UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

Projeto de Conclusão de Curso
Relatório Final - PME2600 - Projeto Integrado III

“Ferramenta Eficiente para Análise Estrutural de Tubos Flexíveis usando Macroelementos Finitos”

Fernando Geremias Toni
Orientador: Prof. Dr. Clóvis de Arruda Martins
Coorientador: Prof. Dr. Rodrigo Provasi Correia

São Paulo
Novembro de 2014

UNIVERSIDADE DE SÃO PAULO ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

“Ferramenta Eficiente para Análise Estrutural de Tubos Flexíveis usando Macroelementos Finitos”

Trabalho de formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Graduação em Engenharia.

Fernando Geremias Toni
Orientador: Prof. Dr. Clóvis de Arruda Martins
Coorientador: Prof. Dr. Rodrigo Provasi Correia

Área de Concentração: Elementos Finitos

São Paulo
Novembro de 2014

Toni, Fernando Geremias
Ferramenta eficiente para análise estrutural de tubos flexíveis usando macroelementos finitos / F.G. Toni. – São Paulo, 2014.
120 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecânica.

1.Método dos elementos finitos 2.Tubos flexíveis (Simulação computacional) 3.Matrizes esparsas 4.Métodos diretos para sistemas lineares I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecânica II.t.

AGRADECIMENTOS

Ao Prof. Dr. Clóvis de Arruda Martins pela orientação deste trabalho de conclusão de curso e pelas discussões que muito têm contribuído à minha formação profissional.

Ao Prof. Dr. Rodrigo Provasi pela coorientação deste trabalho de conclusão de curso.

Aos meus colegas de trabalho do LMO: Eduardo Malta, Marcos Rabelo, Leonardo Garcez, Caio Santos e Rafael Salles, pelo agradável convívio e ambiente de trabalho durante a realização deste projeto.

LISTA DE FIGURAS

Figura 1 – Evolução das reservas provadas de petróleo, por localização (terra e mar) – 2003-2012. Fonte: (AGÊNCIA NACIONAL DO PETRÓLEO, 2013).....15
Figura 2 - Elementos de ligação são responsáveis por interligar as plataformas às unidades de extração presentes no leito do oceano. Fonte: (2B1st Consulting, 2014). 16
Figura 3 – Tubo flexível para aplicações offshore. Fonte: PETROLEO ENERGIA. ..17
Figura 4 – Exemplo de um cabo umbilical para aplicações offshore. Fonte: (VALLOUREC, 2014) 17
Figura 5 – Camadas de um tubo do tipo "Rough Bore Reinforced". Fonte: API RP 17B. 21
Figura 6 – Sistema de coordenadas utilizado na formulação do elemento cilíndrico de parede espessa. Fonte: (PROVASI & MARTINS, 2013-c). 23
Figura 7 – Elemento infinitesimal cilíndrico. Fonte: (PROVASI & MARTINS, 2013-c). 23
Figura 8 – Comparação de deslocamento radial para a superfície superior. Fonte: (PROVASI & MARTINS, 2013-c).....24
Figura 9 – Comparação de deslocamentos radiais para a superfície de capa. Fonte: (PROVASI & MARTINS, 2013-c).....24
Figura 10 – Esquematização do elemento e sistema de coordenas local associado. Fonte: PROVASI (2013). 25
Figura 11 – Elemento de ligação tipo "bridge". Fonte: (PROVASI, 2013). 28
Figura 12 – Dois corpos em situação de pré-contato. Fonte: (PROVASI, 2013).....29
Figura 13 – Exemplos do efeito da discretização do contato: (a) pouco discretizada; (b) discretizada adequadamente. Fonte: (PROVASI, 2013).....30
Figura 14 – Compliance Law para o contato normal: (a) comportamento ideal; (b) comportamento usando uma Compliance Law; (c) comportamento usando o método das penalidades. Fonte: (PROVASI, 2013). 31
Figura 15 – Primeiro caso, configuração inicial. Bloco central, é uma situação com sticking. Bloco a direita, situação com sliding. Fonte: (PROVASI, 2013). 31

Figura 16 - Contato nó a nó: ponto 1 faz parte de um elemento cilíndrico e tem seu deslocamento representado através da expansão em Série de Fourier; o ponto 2 faz parte de um elemento de hélice. 33
Figura 17 - Caso simplificado de apenas 1 elemento e em contato apenas no lado externo: a distribuição dos esforços deve ser conhecida para a aresta superior e para a inferior. 38
Figura 18 - Exemplos de distribuições angulares de esforços expandidos em Série de Fourier. (COOK, MALKUS, PLESHA, & WITT, 2002) 38
Figura 19 - Esforços concentrados na região de contato. Esta figura exemplifica a hipótese para a direção normal. Os modelos matemáticos são válidos para pequenas deformações e o fenômeno foi ampliado nesta imagem apenas para recurso didático. 39
Figura 20 - Expansão em série de Fourier de ordem 20 para uma força concentrada de valor unitário aplicada a π/4 (45°). A integral desta função é igual ao módulo da força aplicada, ou seja, 1N. 40
Figura 21 – Pilha de execução gerada pelo software “dotTrace 5.5.5 Performance”. 43
Figura 22 – Pilha de execução gerada pelo software “ANTS Performance Profiler 8”. 43
Figura 23 - Exemplo de um organograma gerado automaticamente pelo software “ANTS Performance Profiler 8”.....44
Figura 24 - Gráfico de consumo de memória em função do tempo gerado pelo software “dotMemory 4.0”. 45
Figura 25 – Interface exibida pelo programa “dotMemory 4.0” ao se analisar um snapshot.....46
Figura 26 – Gráfico de consumo de memória em função do tempo gerado pelo software “dotMemory 4.0” 47
Figura 27 – Organograma criado através da compilação dos dados gerados pelo software “dotTrace 5.5.5 Performance”. 48
Figura 28 – Organograma gerado automaticamente pelo software “ANTS Performance Profiler”. 48
Figura 29 – Comparação entre o tempo total de execução do programa “UFCad” e o tempo de montagem da matriz global de rigidez em função do tamanho (em

megabytes) da matriz global de rigidez. A variação no tamanho da matriz de rigidez foi obtida aumentando-se o número de macroelementos finitos do modelo e mantendo-se constante a ordem da expansão em Série de Fourier dos elementos de contato. 49
Figura 30 – Comparação entre o tempo total de execução do programa “UFCad” e o tempo de montagem da matriz global de rigidez em função do tamanho (em megabytes) da matriz global de rigidez. A variação no tamanho da matriz de rigidez foi obtida aumentando-se a ordem da expansão em série de Fourier e mantendo-se constante o número de macroelementos finitos do modelo. 50
Figura 31 – Uso de CPU durante a execução do profiler “ANTS”. 51
Figura 32 - Consumo de memória (MB) em função do número de elementos cilíndricos. 52
Figura 33 - Consumo de memória (em megabytes) em função da ordem da expansão em Série de Fourier.....53
Figura 34 - Taxa de convergência em função da atualização ou não das dimensões no cálculo da matriz de rigidez (Newton-Raphson Option (NROPT), 2014).....56
Figura 35 – Comparação de tempo de montagem de matriz global quando: a matriz é calculada apenas 1 vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha).....57
Figura 36 - Comparação de tempo total de execução do programa quando: a matriz é calculada apenas uma vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha). 57
Figura 37 - Razão entre tempo de montagem da matriz global de rigidez e tempo total de execução quando: a matriz é calculada apenas 1 vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha). 58
Figura 38 – Comparação de consumo máximo de memória quando: a matriz é calculada apenas 1 vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha).....58
Figura 39 - Consumo de memória ao longo do tempo para o caso em que a matriz de rigidez é calculada em cada step. 59
Figura 40 - Consumo de memória ao longo do tempo para o caso em que a matriz global de rigidez é calculada apenas uma vez. 59

Figura 41 - Comparação de performance entre os métodos de montagem da matriz global de rigidez. 61
Figura 42 – Aumento na capacidade de armazenamento com a implementação do solver com matrizes esparsas. 64
Figura 43 – Perda de performance na montagem da matriz global de rigidez gerada pela implementação da matriz global de rigidez. 65
Figura 44 – Planificação das trajetórias dois arames de armaduras de tração. 66
Figura 45 – Pontos de intersecção são determinados algebricamente pela expressão apresentada. 68
Figura 46 – Exemplo de checagem de distância entre os nós da armadura externa e o primeiro ponto geométrico de intersecção. 69
Figura 47 – Elementos de contatos criados com base nos critérios da formulação proposta. 69
Figura 48 – Recurso gráfico gerado através do programa Matlab para verificar a formação dos pares de contato. 70
Figura 49 – Resultados de simulação o caso “HelixPlusCylMeshBridgeTest()” (um arame de armadura descrito por elementos de hélice interligados através de elementos de contato rígido com elementos cilíndricos): consumo máximo de memória e tempo de simulação em função do tamanho, em megabytes, da matriz global de rigidez. 75
Figura 50 - Resultados de simulação o caso “HelixPlusCylMeshTestNewContact()” (um arame de armadura descrito por elementos de hélice interligados com elementos cilíndricos através de elementos de contato que permitem sticking e sliding): consumo máximo de memória e tempo de simulação em função do tamanho, em megabytes, da matriz global de rigidez. 75
Figura 51 – Algoritmo GMRES. 76
Figura 52 - Tubo flexível adotado para o caso de estudo. 82
Figura 53 – Curva de tensão deformação do aço 1020. 84
Figura 54 – Curva de tensão deformação do polietileno (HDPE). 85
Figura 55 – Malha estrutura de elementos finitos da capa plástica. 87
Figura 56 - Malha de elementos finitos dos arames das armaduras de tração. 88
Figura 57 - Imagem ampliada da malha de elementos finitos de um arame de tração. 89

Figura 58 - Condições de contorno do modelo.....90
Figura 59 – Tensões de Von Mises para um deslocamento imposto de 40mm. $\sigma Max = 1273\ MPa$, valor bem acima da tensão de plastificação.....91
Figura 60 – Tensões de Von Mises para um deslocamento imposto de 20mm. $\sigma Max = 1146\ MPa$, valor bem acima da tensão de plastificação.....92
Figura 61 – Tensões de Von Mises para um deslocamento imposto de 10mm. $\sigma Max = 837\ MPa$, valor próximo à tensão de plastificação do aço 1020.....92
Figura 62 – Plastificação localiza, devido ao contato do tipo *bonded*. Tensões pontuais elevadas, devido ao fato das armaduras não poderem se acomodar em uma configuração de mínima energia. 93
Figura 63 – Comparação de deslocamentos radiais da armadura interna de tração de modelos com e sem não-linearidades geométricas e materiais plásticos e elásticos. NLG: não linearidades geométricas. Elast: material elástico. Plast: material plástico. 94
Figura 64 – Comparação de deslocamentos radiais da armadura externa de tração de modelos com e sem não-linearidades geométricas e materiais plásticos e elásticos. NLG: não linearidades geométricas. Elast: material elástico. Plast: material plástico. 94
Figura 65 – Resultados do caso de estudo utilizando-se o software Abaqus. 95
Figura 66 - Deslocamentos radiais das armaduras de tração. 96
Figura 67 - Resultados de deslocamento radial para os parâmetros: nel = 30; rdiv = 2; Fourier = 0.....98
Figura 68 – Resultados de deslocamento radial para os parâmetros: nel = 50; rdiv = 2; Fourier = 5.....98
Figura 69 – Comparação de deslocamentos radiais das armaduras internas e externas dos programas UFCad e Abaqus. (UFCad: nel = 30; rdiv = 2; Fourier = 0). 100
Figura 70 – Comparação de deslocamentos radiais das armaduras internas e externas dos programas UFCad e Abaqus. (UFCad: nel = 50; rdiv = 2; Fourier = 5). 101
Figura 71 – Comparação de deslocamentos radiais das armaduras internas e externas dos programas UFCad e Abaqus. (UFCad: nel = 60; rdiv = 2; Fourier = 8). 101

LISTA DE TABELAS

Tabela 1 – Consumo de memória em função do número de elementos do modelo. Parâmetros fixos da análise: matriz global de rigidez fora do loop; ordem de Fourier igual a 0. 51
Tabela 2 - Consumo de memória em função da ordem de expansão da série de Fourier. Parâmetros fixos da análise: matriz global de rigidez fora do loop; número de elementos cilíndricos igual 100; número de elementos de hélice igual à 25. 52
Tabela 3 – Comando em C# para habilitar matrizes com mais de 2GB. 63
Tabela 4 – Resumo dos principais métodos de resolução de sistemas lineares do tipo A.x = b 79
Tabela 5 – Propriedades e parâmetros da capa plástica. 83
Tabela 6 – Propriedades e parâmetros da armadura externa de tração. 83
Tabela 7 – Propriedades e parâmetros da armadura interna de tração. 83
Tabela 8 – Propriedades do aço 1020. 84
Tabela 9 – Propriedades do material polietileno. 85
Tabela 10 – Propriedades do elemento utilizado para modelar a capa plástica 87
Tabela 11 - Propriedades do elemento utilizado para modelar as armaduras de tração. 88
Tabela 12 - Tempo de simulação e consumo de memória do programa Abaqus. .103
Tabela 13 - Tempo de simulação e consumo de memória do programa UFCad. ...103
Tabela 14 – Comparação relativa entre ambos os programas. 104

SUMÁRIO

1 INTRODUÇÃO 15

1.1 MOTIVAÇÃO 15

1.2 OBJETIVOS 17

1.3 ESTRUTURA DO TRABALHO 18

2. REVISÃO BIBLIOGRÁFICA 20

2.1 TUBOS FLEXÍVEIS E CABOS UMBILICAIS 20

2.2 ABORDAGEM AO PROBLEMA 21

2.3 MACROELEMENTOS FINITOS 22

2.3.1 Elemento Cilíndrico de Parede Espessa 22

2.3.2 Elemento de Hélice (ou Tendão) 24

2.3.3 Elemento de Ligação "Bridge" 27

2.3.4 Elemento de Contato 28

3. FORMULAÇÃO CORRIGIDA PARA O MACROELEMENTO FINITO DE CONTATO DO TIPO "NÓ-A-NÓ" ENTRE ELEMENTO CILÍNDRICO E ELEMENTO DE HÉLICE 32

3.1 FORMULAÇÃO ORIGINAL DE MACROELEMENTO FINITO DE CONTATO PROPOSTA POR (PROVASI & MARTINS, 2013-B) 32

3.2 IDENTIFICAÇÃO DA INCONSISTÊNCIA MATEMÁTICA E FORMULAÇÃO CORRIGIDA 35

4. EXPANSÃO EM SÉRIE DE FOURIER DAS REAÇÕES DE CONTATO PARA O ELEMENTO CILÍNDRICO 38

5. IDENTIFICAÇÃO DOS GARGALOS COMPUTACIONAIS 41

5.1 SELEÇÃO DOS *PROFILERS* UTILIZADOS NA IDENTIFICAÇÃO DOS GARGALOS COMPUTACIONAIS 41

5.1.1 Profiler para análise de processamento 42

5.1.2 Profiler para análise de consumo de memória computacional 44

5.1.3 Problemas gerados pela utilização incorreta de Profilers 46

5.2 RESULTADOS E CONCLUSÕES DA ANÁLISE DE PROCESSAMENTO 47

5.3 RESULTADOS E CONCLUSÕES DA ANÁLISE DE CONSUMO DE MEMÓRIA COMPUTACIONAL 51

6. MODIFICAÇÕES NO PROGRAMA UFCAD 55

6.1 MODIFICAÇÃO 01 – A ALTERAÇÃO DO NÚMERO DE VEZES EM QUE A MATRIZ GLOBAL DE RIGIDEZ É COMPUTADA 55

6.2 MODIFICAÇÃO 02 – ALTERAÇÃO NA VARREDURA DE MONTAGEM DA MATRIZ DE RIGIDEZ 60

6.3 MODIFICAÇÃO 03 – PARALELIZAÇÃO DA MONTAGEM DA MATRIZ GLOBAL DE RIGIDEZ 61

6.4 MODIFICAÇÃO 04 – INCLUSÃO DE RECURSO PARA HABILITAR MATRIZES COM MAIS DE 2GB 63

6.5 MODIFICAÇÃO 05 – CONVERSÃO DAS MATRIZES DA CLASSE "*SOLVER*" PARA MATRIZES ESPARSAS 63

6.6 MODIFICAÇÃO 06 – FORMULAÇÃO E IMPLEMENTAÇÃO DE UMA NOVA ESTRATÉGIA DE DETECÇÃO DE CONTATOS ENTRE ARMADURAS DE TRAÇÃO 65

6.6.1 Formulação de um novo método de detecção de contatos entre armaduras de tração. 66

6.6.2 Resultados desta lógica de detecção de contatos 70

6.7 MODIFICAÇÃO 07 – ALTERAÇÃO NA GERAÇÃO DE MALHAS DOS ELEMENTOS DE CONTATO TIPO BRIDGE 70

7. RESOLUÇÃO DO SISTEMA LINEAR 72

7.1 BIBLIOTECAS E MÉTODOS DE SOLUÇÃO TESTADOS 72

7.1.1 Math.NET – Numerics – Solver 72

7.1.2 Math.NET – Numerics – Solver Iterative 73

7.1.3 Math.NET – Numerics – MKL 74

7.1.4 GMRES – Biblioteca Própria 76

7.1.5 Bibliotecas profissionais (pagas) e PARDISO 76

7.2 ALTERNATIVA EMPREGADA 77

7.3 RESUMO DOS MÉTODOS DE SOLUÇÃO DE SISTEMAS LINEARES 78

8. VALIDAÇÃO DE UM CASO DE ESTUDO COM O PROGRAMA UFCAD 81

8.1 DEFINIÇÃO DO CASO DE ESTUDO 81

8.2 PROPRIEDADES E PARÂMETROS GEOMÉTRICOS DOS COMPONENTES 82

8.3 MATERIAIS DO MODELO 83

8.4 ANÁLISE REALIZADA ATRAVÉS DO *SOFTWARE* ABAQUS 85

8.4.1 Características da Implementação *86*
8.4.2 Tipos de Elementos e Características das Malhas de Elementos *86*
8.4.3 Implementação do Contato *89*
8.4.4 Condições de Contorno *90*
8.4.5 Análise de Plastificação *91*
8.4.6 Resultados *95*
8.5 ANÁLISE REALIZADA ATRAVÉS DO *SOFTWARE* UFCAD 96
8.5.1 Características da implementação *96*
8.5.2 Resultados *97*
8.6 BENCHMARKING: ABAQUS X UFCAD 99
8.6.1 Facilidade de implementação *99*
8.6.2 Qualidade dos resultados *99*
8.6.3 Tempo e custo de simulação *102*
8.6.4 Pós-processamento dos dados *103*
8.6.5 Resumo da Análise de Benchmarking *103*
9. CONCLUSÕES DO TRABALHO 105
10. REFERÊNCIAS 107

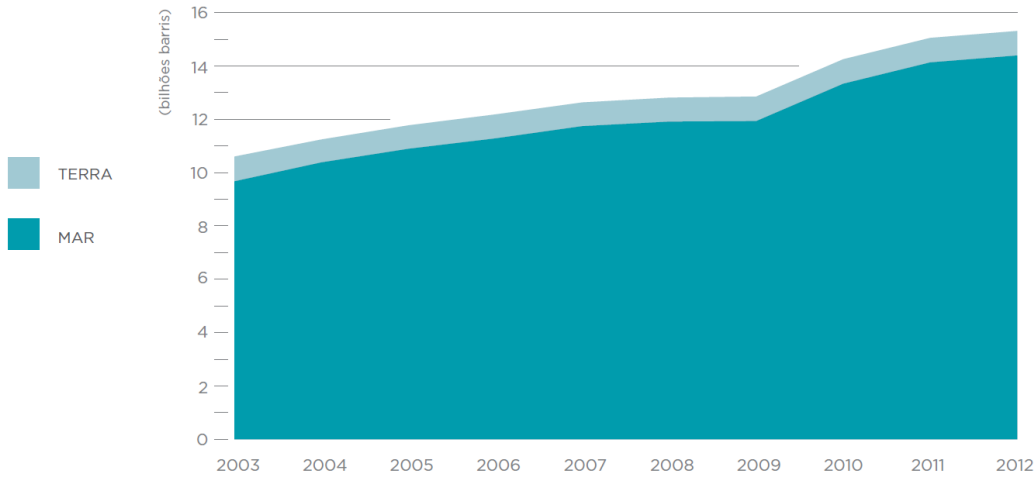
1 INTRODUÇÃO

1.1 Motivação

Presente direta e indiretamente na vida de todos, o petróleo é mais do que uma importante fonte de energia. É um recurso de grande impacto na economia global, altamente estratégico, e que por isso influencia muitas das decisões políticas.

Por ser não-renovável, novas fontes de petróleo precisam ser descobertas para a manutenção ou ampliação da demanda atual. Com o contínuo esgotamento das reservas de fácil exploração, restam as reservas de difícil acesso ou extração, o que exige o desenvolvimento de novas tecnologias de exploração para que as limitações e os desafios sejam superados.

No Brasil, a maior parte das reservas de petróleo encontram-se em mar, como mostram os dados da ANP na Figura 1, o que justifica a produção *offshore* de petróleo em larga escala, caracterizada pela exigência de componentes de elevado nível tecnológico, como plataformas petrolíferas, elementos de ligação e mecanismos de controle.



FONTE: ANP/SDP (Tabela 2.4).
NOTAS: 1. Reservas em 31/12 dos anos de referência.
2. Inclui condensado.
3. Ver em Notas Gerais item sobre "Reservas Brasileiras de Petróleo e Gás Natural".

Figura 1 – Evolução das reservas provadas de petróleo, por localização (terra e mar) – 2003-2012.
Fonte: *(AGÊNCIA NACIONAL DO PETRÓLEO, 2013)*.

Os elementos de ligação, responsáveis por conectar os poços às plataformas e navios, exercem um papel essencial ao processo de produção de petróleo *offshore*, como ilustra a Figura 2. Estes elementos subdividem-se em: tubos flexíveis (Figura 3), cabos umbilicais (Figura 4) e linhas de ancoragem.

Figura 2 - Elementos de ligação são responsáveis por interligar as plataformas às unidades de extração presentes no leito do oceano. Fonte: *(2B1st Consulting, 2014)*.

O projeto destes elementos de ligação é uma atividade de alta complexidade, devido às:

- condições adversas de operação destes elementos, em geral grandes profundidades e sobre o efeito de fenômenos naturais, como correntes marítimas;
- elevado número de componentes, como camadas, mangueiras e armaduras, que irão se interagir uns com os outros durante a operação;
- elevada quantidade de requisitos de funcionalidade e de critérios de projeto.

Figura 3 – Tubo flexível para aplicações *offshore*. Fonte: PETROLEO ENERGIA.

Figura 4 – Exemplo de um cabo umbilical para aplicações *offshore*. Fonte: *(VALLOUREC, 2014)*

As dificuldades intrínsecas ao projeto de tubos flexíveis e cabos umbilicais e também os prejuízos gerados por falhas destes componentes, tanto econômicos quanto ambientais, têm levado à elaboração de novas técnicas de engenharia e também ao desenvolvimento de ferramentas que auxiliem a execução desta tarefa. Seguindo esta linha, (PROVASI, 2013) desenvolveu em seu trabalho de doutorado uma ferramenta de análise direcionada ao projeto de tubos flexíveis e cabos umbilicais, conhecida por UFCad, que implementa uma formulação própria de macroelementos finitos.

1.2 Objetivos

(PROVASI, 2013) priorizou a formulação e validação dos modelos macroelementos finitos, deixando para uma etapa posterior a operacionalização do programa.

Portanto, este projeto de conclusão de curso consiste em uma extensão dos trabalhos realizados em "*Contribuição ao Projeto de Cabos Umbilicais e Tubos Flexíveis: Ferramentas de CAD e Modelo de Macro Elementos*" (PROVASI, 2013), tendo como objetivo o desenvolvimento de uma ferramenta eficiente para análise estrutural de tubos flexíveis, com o intuito de viabilizar a sua utilização em computadores convencionais e em aplicações práticas na indústria. Para que isto seja possível, os gargalos computacionais desta ferramenta devem ser identificados e devem ser executadas ações que resultem em redução do tempo de análise e do consumo de memória e processamento. Deseja-se também a validação de um caso de estudo, comparando a ferramenta desenvolvida com um software consolidado de elementos finitos, com o objetivo de garantir a confiabilidade dos resultados e de se realizar um *benchmarking* entre ambos os programas.

1.3 Estrutura do Trabalho

Este trabalho consiste em 10 capítulos, sendo o primeiro a introdução deste trabalho, e que visa contextualizar este trabalho e deixar claro a motivação para a realização do mesmo e seus respectivos objetivos.

O capítulo 2 consiste em uma revisão bibliográfica dos macroelementos finitos desenvolvidos em (PROVASI, 2013), pois a compreensão destes elementos é de fundamental importância para o entendimento do funcionamento de uma ferramenta de análise que os implemente.

O capítulo 3 traz uma formulação corrigida para o elemento de contato entre o elemento cilíndrico e o elemento de hélice, pois foi encontrada, durante a etapa de revisão bibliográfica, uma falha na formulação original.

Além disso, foi identificada a inexistência de um modelo de forças para a expansão em série de Fourier das reações de contato, o que motivou a formulação de um modelo adequado para tal situação, que encontra-se no capítulo 4.

O capítulo 5 consiste na importante etapa de identificação dos gargalos computacionais, que compuseram a base da estratégia adotada para aumentar a eficiência do programa. Para esta etapa foram utilizados *profilers*, que são ferramentas projetadas para a avaliação de outros programas.

O capítulo 6 traz as modificações realizadas na implementação da ferramenta de análise que resultaram em ganhos de performance e no funcionamento adequado do

programa. As modificações realizadas no cálculo da matriz de rigidez, por exemplo, representaram uma redução de 10 a 20 no tempo de execução desta tarefa.

O capítulo 7 foi dedicado exclusivamente à etapa de resolução do sistema linear, devido à importância que ela representa frente ao funcionamento do programa. Foram testados diversos métodos de resolução de sistemas lineares, inclusive com a implementação de uma biblioteca própria. Apesar disso, alguns problemas e limitações são encontrados, o que será abordado em mais detalhes neste capítulo.

O capítulo 8 consiste na validação de um caso de estudo com a ferramenta de análise, comparando-a com um programa profissional de elementos finitos, o Abaqus. Neste capítulo é apresentado o caso de estudo e são detalhadas as principais etapas que levaram à obtenção dos resultados em ambos os programas. Por fim, os resultados são comparados e realiza-se um *benchmarking* com base em critérios definidos.

No capítulo 9 encontram-se as conclusões deste trabalho e também propostas de trabalhos futuros.

Por fim, no capítulo 10, são listadas as referências nas quais este trabalho se baseou.

2. REVISÃO BIBLIOGRÁFICA

Para o projeto adequado de tubos flexíveis, deve-se conhecer as suas principais características, componentes e condições de operação. Por este motivo, será realizada no início deste capítulo uma breve revisão a respeito deste assunto.

Com a finalidade de tornar eficiente uma ferramenta de análise que implemente macroelementos finitos formulados especificamente para o projeto de tubos flexíveis, é necessário conhecer as características destes elementos, suas hipóteses e suas formulações matemáticas. Por este motivo, realizou-se uma ampla revisão bibliográfica a respeito dos macroelementos finitos formulados por (PROVASI, 2013), que será apresentada a partir do item 2.3.

2.1 Tubos Flexíveis e Cabos Umbilicais

"*Cabos umbilicais são elementos que têm a finalidade de interligar uma unidade flutuante a um poço submerso, executando diversas funções, dentre as quais: controle elétrico e/ou hidráulico, transmissão de sinais elétricos e/ou óticos, transmissão de energia elétrica para bombeamento submerso ou injeção de fluidos no poço.*"

"Tubos, assim como cabos umbilicais, interligam a unidade flutuante ao poço produtor, porém têm como finalidade levar petróleo e gás do poço para a plataforma, injetar fluidos para melhorar a produtividade do poço (como por exemplo, a injeção de gás retirado do próprio poço, processo esse conhecido como Gas Lift), levar o óleo até refinarias ou estações de armazenamento (esses denominados de exportação), além de executar funções de perfuração, entre outras. Para cada uma das funções descritas, existem tipos específicos. Esses tubos são denominados risers quando se encontram suspensos e flowlines quando estão apoiados sobre o solo." (PROVASI, 2013).

Tubos flexíveis são caracterizados por uma elevada quantidade de camadas e componentes, como ilustra a Figura 5. Esses tubos podem conter os seguintes elementos:

- Carcaça intertravada;
- Camadas plásticas;
- Camada circunferencial de pressão;
- Armadura de tração helicoidal;

- Fitas;
- Fillers;
- Núcleo elétrico;
- Mangueiras hidráulicas;
- Núcleo eletro-hidráulico.

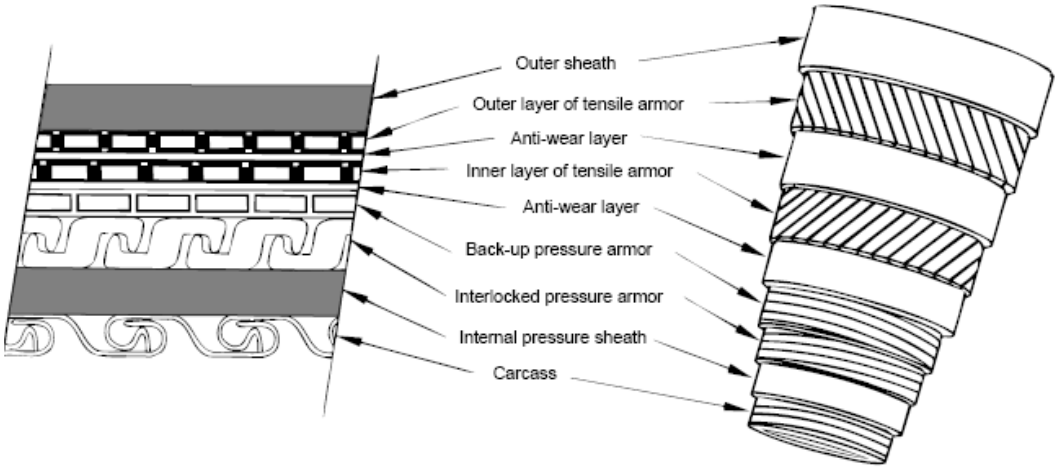


Figura 5 – Camadas de um tubo do tipo "Rough Bore Reinforced". Fonte: API RP 17B.

A escolha da aplicação destes componentes depende de uma série de fatores, como o tipo do tubo (tubo flexível ou cabo umbilical), das especificações e requisitos de projeto que o ele deve atender.

2.2 Abordagem ao problema

O projeto de tubos flexíveis e cabos umbilicais é uma tarefa complexa. Modelos analíticos apresentam muitas limitações e são, em alguns casos, impossíveis de serem realizados. Modelagens numéricas mostram-se uma importante ferramenta de projeto, em especial as que utilizam o Método dos Elementos Finitos.

S*oftwares* convencionais de elementos finitos, como *ANSYS* e *Abaqus*, são muito genéricos, pois visam resolver os mais variados tipos de problema. Assim, ao longo das últimas décadas, modelos específicos têm sido formulados para facilitar o projeto e análise de tubos flexíveis e cabos umbilicais.

(PROVASI, 2013) desenvolveu uma ferramenta de CAD direcionada com a finalidade de tornar o projeto de tubos flexíveis e cabos umbilicais uma tarefa mais simples.

2.3 Macroelementos Finitos

PROVASI (2013) criou elementos próprios, utilizando-se para isso uma formulação de macroelementos finitos, o que facilita a inclusão, a customização dos modelos e a solução numérica destes problemas. Em seu trabalho, foram contemplados os seguintes elementos:

- Elemento Cilíndrico de Parede Espessa;
- Elemento de Hélice;
- Elemento de Ligação “Bridge”;
- Elementos de Contato.

que serão detalhados a seguir.

2.3.1 Elemento Cilíndrico de Parede Espessa

É um elemento que pode ser utilizado para a modelagem de capas plásticas e, em uma abordagem inicial simplificada, para a modelagem de camadas equivalentes, pois este elemento possui uma formulação de material ortotrópico.

(PROVASI & MARTINS, 2013-c) desenvolveram uma extensão do modelo de um elemento cilíndrico de parede espessa formulado por COOK. O modelo desenvolvido pelos autores tem como objetivo determinar os deslocamentos de uma capa cilíndrica sobre quaisquer tipos de carregamentos.

Adotou-se a hipótese de material elástico e linear, e adotou-se um sistema de coordenadas cilíndrico, de acordo com a Figura 6, sendo:

- u – o deslocamento na direção radial
- v – o deslocamento na direção circunferencial
- w – o deslocamento na direção axial

Neste modelo, a expansão em Série de Fourier foi utilizada para descrever os deslocamentos e também os carregamentos atuantes no elemento (forças e momentos).

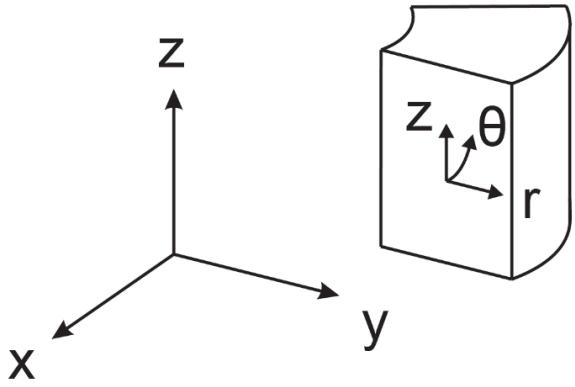


Figura 6 – Sistema de coordenadas utilizado na formulação do elemento cilíndrico de parede espessa. Fonte: *(PROVASI & MARTINS, 2013-c)*.

Através da formulação de elemento infinitesimal cilíndrico, mostrado na Figura 7, determina-se as equações para o equilíbrio das tensões. Utilizando o sistema de coordenadas cilíndrico, calcula-se as relações entre deformação de deslocamentos. Após uma extensa manipulação matemática e utilizando outros conceitos de mecânica dos meios contínuos, obtêm-se a matriz de rigidez do elemento.

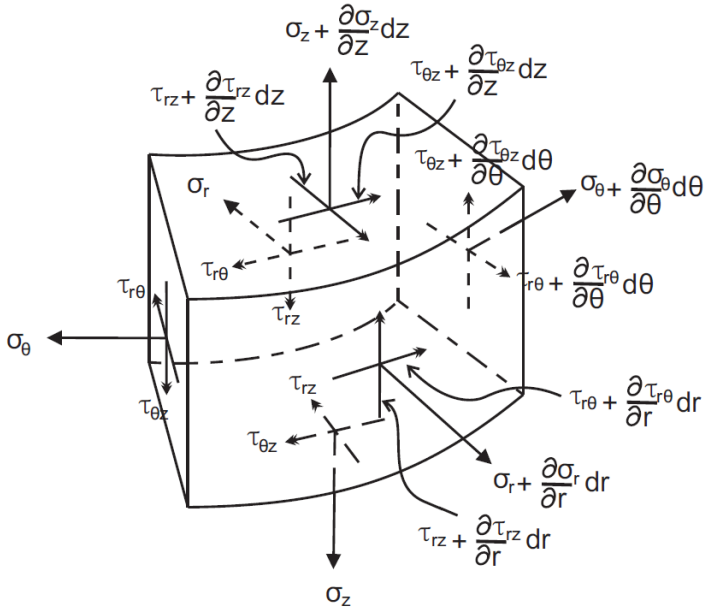


Figura 7 – Elemento infinitesimal cilíndrico. Fonte: *(PROVASI & MARTINS, 2013-c)*.

Como mostram a Figura 8 e a Figura 9, (PROVASI & MARTINS, 2013-c) compararam os resultados obtidos pela sua formulação com os obtidos através do software *Ansys*.

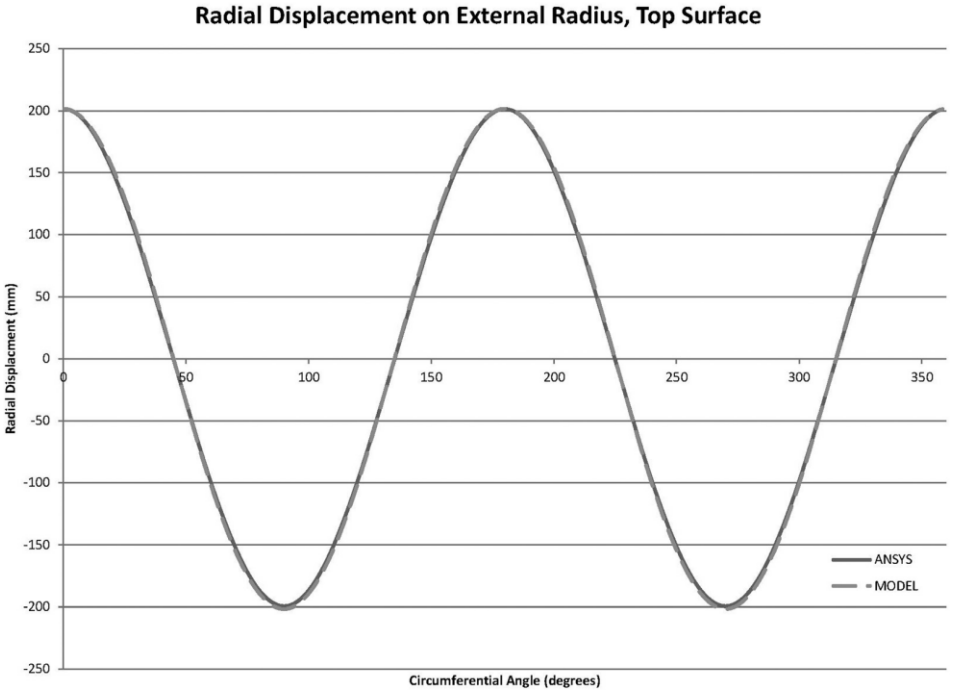


Figura 8 – Comparação de deslocamento radial para a superfície superior. Fonte: (PROVASI & MARTINS, 2013-c).

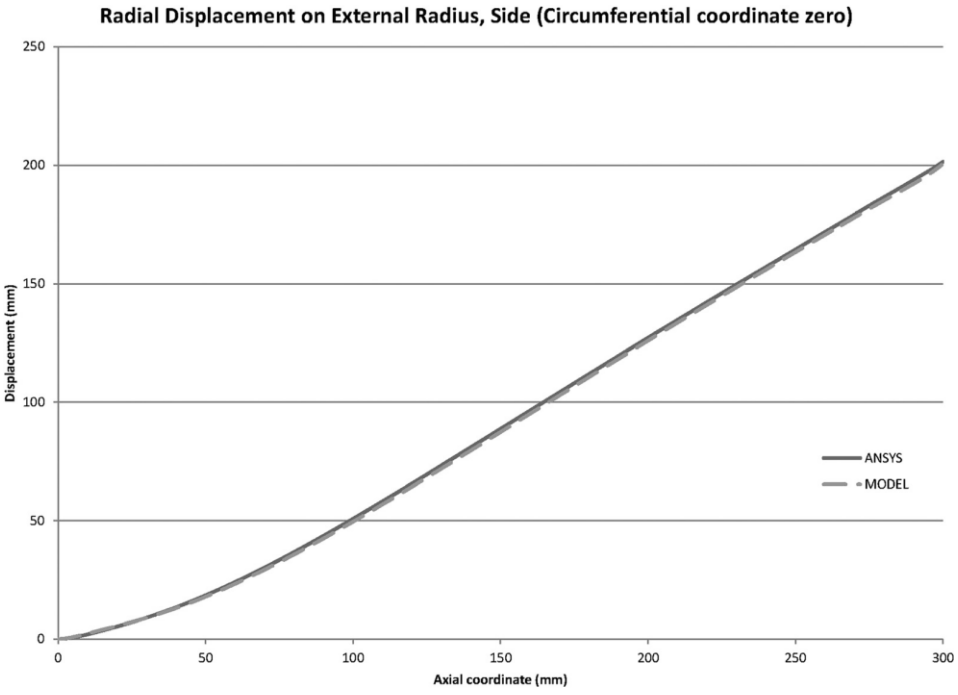


Figura 9 – Comparação de deslocamentos radiais para a superfície de capa. Fonte: (PROVASI & MARTINS, 2013-c)

2.3.2 Elemento de Hélice (ou Tendão)

Baseado na formulação de ZHU e MEGUID, (PROVASI & MARTINS, 2011) desenvolveram um elemento tridimensional para helicoides. Por ser um elemento de

viga que leva em consideração os efeitos da curvatura e tortuosidade, este elemento é aplicável a uma vasta série de problemas, como por exemplo a modelagem dos tendões da armadura de tração.

Para a formulação deste elemento, foram utilizadas as seguintes hipóteses:

- Pequenas deformações e deslocamentos;
- Não ocorre empenamento na seção transversal;
- Material elástico e isotrópico.

Como indicado Figura 10, este elemento possui um sistema de coordenadas local, um triedro de Frenet, que segue a nomenclatura utilizada por ZHU e MEGUID, na qual: $Z \equiv \vec{t}$ é o vetor tangente à curva, $X \equiv \vec{n}$ é o vetor normal, e $Y \equiv \vec{b}$ é o vetor binormal.

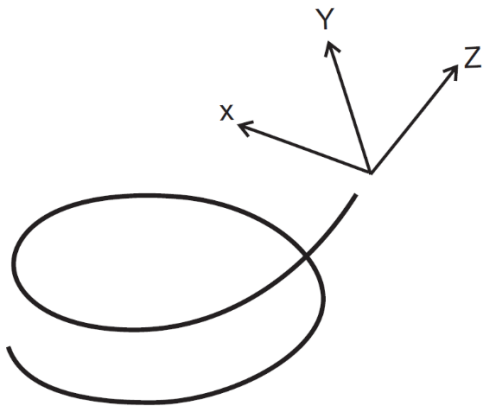


Figura 10 – Esquematização do elemento e sistema de coordenas local associado. Fonte: PROVASI (2013).

No entanto, para integrar o Elemento de Hélice aos demais elementos, que foram formulados em um sistema de coordenadas cilíndricas, é necessária uma rotação do sistema de coordenadas. Assim, além da matriz de rigidez do Elemento de Hélice, será apresentada também a matriz de transformação de coordenadas, e como descrever a matriz de rigidez para o sistema cilíndrico.

As relações de deformação e deslocamento e que incluem os efeitos de curvatura e tortuosidade foram obtidas de (LOVE, 1944):

$$\varepsilon_z = \frac{\partial w(s)}{\partial s} - k\, u(s)$$

$$\omega_x = \frac{\varphi_x(s)}{\partial s} - \tau\varphi_y(s) + k\, \varphi_z(s)$$

$$\omega_y = \frac{\varphi_y(s)}{\partial s} + \tau\varphi_x(s)$$

$$\omega_z = \frac{\varphi_z(s)}{\partial s} - k\varphi_x(s)$$

onde:

- ε_z – é a deformação axial;
- ω_x , ω_y e ω_z – são as distorções angulares ao redor dos eixos x, y e z respectivamente;
- u_x , u_y e u_z – são os deslocamentos nas direções x, y e z respectivamente;
- τ – é a tortuosidade inicial;
- k – é a curvatura inicial.

Baseado nas hipóteses adotadas, os autores formularam o descolamento na direção normal e binormal através de polinômios de quinta ordem.

$$u(s) = \sum_{i=0}^{5} a_i \, s^i$$

$$v(s) = \sum_{i=0}^{5} b_i \, s^i$$

Os deslocamentos u_z e φ_z são determinados pelas duas seguintes expressões, nas quais se obtêm os coeficientes a_6 , a_7 , a_8 e b_6 , b_7 , b_8.

$$u_z = \int (\varepsilon_z + k \, u_x) \, dS$$

$$\varphi_z = \int \left(\omega_z - k \, \frac{\partial u_y}{\partial s} - k \, \tau \, u_x \right) dS$$

Das relações de (LOVE, 1944), obtêm-se as seguintes relações:

$$\varphi_x = - \frac{\partial u_y}{\partial s} - \tau \, u_x$$

$$\varphi_y = \frac{\partial u_x}{\partial s} - \tau \, u_y + k u_z$$

(PROVASI & MARTINS, 2011) utilizaram um elemento de três nós, a partir do qual foi possível determinar todas as constantes necessárias. Com isso, formulou-se uma relação entre os deslocamentos nodais e as constates de integração dada por:

$$\boldsymbol{q = C \, u_{nodal}}$$

Onde:

- $\boldsymbol{q^T} = [a_0 \quad ... \quad a_8 \quad b_0 \quad ... \quad b_8]$ – é o vetor das constantes de integração;
- $\boldsymbol{u_{nodal}^T} = [u_{nodal}^1 \, u_{nodal}^2 \, u_{nodal}^3]$ – é o vetor dos deslocamentos nodais;

- $u^{i}_{nodal} = [u_{xi} \quad u_{yi} \quad u_{zi} \quad \varphi_{xi} \quad \varphi_{xi} \quad \varphi_{xi}]$, com $i = 1, \dots, 3$;
- $\boldsymbol{C} = \begin{bmatrix} C_1 & C_2 & C_3 \\ C_4 & C_5 & C_6 \\ C_7 & C_8 & C_9 \end{bmatrix}$ – cada termo C_i é uma matriz e pode ser encontrado em (PROVASI e MARTINS, 2011).

Os deslocamentos $u^T = [u_{xi} \quad u_{yi} \quad u_{zi} \quad \varphi_{xi} \quad \varphi_{xi} \quad \varphi_{xi}]$ foram expressos em função dos deslocamentos nodais por meio da expressão:

$$\boldsymbol{u = Aq = ACu_{nodal}}$$

$$\boldsymbol{A} = [A_1 \; A_2 \; A_3]$$

Assim, a matriz de rigidez do Elemento de Hélice pode ser calculada por:

$$K_{el} = \; C^T B^T H \; B \; C$$

com:

$$H = \begin{bmatrix} EA & 0 & 0 & 0 \\ 0 & EI_x & 0 & 0 \\ 0 & 0 & EI_y & 0 \\ 0 & 0 & 0 & GJ \end{bmatrix}$$

$$B = [B_1 \quad B_2 \quad B_3]$$

Os termos A_i e B_i estão definidos em (PROVASI & MARTINS, 2011). Para descrever as relações no sistema cilíndrico de coordenadas, fez-se necessário utilizar a seguinte matriz de transformação entre sistemas:

$$T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\cos\alpha & \sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

Onde α é o ângulo de assentamento de hélice.

E a matriz de rigidez para o Elemento de Hélice no sistemas de coordenadas cilíndricos é determinada por:

$$K = \; \mathbb{T}^T \; K_{el} \mathbb{T}$$

$\mathbb{T}$ é uma matriz diagonal, em que cada termo da diagonal é composto pela matriz T.

2.3.3 Elemento de Ligação “Bridge”

O principal objetivo do Elemento de ligação do tipo “bridge” é unir dois nós de diferentes tipos e formulações de maneira rígida, ou seja, sem deslocamento relativo entre eles.

A Figura 11 mostra um exemplo de aplicação deste elemento, em que ocorre uma ligação entre um elemento cilíndrico de parede espessa (que utiliza a formulação em Série de Fourier) e o elemento de hélice (que apresenta formulação convencional).

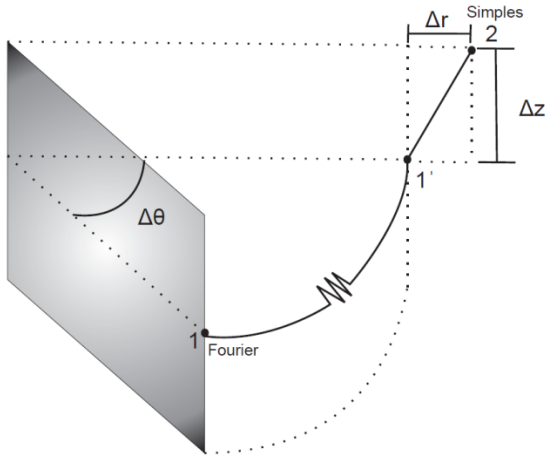


Figura 11 – Elemento de ligação tipo "bridge". Fonte: (PROVASI, 2013).

A formulação completa para este elemento encontra-se em (PROVASI & MARTINS, 2013-a). É interessante ressaltar que, com base na Figura 11, a principal condição *que rege esse elemento é a inexistência de deslocamento relativo entre os nós, dada por:*

$$u_2 - u_1' = 0$$

Utilizando uma formulação através do método das penalidades, os autores obtiveram a matriz de rigidez do elemento de ligação do tipo “bridge”.

2.3.4 Elemento de Contato

A necessidade de um elemento de contato justifica-se pela existência de vazios (“gaps”) entre os elementos e também pelas aplicações de pressão de contato, que podem envolver ou não a presença de atrito.

Como mostra a Figura 12, ocorrerá contato entre dois corpos, quando a distância que os separa for nula. Na prática, o contato irá ocorrer quando essa distância atingir um valor menor ou igual à uma tolerância especificada.

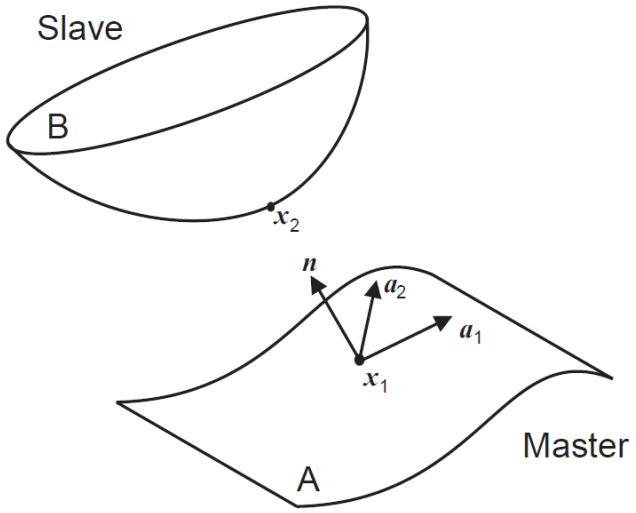


Figura 12 – Dois corpos em situação de pré-contato. Fonte: (PROVASI, 2013).

"*Essa figura (*Figura 12*) ainda exibe o sistema de eixos ortonormal local no contato: a direção normal à superfície* $\boldsymbol{n}$ *e as direções tangentes à superfície* $\boldsymbol{a_1}$ *e* $\boldsymbol{a_2}$*. Observa-se também que são utilizadas duas superfícies: uma denominada máster (mestre) e outra denominada slave (escrava). Normalmente isto é feito para facilitar a descrição de cada uma delas, já que a superfície slave deve ser escolhida de tal forma que possua o maior número de elementos entre as duas. Isso é necessário para que os contatos sejam corretamente detectados após as discretizações necessárias nos métodos matemáticos a serem empregados.*" (PROVASI, 2013).

Se a distância entre as duas superfícies for negativa e de módulo maior que a tolerância especificada, ocorrerá a penetração da superfície *master* sobre a *slave*. A discretização da superfície está associada a qualidade do contato. Caso a superfície seja pouco discretizada, poderá ocorrer penetração, sem que esse fenômeno seja identificado pelos critérios de verificação adotados, o que está indicado na Figura 13. Ao se aumentar o grau de discretização, o problema é resolvido, porém o tempo de simulação também irá aumentar. Deve-se portanto escolher um grau de discretização que forneça níveis aceitáveis de penetração e ao mesmo tempo não inviabilize o tempo de simulação.

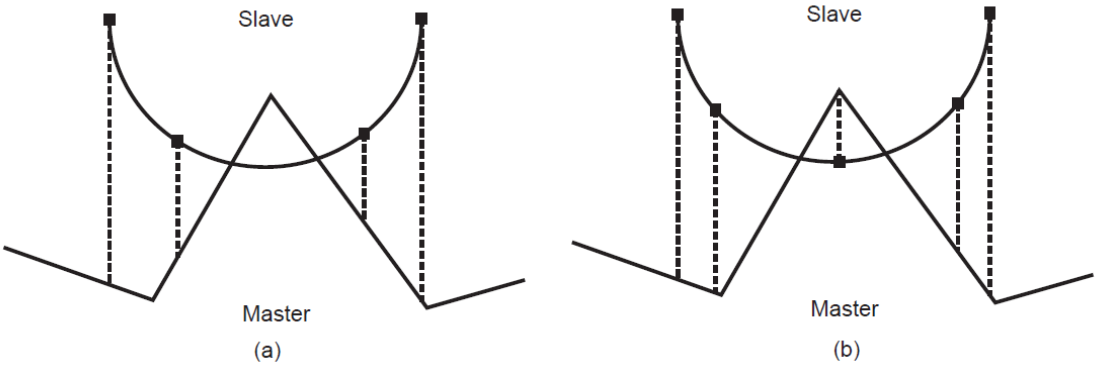


Figura 13 – Exemplos do efeito da discretização do contato: (a) pouco discretizada; (b) discretizada adequadamente. Fonte: (PROVASI, 2013).

A função vazio (ou “gap”) é definida pela seguinte fórmula:

$$g_N = (\boldsymbol{x_1} - \boldsymbol{x_2}) \,.\, \boldsymbol{n}$$

onde $\boldsymbol{x_1}$ e $\boldsymbol{x_2}$ são os pontos das superfícies master e slave, e $\boldsymbol{n}$ é a direção normal. Se o gap for nulo, poderá haver uma pressão de contato que varie de zero à infinito, de modo que a expressão que descreve a condição de contato é dada por:

$$g_N \,.\, p_N = 0$$

“*A Figura 14 mostra o comportamento descrito pela equação acima. Porém a modelagem numérica é complicada dada a não-diferenciabilidade da curva. Para contornar tal problema, adotam-se leis que descrevam o comportamento de maneira mais suave (as chamadas Compliance Laws). O item (b) da Figura 14 exibe um exemplo que varia com uma potência do gap normal. Já o item (c) exibe um comportamento de variação linear com o gap. Este último é conhecido como método das penalidades, no qual varia-se o parâmetro de proporcionalidade tornando o comportamento dessa lei o mais próximo do real quanto se queira. A desvantagem dessa abordagem é que um valor alto desse parâmetro pode comprometer a convergência. Deve-se ressaltar que um valor muito pequeno também compromete o resultado, uma vez que faz com que a curva utilizada encontre-se longe do real comportamento do problema.*” (PROVASI, 2013).

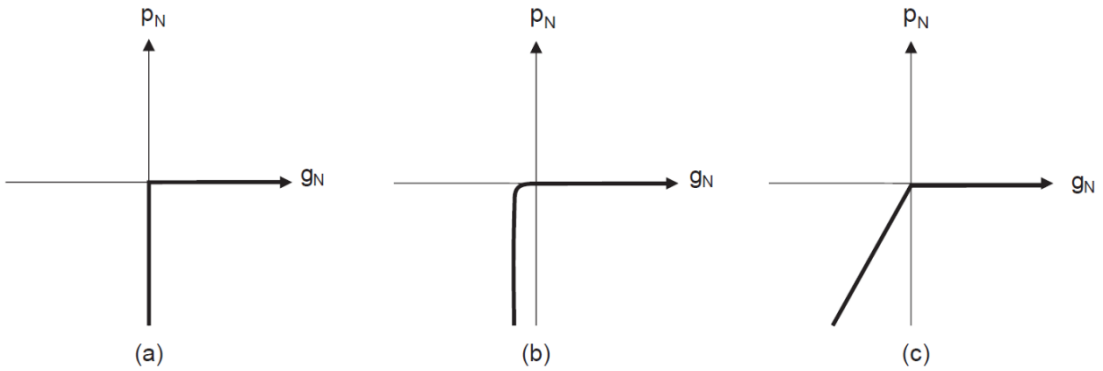


Figura 14 – Compliance Law para o contato normal: (a) comportamento ideal; (b) comportamento usando uma Compliance Law; (c) comportamento usando o método das penalidades. Fonte: (PROVASI, 2013).

(PROVASI, 2013) considerou ainda o caso de contato tangencial, o qual inclui os efeitos do atrito, e que subdivide-se em duas situações, exibidas na Figura 15:

- *sliding* – quando há movimento relativo entre as partes;
- *sticking* – quando não há movimento relativo entre as partes.

Figura 15 – Primeiro caso, configuração inicial. Bloco central, é uma situação com sticking. Bloco a direita, situação com sliding. Fonte: (PROVASI, 2013).

A formulação matemática completa, incluindo as matrizes de rigidez, para a situação de contato considerando *sticking* e para a situação de contato considerando *sliding* pode ser encontrada em (PROVASI, 2013).

3. FORMULAÇÃO CORRIGIDA PARA O MACROELEMENTO FINITO DE CONTATO DO TIPO "NÓ-A-NÓ" ENTRE ELEMENTO CILÍNDRICO E ELEMENTO DE HÉLICE

Durante a revisão bibliográfica do macroelemento finito de contato para nós que utilizam formulações distintas de deslocamento, (PROVASI & MARTINS, 2013-b), identificou-se uma inconsistência no desenvolvimento matemático do modelo. Portanto, o objetivo deste capítulo é apresentar esta inconsistência e propor uma formulação corrigida, que serão realizados no item 3.2.

Por questões lógicas, será apresentada a formulação original no item 3.1, para que o leitor deste trabalho se familiarize com a formulação do elemento em questão e com a notação empregada.

3.1 Formulação original de macroelemento finito de contato proposta por (PROVASI & MARTINS, 2013-b)

Considerando-se a situação ilustrada na Figura 16, com um elemento cilíndrico de parede espessa envolto por um elemento de hélice, tem-se:

- Nó 1 (nó de Fourier): com três graus de liberdade (u, v e w) e os deslocamentos expandidos em série de Fourier.
- Nó 2 (nó de hélice): com três graus de liberdade (u_x, u_y e u_z), pois as rotações foram ignoradas.

Para cada par de nós que podem vir a entrar ou que já estão em contato, conhece-se previamente os seguintes parâmetros:

- θ_0 – o ângulo, medido no sistema cilíndrico de coordenadas, da região de contato;
- P_1 e P_2 – pontos que representam as coordenadas dos nós em questão;
- $\vec{\boldsymbol{n}}_{\boldsymbol{1}}$ – a normal da superfície no ponto P_1;
- $\vec{\boldsymbol{a}}_{\boldsymbol{1}}$ e $\vec{\boldsymbol{a}}_{\boldsymbol{2}}$ – as direções tangenciais da superfície no ponto P_1.

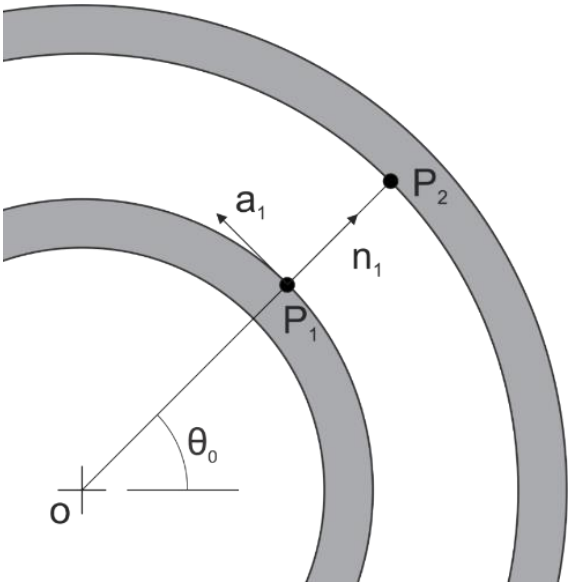


Figura 16 - Contato nó a nó: ponto 1 faz parte de um elemento cilíndrico e tem seu deslocamento representado através da expansão em Série de Fourier; o ponto 2 faz parte de um elemento de hélice.

Definem-se os vetores:

$$\overline{OP_1} = \boldsymbol{X}_1^r \quad \text{e} \quad \overline{OP_1'} = \boldsymbol{x}_1$$
$$\overline{OP_2} = \boldsymbol{X}_2^r \quad \text{e} \quad \overline{OP_2'} = \boldsymbol{x}_2$$

onde:

- $\boldsymbol{X}_1$ – é a posição do nó 1 na configuração inicial (não deformada)
- $\boldsymbol{x}_1$ – é a posição do nó 1 na configuração final (deformada)
- $\boldsymbol{X}_2$ – é a posição do nó 2 na configuração inicial (não deformada)
- $\boldsymbol{x}_2$ – é a posição do nó 1 na configuração final (deformada)

Os deslocamentos dos dois nós são expressos por:

$$\boldsymbol{u_1} = \boldsymbol{x}_1 - \boldsymbol{X}_1 = \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \end{Bmatrix} = \begin{Bmatrix} \sum_{i=0}^{n} \bar{u}_1^i \cos i\theta_0 + \sum_{i=0}^{n} \bar{\bar{u}}_1^i \sin i\theta_0 \\ \sum_{i=0}^{n} \bar{v}_1^i \sin i\theta_0 - \sum_{i=0}^{n} \bar{\bar{v}}_1^i \cos i\theta_0 \\ \sum_{i=0}^{n} \bar{w}_1^i \cos i\theta_0 + \sum_{i=0}^{n} \bar{\bar{w}}_1^i \sin i\theta_0 \end{Bmatrix}$$

$$\boldsymbol{u_2} = \boldsymbol{x}_2 - \boldsymbol{X}_2 = \begin{Bmatrix} u_2 \\ v_2 \\ w_2 \end{Bmatrix}$$

Com o objetivo de facilitar a manipulação algébrica do deslocamento do nó 1 (Fourier), retira-se da somatória o termo de ordem 0. Portanto:

$$\boldsymbol{u_1} = \begin{Bmatrix} \bar{u}_1^0 + \sum_{i=1}^{n} \bar{u}_1^i \cos i\theta_0 + \sum_{i=1}^{n} \bar{\bar{u}}_1^i \sin i\theta_0 \\ -\bar{\bar{v}}_1^0 + \sum_{i=1}^{n} \bar{v}_1^i \sin i\theta_0 - \sum_{i=1}^{n} \bar{\bar{v}}_1^i \cos i\theta_0 \\ \bar{w}_1^0 + \sum_{i=1}^{n} \bar{w}_1^i \cos i\theta_0 + \sum_{i=1}^{n} \bar{\bar{w}}_1^i \sin i\theta_0 \end{Bmatrix}$$

Os deslocamentos podem ser reescritos utilizando-se notação matricial:

$$\boldsymbol{u_1} = \boldsymbol{u_1^0} + \sum_{i=1}^{n} \left[C_i \boldsymbol{\bar{u}_1^i} + S_i \boldsymbol{\bar{\bar{u}}_1^i} \right]$$

Onde:

- $\boldsymbol{u_1} = \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \end{Bmatrix}$
- $\boldsymbol{u_1^0} = \begin{Bmatrix} \bar{u}_1^0 \\ -\bar{\bar{v}}_1^0 \\ \bar{w}_1^0 \end{Bmatrix}$, $\boldsymbol{\bar{u}_1^i} = \begin{Bmatrix} \bar{u}_1^i \\ \bar{v}_1^i \\ \bar{w}_1^i \end{Bmatrix}$ e $\boldsymbol{\bar{\bar{u}}_1^i} = \begin{Bmatrix} \bar{\bar{u}}_1^i \\ \bar{\bar{v}}_1^i \\ \bar{\bar{w}}_1^i \end{Bmatrix}$
- $C_i = \begin{bmatrix} \cos i\theta_0 & 0 & 0 \\ 0 & \sin i\theta_0 & 0 \\ 0 & 0 & \cos i\theta_0 \end{bmatrix}$
- $S_i = \begin{bmatrix} \sin i\theta_0 & 0 & 0 \\ 0 & -\cos i\theta_0 & 0 \\ 0 & 0 & \sin i\theta_0 \end{bmatrix}$

Define-se a função *gap* normal:

$$g_N = (\boldsymbol{x}_2 - \boldsymbol{x}_1) \,.\, \vec{\boldsymbol{n}}_1$$

E também a função gap tangencial:

$$g_T = g_{T_1} \vec{\boldsymbol{a}}_1 + g_{T_2} \vec{\boldsymbol{a}}_2$$

Onde:

$$g_{T_\alpha} = (\boldsymbol{x}_2 - \boldsymbol{x}_1) . \vec{\boldsymbol{a}}_{\boldsymbol{\alpha}} \quad \text{para } \alpha = 1, 2$$

Com a aplicação do princípio virtual, expressa-se o trabalho virtual dos esforços por:

$$\delta W_{contato} = \varepsilon_N g_N \,.\, \delta g_N + \varepsilon_T \boldsymbol{g_T} \,.\, \boldsymbol{\delta g_T}$$

Derivando-se a expressão do trabalho virtual obtêm-se a expressão:

$$\delta(\delta W_{contato}) = \varepsilon_N \delta g_N \,.\, \delta g_N + \varepsilon_T \boldsymbol{\delta g_T} \,.\, \boldsymbol{\delta g_T}$$

Reescrevendo o problema na forma matricial, obtêm-se:

$$\delta(\delta W_{contato}) = \varepsilon_N \delta g_N \delta g_N + \varepsilon_T \boldsymbol{\delta g_T}^{\boldsymbol{T}} \boldsymbol{\delta g_T}$$

Com:

$$\boldsymbol{\delta g_T} = \delta g_{T_1}\, \boldsymbol{a_1} + \delta g_{T_2}\, \boldsymbol{a_2},$$

Onde:

$$\boldsymbol{n} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} \quad , \quad \boldsymbol{a_1} = \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} \text{ e } \boldsymbol{a_2} = \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}$$

Portanto,

$$\delta g_N = \boldsymbol{n^T \delta x_2} - \boldsymbol{n^T \delta x_1}$$
$$\delta g_{T_1} = \boldsymbol{a_1^T \delta x_2} - \boldsymbol{a_1^T \delta x_1}$$
$$\delta g_{T_2} = \boldsymbol{a_2^T \delta x_2} - a_2^T \boldsymbol{\delta x_1}$$

(PROVASI & MARTINS, 2013-b) dizem, então, que $\boldsymbol{\delta x_i} = \boldsymbol{\delta u_i}$, o que lhes permite determinar as variações das funções gap (δg_N, δg_{T_1} e δg_{T_2}) em função das variações de deslocamentos nodais ($\boldsymbol{\delta u_i}$, $\boldsymbol{\delta u_i}$ e $\boldsymbol{\delta u_i}$). Com os valores de variações da função gap, determina-se a matriz de rigidez do elemento de contato por meio da expressão $\delta(\delta W_{contato}) = \varepsilon_N \delta g_N \delta g_N + \varepsilon_T \boldsymbol{\delta g_T}^T \boldsymbol{\delta g_T}$.

3.2 Identificação da inconsistência matemática e formulação corrigida

As relações entre a variação de deslocamento nodal e a variação de posição na configuração deformada estão definidas corretamente em (PROVASI & MARTINS, 2013-b) para os seguintes termos:

$$\boldsymbol{\delta u_2 = \delta x_2} \quad , \quad \boldsymbol{\delta \bar{u}_1^i = \delta \bar{x}_1^i} \quad , \quad \boldsymbol{\delta \bar{\bar{u}}_1^i = \delta \bar{\bar{x}}_1^i}$$

No entanto, esta relação é definida incorretamente para o termo de ordem zero da expansão em Série de Fourier, ou seja,

$$\boldsymbol{\delta x_1^0} \neq \boldsymbol{\delta u_1^0}$$

pois o segundo termo do vetor $\boldsymbol{u_1^0}$ é negativo.

Lembrando-se que $\boldsymbol{u_1^0} = [\bar{u}_1^0 \quad -\bar{\bar{v}}_1^0 \quad \bar{w}_1^0]^T$, onde $-\bar{\bar{v}}_1^0$ é o primeiro termo da somatória $-\sum_{i=0}^{n} \bar{\bar{v}}_1^i \cos i\theta_0$ (quando $i = 0$), propõe-se a realização da seguinte transformação:

$$\boldsymbol{u_1^0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} \bar{u}_1^0 \\ \bar{\bar{v}}_1^0 \\ \bar{w}_1^0 \end{Bmatrix} = B \begin{Bmatrix} \bar{u}_1^0 \\ \bar{\bar{v}}_1^0 \\ \bar{w}_1^0 \end{Bmatrix}$$

Portanto,

$$B\ \boldsymbol{\delta u_1^0} = \boldsymbol{\delta x_1^0}$$

Com:

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$(B = B^T \quad \text{e} \quad BB^T = I)$$

Utilizando-se a modificação proposta acima, as funções δg_N, δg_{T_1} e δg_{T_2} são calculadas por:

$$\delta g_N = \boldsymbol{n^T \delta u_2} - \boldsymbol{n^T} B \boldsymbol{\delta u_1^0} - \sum_{i=1}^{n} \left[\boldsymbol{n^T} C_i \boldsymbol{\delta \bar{u}_1^i} + \boldsymbol{n^T} S_i \boldsymbol{\delta \bar{\bar{u}}_1^i} \right]$$

$$\delta g_{T_1} = \boldsymbol{a_1^T \delta u_2} - \boldsymbol{a_1^T} B \boldsymbol{\delta u_1^0} - \sum_{i=1}^{n} \left[\boldsymbol{a_1^T} C_i \boldsymbol{\delta \bar{u}_1^i} + \boldsymbol{a_1^T} S_i \boldsymbol{\delta \bar{\bar{u}}_1^i} \right]$$

$$g_{T_2} = \boldsymbol{a_2^T \delta u_2} - \boldsymbol{a_2^T} B \boldsymbol{\delta u_1^0} - \sum_{i=1}^{n} \left[\boldsymbol{a_2^T} C_i \boldsymbol{\delta \bar{u}_1^i} + \boldsymbol{a_2^T} S_i \boldsymbol{\delta \bar{\bar{u}}_1^i} \right]$$

Com isso, obtêm-se a matriz de rigidez para o caso de contato com *sticking*:

$$K_{sticking} = \varepsilon_N M_n + \varepsilon_T M_{a1} + \varepsilon_T M_{a2}$$

com:

$$M_n = \begin{bmatrix} \boldsymbol{nn^T} & \boldsymbol{n} B^T \boldsymbol{n^T} C_1^T & \boldsymbol{n} B^T \boldsymbol{n^T} S_1^T & \dots & \boldsymbol{n} B^T \boldsymbol{n^T} C_n^T & \boldsymbol{n} B^T \boldsymbol{n^T} S_n^T & -\boldsymbol{n} B^T \boldsymbol{n^T} \\ C_1^T \boldsymbol{n} B \boldsymbol{n^T} & C_1^T \boldsymbol{nn^T} C_1^T & C_1^T \boldsymbol{nn^T} S_1^T & \dots & C_1^T \boldsymbol{nn^T} C_n^T & C_1^T \boldsymbol{nn^T} S_n^T & -C_1^T \boldsymbol{nn^T} \\ S_1^T \boldsymbol{n} B \boldsymbol{n^T} & S_1^T \boldsymbol{nn^T} C_1^T & S_1^T \boldsymbol{nn^T} S_1^T & \dots & S_1^T \boldsymbol{nn^T} C_n^T & S_1^T \boldsymbol{nn^T} S_n^T & -S_1^T \boldsymbol{nn^T} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ C_n^T \boldsymbol{n} B \boldsymbol{n^T} & C_n^T \boldsymbol{nn^T} C_1^T & C_n^T \boldsymbol{nn^T} S_1^T & \dots & C_n^T \boldsymbol{nn^T} C_n^T & C_n^T \boldsymbol{nn^T} S_n^T & -C_n^T \boldsymbol{nn^T} \\ S_n^T \boldsymbol{n} B \boldsymbol{n^T} & S_n^T \boldsymbol{nn^T} C_1^T & S_n^T \boldsymbol{nn^T} S_1^T & \dots & S_n^T \boldsymbol{nn^T} C_n^T & S_n^T \boldsymbol{nn^T} S_n^T & -S_n^T \boldsymbol{nn^T} \\ -\boldsymbol{n} B \boldsymbol{n^T} & -\boldsymbol{nn^T} C_1^T & -\boldsymbol{nn^T} S_1^T & \dots & -\boldsymbol{nn^T} C_n^T & -\boldsymbol{nn^T} S_n^T & \boldsymbol{nn^T} \end{bmatrix}$$

$$M_{a1} = \begin{bmatrix} \boldsymbol{a_1 a_1^T} & \boldsymbol{a_1} B^T \boldsymbol{a_1^T} & \boldsymbol{a_1} B^T \boldsymbol{a_1^T} S_1^T & \dots & \boldsymbol{a_1} B^T \boldsymbol{a_1^T} C_n^T & \boldsymbol{a_1} B^T \boldsymbol{a_1^T} S_n^T & -\boldsymbol{a_1} B^T \boldsymbol{a_1^T} \\ C_1^T \boldsymbol{a_1} B \boldsymbol{a_1^T} & C_1^T \boldsymbol{a_1 a_1^T} C_1^T & C_1^T \boldsymbol{a_1 a_1^T} S_1^T & \dots & C_1^T \boldsymbol{a_1 a_1^T} C_n^T & C_1^T \boldsymbol{a_1 a_1^T} S_n^T & -C_1^T \boldsymbol{a_1 a_1^T} \\ S_1^T \boldsymbol{a_1} B \boldsymbol{a_1^T} & S_1^T \boldsymbol{a_1 a_1^T} C_1^T & S_1^T \boldsymbol{a_1 a_1^T} S_1^T & \dots & S_1^T \boldsymbol{a_1 a_1^T} C_n^T & S_1^T \boldsymbol{a_1 a_1^T} S_n^T & -S_1^T \boldsymbol{a_1 a_1^T} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ C_n^T \boldsymbol{a_1} B \boldsymbol{a_1^T} & C_n^T \boldsymbol{a_1 a_1^T} C_1^T & C_n^T \boldsymbol{a_1 a_1^T} S_1^T & \dots & C_n^T \boldsymbol{a_1 a_1^T} C_n^T & C_n^T \boldsymbol{a_1 a_1^T} S_n^T & -C_n^T \boldsymbol{a_1 a_1^T} \\ S_n^T \boldsymbol{a_1} B \boldsymbol{a_1^T} & S_n^T \boldsymbol{a_1 a_1^T} C_1^T & S_n^T \boldsymbol{a_1 a_1^T} S_1^T & \dots & S_n^T \boldsymbol{a_1 a_1^T} C_n^T & S_n^T \boldsymbol{a_1 a_1^T} S_n^T & -S_n^T \boldsymbol{a_1 a_1^T} \\ -\boldsymbol{a_1} B \boldsymbol{a_1^T} & -\boldsymbol{a_1 a_1^T} C_1^T & -\boldsymbol{a_1 a_1^T} S_1^T & \dots & -\boldsymbol{a_1 a_1^T} C_n^T & -\boldsymbol{a_1 a_1^T} S_n^T & \boldsymbol{a_1 a_1^T} \end{bmatrix}$$

$$M_{a2} = \begin{bmatrix} \boldsymbol{a_2 a_2^T} & \boldsymbol{a_2 B^T a_2^T} & \boldsymbol{a_2 B^T a_2^T S_1^T} & \dots & \boldsymbol{a_2 B^T a_2^T C_n^T} & \boldsymbol{a_2 B^T a_2^T S_n^T} & \boldsymbol{-a_2 B^T a_2^T} \\ \boldsymbol{C_1^T a_2 B a_2^T} & \boldsymbol{C_1^T a_2 a_2^T C_1^T} & \boldsymbol{C_1^T a_2 a_2^T S_1^T} & \dots & \boldsymbol{C_1^T a_2 a_2^T C_n^T} & \boldsymbol{C_1^T a_2 a_2^T S_n^T} & \boldsymbol{-C_1^T a_2 a_2^T} \\ \boldsymbol{S_1^T a_2 B a_2^T} & \boldsymbol{S_1^T a_2 a_2^T C_1^T} & \boldsymbol{S_1^T a_2 a_2^T S_1^T} & \dots & \boldsymbol{S_1^T a_2 a_2^T C_n^T} & \boldsymbol{S_1^T a_2 a_2^T S_n^T} & \boldsymbol{-S_1^T a_2 a_2^T} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \boldsymbol{C_n^T a_2 B a_2^T} & \boldsymbol{C_n^T a_2 a_2^T C_1^T} & \boldsymbol{C_n^T a_2 a_2^T S_1^T} & \dots & \boldsymbol{C_n^T a_2 a_2^T C_n^T} & \boldsymbol{C_n^T a_2 a_2^T S_n^T} & \boldsymbol{-C_n^T a_2 a_2^T} \\ \boldsymbol{S_n^T a_2 B a_2^T} & \boldsymbol{S_n^T a_2 a_2^T C_1^T} & \boldsymbol{S_n^T a_2 a_2^T S_1^T} & \dots & \boldsymbol{S_n^T a_2 a_2^T C_n^T} & \boldsymbol{S_n^T a_2 a_2^T S_n^T} & \boldsymbol{-S_n^T a_2 a_2^T} \\ \boldsymbol{-a_2 B a_2^T} & \boldsymbol{-a_2 a_2^T C_1^T} & \boldsymbol{-a_2 a_2^T S_1^T} & \dots & \boldsymbol{-a_2 a_2^T C_n^T} & \boldsymbol{-a_2 a_2^T S_n^T} & \boldsymbol{a_2 a_2^T} \end{bmatrix}$$

4. EXPANSÃO EM SÉRIE DE FOURIER DAS REAÇÕES DE CONTATO PARA O ELEMENTO CILÍNDRICO

Embora a expansão em Série de Fourier dos esforços externos estivesse sendo realizada adequadamente para o elemento cilíndrico de parede espessa, notou-se que o mesmo não ocorria para as reações de contato. Isso porque, para este tipo de elemento não basta saber apenas o nó e a intensidade dos esforços aplicados sobre ele. É necessário conhecer também a distribuição destes esforços ao longo da direção angular para o nó em questão, como indicado na Figura 17.

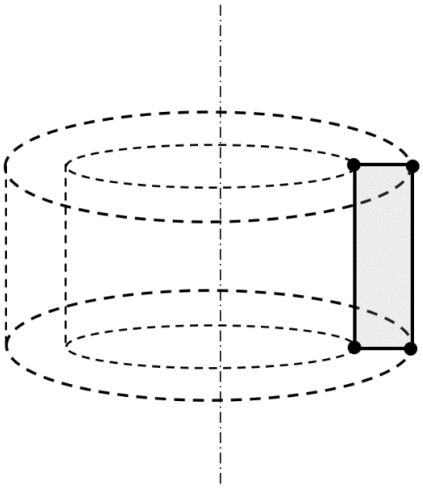


Figura 17 - Caso simplificado de apenas 1 elemento e em contato apenas no lado externo: a distribuição dos esforços deve ser conhecida para a aresta superior e para a inferior.

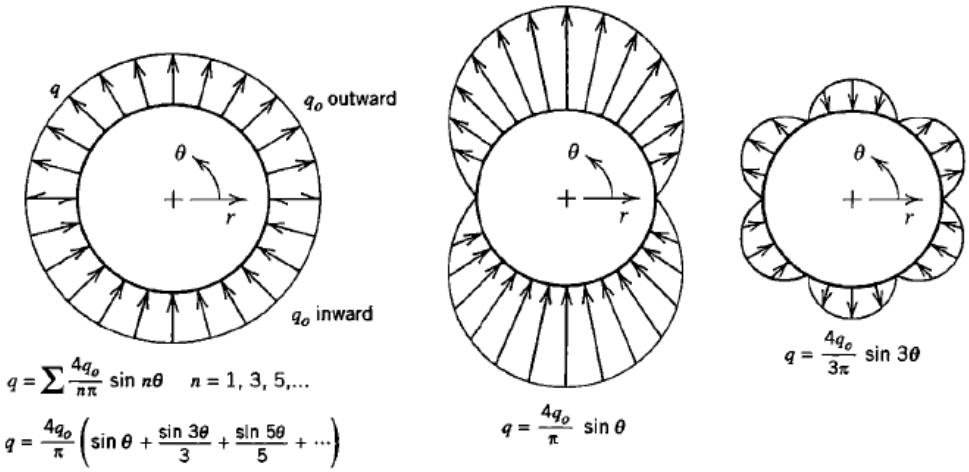


Figura 18 - Exemplos de distribuições angulares de esforços expandidos em Série de Fourier. *(COOK, MALKUS, PLESHA, & WITT, 2002)*

Assim como as forças externas, as reações de contato também devem ser expandidas em Série de Fourier, ou seja,

$$\mathbf{F}_{\boldsymbol{Fourier}} = \begin{Bmatrix} F_{N_{Fourier}} \\ F_{a1_{Fourier}} \\ F_{a2_{Fourier}} \end{Bmatrix} = \begin{Bmatrix} \sum_{i=0}^{n} \bar{F}_{\mathrm{N}}^{i} \cos i\theta + \sum_{i=0}^{n} \bar{\bar{F}}_{\mathrm{N}}^{i} \sin i\theta \\ \sum_{i=0}^{n} \bar{F}_{a1}^{i} \sin i\theta + \sum_{i=0}^{n} \bar{\bar{F}}_{a1}^{i} \cos i\theta \\ \sum_{i=0}^{n} \bar{F}_{a2}^{i} \cos i\theta + \sum_{i=0}^{n} \bar{\bar{F}}_{a2}^{i} \sin i\theta \end{Bmatrix}$$

Para a situação de contato de tipo “nó a nó” entre o elemento cilíndrico e o elemento de hélice, as reações de contato podem ser modeladas como esforços concentrados aplicados na região de contato. Portanto, pode-se utilizar a função Delta de Dirac, $\delta(x - \theta_0)$, para representá-los. A Figura 19 ilustra esta hipótese para a situação de contato normal, na qual F_N é considerada uma força pontual aplicada em θ_0.

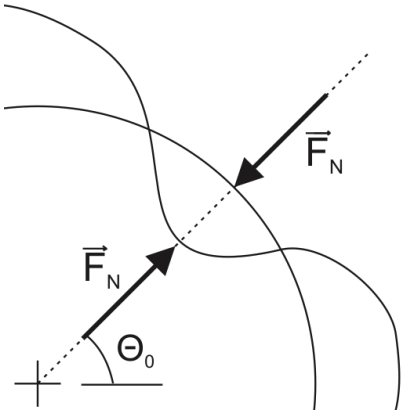


Figura 19 - Esforços concentrados na região de contato. Esta figura exemplifica a hipótese para a direção normal. Os modelos matemáticos são válidos para pequenas deformações e o fenômeno foi ampliado nesta imagem apenas para recurso didático.

A função Delta de Dirac possui as seguintes propriedades:

$$\delta(x - \theta_0) = 0\ , \quad \text{para } x \neq \theta_0$$

$$\int_{-\infty}^{\infty} F_j(x)\, \delta(x - \theta_0)\, dx = F_j(\theta_0)\ , \quad \text{para } j = n, a_1 \text{ e } a_2$$

A expansão em Série de Fourier obtém-se, para $j = n,\ a_1 \text{ e } a_2$:

$$F_{j_{Fourier}} = F_j^0 + \sum_{i=1}^{n} \bar{F}_{\mathrm{j}}^{i} \cos i\theta + \sum_{i=1}^{n} \bar{\bar{F}}_{\mathrm{j}}^{i} \sin i\theta$$

$$F_j^0 = \frac{F_j}{2\pi}$$

$$\bar{F}_{\mathrm{j}}^{i} = \frac{F_j}{\pi} \cos \pi \ i\theta_0$$

$$\bar{\bar{F}}_{\mathrm{j}}^{i} = \frac{F_j}{\pi} \sin \pi \ i\theta_0$$

A Figura 20 exemplifica a expansão em Série de Fourier para a situação de contato normal com uma força concentrada de 1N aplicada para $\theta_0 = 45°$.

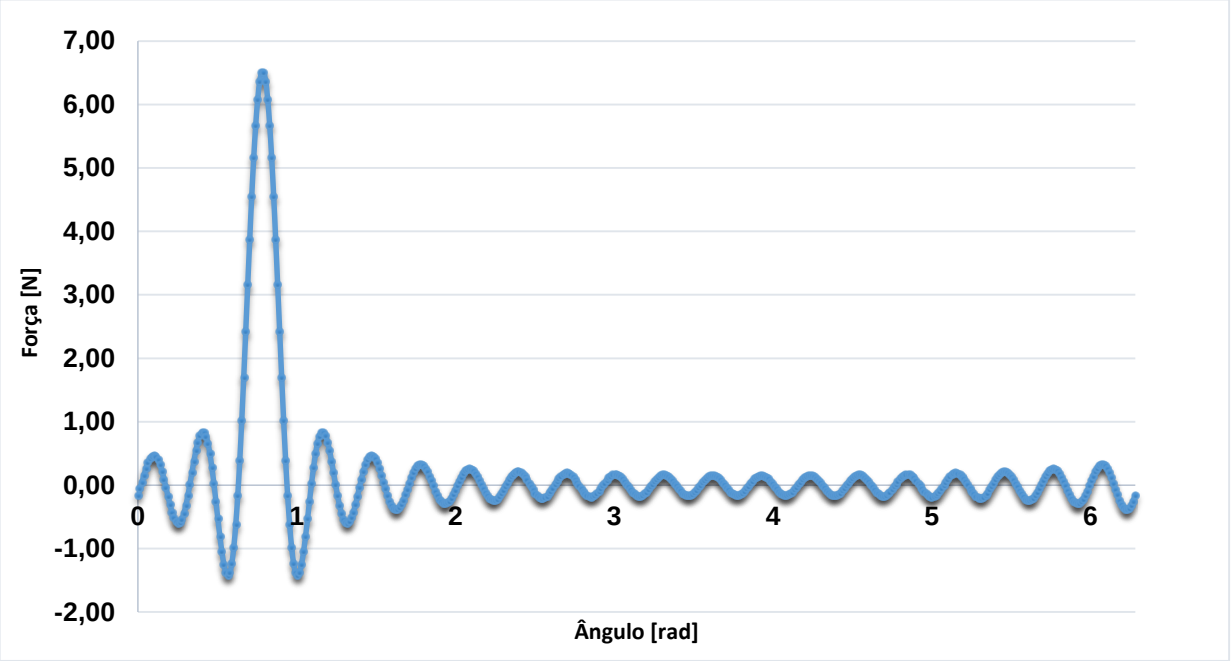


Figura 20 - Expansão em série de Fourier de ordem 20 para uma força concentrada de valor unitário aplicada a π/4 (45°). A integral desta função é igual ao módulo da força aplicada, ou seja, 1N.

5. IDENTIFICAÇÃO DOS GARGALOS COMPUTACIONAIS

Segundo o Princípio de Pareto, 80% dos recursos são utilizados por 20% das operações. Este princípio é utilizado em programação para ilustrar os efeitos gerados por gargalos computacionais, também conhecidos como “hot spots”, que são trechos de um código com elevada proporção de instruções executadas, que demandam elevada quantidade de memória, processamento e tempo. Por este motivo, os gargalos são os principais limitantes à utilização de um programa, que, para se tornar eficiente, é necessária a redução (ou se possível até a eliminação) da influência destes gargalos.

Pelos motivos apresentados, buscou-se a identificação dos gargalos computacionais do *software* UFCad. No entanto, devido à complexidade desta rotina de elementos finitos, com diversas classes e métodos, identificar estes gargalos computacionais não é uma tarefa tão simples. Este fato obrigou a adoção de uma metodologia de identificação, que será apresentada neste capítulo, juntamente com os seus resultados.

É interessante notar que os gargalos computacionais identificados compuseram a base da estratégia de otimização adotada, justificando as modificações no código que se fizeram necessárias e que serão apresentadas nos próximos capítulos.

No item 5.1 deste capítulo serão apresentadas ferramentas específicas para a identificação destes gargalos computacionais, conhecidas como *profilers*, que auxiliaram na identificação dos mesmos.

O UFCad apresentou dois tipos distintos de gargalos computacionais:

- Gargalos de processamento, apresentado no item 5.2;
- Gargalos de memória consumida, no item 9.3.

5.1 Seleção dos *profilers* utilizados na identificação dos gargalos computacionais

Com a crescente complexidade das rotinas implementadas, identificar estes gargalos computacionais torna-se uma tarefa longa e igualmente complexa. Otimizar o código todo é uma tarefa inviável e, na grande maioria das vezes, não necessária. A otimização de trechos de código que não foram selecionados adequadamente trará ganhos pouco substanciais à eficiência do mesmo. Segundo Donald Knuth, professor

emérito da Universidade de Stanford, as pequenas eficiências devem ser esquecidas em 97% dos casos, pois a otimização prematura é a raiz de todos os males (KNUTH, D., 1977).

Uma maneira prática para se analisar o código e encontrar seus gargalos é através da utilização de ferramentas desenvolvidas para esta finalidade, os *profilers*, que monitoram uma série de parâmetros (como consumo de memória, tempo de análise, número de threads, etc) permitindo-se assim avaliá-lo. Como os *profilers* são ferramentas projetadas para o maior número possível de aplicações e finalidades, existem vários opões de análise, como por exemplo linha-por-linha ("*line-by-line*"), amostragem baseada em evento ("*event-based sampling*"), amostragem baseada em tempo ("*time-based sampling*"), etc. A escolha dos métodos e parâmetros depende do caso analisado, de modo geral baseados em alguns critérios como tempo, precisão e custo de análise.

Como há uma distinção entre *profilers* para análise de processamento e *profilers* para análise de consumo de memória, a apresentação dos mesmos seguirá o mesmo padrão, conforme o respectivo tipo.

5.1.1 Profiler para análise de processamento

Para a análise de processamento, foram utilizados os seguintes *profilers*: "*ANTS Performance Profiler 8*" e "*dotTrace 5.5 Performance*", que são ferramentas comerciais que dispõem de licenças para estudante. Ambos os programas são muito semelhantes e se propõem a fazer praticamente as mesmas tarefas, o que permitiu comparar os resultados.

A Figura 21 e a Figura 22 mostram, respectivamente, as interfaces dos programas "*dotTrace*" e "*ANTS*". Nestas imagens é possível ver a pilha de chamada, ou pilha de execução (em inglês "*call stack*"), na qual são listados os métodos e as funções que compõem o programa, bem como seus parâmetros estatísticos mais importantes, por exemplo, a contagem do número de execuções e a porcentagem de tempo que o programa gasta com cada um deles. A organização e disposição destas pilhas de chamadas foram otimizadas pelos desenvolvedores dos softwares de modo a facilitar a identificação dos trechos críticos do código.

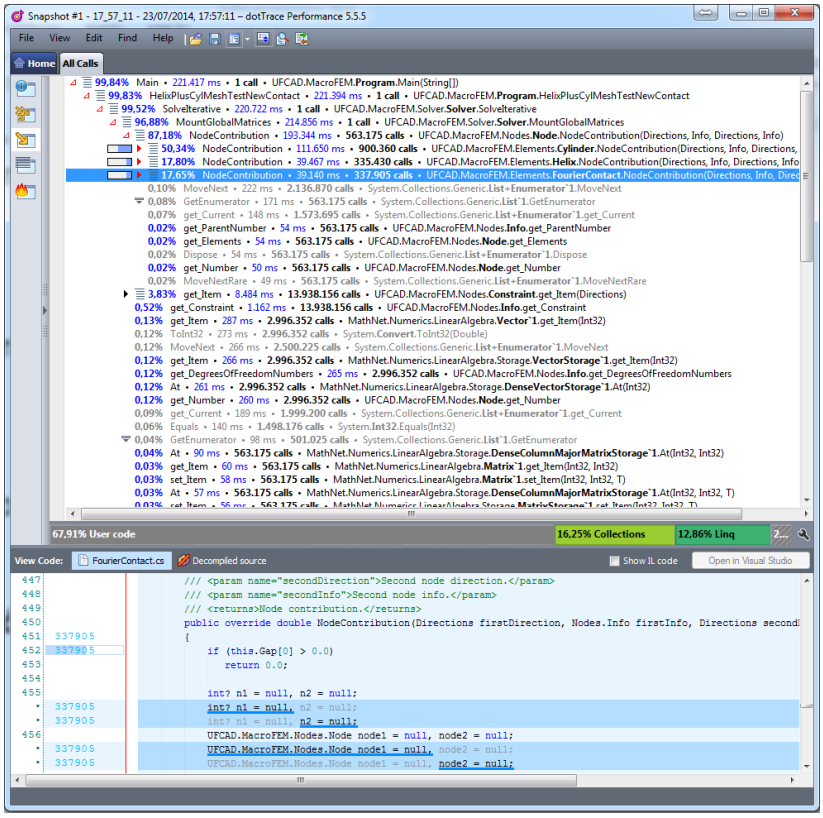


Figura 21 – Pilha de execução gerada pelo software "*dotTrace 5.5.5 Performance*".

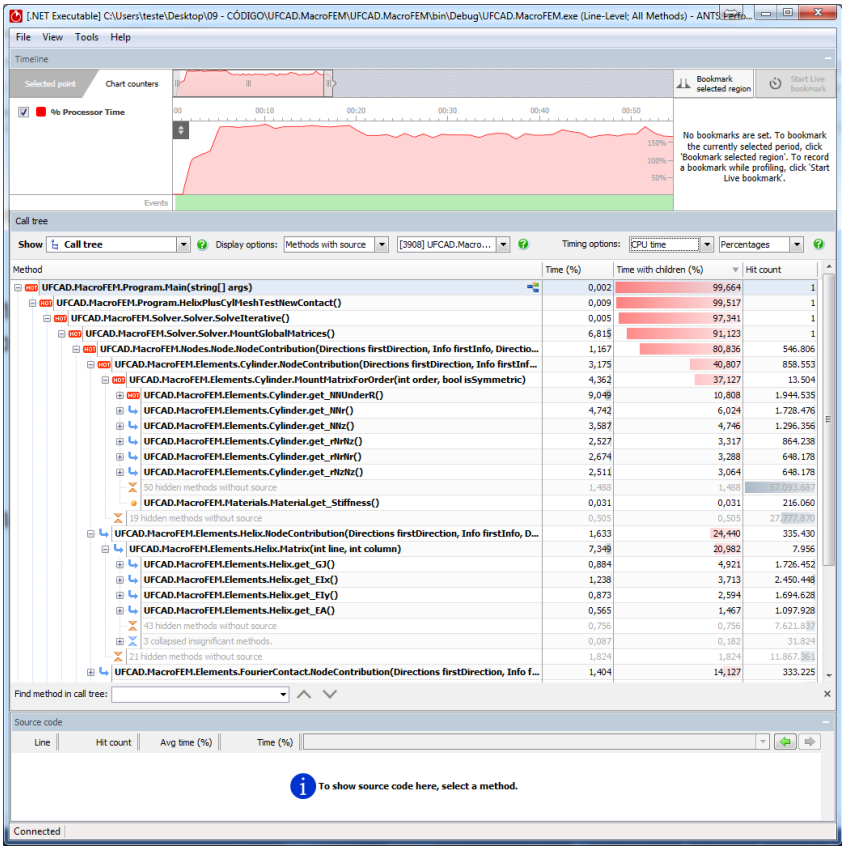


Figura 22 – Pilha de execução gerada pelo software "*ANTS Performance Profiler 8*".

A Figura 23 é um exemplo de organograma gerado automaticamente pelo programa "*ANTS Performance Profiler 8*". Este organograma mostra as relações hierárquicas entre os principais métodos chamados de um código, sendo muito útil para identificar aqueles que demandam proporcionalmente a maior parte do tempo de execução. Neste tipo de representação, os métodos que consomem poucos recursos de processamento são omitidos para não poluí-la visualmente, o que seria prejudicial à interpretação dos dados.

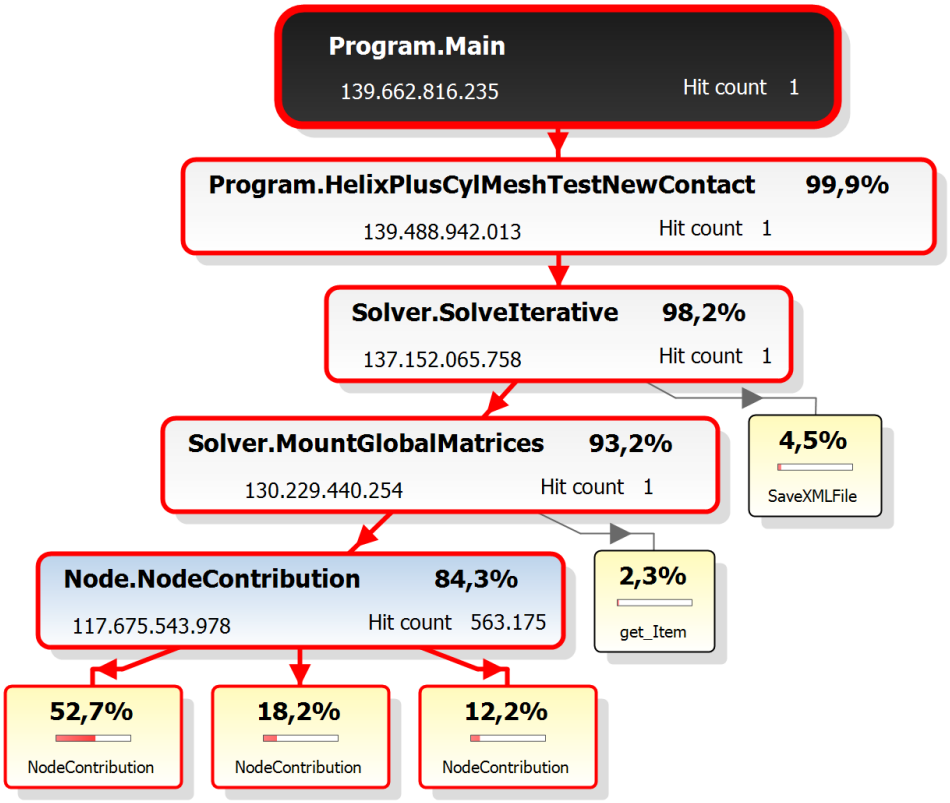


Figura 23 - Exemplo de um organograma gerado automaticamente pelo software *"ANTS Performance Profiler 8*".

5.1.2 Profiler para análise de consumo de memória computacional

Seguindo estratégia semelhante à da análise de processamento, testou-se os seguintes programas para a análise de consumo de memória: "*ANTS Memory Profiler 8"* e "*dotMemory 4.0".* Ambos são programas comerciais que dispõem de licenças para estudantes e há uma certa semelhança entre as análises que se propõem a realizar.

Por melhor se adequar as necessidades desta análise, o *software* "*dotMemory 4.0"* foi o escolhido.

Na Figura 24 pode ser visto um gráfico de consumo de memória ao longo do tempo, que é gerado ao mesmo tempo em que a aplicação é executada. Para se analisar um determinado instante com maiores detalhes, deve-se acionar o comando *snapshot*, o qual captura as informações que estão sendo coletadas e permite uma análise completa do código no instante em questão. Pode ser visto na Figura 25 a interface de um *snapshot* gerada pelo programa "*dotMemory 4.0".* O *snapshot* contém informações importantes, como o tamanho de memória ocupado pelos maiores objetos, vazamentos de memória e links para análises mais detalhadas, como a árvore de chamadas.

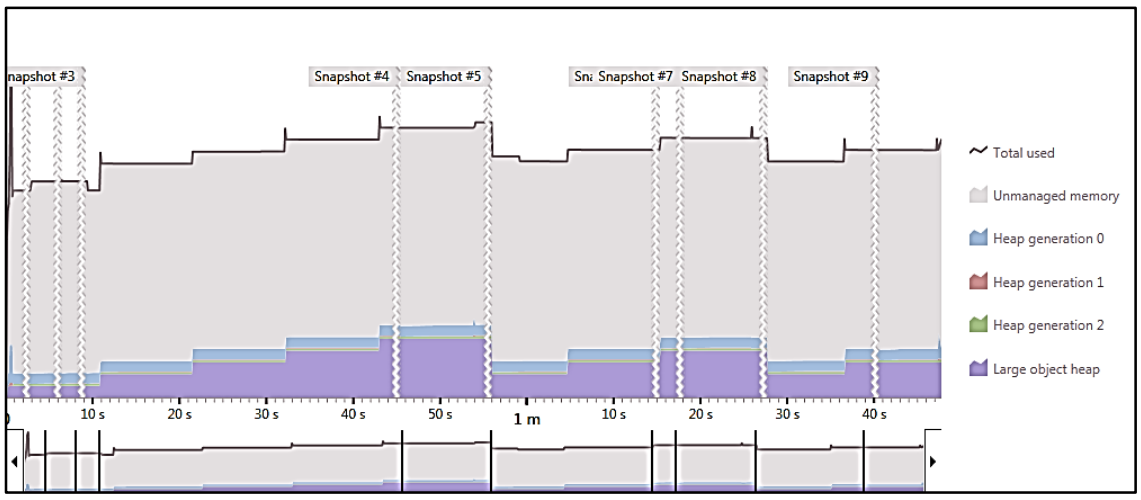


Figura 24 - Gráfico de consumo de memória em função do tempo gerado pelo software "dotMemory 4.0".

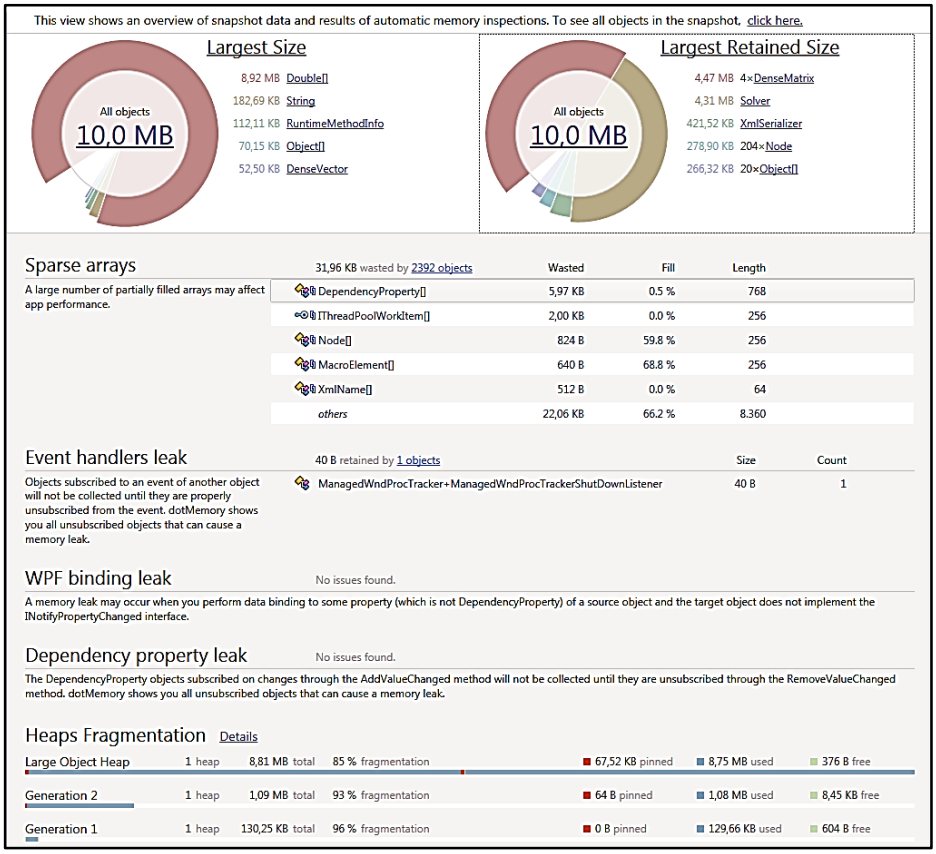


Figura 25 – Interface exibida pelo programa *"dotMemory 4.0*" ao se analisar um *snapshot*.

5.1.3 Problemas gerados pela utilização incorreta de *Profilers*

É importante ressaltar que o *profiler* também consome recursos de memória e processamento do computador, de modo proporcional ao nível de detalhamento e à quantidade de variáveis que estão sendo monitoradas. Como a sua execução ocorre em paralelo com a aplicação analisada, o *profiler* causará interferência nos resultados obtidos. Se esta interferência for muito elevada, o *profiler* pode acabar "mascarando" os gargalos do código e monitorando valores que não condizem com a execução real do programa (como consumo de memória e tempo de análise). Nestes casos extremos, perde-se a capacidade de analisar o código e encontrar seus gargalos. Portanto deve-se sempre questionar os resultados e tomar todos os cuidados para que esta interferência seja minimizada. A Figura 26 é um bom exemplo deste fenômeno. Ao se utilizar o *profiler* "*dotMemory 4.0"* com nível máximo de detalhamento, foram necessários mais 3 minutos para concluir a análise. No entanto, a mesma rotina leva aproximadamente 12 segundos para ser executada. Isso significa que o nível máximo de detalhamento, além de desnecessário, impossibilita a análise de modelos de elementos finitos com elevado número de elementos.

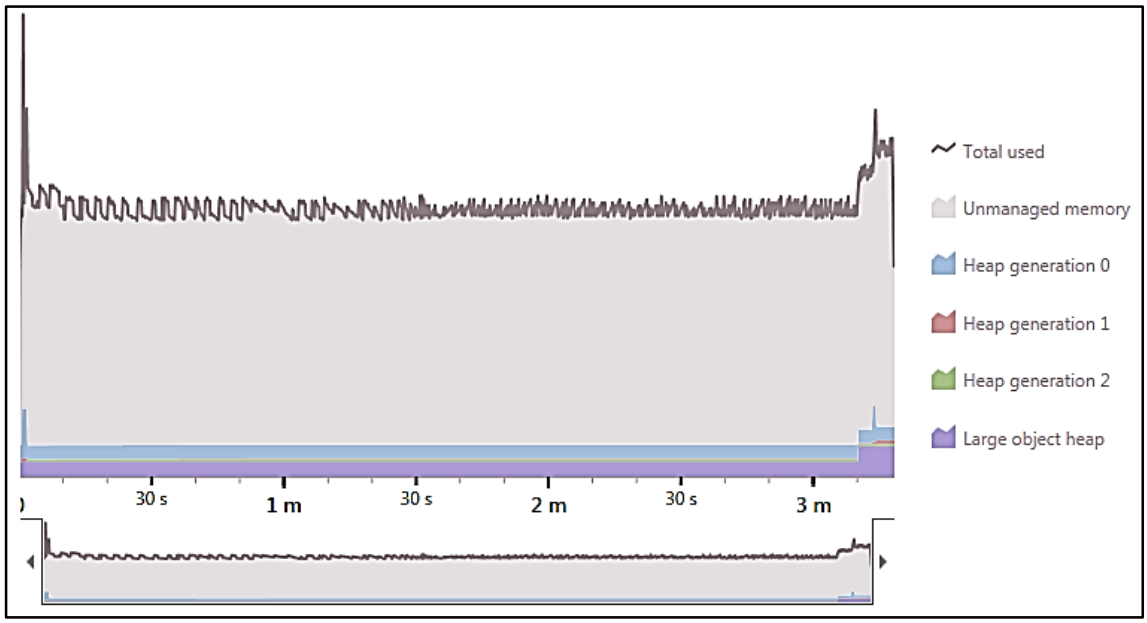


Figura 26 – Gráfico de consumo de memória em função do tempo gerado pelo *software* "*dotMemory 4.0*"

Para as análises realizadas, escolheu-se parâmetros adequados (como por exemplo métodos de amostragem, que reduzem o número de coletas de memória), mas que ainda mantivessem a observabilidade do sistema. Com isso, foi possível realizar a análise pelo *profiler* com tempo compatível com o que a código levar para ser executado. Além disso, adotou-se a precaução de somente comparar casos que tenham utilizados exatamente os mesmos parâmetros de análise.

5.2 Resultados e conclusões da análise de processamento

Para a análise de processamento do "UFCad", implementou-se um caso constituído de uma capa plástica e um arame de armadura conectados por elementos de contato (com a a condição de *sticking* e *sliding)*, para o qual utilizou-se os profilers "*ANTS Performance Profiler 8"* e "*dotTrace 5.5 Performance"* para gerar organogramas como os das Figura 27 e Figura 28.

Os resultados vistos nestas duas figuras são de uma simulação com reduzido número de elementos finitos, porém suficientes para observar que a montagem da matriz de rigidez constitui uma etapa crítica na eficiência do "UFCad", levando mais de 90% do tempo total de execução. Isso ocorre, pois nesta etapa são realizadas muitas

operações matemáticas para o cálculo das matrizes de rigidez de cada um dos elementos finitos que compõem o modelo simulado.

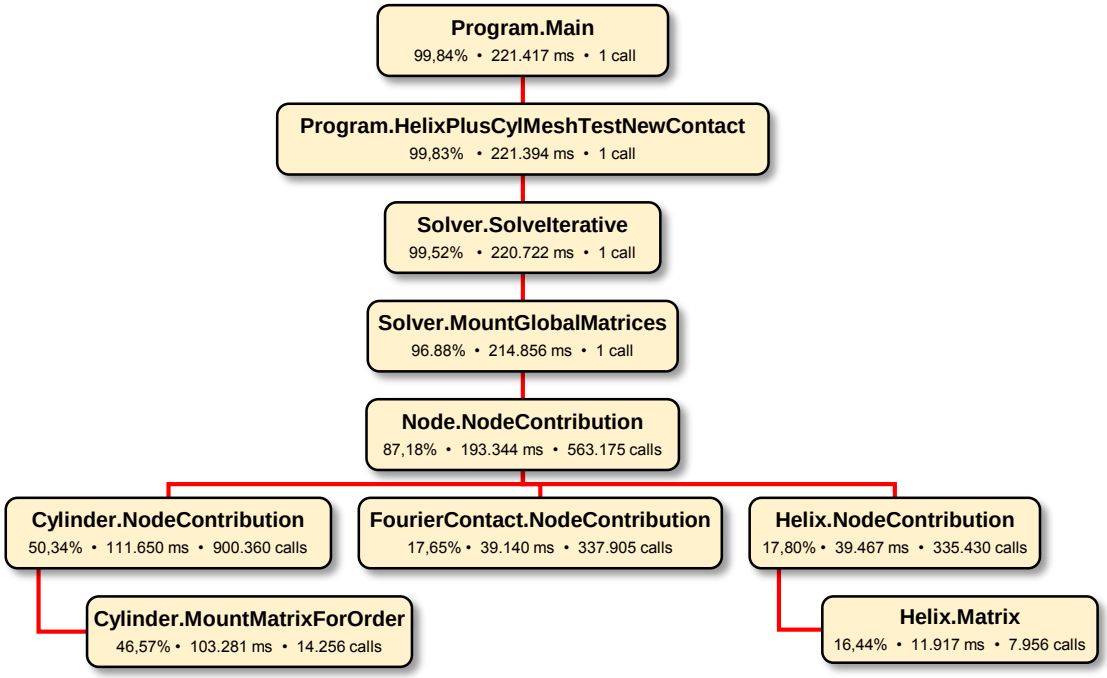


Figura 27 – Organograma criado através da compilação dos dados gerados pelo *software* "*dotTrace 5.5.5 Performance*".

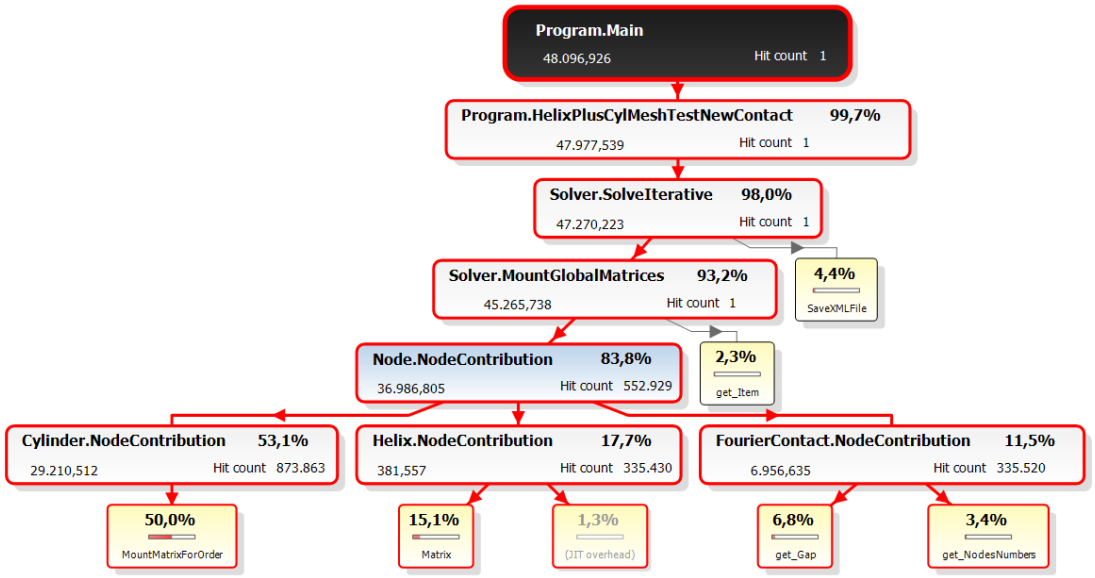


Figura 28 – Organograma gerado automaticamente pelo *software "ANTS Performance Profiler".*

O desempenho do programa "UFCad" é uma função não linear que depende de uma série de fatores, como por exemplo a forma como cada elemento é implementado,

proporção entre os tipos de elemento, trechos que apresentam comportamento variado de acordo com o tamanho dos dados matrizes manipulados, etc.

Um modo interessante para avaliar a influência da montagem da matriz global de rigidez na eficiência do programa é comparar o tempo que a matriz global de rigidez leva para ser montada com o tempo total de execução. Isso foi feito para uma série de situações: variando-se a quantidade de elementos do modelo de macroelementos finitos (Figura 29) e variando-se a ordem da expansão em Série de Fourier para os elementos de contato (Figura 30), o que consequentemente elevou também o tamanho, em megabytes, da matriz global de rigidez. Verifica-se que o valor da proporção de ambos os casos diminui à medida em que o tamanho da matriz global de rigidez aumenta. Isso ocorre, pois a resolução do sistema linear para obtenção dos deslocamentos possui influência crescente sobre a eficiência global do programa com o aumento do tamanho da matriz de rigidez. Este fato, no entanto, será devidamente explorado mais adiante, durante a análise de consumo de memória. O importante, tanto da Figura 29 quanto da Figura 30, é verificar que a montagem da matriz de rigidez requer um tempo significativo, de 50% à 95% do tempo total de simulação.

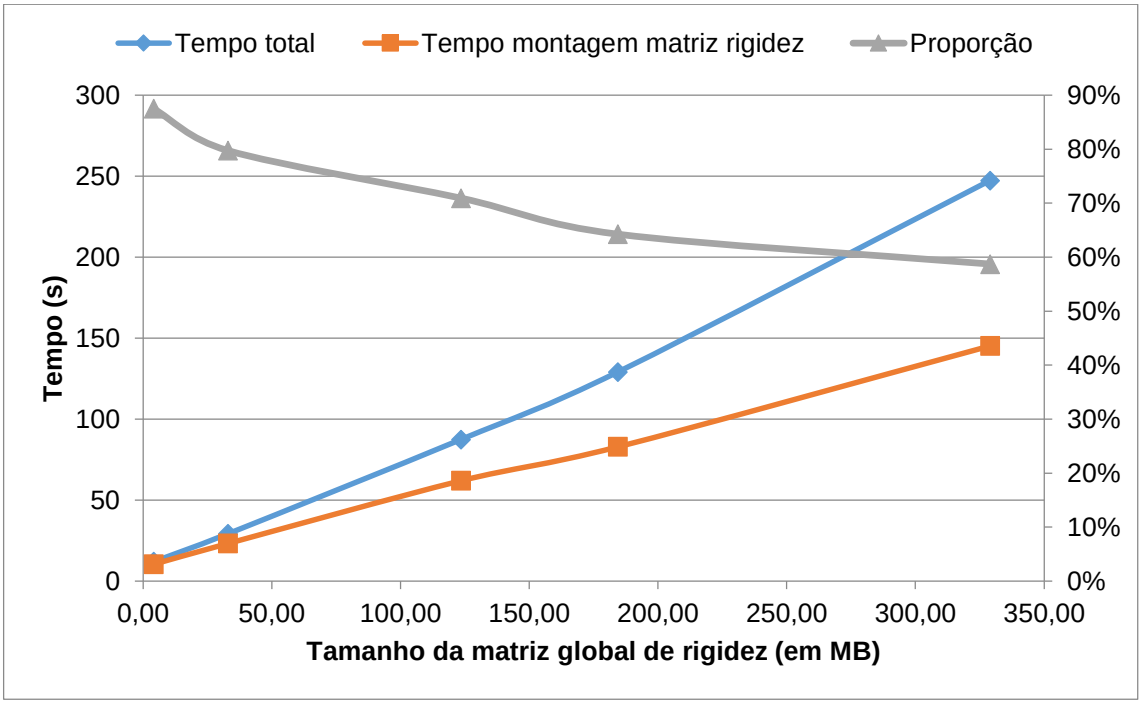


Figura 29 – Comparação entre o tempo total de execução do programa “UFCad” e o tempo de montagem da matriz global de rigidez em função do tamanho (em megabytes) da matriz global de rigidez. A variação no tamanho da matriz de rigidez foi obtida aumentando-se o número de

macroelementos finitos do modelo e mantendo-se constante a ordem da expansão em Série de Fourier dos elementos de contato.

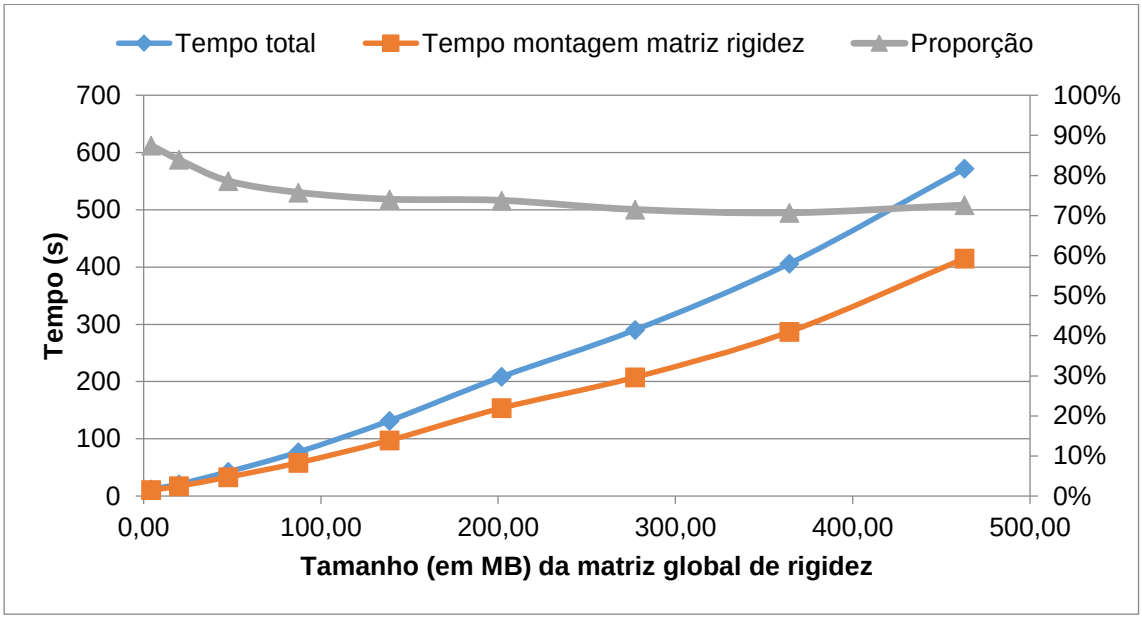


Figura 30 – Comparação entre o tempo total de execução do programa “UFCad” e o tempo de montagem da matriz global de rigidez em função do tamanho (em megabytes) da matriz global de rigidez. A variação no tamanho da matriz de rigidez foi obtida aumentando-se a ordem da expansão em série de Fourier e mantendo-se constante o número de macroelementos finitos do modelo.

Ao se verificar também o uso de CPU durante a execução do “UFCad”, notou-se que o mesmo variava entre 15% à 30%, como indicado na Figura 31. Além disso, a distribuição de processamento entre os núcleos não era homogênea, inclusive com alguns deles não sendo utilizados. Com isso, conclui-se que o programa não está aproveitando todo o potencial de processamento disponível. Modificações com intuito de elevar esta taxa de utilização de CPU trarão ganhos substanciais em relação tempo de simulação.

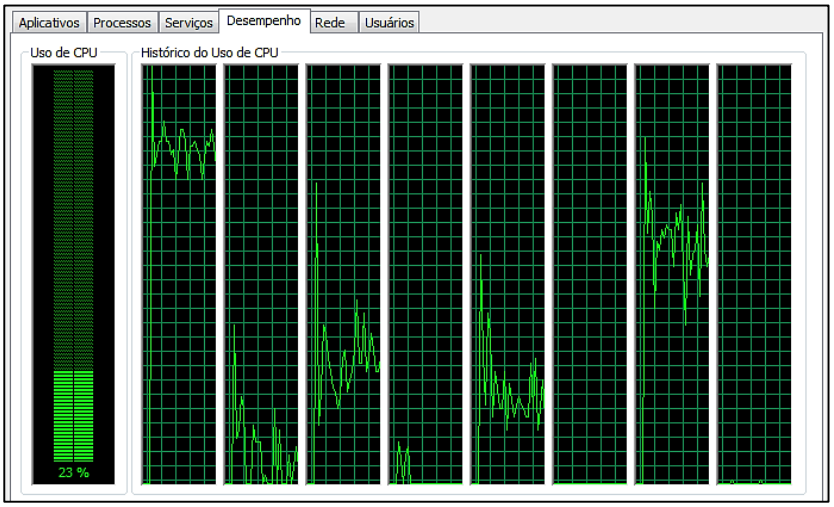


Figura 31 – Uso de CPU durante a execução do *profiler "ANTS".*

Portanto, com os resultados apresentados acima, conclui-se que a estratégia mais adequada para abordar e reduzir os gargalos de processamento é através da introdução de métodos de paralelização computacional no trecho de código responsável pela montagem das matrizes de rigidez, o que elevará a taxa de uso de CPU. Com isso, espera-se aumentar a eficiência do código e reduzirão o tempo de análise, o que impactará diretamente nos custos de análises com elevado número de graus de liberdade.

5.3 Resultados e conclusões da análise de consumo de memória computacional

Para a avaliação do consumo de memória, considerou-se um caso envolvendo um cilindro e um arame interligados por elementos de contato, permitindo deslocamento normal e tangencial entre elementos cilíndricos e helicoidais, sob a condição de contorno imposta de descolamento compressivo.

Realizou-se também uma análise da influência do número de elementos do modelo de elementos finitos. Os resultados desta análise podem ser vistos na Tabela 1 e Figura 32.

Tabela 1 – Consumo de memória em função do número de elementos do modelo. Parâmetros fixos da análise: matriz global de rigidez fora do loop; ordem de Fourier igual a 0.

Nr	Nº Elem. Hélice	Nº Elem. Cilíndricos	Máx. Mem. (MB)	Tempo (s)

#1	25	100	93	12
#2	50	400	163	29.3
#3	75	900	442	87.4
#4	75	1200	642	129.1
#5	100	1600	1096	247.3
#6	**200**	**6400**	**OUT OF MEM.**	**-**

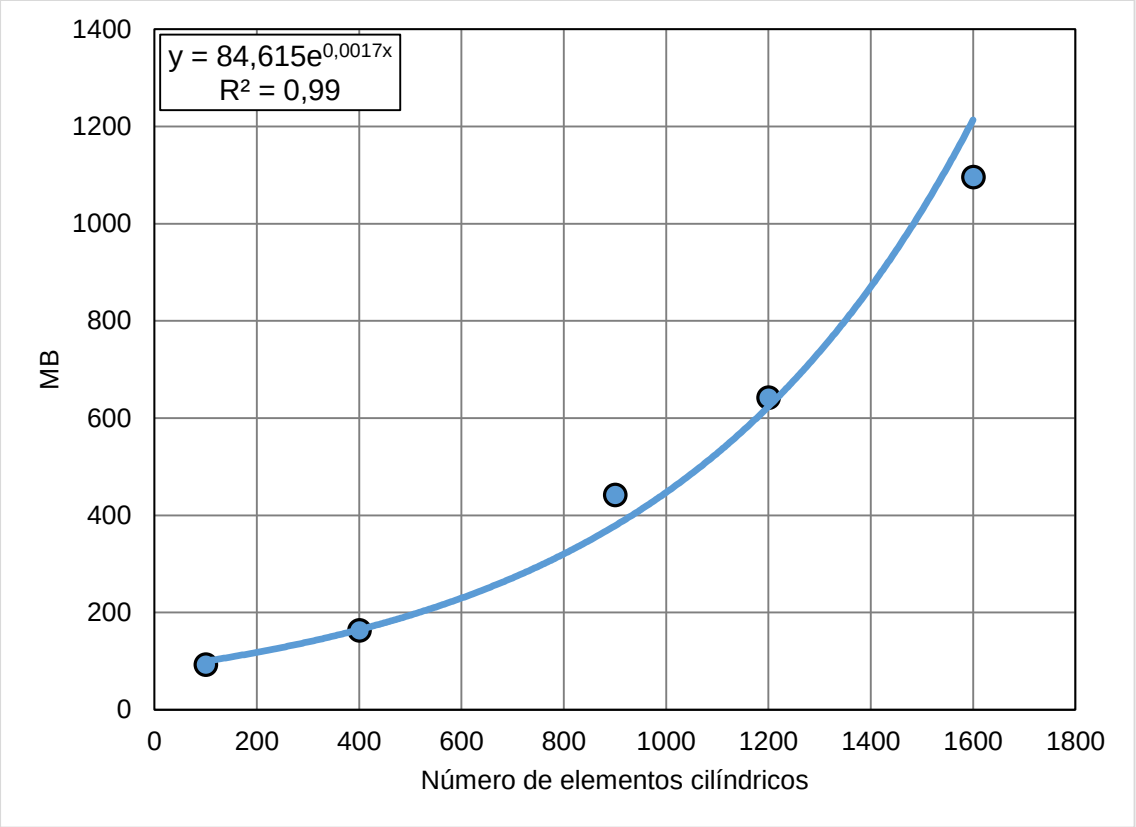


Figura 32 - Consumo de memória (MB) em função do número de elementos cilíndricos.

Também realizou-se uma análise da influência da ordem de expansão em série de Fourier no consumo máximo de memória. Os resultados podem ser conferidos na Tabela 2 e na Figura 33.

Tabela 2 - Consumo de memória em função da ordem de expansão da série de Fourier. Parâmetros fixos da análise: matriz global de rigidez fora do loop; número de elementos cilíndricos igual 100; número de elementos de hélice igual à 25.

Nr	Ord. Fourier	Núm. elem. cilíndricos	Máx. Mem. (MB)	Tempo total (s)
#1	0	100	93	12

#2	1	100	135	20,8
#3	2	100	200	42,3
#4	3	100	313	76,9
#5	4	100	504	131,5
#6	5	100	713	208,4

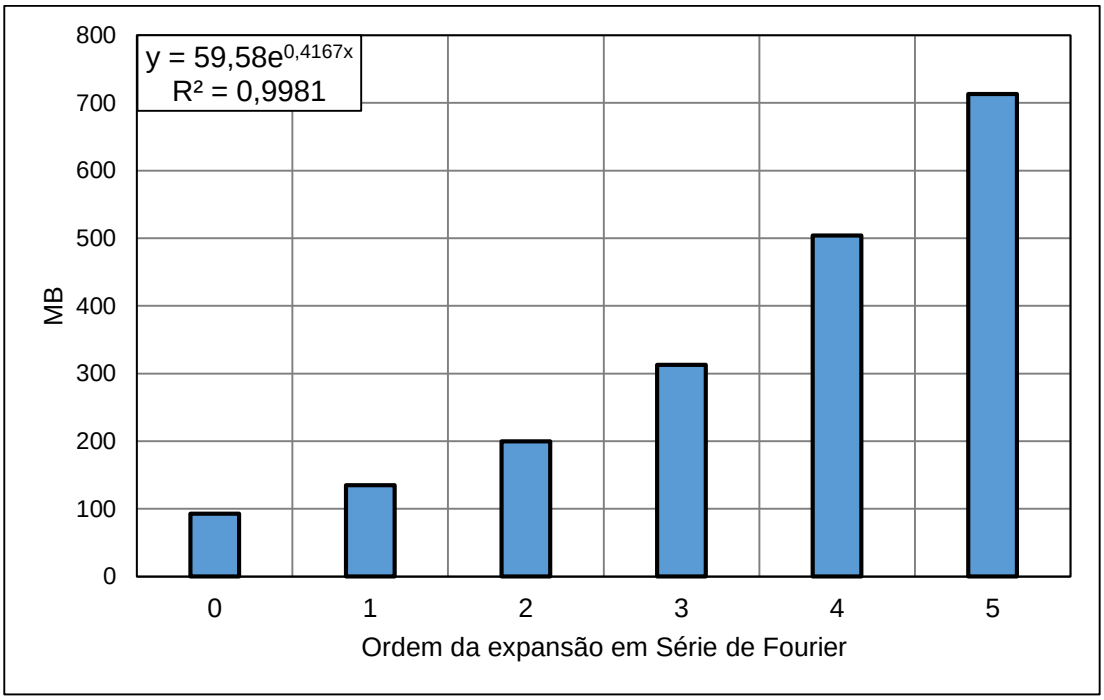


Figura 33 - Consumo de memória (em megabytes) em função da ordem da expansão em Série de Fourier.

O programa “UFCad” apresenta consumo exponencial de memória computacional com relação ao tamanho do modelo de macroelementos finitos, fazendo com que o limite máximo de memória disponível na CPU seja excedido com um número relativamente baixo de elementos. Isto ocorre por dois motivos:

- O programa utiliza matrizes densas ao invés de matrizes esparsas para armazenar os dados;
- Problemas e limitações com a resolução do sistema linear (assunto que será abordado em maiores detalhes no capítulo 7).

Adicionalmente, a configuração padrão da linguagem C# limita a criação e manipulação de matrizes a apenas 2 GB, o que inviabiliza a utilização do programa “UFCad” para a simulação de modelos de elementos finitos de grande escala.

6. MODIFICAÇÕES NO PROGRAMA UFCAD

6.1 Modificação 01 – A alteração do número de vezes em que a matriz global de rigidez é computada

Em função da natureza dos modelos de elementos finitos do programa “UFCad”, deve-se utilizar o *solver-interativo*. Problemas que envolvem contato do tipo *gap* (abertura e fechamento) são não-lineares e requerem métodos iterativos para a convergência. O *solver-interativo* subdivide linearmente o carregamento externo em 10 *steps*, que subdividem-se em um número variável de *substeps*. De modo simplista, o *solver-interativo* aplica o carregamento externo correspondente a um *step*, determina quais pares de contato estarão ativos para e em seguida atualiza as matrizes de rigidez dos elementos de contato correspondentes, caracterizando um *substep*. Para o mesmo carregamento externo, ele determina novamente os pares em contato e atualiza as matrizes de rigidez destes elementos, caracterizando um novo *substep*. Este ciclo persiste até que a convergência para este *step* seja atingida, quando então realiza-se o incremento de um décimo nos carregamentos externos, dando início aos *substeps* do próximo *step*.

Antes da Modificação 01, para cada um dos 10 *steps,* o programa recalculava todas as matrizes de rigidez dos elementos finitos do modelo, montava a matriz global e armazenava-a como uma propriedade. Na transição de uma iteração para a outra, um novo espaço de memória era alocado na memória *heap* do sistema e a matriz de rigidez antiga era eliminada da memória somente quando o sistema realizava a operação “*garbage collector”*.

Como mostra a Figura 34, esta operação somente faria sentido se os valores de deslocamentos fossem incluídos nos cálculos da nova matriz global de rigidez, o que aumentaria a taxa de convergência, ou seja, faria o problema convergir em menor número de iterações.

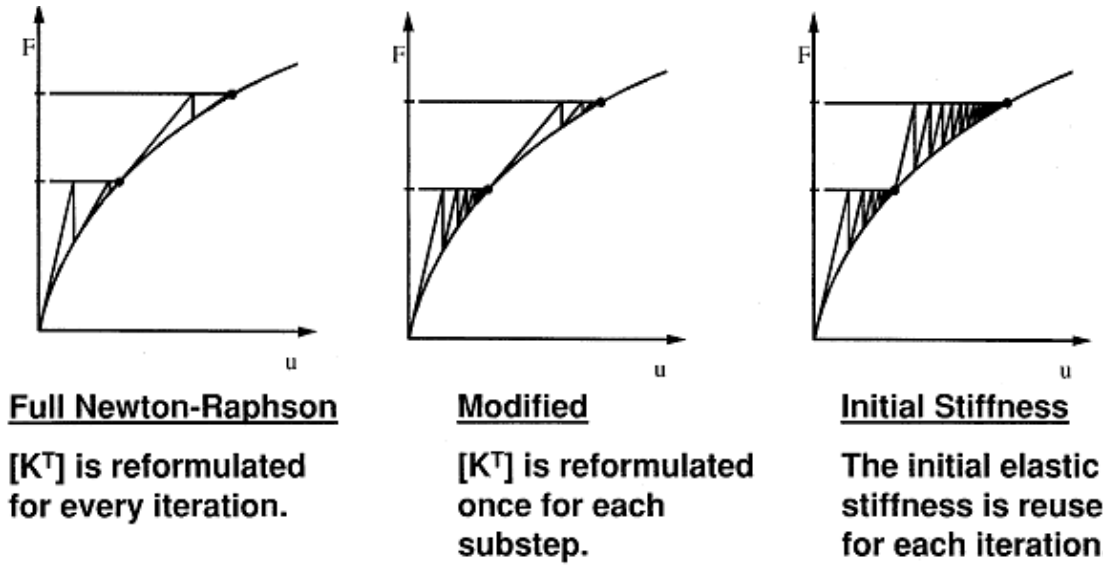


Figura 34 - Taxa de convergência em função da atualização ou não das dimensões no cálculo da matriz de rigidez *(Newton-Raphson Option (NROPT), 2014)*.

No entanto, como a matriz global de rigidez não está sendo reformulada, mas apenas recalculada, gasta-se tempo um tempo muito elevado e totalmente desnecessário nesta operação, com prejuízos de performance e consumo de memória.

Deslocando-se a montagem da matriz global de rigidez para fora do *loop* de iterações, o número de vezes em que a matriz global de rigidez é calculada durante a execução do programa passou de 10 para apenas 1 vez. Ao eliminar-se esta repetição de cálculos, houve um ganho expressivo de performance do programa e também redução do tempo de simulação, conforme o indicado nas Figura 35 à Figura 40.

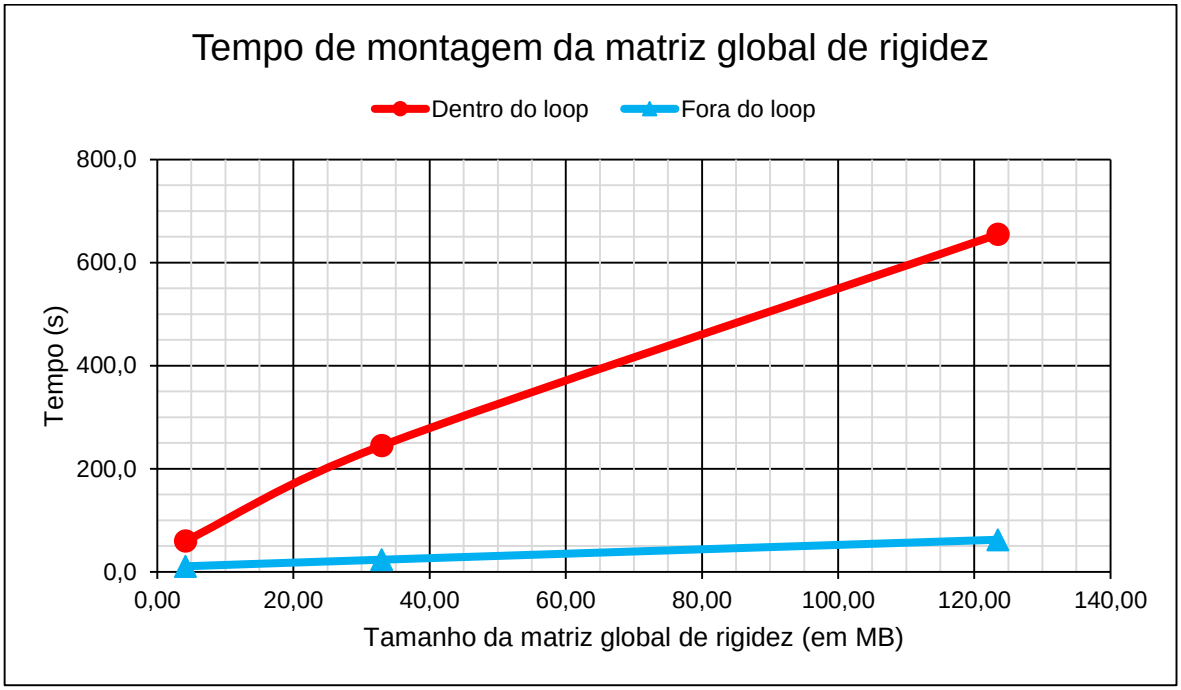


Figura 35 – Comparação de tempo de montagem de matriz global quando: a matriz é calculada apenas 1 vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha).

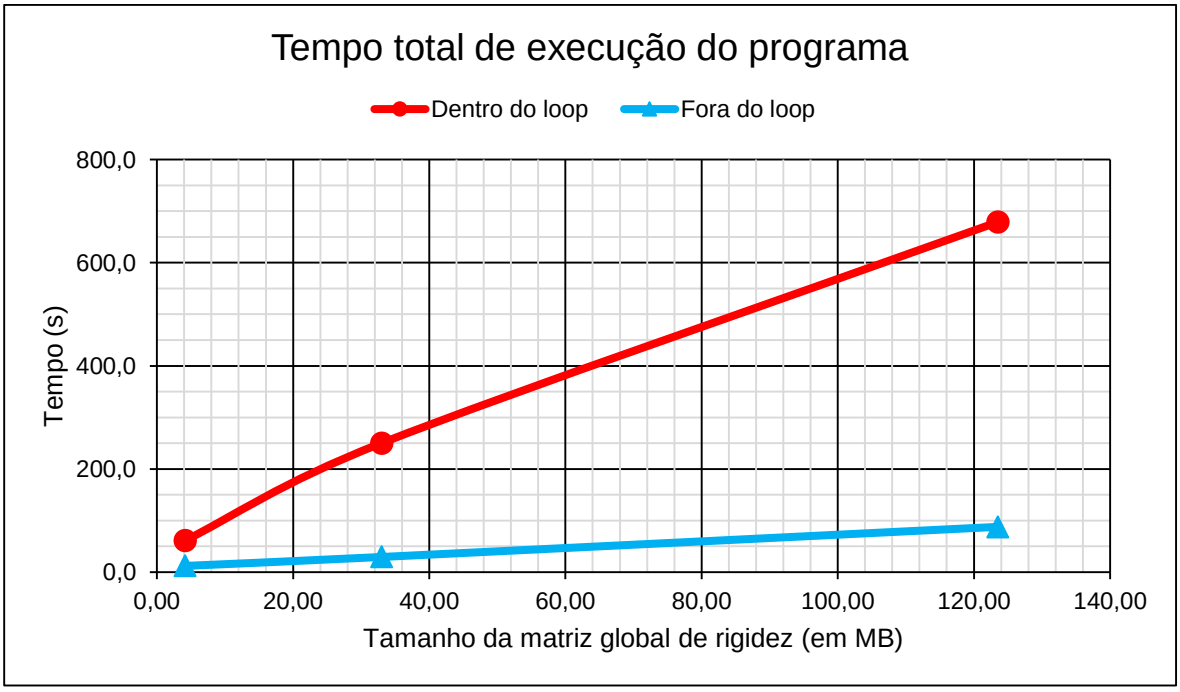


Figura 36 - Comparação de tempo total de execução do programa quando: a matriz é calculada apenas uma vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha).

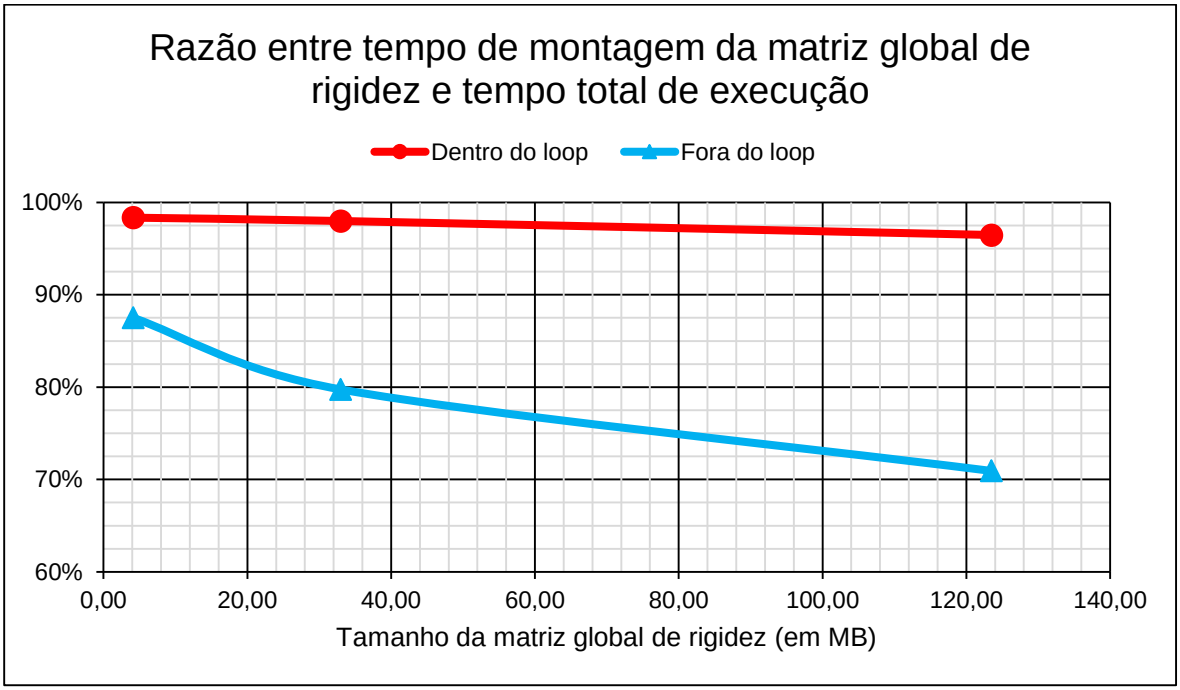


Figura 37 - Razão entre tempo de montagem da matriz global de rigidez e tempo total de execução quando: a matriz é calculada apenas 1 vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha).

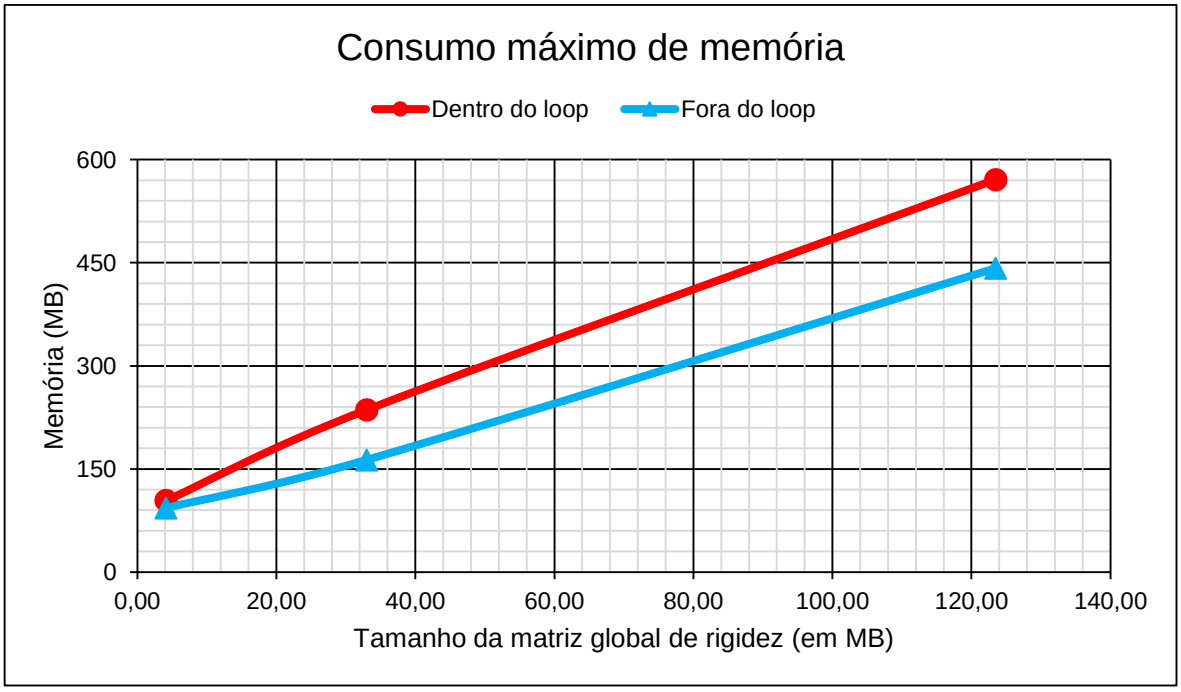


Figura 38 – Comparação de consumo máximo de memória quando: a matriz é calculada apenas 1 vez (fora do loop, curva azul); a matriz é recalculada toda iteração (dentro do loop, curva vermelha).

A Figura 39 mostra gráfico de consumo de memória ao longo do tempo para este caso. Mais do que isso, nela também é possível notar 10 patamares distintos de

valores de memória, cada um deles correspondentes aos 10 *steps* que o programa realiza.

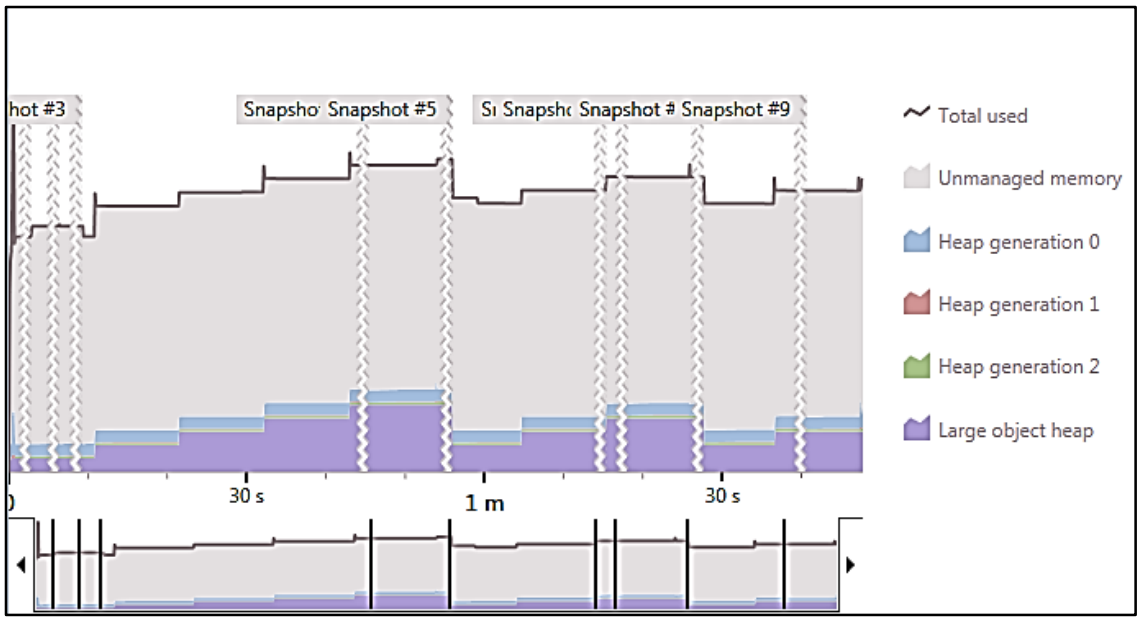


Figura 39 - Consumo de memória ao longo do tempo para o caso em que a matriz de rigidez é calculada em cada *step*.

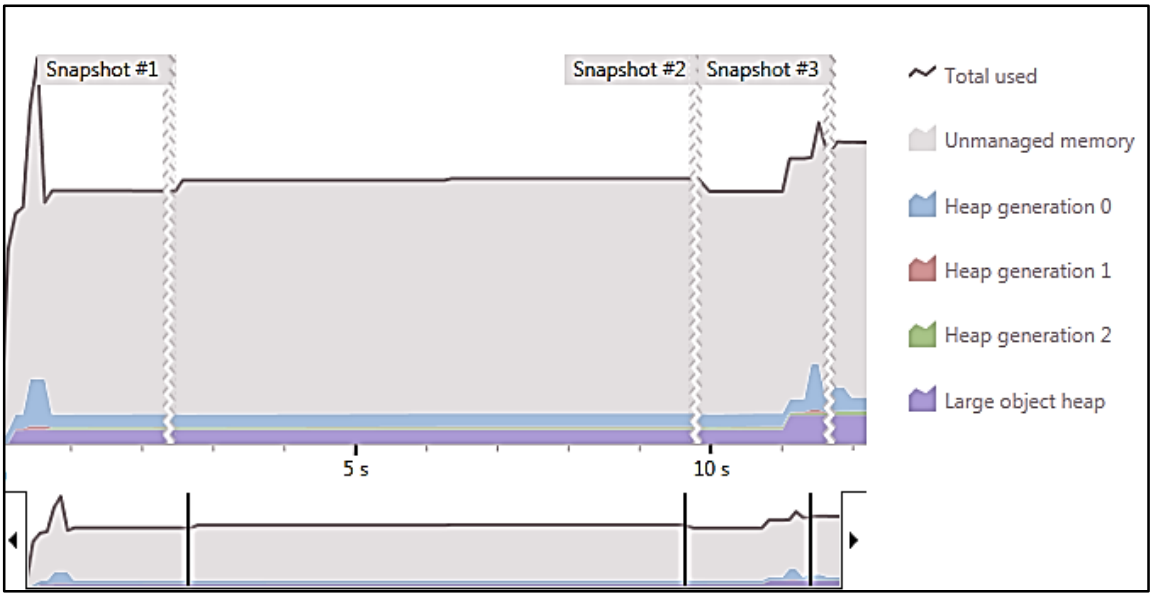


Figura 40 - Consumo de memória ao longo do tempo para o caso em que a matriz global de rigidez é calculada apenas uma vez.

6.2 Modificação 02 – Alteração na varredura de montagem da matriz de rigidez

Também foi alterada a lógica de varredura dos nós para a montagem da matriz de rigidez do programa UFCad. A varredura consiste em identificar corretamente a posição na matriz global de rigidez em que cada uma das matrizes locais de rigidez deve ser adicionada. A varredura é realizada com algumas das propriedades que o UFCad armazena de cada um dos elementos finitos do modelo: o tipo de elemento e os nós que o compõem.

O número de graus de liberdade dos nós de cada tipo de elemento é determinado pela formulação matemática dos elementos. Cada nó do elemento de hélice, por exemplo, possui 6 graus de liberdade, enquanto que, para o elemento cilíndrico, os nós possuem apenas 3 graus de liberdade, uma vez que neste caso as rotações foram desprezadas.

A ***varredura por nó*** consiste em verificar na lista de elementos quais elementos finitos estão associados a cada um dos graus de liberdade do modelo. Isso exige uma quantidade de buscas que cresce quadraticamente com o número de graus de liberdade do modelo, o que se torna um processo demorado e de elevado custo computacional. Esta lógica foi substituída pela ***varredura por elemento***. Sabendo-se quais nós que compõe o elemento finito, é possível determinar a posição na matriz de rigidez, pois existe uma correlação entre a numeração dos nós e suas respectivas localizações na matriz global de rigidez. A varredura por elemento exige uma quantidade de buscas na lista de elementos proporcional à quantidade de elementos, o que representa uma grande vantagem computacional, ilustrada na Figura 41. Neste caso, as eficiências de ambos os tipos de varredura foram comparadas através da medição do tempo de montagem em função do número de graus de liberdade do modelo. É importante dizer que, neste caso, o modelo simulado continha uma capa plástica externa conectada rigidamente à apenas uma armadura de tração. A quantidade de componentes foi mantida constante, de modo que a variação do número de graus de liberdade ocorreu somente pelo refinamento da discretização do modelo. Este gráfico comprova o crescimento quadrático do tempo total de montagem da matriz global de rigidez com o aumento do número de graus de liberdade para a varredura do por nó.

Trata-se, portanto, de uma comparação de casos extremos, devido à baixa proporção entre a quantidade de elementos e graus de liberdade deste caso, o que explicita as

diferenças de performance entre as duas metodologias. Em geral, casos com maior proporção entre a quantidade de elementos e graus de liberdade (vários componentes conectados entre si, por exemplo) irão ocupar a região entre as duas curvas do gráfico da Figura 41.

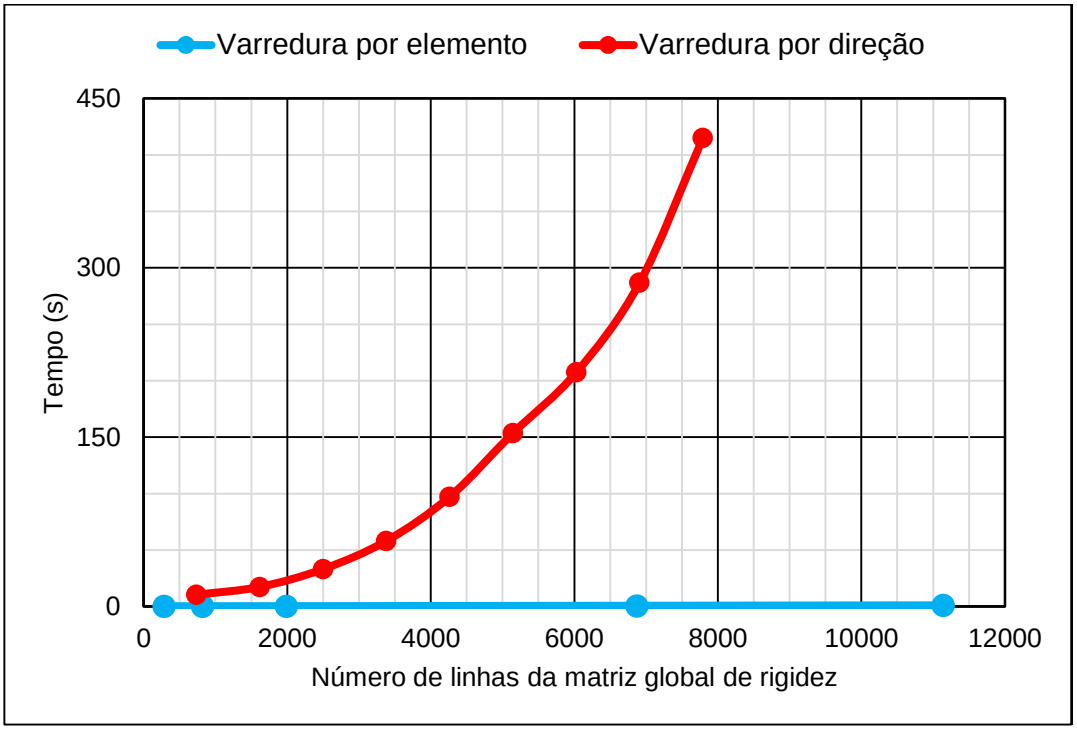


Figura 41 - Comparação de performance entre os métodos de montagem da matriz global de rigidez.

6.3 Modificação 03 – Paralelização da montagem da matriz global de rigidez

Computadores modernos possuem mais de um núcleo de processamento e um código que utilize paralelização pode tirar grande proveito disto. É possível reduzir significativamente o tempo total de simulação ao se utilizar ferramentas específicas de programação. Neste contexto, a paralelização da etapa de montagem de matriz global de rigidez pode trazer grandes vantagens ao programa, devido à elevada quantidade de operações realizadas nesta operação.

Para a realização desta modificação, foi utilizado uma série de recursos disponibilizados pelo C#. *"A classe Parallel na TPL permite paralelizar algumas construções de programação comuns, sem exigir uma reformulação do aplicativo. Internamente, a classe Parallel cria um conjunto próprio de objetos Task e sincroniza automaticamente essas tarefas quando finalizadas. A classe Parallel está localizada no namespace System.Threading.Tasks e dispõe de um pequeno conjunto de*

métodos estáticos para indicar que o código deve ser executado em paralelo, se possível. Esses métodos são os seguintes:

- ***Parallel.For*** *– ele define um loop no qual as iterações podem ocorrer em paralelo ao utilizar tarefas. O método é executado para todo valor entre o valor inicial e um abaixo do valor final especificado, e o parâmetro é preenchido com um inteiro que especifica o valor atual.*
- ***Parallel.ForEach<T>*** *– ele define um loop no qual as iterações podem ocorrer em paralelo. O método é executado para cada item na coleção.*
- ***Parallel.Invoke*** *– você pode utilizar esse método para executar um conjunto de chamadas a métodos sem parâmetros como tarefas paralelas."* (SHARP, 2011).

Nem todas as etapas de um código podem ser paralelizadas. Isto ocorre para etapas que dependem de resultados das anteriores e, neste caso, deve-se garantir a correta execução do programa. Caso isso não seja respeitado, mesmo mantendo-se inalterados todos os parâmetros, o programa irá gerar resultados incorretos e que serão diferentes em todas as vezes que for executado. Um trecho de código é dito "thread-safe" quando ele manipula dados compartilhados entre estruturas de um modo que garanta a segurança de execução dos múltiplos threads ao mesmo tempo (ORACLE, 2010).

Foi realizada a paralelização da etapa de montagem da matriz de rigidez, aumentando-se a taxa de uso processamento, mas esta operação ainda não pôde ser concluída, devido às dificuldades encontradas para se garantir o "*thread safety*". Os testes realizados mostraram desvios na oitava casa decimal, mas que não podem ser desprezados, pois indicam uma falha na sequência de execução das operações. Portanto, esta etapa ainda requer melhorias para que seja integralmente finalizada.

No entanto, é importante notar que os ganhos obtidos com as modificações 6.1 e 6.2 tornaram os ganhos da paralelização apenas residuais, reduzindo a importância deste gargalo computacional.

6.4 Modificação 04 – Inclusão de recurso para habilitar matrizes com mais de 2GB

Foi disponibilizado após o lançamento do ".NET Framework 4.5" um método para desativar a limitação de matrizes de até 2GB de memória. Esta desativação é realizada através dos comandos indicados na Tabela *3*.

A utilização deste recurso permitiu um aumento considerável na quantidade de dados possíveis de serem armazenados na memória RAM, e com isso a simulação de modelos maiores, limitados apenas à quantidade disponível na CPU.

Tabela 3 – Comando em C# para habilitar matrizes com mais de 2GB.

```
<configuration>
  <runtime>
    <gcAllowVeryLargeObjects enabled="true" />
  </runtime>
</configuration>
```

6.5 Modificação 05 – Conversão das Matrizes da Classe "*Solver*" para Matrizes Esparsas

Com o objetivo de reduzir o consumo de memória, realizou-se a conversão das matrizes da classe "*Solver*" de densas para esparsas. Esta modificação trouxe um ganho significativo ao programa quanto à capacidade de armazenamento, como indica a Figura 42, sendo possível implementar modelos com mais de 200000 graus de liberdade por meio da formulação esparsa, que anteriormente estavam limitados somente a 16000 graus de liberdade (para um computador com 8 GB de memória RAM disponíveis).

No entanto, os benefícios desta modificação não puderam ser aproveitados, devido ao fato do programa UFCad ainda não contar com um método eficiente para a resolução de sistemas lineares esparsos de grande escala, o que será apresentado em maiores detalhes no próximo capítulo.

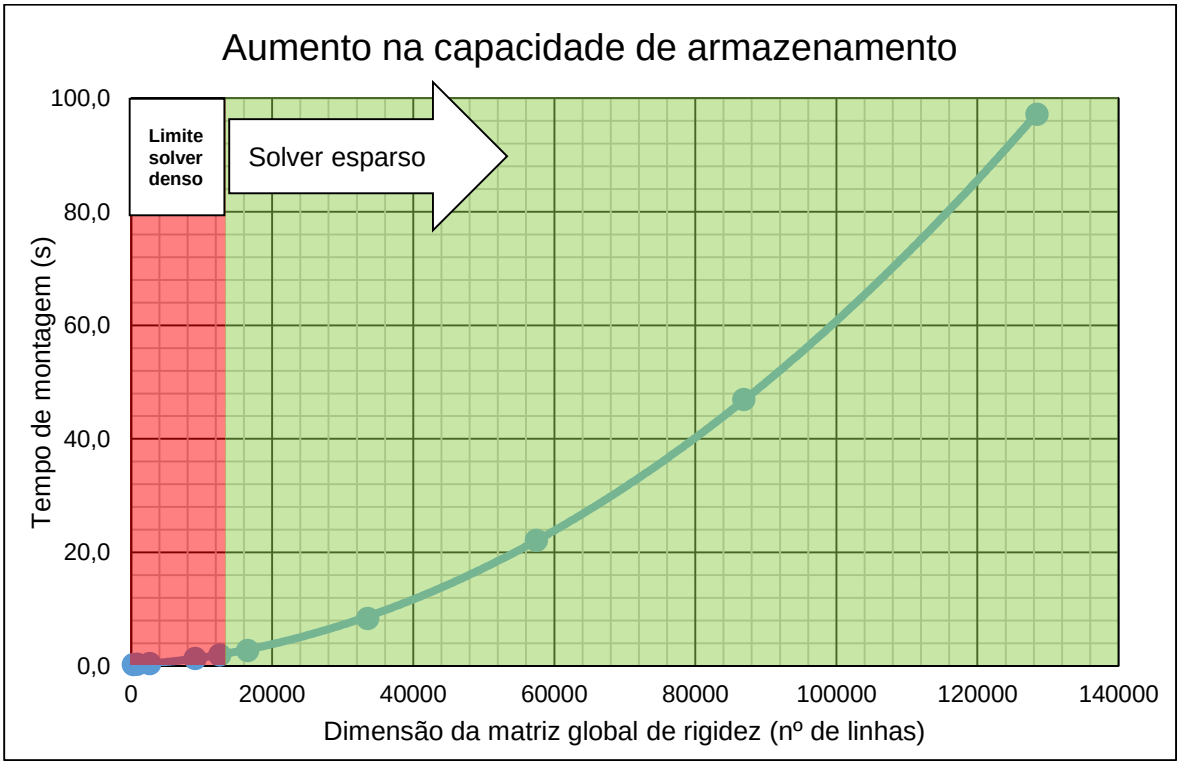


Figura 42 – Aumento na capacidade de armazenamento com a implementação do solver com matrizes esparsas.

A conversão no formato das matrizes gerou uma perda de performance na montagem da matriz global de rigidez, o que pode ser visto na Figura 43. Operações algébricas com matrizes esparsas podem não ser vantajosas ao compará-las com matrizes densas, se a esparsidade for baixa, ou seja, quando a proporção de células nulas é baixa. Isso porque, neste caso, o número de operações matemáticas para ambos os formatos é muito próximo, porém com o agravante de que as matrizes esparsas requererem controles e verificações adicionais. Pode-se mesclar a utilização de matrizes densas e esparsas, sendo a escolha determinada pelos seus respectivos graus de esparsidade. No entanto, mesmo neste caso haverá perda de performance, pois será necessária a conversão de formatos para durante as operações algébricas. Deste modo, otimizar o programa a relação capacidade de armazenamento e performance não é uma tarefa simples ou linear. Assim, na medida do possível, cada matriz deve ser analisada individualmente, atentando-se ao impacto gerado pela alteração no tipo de armazenamento dos dados.

Para o exemplo ilustrado na Figura 42, a perda de performance é totalmente aceitável quando comparada com o aumento gerado na capacidade de armazenamento.

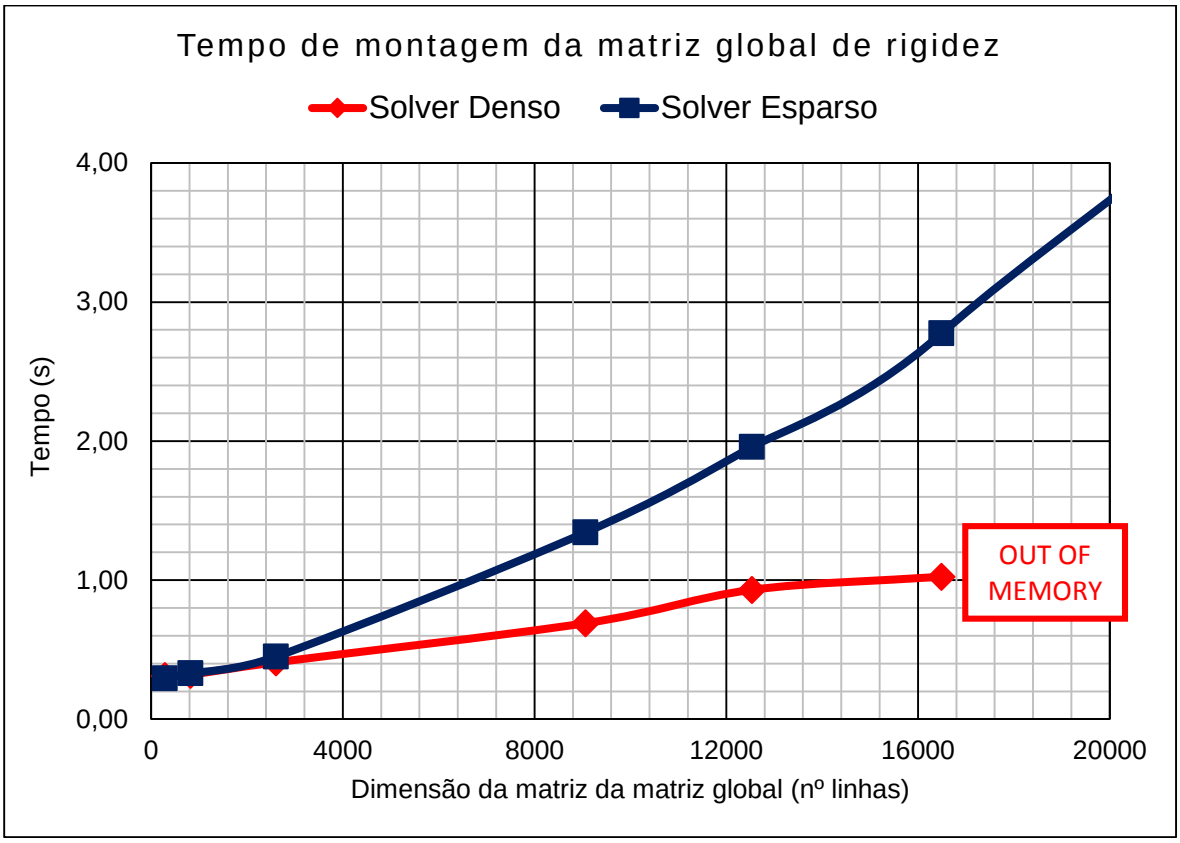


Figura 43 – Perda de performance na montagem da matriz global de rigidez gerada pela implementação da matriz global de rigidez.

6.6 Modificação 06 – Formulação e implementação de uma nova estratégia de detecção de contatos entre armaduras de tração.

Originalmente, o programa utilizava a metodologia de *pinball region* para detectar os contatos entre armaduras, que consiste em procurar para cada nó do modelo outros nós que estejam dentro de uma região de raio especificado. Esta formulação é generalizada e muito empregada no método dos elementos finitos. No entanto, para um problema envolvendo armaduras de tração, com vários componentes que se cruzam e que se encontram muito próximos uns dos outros, esta formulação se mostrou deficiente, pois eram detectados muitos mais pares de contato do que o preciso, conectando partes da estrutura que não deveriam estar conectadas.

Deste modo, foi necessária uma formulação específica para a detecção de contatos entre armaduras de tração, levando em conta o fato de possuírem geometria helicoidal, a qual encontra-se no item 6.6.1. Foram realizadas checagens desta formulação, desde a contagem do número de pares de contato até a utilização de

métodos gráficos. Os resultados, disponíveis no item 6.6.2, comprovam a efetividade da formulação proposta.

6.6.1 Formulação de um novo método de detecção de contatos entre armaduras de tração.

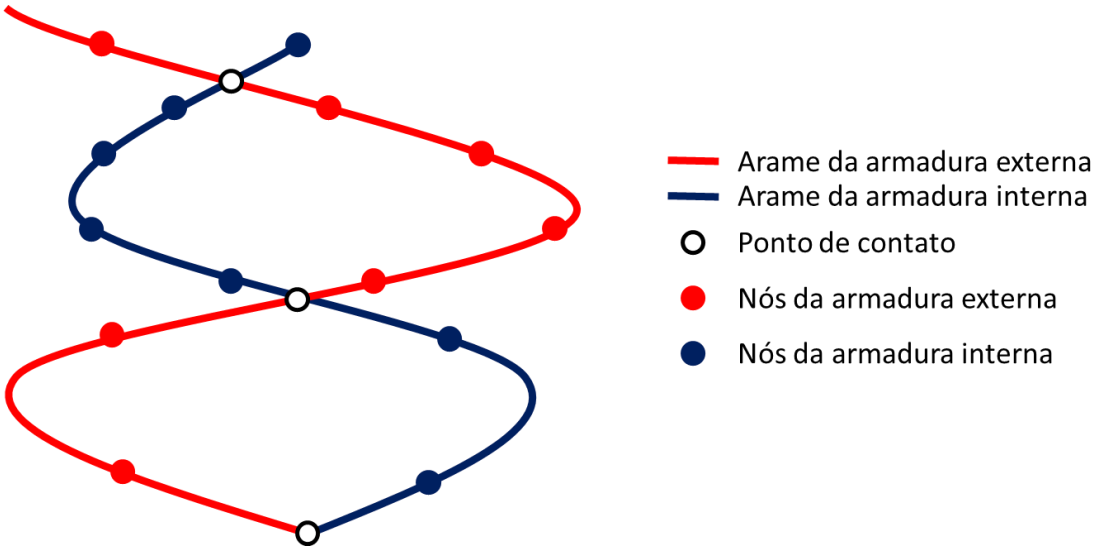


Figura 44 – Planificação das trajetórias dois arames de armaduras de tração.

Parâmetros necessários:

- $\emptyset_{i0}$ – ângulo inicial da armadura interna de tração;
- $\emptyset_{e0}$ – ângulo inicial da armadura externa de tração;
- R_i – raio da armadura interna de tração;
- R_e – raio da armadura externa de tração;
- α_i – ângulo de assentamento da armadura interna;
- α_e – ângulo de assentamento da armadura externa;
- $P_i = \frac{\pi\, 2R_i}{\tan \alpha_i}$ – passo da armadura interna de tração;
- $P_e = \frac{\pi\, 2R_e}{\tan \alpha_e}$ – passo da armadura externa de tração;

Equação da hélice em coordenadas cilíndricas:

$$x_i = R_i \,.\, \cos c_i.t$$

$$y_i = R_i \,.\, \sin c_i.t$$

$$z = t$$

Onde,

$$c_i = \frac{2\pi}{P_i} = \frac{\tan \alpha_i}{R_i}$$

Analogamente, para a armadura externa de tração:

$$x_e = R_e \,.\, \cos c_e.t$$

$$y_e = R_e \,.\, \sin c_e.t$$

$$z = t$$

Com,

$$c_e = \frac{2\pi}{P_e} = \frac{\tan \alpha_e}{R_e}$$

1º Passo: Identificar os pontos de intersecção

Em coordenadas cilíndricas, para haver a interseção entre as armaduras,

$$t_i = t_e = t$$

$$\theta_i = \theta_e = \theta$$

$$\text{para } 0 \leq t \leq L_{max}$$

Assim,

$$\theta_i = c_i.t + \emptyset_{i0}$$

$$\theta_e = c_e.t + \emptyset_{e0}$$

Igualando-se as expressões acima e levando-se em consideração o fato de que θ é uma variável periódica, obtém-se:

$$c_i.t + \emptyset_{i0} = c_e.t + \emptyset_{e0} + k \,.\, 2\pi$$

E portanto,

$$\boldsymbol{t = \frac{\emptyset_{e0} - \emptyset_{i0} + k \,.\, 2\pi}{c_i - c_e}}$$

$$\text{para } \boldsymbol{k = 0, 1, 2, \ldots} \quad \text{e} \quad \boldsymbol{0 \leq t \leq L_{max}}$$

A expressão acima é empregada para determinar algebricamente as coordenadas dos pontos geométricos de interseção entre os dois arames. É interessante notar que a quantidade destes pontos depende da quantidade total de arames em cada armadura,

e dos seus respectivos raios e ângulos de assentamento. No entanto, esta quantidade independe da quantidade de nós, ou seja, do grau de discretização das armaduras.

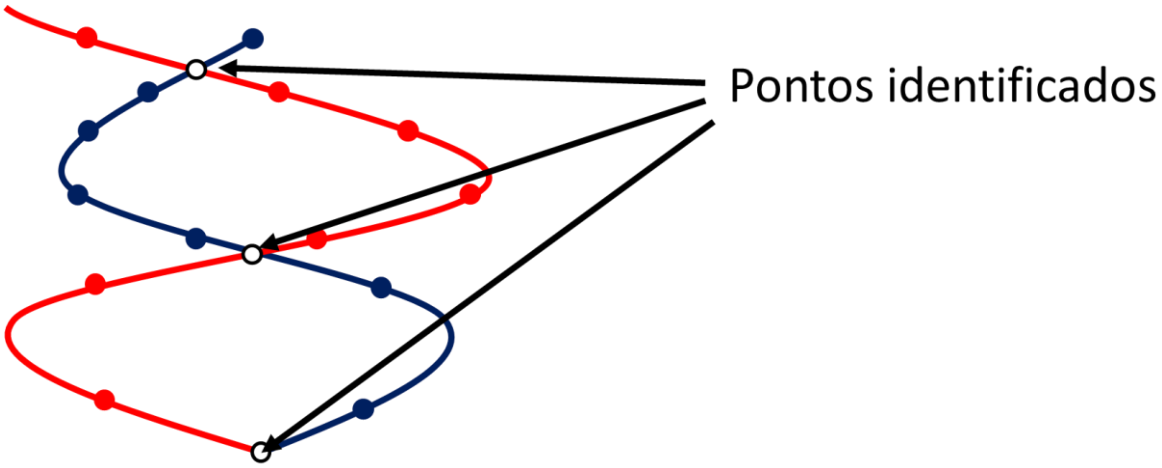


Figura 45 – Pontos de intersecção são determinados algebricamente pela expressão apresentada.

2º Passo: Identificar os nós mais próximos destes pontos geométricos

Após identificados os pontos de interseção, deve-se identificar os nós de cada armadura que mais se aproximam deles. Esta identificação é realizada segundo a metodologia a seguir, ilustrada também na Figura 46:

- Para cada ponto de interseção calculado;
    - Para cada uma das duas hélices;
        - Para cada um dos nós presentes na hélice;
            - Cálculo da distância do nó ao ponto de interseção;
                - Seleção do nó de menor distância.

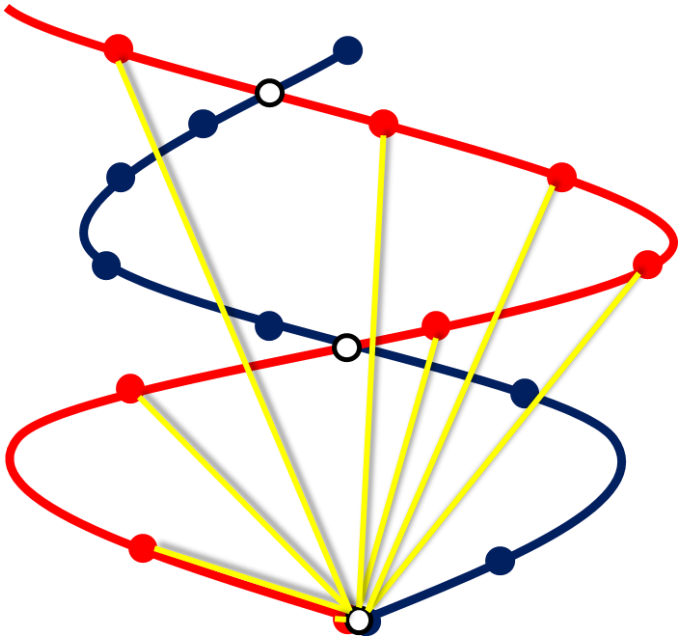


Figura 46 – Exemplo de checagem de distância entre os nós da armadura externa e o primeiro ponto geométrico de intersecção.

3º Passo: Ligar os nós mais próximos dos pontos geométricos

Após determinados os nós que possuem a menor distâncias aos pontos geométricos de intersecção, deve-se associá-los corretamente e em seguida criar o elemento do tipo *bonded* ligando estes nós, procedimento ilustrado na Figura 47.

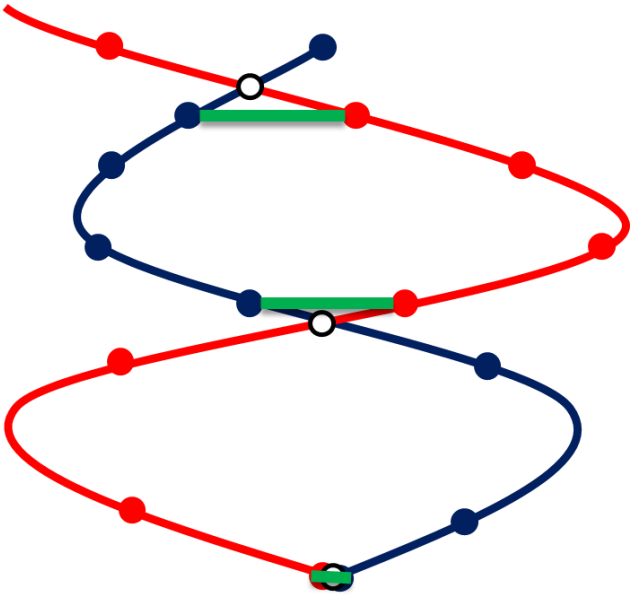


Figura 47 – Elementos de contatos criados com base nos critérios da formulação proposta.

6.6.2 Resultados desta lógica de detecção de contatos

Comparou-se a quantidade de elementos criados com a quantidade teórica para situações de números de arames variados, que mostram-se sempre compatíveis. Adicionalmente, criou-se um método gráfico para esta checagem, ilustrado na Figura 48. Utilizou-se o UFCad para gerar um arquivo com os pares de nós e suas respectivas coordenadas de cada um dos elementos do tipo *bonded* criados. Utilizou-se o programa Matlab para pós-processar estes resultados, convertendo os valores das coordenadas para o sistema cartesiano e respectiva plotagem em um sistema tridimensional.

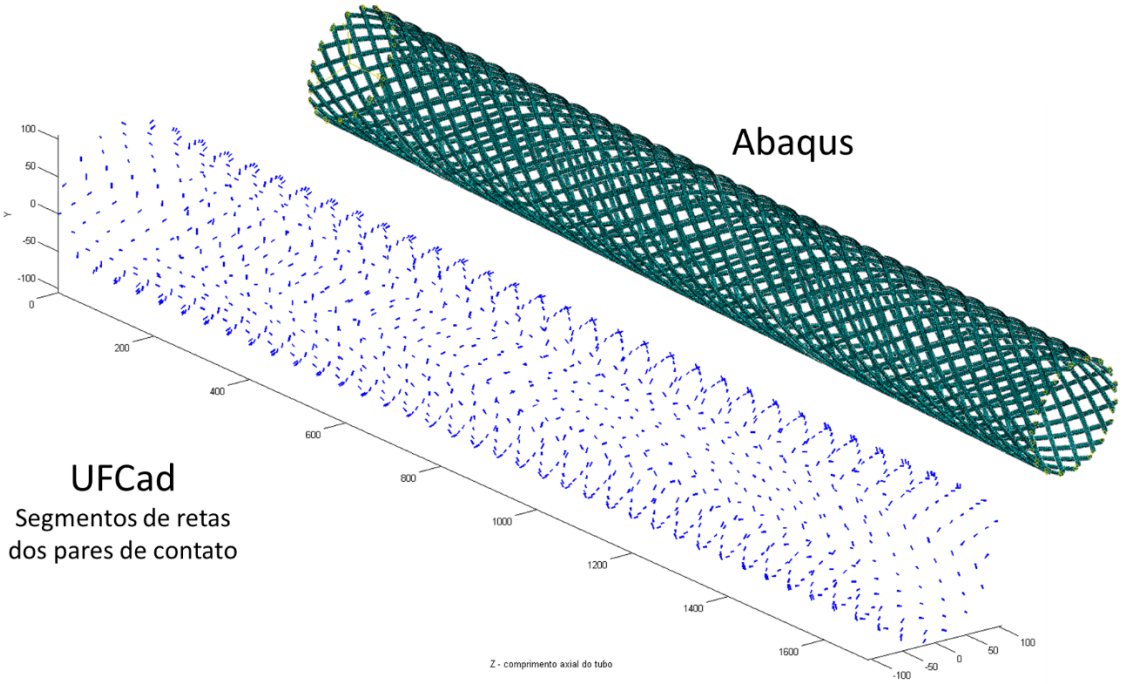


Figura 48 – Recurso gráfico gerado através do programa Matlab para verificar a formação dos pares de contato.

6.7 Modificação 07 – Alteração na Geração de Malhas dos Elementos de Contato Tipo Bridge

Durante a etapa de simulação do caso de estudo (que será apresentado no Capítulo 0), encontrou-se uma falha na geração de malhas dos elementos de contato do tipo bridge. O programa sempre gerava uma quantidade de elementos bridge suficientes

para conectar apenas um arame da armadura de tração. Com isso, para armaduras de tração com mais do que um arame, era gerada uma quantidade insuficiente de elementos *bridge*, deixando de conectar a maior parte dos nós que deveriam estar unidos, invalidando completamente a utilização deste elemento. Após a identificação e correção do problema, todos os nós de todas as armaduras de tração passaram a ser computados, o que foi fundamental para a convergência do problema.

7. RESOLUÇÃO DO SISTEMA LINEAR

A resolução do sistema linear é uma etapa crítica ao funcionamento do programa "UFCad", com reflexos sobre o tempo total de simulação e também ao consumo total de memória. Ao longo deste trabalho, buscou-se várias soluções para contornar as restrições e limitações relativas a solução de sistema linear. Estas ferramentas serão apresentadas nos subitens a seguir.

7.1 Bibliotecas e Métodos de Solução Testados

7.1.1 Math.NET – Numerics – Solver

A biblioteca "*Math.NET – Numerics*" dispõe de uma série de técnicas para a resolução de sistemas lineares que se aplicam tanto para matrizes densas quanto para esparsas. Estas técnicas são:

- Decomposição LU
- Decomposição QR
- Decomposição Gram-Schmidt
- Decomposição SVD
- Decomposição Cholesky
- Decomposição EVD

No início deste trabalho, o programa "UFCad" empregava a biblioteca "*Math.Net Numerics*" para armazenar dados na forma de matrizes e vetores densos por meio de classes específicas por ela disponibilizadas, e também a decomposição do tipo LU para a resolução de sistemas lineares. Durante a realização deste trabalho, testou-se todas as técnicas de resolução listas acima, tanto para sistemas lineares de formato denso quanto esparso. Os métodos de decomposição de Cholesky e EVD exigem matrizes simétricas e portanto não puderam ser utilizados para casos de contato com atrito.

No entanto, estes métodos de solução não são paralelizados e também otimizados para sistemas de grande escala. Este fato inviabiliza a utilização destes métodos para a resolução de sistemas lineares de médio e elevado número de graus de liberdade.

7.1.2 Math.NET – Numerics – Solver Iterative

Durante este trabalho também foram testados métodos iterativos de solução disponibilizados pela biblioteca *Math.Net – Numerics*. Este método de solução requer a definição dos seguintes itens:

$$x = A.\text{SolveIterative}(\text{b}, \text{Iterative solver}, \text{Stop criteria}, \text{Preconditioner})$$

Com os seguintes precondicionadores disponíveis:

- **MILU0Preconditioner()** – "*simple MILU(0) preconditioner*";
- **ILU0Preconditioner()** – "*incomplete, level 0, LU factorization preconditioner*";
- **ILUTPPreconditioner()** – "*incomplete LU factorization with drop tolerance and partial pivoting*".

Os métodos iterativos de solução disponíveis nesta biblioteca são:

- **BiCgStab()** – "*Bi-Conjugate Gradient stabilized iterative matrix solver*";
- **GpBiCg()** – "*Generalized Product Bi-Conjugate Gradient iterative matrix solver*";
- **TFQMR()** – "*Transpose Free Quasi-Minimal Residual (TFQMR) iterative matrix solver*";
- **MlkBiCgStab()** – "*Multiple-Lanczos Bi-Conjugate Gradient stabilized iterative matrix solver*".

Existem uma série de critérios de parada disponíveis e foi selecionado o critério de residual. Assim, o método somente iria chegar ao seu fim quando o residual fosse menor que o valor especificado.

A grande vantagem deste método é que ele pode ser aplicado para a resolução de sistemas lineares esparsos, e as técnicas de precondicionamento podem ajudar a melhorar a convergência de sistemas mal escalados.

Estes métodos iterativos mostraram-se mais eficientes que os métodos do item 7.1.1, porém a sua utilização ainda não é viável para sistemas de grande escala. Além disso, métodos derivados do método de gradiente conjugado não podem ser aplicados para a resolução de sistemas lineares assimétricos.

7.1.3 Math.NET – Numerics – MKL

Ao longo da elaboração deste trabalho, incorporou-se ao programa “UFCad” uma biblioteca adicional, a “*Math.Net Numerics MKL*”, que é uma derivação *freeware* da biblioteca profissional “*Intel® Math Kernel Library*”.

Esta biblioteca dispõe de um *solver LU* para a resolução de sistemas lineares caracterizado pela alta eficiência de execução e elevado grau de paralelização, o que permitiu uma redução considerável do tempo total de simulação.

No entanto, uma grande desvantagem desta biblioteca é que ela não permite a utilização de matrizes esparsas na resolução do sistema linear. Com isso, apesar da ótima eficiência da mesma, não é possível utilizá-la para analisar modelos de macroelementos finitos com elevado número de graus de liberdade. O computador utilizado na avaliação do solver MKL dispunha de 8 GB de memória RAM, o que permitiu simular casos em que a matriz global de rigidez de ordem até [16.000 x 16.000]. Se as dimensões da matriz global de rigidez ultrapassassem o limite especificado, o consumo de memória excedia o disponível na CPU e a execução era abortada.

A Figura 49 mostra os resultados de simulação para um caso contendo um arame de armadura descrito por elementos de hélice interligados através de elementos contato rígido com elementos cilíndricos. Nota-se neste caso que tanto o consumo máximo de memória e o tempo total de simulação possuem relação cúbica com o tamanho da matriz global de rigidez. Este gráfico reforça a conclusão de que apesar a ferramenta MKL ser muito eficiente, o aumento no consumo de memória ocorre de forma muito acentuada, limitando a sua utilização.

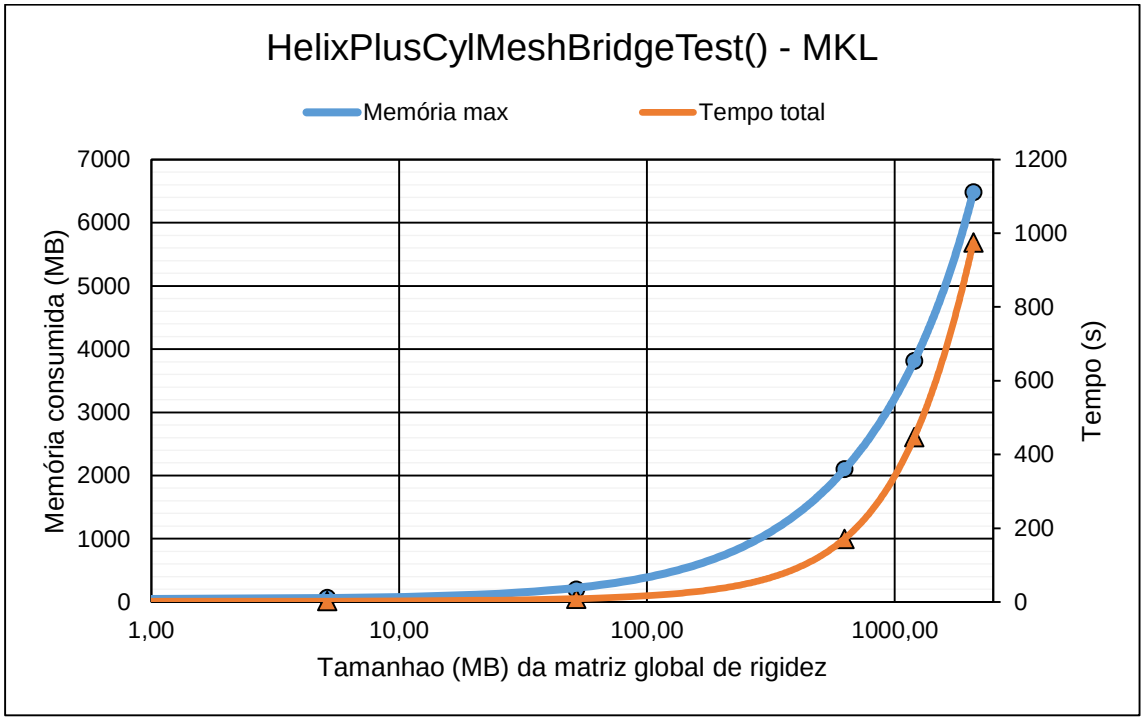


Figura 49 – Resultados de simulação o caso "*HelixPlusCylMeshBridgeTest()*" (um arame de armadura descrito por elementos de hélice interligados através de elementos de contato rígido com elementos cilíndricos): consumo máximo de memória e tempo de simulação em função do tamanho, em megabytes, da matriz global de rigidez.

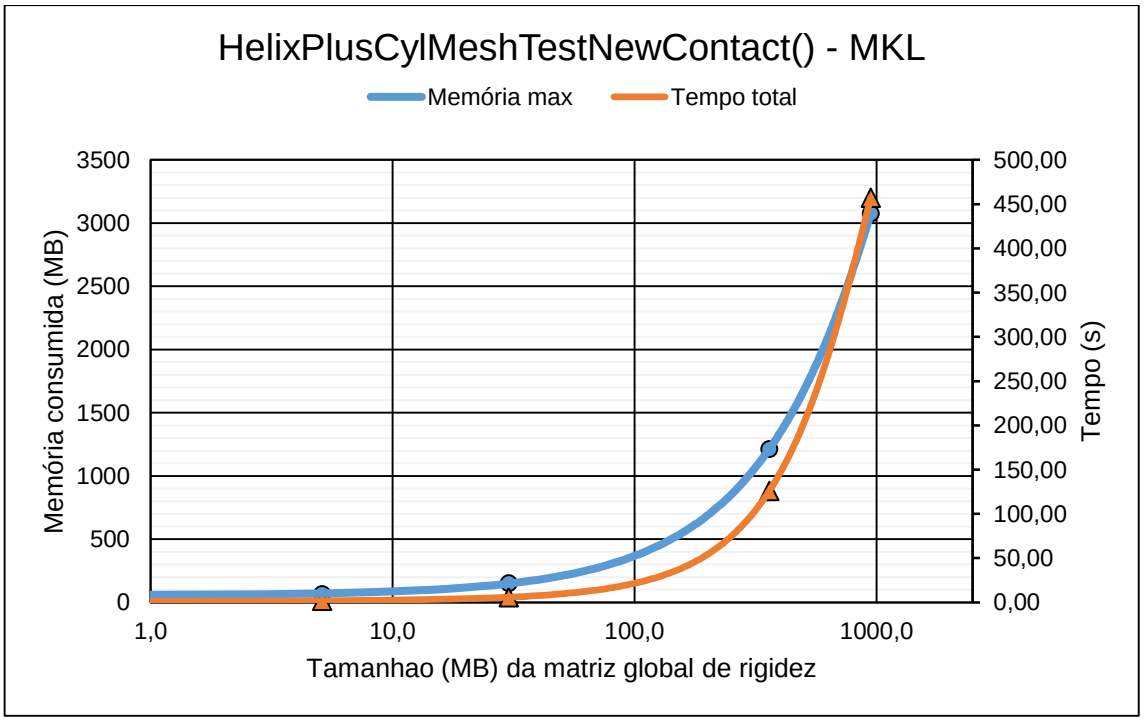


Figura 50 - Resultados de simulação o caso "*HelixPlusCylMeshTestNewContact()*" (um arame de armadura descrito por elementos de hélice interligados com elementos cilíndricos através de

elementos de contato que permitem *sticking* e *sliding*): consumo máximo de memória e tempo de simulação em função do tamanho, em megabytes, da matriz global de rigidez.

7.1.4 GMRES – Biblioteca Própria

Ao longo do trabalho, desenvolveu-se uma biblioteca própria para a solução de sistemas lineares. Implementou-se um método de solução consolidado na computação numérica, o método "*Generalized Minimal Residual Method*" (GMRES), geralmente empregado na solução de grandes sistemas lineares esparsos e não simétricos. Utilizou-se como referência bibliográfica para esta atividade o livro "*Iterative Methods for Sparse Linear Systems*" do autor Yousef Saad. O algoritmo pode ser visto na Figura 51.

1. *Compute* $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, *and* $v_1 := r_0/\beta$
2. *For* $j = 1, 2, \ldots, m$ *Do:*
3. *Compute* $w_j := Av_j$
4. *For* $i = 1, \ldots, j$ *Do:*
5. $h_{ij} := (w_j, v_i)$
6. $w_j := w_j - h_{ij}v_i$
7. *EndDo*
8. $h_{j+1,j} = \|w_j\|_2$. *If* $h_{j+1,j} = 0$ *set* $m := j$ *and go to 11*
9. $v_{j+1} = w_j/h_{j+1,j}$
10. *EndDo*
11. *Define the* $(m+1) \times m$ *Hessenberg matrix* $\bar{H}_m = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$.
12. *Compute* y_m *the minimizer of* $\|\beta e_1 - \bar{H}_m y\|_2$ *and* $x_m = x_0 + V_m y_m$.

Figura 51 – Algoritmo GMRES.

Ao implementar este algoritmo, procurou-se armazenar os dados na forma de matrizes esparsas, além da utilização de suas propriedades para a redução do número de iterações necessárias para se obter a solução. No entanto, como este método de solução ainda não está otimizado, não está paralelizado e ainda não possui pré-condicionador implementado. Portanto este método de solução ainda não pode ser empregado pelo programa "*UFCad*".

7.1.5 Bibliotecas profissionais (pagas) e PARDISO

Uma alternativa para solucionar a curto prazo o problema de limitação de memória e processamento é a aquisição de bibliotecas numéricas comerciais, projetadas,

paralelizadas e otimizadas para a resolução de sistemas lineares esparsos. Esta alternativa no entanto implica em altos custos, não sendo possível adotá-la no momento.

Outra alternativa, é a utilização do "*PARDISO 5.0.0 Solver Project*", uma biblioteca preparada para a finalidade descrita acima e que dispõe de licença grátis para estudante durante o período de um ano. No entanto, está biblioteca funciona apenas para a linguagem de computação C++, o que exige a conversão integral do programa "*UFCad*" para esta linguagem ou a exportação dos dados necessários à resolução do sistema linear para conversão ao C++.

Ainda não há uma alternativa ótima para a resolução do sistema linear. Todas as alternativas disponíveis até o momento possuem vantagens, desvantagens e principalmente limitações. A Tabela 4 resume todas estas informações, com o intuito de facilitar a comparação entre as alternativas de solução.

7.2 Alternativa empregada

Devido à baixa eficiência computacional dos demais métodos, optou-se pela utilização da biblioteca "*Math.Net Numerics MKL*", caracterizada por elevado grau de paralelização e eficiência, e que utiliza 100% dos recursos de processamento disponíveis na CPU durante a solução de sistemas lineares. Comparações realizadas com a biblioteca convencional "*Math.Net Numerics"* indicaram uma redução de duas ordens de grandeza no tempo total de simulação.

No entanto, a biblioteca MKL não pode ser aplicada à resolução de sistemas lineares esparsos. Deste modo, ao se optar por esta biblioteca, não foi possível aproveitar a considerável redução no consumo de memória propiciado pela conversão das matrizes da classe "Solver" para matrizes esparsas (item 6.5).

Portanto, a utilização do UFCad requer a disponibilidade de computadores com elevada quantidade de memória, pois o programa não dispõe de um método eficiente para a resolução de sistemas lineares esparsos, sendo este o principal gargalo existente no momento.

7.3 Resumo dos Métodos de Solução de Sistemas Lineares

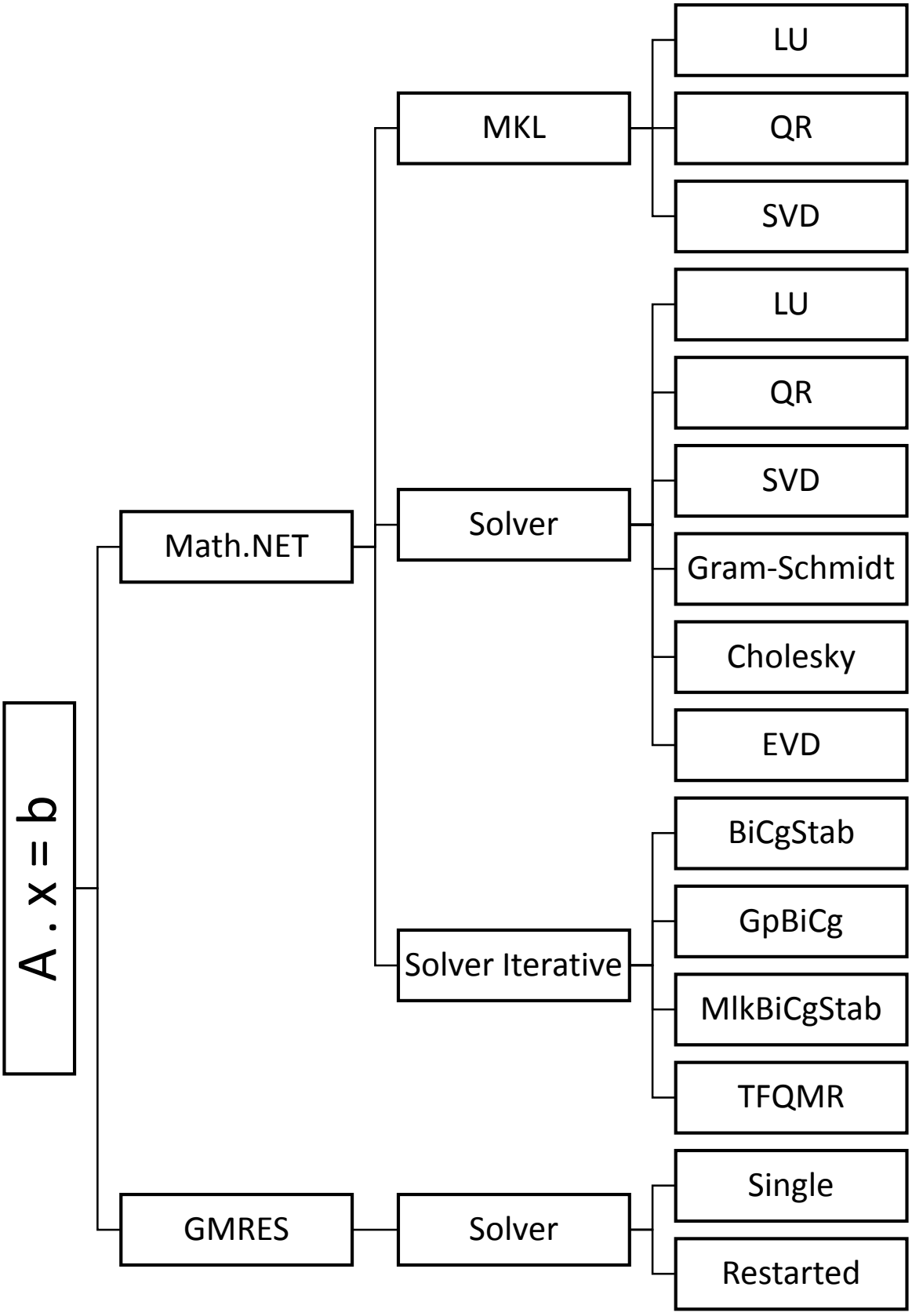


Tabela 4 – Resumo dos principais métodos de resolução de sistemas lineares do tipo A.x = b

Biblioteca	Método	Comando	Características	Limitações	Usos
MKL Math.NET Numerics	LUSolve QRSolve SVDSolve	solver.LUSolve(1, **A**.Values, **A**.RowCount, **b**.Values);	Biblioteca desenvolvida pela Intel, muito eficiente, otimizada e com elevado grau de paralelização.	Funciona apenas com matrizes densas. Consumo exponencial de memória em função do número de graus de liberdade do modelo.	Para modelos com poucos graus de liberdade, em que a quantidade de memória disponível não é limitante.
Math.NET Numerics	Decomposição LU Decomposição QR Decomposição SVD Decomp. Gram-Shmidt	**x** = **A**.LU().Solve(**b**); **x** = **A**.QR().Solve(**b**); **A**.Svd().Solve(**b**, **x**); **A**.GramSchmidt().Solve(**b**, **x**);	Permitem a resolução de sistemas lineares de matrizes e vetores esparsos.	Métodos não paralelizados, de baixa eficiência na resolução de sistemas lineares com matrizes muito grandes.	Quando a biblioteca MKL não for capaz de resolver o sistema linear (limitações de memória), porém com perda expressiva de eficiência.
	Decomposição Cholesky Decomposição EVD	**A**.Cholesky().Solve(**b**, **x**); **A**.Evd().Solve(**b**, **x**);	Permitem a resolução de sistemas lineares de matrizes e vetores esparsos.	A matriz **A** deve ser simétrica e positiva definida. Não pode ser utilizada quando há atrito no modelo.	Apenas para casos em que a matriz global de rigidez é simétrica.
Math.NET Numerics Iterative Solver	BiCgStab() GpBiCg() TFQMR() MlkBiCgStab()	MILU0Preconditioner() ILU0Preconditioner() ILUTPPreconditioner() **x = A**.SolveIterative(**b**, solverteste, iterator, preconditioner)	Permite a solução de sistemas lineares com matrizes e vetores esparsos. Performance superior aos métodos comuns da biblioteca *Math.NET Numerics*.	Performance bem inferior à biblioteca MKL. Algoritmos adequados para matrizes simétricas. Não adequado à solução de problemas de grande escala.	Quando a matriz global de rigidez é armazenada de forma esparsa. Pré-condicionamento pode melhorar convergência da solução.

Biblioteca própria	GMRES	Código próprio	Permite a manipulação de matrizes esparsas. Método GMRES, indicado para sistemas esparsos e não simétricos.	Baixa eficiência, pois ainda não foi otimizado.	Biblioteca ainda em desenvolvimento. Deve ser paralelizada e otimizada.
PARDISO	LUSolve	Comandos em C++	A biblioteca PARDISO dispõe de um solver LU para sistemas lineares com elevado grau de paralelização. Também permite a manipulação de matrizes esparsas.	Não disponível para C#. Deve-se converter o código para C++ ou criar um "*wrapper"* para resolver apenas o sistema linear em C++.	Modelos de elementos finitos de grande escala, pois reduz o consumo de memória e o tempo para resolver o sistema linear.

8. VALIDAÇÃO DE UM CASO DE ESTUDO COM O PROGRAMA UFCAD

Com o objetivo de validar um caso de estudo através do programa UFCad, realizou-se uma comparação com o programa Abaqus. Atualmente desenvolvido pela *Dassault Systemes S.A.*, o Abaqus é uma consolidada ferramenta de análise por elementos finitos que está há mais de 35 anos no mercado.

Neste capítulo, será especificado o caso de estudo, ou seja, os componentes de um tubo flexível que serão modelados em ambos os programas, com o intuito de compará-los em vários aspectos, desde resultados até performance.

No item 8.1 o caso de estudo será detalhado, bem como os motivos que levaram a escolha do mesmo. Nos itens 8.2 e 8.3 serão apresentados os valores dos parâmetros geométricos e as propriedades de materiais, respectivamente.

Nos itens 8.4 e 8.5 serão discutidos os detalhes de implementação em cada programa, incluindo as dificuldades encontradas durante o processo e as alternativas encontradas para contorna-las.

Por fim, no item 8.6 será realizada uma ampla comparação entre ambos os programas, com base em critérios especificados, como qualidade dos resultados, tempo de análise, facilidade de implementação, entre outros.

8.1 Definição do caso de estudo

Como o programa UFCad ainda não dispõe de todos os macroelementos finitos necessários para modelar um tubo flexível completo, adotou-se um tubo mais simples para esta tarefa. Inicialmente, considerou-se o caso de estudo composto pelos seguintes componentes: um núcleo rígido, duas camadas de armaduras de tração e uma capa plástica, todos conectados por elementos de contato que permitissem deslocamentos normais e tangenciais com atrito. No entanto, também não foi possível adotar este caso, pois este o de elemento de contato utilizado ainda apresenta problemas em sua implementação computacional.

Portanto, adotou-se um caso mais simples, ilustrado na Figura 52, porém factível com as ferramentas disponíveis até o momento. Este caso é composto pelos seguintes componentes:

- Armadura de tração interna, com 16 arames;
- Armadura de tração interna, com 18 arames;
- Capa plástica envolvendo as duas armaduras.

Além disso, estes componentes foram conectados rigidamente nos pontos de contato, definidos pelas intersecções entre os arames das armaduras interna e externa de tração, além das interseções dos arames da armadura externa de tração com a capa plástica.

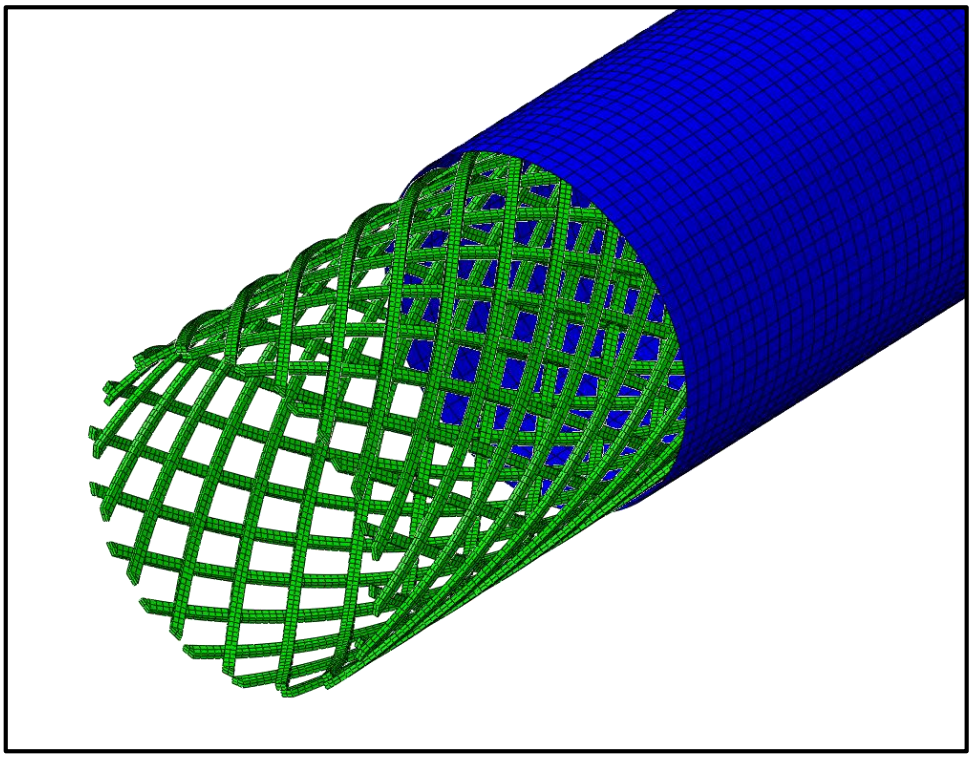


Figura 52 - Tubo flexível adotado para o caso de estudo.

8.2 Propriedades e parâmetros geométricos dos componentes

Neste item estão apresentados os parâmetros geométricos para o caso de estudo escolhido. Para cada tipo componente, resumiu-se as informações de parâmetros nas Tabelas de Tabela 5 à Tabela 7.

Tabela 5 – Propriedades e parâmetros da capa plástica.

Capa Plástica	
Espessura	7 mm
Diâmetro médio	213,5 mm
Material	Polietileno de alta densidade (Tabela 9)

Tabela 6 – Propriedades e parâmetros da armadura externa de tração.

Armadura Externa	
Número de arames	18
Ângulo de assentamento	-38°
Diâmetro médio	206,5 mm
Seção transversal (L x A)	8 mm x 4 mm
Material	Aço1020 (Tabela 8)

Tabela 7 – Propriedades e parâmetros da armadura interna de tração.

Armadura Interna	
Número de arames	16
Ângulo de assentamento	36°
Diâmetro médio	202,5 mm
Seção transversal (L x A)	8 mm x 4 mm
Material	Aço 1020 (Tabela 8)

8.3 Materiais do modelo

As armaduras internas e externas de tração foram modeladas utilizando-se o material Aço 1020, cujas propriedades encontram-se na
Tabela 8.

É importante notar que o programa “UFCad” não permite a adoção de não-linearidades de material. Por este motivo considerou-se o aço 1020 como sendo isotrópico linear elástico, utilizando para tal somente as propriedades somente do

trecho de módulo secante da curva de tensão e deformação. Já para o programa "*Abaqus*", por permitir a utilização de tal recurso, foi adotada a propriedade de material elástico não-linear (com módulo secante e módulo tangente).

Tabela 8 – Propriedades do aço 1020.

Aço 1020	
Formulação	Isotrópica
Densidade	8,05 E-9 ton/mm^3
Módulo de elasticidade	207 GPa
Coeficiente de Poisson	0,30
Módulo tangente	1172 MPa
Tensão de escoamento	650 MPa
Comportamento (UFCad)	Linear elástico
Comportamento (Abaqus)	Elástico não-linear
Deformação plástica	Figura 53

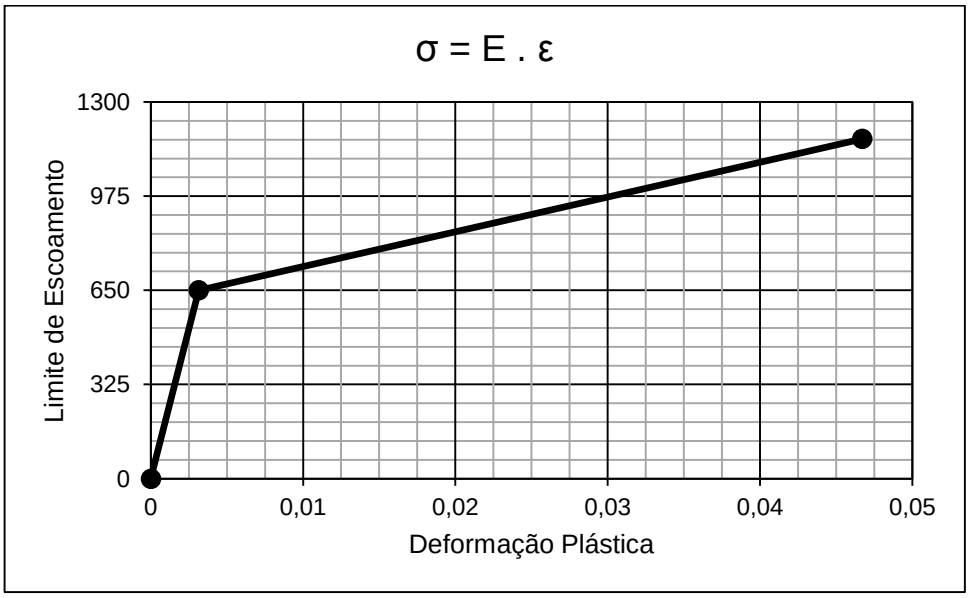


Figura 53 – Curva de tensão deformação do aço 1020.

Para a capa plástica, adotou-se como material um polietileno de alta densidade (HDPE – "*High-density polyethylene*"), cujas propriedades encontram-se na

Tabela 9. Neste caso também há uma diferença entre o comportamento deste material nos dois programas de elementos finitos, sendo linear elástico no "*UFCad*" e elastoplástico no "*Abaqus*".

Tabela 9 – Propriedades do material polietileno.

Polietileno (HDPE)	
Formulação	Isotrópica
Densidade	9,41E-10 ton/mm^3
Módulo de Young	570,88 Mpa
Coeficiente de Poisson	0,45
Tensão de escoamento	20,74 MPa
Comportamento (UFCad)	Linear elástico
Comportamento (Abaqus)	Elastoplástico
Deformação plástica	Figura 54

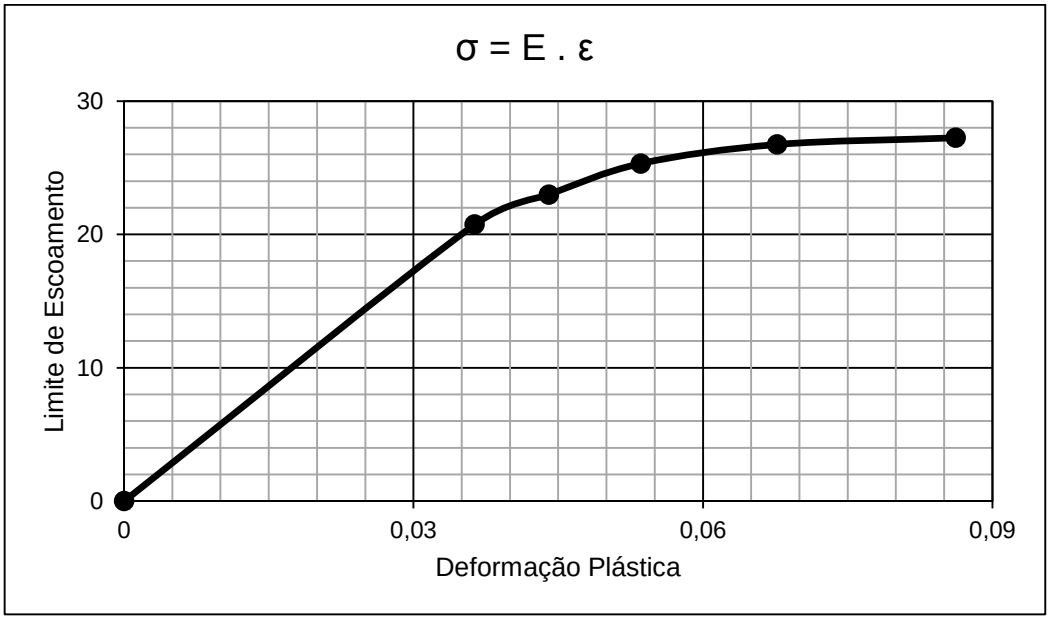


Figura 54 – Curva de tensão deformação do polietileno (HDPE).

8.4 Análise realizada através do *software* Abaqus

Este item contém as informações importantes à resolução do caso de estudo através do *software* Abaqus. Serão detalhadas as principais características da

implementação (item 8.4.1); os tipos de elementos e características das malhas de elementos finitos (item 8.4.2); os problemas com a implementação do contato e as alternativas encontradas para a solução (item 8.4.3); as condições de contorno (item 8.4.4); as precauções quanto a ocorrência de plastificação, o que poderia inviabilizar a comparação com o UFCad (item 8.4.5); e finalmente os resultados (item 8.4.6).

8.4.1 Características da Implementação

Para implementar o modelo, criou-se uma macro com os comandos do programa Abaqus, permitindo a parametrização do mesmo e versatilidade para se alterar os valores dos parâmetros, como, por exemplo, alterações no número de arames das armaduras.

Foram feitos alguns testes com os métodos implícitos e explícitos de integração, mas optou-se pelo método implícito, por este permitir um maior *timestep* (maior fração de carga aplicada por iteração), resultando em um tempo menor de simulação; e também por este método permitir uma implementação mais simples de contato, pois no método implícito foi possível criar dois conjuntos de áreas (método detalhado no item 8.4.3), ao passo que o método explícito requeria a trabalhosa criação de pares individuais de contatos, inviabilizando a parametrização do modelo e consequentemente a utilização deste método de integração.

8.4.2 Tipos de Elementos e Características das Malhas de Elementos

Em função da simplicidade geométrica da capa plástica, ela foi modelada utilizando-se elementos de casca, ao invés de sólido, propiciando uma série de vantagens computacionais, como simplicidade de implementação e ganho de performance. Ao mesmo tempo, o elemento de casca também permite simulações de interações de contato com objetos sólidos. Na Tabela 10 encontram-se as características do elemento empregado na modelagem da capa plástica.

Tabela 10 – Propriedades do elemento utilizado para modelar a capa plástica

Capa plástica	
Element Library:	Standard
Family:	Shell
Geometric Order:	Quadratic
Type:	S8R: 8-node doubly curved thick shell
Redunced integration:	Yes
DOF per node	6

A geometria simples da capa plástica permitiu a criação de uma malha bem estruturada de elementos, Figura 55.

Figura 55 – Malha estrutura de elementos finitos da capa plástica.

Tanto os arames da armadura interna de tração quanto os arames da armadura externa de tração foram modelados com o mesmo tipo de elemento, cujas propriedades encontram-se na Tabela 11. Optou-se pela utilização de elementos sólidos, e não por elementos de viga, devido às dificuldades e problemas de convergência ao se utilizar vigas para descrevê-las.

Tabela 11 - Propriedades do elemento utilizado para modelar as armaduras de tração.

Armaduras de tração	
Element Library:	Standard
Family:	3D Stress
Geometric Order:	Linear
Type:	C3D8: 8-node linear brick
Redunced integration:	No

Criou-se uma malha estruturada para discretizar estes arames, com exceção das extremidades, que exigiram elementos triangulares em função da curvatura destes componentes. Podem ser vistas nas Figura 56 e Figura 57 as malhas de elementos finitos gerada pelo programa para as armaduras de tração.

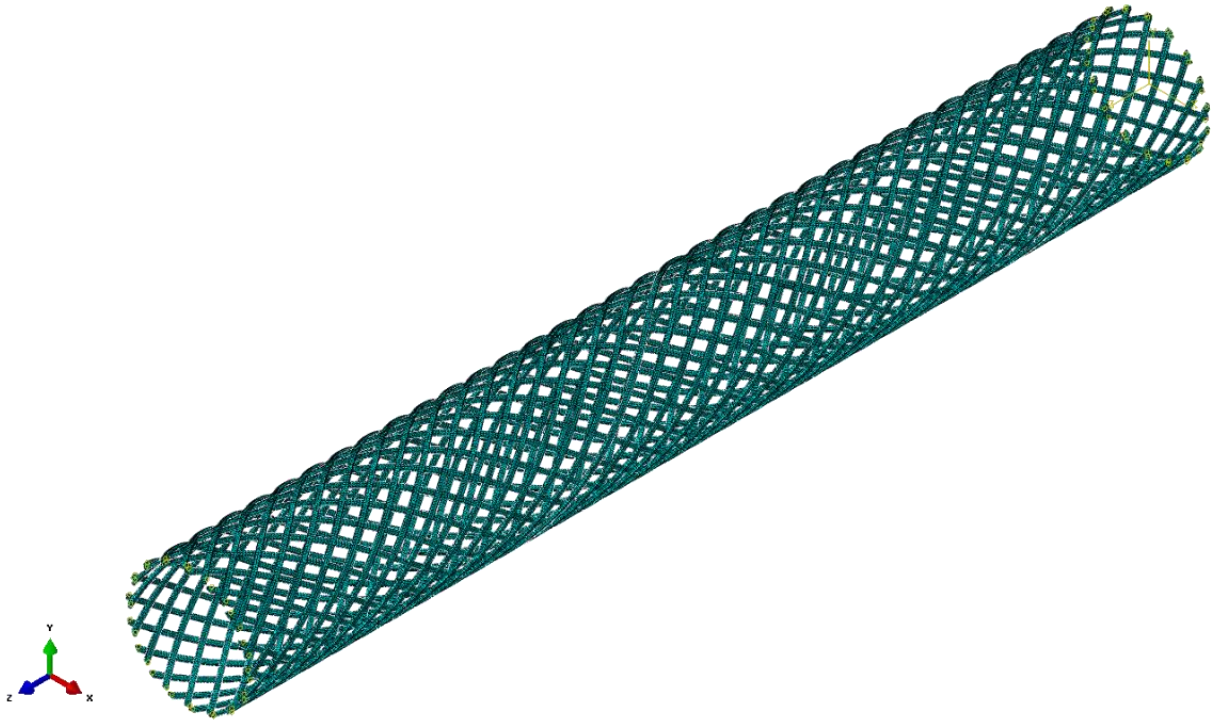


Figura 56 - Malha de elementos finitos dos arames das armaduras de tração.

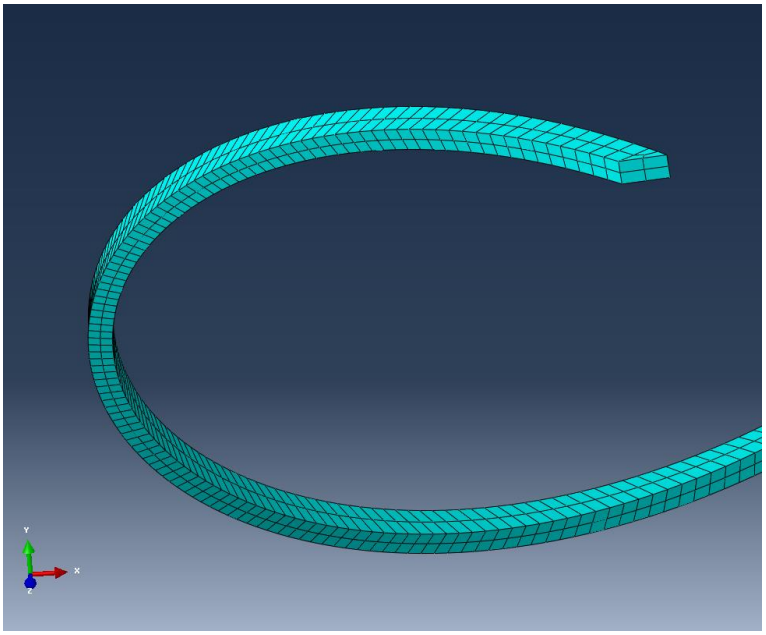


Figura 57 - Imagem ampliada da malha de elementos finitos de um arame de tração.

8.4.3 Implementação do Contato

Durante esta atividade, verificou-se que o programa apresentava comportamentos completamente distintos conforme o tipo de integração escolhido, os métodos implícitos e explícitos.

Devido à característica do contato escolhido, totalmente rígido (*bonded*), quando o método de solução é explícito, o Abaqus não permite a utilização do útil comando "General Contact", o qual identifica automaticamente todos os pares de contato. Isto significa que os pares de contato devem ser informados manualmente ao programa antes da resolução do problema.

Para o contato entre a armadura externa de tração com a armadura interna de tração, por exemplo, um método prático para identificar quais superfícies estão em contato é criar dois conjuntos de superfícies:

- Um conjunto de superfícies composto pelas superfícies externas dos arames da armadura interna de tração;
- Um conjunto de superfícies compostos pelas superfícies internas dos arames da armadura externa de tração.

No entanto, quando o método explícito de integração é selecionado, esta implementação não funciona, pois o programa não permite o contato entre duas

superfícies não contínuas (cada conjunto é composto por uma série de áreas não contínuas). Assim, para continuar a utilizar o método explícito, devem ser especificadas de forma manual todas as combinações de áreas possíveis entre as armaduras internas e externas. Este fato torna a implementação extremamente trabalhosa e inviabiliza qualquer modificação posterior no modelo.

Ao utilizar a formulação implícita de integração, o programa permitiu a utilização dos dois conjuntos de superfícies, mostrando-se um recurso muito útil de solução. Além da vantagem computacional, mencionada no item 8.4.1, a opção pelo método implícito de integração facilitou bastante a implementação do contato, sendo, portanto, o método escolhido.

8.4.4 Condições de Contorno

As condições de contorno do modelo criado estão ilustradas na Figura 58. Selecionou-se os nós da seção transversal da extremidade direita desta figura e todos os graus de liberdade destes nós foram impostos iguais a zero. Quanto aos nós da seção transversal da extremidade esquerda, aplicou-se o deslocamento imposto, cujo valor foi definido pela metodologia apresentada no item 8.4.5.

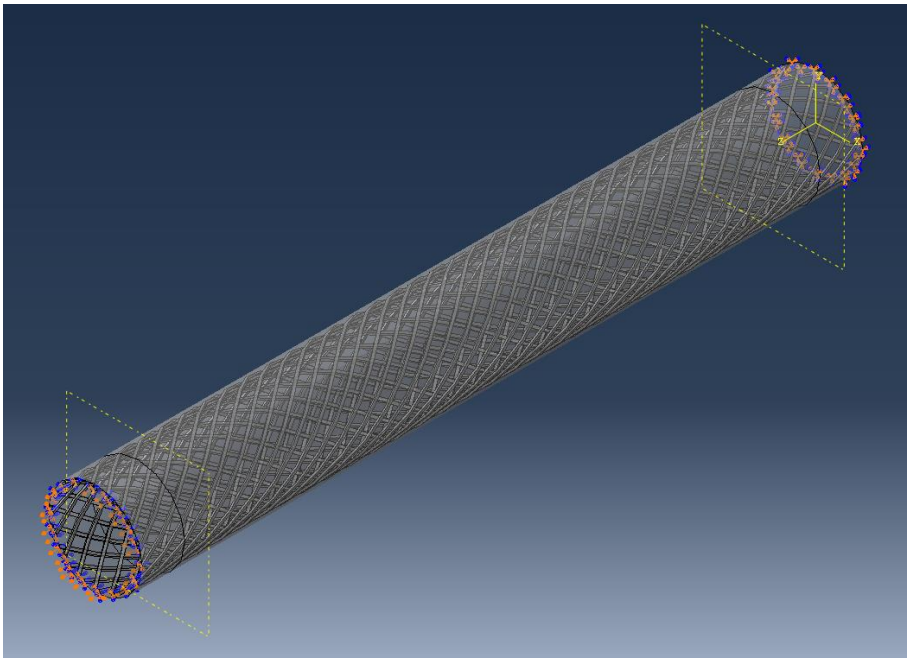


Figura 58 - Condições de contorno do modelo.

8.4.5 Análise de Plastificação

Por ser um programa completo, o Abaqus permite à resolução do problema a inclusão de não-linearidades geométricas e também a utilização de materiais reais (incluindo plastificação e não-linearidades de materiais). Estes efeitos não lineares se acentuam a medida em que as deformações geradas pelo deslocamento imposto aumentam. Como o UFCad permite a solução apenas de problemas lineares (tanto geométricas, quanto de materiais), foi necessária uma análise de tensão para adotar um deslocamento imposto que permitisse a comparação adequada entre ambos os programas. A Figura 59, a Figura 60 e Figura 61 mostram resultados de tensões de Von Mises para deslocamentos impostos de 40 mm, 20 mm e 10 mm, respectivamente. Em todos os casos notou-se uma singularidade, gerando concentrações de tensão, nos pontos de intersecção entre as armaduras, o que pode ser visto na Figura 62. Acredita-se que isso ocorra devido ao fato de as armaduras estarem coladas umas às outras, impedindo-as de deslocar ou rotacionar, e de se acomodarem em uma configuração que minimize a energia interna do sistema. O valor desta tensão poderia ser reduzido se fosse utilizado elementos de contato que permitissem deslocamentos normais e tangenciais com atrito, o que não foi possível devido aos problemas já mencionados.

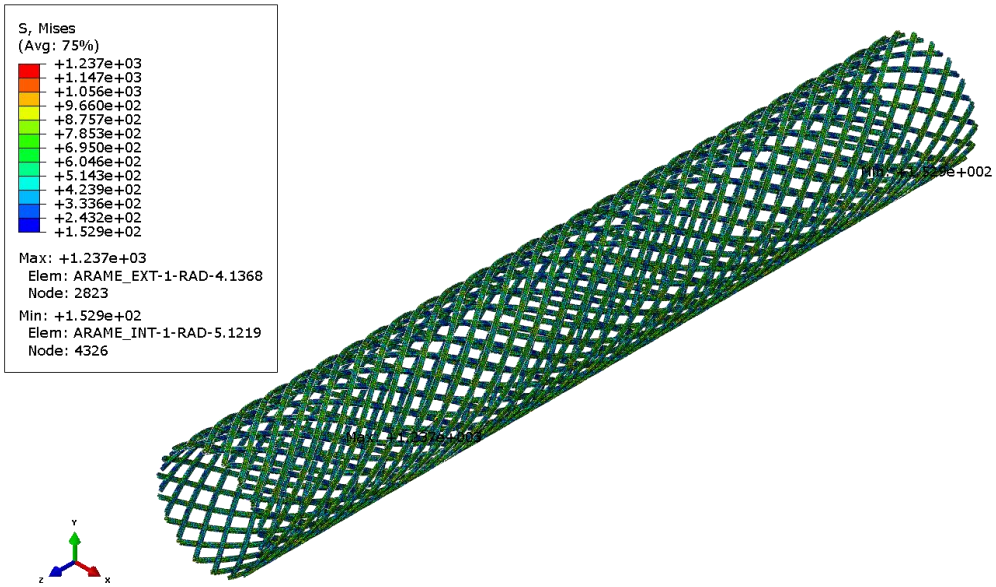


Figura 59 – Tensões de Von Mises para um deslocamento imposto de 40mm. $\sigma_{Max} = 1273\ MPa$, valor bem acima da tensão de plastificação.

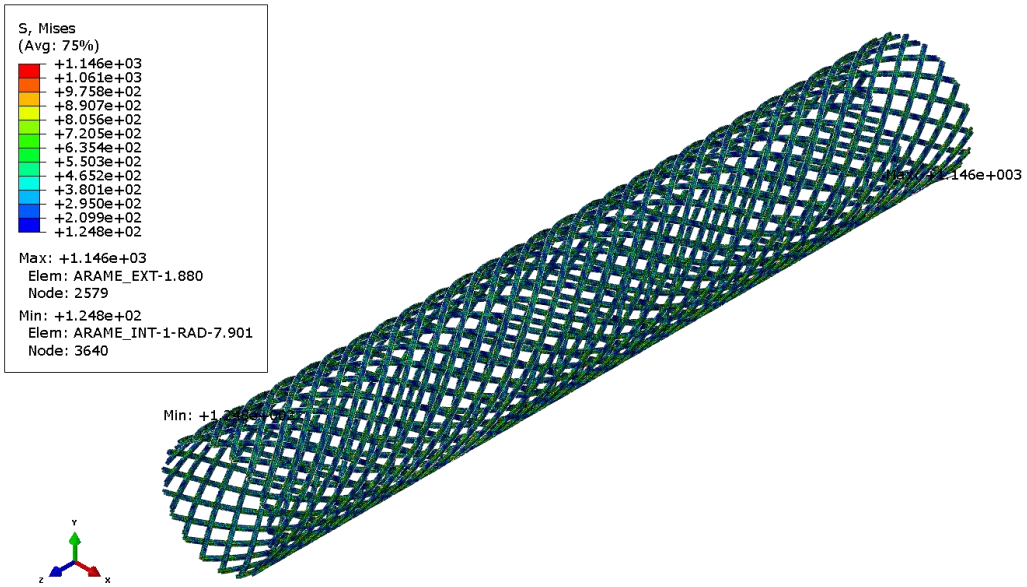


Figura 60 – Tensões de Von Mises para um deslocamento imposto de 20mm. $\sigma_{Max} = 1146\ MPa$, valor bem acima da tensão de plastificação.

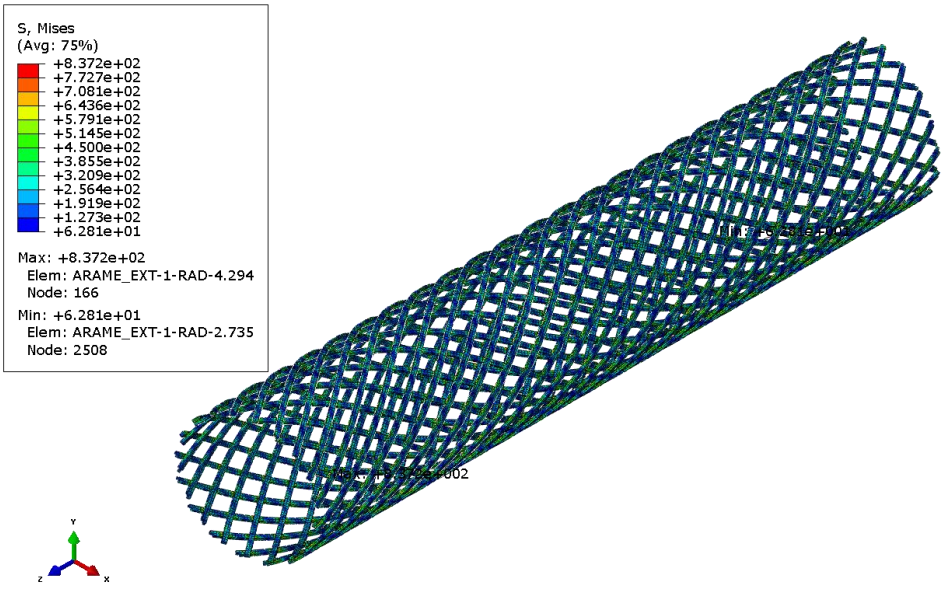


Figura 61 – Tensões de Von Mises para um deslocamento imposto de 10mm. $\sigma_{Max} = 837\ MPa$, valor próximo à tensão de plastificação do aço 1020.

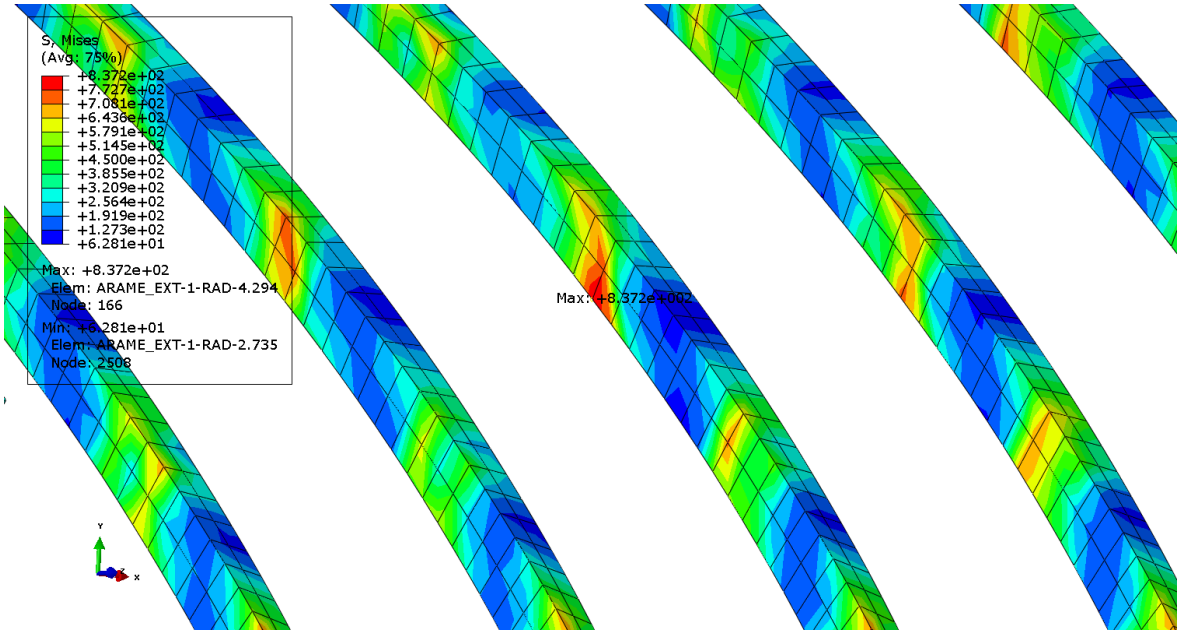


Figura 62 – Plastificação localiza, devido ao contato do tipo *bonded*. Tensões pontuais elevadas, devido ao fato das armaduras não poderem se acomodar em uma configuração de mínima energia.

Com base nos resultados de tensão, adotou-se um deslocamento imposto de 10 mm. Adicionalmente, para este deslocamento imposto realizou-se uma outra comparação, envolvendo os casos:

- Modelo com não-linearidades geométricas e material plástico;
- Modelo sem não-linearidades geométricas e material plástico;
- Modelo sem não-linearidades geométricas e material elástico.

O objetivo desta análise é verificar a validade de se comparar os resultados de um modelo que inclua não-linearidades geométricas e materiais plásticos (Abaqus) com um modelo totalmente linear (Abaqus ou UFCad). Os resultados desta análise encontram-se na Figura 63 e na Figura 64.

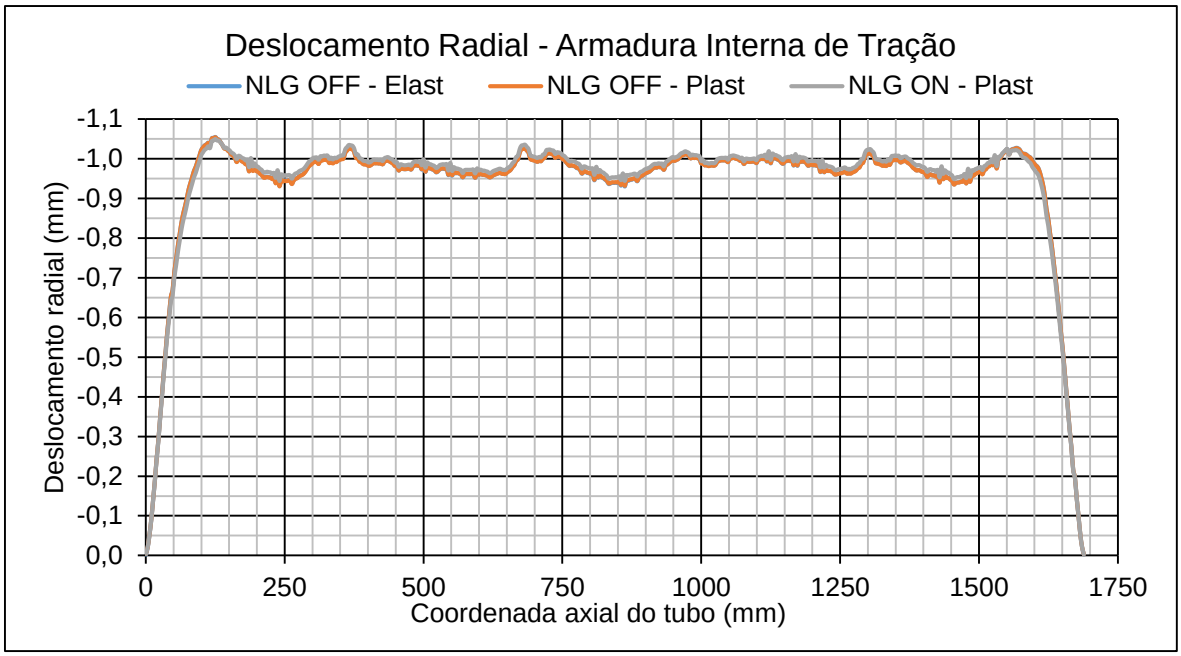


Figura 63 – Comparação de deslocamentos radiais da armadura interna de tração de modelos com e sem não-linearidades geométricas e materiais plásticos e elásticos. NLG: não linearidades geométricas. Elast: material elástico. Plast: material plástico.

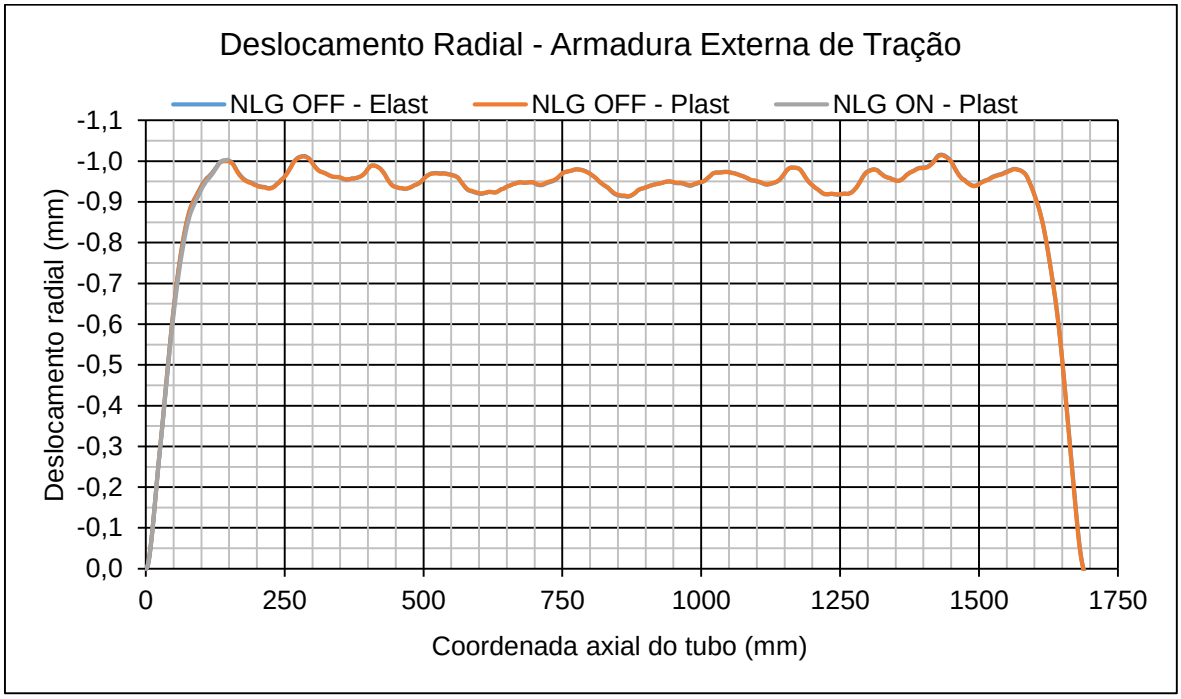


Figura 64 – Comparação de deslocamentos radiais da armadura externa de tração de modelos com e sem não-linearidades geométricas e materiais plásticos e elásticos. NLG: não linearidades geométricas. Elast: material elástico. Plast: material plástico.

Conclui-se que, para o deslocamento imposto selecionado de 10 mm, as diferenças entre o modelo totalmente linear e o modelo não-linear são apenas residuais. Conforme o esperado, os efeitos de plastificação ocorrem apenas localmente, não interferindo nos deslocamentos radiais das armaduras de tração.

8.4.6 Resultados

Os resultados finais da análise para o caso estabelecido utilizando-se o *software* Abaqus encontram-se na Figura 65. Como os contatos deste caso de estudo foram definidos como rígidos e sem separação, era esperado que a estrutura tivesse um comportamento próximo de um tubo homogêneo simples, com um deslocamento aproximadamente constante no trecho central, o que pode ser visto na Figura 65. Adicionalmente, os deslocamentos radiais nas extremidades devem ser nulos, uma vez que foram impostos pelas condições de contorno do problema.

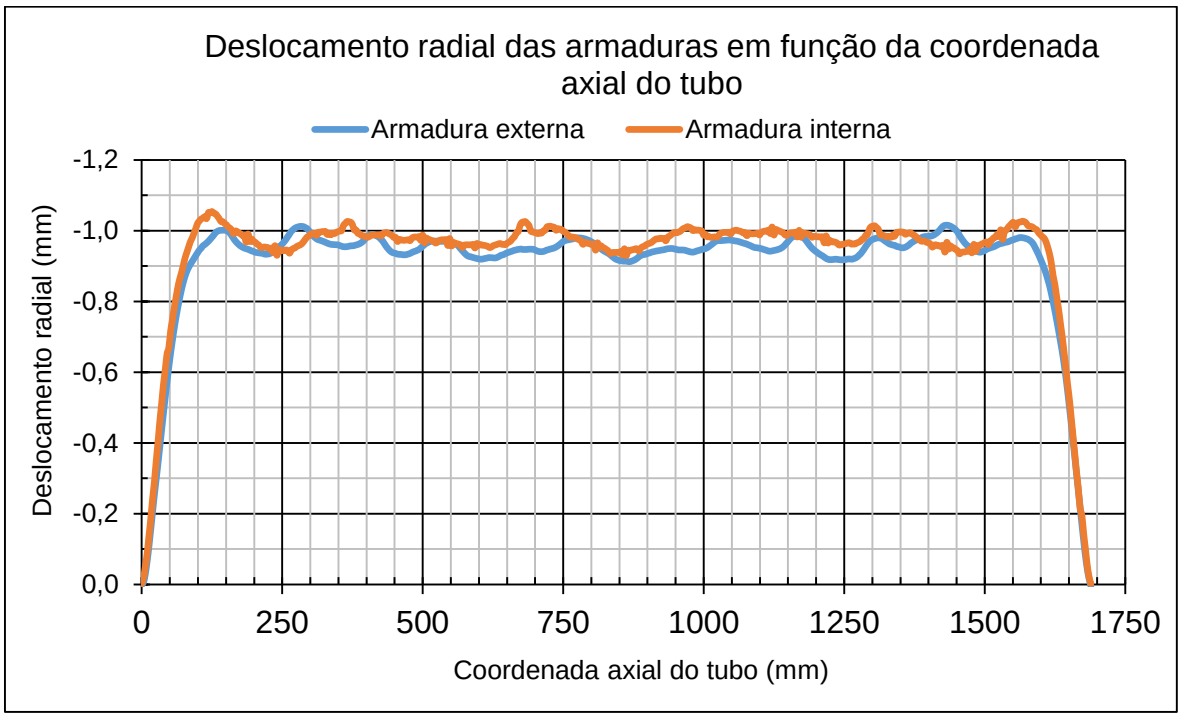


Figura 65 – Resultados do caso de estudo utilizando-se o software Abaqus.

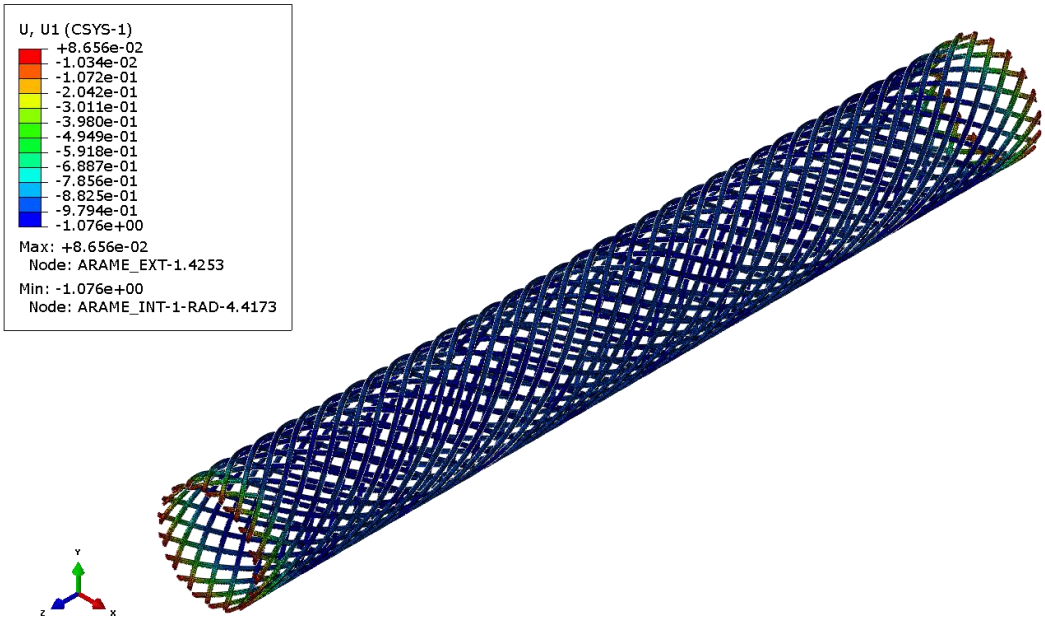


Figura 66 - Deslocamentos radiais das armaduras de tração.

8.5 Análise realizada através do *software* UFCad

O mesmo caso de estudo foi implementado no programa UFCad. Utilizou-se a mesma parametrização aplicada ao programa Abaqus, com o intuito de facilitar eventuais modificações que pudessem ser realizadas no caso de estudo.

Por se tratar de um programa em desenvolvimento, ao longo desta tarefa foram identificados alguns problemas de implementação dos modelos de macroelementos finitos de contato, tanto *bridge* quanto *bonded*, justificando as modificações no código apresentadas nos itens 6.6 e 6.7. Neste ponto, os resultados do programa Abaqus tiveram um papel muito importante, tanto para comparar a qualidade dos resultados do UFCad, quanto para guiar as ações de correção. Foram necessários muitos testes, principalmente para identificar os problemas que implementação, que não eram evidentes a princípio.

8.5.1 Características da implementação

O UFCad ainda não dispõe de interface gráfica e, por isso, a implementação se deu exclusivamente por linha de comando. O código foi implementado de forma estruturada e está segmentado em classes:

- "***Materials***": utilizada para criar os modelos de materiais (isotrópicos ou ortotrópicos);

- "***Elements***": dispõe de todos os modelos de macroelementos finitos formulados até o momento;
- "***Mesh***": utilizada para criar a malha de elementos finitos do modelo;
- "***Loads***": utilizada para impor as condições de contorno do problema;
- "***Solver***": utilizada para a solução do problema de elementos finitos;

Para um usuário familiarizado com a linguagem computacional, com os comandos do UFCad e com o seu funcionamento, a implementação do modelo deste caso de estudo é uma tarefa relativamente simples, uma vez que o UFCad é uma ferramenta de análise específica para tubos flexíveis.

8.5.2 Resultados

Além dos parâmetros de convergência, como "*Load Step*" e "*Penalty Factor*", o programa dispõe para o caso de análise de 3 parâmetros de discretização da malha de elementos finitos:

- "**nel**" – o número de divisões axiais das armaduras de tração;
- "**rdiv**" – o número de divisões radiais da capa plástica;
- "**Fourier**" – a ordem da expansão dos deslocamentos em série de Fourier dos elementos cilíndricos.

Devido a condição de contato nó-a-nó ser necessária para a formulação dos contatos, o número de divisões axiais da capa plástica deve ser o dobro do número de divisões das armaduras de tração.

Após uma análise de influência dos parâmetros, viu-se que o parâmetro "rdiv" possuía baixa influência sobre a convergência dos resultados. O parâmetro "nel" mostrou-se o mais importante de todos. Já o parâmetro "Fourier" foi responsável por introduzir um refinamento dos resultados, pois quanto maior o seu valor, mais termos de ordem superior são computados.

Os resultados do programa UFCad podem ser vistos na Figura 67 e na Figura 68. Uma discussão mais detalhada será realizada no próximo item, na qual eles serão comparados com os resultados do programa Abaqus.

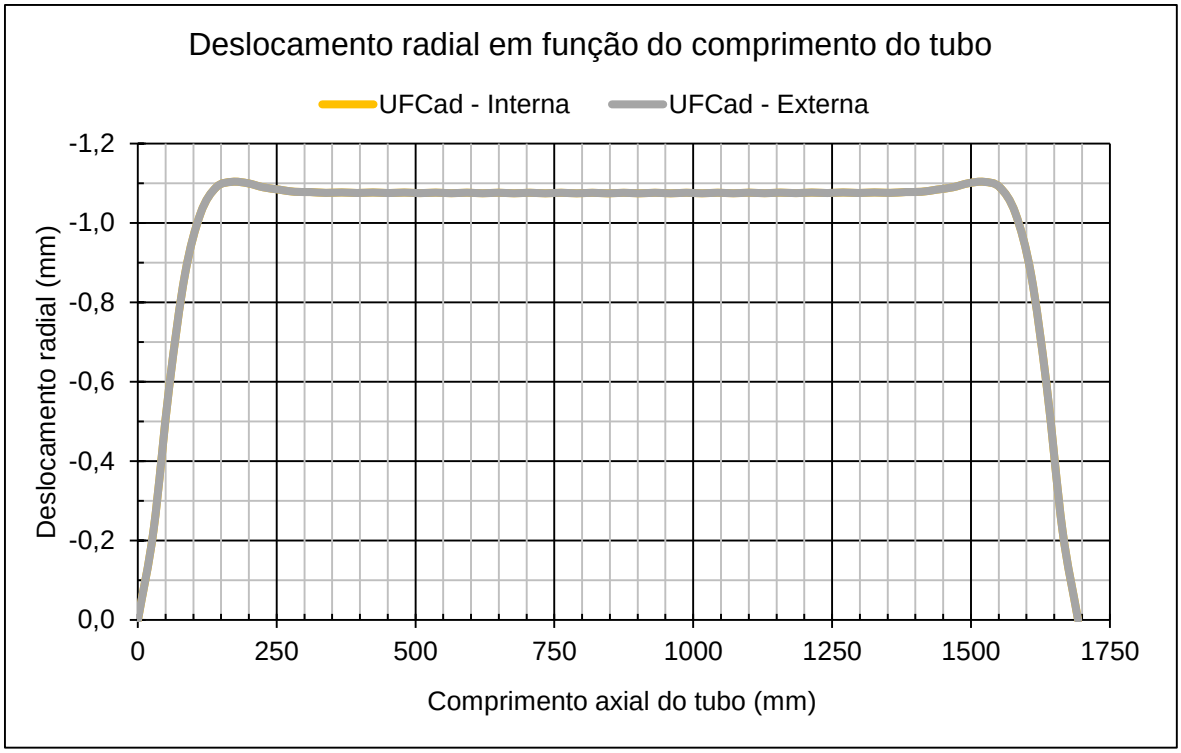


Figura 67 - Resultados de deslocamento radial para os parâmetros: nel = 30; rdiv = 2; Fourier = 0

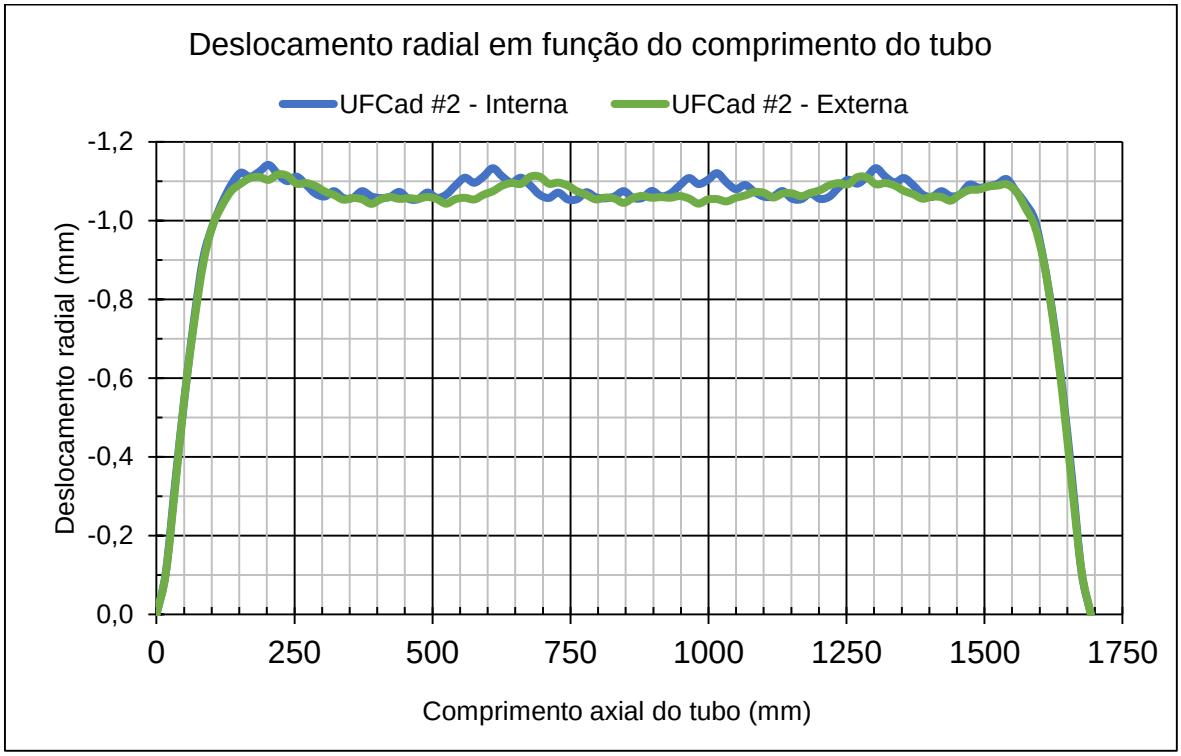


Figura 68 – Resultados de deslocamento radial para os parâmetros: nel = 50; rdiv = 2; Fourier = 5

8.6 Benchmarking: Abaqus x UFCad

A comparação entre ambos os programas será realizada seguindo os critérios:

- Facilidade de implementação
- Qualidade dos resultados
- Tempo e custo de simulação
- Pós-processamento dos dados

8.6.1 Facilidade de implementação

Neste critério, o programa UFCad possui uma boa vantagem em relação ao Abaqus, pois o UFCad é um *software* desenvolvido especificamente para o projeto de tubos flexíveis, enquanto o Abaqus é um programa genérico de elementos finitos. Por este motivo, os macroelementos finitos disponíveis no UFCad levam em consideração a geometria dos componentes, o que simplifica a implementação do modelo e também a definição dos pares do contato. Isso quer dizer que, para implementar o mesmo caso em ambos os programas, o UFCad requer um número bem menor de tarefas e comandos.

Além disso, deve-se levar em conta o fato de ambos os programas requererem treinamento do usuário para que o mesmo esteja apto a operá-lo. O treinamento para o programa UFCad é mais simples e demanda menor tempo.

8.6.2 Qualidade dos resultados

Analisando-se os resultados apresentados na Figura 69, Figura 70 e Figura 71, conclui-se que os resultados do UFCad estão muito próximos do programa Abaqus. Diferenças entre ambos os programas são esperadas, pois as formulações não são exatamente iguais. Por exemplo, no UFCad as armaduras de tração são modelas com vigas curvas e contatos são tipo nó-a-nó, enquanto que, no Abaqus, as armaduras são modelas com elementos sólidos e o contato é superfície com superfície (uma pequena área de intersecção está em contato, diferentemente do contato nó-a-nó, em que apenas dois pontos estão conectados).

Percebe-se que para a primeira figura, os resultados do programa UFCad estão muito lisos (*smooth*), pois foi considerado neste caso apenas os termos de ordem zero da expansão em série de Fourier. Apesar de não ter convergido

completamente, este tipo de análise pode ser útil em uma primeira fase de projeto para estimar com boa precisão a ordem de grandeza dos deslocamentos, pois a demanda computacional deste caso é bem menor que a dos demais.

Devido ao fato das condições de contorno de axissimétricas, era esperado que os deslocamentos apresentassem boa convergência somente com os termos de ordem 0, o que se confirmou nos resultados apresentados. Para carregamentos ou condições de contorno mais complexas, flexão, por exemplo, os termos de ordem superior serão necessários para os resultados convergirem.

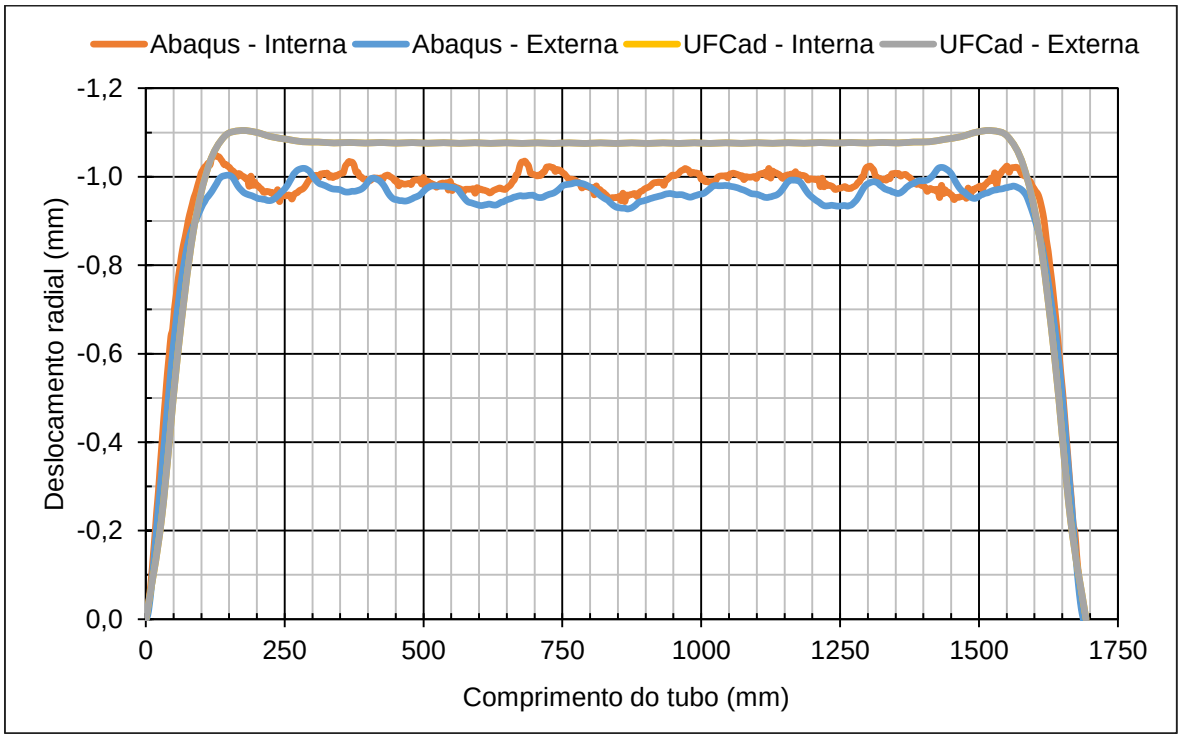


Figura 69 – Comparação de deslocamentos radiais das armaduras internas e externas dos programas UFCad e Abaqus. (UFCad: nel = 30; rdiv = 2; Fourier = 0).

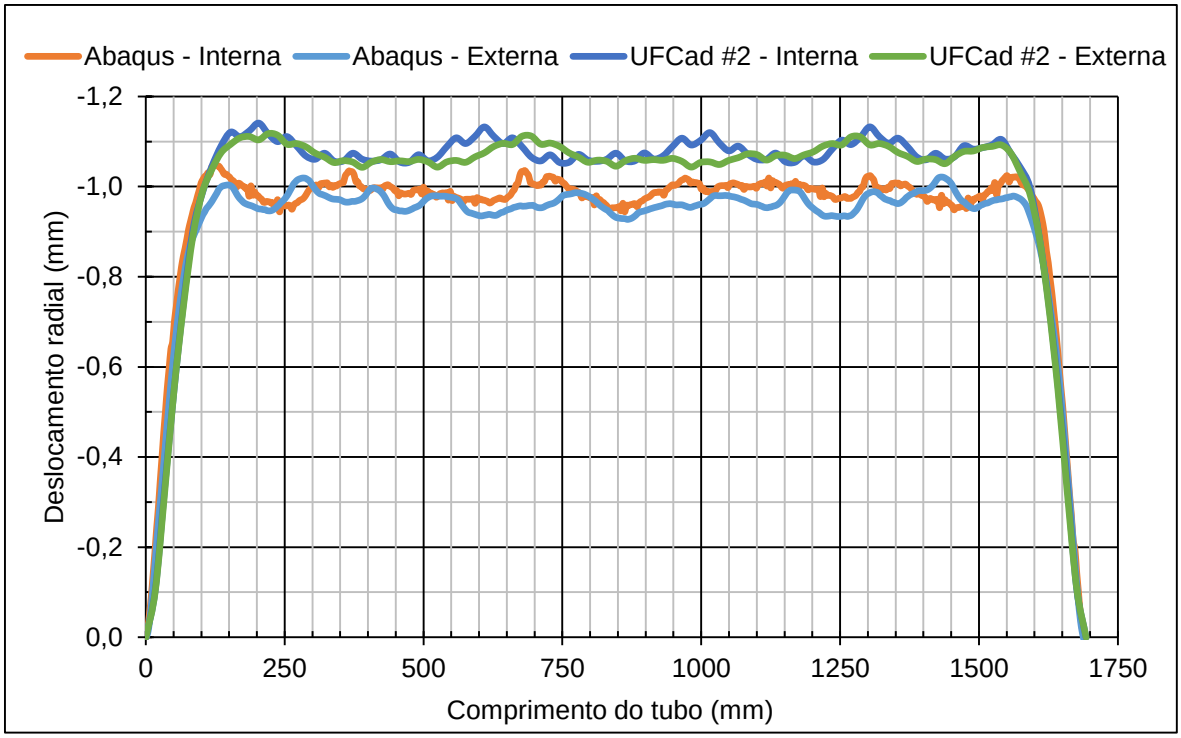


Figura 70 – Comparação de deslocamentos radiais das armaduras internas e externas dos programas UFCad e Abaqus. (UFCad: nel = 50; rdiv = 2; Fourier = 5).

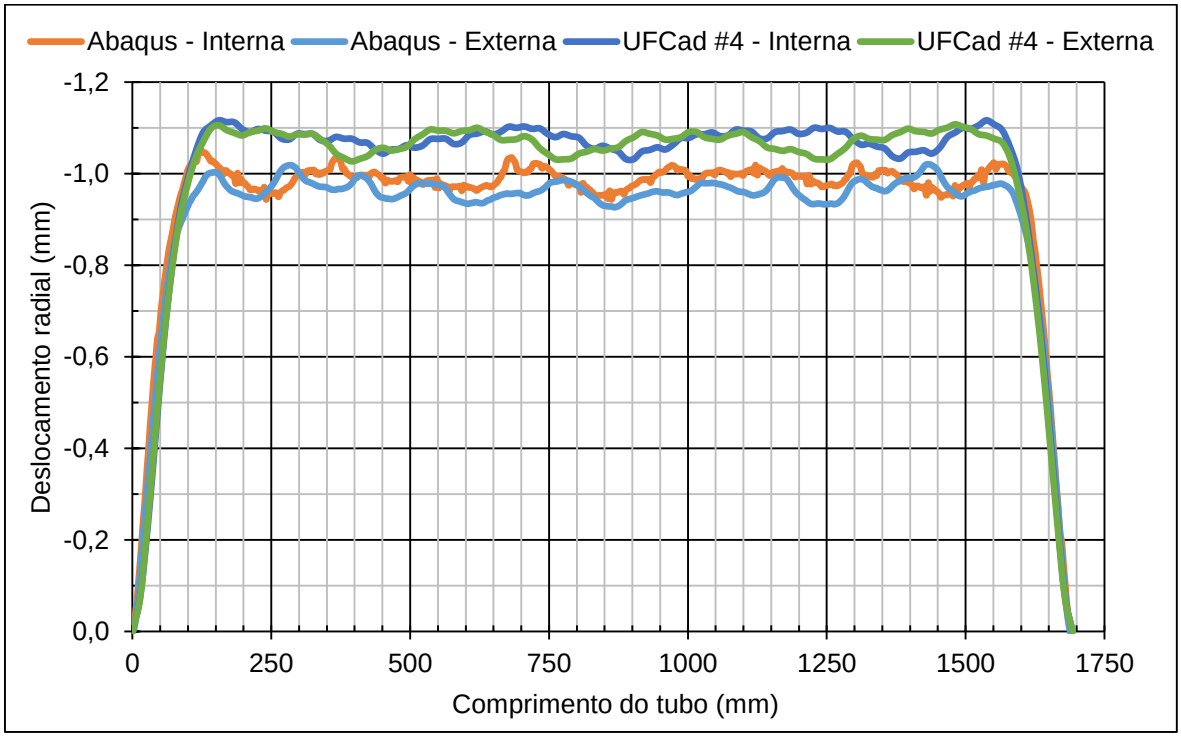


Figura 71 – Comparação de deslocamentos radiais das armaduras internas e externas dos programas UFCad e Abaqus. (UFCad: nel = 60; rdiv = 2; Fourier = 8).

8.6.3 Tempo e custo de simulação

Os resultados do Abaqus foram obtidos em uma máquina com 8 GB de memória RAM e levaram em torno de 1 hora para serem processados. O Abaqus é um software altamente paralelizado, que permite selecionar a quantidade de domínios em que o modelo será subdividido, sendo cada um deles distribuídos entre os núcleos de processamento disponíveis. Além disso, ainda é possível selecionar a porcentagem máxima de memória utilizada pelo computador.

Quanto ao UFCad, a sua principal limitação diz respeito à quantidade de memória demandada, pois o programa ainda não contar com uma técnica eficiente para a resolução de sistemas lineares esparsos. A matriz global de rigidez deve ser armazenada na forma densa para o funcionamento da biblioteca MKL, o que eleva exponencialmente o consumo de memória e limita a utilização do programa às máquinas que dispõe de grande quantidade de memória RAM.

Além disso, carregamentos externos não simétricos (a flexão, por exemplo), exigem vários termos de ordem superior da série de Fourier para a convergência dos resultados. Neste caso, dependendo da complexidade do modelo, mesmo máquinas que dispõem de elevada quantidade de memória podem não ser suficientes para a operação do programa, o que também justifica a busca por um método eficiente de resolução de sistemas lineares esparsos.

Além disso, também é possível reduzir o tempo total de simulação com o programa UFCad com a implementação de técnicas mais avançadas do método dos elementos finitos, como decomposição do problema em subdomínios; aumentando-se o grau de paralelização do programa; encontrando-se métodos mais eficientes de resolução de sistema linear.

Os tempos de resolução do caso de estudo em ambos os programas são da mesma ordem de grandeza, como mostram a Tabela 12 e a Tabela 13. Compreende-se através disto, que a introdução de um método eficiente de resolução de sistemas lineares esparsos pode tornar competitivo o UFCad em relação ao programa Abaqus, principalmente se combinando adequadamente com métodos de paralelização.

Tabela 12 - Tempo de simulação e consumo de memória do programa Abaqus.

Tempo	Max. Memória RAM	Núcleos de processamento e subdomínios
De 40 min à 1h30min	6 GB	**8**

Tabela 13 - Tempo de simulação e consumo de memória do programa UFCad.

Parâmetros	Tempo	Max. Memória RAM
nel = 30; rdiv = 2; Fourier = 0	00 : 05 : 06	9 GB
nel = 50; rdiv = 2; Fourier = 5	01 : 00 : 18	35 GB
nel = 60; rdiv = 2; Fourier = 8	03 : 01 : 00	57 GB

Com isso, conclui-se que apesar de todas as melhorias e avanços introduzidos no programa ao longo deste trabalho, o programa UFCad ainda não encontra-se otimizado o suficiente para ser utilizado por um computador convencional.

8.6.4 Pós-processamento dos dados

O pós-processamento dos dados mostrou-se uma atividade trabalhosa no programa Abaqus. Foi necessária a criação de conjuntos de geometrias para selecionar corretamente os nós das armaduras de tração, e posteriormente várias operações em *Excel* para ordená-los corretamente na sequência referente às suas coordenadas axiais.

Já o pós-processamento de dados no UFCad mostrou-se muito mais fácil, devido à forma como o programa está implementado e aos recursos de sua linguagem computacional (C#), o que simplifica a tarefa de seleção dos nós de cada uma das armaduras de tração. Além disso, o fato de se conhecer o funcionamento interno do programa e a forma como os dados são manipulados representa um grande auxílio ao pós-processamento e análise dos resultados.

8.6.5 Resumo da Análise de Benchmarking

As conclusões obtidas ao longo do item 8.6 podem ser resumidas na Tabela 14.

Tabela 14 – Comparação relativa entre ambos os programas.

Critério	UFCAD	ABAQUS
Facilidade de implementação	✓	✗
Qualidade dos resultados	✓	✓
Tempo e custo de simulação	✗	✓
Pós-processamento dos dados	✓	✗

9. CONCLUSÕES DO TRABALHO

A montagem da matriz global de rigidez mostrou-se uma tarefa crítica no funcionamento do UFCAD, o que justificou as modificações realizadas no programa, sendo elas: a alteração no número de vezes em que a matriz global de rigidez é computada (item 6.1) e a alteração na varredura de montagem da matriz global de rigidez (item 6.2). Estas modificações possibilitaram um ganho expressivo de processamento, com uma redução no tempo de análise de 10 à 20 vezes. Foi realizada a paralelização da etapa de montagem da matriz de rigidez, aumentando-se a taxa de uso processamento, mas esta operação ainda não pôde ser concluída, devido às dificuldades encontradas para se garantir o "*thread safety*". No entanto, os ganhos obtidos com as modificações 6.1 e 6.2 tornaram os ganhos da paralelização apenas residuais.

Além da montagem da matriz global de rigidez, a etapa de resolução de sistemas lineares mostrou-se um importante gargalo. Com o objetivo de eliminar este gargalo, foi incorporada a biblioteca de resolução de sistemas lineares "*Math.Net Numerics MKL*", uma derivação *freeware* da biblioteca profissional "*Intel® Math Kernel Library*", caracterizada por elevado grau de paralelização e eficiência, e que utiliza 100% dos recursos de processamento disponíveis na CPU durante a solução de sistemas lineares. Comparações realizadas com a biblioteca convencional "*Math.Net Numerics"* indicaram uma redução de duas ordens de grandeza no tempo total de simulação. No entanto, a biblioteca MKL não pode ser aplicada à resolução de sistemas lineares esparsos.

Com a alteração das matrizes da classe "*Solver"* para matrizes esparsas (item 6.5), obteve-se uma redução expressiva no consumo de memória. No entanto, este benefício ainda não pode ser aproveitado, pois o UFCAD não dispõe de um método eficiente para a resolução de sistemas lineares esparsos, sendo este o principal gargalo existente no momento.

Neste trabalho foram corrigidos problemas na geração de malhas dos elementos de contato do tipo bridge (item 6.7); foi implementada uma nova lógica de detecção de contatos entre armaduras (item 6.6), mais eficiente computacionalmente e que eliminou os problemas de formação de pares indesejados de contato; e também foi proposta uma formulação modificada para o macroelemento finito de contato do

tipo nó-a-nó entre elemento cilíndrico e elemento de hélice (capítulo 3). Todas estas modificações e correções realizadas no programa UFCAD tornaram-no capaz de simular um tubo com maior número de componentes.

A validação de um caso de estudo realizada no capítulo 8, comparando o UFCad com o programa profissional de elementos finitos Abaqus, permitiu a realização de um *benchmarking* entre ambos os programas sob diversos critérios, cujos resultados encontram-se resumidos na Tabela 14. Os resultados de deslocamentos radiais das armaduras de tração de ambos os programas (Figura 71) estão muito próximos, o que permite validar os resultados de deslocamento o tubo flexível implementado e simulado. Diferenças entre ambos os programas são esperadas, pois as formulações não são exatamente iguais. Por exemplo, as armaduras de tração foram modelas utilizando-se vigas curvas no UFCAD e elementos sólidos no ABAQUS; foi utilizado o contato do tipo nó-a-nó no UFCAD e do tipo superfície com superfície no ABAQUS, no qual uma pequena área de intersecção está em contato, diferentemente do contato nó-a-nó, em que apenas dois pontos estão conectados. Mesmo ainda não estando otimizado o suficiente para ser utilizado por computadores convencionais, os resultados obtidos com este trabalho contribuíram para aumentar a eficiência do programa UFCad, com ganhos consideráveis de processamento e caminhos bem definidos para a redução do consumo de memória, já sendo possível a sua utilização para análise estrutural de tubos flexíveis de baixa à média complexidade.

Como trabalhos futuros, é de grande valia o desenvolvimento de um método eficiente para a resolução de sistemas lineares esparsos, o que eliminaria o principal gargalo existente no programa atualmente, que é o consumo elevado de memória. Além disso, também é possível reduzir o tempo total de simulação com o programa UFCad com a implementação de técnicas mais avançadas do método dos elementos finitos, como decomposição do problema em subdomínios e aumentando-se o grau de paralelização do programa.

10. REFERÊNCIAS

2B1st Consulting. (2014). *Umbilical*. Fonte: 2B1st Consulting: http://www.2b1stconsulting.com/umbilical/

AGÊNCIA NACIONAL DO PETRÓLEO. (2013). *Anuário Estatístico Brasileiro do Petróleo, Gás Natural e Biocombustíveis.* Rio de Janeiro: Ministério de Minas e Energia.

API RP 17B. (2002). *Recommended Pratice for Flexible Pipe.* American Petroleum Institute.

BATHE, K. J. (1995). *Finite Element Procedures* (2. ed. [S.I.] ed.). Prentice Hall.

COOK, R. D., MALKUS, D. S., PLESHA, M. E., & WITT, R. J. (2002). *Concepts and Applications of Finite Element Analysis* (4. ed. [S.I.] ed.). John Wiley & Sons.

KNUTH, D. (1977). Structured Programming with go to Statements. 41.

LOVE, A. E. (1944). *A Treatise On The Mathematical Theory of Elasticity* (4. ed. ed.). New York: Dover.

MSDN. (2014). *Introdução a consultas LINQ (C#)*. Fonte: Microsoft Developer Networker: http://msdn.microsoft.com/pt-br/library/bb397906.aspx

MSDN. (2014). *Paralelismo de dados (biblioteca de tarefas paralelas)*. Fonte: Microsoft Developer Network: http://msdn.microsoft.com/pt-br/library/dd537608(v=vs.110).aspx

MSDN. (2014). *Visual C#*. Fonte: Microsoft Developer Network: http://msdn.microsoft.com/pt-br/library/kx37x362.aspx

Newton-Raphson Option (NROPT). (2014). Fonte: ANSYS CAE_Course: http://office.es.ncku.edu.tw/leehh/ANSYS/ANSYS/CAE_Course/Chap16_Nonlinear/NROPT.htm

ORACLE. (2010). *A procedure is thread safe when the procedure is logically correct when executed simultaneously by several threads*. Fonte: Multithreaded Programming Guide.

ORACLE. (2014). *Lesson: Object-Oriented Programming Concepts*. Fonte: ORACLE - "The Java Tutorials": http://docs.oracle.com/javase/tutorial/java/concepts/index.html

PETROBRAS. (12 de 2013). *Petrobras S.A.* Fonte: <http://www.petro-bras.com.br/>

PETRÓLEO E ENERGIA. (12 de 2013). *PETROLEO E ENERGIA*. Fonte: <http://www.petroleoenergia.com.br/-petroleo/wp-content/uploads/2013/09/Evonik-PA12_Tubos-Flexiveis.jpg>

PROVASI, R. (2013). *Contribuição ao Projeto de Cabos Umbilicais e Tubos Flexíveis: Ferramentas de CAD e Modelo de Macro Elementos.* Tese de Doutorado, Escola Politécnica da Universidade de São Paulo.

PROVASI, R., & MARTINS, C. A. (2010-a). A Finite Macro-Element for Cylindrical Layer Modeling. *International Conference on Ocean, Offshore and Arctic Engineering - OMAE2010.* Shangai, China.

PROVASI, R., & MARTINS, C. A. (2010-b). CAD Software for Cable Design: a Three-Dimensional Visualization Tool. *International Conference on Ocean, Offshore and Arctic Engineering - OMAE2010.* Shangai, China.

PROVASI, R., & MARTINS, C. A. (2011). A three-dimensional curved beam element for helical components modeling. *International Conference on Ocean, Offshore and Arctic Engineering - OMAE2011*. Rotterdam, Netherlands.

PROVASI, R., & MARTINS, C. A. (2013-a). A rigid connection for macro-elements with different node displacement natures. *International Offshore and Polar Engineering*

Anchorage, International Society of Offshore and Polar Engineers (ISOPE). Alaska, USA.

PROVASI, R., & MARTINS, C. A. (2013-b). A Contact Element for Macro-Elements with Different Node Displacement Natures. *International Offshore and Polar Engineering Anchorage, International Society of Offshore and Polar Engineers (ISOPE).* Alaska, USA.

PROVASI, R., & MARTINS, C. A. (2013-c). A Finite Macro-Element for Orthotropic Cylindrical Layer Modeling. *Journal of Offshore Mechanics and Arctic Engineering, Volume 135, Issue 3*.

PROVASI, R., MEIRELLES, C., & MARTINS, C. (2009). CAD Software For Cable Design: A Consistent Modeling Method To Describe The Cable Structure And Associated Interface. *International Congress of Mechanical Engineering COBEM*. Gramado, Brazil.

SHARP, J. (2011). *Microsoft Visual C# 2010: passo a passo.* Bookman.

VALLOUREC. (2014). Fonte: http://finance.yahoo.com/news/vallourec-vallourec-widens-premium-offer-063301518.html

WIKIPEDIA. (2014). *C Sharp*. Fonte: Wikipedia: http://pt.wikipedia.org/wiki/C_Sharp

WIKIPEDIA. (2014). *Orientação a objetos*. Fonte: Wikipedia: http://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objetos

WRIGGERS, P. (2002). *Computational Contact Mechanics.* Wiley.

ZHU, Z. H., & MEGUID, S. A. (2004). Analysis of three-dimensional locking-free curved beam element. *International Journal of Computational Engineering Science, Vol. 5, No. 3*, pp. 535-556.