução da incerteza do modelo;
- Empregar resultados armazenados pelo Gêmeo Digital para fazer a análise de fadiga ou aplicar uma carga cíclica para essa finalidade.

Bibliografia

ABS (2010). *Spectral-based fatigue analysis for floating production, storage and offloading (FPSO) installations*. American Bureau of Shipping. USA.

– (2020). *Advisory on Structural Health Monitoring: The Application of Sensor-Based Approaches*. American Bureau of Shipping. USA.

– (2022a). *Rules for building and classing - Floating production installations*. American Bureau of Shipping. USA.

– (2022b). *Rules for survey after construction*. American Bureau of Shipping. USA.

Ang, J. *et al.* (set. de 2017). "Efficient Hull Form Design Optimization using Hybrid Evolutionary Algorithm-Morphing Approach". Em: *International Conference on Computer Applications in Shipbuilding*. URL: http://eprints.gla.ac.uk/169729/.

ANP (2016). *Relatório Anual de Segurança Operacional das Atividades de Exploração e Produção de Petróleo e Gás Natural - 2016*. Rel. técn. Agência Nacional do Petróleo, Gás Natural e Biocombustíveis. URL: https://www.gov.br/anp/pt-br/assuntos/exploracao-e-producao-de-oleo-e-gas/seguranca-operacional-e-meio-ambiente/arq/raso/2016-relatorio-anual-seguranca-operacional.pdf.

– (2019). *Relatório Anual de Segurança Operacional das Atividades de Exploração e Produção de Petróleo e Gás Natural - 2019*. Rel. técn. Agência Nacional do Petróleo, Gás Natural e Biocombustíveis. URL: http://www.anp.gov.br/arquivos/exploracao-producao/sgom/dd/rso/2019-relatorio-anual-seguranca-operacional.pdf.

– (2020). *Relatório Anual de Segurança Operacional das Atividades de Exploração e Produção de Petróleo e Gás Natural - 2020*. Rel. técn. Agência Nacional do Petróleo, Gás Natural e Biocombustíveis. URL: https://www.gov.br/anp/pt-br/assuntos/exploracao-e-producao-de-oleo-e-gas/seguranca-operacional-e-meio-ambiente/arq/raso/2020-relatorio-anual-seguranca-operacional.pdf.

ANP (2021a). *Central de Sistemas ANP - Do poço ao posto.* Rel. técn. Agência Nacional do Petróleo, Gás Natural e Biocombustíveis. URL: http://app.anp.gov.br/anp-csa-web/.

– (2021b). *Lista de Plataformas em Operação - Dados Abertos.* Rel. técn. Agência Nacional do Petróleo, Gás Natural e Biocombustíveis. URL: https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/lista-de-plataformas-em-operacao.

– (2021c). *Painel dinâmico de descomissionamento de instalações de E&P.* URL: https://www.gov.br/anp/pt-br/centrais-de-conteudo/paineis-dinamicos-da-anp.

– (2022). *Anuário Estatístico Brasileiro do Petróleo, Gás Natural e Biocombustíveis 2022 - Dados Abertos.* Rel. técn. Agência Nacional do Petróleo, Gás Natural e Biocombustíveis. URL: https://www.gov.br/anp/pt-br/centrais-de-conteudo/dados-abertos/anuario-estatistico-2022.

ANSYS (2020). *ANSYS Mechanical 2020 documentation.* URL: https://ansyshelp.ansys.com/.

Ardhuin, f. *et al.* (2019). "Observing Sea States". Em: *Frontiers in Marine Science* 6, p. 124. ISSN: 2296-7745. DOI: 10.3389/fmars.2019.00124. URL: https://www.frontiersin.org/article/10.3389/fmars.2019.00124.

Augusto, O.B. (2007). "Módulo 4: Análise Estrutural de Navios". Em: ed. por Franz-Josef Kahlen, Shannon Flumerfelt e Anabela Alves.

Bacchiega, G. e G. Bondani (fev. de 2018). "Creating an Embedded Digital Twin: monitor, understand and predict Device Health Failure". Em.

Bai, Y., E. Bendiksen e P. Terndruppedersen (dez. de 1993). "Collapse analysis of ship hulls". Em: *Marine Structures* 6, pp. 485–507. DOI: 10.1016/0951-8339(93)90034-Z.

Boggs, D. *et al.* (2020). "2020 Worldwide Survey of Floating Production, Storage And Offloading (FPSO) Units". Em: *Offshore Magazine / Endeavor Business Mediae.* URL: https://www.offshore-mag.com/resources/maps-posters.

Bole, M., G. Powell e E. Rousseau (out. de 2017). "Taking Control of the Digital Twin". Em.

Bolton, R. *et al.* (2018). "Customer experience challenges: bringing together digital, physical and social realms". Em: *Journal of Service Management* 29, pp. 776–808. ISSN: 1757-5818. DOI: https://doi.org/10.1108/JOSM-04-2018-0113.

Caldwell, R. (set. de 2013). "Hull Inspection Techniques Strategies". Em: DOI: 10.2118/166570-MS.

CEIDA (2003). “Double Hull Tankers – Are they the answer?” Em: *Oil Companies International Marine Forum*. URL: http://www.ceida.org/prestige/Documentacion/dobrecascopetroleiros.pdf.

Cerrone, A. *et al.* (set. de 2014). “On the Effects of Modeling As-Manufactured Geometry: Toward Digital Twin”. Em: *International Journal of Aerospace Engineering* 2014. DOI: 10.1155/2014/439278.

Chakrabarti, Subrata, John Halkyard e Cuneyt Capanoglu (2005). *Chapter 1 - Historical Development of Offshore Structures*. Ed. por SUBRATA K. CHAKRABARTI. London: Elsevier, pp. 1–38. ISBN: 978-0-08-044381-2. DOI: https://doi.org/10.1016/B978-008044381-2.50004-7. URL: https://www.sciencedirect.com/science/article/pii/B9780080443812500047.

Christodoulou, Ioanna (2015). “Challenges and Opportunities that Define the Success of an FPSO Project”. Diss. de mestr. University of Nordland. URL: http://hdl.handle.net/11250/299812.

Cousins, S. (2017). “3D mapping Helsinki: How mega digital models can help city planners”. Em: *Construction Research and Innovation* 8.4, pp. 102–106. DOI: 10.1080/20450249.2017.1396747. eprint: https://doi.org/10.1080/20450249.2017.1396747. URL: https://doi.org/10.1080/20450249.2017.1396747.

Danielsen-Haces, A. (2018). “Digital Twin development: condition monitoring and simulation comparison for the ReVolt autonomous model ship”. Diss. de mestr. Norwegian University of Science e Technology.

DeCola, E. (2009). “A Review of Double Hull Tanker Oil Spill Prevention Considerations”. Em: *Nuka Research Planning Group, LLC.*, p. 34.

Devanney, Jack (2006). *The Tankship Tromedy, The Impending Disasters in Tankers0*. CTX Press, Tavernier, Florida. ISBN: 0-9776479-0-0.

DNV (2014). *Environmental conditions and environmental loads - DNV-RP-C205*. Det Norske Veritas. Norway.

– (2015). *Thickness diminution for mobile offshore units - DNVGL-CG-0172*. Det Norske Veritas. Norway.

Dobson, B. (jul. de 2013). “The Naval Engineering Workforce - A UK NEST Review”. Em.

Eide, Anders W. (2008). “Fundamental analysis and valuation of Prosafe Production”. Diss. de mestr. Copenhagen Business School.

Emi, H. *et al.* (1994). “A study on developing a rational corrosion protection system of hull structures”. Em: *NK Tech Bull* 246, pp. 65–79.

Ereiz, Suzana, Ivan Duvnjak e Javier Fernando Jiménez-Alonso (2022). "Review of finite element model updating methods for structural applications". Em: *Structures* 41, pp. 684–723. ISSN: 2352-0124. DOI: https://doi.org/10.1016/j.istruc.2022.05.041. URL: https://www.sciencedirect.com/science/article/pii/S2352012422004039.

Farias, B.V. e T.A. Netto (2012). "FPSO hull structural integrity evaluation via Bayesian updating of inspection data". Em: *Ocean Engineering* 56, pp. 10–19. ISSN: 0029-8018. DOI: https://doi.org/10.1016/j.oceaneng.2012.08.001. URL: https://www.sciencedirect.com/science/article/pii/S0029801812003150.

Ferreira, Nayara Nunes *et al.* (2020). "Guidelines for life extension process management in oil and gas facilities". Em: *Journal of Loss Prevention in the Process Industries* 68, p. 104290. ISSN: 0950-4230. DOI: https://doi.org/10.1016/j.jlp.2020.104290. URL: https://www.sciencedirect.com/science/article/pii/S0950423020305775.

Friswell, Michael e John E Mottershead (1995). *Finite element model updating in structural dynamics*. Vol. 38. Springer Science & Business Media.

Glaessgen, E.H. e D.S. Stargel (abr. de 2012). "The digital twin paradigm for future NASA and U.S. air force vehicles". Em: ISBN: 978-1-60086-937-2. DOI: 10.2514/6.2012-1818. URL: https://ntrs.nasa.gov/search.jsp?R=20120081782019-09-21T14:00:01+00:00Z.

Goh, G. (2015). *Building Singapore's 'digital twin'*. URL: https://www.digitalnewsasia.com/digital-economy/building-singapores-digital-twin (acesso em 15/09/2015).

Gordo, J.M. e M. Leal (2018). "A tool for analysis of costs on the manufacturing of the hull". Em: *Maritime Transportation and Harvesting of Sea Resources – Guedes Soares Teixeira (Eds)*. ISSN: 978-0-8153-7993-5.

Grange, E. (2018). "A Roadmap for Adopting a Digital Lifecycle Approach to Offshore Oil and Gas Production". Em: *Offshore Technology Conference*, p. 15. ISSN: 978-1-61399-571-6. DOI: 10.4043/28669-MS. URL: https://doi.org/10.4043/28669-MS.

Grieves, M. (ago. de 2016). "Origins of the Digital Twin Concept". Em: DOI: 10.13140/RG.2.2.26367.61609.

Grieves, Michael (2005a). Em: *Product Lifecycle Management: Driving the Next Generation of Lean Thinkin*. McGraw-Hill Education, p. 288. ISBN: 978-0071452304.

Grieves, Michael (jan. de 2005b). “Product lifecycle management: the new paradigm for enterprises”. Em: *International Journal of Product Development - Int J Prod Dev* 2. DOI: 10.1504/IJPD.2005.006669.

– (jul. de 2019). “Virtually Intelligent Product Systems: Digital and Physical Twins”. Em: pp. 175–200. ISBN: 978-1624105647. DOI: 10.2514/5.9781624105654.0175.0200.

Grieves, Michael e John Vickers (2017). “Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems”. Em: *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*. Ed. por Franz-Josef Kahlen, Shannon Flumerfelt e Anabela Alves. Cham: Springer International Publishing, pp. 85–113. ISBN: 978-3-319-38756-7. DOI: 10.1007/978-3-319-38756-7_4. URL: https://doi.org/10.1007/978-3-319-38756-7_4.

Handscomb, C., S. Sharabura e J. Woxholth (2016). *The oil and gas organization of the future*. URL: https://www.mckinsey.com/industries/oil-and-gas/our-insights/the-oil-and-gas-organization-of-the-future (acesso em 14/10/2019).

Heems, W. J. H. (2018). “A Simulation-Based Digital Twin for Monitoring and Predictive Performance of a HVAC System in a Cleanroom Environment”. Diss. de mestr. Eindhoven: Eindhoven University of Technology.

Hughes, O.F. (1983). *Ship Structural Design: A Rationally-based, Computer-aided, Optimization Approach*. Environmental Science and Technology. Wiley. ISBN: 9780471032410. URL: https://books.google.com.br/books?id=dm4ZAQAAIAAJ.

Kadir, B.A. (2017). *The Nine Pillars of Industry 4.0*. URL: https://www.4thpost.com/single-post/2017/07/23/The-nine-pillars-of-Industry-40.

Karve, Pranav M. *et al.* (fev. de 2020). “Digital Twin Approach for Damage-Tolerant Mission Planning Under Uncertainty”. Em: *Engineering Fracture Mechanics* 225, p. 106766. DOI: 10.1016/j.engfracmech.2019.106766. URL: https://www.sciencedirect.com/science/article/pii/S0013794419306496.

Khan, W. Z. *et al.* (2017). “A reliable Internet of Things based architecture for oil and gas industry”. Em: *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pp. 705–710. DOI: 10.23919/ICACT.2017.7890184.

Klare, Michael T. (mar. de 2009). *Rising powers, shrinking planet: the new geopolitics of energy*. Ed. por Henry Holt Company, p. 339. ISBN: 978-0805089219.

Kritzinger, W. *et al.* (2018). “Digital Twin in manufacturing: A categorical literature review and classification”. Em: *IFAC-PapersOnLine* 51.11. 16th IFAC Symposium

on Information Control Problems in Manufacturing INCOM 2018, pp. 1016 –1022. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.08.474. URL: http://www.sciencedirect.com/science/article/pii/S2405896318316021.

Lee, J., B. Bagheri e H. Kao (2015). "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems". Em: *Manufacturing Letters* 3, pp. 18 –23. ISSN: 2213-8463. DOI: https://doi.org/10.1016/j.mfglet.2014.12.001. URL: http://www.sciencedirect.com/science/article/pii/S221384631400025X.

Liu, Yuchao e Huilong Ren (2022). "Acquisition method of evaluation stress for the digital twin model of ship monitoring structure". Em: *Applied Ocean Research* 129, p. 103368. ISSN: 0141-1187. DOI: https://doi.org/10.1016/j.apor.2022.103368. URL: https://www.sciencedirect.com/science/article/pii/S0141118722002991.

Lu, Hongfang *et al.* (2019). "Oil and Gas 4.0 era: A systematic review and outlook". Em: *Computers in Industry* 111, pp. 68–90. DOI: 10.1016/j.compind.2019.06.007.

Lu, Yuqian *et al.* (2020). "Digital Twin-Driven Smart Manufacturing: Connotation, Reference Model, Applications and Research Issues". Em: *Robotics and Computer-Integrated Manufacturing* 61, p. 101837. DOI: 10.1016/j.rcim.2019.101837. URL: https://www.sciencedirect.com/science/article/pii/S0736584519302480?via%3Dihub.

Lydon, G. P. *et al.* (2019). "Coupled simulation of thermally active building systems to support a digital twin". Em: *Energy and Buildings* 202, p. 109298. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2019.07.015. URL: http://www.sciencedirect.com/science/article/pii/S0378778819305201.

MacMillan, A. (jan. de 2001). "Effective FPSO/FSO hull structural design". Em: 3. DOI: 10.4043/13212-MS.

Malta, Edgard Borges (2010). "Métodos e processos para a análise experimental de sistemas oceânicos de produção de petróleo e gás". Diss. de mestr. Engenharia Naval e Oceânica da Escola Politécnica da USP. DOI: https://doi.org/10.11606/D.3.2010.tde-17082010-113906.

Drilling Digital Twin Success Stories the Last 10 Years (abr. de 2018). Vol. Day 1 Wed, April 18, 2018. SPE Norway Subsurface Conference. D011S007R001. DOI: 10.2118/191336-MS. eprint: https://onepetro.org/SPEBERG/proceedings-pdf/18BERG/1-18BERG/D011S007R001/1189274/spe-191336-ms.pdf. URL: https://doi.org/10.2118/191336-MS.

Probabilistic Modelling of Marine Corrosion of Steel Specimens (jun. de 1995). Vol. All Days. International Ocean and Polar Engineering Conference. ISOPE-I-95-311. eprint: https://onepetro.org/ISOPEIOPEC/proceedings-pdf/ISOPE95/All-ISOPE95/ISOPE-I-95-311/1969289/isope-i-95-311.pdf.

Moghadam, Farid K. e Amir R. Nejad (2022). "Online condition monitoring of floating wind turbines drivetrain by means of digital twin". Em: *Mechanical Systems and Signal Processing* 162, p. 108087. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2021.108087. URL: https://www.sciencedirect.com/science/article/pii/S0888327021004738.

Negri, E., L. Fumagalli e M. Macchi (2017). "A Review of the Roles of Digital Twin in CPS-based Production Systems". Em: *Procedia Manufacturing* 11. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy, pp. 939 –948. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2017.07.198. URL: http://www.sciencedirect.com/science/article/pii/S2351978917304067.

NRC, National Research Council (1998). *Double Hull Tanker Legislation: An Assessment of The Oil Pollution Act of 1990*. National Academy Press, Washington, D.C.

Okumoto, Y. *et al.* (jan. de 2009). *Design of ship hull structures: A practical guide for engineers*, pp. 1–578. DOI: 10.1007/978-3-540-88445-3.

Petrobras (2021). *Tipos de plataformas.* URL: https://petrobras.com.br/infograficos/tipos-de-plataformas/desktop/index.html#.

Petrobras, Econservation e Kick (2019). *Relatório de Impacto Ambiental - Revitalização Marlim/Voador - 2019.* Rel. técn. Petrobras.

Piascik, R. *et al.* (2010). "Technology Area 12: Materials, Structures, Mechanical Systems, and Manufacturing Road Map." Em: *NASA Office of Chief Technologist*, p. 36.

Poddar, Tushar (mar. de 2018). "Digital Twin Bridging Intelligence Among Man, Machine and Environment". Em: *Offshore Technology Conference*. URL: https://www.onepetro.org/conference-paper/OTC-28480-MS?sort=&start=0&q=2018+poddar+&from_year=&peer_reviewed=&published_between=&fromSearchResults=true&to_year=&rows=25#.

Pylianidis, Christos, Sjoukje Osinga e Ioannis N. Athanasiadis (2021). "Introducing digital twins to agriculture". Em: *Computers and Electronics in Agriculture* 184, p. 105942. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.

2020.105942. URL: https://www.sciencedirect.com/science/article/pii/S0168169920331471.

Rasheed, A., O. San e T. Kvamsdal (out. de 2019). "Digital Twin: Values, Challenges and Enablers". Em.

Reifsnider, K. e P. Majumdar (abr. de 2013). "Multiphysics Stimulated Simulation Digital Twin Methods for Fleet Management". Em: ISBN: 978-1-62410-223-3. DOI: 10.2514/6.2013-1578.

Renzi, D. *et al.* (2017). "Developing a Digital Twin for Floating Production Systems Integrity Management". Em: *Offshore Technology Conference*, p. 8. ISSN: 978-1-61399-541-9. DOI: 10.4043/28012-MS. URL: https://www.onepetro.org/conference-paper/OTC-28012-MS.

Riva, I.R. (2004). "Análise de Fadiga em Estruturas Metálicas com Ênfase em Offshore". Diss. de mestr. Rio de Janeiro, RJ, Brasil: UFRJ.

Rosen, R. *et al.* (2015). "About The Importance of Autonomy and Digital Twins for the Future of Manufacturing". Em: *IFAC-PapersOnLine* 48.3. 15th IFAC Symposium onInformation Control Problems inManufacturing, pp. 567 –572. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.06.141. URL: http://www.sciencedirect.com/science/article/pii/S2405896315003808.

Rystad (2020). *Report*. Rel. técn. Rystad Energy Consulting. URL: https://www.rystadenergy.com/newsevents/news/newsletters/CompanyArchive/cn-march-2020/ (acesso em 10/03/2020).

– (2022). *Press release - Global oil and gas exploration shrinks as companies shift focus to lower-risk core assets and regions*. Rel. técn. Rystad Energy Consulting. URL: https://www.rystadenergy.com/news/global-oil-and-gas-exploration-shrinks-as-companies-shift-focus-to-lower-risk-cor (acesso em 05/01/2023).

Shahbaznia, Mahdi, Morteza Raissi Dehkordi e Akbar Mirzaee (2020). "An Improved Time-Domain Damage Detection Method for Railway Bridges Subjected to Unknown Moving Loads". Em: *Periodica Polytechnica Civil Engineering* 64.3, 928–938. DOI: 10.3311/PPci.15813. URL: https://pp.bme.hu/ci/article/view/15813.

Shetelig, H. (2013). "Shipbuilding Cost Estimation - Parametric Approach". Diss. de mestr. Norwegian University of Science e Technology.

Single vs Double Sides for FPSO Hulls - A Decision Model (mar. de 2004). Vol. All Days. SPE International Conference and Exhibition on Health, Safety, Environment, and Sustainability. SPE-86669-MS. DOI: 10.2118/86669-MS. eprint: https:

//onepetro.org/SPEHSE/proceedings-pdf/04HSE/All-04HSE/SPE-86669-MS/1858935/spe-86669-ms.pdf. URL: https://doi.org/10.2118/86669-MS.

Soares, C.Guedes e Y. Garbatov (1999). "Reliability of maintained, corrosion protected plates subjected to non-linear corrosion and compressive loads". Em: *Marine Structures* 12.6, pp. 425–445. ISSN: 0951-8339. DOI: https://doi.org/10.1016/S0951-8339(99)00028-3. URL: https://www.sciencedirect.com/science/article/pii/S0951833999000283.

Söderberg, R. *et al.* (2017). "Toward a Digital Twin for real-time geometry assurance in individualized production". Em: *CIRP Annals* 66.1, pp. 137 –140. ISSN: 0007-8506. DOI: https://doi.org/10.1016/j.cirp.2017.04.038. URL: http://www.sciencedirect.com/science/article/pii/S0007850617300380.

Southwell, C.R., J.D. Bultman e C.W. Hummer (1979). "Estimating service life of steel in seawater". Em: *Seawater Corrosion Handbook*. Ed. por Schumacher M. Park Ridge, NJ: Noyes Data Corp., pp. 374–387.

Tammer, M. *et al.* (jan. de 2014). "Current performance and future practices in FPSO hull condition assessments". Em: *Proceedings of the International Offshore and Polar Engineering Conference*, pp. 332–338.

Tao, Fei *et al.* (2018). "Digital twin-driven product design framework". Em: *International Journal of Production Research* 57.12, pp. 3935–3953. DOI: 10.1080/00207543.2018.1443229.

Terpstra, T. (jan. de 2001). "FPSO design and conversion: A designer's approach". Em: 3, pp. 319–339.

TRB (2001). *Transportation Research Board and National Academies of Sciences, Engineering, and Medicine - Environmental Performance of Tanker Designs in Collision and Grounding: Method for Comparision – Special Report 259*. Washington, DC: The National Academies Press. DOI: 10.17226/10199. URL: https://www.nap.edu/catalog/10199/environmental-performance-of-tanker-designs-in-collision-and-grounding-method.

Tuegel, E.J. *et al.* (2011). "Reengineering Aircraft Structural Life Prediction Using a Digital Twin". Em: *International Journal of Aerospace Engineering* 2011, p. 14. ISSN: 10.1155/2011/154798. DOI: http://dx.doi.org/10.1155/2011/154798. URL: https://www.hindawi.com/journals/ijae/2011/154798/abs/.

Tuegel, Eric (2012). "The Airframe Digital Twin: Some Challenges to Realization". Em: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference&ltBR&gt20th AIAA/ASME/AHS Adaptive Structures*

Conference&ltBR&gt14th AIAA. American Institute of Aeronautics e Astronautics. DOI: 10.2514/6.2012-1812.

VanDerHorn, Eric e G. Wang (set. de 2011). "A statistical study on the material properties of shipbuilding steels". Em: pp. 371–378.

Verdouw, Cor *et al.* (2021). "Digital twins in smart farming". Em: *Agricultural Systems* 189, p. 103046. ISSN: 0308-521X. DOI: https://doi.org/10.1016/j.agsy.2020.103046. URL: https://www.sciencedirect.com/science/article/pii/S0308521X20309070.

WAMIT (2018). *WAMIT User Manual 7.3*. Rel. técn. WAMIT Inc. URL: http://www.wamit.com/manualupdate/v73_manual.pdf.

Wang, H.K. *et al.* (mar. de 2015). "The Use of High-Performance Fatigue Mechanics and the Extended Kalman/Particle Filters, for Diagnostics and Prognostics of Aircraft Structures". Em: *CMES - Computer Modeling in Engineering and Sciences* 105, pp. 1–24. DOI: 10.1201/b18009-2.

Yang, C. e F. Huang (2016). "An overview of simulation-based hydrodynamic design of ship hull forms". Em: *Journal of Hydrodynamics, Ser. B* 28.6, pp. 947–960. ISSN: 1001-6058. DOI: https://doi.org/10.1016/S1001-6058(16)60696-0. URL: http://www.sciencedirect.com/science/article/pii/S1001605816606960.

Yip, Tsz Leung, Wayne K. Talley e Di Jin (2011). "The effectiveness of double hulls in reducing vessel-accident oil spillage". Em: *Marine Pollution Bulletin* 62.11, pp. 2427–2432. ISSN: 0025-326X. DOI: https://doi.org/10.1016/j.marpolbul.2011.08.026. URL: https://www.sciencedirect.com/science/article/pii/S0025326X11004504.

"Chapter 9 - Ocean Wave Measurement" (1999). Em: *Wind Generated Ocean Waves*. Ed. por Ian R. Young. Vol. 2. Elsevier Ocean Engineering Series. Elsevier, pp. 227–254. DOI: https://doi.org/10.1016/S1571-9952(99)80011-7. URL: https://www.sciencedirect.com/science/article/pii/S1571995299800117.

Zacharias, Daniel Constantino, Carine Malagolini Gama e Adalgiza Fornaro (2021). "Mysterious oil spill on Brazilian coast: Analysis and estimates". Em: *Marine Pollution Bulletin* 165, p. 112125. ISSN: 0025-326X. DOI: https://doi.org/10.1016/j.marpolbul.2021.112125. URL: https://www.sciencedirect.com/science/article/pii/S0025326X21001594.

Zanganeh, Razieh e Krish Thiagarajan (2018). "Prediction of the mean heading of a turret moored FPSO in bi-modal and bi-directional sea states". Em: *Applied Ocean Research* 78, pp. 156–166. ISSN: 0141-1187. DOI: https://doi.org/10.

1016/j.apor.2018.04.006. URL: https://www.sciencedirect.com/science/article/pii/S0141118717306466.

A. Apêndice - Elaboração do modelo de elementos finitos

A.1. Aspectos e diretrizes gerais

Os modelos em elementos finitos para os estudos de caso apresentados nesse trabalho seguiram as diretrizes apresentadas em ABS, 2022a, as quais propõem recomendações para a construção de um modelo global de viga-navio com três tanques de carga com extensão de duas cavernas em cada extremidade (ver Figura A.1), incluindo todos os elementos estruturais primários e elementos estruturais secundários que afetem o carregamento global.

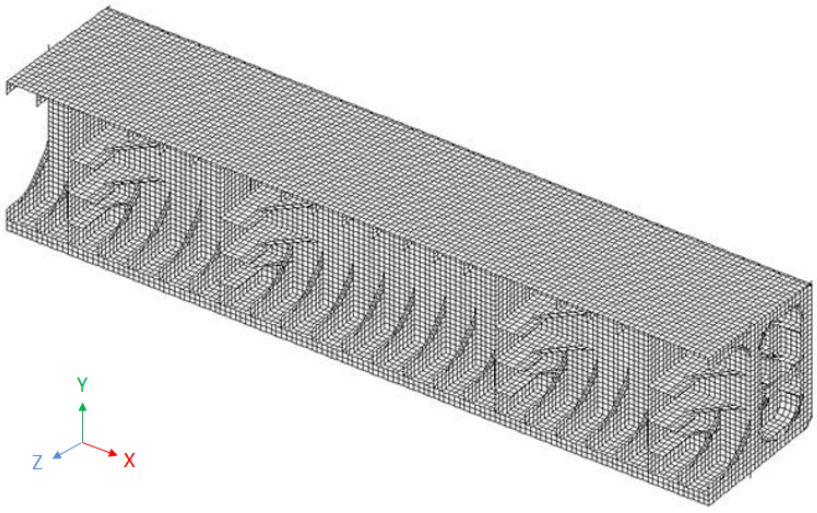


Figura A.1.: Exemplo de modelo de elementos finitos global de três tanques (adaptado de ABS, 2022a).

O modelo global de elementos finitos de três tanques de carga pode ser utilizado para análise estrutural dos tanques de carga e lastro localizados no intervalo de 0,4L (comprimento) à meia nau, avaliando tensão de viga-navio e também resistência ao

escoamento e flambagem de elementos primários longitudinais e transversais. Caso desejado, a partir desse modelo é possível obter condições de contorno adequadas para serem utilizadas em uma análise local com malha refinada.

Algumas instruções para criação do modelo podem ser destacadas nos tópicos a seguir:

- O modelo global precisa incluir todos os membros estruturais principais de carga, quaisquer membros estruturais primários e secundários que influenciem na distribuição geral da carga devem ser considerados apropriadamente.
- A idealização estrutural deve ser baseada na rigidez e na resposta antecipada da estrutura, não apenas na geometria da estrutura.
- Representar bem a geometria estrutural implica em ter um bom modelo. As propriedades estruturais é que são a base para uma boa qualidade de resposta a ser provida pelo modelo.
- É desejável obter um bom modelo para os três tanques de carga. No entanto, o tanque central deve sempre ser priorizado, onde resultados mais precisos são esperados. Se aproximações tiverem que ser feitas, que sejam apenas nos tanques das extremidades onde há maior interferência das condições de contorno.
- A razão de aspecto [1] dos elementos não deve ultrapassar 3,0.
- A razão de aspecto dos elementos em áreas de tensão elevada deve ser próxima a 1,0.
- Evitar o uso de elementos triangulares (especialmente em áreas de tensão elevada).
- Ignorar orifícios em estruturas longitudinais ou transversais e não utilizar nenhum artifício para modelar o orifício (não representar o elemento de placa, reduzir a espessura, etc.).
- No mínimo 3 elementos entre as cavernas.
- Elementos de alta ordem podem ser utilizados, mas não são necessários.
- Adição de duas cavernas após extremidade dos três tanques do modelo.
- Tipos de elementos mais indicados:

[1] Elementos de casca são utilizados para modelagem do casco, tendo a razão de aspecto definida pelo razão entre os seus lados.

  - Elementos unidimensionais (barra/treliça) - rigidez axial, área de seção transversal constante ao longo do comprimento do elemento.
  - Elementos de viga - sem deslocamento - *offset*, com cisalhamento axial, torcional e bidirecional; rigidez à flexão, com propriedades constantes ao longo do comprimento do elemento.
  - Elementos de casca - com rigidez no plano e rigidez de flexão fora do plano com espessura constante.

O sistema de coordenadas utilizado no modelo estrutural de elementos finitos é sugerido a seguir:

- Origem: linha de base com linha de centro na antepara transversal estanque do tanque mais a ré.
- Eixo-x: longitudinal (positivo para vante).
- Eixo-y: vertical (positivo para cima).
- Eixo-z: transversal (positivo para boreste).

As unidades utilizadas sugeridas são:

- Comprimento: cm
- Massa: kg
- Tensão: kgf/cm^2

A.2. Condições de contorno

As condições de contorno são determinadas por elementos longitudinais ligados às extremidades do modelo de elementos finitos, os quais devem estar rigidamente conectados ao ponto independente definido pela linha central e linha neutra da seção (ver Figura A.2 e Tabela A.1). As forças cisalhantes devem ser representadas por molas (elementos unidimensionais apenas com rigidez axial). As molas horizontais devem estar conectadas ao convés e ao fundo, e as molas verticais devem estar conectadas à lateral do casco e às anteparas longitudinais. A segunda extremidade de molas deve estar fixa em todos os graus de liberdade.

A rigidez da mola é equivalente ao apoio dado à antepara de extremidade considerada pelos membros estruturais longitudinais à seção transversal. A resultante das molas na seção transversal pode ser determinada pela Equação A.1:

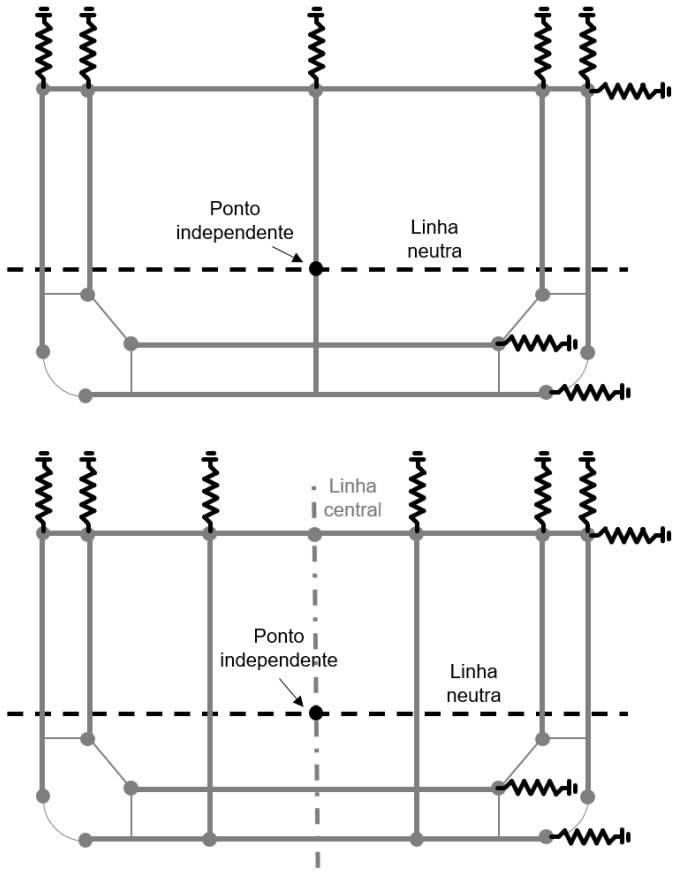


Figura A.2.: Restrições de mola nas extremidades dos modelos FE (adaptado de ABS, 2022a).

Tabela A.1.: Condições de contorno (adaptado de ABS, 2022a).

Elementos longitudinais	LR	-	-	-	LR	LR
Ponto independente	Engaste	-	-	-	-	-
Convés e fundo	-	Mola	-	-	-	-
Costado e anteparas longitudinais	-	-	Mola	-	-	-
Extremidade inicial						
Elementos longitudinais	LR	-	-		LR	LR
Ponto independente	-	-	-	-	-	-
Convés e fundo	-	Mola	-	-	-	-
Costado e anteparas longitudinais	-	-	Mola	-	-	-

onde:
LR = Ligação Rígida.
- = Sem restrição aplicada (livre).

$$A = \left(\frac{1}{1+\nu}\right)\frac{A_s l}{l_{tk} n} \tag{A.1}$$

onde, A é a área da seção transversal da mola (representada pelo elemento unidimensional), A_s é a área de cisalhamento do elemento estrutural individual conectado, ν é a razão de Poisson do material , l_{tk} é o comprimento do tanque de carga (entre anteparas do tanque intermediário), n é o número de pontos nodais aos quais os elementos de mola são aplicados ao membro estrutural conectado e l é o comprimento de a mola (geralmente adotado igual a 1). As unidades utilizadas na Equação A.1 devem ser consistentes com as unidades utilizadas no modelo de elementos finitos.

A.3. Construção do modelo de elementos finitos

Para que o modelo de elementos finitos construído seja adequado à metodologia adaptativa, algumas orientações devem ser seguidas:

1. Não é necessário aplicar pressões hidrodinâmicas no modelo de elementos finitos. Para acrescentar esse efeito deve ser realizado o acoplamento dos dados da análise hidrodinâmica de maneira compatível com o arquivo de entrada para pressões hidrodinâmicas (ver Subseção B.2.1);
2. Todo e qualquer carregamento (exemplo: peso próprio da estrutura, pressões hidrostáticas, pontos de massa, forças remotas, etc.) deve ser incluído no modelo de elementos finitos (a única exceção é feita para pressões hidrodinâmicas);
3. É recomendada a construção do modelo de elementos finitos utilizando os valores reais de espessura de chapas e parâmetros das seções das vigas. Esses valores podem ser lidos e listados (ver Subseção C.1.7) facilitando a criação do arquivo de dados de inspeção (ver Subseção B.2.3) utilizado para atualização desses valores;
4. A construção do modelo de elementos finitos pode ser feita com valores fictícios de espessura de chapas e parâmetros das seções das vigas. No entanto, a atualização desses valores por meio de um arquivo de dados de inspeção (ver Subseção B.2.3) durante a operação do Gêmeo Digital consome mais tempo e memória computacional;

5. É recomendado que o modelo seja executado pelo *solver* utilizado para assegurar que o modelo esteja operando conforme o almejado;
6. Uma vez que há a consideração de um problema linear estático, é considerado que a resolução do problema é feita em um único passo de simulação. Assim, o controle de passos deve ser alterado nas configurações para assegurar que a análise numérica ocorrerá em um único passo;
7. Toda e qualquer propriedade ou carga (propriedades dos materiais, densidade dos fluidos, etc.) não alterada pelos sistemas acoplados é mantida como a definida no modelo de elementos finitos;
8. O sistema de unidades adotado pelas variáveis e arquivos devem ser consistentes com o sistema de unidades adotado no modelo de elementos finitos;
9. Devido à necessidade de se exportar a malha da superfície do casco abaixo do calado para programas de análise hidrodinâmica, essa região não pode ser modelada em mais de uma entidade geométrica (*body*). Além disso, o arquivo de modelo de elementos finitos deve ter essa entidade nomeada como "Hull"para reconhecimento dos algoritmos;
10. Após finalização, o arquivo do modelo de elementos finitos (de extensão "`.dat`") deve ser salvo no diretório "`A02 dat base`".

A.4. Inserção de códigos de substituição

Essa etapa é necessária caso seja pretendido durante a operação do Gêmeo Digital a realização de qualquer uma das rotinas de atualização abaixo:

1. Alternar entre considerar ou não o peso próprio da estrutura;
2. Atualizar nível dos tanques (controle da aplicação de pressões hidrostáticas exercidas pelo conteúdo de cada tanque);
3. Atualizar pontos de massa (centro de gravidade, massas e momentos de inércia) para o conteúdo de cada tanque;
4. Atualizar os valores de carregamentos (exceto pressão hidrodinâmica).

Após construção do modelo de elementos finitos descrito na Seção A.3, o arquivo do modelo de elementos finitos salvo no diretório "`A02 dat base`" deve ser copiado no mesmo diretório e renomeado (é sugerido manter o mesmo nome do arquivo original com adição do sufixo "_codes"). Esse arquivo deve ser editado da seguinte forma:

1. Para inserir manualmente o código para consideração ou não do peso próprio da estrutura em relação ao eixo transversal (vertical):

   a) Identificar e substituir os valores das componentes X, Y e Z de aceleração do comando "`acel`" respectivamente pelos códigos: "`***Value_SEGX***`", "`***Value_SEGY***`" e "`***Value_SEGZ***`".

   Este comando aparecerá dentro das definições do bloco "`/com,****************** SOLVE FOR LS 1 OF 1 ****************`"

   Exemplo:

   ```
   acel,0.,0.,9.80665
   ```

   deve ser substituído por:

   ```
   acel,***Value_SEGX***,***Value_SEGY***,***Value_SEGZ***
   ```

2. Para inserir manualmente os códigos para atualização do nível dos tanques:

   a) Identificar e substituir os valores do nível da água (externo e interno) pelos códigos.

   Estes valores aparecerão dentro das definições do bloco "`/com,***********
   Send User Defined Coordinate System(s) ***********`"

   Exemplo:

   ```
   local,12,0,0.,0.,16.24,0.,0.,-90.
   local,13,0,0.,0.,17.25,0.,0.,-90.
   local,14,0,0.,0.,18.15,0.,0.,-90.
   local,15,0,0.,0.,19.25,0.,0.,-90.
   local,16,0,0.,0.,18.25,0.,0.,-90.
   local,17,0,0.,0.,17.15,0.,0.,-90.
   ```

   deve ser substituído por:

```
local,12,0,0.,0.,***HSP_Zplan***,0.,0.,-90.
local,13,0,0.,0.,***HSP_T01_Zplan***,0.,0.,-90.
local,14,0,0.,0.,***HSP_T02_Zplan***,0.,0.,-90.
local,15,0,0.,0.,***HSP_T03_Zplan***,0.,0.,-90.
local,16,0,0.,0.,***HSP_T04_Zplan***,0.,0.,-90.
local,17,0,0.,0.,***HSP_T05_Zplan***,0.,0.,-90.
```

- Note que a quantidade de variáveis é dependente da quantidade de tanques que receberam aplicação de pressão hidrostática no modelo.
- Note que a posição da variável (separada por vírgulas no arquivo) a ser substituída é dependente do eixo transversal (vertical) adotado no modelo. Sendo:
  - A quarta variável se adotado o eixo X para eixo transversal (vertical) do modelo;
  - A quinta variável se adotado o eixo Y para eixo transversal (vertical) do modelo;
  - A sexta variável se adotado o eixo Z para eixo transversal (vertical) do modelo.

3. Para inserir manualmente os códigos para atualização dos pontos de massa:

   a) Identificar e substituir as coordenadas X, Y e Z do centro de gravidade do conteúdo de cada tanque pelos códigos.

   Estes valores aparecerão como nós para pontos remotos dentro das definições do bloco `"/com,********** Nodes for all Remote Points **********"`

   Exemplo:

1839167	-6.267499924E+01	0.000000000E+00	1.375300000E+01
1839168	-4.090000000E+01	0.000000000E+00	1.375300000E+01
1839169	0.000000000E+00	0.000000000E+00	1.375300000E+01
1839170	4.090000000E+01	0.000000000E+00	1.375300000E+01
1839171	6.267499924E+01	0.000000000E+00	1.375300000E+01

   deve ser substituído por:

1839167	***Value_T1Xloc***	***Value_T1Yloc***	***Value_T1Zloc***
1839168	***Value_T2Xloc***	***Value_T2Yloc***	***Value_T2Zloc***
1839169	***Value_T3Xloc***	***Value_T3Yloc***	***Value_T3Zloc***
1839170	***Value_T4Xloc***	***Value_T4Yloc***	***Value_T4Zloc***
1839171	***Value_T5Xloc***	***Value_T5Yloc***	***Value_T5Zloc***

- Note que a quantidade de variáveis é dependente da quantidade de pontos de massa criadas para os tanques.

b) Identificar e substituir os valores da massa e momentos de inércia (I_{XX}, I_{YY} e I_{ZZ}) do conteúdo de cada tanque pelos códigos.

Estes valores aparecerão dentro das definições dos blocos "`/com,**********Construct Remote Mass Using Remote Attachment **********`" criados para cada ponto de massa adicionado no modelo.

Observação: O valor da massa é repetido três vezes no arquivo (em vermelho no exemplo abaixo), no entanto, os três valores devem ser substituídos apenas por um código para representar a massa.

Exemplo:

r,_tid, 26169642.58,26169642.58,26169642.58,7414142112.36,2907053611.67,9170911352.39

deve ser substituído por:

r,_tid,***Value_T2_M***,***Value_T2IXX***,***Value_T2IYY***,***Value_T2IZZ***

- Note que esse passo deve ser repetido até o total de pontos de massa criados no modelo.
- Note que a posição das variáveis (separadas por vírgulas no arquivo) a serem substituídas seguem a ordem:
  - A terceira, quarta e quinta variável representam o valor da massa do ponto de massa;
  - A sexta variável representa o valor do momento de inércia no eixo X;
  - A sétima variável representa o valor do momento de inércia no eixo Y;

- A oitava variável representa o valor do momento de inércia no eixo Z;

4. Para inserir manualmente os códigos para atualização dos valores de força remota (para consideração das forças de tração no topo das linhas de ancoragem):

   a) Identificar e substituir as coordenadas X, Y e Z do ponto remoto.

   Este valor aparecerá como nó para ponto remoto dentro das definições do bloco `"/com,********** Nodes for all Remote Points **********"`

   Exemplo:

```
1839162  1.500000000E+02  0.000000000E+00  1.375300000E+01
```

   deve ser substituído por:

```
1839162  ***Value_RFXloc***  ***Value_RFYloc***  ***Value_RFZloc***
```

   b) Identificar e substituir os componentes de força X, Y e Z.

   Estes valores aparecerão dentro das definições do bloco `"/com,****************** SOLVE FOR LS 1 OF 1 ****************"` criados para cada ponto de massa adicionado no modelo.

   Exemplo:

```
nsel,s,node,,1839162
f,all,fx,0.
f,all,fy,0.
f,all,fz,-10000000.
```

   deve ser substituído por:

```
nsel,s,node,,157846
f,all,fx,***Value_RFXcomp***
f,all,fy,***Value_RFYcomp***
f,all,fz,***Value_RFZcomp***
```

A.5. Inserção de códigos de substituição para dados de inspeção

Essa etapa é necessária caso seja pretendido durante a operação do Gêmeo Digital a atualização de espessura de chapas e, ou dimensões de seções das vigas.

Após construção do modelo de elementos finitos descrito na Seção A.4 (caso a etapa descrita na Seção A.4 não tenha sido realizada, deve ser utilizado o modelo de elementos finitos criado na Seção A.3), o arquivo do modelo de elementos finitos salvo no diretório "`A02 dat base`" deve ser copiado no mesmo diretório e renomeado (é sugerido manter o mesmo nome do arquivo original com adição do sufixo "_codes_-inspection"). Esse arquivo deve ser editado da seguinte forma:

1. Todos os dados das geometrias construídas em elementos de casca do bloco "`/com,********** Send Sheet Properties **********`" devem ser deletados.
2. Todos os dados dos elementos de vigas do bloco "`/com,********** Send Beam Properties **********`" devem ser deletados.

Notas:

1. Apenas os dados das geometrias construídas em elementos de casca do bloco devem ser deletados, a linha com dado de inicialização do bloco deve ser mantida.
2. Apenas os dados dos elementos de vigas do bloco devem ser deletados, a linha com dado de inicialização do bloco deve ser mantida.

Após todas as alterações manuais, o arquivo deve ser salvo no diretório "`A02 dat base`".

B. Apêndice - Manual de utilização e guia de interface do usuário (GUI)

A manipulação de arquivos de configurações da operação (os quais controlam os sistemas acoplados, cenários de operação e resultados requeridos do DSS) e a execução dos algoritmos do Gêmeo Digital pode ser feita de duas maneiras:

1. Edição e execução manual: editando os arquivos manualmente; e, em seguida executando cada um dos algoritmos na ordem conveniente.
2. Edição e execução assistida: editando os arquivos com auxílio da Guia de Interface do Usuário (*Graphical User Interface* - GUI); e, em seguida executando cada um dos algoritmos usando a GUI na ordem conveniente ou permitindo que a GUI execute os algoritmos de maneira automática.

A segunda maneira é a mais recomendada, uma vez que facilita a compreensão de cada uma das etapas da metodologia criada. Além disso, dessa maneira é possível manter o Gêmeo Digital operando de forma automatizada, restando mais tempo para o usuário analisar os resultados gerados pelo DSS. A primeira maneira requer discernimento de cada etapa da metodologia e funcionamento dos algoritmos pelo usuário. Devido a esses fatores, os arquivos de configurações e a execução dos algoritmos do Gêmeo Digital são descritos nas próximas seções deste apêndice seguindo o fluxo de trabalho da GUI.

A GUI permite salvar ou carregar os dados em qualquer uma de suas em 5 abas, as quais compreendem as etapas do fluxo de trabalho necessário para operação do Gêmeo Digital. Nas próximas subseções são descritos quais arquivos, variáveis e algoritmos são manipulados por cada uma dessas etapas.

B.1. Aba *FE model*

Na aba *FE model* (ver Figura B.1) devem ser informados todos os arquivos de modelos criados, diretório dos arquivos e alguns dados do modelo em elementos finitos.

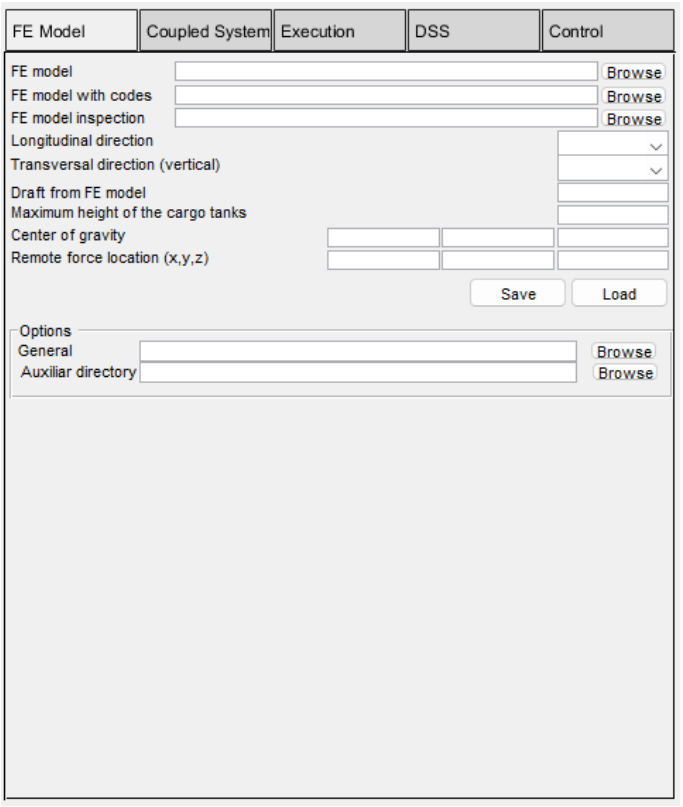


Figura B.1.: Aba de configuração de informação dos modelos em elementos finitos.

Os campos e dados esperados da aba *FE model* são descritos abaixo:

1. FE model: caminho do arquivo de modelo de elementos finitos (por padrão dentro do diretório: "A02 dat base") criado na Seção A.3 é atribuído à variável "`FemFile`" do tipo *char*;
2. FE model with codes: caminho do arquivo de modelo de elementos finitos (por padrão dentro do diretório: "A02 dat base") criado na Seção A.4 é atribuído à variável "`FemFileCodes`" do tipo *char*;
3. FE model inspection: caminho do arquivo de modelo de elementos finitos (por padrão dentro do diretório: "A02 dat base") criado na Seção A.5 é atribuído à variável "`FemFileInspection`" do tipo *char*;

4. Longitudinal direction: especifica a direção longitudinal do modelo de elementos finitos entre as 3 disponíveis: X, Y ou Z para ser atribuído à variável "`Direction`" do tipo *char*;
5. Transversal direction (vertical): especifica a direção transversal do modelo de elementos finitos (no sentido vertical - altura) entre as 3 disponíveis: X, Y ou Z para ser atribuído à variável "`Direction_transv_vertical`" do tipo *char*;
6. Draft from FE model: a distância do plano ortogonal em relação ao eixo transversal (vertical) até o fundo do casco do modelo (ver exemplos em Figura B.2 e Figura B.3) para ser atribuída a variável "`DraftModel`" do tipo *double*;

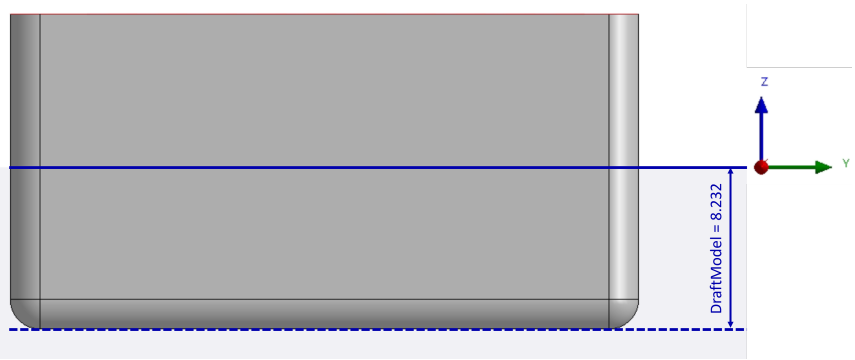


Figura B.2.: Exemplo de modelo com variável `DraftModel` = 8.232.

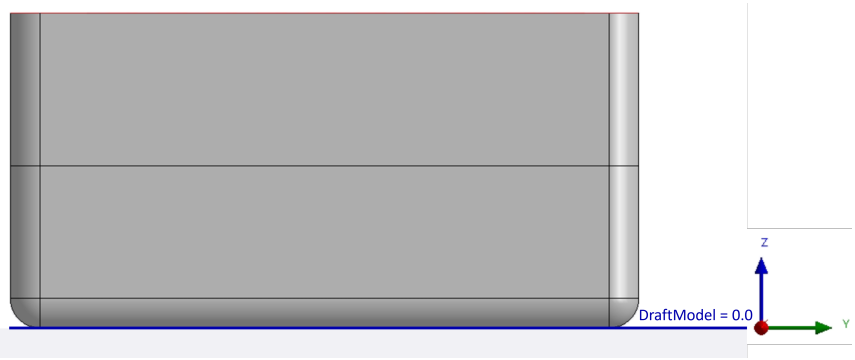


Figura B.3.: Exemplo de modelo com variável `DraftModel` = 0.0.

7. Maximum height of the cargo tanks: nível (altura) máximo que pode ser atingido pelos tanques de carga para ser atribuída a variável "`HTank`" do tipo *double*;
8. Center of gravity: as coordenadas X, Y e Z do centro de gravidade da embarcação, para serem atribuídas, respectivamente, às variáveis "`cgloc_X`", "`cgloc_Y`" e "`cgloc_Z`" do tipo *double*;

9. Remote force location: as coordenadas X, Y e Z da força remota aplicada à embarcação, para serem atribuídas, respectivamente, às variáveis "`floc_X`", "`floc_Y`" e "`floc_Z`" do tipo *double*;
10. General: caminho do diretório dos modelos (por padrão dentro do diretório: "A02 dat base") é atribuído à variável "`FolderFem`" do tipo *char*;
11. Auxiliar directory: caminho do diretório auxiliar (utilizado para armazenar varáveis estáticas durante a rotina do Gêmeo Digital) é atribuído à variável "`FolderLock`" do tipo *char*.

B.2. Aba *Coupled system*

A aba *Coupled system* (ver Figura B.4) gera o arquivo "Options.txt". O caminho deste arquivo (por padrão dentro do diretório: "A01 coupled systems") gerado automaticamente é atribuído à variável "`FileOpt`" do tipo *char*.

O arquivo de opções é utilizado para definir cenários e sistemas acoplados considerados na análise estrutural e a maneira como os dados serão atualizados na rotina do Gêmeo Digital de acordo com o conjunto de dados empregado (*data set*).

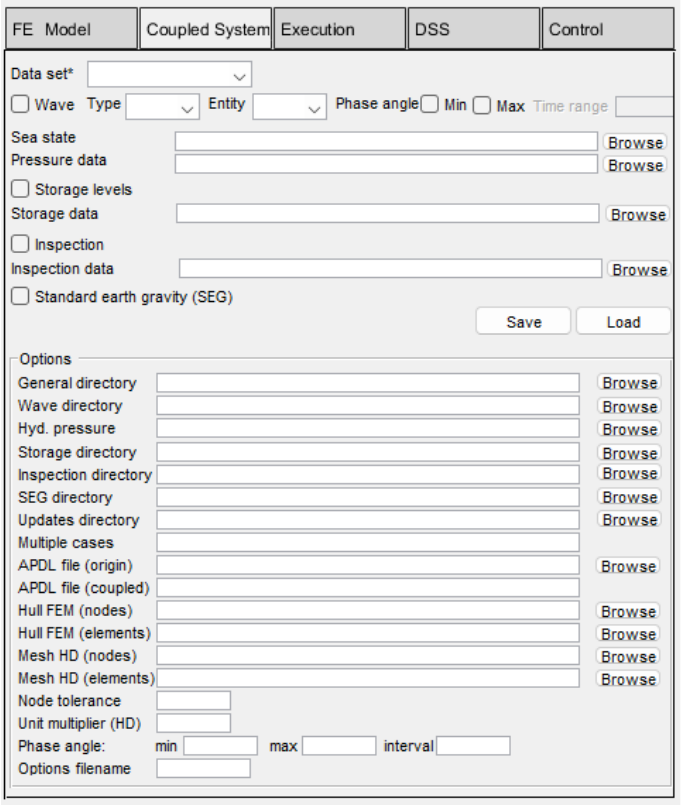


Figura B.4.: Aba de configuração de informação dos sistemas acoplados.

O arquivo de opções e suas entradas, definidos na aba *Coupled system*, esperadas são demonstrados na Tabela B.1.

Tabela B.1.: Arquivo de opções dos sistemas acoplados com suas entradas esperadas.

#	Item	Option	File
0	Data_set	Entry_01	NA
1	Wave	Entry_02	Entry_03
2	Wave_analysis	Entry_04	Entry_05
3	Min_pres_phase_angle	Entry_06	NA
4	Max_pres_phase_angle	Entry_07	NA
5	Inspection	Entry_08	Entry_09
6	Storage_levels	Entry_10	Entry_11
7	Standard_earth_gravity	Entry_12	NA

As entradas definidas pelos campos e dados esperados da aba *Coupled system* são descritos abaixo:

`Entry_01`: a maneira como os dados serão atualizados de acordo com o conjunto de dados empregado (*data set*). As opções são: `Wave`, `Storage_levels` ou `Inspection`.

É definido na GUI pelo menu de seleção "Data set*".

`Entry_02`: classificação dos dados de onda. As opções são: `off`, `regular` ou `irregular`. É habilitado na GUI pela caixa de seleção "Wave" e definido pelo menu de seleção "Type".

`Entry_03`: a extensão do arquivo utilizado para os dados de onda. As opções são: `NA` ou `*.mat`. Esta opção deve ser `*.mat` caso sejam selecionadas as opções `regular` ou `irregular` para `Entry_02`. Se selecionada a opção `off` para `Entry_02`, esta opção deve ser `NA`. É definido automaticamente pela GUI de acordo com a opção selecionada para a caixa de seleção "Wave".

`Entry_04`: a referência da entidade usada na análise de pressões hidrodinâmicas. As opções são `NODE` (para quando os valores de pressão hidrodinâmica sejam referentes aos nós) ou `ELEMENT` (para quando os valores de pressão hidrodinâmica sejam referentes aos elementos). É definido na GUI pela caixa de seleção "Entity".

`Entry_05`: a extensão do arquivo utilizado para os dados da análise hidrodinâmica. Esta opção deve ser `*.mat` caso seja selecionada a opção `WAMIT` para `Entry_04`. É definido automaticamente pela GUI de acordo com a opção selecionada para a caixa de seleção "Wave".

`Entry_06`: especifica se será criado um caso de simulação relativo ao ângulo de fase onde será identificado o menor valor de pressão. As opções são: `TRUE` ou `FALSE`. É definido na GUI pela caixa de seleção "Min" após a legenda "Phase angle".

`Entry_07`: especifica se será criado um caso de simulação relativo ao ângulo de fase onde será identificado o maior valor de pressão. As opções são: `TRUE` ou `FALSE`. É definido na GUI pela caixa de seleção "Max" após a legenda "Phase angle".

`Entry_08`: especifica se será utilizado arquivo com dados de inspeção para atualização do modelo. As opções são: `TRUE` ou `FALSE`. É definido na GUI pela caixa de seleção "Inspection".

`Entry_09`: a extensão do arquivo utilizado para o arquivo de inspeção. Esta opção deve ser `*.txt` caso seja selecionada a opção `TRUE` para `Entry_08`. Caso seja selecionada a opção `FALSE` para `Entry_08`, esta opção deve ser `NA`. É definido automaticamente pela GUI de acordo com a opção selecionada na caixa de seleção "Inspection".

`Entry_10`: especifica se será utilizado arquivo com dados do nível dos tanques para atualização do modelo. As opções são: `TRUE` ou `FALSE`. É definido na GUI pela caixa de seleção "Storage levels".

`Entry_11`: a extensão do arquivo utilizado para o arquivo de nível dos tanques. Esta opção deve ser `*.txt` caso seja selecionada a opção `TRUE` para `Entry_10`. Caso seja selecionada a opção `FALSE` para `Entry_10`, esta opção deve ser `NA`. É definido automaticamente pela GUI de acordo com a opção selecionada na caixa de seleção "Storage levels".

`Entry_12`: especifica se será considerado peso próprio da estrutura. As opções são: `TRUE` ou `FALSE`. É definido na GUI pela caixa de seleção "Standard earth gravity (SEG)".

Notas:

1. A maneira como o Gêmeo Digital opera é, substancialmente definida pela entrada `Entry_01` do arquivo de opções.
2. A utilização das opções `regular` ou `irregular` para a entrada `Entry_02` implica na utilização de um arquivo com cargas hidrodinâmicas pré-calculadas (ver Subseção B.2.1).
3. Todas as entradas devem ser separadas por um caractere de espaço ou tabulação, sendo essa a única função desse tipo de caractere no arquivo de opções.
4. Entradas que não são necessárias devem ser nomeadas como `NA` (não aplicável).
5. Se selecionada a opção `off` para `Entry_02`, as entradas `Entry_04` e `Entry_05` são negligenciadas.
6. Não se deve selecionar a opção `FALSE` para as entradas `Entry_06` e `Entry_07` ao mesmo tempo. Caso isso ocorra, uma dessas entradas será automaticamente alterada para `TRUE`.

Um exemplo do arquivo de opções é apresentado na Tabela B.2.

Tabela B.2.: Exemplo de arquivo de opções dos sistemas acoplados.

#	Item	Option	File
0	Data_set	Wave	NA
1	Wave	irregular	*.mat
2	Wave_analysis	WAMIT	*.mat
3	Min_pres_phase_angle	TRUE	NA
4	Max_pres_phase_angle	TRUE	NA
5	Inspection	TRUE	*.txt
6	Storage_levels	TRUE	*.txt
7	Standard_earth_gravity	TRUE	NA

B.2.1. Dados de cargas hidrodinâmicas

Os dados de cargas hidrodinâmicas previamente calculados por outro programa precisa seguir o formato e padrão de arquivo descritos nessa seção.

O arquivo com dados de cargas hidrodinâmicas, é utilizado para atualizar os valores de pressão hidrodinâmica na superfície do casco do modelo de elementos finitos de acordo um certo estado de mar advindo de uma onda regular ou irregular.

Este arquivo deve ter a extensão ".mat" em *structure array* e tem por padrão sua localização definida dentro do diretório: "A01 coupled systems\Wave\HD". É definido na GUI pelo campo "Pressure data" habilitado pela caixa de seleção "Wave".

O arquivo é subdividido com as seguintes variáveis:

1. Nome da variável: `Verticex`
   Tipo de variável: *double array*
   Informação das três coordenadas de cada nó da malha da região da superfície do casco.
   Dimensão: número de nós (linhas) x 3 (colunas)

2. Nome da variável: `Faces`
   Tipo de variável: *double array*
   Informação das faces (elementos ou painéis) com a identificação das coordenadas usadas.
   Dimensão: número de elementos (linhas) x número de nós utilizados pelo elemento (colunas)

3. Nome da variável: `FaceVertexCData`
   Tipo de variável: *cell array*
   Informação dos valores de pressão hidrodinâmica por nó ou elemento (de acordo com o definido pela entrada `Entry_04` do arquivo de opções (ver Tabela B.1).
   Dimensão: número de períodos (linhas) x número de incidências (colunas)

4. Nome da variável: `Hs`
   Tipo de variável: *double*
   Informação da altura de onda relativo ao processamento dos dados de pressão hidrodinâmica.

5. Nome da variável: `Incidence`
   Tipo de variável: *double array*

Informação dos valores de incidência de onda relativo ao processamento dos dados de pressão hidrodinâmica.
Dimensão: número de incidências (linhas) x 1 (coluna)

6. Nome da variável: `Period`
   Tipo de variável: *double array*
   Informação dos valores de períodos de onda relativo ao processamento dos dados de pressão hidrodinâmica.
   Dimensão: número de períodos (linhas) x 1 (coluna)

A Figura B.5 exemplifica o arquivo de pressões.

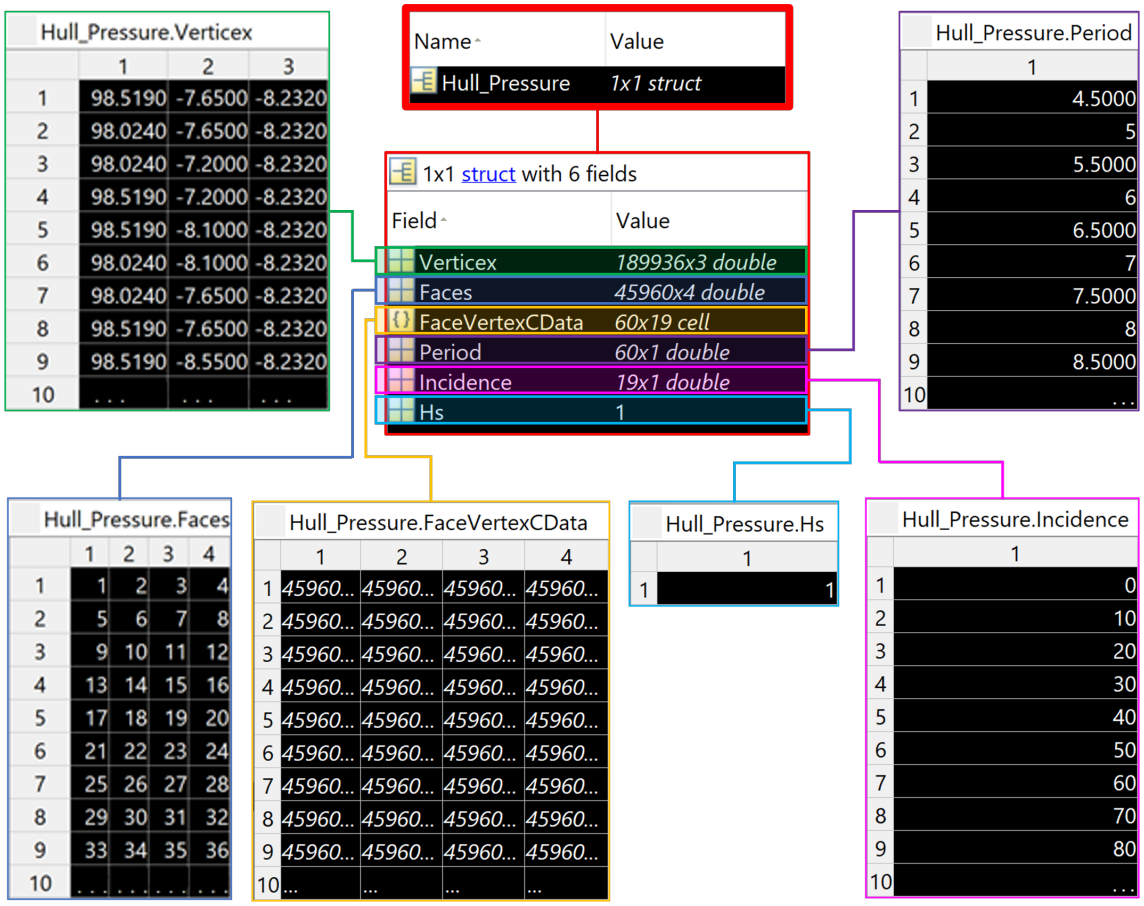


Figura B.5.: Exemplo de arquivo de pressões hidrodinâmicas e sua estrutura.

Arquivos de extensão ".txt" (por padrão dentro do diretório: "A01 coupled systems\Wave\HD") contendo as informações dos nós e elementos das malhas dos modelos em elementos finitos e hidrodinâmicos (somente da superfície do casco) são requeridos para execução do algoritmo definido em Subseção C.1.9. Caso a variável

`FaceVertexCData` forneça o valor de pressão por nós, é dispensado o arquivo com informação dos elementos da malha hidrodinâmica.

O arquivo com informações dos elementos da malha de elementos finitos e entradas esperadas é demonstrado na Tabela B.3:

Tabela B.3.: Arquivo com informações dos elementos da malha de elementos finitos e entradas esperadas.

Element Number	Element Type	Nodes			
Entry_01	Entry_02	Entry_03	Entry_04	Entry_05	Entry_06

`Entry_01`: ID do elemento.

`Entry_02`: Tipo do elemento.

`Entry_03`: ID do nó 1.

`Entry_04`: ID do nó 2.

`Entry_05`: ID do nó 3.

`Entry_06`: ID do nó 4.

Notas:

1. Novas linhas devem ser adicionadas para a informação dos elementos seguindo o disposto nas entradas `Entry_01:06`.

Um exemplo de arquivo com informações dos elementos da malha de elementos finitos e entradas esperadas é apresentado na Tabela B.4.

Tabela B.4.: Arquivo com informações dos elementos da malha de elementos finitos e entradas esperadas.

Element Number	Element Type	Nodes			
1	Quad4	12216	12217	203022	203021
2	Quad4	12101	203022	12217	12100
3	Quad4	203021	203020	12215	12216
4	Quad4	203020	203019	12214	12215

O arquivo com informações dos elementos da malha do modelo hidrodinâmico e entradas esperadas é demonstrado na Tabela B.5:

Tabela B.5.: Arquivo com informações dos elementos da malha do modelo hidrodinâmico e entradas esperadas.

Entry_01	Entry_02	Entry_03	Entry_04

`Entry_01`: ID do nó 1.

`Entry_02`: ID do nó 2.

`Entry_03`: ID do nó 3.

`Entry_04`: ID do nó 4.

Notas:

1. Novas linhas devem ser adicionadas para a informação dos elementos seguindo o disposto nas entradas `Entry_01:04`.
2. Cada elemento pode ser criado com nós únicos e exclusivos. Assim, a malha do modelo hidrodinâmico pode conter nós duplicados.
3. Na malha do modelo hidrodinâmico o ID do elemento é definido pelo índice da linha que o elemento se encontra.

Um exemplo de arquivo com informações dos elementos da malha do modelo hidrodinâmico e entradas esperadas é apresentado na Tabela B.6.

Tabela B.6.: Exemplo de arquivo com informações dos elementos da malha do modelo hidrodinâmico e entradas esperadas.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

O arquivo com informações dos nós da malha de elementos finitos e entradas esperadas é demonstrado na Tabela B.7:

Tabela B.7.: Arquivo com informações dos nós da malha de elementos finitos e entradas esperadas.

Node Number	X Location	Y Location	Z Location
Entry_01	Entry_02	Entry_03	Entry_04

`Entry_01`: ID do nó.

`Entry_02`: Coordenada X.

`Entry_03`: Coordenada Y.

`Entry_04`: Coordenada Z.

Notas:

1. Novas linhas devem ser adicionadas para a informação dos nós seguindo o disposto nas entradas `Entry_01:04`.

Um exemplo de arquivo com informações dos nós da malha de elementos finitos e entradas esperadas é apresentado na Tabela B.8.

Tabela B.8.: Exemplo de Arquivo com informações dos nós da malha de elementos finitos e entradas esperadas.

Node Number	X Location	Y Location	Z Location
1	37.5	16.	0.
2	37.5	16.	-0.802
3	36.5	16.	-0.802
4	36.5	16.	0.

O arquivo com informações dos nós da malha do modelo hidrodinâmico e entradas esperadas é demonstrado na Tabela B.9:

Tabela B.9.: Arquivo com informações dos nós da malha do modelo hidrodinâmico e entradas esperadas.

Entry_01	Entry_02	Entry_03

`Entry_01`: Coordenada X.

`Entry_02`: Coordenada Y.

`Entry_03`: Coordenada Z.

Notas:

1. Novas linhas devem ser adicionadas para a informação dos nós seguindo o disposto nas entradas `Entry_01:03`.
2. A malha do modelo hidrodinâmico pode conter nós duplicados.
3. Na malha do modelo hidrodinâmico o ID do nó é definido pelo índice da linha que o nó se encontra.

Um exemplo de arquivo com informações dos nós da malha do modelo hidrodinâmico e entradas esperadas é apresentado na Tabela B.10.

Tabela B.10.: Exemplo de arquivo com informações dos nós da malha do modelo hidrodinâmico e entradas esperadas.

68.125	-1.853	-14.27
68.125	-4.601	-14.27
68.125	-5.517	-14.27
68.125	-6.433	-14.27

B.2.1.1. Dados de onda regular

O arquivo de atualização com dados de onda regular deve ter a extensão ".mat" em *string array* e tem por padrão sua localização definida dentro do diretório: "A01 coupled systems\Wave". É definido na GUI pelo campo "Sea state" habilitado pela caixa de seleção "Wave"e é atribuído à variável "`FileWaveReg`" do tipo *char*.

O arquivo com dados de onda regular e suas entradas esperadas são demonstrados na Tabela B.11:

Tabela B.11.: Arquivo com dados de onda regular e entradas esperadas.

Time	Entry_01	...continued
Amplitude	Entry_02	...continued
Period	Entry_03	...continued
WaveDirection	Entry_04	...continued
acel_X	Entry_05	...continued
acel_Y	Entry_06	...continued
acel_Z	Entry_07	...continued
dcgomg_X	Entry_08	...continued
dcgomg_Y	Entry_09	...continued
dcgomg_Z	Entry_10	...continued
cgomga_X	Entry_11	...continued
cgomga_Y	Entry_12	...continued
cgomga_Z	Entry_13	...continued
omega_X	Entry_14	...continued
omega_Y	Entry_15	...continued
omega_Z	Entry_16	...continued
domega_X	Entry_17	...continued
domega_Y	Entry_18	...continued
domega_Z	Entry_19	...continued
FX	Entry_20	...continued
FY	Entry_21	...continued
FZ	Entry_22	...continued

`Entry_01`: tempo em segundos (t_i) da verificação dos dados (deve ser único).

`Entry_02`: dado da amplitude de onda em relação ao tempo (t_i).

`Entry_03`: dado do período de onda em relação ao tempo (t_i).

`Entry_04`: dado de direção de onda em relação ao tempo (t_i).

`Entry_05`: componente X de aceleração linear do sistema global de coordenadas em relação ao tempo (t_i).

`Entry_06`: componente Y de aceleração linear do sistema global de coordenadas em relação ao tempo (t_i).

`Entry_07`: componente Z de aceleração linear do sistema global de coordenadas em relação ao tempo (t_i).

`Entry_08`: componente X de aceleração angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_09`: componente Y de aceleração angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_10`: componente Z de aceleração angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_11`: componente X de velocidade angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_12`: componente Y de velocidade angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_13`: componente Z de velocidade angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_14`: componente X de aceleração angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_15`: componente Y de aceleração angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_16`: componente Z de aceleração angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_17`: componente X de velocidade angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_18`: componente Y de velocidade angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_19`: componente Z de velocidade angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_20`: componente X utilizado para aplicação de força remota no modelo em relação ao tempo (t_i).

`Entry_21`: componente Y utilizado para aplicação de força remota no modelo em relação ao tempo (t_i).

`Entry_22`: componente Z utilizado para aplicação de força remota no modelo em relação ao tempo (t_i).

Notas:

1. Caso não seja utilizada a opção `Wave` para a entrada `Entry_01` e `regular` para a entrada `Entry_02` do arquivo de opções (ver Tabela B.1). O arquivo de dados de onda regular precisa conter apenas duas colunas para atualização do modelo (ver exemplo apresentado na Tabela B.12).
2. Caso seja utilizada a opção `Wave` para a entrada `Entry_01` e `regular` para a entrada `Entry_02` do arquivo de opções (ver Tabela B.1). A segunda coluna é negligenciada pelo código e casos de simulação são criados com os conjuntos de dados informados após a segunda coluna (ver exemplo apresentado na Tabela B.13). Neste caso, as colunas adicionadas devem seguir os mesmos critérios das entradas `Entry_01:22`.
3. Os valores utilizados nas entradas `Entry_05:07` e `Entry_20:22` devem ser expressos nas mesmas unidades utilizadas para criação do modelo de elementos finitos.

4. Os valores utilizados nas entradas `Entry_08:19` devem ser expressos em radianos/tempo^2, sendo a unidade de tempo a mesma utilizada para criação do modelo de elementos finitos.
5. Para maiores detalhes sobre as entradas `Entry_05:07`, consulte o comando *ACEL* em ANSYS, 2020.
6. Para maiores detalhes sobre as entradas `Entry_08:10`, consulte o comando *DCGOMG* em ANSYS, 2020.
7. Para maiores detalhes sobre as entradas `Entry_11:13`, consulte o comando *CGOMGA* em ANSYS, 2020.
8. Para maiores detalhes sobre as entradas `Entry_14:16`, consulte o comando *OMEGA* em ANSYS, 2020.
9. Para maiores detalhes sobre as entradas `Entry_17:19`, consulte o comando *DOMEGA* em ANSYS, 2020.
10. É suportada apenas uma força remota no modelo.

Exemplos de arquivos com dados de onda regular são apresentados nas Tabela B.12 e Tabela B.13.

Tabela B.12.: Exemplo de arquivo com dados de onda regular e suas entradas esperadas sem utilizar as opções ***Wave*** e ***regular*** , respectivamente, como entradas `Entry_01` e `Entry_02` do arquivo de opções.

```
Time            0          ...continued
Amplitude       1          ...continued
Period          10         ...continued
WaveDirection   0          ...continued
acel_X          -0.04267   ...continued
acel_Y          -0.00012   ...continued
acel_Z          0.43143    ...continued
dcgomg_X        0.00128    ...continued
dcgomg_Y        0.00900    ...continued
dcgomg_Z        0.00129    ...continued
omega_X         0          ...continued
omega_Y         0          ...continued
omega_Z         0          ...continued
domega_X        0          ...continued
domega_Y        0          ...continued
domega_Z        0          ...continued
cgomga_X        0          ...continued
cgomga_Y        0          ...continued
cgomga_Z        0          ...continued
FX              57998.8    ...continued
FY              647.4      ...continued
FZ              -5689488   ...continued
```

Tabela B.13.: Exemplo de arquivo com dados de onda regular e suas entradas esperadas utilizando as opções ***Wave*** e ***regular*** , respectivamente, como entradas `Entry_01` e `Entry_02` do arquivo de opções.

Time	0	1	2	...continued
Amplitude	1	1.2	0.9	...continued
Period	10	9.5	11	...continued
WaveDirection	0	2	17	...continued
acel_X	-0.04267	-0.04693	-0.05163	...continued
acel_Y	-0.00012	-0.00013	-0.00015	...continued
acel_Z	0.43143	0.47457	0.52203	...continued
dcgomg_X	-0.00128	-0.00140	-0.00154	...continued
dcgomg_Y	-0.00089	-0.00098	-0.00108	...continued
dcgomg_Z	0.01294	0.01423	0.01566	...continued
omega_X	0	0	0	...continued
omega_Y	0	0	0	...continued
omega_Z	0	0	0	...continued
domega_X	0	0	0	...continued
domega_Y	0	0	0	...continued
domega_Z	0	0	0	...continued
cgomga_X	0	0	0	...continued
cgomga_Y	0	0	0	...continued
cgomga_Z	0	0	0	...continued
FX	57998.8	63798.68	70178.548	...continued
FY	647.4	712.14	783.354	...continued
FZ	-5689488	-6258436.8	-6884280.48	...continued

B.2.1.2. Dados de onda irregular

O arquivo de atualização com dados de onda irregular deve ter a extensão ".mat" em *string array* e tem por padrão sua localização definida dentro do diretório: "A01 coupled systems\Wave". É definido na GUI pelo campo "Sea state" habilitado pela caixa de seleção "Wave"e é atribuído à variável "`FileWaveIrr`" do tipo *char*.

Este tipo de arquivo pode ser utilizado apenas com a combinação das opções `Wave` para a entrada `Entry_01` e `irregular` para a entrada `Entry_02` do arquivo de opções (ver Tabela B.1).

Essa combinação também requer o intervalo de tempo para divisão da onda irregular informada em trechos. É definido na GUI pelo campo "Time range" habilitado pela caixa de seleção "Wave" e seleção do menu "Type" como "irregular", seu valor é atribuído à variável "`DivTimeIrrWave`" do tipo *double*.

Notas:

1. Este valor de tempo deve ser menor ou igual ao maior valor de tempo (t_i) informado no arquivo de onda irregular.
2. Caso desejado que a onda irregular seja analisada em um único trecho, informar o valor máximo de tempo (t_i) para a variável "`DivTimeIrrWave`".

A variável "`DivTimeIrrWave`" deve conter o intervalo de tempo para divisão da onda irregular informada em trechos.

O arquivo com dados de onda regular e suas entradas esperadas são demonstrados na Tabela B.14:

Tabela B.14.: Arquivo com dados de onda irregular e entradas esperadas.

`Time`	`Entry_01`	`...continued`
`SeaElevation`	`Entry_02`	`...continued`
`WaveDirection`	`Entry_03`	`...continued`
`acel_X`	`Entry_04`	`...continued`
`acel_Y`	`Entry_05`	`...continued`
`acel_Z`	`Entry_06`	`...continued`
`dcgomg_X`	`Entry_07`	`...continued`
`dcgomg_Y`	`Entry_08`	`...continued`
`dcgomg_Z`	`Entry_09`	`...continued`
`omega_X`	`Entry_10`	`...continued`
`omega_Y`	`Entry_11`	`...continued`
`omega_Z`	`Entry_12`	`...continued`
`domega_X`	`Entry_13`	`...continued`
`domega_Y`	`Entry_14`	`...continued`
`domega_Z`	`Entry_15`	`...continued`
`cgomga_X`	`Entry_16`	`...continued`
`cgomga_Y`	`Entry_17`	`...continued`
`cgomga_Z`	`Entry_18`	`...continued`
`Entry_22`	`Entry_19`	`...continued`
`Entry_23`	`Entry_20`	`...continued`
`Entry_24`	`Entry_21`	`...continued`
`...continued`	`...continued`	`...continued`

`Entry_01`: tempo (t_i) em segundos da verificação dos dados (deve ser único).

`Entry_02`: dado da elevação do mar em relação ao tempo (t_i).

`Entry_03`: dado de direção de onda em relação ao tempo (t_i).

`Entry_04`: componente X de aceleração linear do sistema global de coordenadas em relação ao tempo (t_i).

`Entry_05`: componente Y de aceleração linear do sistema global de coordenadas em relação ao tempo (t_i).

`Entry_06`: componente Z de aceleração linear do sistema global de coordenadas em relação ao tempo (t_i).

`Entry_07`: componente X de aceleração angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_08`: componente Y de aceleração angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_09`: componente Z de aceleração angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_10`: componente X de velocidade angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_11`: componente Y de velocidade angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_12`: componente Z de velocidade angular da origem global sobre cada um dos eixos do sistema de coordenadas de aceleração (centro de gravidade da embarcação) em relação ao tempo (t_i).

`Entry_13`: componente X de aceleração angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_14`: componente Y de aceleração angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_15`: componente Z de aceleração angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_16`: componente X de velocidade angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_17`: componente Y de velocidade angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_18`: componente Z de velocidade angular da estrutura sobre cada um dos eixos do sistema de coordenadas global (regra da mão direita) em relação ao tempo (t_i).

`Entry_19`: componente X utilizado para aplicação de força remota no modelo em relação ao tempo (t_i).

`Entry_20`: componente Y utilizado para aplicação de força remota no modelo em relação ao tempo (t_i).

`Entry_21`: componente Z utilizado para aplicação de força remota no modelo em relação ao tempo (t_i).

`Entry_22`: especificação do número da linha de ancoragem e componente de força na direção X. Devem ser concatenados os termos: "Line"+ número da linha de ancoragem + "FX".

`Entry_23`: especificação do número da linha de ancoragem e componente de força na direção Y. Devem ser concatenados os termos: "Line"+ número da linha de ancoragem + "FY".

`Entry_24`: especificação do número da linha de ancoragem e componente de força na direção Z. Devem ser concatenados os termos: "Line"+ número da linha de ancoragem + "FZ".

Notas:

1. Os valores utilizados nas entradas `Entry_04:06` e `Entry_19:21` devem ser expressos nas mesmas unidades utilizadas para criação do modelo de elementos finitos.
2. Os valores utilizados nas entradas `Entry_07:18` devem ser expressos em radianos/tempo^2, sendo a unidade de tempo a mesma utilizada para criação do modelo de elementos finitos.
3. Para maiores detalhes sobre as entradas `Entry_04:06`, consulte o comando *ACEL* em ANSYS, 2020.
4. Para maiores detalhes sobre as entradas `Entry_07:09`, consulte o comando *DCGOMG* em ANSYS, 2020.

5. Para maiores detalhes sobre as entradas `Entry_10:12`, consulte o comando *CGOMGA* em ANSYS, 2020.
6. Para maiores detalhes sobre as entradas `Entry_13:15`, consulte o comando *OMEGA* em ANSYS, 2020.
7. Para maiores detalhes sobre as entradas `Entry_16:18`, consulte o comando *DOMEGA* em ANSYS, 2020.
8. A adição de dados na série temporal deve ser feita por meio da adição de novas colunas com observação ao que é descrito nas entradas `Entry_01:21`.
9. A adição de forças das linhas de ancoragem deve ser feita pela adição de novas linhas com observação ao que é descrito nas entradas `Entry_22:24` e `Entry_19:21`.
10. As entradas `Entry_22:24` devem ser únicas.

Um exemplo de arquivo com dados de onda irregular é apresentado na Tabela B.15.

Tabela B.15.: Exemplo de arquivo com dados de onda irregular e suas entradas esperadas.

Time	0	0.1	0.2	0.3	...continued
SeaElevation	-0.2805	-0.2309	-0.1772	-0.1209	...continued
WaveDirection	0	0	0	0	...continued
acel_X	-0.028517	-0.02803	-0.027464	-0.02682	...continued
acel_Y	0.00034727	0.00033807	0.00032828	0.00031797	...continued
acel_Z	-0.0057122	-0.01186	-0.017883	-0.02377	...continued
dcgomg_X	-0.00128	-0.0014	-0.00154	-0.00154	...continued
dcgomg_Y	-0.00089	-0.00098	-0.00108	-0.001078	...continued
dcgomg_Z	0.01294	0.01423	0.01566	0.015653	...continued
omega_X	0	0	0	0	...continued
omega_Y	0	0	0	0	...continued
omega_Z	0	0	0	0	...continued
domega_X	0	0	0	0	...continued
domega_Y	0	0	0	0	...continued
domega_Z	0	0	0	0	...continued
cgomga_X	0	0	0	0	...continued
cgomga_Y	0	0	0	0	...continued
cgomga_Z	0	0	0	0	...continued
Line1FX	-151084.4	-151075.6	-151069.5	-151066.4	...continued
Line1FY	303684.1	303731.7	303784.7	303843.1	...continued
Line1FZ	-611355.5	-611579.8	-611801.8	-612020.1	...continued
Line2FX	-125407.1	-125395.8	-125387.4	-125381.9	...continued
Line2FY	319780.3	319840.2	319906.4	319978	...continued
Line2FZ	-614837.1	-615051.5	-615264.3	-615474.4	...continued
Line3FX	-98195.4	-98184.44	-98176.55	-98171.43	...continued
Line3FY	334057.7	334139.8	334230.3	334326.9	...continued
Line3FZ	-618703.7	-618941.7	-619180	-619416.4	...continued

B.2.2. Dados do nível dos tanques

O arquivo com dados do nível dos tanques é utilizado para atualizar os valores pressão hidrostática (interna e externa) aplicado no modelo. Este arquivo é dependente de como foi construído o modelo, pois o número de tanques do modelo e os códigos utilizados para substituírem as pressões hidrostáticas no modelo (apresentados na Seção A.4) devem ser usados aqui.

O arquivo de atualização dos níveis dos tanques deve ter a extensão ".txt" e tem por padrão sua localização definida dentro do diretório: "A01 coupled systems\Storage Levels". É definido na GUI pelo campo "Storage data" habilitado pela caixa de seleção "Storage levels" e é atribuído à variável "`FileStor`" do tipo *char*;

O arquivo de opções e suas entradas esperadas são demonstrados na Tabela B.16:

Tabela B.16.: Arquivo com dados esperados para nível dos tanques.

DATE	Entry_10	...continued
Status_HSP	Entry_11	...continued
HSP_par	Entry_12	...continued
Entry_01	Entry_13	...continued
Entry_02	Entry_14	...continued
Entry_03	Entry_15	...continued
Entry_04	Entry_16	...continued
Entry_05	Entry_17	...continued
Entry_06	Entry_18	...continued
Entry_07	Entry_19	...continued
Entry_08	Entry_20	...continued
Entry_09	Entry_21	...continued
...continued	...continued	...continued

`Entry_01`: código utilizado para pressão hidrostática externa ao modelo.

`Entry_02`: código utilizado para pressão hidrostática interna do tanque (T_i).

`Entry_03`: código utilizado para coordenada X do centro de massa do tanque (T_i).

`Entry_04`: código utilizado para coordenada Y do centro de massa do tanque (T_i).

`Entry_05`: código utilizado para coordenada Z do centro de massa do tanque (T_i).

`Entry_06`: código utilizado para valor da massa do tanque (T_i).

`Entry_07`: código utilizado para valor do momento de inércia no eixo X da massa do tanque (T_i).

`Entry_08`: código utilizado para valor do momento de inércia no eixo Y da massa do tanque (T_i).

`Entry_09`: código utilizado para valor do momento de inércia no eixo Z da massa do tanque (T_i).

`Entry_10`: data da verificação ou nome dado ao modelo (deve ser único).

`Entry_11`: entrada válida apenas para quando for utilizada a opção `StorageLevels` para a entrada `Entry_01` do arquivo de opções (ver Tabela B.1). As opções são: `TRUE` para utilizar o conjunto de dados informado na coluna para se criar um novo caso de simulação ou `FALSE` para ignorar o conjunto de dados da coluna. Quando não for utilizada a opção `StorageLevels` para a entrada `Entry_01` do arquivo de opções, esta opção deve ser preenchida apenas com um hífen: `-`.

`Entry_12`: número total de entradas relacionadas ao nível dos tanques a serem modificadas no modelo (quantidade de variáveis a ser contada da linha posterior até o final da coluna).

`Entry_13`: valor em porcentagem da altura do calado em relação à altura máxima do nível dos tanques (informada na Seção B.1).

`Entry_14`: valor em porcentagem do nível do tanque (T_i) em relação à altura máxima do nível dos tanques (informada na Seção B.1).

`Entry_15`: valor de atualização para coordenada X do centro de massa do tanque (T_i).

`Entry_16`: valor de atualização para coordenada Y do centro de massa do tanque (T_i).

`Entry_17`: valor de atualização para coordenada Z do centro de massa do tanque (T_i).

`Entry_18`: valor de atualização para valor da massa do tanque (T_i).

`Entry_19`: valor de atualização para valor do momento de inércia no eixo X da massa do tanque (T_i).

`Entry_20`: valor de atualização para valor do momento de inércia no eixo Y da massa do tanque (T_i).

`Entry_21`: valor de atualização para valor do momento de inércia no eixo Z da massa do tanque (T_i).

Notas:

1. Todas as entradas devem ser separadas por um caractere de espaço ou tabulação.
2. Caso não seja utilizada a opção `StorageLevels` para a entrada `Entry_01` do arquivo de opções (ver Tabela B.1). O arquivo de nível dos tanques precisa conter apenas duas colunas para atualização do modelo (ver exemplo apresentado na Tabela B.17).
3. Caso seja utilizada a opção `StorageLevels` para a entrada `Entry_01` do arquivo de opções (ver Tabela B.1). A segunda coluna é negligenciada pelo código e casos de simulação são criados com os conjuntos de dados informados após a segunda coluna (ver exemplo apresentado na Tabela B.18). Neste

caso, as colunas adicionadas devem seguir os mesmos critérios das entradas `Entry_10:21`.

4. Os valores de entradas relacionados a coordenadas (`Entry_15:17`), massa (`Entry_18`) ou momentos de inércia (`Entry_19:21`) devem ser expressos na mesma unidade utilizada para criação do modelo de elementos finitos.
5. Os nomes atribuídos à entrada `Entry_10` devem ser únicos quando utilizados após a segunda coluna.
6. Caso o modelo de elementos finitos contenha mais de um tanque, as entradas `Entry_02` e `Entry_14` devem ser repetidas com as informações dos tanques adicionais nas linhas em sequência até o número total de tanques definidos no modelo. Somente após descrever a informação de todos os tanques, deve ser dada procedência para as informações das entradas `Entry_03:09` e `Entry_-15:21`.
7. Caso o modelo de elementos finitos contenha mais de um tanque. As entradas `Entry_03:09` e `Entry_15:21` devem ser repetidas com as informações dos tanques adicionais nas linhas em sequência até o número total de tanques definidos no modelo.
8. As entradas `Entry_13` e `Entry_14` devem ser especificadas em porcentagem em relação à altura máxima do nível dos tanques (informada na Seção B.1). Por exemplo: se a altura do calado ou nível do tanque é de 9 metros e altura máxima do nível dos tanques informada foi 10 metros, o valor especificado deve ser 90.
9. Antes do primeiro ciclo de execução do Gêmeo Digital, o algoritmo apresentado na Subseção C.1.10 deve ser executado para a criação do arquivo auxiliar '- StorageLevelsOFF.txt' a partir do arquivo de dados para nível dos tanques.

Exemplos de arquivos de nível dos tanques são apresentados nas Tabela B.17 e Tabela B.18.

Tabela B.17.: Exemplo de arquivo de nível com 2 tanques sem utilizar opção *StorageLevels* como entrada Entry_01 do arquivo de opções.

Date	Current
Status_HSP	-
HSP_par	17
HSP_Zplan	51
HSP_T01_Zplan	70
HSP_T02_Zplan	70
Value_T1Xloc	-42.5
Value_T1Yloc	0
Value_T1Zloc	2.46
Value_T1_M	5600000
Value_T1IXX	536405333.3
Value_T1IYY	350205333.3
Value_T1IZZ	769533333.3
Value_T2Xloc	-25
Value_T2Yloc	0
Value_T2Zloc	2.46
Value_T2_M	5600000
Value_T2IXX	536405333.3
Value_T2IYY	350205333.3
Value_T2IZZ	769533333.3

Tabela B.18.: Exemplo de arquivo de nível com 2 tanques utilizando opção *StorageLevels* como entrada Entry_01 do arquivo de opções.

Date	Current	2020Apr24_06:45:07	2020Apr24_06:45:07
Status_HSP	-	TRUE	TRUE
HSP_par	17	17	17
HSP_Zplan	51	56.1	61.71
HSP_T01_Zplan	70	77	84.7
HSP_T02_Zplan	70	77	84.7
Value_T1Xloc	-42.5	-46.8	-51.4
Value_T1Yloc	0	0	0
Value_T1Zloc	2.46	2.7	2.97
Value_T1_M	5600000	6160000	6776000
Value_T1IXX	536405333.3	590045866.6	649050453.3
Value_T1IYY	350205333.3	385225866.6	423748453.3
Value_T1IZZ	769533333.3	846486666.6	931135333.3
Value_T2Xloc	-25	-27.5	-30.3
Value_T2Yloc	0	0	0
Value_T2Zloc	2.46	2.7	2.97
Value_T2_M	5600000	6160000	6776000
Value_T2IXX	536405333.3	590045866.6	649050453.3
Value_T2IYY	350205333.3	385225866.6	423748453.3
Value_T2IZZ	769533333.3	846486666.6	931135333.3

B.2.3. Dados de inspeção

Os arquivos com dados de inspeção são utilizados para atualizar valores de espessura de chapas e vigas do modelo. Estes arquivos são dependentes de como foi construído o modelo, pois o número entidades geométricas (chapas e vigas) e o número de parâmetros relativos ao tipo de elementos utilizado para cada entidade deve ser considerado, o que também influencia como os arquivos de inspeção deve ser nomeado (descrito pela entrada `*_Entry_01`).

`*_Entry_01`: Os arquivos de inspeção devem ser criados separadamente de acordo com o tipo de entidade geométrica utilizada e o número de parâmetros que eles necessitam. Por isso, o nome do arquivo de extensão `".txt"` deve ser concatenado com o sufixo relacionado à entrada `Entry_01` para especificar o tipo de entidade geométrica a que se refere (o tipo de entidade geométrica é exibido no arquivo de modelo de elementos finitos, alguns exemplos são: `_shell`, `_beam_rect` ou `_beam_L`).

Figura B.6.: Parâmetros de dimensão (B e H) de entidade geométrica retangular utilizado para vigas com seção retangular.

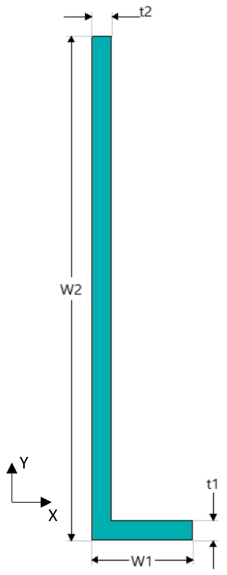


Figura B.7.: Parâmetros de dimensão ($W1$ e $W2$) e espessura ($t1$ e $t2$) utilizado para vigas com seção L.

Notas:

1. As entradas esperadas variam de acordo com o número e ordem de parâmetros requeridos para cada tipo de elemento. São suportadas atualização de valores no modelo para os tipos de entidades geométricas a seguir:
   - `_shell`: elemento de casca com um parâmetro de espessura (t1) com seção de *offset* a partir do meio da casca;
   - `_beam_rect`: elemento de viga com seção retangular utilizando parâmetro de base (B1) e altura (H1) (ver Figura B.6);
   - `_beam_L`: elemento de viga com seção L utilizando dois parâmetros de dimensão (W1 e W2), dois parâmetros de espessura (t1 e t2), um parâmetro de *offset* na direção X (O1) e um parâmetro de *offset* na direção Y (O2) (ver Figura B.7).
2. Por padrão a localização dos arquivos de inspeção é definida dentro do diretório: "A01 coupled systems\Inspection".
3. O arquivo de inspeção é definido na GUI pelo campo "Inspection data" habilitado pela caixa de seleção "Inspection" e é atribuído à variável "`FileInsp`" do tipo *char*. É necessário, e obrigatório, informar somente o arquivo com entidades geométricas de elemento de casca (`*_shell.txt`). Os outros arquivos de inspeção serão lidos automaticamente do diretório definido.

O arquivo com dados de inspeção e suas entradas esperadas é demonstrado na Tabela B.19.

Tabela B.19.: Arquivo com dados de inspeção e entradas esperadas.

```
DATE          BODY_NUMBER   Entry_05      ...continued
Entry_02      Entry_03      Entry_04      ...continued
...continued  ...continued  ...continued  ...continued
```

`Entry_02`: nome da geometria do corpo (*body*) utilizado no modelo de elementos finitos.

`Entry_03`: número da geometria do corpo (*body*) utilizado no modelo de elementos finitos.

`Entry_04`: valor para atualização do parâmetro da geometria do corpo (*body*) no modelo de elementos finitos.

`Entry_05`: data da inspeção ou nome dado ao modelo (deve ser único).

Notas:

1. Todas as entradas devem ser separadas por um caractere de espaço ou tabulação.
2. Caso não seja utilizada a opção `Inspection` para a entrada `Entry_01` do arquivo de opções (ver Tabela B.1). O arquivo de inspeção precisa conter o número de colunas suficiente para especificar os parâmetros do tipo de geometria (ver exemplo apresentado na Tabela B.20).
3. Caso seja utilizada a opção `Inspection` para a entrada `Entry_01` do arquivo de opções (ver Tabela B.1). A segunda coluna do arquivo de inspeção (para especificar o primeiro conjunto de parâmetros) é negligenciada pelo código e casos de simulação são criados com os conjuntos de dados informados após essa coluna (ver exemplo apresentado na Tabela B.21). Neste caso, as colunas adicionadas devem seguir os mesmos critérios das entradas `Entry_04` e `Entry_05`.
4. Os valores utilizados na entrada `Entry_04` devem ser expressos na mesma unidade utilizada para criação do modelo de elementos finitos.
5. Caso a geometria do corpo (*body*) necessite de mais de um parâmetro, as entradas `Entry_04` e `Entry_05` devem ser repetidas de acordo à quantidade e

sequência dos parâmetros até o número total necessário de parâmetros da entidade geométrica. Somente após descrever todos os parâmetros necessários, deve ser dada procedência, se preciso for, para adição de novas colunas contendo a atualização de parâmetros para novos casos (ver exemplo apresentado na Tabela B.21).

6. Novas linhas devem ser adicionadas observando o que é descrito para as entradas `Entry_02:04` até que a quantidade total da mesma entidade geométrica seja especificada no arquivo.

Exemplos de arquivos de inspeção para vigas de seção retangular são apresentados na Tabela B.20 e Tabela B.21.

Tabela B.20.: Exemplo de arquivo de inspeção com dados para entidades geométricas de vigas de seção retangular sem utilizar opção ***Inspection*** *para o* Data set (***Exemplo01_beam_rect.txt***).

DATE	BODY_NUMBER	CURRENT_B1	CURRENT_H1
!Beam_A	1	0.3	0.017
!Beam_B	2	0.25	0.015
!Beam_C	3	0.3	0.017
!Beam_D	4	0.3	0.017
...continued	...continued	...continued	...continued

Tabela B.21.: Exemplo de arquivo de inspeção com dados para entidades geométricas de vigas de seção retangular utilizando a opção ***Inspection*** *para o* Data set (***Exemplo02_beam_rect.txt***).

DATE	BODY_NUMBER	CURRENT_B1	CURRENT_H1	2001_B1	2001_H1	2002_B1	2002_H1
!Beam_A	1	0.3	0.017	0.3	0.017	0.29995	0.01695
!Beam_B	2	0.25	0.015	0.25	0.015	0.24995	0.01495
!Beam_C	3	0.3	0.017	0.3	0.017	0.29995	0.01695
!Beam_D	4	0.3	0.017	0.3	0.017	0.29995	0.017
...continued	...continued	...continued	...continued	...continued	...continued	...continued	...continued

B.2.4. Outras definições

A aba *Coupled system* determina outras definições que não estão diretamente relacionadas ao arquivo de opções. Estes campos e dados esperados são descritos abaixo:

1. General directory: diretório geral dos arquivos de sistemas acoplados (por padrão dentro do diretório: "A01 coupled systems") é atribuído à variável "`FolderCS`" do tipo *char*;
2. Wave directory: diretório dos arquivos com dados de ondas (por padrão dentro do diretório: "A01 coupled systems\Wave") é atribuído à variável "`FolderWave`" do tipo *char*;
3. Hyd. pressure: diretório dos arquivos com dados da análise hidrodinâmica (por padrão dentro do diretório: "A01 coupled systems\Wave \HD") é atribuído à variável "`FolderWamitProc`" do tipo *char*;
4. Storage directory: diretório dos arquivos com dados da de nível dos tanques (por padrão dentro do diretório: "A01 coupled systems\Storage Levels") é atribuído à variável "`FolderStor`" do tipo *char*;
5. Inspection directory: diretório dos arquivos com dados de inspeção (por padrão dentro do diretório: "A01 coupled systems\Inspection") é atribuído à variável "`FolderInsp`" do tipo *char*;
6. SEG directory: diretório do arquivo para estabelecer peso próprio da estrutura igual a 0 (por padrão dentro do diretório: "A01 coupled systems\Standard Earth Gravity") é atribuído à variável "`FolderSEG`" do tipo *char*;
7. Updates directory: diretório auxiliar para arquivos necessários durante atualização dos modelos (por padrão dentro do diretório: "A01 coupled systems\- Update") é atribuído à variável "`FolderUpdates`" do tipo *char*;
8. Multiple cases: nome do arquivo contendo a matrix de atualização aplicada ao modelo de elementos finitos (por padrão: "MCContent") é atribuído à variável "`MCFile`" do tipo *char*;
9. APDL file (origin): caminho do arquivo auxiliar de extensão .txt necessário durante atualização dos modelos em elementos finitos (por padrão: "A01 coupled systems\- Update\APDL_post_origin.txt") é atribuído à variável "`ApdlPostFileOri`" do tipo *char*;

10. APDL file (coupled): nome do arquivo auxiliar gerado de acordo com dados requeridos pelo DSS (por padrão: "APDL_post.txt") é atribuído à variável "`ApdlPostFile`" do tipo *char*;
11. Hull FEM (nodes): caminho do arquivo auxiliar de extensão .txt com informação dos nós do modelo de elementos finitos relativos à superfície do casco (por padrão: "A01 coupled systems\Wave\HD*.txt") é atribuído à variável "`HullNodesFile`" do tipo *char* (ver Subseção B.2.1);
12. Hull FEM (elements): caminho do arquivo auxiliar de extensão .txt com informação dos elementos da malha do modelo de elementos finitos relativos à superfície do casco (por padrão: "A01 coupled systems\Wave\HD*.txt") é atribuído à variável "`HullElementsFile`" do tipo *char* (ver Subseção B.2.1);
13. Mesh HD (nodes): caminho do arquivo auxiliar de extensão .txt com informação dos nós da malha do modelo utilizado na análise hidrodinâmica relativos à superfície do casco (por padrão: "A01 coupled systems\Wave\HD*.txt") é atribuído à variável "`WamitNodesFile`" do tipo *char* (ver Subseção B.2.1);
14. Mesh HD (elements): caminho do arquivo auxiliar de extensão .txt com informação dos elementos da malha do modelo utilizado na análise hidrodinâmica relativos à superfície do casco (por padrão: "A01 coupled systems\Wave\HD*.txt") é atribuído à variável "`WamitElementsFile`" do tipo *char* (ver Subseção B.2.1);
15. Node tolerance: valor de tolerância utilizado para para coordenadas do nós em equivalência das malhas, é atribuído à variável "`TolWamit`" do tipo *double*;
16. Unit multiplier (HD): valor utilizado como multiplicador para valores obtidos de pressão hidrodinâmica (utilizado em caso de unidades de pressão diferentes entre modelo de elementos finitos e de análise hidrodinâmica), é atribuído à variável "`Unit`" do tipo *double*;
17. Phase angle - min, max e interval: respectivamente, valores mínimo, máximo e de intervalo utilizados para encontrar o ângulo de fase onde ocorre a maior e, ou a menor pressão hidrodinâmica aplicada na superfície do casco do modelo de elementos finitos (por padrão: min = 0; max = 360; interval = 10). São atribuídos, respectivamente, às variáveis "`MinPha`", "`MaxPha`" e "`BrkPha`", todas do tipo *double*;
18. Options filename: nome do arquivo de opções gerado (por padrão: "Options") é atribuído à variável "`Options`" do tipo *char*.

B.3. Aba *Execution*

Na aba *Execution* (ver Figura B.8) devem ser informados dados relacionados ao *solver* em elementos finitos e resultados requeridos durante as análises.

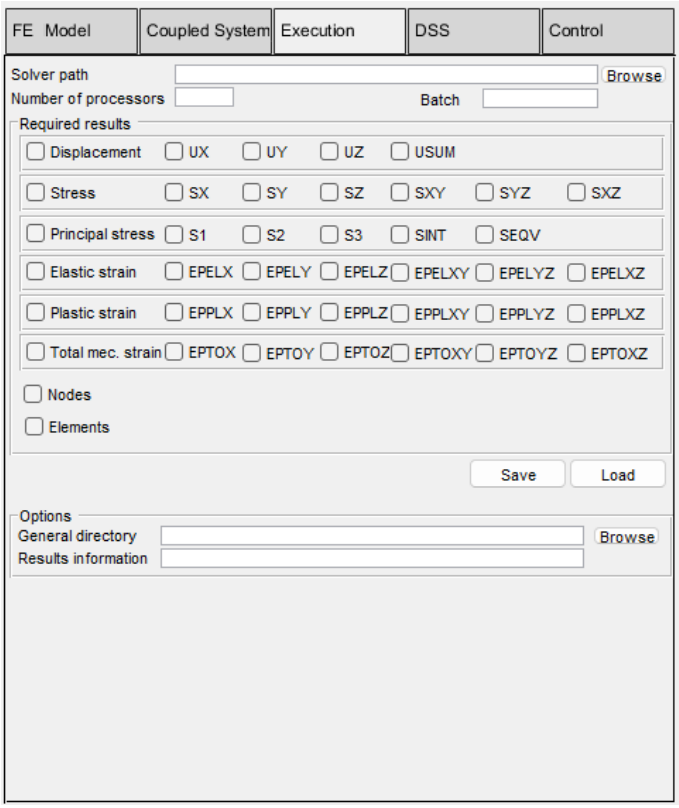


Figura B.8.: Aba de configuração de execução do *solver* em elementos finitos.

Os campos e dados esperados da aba *Execution* são descritos abaixo:

1. Solver path: caminho do *solver* em elementos finitos utilizado para as análises, é atribuído à variável "`PathAnsysSolver`" do tipo *char*;
2. Number of processors: especifica o número de processadores a serem usados ao executar o *solver*, é atribuído à variável "`np`" do tipo *double*;
3. Batch: especifica o nome do arquivo em lotes criado para ser submetido ao *solver* (por padrão: - Run_Cases"), é atribuído à variável "`FileNameBat`" do tipo *char*;
4. General directory: especifica o diretório onde os modelos em elementos finitos atualizados e suas soluções serão geradas (por padrão: "A03 assemble"), é atribuído à variável "`FolderFemCases`" do tipo *char*;

Tabela B.22.: Resultados com detalhe do nome da variável e número de componentes do resultado.

Resultado	Descrição	Variável	Componentes
Displacement	Deslocamento	`U`	4
Stress	Tensão	`S`	6
Principal	Tensão principal	`PRIN`	5
Elastic strain	Deformação elástica	`EPEL`	6
Plastic strain	Deformação plástica	`EPPL`	6
Total mec. strain	Deformação mecânica total	`EPTO`	6
Nodes	Nós do modelo	`NLIST`	-
Elements	Elementos do modelo	`ELIST`	-

5. Results information: especifica o nome do arquivo que contém todas as variáveis utilizadas para atualizar cada um dos modelos em elementos finitos (por padrão: - Results_info"), é atribuído à variável "`ResInfo`" do tipo *char*.

Notas:

1. O número máximo de processadores é dependente da licença ativa em uso do *solver*.
2. Para maiores detalhes consulte o comando em lote *-np* em ANSYS, 2020.

As variáveis que especificam os resultados requeridos durante a execução do *solver* são do tipo *logical* e podem ter as seguintes opções: "`true`" ou "`false`".

A Tabela B.22 apresenta cada um dos resultados que podem ser solicitados, o nome da variável lógica utilizada e o número de componentes para cada resultado.

A GUI atribui automaticamente os valores das variáveis de resultado de acordo com a definição feita pelo usuário nas caixas de seleção da aba *Execution*.

Notas:

1. É recomendado manter as variáveis "`NLIST`", "`ELIST`", "`EPPL`" e "`EPTO`" com a opção "`false`" para economia de tempo e memória computacional.
2. É recomendado manter as variáveis de resultados que não sejam necessárias para o sistema de apoio à decisão com a opção "`false`" para economia de tempo e memória computacional.

Vetores do tipo *double* (de dimensão 1 x número de componentes necessário para cada resultado) definem as componentes requeridas nos resultados por meio dos valores "1"e "0"para cada uma das posições do vetor.

A variável `UGmsh` define o requerimento de cada uma das componentes de resultado de deslocamento:

```
UGmsh = [UX,UY,UZ,USUM]
```

- Primeira posição: componente de deslocamento na direção X;
- Segunda posição: componente de deslocamento na direção Y;
- Terceira posição: componente de deslocamento na direção Z;
- Quarta posição: deslocamento total;

Os valores de `UGmsh` podem ser definidos por meio das caixas de seleção "UX, UY, UZ e USUM"na GUI.

A variável `SGmsh` define o requerimento de cada uma das componentes de resultado de tensão:

```
SGmsh = [SX,SY,SZ,SXY,SYZ,SXZ]
```

- Primeira posição: componente de tensão X;
- Segunda posição: componente de tensão Y;
- Terceira posição: componente de tensão Z;
- Quarta posição: componente de tensão XY;
- Quinta posição: componente de tensão YZ;
- Sexta posição: componente de tensão XZ;

Os valores de `SGmsh` podem ser definidos por meio das caixas de seleção "SX, SY, SZ, SXY, SYZ e SXZ"na GUI.

A variável `PRINGmsh` define o requerimento de cada uma das componentes de resultado de tensão principal:

```
PRINGmsh = [S1,S2,S3,SINT,SEQV]
```

- Primeira posição: tensão principal $S1$;
- Segunda posição: tensão principal $S2$;
- Terceira posição: tensão principal $S3$;
- Quarta posição: intensidade da tensão (*stress intensity*) $SINT$;
- Quinta posição: tensão equivalente $SEQV$;

Os valores de `PRINGmsh` podem ser definidos por meio das caixas de seleção "S1, S2, S3, SINT e SEQV"na GUI.

A variável `EPELGmsh` define o requerimento de cada uma das componentes de resultado de deformação elástica:

```
EPELGmsh = [EPELX,EPELY,EPELZ,EPELXY,EPELYZ,EPELXZ]
```

- Primeira posição: componente de deformação elástica X;
- Segunda posição: componente de deformação elástica Y;
- Terceira posição: componente de deformação elástica Z;
- Quarta posição: componente de deformação elástica XY;
- Quinta posição: componente de deformação elástica YZ;
- Sexta posição: componente de deformação elástica XZ;

Os valores de `SGmsh` podem ser definidos por meio das caixas de seleção "EPELX, EPELY, EPELZ, EPELXY, EPELYZ e EPELXZ"na GUI.

A variável `EPPLGmsh` define o requerimento de cada uma das componentes de resultado de deformação plástica:

```
EPPLGmsh = [EPPLX,EPPLY,EPPLZ,EPPLXY,EPPLYZ,EPPLXZ]
```

- Primeira posição: componente de deformação plástica X;
- Segunda posição: componente de deformação plástica Y;
- Terceira posição: componente de deformação plástica Z;
- Quarta posição: componente de deformação plástica XY;
- Quinta posição: componente de deformação plástica YZ;
- Sexta posição: componente de deformação plástica XZ;

Os valores de `SGmsh` podem ser definidos por meio das caixas de seleção "EPPLX, EPPLY, EPPLZ, EPPLXY, EPPLYZ e EPPLXZ"na GUI.

A variável `EPTOGmsh` define o requerimento de cada uma das componentes de resultado de deformação mecânica total:

```
EPTOGmsh = [EPTOX,EPTOY,EPTOZ,EPTOXY,EPTOYZ,EPTOXZ]
```

- Primeira posição: componente de deformação total X;

- Segunda posição: componente de deformação total Y;
- Terceira posição: componente de deformação total Z;
- Quarta posição: componente de deformação total XY;
- Quinta posição: componente de deformação total YZ;
- Sexta posição: componente de deformação total XZ;

Os valores de `SGmsh` podem ser definidos por meio das caixas de seleção "EPTOX, EPTOY, EPTOZ, EPTOXY, EPTOYZ e EPTOXZ"na GUI.

B.4. Aba DSS

Na aba DSS (ver Figura B.9) devem ser informados dados relacionados à verificação dos resultados gerados durante as análises.

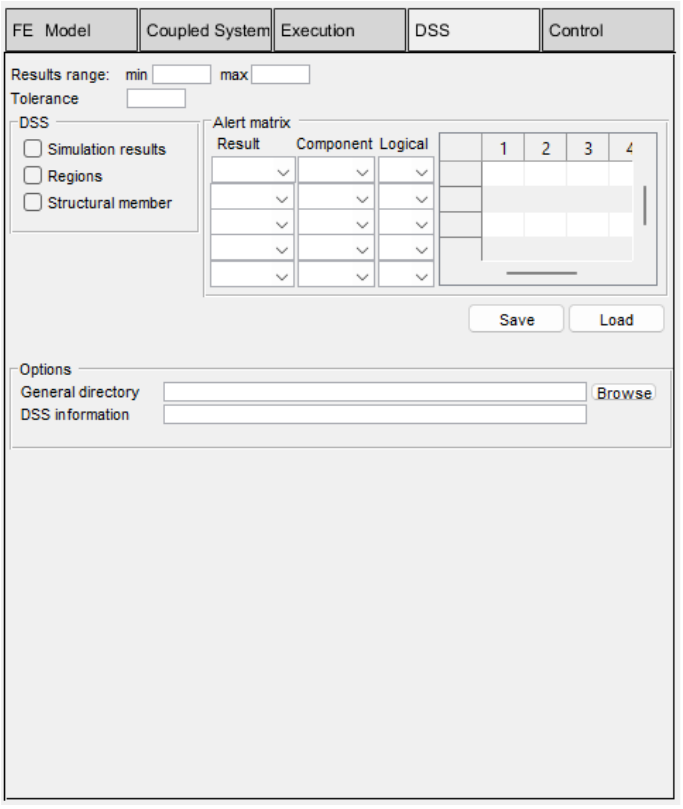


Figura B.9.: Aba de configuração de execução do DSS.

A análise dos resultados pode ser restringida à um intervalo definido no eixo longitudinal do modelo. Para isso, os campos e dados esperados são descritos abaixo:

1. Results range - min: menor valor do intervalo no eixo longitudinal para extração de resultados do modelo, é atribuído à variável "`MinRange`" do tipo *double*;
2. Results range - max: maior valor do intervalo no eixo longitudinal para extração de resultados do modelo, é atribuído à variável "`MaxRange`" do tipo *double*;
3. Tolerance: especifica um valor de tolerância para definir os elementos dentro do intervalo determinado, é atribuído à variável "`TolRange`" do tipo *double*.

As caixas de seleção do painel DSS definem quais resultados e de que maneira serão gerados para visualização. As caixas de seleção são descritas abaixo:

1. Simulation results: especifica se resultados gerados pelo *solver* devem ser salvos no formato .msh (Gmsh), é atribuído à variável "`PlotAnsGmsh`" do tipo *logical*;
2. Regions: especifica se deve ser gerada a visualização no formato .msh (Gmsh) das regiões que ultrapassam algum limite da matriz de alerta, é atribuído à variável "`PlotHotspots`" do tipo *logical*;
3. Structural member: especifica se deve ser gerada a visualização no formato .msh (Gmsh) dos membros estruturais que ultrapassam algum limite da matriz de alerta, é atribuído à variável "`PlotAlert`" do tipo *logical*;

A variável matriz de alerta ("`AlertMatrix`") do tipo *string array* e suas entradas esperadas são demonstrados na Tabela B.23:

Tabela B.23.: Variável matriz de alerta ("`AlertMatrix`") e suas entradas esperadas.

`Result`	`Component`	`Logical`	`Entry_05`	`...continued`
`Entry_01`	`Entry_02`	`Entry_03`	`Entry_04`	`...continued`
`...continued`	`...continued`	`...continued`	`...continued`	`...continued`

`Entry_01`: Resultado requerido para checar limite de valores. As opções são: "`Displacement`", "`Stress`", "`Elastic_Strain`", "`Plastic_Strain`", "`Total_Strain`" e "`Principal_Stress`".

`Entry_02`: A posição da componente de resultado analisado no alerta.

`Entry_03`: Sinal lógico utilizado para comparação. As opções são: "`>`" (maior que) ou "`<`" (menor que).

`Entry_04`: Valor para comparação do nível de alerta (A_i).

`Entry_05`: Nível de alerta (A_i).

Um exemplo da matriz de alerta é apresentado na Tabela B.24.

Tabela B.24.: Exemplo de matriz de alerta ("*AlertMatrix*") e suas entradas esperadas.

Result	Component	Logical	1	2
Displacement	2	>	0.002054	0.002641
Displacement	4	>	0.20652	0.2084
Principal_Stress	1	>	25700000	38600000
Principal_Stress	3	<	-17500000	-26300000
Stress	6	>	17000000	20000000

Notas:

1. A entrada `Entry_02` deve ser consistente com a opção atribuída à entrada `Entry_01` e às posições de cada componente das variáveis: `UGmsh`, `SGmsh`, `EPELGmsh`, `EPPLGmsh`, `EPTOGmsh` e `PRINGmsh` (conforme apresentado em Seção B.3).
2. Novos níveis de alerta (A_i) podem ser adicionados por meio de novas colunas.
3. Os valores de novos níveis de alerta (A_i) adicionados precisam ser consistentes com o sinal lógico e valores de comparação atribuídos para os níveis de alerta anteriores criando intervalos para análise dos valores.
4. O valor "0" não pode ser usado para a entrada `Entry_05` que especifica o nível de alerta, pois o nível de alerta "0" é restrito para regiões ou membros que não atingiram nenhum nível de alerta.
5. O módulo dos níveis de alerta precisam seguir valores crescentes, assim: $|A_i| < |A_{i+1}|$.

B.5. Aba *Control*

A aba *Control* (ver Figura B.10) é utilizada para manipular o Gêmeo Digital executando os algoritmos.

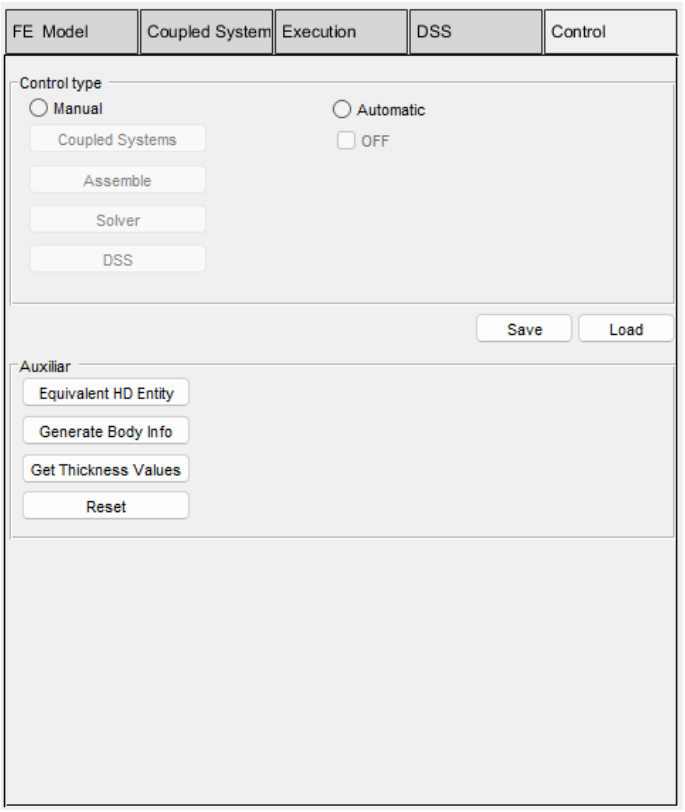


Figura B.10.: Aba de configuração de controle do Gêmeo Digital.

A definição dos itens do painel *Control type* são descritos abaixo:

1. Manual: botão de opção atribuído à variável "`radiobuttonControlManual`" do tipo *string*. Quando selecionado, determina que os algoritmos (ver algoritmos primários em Seção C.1) serão executados por meio botões de ação abaixo:
    - Coupled Systems: botão de ação que executa o algoritmo definido em Subseção C.1.1.
    - Assemble: botão de ação que executa o algoritmo definido em Subseção C.1.2.
    - Solver: botão de ação que executa o algoritmo definido em Subseção C.1.3.
    - DSS: botão de ação que executa o algoritmo definido em Subseção C.1.4.
2. Automatic: botão de opção atribuído à variável "`radiobuttonControlAutomatic`" do tipo *string*. Quando selecionado, determina que os algoritmos classificados como primários em Seção C.1 serão executados em sequência por meio da caixa de seleção abaixo:

- OFF/ON: caixa de seleção atribuída à variável "`checkboxControlAutomatic`" do tipo *string*. Ao ser definido como "ON", executa os algoritmos em sequência após atualização dos dados dos sistemas acoplados. Ao ser definido como "OFF", interrompe a execução dos algoritmos após o final da sequência.

A definição dos itens do painel *Auxiliar* são descritos abaixo:

1. Equivalent HD Entity: botão de ação que executa o algoritmo definido em Subseção C.1.9.
2. Generate Body Info: botão de ação que executa o algoritmo definido em Subseção C.1.8.
3. Get Thickness Values: botão de ação que executa o algoritmo definido em Subseção C.1.7.
4. Reset: botão de ação que executa o algoritmo definido em Subseção C.1.11.

C. Apêndice - Algoritmos

C.1. Definição dos algoritmos

Os algoritmos criados no MATLAB podem ser classificados em diferentes categorias de acordo com sua necessidade e função no desempenho do gêmeo digital:

- Primário: independente dos dados recebidos dos sistemas acoplados, é fundamental e necessário durante a rotina de execução das análises numéricas do gêmeo digital;
- Secundário: dependendo dos dados recebidos dos sistemas acoplados, é fundamental e necessário durante a rotina de execução das análises numéricas do gêmeo digital;
- Auxiliar: é fundamental na fase de preparação ou início da execução do gêmeo digital, mas não é requerido durante a rotina de execução das análises numéricas do gêmeo digital;
- Opcional: de uso facultativo, pode auxiliar no funcionamento do gêmeo digital.

A Tabela C.1 resume o objetivo e a categoria de cada algoritmo criado para o gêmeo digital.

Tabela C.1.: Resumo do objetivo e categoria dos algoritmos.

Algoritmo	**Objetivo**	**Categoria**
A01_coupled_systems	Obter e organizar dados dos sistemas acoplados	Primário
A03_assemble	Atualizar os modelos em elementos finitos	Primário
B04_solver	Criar um arquivo em lotes e executar o Solver em elementos finitos	Primário
C05_post_results	Organizar e gera resultados do SAD	Primário
Calc_Press	Obter pressão hidrodinâmica para a malha em elementos finitos	Secundário
Ini_A00_folders	Verificar e criar diretórios necessários (caso não existam)	Auxiliar
Ini_A00_get_parameters	Obter valores de espessura de elementos de casca e viga do modelo de elementos finitos	Auxiliar
Ini_A00_post_bodies	Obter e salvar informações das entidades do modelo de elementos finitos	Auxiliar
Ini_A01_equiv	Identifica nós e elementos equivalentes entre a malha em elementos finitos e malha do modelo hidrodinâmico	Auxiliar
Ini_A01_storage_levels_off	Cria arquivo utilizado para a condição de carregamento dos tanques igual a 0	Auxiliar
Res_All_Data	Deletar todo o conteúdo dos diretórios gerados por Ini_A00_Folders	Opcional
Ini_A00_initial_data	Inicializa variáveis (apenas para execução manual do Gêmeo Digital)	Opcional
Opt_B05_post_statistics	Gera dados estatísticos do consumo de tempo e memória com base nos arquivos de registros	Opcional
Opt_C06_plot_bodies	Gera todas as entidades do modelo de elementos finitos no formato de extensão do Gmsh	Opcional

C.1.1. Algoritmo A01_coupled_systems

Algoritmo	Objetivo	Categoria
A01_coupled_systems	Obter e organizar dados dos sistemas acoplados	Primário

```
%% A01_coupled_systems
%  Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%  Jul 17, 2021
%  This script/function reads and organize information from coupled systems to be used by A03_assemble
%  [pt-br] Este script/fun  o  l  e organiza a informa  o dos sistemas acoplados para ser usada pelo A03_assemble

%% Steps:
%  Load variables
%  Read options file
%  Get flags from options
%  Read initial parameters to set SEG or Wave OFF
%  Get significant wave height (Hs) and zero-crossing period for an irregular wave divided into parts by a fixed time value
%  Get accelerations and forces from cables for a wave divided into parts by a fixed time value
%  Create Multiple Cases array (MCContent) according to Flow-based (FB)
%  Update MCContent with information from coupled systems
%  Update MCContent with wave information
%  Update MCContent with SEG information
%  Update MCContent with Storage Levels information
%  Save MCContent
%  End of code

%% Load variables
fprintf(1,"Initializing A01_coupled_systems \n")
tic

%% Check if script was requested by GUI (using Manual or Automatic control)
try
   if control == "Manual"
       fprintf("Control: GUI - %s \n",control)
   elseif control == "Automatic"
       fprintf("Control: GUI - %s \n",control)
```

```
32      else
33          control = "None";
34          fprintf("Control: GUI - %s \n",control)
35      end
36  catch
37      control = "None";
38      fprintf("Control: GUI - %s \n",control)
39  end
40
41  %% Load variables by running without GUI
42  if control == "None"
43      Ini_A00_initial_data
44      clearvars -except Var_A01_coupled_systems
45
46      FileOpt = Var_A01_coupled_systems{1};                    %
            Options file
47      FolderCS = Var_A01_coupled_systems{2};                   %
            Folder for coupled systems
48      FileWaveReg = Var_A01_coupled_systems{3};                %
            Filename for regular wave file
49      FileWaveIrr = Var_A01_coupled_systems{4};                %
            Filename for irregular wave file
50      FolderWave = Var_A01_coupled_systems{5};                 %
            Folder for wave files
51      FileStor = Var_A01_coupled_systems{6};                   %
            Filename for storage levels file
52      FolderStor = Var_A01_coupled_systems{7};                 %
            Folder for storage levels
53      FileInsp = Var_A01_coupled_systems{8};                   %
            Filename for inspection file
54      FolderSEG = Var_A01_coupled_systems{9};                  %
            Folder for Standard Earth Gravity
55      DraftModel = Var_A01_coupled_systems{10};                % Draft
             value used in model
56      DivTimeIrrWave = Var_A01_coupled_systems{11};            % Time
            for division of irregular wave for analysis
57      cgloc_X = Var_A01_coupled_systems{12};                   % CG: X
             coordinate
58      cgloc_Y = Var_A01_coupled_systems{13};                   % CG: Y
             coordinate
59      cgloc_Z = Var_A01_coupled_systems{14};                   % CG: Z
             coordinate
60      floc_X = Var_A01_coupled_systems{15};                    % X
```

```
        coordinate for remote force
61      floc_Y = Var_A01_coupled_systems{16};                    % Y
        coordinate for remote force
62      floc_Z = Var_A01_coupled_systems{17};                    % Z
        coordinate for remote force
63      HTank = Var_A01_coupled_systems{18};                     %
        Maximum height value for capacity of the cargo tanks
64      FolderUpdates = Var_A01_coupled_systems{19};             %
        Folder for *.txt file
65      MCFile = Var_A01_coupled_systems{20};                    % File
        for definition of multiple cases (automatically generated)
66      Direction_transv_vertical = Var_A01_coupled_systems{21}; %
        Transversal (vertical) direction of model: x, y or z /  / This
        variable is locked after 1st running
67
68  end
69
70  %% Read options file
71  FileName = strcat(FolderCS,'\',FileOpt,'.txt');      % Get .txt file
        name
72  FileDataOpt = extractFileText(FileName);             % Get data from .
        txt file
73  FileDataOpt = regexprep(FileDataOpt,'\n','&%^');     % Replace break
        lines with codes
74  FileDataOpt = strip(split(FileDataOpt,'&%^'));       % Split characters
        and delete codes
75  FileDataOpt = string(FileDataOpt);                   % Convert to string
76  FileDataOpt = regexp(FileDataOpt,'\s','split');
77  if FileDataOpt{end,1} == ""
78      FileDataOpt(end,:)=[];
79  end
80  FileDataOpt = vertcat(FileDataOpt{:});
81
82  %  Check flow-based (FB)
83  FB = FileDataOpt(2,3);                               % Get string (FB)
84  FBIdx = find(FB == FileDataOpt(:,2));                % Get index for
        string
85  if FileDataOpt(FBIdx,3) == "FALSE"                   % Stop code if
        status for FB = "FALSE"
86      fprintf("Item: Option \n");
87      fprintf("Flow-based: %s \n",FB);
88      fprintf("%s: %s \n\n",FB,FileDataOpt(FBIdx,3));
89      fprintf("Option for item '%s' cannot be 'FALSE' \n",FB);
```

```
90      return
91  end
92
93  %% Get flags from options
94  %  Read wave files and get flags (regular or irregular)
95  Item = "Wave";
96  ItemIdx = find(Item == FileDataOpt(:,2));
97  if FileDataOpt(ItemIdx,3) ~= "FALSE"
98      if FileDataOpt(ItemIdx,3) == "regular"                  % Load
            information
99          load(FileWaveReg,'RegularWave');
100         FLWave = 1;                                         % Flag for
                regular wave type
101     elseif FileDataOpt(ItemIdx,3) == "irregular"            % Load
            information
102         load(FileWaveIrr,'IrregularWave');
103         FLWave = 2;                                         % Flag for
                irregular wave type
104     elseif FileDataOpt(ItemIdx,3) == "off"
105         FLWave = 0;                                         % Flag for
                hydrodynamic wave analysis off
106         Status_HDP = 0;
107     else
108         disp("Incorrect option for Wave.")
109         return
110     end
111 end
112
113 %  Check Wave_analysis (0 = OFF; 1 = NODE; 2 = ELEMENT)
114 ItemIdx = find("Wave_analysis" == FileDataOpt(:,2));    % Get index for
        string
115 Status_HDP = FileDataOpt(ItemIdx,3);
116
117 if Status_HDP == "NODE" && FLWave ~= 0
118     Status_HDP = 1;
119 elseif Status_HDP == "ELEMENT" && FLWave ~= 0
120     Status_HDP = 2;
121 else
122     Status_HDP = 0;
123 end
124
125 %  Check analysis in min pressure phase angle
126 ItemIdx = find("Min_pres_phase_angle" == FileDataOpt(:,2));     % Get
```

```
    index for string
Status_MinPA = FileDataOpt(ItemIdx,3);

%  Check analysis in max pressure phase angle
ItemIdx = find("Max_pres_phase_angle" == FileDataOpt(:,2));      % Get
    index for string
Status_MaxPA = FileDataOpt(ItemIdx,3);

%  Read standard earth gravity status and get flag
Item = "Standard_earth_gravity";
ItemIdx = find(Item == FileDataOpt(:,2));
if FileDataOpt(ItemIdx,3) ~= "FALSE"
    FLSEG = 1;                                                   % Flag
        for standard earth gravity status "TRUE"
else
    FLSEG = 0;                                                   % Flag
        for standard earth gravity status "FALSE"
end

%  Read storage levels file and get flag
Item = "Storage_levels";
ItemIdx = find(Item == FileDataOpt(:,2));
if FileDataOpt(ItemIdx,3) ~= "FALSE"
    FLStor = 1;                                                  % Flag
        for Storage_levels status "TRUE"

    FileNameStor = FileStor;                                     % Get .
        txt file name
    FileDataStor = extractFileText(FileNameStor);                % Get
        data from .txt file
    FileDataStor = regexprep(FileDataStor,'\n','&%^');           %
        Replace break lines with codes
    FileDataStor = strip(split(FileDataStor,'&%^'));             % Split
         characters and delete codes
    FileDataStor = string(FileDataStor);                         %
        Convert to string

    if FileDataStor(end,1) == ""                                 %
        Delete last row if empty
         FileDataStor = FileDataStor(1:end-1,1);
    end

    FileDataStor = regexp(FileDataStor,'\s','split');            % Split
```

```
        string
159     if FileDataStor{end} == ""                                          %
        Delete last row if empty
160         FileDataStor = FileDataStor(1:size(FileDataStor,1)-1);
161     end
162     FileDataStor = vertcat(FileDataStor{:});
163
164     % Convert Storage Levels from % to meters
165     HSPFileDataStorIdx = char(FileDataStor(:,1));
166     HSPFileDataStorIdx = string(HSPFileDataStorIdx(:,1:6));
167     HSPFileDataStorIdx = find("***HSP" == HSPFileDataStorIdx);  % Index
        for Hidrostatic Pressure in FileDataStor
168     HSPFileDataStorIdx = HSPFileDataStorIdx(2:end);             %
        Delete line related to parameters
169
170     for i=1:size(HSPFileDataStorIdx,1)                                  %
        Convert Storage Levels in % to Storage Levels in m for all HSP
171         for j=2:size(FileDataStor,2)
172
173             LevelTank = round((double(FileDataStor(HSPFileDataStorIdx(i
                ),j))/100 * HTank) - DraftModel,3);
174             FileDataStor(HSPFileDataStorIdx(i),j) = string(LevelTank);
175         end
176     end
177
178 else
179     FLStor = 0;                                                         % Flag
        for Storage_levels status "FALSE"
180 end
181
182 % Read file for Storage Levels off
183 FileName = strcat(FolderStor,'\','- StorageLevelsOFF.txt');     % Get .
    txt file name
184 FileDataStorOFF = extractFileText(FileName);                    % Get
    data from .txt file
185 FileDataStorOFF = regexprep(FileDataStorOFF,'\n','&%^');        %
    Replace break lines with codes
186 FileDataStorOFF = strip(split(FileDataStorOFF,'&%^'));          % Split
     characters and delete codes
187 FileDataStorOFF = string(FileDataStorOFF);                     %
    Convert to string
188
189 if FileDataStorOFF(end,1) == ""                                % Delete
```

```
    last row if empty
190     FileDataStorOFF = FileDataStorOFF(1:end-1,1);
191 end
192
193 FileDataStorOFF = regexp(FileDataStorOFF,'\s','split');
194 FileDataStorOFF = vertcat(FileDataStorOFF{:});
195
196 HSPFileDataStorIdx = char(FileDataStorOFF(:,1));
197 HSPFileDataStorIdx = string(HSPFileDataStorIdx(:,1:6));
198 HSPFileDataStorIdx = find("***HSP" == HSPFileDataStorIdx);      % Index
    for Hidrostatic Pressure in FileDataStor
199 HSPFileDataStorIdx = HSPFileDataStorIdx(2:end);                 %
    Delete line related to parameters
200
201 for i=1:size(HSPFileDataStorIdx,1)                              %
    Convert Storage Levels in % to Storage Levels in m for all HSP
202     for j=2:size(FileDataStorOFF,2)
203         LevelTank = round((double(FileDataStorOFF(HSPFileDataStorIdx(i)
            ,j))/100 * HTank) - DraftModel,3) - 1;
204         FileDataStorOFF(HSPFileDataStorIdx(i),j) = string(LevelTank);
205     end
206 end
207
208 %  Read inspection file and get flag
209 Item = "Inspection";
210 ItemIdx = find(Item == FileDataOpt(:,2));
211 if FileDataOpt(ItemIdx,3) ~= "FALSE"
212     FileNameInsp = FileInsp;                                    % Get .
        txt file name
213     FileDataInsp = extractFileText(FileNameInsp);               % Get
        data from .txt file
214     FileDataInsp = regexprep(FileDataInsp,'\n','&%^');          %
        Replace break lines with codes
215     FileDataInsp = strip(split(FileDataInsp,'&%^'));            % Split
         characters and delete codes
216     FileDataInsp = string(FileDataInsp);                        %
        Convert to string
217
218     if FileDataInsp(end,1) == ""                                %
        Delete last row if empty
219         FileDataInsp = FileDataInsp(1:end-1,1);
220     end
221
```

```
222     FileDataInsp = regexp(FileDataInsp,'\s','split');           % Split
            string
223     FileDataInsp = vertcat(FileDataInsp{:});
224     FLInsp = 1;                                                 % Flag
           for Inspection status "TRUE"
225 else
226     FLInsp = 0;                                                 % Flag
           for Inspection status "FALSE"
227 end
228
229 %% Read initial parameters to set SEG or Wave OFF
230 FileName = strcat(FolderWave,'\','- WaveOFF.txt');              % Get .
       txt file name
231 FileDataWaveOFF = extractFileText(FileName);                    % Get
       data from .txt file
232 FileDataWaveOFF = regexprep(FileDataWaveOFF,'\n','&%^');        %
       Replace break lines with codes
233 FileDataWaveOFF = strip(split(FileDataWaveOFF,'&%^'));         % Split
        characters and delete codes
234 FileDataWaveOFF = string(FileDataWaveOFF);                     %
       Convert to string
235
236 if FileDataWaveOFF(end,1) == ""                               % Delete
        last row if empty
237     FileDataWaveOFF = FileDataWaveOFF(1:end-1,1);
238 end
239
240 FileDataWaveOFF = regexp(FileDataWaveOFF,'\s','split');
241 FileDataWaveOFF = vertcat(FileDataWaveOFF{:});
242
243 FileName = strcat(FolderSEG,'\','- SEGOFF.txt');                % Get .
       txt file name
244 FileDataSEGOFF = extractFileText(FileName);                     % Get
       data from .txt file
245 FileDataSEGOFF = regexprep(FileDataSEGOFF,'\n','&%^');         %
       Replace break lines with codes
246 FileDataSEGOFF = strip(split(FileDataSEGOFF,'&%^'));          % Split
        characters and delete codes
247 FileDataSEGOFF = string(FileDataSEGOFF);                      %
       Convert to string
248 FileDataSEGOFF = regexp(FileDataSEGOFF,'\s','split');
249 FileDataSEGOFF = vertcat(FileDataSEGOFF{:});
250
```

```
251 MCContent = [FileDataWaveOFF(2:end,1:2);FileDataSEGOFF(2:end,1:2);
        FileDataStorOFF(2:end,1:2)];
252
253 MCContent(1:end,2:3) = MCContent;
254 MCContent(:,1) = [1:size(MCContent,1)];
255
256 %% Get significant wave height (Hs) and zero-crossing period for an
        irregular wave divided into parts by a fixed time value
257 if FLWave == 2
258     TimeIdx = find("Time" == IrregularWave(:,1));                    %
            Get index
259     SeaElevationIdx = find("SeaElevation" == IrregularWave(:,1));    %
            Get index
260
261     t = str2double(IrregularWave(TimeIdx,2:end))';                   %
            Get values
262     s = str2double(IrregularWave(SeaElevationIdx,2:end))';           %
            Get values
263     SeaElevation = [t,s];                                            %
            Get time and sea elevation
264
265     MaxTime = max(SeaElevation(:,1));                                %
            Max time of irregular wave
266
267     DivIrrWave = floor(MaxTime/DivTimeIrrWave);                      %
            Number of divisions for irregular wave
268     DivTime = [];                                                    %
            Array to get division of wave
269
270     IdxTime_0 = 1;                                                   %
            Index for initial time
271     TzHs = ["Tz","Hs"];                                              %
            Array for Ti (zero crossing period) and Hs (significant wave
            height)
272     TiHsAux = [];                                                    %
            Auxiliar array to calculate Ti and Hs for part (division) of
            the irregular wave
273     TiHi = [];                                                       %
            Auxiliar array
274
275     for i=1:DivIrrWave                                               %
            Loop through parts of irregular wave
276         RangeIrrWave = [];                                           %
```

```
            Array for range (part) of irregular wave
277         Time_1 = i * DivTimeIrrWave;                                    %
            Final time for range
278
279         [d,IdxTime_1] = min( abs(SeaElevation(:,1)-Time_1 ) );          %
            Index for final time
280
281         RangeIrrWave = SeaElevation(IdxTime_0:IdxTime_1,:);             %
            Get part of irregular wave
282         FL_CrossZero = 0;                                               %
            Flag to check if H has crossed zero value
283         MaxAmp = 0;                                                     %
            Auxiliar variable to get maximum value for Hi
284         MinAmp = 0;                                                     %
            Auxiliar variable to get minimum value for Hi
285         FL_StartChecking = 0;                                           %
            Flag to check if H has crossed zero value for first time
286         for t=2:size(RangeIrrWave,1)
287
288             if (RangeIrrWave(t,2) / RangeIrrWave(t-1,2)) < 0            %
                Identify A = 0;
289                 FL_CrossZero = FL_CrossZero + 1;                        %
                    Amplitude line has crossed 0
290
291                 % Interpolate first value for Ti (zero crossing period)
292                 if FL_CrossZero == 1 && FL_StartChecking == 0;          %
                    Calculte T_0 for the first time by interpolating
                    value
293                     Ti_0 = ((RangeIrrWave(t,1) - RangeIrrWave(t-1,1)) *
                          (abs(RangeIrrWave(t-1,2))) / (abs(RangeIrrWave
                        (t,2)) + abs(RangeIrrWave(t-1,2))) )+ (
                        RangeIrrWave(t-1,1));
294                 end
295
296                 FL_StartChecking = 1;                                   %
                    First time crossing zero
297
298                 % Interpolate first value for Ti (zero crossing period)
299                 if FL_CrossZero == 3;                                   %
                    Calculte T_1 by interpolating value
300                     Ti_1 = ((RangeIrrWave(t,1) - RangeIrrWave(t-1,1)) *
                          (abs(RangeIrrWave(t-1,2))) / (abs(RangeIrrWave
                        (t,2)) + abs(RangeIrrWave(t-1,2))) )+ (
```

```
                    RangeIrrWave(t-1,1));
301             end
302         end
303
304         % Get maximum value for Hi if first time crossing zero has
                passed
305         if RangeIrrWave(t,2) > MaxAmp && FL_StartChecking == 1
306             MaxAmp = RangeIrrWave(t,2);
307         end
308
309         % Get minimum value for Hi if first time crossing zero has
                passed
310         if RangeIrrWave(t,2) < MinAmp && FL_StartChecking == 1
311             MinAmp = RangeIrrWave(t,2);
312         end
313
314         if FL_CrossZero == 3;                          % Cycle has
                ended
315             Ti = Ti_1 - Ti_0;                          % Calculte Ti (
                    zero crossing period)
316             Hi = MaxAmp + abs(MinAmp);                 % Calculte Hi (
                    wave height)
317             TiHi = [Ti,Hi];
318             TiHsAux = [TiHsAux;TiHi];                  % Add Ti and Hi
                     to auxiliar array TiHsAux
319
320             Ti_0 = Ti_1;                               % Ti_0 for next
                     Ti
321
322             FL_CrossZero = 1;                          % Reinitialize
                    process to get Ti and Hi
323             MaxAmp = 0;                                % Reinitialize
                    process to get Ti and Hi
324             MinAmp = 0;                                % Reinitialize
                    process to get Ti and Hi
325         end
326
327     end
328
329     DivTime = [DivTime;IdxTime_0];                     % Idx Div Time
330     IdxTime_0 = IdxTime_1;                             % Idx of time
            to start new part of the wave
331
```

```
332         Hi = sort(TiHsAux(:,2));                                % Sort Hi
                values in ascending order
333
334         OneThird = ceil(size(Hi,1)/3);                          % Calculates
                1/3 of size
335
336         HsHigherOneThird = Hi(OneThird*2+1:size(Hi,1)); % Get 1/3 of Hi
                 (highest values of wave heights)
337
338         TzHs(i+1,1) = string(mean(TiHsAux(:,1)));               % Get Tz (
                average of Ti) for part of irregular wave
339         TzHs(i+1,2) = string(mean(HsHigherOneThird));   % Get Hs (
                average of highest 1/3 of Hi) for part of irregular wave
340
341         TiHsAux = [];                                           % Clear
                auxiliar array TiHsAux
342
343     end
344 end
345 clear i TiHsAux HsHigherOneThird OneThird Hi IdxTime_0 IdxTime_1 MinAmp
        MaxAmp Ti_0 Ti_1 Ti Hi TiHi FL_CrossZero s t Time_1
    SeaElevationIdx
346
347 %% Get accelerations and forces from cables for a wave divided into
    parts by a fixed time value
348 if FLWave == 2                                                  % Check
     if flag is for a irregular wave
349     ResAcel = [];                                               % Aux
           array to get resultant vectors of linear acceleration
350     ResDcgomg = [];                                             % Aux
           array to get resultant vectors of angular acceleration
351     ResCgomga = [];                                             % Aux
           array to get resultant cgomga
352     ResOmega = [];                                              % Aux
           array to get resultant omega
353     ResDomega = [];                                             % Aux
           array to get resultant domega
354
355     ResLineF = [];                                              % Aux
           array to get resultant vectors of line forces
356
357     % Find index for linear accelerations
358     AcelXIdx = find("acel_X" == IrregularWave(:,1));        % Get index
```

```
359     AcelYIdx = find("acel_Y" == IrregularWave(:,1));          % Get index
360     AcelZIdx = find("acel_Z" == IrregularWave(:,1));          % Get index
361
362     % Get linear acceleration components
363     AcelX = str2double(IrregularWave(AcelXIdx,2:end));
364     AcelY = str2double(IrregularWave(AcelYIdx,2:end));
365     AcelZ = str2double(IrregularWave(AcelZIdx,2:end));
366
367     % Find index for angular accelerations
368     DcgomgXIdx = find("dcgomg_X" == IrregularWave(:,1));      % Get index
369     DcgomgYIdx = find("dcgomg_Y" == IrregularWave(:,1));      % Get index
370     DcgomgZIdx = find("dcgomg_Z" == IrregularWave(:,1));      % Get index
371
372     % Get angular acceleration components
373     DcgomgX = str2double(IrregularWave(DcgomgXIdx,2:end));
374     DcgomgY = str2double(IrregularWave(DcgomgYIdx,2:end));
375     DcgomgZ = str2double(IrregularWave(DcgomgZIdx,2:end));
376
377     % Find index for cgomga
378     CgomgaXIdx = find("cgomga_X" == IrregularWave(:,1));      % Get index
379     CgomgaYIdx = find("cgomga_Y" == IrregularWave(:,1));      % Get index
380     CgomgaZIdx = find("cgomga_Z" == IrregularWave(:,1));      % Get index
381
382     % Get cgomga components
383     CgomgaX = str2double(IrregularWave(CgomgaXIdx,2:end));
384     CgomgaY = str2double(IrregularWave(CgomgaYIdx,2:end));
385     CgomgaZ = str2double(IrregularWave(CgomgaZIdx,2:end));
386
387     % Find index for omega
388     OmegaXIdx = find("omega_X" == IrregularWave(:,1));        % Get index
389     OmegaYIdx = find("omega_Y" == IrregularWave(:,1));        % Get index
390     OmegaZIdx = find("omega_Z" == IrregularWave(:,1));        % Get index
391
392     % Get omega components
393     OmegaX = str2double(IrregularWave(OmegaXIdx,2:end));
394     OmegaY = str2double(IrregularWave(OmegaYIdx,2:end));
395     OmegaZ = str2double(IrregularWave(OmegaZIdx,2:end));
396
397     % Find index for domega
398     DomegaXIdx = find("domega_X" == IrregularWave(:,1));      % Get index
399     DomegaYIdx = find("domega_Y" == IrregularWave(:,1));      % Get index
400     DomegaZIdx = find("domega_Z" == IrregularWave(:,1));      % Get index
401
```

```
402     % Get domega components
403     DomegaX = str2double(IrregularWave(DomegaXIdx,2:end));
404     DomegaY = str2double(IrregularWave(DomegaYIdx,2:end));
405     DomegaZ = str2double(IrregularWave(DomegaZIdx,2:end));
406
407     % Find index for lines
408     FxIdx = [];                                          % Aux array
            to get index for X forces of lines
409     FyIdx = [];                                          % Aux array
            to get index for Y forces of lines
410     FzIdx = [];                                          % Aux array
            to get index for Z forces of lines
411
412     for i=1:size(IrregularWave,1)                        % Loop to
           find items with Prefix = "Line" and Sufix = "X", "Y" or "Z"
413         ItemName = char(IrregularWave(i,1));
414
415         if length(ItemName) > 4
416             ItemName = char(IrregularWave(i,1));
417             ItemNamePrefix = string(ItemName(:,1:4));
418             ItemNameSufix = string(ItemName(:,length(ItemName)));
419
420             if ItemNamePrefix == "Line" && ItemNameSufix == "X"
421                 FxIdx = [FxIdx;i];
422             end
423
424             if ItemNamePrefix == "Line" && ItemNameSufix == "Y"
425                 FyIdx = [FyIdx;i];
426             end
427
428             if ItemNamePrefix == "Line" && ItemNameSufix == "Z"
429                 FzIdx = [FzIdx;i];
430             end
431         end
432     end
433
434     % Sum of force line components
435     SumFx = sum(str2double(IrregularWave(FxIdx,2:end)));
436     SumFy = sum(str2double(IrregularWave(FyIdx,2:end)));
437     SumFz = sum(str2double(IrregularWave(FzIdx,2:end)));
438
439     % Calculate vector resultant for acceleration, velocity, and  force
            componentes
```

```
440     for i=1:(size(IrregularWave,2)-1)
441         ResAcel = [ResAcel,((AcelX(i)^2) + (AcelY(i)^2) + (AcelZ(i)^2))
                ^ (1/2)];           % Linear Acceleration
442         ResDcgomg = [ResDcgomg,((DcgomgX(i)^2) + (DcgomgY(i)^2) + (
               DcgomgZ(i)^2)) ^ (1/2)]; % Angular Acceleration
443         ResCgomga = [ResCgomga,((CgomgaX(i)^2) + (CgomgaY(i)^2) + (
               CgomgaZ(i)^2)) ^ (1/2)]; % cgomga
444         ResOmega = [ResOmega,((OmegaX(i)^2) + (OmegaY(i)^2) + (OmegaZ(i
               )^2)) ^ (1/2)];      % omega
445         ResDomega = [ResDomega,((DomegaX(i)^2) + (DomegaY(i)^2) + (
               DomegaZ(i)^2)) ^ (1/2)]; % domega
446         ResLineF = [ResLineF,((SumFx(i)^2) + (SumFy(i)^2) + (SumFz(i)
               ^2)) ^ (1/2)];         % Forces
447     end
448
449     IdxMaxAcel = [];    % Idx for all max resultants of linear acc
450     IdxMaxDcgomg = [];  % Idx for all max resultants of angular acc
451     IdxMaxCgomga = [];  % Idx for all max resultants of cgomga
452     IdxMaxOmega = [];   % Idx for all max resultants of omega
453     IdxMaxDomega = [];  % Idx for all max resultants of domega
454     IdxMaxLineF = [];   % Idx for all max resultants of forces
455
456     % Get index for max vector resultant (linear acc, angular acc and
           force) for each part of irr wave
457     for i=1:size(DivTime,1)
458         if i ~= size(DivTime,1)
459             [MaxValue,MaxIdx] = max(ResAcel(DivTime(i):DivTime(i+1)));
460             MaxIdx = MaxIdx + DivTime(i) - 1;
461             IdxMaxAcel = [IdxMaxAcel;MaxIdx];
462
463             [MaxValue,MaxIdx] = max(ResDcgomg(DivTime(i):DivTime(i+1)))
                   ;
464             MaxIdx = MaxIdx + DivTime(i) - 1;
465             IdxMaxDcgomg = [IdxMaxDcgomg;MaxIdx];
466
467             [MaxValue,MaxIdx] = max(ResCgomga(DivTime(i):DivTime(i+1)))
                   ;
468             MaxIdx = MaxIdx + DivTime(i) - 1;
469             IdxMaxCgomga = [IdxMaxCgomga;MaxIdx];
470
471             [MaxValue,MaxIdx] = max(ResOmega(DivTime(i):DivTime(i+1)));
472             MaxIdx = MaxIdx + DivTime(i) - 1;
473             IdxMaxOmega = [IdxMaxOmega;MaxIdx];
```

```
474
475             [MaxValue,MaxIdx] = max(ResDomega(DivTime(i):DivTime(i+1)))
                    ;
476             MaxIdx = MaxIdx + DivTime(i) - 1;
477             IdxMaxDomega = [IdxMaxDomega;MaxIdx];
478
479             [MaxValue,MaxIdx] = max(ResLineF(DivTime(i):DivTime(i+1)));
480             MaxIdx = MaxIdx + DivTime(i) - 1;
481             IdxMaxLineF = [IdxMaxLineF;MaxIdx];
482         else
483             [MaxValue,MaxIdx] = max(ResAcel(DivTime(i):end));
484             MaxIdx = MaxIdx + DivTime(i) - 1;
485             IdxMaxAcel = [IdxMaxAcel;MaxIdx];
486
487             [MaxValue,MaxIdx] = max(ResDcgomg(DivTime(i):end));
488             MaxIdx = MaxIdx + DivTime(i) - 1;
489             IdxMaxDcgomg = [IdxMaxDcgomg;MaxIdx];
490
491             [MaxValue,MaxIdx] = max(ResCgomga(DivTime(i):end));
492             MaxIdx = MaxIdx + DivTime(i) - 1;
493             IdxMaxCgomga = [IdxMaxCgomga;MaxIdx];
494
495             [MaxValue,MaxIdx] = max(ResOmega(DivTime(i):end));
496             MaxIdx = MaxIdx + DivTime(i) - 1;
497             IdxMaxOmega = [IdxMaxOmega;MaxIdx];
498
499             [MaxValue,MaxIdx] = max(ResDomega(DivTime(i):end));
500             MaxIdx = MaxIdx + DivTime(i) - 1;
501             IdxMaxDomega = [IdxMaxDomega;MaxIdx];
502
503             [MaxValue,MaxIdx] = max(ResLineF(DivTime(i):end));
504             MaxIdx = MaxIdx + DivTime(i) - 1;
505             IdxMaxLineF = [IdxMaxLineF;MaxIdx];
506         end
507     end
508
509     % Copy information from TzHs array and add new columns
510     IrregularWaveRes = TzHs;
511     Aux = ["Direction","acel_X","acel_Y","acel_Z","dcgomg_X","dcgomg_Y
        ","dcgomg_Z","cgomga_X","cgomga_Y","cgomga_Z","omega_X","
        omega_Y","omega_Z","domega_X","domega_Y","domega_Z","FX","FY","
        FZ"];
512     IrregularWaveRes(1,size(IrregularWaveRes,2)+1:size(Aux,2)+size(
```

```
        IrregularWaveRes,2)) = Aux;
513     Aux = find("WaveDirection" == IrregularWave(:,1));
                                  % Get index
514     IrregularWaveRes(2:end,3) = IrregularWave(Aux,2);
                                   % Apply wave direction
515
516     % Add components (linear acc, angular acc and force) according to
           max vector resultant for each part of irr wave
517     for i=1:DivIrrWave
518         IrregularWaveRes(i+1,4) = IrregularWave(AcelXIdx,IdxMaxAcel(i)
               +1);       % Max Linear Acc X for part of wave
519         IrregularWaveRes(i+1,5) = IrregularWave(AcelYIdx,IdxMaxAcel(i)
               +1);       % Max Linear Acc Y for part of wave
520         IrregularWaveRes(i+1,6) = IrregularWave(AcelZIdx,IdxMaxAcel(i)
               +1);       % Max Linear Acc Z for part of wave
521
522         IrregularWaveRes(i+1,7) = IrregularWave(DcgomgXIdx,IdxMaxDcgomg
               (i)+1);   % Max Angular Acc X for part of wave
523         IrregularWaveRes(i+1,8) = IrregularWave(DcgomgYIdx,IdxMaxDcgomg
               (i)+1);   % Max Angular Acc Y for part of wave
524         IrregularWaveRes(i+1,9) = IrregularWave(DcgomgZIdx,IdxMaxDcgomg
               (i)+1);   % Max Angular Acc Z for part of wave
525
526         IrregularWaveRes(i+1,10) = IrregularWave(CgomgaXIdx,
               IdxMaxCgomga(i)+1); % Max cgomga X for part of wave
527         IrregularWaveRes(i+1,11) = IrregularWave(CgomgaYIdx,
               IdxMaxCgomga(i)+1); % Max cgomga Y for part of wave
528         IrregularWaveRes(i+1,12) = IrregularWave(CgomgaZIdx,
               IdxMaxCgomga(i)+1); % Max cgomga Z for part of wave
529
530         IrregularWaveRes(i+1,13) = IrregularWave(OmegaXIdx,IdxMaxOmega(
               i)+1);    % Max omega X for part of wave
531         IrregularWaveRes(i+1,14) = IrregularWave(OmegaYIdx,IdxMaxOmega(
               i)+1);    % Max omega Y for part of wave
532         IrregularWaveRes(i+1,15) = IrregularWave(OmegaZIdx,IdxMaxOmega(
               i)+1);    % Max omega Z for part of wave
533
534         IrregularWaveRes(i+1,16) = IrregularWave(DomegaXIdx,
               IdxMaxDomega(i)+1); % Max domega X for part of wave
535         IrregularWaveRes(i+1,17) = IrregularWave(DomegaYIdx,
               IdxMaxDomega(i)+1); % Max domega Y for part of wave
536         IrregularWaveRes(i+1,18) = IrregularWave(DomegaZIdx,
               IdxMaxDomega(i)+1); % Max domega Z for part of wave
```

```
537
538         IrregularWaveRes(i+1,19) = SumFx(IdxMaxLineF(i));
                                    % Max Line Force X for part of wave
539         IrregularWaveRes(i+1,20) = SumFy(IdxMaxLineF(i));
                                    % Max Line Force Y for part of wave
540         IrregularWaveRes(i+1,21) = SumFz(IdxMaxLineF(i));
                                    % Max Line Force Z for part of wave
541     end
542
543     IrregularWaveRes = IrregularWaveRes';
544     clear SumFx SumFy SumFz AccAngXIdx AccAngYIdx AccAngZIdx AccLinXIdx
          AccLinYIdx AccLinZIdx IdxMaxLineF IdxMaxAccAng IdxMaxAccLin
        AccAngX AccAngY AccAngZ AccLinX AccLinY AccLinZ d FxIdx FyIdx
        FzIdx Item ItemIdx ItemName ItemNamePrefix ItemNameSufix MaxIdx
          MaxValue RangeIrrWave ResAccAng ResAccLin ResLineF TimeIdx
545 end
546
547 %% Create Multiple Cases array (MCContent) according to FB
548 % Update initial content according to FB
549 switch FB
550     case "Wave"
551         if FLWave == 1
                                                                    %
             Case regular wave type
552             MCContent(1,4:(size(RegularWave,2)+2)) = RegularWave(1,2:
                    end);
553         elseif FLWave == 2
                                                                % Case
              irregular wave type
554             MCContent(1,4:(size(TzHs,1)-1+3)) = [0:(size(TzHs,1)-2)];
555         end
556     case "Storage_levels"
557             MCContent(1,4:(size(FileDataStor,2)+1)) = [0:(size(
                    FileDataStor,2)-3)];
558     case "Inspection"
559             MCContent(1,4:(size(FileDataInsp,2))) = [0:(size(
                    FileDataInsp,2)-4)];
560 end
561 MCContent(2,4:end) = "TRUE";
562
563 %% Update MCContent with information from coupled systems
564 %% Update MCContent with wave information
565 MCContent(3,4:end) = string(Status_HDP);
```

```
                                        % Set ***Status_HDP*** according
    Wave_analysis
566 MCContent(4,4:end) = MCContent(4,3);
                                                % Repeat HDP parmeters
567 MCContent(5,4:end) = "1";
                                                        % Set HDP
    Interpolation method (only 1 is available now)
568 MCContent(6,4:end) = Status_MinPA;
                                                % Set analysis according
    to min pressure phase angle
569 MCContent(7,4:end) = Status_MaxPA;
                                                % Set analysis according
    to min pressure phase angle
570
571 if FB == "Wave" && FLWave == 1
                                                    % FB = Wave and
    Regular Wave
572
573     MCContent(8,4:end) = RegularWave(2,2:end);
                                    % Set amplitude
574     MCContent(9,4:end) = RegularWave(3,2:end);
                                    % Set period
575     MCContent(10,4:end) = RegularWave(4,2:end);
                                % Set direction
576
577     MCContent(11,4:end) = sprintf('%.9E',cgloc_X);
                                % Set CG component X
578     MCContent(12,4:end) = sprintf('%.9E',cgloc_Y);
                                % Set CG component Y
579     MCContent(13,4:end) = sprintf('%.9E',cgloc_Z);
                                % Set CG component Z
580
581     for a=2:size(RegularWave,2)
582         MCContent(14,a+2) = sprintf('%.9E',RegularWave(5,a));
                        % Set linear acceleration component X
583         MCContent(15,a+2) = sprintf('%.9E',RegularWave(6,a));
                        % Set linear acceleration component Y
584         MCContent(16,a+2) = sprintf('%.9E',RegularWave(7,a));
                        % Set linear acceleration component Z
585
586         MCContent(17,a+2) = sprintf('%.9E',RegularWave(8,a));
                        % Set angular acceleration component X
587         MCContent(18,a+2) = sprintf('%.9E',RegularWave(9,a));
```

```
                        % Set angular acceleration component Y
588         MCContent(19,a+2) = sprintf('%.9E',RegularWave(10,a));
                       % Set angular acceleration component Z
589
590         MCContent(20,a+2) = sprintf('%.9E',RegularWave(11,a));
                       % Set cgomga component X
591         MCContent(21,a+2) = sprintf('%.9E',RegularWave(12,a));
                       % Set cgomga component Y
592         MCContent(22,a+2) = sprintf('%.9E',RegularWave(13,a));
                       % Set cgomga component Z
593
594         MCContent(23,a+2) = sprintf('%.9E',RegularWave(14,a));
                       % Set omega component X
595         MCContent(24,a+2) = sprintf('%.9E',RegularWave(15,a));
                       % Set omega component Y
596         MCContent(25,a+2) = sprintf('%.9E',RegularWave(16,a));
                       % Set omega component Z
597
598         MCContent(26,a+2) = sprintf('%.9E',RegularWave(17,a));
                       % Set domega component X
599         MCContent(27,a+2) = sprintf('%.9E',RegularWave(18,a));
                       % Set domega component Y
600         MCContent(28,a+2) = sprintf('%.9E',RegularWave(19,a));
                       % Set domega component Z
601
602         MCContent(34,a+2) = sprintf('%.0f.',RegularWave(20,a));
                      % Set component X for Remote Force
603         MCContent(35,a+2) = sprintf('%.0f.',RegularWave(21,a));
                      % Set component Y for Remote Force
604         MCContent(36,a+2) = sprintf('%.0f.',RegularWave(22,a));
                      % Set component Z for Remote Force
605
606     end
607
608     MCContent(29,4:end) = "TRUE";
                                                  % Set Status for
        Remote Force
609     MCContent(30,4:end) = "6";
                                                     % Set number of
        parameters for Remote Force
610
611     MCContent(31,4:end) = sprintf('%.9E',floc_X);
                                % Set component X for location of
```

```
        Remote Force
612     MCContent(32,4:end) = sprintf('%.9E',floc_Y);
                                    % Set component Y for location of
        Remote Force
613     MCContent(33,4:end) = sprintf('%.9E',floc_Z);
                                    % Set component Z for location of
        Remote Force
614
615 elseif FB == "Wave" && FLWave == 2    % FB = Wave and Irregular Wave
616
617     MCContent(8,4:end) = string(double(IrregularWaveRes(2,2:end))/2);
                % Set amplitude
618     MCContent(9,4:end) = IrregularWaveRes(1,2:end);
                                  % Set period
619     MCContent(10,4:end) = IrregularWaveRes(3,2:end);
                                 % Set direction
620
621     MCContent(11,4:end) = sprintf('%.9E',cgloc_X);
                                    % Set CG component X
622     MCContent(12,4:end) = sprintf('%.9E',cgloc_Y);
                                    % Set CG component X
623     MCContent(13,4:end) = sprintf('%.9E',cgloc_Z);
                                    % Set CG component X
624
625     for a=2:size(IrregularWaveRes,2)
626         MCContent(14,a+2) = sprintf('%.9E',IrregularWaveRes(4,a));
                    % Set linear acceleration component X
627         MCContent(15,a+2) = sprintf('%.9E',IrregularWaveRes(5,a));
                    % Set linear acceleration component Y
628         MCContent(16,a+2) = sprintf('%.9E',IrregularWaveRes(6,a));
                    % Set linear acceleration component Z
629
630         MCContent(17,a+2) = sprintf('%.9E',IrregularWaveRes(7,a));
                    % Set angular acceleration component X
631         MCContent(18,a+2) = sprintf('%.9E',IrregularWaveRes(8,a));
                    % Set angular acceleration component Y
632         MCContent(19,a+2) = sprintf('%.9E',IrregularWaveRes(9,a));
                    % Set angular acceleration component Z
633
634         MCContent(20,a+2) = sprintf('%.9E',IrregularWaveRes(10,a));
                  % Set cgomga component X
635         MCContent(21,a+2) = sprintf('%.9E',IrregularWaveRes(11,a));
                  % Set cgomga component Y
```

```
636         MCContent(22,a+2) = sprintf('%.9E',IrregularWaveRes(12,a));
                    % Set cgomga component Z
637
638         MCContent(23,a+2) = sprintf('%.9E',IrregularWaveRes(13,a));
                    % Set omega component X
639         MCContent(24,a+2) = sprintf('%.9E',IrregularWaveRes(14,a));
                    % Set omega component Y
640         MCContent(25,a+2) = sprintf('%.9E',IrregularWaveRes(15,a));
                    % Set omega component Z
641
642         MCContent(26,a+2) = sprintf('%.9E',IrregularWaveRes(16,a));
                    % Set domega component X
643         MCContent(27,a+2) = sprintf('%.9E',IrregularWaveRes(17,a));
                    % Set domega component Y
644         MCContent(28,a+2) = sprintf('%.9E',IrregularWaveRes(18,a));
                    % Set domega component Z
645
646         MCContent(34,a+2) = sprintf('%.0f.',IrregularWaveRes(19,a));
                   % Set component X for Remote Force
647         MCContent(35,a+2) = sprintf('%.0f.',IrregularWaveRes(20,a));
                   % Set component Y for Remote Force
648         MCContent(36,a+2) = sprintf('%.0f.',IrregularWaveRes(21,a));
                   % Set component Z for Remote Force
649     end
650
651     MCContent(29,4:end) = "TRUE";
                                                    % Set Status for
        Remote Force
652     MCContent(30,4:end) = "6";
                                                       % Set number of
        parameters for Remote Force
653
654     MCContent(31,4:end) = sprintf('%.9E',floc_X);
                                  % Set component X for location of
        Remote Force
655     MCContent(32,4:end) = sprintf('%.9E',floc_Y);
                                  % Set component Y for location of
        Remote Force
656     MCContent(33,4:end) = sprintf('%.9E',floc_Z);
                                  % Set component Z for location of
        Remote Force
657
658 elseif FLWave == 0    % For Wave OFF
```

```
659
660     MCContent(8,4:end) = 0;
                                                          % Set amplitude
661     MCContent(9,4:end) = 0;
                                                          % Set period
662     MCContent(10,4:end) = 0;
                                                         % Set direction
663
664     MCContent(11,4:end) = sprintf('%.9E',cgloc_X);
                                % Set CG component X
665     MCContent(12,4:end) = sprintf('%.9E',cgloc_Y);
                                % Set CG component Y
666     MCContent(13,4:end) = sprintf('%.9E',cgloc_Z);
                                % Set CG component Z
667
668     MCContent(14,4:end) = sprintf('%.9E',0);
                                      % Set linear acceleration
        component X
669     MCContent(15,4:end) = sprintf('%.9E',0);
                                      % Set linear acceleration
        component Y
670     MCContent(16,4:end) = sprintf('%.9E',0);
                                      % Set linear acceleration
        component Z
671
672     MCContent(17,4:end) = sprintf('%.9E',0);
                                      % Set angular acceleration
        component X
673     MCContent(18,4:end) = sprintf('%.9E',0);
                                      % Set angular acceleration
        component Y
674     MCContent(19,4:end) = sprintf('%.9E',0);
                                      % Set angular acceleration
        component Z
675
676     MCContent(20,4:end) = sprintf('%.9E',0);
                                      % Set cgomga component X
677     MCContent(21,4:end) = sprintf('%.9E',0);
                                      % Set cgomga component Y
678     MCContent(22,4:end) = sprintf('%.9E',0);
                                      % Set cgomga component Z
679
680     MCContent(23,4:end) = sprintf('%.9E',0);
```

```
                                        % Set omega component X
681     MCContent(24,4:end) = sprintf('%.9E',0);
                                        % Set omega component Y
682     MCContent(25,4:end) = sprintf('%.9E',0);
                                        % Set omega component Z
683
684     MCContent(26,4:end) = sprintf('%.9E',0);
                                        % Set domega component X
685     MCContent(27,4:end) = sprintf('%.9E',0);
                                        % Set domega component Y
686     MCContent(28,4:end) = sprintf('%.9E',0);
                                        % Set domega component Z
687
688     MCContent(34,4:end) = sprintf('%.0f.',0);
                                       % Set component X for Remote
        Force
689     MCContent(35,4:end) = sprintf('%.0f.',0);
                                       % Set component Y for Remote
        Force
690     MCContent(36,4:end) = sprintf('%.0f.',0);
                                       % Set component Z for Remote
        Force
691
692     MCContent(20,4:end) = "FALSE";
                                                % Set Status for
        Remote Force
693     MCContent(21,4:end) = "6";
                                                    % Set number of
        parameters for Remote Force
694
695     MCContent(22,4:end) = sprintf('%.9E',0);
                                        % Set component X for location
        of Remote Force
696     MCContent(23,4:end) = sprintf('%.9E',0);
                                        % Set component Y for location
        of Remote Force
697     MCContent(24,4:end) = sprintf('%.9E',0);
                                        % Set component Z for location
        of Remote Force
698
699 elseif FB ~= "Wave" && FLWave == 1   % For other FB and Regular Wave (
    only 1st case from regular wave file is considered)
700
```

```
701     MCContent(8,4:end) = RegularWave(2,2);
                                          % Set amplitude
702     MCContent(9,4:end) = RegularWave(3,2);
                                          % Set period
703     MCContent(10,4:end) = RegularWave(4,2);
                                         % Set direction
704
705     MCContent(11,4:end) = sprintf('%.9E',cgloc_X);
                                % Set CG component X
706     MCContent(12,4:end) = sprintf('%.9E',cgloc_Y);
                                % Set CG component Y
707     MCContent(13,4:end) = sprintf('%.9E',cgloc_Z);
                                % Set CG component Z
708
709     MCContent(14,4:end) = sprintf('%.9E',RegularWave(5,2));
                          % Set linear acceleration component X
710     MCContent(15,4:end) = sprintf('%.9E',RegularWave(6,2));
                          % Set linear acceleration component Y
711     MCContent(16,4:end) = sprintf('%.9E',RegularWave(7,2));
                          % Set linear acceleration component Z
712
713     MCContent(17,4:end) = sprintf('%.9E',RegularWave(8,2));
                          % Set angular acceleration component X
714     MCContent(18,4:end) = sprintf('%.9E',RegularWave(9,2));
                          % Set angular acceleration component Y
715     MCContent(19,4:end) = sprintf('%.9E',RegularWave(10,2));
                         % Set angular acceleration component Z
716
717     MCContent(20,4:end) = sprintf('%.9E',RegularWave(11,2));
                         % Set cgomga component X
718     MCContent(21,4:end) = sprintf('%.9E',RegularWave(12,2));
                         % Set cgomga component Y
719     MCContent(22,4:end) = sprintf('%.9E',RegularWave(13,2));
                         % Set cgomga component Z
720
721     MCContent(23,4:end) = sprintf('%.9E',RegularWave(14,2));
                         % Set omega component X
722     MCContent(24,4:end) = sprintf('%.9E',RegularWave(15,2));
                         % Set omega component Y
723     MCContent(25,4:end) = sprintf('%.9E',RegularWave(16,2));
                         % Set omega component Z
724
725     MCContent(26,4:end) = sprintf('%.9E',RegularWave(17,2));
```

```
                        % Set domega component X
726     MCContent(27,4:end) = sprintf('%.9E',RegularWave(18,2));
                        % Set domega component Y
727     MCContent(28,4:end) = sprintf('%.9E',RegularWave(19,2));
                        % Set domega component Z
728
729     MCContent(34,4:end) = sprintf('%.0f.',RegularWave(20,2));
                       % Set component X for Remote Force
730     MCContent(35,4:end) = sprintf('%.0f.',RegularWave(21,2));
                       % Set component Y for Remote Force
731     MCContent(36,4:end) = sprintf('%.0f.',RegularWave(22,2));
                       % Set component Z for Remote Force
732
733     MCContent(29,4:end) = "TRUE";
                                                  % Set Status for
        Remote Force
734     MCContent(30,4:end) = "6";
                                                     % Set number of
        parameters for Remote Force
735
736     MCContent(31,4:end) = sprintf('%.9E',floc_X);
                                  % Set component X for location of
        Remote Force
737     MCContent(32,4:end) = sprintf('%.9E',floc_Y);
                                  % Set component Y for location of
        Remote Force
738     MCContent(33,4:end) = sprintf('%.9E',floc_Z);
                                  % Set component Z for location of
        Remote Force
739
740 elseif FB ~= "Wave" && FLWave == 2   % For other FB and Irregular Wave
    (only 1st case from regular wave file is considered)
741
742     %%% ATENCAO, PRECISA ATUALIZAR ***
743     MCContent(8,4:end) = string(double(IrregularWaveRes(2,2))/2);
                  % Set amplitude
744     MCContent(9,4:end) = IrregularWaveRes(1,2);
                                    % Set period
745     MCContent(10,4:end) = IrregularWaveRes(3,2);
                                   % Set direction
746
747     MCContent(11,4:end) = sprintf('%.9E',cgloc_X);
                                 % Set CG component X
```

```
748     MCContent(12,4:end) = sprintf('%.9E',cgloc_Y);
                                % Set CG component X
749     MCContent(13,4:end) = sprintf('%.9E',cgloc_Z);
                                % Set CG component X
750
751     MCContent(14,a+2) = sprintf('%.9E',IrregularWaveRes(4,2));
                    % Set linear acceleration component X
752     MCContent(15,a+2) = sprintf('%.9E',IrregularWaveRes(5,2));
                    % Set linear acceleration component Y
753     MCContent(16,a+2) = sprintf('%.9E',IrregularWaveRes(6,2));
                    % Set linear acceleration component Z
754
755     MCContent(17,a+2) = sprintf('%.9E',IrregularWaveRes(7,2));
                    % Set angular acceleration component X
756     MCContent(18,a+2) = sprintf('%.9E',IrregularWaveRes(8,2));
                    % Set angular acceleration component Y
757     MCContent(19,a+2) = sprintf('%.9E',IrregularWaveRes(9,2));
                    % Set angular acceleration component Z
758
759     MCContent(20,a+2) = sprintf('%.9E',IrregularWaveRes(10,2));
                   % Set cgomga component X
760     MCContent(21,a+2) = sprintf('%.9E',IrregularWaveRes(11,2));
                   % Set cgomga component Y
761     MCContent(22,a+2) = sprintf('%.9E',IrregularWaveRes(12,2));
                   % Set cgomga component Z
762
763     MCContent(23,a+2) = sprintf('%.9E',IrregularWaveRes(13,2));
                   % Set omega component X
764     MCContent(24,a+2) = sprintf('%.9E',IrregularWaveRes(14,2));
                   % Set omega component Y
765     MCContent(25,a+2) = sprintf('%.9E',IrregularWaveRes(15,2));
                   % Set omega component Z
766
767     MCContent(26,a+2) = sprintf('%.9E',IrregularWaveRes(16,2));
                   % Set domega component X
768     MCContent(27,a+2) = sprintf('%.9E',IrregularWaveRes(17,2));
                   % Set domega component Y
769     MCContent(28,a+2) = sprintf('%.9E',IrregularWaveRes(18,2));
                   % Set domega component Z
770
771     MCContent(34,a+2) = sprintf('%.0f.',IrregularWaveRes(19,2));
                  % Set component X for Remote Force
772     MCContent(35,a+2) = sprintf('%.0f.',IrregularWaveRes(20,2));
```

```
                    % Set component Y for Remote Force
773     MCContent(36,a+2) = sprintf('%.0f.',IrregularWaveRes(21,2));
                    % Set component Z for Remote Force
774
775     MCContent(29,4:end) = "TRUE";
                                                    % Set Status for
        Remote Force
776     MCContent(30,4:end) = "6";
                                                      % Set number of
        parameters for Remote Force
777
778     MCContent(31,4:end) = sprintf('%.9E',floc_X);
                                    % Set component X for location of
        Remote Force
779     MCContent(32,4:end) = sprintf('%.9E',floc_Y);
                                    % Set component Y for location of
        Remote Force
780     MCContent(33,4:end) = sprintf('%.9E',floc_Z);
                                    % Set component Z for location of
        Remote Force
781
782 end
783 LR = size(FileDataWaveOFF,1) - 1;  % Last Row For Wave Info
784
785 %% Update MCContent with SEG information
786 if FLSEG == 1
787     MCContent(LR+1,4:end) = "TRUE";
788     MCContent(LR+2,4:end) = MCContent(LR+2,3);
789     switch Direction_transv_vertical
790         case 'x'
791             MCContent(LR+3,4:end) = "9.80665";
792             MCContent(LR+4,4:end) = "0.";
793             MCContent(LR+5,4:end) = "0.";
794         case 'y'
795             MCContent(LR+3,4:end) = "0.";
796             MCContent(LR+4,4:end) = "9.80665";
797             MCContent(LR+5,4:end) = "0.";
798         case 'z'
799             MCContent(LR+3,4:end) = "0.";
800             MCContent(LR+4,4:end) = "0.";
801             MCContent(LR+5,4:end) = "9.80665";
802     end
803 elseif FLSEG == 0
```

```
804     MCContent(LR+1,4:end) = "FALSE";
805     MCContent(LR+2,4:end) = MCContent(LR+2,3);
806     MCContent(LR+3,4:end) = "0.";
807     MCContent(LR+4,4:end) = "0.";
808     MCContent(LR+5,4:end) = "0.";
809 end
810 LR = LR + (size(FileDataSEGOFF,1) - 1);  % Last Row For Wave Info
811
812 %% Update MCContent with Storage Levels information
813
814 % Update Status
815 GetIdx = find(contains(MCContent(:,2),FileDataStorOFF(2,1)));
816 if FLStor == 1              % Set according to Flag for Storage Levels
817     MCContent(GetIdx,4:end) = "TRUE";
818 else
819     MCContent(GetIdx,4:end) = "FALSE";
820 end
821
822 % Update number of parameters
823 MCContent(GetIdx+1,4:end) = MCContent(GetIdx+1,3);
824
825 R0 = GetIdx+2;
826 R1 = find(contains(MCContent(:,2),FileDataStorOFF(end,1)));
827
828 % Update values according to FileDataStor
829 if FB == "Storage_levels"
830     MCContent(R0:R1,4:end) = FileDataStor(4:end,3:end);
831 elseif FB ~= "Storage_levels" && FLStor == 1
832     for j=4:size(MCContent,2)
833         MCContent(R0:R1,j) = FileDataStor(4:end,2);
834     end
835 elseif FB ~= "Storage_levels" && FLStor == 0
836     for j=4:size(MCContent,2)
837         MCContent(R0:R1,j) = FileDataStorOFF(4:end,2);
838     end
839 end
840
841 % Update Values from Mass (exception: M.,M.,M.)
842 GetIdx = find(contains(MCContent(:,2),'_M***'));
843 for i=1:size(GetIdx,1)
844     for j=3:size(MCContent,2)
845         MCContent(GetIdx(i),j) = strcat(MCContent(GetIdx(i),j),',',
            MCContent(GetIdx(i),j),',',MCContent(GetIdx(i),j));
```

```
846     end
847 end
848
849 %% Save MCContent
850 writematrix(MCContent,strcat(FolderUpdates,'\',MCFile,'.txt'),'
        Delimiter',' ');
851
852 %%  End of code
853 fclose all;
854 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
855 fprintf('Elapsed time running A01_coupled_systems.m is %s \n',
        ElapsedTime)
856 clearvars
857 return
```

C.1.2. Algoritmo A03_assemble

Algoritmo	Objetivo	Categoria
A03_assemble	Atualizar os modelos em elementos finitos	Primário

```
%%  A03_assemble
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   Sep 14, 2020
%   This script/function reads a *.dat file (*_codes.dat or *_codes_beam.dat) from folder "A02 dat base" and update it with information from Options file (file "Options.txt" from folder "A01 coupled systems") and MCContent created by A01_coupled_systems. Then the modified *.dat file is saved in folder "A03 assemble"
%   [pt-br] Este script/ f u n   o  l   um arquivo *.dat (*_codes.dat ou *_codes_beam.dat) da pasta "A02 dat base" e o atualiza com i n f o r m a  e s  %   do arquivo Options (arquivo "Options.txt" da pasta "A01 %   coupled systems") e MCContent criado pelo A01_coupled_systems. Em seguida, o arquivo *.dat modificado salvo na pasta "A03 assemble"
%   Updated on Mar 05, 2021
%   Updated on Apr 22, 2021

%% Notes:
%

%% Steps:
%  Load variables
%  Update APDL Commands - Post  with requested results
%  Copy: Generic FEM model according to options file
%  Read definition of Multiple Cases (MCContent)
%  Assign default values for non-HDP parameters with Status set for FALSE
%  Check hydrodynamic analysis required
%  If HD was required: adjustment according required and available parameters
%  Check for previous results ("A03 assemble\- Results_info.txt")
%  Compare: required cases (MCContent) X previous results (-Results_info); Update information for required simulations
%  Go through simulation cases
%  End of code

```

```
25 %% Load variables
26 fprintf(1,"Initializing A03_assemble \n")
27 tic
28
29 % Check if script was requested by GUI (in Manual or Automatic control)
30 try
31    if control == "Manual"
32        fprintf("Control: GUI - %s \n",control)
33    elseif control == "Automatic"
34        fprintf("Control: GUI - %s \n",control)
35    else
36        control = "None";
37        fprintf("Control: GUI - %s \n",control)
38    end
39 catch
40    control = "None";
41    fprintf("Control: GUI - %s \n",control)
42 end
43
44 % Load variables by running without GUI
45 if control == "None"
46     Ini_A00_initial_data
47     clearvars -except Var_A03_assemble
48
49     FolderLock = Var_A03_assemble{1};% Folder to lock variables after
           first time running scripts
50     PathApdlPostFileOri = Var_A03_assemble{2};% Path for *.txt file
51     PathApdlPostFile = Var_A03_assemble{3};% Path for *.txt file
52     FolderCS = Var_A03_assemble{4};% Folder for coupled systems
53     FileOpt = Var_A03_assemble{5};% Filename for options file
54     FolderInsp = Var_A03_assemble{6};% Folder for inspection
55     PathFemFileCodes = Var_A03_assemble{7};% Path for generic FEM model
           files
56     FolderUpdates = Var_A03_assemble{8};% Folder for *.txt file
57     MCFile = Var_A03_assemble{9};% File for definition of multiple
           cases (automatically generated)
58     FolderFemCases = Var_A03_assemble{10};% Folder for FEM cases
59     ResInfo = Var_A03_assemble{11};% Filename to save results
           information (automatically generated)
60     FolderHDProcessed = Var_A03_assemble{12};% Path for hydrodynamic
           analysis folder
61     PathHDProcessed = Var_A03_assemble{13};% Path for hydrodynamic
           analysis file
```

```
62      MinPha = Var_A03_assemble{14};% Minimum value for phase angle range
63      BrkPha = Var_A03_assemble{15};% Maximum value for phase angle range
64      MaxPha = Var_A03_assemble{16};% Break for phase angle range
65      Unit = Var_A03_assemble{17};% Unit multiplies used for pressure: 1
66      PathFemFileInspection = Var_A03_assemble{18};
67
68  end
69
70  % Check locked variables
71  try
72      load(strcat(FolderLock,'\ReqResStatus_locked.mat'),'
            ReqResStatus_locked');  % Load information
73      ReqResStatus = ReqResStatus_locked;
74  catch
75  %     ReqResStatus_locked = ReqResStatus;
76      save(strcat(FolderLock,'\ReqResStatus_locked.mat'),'
            ReqResStatus_locked');    % Save information
77      fprintf("Locking variable: ReqResStatus \n")
78  end
79
80  %% Update APDL Commands - Post  with requested results
81  fid = fopen(PathApdlPostFileOri,'r','n','UTF-8'); % APDL Commands -
        Post file to be copied
82  fApdl = fread(fid,'*char')';
83  fclose(fid);
84
85  ReqResStatus = ReqResStatus_locked;
86
87  % Turn results off if 'false'
88  if ReqResStatus(1) == false
89      ToBeRep = 'NLIST,,,,COORD           ! print nodes w. coordinates';
90      ToRep = '!NLIST,,,,COORD            ! print nodes w. coordinates';
91      fApdl = strrep(fApdl,ToBeRep,ToRep);
92  end
93
94  if ReqResStatus(2) == false
95      ToBeRep = 'ELIST                            ! print element
            connectivity table';
96      ToRep = '!ELIST                             ! print element
            connectivity table';
97      fApdl = strrep(fApdl,ToBeRep,ToRep);
98  end
99
```

```
100 if ReqResStatus(3) == false
101     ToBeRep = 'PRNSOL,U                 ! print all displacements';
102     ToRep = '!PRNSOL,U                  ! print all displacements';
103     fApdl = strrep(fApdl,ToBeRep,ToRep);
104 end
105
106 if ReqResStatus(4) == false
107     ToBeRep = 'PRNSOL,S                 ! print all stress components';
108     ToRep = '!PRNSOL,S                  ! print all stress components';
109     fApdl = strrep(fApdl,ToBeRep,ToRep);
110 end
111
112 if ReqResStatus(5) == false
113     ToBeRep = 'PRNSOL,EPEL                      ! print all elastic
            strain components';
114     ToRep = '!PRNSOL,EPEL                       ! print all elastic
            strain components';
115     fApdl = strrep(fApdl,ToBeRep,ToRep);
116 end
117
118 if ReqResStatus(6) == false
119     ToBeRep = 'PRNSOL,EPPL                      ! print all plastic
            strain components';
120     ToRep = '!PRNSOL,EPPL                       ! print all plastic
            strain components';
121     fApdl = strrep(fApdl,ToBeRep,ToRep);
122 end
123
124 if ReqResStatus(7) == false
125     ToBeRep = 'PRNSOL,EPTO                      ! print all total
            mechanical strain components (EPEL + EPPL + EPCR)';
126     ToRep = '!PRNSOL,EPTO                       ! print all total
            mechanical strain components (EPEL + EPPL + EPCR)';
127     fApdl = strrep(fApdl,ToBeRep,ToRep);
128 end
129
130 if ReqResStatus(7) == false
131     ToBeRep = 'PRNSOL,EPTO                      ! print all total
            mechanical strain components (EPEL + EPPL + EPCR)';
132     ToRep = '!PRNSOL,EPTO                       ! print all total
            mechanical strain components (EPEL + EPPL + EPCR)';
133     fApdl = strrep(fApdl,ToBeRep,ToRep);
134 end
```

```
135
136 if ReqResStatus(8) == false
137     ToBeRep = 'PRNSOL,S,PRIN    ! print all principal stresses (S1, S2,
            S3 principal stresses, SINT stress intensity, and SEQV
           equivalent stress)';
138     ToRep = '!PRNSOL,S,PRIN     ! print all principal stresses (S1, S2,
            S3 principal stresses, SINT stress intensity, and SEQV
           equivalent stress)';
139     fApdl = strrep(fApdl,ToBeRep,ToRep);
140 end
141
142 fid = fopen(strcat(FolderUpdates,'\',PathApdlPostFile),'w');
143 % fid = fopen(PathApdlPostFile,'w');
144 fprintf(fid,'%s \t',fApdl);
145 fclose(fid);
146
147 %% Copy: Generic FEM model according to options file
148 FLInsp = 0;         % Flag for Inspection status "FALSE"
149 FileName = strcat(FolderCS,'\',FileOpt,'.txt');     % Get .txt file
       name
150 FileDataOpt = extractFileText(FileName);                    % Get data
       from .txt file
151 FileDataOpt = regexprep(FileDataOpt,'\n','&%^');            % Replace
       break lines with codes
152 FileDataOpt = strip(split(FileDataOpt,'&%^'));             % Split
       characters and delete codes
153 FileDataOpt = string(FileDataOpt);                         % Convert
       to string
154 FileDataOpt = regexp(FileDataOpt,'\s','split');
155 if FileDataOpt{end,1} == ""
156     FileDataOpt(end,:) = [];
157 end
158 FileDataOpt = vertcat(FileDataOpt{:});
159
160 Item = "Inspection";
161 ItemIdx = find(Item == FileDataOpt(:,2));
162 if FileDataOpt(ItemIdx,3) ~= "FALSE"
163     FLInsp = 1;         % Flag for Inspection status "TRUE"
164
165     % Get all .txt files from Inspection folder
166     path_directory = strcat(FolderInsp,'\*.txt');
167     TxtFiles = dir(path_directory);
168     InspTxtFilesInfo = struct2cell(TxtFiles);                %
```

```
        Create cell for *.txt files
    InspTxtFilesInfo = InspTxtFilesInfo(1,:)';

    % Get type of inspection file from each .txt file
    for r=1:size(InspTxtFilesInfo,1)
        Filename = string(InspTxtFilesInfo(r,1));
        CheckFilename = split(Filename,["_","."]);
        for s=1:size(CheckFilename,1)
            FilenameBreak = CheckFilename(s,1);
            if strcmpi(FilenameBreak,"shell")
                InspTxtFilesInfo{r,2} = "shell";
            elseif strcmpi(FilenameBreak,"beam")
                InspTxtFilesInfo{r,2} = "beam";
            elseif strcmpi(FilenameBreak,"rect")
                InspTxtFilesInfo{r,3} = "rect";
            elseif strcmpi(FilenameBreak,"l")
                InspTxtFilesInfo{r,3} = "l";
            end
        end

        fid = fopen(strcat(FolderInsp,'\',Filename), 'r','n','UTF-8');
                    % dat file from mechanical to be read
        data = readtable(strcat(FolderInsp,'\',Filename));
        fclose(fid);
        InspTxtFilesInfo{r,4} = data;
        if InspTxtFilesInfo{r,2} == "shell"
            InspShell = data;
        elseif InspTxtFilesInfo{r,2} == "beam" && InspTxtFilesInfo{r,3}
            == "rect"
            InspBeamRect = data;
        elseif InspTxtFilesInfo{r,2} == "beam" && InspTxtFilesInfo{r,3}
            == "l"
            InspBeamL = data;
        end
    end

else
    FLInsp = 0;         % Flag for Inspection status "FALSE"
end

% Load different file if Inspection is TRUE
if FLInsp == 0
    fid = fopen(PathFemFileCodes, 'r','n','UTF-8');          % dat file
```

```
        from mechanical to be read
    f=fread(fid,'*char')';
    fclose(fid);
else
    fid = fopen(PathFemFileInspection, 'r','n','UTF-8'); % dat file
       from mechanical to be read
    f=fread(fid,'*char')';
    fclose(fid);
end

% Update FEM model with APDL Commands - Post
fid = fopen(strcat(FolderUpdates,'\',PathApdlPostFile), 'r','n','UTF-8'
   ); % APDL Commands - Post to be read
% fid = fopen(PathApdlPostFile, 'r','n','UTF-8'); % APDL Commands -
   Post to be read
fApdl=fread(fid,'*char')';
fclose(fid);

f = strrep(f,'/post1',fApdl);

% return
%% Read definition of Multiple Cases (MCContent)
%   Read: Definition of Multiple Cases
fid = fopen(strcat(FolderUpdates,'\',MCFile,'.txt'),'r');
                           % Open file
MCContent = [];                                    % Create array
while ~feof(fid)                                   % Read until end of
    file
    tline = fgetl(fid);
    tlinesplit = regexp(tline,'\s','split');       % Whitespace
       delimiter
    MCContent = [MCContent;string(tlinesplit)];    % Update array
end
fclose(fid);

Codes = MCContent(:,2);                            % List of codes

%% Assign default values for non-HDP parameters with Status set for
   FALSE
for r=11:size(MCContent,1)                     % Check non-HDP
   parameters with FALSE for Status
    CurrentCode = MCContent(r,2);
    try                                        % Check first letters
```

```
242         CheckLetters = char(CurrentCode(1,1));
243         CheckFL = char(CheckLetters(1:9));
244         if CheckFL == '***Status'               % Check if it is a
                Status parameter
245             for c=4:size(MCContent,2)
246                 NumPar = str2num(MCContent(r+1,c));             %
                        Number of parameters
247                 if MCContent(r,c) == "FALSE"                    % Check
                        is Status is set to FALSE
248                     MCContent(r+2:r+1+NumPar,c) = MCContent(r+2:r+1+
                            NumPar,3); % Repalce by default values
249                 end
250             end
251         end
252     end
253 end
254
255 %% Check hydrodynamic analysis required
256 HdpCodeStatus = "***Status_HDP***";             % Code for hydrodynamic
        analysis
257 IdxHdpCode = find(Codes == HdpCodeStatus);      % Index code
258 HdpCodeStatusAll = MCContent(IdxHdpCode,4:end); % Get all Status values
        for HDP (1 for pressure on nodes; 2 for pressure on elements)
259
260 for i=4:size(HdpCodeStatusAll,2)                % Inform user if any
        hydrodynamic analysis is off
261     if HdpCodeStatusAll(1,i) ~= "1" & HdpCodeStatusAll(1,i) ~= "2"
262         fprintf('Case %s: HDP is OFF. \n',MCContent(1,i+3));
263     end
264 end
265
266 %% If HD was required: adjustment according required and available
        parameters
267 % Load HD processed file if required
268 if ismember("1",HdpCodeStatusAll) || ismember("2",HdpCodeStatusAll)
269     try
270         load(PathHDProcessed);    % Load HD processed file
271     catch
272         fprintf('There is no Hydrodynamic analysis data in folder: %s',
                FolderHDProcessed)
273         return
274     end
275
```

```
276 % Check and load equivalent entity list from structural and HD meshes (
    nodes or elements)
277     try
278         load(strcat(FolderLock,'\Equivalent_Entity_List.mat'));
                % HD equivalent mesh
279     catch
280         fprintf('There is no information for equivalent entity used for
                Hydrodynamic analysis in folder: %s',FolderHDProcessed)
281 
282         prompt = 'Do you want to generate information equivalent entity
                list? Y/N [Y]: ';
283         UserAns = input(prompt,'s');
284         if UserAns == "Y" | UserAns == "y" | UserAns == "Yes" | UserAns
                == "YES" | UserAns == "yes"
285             Ini_A01_equiv    % Call script to generate info
286         else
287             disp('Information for  equivalent entities is required to
                   proceed');
288             return
289         end
290     end
291 
292     MatFilesInfo = struct2cell(Hull_Pressure);      % Create cell for
        *.mat files
293 
294     StructuFieldNames = fieldnames(Hull_Pressure);
295     StructuFieldNames = convertCharsToStrings(StructuFieldNames);
296 
297     IdxField = find("Period" == StructuFieldNames);
298     InfoPeriodsHD = cat(1, MatFilesInfo{IdxField:IdxField});       %
        Get info for periods
299 
300     IdxField = find("Incidence" == StructuFieldNames);
301     InfoDirectionsHD = cat(1, MatFilesInfo{IdxField:IdxField});       %
         Get info for wave directions
302 
303     IdxField = find("Hs" == StructuFieldNames);
304     InfoAmplitudeHD = cat(1, MatFilesInfo{IdxField:IdxField})/2;     %
        Get info for amplitude
305 
306     IdxField = find("FaceVertexCData" == StructuFieldNames);
307 %     InfoHDNodes = cat(1, MatFilesInfo{1:1});     % Get info for nodes
308 
```

```
309     RequiredPeriod = [];
310     RequiredDirection = [];
311 
312     for i=4:size(MCContent,2)
313         CheckHD = MCContent(3,i);
314         if CheckHD ~= "0"                    % Pressure applied to NODES
315             % Check closest available value for Period
316             RequiredPeriod = str2double(MCContent(9,i));
317             [val,idx] = min(abs(InfoPeriodsHD-RequiredPeriod));
318             ClosestVal = InfoPeriodsHD(idx);
319 
320             if ClosestVal ~= RequiredPeriod
321                 fprintf('Case %s: Required Period %f was changed to
                        closest available value: %f \n',MCContent(1,i),
                        RequiredPeriod,ClosestVal)
322                 MCContent(9,i) = ClosestVal;
323             end
324 
325             % Check closest available value for Direction
326             RequiredDirection = str2double(MCContent(10,i));
327             [val,idx]=min(abs(InfoDirectionsHD-RequiredDirection));
328             ClosestVal=InfoDirectionsHD(idx);
329             if ClosestVal ~= RequiredDirection
330                 fprintf('Case %s: Required Direction %f was changed to
                        closest available value: %f \n',MCContent(1,i),
                        RequiredDirection,ClosestVal)
331                 MCContent(10,i) = ClosestVal;
332             end
333         end
334     end
335     RequiredPeriod = [];
336     RequiredDirection = [];
337 end
338 
339 %% Check for previous results ("A03 assemble\- Results_info.txt")
340 try % Try to open Data Base file
341     fid = fopen(strcat(FolderFemCases,'\',ResInfo,'.txt'),'r');     %
        Open file
342     ResContent = [];
343     while ~feof(fid)                                                %
        Read until end of file
344         tline = fgetl(fid);
345         tlinesplit = regexp(tline,'\s','split');                    %
```

```
            Whitespace delimiter
346         ResContent = [ResContent;string(tlinesplit)];                %
            Update array
347     end
348
349     fclose(fid);
350 end
351
352 %% Compare: required cases (MCContent) X previous results (-
    Results_info); Update information for required simulations
353 BatchNumber = 0;     % Number of simulation in batch mode
354 if isempty(ResContent)      % If there is no Data Base = 1st batch mode
355     BatchNumber = BatchNumber + 1;                                   %
        Start number of simulation in batch mode
356     ResContent = MCContent;                                          %
        Copy content from required cases (Multiple Cases)
357     ResContent(1,4:end) = strcat(string(BatchNumber),".",ResContent
        (1,4:end));  % Append Batch Number
358     MCContent(1,4:end) = strcat(string(BatchNumber),".",MCContent(1,4:
        end));    % Append Batch Number
359     ResContent(:,3) = "-";                                           %
        Erase default values to be saved
360
361     % Save Data Base file
362     WriteResContent = fopen(strcat(FolderFemCases,'\',ResInfo,'.txt'),'
        w');
363
364     for i=1:size(ResContent,1)
365         fprintf(WriteResContent,'%s \n',ResContent(i,:));
366     end
367     fclose(WriteResContent);
368
369 else                        % If there is a Data Base
370     MaxBatch = 0;
371     for i=4:size(ResContent,2) % Find max number for batch
372         Aux = split(ResContent(1,i),'.');
373         if str2double(Aux(1)) > MaxBatch
374             MaxBatch = str2num(Aux(1));
375         end
376     end
377
378     BatchNumber = MaxBatch + 1;     % Number of simulation in batch
        mode
```

```
379
380     % Compare requested cases (MCContent) to Results Data Base (
            ResContent)
381     Size2MCContent = size(MCContent,2);             % Get current size
            of MCContent
382     Size2ResContent = size(ResContent,2);           % Get current size
            of ResContent
383     RepeatedCase = [];                              % Variable to check
             if it is a repeated case
384     AuxMCContent = [];                              % Aux variable for
            MCContent
385     AuxResContent = [];                             % Aux variable for
            ResContent
386     for i=4:Size2MCContent                          % Check each case
            from requested cases
387         for j=4:Size2ResContent                     % Check each case
                from data base
388             RepeatedCase = isequal(MCContent(2:end,i),ResContent(2:end,
                    j)); % Compare requested cases with data base
389             if RepeatedCase == 1                    % Get info for
                    reapeated case
390                 AuxMCContent = i;
391                 AuxResContent = j;
392             end
393         end
394
395         if isempty(AuxMCContent)                    % It's not a
                repeated case
396             MCContent(1,i) = strcat(string(BatchNumber),".", MCContent
                    (1,i));
397             ResContent = [ResContent';MCContent(:,i)'];
398         else                                        % It's a repeated
                case
399             fprintf('Case %s: Result for this case already exists. \n',
                    MCContent(1,i));
400             ResContent = [ResContent';MCContent(:,AuxMCContent)'];
401             ResContent(1,size(ResContent,2)) = ResContent(1,
                    AuxResContent);
402             MCContent(2,AuxMCContent) = "FALSE";
403             fprintf('Case %s: Case Status was set to False. \n',
                    MCContent(1,i));
404         end
405
```

```
406         % Erase variables
407         RepeatedCase = [];
408         AuxMCContent = [];
409         AuxResContent = [];
410     end
411 end
412 
413 fprintf("Writing cases for simulation in batch mode: %d \n",BatchNumber
    );
414 
415 % Save Data Base file
416 %     writematrix(ResContent,strcat(PathFemCases,ResInfo),'Delimiter
    ','\t');  % Save Data Base file  % This command doesn't work for
    old Matlab versions
417 WriteResContent = fopen(strcat(FolderFemCases,'\',ResInfo,'.txt'),'w');
418 for i=1:size(ResContent,1)
419     fprintf(WriteResContent,'%s ',ResContent(i,:));
420     fprintf(WriteResContent,'\n',ResContent(i,:));
421 end
422 fclose(WriteResContent);
423 
424 %% Go through simulation cases
425 CheckFL = [];
426 CheckLL = [];
427 for  c=4:size(MCContent,2)                                  % Through
    columns Status_Case
428     if MCContent(2,c) == "TRUE"
429         CopyFmin = f;                                       % Copy input
            file with codes for min pres case
430         CopyFmax = f;                                       % Copy input
            file with codes for max pres case
431         CopyFNoHDP = f;                                     % Copy input
            file with codes for case without HDP
432 
433         for r=3:size(MCContent,1)                           % Through rows
434             CurrentCode = MCContent(r,2);
435             try                                             % Check first
                letters
436                 CheckLetters = char(CurrentCode(1,1));
437                 CheckFL = CheckLetters(1:9);
438             catch
439                 % do nothing
440             end
```

```
441
442             if CheckFL == '***Status'                    % Check if it
                    is a Status parameter
443                 try                                      % Check last
                        letters
444                     CheckLL = CheckLetters(11:16);
445                 catch
446                     CheckLL = 'NANANA';
447                 end
448
449                 if CheckLL == 'HDP***'                   % Check if it
                        is HDP parameter
450                     if MCContent(3,c) == "1"
451
452                         NumPar = str2num(MCContent(r+1,c));
                                        % Number of parameters
453                         HdpCodes = MCContent(r+2:r+1+NumPar,2);
                                % HDP codes
454                         HdpValues = MCContent(r+2:r+1+NumPar,c);
                                % HDP values
455
456
457
458                         InterpMethod = HdpValues(1,1);
                                                % Interpolation method
                                by user case
459 %                     if InterpMethod ~= "0"
460 %                         if InterpMethod ~= "1" & InterpMethod ~= "2"
461 %                             InterpMethod  = MCContent(5,3);
                        % Interpolation method by user default
462 %                             if InterpMethod ~= "1" & InterpMethod ~=
    "2"
463 %                                 InterpMethod  = "2";
                            % Interpolation method by global default
464 %                                 fprintf("Inputs by user case and user
     default were invalid for Interpolation Method. Interpolation
    Method was set to global default: 2.");
465 %                             end
466 %                         end
467
468                         CaseMinStatus = HdpValues(2,1);
                                                % Case Min
                            Pressure Status by user case
```

```
469                     CaseMaxStatus = HdpValues(3,1);
                                                % Case Max
                        Pressure Status by user case
470
471                     if CaseMinStatus ~= "TRUE" & CaseMaxStatus ~= "
                        TRUE"
472                         CaseMinStatus = MCContent(6,3);
                                                % Case Min
                            Pressure Status by user default
473                         CaseMaxStatus = MCContent(7,3);
                                                % Case Max
                            Pressure Status by user default
474                         if CaseMinStatus ~= "TRUE" & CaseMaxStatus
                            ~= "TRUE"
475                             CaseMinStatus = "TRUE";
                                                    % Case
                                Min Pressure Status by global
                                default
476                             CaseMaxStatus = "TRUE";
                                                    % Case
                                Max Pressure Status by global
                                default
477                         end
478                     end
479
480                     RequiredAmplitude = str2num(HdpValues(4,1));
                                    % Required amplitude
481                     RequiredPeriod = str2num(HdpValues(5,1));
                                    % Required amplitude
482                     RequiredDirection = str2num(HdpValues(6,1));
                                    % Required amplitude
483
484                     % Get pressures according to required
                        parameters
485                     counter = 0;
486                     PhaseAngleInfo = [];        % Matrix to append
                        information of phase angles
487                     PressureCaseAppend = {[]};  % Cell to append
                        pressures for each phase angle
488
489                                 %%%%% &&&&& FE_model
490                                 %%%%% load('G:\Shared
                                        drives\Poli MSc
```

```
                                                    Kennedy\ANSYS\
                                                    all_v12\A01 coupled
                                                     systems\FE_model\
                                                    Wave\HD\
                                                    FE_model_Hull_Pressure
                                                    .mat')
491                                                %%%%% MinPha = 0;
492                                                %%%%% BrkPha = 10;
493                                                %%%%% MaxPha = 360;
494                                                %%%%% RequiredAmplitude
                                                        = 3;
495                                                %%%%% RequiredPeriod =
                                                    10;
496                                                %%%%% RequiredDirection
                                                        = 0;
497                                                %%%%% pha = 260;
498                                                %%%%% PressureCase =
                                                    Calc_Press(
                                                    RequiredAmplitude
                                                    *2,RequiredPeriod,
                                                    RequiredDirection,
                                                    pha,Hull_Pressure);
                                                      % Get HD
                                                    Pressures
499
500                                    for pha = MinPha:BrkPha:MaxPha    % loop
                                           according to phase angle range and
                                           intervals
501                                         counter = counter + 1;
502                                         % Pressure = Calc_Press(Hs [m], Period [s],
                                              Direction [deg], Phase [deg],
                                             Hull_Pressure)
503                                         PressureCase = Calc_Press(RequiredAmplitude
                                             *2,RequiredPeriod,RequiredDirection,pha
                                             ,Hull_Pressure);  % Get HD Pressures
504
505                                         if MCContent(3,c) == "1"            % HDP
                                             pressure applied to NODES
506                                          PressureCase = PressureCase(EquivNodes
                                                 (:,2));                          %
                                                 Delete pressures for nodes out of
                                                 mesh
507                                          PressureCase = [EquivNodes(:,1:2)';
```

```
                    PressureCase'];
                                        % Append Fem
                    element ID and HD element ID to
                    pressures
508             elseif MCContent(3,c) == "2"            %
                    HDP pressure applied to ELEMENTS
509                 PressureCase = PressureCase(
                    EquivElements(:,2));
                                        % Delete
                    pressures for elements out of mesh
510                 PressureCase = [EquivElements(:,1:2)';
                    PressureCase'];
                                        % Append Fem
                    element ID and HD element ID to
                    pressures
511             end
512
513             PhaseAngleInfo = [PhaseAngleInfo;pha];   %
                    Append information of phase angles
514             PressureCaseAppend{[counter]} =
                    PressureCase; % Append pressures for
                    each phase angle
515         end
516
517         % Get phase angle with min pressure
518
519         MinPres = +1E+99;       % Aux variable to check
                min pressure
520         MinPhaPres = [];           % Aux variable to
                phase angle for min pressure
521         MinPhaIdx = [];                    % Aux
                variable to get idx of phase angle
522         for pc=1:size(PhaseAngleInfo,1)
523             MinValue = min(PressureCaseAppend{pc}(:,3))
                    ;
524             if MinValue < MinPres
525                 MinPres = MinValue;
526                 MinPhaPres = PhaseAngleInfo(pc);
527                 MinPhaIdx = pc;
528             end
529         end
530
531         CaseMinPres = PressureCaseAppend{MinPhaIdx};
```

```
532
533                         CaseMinPres = sortrows(CaseMinPres,1);
534                         CaseMinPres(:,1:2) = [];
535                         CaseMinPres = CaseMinPres * Unit;
536
537                         % Write input case for min pressure
538                         ExpToFind = 'outres,misc,all,_elmisc';       %
                               After this line command snippet will be
                               inserted
539                         InsertPressureMin = [ExpToFind newline '!
                               ****** Begin Command Snippet ******'];
540                         CaseInfo = ['! Amplitude: ' num2str(
                               RequiredAmplitude) ' m, Frequency: '
                               num2str(1/RequiredPeriod) ' Hz / ' num2str(
                               RequiredPeriod) ' sec, Direction: ' num2str
                               (RequiredDirection) '  , Phase Angle: '
                               num2str(MinPhaPres) ' [degrees]'];
541                         InsertPressureMin = [InsertPressureMin newline
                               CaseInfo];
542
543                         if CaseMaxStatus == "TRUE"
544                             InsertPressureMax = [ExpToFind newline '!
                                   ****** Begin Command Snippet ******'];
545                             % Invert phase angle
546                             MaxPhaPres = MinPhaPres - 180;
547                             if MaxPhaPres < 0
548                                 MaxPhaPres = 360 + MaxPhaPres;
549                             end
550                             CaseInfo = ['! Amplitude: ' num2str(
                                   RequiredAmplitude) ' m, Frequency: '
                                   num2str(1/RequiredPeriod) ' Hz / '
                                   num2str(RequiredPeriod) ' sec,
                                   Direction: ' num2str(RequiredDirection)
                                    '  , Phase Angle: ' num2str(MaxPhaPres
                                   ) ' [degrees]'];
551                             InsertPressureMax = [InsertPressureMax
                                   newline CaseInfo];
552                         end
553
554                         if MCContent(3,c) == "1"            % HDP
                               pressure applied to NODES
555                             SfeInfoMin = string([]);
556                             SfeInfoMax = string([]);
```

```
557
558                         SfeInfoMin(:,2) = string(FEMElements(:,1));
                                %ElementsHull = FEMElements
559                         SfeInfoMin(:,1) = 'sfe,  ';
560                         SfeInfoMin(:,3) = ',     2,pres,0,  ';
561
562                         SfeInfoMin(:,5) = ',  ';
563                         SfeInfoMin(:,7) = ',  ';
564                         SfeInfoMin(:,9) = ',  ';
565
566                         NodesNeeded = str2double(FEMElements(:,3:
                               end));
567                         HullNodePressure = sortrows(EquivNodes,2);
568                         HullNodePressure(:,3) = CaseMinPres;
569                         HullNodePressure(:,2) = [];
570
571                         [~,idx] = ismember(NodesNeeded(:,1),
                               HullNodePressure(:,1));
572                         PressureUpdate = HullNodePressure(idx,2);
573                         SfeInfoMin(:,4) = compose("%1.9E",
                               PressureUpdate);
574
575                         [~,idx] = ismember(NodesNeeded(:,2),
                               HullNodePressure(:,1));
576                         PressureUpdate = HullNodePressure(idx,2);
577                         SfeInfoMin(:,6) = compose("%1.9E",
                               PressureUpdate);
578
579                         [~,idx] = ismember(NodesNeeded(:,3),
                               HullNodePressure(:,1));
580                         PressureUpdate = HullNodePressure(idx,2);
581                         SfeInfoMin(:,8) = compose("%1.9E",
                               PressureUpdate);
582
583                         [~,idx] = ismember(NodesNeeded(:,4),
                               HullNodePressure(:,1));
584                         PressureUpdate = HullNodePressure(idx,2);
585                         SfeInfoMin(:,10) = compose("%1.9E",
                               PressureUpdate);
586
587                         if CaseMaxStatus == "TRUE"
588                             SfeInfoMax = SfeInfoMin;
589                             SfeInfoMax(:,4) = compose("%1.9E",
```

```
                                    string(str2double(SfeInfoMin(:,4))
                                    * -1));
590                             SfeInfoMax(:,6) = compose("%1.9E",
                                    string(str2double(SfeInfoMin(:,6))
                                    * -1));
591                             SfeInfoMax(:,8) = compose("%1.9E",
                                    string(str2double(SfeInfoMin(:,8))
                                    * -1));
592                             SfeInfoMax(:,10) = compose("%1.9E",
                                    string(str2double(SfeInfoMin(:,10))
                                     * -1));
593
594                             SfeInfoMax(:,11) = '###';
595                             SfeInfoMax(end,11) = '';
596                             SfeInfoMax = strjoin(SfeInfoMax');
597
598                             InsertPressureMax = [InsertPressureMax
                                    newline char(SfeInfoMax)];
599
600                             InsertPressureMax = regexprep(
                                    InsertPressureMax,'###','\n');
601                             InsertPressureMax = strrep(
                                    InsertPressureMax,' ,',',');
602                             InsertPressureMax = strrep(
                                    InsertPressureMax,' sfe,','sfe,');
603                         end
604
605                         SfeInfoMin(:,11) = '###';
606                         SfeInfoMin(end,11) = '';
607                         SfeInfoMin = strjoin(SfeInfoMin');
608
609                         InsertPressureMin = [InsertPressureMin
                                newline char(SfeInfoMin)];
610
611                         InsertPressureMin = regexprep(
                                InsertPressureMin,'###','\n');
612                         InsertPressureMin = strrep(
                                InsertPressureMin,' ,',',');
613                         InsertPressureMin = strrep(
                                InsertPressureMin,' sfe,','sfe,');
614
615                     elseif MCContent(3,c) == "2"            % HDP
                            pressure applied to ELEMENTS
```

```
616
617                         FEMElementPressure = sortrows(EquivElements
                                ,2);
618                         FEMElementPressure(:,3) = CaseMinPres;
619                         FEMElementPressure(:,2) = [];
620                         FEMElementPressure = sortrows(
                                FEMElementPressure,1);
621
622                         SfeInfoMin = string([]);
623                         SfeInfoMax = string([]);
624
625                         SfeInfoMin(:,2) = string(FEMElementPressure
                                (:,1));
626                         SfeInfoMin(:,1) = 'sfe,  ';
627                         SfeInfoMin(:,3) = ',     2,pres,0,  ';
628
629                         SfeInfoMin(:,5) = ',  ';
630                         SfeInfoMin(:,7) = ',  ';
631                         SfeInfoMin(:,9) = ',  ';
632
633                         SfeInfoMin(:,4) = compose("%1.9E",
                                FEMElementPressure(:,2));
634                         SfeInfoMin(:,6) = compose("%1.9E",
                                FEMElementPressure(:,2));
635                         SfeInfoMin(:,8) = compose("%1.9E",
                                FEMElementPressure(:,2));
636                         SfeInfoMin(:,10) = compose("%1.9E",
                                FEMElementPressure(:,2));
637
638                         if CaseMaxStatus == "TRUE"
639                             SfeInfoMax = SfeInfoMin;
640
641                             SfeInfoMin(:,4) = compose("%1.9E",
                                    FEMElementPressure(:,2) * -1);
642                             SfeInfoMin(:,6) = compose("%1.9E",
                                    FEMElementPressure(:,2) * -1);
643                             SfeInfoMin(:,8) = compose("%1.9E",
                                    FEMElementPressure(:,2) * -1);
644                             SfeInfoMin(:,10) = compose("%1.9E",
                                    FEMElementPressure(:,2) * -1);
645
646                             SfeInfoMax(:,11) = '###';
647                             SfeInfoMax(end,11) = '';
```

```
648                                 SfeInfoMax = strjoin(SfeInfoMax');
649
650                                 InsertPressureMax = [InsertPressureMax
                                        newline char(SfeInfoMax)];
651
652                                 InsertPressureMax = regexprep(
                                        InsertPressureMax,'###','\n');
653                                 InsertPressureMax = strrep(
                                        InsertPressureMax,' ,',',');
654                                 InsertPressureMax = strrep(
                                        InsertPressureMax,' sfe,','sfe,');
655                             end
656
657                             SfeInfoMin(:,11) = '###';
658                             SfeInfoMin(end,11) = '';
659                             SfeInfoMin = strjoin(SfeInfoMin');
660
661                             InsertPressureMin = [InsertPressureMin
                                    newline char(SfeInfoMin)];
662
663                             InsertPressureMin = regexprep(
                                    InsertPressureMin,'###','\n');
664                             InsertPressureMin = strrep(
                                    InsertPressureMin,' ,',',');
665                             InsertPressureMin = strrep(
                                    InsertPressureMin,' sfe,','sfe,');
666
667                         end
668
669                         NumParHDP =  str2double(MCContent(4,c)); %
                                Number of parameters for HDP
670
671                             % CG input
672                         RowParamater = find("***cgloc_X***" ==
                                MCContent(:,2));
673     %                       if str2double(MCContent(RowParamater,c))
        == 0 && str2double(MCContent(RowParamater+1,c)) == 0 &&
        str2double(MCContent(RowParamater+2,c)) == 0
674                             cg_info = strcat("cgloc,  ",MCContent(
                                    RowParamater,c),",  ",MCContent(
                                    RowParamater+1,c),",  ",MCContent(
                                    RowParamater+2,c));
675     %                       end
```

```
676
677                         % Acceleration input
678                         RowParamater = find("***acel_X***" == MCContent
                                (:,2));
679                         if str2double(MCContent(RowParamater,c)) == 0
                                && str2double(MCContent(RowParamater+1,c))
                                == 0 && str2double(MCContent(RowParamater
                                +2,c)) == 0
680                             acel_info = "! acel,    0.000000000E+00,
                                    0.000000000E+00,  0.000000000E+00";
681                         else
682                             acel_info = strcat("acel,    ",MCContent(
                                    RowParamater,c),",   ",MCContent(
                                    RowParamater+1,c),",   ",MCContent(
                                    RowParamater+2,c));
683                         end
684
685                         % Ang. Acceleration input
686                         RowParamater = find("***dcgomg_X***" ==
                                MCContent(:,2));
687                         if str2double(MCContent(RowParamater,c)) == 0
                                && str2double(MCContent(RowParamater+1,c))
                                == 0 && str2double(MCContent(RowParamater
                                +2,c)) == 0
688                             dcgomg_info = "! dcgomg,    0.000000000E+00,
                                     0.000000000E+00,  0.000000000E+00";
689                         else
690                             dcgomg_info = strcat("dcgomg,    ",MCContent
                                    (RowParamater,c),",   ",MCContent(
                                    RowParamater+1,c),",   ",MCContent(
                                    RowParamater+2,c));
691                         end
692
693                         % cgomga input
694                         RowParamater = find("***cgomga_X***" ==
                                MCContent(:,2));
695                         if str2double(MCContent(RowParamater,c)) == 0
                                && str2double(MCContent(RowParamater+1,c))
                                == 0 && str2double(MCContent(RowParamater
                                +2,c)) == 0
696                             cgomga_info = "! cgomga,    0.000000000E+00,
                                     0.000000000E+00,  0.000000000E+00";
697                         else
```

```
698                         cgomga_info = strcat("cgomga,    ",MCContent
                                (RowParamater,c),",   ",MCContent(
                                RowParamater+1,c),",   ",MCContent(
                                RowParamater+2,c));
699                     end
700
701                     % omega input
702                     RowParamater = find("***omega_X***" ==
                            MCContent(:,2));
703                     if str2double(MCContent(RowParamater,c)) == 0
                            && str2double(MCContent(RowParamater+1,c))
                            == 0 && str2double(MCContent(RowParamater
                            +2,c)) == 0
704                         omega_info = "! omega    0.000000000E+00,
                                0.000000000E+00,  0.000000000E+00";
705                     else
706                         omega_info = strcat("omega,    ",MCContent(
                                RowParamater,c),",   ",MCContent(
                                RowParamater+1,c),",   ",MCContent(
                                RowParamater+2,c));
707                     end
708
709                     % domega input
710                     RowParamater = find("***domega_X***" ==
                            MCContent(:,2));
711                     if str2double(MCContent(RowParamater,c)) == 0
                            && str2double(MCContent(RowParamater+1,c))
                            == 0 && str2double(MCContent(RowParamater
                            +2,c)) == 0
712                         domega_info = "! domega    0.000000000E+00,
                                0.000000000E+00,  0.000000000E+00";
713                     else
714                         domega_info = strcat("domega,    ",MCContent
                                (RowParamater,c),",   ",MCContent(
                                RowParamater+1,c),",   ",MCContent(
                                RowParamater+2,c));
715                     end
716
717                     % Add information for acceleration, cg location
                            and rotational acceleration
718                     InsertPressureMin = [InsertPressureMin newline
                            char(acel_info)];
719                     InsertPressureMin = [InsertPressureMin newline
```

```
                char(cg_info)];
720             InsertPressureMin = [InsertPressureMin newline
                char(dcgomg_info)];
721             InsertPressureMin = [InsertPressureMin newline
                char(cgomga_info)];
722             InsertPressureMin = [InsertPressureMin newline
                char(omega_info)];
723             InsertPressureMin = [InsertPressureMin newline
                char(domega_info)];
724
725             InsertPressureMin = [InsertPressureMin newline
                '/prep7'];
726             InsertPressureMin = [InsertPressureMin newline
                'etcontrol,off'];
727             InsertPressureMin = [InsertPressureMin newline
                '*get,_lastetyp,etyp,,num,max'];
728             InsertPressureMin = [InsertPressureMin newline
                '*do,ii,1,_lastetyp'];
729             InsertPressureMin = [InsertPressureMin newline
                '  *get,_etyp,etyp,ii,attr,enam'];
730             InsertPressureMin = [InsertPressureMin newline
                '  *if,_etyp,eq,188,then'];
731             InsertPressureMin = [InsertPressureMin newline
                '  keyopt,ii,3,3'];
732             InsertPressureMin = [InsertPressureMin newline
                '  *endif'];
733             InsertPressureMin = [InsertPressureMin newline
                '*enddo'];
734             InsertPressureMin = [InsertPressureMin newline
                'etlist'];
735             InsertPressureMin = [InsertPressureMin newline
                '/solu'];
736             InsertPressureMin = [InsertPressureMin newline
                '! ****** End   Command Snippet ******'];
737
738             if CaseMaxStatus == "TRUE"
739                 InsertPressureMax = [InsertPressureMax
                    newline char(acel_info)];
740                 InsertPressureMax = [InsertPressureMax
                    newline char(cg_info)];
741                 InsertPressureMax = [InsertPressureMax
                    newline char(dcgomg_info)];
742                 InsertPressureMax = [InsertPressureMax
```

```
                                    newline char(cgomga_info)];
743                                 InsertPressureMax = [InsertPressureMax
                                    newline char(omega_info)];
744                                 InsertPressureMax = [InsertPressureMax
                                    newline char(domega_info)];
745
746                                 InsertPressureMax = [InsertPressureMax
                                    newline '/prep7'];
747                                 InsertPressureMax = [InsertPressureMax
                                    newline 'etcontrol,off'];
748                                 InsertPressureMax = [InsertPressureMax
                                    newline '*get,_lastetyp,etyp,,num,max'
                                    ];
749                                 InsertPressureMax = [InsertPressureMax
                                    newline '*do,ii,1,_lastetyp'];
750                                 InsertPressureMax = [InsertPressureMax
                                    newline '  *get,_etyp,etyp,ii,attr,enam
                                    '];
751                                 InsertPressureMax = [InsertPressureMax
                                    newline '  *if,_etyp,eq,188,then'];
752                                 InsertPressureMax = [InsertPressureMax
                                    newline '  keyopt,ii,3,3'];
753                                 InsertPressureMax = [InsertPressureMax
                                    newline '  *endif'];
754                                 InsertPressureMax = [InsertPressureMax
                                    newline '*enddo'];
755                                 InsertPressureMax = [InsertPressureMax
                                    newline 'etlist'];
756                                 InsertPressureMax = [InsertPressureMax
                                    newline '/solu'];
757                                 InsertPressureMax = [InsertPressureMax
                                    newline '! ****** End   Command Snippet
                                     ******'];
758
759                                 CopyFmax = strrep(CopyFmax,ExpToFind,
                                    InsertPressureMax);  % Insert Pressures
760                             end
761
762                             CopyFmin = strrep(CopyFmin,ExpToFind,
                                InsertPressureMin);  % Insert Pressures
763     %                       end
764
765                             CheckLL = [];
```

```
766                     end
767                 else
768                     NumPar = str2num(MCContent(r+1,c));             %
                        Number of parameters
769                     RepCodes = MCContent(r+2:r+1+NumPar,2);         %
                        Codes to replace
770                     RepValues = MCContent(r+2:r+1+NumPar,c);
                        % Values to replace
771
772                     % Replacing code by values for case of min pressure
773                     try
774                         if CaseMinStatus == "TRUE"
775                             for w = 1:size(RepCodes,1)
776                                 CopyFmin = strrep(CopyFmin,RepCodes(w),
                                    RepValues(w));
777
778                                 % Update inspection data if required
779                                 if FLInsp == 1
780
781                                         % Update for shells
782                                         SheetProp = ['/com,***********
                                            Send Sheet Properties
                                            ***********' newline]; %
                                            Start shell properties info
783                                         SheetPropAux = ['secoff,mid'
                                            newline newline];
784
785                                         for t=1:size(InspShell,1)
786                                             AddLine1 = char(strcat("
                                                sectype,",string(
                                                InspShell{t,2}),",shell
                                                "));
787                                             AddLine2 = char(strcat("
                                                secdata,",sprintf('%.6e
                                                ', InspShell{t,c})));
788                                             AddLine3 = [AddLine1
                                                newline AddLine2
                                                newline SheetPropAux];
789
790                                             SheetProp = [SheetProp
                                                AddLine3];
791                                         end
792
```

```
793                                 ExpToFind = '/com,**********
                                        Send Sheet Properties
                                        **********';
794                                 CopyFmin = strrep(CopyFmin,
                                        ExpToFind,SheetProp);
795
796                                 % Update for Beams
797                                 BeamProp = ['/com,**********
                                        Send Beam Properties
                                        **********' newline];
798                                 BeamRectPropAux = ['secoffset,
                                        cent' newline newline];
799
800                                 for t=1:size(InspBeamRect,1)
801                                     AddLine1 = char(strcat("
                                            SECTYPE,",string(
                                            InspBeamRect{t,2}),",
                                            BEAM,RECT"));
802                                     AddLine2 = char(strcat("
                                            SECDATA,",sprintf('%.6e
                                            ', InspBeamRect{t,c
                                            *2-3}),",",sprintf('%.6
                                            e', InspBeamRect{t,c
                                            *2-2})));
803                                     AddLine3 = [AddLine1
                                            newline AddLine2
                                            newline BeamRectPropAux
                                            ];
804                                     BeamProp = [BeamProp
                                            AddLine3];
805                                 end
806
807                                 for t=1:size(InspBeamL,1)
808                                     AddLine1 = char(strcat("
                                            SECTYPE,",string(
                                            InspBeamL{t,2}),",BEAM,
                                            L"));
809                                     AddLine2 = char(strcat("
                                            SECDATA,",sprintf('%.6e
                                            ', InspBeamL{t,c*6-15})
                                            ,",",sprintf('%.6e',
                                            InspBeamL{t,c*6-14})
                                            ,",",sprintf('%.6e',
```

```
                                                    InspBeamL{t,c*6-13})
                                                    ,",",sprintf('%.6e',
                                                    InspBeamL{t,c*6-12})));
810                                                 AddLine3 = char(strcat("
                                                    secoffset,user,",
                                                    sprintf('%.6e',
                                                    InspBeamL{t,c*6-11})
                                                    ,",",sprintf('%.6e',
                                                    InspBeamL{t,c*6-10})));
811
812                                                 AddLine4 = [AddLine1
                                                    newline AddLine2
                                                    newline AddLine3
                                                    newline];
813                                                 BeamProp = [BeamProp
                                                    AddLine4 newline];
814                                             end
815
816                                             ExpToFind = '/com,**********
                                                Send Beam Properties
                                                **********';
817                                             CopyFmin = strrep(CopyFmin,
                                                ExpToFind,BeamProp);
818
819                                         end
820
821                                     end
822                                 end
823                             end
824
825                             % % % Replacing code by values for case of min
                                pressure
826 %                           try
827 %                               if CaseMinStatus == "TRUE"
828 %                                   CopyFmin = regexprep(CopyFmin,cellstr(
    RepCodes'),cellstr(RepValues'),'once');
829 %                               end
830 %                           end
831
832                             % Replacing code by values for case of max pressure
833                             try
834                                 if CaseMaxStatus == "TRUE"
835                                     for w = 1:size(RepCodes,1)
```

```
836                         CopyFmax = strrep(CopyFmax,RepCodes(w),
                                RepValues(w));
837
838                         % Update inspection data if required
839                         if FLInsp == 1
840
841                                 % Update for shells
842                                 SheetProp = ['/com,**********
                                        Send Sheet Properties
                                        **********' newline]; %
                                        Start shell properties info
843                                 SheetPropAux = ['secoff,mid'
                                        newline newline];
844
845                                 for t=1:size(InspShell,1)
846                                     AddLine1 = char(strcat("
                                            sectype,",string(
                                            InspShell{t,2}),",shell
                                            "));
847                                     AddLine2 = char(strcat("
                                            secdata,",sprintf('%.6e
                                            ', InspShell{t,c})));
848                                     AddLine3 = [AddLine1
                                            newline AddLine2
                                            newline SheetPropAux];
849
850                                     SheetProp = [SheetProp
                                            AddLine3];
851                                 end
852
853                                 ExpToFind = '/com,**********
                                        Send Sheet Properties
                                        **********';
854                                 CopyFmax = strrep(CopyFmax,
                                        ExpToFind,SheetProp);
855
856                                 % Update for Beams
857                                 BeamProp = ['/com,**********
                                        Send Beam Properties
                                        **********' newline];
858                                 BeamRectPropAux = ['secoffset,
                                        cent' newline newline];
859
```

```
860                                 for t=1:size(InspBeamRect,1)
861                                     AddLine1 = char(strcat("
                                        SECTYPE,",string(
                                        InspBeamRect{t,2}),",
                                        BEAM,RECT"));
862                                     AddLine2 = char(strcat("
                                        SECDATA,",sprintf('%.6e
                                        ', InspBeamRect{t,c
                                        *2-3}),",",sprintf('%.6
                                        e', InspBeamRect{t,c
                                        *2-2})));
863                                     AddLine3 = [AddLine1
                                        newline AddLine2
                                        newline BeamRectPropAux
                                        ];
864                                     BeamProp = [BeamProp
                                        AddLine3];
865                                 end
866
867                                 for t=1:size(InspBeamL,1)
868                                     AddLine1 = char(strcat("
                                        SECTYPE,",string(
                                        InspBeamL{t,2}),",BEAM,
                                        L"));
869                                     AddLine2 = char(strcat("
                                        SECDATA,",sprintf('%.6e
                                        ', InspBeamL{t,c*6-15})
                                        ,",",sprintf('%.6e',
                                        InspBeamL{t,c*6-14})
                                        ,",",sprintf('%.6e',
                                        InspBeamL{t,c*6-13})
                                        ,",",sprintf('%.6e',
                                        InspBeamL{t,c*6-12})));
870                                     AddLine3 = char(strcat("
                                        secoffset,user,",
                                        sprintf('%.6e',
                                        InspBeamL{t,c*6-11})
                                        ,",",sprintf('%.6e',
                                        InspBeamL{t,c*6-10})));
871
872                                     AddLine4 = [AddLine1
                                        newline AddLine2
                                        newline AddLine3
```

```
                                                    newline];
873                                             BeamProp = [BeamProp
                                                    AddLine4 newline];
874                                         end
875
876                                         ExpToFind = '/com,**********
                                                Send Beam Properties
                                                **********';
877                                         CopyFmax = strrep(CopyFmax,
                                                ExpToFind,BeamProp);
878
879                                     end
880
881                                 end
882                             end
883                         end
884
885                         % % % Replacing code by values for case of max
                                pressure
886 %                       try
887 %                           if CaseMaxStatus == "TRUE"
888 %                               CopyFmax = regexprep(CopyFmax,cellstr(
    RepCodes'),cellstr(RepValues'),'once');
889 %                           end
890 %                       end
891
892                         % Replacing code by values for case without HDP
893                         if MCContent(3,c) == "0"                % HDP off
894                             for w = 1:size(RepCodes,1)
895                                 CopyFNoHDP = strrep(CopyFNoHDP,RepCodes(w),
                                        RepValues(w));
896
897                                 % Update inspection data if required
898                                 if FLInsp == 1
899
900                                     % Update for shells
901                                     SheetProp = ['/com,********** Send
                                            Sheet Properties **********'
                                        newline]; % Start shell
                                        properties info
902                                     SheetPropAux = ['secoff,mid'
                                        newline newline];
903
```

```
904                                 for t=1:size(InspShell,1)
905                                     AddLine1 = char(strcat("sectype
                                        ,",string(InspShell{t,2})
                                        ,",shell"));
906                                     AddLine2 = char(strcat("secdata
                                        ,",sprintf('%.6e',
                                        InspShell{t,c})));
907                                     AddLine3 = [AddLine1 newline
                                        AddLine2 newline
                                        SheetPropAux];
908
909                                     SheetProp = [SheetProp AddLine3
                                        ];
910                                 end
911
912                                 ExpToFind = '/com,********** Send
                                    Sheet Properties **********';
913                                 CopyFNoHDP = strrep(CopyFNoHDP,
                                    ExpToFind,SheetProp);
914
915                                 % Update for Beams
916                                 BeamProp = ['/com,********** Send
                                    Beam Properties **********'
                                    newline];
917                                 BeamRectPropAux = ['secoffset,cent'
                                     newline newline];
918
919                                 for t=1:size(InspBeamRect,1)
920                                     AddLine1 = char(strcat("SECTYPE
                                        ,",string(InspBeamRect{t
                                        ,2}),",BEAM,RECT"));
921                                     AddLine2 = char(strcat("SECDATA
                                        ,",sprintf('%.6e',
                                        InspBeamRect{t,c*2-3}),",",
                                        sprintf('%.6e',
                                        InspBeamRect{t,c*2-2})));
922                                     AddLine3 = [AddLine1 newline
                                        AddLine2 newline
                                        BeamRectPropAux];
923                                     BeamProp = [BeamProp AddLine3];
924                                 end
925
926                                 for t=1:size(InspBeamL,1)
```

```
927                                         AddLine1 = char(strcat("SECTYPE
                                                ,",string(InspBeamL{t,2})
                                                ,",BEAM,L"));
928                                         AddLine2 = char(strcat("SECDATA
                                                ,",sprintf('%.6e',
                                                InspBeamL{t,c*6-15}),",",
                                                sprintf('%.6e', InspBeamL{t
                                                ,c*6-14}),",",sprintf('%.6e
                                                ', InspBeamL{t,c*6-13})
                                                ,",",sprintf('%.6e',
                                                InspBeamL{t,c*6-12})));
929                                         AddLine3 = char(strcat("
                                                secoffset,user,",sprintf('
                                                %.6e', InspBeamL{t,c*6-11})
                                                ,",",sprintf('%.6e',
                                                InspBeamL{t,c*6-10})));
930
931                                         AddLine4 = [AddLine1 newline
                                                AddLine2 newline AddLine3
                                                newline];
932                                         BeamProp = [BeamProp AddLine4
                                                newline];
933                                     end
934
935                                     ExpToFind = '/com,********** Send
                                            Beam Properties **********';
936                                     CopyFNoHDP = strrep(CopyFNoHDP,
                                            ExpToFind,BeamProp);
937
938                                 end
939
940                             end
941                         end
942
943
944
945                     end
946                 end
947             end
948
949
950 % if c==5
951 %     return
```

```
952 % end
953 %
954
955
956
957         % Save case min
958         try
959             if CaseMinStatus == "TRUE"
960                 CaseName = strcat(FolderFemCases,'\',MCContent(1,c),'
                        _min.dat');
961                 CopyFmin = strrep(CopyFmin,'***RESULT***',strcat(
                        MCContent(1,c),'_min'));
962                 fid  = fopen(CaseName,'w');
963                 fprintf(fid,'%s',CopyFmin);
964                 fclose(fid);
965             end
966         catch
967             % do nothing
968         end
969         CaseMinStatus = [];
970
971         % Save case max
972         try
973             if CaseMaxStatus == "TRUE"
974                 CaseName = strcat(FolderFemCases,'\',MCContent(1,c),'
                        _max.dat');
975                 CopyFmax = strrep(CopyFmax,'***RESULT***',strcat(
                        MCContent(1,c),'_max'));
976                 fid  = fopen(CaseName,'w');
977                 fprintf(fid,'%s',CopyFmax);
978                 fclose(fid);
979             end
980         catch
981             % do nothing
982         end
983         CaseMaxStatus = [];
984
985         % Save case without HDP
986         if MCContent(3,c) == "0"                % HDP off
987             CaseName = strcat(FolderFemCases,'\',MCContent(1,c),'.dat')
                    ;
988             CopyFNoHDP = strrep(CopyFNoHDP,'***RESULT***',strcat(
                    MCContent(1,c)));
```

```
989             fid  = fopen(CaseName,'w');
990             fprintf(fid,'%s',CopyFNoHDP);
991             fclose(fid);
992         end
993 
994         fclose all;
995     else
996         CaseNumber = MCContent(1,c);
997         fprintf('Case %s is OFF. \n',CaseNumber);
998     end
999 end
1000 
1001 fprintf("Cases for simulation in batch mode are ready. \n");
1002 fprintf("Batch number: %d \n",BatchNumber);
1003 
1004 %%  End of code
1005 fclose all;
1006 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
1007 fprintf('Elapsed time running A03_assemble.m is %s \n',ElapsedTime)
1008 clearvars
1009 return
```

C.1.3. Algoritmo B04_solver

Algoritmo	Objetivo	Categoria
B04_solver	Criar um arquivo em lotes e executar o Solver em elementos finitos	Primário

```
%% B04_solver
%  Sep 15, 2020
%  Author: Kennedy Neves
%  This script/function creates and runs a file to be used by ANSYS Mechanical solver in batch mode
%  [pt-br] Este script/fun  o cria e executa um arquivo para ser usado pelo solver ANSYS Mechanical em modo de lote

%% Notes:
%

%% Steps:
%  Load variables
%  Read *.dat files
%  Read *.out files
%  Create*.bat file
%  Save *.bat file
%  Clear folder
%  End of code

%% Load variables
fprintf(1,"Initializing B04_solver \n")
tic

% Check if script was requested by GUI (in Manual or Automatic control)
try
   if control == "Manual"
       fprintf("Control: GUI - %s \n",control)
   elseif control == "Automatic"
       fprintf("Control: GUI - %s \n",control)
   else
       control = "None";
       fprintf("Control: GUI - %s \n",control)
   end
catch
   control = "None";
   fprintf("Control: GUI - %s \n",control)
```

```
36 end
37
38 % Load variables by running without GUI
39 if control == "None"
40     Ini_A00_initial_data
41     clearvars -except Var_B04_solver
42
43     % Input: FEM model cases
44     FolderFemCases = Var_B04_solver{1};      % Folder with FEM model
           files
45     PathAnsysSolver = Var_B04_solver{2};     % Solver path
46     np = Var_B04_solver{3};                  % Number of processors
47     FileNameBat = Var_B04_solver{4};         % Batch file name
48 end
49
50 %% Read *.dat files
51 GetFemCases = strcat(FolderFemCases,'\*.dat');      % path and extension
52 GetMatFiles = dir(GetFemCases);                      % read *.dat files
       info
53 MatFilesInfo = struct2cell(GetMatFiles);             % create cell for
       files
54 AvaFemCases = string(MatFilesInfo(1,:))';            % available FEM
       cases
55 if isempty(AvaFemCases)
56     fprintf(2,"There are no FEM models to run \n")
57     return
58 end
59
60 %% Read *.out files
61 GetFemCases = strcat(FolderFemCases,'\*.out');      % path and
       extension
62 GetMatFiles = dir(GetFemCases);                          % read *.dat
       files info
63 MatFilesInfo = struct2cell(GetMatFiles);                 % create cell
       for files
64 AvaLogFemCases = string(MatFilesInfo(1,:))';            % available log
        files
65
66 AvaFemCasesCheck = strrep(AvaFemCases,".dat","");       % list of
       available FEM cases to check
67 AvaLogFemCasesCheck = strrep(AvaLogFemCases,".out",""); % list of
       available log files to check
68
```

```
idx=[];                                                 % matrix to get
    index of FEM cases already ran
for i=1:size(AvaFemCasesCheck,1)
    for j=1:size(AvaLogFemCasesCheck,1)
        Check = find(AvaFemCasesCheck(i,1) == AvaLogFemCasesCheck(j,1))
            ;
        if Check == 1;
            idx = [idx;i];
        end
    end
end

AvaFemCases(idx) = [];                                  % exclude cases
    already ran

%% Create*.bat file
BatFileContent = 'SET ANSYS201_PRODUCT=ANSYS';
BatFileContent = [BatFileContent newline 'SET ANS_CONSEC=YES'];
BatFileContent = [BatFileContent newline 'SET ANSYS_LOCK=OFF'];
BatFileContent = [BatFileContent newline 'SET KMP_STACKSIZE=2048k'];

newl = [];
for i=1:size(AvaFemCases,1)
    CurrentCase = char(strcat('"',pwd,'\',FolderFemCases,'\',
        AvaFemCases(i),'"'));
%     newl = [PathAnsysSolver ' -b -i ' CurrentCase];
    newl = [PathAnsysSolver ' -b -np ' char(string(np)) ' -i '
        CurrentCase];    %*** TO BE TESTED (including -np command)
    newl = [newl ' -o '];
    CurrentCaseOut = strrep(AvaFemCases(i),'.dat','.out');
    CurrentCaseOut = char(strcat('"',pwd,'\',FolderFemCases,'\',
        CurrentCaseOut,'"'));
    newl = [newl CurrentCaseOut];
    newl = [newl ' -dir '];
    newl = [newl '"',pwd,'\',FolderFemCases,'"'];
    BatFileContent = [BatFileContent newline newl];
end

%% Save *.bat file
BatFileContent = string(BatFileContent);
fid  = fopen(strcat(FolderFemCases,'\',FileNameBat,'.bat'),'w');
fprintf(fid,'%s',BatFileContent);
fclose(fid);
```

```
106
107 fclose all; % close all files
108 fprintf("Running simulations in batch mode. \n");
109 system(strcat(FolderFemCases,'\',FileNameBat,'.bat')); % run bat file
110 fprintf("End of simulations in batch mode. \n");
111
112 %% Clear folder
113 fprintf("Cleaning results folder. \n");
114 % Read all files
115 GetAllFiles = dir(FolderFemCases);                % read all files
116 MatFilesInfo = struct2cell(GetAllFiles);        % create cell for files
117 AllFiles = string(MatFilesInfo(1,:))';          % all files
118
119 % Delete files out of allowed extensions
120 AllowedExtension = ["bat";"txt";"out";"dat"];   % Allowed extensions
121 recycle('on');                                  % Turn on file
    recycling
122 for i=1:size(AllFiles,1)
                                                    % Loop through
    filenames (with extension)
123     try
124         FileNameLength = strlength(AllFiles(i));
125         Last3Letters = char(AllFiles(i));
126         Last3Letters = Last3Letters((FileNameLength-2):FileNameLength);
127         Last3Letters = string(Last3Letters);
                                        % Get last 3 letters of
            extension
128
129         if isempty(find(Last3Letters == AllowedExtension))
                            % Delete file if it has not any allowed
            extension
130             delete(strcat(FolderFemCases,'\',AllFiles(i)));
131         end
132     catch
133         % do nothing
134     end
135 end
136
137 % Read all files
138 GetAllFiles = dir(FolderFemCases);                % read all files
139 MatFilesInfo = struct2cell(GetAllFiles);        % create cell for files
140 AllFiles = string(MatFilesInfo(1,:))';          % all files
141
```

```
142 for i=1:size(AllFiles,1)
                                                    % Loop through
    filenames (with extension)
143     try
144         First4Letters = char(AllFiles(i));
145         First4Letters = First4Letters(1:4);
146         First4Letters = string(First4Letters);
                                          % Get last 3 letters of
            extension
147 
148         if First4Letters == "file"  % isempty(find(Last3Letters ==
            AllowedExtension))                  % Delete file if it has
             not any allowed extension
149             delete(strcat(FolderFemCases,'\',AllFiles(i)));
150         end
151     catch
152         % do nothing
153     end
154 end
155 
156 recycle('off');                              % Turn off file
    recycling
157 
158 %%  End of code
159 fclose all;
160 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
161 fprintf('Elapsed time running C03_solver.m is %s \n',ElapsedTime)
162 clearvars
163 return
```

C.1.4. Algoritmo C05_post_results

Algoritmo	Objetivo	Categoria
C05_post_results	Organizar e gera resultados do SAD	Primário

```
%% C05_post_results
%  Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%  May 10, 2021
%  This script/function plot all required results in software Gmsh
%  [pt-br] Este script/fun  o plotta todos os resultados requeridos no programa Gmsh

%% Notes:
%

%% Steps:
%  Load variables
%  Check for bodies info to get Nodes in Range
%  Check for bodies info to get Nodes in Range
%  Process required results
%  Check for results already processed
%  Get all result files (*.txt) from folder "A03 assemble"
%  Expressions to find accorddng to ResPrint
%  Read .txt files
%  Process result files (*.txt)
%  Check alert matrix
%  Write nodes, elements and bodies for .msh file
%  Plot simulation results from ANSYS in Gmsh (according to ResGmsh)
%  Plot hotspots in Gmsh (accorging to Alert Matrix)
%  Plot bodies in alert in Gmsh (according to Alert Matrix)
%  Save processed results
%  End of code

%% Load variables
fprintf(1,"Initializing C05_post_results \n")
tic

% Check if script was requested by GUI (in Manual or Automatic control)
try
   if control == "Manual"
       fprintf("Control: GUI - %s \n",control)
   elseif control == "Automatic"
```

```
37          fprintf("Control: GUI - %s \n",control)
38      else
39          control = "None";
40          fprintf("Control: GUI - %s \n",control)
41      end
42  catch
43      control = "None";
44      fprintf("Control: GUI - %s \n",control)
45  end
46
47  % Load variables by running without GUI
48  if control == "None"
49      Ini_A00_initial_data
50      clearvars -except Var_C05_post_results
51
52      % FolderLock = 'Aux locked\PJ_Cilamce';
53      % FolderPost = 'A03 assemble\PJ_Cilamce';
54      % FolderFemCases = 'A03 assemble\PJ_Cilamce';
55      % ResPrint
56      % ResPrint = [" PRINT U";" PRINT S";" PRINT EPEL";" PRINT EPPL";"
            PRINT EPTO";" PRINT S"];
57
58
59      FolderLock = Var_C05_post_results{1}; % 'Locked';
                                                              % Folder for
            locked variables
60      FolderPost = Var_C05_post_results{2}; % Folder for Post-processed
            files
61      FolderFemCases = Var_C05_post_results{3}; % Folder to generate FEM
            models and results
62      ResInfo = Var_C05_post_results{4};        % File with results
            information
63      ResItem = Var_C05_post_results{5};  % ["Displacement";"Stress";"
            Elastic_Strain";"Plastic_Strain";"Total_Strain"]; % Possible
            result status
64      ResPrint = Var_C05_post_results{6}; % [" PRINT U";" PRINT S";"
            PRINT EPEL";" PRINT EPPL";" PRINT EPTO"]; % Expression to find
            according to possible results
65      ResComp = Var_C05_post_results{7};  % [4;6;6;6;6]; % Total of
            components for each result
66      ResCompView = Var_C05_post_results{8}; % Component views
67      AlertMatrix = Var_C05_post_results{9};
                                                % Alert Matrix
```

```
68      ResGmsh = Var_C05_post_results{10};
                                                % Results required to be
            plotted on *.msh files format
69      PrintGmsh = Var_C05_post_results{11};
                                              % Required components from
            results to be plotted on *.msh files format
70
71  end
72
73  %     FolderLock
74  %     FolderPost
75  %     FolderFemCases
76  %     ResInfo
77  %     ResItem
78  %     ResPrint
79  %     ResComp
80  %     ResCompView
81  %     AlertMatrix
82  %     ResGmsh
83  %     PrintGmsh
84
85
86  %% Check for bodies info to get Nodes in Range
87  try
88      CheckBodyInfoStructure = load(strcat(FolderLock,'\',"BodyInfo.mat")
            );
89
90      VarInfo = struct2cell(CheckBodyInfoStructure);      % Create cell
            for *.mat files
91
92      BodyInfo = cat(1, VarInfo{1});          % Get info for bodies
93      BodyList = cat(1, VarInfo{2});          % Get list of bodies
94      ElementsRange = cat(1, VarInfo{3});     % Get info for elements in
            range
95      NodeBodyElType = cat(1, VarInfo{4});    % Get info for bodies (Body
             Name, Element Type and Range)
96      NodesFem = cat(1, VarInfo{5});          % Get info for all nodes
97      NodesRange = cat(1, VarInfo{6});        % Get info for nodes in
            range
98
99  catch
100     fprintf(2,"Information for bodies is not ready \n")
101     fprintf(2,"Please run Ini_A00_post_bodies \n")
```

```
102     return
103 end
104 
105 
106 %% Process required results
107 % Check locked variables
108 try
109     load(strcat(FolderLock,'\','ReqResStatus_locked','.','mat'),'
            ReqResStatus_locked');  % Load information
110 catch
111     fprintf(2,"Information for status of results is missing. Please run
            A03_assemble. \n")
112     return
113 end
114 
115 %% Check for results already processed
116 ProcessedTxtFiles = [];
117 try
118     load(strcat(FolderPost,'\','- Processed_Results','.','mat'));     %
            Load processed results if available
119     for i=3:size(ResProc,2)
120         ProcessedTxtFiles = [ProcessedTxtFiles;ResProc{1,i}];    % Get
               names for results already processed
121     end
122 catch
123     ResAlert = [];
124     % do nothing
125 end
126 
127 %% Get all result files (*.txt) from folder "A03 assemble"
128 GetListTxt = dir(strcat(FolderFemCases,'\*.txt'));
129 GetListTxt = struct2cell(GetListTxt);                    % Get filenames
        for *.txt files
130 
131 ListTxtFiles = strings;
132 for i=1:size(GetListTxt,2)
133     ListTxtFiles(i,1) = GetListTxt{1,i};                 % Get all *.txt
            files available
134 end
135 
136 FindResInfo = find(ListTxtFiles == string(ResInfo));     % Find results
       file
137 ListTxtFiles(FindResInfo) = [];                          % Remove
```

```
    results file from list
% Remove cases already processed
if isempty(ProcessedTxtFiles)   % In this case there is no previous
    processed results

    % Create cell to get processed results
    ResProc{1,1} = "Result";
    ResProc{1,2} = "Status";
    for i=1:size(ResItem,1)
        ResProc{i+1,1} = ResItem(i);
    end

    for i=1:size(ResItem,1)
        ResProc{i+1,2} = ReqResStatus_locked(i+2);
    end

    StartNewCases = 3;  % Column to start processing new cases
    try
        IdxDel = find(strcat(ResInfo,'.txt') == ListTxtFiles);
        ListTxtFiles(IdxDel) = [];
    catch
        % do nothing
    end

    for j=1:size(ListTxtFiles,1)
        ResProc{1,j+2} = ListTxtFiles(j);
    end

else % In this case there are previous processed results
    % Remove cases already processed
    AlreadyProc = [];
    for i=1:size(ListTxtFiles)
        if find(ListTxtFiles(i) == ProcessedTxtFiles)
            AlreadyProc = [AlreadyProc;i];
        end
    end
    ListTxtFiles(AlreadyProc) = [];
    NewCases = size(ListTxtFiles,1);

    try
        IdxDel = find(strcat(ResInfo,'.txt') == ListTxtFiles);
        ListTxtFiles(IdxDel) = [];
    catch
```

```
179         % do nothing
180     end
181     % Update information for results to be processed
182     StartNewCases = size(ResProc,2);
183     for i=1:NewCases
184         ResProc{1,StartNewCases+i} = ListTxtFiles(i);
185     end
186     StartNewCases = StartNewCases +1; % Column to start processing new
            cases
187
188 end
189
190 clearvars i j FindResInfo ProcessedTxtFiles
191
192 %% Expressions to find accorddng to ResPrint
193 ResPrint(1,2) = "NODE        UX            UY            UZ
        USUM";
194 ResPrint(2,2) = "NODE      SX            SY            SZ
        SXY           SYZ           SXZ";
195 ResPrint(3,2) = "NODE      EPELX         EPELY         EPELZ
        EPELXY        EPELYZ        EPELXZ";
196 ResPrint(4,2) = "NODE      EPPLX         EPPLY         EPPLZ
        EPPLXY        EPPLYZ        EPPLXZ";
197 ResPrint(5,2) = "NODE      EPTOX         EPTOY         EPTOZ
        EPTOXY        EPTOYZ        EPTOXZ";
198 ResPrint(6,2) = "NODE      S1            S2            S3
        SINT          SEQV";
199
200
201 %% Process result files (*.txt)
202 fprintf("Processing results... \n")
203 for j=StartNewCases:size(ResProc,2)                          % Go
        through Cases
204
205     FileName = strcat(FolderFemCases,'\',ResProc{1,j});      % Get .
            txt file name
206     FileData = extractFileText(FileName);                    % Get
            data from .txt file
207     FileData = regexprep(FileData,'\n','&%^');               %
            Replace break lines with codes
208     FileData = char(FileData);                               %
            Convert to char
209     FileData = strip(split(FileData,'&%^'));                 % Split
```

```
        characters and delete codes
210     StringFileData = string(FileData);                          %
            Convert to string
211
212     for i=2:size(ResProc,1)
                                                                    % Go through
            Results
213         if ResProc{i,2} == true
                                                                    % Get idx for each
                result if true
214             IdxResPrint = find(ResPrint(i-1,2) == StringFileData(:,1));
215             ResPrint(i-1,3) = string(IdxResPrint);
216         else

                % Write "aux" if false
217               ResPrint(i-1,3) = "aux";
218         end
219     end
220
221     for i=size(ResPrint,1):-1:1
                                                                % Replace index with
            next value in case it is false
222         if ResPrint(i,3) == "aux"
223             ResPrint(i,3) = ResPrint(i+1,3);
224         end
225     end
226
227     for i=2:size(ResProc,1)                                         % Go
            through Results
228         if ResProc{i,2} == true                                     % Get
                idx for each result if true
229
230             A = str2double(ResPrint(i-1,3)) + 1;                    % Get
                    idx for beginning of result
231
232             if i ~= size(ResProc,1)                                 % Get
                    idx for end of result
233                 B = str2double(ResPrint(i,3));
234                 ResCurrent = FileData(A:B,1);
235             else
236                 ResCurrent = FileData(A:end,1);
237             end
238
```

```
239 %             return %$$$
240 
241 %             for t=1:size(ResCurrent,1)-1
242 %                 TestRow = ResCurrent{end-t+1};
243 %                 if ~isempty(TestRow)
244 %                     if isstrprop(ResCurrent{end-t+1}(1),'digit') == 1
245 %                         t;
246 %                         return
247 %                     end
248 %                 end
249 %             end
250 %
251 %
252 %             ResCurrent = cellfun(@str2double,ResCurrent,'
    UniformOutput',false);    % Get result
253 
254 
255 
256 
257             ResCurrent = cellfun(@str2num,ResCurrent,'UniformOutput',
                false);    % Get result
258             ResCurrent = cell2mat(ResCurrent);
                                                % Convert result to
                matrix
259 
260             inster = intersect(ResCurrent(:,1),NodesRange(:,1));
                                % Get instersection with nodes in range
261 
262             [tf,loc] = ismember(inster,ResCurrent(:,1));
                                          % Get idx of nodes in range
263             ResCurrent = ResCurrent(loc,:);
                                                    % Keep only nodes
                in range
264 
265             ResProc{i,j} = ResCurrent;
                                                          % Append
                processed result
266 
267         end
268     end
269 
270 fprintf("Results for case '%s' are processed \n",ResProc{1,j})
271 end
```

```
272
273 %% Check Alert Matrix
274 ResAlertMatrix = {};            % Matrix to append results reaching
        alert levels
275 if ~isempty(AlertMatrix)        % If Alert Matrix is not empty
276
277     % If it's the first time running
278     if isempty(ResAlert)
279         ResAlert = ResProc;          % Copy data from ResProc
280         ResAlert(:,StartNewCases:end)= [];
281     end
282
283     for j=StartNewCases:size(ResProc,2)      % Loop through ResProc
            columns
284         ResAlert(1,j)=ResProc(1,j);          % Matrix to append
                ResAlertMatrix
285         for i=2:size(ResProc,1)              % Loop through ResProc rows
286
287             if ismember(ResAlert{i,1}(1,1),AlertMatrix(2:end,1))
                                                 % Proceed only if result is
                    requested by AlertMatrix
288                 [AlertMatrixIdx,~] = find(ResAlert{i,1}(1,1) ==
                        AlertMatrix(:,1));
289                 ResCurrentAllComp = ResProc{i,j};
                                                          % Current
                        result with Node ID and all Components
290
291                 % Creating array ResAlertMatrix to append alert results
292                 for a=1:size(AlertMatrix,1)
293                     for b=1:size(AlertMatrix,2)
294                         ResAlertMatrix{a,b} = AlertMatrix(a,b);
295                     end
296                 end
297
298                 % Make alert results empty
299                 for a=2:size(AlertMatrix,1)
300                     for b=4:size(AlertMatrix,2)
301                         ResAlertMatrix{a,b} = [];
302                     end
303                 end
304
305                 % Loop to update ResAlertMatrix with nodes in alert
                        level
```

```
306                 for ii=1:size(AlertMatrixIdx,1)
307                     AlertResult = AlertMatrix(AlertMatrixIdx(ii),1);
                                                % Get result from
                        AlertMatrix
308                     AlertComponent = str2double(AlertMatrix(
                        AlertMatrixIdx(ii),2));     % Get component
                        from AlertMatrix
309                     AlertLogical = AlertMatrix(AlertMatrixIdx(ii),3);
                                                % Get logical symbol from
                        AlertMatrix
310
311                     ResCurrent = [ResCurrentAllComp(:,1),
                        ResCurrentAllComp(:,AlertComponent+1)];      %
                        Current result with Node ID and Component from
                        alert matrix
312
313                     for jj=4:size(AlertMatrix,2)
314                         ResAlertMatrix{1,jj} = AlertMatrix(1,jj);
                                                                     %
                            Update ResAlertMatrix with Alert Level
                            informed in AlertMatrix
315                         AlertValue = str2double(AlertMatrix(
                            AlertMatrixIdx(ii),jj));                  %
                             Get current value for alert level
316                         try
317                             AlertValueNext = str2double(AlertMatrix(
                                AlertMatrixIdx(ii),jj+1));          %
                                Get next value for alert level
318                         catch
319                             AlertValueNext = [];

                                % Make it empty if it is the last
                                column
320                         end
321
322                         if AlertLogical == ">"
                                                                    %
                            Loop for logical greater than (>)
323
324                             % Exception if it is the last column of
                                AlertMatrix
325                             if isempty(AlertValueNext)
                                                                % Make
```

```
                        next value for alert level = infinite
326                     AlertValueNext = Inf;
327                 end
328
329                 % Get nodes in range for alert level
330                 [NodeIdx,~] = find(ResCurrent(:,2) >
                        AlertValue & ResCurrent(:,2) <
                        AlertValueNext);
331                 AlertNodes = ResCurrent(NodeIdx,:);
332                 AlertNodes(:,2) = str2double(ResAlertMatrix
                        {1,jj});     % Assign alert level from
                        ResAlerMatrix
333
334                 % Assign information for Body Name related
                        to nodes in alert level (AlertNodes=[
                        NodeID,AlertLevel,BodyName])
335                 [~,Idx] = ismember(AlertNodes(:,1),
                        str2double(NodeBodyElType(:,1)) );
336                 AlertNodes = string(AlertNodes);
337                 AlertNodes(:,3) = NodeBodyElType(Idx,2);
338
339                 % Update ResAlertMatrix with AlertNodes
340                 ResAlertMatrix{AlertMatrixIdx(ii),jj} =
                        AlertNodes;
341
342             elseif AlertLogical == "<"
                                                          %
                    Loop for logical less than (<)
343                 % Exception if it is the last column of
                        AlertMatrix
344                 if isempty(AlertValueNext)
                                                    % If
                        it is the last column, make next value
                        for alert level = -infinite
345                     AlertValueNext = -Inf;
346                 end
347
348                 % Get nodes in range for alert level
349                 [NodeIdx,~] = find(ResCurrent(:,2) <
                        AlertValue & ResCurrent(:,2) >
                        AlertValueNext);
350                 AlertNodes = ResCurrent(NodeIdx,:);
351                 AlertNodes(:,2) = str2double(ResAlertMatrix
```

```
                                {1,jj});     % Assign alert level from
                                ResAlerMatrix
352
353                             % Assign information for Body Name related
                                to nodes in alert level (AlertNodes=[
                                NodeID,AlertLevel,BodyName])
354                             [~,Idx] = ismember(AlertNodes(:,1),
                                str2double(NodeBodyElType(:,1)) );
355                             AlertNodes = string(AlertNodes);
356                             AlertNodes(:,3) = NodeBodyElType(Idx,2);
357
358                             % Update ResAlertMatrix with AlertNodes
359                             ResAlertMatrix{AlertMatrixIdx(ii),jj} =
                                AlertNodes;
360
361                         end
362
363                         % Update ResAlert
364                         ResAlert{i,j}=ResAlertMatrix;
365                     end
366
367                 end
368
369             end
370         end
371     end
372 end
373
374 %% Write nodes, elements and bodies for .msh file
375 % Initial lines for *.msh file
376 MshInitializeAll = [];
377 MshInitializeAll = "$MeshFormat";
378 MshInitializeAll = [MshInitializeAll;"2.1 0 8"];
379 MshInitializeAll = [MshInitializeAll;"$EndMeshFormat"];
380 MshInitializeAll = [MshInitializeAll;""];
381
382 % Add nodes info for *.msh file
383 MshInitializeAll = [MshInitializeAll;"$Nodes"];
384 MshInitializeAll = [MshInitializeAll;string(size(NodesRange,1))];
385 AddNodes = sprintf('%d  %0.8f  %0.8f  %0.8f\n',NodesRange');
386 AddNodes = AddNodes(1:size(AddNodes,2)-1);
387 MshInitializeAll = [MshInitializeAll;AddNodes];
388 MshInitializeAll = [MshInitializeAll;"$EndNodes"]; % End nodes
```

```
389 MshInitializeAll = [MshInitializeAll;""];
390
391 % Add elements info for *.msh file
392 % MshInitializeAll = MshInitializeAll;
393 MshInitializeAll = [MshInitializeAll;"$Elements"]; % Elements
394 MshInitializeAll = [MshInitializeAll;string(size(ElementsRange,1))];
395
396 Elements4NIdx = (1:size(ElementsRange,1))';
397 Elements2NIdx = find(isnan(ElementsRange(:,5)));    % Get idx for 2-
    node elements
398 Elements4NIdx(Elements2NIdx) = [];                  % Get idx for 4-
    node elements
399
400 Elements4NRange = ElementsRange(Elements4NIdx,:);   % Get all 4-node
    elements from Elements in Range
401 AddElements4N = sprintf('%d     3       3       1       1       0
          %d  %d  %d %d \n',Elements4NRange'); % Format for 4-node
    elements
402 AddElements4N = AddElements4N(1:size(AddElements4N,2)-1);   % Delete
    last line
403
404 Elements2N = ElementsRange(Elements2NIdx,1:3);      % Get all 2-node
    elements from Elements in Range
405 AddElements2N = sprintf('%d     1       3       2       2       0
          %d  %d \n',Elements2N'); % Format for 2-node elements
406 AddElements2N = AddElements2N(1:size(AddElements2N,2)-1);   % Delete
    last line
407
408 MshInitializeAll = [MshInitializeAll;AddElements4N];  % Add 4-node
    elements to *.msh file
409 MshInitializeAll = [MshInitializeAll;AddElements2N];  % Add 2-node
    elements to *.msh file
410 MshInitializeAll = [MshInitializeAll;"$EndElements"]; % End elements
411 MshInitializeAll = [MshInitializeAll;""]; % End elements
412 % MshInitializeAll = [MshInitializeAll;""]; %
413
414 %% Plot simulation results from ANSYS in Gmsh (according to ResGmsh)
415 if ResGmsh(1,1) == true         % Plot results from simulations
416
417     for j=StartNewCases:size(ResProc,2)                          %
        Loop through ResProc columns
418         Counter = 0;
419         Msh=[];
```

```
420         for i=2:size(ResProc,1)
421             Counter = Counter+1;
422             ResCurrent = ResProc{i,j};                              % Get
                    current result
423             ResultType = ResProc{i,1};                              % Get
                    result type
424
425             switch ResultType
426                 case "Displacement"
427
428                     CheckGmshPrint = PrintGmsh{i-1,1};                  %
                          Check if result was request to be printed on
                        *.msh file format
429                     for jj=1:size(CheckGmshPrint,2)
430                         if CheckGmshPrint(1,jj) == 1 && ~isempty(
                            ResCurrent)     % Check if result was
                            requested to be printed on *.msh file
                            format and if result was generated by Ansys
431
432                          if jj~=size(CheckGmshPrint,2)
                                            % Case Displacement is UX
                            , UY or UZ (the same process to proceed
                             with results)
433                             Msh = [Msh;"$NodeData"];
                                            % $NodeData
434                             Msh = [Msh;"1"];
                                                    %
435
436                             View = strcat('"',ResItem(i-1,1),'_',
                                ResCompView(i-1,jj),'"');      %
                                View name
437                             Msh = [Msh;View];
438
439                             Msh = [Msh;"1"];
                                                    %
440                             Msh = [Msh;string(Counter)];
                                        % Counter (replacing
                                time step)
441                             Msh = [Msh;"4"];
                                                    %
442                             Msh = [Msh;"0"];
                                                    % Step
                                number (It's not used, so it can be
```

```
                                always 0)
443                         Msh = [Msh;"1"];
                                                % Number of
                                components
444                         Msh = [Msh;string(size(ResCurrent,1))];
                                % Number of nodes
445                         Msh = [Msh;"0"];
                                                %
446
447                         % Add nodes info from current result
448                         Aux = [ResCurrent(:,1),ResCurrent(:,jj
                                +1)];
449                         AddNodes = sprintf('%d  %0.8f \n',Aux')
                                ;
450                         AddNodes = AddNodes(1:size(AddNodes,2)
                                -1);
451                         Msh = [Msh;AddNodes];
452
453                         Msh = [Msh;"$EndNodeData"];
                                            % $EndNodeData
454                         Msh = [Msh;""];
455
456                     else

                        % EXCEPTION: Case Displacement is USUM
                        (it uses 3 components UX, UY and UZ)
457
458                         Msh = [Msh;"$NodeData"];
                                                % $NodeData
459                         Msh = [Msh;"1"];
                                                %
460
461                         View = strcat('"',ResItem(i-1,1),'_',
                                ResCompView(i-1,jj),'"');      %
                                View name
462
463                         Msh = [Msh;View];
464
465                         Msh = [Msh;"1"];
                                                %
466                         Msh = [Msh;string(Counter)];
                                            % Counter (replacing
                                time step)
```

```
467                             Msh = [Msh;"4"];
                                                %
468                             Msh = [Msh;"0"];
                                                % Step
                                    number (It's not used, so it can be
                                     always 0)
469                             Msh = [Msh;"3"];
                                                % Number of
                                    components (Displacement has 3
                                    components)
470                             Msh = [Msh;string(size(ResCurrent,1))];
                                     % Number of nodes
471                             Msh = [Msh;"0"];
                                                %
472
473                             % Add nodes info from current result
474                             AddNodes = sprintf('%d  %0.8f  %0.8f
                                    %0.8f \n',ResCurrent(:,1:4)');
475                             AddNodes = AddNodes(1:size(AddNodes,2)
                                    -1);
476                             Msh = [Msh;AddNodes];
477
478                             Msh = [Msh;"$EndNodeData"];
                                                % $EndNodeData
479                             Msh = [Msh;""];
480
481                         end
482
483
484                     elseif CheckGmshPrint(1,jj) == 1  && isempty(
                            ResCurrent)
485                         fprintf(2,"Results for %s were not
                                previously generated during simulation
                                of %s. %s cannot be printed. \n",
                                ResultType,ResProc{i,1},ResCompView(i
                                -1,jj));
486                     end
487                 end
488
489             case "Stress"
490                 CheckGmshPrint = PrintGmsh{i-1,1};                %
                        Check if result was request to be printed on
                       *.msh file format
```

```
491                     for jj=1:size(CheckGmshPrint,2)
492                         if CheckGmshPrint(1,jj) == 1 && ~isempty(
                                ResCurrent)  % Check if result was
                                requested to be printed on *.msh file
                                format and if result was generated by Ansys
493
494                             Msh = [Msh;"$NodeData"];                %
                                    $NodeData
495                             Msh = [Msh;"1"];                        %
496
497                             View = strcat('"',ResItem(i-1,1),'_',
                                    ResCompView(i-1,jj),'"');       % View
                                    name
498                             Msh = [Msh;View];
499
500                             Msh = [Msh;"1"];                        %
501                             Msh = [Msh;string(Counter)];            %
                                    Counter (replacing time step)
502                             Msh = [Msh;"4"];                        %
503                             Msh = [Msh;"0"];                        %
                                    Step number (It's not used, so it can
                                    be always 0)
504                             Msh = [Msh;"1"];                        %
                                    Number of components
505                             Msh = [Msh;string(size(ResCurrent,1))]; %
                                    Number of nodes
506                             Msh = [Msh;"0"];                        %
507
508                             % Add nodes info from current result
509                             Aux = [ResCurrent(:,1),ResCurrent(:,jj+1)];
510                             AddNodes = sprintf('%d  %0.8f \n',Aux');
511                             AddNodes = AddNodes(1:size(AddNodes,2)-1);
512                             Msh = [Msh;AddNodes];
513
514                             Msh = [Msh;"$EndNodeData"];             %
                                    $EndNodeData
515                             Msh = [Msh;""];
516                         elseif CheckGmshPrint(1,jj) == 1  && isempty(
                                ResCurrent)
517                             fprintf(2,"Results for %s were not
                                    previously generated during simulation
                                    of %s. %s cannot be printed. \n",
                                    ResultType,ResProc{i,1},ResCompView(i
```

```
                                -1,jj));
518                         end
519                     end
520
521                 case "Elastic_Strain"
522                     CheckGmshPrint = PrintGmsh{i-1,1};                  %
                            Check if result was request to be printed on
                           *.msh file format
523                     for jj=1:size(CheckGmshPrint,2)
524                         if CheckGmshPrint(1,jj) == 1 && ~isempty(
                               ResCurrent)  % Check if result was
                               requested to be printed on *.msh file
                               format and if result was generated by Ansys
525
526                             Msh = [Msh;"$NodeData"];                    %
                                $NodeData
527                             Msh = [Msh;"1"];                            %
528
529                             View = strcat('"',ResItem(j-1,1),'_',
                                ResCompView(j-1,jj),'"');       % View
                                name
530                             Msh = [Msh;View];
531
532                             Msh = [Msh;"1"];                            %
533                             Msh = [Msh;string(Counter)];                %
                                Counter (replacing time step)
534                             Msh = [Msh;"4"];                            %
535                             Msh = [Msh;"0"];                            %
                                Step number (It's not used, so it can
                                be always 0)
536                             Msh = [Msh;"1"];                            %
                                Number of components
537                             Msh = [Msh;string(size(ResCurrent,1))]; %
                                Number of nodes
538                             Msh = [Msh;"0"];                            %
539
540                             % Add nodes info from current result
541                             Aux = [ResCurrent(:,1),ResCurrent(:,jj+1)];
542                             AddNodes = sprintf('%d  %0.8f \n',Aux');
543                             AddNodes = AddNodes(1:size(AddNodes,2)-1);
544                             Msh = [Msh;AddNodes];
545
546                             Msh = [Msh;"$EndNodeData"];                 %
```

```
                        $EndNodeData
547                     Msh = [Msh;""];
548                 elseif CheckGmshPrint(1,jj) == 1 && isempty(
                        ResCurrent)
549                     fprintf(2,"Results for %s were not
                        previously generated during simulation
                        of %s. %s cannot be printed. \n",
                        ResultType,ResProc{i,1},ResCompView(i
                        -1,jj));
550                 end
551             end
552
553         case "Plastic_Strain"
554             CheckGmshPrint = PrintGmsh{i-1,1};                %
                    Check if result was request to be printed on
                    *.msh file format
555             for jj=1:size(CheckGmshPrint,2)
556                 if CheckGmshPrint(1,jj) == 1  && ~isempty(
                        ResCurrent)  % Check if result was
                        requested to be printed on *.msh file
                        format and if result was generated by Ansys
557
558                     Msh = [Msh;"$NodeData"];                  %
                        $NodeData
559                     Msh = [Msh;"1"];                          %
560
561                     View = strcat('"',ResItem(i-1,1),'_',
                        ResCompView(i-1,jj),'"');      % View
                        name
562                     Msh = [Msh;View];
563
564                     Msh = [Msh;"1"];                          %
565                     Msh = [Msh;string(Counter)];              %
                        Counter (replacing time step)
566                     Msh = [Msh;"4"];                          %
567                     Msh = [Msh;"0"];                          %
                        Step number (It's not used, so it can
                        be always 0)
568                     Msh = [Msh;"1"];                          %
                        Number of components
569                     Msh = [Msh;string(size(ResCurrent,1))]; %
                        Number of nodes
570                     Msh = [Msh;"0"];                          %
```

```
571
572                         % Add nodes info from current result
573                         Aux = [ResCurrent(:,1),ResCurrent(:,jj+1)];
574                         AddNodes = sprintf('%d  %0.8f \n',Aux');
575                         AddNodes = AddNodes(1:size(AddNodes,2)-1);
576                         Msh = [Msh;AddNodes];
577
578                         Msh = [Msh;"$EndNodeData"];             %
                               $EndNodeData
579                         Msh = [Msh;""];
580                     elseif CheckGmshPrint(1,jj) == 1  && isempty(
                           ResCurrent)
581                         fprintf(2,"Results for %s were not
                               previously generated during simulation
                               of %s. %s cannot be printed. \n",
                               ResultType,ResProc{i,1},ResCompView(i
                               -1,jj));
582                     end
583                 end
584
585             case "Total_Strain"
586                 CheckGmshPrint = PrintGmsh{i-1,1};                %
                       Check if result was request to be printed on
                      *.msh file format
587                 for jj=1:size(CheckGmshPrint,2)
588                     if CheckGmshPrint(1,jj) == 1  && ~isempty(
                           ResCurrent)  % Check if result was
                           requested to be printed on *.msh file
                           format and if result was generated by Ansys
589
590                         Msh = [Msh;"$NodeData"];                %
                               $NodeData
591                         Msh = [Msh;"1"];                        %
592
593                         View = strcat('"',ResItem(i-1,1),'_',
                               ResCompView(i-1,jj),'"');      % View
                               name
594                         Msh = [Msh;View];
595
596                         Msh = [Msh;"1"];                        %
597                         Msh = [Msh;string(Counter)];            %
                               Counter (replacing time step)
598                         Msh = [Msh;"4"];                        %
```

```
599                         Msh = [Msh;"0"];                            %
                                Step number (It's not used, so it can
                                be always 0)
600                         Msh = [Msh;"1"];                            %
                                Number of components
601                         Msh = [Msh;string(size(ResCurrent,1))]; %
                                Number of nodes
602                         Msh = [Msh;"0"];                            %
603
604                         % Add nodes info from current result
605                         Aux = [ResCurrent(:,1),ResCurrent(:,jj+1)];
606                         AddNodes = sprintf('%d  %0.8f \n',Aux');
607                         AddNodes = AddNodes(1:size(AddNodes,2)-1);
608                         Msh = [Msh;AddNodes];
609
610                         Msh = [Msh;"$EndNodeData"];                 %
                                $EndNodeData
611                         Msh = [Msh;""];
612                     elseif CheckGmshPrint(1,jj) == 1  && isempty(
                            ResCurrent)
613                         fprintf(2,"Results for %s were not
                                previously generated during simulation
                                of %s. %s cannot be printed. \n",
                                ResultType,ResProc{i,1},ResCompView(i
                                -1,jj));
614                     end
615                 end
616
617             case "Principal_Stress"
618                 CheckGmshPrint = PrintGmsh{i-1,1};                  %
                        Check if result was request to be printed on
                        *.msh file format
619                 for jj=1:size(CheckGmshPrint,2)
620                     if CheckGmshPrint(1,jj) == 1  && ~isempty(
                            ResCurrent)  % Check if result was
                            requested to be printed on *.msh file
                            format and if result was generated by Ansys
621
622                         Msh = [Msh;"$NodeData"];                    %
                                $NodeData
623                         Msh = [Msh;"1"];                            %
624
625                         View = strcat('"',ResItem(i-1,1),'_',
```

```
                                ResCompView(i-1,jj),'"');       % View
                                    name
626                             Msh = [Msh;View];
627
628                             Msh = [Msh;"1"];                            %
629                             Msh = [Msh;string(Counter)];                %
                                    Counter (replacing time step)
630                             Msh = [Msh;"4"];                            %
631                             Msh = [Msh;"0"];                            %
                                    Step number (It's not used, so it can
                                    be always 0)
632                             Msh = [Msh;"1"];                            %
                                    Number of components
633                             Msh = [Msh;string(size(ResCurrent,1))]; %
                                    Number of nodes
634                             Msh = [Msh;"0"];                            %
635
636                             % Add nodes info from current result
637                             Aux = [ResCurrent(:,1),ResCurrent(:,jj+1)];
638                             AddNodes = sprintf('%d  %0.8f \n',Aux');
639                             AddNodes = AddNodes(1:size(AddNodes,2)-1);
640                             Msh = [Msh;AddNodes];
641
642                             Msh = [Msh;"$EndNodeData"];                 %
                                    $EndNodeData
643                             Msh = [Msh;""];
644                         elseif CheckGmshPrint(1,jj) == 1  && isempty(
                                ResCurrent)
645                             fprintf(2,"Results for %s were not
                                    previously generated during simulation
                                    of %s. %s cannot be printed. \n",
                                    ResultType,ResProc{i,1},ResCompView(i
                                    -1,jj));
646                         end
647                     end
648                 end
649
650             end
651
652
653
654             % Save *.msh file
655             Msh = [MshInitializeAll;Msh];
```

```
656         FileName = strcat(erase(ResProc{1,j},'.txt'),'_RES','.msh');
657         fid = fopen(strcat(FolderPost,'\',FileName),'w');
658         fprintf(fid,'%s \n',Msh);
659         fclose(fid);
660 
661     end
662 
663 end
664 
665 %% Plot hotspots in Gmsh (accorging to Alert Matrix)
666 if ResGmsh(2,1) == true         % Plot hotspots
667     Counter=0;                                                          %
        Counter (replacing time step)
668     for j=StartNewCases:size(ResAlert,2)                                %
        Loop through ResAlert columns
669         Msh=[];                                                         %
            Start a new *.msh file
670         Counter=Counter+1;                                              %
            Counter (replacing time step for each new *.msh generated)
671         for i=2:size(ResAlert,1)                                        %
            Loop through ResAlert rows
672 
673             ResCurrent = ResAlert{i,j};                                 %
                Get current result
674             ResultType = ResAlert{i,1};                                 %
                Get result type
675 
676             switch ResultType
677                 case "Displacement"
678 
679                     for ii=2:size(ResCurrent,1)                         %
                        Loop through ResCurrent rows
680 
681                         Aux = [ResProc{i,j}(:,1)];
682                         Aux(:,2) = 0;                                   %
                            Give Hotspot with Alert Zero to all nodes
683                         for jj=4:size(ResCurrent,2)                     %
                            Loop through ResCurrent columns
684 
685                             try
686                                 UpdateHS = str2double(ResCurrent{ii,jj
                                    }(:,1:2));    % Update Hotspots
687                             catch
```

```
688                                 UpdateHS = [];

                                        % Make Hotspots matrix empty if
                                        there are no results
689                                 end
690
691                                 if ~isempty(UpdateHS)
                                                                        %
                                        Skip if Hotspots matrix is empty
692                                  [~,idx] =  ismember(UpdateHS(:,1),Aux
                                        (:,1));        % Get idx from Aux
693                                  Aux(idx,2) = str2double(ResCurrent{1,jj
                                        });          % Apply alert from
                                        Alert Matrix
694                                 end
695                             end
696
697                             if ~isempty(UpdateHS)
                                                                        %
                                    Skip if Hotspots matrix is empty
698                                 % Add hotspot information to *.msh file
699                                 Msh = [Msh;"$NodeData"];                %
                                        $NodeData
700                                 Msh = [Msh;"1"];                        %
701
702                                 ComponentColum = find(ResCurrent{ii,1} ==
                                        ResItem);                           %
                                     Get column for component name
703                                 GetComponentName = ResCompView(
                                        ComponentColum,str2double(ResCurrent{ii
                                        ,2}));    % Get component name
704
705                                 View = strcat('"','A',string(ii-1),'_',
                                        ResCurrent{ii,1},'_',GetComponentName,'
                                        "'); % View name
706                                 Msh = [Msh;View];
707
708                                 Msh = [Msh;"1"];                        %
709                                 Msh = [Msh;string(Counter)];            %
                                        Counter (replacing time step)
710                                 Msh = [Msh;"4"];                        %
711                                 Msh = [Msh;"0"];                        %
                                        Step number (It's not used, so it can
```

```
                        be always 0)
712                     Msh = [Msh;"1"];                        %
                        Number of components
713                     Msh = [Msh;string(size(Aux,1))];        %
                        Number of nodes
714                     Msh = [Msh;"0"];                        %
715
716                     % Add nodes info from current result with
                        alert levels
717                     AddNodes = sprintf('%d  %0.0f \n',Aux');
718                     AddNodes = AddNodes(1:size(AddNodes,2)-1);
719                     Msh = [Msh;AddNodes];
720
721                     Msh = [Msh;"$EndNodeData"];             %
                        $EndNodeData
722                     Msh = [Msh;""];                         %
723                 end
724
725             end
726
727         case "Stress"
728
729             for ii=2:size(ResCurrent,1)                     %
                    Loop through ResCurrent rows
730
731                 Aux = [ResProc{i,j}(:,1)];
732                 Aux(:,2) = 0;                               %
                    Give Hotspot with Alert Zero to all nodes
733                 for jj=4:size(ResCurrent,2)                 %
                    Loop through ResCurrent columns
734
735                     try
736                         UpdateHS = str2double(ResCurrent{ii,jj
                            }(:,1:2));    % Update Hotspots
737                     catch
738                         UpdateHS = [];

                            % Make Hotspots matrix empty if
                            there are no results
739                     end
740
741                     if ~isempty(UpdateHS)
                                                            %
```

```
                    Skip if Hotspots matrix is empty
742                 [~,idx] =  ismember(UpdateHS(:,1),Aux
                        (:,1));         % Get idx from Aux
743                 Aux(idx,2) = str2double(ResCurrent{1,jj
                        });           % Apply alert from
                        Alert Matrix
744             end
745         end
746
747         if ~isempty(UpdateHS)
                                                        %
                    Skip if Hotspots matrix is empty
748             % Add hotspot information to *.msh file
749             Msh = [Msh;"$NodeData"];                  %
                    $NodeData
750             Msh = [Msh;"1"];                          %
751
752             ComponentColum = find(ResCurrent{ii,1} ==
                    ResItem);                              %
                     Get column for component name
753             GetComponentName = ResCompView(
                    ComponentColum,str2double(ResCurrent{ii
                    ,2}));     % Get component name
754
755             View = strcat('"','A',string(ii-1),'_',
                    ResCurrent{ii,1},'_',GetComponentName,'
                    "'); % View name
756             Msh = [Msh;View];
757
758             Msh = [Msh;"1"];                          %
759             Msh = [Msh;string(Counter)];              %
                    Counter (replacing time step)
760             Msh = [Msh;"4"];                          %
761             Msh = [Msh;"0"];                          %
                    Step number (It's not used, so it can
                    be always 0)
762             Msh = [Msh;"1"];                          %
                    Number of components
763             Msh = [Msh;string(size(Aux,1))];          %
                    Number of nodes
764             Msh = [Msh;"0"];                          %
765
766             % Add nodes info from current result with
```

```
                                alert levels
767                         AddNodes = sprintf('%d  %0.0f \n',Aux');
768                         AddNodes = AddNodes(1:size(AddNodes,2)-1);
769                         Msh = [Msh;AddNodes];
770
771                         Msh = [Msh;"$EndNodeData"];             %
                               $EndNodeData
772                         Msh = [Msh;""];                         %
773                     end
774
775                 end
776
777             case "Elastic_Strain"
778
779                 for ii=2:size(ResCurrent,1)                     %
                       Loop through ResCurrent rows
780
781                     Aux = [ResProc{i,j}(:,1)];
782                     Aux(:,2) = 0;                               %
                           Give Hotspot with Alert Zero to all nodes
783                     for jj=4:size(ResCurrent,2)                 %
                           Loop through ResCurrent columns
784
785                         try
786                             UpdateHS = str2double(ResCurrent{ii,jj
                                   }(:,1:2));    % Update Hotspots
787                         catch
788                             UpdateHS = [];

                                % Make Hotspots matrix empty if
                                there are no results
789                         end
790
791                         if ~isempty(UpdateHS)
                                                                    %
                               Skip if Hotspots matrix is empty
792                             [~,idx] =  ismember(UpdateHS(:,1),Aux
                                   (:,1));        % Get idx from Aux
793                             Aux(idx,2) = str2double(ResCurrent{1,jj
                                   });          % Apply alert from
                                Alert Matrix
794                         end
795                     end
```

```
796
797                     if ~isempty(UpdateHS)
                                                                %
                        Skip if Hotspots matrix is empty
798                     % Add hotspot information to *.msh file
799                     Msh = [Msh;"$NodeData"];                %
                            $NodeData
800                     Msh = [Msh;"1"];                        %
801
802                     ComponentColum = find(ResCurrent{ii,1} ==
                            ResItem);                           %
                             Get column for component name
803                     GetComponentName = ResCompView(
                            ComponentColum,str2double(ResCurrent{ii
                            ,2}));    % Get component name
804
805                     View = strcat('"','A',string(ii-1),'_',
                            ResCurrent{ii,1},'_',GetComponentName,'
                            "'); % View name
806                     Msh = [Msh;View];
807
808                     Msh = [Msh;"1"];                        %
809                     Msh = [Msh;string(Counter)];            %
                            Counter (replacing time step)
810                     Msh = [Msh;"4"];                        %
811                     Msh = [Msh;"0"];                        %
                            Step number (It's not used, so it can
                            be always 0)
812                     Msh = [Msh;"1"];                        %
                            Number of components
813                     Msh = [Msh;string(size(Aux,1))];        %
                            Number of nodes
814                     Msh = [Msh;"0"];                        %
815
816                     % Add nodes info from current result with
                            alert levels
817                     AddNodes = sprintf('%d  %0.0f \n',Aux');
818                     AddNodes = AddNodes(1:size(AddNodes,2)-1);
819                     Msh = [Msh;AddNodes];
820
821                     Msh = [Msh;"$EndNodeData"];             %
                            $EndNodeData
822                     Msh = [Msh;""];                         %
```

```
823                     end
824
825                 end
826
827             case "Plastic_Strain"
828
829                 for ii=2:size(ResCurrent,1)                         %
                        Loop through ResCurrent rows
830
831                     Aux = [ResProc{i,j}(:,1)];
832                     Aux(:,2) = 0;                                   %
                        Give Hotspot with Alert Zero to all nodes
833                     for jj=4:size(ResCurrent,2)                     %
                        Loop through ResCurrent columns
834
835                         try
836                             UpdateHS = str2double(ResCurrent{ii,jj
                                }(:,1:2));    % Update Hotspots
837                         catch
838                             UpdateHS = [];

                                % Make Hotspots matrix empty if
                                there are no results
839                         end
840
841                         if ~isempty(UpdateHS)
                                                                    %
                            Skip if Hotspots matrix is empty
842                             [~,idx] =  ismember(UpdateHS(:,1),Aux
                                (:,1));        % Get idx from Aux
843                             Aux(idx,2) = str2double(ResCurrent{1,jj
                                });          % Apply alert from
                                Alert Matrix
844                         end
845                     end
846
847                     if ~isempty(UpdateHS)
                                                                    %
                        Skip if Hotspots matrix is empty
848                         % Add hotspot information to *.msh file
849                         Msh = [Msh;"$NodeData"];                    %
                            $NodeData
850                         Msh = [Msh;"1"];                            %
```

```
851
852                         ComponentColum = find(ResCurrent{ii,1} ==
                                ResItem);                                %
                                 Get column for component name
853                         GetComponentName = ResCompView(
                                ComponentColum,str2double(ResCurrent{ii
                                ,2}));     % Get component name
854
855                         View = strcat('"','A',string(ii-1),'_',
                                ResCurrent{ii,1},'_',GetComponentName,'
                                "'); % View name
856                         Msh = [Msh;View];
857
858                         Msh = [Msh;"1"];                               %
859                         Msh = [Msh;string(Counter)];                   %
                                Counter (replacing time step)
860                         Msh = [Msh;"4"];                               %
861                         Msh = [Msh;"0"];                               %
                                Step number (It's not used, so it can
                                be always 0)
862                         Msh = [Msh;"1"];                               %
                                Number of components
863                         Msh = [Msh;string(size(Aux,1))];               %
                                Number of nodes
864                         Msh = [Msh;"0"];                               %
865
866                         % Add nodes info from current result with
                                alert levels
867                         AddNodes = sprintf('%d  %0.0f \n',Aux');
868                         AddNodes = AddNodes(1:size(AddNodes,2)-1);
869                         Msh = [Msh;AddNodes];
870
871                         Msh = [Msh;"$EndNodeData"];                    %
                                $EndNodeData
872                         Msh = [Msh;""];                                %
873                     end
874
875                 end
876
877             case "Total_Strain"
878
879                 for ii=2:size(ResCurrent,1)                            %
                        Loop through ResCurrent rows
```

```
880
881                     Aux = [ResProc{i,j}(:,1)];
882                     Aux(:,2) = 0;                                   %
                            Give Hotspot with Alert Zero to all nodes
883                     for jj=4:size(ResCurrent,2)                     %
                            Loop through ResCurrent columns
884
885                         try
886                             UpdateHS = str2double(ResCurrent{ii,jj
                                }(:,1:2));    % Update Hotspots
887                         catch
888                             UpdateHS = [];
                                % Make Hotspots matrix empty if
                                there are no results
889                         end
890
891                         if ~isempty(UpdateHS)
                                                                        %
                                Skip if Hotspots matrix is empty
892                             [~,idx] =  ismember(UpdateHS(:,1),Aux
                                (:,1));        % Get idx from Aux
893                             Aux(idx,2) = str2double(ResCurrent{1,jj
                                });          % Apply alert from
                                Alert Matrix
894                         end
895                     end
896
897                     if ~isempty(UpdateHS)
                                                                      %
                            Skip if Hotspots matrix is empty
898                         % Add hotspot information to *.msh file
899                         Msh = [Msh;"$NodeData"];                    %
                            $NodeData
900                         Msh = [Msh;"1"];                            %
901
902                         ComponentColum = find(ResCurrent{ii,1} ==
                            ResItem);                                    %
                             Get column for component name
903                         GetComponentName = ResCompView(
                            ComponentColum,str2double(ResCurrent{ii
                            ,2}));    % Get component name
904
```

```
905                         View = strcat('"','A',string(ii-1),'_',
                                ResCurrent{ii,1},'_',GetComponentName,'
                                "'); % View name
906                         Msh = [Msh;View];
907
908                         Msh = [Msh;"1"];                          %
909                         Msh = [Msh;string(Counter)];              %
                                Counter (replacing time step)
910                         Msh = [Msh;"4"];                          %
911                         Msh = [Msh;"0"];                          %
                                Step number (It's not used, so it can
                                be always 0)
912                         Msh = [Msh;"1"];                          %
                                Number of components
913                         Msh = [Msh;string(size(Aux,1))];          %
                                Number of nodes
914                         Msh = [Msh;"0"];                          %
915
916                         % Add nodes info from current result with
                                alert levels
917                         AddNodes = sprintf('%d  %0.0f \n',Aux');
918                         AddNodes = AddNodes(1:size(AddNodes,2)-1);
919                         Msh = [Msh;AddNodes];
920
921                         Msh = [Msh;"$EndNodeData"];               %
                                $EndNodeData
922                         Msh = [Msh;""];                           %
923                     end
924
925                 end
926
927             case "Principal_Stress"
928
929                 for ii=2:size(ResCurrent,1)                       %
                        Loop through ResCurrent rows
930
931                     Aux = [ResProc{i,j}(:,1)];
932                     Aux(:,2) = 0;                                 %
                            Give Hotspot with Alert Zero to all nodes
933                     for jj=4:size(ResCurrent,2)                   %
                            Loop through ResCurrent columns
934
935                         try
```

```
936                             UpdateHS = str2double(ResCurrent{ii,jj
                                    }(:,1:2));    % Update Hotspots
937                         catch
938                             UpdateHS = [];

                                % Make Hotspots matrix empty if
                                there are no results
939                         end
940
941                         if ~isempty(UpdateHS)
                                                                  %
                            Skip if Hotspots matrix is empty
942                             [~,idx] =  ismember(UpdateHS(:,1),Aux
                                    (:,1));        % Get idx from Aux
943                             Aux(idx,2) = str2double(ResCurrent{1,jj
                                    });          % Apply alert from
                                Alert Matrix
944                         end
945                     end
946
947                     if ~isempty(UpdateHS)
                                                                  %
                        Skip if Hotspots matrix is empty
948                         % Add hotspot information to *.msh file
949                         Msh = [Msh;"$NodeData"];                %
                            $NodeData
950                         Msh = [Msh;"1"];                        %
951
952                         ComponentColum = find(ResCurrent{ii,1} ==
                                ResItem);                          %
                             Get column for component name
953                         GetComponentName = ResCompView(
                                ComponentColum,str2double(ResCurrent{ii
                                ,2}));    % Get component name
954
955                         View = strcat('"','A',string(ii-1),'_',
                                ResCurrent{ii,1},'_',GetComponentName,'
                                "'); % View name
956                         Msh = [Msh;View];
957
958                         Msh = [Msh;"1"];                        %
959                         Msh = [Msh;string(Counter)];            %
                            Counter (replacing time step)
```

```
960                             Msh = [Msh;"4"];                        %
961                             Msh = [Msh;"0"];                        %
                                    Step number (It's not used, so it can
                                    be always 0)
962                             Msh = [Msh;"1"];                        %
                                    Number of components
963                             Msh = [Msh;string(size(Aux,1))];        %
                                    Number of nodes
964                             Msh = [Msh;"0"];                        %
965
966                             % Add nodes info from current result with
                                    alert levels
967                             AddNodes = sprintf('%d  %0.0f \n',Aux');
968                             AddNodes = AddNodes(1:size(AddNodes,2)-1);
969                             Msh = [Msh;AddNodes];
970
971                             Msh = [Msh;"$EndNodeData"];             %
                                    $EndNodeData
972                             Msh = [Msh;""];                         %
973                         end
974
975                     end
976
977             end
978
979         end
980
981         % Save *.msh file
982         Msh = [MshInitializeAll;Msh];
983         FileName = strcat(erase(ResAlert{1,j},'.txt'),'_HS','.msh');
984         fid = fopen(strcat(FolderPost,'\',FileName),'w');
985         fprintf(fid,'%s \n',Msh);
986         fclose(fid);
987
988     end
989
990 end
991
992 %% Plot bodies in alert in Gmsh (according to Alert Matrix)
993 if ResGmsh(3,1) == true         % Plot hotspots
994     Counter=0;                                                      %
        Counter (replacing time step)
995     for j=StartNewCases:size(ResAlert,2)                            %
```

```
        Loop through ResAlert columns
996     Counter=Counter+1;                                              %
           Counter (replacing time step for each new *.msh generated)
997     for i=2:size(ResAlert,1)                                        %
           Loop through ResAlert rows
998
999         ResCurrent = ResAlert{i,j};                                 %
               Get current result
1000        ResultType = ResAlert{i,1};                                 %
               Get result type
1001
1002        if ~isempty(ResAlert)
1003
1004            switch ResultType
1005                case "Displacement"
1006
1007                    for ii=2:size(ResCurrent,1)
                                                            % Loop
                           through ResCurrent rows
1008                        BodiesInAlert = [];
                                                                    %
                           Start bodies in alert for each row in
                           ResCurrent
1009                        Aux = [];
1010                        Msh=[];

                           % Start a new *.msh file
1011
1012                        for jj=4:size(ResCurrent,2)
                                                            % Loop
                           through ResCurrent columns
1013
1014                            try
1015                                Aux = ResCurrent{ii,jj}(:,:);
                                                        % Update
                                   Bodies
1016                            catch
1017                                Aux = [];

                                   % Make Hotspots matrix empty if
                                    there are no results
1018                            end
1019
```

```
1020                    if ~isempty(Aux)
                            % Skip if Hotspots matrix is empty
1021
1022                        Aux = unique(Aux(:,3));
1023
1024                        % Check if any body has a previous
                                alert level (they should be
                                updated with highest alert
                                level)
1025                        try
1026                            [~,idx] = ismember(Aux(:,1),
                                    BodiesInAlert(:,1));
1027                            if idx ~= 0
1028                                BodiesInAlert(idx,:) = [];
1029                            end
1030                        catch
1031                            % do nothing
1032                        end
1033
1034                        Aux(:,2) = ResCurrent{ii,jj}(1,2);
1035                        BodiesInAlert=[BodiesInAlert;Aux];
                                                % Update
                                Bodies
1036
1037                    end
1038                end
1039
1040                if ~isempty(BodiesInAlert)
                                                        %
                    Skip if BodiesInAlert matrix is empty
1041
1042                    ElementsOtherBodies = ElementsRange
                            (:,1);                  % Get all
                            element IDs
1043                    for iii=1:size(BodiesInAlert,1)
1044                        idx = find(BodiesInAlert(iii,1)==
                                BodyList(:,1));       % Find
                                index of body in BodyInfo
1045                        ElementsInAlert = str2double(
                                BodyInfo{1,idx}(2:end,1));
                                % Get elements from BodyInfo
                                related to BodiesInAlert
```

```
1046                    ElementsInAlert(:,2) = str2double(
                            BodiesInAlert(iii,2));    %
                            Assign alert level
1047
1048                    [~,idx] = ismember(ElementsInAlert
                            (:,1),ElementsOtherBodies);   %
                             Get index of ElementsInAlert
                            from ElementsOtherBodies
1049                    ElementsOtherBodies(idx) = [];

                            % Delete ElementsInAlert from
                            ElementsOtherBodies
1050
1051                    Msh = [Msh;"$ElementData"];
                                    % Start element data
1052                    Msh = [Msh;"1"];
1053
1054                    View = strcat('"','_',BodiesInAlert
                            (iii,1),'"');  % View name
1055                    View = strrep(View,' ','_');
1056                    Msh = [Msh;View];             % Add
                            view name
1057
1058                    Msh = [Msh;"1"];
1059                    Msh = [Msh;string(Counter)];
                                    % Counter (replacing
                             time step)
1060                    Msh = [Msh;"4"];
1061                    Msh = [Msh;"0"];
                                                % Step
                            number (It's not used, so it
                            can be always 0)
1062                    Msh = [Msh;"1"];
                                                % Number
                             of components
1063
1064                    Msh = [Msh;size(ElementsInAlert,1)
                            ];    % Number of elements
1065                    Msh = [Msh;"0"];
1066
1067                    % Add nodes info from current
                            result with alert levels
1068                    AddElements = sprintf('%d  %0.0f \n
```

```
                                    ',ElementsInAlert');
1069                                AddElements = AddElements(1:size(
                                        AddElements,2)-1);
1070                                Msh = [Msh;AddElements];
1071
1072                                Msh = [Msh;"$EndElementData"];
                                            % End Element Data
1073                                Msh = [Msh;""];
1074
1075                            end
1076
1077                        end
1078
1079                        if ~isempty(BodiesInAlert)
                                                                %
                             Skip if BodiesInAlert matrix is empty
1080
1081                            % Bodies with level alert = zero
1082                            ElementsOtherBodies(:,2) = 0;    % Give
                                zero for all elements from bodies
                                with alert level = 0
1083
1084                            Msh = [Msh;"$ElementData"];          %
                                Start element data
1085                            Msh = [Msh;"1"];
1086
1087                            AddBodyName = strcat('"','Other_Bodies'
                                ,'"');
1088                            Msh = [Msh;AddBodyName];             %
                                Add body name
1089
1090                            Msh = [Msh;"1"];
1091                            Msh = [Msh;string(Counter)];
                                            % Counter (replacing
                                time step)
1092                            Msh = [Msh;"4"];
1093                            Msh = [Msh;"0"];
                                                        % Step
                                number (It's not used, so it can be
                                 always 0)
1094                            Msh = [Msh;"1"];
                                                        % Number of
                                components
```

```
1095
1096                    Msh = [Msh;size(ElementsOtherBodies,1)
                            ];    % Number of elements
1097                    Msh = [Msh;"0"];
1098
1099                    % Add nodes info from current result
                            with alert levels
1100                    AddElements = sprintf('%d  %0.0f \n',
                            ElementsOtherBodies');
1101                    AddElements = AddElements(1:size(
                            AddElements,2)-1);
1102                    Msh = [Msh;AddElements];
1103
1104                    Msh = [Msh;"$EndElementData"];      %
                            End Element Data
1105                    Msh = [Msh;""];
1106
1107                    % Save *.msh file
1108                    Msh = [MshInitializeAll;Msh];
1109                    FileName = strcat(erase(ResAlert{1,j},'
                            .txt'),'_SAD_A',string(ii-1),'.msh'
                            );
1110                    fid = fopen(strcat(FolderPost,'\',
                            FileName),'w');
1111                    fprintf(fid,'%s \n',Msh);
1112                    fclose(fid);
1113
1114                    % Create *.msh.opt file
1115                    Aux = ["View[***VN***].ColormapNumber =
                             4;"];           % ***VN*** stands
                            for View Number
1116                    Aux = [Aux;"View[***VN***].CustomMin=
                            0;"];
1117                    Aux = [Aux;"View[***VN***].CustomMax =
                            ***MAL***;"];    % ***MAL*** stands
                             for Maximum Alert Level
1118                    Aux = [Aux;"View[***VN***].RangeType =
                            2;"];
1119                    Aux = [Aux;"View[***VN***].Visible =
                            0;"];
1120
1121                    MAL=0;      % Get maximum alert level
                            for current result
```

```
1122                            for iiii=4:size(ResCurrent,2)
1123                                if str2double(ResCurrent{1,iiii}) >
                                        MAL
1124                                    MAL = str2double(ResCurrent{1,
                                            iiii});
1125                                end
1126                            end
1127
1128                            % Create .msh.opt content for all views
1129                            Opt=[];
1130                            for iiii=1:size(BodiesInAlert,1)+1
1131                                ViewOptions = strrep(Aux,'***VN***'
                                        ,string(iiii-1));
1132                                ViewOptions = strrep(ViewOptions,'
                                        ***MAL***',string(MAL));
1133                                Opt=[Opt;ViewOptions];
1134                            end
1135
1136                            % Save *.msh.opt file
1137                            FileName = strcat(erase(ResAlert{1,j},'
                                    .txt'),'_SAD_A',string(ii-1),'.msh.
                                    opt');
1138                            fid = fopen(strcat(FolderPost,'\',
                                    FileName),'w');
1139                            fprintf(fid,'%s \n',Opt);
1140                            fclose(fid);
1141
1142                        end
1143                    end
1144
1145                case "Stress"
1146
1147                    for ii=2:size(ResCurrent,1)
                                                        % Loop
                            through ResCurrent rows
1148                        BodiesInAlert = [];
                                                                %
                            Start bodies in alert for each row in
                            ResCurrent
1149                        Aux = [];
1150                        Msh=[];

                            % Start a new *.msh file
```

```
1151
1152                        for jj=4:size(ResCurrent,2)
                                                    % Loop
                               through ResCurrent columns
1153
1154                            try
1155                                Aux = ResCurrent{ii,jj}(:,:);
                                                    % Update
                                       Bodies
1156                            catch
1157                                Aux = [];

                                       % Make Hotspots matrix empty if
                                        there are no results
1158                            end
1159
1160                            if ~isempty(Aux)

                                   % Skip if Hotspots matrix is empty
1161
1162                                Aux = unique(Aux(:,3));
1163
1164                                % Check if any body has a previous
                                       alert level (they should be
                                       updated with highest alert
                                       level)
1165                                try
1166                                   [~,idx] = ismember(Aux(:,1),
                                          BodiesInAlert(:,1));
1167                                   if idx ~= 0
1168                                       BodiesInAlert(idx,:) = [];
1169                                   end
1170                                catch
1171                                    % do nothing
1172                                end
1173
1174                                Aux(:,2) = ResCurrent{ii,jj}(1,2);
1175                                BodiesInAlert=[BodiesInAlert;Aux];
                                                    % Update
                                       Bodies
1176
1177                            end
1178                        end
```

```
1179
1180                        if ~isempty(BodiesInAlert)
                                                                %
                            Skip if BodiesInAlert matrix is empty
1181
1182                            ElementsOtherBodies = ElementsRange
                                (:,1);                  % Get all
                                element IDs
1183                            for iii=1:size(BodiesInAlert,1)
1184                                idx = find(BodiesInAlert(iii,1)==
                                    BodyList(:,1));        % Find
                                    index of body in BodyInfo
1185                                ElementsInAlert = str2double(
                                    BodyInfo{1,idx}(2:end,1));
                                    % Get elements from BodyInfo
                                    related to BodiesInAlert
1186                                ElementsInAlert(:,2) = str2double(
                                    BodiesInAlert(iii,2));    %
                                    Assign alert level
1187
1188                                [~,idx] = ismember(ElementsInAlert
                                    (:,1),ElementsOtherBodies);   %
                                     Get index of ElementsInAlert
                                    from ElementsOtherBodies
1189                                ElementsOtherBodies(idx) = [];

                                    % Delete ElementsInAlert from
                                    ElementsOtherBodies
1190
1191                                Msh = [Msh;"$ElementData"];
                                            % Start element data
1192                                Msh = [Msh;"1"];
1193
1194                                View = strcat('"','_',BodiesInAlert
                                    (iii,1),'"');  % View name
1195                                View = strrep(View,' ','_');
1196                                Msh = [Msh;View];            % Add
                                    view name
1197
1198                                Msh = [Msh;"1"];
1199                                Msh = [Msh;string(Counter)];
                                                % Counter (replacing
                                     time step)
```

```
1200                                Msh = [Msh;"4"];
1201                                Msh = [Msh;"0"];
                                                        % Step
                                    number (It's not used, so it
                                    can be always 0)
1202                                Msh = [Msh;"1"];
                                                        % Number
                                     of components
1203
1204                                Msh = [Msh;size(ElementsInAlert,1)
                                    ];    % Number of elements
1205                                Msh = [Msh;"0"];
1206
1207                                % Add nodes info from current
                                    result with alert levels
1208                                AddElements = sprintf('%d  %0.0f \n
                                    ',ElementsInAlert');
1209                                AddElements = AddElements(1:size(
                                    AddElements,2)-1);
1210                                Msh = [Msh;AddElements];
1211
1212                                Msh = [Msh;"$EndElementData"];
                                            % End Element Data
1213                                Msh = [Msh;""];
1214
1215                            end
1216
1217                        end
1218
1219                        if ~isempty(BodiesInAlert)
                                                                %
                             Skip if BodiesInAlert matrix is empty
1220
1221                            % Bodies with level alert = zero
1222                            ElementsOtherBodies(:,2) = 0;   % Give
                                zero for all elements from bodies
                                with alert level = 0
1223
1224                            Msh = [Msh;"$ElementData"];         %
                                Start element data
1225                            Msh = [Msh;"1"];
1226
1227                            AddBodyName = strcat('"','Other_Bodies'
```

```
                    ,'"');
1228                Msh = [Msh;AddBodyName];            %
                    Add body name
1229
1230                Msh = [Msh;"1"];
1231                Msh = [Msh;string(Counter)];
                                % Counter (replacing
                    time step)
1232                Msh = [Msh;"4"];
1233                Msh = [Msh;"0"];
                                              % Step
                    number (It's not used, so it can be
                     always 0)
1234                Msh = [Msh;"1"];
                                              % Number of
                    components
1235
1236                Msh = [Msh;size(ElementsOtherBodies,1)
                    ];    % Number of elements
1237                Msh = [Msh;"0"];
1238
1239                % Add nodes info from current result
                    with alert levels
1240                AddElements = sprintf('%d  %0.0f \n',
                    ElementsOtherBodies');
1241                AddElements = AddElements(1:size(
                    AddElements,2)-1);
1242                Msh = [Msh;AddElements];
1243
1244                Msh = [Msh;"$EndElementData"];      %
                    End Element Data
1245                Msh = [Msh;""];
1246
1247                % Save *.msh file
1248                Msh = [MshInitializeAll;Msh];
1249                FileName = strcat(erase(ResAlert{1,j},'
                    .txt'),'_SAD_A',string(ii-1),'.msh'
                    );
1250                fid = fopen(strcat(FolderPost,'\',
                    FileName),'w');
1251                fprintf(fid,'%s \n',Msh);
1252                fclose(fid);
1253
```

```
1254                    % Create *.msh.opt file
1255                    Aux = ["View[***VN***].ColormapNumber =
                            4;"];           % ***VN*** stands
                            for View Number
1256                    Aux = [Aux;"View[***VN***].CustomMin=
                            0;"];
1257                    Aux = [Aux;"View[***VN***].CustomMax =
                            ***MAL***;"];    % ***MAL*** stands
                             for Maximum Alert Level
1258                    Aux = [Aux;"View[***VN***].RangeType =
                            2;"];
1259                    Aux = [Aux;"View[***VN***].Visible =
                            0;"];
1260
1261                    MAL=0;      % Get maximum alert level
                            for current result
1262                    for iiii=4:size(ResCurrent,2)
1263                        if str2double(ResCurrent{1,iiii}) >
                                MAL
1264                            MAL = str2double(ResCurrent{1,
                                    iiii});
1265                        end
1266                    end
1267
1268                    % Create .msh.opt content for all views
1269                    Opt=[];
1270                    for iiii=1:size(BodiesInAlert,1)+1
1271                        ViewOptions = strrep(Aux,'***VN***'
                                ,string(iiii-1));
1272                        ViewOptions = strrep(ViewOptions,'
                                ***MAL***',string(MAL));
1273                        Opt=[Opt;ViewOptions];
1274                    end
1275
1276                    % Save *.msh.opt file
1277                    FileName = strcat(erase(ResAlert{1,j},'
                            .txt'),'_SAD_A',string(ii-1),'.msh.
                            opt');
1278                    fid = fopen(strcat(FolderPost,'\',
                            FileName),'w');
1279                    fprintf(fid,'%s \n',Opt);
1280                    fclose(fid);
1281
```

```
1282                            end
1283                        end
1284
1285                    case "Elastic_Strain"
1286
1287                        for ii=2:size(ResCurrent,1)
                                                        % Loop
                               through ResCurrent rows
1288                            BodiesInAlert = [];
                                                                %
                                Start bodies in alert for each row in
                                ResCurrent
1289                            Aux = [];
1290                            Msh=[];

                                % Start a new *.msh file
1291
1292                            for jj=4:size(ResCurrent,2)
                                                        % Loop
                                through ResCurrent columns
1293
1294                             try
1295                                 Aux = ResCurrent{ii,jj}(:,:);
                                                        % Update
                                     Bodies
1296                             catch
1297                                 Aux = [];

                                     % Make Hotspots matrix empty if
                                      there are no results
1298                             end
1299
1300                             if ~isempty(Aux)

                                 % Skip if Hotspots matrix is empty
1301
1302                                 Aux = unique(Aux(:,3));
1303
1304                                 % Check if any body has a previous
                                     alert level (they should be
                                     updated with highest alert
                                     level)
1305                                 try
```

```
1306                                    [~,idx] = ismember(Aux(:,1),
                                           BodiesInAlert(:,1));
1307                                    if idx ~= 0
1308                                        BodiesInAlert(idx,:) = [];
1309                                    end
1310                                catch
1311                                    % do nothing
1312                                end
1313
1314                                Aux(:,2) = ResCurrent{ii,jj}(1,2);
1315                                BodiesInAlert=[BodiesInAlert;Aux];
                                                          % Update
                                    Bodies
1316
1317                            end
1318                        end
1319
1320                        if ~isempty(BodiesInAlert)
                                                                  %
                             Skip if BodiesInAlert matrix is empty
1321
1322                            ElementsOtherBodies = ElementsRange
                                (:,1);                    % Get all
                                element IDs
1323                            for iii=1:size(BodiesInAlert,1)
1324                                idx = find(BodiesInAlert(iii,1)==
                                    BodyList(:,1));        % Find
                                    index of body in BodyInfo
1325                                ElementsInAlert = str2double(
                                    BodyInfo{1,idx}(2:end,1));
                                    % Get elements from BodyInfo
                                    related to BodiesInAlert
1326                                ElementsInAlert(:,2) = str2double(
                                    BodiesInAlert(iii,2));    %
                                    Assign alert level
1327
1328                                [~,idx] = ismember(ElementsInAlert
                                    (:,1),ElementsOtherBodies);   %
                                     Get index of ElementsInAlert
                                    from ElementsOtherBodies
1329                                ElementsOtherBodies(idx) = [];

                                    % Delete ElementsInAlert from
```

```
                    ElementsOtherBodies
1330
1331                Msh = [Msh;"$ElementData"];
                            % Start element data
1332                Msh = [Msh;"1"];
1333
1334                View = strcat('"','_',BodiesInAlert
                    (iii,1),'"');  % View name
1335                View = strrep(View,' ','_');
1336                Msh = [Msh;View];             % Add
                    view name
1337
1338                Msh = [Msh;"1"];
1339                Msh = [Msh;string(Counter)];
                              % Counter (replacing
                     time step)
1340                Msh = [Msh;"4"];
1341                Msh = [Msh;"0"];
                                          % Step
                    number (It's not used, so it
                    can be always 0)
1342                Msh = [Msh;"1"];
                                          % Number
                     of components
1343
1344                Msh = [Msh;size(ElementsInAlert,1)
                    ];    % Number of elements
1345                Msh = [Msh;"0"];
1346
1347                % Add nodes info from current
                    result with alert levels
1348                AddElements = sprintf('%d  %0.0f \n
                    ',ElementsInAlert');
1349                AddElements = AddElements(1:size(
                    AddElements,2)-1);
1350                Msh = [Msh;AddElements];
1351
1352                Msh = [Msh;"$EndElementData"];
                            % End Element Data
1353                Msh = [Msh;""];
1354
1355            end
1356
```

```
1357                        end
1358
1359                        if ~isempty(BodiesInAlert)
                                                                %
                             Skip if BodiesInAlert matrix is empty
1360
1361                            % Bodies with level alert = zero
1362                            ElementsOtherBodies(:,2) = 0;   % Give
                                zero for all elements from bodies
                                with alert level = 0
1363
1364                            Msh = [Msh;"$ElementData"];         %
                                Start element data
1365                            Msh = [Msh;"1"];
1366
1367                            AddBodyName = strcat('"','Other_Bodies'
                                ,'"');
1368                            Msh = [Msh;AddBodyName];            %
                                Add body name
1369
1370                            Msh = [Msh;"1"];
1371                            Msh = [Msh;string(Counter)];
                                            % Counter (replacing
                                time step)
1372                            Msh = [Msh;"4"];
1373                            Msh = [Msh;"0"];
                                                        % Step
                                number (It's not used, so it can be
                                 always 0)
1374                            Msh = [Msh;"1"];
                                                        % Number of
                                components
1375
1376                            Msh = [Msh;size(ElementsOtherBodies,1)
                                ];    % Number of elements
1377                            Msh = [Msh;"0"];
1378
1379                            % Add nodes info from current result
                                with alert levels
1380                            AddElements = sprintf('%d  %0.0f \n',
                                ElementsOtherBodies');
1381                            AddElements = AddElements(1:size(
                                AddElements,2)-1);
```

```
1382                    Msh = [Msh;AddElements];
1383
1384                    Msh = [Msh;"$EndElementData"];      % End Element Data
1385                    Msh = [Msh;""];
1386
1387                    % Save *.msh file
1388                    Msh = [MshInitializeAll;Msh];
1389                    FileName = strcat(erase(ResAlert{1,j},'.txt'),'_SAD_A',string(ii-1),'.msh');
1390                    fid = fopen(strcat(FolderPost,'\',FileName),'w');
1391                    fprintf(fid,'%s \n',Msh);
1392                    fclose(fid);
1393
1394                    % Create *.msh.opt file
1395                    Aux = ["View[***VN***].ColormapNumber = 4;"];           % ***VN*** stands for View Number
1396                    Aux = [Aux;"View[***VN***].CustomMin= 0;"];
1397                    Aux = [Aux;"View[***VN***].CustomMax = ***MAL***;"];    % ***MAL*** stands for Maximum Alert Level
1398                    Aux = [Aux;"View[***VN***].RangeType = 2;"];
1399                    Aux = [Aux;"View[***VN***].Visible = 0;"];
1400
1401                    MAL=0;      % Get maximum alert level for current result
1402                    for iiii=4:size(ResCurrent,2)
1403                        if str2double(ResCurrent{1,iiii}) > MAL
1404                            MAL = str2double(ResCurrent{1,iiii});
1405                        end
1406                    end
1407
1408                    % Create .msh.opt content for all views
1409                    Opt=[];
1410                    for iiii=1:size(BodiesInAlert,1)+1
```

```
1411                                    ViewOptions = strrep(Aux,'***VN***'
                                            ,string(iiii-1));
1412                                    ViewOptions = strrep(ViewOptions,'
                                            ***MAL***',string(MAL));
1413                                    Opt=[Opt;ViewOptions];
1414                                end
1415
1416                                % Save *.msh.opt file
1417                                FileName = strcat(erase(ResAlert{1,j},'
                                        .txt'),'_SAD_A',string(ii-1),'.msh.
                                        opt');
1418                                fid = fopen(strcat(FolderPost,'\',
                                        FileName),'w');
1419                                fprintf(fid,'%s \n',Opt);
1420                                fclose(fid);
1421
1422                            end
1423                        end
1424
1425                    case "Plastic_Strain"
1426
1427                        for ii=2:size(ResCurrent,1)
                                                        % Loop
                                through ResCurrent rows
1428                            BodiesInAlert = [];
                                                                %
                                Start bodies in alert for each row in
                                ResCurrent
1429                            Aux = [];
1430                            Msh=[];

                                % Start a new *.msh file
1431
1432                            for jj=4:size(ResCurrent,2)
                                                        % Loop
                                through ResCurrent columns
1433
1434                                try
1435                                    Aux = ResCurrent{ii,jj}(:,:);
                                                    % Update
                                        Bodies
1436                                catch
1437                                    Aux = [];
```

```
                                % Make Hotspots matrix empty if
                                 there are no results
1438                        end
1439
1440                        if ~isempty(Aux)

                               % Skip if Hotspots matrix is empty
1441
1442                            Aux = unique(Aux(:,3));
1443
1444                            % Check if any body has a previous
                                   alert level (they should be
                                   updated with highest alert
                                   level)
1445                            try
1446                               [~,idx] = ismember(Aux(:,1),
                                      BodiesInAlert(:,1));
1447                               if idx ~= 0
1448                                   BodiesInAlert(idx,:) = [];
1449                               end
1450                            catch
1451                                % do nothing
1452                            end
1453
1454                            Aux(:,2) = ResCurrent{ii,jj}(1,2);
1455                            BodiesInAlert=[BodiesInAlert;Aux];
                                                        % Update
                                   Bodies
1456
1457                        end
1458                    end
1459
1460                    if ~isempty(BodiesInAlert)
                                                              %
                        Skip if BodiesInAlert matrix is empty
1461
1462                        ElementsOtherBodies = ElementsRange
                               (:,1);                  % Get all
                               element IDs
1463                        for iii=1:size(BodiesInAlert,1)
1464                            idx = find(BodiesInAlert(iii,1)==
                                   BodyList(:,1));        % Find
```

```
                                    index of body in BodyInfo
1465                                ElementsInAlert = str2double(
                                    BodyInfo{1,idx}(2:end,1));
                                    % Get elements from BodyInfo
                                    related to BodiesInAlert
1466                                ElementsInAlert(:,2) = str2double(
                                    BodiesInAlert(iii,2));    %
                                    Assign alert level
1467
1468                                [~,idx] = ismember(ElementsInAlert
                                    (:,1),ElementsOtherBodies);   %
                                     Get index of ElementsInAlert
                                    from ElementsOtherBodies
1469                                ElementsOtherBodies(idx) = [];

                                    % Delete ElementsInAlert from
                                    ElementsOtherBodies
1470
1471                                Msh = [Msh;"$ElementData"];
                                            % Start element data
1472                                Msh = [Msh;"1"];
1473
1474                                View = strcat('"','_',BodiesInAlert
                                    (iii,1),'"');  % View name
1475                                View = strrep(View,' ','_');
1476                                Msh = [Msh;View];             % Add
                                    view name
1477
1478                                Msh = [Msh;"1"];
1479                                Msh = [Msh;string(Counter)];
                                              % Counter (replacing
                                     time step)
1480                                Msh = [Msh;"4"];
1481                                Msh = [Msh;"0"];
                                                              % Step
                                    number (It's not used, so it
                                    can be always 0)
1482                                Msh = [Msh;"1"];
                                                              % Number
                                     of components
1483
1484                                Msh = [Msh;size(ElementsInAlert,1)
                                    ];    % Number of elements
```

```
1485                                    Msh = [Msh;"0"];
1486
1487                                    % Add nodes info from current
                                            result with alert levels
1488                                    AddElements = sprintf('%d  %0.0f \n
                                            ',ElementsInAlert');
1489                                    AddElements = AddElements(1:size(
                                            AddElements,2)-1);
1490                                    Msh = [Msh;AddElements];
1491
1492                                    Msh = [Msh;"$EndElementData"];
                                                % End Element Data
1493                                    Msh = [Msh;""];
1494
1495                                end
1496
1497                            end
1498
1499                            if ~isempty(BodiesInAlert)
                                                                        %
                                 Skip if BodiesInAlert matrix is empty
1500
1501                                % Bodies with level alert = zero
1502                                ElementsOtherBodies(:,2) = 0;    % Give
                                        zero for all elements from bodies
                                        with alert level = 0
1503
1504                                Msh = [Msh;"$ElementData"];          %
                                        Start element data
1505                                Msh = [Msh;"1"];
1506
1507                                AddBodyName = strcat('"','Other_Bodies'
                                        ,'"');
1508                                Msh = [Msh;AddBodyName];             %
                                        Add body name
1509
1510                                Msh = [Msh;"1"];
1511                                Msh = [Msh;string(Counter)];
                                                % Counter (replacing
                                        time step)
1512                                Msh = [Msh;"4"];
1513                                Msh = [Msh;"0"];
                                                            % Step
```

```
                                    number (It's not used, so it can be
                                     always 0)
1514                                Msh = [Msh;"1"];
                                                              % Number of
                                    components
1515
1516                                Msh = [Msh;size(ElementsOtherBodies,1)
                                    ];    % Number of elements
1517                                Msh = [Msh;"0"];
1518
1519                                % Add nodes info from current result
                                    with alert levels
1520                                AddElements = sprintf('%d  %0.0f \n',
                                    ElementsOtherBodies');
1521                                AddElements = AddElements(1:size(
                                    AddElements,2)-1);
1522                                Msh = [Msh;AddElements];
1523
1524                                Msh = [Msh;"$EndElementData"];      %
                                    End Element Data
1525                                Msh = [Msh;""];
1526
1527                                % Save *.msh file
1528                                Msh = [MshInitializeAll;Msh];
1529                                FileName = strcat(erase(ResAlert{1,j},'
                                    .txt'),'_SAD_A',string(ii-1),'.msh'
                                    );
1530                                fid = fopen(strcat(FolderPost,'\',
                                    FileName),'w');
1531                                fprintf(fid,'%s \n',Msh);
1532                                fclose(fid);
1533
1534                                % Create *.msh.opt file
1535                                Aux = ["View[***VN***].ColormapNumber =
                                     4;"];           % ***VN*** stands
                                    for View Number
1536                                Aux = [Aux;"View[***VN***].CustomMin=
                                    0;"];
1537                                Aux = [Aux;"View[***VN***].CustomMax =
                                    ***MAL***;"];    % ***MAL*** stands
                                     for Maximum Alert Level
1538                                Aux = [Aux;"View[***VN***].RangeType =
                                    2;"];
```

```
1539                            Aux = [Aux;"View[***VN***].Visible =
                                    0;"];
1540
1541                            MAL=0;      % Get maximum alert level
                                    for current result
1542                            for iiii=4:size(ResCurrent,2)
1543                                if str2double(ResCurrent{1,iiii}) >
                                        MAL
1544                                    MAL = str2double(ResCurrent{1,
                                        iiii});
1545                                end
1546                            end
1547
1548                            % Create .msh.opt content for all views
1549                            Opt=[];
1550                            for iiii=1:size(BodiesInAlert,1)+1
1551                                ViewOptions = strrep(Aux,'***VN***'
                                    ,string(iiii-1));
1552                                ViewOptions = strrep(ViewOptions,'
                                    ***MAL***',string(MAL));
1553                                Opt=[Opt;ViewOptions];
1554                            end
1555
1556                            % Save *.msh.opt file
1557                            FileName = strcat(erase(ResAlert{1,j},'
                                    .txt'),'_SAD_A',string(ii-1),'.msh.
                                    opt');
1558                            fid = fopen(strcat(FolderPost,'\',
                                    FileName),'w');
1559                            fprintf(fid,'%s \n',Opt);
1560                            fclose(fid);
1561
1562                        end
1563                    end
1564
1565                case "Total_Strain"
1566
1567                    for ii=2:size(ResCurrent,1)
                                                        % Loop
                            through ResCurrent rows
1568                            BodiesInAlert = [];
                                                                %
                                Start bodies in alert for each row in
```

```
                ResCurrent
1569            Aux = [];
1570            Msh=[];

                % Start a new *.msh file
1571
1572            for jj=4:size(ResCurrent,2)
                                                % Loop
                through ResCurrent columns
1573
1574             try
1575                 Aux = ResCurrent{ii,jj}(:,:);
                                              % Update
                    Bodies
1576             catch
1577                 Aux = [];

                    % Make Hotspots matrix empty if
                     there are no results
1578             end
1579
1580             if ~isempty(Aux)

                   % Skip if Hotspots matrix is empty
1581
1582                 Aux = unique(Aux(:,3));
1583
1584                 % Check if any body has a previous
                    alert level (they should be
                    updated with highest alert
                    level)
1585                 try
1586                    [~,idx] = ismember(Aux(:,1),
                       BodiesInAlert(:,1));
1587                    if idx ~= 0
1588                        BodiesInAlert(idx,:) = [];
1589                    end
1590                 catch
1591                     % do nothing
1592                 end
1593
1594                 Aux(:,2) = ResCurrent{ii,jj}(1,2);
1595                 BodiesInAlert=[BodiesInAlert;Aux];
```

```
                                                    % Update
                                    Bodies
1596
1597                            end
1598                        end
1599
1600                        if ~isempty(BodiesInAlert)
                                                                %
                             Skip if BodiesInAlert matrix is empty
1601
1602                            ElementsOtherBodies = ElementsRange
                                (:,1);                  % Get all
                                element IDs
1603                            for iii=1:size(BodiesInAlert,1)
1604                                idx = find(BodiesInAlert(iii,1)==
                                    BodyList(:,1));        % Find
                                    index of body in BodyInfo
1605                                ElementsInAlert = str2double(
                                    BodyInfo{1,idx}(2:end,1));
                                    % Get elements from BodyInfo
                                    related to BodiesInAlert
1606                                ElementsInAlert(:,2) = str2double(
                                    BodiesInAlert(iii,2));    %
                                    Assign alert level
1607
1608                                [~,idx] = ismember(ElementsInAlert
                                    (:,1),ElementsOtherBodies);   %
                                     Get index of ElementsInAlert
                                    from ElementsOtherBodies
1609                                ElementsOtherBodies(idx) = [];

                                    % Delete ElementsInAlert from
                                    ElementsOtherBodies
1610
1611                                Msh = [Msh;"$ElementData"];
                                                % Start element data
1612                                Msh = [Msh;"1"];
1613
1614                                View = strcat('"','_',BodiesInAlert
                                    (iii,1),'"');  % View name
1615                                View = strrep(View,' ','_');
1616                                Msh = [Msh;View];            % Add
                                    view name
```

```
1617
1618                                Msh = [Msh;"1"];
1619                                Msh = [Msh;string(Counter)];
                                                % Counter (replacing
                                    time step)
1620                                Msh = [Msh;"4"];
1621                                Msh = [Msh;"0"];
                                                                % Step
                                    number (It's not used, so it
                                    can be always 0)
1622                                Msh = [Msh;"1"];
                                                                % Number
                                    of components
1623
1624                                Msh = [Msh;size(ElementsInAlert,1)
                                    ];    % Number of elements
1625                                Msh = [Msh;"0"];
1626
1627                                % Add nodes info from current
                                    result with alert levels
1628                                AddElements = sprintf('%d  %0.0f \n
                                    ',ElementsInAlert');
1629                                AddElements = AddElements(1:size(
                                    AddElements,2)-1);
1630                                Msh = [Msh;AddElements];
1631
1632                                Msh = [Msh;"$EndElementData"];
                                            % End Element Data
1633                                Msh = [Msh;""];
1634
1635                            end
1636
1637                        end
1638
1639                        if ~isempty(BodiesInAlert)
                                                                    %
                            Skip if BodiesInAlert matrix is empty
1640
1641                            % Bodies with level alert = zero
1642                            ElementsOtherBodies(:,2) = 0;   % Give
                                zero for all elements from bodies
                                with alert level = 0
1643
```

```
1644                    Msh = [Msh;"$ElementData"];         %
                            Start element data
1645                    Msh = [Msh;"1"];
1646
1647                    AddBodyName = strcat('"','Other_Bodies'
                            ,'"');
1648                    Msh = [Msh;AddBodyName];            %
                            Add body name
1649
1650                    Msh = [Msh;"1"];
1651                    Msh = [Msh;string(Counter)];
                                        % Counter (replacing
                            time step)
1652                    Msh = [Msh;"4"];
1653                    Msh = [Msh;"0"];
                                                % Step
                            number (It's not used, so it can be
                             always 0)
1654                    Msh = [Msh;"1"];
                                                % Number of
                            components
1655
1656                    Msh = [Msh;size(ElementsOtherBodies,1)
                            ];    % Number of elements
1657                    Msh = [Msh;"0"];
1658
1659                    % Add nodes info from current result
                            with alert levels
1660                    AddElements = sprintf('%d  %0.0f \n',
                            ElementsOtherBodies');
1661                    AddElements = AddElements(1:size(
                            AddElements,2)-1);
1662                    Msh = [Msh;AddElements];
1663
1664                    Msh = [Msh;"$EndElementData"];      %
                            End Element Data
1665                    Msh = [Msh;""];
1666
1667                    % Save *.msh file
1668                    Msh = [MshInitializeAll;Msh];
1669                    FileName = strcat(erase(ResAlert{1,j},'
                            .txt'),'_SAD_A',string(ii-1),'.msh'
                            );
```

```
1670                    fid = fopen(strcat(FolderPost,'\',
                            FileName),'w');
1671                    fprintf(fid,'%s \n',Msh);
1672                    fclose(fid);
1673
1674                    % Create *.msh.opt file
1675                    Aux = ["View[***VN***].ColormapNumber =
                             4;"];           % ***VN*** stands
                            for View Number
1676                    Aux = [Aux;"View[***VN***].CustomMin=
                            0;"];
1677                    Aux = [Aux;"View[***VN***].CustomMax =
                            ***MAL***;"];    % ***MAL*** stands
                             for Maximum Alert Level
1678                    Aux = [Aux;"View[***VN***].RangeType =
                            2;"];
1679                    Aux = [Aux;"View[***VN***].Visible =
                            0;"];
1680
1681                    MAL=0;      % Get maximum alert level
                            for current result
1682                    for iiii=4:size(ResCurrent,2)
1683                        if str2double(ResCurrent{1,iiii}) >
                                 MAL
1684                            MAL = str2double(ResCurrent{1,
                                    iiii});
1685                        end
1686                    end
1687
1688                    % Create .msh.opt content for all views
1689                    Opt=[];
1690                    for iiii=1:size(BodiesInAlert,1)+1
1691                        ViewOptions = strrep(Aux,'***VN***'
                                ,string(iiii-1));
1692                        ViewOptions = strrep(ViewOptions,'
                                ***MAL***',string(MAL));
1693                        Opt=[Opt;ViewOptions];
1694                    end
1695
1696                    % Save *.msh.opt file
1697                    FileName = strcat(erase(ResAlert{1,j},'
                            .txt'),'_SAD_A',string(ii-1),'.msh.
                            opt');
```

```
1698                                fid = fopen(strcat(FolderPost,'\',
                                       FileName),'w');
1699                                fprintf(fid,'%s \n',Opt);
1700                                fclose(fid);
1701
1702                            end
1703                        end
1704
1705                    case "Principal_Stress"
1706
1707                        for ii=2:size(ResCurrent,1)
                                                          % Loop
                               through ResCurrent rows
1708                            BodiesInAlert = [];
                                                                  %
                               Start bodies in alert for each row in
                               ResCurrent
1709                            Aux = [];
1710                            Msh=[];

                               % Start a new *.msh file
1711
1712                            for jj=4:size(ResCurrent,2)
                                                          % Loop
                               through ResCurrent columns
1713
1714                             try
1715                                 Aux = ResCurrent{ii,jj}(:,:);
                                                        % Update
                                    Bodies
1716                             catch
1717                                 Aux = [];

                                    % Make Hotspots matrix empty if
                                     there are no results
1718                             end
1719
1720                             if ~isempty(Aux)

                                % Skip if Hotspots matrix is empty
1721
1722                                 Aux = unique(Aux(:,3));
1723
```

```
1724                                % Check if any body has a previous
                                        alert level (they should be
                                        updated with highest alert
                                        level)
1725                                try
1726                                    [~,idx] = ismember(Aux(:,1),
                                            BodiesInAlert(:,1));
1727                                    if idx ~= 0
1728                                        BodiesInAlert(idx,:) = [];
1729                                    end
1730                                catch
1731                                    % do nothing
1732                                end
1733
1734                                Aux(:,2) = ResCurrent{ii,jj}(1,2);
1735                                BodiesInAlert=[BodiesInAlert;Aux];
                                                        % Update
                                        Bodies
1736
1737                            end
1738                        end
1739
1740                        if ~isempty(BodiesInAlert)
                                                                %
                                Skip if BodiesInAlert matrix is empty
1741
1742                            ElementsOtherBodies = ElementsRange
                                    (:,1);                  % Get all
                                    element IDs
1743                            for iii=1:size(BodiesInAlert,1)
1744                                idx = find(BodiesInAlert(iii,1)==
                                        BodyList(:,1));         % Find
                                        index of body in BodyInfo
1745                                ElementsInAlert = str2double(
                                        BodyInfo{1,idx}(2:end,1));
                                        % Get elements from BodyInfo
                                        related to BodiesInAlert
1746                                ElementsInAlert(:,2) = str2double(
                                        BodiesInAlert(iii,2));    %
                                        Assign alert level
1747
1748                                [~,idx] = ismember(ElementsInAlert
                                        (:,1),ElementsOtherBodies);   %
```

```
                                    Get index of ElementsInAlert
                                   from ElementsOtherBodies
1749                               try
1750                                   ElementsOtherBodies(idx) = [];
                                                                  %
                                       Delete ElementsInAlert from
                                        ElementsOtherBodies
1751                               catch
1752                                   % do nothing
1753                                   break
1754                               end
1755
1756                               Msh = [Msh;"$ElementData"];
                                          % Start element data
1757                               Msh = [Msh;"1"];
1758
1759                               View = strcat('"','_',BodiesInAlert
                                   (iii,1),'"');  % View name
1760                               View = strrep(View,' ','_');
1761                               Msh = [Msh;View];            % Add
                                   view name
1762
1763                               Msh = [Msh;"1"];
1764                               Msh = [Msh;string(Counter)];
                                             % Counter (replacing
                                    time step)
1765                               Msh = [Msh;"4"];
1766                               Msh = [Msh;"0"];
                                                              % Step
                                   number (It's not used, so it
                                   can be always 0)
1767                               Msh = [Msh;"1"];
                                                              % Number
                                    of components
1768
1769                               Msh = [Msh;size(ElementsInAlert,1)
                                   ];    % Number of elements
1770                               Msh = [Msh;"0"];
1771
1772                               % Add nodes info from current
                                   result with alert levels
1773                               AddElements = sprintf('%d  %0.0f \n
                                   ',ElementsInAlert');
```

```
1774                                AddElements = AddElements(1:size(
                                        AddElements,2)-1);
1775                                Msh = [Msh;AddElements];
1776
1777                                Msh = [Msh;"$EndElementData"];
                                            % End Element Data
1778                                Msh = [Msh;""];
1779
1780                            end
1781
1782                        end
1783
1784                        if ~isempty(BodiesInAlert)
                                                                %
                             Skip if BodiesInAlert matrix is empty
1785
1786                            % Bodies with level alert = zero
1787                            ElementsOtherBodies(:,2) = 0;   % Give
                                zero for all elements from bodies
                                with alert level = 0
1788
1789                            Msh = [Msh;"$ElementData"];         %
                                Start element data
1790                            Msh = [Msh;"1"];
1791
1792                            AddBodyName = strcat('"','Other_Bodies'
                                ,'"');
1793                            Msh = [Msh;AddBodyName];            %
                                Add body name
1794
1795                            Msh = [Msh;"1"];
1796                            Msh = [Msh;string(Counter)];
                                            % Counter (replacing
                                time step)
1797                            Msh = [Msh;"4"];
1798                            Msh = [Msh;"0"];
                                                        % Step
                                number (It's not used, so it can be
                                 always 0)
1799                            Msh = [Msh;"1"];
                                                        % Number of
                                components
1800
```

```
1801                    Msh = [Msh;size(ElementsOtherBodies,1)
                            ];    % Number of elements
1802                    Msh = [Msh;"0"];
1803
1804                    % Add nodes info from current result
                            with alert levels
1805                    AddElements = sprintf('%d  %0.0f \n',
                            ElementsOtherBodies');
1806                    AddElements = AddElements(1:size(
                            AddElements,2)-1);
1807                    Msh = [Msh;AddElements];
1808
1809                    Msh = [Msh;"$EndElementData"];      %
                            End Element Data
1810                    Msh = [Msh;""];
1811
1812                    % Save *.msh file
1813                    Msh = [MshInitializeAll;Msh];
1814                    FileName = strcat(erase(ResAlert{1,j},'
                            .txt'),'_SAD_A',string(ii-1),'.msh'
                            );
1815                    fid = fopen(strcat(FolderPost,'\',
                            FileName),'w');
1816                    fprintf(fid,'%s \n',Msh);
1817                    fclose(fid);
1818
1819                    % Create *.msh.opt file
1820                    Aux = ["View[***VN***].ColormapNumber =
                             4;"];          % ***VN*** stands
                            for View Number
1821                    Aux = [Aux;"View[***VN***].CustomMin=
                            0;"];
1822                    Aux = [Aux;"View[***VN***].CustomMax =
                            ***MAL***;"];    % ***MAL*** stands
                             for Maximum Alert Level
1823                    Aux = [Aux;"View[***VN***].RangeType =
                            2;"];
1824                    Aux = [Aux;"View[***VN***].Visible =
                            0;"];
1825
1826                    MAL=0;      % Get maximum alert level
                            for current result
1827                    for iiii=4:size(ResCurrent,2)
```

```
1828                                    if str2double(ResCurrent{1,iiii}) >
                                            MAL
1829                                        MAL = str2double(ResCurrent{1,
                                               iiii});
1830                                    end
1831                                end
1832
1833                                % Create .msh.opt content for all views
1834                                Opt=[];
1835                                for iiii=1:size(BodiesInAlert,1)+1
1836                                    ViewOptions = strrep(Aux,'***VN***'
                                           ,string(iiii-1));
1837                                    ViewOptions = strrep(ViewOptions,'
                                           ***MAL***',string(MAL));
1838                                    Opt=[Opt;ViewOptions];
1839                                end
1840
1841                                % Save *.msh.opt file
1842                                FileName = strcat(erase(ResAlert{1,j},'
                                       .txt'),'_SAD_A',string(ii-1),'.msh.
                                       opt');
1843                                fid = fopen(strcat(FolderPost,'\',
                                       FileName),'w');
1844                                fprintf(fid,'%s \n',Opt);
1845                                fclose(fid);
1846
1847                            end
1848                        end
1849
1850                end
1851            end
1852
1853        end
1854
1855    end
1856
1857 end
1858
1859 clearvars -except FolderPost ResProc ResAlert AlertMatrix
1860
1861 %% Save processed results
1862 save(strcat(FolderPost,'\','- Processed_Results','.','mat'));     % Save
        information
```

```
% load(strcat(FolderPost,'\','- Processed_Results','.','mat'));     %
    Load information
fprintf("Processed results were saved \n")

%%  End of code
fclose all;
ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
fprintf('Elapsed time running C03_Post_Results.m is %s \n',ElapsedTime)
clear
return
```

C.1.5. Algoritmo Calc_Press

Algoritmo	Objetivo	Categoria
Calc_Press	Obter pressão hidrodinâmica para a malha em elementos finitos	Secundário

```
%% load("E:\Projetos\Enauta-DT\Modelos\Hull_Pressure_amort.mat");
%% Pressao = Calc_Press(1,11,0,90, Hull_Pressure);

function Pressao = Calc_Press(Hs, Periodo, incidencia, fase,
    Hull_Pressure)
    ID_inc = find(Hull_Pressure.Incidence==incidencia);
    if isempty(ID_inc); disp('Wave incidence not available'); Pressao
        =0; return; end
    ID_per = find(Hull_Pressure.Period==Periodo);
    if isempty(ID_per); disp('Period not available'); Pressao=0; return
        ; end
    Pressao =  real(Hull_Pressure.FaceVertexCData{ID_per,ID_inc}(:,1).*
        Hs.*exp(i*(fase+Hull_Pressure.FaceVertexCData{ID_per,ID_inc
        }(:,2))*pi/180));
end
```

C.1.6. Algoritmo Ini_A00_folders

Algoritmo	Objetivo	Categoria
Ini_A00_folders	Verificar e criar diretórios necessários (caso não existam)	Auxiliar

```
%%  Ini_A00_Folders
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   Nov 22, 2021
%   This script/function check for all needed folders and creates them if they do not exist
%  [pt-br] Este script/fun  o verifica todas as pastas necessrias e as cria caso elas n o existam

%% Notes:
%

%% Steps:
%  Load variables
%  Check if folder exists and create if doesn't
%  End of code

%% Load variables
fprintf(1,"Initializing Ini_A00_Folders \n")
tic

Ini_A00_initial_data
clearvars -except Var_Ini_A00_Folders

%% Check if folder exists and create if doesn't
for i=1:size(Var_Ini_A00_Folders,1)
    if ~exist(Var_Ini_A00_Folders{i}, 'dir')
       mkdir(Var_Ini_A00_Folders{i})
       fprintf(2,'Folder "%s" was created.\n',Var_Ini_A00_Folders{i})
    else
       fprintf(1,'Folder "%s" already exists.\n',Var_Ini_A00_Folders{i})
    end
end

%% Check if file "- SEGOFF.txt" exists and create if doesn't
if ~exist(strcat(Var_Ini_A00_Folders{3},'\- SEGOFF.txt'), 'file')
```

```
34
35      newline = ["Item    Value_[SI]      format"];
36      newline = [newline; "***Status_SEG***       -       %s"];
37      newline = [newline; "***SEG_par***  3       %.0f"];
38      newline = [newline; "***Value_SEGX***       0.      %.0f."];
39      newline = [newline; "***Value_SEGY***       0.      %.0f."];
40      newline = [newline; "***Value_SEGZ***       0.      %.0f."];
41
42      fid = fopen(strcat(Var_Ini_A00_Folders{3},'\- SEGOFF.txt'),'w');
43      fprintf(fid,'%s \n',newline);
44      fclose(fid);
45
46      fprintf(2,'File "%s" was created.\n',strcat(Var_Ini_A00_Folders{3},
            '\- SEGOFF.txt'))
47  else
48     fprintf(1,'File "%s" already exists.\n',strcat(Var_Ini_A00_Folders
           {3},'\- SEGOFF.txt'))
49  end
50
51  %% Check if file "- WaveOFF.txt" exists and create if doesn't
52  if ~exist(strcat(Var_Ini_A00_Folders{3},'\- WaveOFF.txt'), 'file')
53
54      newline = ["Item    Value_[SI]      format"];
55      newline = [newline; "***Case***     -       %s"];
56      newline = [newline; "***Status_Case***      -       %s"];
57      newline = [newline; "***Status_HDP***       -       %s"];
58      newline = [newline; "***HDP_par***  15      %.0f"];
59      newline = [newline; "***HDP_interp_met***   -       %s"];
60      newline = [newline; "***HDP_min_pres***     -       %s"];
61      newline = [newline; "***HDP_max_pres***     TRUE    %s"];
62      newline = [newline; "***HDP_Amplitude***    -       %s"];
63      newline = [newline; "***HDP_Period***       -       %s"];
64      newline = [newline; "***HDP_Direction***    -       %s"];
65      newline = [newline; "***cgloc_X***  0.000000000e+00 %.9E"];
66      newline = [newline; "***cgloc_Y***  0.000000000e+00 %.9E"];
67      newline = [newline; "***cgloc_Z***  0.000000000e+00 %.9E"];
68      newline = [newline; "***acel_X***   0.000000000e+00 %.9E"];
69      newline = [newline; "***acel_Y***   0.000000000e+00 %.9E"];
70      newline = [newline; "***acel_Z***   0.000000000e+00 %.9E"];
71      newline = [newline; "***dcgomg_X*** 0.000000000e+00 %.9E"];
72      newline = [newline; "***dcgomg_Y*** 0.000000000e+00 %.9E"];
73      newline = [newline; "***dcgomg_Z*** 0.000000000e+00 %.9E"];
74      newline = [newline; "***omega_X***  0.000000000e+00 %.9E"];
```

```
75      newline = [newline; "***omega_Y***  0.000000000e+00 %.9E"];
76      newline = [newline; "***omega_Z***  0.000000000e+00 %.9E"];
77      newline = [newline; "***domega_X*** 0.000000000e+00 %.9E"];
78      newline = [newline; "***domega_Y*** 0.000000000e+00 %.9E"];
79      newline = [newline; "***domega_Z*** 0.000000000e+00 %.9E"];
80      newline = [newline; "***cgomga_X*** 0.000000000e+00 %.9E"];
81      newline = [newline; "***cgomga_Y*** 0.000000000e+00 %.9E"];
82      newline = [newline; "***cgomga_Z*** 0.000000000e+00 %.9E"];
83      newline = [newline; "***Status_RF***        -       %s"];
84      newline = [newline; "***RF_par***   6       %.0f"];
85      newline = [newline; "***Value_RFXloc***     0.000000000e+00 %.9E"];
86      newline = [newline; "***Value_RFYloc***     0.000000000e+00 %.9E"];
87      newline = [newline; "***Value_RFZloc***     0.000000000e+00 %.9E"];
88      newline = [newline; "***Value_RFXcomp***    0.      %.0f."];
89      newline = [newline; "***Value_RFYcomp***    0.      %.0f."];
90      newline = [newline; "***Value_RFZcomp***    0.      %.0f."];
91
92      fid = fopen(strcat(Var_Ini_A00_Folders{6},'\- WaveOFF.txt'),'w');
93      fprintf(fid,'%s \n',newline);
94      fclose(fid);
95
96      fprintf(2,'File "%s" was created.\n',strcat(Var_Ini_A00_Folders{3},
            '\- WaveOFF.txt'))
97  else
98     fprintf(1,'File "%s" already exists.\n',strcat(Var_Ini_A00_Folders
           {3},'\- WaveOFF.txt'))
99  end
100
101 %%  End of code
102 fclose all;
103 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
104 fprintf('Elapsed time running Ini_A00_Folders.m is %s \n',ElapsedTime)
105 return
```

C.1.7. Algoritmo Ini_A00_get_parameters

Algoritmo	Objetivo	Categoria
Ini_A00_get_parameters	Obter valores de espessura de elementos de casca e viga do modelo de elementos finitos	Auxiliar

```
%%  Ini_A00_get_parameters
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   Dec 07, 2021
%   This script gets thickness values for shell and beams bodies from generic FEM model ("A02 dat base\*_origin.dat")
%  [pt-br] Este script/fun  o obt m valores de espessura de entidades de casca e viga do modelo MEF gen rico ("A02 dat base\*_origin.dat")

%% Notes:
%

%% Steps:
%  Load variables
%  Read Generic FEM model
%  Find and get information from FEM Model
%  Get info for shells, beams and model
%  Add Shell info to Model Summary
%  Add Beam (Rect and L section) info to Model Summary
%  Get info for Shell Summary Info
%  Get info for Beam Rect Summary Info
%  Get info for Beam L Summary Info
%  Save initial information for Shell (values from *_origin.dat)
%  Save initial information for Beam Rect (values from *_origin.dat)
%  Save initial information for Beam L (values from *_origin.dat)
%  End of code

%% Load variables
fprintf(1,"Initializing Ini_A00_get_parameters \n")
tic

%% Check if script was requested by GUI (using Manual or Automatic control)
try
   if control == "Manual"
```

```
32          fprintf("Control: GUI - %s \n",control)
33      elseif control == "Automatic"
34          fprintf("Control: GUI - %s \n",control)
35      else
36          control = "None";
37          fprintf("Control: GUI - %s \n",control)
38      end
39  catch
40      control = "None";
41      fprintf("Control: GUI - %s \n",control)
42  end
43
44  %% Load variables by running without GUI
45  if control == "None"
46      Ini_A00_initial_data
47      clearvars -except Var_Ini_A00_get_parameters
48
49      PathFemFile = Var_Ini_A00_get_parameters{1};
                                                              % Folder
          for Generic FEM model file
50      FolderCS = Var_Ini_A00_get_parameters{2};      % Folder for coupled
           systems
51      FolderLock = Var_Ini_A00_get_parameters{3};              % Folder
          for inspection
52  end
53
54  %% Read Generic FEM model
55  fprintf("Reading Generic FEM model. \n");
56  fid = fopen(PathFemFile,'r','n','UTF-8'); % Open Generic FEM model
57  fFemFile = fread(fid,'*char')';              % Read Generic FEM model
58  fclose(fid);
59
60  fFemFile = string(fFemFile);
61  fFemFile = regexprep(fFemFile,'\n','&%^');            % Replace break
      lines with codes
62  fFemFile = strip(split(fFemFile,'&%^'));              % Split
      characters and delete codes
63
64  %% Find and get information from FEM Model
65  StartShellInfo = "/com,********** Send Sheet Properties **********";
66  StartBeamInfo = "/com,********** Send Beam Properties **********";
67  EndBeamInfo = "!************************ Model Summary
      ********************";
```

```
68  EndModelSummary = "!************************ End Model Summary
        ********************";
69
70  StartShellInfoIdx = find(strcmp(fFemFile, StartShellInfo));
71  StartBeamInfoIdx = find(strcmp(fFemFile, StartBeamInfo));
72  EndBeamInfoIdx = find(strcmp(fFemFile, EndBeamInfo));
73  EndModelSummaryIdx = find(strcmp(fFemFile, EndModelSummary));
74
75  %% Get info for shells, beams and model
76  ShellInfo = fFemFile(StartShellInfoIdx+1:StartBeamInfoIdx-1);
77  BeamInfo = fFemFile(StartBeamInfoIdx+1:EndBeamInfoIdx-1);
78  ModelSummaryInfo = fFemFile(EndBeamInfoIdx+1:EndModelSummaryIdx-1);
79
80  ModelSummaryInfo = strip(split(ModelSummaryInfo,','));              %
        Split characters
81
82  %% Add Shell info to Model Summary
83  for i=1:size(ShellInfo,1)
84      RowCurrent = strip(split(ShellInfo(i,1),','))';                 %
            Split characters
85      if RowCurrent(1,1) == "sectype" || RowCurrent(1,1) == "SECTYPE"
86          ModelSummaryIdx = find(strcmp(ModelSummaryInfo(:,4), RowCurrent
                (1,2)));
87          ModelSummaryInfo(ModelSummaryIdx,5) = RowCurrent(1,3);
88          RowAfter = strip(split(ShellInfo(i+1,1),','))';
                                % Split characters
89          ModelSummaryInfo(ModelSummaryIdx,7) = RowAfter(1,2);
90      end
91  end
92
93  %% Add Beam (Rect and L section) info to Model Summary
94  for i=1:size(BeamInfo,1)
95      RowCurrent = strip(split(BeamInfo(i,1),','))';                  %
            Split characters
96      if RowCurrent(1,1) == "sectype" || RowCurrent(1,1) == "SECTYPE"
97          ModelSummaryIdx = find(strcmp(ModelSummaryInfo(:,4), RowCurrent
                (1,2)));
98          ModelSummaryInfo(ModelSummaryIdx,5) = RowCurrent(1,3);
99          if RowCurrent(1,4) == "RECT" || RowCurrent(1,4) == "rect"
100             ModelSummaryInfo(ModelSummaryIdx,6) = RowCurrent(1,4);
101             RowAfter = strip(split(BeamInfo(i+1,1),','))';
                                    % Split characters
102             ModelSummaryInfo(ModelSummaryIdx,7) = RowAfter(1,2);
```

```
103             ModelSummaryInfo(ModelSummaryIdx,8) = RowAfter(1,3);
104         elseif RowCurrent(1,4) == "l" || RowCurrent(1,4) == "L"
105             ModelSummaryInfo(ModelSummaryIdx,6) = RowCurrent(1,4);
106             RowAfter = strip(split(BeamInfo(i+1,1),','))';
                                % Split characters
107             ModelSummaryInfo(ModelSummaryIdx,7) = RowAfter(1,2);
108             ModelSummaryInfo(ModelSummaryIdx,8) = RowAfter(1,3);
109             ModelSummaryInfo(ModelSummaryIdx,9) = RowAfter(1,4);
110             ModelSummaryInfo(ModelSummaryIdx,10) = RowAfter(1,5);
111             RowAfter = strip(split(BeamInfo(i+2,1),','))';
                                % Split characters
112             ModelSummaryInfo(ModelSummaryIdx,11) = RowAfter(1,3);
113             ModelSummaryInfo(ModelSummaryIdx,12) = RowAfter(1,4);
114         end
115     end
116 end
117
118 clearvars -except ModelSummaryInfo FemFileOri FolderCS FolderLock
        PathFemFile
119
120 %% Get info for Shell Summary Info
121 ShellIdx = strcmpi(ModelSummaryInfo(:,5),"shell");
122 ShellIdx = find(ShellIdx == 1 );
123 ShellSummaryInfo = ModelSummaryInfo(ShellIdx,1:7);
124 ShellSummaryInfo(:,2:3) = [];
125 ShellSummaryInfo(:,3:4) = [];
126 TopRows = ["DATE","BODY_NUMBER","CURRENT_t1"];
127 ShellSummaryInfo = [TopRows;ShellSummaryInfo];
128
129 %% Get info for Beam Rect Summary Info
130 BeamRectIdx = strcmpi(ModelSummaryInfo(:,5),"beam") & strcmpi(
        ModelSummaryInfo(:,6),"rect");
131 BeamRectIdx = find(BeamRectIdx == 1 );
132 BeamRectSummaryInfo = ModelSummaryInfo(BeamRectIdx,1:8);
133 BeamRectSummaryInfo(:,2:3) = [];
134 BeamRectSummaryInfo(:,3:4) = [];
135 TopRows = ["DATE","BODY_NUMBER","CURRENT_B1","CURRENT_H1"];
136 BeamRectSummaryInfo = [TopRows;BeamRectSummaryInfo];
137
138 %% Get info for Beam L Summary Info
139 BeamLIdx = strcmpi(ModelSummaryInfo(:,5),"beam") & strcmpi(
        ModelSummaryInfo(:,6),"l");
140 BeamLIdx = find(BeamLIdx == 1 );
```

```
141 BeamLSummaryInfo = ModelSummaryInfo(BeamLIdx,1:12);
142 BeamLSummaryInfo(:,2:3) = [];
143 BeamLSummaryInfo(:,3:4) = [];
144 TopRows = ["DATE","BODY_NUMBER","CURRENT_W1","CURRENT_W2","CURRENT_t1
        ","CURRENT_t2","CURRENT_O1","CURRENT_O2"];
145 BeamLSummaryInfo = [TopRows;BeamLSummaryInfo];
146
147 fprintf("Saving information from '%s'. \n",PathFemFile);
148
149 InitialName = strsplit(string(PathFemFile),'\');
150 InitialName = InitialName(1,end);
151
152 %% Save initial information for Shell (values from *_origin.dat)
153 filename = strcat(FolderLock,'\',InitialName,'_shell.txt');
154 fprintf(1,"Saving information for Shell ('%s'). \n",filename);
155 writematrix(ShellSummaryInfo,filename,'Delimiter','tab');
156
157 %% Save initial information for Beam Rect (values from *_origin.dat)
158 filename = strcat(FolderLock,'\',InitialName,'_beam_rect.txt');
159 fprintf(1,"Saving information for Beam Rect ('%s'). \n",filename);
160 writematrix(BeamRectSummaryInfo,filename,'Delimiter','tab');
161
162 %% Save initial information for Beam L (values from *_origin.dat)
163 filename = strcat(FolderLock,'\',InitialName,'_beam_L.txt');
164 fprintf(1,"Saving information for Beam L ('%s'). \n",filename);
165 writematrix(BeamLSummaryInfo,filename,'Delimiter','tab');
166
167 %%  End of code
168 fclose all;
169 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
170 fprintf('Elapsed time running Ini_A00_get_parameters.m is %s \n',
        ElapsedTime)
171 clearvars
172 return
```

C.1.8. Algoritmo Ini_A00_post_bodies

Algoritmo	Objetivo	Categoria
Ini_A00_post_bodies	Obter e salvar informações das entidades do modelo de elementos finitos	Auxiliar

```
%%  Ini_A00_post_bodies
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   May 10, 2021
%   This script/function get and save information from generic FEM model bodies
%       [pt-br] Este script/fun  o obt m e salva informa  es das entidades do modelo gen rico MEF

%% Notes:
%  This scrip/function needs to be run only once

%% Steps:
%  Load variables
%  Check for bodies info or processed information for bodies
%  Process bodies info if it doesn't exist
%  Clear and save variables
%  End of code

%% Load variables
fprintf(1,"Initializing Ini_A00_post_bodies \n")
tic

% Check if script was requested by GUI (in Manual or Automatic control)
try
   if control == "Manual"
       fprintf("Control: GUI - %s \n",control)
   elseif control == "Automatic"
       fprintf("Control: GUI - %s \n",control)
   else
       control = "None";
       fprintf("Control: GUI - %s \n",control)
   end
catch
   control = "None";
   fprintf("Control: GUI - %s \n",control)
```

```
34 end
35
36 % Load variables by running without GUI
37 if control == "None"
38     Ini_A00_initial_data
39     clearvars -except Var_Ini_A00_post_bodies
40
41     FolderLock = Var_Ini_A00_post_bodies{1};
42     Direction = Var_Ini_A00_post_bodies{2};
43     MinRange = Var_Ini_A00_post_bodies{3};
44     MaxRange = Var_Ini_A00_post_bodies{4};
45     TolRange = Var_Ini_A00_post_bodies{5};
46     PathFemFile = Var_Ini_A00_post_bodies{6};
47 end
48
49 % Check locked results
50 try
51     load(strcat(FolderLock,'\','Direction_locked','.','mat'),'
           Direction_locked');  % Load information
52     Direction = Direction_locked;
53 catch
54     Direction_locked = Direction;
55     save(strcat(FolderLock,'\','Direction_locked','.','mat'),'
           Direction_locked');    % Save information
56     fprintf(1,"Locking variable: Direction \n")
57 end
58
59 try
60     load(strcat(FolderLock,'\','MinRange_locked','.','mat'),'
           MinRange_locked');  % Load information
61     MinRange = MinRange_locked;
62 catch
63     MinRange_locked = MinRange;
64     save(strcat(FolderLock,'\','MinRange_locked','.','mat'),'
           MinRange_locked');    % Save information
65     fprintf(1,"Locking variable: MinRange \n")
66 end
67
68 try
69     load(strcat(FolderLock,'\','MaxRange_locked','.','mat'),'
           MaxRange_locked');  % Load information
70     MaxRange = MaxRange_locked;
71 catch
```

```
72      MaxRange_locked = MaxRange;
73      save(strcat(FolderLock,'\','MaxRange_locked','.','mat'),'
            MaxRange_locked');    % Save information
74      fprintf(1,"Locking variable: MaxRange \n")
75  end
76
77  try
78      load(strcat(FolderLock,'\','TolRange_locked','.','mat'),'
            TolRange_locked');  % Load information
79      TolRange = TolRange_locked;
80  catch
81      TolRange_locked = TolRange;
82      save(strcat(FolderLock,'\','TolRange_locked','.','mat'),'
            TolRange_locked');    % Save information
83      fprintf(1,"Locking variable: TolRange \n")
84  end
85
86  clear Direction_locked MinRange_locked MaxRange_locked TolRange_locked
87
88  %% Check for bodies info or processed information for bodies
89  % Check for bodies info
90  CheckBodyInfoStructure = [];
91  try
92      CheckBodyInfoStructure = load(strcat(FolderLock,'\',"BodyInfo.mat")
            );
93      fprintf(1,"Information for bodies was loaded from file: BodyInfo.
            mat \n")
94
95      VarInfo = struct2cell(CheckBodyInfoStructure);      % Create cell
            for *.mat files
96
97      BodyInfo = cat(1, VarInfo{1});          % Get info for bodies
98      BodyList = cat(1, VarInfo{2});          % Get list of bodies
99      ElementsRange = cat(1, VarInfo{3});     % Get info for elements in
            range
100     NodeBodyElType = cat(1, VarInfo{5});     % Get info for bodies (
            Body Name, Element Type and Range)
101     NodesFem = cat(1, VarInfo{6});          % Get info for all nodes
102     NodesRange = cat(1, VarInfo{7});        % Get info for nodes in
            range
103
104     clearvars -except BodyInfo NodesFem NodesRange ElementsRange
            NodeBodyElType FolderLock BodyList
```

```
105     CheckBodyInfoStructure = "IsNotEmpty";
106 catch
107     % do nothing
108 end
109 
110 %% Process bodies info if it doesn't exist
111 BodyList = [];
112 if isempty(CheckBodyInfoStructure)
113     fprintf(1,"Processing information for bodies \n")
114 
115     % Input for range of results
116     MinRange = MinRange - TolRange;
117     MaxRange = MaxRange + TolRange;
118 
119     % Read Node info
120 
121     % Read Generic FEM model
122     fprintf(1,"Reading Generic FEM model. \n");
123     fid = fopen(PathFemFile,'r','n','UTF-8'); % Open Generic FEM model
124     fFemFile = fread(fid,'*char')';              % Read Generic FEM
        model
125     fclose(fid);
126 
127     fFemFile = string(fFemFile);
128     fFemFile = regexprep(fFemFile,'\n','&%^');            % Replace
        break lines with codes
129     fFemFile = strip(split(fFemFile,'&%^'));              % Split
        characters and delete codes
130 
131     % Find and get information from FEM Model
132     StartNodeInfo = "/com,*********** Nodes for the whole assembly
        ***********";    % Expression to find nodes - start
133     EndNodeInfo = "/com,*********** Nodes for all Remote Points
        ***********";       % Expression to find nodes - end
134 
135     StartNodeInfoIdx = find(strcmp(fFemFile, StartNodeInfo));
                              % Index for expression to find nodes -
        start
136     EndNodeInfoIdx = find(strcmp(fFemFile, EndNodeInfo));
                                  % Index for expression to find nodes
        - end
137 
138     % Array to receive all nodes from assembly
```

```
139     NodesFem = fFemFile(StartNodeInfoIdx+3:EndNodeInfoIdx-2);
140     NodesFem = regexprep(NodesFem,' +',' ');
141     NodesFem = split(NodesFem,' ');
142     NodesFem = str2double(NodesFem);
143 
144     % Create array for nodes in range
145     if Direction == 'x'         % Direction of model: x
146         OutRange = find(NodesFem(:,2) < (MinRange) | NodesFem(:,2) > (
                MaxRange));
147         NodesRange = NodesFem;
148         NodesRange(OutRange,:) = [];
149     end
150 
151     if Direction == 'y'         % Direction of model: x
152         OutRange = find(NodesFem(:,3) < (MinRange) | NodesFem(:,3) > (
                MaxRange));
153         NodesRange = NodesFem;
154         NodesRange(OutRange,:) = [];
155     end
156 
157     if Direction == 'z'         % Direction of model: x
158         OutRange = find(NodesFem(:,4) < (MinRange) | NodesFem(:,4) > (
                MaxRange));
159         NodesRange = NodesFem;
160         NodesRange(OutRange,:) = [];
161     end
162 
163     clearvars NodeX NodeY NodeZ NodeId NodeMatrixNewLine StringLine
            NumChar ExpToFind Cond Direction
164 
165     StartBodyInfo = regexp(fFemFile, '/com,********** Elements for
            Body\>');                % Expression to find bodies - start
166     StartBodyInfo = find(~cellfun(@isempty,StartBodyInfo));
                                        % Index for expression to find
            bodies - start
167 
168     EndBodyInfo = "/com,********** Send User Defined Coordinate System
            (s) **********";    % Expression to find bodies - end
169     EndBodyInfoIdx = find(strcmp(fFemFile, EndBodyInfo));
                                          % Index for expression to
            find bodies - end
170 
171     % Loop to get information for bodies
```

```
172     for y=1:size(StartBodyInfo,1)
173         % Get body name
174         GetBodyName = char(fFemFile(StartBodyInfo(y,1)));
175         k = strfind(GetBodyName,'"');
176         GetBodyName = GetBodyName(k(1)+1:k(2)-1);
177 
178         % Get element type for body
179         GetElType = char(fFemFile(StartBodyInfo(y,1)+1));
180         ToErase = ['et,',char(string(y)),','];
181         GetElType  = erase(GetElType,ToErase);
182 
183         % Get elements for body
184         if y==size(StartBodyInfo,1)
                                                        % Exception for
              last body
185             GetEl = fFemFile(StartBodyInfo(y,1)+5:EndBodyInfoIdx-4);
186         else
187             GetEl = fFemFile(StartBodyInfo(y,1)+5:StartBodyInfo(y+1,1)
                  -3);
188         end
189         GetEl = regexprep(GetEl,' +',' ');
190         GetEl = split(GetEl,' ');
191         GetEl(:,1:10)=[];
192         if string(GetElType) == "188"
                                                      % Get elements
              for element type 188
193             GetEl(:,4)=[];
194         end
195 
196         % Delete information for elements out of range
197         [NodeTest,NodeID]=ismember(str2double(GetEl(:,2:end)),OutRange)
              ;
198         NodeTest = sum(NodeTest,2);
199         NodeTest = find(NodeTest == 0);
200         GetEl = GetEl(NodeTest,:);
201 
202         % Update information from body (Name, Element type and Elements
              )
203         BodyInfo{y}(1,1) = string(GetBodyName);
                                        % Update body name
204         BodyInfo{y}(1,2) = GetElType;
                                                % Update element type
              for body
```

```
205         BodyInfo{y}(2:size(GetEl,1)+1,1:size(GetEl,2)) = GetEl;
                        % Update elements for body
206
207         Aux=[string(GetBodyName),string(GetElType)];
208         BodyList = [BodyList;Aux];
                                                    % Create body list
            ; Body name, Element type
209     end
210
211     % Check and delete bodies out of range
212     CheckBodyInfo = [];
213     for i=1:size(BodyInfo,2)
214         if size(BodyInfo{1,i},1) <= 1
215             CheckBodyInfo = [CheckBodyInfo;i];  % Get body ID to delete
216         end
217     end
218     BodyInfo(CheckBodyInfo) = [];               % Delete bodies out of
        range from BodyInfo
219     BodyList(CheckBodyInfo,:) = [];               % Delete bodies out
        of range from BodyList
220
221     % Get list of nodes related to each body and elements in range
222     ElementsRange = [];
223     NodeBodyElType = [];
224     for i=1:size(BodyInfo,2)
225         % Get list of nodes in range from body
226         NodesFromBody = BodyInfo{i}(2:end,2:end);
227         NodesFromBody = unique(NodesFromBody);
228         if size(NodesFromBody,1)==1                 % Exception for
            single element body in range
229             NodesFromBody=NodesFromBody';
230         end
231         NodesFromBody = sort(str2double(NodesFromBody));
232
233         % Array to get Node ID and related body
234         Aux = [];
235         Aux = string(NodesFromBody);
236         Aux(:,2) = BodyInfo{i}(1,1);
237         Aux(:,3) = BodyInfo{i}(1,2);
238         NodeBodyElType = [Aux;NodeBodyElType];
239
240         Aux = BodyInfo{i}(2:end,1:end);
241         if size(Aux,2) ~= 5
```

```
242             Aux(:,size(Aux,2)+1:5) = "";
243         end
244
245         % Get Elements in range
246         ElementsRange = [ElementsRange;Aux];
247     end
248
249     % Sort elements in range
250     ElementsRange = sortrows(str2double(ElementsRange),1);
251
252 %% Clear and save variables
253     clearvars -except BodyInfo NodesFem NodesRange ElementsRange
            NodeBodyElType FolderLock BodyList
254     save(strcat(FolderLock,'\','BodyInfo','.','mat'));  % Save
            information
255     % load(strcat(FolderLock,'\','BodyInfo','.','mat'));  % Save
            information
256     fprintf(1,"Information for bodies was processed and saved: BodyInfo
            .mat \n")
257     fclose all;
258 end
259
260 %%  End of code
261 clearvars
262 fclose all;
263 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
264 fprintf(1,'Elapsed time running D02_Post_Bodies.m is %s \n',ElapsedTime
        )
265 return
```

C.1.9. Algoritmo Ini_A01_equiv

Algoritmo	Objetivo	Categoria
Ini_A01_equiv	Identifica nós e elementos equivalentes entre a malha em elementos finitos e malha do modelo hidrodinâmico	Auxiliar

```
%%  Ini_A01_equiv
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   Sep 14, 2020
%   This script/function finds the equivalent nodes or elements from Mesh 1 (used in hydrodynamic analysis) in Mesh 2 (used in generic FEM model)

%       [pt-br] Este script/função encontra os nós ou elementos equivalentes da Malha 1 (usada na análise hidrodinâmica) na Malha 2 (malha do modelo MEF genérico)
%   Updated on Mar 26, 2021
%   Updated on Apr 21, 2021
%   Updated on Feb 17, 2022

%% Notes:
%  This scrip/function needs to be run only once
%  All nodes or elements from Mesh 1 must have an equivalent in Mesh 2
%  Required files:

%  If hydrodynamic analysis was processed to provide pressure on "nodes":
%       File with nodes from Mesh 1 (format: coordinate_X coordinate_Y coordinate_Z ; use tabulation to separate values; node ID is not needed)
%       IF APPLYING PRESSURE ON ELEMENTS: File with elements from Mesh 1 (format: node_ID_1 node_ID_2 node_ID_3 node_ID_4 ; use tabulation to separate values; element ID is not needed)
%       File with nodes from Mesh 2 (format: node_ID coordinate_X coordinate_Y coordinate_Z ; use tabulation to separate values; first line is neglected)
%       File with elements from Mesh 2 (format: element_ID element_type node_ID_1 node_ID_2 node_ID_3 node_ID_4; use tabulation to separate values; first line is neglected; element type is provided by ANSYS Mechanical)
```

```
21
22 %  If hydrodynamic analysis was processed to provide pressure on "
       elements":
23 %  ...
24
25 %% Steps:
26
27 %% Load variables
28 fprintf(1,"Initializing Ini_A01_equiv \n")
29 tic
30
31 % Check if script was requested by GUI (in Manual or Automatic control)
32 try
33    if control == "Manual"
34        fprintf("Control: GUI - %s \n",control)
35    elseif control == "Automatic"
36        fprintf("Control: GUI - %s \n",control)
37    else
38        control = "None";
39        fprintf("Control: GUI - %s \n",control)
40    end
41 catch
42    control = "None";
43    fprintf("Control: GUI - %s \n",control)
44 end
45
46 % Load variables by running without GUI
47 if control == "None"
48     Ini_A00_initial_data
49     clearvars -except Var_Ini_A01_equiv
50
51     FolderLock = Var_Ini_A01_equiv{1};
52     FolderCS = Var_Ini_A01_equiv{2};
53     FileOpt = Var_Ini_A01_equiv{3};
54     FEMNodesFile = Var_Ini_A01_equiv{4};
55     HDNodesFile = Var_Ini_A01_equiv{5};
56     FEMElementsFile = Var_Ini_A01_equiv{6};
57     HDElementsFile = Var_Ini_A01_equiv{7};
58     Direction = Var_Ini_A01_equiv{8};
59     Direction_transv_vertical = Var_Ini_A01_equiv{9};
60     TolHDMesh = Var_Ini_A01_equiv{10};
61
62 end
```

```
63
64 %% Read options file
65 FileDataOpt = extractFileText(strcat(FolderCS,'\',FileOpt,'.txt'));
                        % Get data from .txt file
66 FileDataOpt = regexprep(FileDataOpt,'\n','&%^');              % Replace
    break lines with codes
67 FileDataOpt = strip(split(FileDataOpt,'&%^'));                % Split
    characters and delete codes
68 FileDataOpt = string(FileDataOpt);                            % Convert
    to string
69 FileDataOpt = regexp(FileDataOpt,'\s','split');
70 if FileDataOpt{end,1} == ""
71     FileDataOpt(end,:) = [];
72 end
73
74 FileDataOpt = vertcat(FileDataOpt{:});
75
76 %  Check entity to apply pressure
77 FB = 'Wave_analysis';                                         % Get string
    (FB)
78 FBIdx = find(FB == FileDataOpt(:,2));                         % Get index
    for string
79 if FileDataOpt(FBIdx,3) == "NODE"                             % Check
    option for entity used to apply pressures informed by Options File
80     fprintf("Hydrodynamic analysis with pressure applied on nodes \n");
81     PressureEntity = 1;      % Nodes
82 elseif FileDataOpt(FBIdx,3) == "ELEMENT"                      % Check
    option for entity used to apply pressures informed by Options File
83     fprintf("Hydrodynamic analysis with pressure applied on elements \n
        ");
84     PressureEntity = 2;      % Elements
85 else
86     fprintf("Please check option for entity used to apply pressures
        informed by Options File \n");
87     fprintf("Option %s is not supported \n", FileDataOpt(FBIdx,3));
88     return
89 end
90
91 clearvars FileDataOpt FBIdx FB
92
93 %% Read file with nodes from FEM mesh
94 FileDataOpt = extractFileText(FEMNodesFile);                  % Get
    data from .txt file
```

```
95  FileDataOpt = regexprep(FileDataOpt,'\n','&%^');            % Replace
        break lines with codes
96  FileDataOpt = strip(split(FileDataOpt,'&%^'));               % Split
        characters and delete codes
97  FileDataOpt = string(FileDataOpt);                           % Convert
        to string
98  FileDataOpt = FileDataOpt(2:end,:);                           % Convert
        to string
99  if FileDataOpt(end,end) == ""
100     FileDataOpt = FileDataOpt(1:end-1,:);                          %
            Convert to string
101 end
102 FileDataOpt = regexp(FileDataOpt,'\s','split');
103 FileDataOpt = vertcat(FileDataOpt{:});
104 FileDataOpt = double(FileDataOpt);
105 FEMNodes = FileDataOpt;
106
107 clearvars FileDataOpt FEMNodesFile
108
109 %% Read file with nodes from HD mesh
110 FileDataOpt = extractFileText(HDNodesFile);                     % Get
        data from .txt file
111 FileDataOpt = regexprep(FileDataOpt,'\n','&%^');            % Replace
        break lines with codes
112 FileDataOpt = strip(split(FileDataOpt,'&%^'));               % Split
        characters and delete codes
113 FileDataOpt = string(FileDataOpt);                           % Convert
        to string
114 % FileDataOpt = FileDataOpt(2:end,:);                          %
        Convert to string
115 if FileDataOpt(end,end) == ""
116     FileDataOpt = FileDataOpt(1:end-1,:);                          %
            Convert to string
117 end
118 FileDataOpt = regexp(FileDataOpt,'\s','split');
119 FileDataOpt = vertcat(FileDataOpt{:});
120 FileDataOpt = double(FileDataOpt);
121 HDNodes = FileDataOpt;
122
123 clearvars FileDataOpt HDNodesPath
124
125 %% Read file with elements from FEM mesh
126 FileDataOpt = extractFileText(FEMElementsFile);                  %
```

```
    Get data from .txt file
FileDataOpt = regexprep(FileDataOpt,'\n','&%^');            % Replace
    break lines with codes
FileDataOpt = strip(split(FileDataOpt,'&%^'));               % Split
    characters and delete codes
FileDataOpt = string(FileDataOpt);                           % Convert
    to string
FileDataOpt = FileDataOpt(2:end,:);                           % Convert
    to string
if FileDataOpt(end,end) == ""
    FileDataOpt = FileDataOpt(1:end-1,:);                          %
        Convert to string
end
FileDataOpt = regexp(FileDataOpt,'\s','split');

% Get max values for columns
[s,d] = cellfun(@size,FileDataOpt);
out = max([s,d]);
MaxCol = out(1,2);

% Fill cells with empty cell they don't have MaxCol size for column
for i=1:size(FileDataOpt,1)
    CheckSize = size(FileDataOpt{i},2);

    if CheckSize < MaxCol
        for jj=(size(FileDataOpt{i},2)+1):MaxCol
            FileDataOpt{i}(1,jj) = "";
        end
    end
end
FileDataOpt = vertcat(FileDataOpt{:});
FEMElements = FileDataOpt;

clearvars -except FEMElements FEMNodes HDNodes PressureEntity
    FolderLock HDElementsFile Direction Direction_transv_vertical
    TolHDMesh

%% Read file with elements from HD mesh
if PressureEntity == 2
    FileDataOpt = extractFileText(HDElementsFile);                  %
        Get data from .txt file
    FileDataOpt = regexprep(FileDataOpt,'\n','&%^');            %
        Replace break lines with codes
```

```
160     FileDataOpt = strip(split(FileDataOpt,'&%^'));              % Split
            characters and delete codes
161     FileDataOpt = string(FileDataOpt);                          %
            Convert to string
162     if FileDataOpt(end,end) == ""
163         FileDataOpt = FileDataOpt(1:end-1,:);
                % Convert to string
164     end
165     FileDataOpt = regexp(FileDataOpt,'\s','split');
166     FileDataOpt = vertcat(FileDataOpt{:});
167     FileDataOpt = double(FileDataOpt);
168     HDElements = FileDataOpt;
169
170     clearvars -except FEMElements FEMNodes HDNodes PressureEntity
            FolderLock HDElements Direction Direction_transv_vertical
            TolHDMesh
171 end
172
173 clearvars -except FEMElements FEMNodes HDNodes PressureEntity
        FolderLock HDElements Direction Direction_transv_vertical TolHDMesh
174
175 %% Procedure to find equivalent nodes
176 if PressureEntity == 1   % Pressure entity = NODES
177
178     % Add Node ID to HD nodes
179     HDNodes(:,2:4) = HDNodes;
180     HDNodes(1:end,1) = 1:size(HDNodes,1);
181
182     % Round values to 6 decimals
183     FEMNodes(:,2:4) = round(FEMNodes(:,2:4),6);
184     HDNodes(:,2:4) = round(HDNodes(:,2:4),6);
185
186     %  Get min/max value for longitudinal direction of FEM model and
            delete any HD node out of range
187     switch Direction
188         case 'x'
189             MinValLongDir = min(FEMNodes(:,2));
190             MaxValLongDir = max(FEMNodes(:,2));
191
192             [IdxMinOut,~] = find(HDNodes(:,2) < MinValLongDir -
                    TolHDMesh);
193             [IdxMaxOut,~] = find(HDNodes(:,2) > MaxValLongDir +
                    TolHDMesh);
```

```
194         case 'y'
195             MinValLongDir = min(FEMNodes(:,3));
196             MaxValLongDir = max(FEMNodes(:,3));
197 
198             [IdxMinOut,~] = find(HDNodes(:,3) < MinValLongDir -
                    TolHDMesh);
199             [IdxMaxOut,~] = find(HDNodes(:,3) > MaxValLongDir +
                    TolHDMesh);
200         case 'z'
201             MinValLongDir = min(FEMNodes(:,4));
202             MaxValLongDir = max(FEMNodes(:,4));
203 
204             [IdxMinOut,~] = find(HDNodes(:,4) < MinValLongDir -
                    TolHDMesh);
205             [IdxMaxOut,~] = find(HDNodes(:,4) > MaxValLongDir +
                    TolHDMesh);
206     end
207 
208     % Delete HD nodes out of range
209     IdxOut = [IdxMinOut;IdxMaxOut]; % Index of HD nodes out of FEM
            model range
210     HDNodes(IdxOut,:) = [];
211 
212     % Offset for non-longitudinal directions (to get coordinates
            starting at 0)
213     switch Direction
214         case 'x'
215             % FEM nodes
216             MinValDir_B_FEM = min(FEMNodes(:,3));
217             MinValDir_C_FEM = min(FEMNodes(:,4));
218 
219             FEMNodes(:,3) = FEMNodes(:,3) - MinValDir_B_FEM;
220             FEMNodes(:,4) = FEMNodes(:,4) - MinValDir_C_FEM;
221 
222             % HD nodes
223             MinValDir_B_HD = min(HDNodes(:,3));
224             MinValDir_C_HD = min(HDNodes(:,4));
225 
226             HDNodes(:,3) = HDNodes(:,3) - MinValDir_B_HD;
227             HDNodes(:,4) = HDNodes(:,4) - MinValDir_C_HD;
228         case 'y'
229             % FEM nodes
230             MinValDir_B_FEM = min(FEMNodes(:,2));
```

```
231             MinValDir_C_FEM = min(FEMNodes(:,4));
232
233             FEMNodes(:,2) = FEMNodes(:,2) - MinValDir_B_FEM;
234             FEMNodes(:,4) = FEMNodes(:,4) - MinValDir_C_FEM;
235
236             % HD nodes
237             MinValDir_B_HD = min(HDNodes(:,2));
238             MinValDir_C_HD = min(HDNodes(:,4));
239
240             HDNodes(:,2) = HDNodes(:,2) - MinValDir_B_HD;
241             HDNodes(:,4) = HDNodes(:,4) - MinValDir_C_HD;
242         case 'z'
243             % FEM nodes
244             MinValDir_B_FEM = min(FEMNodes(:,2));
245             MinValDir_C_FEM = min(FEMNodes(:,3));
246
247             FEMNodes(:,2) = FEMNodes(:,2) - MinValDir_B_FEM;
248             FEMNodes(:,3) = FEMNodes(:,3) - MinValDir_C_FEM;
249
250             % HD nodes
251             MinValDir_B_HD = min(HDNodes(:,2));
252             MinValDir_C_HD = min(HDNodes(:,3));
253
254             HDNodes(:,2) = HDNodes(:,2) - MinValDir_B_HD;
255             HDNodes(:,3) = HDNodes(:,3) - MinValDir_C_HD;
256     end
257
258     % Delete FEM nodes above max value of nodes from HD pressure nodes
            (in transversal direction)
259     switch Direction_transv_vertical
260         case 'x'
261             MaxValDir_A = max(HDNodes(:,2));
262             [IdxMaxOut,~] = find(FEMNodes(:,2) > MaxValDir_A +
                    TolHDMesh);
263             FEMNodes(IdxMaxOut,:) = [];
264         case 'y'
265             MaxValDir_A = max(HDNodes(:,3));
266             [IdxMaxOut,~] = find(FEMNodes(:,3) > MaxValDir_A +
                    TolHDMesh);
267             FEMNodes(IdxMaxOut,:) = [];
268         case 'z'
269             MaxValDir_A = max(HDNodes(:,4));
270             [IdxMaxOut,~] = find(FEMNodes(:,4) > MaxValDir_A +
```

```
                TolHDMesh);
271             FEMNodes(IdxMaxOut,:) = [];
272     end
273
274     FEMNodes = sortrows(FEMNodes,1);
275
276     if size(FEMNodes,1) ~= size(HDNodes,1)
277         fprintf(2,"Non-coincident meshes are not supported \n");
278         return
279     end
280
281     % Delete FEM model elements above draft
282     Aux = str2double(FEMElements(:,3:end));
283     ElIdxKeep = [];
284     for i = 1:size(Aux,1)
285         if sum(ismember(Aux(i,:),FEMNodes(:,1))) == size(Aux(i,:),2)
286             ElIdxKeep = [ElIdxKeep;i];
287         end
288     end
289     FEMElements = FEMElements(ElIdxKeep,:);
290
291     % Find equivalent nodes
292     EquivNodes = FEMNodes(:,1);
293     for i=1:size(FEMNodes,1)
294
295         % Get coordinates
296         Xcoord = FEMNodes(i,2);
297         Ycoord = FEMNodes(i,3);
298         Zcoord = FEMNodes(i,4);
299
300         % Get nodes from FEM model in the same X, Y and Z range from HD
                mesh
301         EquivRangeX = [];
302         EquivRangeY = [];
303         EquivRangeZ = [];
304         EquivRangeX = find((HDNodes(:,2) - TolHDMesh) <= Xcoord & (
              HDNodes(:,2) + TolHDMesh) >= Xcoord);
305         EquivRangeY = find((HDNodes(:,3) - TolHDMesh) <= Ycoord & (
              HDNodes(:,3) + TolHDMesh) >= Ycoord);
306         EquivRangeZ = find((HDNodes(:,4) - TolHDMesh) <= Zcoord & (
              HDNodes(:,4) + TolHDMesh) >= Zcoord);
307
308         % Get index of equivalent element from FEM mesh
```

```
309         EquivRange = intersect(EquivRangeX,EquivRangeY);
310         EquivRange = intersect(EquivRange,EquivRangeZ);
311 
312         if size(EquivRange,1) ~= 1
313             fprintf(2,'More than 1 equivalent element was found \nCheck
                    the meshes or improve tolerance parameter \n');
314             return
315         end
316 
317         EquivNodes(i,2) = EquivRange;
318     end
319 
320     % Return node coordinates with offset for non-longitudinal
          directions
321     switch Direction
322         case 'x'
323             % FEM nodes
324             FEMNodes(:,3) = FEMNodes(:,3) + MinValDir_B_FEM;
325             FEMNodes(:,4) = FEMNodes(:,4) + MinValDir_C_FEM;
326 
327             % HD nodes
328             HDNodes(:,3) = HDNodes(:,3) + MinValDir_B_HD;
329             HDNodes(:,4) = HDNodes(:,4) + MinValDir_C_HD;
330         case 'y'
331             % FEM nodes
332             FEMNodes(:,2) = FEMNodes(:,2) + MinValDir_B_FEM;
333             FEMNodes(:,4) = FEMNodes(:,4) + MinValDir_C_FEM;
334 
335             % HD nodes
336             HDNodes(:,2) = HDNodes(:,2) + MinValDir_B_HD;
337             HDNodes(:,4) = HDNodes(:,4) + MinValDir_C_HD;
338         case 'z'
339             % FEM nodes
340             FEMNodes(:,2) = FEMNodes(:,2) + MinValDir_B_FEM;
341             FEMNodes(:,3) = FEMNodes(:,3) - MinValDir_C_FEM;
342 
343             % HD nodes
344             HDNodes(:,2) = HDNodes(:,2) + MinValDir_B_HD;
345             HDNodes(:,3) = HDNodes(:,3) - MinValDir_C_HD;
346     end
347 
348 %% Procedure to find equivalent elements
349 elseif PressureEntity == 2    % Pressure entity = ELEMENTS
```

```
350
351     % Add Element ID to HD elements
352     HDElements(:,2:5) = HDElements;
353     HDElements(1:end,1) = 1:size(HDElements,1);
354
355     % Add Node ID to HD nodes
356     HDNodes(:,2:4) = HDNodes;
357     HDNodes(1:end,1) = 1:size(HDNodes,1);
358
359     % Round values to 6 decimals
360     FEMNodes(:,2:4) = round(FEMNodes(:,2:4),6);
361     HDNodes(:,2:4) = round(HDNodes(:,2:4),6);
362
363     %  Get min/max value for longitudinal direction of FEM model and
            delete any HD node out of range
364     switch Direction
365         case 'x'
366             MinValLongDir = min(FEMNodes(:,2));
367             MaxValLongDir = max(FEMNodes(:,2));
368
369             [IdxMinOut,~] = find(HDNodes(:,2) < MinValLongDir -
                    TolHDMesh);
370             [IdxMaxOut,~] = find(HDNodes(:,2) > MaxValLongDir +
                    TolHDMesh);
371         case 'y'
372             MinValLongDir = min(FEMNodes(:,3));
373             MaxValLongDir = max(FEMNodes(:,3));
374
375             [IdxMinOut,~] = find(HDNodes(:,3) < MinValLongDir -
                    TolHDMesh);
376             [IdxMaxOut,~] = find(HDNodes(:,3) > MaxValLongDir +
                    TolHDMesh);
377         case 'z'
378             MinValLongDir = min(FEMNodes(:,4));
379             MaxValLongDir = max(FEMNodes(:,4));
380
381             [IdxMinOut,~] = find(HDNodes(:,4) < MinValLongDir -
                    TolHDMesh);
382             [IdxMaxOut,~] = find(HDNodes(:,4) > MaxValLongDir +
                    TolHDMesh);
383     end
384
385     % Delete HD nodes out of range
```

```
386     IdxOut = [IdxMinOut;IdxMaxOut]; % Index of HD nodes out of FEM
            model range
387     HDNodes(IdxOut,:) = [];
388
389     % Delete FEM elements above max value of nodes from HD pressure
            nodes (in transversal direction)
390     NodesCheck = [];
391     for j=2:size(HDElements,2)
392         NodesCheck = [NodesCheck,ismember(HDElements(:,j),HDNodes(:,1))
                ];
393     end
394     NodesCheck = sum(NodesCheck,2);
395
396     IdxOut = [];
397     for i=1:size(NodesCheck,1)
398         if NodesCheck(i) ~= 4
399             IdxOut = [IdxOut;i];
400         end
401     end
402
403     HDElements(IdxOut,:) = [];
404
405     % Offset for non-longitudinal directions (to get coordinates
            starting at 0)
406     switch Direction
407         case 'x'
408             % FEM nodes
409             MinValDir_B_FEM = min(FEMNodes(:,3));
410             MinValDir_C_FEM = min(FEMNodes(:,4));
411
412             FEMNodes(:,3) = FEMNodes(:,3) - MinValDir_B_FEM;
413             FEMNodes(:,4) = FEMNodes(:,4) - MinValDir_C_FEM;
414
415             % HD nodes
416             MinValDir_B_HD = min(HDNodes(:,3));
417             MinValDir_C_HD = min(HDNodes(:,4));
418
419             HDNodes(:,3) = HDNodes(:,3) - MinValDir_B_HD;
420             HDNodes(:,4) = HDNodes(:,4) - MinValDir_C_HD;
421         case 'y'
422             % FEM nodes
423             MinValDir_B_FEM = min(FEMNodes(:,2));
424             MinValDir_C_FEM = min(FEMNodes(:,4));
```

```
425
426             FEMNodes(:,2) = FEMNodes(:,2) - MinValDir_B_FEM;
427             FEMNodes(:,4) = FEMNodes(:,4) - MinValDir_C_FEM;
428
429             % HD nodes
430             MinValDir_B_HD = min(HDNodes(:,2));
431             MinValDir_C_HD = min(HDNodes(:,4));
432
433             HDNodes(:,2) = HDNodes(:,2) - MinValDir_B_HD;
434             HDNodes(:,4) = HDNodes(:,4) - MinValDir_C_HD;
435         case 'z'
436             % FEM nodes
437             MinValDir_B_FEM = min(FEMNodes(:,2));
438             MinValDir_C_FEM = min(FEMNodes(:,3));
439
440             FEMNodes(:,2) = FEMNodes(:,2) - MinValDir_B_FEM;
441             FEMNodes(:,3) = FEMNodes(:,3) - MinValDir_C_FEM;
442
443             % HD nodes
444             MinValDir_B_HD = min(HDNodes(:,2));
445             MinValDir_C_HD = min(HDNodes(:,3));
446
447             HDNodes(:,2) = HDNodes(:,2) - MinValDir_B_HD;
448             HDNodes(:,3) = HDNodes(:,3) - MinValDir_C_HD;
449     end
450
451     % Delete FEM nodes above max value of nodes from HD pressure nodes
        (in transversal direction)
452     switch Direction_transv_vertical
453         case 'x'
454             MaxValDir_A = max(HDNodes(:,2));
455             [IdxMaxOut,~] = find(FEMNodes(:,2) > MaxValDir_A +
                    TolHDMesh);
456             FEMNodes(IdxMaxOut,:) = [];
457         case 'y'
458             MaxValDir_A = max(HDNodes(:,3));
459             [IdxMaxOut,~] = find(FEMNodes(:,3) > MaxValDir_A +
                    TolHDMesh);
460             FEMNodes(IdxMaxOut,:) = [];
461         case 'z'
462             MaxValDir_A = max(HDNodes(:,4));
463             [IdxMaxOut,~] = find(FEMNodes(:,4) > MaxValDir_A +
                    TolHDMesh);
```

```
464             FEMNodes(IdxMaxOut,:) = [];
465     end
466
467     FEMNodes = sortrows(FEMNodes,1);
468
469     % Find equivalent nodes
470     EquivNodes(:,2) = HDNodes(:,1);
471
472     for i=1:size(HDNodes,1)
473
474         % Get coordinates
475         Xcoord = HDNodes(i,2);
476         Ycoord = HDNodes(i,3);
477         Zcoord = HDNodes(i,4);
478
479         % Get nodes from FEM model in the same X, Y and Z range from HD
                mesh
480         EquivRangeX = [];
481         EquivRangeY = [];
482         EquivRangeZ = [];
483         EquivRangeX = find((FEMNodes(:,2) - TolHDMesh) <= Xcoord & (
                FEMNodes(:,2) + TolHDMesh) >= Xcoord);
484         EquivRangeY = find((FEMNodes(:,3) - TolHDMesh) <= Ycoord & (
                FEMNodes(:,3) + TolHDMesh) >= Ycoord);
485         EquivRangeZ = find((FEMNodes(:,4) - TolHDMesh) <= Zcoord & (
                FEMNodes(:,4) + TolHDMesh) >= Zcoord);
486
487         % Get index of equivalent element from FEM mesh
488         EquivRange = intersect(EquivRangeX,EquivRangeY);
489         EquivRange = intersect(EquivRange,EquivRangeZ);
490
491         if isempty(EquivRange)
492             EquivNodes(i,1) = 0;    % Nodes from HD mesh without an
                    equivalent has ID = 0
493         else
494             EquivNodes(i,1) = FEMNodes(EquivRange,1);
495         end
496     end
497
498     % Find equivalent elements
499     EquivElements(:,2) = HDElements(:,1);
500
501     % Find FEM nodes converted from HD nodes used in HD mesh
```

```
502     FEMNodesConv = [];
503     for i=1:size(EquivElements,1)
504         HDCurrentNodes = sort(HDElements(i,2:5));
505         [idx,~] = find(HDCurrentNodes == EquivNodes(:,2));
506         FEMNodesConv = [FEMNodesConv;EquivNodes(idx,1)'];
507     end
508 
509     % Sort FEM nodes converted and FEM nodes from FEM elements
510     FEMNodesConv = sort(FEMNodesConv,2);
511 
512     FEMNodesSorted = sort(str2double(FEMElements(:,3:end)),2);
513     FEMNodesSorted(:,2:5) = FEMNodesSorted;
514     FEMNodesSorted(:,1) = str2double(FEMElements(:,1));
515 
516     % Find equivalent elements
517     for i=1:size(FEMNodesConv,1)
518         PossibleElementsIdx = [];
519         for j=2:5
520             [idx,~] = find(FEMNodesConv(i,1) == FEMNodesSorted(:,j));
521             PossibleElementsIdx = [PossibleElementsIdx;idx];
522         end
523 
524         PossibleElements = FEMNodesSorted(PossibleElementsIdx,:);
525         CheckElement = sum(ismember(string(PossibleElements(:,2:5)),
                string(FEMNodesConv(i,:))),2);
526         ElementIdx = find(4 == CheckElement);
527 
528         EquivElements(i,1) = PossibleElements(ElementIdx,1);
529     end
530 
531     EquivElements = sortrows(EquivElements,1);
532 
533     % Return node coordinates with offset for non-longitudinal
            directions
534     switch Direction
535         case 'x'
536             % FEM nodes
537             FEMNodes(:,3) = FEMNodes(:,3) + MinValDir_B_FEM;
538             FEMNodes(:,4) = FEMNodes(:,4) + MinValDir_C_FEM;
539 
540             % HD nodes
541             HDNodes(:,3) = HDNodes(:,3) + MinValDir_B_HD;
542             HDNodes(:,4) = HDNodes(:,4) + MinValDir_C_HD;
```

```
543         case 'y'
544             % FEM nodes
545             FEMNodes(:,2) = FEMNodes(:,2) + MinValDir_B_FEM;
546             FEMNodes(:,4) = FEMNodes(:,4) + MinValDir_C_FEM;
547 
548             % HD nodes
549             HDNodes(:,2) = HDNodes(:,2) + MinValDir_B_HD;
550             HDNodes(:,4) = HDNodes(:,4) + MinValDir_C_HD;
551         case 'z'
552             % FEM nodes
553             FEMNodes(:,2) = FEMNodes(:,2) + MinValDir_B_FEM;
554             FEMNodes(:,3) = FEMNodes(:,3) - MinValDir_C_FEM;
555 
556             % HD nodes
557             HDNodes(:,2) = HDNodes(:,2) + MinValDir_B_HD;
558             HDNodes(:,3) = HDNodes(:,3) - MinValDir_C_HD;
559     end
560 
561 end
562 
563 %% Clear variables and save Equivalent Entity List
564 clearvars -except EquivNodes EquivElements FolderLock FEMElements
        FEMNodes HDNodes HDElements
565 save(strcat(FolderLock,'\Equivalent_Entity_List.mat'));  % Save
        Equivalent matrix
566 % load('Aux locked\PJ_Cilamce\Equivalent_Entity_List.mat')
567 
568 %%  End of code
569 fclose all;
570 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
571 fprintf('Elapsed time running Ini_A01_equiv.m is %s \n',ElapsedTime)
572 clearvars
573 return
```

C.1.10. Algoritmo Ini_A01_storage_levels_off

Algoritmo	Objetivo	Categoria
Ini_A01_storage_levels_off	Cria arquivo utilizado para a condição de carregamento dos tanques igual a 0	Auxiliar

```
%%  Ini_A01_storage_levels_off
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   Nov 22, 2021
%   This script/function reads Storage Levels file (StorageLevels.txt) from directory "A01 coupled systems\Storage Levels" and creates file "- StorageLevelsOFF.txt"
%  [pt-br] Este script/fun  o  l  o arquivo de dados de n vel dos tanques (StorageLevels.txt) do diret rio "A01 coupled systems\Storage Levels" e cria o arquivo "- StorageLevelsOFF.txt"

%% Notes:
%

%% Steps:
%  Load variables
%  Check if file exists and create if it doesn't
%  End of code

%% Load variables
fprintf(1,"Initializing Ini_A01_storage_levels_off \n")
tic

Ini_A00_initial_data
clearvars -except Var_Ini_A01_storage_levels_off


strcat(Var_Ini_A01_storage_levels_off{1},'\',
    Var_Ini_A01_storage_levels_off{2},'.',
    Var_Ini_A01_storage_levels_off{3})

%% Check if file "- StorageLevelsOFF.txt" exists and create if doesn't
% if ~exist(strcat(Var_Ini_A01_storage_levels_off{1},'\-
    StorageLevelsOFF.txt'), 'file')

    if exist(strcat(Var_Ini_A01_storage_levels_off{1},'\',
```

```
        Var_Ini_A01_storage_levels_off{2},'.',
        Var_Ini_A01_storage_levels_off{3}), 'file')
29
30          FileDataOpt = extractFileText(strcat(
                Var_Ini_A01_storage_levels_off{1},'\',
                Var_Ini_A01_storage_levels_off{2},'.',
                Var_Ini_A01_storage_levels_off{3}));                    %
                Get data from .txt file
31          FileDataOpt = regexprep(FileDataOpt,'\n','&%^');            %
                Replace break lines with codes
32          FileDataOpt = strip(split(FileDataOpt,'&%^'));              %
                Split characters and delete codes
33          FileDataOpt = string(FileDataOpt);                          %
                Convert to string
34          FileDataOpt = regexp(FileDataOpt,'\s','split');
35          FileDataOpt = vertcat(FileDataOpt{:});
36
37          FileDataOpt(:,3:end) = [];
38          FileDataOpt(4:end,2) = "0";
39
40          fid = fopen(strcat(Var_Ini_A01_storage_levels_off{1},'\-
                StorageLevelsOFF.txt'),'w');
41          fprintf(fid,'%s %s \n',FileDataOpt');
42          fclose(fid);
43
44          fprintf(2,'File "%s" was created.\n',strcat(
                Var_Ini_A01_storage_levels_off{1},'\- StorageLevelsOFF.txt'
                ))
45
46      else
47          fprintf(2,'File "%s" does not exist. \n',strcat(
                Var_Ini_A01_storage_levels_off{1},'\',
                Var_Ini_A01_storage_levels_off{2},'.',
                Var_Ini_A01_storage_levels_off{3}))
48          return
49      end
50
51  % else
52  %     fprintf(1,'File "%s" already exists.\n',strcat(
        Var_Ini_A01_storage_levels_off{1},'\- StorageLevelsOFF.txt'))
53  % end
54
55  %%  End of code
```

```
56 fclose all;
57 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
58 fprintf('Elapsed time running Ini_A01_storage_levels_off.m is %s \n',
       ElapsedTime)
59 return
```

C.1.11. Algoritmo Reset

Algoritmo	Objetivo	Categoria
Reset	Deletar todo o conteúdo dos diretórios gerados por Ini_A00_Folders	Opcional

```
%%  Reset
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   Jun 06, 2021
%   This script/function deletes all content from folders generated by Ini_A00_Folders.m

tic

% Check if script was requested by GUI (in Manual or Automatic control)
try
   if control == "Manual"
       fprintf("Control: GUI - %s \n",control)
   elseif control == "Automatic"
       fprintf("Control: GUI - %s \n",control)
   else
       control = "None";
       fprintf("Control: GUI - %s \n",control)
   end
catch
   control = "None";
   fprintf("Control: GUI - %s \n",control)
end

% Load variables by running without GUI
if control == "None"
    Ini_A00_initial_data
    clearvars -except Var_Reset

    FolderUpdates = Var_Reset{1};
    MCFile = Var_Reset{2};
    PathApdlPostFile = Var_Reset{3};
    FolderFemCases = Var_Reset{4};
    FolderLock = Var_Reset{5};
    FolderPost = Var_Reset{6};
end

```

```
answer = questdlg('Data and files will be deleted ', ...
        'Reset', ...
        'Yes','No','No');

if answer == 'Yes'
    try
        delete(strcat(FolderUpdates,'\',MCFile,'.txt'))
        fprintf(2,'File "%s" was deleted.\n',strcat(FolderUpdates,'\',
           MCFile,'.txt'))
    catch
        % do nothing
    end

    try
        delete(strcat(FolderUpdates,'\',PathApdlPostFile))
        fprintf(2,'File "%s" was deleted.\n',strcat(FolderUpdates,'\',
           PathApdlPostFile))
    catch
        % do nothing
    end

    try
        delete(strcat(FolderFemCases,'\*'))
        fprintf(2,'Content from folder "%s" was deleted.\n',
           FolderFemCases)
    catch
        % do nothing
    end

    try
        delete(strcat(FolderPost,'\*'))
        fprintf(2,'Content from folder "%s" was deleted.\n',FolderPost)
    catch
        % do nothing
    end

    listing = dir(FolderLock);
    listing = struct2cell(listing);
    for i=1:size(listing,2)
        try
            filename = string(listing(1,i));
            if filename ~= "BodyInfo.mat" && filename ~= "
               Equivalent_Entity_List.mat"
```

```
75                  delete(strcat(FolderLock,'\',filename))
76              end
77          catch
78              % do nothing
79          end
80      end
81
82  end
83
84  %%  End of code
85  fclose all;
86  ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
87  fprintf('Elapsed time running Res_All_Data.m is %s \n',ElapsedTime)
88  return
```

Algoritmo	Objetivo	Categoria
Ini_A00_initial_data	Inicializa variáveis (apenas para execução manual do Gêmeo Digital)	Opcional

```
%%  A00_Initial_Data
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   May 10, 2021
%   This script/function initializes all variables needed for all scripts
%       [pt-br] Este script/fun  o inicializa todas as vari veis necess rias para todos os scripts

%% Notes:
%  Warning: On the MatLab Home tab, go to the Environment section, click Preferences. Select MATLAB > General > Java Heap Memory > set to max.
%  Warning: Everything that is an user input is under "%   User input", variables that are not an user input can be changed (but it is not recommended to do that)

%% Steps:
%  1. Input parameters related to generic FEM model (A-green)
%  2. Input parameters related to coupled systems (A-blue)
%  3. Input parameters related to models assembling (A-purple)
%  4. Input parameters related to solving process (B-yellow)
%  5. Input parameters related to post-processing (C-red)
%  6. Cells with variables needed by each script
%  End of code
fprintf(1,"Loading all variables \n")









% % % %% FE_model_Cilamce
% % % %  Input for A01_coupled_systems
% % % FileOpt = 'Options';
```

```
    % Filename for options file
32 % % % FileWaveReg = 'A01 coupled systems\FE_model_Cilamce\Wave\
    RegularWave.mat';                    % Filename for regular wave
    file
33 % % % FileWaveIrr = 'A01 coupled systems\FE_model_Cilamce\Wave\
    IrregularWave.mat';                  % Filename for irregular wave
    file
34 % % % FileStor = 'A01 coupled systems\FE_model_Cilamce\Storage Levels\
    StorageLevels.txt';           % Filename for storage levels file
35 % % % FileInsp = '';

    % Filename for inspection file of plates (with only 1 parameter)
36 % % % FolderCS = 'A01 coupled systems\FE_model_Cilamce';
                                                % Folder for coupled
    systems
37 % % % FolderWave = 'A01 coupled systems\FE_model_Cilamce\Wave';
                                          % Folder for wave files
38 % % % FolderStor = 'A01 coupled systems\FE_model_Cilamce\Storage Levels
    ';                              % Folder for storage levels
39 % % % FolderInsp = 'A01 coupled systems\FE_model_Cilamce\Inspection';
                                     % Folder for inspection
40 % % % FolderSEG = 'A01 coupled systems\FE_model_Cilamce\Standard Earth
    Gravity';                     % Folder for Standard Earth Gravity
41 % % % DraftModel = 8.2320;

    % Cut plane related to Z coordinate zero
42 % % % DivTimeIrrWave = 1200;
                                                                      %
    Time for division of irregular wave for analysis
43 % % % cgloc_X = 0.0;

    % CG: X coordinate from FEM model
44 % % % cgloc_Y = 0.0;

    % CG: Y coordinate from FEM model
45 % % % cgloc_Z = 2.4580;

    % CG: Z coordinate from FEM model
46 % % % floc_X = 60.0;

    % X coordinate for remote force from FEM model
47 % % % floc_Y = 0.0;
```

```
    % Y coordinate for remote force from FEM model
48 % % % floc_Z = 2.458;

    % Z coordinate for remote force from FEM model
49 % % % HTank = 16.0;

    % Maximum height value for capacity of the cargo tanks
50 % % % FolderUpdates = 'A01 coupled systems\FE_model_Cilamce\- Update';
                                % Folder for *.txt file
51 % % % MCFile = 'MCContent';
                                                                    %
     File for definition of multiple cases (automatically generated)
52 % % % Direction_transv_vertical = 'z';
                                                                %
    Transversal (vertical) direction of model: x, y or z /  / This
    variable is locked after 1st running
53 % % %
54 % % % Var_A01_coupled_systems = {FileOpt;FolderCS;FileWaveReg;
    FileWaveIrr;FolderWave;FileStor;FolderStor;FileInsp;FolderSEG;
    DraftModel;DivTimeIrrWave;cgloc_X;cgloc_Y;cgloc_Z;floc_X;floc_Y;
    floc_Z;HTank;FolderUpdates;MCFile;Direction_transv_vertical};
55 % % %
56 % % % %  Input for A03_assemble
57 % % % FolderLock = 'Aux locked\FE_model_Cilamce';
                                                    % Folder to lock
    variables after first time running scripts
58 % % % PathApdlPostFileOri = 'A01 coupled systems\FE_model_Cilamce\-
    Update\APDL_post_origin.txt';   % Path for *.txt file
59 % % % PathApdlPostFile = 'APDL_post';
                                                        % Path for
    *.txt file
60 % % % PathFemFileCodes = 'A02 dat base\FE_model_Cilamce\FE_model_codes.
    dat';                               % Path for generic FEM model
    file
61 % % % PathFemFileInspection = 'A02 dat base\FE_model_Cilamce\
    FE_model_codes_inspection.dat';                 % Path for generic
     FEM model file (inspection)
62 % % % FolderFemCases = 'A03 assemble\FE_model_Cilamce';
                                            % Folder for FEM cases
63 % % % ResInfo = '- Results_info';
                                                                  %
    Filename to save results information (automatically generated)
```

```
64 % % % FolderHDProcessed = 'A01 coupled systems\FE_model_Cilamce\Wave\HD
       ';                              % Path for hydrodynamic analysis
       folder
65 % % % PathHDProcessed = 'A01 coupled systems\FE_model_Cilamce\Wave\HD\
       FE_model_Hull_Pressure.mat';        % Path for hydrodynamic
       analysis file
66 % % % MinPha = 0;

       % Minimum value for phase angle range
67 % % % MaxPha = 360;

       % Maximum value for phase angle range
68 % % % BrkPha = 10;

       % Break for phase angle range
69 % % % Unit = 1000;

       % Unit multiplies used for pressure: 1 = Pa ; 1000 = KPa ...
70 % % %
71 % % % Var_A03_assemble = {FolderLock;PathApdlPostFileOri;
       PathApdlPostFile;FolderCS;FileOpt;FolderInsp;PathFemFileCodes;
       FolderUpdates;MCFile;FolderFemCases;ResInfo;FolderHDProcessed;
       PathHDProcessed;MinPha;BrkPha;MaxPha;Unit;PathFemFileInspection};
72 % % %
73 % % % % Input B04_solver
74 % % % PathAnsysSolver = '"C:\Program Files\ANSYS Inc\v202\ansys\bin\
       winx64\ANSYS202.exe"';    % ANSYS Mechanical solver path
75 % % % np = '4';

       % Number of processors
76 % % % FileNameBat = '- Run_Cases';
                                                                  % Batch
       file name
77 % % %
78 % % % Var_B04_solver = {FolderFemCases;PathAnsysSolver;np;FileNameBat};
79 % % %
80 % % % % Input for C05_post_results
81 % % % FolderPost = 'C05 post\FE_model_Cilamce';
                                                      % Folder for Post-
       processed files
82 % % % ResItem = ["Displacement";"Stress";"Elastic_Strain";"
       Plastic_Strain";"Total_Strain";"Principal_Stress"]; % Possible
       result status
```

```
83 % % % ResPrint = [" PRINT U";" PRINT S";" PRINT EPEL";" PRINT EPPL";" PRINT EPTO";" PRINT S"]; % Expression to find according to possible results
84 % % % ResComp = [4;6;6;6;6;5];                                                % Total of components for each result
85 % % % ResCompView = ["UX","UY","UZ","USUM","","";"SX","SY","SZ","SXY","SYZ","SXZ";"EPELX","EPELY","EPELZ","EPELXY","EPELYZ","EPELXZ";"EPPLX","EPPLY","EPPLZ","EPPLXY","EPPLYZ","EPPLXZ";"EPTOX","EPTOY","EPTOZ","EPTOXY","EPTOYZ","EPTOXZ";"S1","S2","S3","SINT","SEQV",""]; % Component views
86 % % % AlertMatrix = ["Result","Component","Logical",1,2;"Principal_Stress",1,">",165E+06,180E+06]; % AlertMatrix = [];
87 % % % PlotAnsGmsh = false;    % Plot simulation results from ANSYS in Gmsh
88 % % % PlotHotspots = true;    % Plot hotspots in Gmsh
89 % % % PlotAlert = true;       % Plot bodies in alert in Gmsh
90 % % % ResGmsh = [PlotAnsGmsh;PlotHotspots;PlotAlert];
91 % % %
92 % % % UGmsh = [0,0,0,1];          % Plot Displacements on Gmsh [UX,UY,UZ,USUM]
93 % % % SGmsh = [0,0,0,0,0,0];      % Plot Stress components on Gmsh [SX,SY,SZ,SXY,SYZ,SXZ]
94 % % % EPELGmsh = [0,0,0,0,0,0];   % Plot Elastic Strain Components on Gmsh [EPELX,EPELY,EPELZ,EPELXY,EPELYZ,EPELXZ]
95 % % % EPPLGmsh = [0,0,0,0,0,0];   % Plot Plastic Strain Components on Gmsh [EPPLX,EPPLY,EPPLZ,EPPLXY,EPPLYZ,EPPLXZ]
96 % % % EPTOGmsh = [0,0,0,0,0,0];   % Plot Total Mechanical Strain on Gmsh [EPTOX,EPTOY,EPTOZ,EPTOXY,EPTOYZ,EPTOXZ]
97 % % % PRINGmsh = [1,0,1,0,0];     % Plot Principal Stress Components on Gmsh [S1,S2,S3,SINT,SEQV]
98 % % % PrintGmsh = {UGmsh;SGmsh;EPELGmsh;EPPLGmsh;EPTOGmsh;PRINGmsh}; % All components to be printed on Gmsh / Only if PlotAnsGmsh = true / (IT MUST NOT BE CHANGED)
99 % % %
100 % % %
101 % % % Var_C05_post_results = {FolderLock;FolderPost;FolderFemCases;ResInfo;ResItem;ResPrint;ResComp;ResCompView;AlertMatrix;ResGmsh;PrintGmsh};
102 % % %
103 % % % % Input for Ini_A01_equiv
104 % % % FEMNodesFile = 'A01 coupled systems\FE_model_Cilamce\Wave\HD\FE_model_nodes_FEM.txt';
```

```
105 % % % HDNodesFile = 'A01 coupled systems\FE_model_Cilamce\Wave\HD\
    FE_model_nodes_HD.txt';
106 % % % FEMElementsFile = 'A01 coupled systems\FE_model_Cilamce\Wave\HD\
    FE_model_elements_FEM.txt';
107 % % % HDElementsFile = 'A01 coupled systems\FE_model_Cilamce\Wave\HD\
    FE_model_elements_HD.txt';
108 % % % Direction = 'x';
109 % % % TolHDMesh = 0.090;
110 % % %
111 % % % Var_Ini_A01_equiv = {FolderLock;FolderCS;FileOpt;FEMNodesFile;
    HDNodesFile;FEMElementsFile;HDElementsFile;Direction;
    Direction_transv_vertical;TolHDMesh};
112 % % %
113 % % % % Input for Ini_A00_post_bodies
114 % % % MinRange = -37.5;         % Minimum value for range of results
    related to longitudinal direction of the model / This variable is
    locked after 1st running
115 % % % MaxRange =  37.5;         % Maximum value for range of results
    related to longitudinal direction of the model / This variable is
    locked after 1st running
116 % % % TolRange = 0.09;            % Tolerance value for range of
    results / This variable is locked after 1st running
117 % % % PathFemFile = 'A02 dat base\FE_model_Cilamce\FE_model.dat';
118 % % %
119 % % % Var_Ini_A00_post_bodies = {FolderLock;Direction;MinRange;MaxRange
    ;TolRange;PathFemFile};
120 % % %
121 % % % % Input for Reset
122 % % %
123 % % % Var_Reset = {FolderUpdates;MCFile;PathApdlPostFile;FolderFemCases
    ;FolderLock;FolderPost};
124 % % %
125 % % % % Input for Var_Ini_A00_get_parameters
126 % % % PathFemFile = "A02 dat base\FE_model_Cilamce\FE_model.dat";
127 % % %
128 % % % Var_Ini_A00_get_parameters = {PathFemFile;FolderCS;FolderLock;
    FileInsp};
129 % % %
130 % % %
131 % % % clearvars -except Var_A01_coupled_systems Var_A03_assemble
    Var_B04_solver Var_C05_post_results Var_Ini_A00_Folders
    Var_Ini_A01_storage_levels_off Var_Ini_A00_get_parameters
    Var_Ini_A00_post_bodies Var_Ini_A01_equiv Var_Ret_A02_AQWA_process
```

```
    Var_Opt_B05_post_statistics Var_Opt_C06_plot_bodies
    Var_Res_All_Data Var_Res_Model_Data
% % % fprintf(1,"All variables loaded from Ini_A00_initial_data.m \n")
% % %
% % % %%  End of code
% % % fclose all;
% % % % ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
% % % % fprintf('Elapsed time running A00_Initial_Data.m is %s \n',
    ElapsedTime)
% % % return
% % %
% % %
```

```
%% FE_model
%  Input for A01_coupled_systems
FileOpt = 'Options';

    % Filename for options file

FileWaveReg = 'A01 coupled systems\FE_model\Wave\RegularWave.mat';
                              % Filename for regular wave file
FileWaveIrr = 'A01 coupled systems\FE_model\Wave\IrregularWave.mat';
                            % Filename for irregular wave file
FileStor = 'A01 coupled systems\FE_model\Storage Levels\StorageLevels.
    txt';                   % Filename for storage levels file
FileInsp = 'A01 coupled systems\FE_model\Inspection\FE_model_shell.txt'
    ;

    % Filename for inspection file of plates (with only 1 parameter)
```

```
164 FolderCS = 'A01 coupled systems\FE_model';
                                                    % Folder for
    coupled systems
165 FolderWave = 'A01 coupled systems\FE_model\Wave';
                                                % Folder for wave files
166 FolderStor = 'A01 coupled systems\FE_model\Storage Levels';
                                        % Folder for storage levels
167 FolderInsp = 'A01 coupled systems\FE_model\Inspection';
                                            % Folder for inspection
168 FolderSEG = 'A01 coupled systems\FE_model\Standard Earth Gravity';
                                % Folder for Standard Earth Gravity
169 DraftModel = 0.0;

    % Cut plane related to Z coordinate zero
170 DivTimeIrrWave = 1200;
                                                                      %
    Time for division of irregular wave for analysis
171 cgloc_X = 2.32;

    % CG: X coordinate from FEM model
172 cgloc_Y = 0.0;

    % CG: Y coordinate from FEM model
173 cgloc_Z = 14.27;

    % CG: Z coordinate from FEM model
174 floc_X = -122.625;

    % X coordinate for remote force from FEM model
175 floc_Y = 0.0;

    % Y coordinate for remote force from FEM model
176 floc_Z = 14.27;

    % Z coordinate for remote force from FEM model
177 HTank = 29.5;

    % Maximum height value for capacity of the cargo tanks
178 FolderUpdates = 'A01 coupled systems\FE_model\- Update';
                                            % Folder for *.txt file
179 MCFile = 'MCContent';
                                                                    %
     File for definition of multiple cases (automatically generated)
```

```
180 Direction_transv_vertical = 'z';
                                                                %
    Transversal (vertical) direction of model: x, y or z /  / This
    variable is locked after 1st running
181
182 Var_A01_coupled_systems = {FileOpt;FolderCS;FileWaveReg;FileWaveIrr;
    FolderWave;FileStor;FolderStor;FileInsp;FolderSEG;DraftModel;
    DivTimeIrrWave;cgloc_X;cgloc_Y;cgloc_Z;floc_X;floc_Y;floc_Z;HTank;
    FolderUpdates;MCFile;Direction_transv_vertical};
183
184 %  Input for A03_assemble
185 FolderLock = 'Aux locked\FE_model';
                                                          % Folder to
     lock variables after first time running scripts
186 PathApdlPostFileOri = 'A01 coupled systems\FE_model\- Update\
    APDL_post_origin.txt';          % Path for *.txt file
187 PathApdlPostFile = 'APDL_post';
                                                          % Path for
    *.txt file
188 PathFemFileCodes = 'A02 dat base\FE_model\FE_model_codes.dat';
                                      % Path for generic FEM model file
189 PathFemFileInspection = 'A02 dat base\FE_model\
    FE_model_codes_inspection.dat';                  % Path for generic
     FEM model file (inspection)
190 FolderFemCases = 'A03 assemble\FE_model';
                                                       % Folder for FEM
    cases
191 ResInfo = '- Results_info';
                                                                  %
    Filename to save results information (automatically generated)
192 FolderHDProcessed = 'A01 coupled systems\FE_model\Wave\HD';
                                    % Path for hydrodynamic analysis
    folder
193 PathHDProcessed = 'A01 coupled systems\FE_model\Wave\HD\
    FE_model_Hull_Pressure.mat';            % Path for hydrodynamic
    analysis file
194 MinPha = 0;

    % Minimum value for phase angle range
195 MaxPha = 360;

    % Maximum value for phase angle range
196 BrkPha = 10;
```

```
    % Break for phase angle range
197 Unit = 1000;

    % Unit multiplies used for pressure: 1 = Pa ; 1000 = KPa ...
198
199 Var_A03_assemble = {FolderLock;PathApdlPostFileOri;PathApdlPostFile;
    FolderCS;FileOpt;FolderInsp;PathFemFileCodes;FolderUpdates;MCFile;
    FolderFemCases;ResInfo;FolderHDProcessed;PathHDProcessed;MinPha;
    BrkPha;MaxPha;Unit;PathFemFileInspection};
200
201 % Input B04_solver
202 PathAnsysSolver = '"C:\Program Files\ANSYS Inc\v202\ansys\bin\winx64\
    ANSYS202.exe"';    % ANSYS Mechanical solver path
203 np = '4';

    % Number of processors
204 FileNameBat = '- Run_Cases';
                                                                % Batch
    file name
205
206 Var_B04_solver = {FolderFemCases;PathAnsysSolver;np;FileNameBat};
207
208 % Input for C05_post_results
209 FolderPost = 'C05 post\FE_model';
                                                % Folder for Post-
    processed files
210 ResItem = ["Displacement";"Stress";"Elastic_Strain";"Plastic_Strain";"
    Total_Strain";"Principal_Stress"]; % Possible result status
211 ResPrint = [" PRINT U";" PRINT S";" PRINT EPEL";" PRINT EPPL";" PRINT
    EPTO";" PRINT S"]; % Expression to find according to possible
    results
212 ResComp = [4;6;6;6;6;5];
                                                                    %
    Total of components for each result
213 ResCompView = ["UX","UY","UZ","USUM","","";"SX","SY","SZ","SXY","SYZ","
    SXZ";"EPELX","EPELY","EPELZ","EPELXY","EPELYZ","EPELXZ";"EPPLX","
    EPPLY","EPPLZ","EPPLXY","EPPLYZ","EPPLXZ";"EPTOX","EPTOY","EPTOZ","
    EPTOXY","EPTOYZ","EPTOXZ";"S1","S2","S3","SINT","SEQV",""]; %
    Component views
214 AlertMatrix = ["Result","Component","Logical",1,2;"Principal_Stress
    ",1,">",250E+06,380E+06]; % AlertMatrix = [];
215 PlotAnsGmsh = true;
```

```
PlotHotspots = true;
PlotAlert = true;
ResGmsh = [PlotAnsGmsh;PlotHotspots;PlotAlert];

UGmsh = [0,0,0,1];          % Plot Displacements on Gmsh [UX,UY,UZ,USUM]
SGmsh = [0,0,0,0,0,0];      % Plot Stress components on Gmsh [SX,SY,SZ,SXY,SYZ,SXZ]
EPELGmsh = [0,0,0,0,0,0];   % Plot Elastic Strain Components on Gmsh [EPELX,EPELY,EPELZ,EPELXY,EPELYZ,EPELXZ]
EPPLGmsh = [0,0,0,0,0,0];   % Plot Plastic Strain Components on Gmsh [EPPLX,EPPLY,EPPLZ,EPPLXY,EPPLYZ,EPPLXZ]
EPTOGmsh = [0,0,0,0,0,0];   % Plot Total Mechanical Strain on Gmsh [EPTOX,EPTOY,EPTOZ,EPTOXY,EPTOYZ,EPTOXZ]
PRINGmsh = [1,0,1,0,0];     % Plot Principal Stress Components on Gmsh [S1,S2,S3,SINT,SEQV]
PrintGmsh = {UGmsh;SGmsh;EPELGmsh;EPPLGmsh;EPTOGmsh;PRINGmsh}; % All components to be printed on Gmsh / Only if PlotAnsGmsh = true / (IT MUST NOT BE CHANGED)

Var_C05_post_results = {FolderLock;FolderPost;FolderFemCases;ResInfo;ResItem;ResPrint;ResComp;ResCompView;AlertMatrix;ResGmsh;PrintGmsh};

% Input for Ini_A01_equiv
FEMNodesFile = 'A01 coupled systems\FE_model\Wave\HD\FE_model_nodes_FEM.txt';
HDNodesFile = 'A01 coupled systems\FE_model\Wave\HD\FE_model_nodes_HD.txt';
FEMElementsFile = 'A01 coupled systems\FE_model\Wave\HD\FE_model_elements_FEM.txt';
HDElementsFile = '';
Direction = 'x';
TolHDMesh = 0.0090;

Var_Ini_A01_equiv = {FolderLock;FolderCS;FileOpt;FEMNodesFile;HDNodesFile;FEMElementsFile;HDElementsFile;Direction;Direction_transv_vertical;TolHDMesh};

% Input for Ini_A00_post_bodies
MinRange = -24.525;         % Minimum value for range of results related to longitudinal direction of the model / This variable is locked after 1st running
```

```
242 MaxRange =  24.525;          % Maximum value for range of results
    related to longitudinal direction of the model / This variable is
    locked after 1st running
243 TolRange = 0.09;             % Tolerance value for range of results /
    This variable is locked after 1st running
244 PathFemFile = 'A02 dat base\FE_model\FE_model_origin.dat';
245
246 Var_Ini_A00_post_bodies = {FolderLock;Direction;MinRange;MaxRange;
    TolRange;PathFemFile};
247
248 % Input for Reset
249
250 Var_Reset = {FolderUpdates;MCFile;PathApdlPostFile;FolderFemCases;
    FolderLock;FolderPost};
251
252 % Input for Var_Ini_A00_get_parameters
253 PathFemFile = "A02 dat base\FE_model\FE_model_origin.dat";
254
255 Var_Ini_A00_get_parameters = {PathFemFile;FolderCS;FolderLock;FileInsp
    };
256
257
258
259 clearvars -except Var_A01_coupled_systems Var_A03_assemble
    Var_B04_solver Var_C05_post_results Var_Ini_A00_Folders
    Var_Ini_A01_storage_levels_off Var_Ini_A00_get_parameters
    Var_Ini_A00_post_bodies Var_Ini_A01_equiv Var_Ret_A02_AQWA_process
    Var_Opt_B05_post_statistics Var_Opt_C06_plot_bodies Var_Reset
260
261 fprintf(1,"All variables loaded from Ini_A00_initial_data.m \n")
262
263 %%  End of code
264 fclose all;
265 % ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
266 % fprintf('Elapsed time running A00_Initial_Data.m is %s \n',
    ElapsedTime)
267 return
```

Algoritmo	Objetivo	Categoria
Opt_B05_post_statistics	Gera dados estatísticos do consumo de tempo e memória com base nos arquivos de registros	Opcional

```
%%  Opt_B05_post_statistics
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   May 10, 2021
%   This script/function ***
%   This script/function ***

%% Requirements and Warnings
%

%% Summary of steps
tic

%% Load variables
Ini_A00_initial_data
clearvars -except Var_Opt_B05_post_statistics

if Var_Opt_B05_post_statistics{2} == false
    return
else
    %% Input: Results - Data Base
    FolderFemCases = Var_Opt_B05_post_statistics{2}; % Folder to generate FEM models and results

    %% Output: Post-processed files
    FolderPost = Var_Opt_B05_post_statistics{3};                                                % Folder for Post-processed files

    %% Get Statistics
    Statistics = [];
    try     % Check if there is statistics information
        load(strcat(FolderPost,'\','Statistics','.','mat'),'Statistics'); % Try to load statistics information
    catch
        Statistics = ["Case";"Release";"Build";"Update";"Platform";"Date Run";"Time";"Process ID";"Operating System";"Processor
```

```
          Model";"Compiler";"Number of machines requested";"Total
          number of cores available";"Number of physical cores
          available";"Number of processes requested";"Number of
          threads per process requested";"Total number of cores
          requested";"MPI Type";"MPI Version";"GPU Acceleration";"
          Total CPU time for main thread ";"Total CPU time summed for
           all threads";"Elapsed time spent obtaining a license ";"
          Elapsed time spent pre-processing model";"Elapsed time
          spent solution - preprocessing";"Elapsed time spent
          computing solution";"Elapsed time spent solution -
          postprocessing";"Elapsed time spent post-processing model
          ";"Equation solver used";"Equation solver computational
          rate";"Equation solver effective I/O rate";"Maimum total
          memory used";"Maimum total memory allocated";"Total
          physical memory available";"Total amount of I/O written to
          disk";"Total amount of I/O read from disk"];
32    end
33
34    if ~isempty(Statistics)
35        ProcOutFiles = Statistics(1,2:end)';    % Get *.out files
              already processed
36    end
37
38    GetListOut = dir(strcat(FolderFemCases,'\*.out'));
39    GetListOut = struct2cell(GetListOut);                    % Get
          filenames for *.out files
40
41    if isempty(GetListOut)                                   % Stop code
           if there is no .out files to be processed
42        fprintf('There is no result to be processed \n')
43        return
44    end
45
46    ListOutFiles = strings;
47    for i=1:size(GetListOut,2)
48        ListOutFiles(i,1) = GetListOut{1,i};                 % Get all
              *.out files available
49    end
50
51    CheckOutFileList = [];
52    for i=1:size(ListOutFiles,1)                             % Check *.
          out files not processed
53        CheckOutFile = find(ListOutFiles(i) == ProcOutFiles);
```

```
54          if isempty(CheckOutFile)
55              CheckOutFileList = [CheckOutFileList;ListOutFiles(i)];
56          end
57      end
58
59      ListOutFiles = CheckOutFileList;                          % List of
            *.out files to be processed
60      SS = size(Statistics,2);                                  % Current
            size of statistics matrix
61
62      for i=1:size(ListOutFiles,1)
                                                      % Loop through *.out
            files
63          fid = fopen(strcat(FolderFemCases,'\',ListOutFiles(i,1)),'r');
                  % Open *.out file
64
65          ExpToFind = "+--------- D I S T R I B U T E D   A N S Y S   S T
              A T I S T I C S ------------+";      % Expression to find
66          NumChar = strlength(ExpToFind);
67          Cond = true;
68
69          while Cond                                                 %
               Find expression
70              StringLine = fgetl(fid);                          % Read
                   line from dat file
71              try
72                  if strcmpi(StringLine(1:NumChar), ExpToFind)      %
                       Stop reading if expression is found
73                      Cond = false;
74                  end
75              end
76          end
77
78          CC = i + SS;                              % Current column
79          Statistics(1,CC) = ListOutFiles(i);
80
81          StringLine = regexp(fgetl(fid),'\s','split');         %
               Whitespace delimiter
82          StringLine = regexp(fgetl(fid),'\s','split');         %
               Whitespace delimiter
83          Statistics(2,CC) = strcat(string(StringLine{2})," ",string(
               StringLine{3}));
84          Statistics(3,CC) = strcat(string(StringLine{16})," ",string(
```

```
            StringLine{17}));
85          Statistics(4,CC) = strcat(string(StringLine{24}));
86          Statistics(5,CC) = strcat(string(StringLine{28}),' ',string(
                StringLine{29}));
87
88          StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
89          Statistics(6,CC) = strcat(string(StringLine{3}));
90          Statistics(7,CC) = strcat(string(StringLine{7}));
91          Statistics(8,CC) = strcat(string(StringLine{14}));
92
93          StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
94          Statistics(9,CC) = strcat(string(StringLine{3}),' ',string(
                StringLine{4}),' ',string(StringLine{6}),' ',string(
                StringLine{7}));
95
96          StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
97          StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
98          Statistics(10,CC) = strcat(string(StringLine{3}),' ',string(
                StringLine{4}),' ',string(StringLine{6}),' ',string(
                StringLine{7}),' ',string(StringLine{8}),' ',string(
                StringLine{9}));
99
100         StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
101         StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
102         Statistics(11,CC) = strcat(string(StringLine{2}),' ',string(
                StringLine{3}),' ',string(StringLine{4}),' ',string(
                StringLine{5}),' ',string(StringLine{6}),' ',string(
                StringLine{8}),' ',string(StringLine{9}));
103
104         StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
105         Statistics(11,CC) = strcat(Statistics(11,CC),' / ',string(
                StringLine{11}),' ',string(StringLine{12}),' ',string(
                StringLine{13}),' ',string(StringLine{14}),' ',string(
                StringLine{15}),' ',string(StringLine{17}),' ',string(
                StringLine{18}));
106
```

```
107         StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
108         Statistics(11,CC) = strcat(Statistics(11,CC)," / ",string(
                StringLine{11})," ",string(StringLine{12})," ",string(
                StringLine{13})," ",string(StringLine{14})," ",string(
                StringLine{15})," ",string(StringLine{16})," ",string(
                StringLine{17})," ",string(StringLine{18})," ",string(
                StringLine{19}));
109
110         StringLine = regexp(fgetl(fid),'\s','split');        %
                Whitespace delimiter
111         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
112         Statistics(12,CC) = strcat(string(StringLine{2}));
113
114         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
115         Statistics(13,CC) = strcat(string(StringLine{2}));
116
117         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
118         Statistics(14,CC) = strcat(string(StringLine{2}));
119
120         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
121         Statistics(15,CC) = strcat(string(StringLine{2}));
122
123         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
124         Statistics(16,CC) = strcat(string(StringLine{2}));
125
126         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
127         Statistics(17,CC) = strcat(string(StringLine{2}));
128
129         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
130         Statistics(18,CC) = strcat(string(StringLine{2}));
131
132         StringLine = regexp(fgetl(fid),':','split');            %
                Whitespace delimiter
133         Statistics(19,CC) = strcat(string(StringLine{2}));
134
```

```
135         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
136         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
137         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
138         Statistics(20,CC) = strcat(string(StringLine{2}));
139
140         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
141         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
142         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
143         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
144         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
145         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
146         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
147         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
148         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
149         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
150         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
151         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
152         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
153         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
154         Statistics(21,CC) = strcat(string(StringLine{2}));
155
156         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
157         Statistics(22,CC) = strcat(string(StringLine{2}));
158
159         StringLine = regexp(fgetl(fid),':','split');            %
```

```
            Whitespace delimiter
160         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
161         Statistics(23,CC) = strcat(string(StringLine{2}));
162
163         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
164         Statistics(24,CC) = strcat(string(StringLine{2}));
165
166         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
167         Statistics(25,CC) = strcat(string(StringLine{2}));
168
169         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
170         Statistics(26,CC) = strcat(string(StringLine{2}));
171
172         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
173         Statistics(27,CC) = strcat(string(StringLine{2}));
174
175         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
176         Statistics(28,CC) = strcat(string(StringLine{2}));
177
178         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
179         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
180         Statistics(29,CC) = strcat(string(StringLine{2}));
181
182         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
183         Statistics(30,CC) = strcat(string(StringLine{2}));
184
185         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
186         Statistics(31,CC) = strcat(string(StringLine{2}));
187
188         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
189         StringLine = regexp(fgetl(fid),':','split');            %
            Whitespace delimiter
```

```
190         Statistics(32,CC) = strcat(string(StringLine{2}));
191
192         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
193         Statistics(33,CC) = strcat(string(StringLine{2}));
194
195         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
196         Statistics(34,CC) = strcat(string(StringLine{2}));
197
198         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
199         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
200         Statistics(35,CC) = strcat(string(StringLine{2}));
201
202         StringLine = regexp(fgetl(fid),':','split');            %
               Whitespace delimiter
203         Statistics(36,CC) = strcat(string(StringLine{2}));
204
205         fclose(fid);
206     end
207
208     % Save statistics
209     save(strcat(FolderPost,'\','- Statistics','.','mat'),'Statistics');
            % Save information
210     fprintf("Information for statistics was processed and saved:
           BodyInfo.mat \n")
211 end
212
213 %%  End of code
214 fclose all;
215 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
216 fprintf('Elapsed time running D01_Post_Statistics.m is %s \n',
       ElapsedTime)
217 return
```

Algoritmo	Objetivo	Categoria
Opt_C06_plot_bodies	Gera todas as entidades do modelo de elementos finitos no formato de extensão do Gmsh	Opcional

```
%%  Opt_C06_plot_bodies
%   Author: Kennedy Neves - kennedyneves@usp.br / kennedyleandrosn@gmail.com
%   May 10, 2021
%   This script/function plots all bodies from generic FEM model in Gmsh file format
%       [pt-br] Este script/fun  o plotta todas as entidades do modelo gen rico no formato de extens o do Gmsh

%% Notes:
%  This scrip/function is optinal
%  This scrip/function is used only to better see bodies from generic FEM model in Gmsh

%% Steps:
%  Load variables
%  Write nodes, elements and bodies for .msh file
%  Save *.msh file
%  Create options file for *msh file
%  Save *.msh.opt file
%  End of code

%% Load variables
fprintf(1,"Initializing Opt_C06_plot_bodies \n")
tic

Ini_A00_initial_data
clearvars -except Var_Opt_C06_plot_bodies

% Output: Post-processed files
FolderPost = Var_Opt_C06_plot_bodies{1}; % Folder for Post-processed files

% Folder to get BodyInfo.mat
FolderLock = Var_Opt_C06_plot_bodies{2};   % Folder for locked variables

```

```
% Check body info
CheckBodyInfoStructure = [];
try
    CheckBodyInfoStructure = load(strcat(FolderLock,'\',"BodyInfo.mat")
        );
    fprintf(1,'Information for bodies loaded from file: BodyInfo.mat')

    VarInfo = struct2cell(CheckBodyInfoStructure);      % Create cell
        for *.mat files

    BodyInfo = cat(1, VarInfo{1});          % Get info for bodies
    ElementsRange = cat(1, VarInfo{2});     % Get info for elements in
        range
    NodeBodyElType = cat(1, VarInfo{4});     % Get info for bodies (
        Body Name, Element Type and Range)
    NodesFem = cat(1, VarInfo{5});          % Get info for all nodes
    NodesRange = cat(1, VarInfo{6});        % Get info for nodes in
        range

catch
    disp('File BodyInfo.mat is missing')
    return
end

%% Write nodes, elements and bodies for .msh file
% Initial lines for *.msh file
Msh = [];
Msh = "$MeshFormat";
Msh = [Msh;"2.1 0 8"];
Msh = [Msh;"$EndMeshFormat"];
Msh = [Msh;""];

% Add nodes info to *.msh file
Msh = [Msh;"$Nodes"];
Msh = [Msh;string(size(NodesRange,1))];
AddNodes = sprintf('%d  %0.8f  %0.8f  %0.8f\n',NodesRange');
AddNodes = AddNodes(1:size(AddNodes,2)-1);
Msh = [Msh;AddNodes];
Msh = [Msh;"$EndNodes"]; % End nodes
Msh = [Msh;""];

% Add elements info to *.msh file
Msh = [Msh;"$Elements"]; % Elements
```

```
70  Msh = [Msh;string(size(ElementsRange,1))];
71
72  Elements4NIdx = (1:size(ElementsRange,1))';
73  Elements2NIdx = find(isnan(ElementsRange(:,5)));    % Get idx for 2-
        node elements
74  Elements4NIdx(Elements2NIdx) = [];                  % Get idx for 4-
        node elements
75
76  Elements4NRange = ElementsRange(Elements4NIdx,:);   % Get all 4-node
        elements from Elements in Range
77  AddElements4N = sprintf('%d     3       3       1       1       0
              %d  %d  %d %d \n',Elements4NRange'); % Format for 4-node
        elements
78  AddElements4N = AddElements4N(1:size(AddElements4N,2)-1);   % Delete
        last line
79
80  Elements2N = ElementsRange(Elements2NIdx,1:3);      % Get all 2-node
        elements from Elements in Range
81  AddElements2N = sprintf('%d     1       3       2       2       0
              %d  %d \n',Elements2N'); % Format for 2-node elements
82  AddElements2N = AddElements2N(1:size(AddElements2N,2)-1);   % Delete
        last line
83
84  Msh = [Msh;AddElements4N];  % Add 4-node elements to *.msh file
85  Msh = [Msh;AddElements2N];  % Add 2-node elements to *.msh file
86  Msh = [Msh;"$EndElements"]; % End elements
87  Msh = [Msh;""];
88
89  % % % Exception for Body Name of Upper Hull / DO NOT USE DUPLICATED
        NAMES FOR BODIES IN FUTURE
90  % % BodyInfo{1,1}(1,1) = "upper hull 1";
91  % % BodyInfo{1,3}(1,1) = "upper hull 2";
92
93  BodyName = [];         % Array to get view number on Gmsh
94  for i=1:size(BodyInfo,2)
95      NumRow = size(BodyInfo{1,i},1);
96      if NumRow >= 2
97          Msh = [Msh;"$ElementData"];          % Start element data
98          Msh = [Msh;"1"];
99          AddBodyName = strcat('"',BodyInfo{1,i}(1,1),'"');
100         BodyName = [BodyName;AddBodyName];
101         Msh = [Msh;AddBodyName];             % Add body name
102         Msh = [Msh;"1"];
```

```
103         Msh = [Msh;"1"];                        % Time Step ***
104         Msh = [Msh;"4"];
105         Msh = [Msh;"0"];                        % Step Number ***
106         Msh = [Msh;"1"];                        % Number of components
107         Msh = [Msh;size(BodyInfo{1,i},1)-1];                    %
                Number of elements
108         Msh = [Msh;"0"];
109         AddElementData = BodyInfo{1,i}(2:end,1);    % Element Data
110         AddElementData(:,2) = i;                    % Body idx
111
112         AddElementData = sprintf('%d    %d \n',str2double(
                AddElementData)'); % Format for element data
113         AddElementData = AddElementData(1:size(AddElementData,2)-1);
                % Delete last line
114         Msh = [Msh;AddElementData];         % Add Element Data
115         Msh = [Msh;"$EndElementData"];      % End Element Data
116         Msh = [Msh;""];
117     end
118 end
119
120 %% Save *.msh file
121 fid = fopen(strcat(FolderPost,'\','Bodies.msh'),'w');
122 fprintf(fid,'%s \n',Msh);
123 fclose(fid);
124
125 %% Create options file for *msh file
126 % Data for colors
127 ColorData(1,1) = "{{255,0,0,255}};";
128 ColorData(2,1) = "{{0,255,0,255}};";
129 ColorData(3,1) = "{{0,0,255,255}};";
130 ColorData(4,1) = "{{255,255,0,255}};";
131 ColorData(5,1) = "{{255,0,255,255}};";
132 ColorData(6,1) = "{{0,255,255,255}};";
133 ColorData(7,1) = "{{255,100,0,255}};";
134 ColorData(8,1) = "{{255,0,100,255}};";
135 ColorData(9,1) = "{{0,255,100,255}};";
136 ColorData(10,1) = "{{100,255,0,255}};";
137 ColorData(11,1) = "{{100,0,255,255}};";
138 ColorData(12,1) = "{{0,100,255,255}};";
139 ColorData(13,1) = "{{255,100,255,255}};";
140 ColorData(14,1) = "{{255,255,100,255}};";
141 ColorData(15,1) = "{{100,255,255,255}};";
142 ColorData(16,1) = "{{128,0,0,255}};";
```

```
143 ColorData(17,1) = "{{0,128,0,255}};";
144 ColorData(18,1) = "{{0,0,128,255}};";
145 ColorData(19,1) = "{{128,128,0,255}};";
146 ColorData(20,1) = "{{128,0,128,255}};";
147 ColorData(21,1) = "{{0,128,128,255}};";
148 ColorData(22,1) = "{{128,50,0,255}};";
149 ColorData(23,1) = "{{128,0,50,255}};";
150 ColorData(24,1) = "{{0,128,50,255}};";
151 ColorData(25,1) = "{{50,128,0,255}};";
152 ColorData(26,1) = "{{50,0,128,255}};";
153 ColorData(27,1) = "{{0,50,128,255}};";
154 ColorData(28,1) = "{{128,50,128,255}};";
155 ColorData(29,1) = "{{128,128,50,255}};";
156 ColorData(30,1) = "{{50,128,128,255}};";
157
158 expression = {'(^|\.)\s*.','"','[0-9]',' A ',' B ',' C ',' D ',' E ','
    F ',' G ',' H ',};
159 replace = '';
160 BodyNameComp = regexprep(BodyName,expression,replace); % Body names to
    compare
161 UniqueBodyNameComp = unique(BodyNameComp);             % Groups of
    bodies
162
163 % Assign a color for each groups of bodies
164 TotalOfColors = size(ColorData,1);
165 for i=1:size(UniqueBodyNameComp,1)
166     ColorID = (i - ((floor(i/TotalOfColors))*TotalOfColors))+1;
167     UniqueBodyNameComp(i,2) = ColorID;
168 end
169
170 % Info for options with colors for each body
171 MshOpt = [];
172 MshOpt = [MshOpt;"General.RotationX = 290.000;"];
173 MshOpt = [MshOpt;"General.RotationY = 0.575;"];
174 MshOpt = [MshOpt;"General.RotationZ = 40.000;"];
175
176 MshOpt = [MshOpt;"General.TrackballQuaternion0 = 0.5278058978190083;"];
177 MshOpt = [MshOpt;"General.TrackballQuaternion1 =
    -0.2000172533661817;"];
178 MshOpt = [MshOpt;"General.TrackballQuaternion2 =
    -0.2841926020513545;"];
179 MshOpt = [MshOpt;"General.TrackballQuaternion3 = 0.775015224058598;"];
180
```

```
181 for i=1:size(BodyNameComp,1)
182 %     for j=1:size(UniqueBodyNameComp,1)
183 %         if BodyNameComp(i) == UniqueBodyNameComp(j)
184             NewLine = strcat('View[',string(i-1),'].Visible = 0;');
185             MshOpt = [MshOpt;NewLine];
186 
187             ColorIdx = find(BodyNameComp(i,1)==UniqueBodyNameComp(:,1))
                    ;
188             ColorIdx = UniqueBodyNameComp(ColorIdx,2);
189             ColorIdx = ColorData(str2double(ColorIdx),1);
190             NewLine = strcat('View[',string(i-1),'].ColorTable = ',
                    ColorIdx);
191 
192             MshOpt = [MshOpt;NewLine];
193             NewLine = strcat('View[',string(i-1),'].ShowScale = 0;');
194             MshOpt = [MshOpt;NewLine];
195 %             break
196 %         end
197 %     end
198 end
199 
200 
201 % % % for i=1:size(BodyNameComp,1)
202 % % %     for j=1:size(UniqueBodyNameComp,1)
203 % % %         if BodyNameComp(i) == UniqueBodyNameComp(j)
204 % % %             NewLine = strcat('View[',string(i-1),'].Visible =
        0;');
205 % % %             MshOpt = [MshOpt;NewLine];
206 % % %             NewLine = strcat('View[',string(i-1),'].ColorTable =
        ',ColorData(j));
207 % % %             MshOpt = [MshOpt;NewLine];
208 % % %             NewLine = strcat('View[',string(i-1),'].ShowScale =
        0;');
209 % % %             MshOpt = [MshOpt;NewLine];
210 % % %             break
211 % % %         end
212 % % %     end
213 % % % end
214 
215 %% Save *.msh.opt file
216 fid = fopen(strcat(FolderPost,'\','Bodies.msh.opt'),'w');
217 fprintf(fid,'%s \n',MshOpt);
218 fclose(fid);
```

```
219
220 %%  End of code
221 fclose all;
222 ElapsedTime = datestr(toc/86400, 'HH:MM:SS.FFF');
223 fprintf('Elapsed time running D04_Plot_Bodies.m is %s \n',ElapsedTime)
224 return
```

D. Apêndice - Modelos de elementos finitos dos estudos de caso

Ambos os modelos de elementos finitos foram criados no *software* comercial ANSYS Mechanical® utilizando os elementos:

- Elemento de casca - `SHELL181` (ANSYS, 2020): adequado para analisar estruturas de cascas finas a moderadamente espessas. É um elemento de quatro nós com seis graus de liberdade em cada nó: translações nas direções x, y e z, e rotações sobre os eixos x, y e z. Adequado para aplicações lineares, de grande rotação e, ou de grande deformação não linear. A mudança na espessura da casca é contabilizada em análises não lineares. No domínio do elemento, os esquemas de integração total e reduzida são suportados. O elemento considera os efeitos de deformação da estrutura para pressões distribuídas. A Figura D.1 mostra a geometria, as localizações dos nós e o sistema de coordenadas do elemento para este elemento. O elemento é definido pelas informações da seção do *shell* e por quatro nós globais (I, J, K e L):

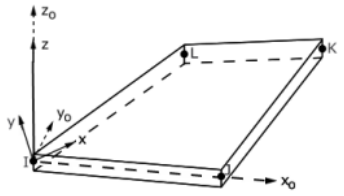


Figura D.1.: Geometria do elemento de casca, localizações dos nós e o sistema de coordenadas (ANSYS, 2020).

- Elemento de viga - `BEAM188` (ANSYS, 2020): adequado para analisar estruturas de vigas delgadas a moderadamente compactas ou espessas. O elemento é baseado na teoria de vigas de Timoshenko, que inclui efeitos de deformação por cisalhamento. O elemento fornece opções para deformação torcional irrestrita e restrita de seções transversais. É um elemento de viga tridimensional

linear, quadrático ou cúbico de dois nós. Tem seis ou sete graus de liberdade em cada nó. Isso inclui translações nas direções x, y e z, e rotações nas direções x, y e z. Um sétimo grau de liberdade (magnitude de deformação) é opcional. Este elemento é adequado para aplicações lineares, de grande rotação e, ou de grande deformação não linear. O elemento inclui termos de rigidez à tensão, por padrão, em qualquer análise com grande deflexão. Os termos de rigidez à tensão fornecidos permitem que os elementos analisem problemas de estabilidade de lateral, na flexão e torção. Elasticidade, plasticidade, fluência e outros modelos de materiais não lineares são suportados. Uma seção transversal associada a este tipo de elemento pode ser uma seção construída referenciando mais de um material. Massa adicional, massa adicional e carga hidrodinâmica, e carga de empuxo estão disponíveis.

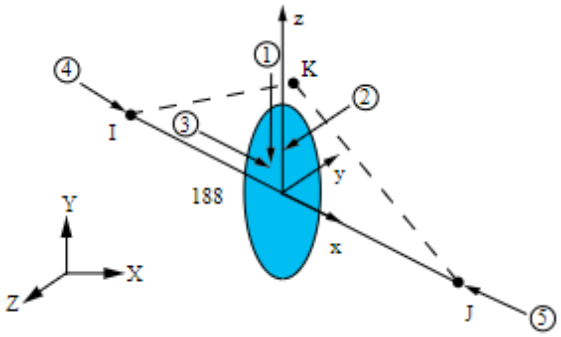
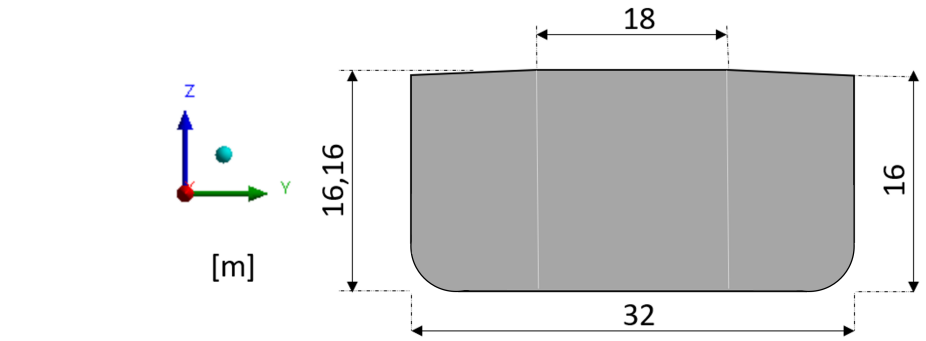


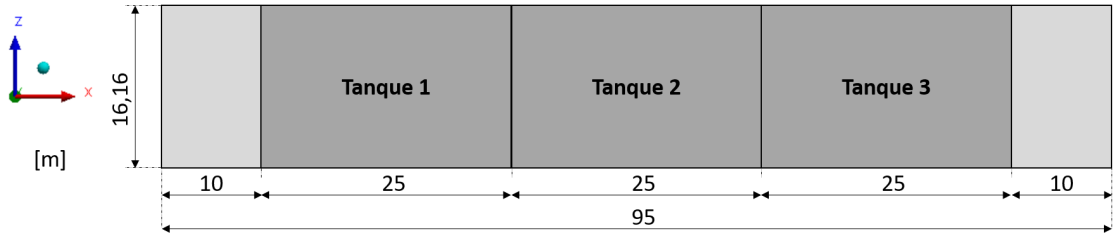
Figura D.2.: Geometria do elemento de viga, localizações dos nós e o sistema de coordenadas (ANSYS, 2020).

D.1. Modelo de elementos finitos - Estudo de caso 1

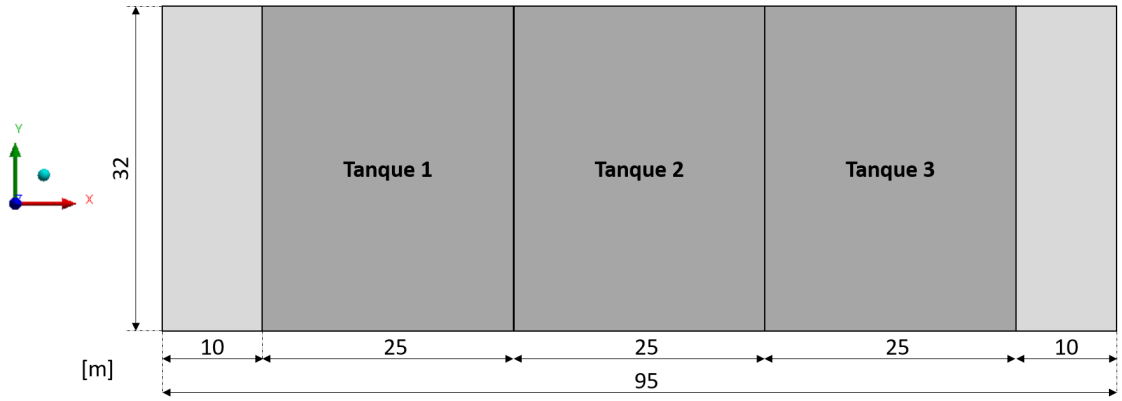
As figuras a seguir apresentam mais detalhes do modelo criado para o estudo de caso 1.



(a) vista frontal

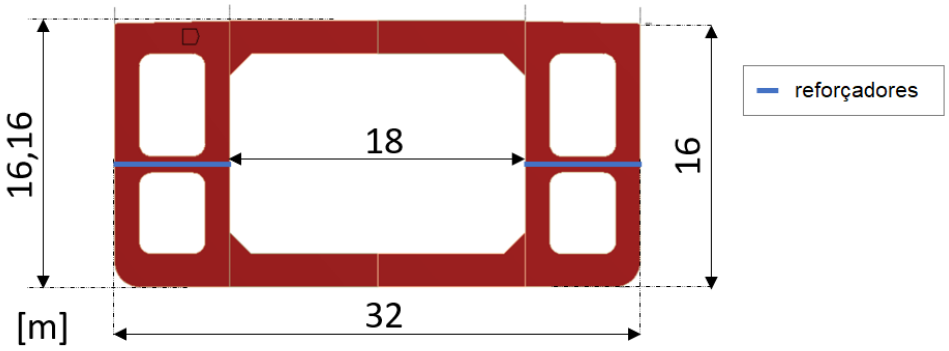


(b) vista lateral

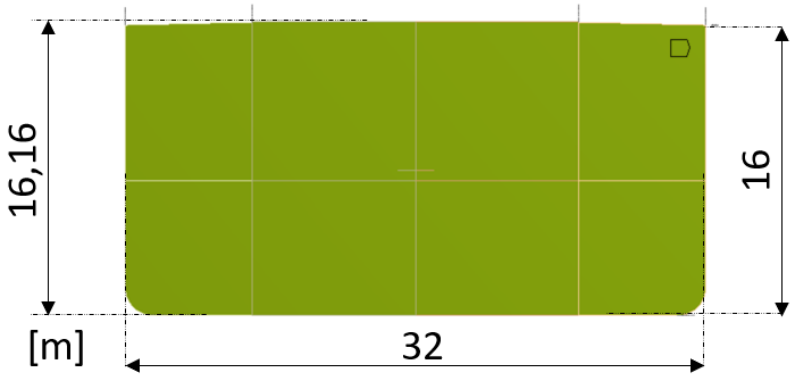


(c) vista superior

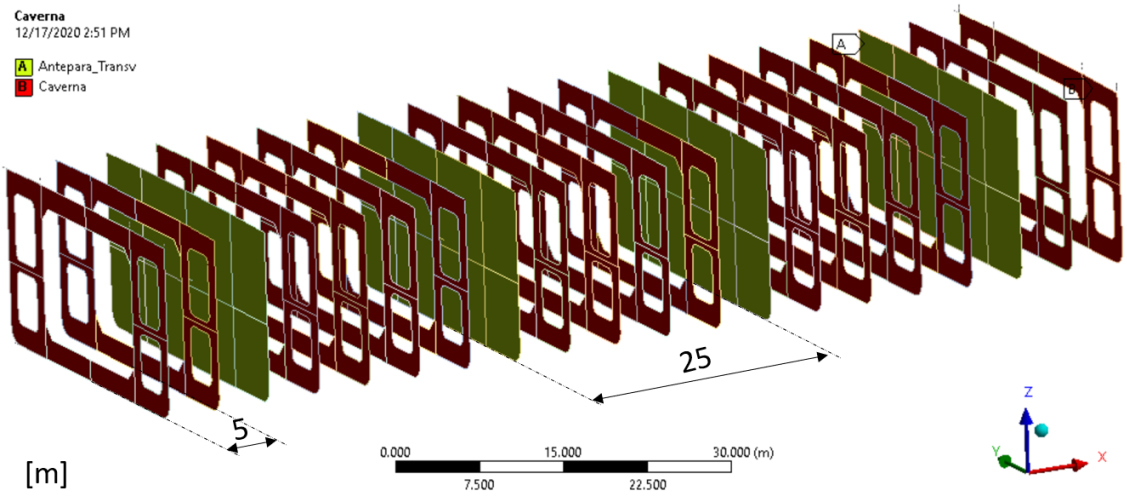
Figura D.3.: Vistas e dimensões do modelo de elementos finitos.



(a) dimensões e contorno da caverna



(b) dimensões e contorno da antepara transversal



(c) espaçamento e disposição das cavernas e anteparas transversais

Figura D.4.: Cavernas e anteparas transversais.

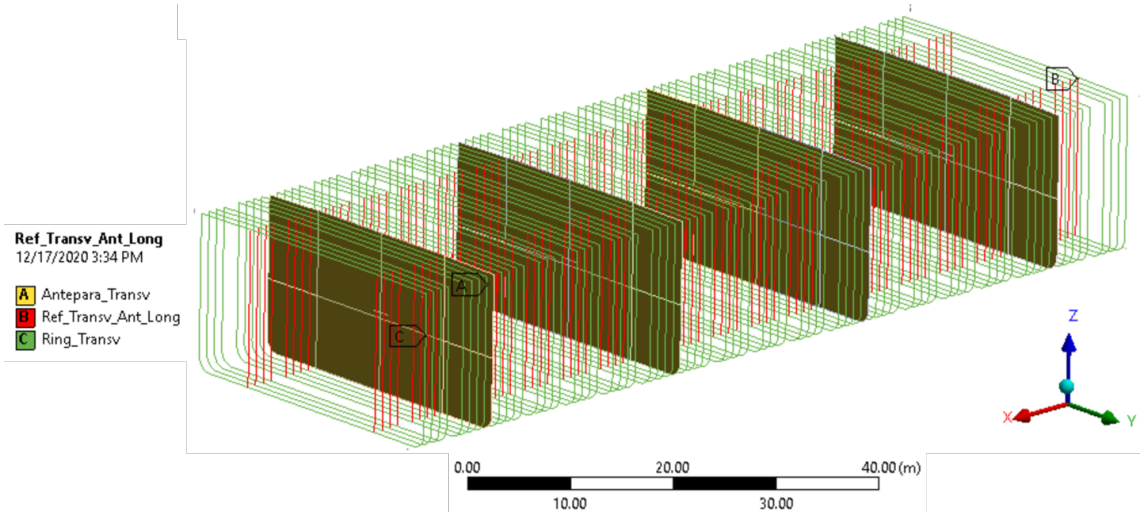
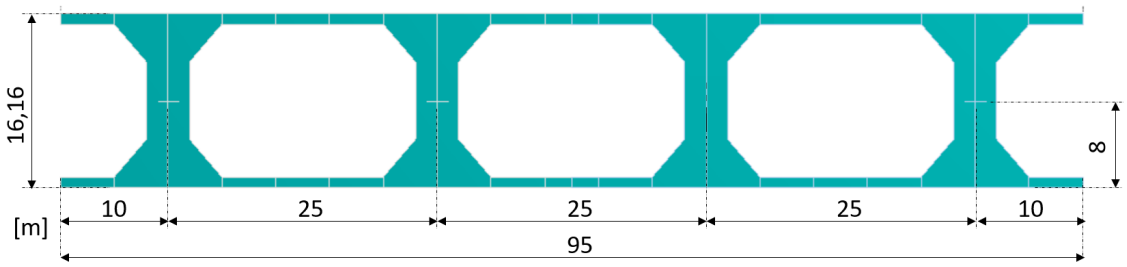
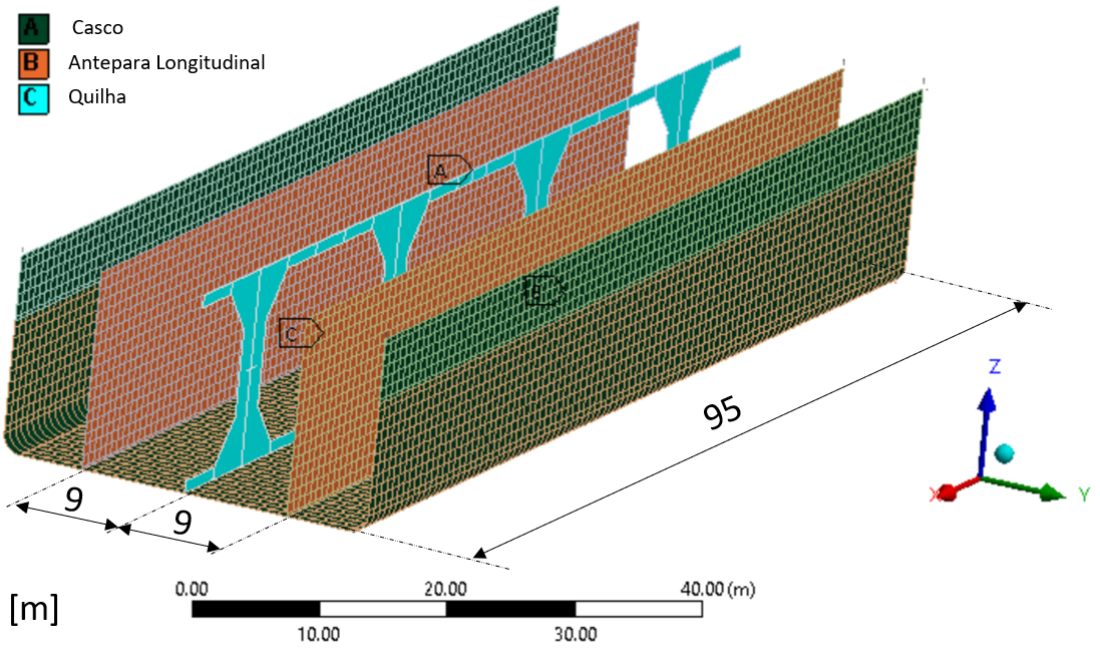


Figura D.5.: Disposição dos anéis transversais e reforços da antepara longitudinal.



(a) estrutura no plano da quilha



(b) anteparas longitudinais, casco e estrutura no plano da quilha

Figura D.6.: Dimensões, espaçamento e disposição.

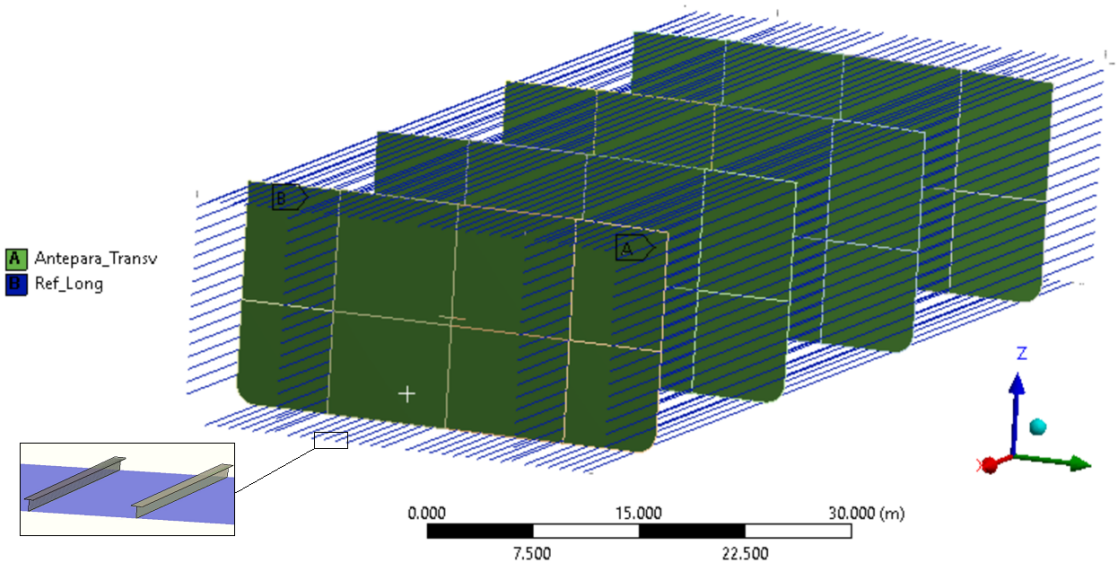


Figura D.7.: Disposição das vigas longitudinais.

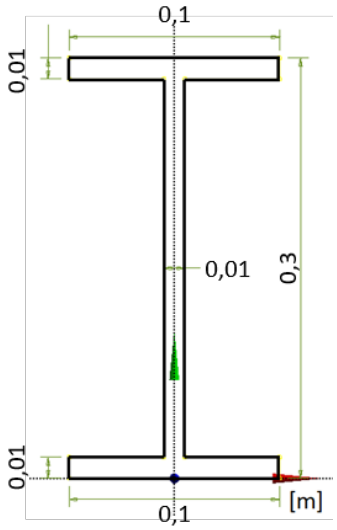


Figura D.8.: Dimensões da seção transversal e espessura das vigas.

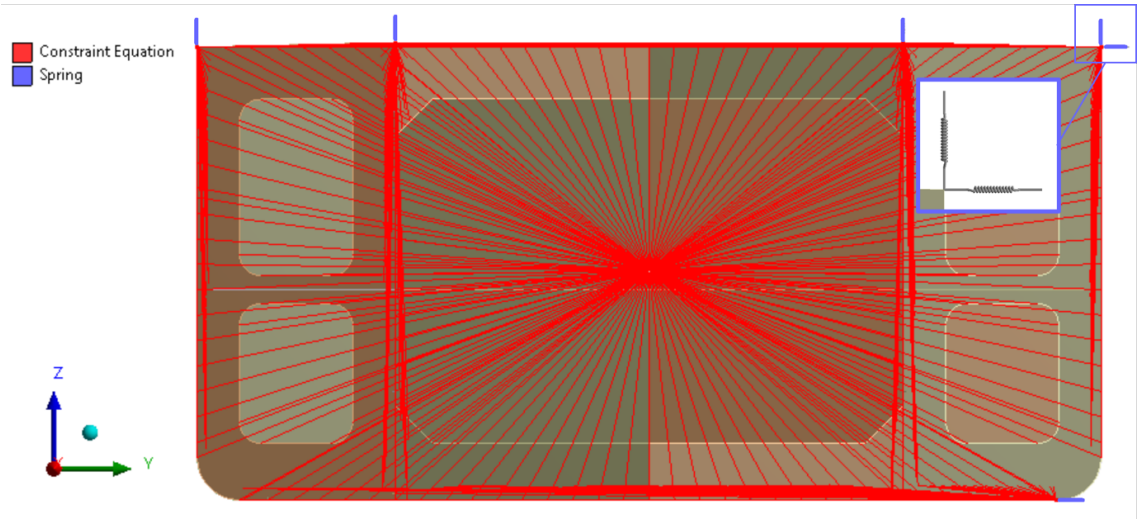


Figura D.9.: Condições de contorno do modelo com detalhes das molas.

D.2. Modelo de elementos finitos - Estudo de caso 2

As figuras a seguir apresentam mais detalhes do modelo de elementos finitos criado no *software* comercial Ansys Mechanical® utilizado no estudo de caso 2.

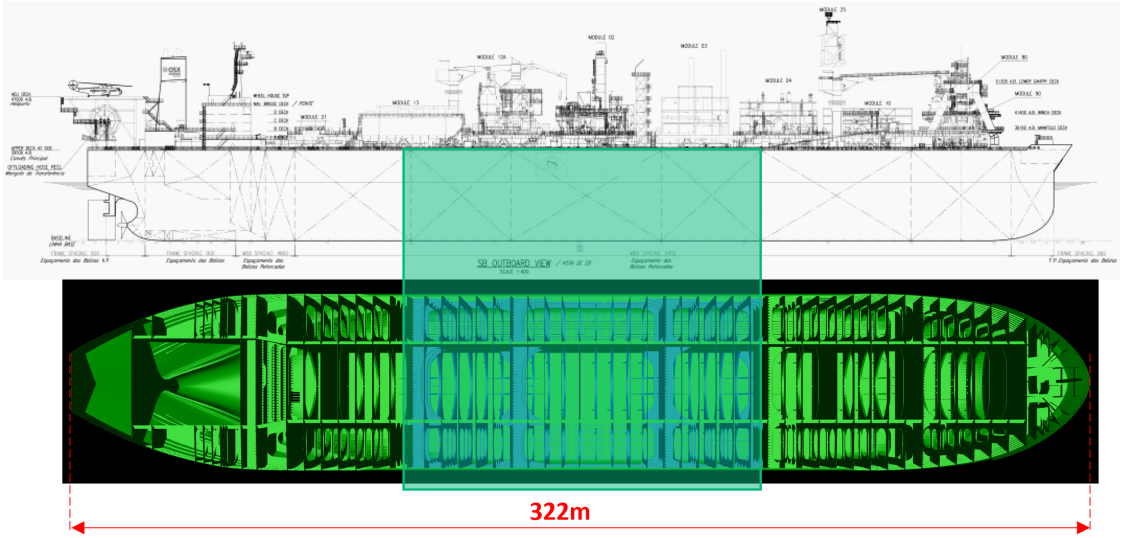


Figura D.10.: Arranjo da embarcação com detalhe dos três tanques centrais em vista superior.

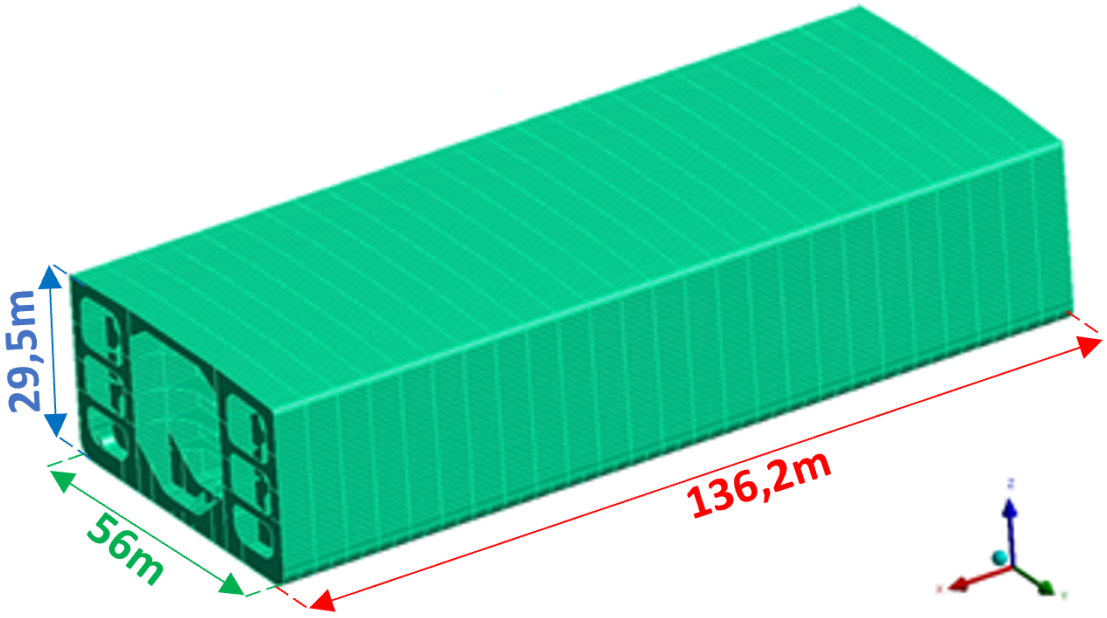


Figura D.11.: Dimensões e modelo de elementos finitos de três tanques de carga com adição de duas cavernas.

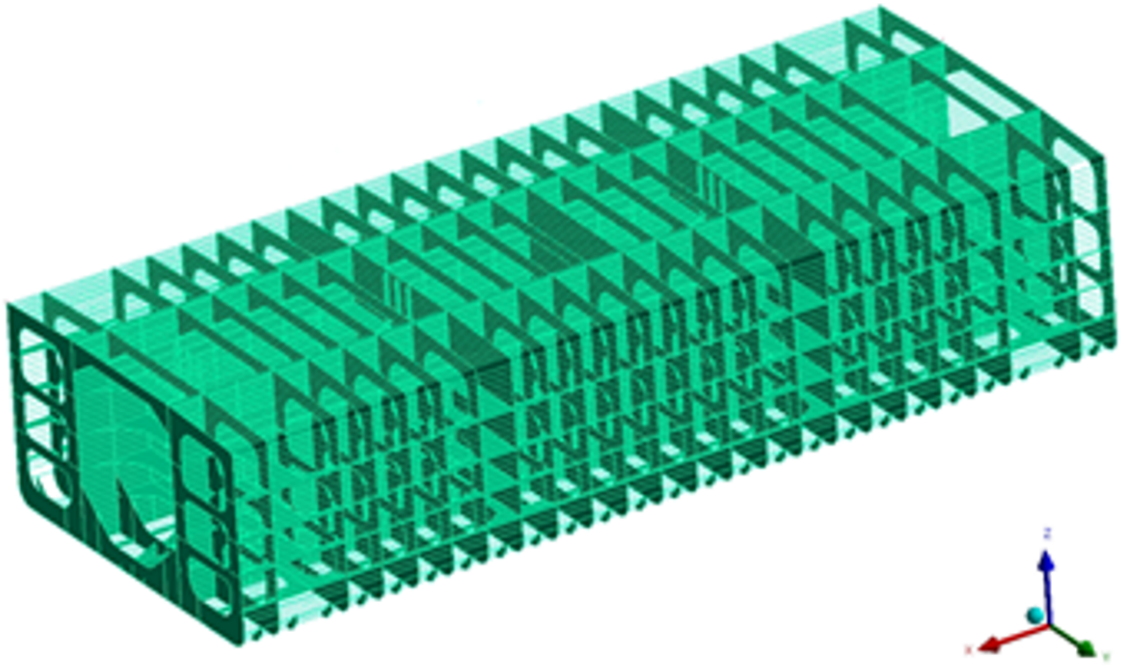
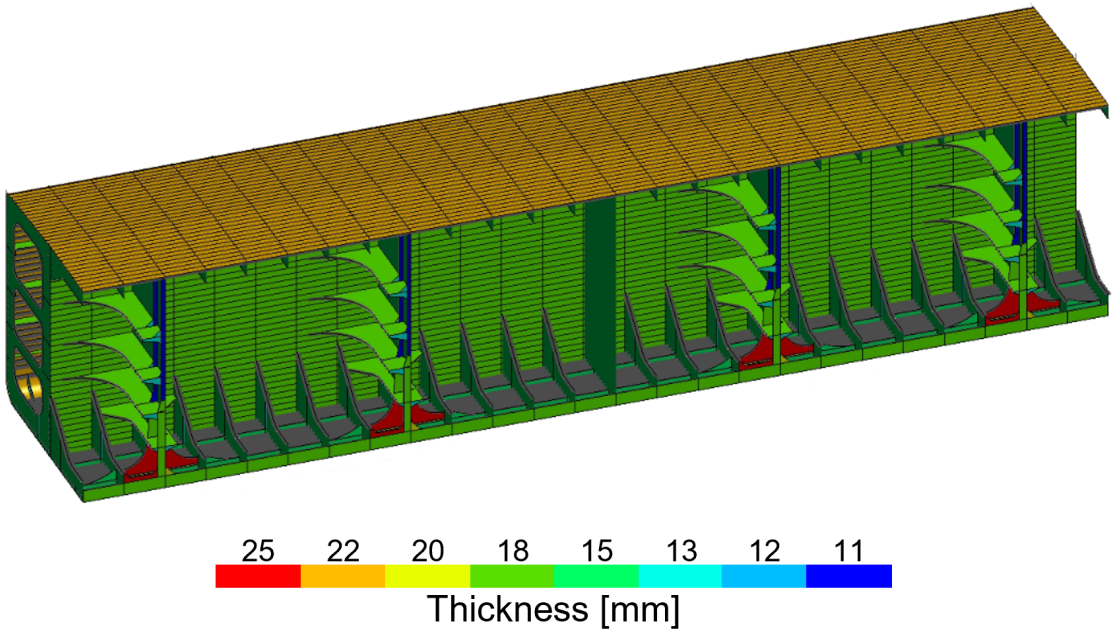
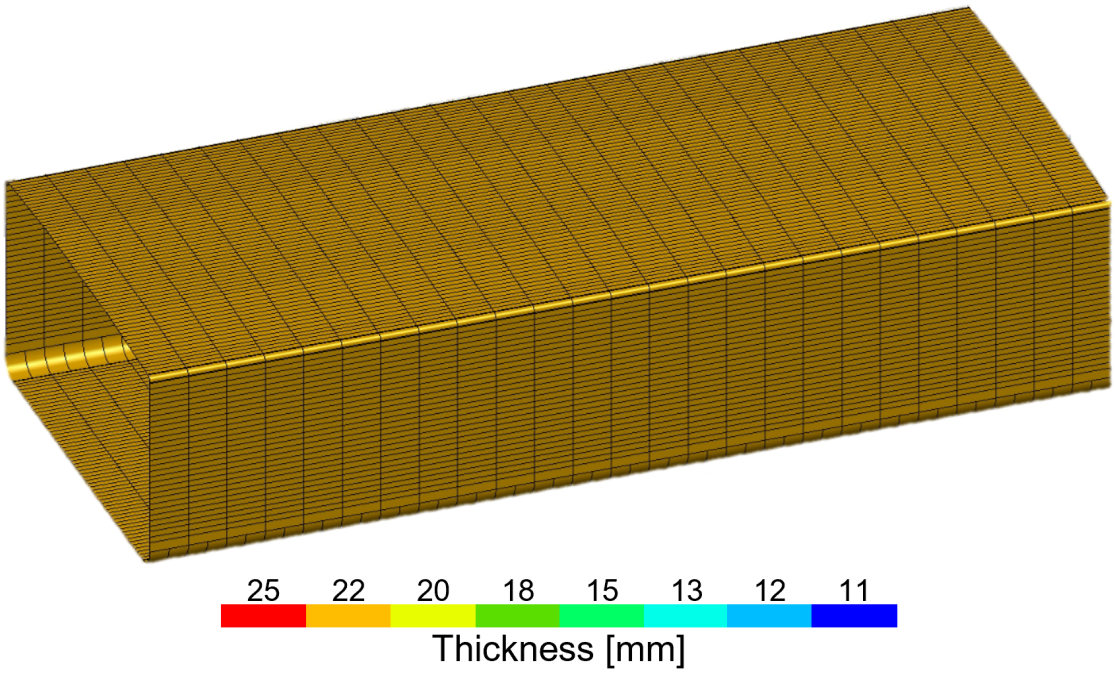


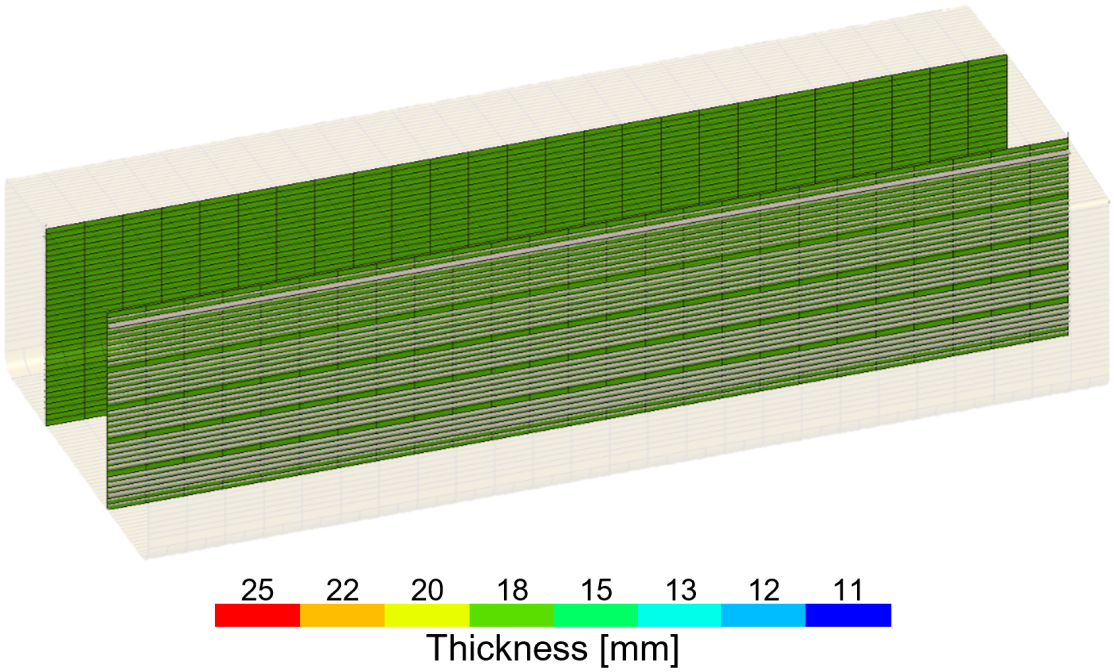
Figura D.12.: Modelo de elementos finitos de três tanques sem visualização do casco.



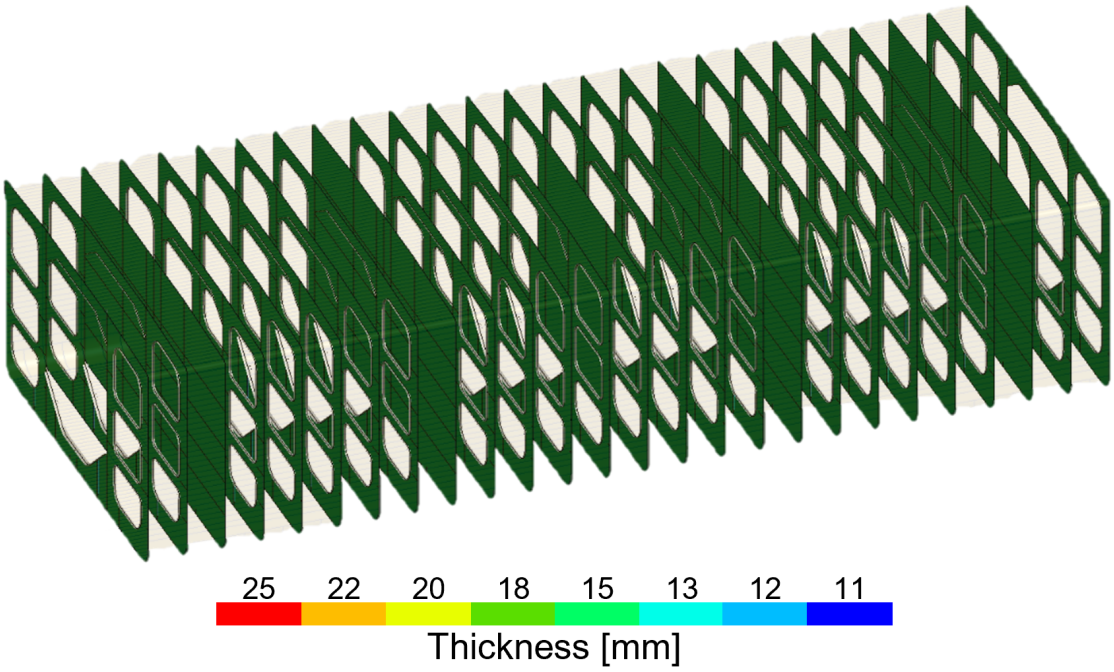
(a) Vista isométrica do plano de seção longitudinal.



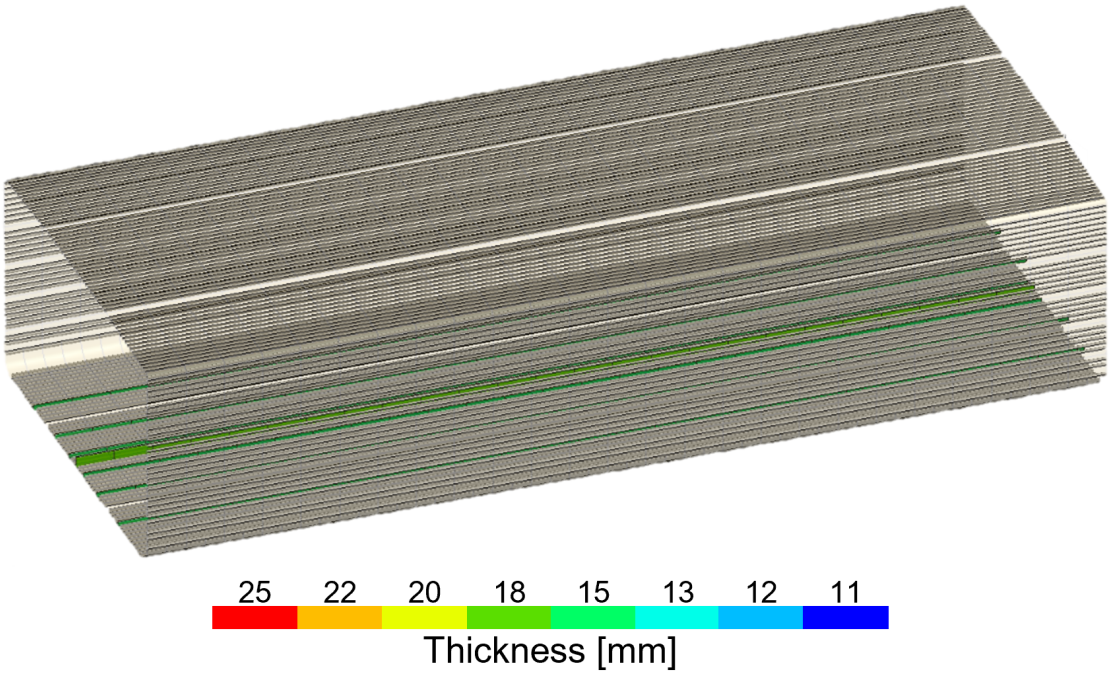
(b) Convés, laterais e fundo do casco.



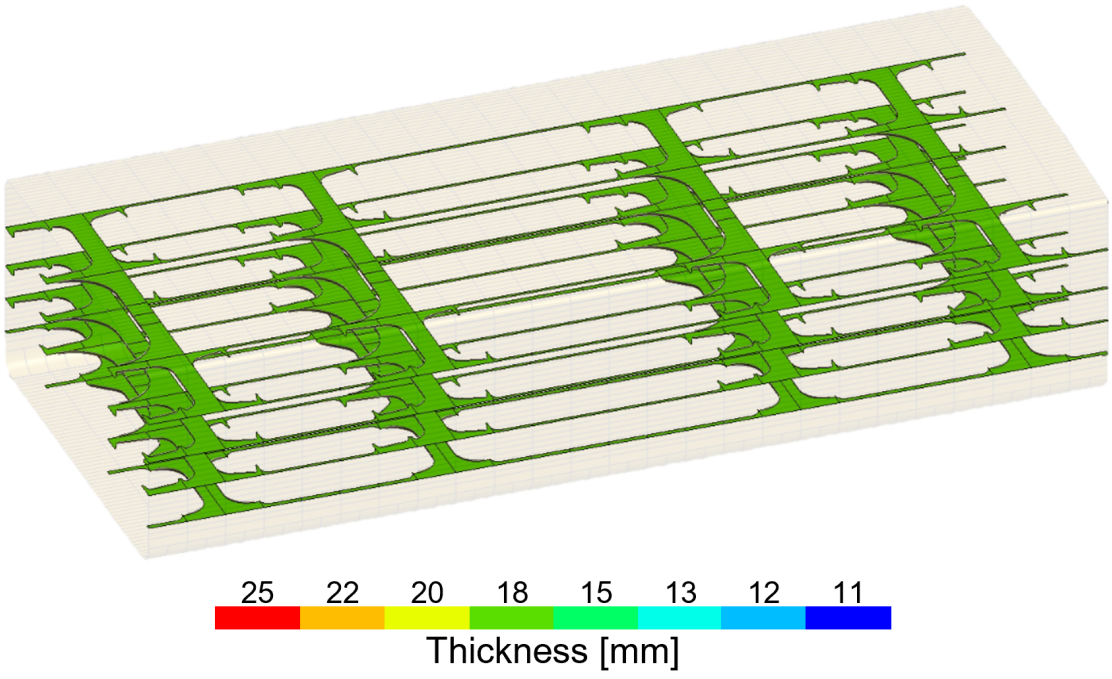
(c) Anteparas longitudinais e reforçadores.



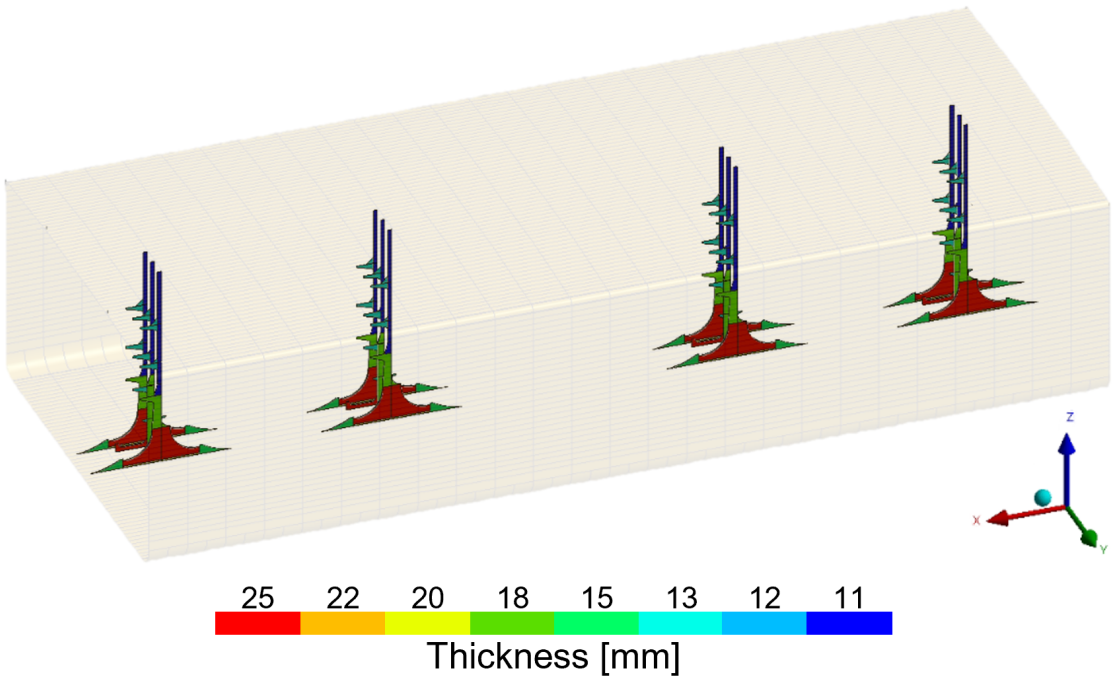
(d) Anteparas transversais e cavernas.



(e) Vigas longitudinais, quilha e reforçadores.



(f) Escoas.



(g) Anteparas não-estanques.

Figura D.12.: Membros estruturais do modelo de elementos finitos com espessuras.

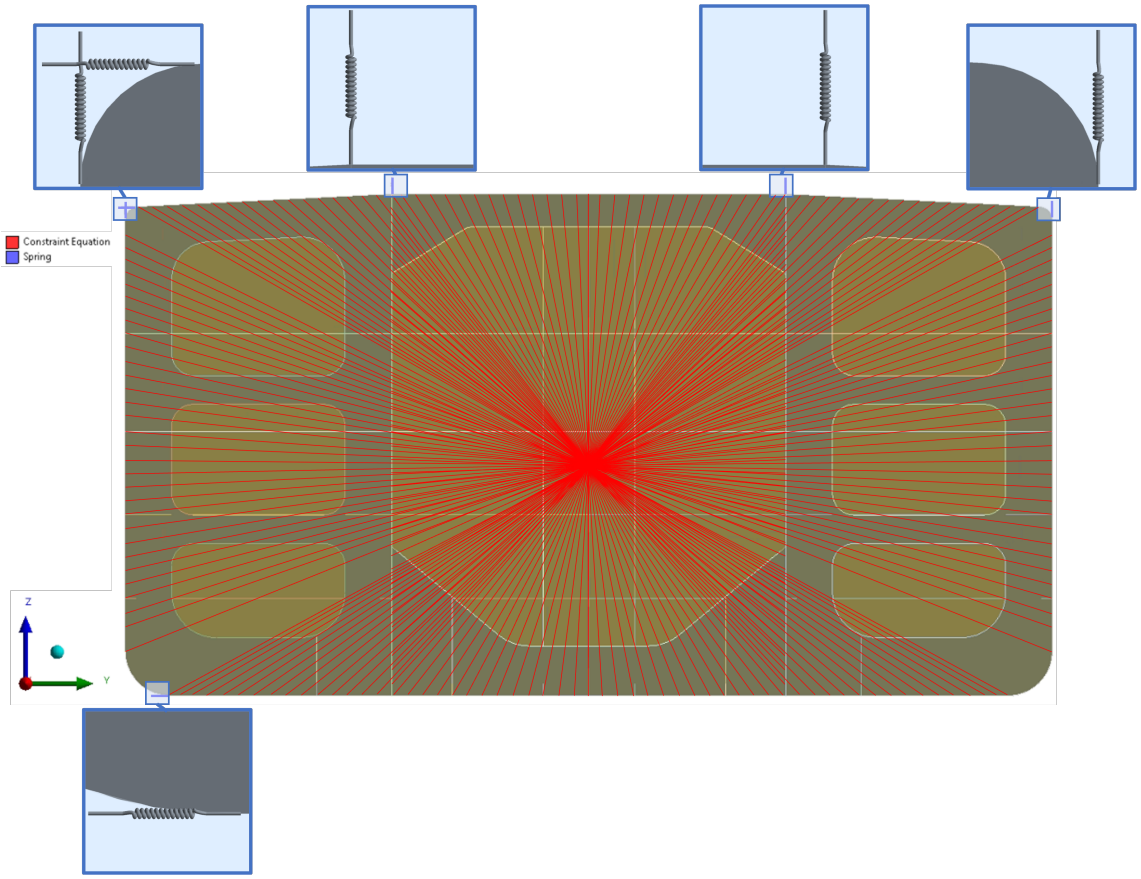


Figura D.13.: Condições de contorno do modelo com detalhes das molas.