**UNIVERSIDADE FEDERAL DO PARANÁ**

**ENGENHARIA MECÂNICA**

**JÉSSICA MENEGUEL**

**LEONARDO SIRINO**

**LOCALIZAÇÃO DE FONTES ACÚSTICAS EM CORPOS CILINDRÍCOS DE EXTREMIDADES ELIPSOIDAIS**

**CURITIBA**

**2018**

**JÉSSICA MENEGUEL**

**LEONARDO SIRINO**

****

**LOCALIZAÇÃO DE FONTES ACÚSTICAS EM CORPOS CILINDRÍCOS DE EXTREMIDADES ELIPSOIDAIS**

Trabalho de Conclusão do Curso de Graduação em Engenharia Mecânica da Universidade Federal do Paraná, apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Mecânica.

Orientador: Prof. Dr. Luciano Kiyoshi Araki

**CURITIBA**

**2018**

**TERMO DE APROVAÇÃO**

JÉSSICA MENEGUEL

LEONARDO SIRINO

LOCALIZAÇÃO DE FONTES ACÚSTICAS EM CORPOS CILINDRÍCOS DE EXTREMIDADES ELIPSOIDAIS

Trabalho de conclusão de curso aprovado como requisito parcial para obtenção do grau de Bacharel em Engenharia Mecânica pela Universidade Federal do Paraná.

BANCA EXAMINADORA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr./Ms……..

Departamento e Instituição onde atua o/a professor(a)

Presidente da Banca

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr./Ms……..

Departamento e Instituição onde atua o/a professor(a)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Dr./Ms……..

Departamento e Instituição onde atua o/a professor(a)

Curitiba

2018

AGRADECIMENTOS

(Opcional) São menções que o autor faz a pessoas e/ou instituições das quais eventualmente recebeu apoio para o desenvolvimento do trabalho. Os agradecimentos aparecem em folha distinta após a dedicatória, pode ser escrito no final da página, sendo o texto justificado a direita e em negrito.

Exemplo 1:

A

***Prof. Marcio Luiz Fernandes da UNIOESTE***

***pelas orientações xxxxxxxxxxxxxxxx.***

***Joaquim da Silva***

***por xxxxxxxxxxxxxxxxxx.***

***Carmem Cristina***

***devido xxxxxxxxxxxxxxxxxxxxxx.***

***Exemplo 2:***

***A todos que, direta ou indiretamente, contribuíram para a realização deste trabalho, xxxxxx xxxx x x xxxxxxxxxxxxxxxxx xxxxxxx xx xxxx x xxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.***

LISTA DE ILUSTRAÇÕES

[Figura 1 - Onda longitudinal 14](#_Toc531543516)

[Figura 2 - Onda transversal 14](#_Toc531543517)

[Figura 3 - Onda de Rayleigh 15](#_Toc531543518)

[Figura 4 - Onda de Lamb 15](#_Toc531543519)

[Figura 5 - Efeito Kaiser 16](#_Toc531543520)

[Figura 6 - Elementos de um sensor de EA 17](#_Toc531543521)

[Figura 7 - Fluxograma do sinal de EA 17](#_Toc531543522)

[Figura 8 – Tempos de um sinal de EA 19](#_Toc531543523)

[Figura 9 – Exemplo de tela de monitoramento para testes de emissão acústica 19](#_Toc531543524)

[Figura 10 - Processo de calandramento para confecção do corpo cilíndrico 20](#_Toc531543525)

[Figura 11 - Processo de conformação para fabricação de tampo elipsoidal 21](#_Toc531543526)

[Figura 12 - Gráfico para definição dos grupos de risco para vasos de pressão 22](#_Toc531543527)

[Figura 13 - Gráfico de teste hidrostático do grupo de risco 1 23](#_Toc531543528)

[Figura 14 - Gráfico de teste hidrostático do grupo de risco 2 23](#_Toc531543529)

[Figura 15 - Gráfico de teste hidrostático do grupo de risco 3 24](#_Toc531543530)

[Figura 16 - Localização planar pelo método da hipérbole. T1, T2 e T3 são os tempos de chegada das ondas mecânicas nos sensores correspondentes 25](#_Toc531543531)

[Figura 17 - Localização planar com dois sensores 26](#_Toc531543532)

[Figura 18 - Grandezas em um elipsóide de revolução 28](#_Toc531543533)

[Figura 19 - Identificação das coordenadas x e s 33](#_Toc531543534)

[Figura 20 - Vista superior do tampo - coordenadas x0' e y0' 34](#_Toc531543535)

[Figura 21 - Secante da elipse 35](#_Toc531543536)

[Figura 22 - Vista superior do plano de seccionamento 37](#_Toc531543537)

[Figura 23 - Elipse auxiliar 38](#_Toc531543538)

[Figura 24 - Regressão Posição x Comprimento do arco 40](#_Toc531543539)

[Figura 25 - Regressão Comprimento do arco x Posição 41](#_Toc531543540)

[Figura 26 - Distâncias entre pontos no tampo para diferentes métodos de cálculo 42](#_Toc531543541)

[Figura 27 – Erro no cálculo da distância entre pontos no tampo para diferentes métodos 43](#_Toc531543542)

[Figura 28 - Erro máximo do método de seccionamento 44](#_Toc531543543)

[Figura 29 - Croqui da distribuição dos sensores no vaso 49](#_Toc531543544)

[Figura 30 – Foto da montagem dos sensores no vaso de pressão 49](#_Toc531543545)

LISTA DE TABELAS

[Tabela 1 - Tabela de verificação do acoplamento dos sensores 49](#_Toc531543546)

LISTA DE ABREVIATURAS E SIGLAS

|  |  |
| --- | --- |
| EA | Emissão acústica |
| END | Ensaio não destrutivo |
| TH | Teste hidrostático |
|  |  |
|  |  |
|  |  |

SUMÁRIO

[1. Introdução 11](#_Toc531543547)

[2. Revisão bibliográfica 13](#_Toc531543548)

[2.1. Emissão acústica 13](#_Toc531543549)

[2.1.1. A origem da técnica 13](#_Toc531543550)

[2.1.2. Modos de propagação de onda 13](#_Toc531543551)

[2.1.3. Efeito Kaiser 15](#_Toc531543552)

[2.1.4. Equipamentos 16](#_Toc531543553)

[2.1.5. Processamento do sinal de EA 18](#_Toc531543554)

[2.2. Vasos de pressão 20](#_Toc531543555)

[2.3. Localização 24](#_Toc531543556)

[2.4. Geodésica 27](#_Toc531543557)

[2.5. Otimização 28](#_Toc531543558)

[2.5.1. Evolução diferencial 28](#_Toc531543559)

[2.5.1.1. Mutação 29](#_Toc531543560)

[2.5.1.2. Cruzamento 29](#_Toc531543561)

[2.5.1.3. Seleção 30](#_Toc531543562)

[2.5.1.1. Implementação 30](#_Toc531543563)

[2.5.2. Método de Newton 30](#_Toc531543564)

[3. Método de seccionamento 32](#_Toc531543565)

[3.1.1. Coordenadas auxiliares 32](#_Toc531543566)

[3.1.2. Plano de seccionamento e elipse auxiliar 36](#_Toc531543567)

[3.1.3. Aproximações 39](#_Toc531543568)

[3.1.4. Verificação 41](#_Toc531543569)

[4. Algoritmo de localização 45](#_Toc531543570)

[5. Procedimento Experimental 48](#_Toc531543571)

[5.1. Análise numérica 48](#_Toc531543572)

[5.2. Análise empírica 48](#_Toc531543573)

[6. Resultados 50](#_Toc531543574)

[7. Bibliografia 51](#_Toc531543575)

RESUMO

A técnica de Emissão Acústica (EA) é um ensaio não destrutivo de grande aplicabilidade na engenharia mecânica, podendo ser usada para testes pontuais em equipamentos ou para o monitoramento continuado de grandes estruturas. Uma das grandes vantagens dessa técnica é a possibilidade de monitorar grandes regiões do equipamento em operação com uso de poucos sensores, detectando defeitos na estrutura e apontando sua localização. Os defeitos, que atuam como fontes acústicas durante a solicitação da estrutura, podem ser localizados a partir dos tempos de chegada do sinal nos sensores, recaindo em um problema geométrico que dependerá da forma da estrutura analisada. Quando se analisam vasos de pressão, caldeiras e tanques é comum encontrar geometrias na forma de corpos cilíndricos com extremidades elipsoidais; as técnicas atuais tratam esse tipo de geometria de maneira aproximada, promovendo distorção na geometria para se realizar a localização, gerando certa imprecisão nos resultados, principalmente para fontes sonoras nos tampos. O presente trabalho propõe uma alternativa para a técnica de localização em corpos cilíndricos com extremidades elipsoidais de modo a minimizar distorções na geometria e fornecer consequentemente resultados mais precisos.

**Palavras-chaves**: Algoritmo genético, Emissão acústica, Localização, Vasos de pressão

# Introdução

Estruturas de corpo cilíndrico com tampo elipsoidal, como vasos de pressão e tanques, são comumente empregadas no armazenamento de fluídos na indústria mecânica. Na fabricação esses equipamentos passam por processos de laminação, conformação e soldagem, que podem gerar defeitos e induzir tensões na estrutura. Durante a operação, frequentemente são submetidos a ciclos térmicos e mecânicos, propiciando que os defeitos gerados na fabricação cresçam. A falha desses equipamentos ocorre em geral por trincas e vazamentos, e pode acarretar consequências catastróficas, pelo fato de que essas estruturas frequentemente armazenam fluídos a alta temperatura e pressão. Para garantir uma operação segura os vasos de pressão devem ser obrigatoriamente submetidos ao teste hidrostático (TH) em sua fase de fabricação; o TH pode ser repetido durante a vida útil do equipamento para a verificação de vazamentos ou outros defeitos [1]. A pressão do TH deve ser definida pelo profissional responsável pelo vaso, e se situa geralmente em 1,5 vezes a pressão máxima de trabalho admissível (PMTA) [2].

Através da técnica de Emissão Acústica (EA) é possível monitorar os ensaios hidrostáticos, podendo-se identificar o crescimento de trincas na estrutura e vazamento de pequena dimensão. É possível também monitorar vasos de pressão e tanques durante operação, identificando zonas críticas em tempo real, tornando a manutenção preventiva do equipamento mais eficiente.

Algumas das maiores vantagens da EA sobre as demais técnicas de ensaios não destrutivos é sua capacidade em monitorar uma estrutura de maneira global e não intrusiva, apontando a localização de regiões na estrutura que apresentam anomalias. Portanto, custos são reduzidos pelo fato de o ensaio interferir pouco na operação do equipamento e ter curta duração, e o reparo necessário devido aos eventuais defeitos encontrados ser restringido a uma área limitada indicada nos resultados. Além disso, há economia relacionada à não necessidade de escavar tubulações enterradas e remover revestimentos quando da aplicação da técnica, entre outros fatores.

A localização de anomalias que são fontes de EA é feita partindo-se do pressuposto de que a onda se propaga em frentes de onda esféricas, atingindo os sensores com diferentes tempos de chegada. A partir do tempo que o sinal demorou para chegar em diferentes sensores e da posição de cada um desses, é possível por triangulação calcular a posição da fonte causadora do sinal.

Entretanto, devido à complexidade geométrica de elementos cilíndricos com tampos elipsoidais, como os vasos de pressão, as técnicas atuais de localização aplicadas em sistemas comerciais empregam modelagens simplificadas dessas estruturas, geralmente planificando-a. Logo, é calculada a posição da fonte a partir de um caminho aproximado percorrido pela onda, gerando resultados imprecisos principalmente na região dos tampos, que é muito deformada na planificação.

Neste trabalho a trajetória das ondas sonoras em corpos cilíndricos com tampos elipsoidais é determinada através do cálculo da distância entre dois pontos pela aplicação de um método aqui denominado de Método do Seccionamento, que será comparada à menor distância entre pontos em um elipsoide de revolução, a geodésica. A partir dessas distâncias procura-se obter resultados mais acurados que os métodos tradicionais de planificação para a localização de defeitos através da técnica de EA, com resultados semelhantes à aplicação de geodésicas, mas com velocidade de processamento que permita sua aplicação em monitoramento em tempo real.

# Revisão bibliográfica

## Emissão acústica

Emissão Acústica (EA) é uma técnica de ensaio não destrutivo (END) fundamentada no princípio básico de que processos de degradação dos materiais geram ondas mecânicas transientes, passíveis de detecção por sensores piezelétricos. A principal fonte de sinais quando se trata de emissão acústica é a deformação plástica, que ocorre de maneira generalizada quando há sobrecarga na estrutura, ou localizada, na ponta de uma trinca em processo de propagação, por exemplo. Também existem as chamadas pseudofontes, tais como: vazamento, cavitação, descargas parciais, fricção, entre outros; todos esses eventos geram ondas mecânicas no material que também podem ser detectadas e localizadas.

### A origem da técnica

O primeiro registro do uso da técnica de EA data do século VIII pelo alquimista árabe Jabiribn Hayyan, que reportou que o estanho emite um “som áspero” quando trabalhado enquanto o ferro “soa muito” durante o forjamento. Esse foi o princípio do uso da técnica de EA, quando se analisavam apenas as fontes audíveis. Esse tipo de relato continuou com Robert Anderson testando corpos de prova de alumínio além de seu ponto de escoamento. Erich Scheil também relatou ruído audível durante a formação de martensita no aço [3].

O começo da era moderna da técnica de EA teve início com um dos trabalhos mais importante até hoje, o trabalho de PhD de Joseph Kaiser, intitulado Investigação da ocorrência de ruído durante o ensaio de tração (*Untersuchung über das Auftreten von Geräuschen beim Zugversuch*), que registrou o primeiro relato do que hoje é conhecido como efeito Kaiser. Jopseh Kaiser observou que amostras que já haviam sido submetidas a uma determinada força, quando solicitadas mecanicamente novamente, só voltavam a emitir ruído após a máxima força aplicada no teste anterior ser ultrapassada. Nos testes de Kaiser já foram usados sensores piezelétricos para a detecção de ruído, mesmo que de forma rudimentar se comparada à tecnologia atual.

### Modos de propagação de onda

As ondas de EA, como qualquer onda mecânica, necessita de meio de propagação e apresentam características como frequência, período, comprimento de onda, amplitude, fase, entre outras. [3]

O meio de transporte de uma onda é composto por arranjos atômicos conectados por ligações atômicas, que são responsáveis pela transferência da energia cinética entre as partículas na presença de perturbações. A onda atuante sobre esse arranjo apresentará dois movimentos: um que determina a direção de propagação da onda e outro que determina o eixo de oscilação das partículas. Combinações diferentes desses dois movimentos geram diferentes modos de propagação da onda. [3]

* Onda longitudinal: O movimento de oscilação das partículas é paralelo à direção de propagação da onda. Esse tipo de onda gera a formação de regiões de rarefação e regiões de compressão no material, devido ao movimento dos planos das partículas, como mostrado na Figura 1.

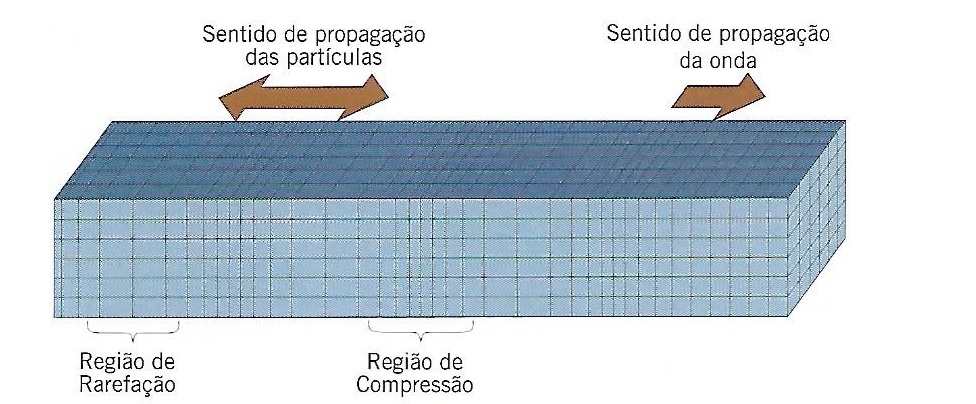


Figura 1 - Onda longitudinal

Fonte: Filippin, 2017

* Onda transversal: O movimento de oscilação das partículas é perpendicular à direção de propagação da onda. Não há formação de regiões de rarefação e compressão, pois os planos das partículas se mantêm equidistantes, como mostrado na Figura 2.

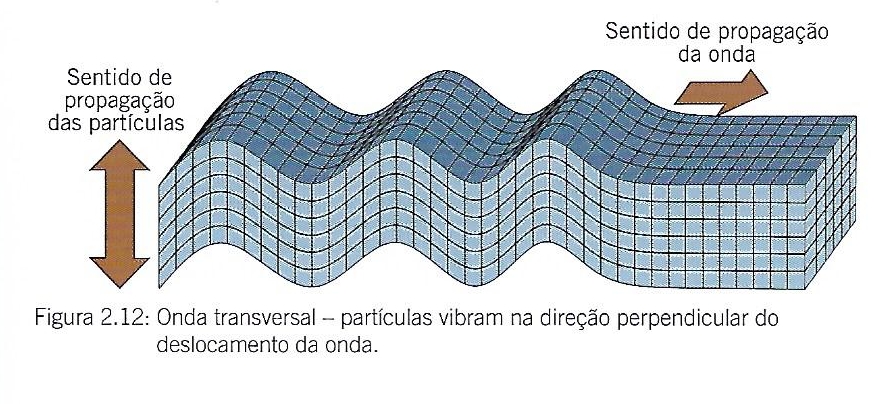


Figura 2 - Onda transversal

Fonte: Filippin, 2017

* Ondas superficiais: São compostas pela combinação de ondas longitudinais e transversais, que se propagam pela superfície do material. São subdivididas entre diversos tipos de onda, entre as ondas de Rayleigh e as ondas de Lamb, apresentadas na Figura 3 e na Figura 4.

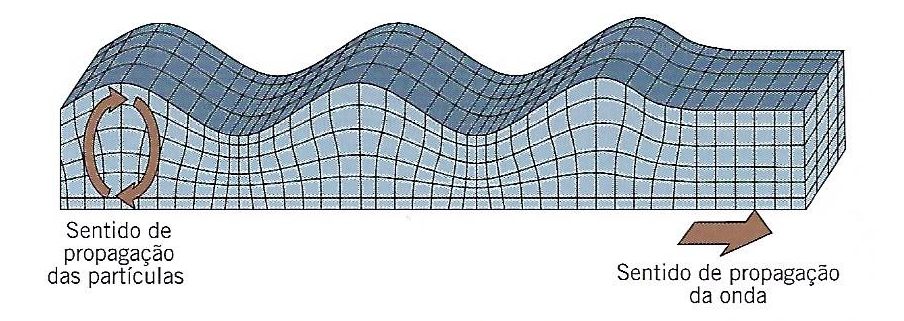


Figura 3 - Onda de Rayleigh

Fonte: Filippin, 2017

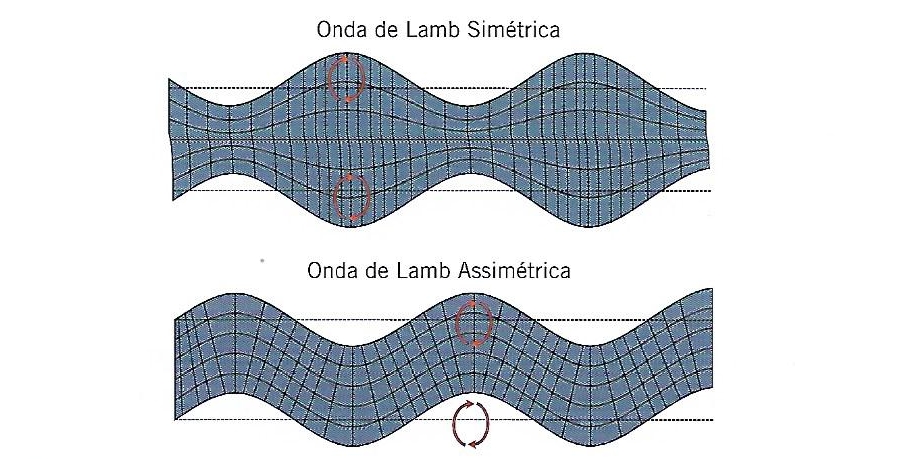


Figura 4 - Onda de Lamb

Fonte: Filippin, 2017

### Velocidade da onda

### Efeito Kaiser

O trabalho de Kaiser resultou em um dos princípios básicos da técnica de EA, o Efeito Kaiser. Segundo esse, para uma classe de materiais que obedeça a esse princípio, aplicando-se carregamento menor que um carregamento crítico, só serão observados sinais de emissão acústica após o carregamento ultrapassar a carga anteriormente aplicada na estrutura, como mostrado na Figura 5. Nesse caso, garante-se que haverá efeitos ativos, portanto, atividade acústica estará presente. Fazem parte dessa classe de materiais que obedecem a este princípio os materiais metálicos [4].

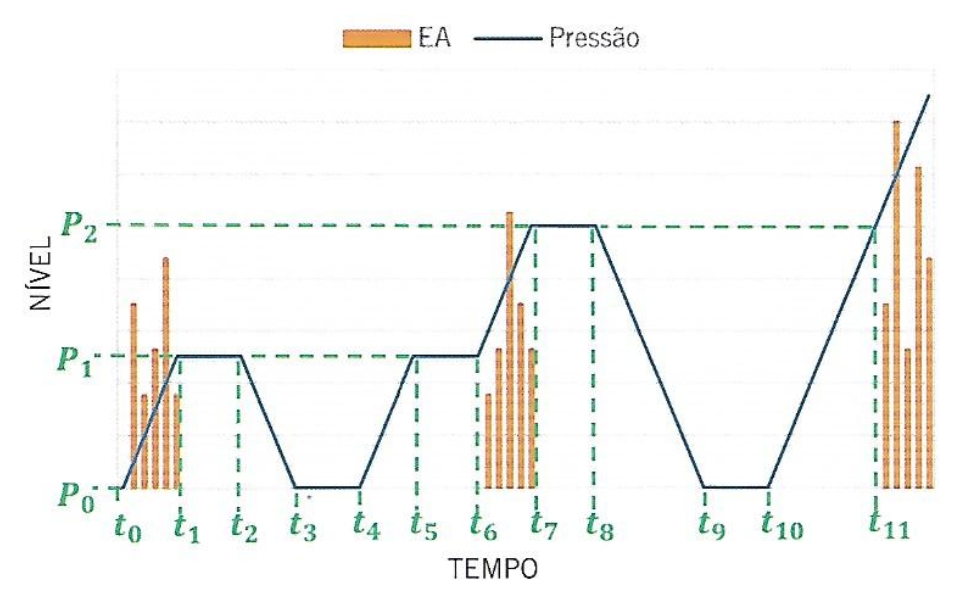


Figura 5 - Efeito Kaiser

Fonte: Filippin, 2017

Se a estrutura possuir um defeito que promova a concentração de tensões localizada, a aplicação de um carregamento menor que o anterior ainda pode gerar tensões localizadas mais elevadas que as anteriores. Esse fenômeno pode ser observado em uma estrutura com trinca, onde o carregamento nominal promova o crescimento desta, o que faz aumentar o fator de intensificação de tensões e gera tensões mais elevadas com o mesmo carregamento nominal, situação que é observada no crescimento de trinca por fadiga em uma estrutura [3] [4].

### Equipamentos

O uso moderno de EA não se limita às fontes audíveis, sensores piezelétricos são usados para captar ondas mecânicas no material, isso torna possível a detecção de ondas com frequências muito mais elevadas e amplitude menores que o ouvido humano seria capaz de detectar. O sensor de EA é geralmente constituído de um cristal piezelétrico no interior de um invólucro de proteção, onde pode estar também o amplificador integrado, denominado pré-amplificador. Na Figura 6 são apresentados os componentes de um sensor de EA.

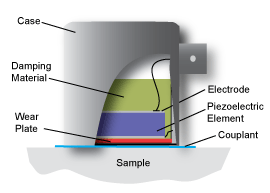


Figura 6 - Elementos de um sensor de EA

Fonte: https://www.nde-ed.org/EducationResources/CommunityCollege/Other%20Methods/AE/AE\_Equipment.php

Entre o sensor e a estruturada analisada há um meio acoplante, geralmente líquido e bastante viscoso; isso garante maior integridade na transmissão do sinal ao sensor.

O sinal de EA, ao passar para sensor, faz com que o cristal piezelétrico se deforme, então este produz uma diferença de potencial proporcional a esta deformação. Esse sinal elétrico é então amplificado e transmitido através de cabos, geralmente coaxiais.

Existem equipamentos comerciais especializados na aquisição e processamento de sinais de EA, mas todos seguem a mesma estrutura básica, conforme apresentado na Figura 7.

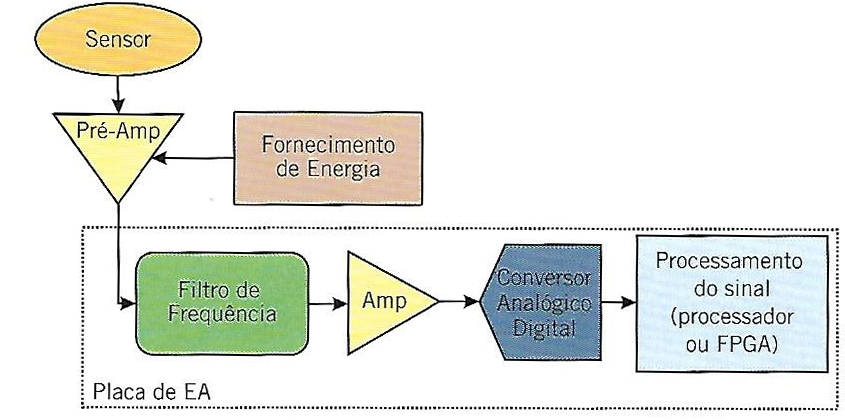


Figura 7 - Fluxograma do sinal de EA

Fonte: Filippin, 2017

O sinal, já amplificado pelo pré-amplificador chega ao equipamento e passa por um condicionamento, que consiste em filtros de frequência; então é amplificado novamente e enviado ao conversor analógico-digital (*Analog Digital Converter* – ADC). Este sinal, agora digitalizado, deve ser processado para que se retirem as informações pertinentes; esse processamento pode ser feito por um processador convencional, um circuito dedicado ou, mais frequentemente, um chip FPGA (*Field Programmable Gate Array*).

O processamento de sinais de EA é uma tarefa de grande custo computacional, já que as frequências de amostragem geralmente são elevadas (acima de 1 MHz) para que possa se registrar de maneira fidedignas sinais de EA com frequências bastante elevadas. Usar um processador convencional para esta tarefa pode limitar o número de canais de um sistema a um valor impraticável. Por esse motivo se torna interessante o uso de FPGA’s. Os FPGA’s são circuitos integrados programáveis que permitem que as operações realizadas no sinal de EA sejam diretamente implementadas em hardware, fazendo com que tenha desempenho semelhante à de circuitos dedicados, mas ainda com a flexibilidade próxima a de um processador. Outra vantagem do uso de FPGA’s é o de tornar o processamento distribuído, uma vez que podem ser adicionados mais chips conforme se aumente o número de canais, sendo esta uma prática comum entre as fornecedoras de equipamentos, sendo que cada placa de expansão de canais possui seu próprio FPGA.

### Processamento do sinal de EA

Devido às altas taxas de amostragem utilizadas, é inviável a análise e registro contínuo do sinal de um sensor de EA. Por esse motivo usam-se as informações dos *hits.* Os *hits* são trechos do sinal de algum sensor que em algum momento ultrapassaram um valor pré-determinado, denominado limiar de detecção. A duração desses *hits* é definida com base em alguns parâmetros, que também são escolhidos previamente. A Figura 8 apresenta como esses parâmetros são observados em um sinal.

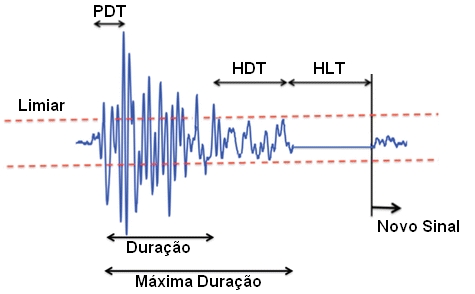


Figura 8 – Tempos de um sinal de EA

Fonte: Shen, 2015

Definido um *hit*, deve se calcular algumas métricas para se caracterizar este trecho de sinal. Existe uma grande variedade de parâmetros que podem ser extraídos de um *hit,* mas os principais estão relacionados à sua intensidade, frequência, duração e energia.

A análise de um ensaio de EA se dá por meio destas métricas, suas correlações e sua evolução no decorrer do tempo. Portanto é muito comum o uso de gráficos durante a execução de algum ensaio para o acompanhamento em tempo real, a Figura 9 apresenta um exemplo de tela usada para o monitoramento de um ensaio.

|  |  |
| --- | --- |
|  |  |

Figura 9 – Exemplo de tela de monitoramento para testes de emissão acústica

Fonte: Filippin, 2017

## Vasos de pressão

Vasos de pressão são reservatórios de diferentes tipos, dimensões ou finalidades, projetados para resistir com segurança a pressões internas diferentes da pressão atmosférica ou da pressão externa. A NR-13 é a norma regulamentadora que “estabelece os requisitos mínimos para gestão da integridade estrutural de caldeiras a vapor, vasos de pressão e suas tubulações de interligação nos aspectos relacionados à instalação, inspeção, operação e manutenção, visando à segurança e à saúde dos trabalhadores” [1] e divide os vasos em quatro diferentes categorias, de acordo com a composição, pressão e volume do fluído armazenado.

Os vasos de pressão são fabricados a partir de chapas laminadas, que são calandradas para confecção do corpo cilíndrico (Figura 10) e conformadas para confecção do tampo (Figura 11). As diferentes partes do vaso são unidas posteriormente por soldagem. Aços ao carbono são frequentemente empregados na fabricação dessas estruturas devido às suas características de boa conformabilidade, boa soldabilidade, baixo custo, condição de serviço, natureza e grau dos esforços aplicados, disponibilidade e segurança.

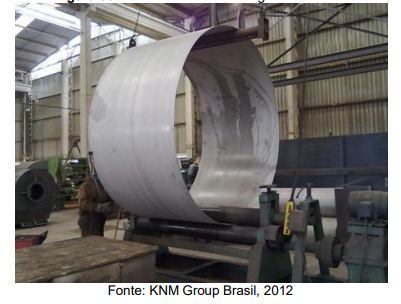


Figura 10 - Processo de calandramento para confecção do corpo cilíndrico

Fonte: KNM Group Brasil, 2012



Figura 11 - Processo de conformação para fabricação de tampo elipsoidal

Fonte: Gianturco, 2012

A etapa de fabricação pode induzir diferentes defeitos na estrutura de um vaso de pressão, sendo defeitos comuns da laminação as gotas frias, vazios, fendilhamento, ondulação, trincas, dobras, inclusões, e segregações; da calandragem, espessura irregular e bolhas, da conformação trincas, ondulações e rugas e abaulamento; e por fim, da soldagem, porosidade, falta de penetração ou fusão, mordeduras, trincas, empenamento, entre outros.

Nas indústrias de processo três condições específicas tornam necessário um alto grau de confiabilidade para os equipamentos como vasos de pressão: trabalho em regime contínuo, submetendo os equipamentos a um regime severo de operação; cadeia contínua de equipamentos, na qual a falha ou paralisação de um único equipamento pode causar a paralisação de toda a instalação; armazenamento de fluídos inflamáveis, tóxicos ou em elevadas pressões e/ou temperaturas, condições nos quais uma falha pode resultar em um acidente grave.

Para prevenir tais falhas, a NR-13 prescreve realização de inspeção inicial de segurança, no local de operação do vaso pressão e antes da entrada em funcionamento, inspeção periódica, em períodos definidos pela mesma norma de acordo com a categoria do vaso de pressão em questão, e inspeção após qualquer dano ao vaso, reparo ou alteração importante, antes da volta em funcionamento para vasos inativos por mais de 12 meses ou movimentação do vaso [1].

Essas inspeções contam com exame visual externo e interno da estrutura e, no caso da inspeção inicial, com o teste de pressão hidrostático, no qual o vaso é preenchido com água e pressurizado até um dado valor de pressão, com a finalidade de avaliar a integridade, estanqueidade e a resistência estrutural dos componentes sujeitos à pressão, dentro das condições estabelecidas para a sua realização [1].

A N-2688 - Teste de Pressão em Serviço de Vasos de Pressão e Caldeiras da Petrobrás define três grupos de risco para vasos de pressão, de acordo com a pressão de teste e volume da estrutura verificada, como mostrado na Figura 12.

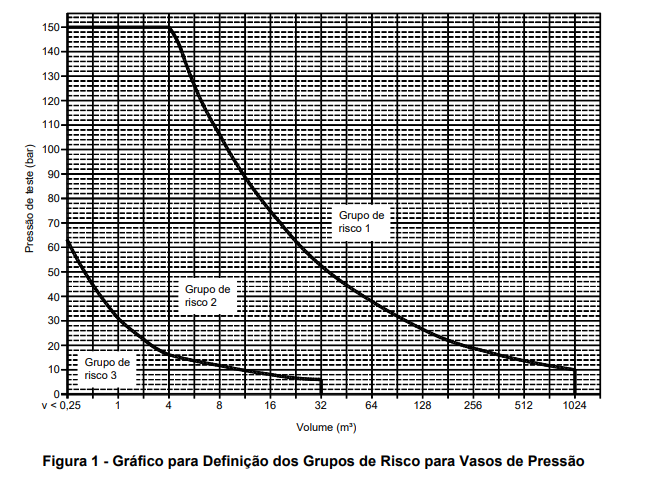


Figura 12 - Gráfico para definição dos grupos de risco para vasos de pressão

Fonte: N-2688

Segundo a N-2688, a pressão de teste é definida pelo profissional habilitado responsável pelo vaso de pressão e deve considerar os seguintes aspectos:

1. Código e norma de projeto de fabricação;
2. Código de inspeção em serviço aplicável;
3. Relação entre as condições de projeto e condições de operação;
4. Potencial de risco e localização do vaso na unidade industrial;
5. Histórico de resultado das inspeções de segurança internas e externas anteriores;
6. Histórico de resultado de testes de pressão anteriores;
7. Existência de descontinuidades no equipamento;
8. Avaliação da PMTA na condição atual do equipamento.

Geralmente, a pressão de teste situa entre 1,5 vezes a PMTA do vaso [2] e deve ser aplicada na estrutura de acordo com o grau de risco, como apresentados nas figuras Figura 13Figura 14 e Figura 15.

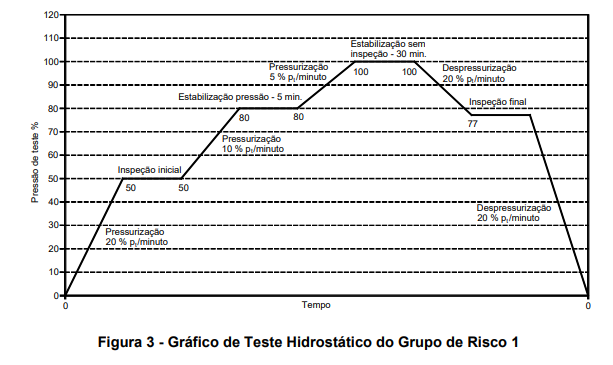


Figura 13 - Gráfico de teste hidrostático do grupo de risco 1

Fonte: N-2688

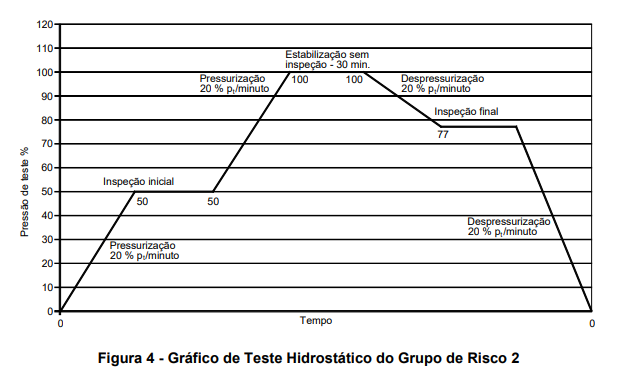


Figura 14 - Gráfico de teste hidrostático do grupo de risco 2

Fonte: N-2688

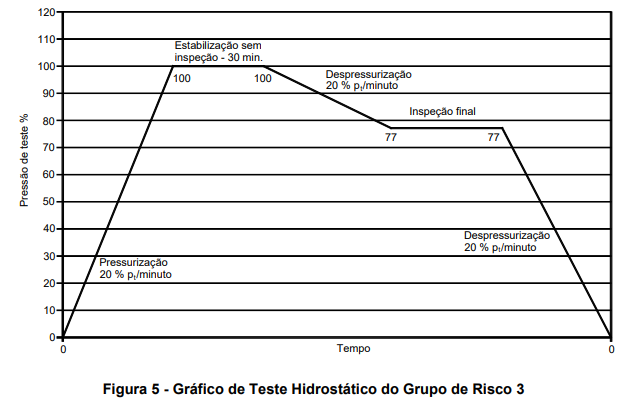


Figura 15 - Gráfico de teste hidrostático do grupo de risco 3

Fonte: N-2688

Após o ensaio hidrostático, deve ser realizada uma inspeção visual observando deformações, vazamentos e recalques, e é recomendada inspeção visual interna para avaliação da integridade do revestimento em equipamentos cladeados ou revestidos com tiras soldadas (*strip lining*) [5].

Muitas vezes a emissão acústica pode ser aplicada em conjunto com o ensaio pneumático em vasos de pressão, em substituição ao ensaio hidrostático, ou ainda, pode ser aplicada junto desse para acompanhamento de defeitos subcríticos ou não visíveis a nível macroscópico. O crescimento de trincas é acompanhado da geração de vários sinais (Hits) e vazamentos levam ao crescimento do sinal RMS [3] [4].

A integridade da estrutura também pode ser avaliada com o auxílio do efeito Kaiser. Para isso são realizados dois ciclos de pressão no vaso de pressão, desta forma, se for observada grande atividade acústica no segundo ciclo, há indício de defeitos na estrutura [3] [4].

## Localização

Nos softwares comerciais de EA, a localização de defeitos em componentes de geometria cilíndrica com tampos elipsoidais é determinada pela adaptação dos algoritmos de localização planar. A localização planar emprega nesses sistemas o método da diferença no tempo de chegada, ou do inglês (*Time Difference of Arrival* - TDOA), que localiza a fonte por cálculo geométrico em função das diferenças entre os tempos de chegada dos sinais detectados nos diferentes sensores arranjados na estrutura. Uma interpretação geométrica do cálculo da localização planar são as hipérboles, que são definidas como curvas na quais é constante a diferença das distâncias de cada um dos seus pontos a dois pontos fixos ou focos. Logo, para fontes localizadas sobre uma hipérbole cujos focos são dois sensores i e j o valor da diferença entre os tempos de chegada detectados em tais sensores, ti e tj, é constante. Quando é inserido mais um sensor no arranjo é possível traçar mais duas hipérboles. A intersecção das três hipérboles assim geradas ocorre sobre a fonte de emissão acústica, como é mostrado na Figura 16.

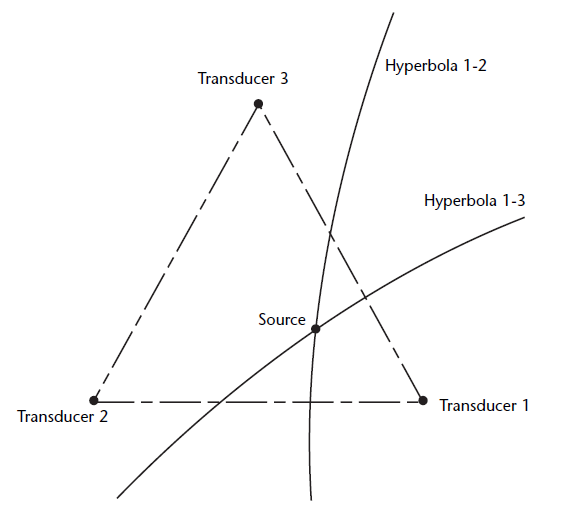


Figura 16 - Localização planar pelo método da hipérbole. T1, T2 e T3 são os tempos de chegada das ondas mecânicas nos sensores correspondentes

Fonte: 2008, Christian U. Grosse, Acoustic Emission Testing

Dado um sensor 1, um sensor 2 e uma fonte de emissão acústica com coordenadas (Xs, Ys), temos r1 a distância da fonte e o sensor 2, R a distância entre a fonte e sensor 1, θ o ângulo formado entre a linha que conecta o sensor 1 ao sensor q e a vertical, D a distância entre os dois sensores e Z a distância da fonte à reta que conecta os dois sensores.

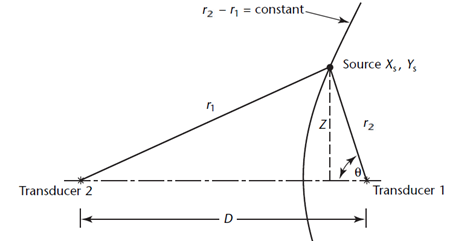


Figura 17 - Localização planar com dois sensores

Fonte: 2008, Christian U. Grosse, Acoustic Emission Testing

Sendo V a velocidade da onda detectada no material, obtemos:

|  |  |  |
| --- | --- | --- |
|  |  | (2.1) |
|  |  | (2.2) |
|  |  | (2.3) |

Substituindo a equação (2.2) na (2.3):

|  |  |  |
| --- | --- | --- |
|  |  | (2.4) |

Simplificando a expressão (2.4) :

|  |  |  |
| --- | --- | --- |
|  |  | (2.5) |

Isolando na equação (2.1) e substituindo em (2.7) se chega a:

|  |  |  |
| --- | --- | --- |
|  |  | (2.6) |

Inserindo um terceiro sensor, não alinhado aos dois anteriores, é possível obter outra equação semelhante a (2.8). A resolução simultânea das duas equações, ou seja, a intersecção entre duas hipérboles, resulta na posição da fonte de EA [3].

A localização planar de uma fonte de EA requer a utilização de no mínimo três sensores. Caso mais de três sensores forem usados se obtém um sistema sobredeterminado de equações, e métodos estatísticos, como o método dos mínimos quadrados podem ser empregados.

Para tanto é estabelecida uma função erro, calculada pela diferença entre os tempos de chegada medidos e os tempos de chegada calculados, pressupondo-se que o evento aconteceu em determinada posição (x,y).

|  |  |  |
| --- | --- | --- |
|  |  | (2.7) |
|  |  | (2.8) |
|  |  | (2.9) |

Onde:

|  |  |  |
| --- | --- | --- |
|  |  | (2.10) |

A localização da fonte é então calculada a partir da minimização da função erro, com um palpite inicial que pode ser a média geométrica da posição dos três sensores que apresentaram menor tempo de chegada. Entretanto, para calcular a distância relativa entre cada sensor emprega-se a planificação do corpo, gerando resultados insatisfatórios, principalmente para os tampos, que são as áreas mais deformadas. Para isso aplica-se nesse trabalho o conceito de geodésicas.

## Geodésica

Geodésica é a curva de menor comprimento que une dois pontos. No espaço euclidiano essa curva é um segmento de reta, mas na geometria riemanniana tal curva pode não ser uma reta.

Existem dois problemas aplicáveis na definição das geodésicas: no primeiro, chamado problema direto, define-se um ponto inicial (A), uma distância () e direção, ou azimute no ponto A (), e procura-se o ponto final (B); no problema indireto, tem-se o ponto inicial (A) e final (B), e procura-se a distância entre os pontos (). Esses e outros parâmetros cuja definição auxilia na resolução dos problemas, como longitude () e diferença de longitude (), são apresentados na Figura 18.

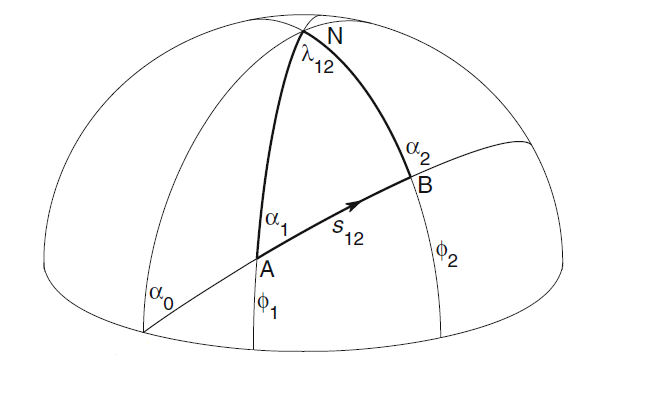


Figura 18 - Grandezas em um elipsóide de revolução

Fonte: Karney, 2011

As geodésicas são muito aplicadas em elipsoides de revolução, já que tal geometria representa adequadamente o formato da Terra. Essa geometria representa adequadamente também os tampos esféricos e elipsoidais de elementos de corpo cilíndrico, abordados nesse trabalho. Portanto, é de interesse a solução do problema indireto.

Karney (2011) apresenta uma solução numérica para esse problema que faz uso da capacidade computacional disponível atualmente. Nesse trabalho, é definido uma esfera auxiliar para o cálculo da distância e obtém-se como solução uma integral, cuja solução é aproximada por expansão de Taylor. Existe uma biblioteca em Python distribuída livremente que oferece essas ferramentas. Chamada *Geographiclib*, tal biblioteca disponibiliza os pacotes *Geodesic* e *GeodesicLine*, que apresentam erro de 36 nm no cálculo de distâncias em um elipsoide para fator de achatamento de 0,5 e um quarto da distância meridional igual a 10.000 km [6], e, portanto, são válidas para os propósitos desse trabalho.

## Otimização

### Evolução diferencial

A Evolução Diferencial (DE) é um algoritmo de otimização estocástico proposto por Rainer Storn e Kenneth Price em 1995 [7]. Uma das vantagens deste método é de não ser preciso determinar o gradiente da função objetivo, pois existem situações em que a determinação analítica do gradiente é impossível e sua determinação numérica pode levar a erros elevados.

O método de evolução diferencial pode ser descrito como uma manipulação de pontos candidatos à solução (PCS – ponto candidato à solução), através de mutações e cruzamentos, até que seja atingido um critério de parada.

Essas duas operações são aplicadas num conjunto inicial de PCSs, denominado de geração inicial, gerando um novo conjunto de indivíduos que será submetido à um processo de seleção, formando então uma nova geração que é novamente submetida à duas operações e então o ciclo se repete.

O PCS pode possuir diversos parâmetros, portanto este é um vetor de dimensão igual ao número de parâmetros existentes. O objetivo do método de otimização é minimizar o valor de uma função objetivo quando esta recebe como argumento os parâmetros do PCS.

* + - 1. Mutação

A operação de mutação gera um PCS a partir de três outros, para isso é utilizada a equação abaixo [8]:

|  |  |  |
| --- | --- | --- |
|  |  | (2.11) |

Onde:

* é o novo PCS
* é determinado o ponto base, onde será inserida a mutação
* é denominado de coeficiente de ponderação da mutação, que determina a influência da mutação na formação de novos PCSs
* são dois pontos escolhidos aleatoriamente dentro da geração atual.

Para a operação de mutação ainda deve ser definido um valor de taxa de recombinação, este está relacionado à probabilidade de ocorrer mutação numa população.

* + - 1. Cruzamento

O cruzamento, também denominado de *crossover,* é aplicado na população após a mutação, com o objetivo de se aumentar a diversidade da população (citar Price, K. V., 1999. An introduction to differential evolution. New Ideas in Optimization, 79–108.)

Esta operação consiste em gerar um novo PCS a partir da alteração de alguns parâmetros do ponto original. Para isso é seguida a regra abaixo

Seja

|  |  |  |
| --- | --- | --- |
|  |  | (2.12) |

Onde:

* é um valor gerado aleatoriamente
* é a taxa de recombinação para o cruzamento
* é o fator de ponderação para o cruzamento (colocar índices?)
* e são as j-ésimas componentes de dois pontos escolhidos aleatoriamente
  + - 1. Seleção

A etapa de seleção tem como objetivo diminuir o tamanho da geração, deixando apenas os indivíduos com maior aptidão para próxima geração.

Existem diversas abordagens utilizadas para a seleção de indivíduos, uma das mais simples, mas ainda assim amplamente utilizada é a competição, onde dois indivíduos da população são escolhidos aleatoriamente e o com menor aptidão é eliminado, sendo esse processo repetido até a população atingir um tamanho especificado.

* + - 1. Implementação

O pacote *SciPy* possui uma implementação do algoritmo de evolução diferencial bem avançada, onde é possível configurar os métodos de mutação, cruzamento e seleção com muita flexibilidade. Esse pacote está disponível para linguagem Python, mas possui implementações de alguns processos em outras linguagens, o que lhe confere excelente desempenho.

Pelos motivos apresentados acima, o método de evolução diferencial oferecido pelo pacote *SciPy* será utilizado para a solução dos problemas de otimização do presente trabalho.

### Método de Newton

# Método de seccionamento

A aplicação da biblioteca *Geographiclib* para localização de defeitos de emissão acústica é inviável devido ao tempo de execução das rotinas. Entretanto, não é necessária grande acurácia da localização devido a existência de outras fontes de erro mais significantes, como dispersão de onda e desvios de geometria e de medição.

Em vista disso, nesse trabalho será proposta uma nova técnica para o cálculo de distâncias nestes elipsoides. Essa técnica será referenciada no decorrer do trabalho como Seccionamento.

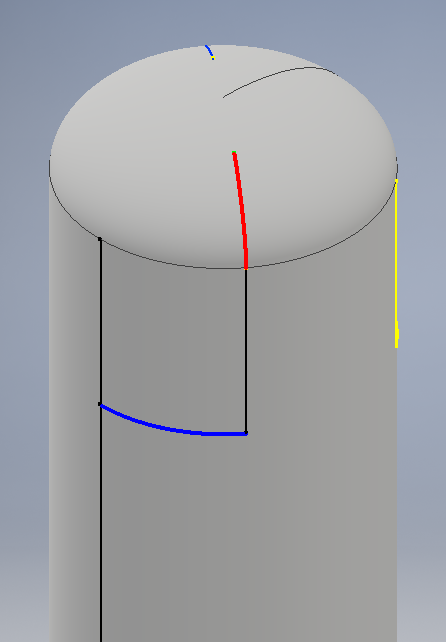
O método de seccionamento consiste em assumir que a geodésica entre dois pontos no elipsoide está contida em um plano que seja paralelo ao eixo de revolução do elipsoide e que contenha os dois pontos. Esta abordagem, é descrita na sequência.

### Coordenadas auxiliares

A primeira etapa consiste em determinar algumas coordenadas auxiliares dos pontos dos quais se deseja calcular a distância. Essas coordenadas referenciam os pontos na vista superior do tampo e determinam sua altura relativa neste.

Estes pontos devem ser determinados a partir das informações conhecidas dos pontos, as quais são a distância do ponto a uma geratriz do corpo cilíndrico e a distância do ponto à interface tampo/corpo. Estas são as informações usadas para se localizar um ponto na superfície do tampo por serem as mais fáceis de se obter. Em uma situação de trabalho em campo, estas informações podem ser obtidas com o auxílio de uma fita métrica flexível.

No decorrer do presente trabalho essas coordenadas serão denominadas x e s, respectivamente. Na Figura 19 há uma representação dessas coordenadas em um modelo de vaso de pressão de tampo elipsoidal para um certo ponto P0.



s

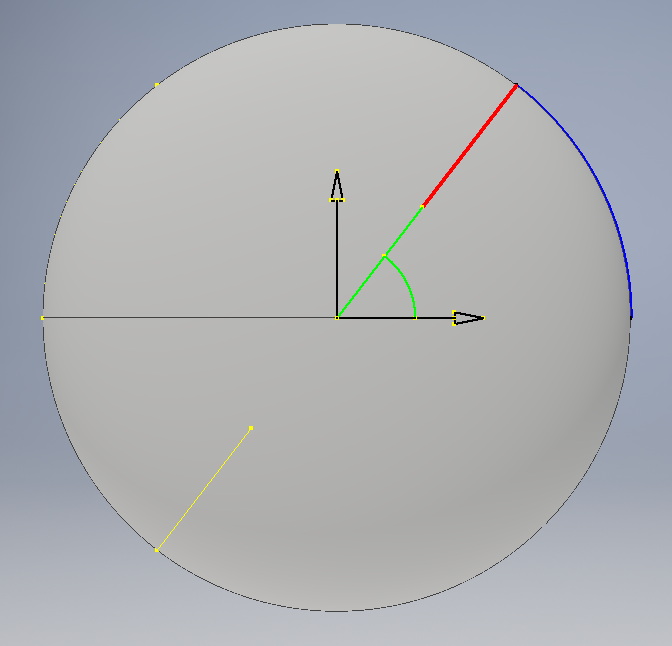
x

P0

Figura 19 - Identificação das coordenadas x e s

Quando o tampo é observado a partir de uma vista superior, o ponto P0 possuirá coordenadas diferentes. Essas coordenadas serão denominadas x0’ e y0’, como pode ser observado na Figura 20.

Essas coordenadas serão posteriormente utilizadas para se determinar o plano que contém a geodésica.



**x’**

**y’**

P0: (x0’, y0’)

**λ**

**R**

Figura 20 - Vista superior do tampo - coordenadas x0' e y0'

Para a determinação de x0’ e y0’, são usadas duas informações: a distância do ponto P0 até o centro do sistema de coordenadas, denominada R, e o ângulo que este forma com o eixo x’, denominado de λ, que é a longitude do ponto no elipsoide.

A longitude λ pode ser determinada com o valor de x a partir da equação (3.1)

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

Onde D é o diâmetro externo do cilindro.

Para a determinação de R é necessário um método incremental para o cálculo do comprimento de arco de elipse. Este método consiste em aproximar o comprimento do arco de elipse a partir a soma do comprimento de secantes. Na Figura 21 é apresentada a definição de uma secante da elipse.

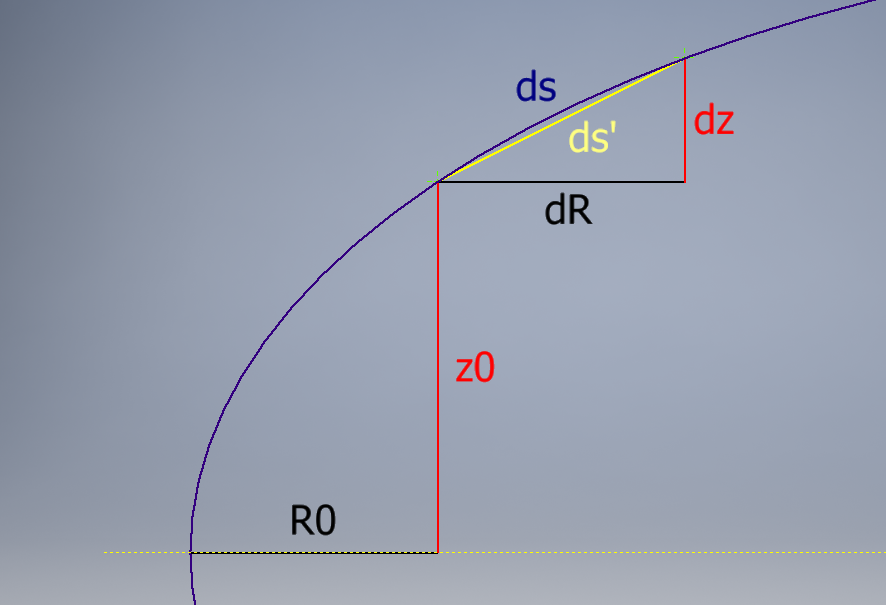


Figura 21 - Secante da elipse

Para um dR suficientemente pequeno, pode-se aproximar o valor de ds como sendo igual a ds’, sendo que este é calculado a partir das equações abaixo.

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |

|  |  |  |
| --- | --- | --- |
|  |  | (3.3) |

Onde:

* : raio do cilindro
* : razão de achatamento do tampo

O valor de R é determinado a partir do procedimento incremental descrito no pseudocódigo a seguir:

R = -a

s’ = 0

z0 = 0

ENQUANTO s’ < s:

R = R + dR

z = a \* f \* (1 – R^2 / a^2)^(1/2)

dz = z - z0

ds = (dR^2 + dz^2)^(1/2)

s’ = s’ + ds

z0 = z

FIM ENQUANTO

RETORNE R, z

Este algoritmo retornará o valor de R e z, que serão usados para calcular as coordenadas auxiliares a partir das seguintes equações:

|  |  |  |
| --- | --- | --- |
|  |  | (3.4) |
|  |  | (3.5) |
|  |  | (3.6) |

Onde os parâmetros são definidos na Figura 20 e é a altura do ponto relativo a base do tampo.

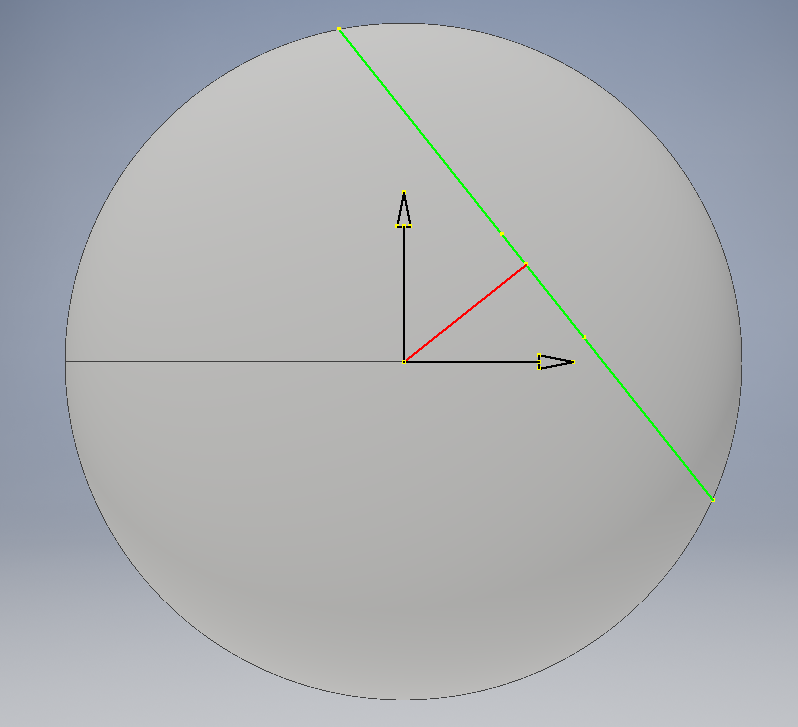
### Plano de seccionamento e elipse auxiliar

Com a informação das coordenadas auxiliares de dois pontos no elipsoide, pode-se determinar a distância do plano que contém esses pontos até o eixo de revolução do elipsoide. Para isso, é observada a vista superior do elipsoide, onde o plano que contém os dois pontos é observado como uma reta e o problema se reduz à determinação da distância entre reta e ponto.

Nesse sistema de coordenadas, o eixo de revolução do elipsoide se resume à origem do sistema de coordenadas e a distância *d* pode ser determinada a partir da equação (3.7), cujos parâmetros são definidos na Figura 22.

|  |  |  |
| --- | --- | --- |
|  |  | (3.7) |

A intersecção deste plano com o elipsoide resultará em uma elipse menor ou igual à elipse que deu origem ao elipsoide. Por se tratar de um elipsoide de revolução, as dimensões da elipse formada na interseção com o plano dependerão apenas da distância do plano até o eixo de revolução. Além disso, a razão de achatamento será a mesma para qualquer distância *d*, portanto só é preciso determinar um novo valor de raio maior (*a’*) para a elipse formada.



**x’**

**y’**

P1: (x1’, y1’)

P2: (x2’, y2’)

**d**

Figura 22 - Vista superior do plano de seccionamento

O valor de *a’* pode ser determinado fazendo o plano de intersecção normal a um dos eixos do sistema de coordenadas original e combinando a equação deste plano com a equação do elipsoide.

|  |  |  |
| --- | --- | --- |
| Plano: |  | (3.8) |

|  |  |  |
| --- | --- | --- |
| Elipsoide: |  | (3.9) |
|  |  |  |

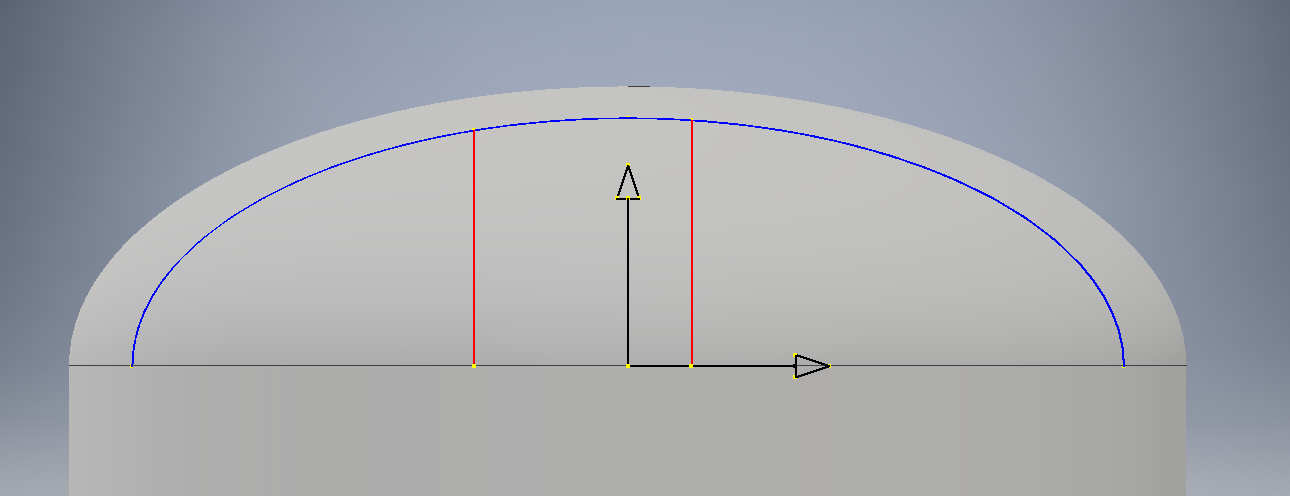
Combinando essas duas equações e realizando algumas manipulações algébricas se chega a:

|  |  |  |
| --- | --- | --- |
|  |  | (3.10) |

Portanto, os semieixos da elipse serão:

|  |  |  |
| --- | --- | --- |
|  |  | (3.11) |
|  |  | (3.12) |

Assim, observa-se que a razão de achatamento se mantém e os dois semieixos são reduzidos por um mesmo fator.



P1: (R1’, z1’)

P2: (R2’, z2’)

**z’**

**R’**

Figura 23 - Elipse auxiliar

Nessa elipse os pontos P1 e P2 possuirão coordenadas R’ e z’. A coordenada R’ pode ser determinada a partir do procedimento incremental descrito anteriormente para a determinação das coordenadas auxiliares; a coordenada z’ não se altera.

Agora pode-se calcular o comprimento do arco de elipse que liga os pontos P1 e P2. Para isso é utilizado o procedimento descrito pelo pseudocódigo a seguir.

Ri = mínimo(R1, R2)

Rf = máximo(R1, R2)

R = Ri

z0 = a’ \* f \* (1 – R^2 / a’^2)^(1/2)

s = 0

ENQUANTO R < Rf:

R = R + dR

z = a \* f \* (1 – R^2 / a^2)^(1/2)

dz = z - z0

ds = (dR^2 + dz^2)^(1/2)

s = s + ds

z0 = z

FIM ENQUANTO

RETORNA s

O valor retornado por esse procedimento será a distância entre os pontos P1 e P2 no elipsoide, que, para este método, será assumido como sendo a geodésica entre os pontos.

### Aproximações

Os procedimentos incrementais descritos anteriormente podem demandar grande custo computacional e tempo de processamento elevado se o incremento escolhido for muito pequeno. Entretanto, se o incremento for muito grande, isso resultará em erros elevados.

Para se contornar essa situação, foram ajustadas funções polinomiais para se representar os dados obtidos por esses procedimentos incrementais. Essas funções dependem do valor de *a* e *f.* A dependência com *a* é linear, portanto podem ser obtidas curvas normalizadas cujo resultado é posteriormente multiplicado por *a*. Para *f* não existe uma relação simples, portanto as curvas devem ser previamente obtidas em função de *f*. Nos gráficos abaixo foi adotado *f* igual a 0.5, que é o valor mais comum para tampos elípticos de vasos de pressão industriais [9].

Para se realizar as regressões foram usados os dados obtidos com os procedimentos incrementais descritos anteriormente. O algoritmo de regressão foi o disponível no pacote *Numpy*, um pacote livre do Python para métodos numéricos. Para a regressão da posição foi usado um polinômio de ordem 7 e para a regressão do comprimento de arco um de ordem 10.

A regressão da posição pode ser observada na Figura 24.

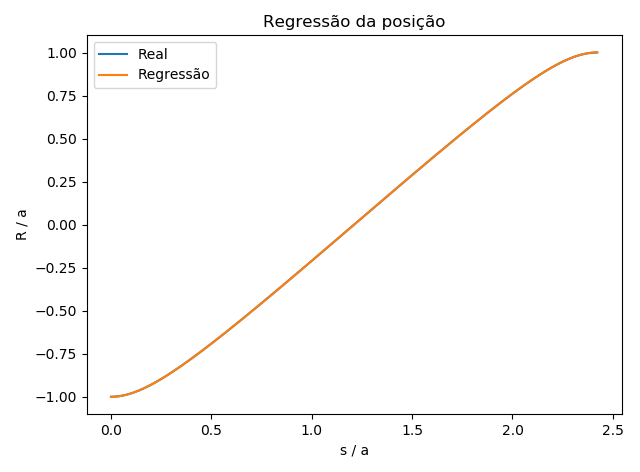


Figura 24 - Regressão Posição x Comprimento do arco

A função obtida foi:

|  |  |  |
| --- | --- | --- |
|  |  | (3.13) |

A regressão do arco pode ser observada na Figura 25.

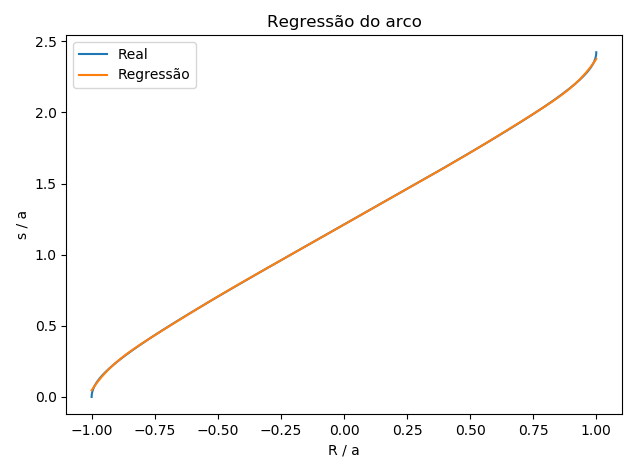


Figura 25 - Regressão Comprimento do arco x Posição

A função obtida foi:

|  |  |  |
| --- | --- | --- |
|  |  | (3.14) |

Com essas funções, não há a necessidade do procedimento incremental e a determinação da distância entre os pontos é muito mais rápida.

### Verificação

Para se verificar a precisão do método proposto para se calcular a distância entre pontos no elipsoide foi realizado o seguinte procedimento:

* Foi definido um ponto de referência com coordenadas (0, 0). Esse ponto fica na interface entre o tampo e o corpo cilíndrico.
* O segundo ponto teve suas coordenadas variadas de forma a cobrir todo o tampo. Para cada coordenada foi calculada a distância real (através da biblioteca *Geographiclib)*, a distância através do método de seccionamento e através do método de planificação
* O diâmetro foi mantido constante e todos os valores de distância e erro serão apresentados normalizados em relação ao diâmetro. A razão de achatamento foi mantida constante e igual a 0,5

Na Figura 26 são apresentadas as distâncias calculadas pelos diferentes métodos, sendo cada um representado por uma cor.

Para cada método existem 5 curvas, cada uma relativa a uma coordenada S do ponto variável. Essa coordenada foi variada de 0 até o semiperímetro do tampo, ou seja, até o vértice do tampo.

O eixo x está relacionado com a posição X normalizada do ponto variável.

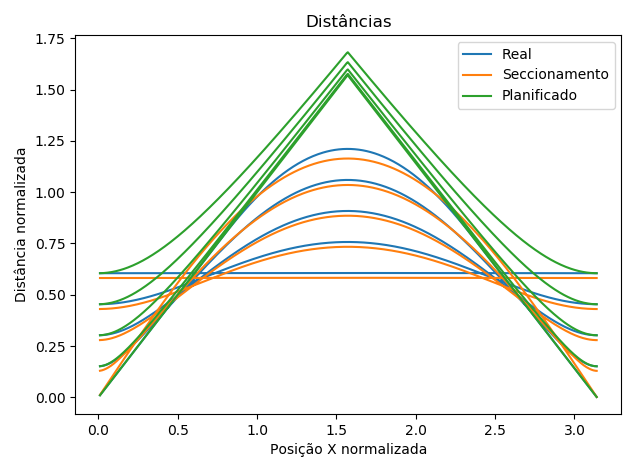


Figura 26 - Distâncias entre pontos no tampo para diferentes métodos de cálculo

Na Figura 27 é apresentado o erro absoluto normalizado para os diferentes métodos. Novamente são apresentadas 5 curvas por método, cada uma relativa a uma coordenada S do ponto variável.

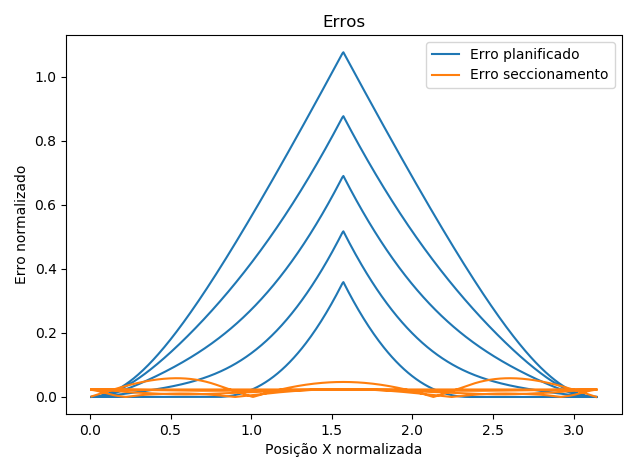


Figura 27 – Erro no cálculo da distância entre pontos no tampo para diferentes métodos

Na Figura 28 é apresentado o erro normalizado máximo no cálculo da distância pelo método de seccionamento. Esse erro representa o máximo erro encontrado para qualquer coordenada X para uma coordenada S fixa. Neste gráfico pode se observar que foram encontrados erros relativamente pequenos, chegando no máximo a 6% do valor do diâmetro.



Figura 28 - Erro máximo do método de seccionamento

# Algoritmo de localização

O algoritmo de localização tem como objetivo determinar a posição da fonte de sinal a partir dos tempos de chegada nos diferentes sensores instalados na estrutura. O algoritmo busca então um ponto na estrutura que proporcione os mesmos tempos de chegada nos sensores que as observadas pelo sinal real.

Como não é conhecido o tempo em que o sinal foi originado, é preciso definir uma outra referência de tempo. Como referência de tempo foi utilizado o tempo de chegada ao primeiro sensor.

Portanto, o que é medido experimentalmente é um conjunto de tempos de chegada em diferentes sensores:

|  |  |  |
| --- | --- | --- |
|  |  | (4.1) |

Onde é o tempo de chegada ao sensor i.

Esses tempos de chegada são então diminuídos do tempo de chegada ao primeiro sensor:

|  |  |  |
| --- | --- | --- |
|  |  | (4.2) |
|  |  | (4.3) |

Um ponto qualquer na estrutura possuirá coordenadas . Este ponto estará a uma determinada distância de cada sensor, desta forma, pode se definir um vetor de distâncias.

|  |  |  |
| --- | --- | --- |
|  |  | (4.4) |

Essas distâncias podem ser calculadas de diversas formas, no presente trabalho foi proposto o método de seccionamento, mas também foram testados os métodos planificado e através da biblioteca *Geographiclib.*

A partir do valor das distâncias e conhecendo a velocidade do som no material da estrutura, pode se determinar o vetor de tempos de chegada, como dessa vez se trata de uma fonte de sinal virtual, esse vetor será denominado de vetor de tempos de chegada teóricos.

|  |  |  |
| --- | --- | --- |
|  |  | (4.5) |

Onde é velocidade do som no material.

Para tornar compatível a informação de tempo dos sinais medidos com os tempos de chegada teóricos, deve se utilizar a mesma referência de tempo. Para os sinais medidos o valor de foi definido como o menor valor de tempo de chegada, ou seja, o tempo de chegada ao sensor mais próximo da fonte. O teórico será definido como o tempo de chegada ao sensor mais próximo da fonte de sinal real.

Com esta referência de tempo, pode se determinar o vetor de diferenças de tempo de chegada teóricas:

|  |  |  |
| --- | --- | --- |
|  |  | (4.6) |

O objetivo do algoritmo de minimização é determinar as coordenadas da fonte virtual que torne o mais próximo possível de .

Para se determinar a proximidade de dois vetores existem diversas abordagens. No presente trabalho se optou por usar uma média quadrática ponderada. Portanto, a proximidade dos vetores e , que também será a função custo do algoritmo de minimização, foi definida como:

|  |  |  |
| --- | --- | --- |
|  |  | (4.7) |

Sendo:

|  |  |  |
| --- | --- | --- |
|  |  | (4.8) |
|  |  | (4.9) |

Onde:

* : vetor de ponderação
* : coeficiente de normalização
* : ganho
* : diâmetro do corpo cilíndrico
* : altura do corpo cilíndrico

O vetor de ponderação tem por objetivo priorizar fazer com o algoritmo de minimização priorize os sensores mais próximos. Este vetor foi construído desta forma pois os sinais que chegam a sensores muito distante estão sujeitos à várias fontes de erro, como a atenuação do material, dispersão de onda, desvios de geometria e acessórios no vaso de pressão. O vetor de ponderação é ainda influenciado pelo parâmetro ganho, que determina o quão penalizadas serão as informações dos sensores distantes.

O coeficiente de normalização tem por objetivo tornar os valores da função objetivo semelhantes independentemente das dimensões da estrutura. Seu valor foi definido como o tempo decorrido para o som percorrer a maior distância entre dois pontos no corpo cilíndrico.

Portanto, o algoritmo de evolução diferencial buscará as coordenadas do ponto teórico que minimizem o valor da função objetivo

# Procedimento Experimental

## Análise numérica

Como uma primeira abordagem para se analisar a eficácia do método de localização a partir das distâncias calculadas com o método de seccionamento foram realizados uma sequência de testes numéricos.

No primeiro teste, foram gerados 25 pontos distribuídos por toda a superfície de uma geometria cilíndrica com tampos elipsoidais. Em seguida, foram definidas as posições de 10 sensores em um padrão triangular, 6 deles no corpo cilíndrico e 2 em cada tampo elipsoidal, e calculadas as distâncias de cada ponto em relação a cada sensor. Então, foi definida uma velocidade de propagação de onda igual a 5 km/s e se obteve as diferenças de tempo de chegada do sinal.

A partir de tal diferença de tempo de chegada do sinal, foi resolvido o problema de localização pela equação (2.9) e pelo método do seccionamento, apresentado no capítulo 3. Na Figura são apresentados os resultados para esse teste numérico.

No segundo teste, admitiu-se a região de intersecção entre tampo e vaso como zona de interesse. Foram distribuídos 25 pontos nessa zona de interesse, e adotou-se a mesma posição dos 10 sensores anteriores. Definindo-se novamente a velocidade de propagação de onda igual a 5 km/s, obtendo-se as diferenças de tempo e resolvendo o problema de localização com os dois métodos, obteve-se os resultados apresentados na Figura.

## Análise empírica

Esse ensaio foi realizado nas dependências do prédio LEME do instituto de pesquisa Lactec, utilizando os seguintes materiais:

1. Sistema de emissão acústica da marca \*\*\*\*;
2. Sensores da marca, de banda...;
3. Acoplante: graxa;
4. Punção para geração das ondas mecânicas;
5. Vaso de pressão;

Os sensores foram fixados ao vaso de pressão com auxílio de porta sensor desenvolvido pelo Lactec e com o uso de acoplante, na mesma posição daqueles da análise numérica. O acoplamento foi verificado de acordo com a norma...,, obtendo-se os seguintes resultados:



Figura 29 - Croqui da distribuição dos sensores no vaso

Fonte: Autores

Tabela 1 - Tabela de verificação do acoplamento dos sensores

|  |  |  |  |
| --- | --- | --- | --- |
| Sensor | 1ª Medição | 2 ª Medição | 3 ª Medição |
| Sensor 1 |  |  |  |
| Sensor 2 |  |  |  |



Figura 30 – Foto da montagem dos sensores no vaso de pressão

Fonte: Autores

# Resultados

# Bibliografia

x

|  |  |
| --- | --- |
| 1. | MINISTÉRIO DO TRABALHO. **NR-13: Caldeiras, Vasos de Pressão e Tubulação**. [S.l.], p. 23. 2017. |
| 2. | FILHO, J. D. S. P. **Análise de Efeitos de Teste Hidrostático em Vasos de Pressão**. Florianópolis. 2004. |
| 3. | CARLO GIUSEPPE FILIPPIN, J. B. J. A. D. R. N. E. A. **Emissão Acústica - Conceitos e Aplicações**. Curitiba: Grafo Estúdio, 2017. |
| 4. | SÉRGIO DAMASCENO SOARES, N. C. D. M. **Apostila de Emissão Acústica**. [S.l.]: [s.n.]. |
| 5. | PETROBRÁS. **N-2688: Teste de Pressão em Serviço de Vasos de**. [S.l.], p. 15. 2014. |
| 6. | KARNEY, C. F. F. Geographiclib, 10 maio 2017. Disponivel em: <https://geographiclib.sourceforge.io/html/index.html>. |
| 7. | ARAUJO, R. L. D. **Evolução Diferencial para Problemas de Otimização com Restrições Lineares**. Universidade Federal de Juiz de Fora. Juiz de Fora, p. 82. 2016. |
| 8. | RAINER STORN, K. P. **Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces**. [S.l.], p. 15. 1996. |
| 9. | DONATO, G. V. P. **Inspeção de vasos de pressão**. Instituto Brasileiro de Petróleo, Gás e Biocombustíveis. [S.l.], p. 147. 2013. |
| 10. | NATIONAL SCIENCE FOUNDATION. Introduction to Acoustic Emission Testing. **NDT Resource Center**, 2001. Disponivel em: <https://www.nde-ed.org/EducationResources/CommunityCollege/Other%20Methods/AE/AE\_Index.php>. |
| 11. | KAISER, J. **Untersuchung über das Auftreten von Geräuschen beim Zugversuch**. München. 1950. |
| 12. | KARNEY, C. F. F. Algorithms for geodesics, 26 jun. 2012. 13. |
| 13. | PRICE, K. V. An introduction to differential evolution. New Ideas in Optimization. [S.l.]: [s.n.], 1999. p. 79–108. |
| 14. | SCIPY.ORG. **SciPy**, 2016. Disponivel em: <https://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.optimize.differential\_evolution.html>. Acesso em: 24 Novembro 2018. |

x

Anexo I: Código Python

import geographiclib.geodesic as geo

import matplotlib.pyplot as plt

import scipy.optimize as opt

import math as m

import numpy as np

import copy

import time

from Routines.fastCalc import \*

from Routines.SectionAux import elipseArcReg

class VesselPoint():

"""Classe para definir as propriedades de um ponto no vaso

"""

def \_\_init\_\_(self, Xcord, Ycord, ID):

self.Xcord = Xcord

self.Ycord = Ycord

self.ID = ID

"""

if type(Xcord) is not float and type(Xcord) is not np.float64:

print("Erro! ID: " + str(ID))

print(Xcord)

print(type(Xcord))

print("\n")

"""

def SetXcord(self, Xcord):

self.Xcord = Xcord

def SetLon(self, Lon):

self.Lon = Lon

def SetLat(self, Lat):

self.Lat = Lat

def SetOnCap(self, OnCap):

self.OnCap = OnCap

def SetCap(self, Cap):

self.Cap = Cap

def SetXcap(self, X):

self.Xcap = X

def SetYcap(self, Y):

self.Ycap = Y

def SetZcap(self, Z):

self.Zcap = Z

def \_\_str\_\_(self):

text = "ID: " + \

str(self.ID) + " x: " + str(self.Xcord) + " y: " + str(self.Ycord)

return text

class CylindricalLocation():

def \_\_init\_\_(self, diameter, height):

self.diameter = diameter

self.height = height

self.SensorList = []

self.veloc = 1

self.\_\_SensorID = -1

self.\_\_tempSensorList = None

self.CalcMode = 'geodesic'

"""Modos:

geodesic - usando biblioteca do Python - GeoplotLib

section - usando seccionamento do tampo

"""

self.SectionMode = 'reg'

"""Modos:

reg: usa regressão para calcular arco

inc: usa método incremental para calcular o arco

"""

self.\_\_ellipseDivs = 500

self.\_\_DivsTolerance = 100

self.numba = True

# Inicialização dos tempos acumulados

self.t\_samecap = 0

self.t\_wallcap = 0

self.t\_wall = 0

self.t\_captocap = 0

self.i\_samecap = 0

self.i\_wallcap = 0

self.i\_wall = 0

self.i\_captocap = 0

# Setters & getters

def setCalcMode(self, mode):

self.CalcMode = mode

"""Modos:

geodesic - usando biblioteca do Python - GeoplotLib

section - usando seccionamento do tampo

"""

def setSectionMode(self, mode):

self.SectionMode = mode

"""Modos:

reg: usa regressão para calcular arco

inc: usa método incremental para calcular o arco

"""

def set\_f(self, f):

self.f = f

self.cap = geo.Geodesic(self.diameter / 2, f)

result = self.cap.Inverse(lat1=0, lon1=0, lat2=90, lon2=0)

self.SemiPerimeter = result.get("s12")

self.\_\_regPolys()

def set\_semiPerimeter(self, SemiPerimeter):

"""Definição do valor do semiperímetro e calculo do valor de f correspondente

Arguments:

SemiPerimeter {[float]} -- [medida do semiperímetro]

Returns:

f[float] -- [razão de achatamento]

"""

self.SemiPerimeter = SemiPerimeter

def CalcSemiPerimeter(f):

cap = geo.Geodesic(self.diameter / 2, f)

result = cap.Inverse(lat1=0, lon1=0, lat2=90, lon2=0)

CalcSP = result.get('s12')

return CalcSP

res = opt.minimize(lambda x: (CalcSemiPerimeter(

x) - SemiPerimeter)\*\*2, bounds=[(0, 0.999)], method='L-BFGS-B', x0=0.5)

f = res.get("x")[0]

self.set\_f(f)

self.cap = geo.Geodesic(self.diameter / 2, self.f)

def SetVelocity(self, velocity):

"Definição da velocidade em mm/s"

self.veloc = velocity

def GetSensorCoords(self, ID):

sensor = self.\_\_getSensorbyID(ID)

return (sensor.Xcord, sensor.Ycord)

# Periféricos

def PrintAllSensors(self):

print("Sensores originais:")

for sensor in self.SensorList:

print(sensor)

def FindFurthestPoint(self):

# Revisar este função

def CalcDistRemotePoint(x):

distances = self.calcAllDist(SourceX=x[0], SourceY=x[1], IDs=[-1])

return np.min(distances)

def CallBack(xk):

print(xk)

maxDist = CalcDistRemotePoint(xk)

print("Max distance: " + str(maxDist))

BruteRes = opt.brute(lambda x: -CalcDistRemotePoint(x), ranges=[(

0, self.diameter \* m.pi), (-self.SemiPerimeter, self.height + self.SemiPerimeter)], Ns=5)

# print(BruteRes)

res = opt.minimize(lambda x: -CalcDistRemotePoint(x), method='L-BFGS-B', bounds=[(

0, self.diameter \* m.pi), (-self.SemiPerimeter, self.height + self.SemiPerimeter)], x0=BruteRes, callback=CallBack, options={'maxfun': 200, 'ftol': 0.0000001})

return res.get('x')

def \_\_initializeTimes(self):

self.t\_samecap = 0

self.t\_wallcap = 0

self.t\_wall = 0

self.t\_captocap = 0

self.i\_samecap = 0

self.i\_wallcap = 0

self.i\_wall = 0

self.i\_captocap = 0

def \_\_printTimes(self):

print("\nTempos de cada modo de cálculo de distâncias")

try:

print("Mesmo tampo: " + str(round(self.t\_samecap, 3)) +

" em " + str(self.i\_samecap) + " avaliações / tempo médio: " + str(round(self.t\_samecap / self.i\_samecap, 4)) + " s")

except:

pass

try:

print("Corpo tampo: " + str(round(self.t\_wallcap, 3)) +

" em " + str(self.i\_wallcap) + " avaliações / tempo médio: " + str(round(self.t\_wallcap / self.i\_wallcap, 4)) + " s")

except:

pass

try:

print("Tampo tampo: " + str(round(self.t\_captocap, 3)) +

" em " + str(self.i\_captocap) + " avaliações / tempo médio: " + str(round(self.t\_captocap / self.i\_captocap, 4)) + " s")

except:

pass

try:

print("Corpo: " + str(round(self.t\_wall, 3)) +

" em " + str(self.i\_wall) + " avaliações / tempo médio: " + str(round(self.t\_wall / self.i\_wall, 4)) + " s")

except:

pass

print("\n")

# Auxiliares

def \_\_getSensorbyID(self, ID):

for sensor in self.SensorList:

if sensor.ID == ID:

return sensor

return None

def \_\_AuxCoords(self, Coords):

"""

Função para calcular as coordenadas auxliares (latitude e longitude) quando a coordenada estiver no tampo

"""

if Coords.Ycord >= self.height or Coords.Ycord <= 0:

if self.CalcMode == 'geodesic':

"""Calculo de variáveis auxiliares necessárias para o uso da geodésicas

"""

lon = Coords.Xcord / (self.diameter \* m.pi) \* 360 - 180

if Coords.Ycord >= self.height:

Coords.SetCap("sup")

s12 = Coords.Ycord - self.height

else:

s12 = abs(Coords.Ycord)

Coords.SetCap("inf")

EndCap = self.cap.Direct(lat1=0, lon1=lon, s12=s12, azi1=0)

lat = EndCap.get("lat2")

Coords.SetLat(lat)

Coords.SetLon(lon)

elif self.CalcMode == 'section':

"""Cálculo de variáveis auxiliares para o uso do seccionamento do tampo

"""

lon = Coords.Xcord / (self.diameter \* m.pi) \* 2 \* m.pi - m.pi

if Coords.Ycord >= self.height:

Coords.SetCap("sup")

s = Coords.Ycord - self.height

else:

s = abs(Coords.Ycord)

Coords.SetCap("inf")

(R, z) = self.\_\_sectionPos(s)

R = abs(R)

Coords.Xcap = R \* m.cos(lon)

Coords.Ycap = R \* m.sin(lon)

Coords.Zcap = z

else:

print("Modo inválido")

Coords.SetOnCap(True)

else:

Coords.SetOnCap(False)

def \_\_AllSensorsAuxCoords(self):

for sensor in self.SensorList:

self.\_\_AuxCoords(sensor)

def AddSensor(self, Xcord, Ycord):

# Conditions

C1 = Xcord >= 0 and Xcord <= self.diameter \* m.pi

C2 = Ycord > - self.SemiPerimeter \* \

1.01 and Ycord < (self.height + self.SemiPerimeter) \* 1.01

if C1 and C2:

self.\_\_SensorID += 1

ID = self.\_\_SensorID

SensorCoords = VesselPoint(Xcord, Ycord, ID)

self.\_\_AuxCoords(SensorCoords)

self.SensorList.append(SensorCoords)

else:

print("As coordenadas deste ponto estão fora do vaso")

def StructuredSensorDistribution(self, lines, sensorsInLine, x0, y0, dx, dy, aligned):

for i in range(0, lines):

if not aligned and (-1)\*\*(i + 1) == 1:

x1 = x0 + dx / 2

else:

x1 = x0

y1 = y0 + i \* dy

for j in range(0, sensorsInLine):

x = x1 + j \* dx

y = y1

self.AddSensor(Xcord=x, Ycord=y)

def \_\_removeSensors(self, IDs):

# Remove os sensores que não estão na lista de IDs

if IDs == [-1]:

pass

else:

ValidSensors = []

InvalidSensors = []

for sensor in self.SensorList:

try:

IDs.index(sensor.ID)

ValidSensors.append(sensor)

except:

InvalidSensors.append(sensor)

self.SensorList = ValidSensors

self.\_\_tempSensorList = InvalidSensors

def \_\_returnSensors(self):

# Os sensores sempre voltam ordenados às suas posições

if not self.\_\_tempSensorList == None:

temp = self.SensorList + self.\_\_tempSensorList

self.SensorList = [None] \* len(temp)

for sensor in temp:

self.SensorList[sensor.ID] = sensor

def \_\_orderMembers(self, TimesToSensors):

type = [('ID', int), ('time', float)]

data = np.array(TimesToSensors, dtype=type)

OrderedMembers = np.sort(data, order='ID')

return OrderedMembers

def \_\_orderByTime(self, TimesToSensors):

dtype = [("ID", int), ("time", float)]

NPtimes = np.array(TimesToSensors, dtype=dtype)

NPtimes = np.sort(NPtimes, order="time")

return NPtimes

# Seccionamento

def \_\_regPolys(self):

polArc, polPos = elipseArcReg(self.f, 7, 100000)

self.polArc = polArc

self.polArc\_f = polArc[::-1]

self.polPos = polPos

self.polPos\_f = polPos[::-1]

def \_\_sectionPos(self, s):

if self.SectionMode == "reg":

a = self.diameter / 2

f = self.f

s = s / a

pol = self.polPos

polF = self.polPos\_f

if self.numba:

y = sectionPos(s, polF)

R = y \* a

else:

R = np.polyval(pol, s) \* a

try:

z = a \* f \* m.sqrt(1 - R\*\*2 / a\*\*2)

except ValueError:

if abs(a - abs(R)) < (a / self.\_\_DivsTolerance):

R = a

z = 0

else:

print("SectionPos")

print("a: " + str(a))

print("s: " + str(s))

print("R: " + str(R))

z = np.nan

R = np.nan

elif self.SectionMode == "inc":

sf = s

f = self.f

a = self.diameter / 2

R1 = a

s = 0

z1 = 0

dR = 2 \* a / self.\_\_ellipseDivs

while s < sf:

R2 = R1 - dR

z2 = a \* f \* m.sqrt(1 - R2\*\*2 / a\*\*2)

ds = m.sqrt((R2 - R1)\*\*2 + (z2 - z1)\*\*2)

s += ds

R1 = R2

z1 = z2

R = R1

z = z1

else:

print("Modo inexistente")

return (R, z)

def \_\_sectionArc(self, a, R1, R2):

Ri = min([R1, R2])

Rf = max([R1, R2])

f = self.f

if self.SectionMode == "reg":

Ri = Ri / a

Rf = Rf / a

pol = self.polArc

polF = self.polArc\_f

if self.numba:

y = sectionArc(Ri, Rf, polF)

s = y \* a

else:

si = np.polyval(pol, Ri) \* a

sf = np.polyval(pol, Rf) \* a

s = sf - si

elif self.SectionMode == "inc":

s = 0

z1 = a \* f \* m.sqrt(1 - Ri\*\*2 / a\*\*2)

dR = (Rf - Ri) / self.\_\_ellipseDivs

radius = np.linspace(Ri, Rf, num=self.\_\_ellipseDivs)

for R in radius:

z2 = a \* f \* m.sqrt(1 - R\*\*2 / a\*\*2)

ds = m.sqrt(dR\*\*2 + (z2 - z1)\*\*2)

s += ds

z1 = z2

else:

print("Modo inexistente")

return s

def \_\_centerLineDistance(self, point1, point2):

a = self.diameter / 2

N = self.\_\_DivsTolerance

x1 = point1.Xcap

y1 = point1.Ycap

x2 = point2.Xcap

y2 = point2.Ycap

if abs(x1 - x2) < a / N and abs(y1 - y2) < a / N:

# Evita erros no cálculo da distância até o centro quando P1 e P2 estão muito próximos

d = a

else:

d = np.abs(x2 \* y1 - y2 \* x1) / \

np.sqrt((y2 - y1)\*\*2 + (x2 - x1)\*\*2)

return d

def \_\_reductionFactor(self, point1, point2):

"""Redução do diâmetro da elipse em função da sua distância do centro

"""

a = self.diameter / 2

d = self.\_\_centerLineDistance(point1, point2)

redF = np.sqrt((a\*\*2 - d\*\*2) / a\*\*2)

if np.isnan(redF) or a == d:

redF = 0

return redF, d

# Distâncias e tempos

def ExternalCalcDist(self, x1, y1, x2, y2):

auxMode = self.CalcMode

self.setCalcMode('geodesic')

P1 = VesselPoint(x1, y1, -3)

P2 = VesselPoint(x2, y2, -3)

self.\_\_AuxCoords(P1)

self.\_\_AuxCoords(P2)

dist = self.\_\_calcDist(P1, P2)

self.setCalcMode(auxMode)

return dist

def \_\_calcDist(self, P1, P2):

"""

Função para calcular distância entre dois pontos em posições quaisquer do vaso

"""

if P1.OnCap and P2.OnCap:

if P1.Cap == P2.Cap: # Dois pontos no mesmo tampo

t0 = time.time()

dist = self.\_\_DistSameCap(P1, P2)

t1 = time.time()

self.t\_samecap += t1 - t0

self.i\_samecap += 1

else: # Pontos em tampos opostos

if P1.Cap == "sup":

Psup = P1

Pinf = P2

else:

Psup = P2

Pinf = P1

t0 = time.time()

dist = self.\_\_DistCaptoCap(Psup, Pinf)

t1 = time.time()

self.t\_captocap += t1 - t0

self.i\_captocap += 1

# Distância entre um ponto no casco e outro no tampo

elif P1.OnCap ^ P2.OnCap:

t0 = time.time()

dist = self.\_\_DistWalltoCap(P1, P2)

t1 = time.time()

self.t\_wallcap += t1 - t0

self.i\_wallcap += 1

else: # Distância entre pontos no casco

t0 = time.time()

if self.numba:

# Identificar onde as coordenadas estão sendo definidas como vetores

x1 = float(P1.Xcord)

x2 = float(P2.Xcord)

y1 = float(P1.Ycord)

y2 = float(P2.Ycord)

d = float(self.diameter)

dist = wallDist(x1, y1, x2, y2, d)

else:

dist1 = np.sqrt((P1.Xcord - P2.Xcord) \*\*

2 + (P1.Ycord - P2.Ycord)\*\*2)

# Clone à direita

dist2 = np.sqrt((P1.Xcord - P2.Xcord + self.diameter \*

m.pi) \*\* 2 + (P1.Ycord - P2.Ycord)\*\*2)

# Clone à esquerda

dist3 = np.sqrt((P1.Xcord - P2.Xcord - self.diameter \*

m.pi) \*\* 2 + (P1.Ycord - P2.Ycord)\*\*2)

dist = np.min([dist1, dist2, dist3])

t1 = time.time()

self.t\_wall += t1 - t0

self.i\_wall += 1

return dist

def \_\_DistSameCap(self, P1, P2):

if self.CalcMode == 'geodesic':

res = self.cap.Inverse(

lat1=P1.Lat, lat2=P2.Lat, lon1=P1.Lon, lon2=P2.Lon)

dist = res.get("s12")

elif self.CalcMode == 'section':

redF, d = self.\_\_reductionFactor(P1, P2)

if redF != 0:

if False:

# Método otimizado com o Numba

x1 = float(P1.Xcap)

x2 = float(P2.Xcap)

y1 = float(P1.Ycap)

y2 = float(P2.Ycap)

diam = float(self.diameter)

divs = float(self.\_\_DivsTolerance)

tol = diam / divs

u1, u2 = distCap(redF, x1, y1, x2, y2, tol, d)

if True:

# Método convencional - numpy

r1q = P1.Xcap\*\*2 + P1.Ycap\*\*2

try:

u1 = m.sqrt(r1q - d\*\*2)

except ValueError:

if abs(r1q - d) < self.diameter / self.\_\_DivsTolerance:

u1 = 0

else:

u1 = np.nan

r2q = P2.Xcap\*\*2 + P2.Ycap\*\*2

try:

u2 = m.sqrt(r2q - d\*\*2)

except ValueError:

if abs(r2q - d) < self.diameter / self.\_\_DivsTolerance:

u2 = 0

else:

u2 = np.nan

v1c = np.array([-P1.Xcap, -P1.Ycap])

v2c = np.array([-P2.Xcap, -P2.Ycap])

v12 = np.array([P2.Xcap - P1.Xcap, P2.Ycap - P1.Ycap])

cosTheta1 = np.dot(v1c, v12) / \

(np.linalg.norm(v1c) \* np.linalg.norm(v12))

cosTheta2 = np.dot(v2c, v12) / \

(np.linalg.norm(v2c) \* np.linalg.norm(v12))

if cosTheta1 <= 0:

# Ponto está do outro lado do centro da elipse

u1 = -u1

if cosTheta2 <= 0:

# Ponto está do outro lado do centro da elipse

u2 = -u2

a = redF \* self.diameter / 2

dist = self.\_\_sectionArc(a, u1, u2)

else:

dist = 0

else:

print("Modo inexistente")

return dist

def \_\_DistWalltoCap(self, P1, P2):

if P1.OnCap:

Pcap = P1

Pwall = P2

else:

Pcap = P2

Pwall = P1

if Pcap.Cap == "sup":

Yaux = self.height

else:

Yaux = 0

AuxPoint = VesselPoint(0.0, Yaux, -2)

AuxPoint.SetCap(Pcap.Cap)

def CalcWallToCap(Xaux):

"""[Função para cálculo de distância entre um ponto no casco e outro no tampo]

Arguments:

Xaux {[float]} -- [Posição x do ponto auxiliar: ponto de transição casco-tampo]

Returns:

[totalDist] -- [Distância total entre pontos]

"""

AuxPoint.SetXcord(Xaux)

# Abordagem não otimizada

self.\_\_AuxCoords(AuxPoint)

AuxPoint.SetOnCap(False)

# Coordenadas auxiliares seccionamento - método otimizado

"""

lon = Xaux / (self.diameter \* m.pi) \* 2 \* m.pi - m.pi

R = self.diameter / 2

Xcap = R \* np.cos(lon)

Ycap = R \* np.sin(lon)

Zcap = 0

AuxPoint.Xcap = Xcap

AuxPoint.Ycap = Ycap

AuxPoint.Zcap = Zcap

# Coordenadas auxiliares geodesic

AuxPoint.Lat = 0

AuxPoint.Lon = lon

"""

dist1 = self.\_\_calcDist(Pwall, AuxPoint)

AuxPoint.SetOnCap(True)

dist2 = self.\_\_calcDist(AuxPoint, Pcap)

totalDist = dist1 + dist2

return totalDist

InitGuess = opt.brute(

CalcWallToCap, ((0, m.pi \* self.diameter),), Ns=6)

FinalSearch = opt.minimize(

CalcWallToCap, x0=InitGuess, method="BFGS", options={'maxiter': 5})

# print(FinalSearch) -- Resultado da minimização

dist = FinalSearch.get("fun")

return dist

def \_\_DistCaptoCap(self, Psup, Pinf):

AuxPoint1 = VesselPoint(0.0, self.height, -2)

AuxPoint2 = VesselPoint(0.0, 0.0, -2)

def CalcCaptoCap(Xaux):

"""[Função para calcular a distância entre pontos em tampos opostos]

Arguments:

Xaux {[float array]} -- [vetor com as coordenadas x dos pontos auxiliares]

Returns:

[dist] -- [distância entre os pontos]

"""

AuxPoint1.SetXcord(Xaux[0])

AuxPoint2.SetXcord(Xaux[1])

self.\_\_AuxCoords(AuxPoint1)

self.\_\_AuxCoords(AuxPoint2)

AuxPoint1.SetOnCap(True)

dist1 = self.\_\_calcDist(Psup, AuxPoint1)

AuxPoint1.SetOnCap(False)

AuxPoint2.SetOnCap(False)

dist2 = self.\_\_calcDist(AuxPoint1, AuxPoint2)

AuxPoint2.SetOnCap(True)

dist3 = self.\_\_calcDist(AuxPoint2, Pinf)

totalDist = dist1 + dist2 + dist3

return totalDist

# Chute inicial com otimização bruta

SearchRange = (0, m.pi \* self.diameter)

InitGuess = opt.brute(CalcCaptoCap, (SearchRange, SearchRange), Ns=6)

FinalSearch = opt.minimize(CalcCaptoCap, x0=InitGuess, method="BFGS")

# print(FinalSearch) # -- Resultado da minimização

dist = FinalSearch.get("fun")

MinPosSup = FinalSearch.get("x")[0]

MinPosInf = FinalSearch.get("x")[1]

AuxPoint1.SetXcord(MinPosSup)

AuxPoint2.SetXcord(MinPosInf)

return dist

def \_\_DistVClone(self, Source, Sensor):

SemiHeight = self.height / 2

dx1 = abs(Source.Xcord - Sensor.Xcord)

dx2 = abs(Source.Xcord - Sensor.Xcord + m.pi \* self.diameter)

dx3 = abs(Source.Xcord - Sensor.Xcord - m.pi \* self.diameter)

dx = np.min([dx1, dx2, dx3])

dy = Sensor.Ycord - Source.Ycord

d = np.sqrt(dx\*\*2 + dy\*\*2)

# É apenas uma aproximação, tem como calcular melhor esse valor

capDistance = dx \* self.SemiPerimeter / (m.pi \* self.diameter)

case1 = (Source.Ycord + Sensor.Ycord + capDistance) <= d

case2 = (2 \* self.height - Source.Ycord +

Sensor.Ycord + capDistance) <= d

Cond1 = case1 or case2

Cond2 = not (Source.OnCap or Sensor.OnCap)

if Cond1 and Cond2:

if Source.Ycord > SemiHeight:

YAuxCord = self.height

else:

YAuxCord = 0

AuxPoint1 = VesselPoint(Source.Xcord, YAuxCord, -2)

AuxPoint2 = VesselPoint(Sensor.Xcord, YAuxCord, -2)

self.\_\_AuxCoords(AuxPoint1)

self.\_\_AuxCoords(AuxPoint2)

AuxPoint1.SetOnCap(False)

dist1 = self.\_\_calcDist(Source, AuxPoint1)

AuxPoint1.SetOnCap(True)

AuxPoint2.SetOnCap(True)

dist2 = self.\_\_calcDist(AuxPoint1, AuxPoint2)

AuxPoint2.SetOnCap(False)

dist3 = self.\_\_calcDist(AuxPoint2, Sensor)

dist = dist1 + dist2 + dist3

else:

dist = -1

return dist

def calcAllDist(self, SourceX, SourceY, IDs):

"""

# Inicialização dos tempos acumulados de cálculo de distâncias - medição de performance

self.\_\_initializeTimes()

"""

self.\_\_removeSensors(IDs)

Source = VesselPoint(SourceX, SourceY, -1)

self.\_\_AuxCoords(Source)

t0 = time.time()

MinDistances = np.zeros(len(self.SensorList))

i = -1

for sensor in self.SensorList:

i += 1

distDirect = self.\_\_calcDist(Source, sensor)

distVClone = -1

distVClone = self.\_\_DistVClone(Source, sensor)

if distVClone == -1:

distVClone = distDirect \* 10

else:

"""

print("Distância direta: " + str(distDirect))

print("Clone vertical :" + str(distVClone))

print("\n")

"""

Distances = [distDirect, distVClone]

MinDistances[i] = np.min(Distances)

self.\_\_returnSensors()

t1 = time.time()

dt = t1 - t0

# Report dos tempos para calcular todas as distâncias

"""

self.\_\_printTimes()

print("Tempo total: " + str(round(dt, 5)))

"""

return MinDistances

def \_\_SimplifiedDistances(self, x, y, IDs):

self.\_\_removeSensors(IDs)

distances = []

for sensor in self.SensorList:

P1 = sensor

if self.numba:

# Identificar onde as coordenadas estão sendo definidas como vetores

x1 = float(P1.Xcord)

x2 = x

y1 = float(P1.Ycord)

y2 = y

d = float(self.diameter)

dist = wallDist(x1, x2, y1, y2, d)

else:

dist1 = np.sqrt((P1.Xcord - x) \*\* 2 + (P1.Ycord - y)\*\*2)

# Clone à direita

dist2 = np.sqrt(

(P1.Xcord - x + self.diameter \* m.pi) \*\* 2 + (P1.Ycord - y)\*\*2)

# Clone à esquerda

dist3 = np.sqrt(

(P1.Xcord - x - self.diameter \* m.pi) \*\* 2 + (P1.Ycord - y)\*\*2)

dist = np.min([dist1, dist2, dist3])

distances.append(dist)

self.\_\_returnSensors()

return distances

def returnDeltaT(self, x, y, IDs, mode):

auxMode = self.CalcMode

auxSection = self.SectionMode

if mode == 'geodesic':

self.setCalcMode('geodesic')

self.\_\_AllSensorsAuxCoords()

distances = self.calcAllDist(x, y, IDs)

elif mode == 'reg':

self.setCalcMode('section')

self.setSectionMode('reg')

self.\_\_AllSensorsAuxCoords()

distances = self.calcAllDist(x, y, IDs)

elif mode == 'inc':

self.setCalcMode('section')

self.setSectionMode('inc')

self.\_\_AllSensorsAuxCoords()

distances = self.calcAllDist(x, y, IDs)

elif mode == 'simple':

"Modo simplificado - planificado"

distances = self.\_\_SimplifiedDistances(x, y, IDs)

elif mode == 'original':

"Usar modo atual"

distances = self.calcAllDist(x, y, IDs)

else:

print("Modo inexistente")

if mode != 'original' and mode != 'simple':

self.setCalcMode(auxMode)

self.setSectionMode(auxSection)

self.\_\_AllSensorsAuxCoords()

NPdist = np.array(distances)

times = (NPdist - NPdist[0]) / self.veloc

return times

# Localização

def \_\_InitialKick(self, TimesToSensors):

temp = self.\_\_orderByTime(TimesToSensors)

times = []

Nsensors = 3

for element in temp[:Nsensors]:

(ID, time) = element

times.append(time)

weights = np.array(times / np.max(times))

weights += 1

x = 0

y = 0

j = 0

sum = 0

for element in temp[:Nsensors]:

(ID, time) = element

sensor = self.\_\_getSensorbyID(ID)

x += sensor.Xcord / weights[j]

y += sensor.Ycord / weights[j]

sum += 1 / weights[j]

j += 1

x0 = [x / sum, y / sum]

"""

# Definindo como sendo o sensor mais próximo

(ID, time) = temp[0]

sensor = self.\_\_getSensorbyID(ID)

x0 = [sensor.Xcord, sensor.Ycord]

"""

return (x0)

def simpleLocation(self, TimesToSensors):

# Revisar este método

x0 = self.\_\_InitialKick(TimesToSensors)

data = self.\_\_orderMembers(TimesToSensors)

(firstID, t0) = data[0]

IDs = []

MeasTimes = []

for member in data:

(ID, TOF) = member

IDs.append(ID)

MeasTimes.append(TOF - t0)

MeasTimes = np.array(MeasTimes)

gain = 5

A = np.sqrt(self.height\*\*2 + (self.diameter \* np.pi / 2)\*\* 2) / self.veloc

weights = (MeasTimes - np.min(MeasTimes)) / A

weights = np.exp(-gain \* weights)

def CalcResidue(x):

tcalc = self.returnDeltaT(x[0], x[1], IDs, 'simple')

residue = np.sqrt(np.sum(((tcalc - MeasTimes)/ A)\*\*2))

f = np.log10(residue)

return f

# options={"gtol": 1E-4}

bounds = [(-0.01 \* self.diameter \* m.pi, 1.01 \* self.diameter \* m.pi),

(-1.01 \* self.SemiPerimeter, self.height + 1.01 \* self.SemiPerimeter)]

maxiter = 3000

polish = False

"""

res = opt.minimize(CalcResidue, x0, method='BFGS')

"""

res = opt.differential\_evolution(CalcResidue, bounds=bounds, maxiter=maxiter, polish=polish)

return res.get("x")

def completeLocation(self, TimesToSensors):

# Inicialização dos tempos acumulados

# self.\_\_initializeTimes()

#x0 = self.\_\_InitialKick(TimesToSensors)

data = self.\_\_orderMembers(TimesToSensors)

(firstID, t0) = data[0]

IDs = []

MeasTimes = []

for member in data:

(ID, TOF) = member

IDs.append(ID)

MeasTimes.append(TOF - t0)

MeasTimes = np.array(MeasTimes)

gain = 10

A = np.sqrt(self.height\*\*2 + (self.diameter \* np.pi / 2)

\*\* 2) / self.veloc

weights = (MeasTimes - np.min(MeasTimes)) / A

weights = np.exp(-gain \* weights)

def CalcResidue(x):

tcalc = self.returnDeltaT(x[0], x[1], IDs, 'original')

residue = np.sqrt(np.sum(((tcalc - MeasTimes) \* weights / A)\*\*2))

f = np.log10(residue)

return f

# options={"gtol": 1E-4}

bounds = [(-0.01 \* self.diameter \* m.pi, 1.01 \* self.diameter \* m.pi),

(-1.01 \* self.SemiPerimeter, self.height + 1.01 \* self.SemiPerimeter)]

maxiter = 3000

polish = False

"""

res = opt.minimize(CalcResidue, x0=x0, method='L-BFGS-B', options={"gtol": 1E-4}, bounds=bounds)

"""

res = opt.differential\_evolution(CalcResidue, bounds=bounds, maxiter=maxiter, polish=polish)

# print(res) # - Resultado da otimização

# self.\_\_printTimes()

return res.get("x")