UNIVERSIDADE FEDERAL DO PARANÁ

ENGENHARIA MECÂNICA

JÉSSICA MENEGUEL
LEONARDO SIRINO

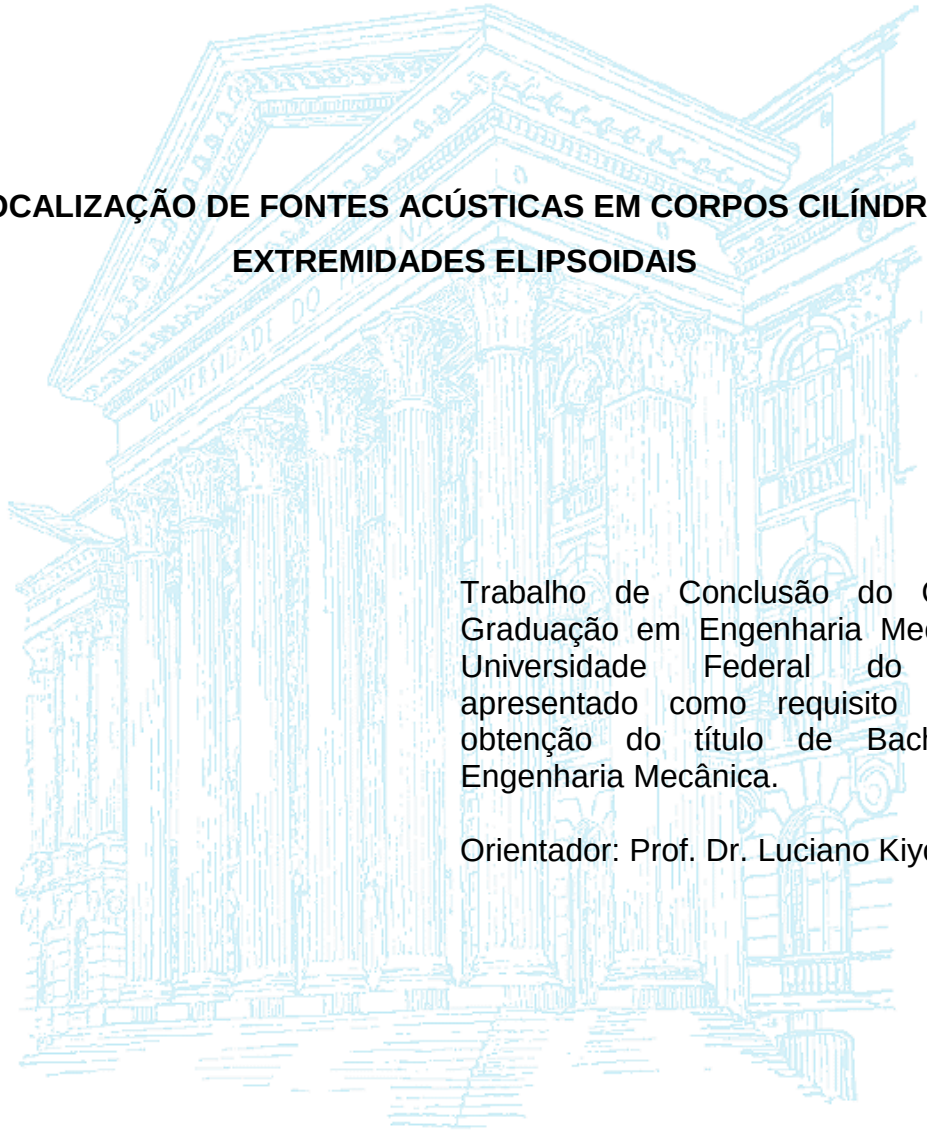
LOCALIZAÇÃO DE FONTES ACÚSTICAS EM CORPOS CILÍNDRICOS DE EXTREMIDADES ELIPSOIDAIS

CURITIBA
2018

JÉSSICA MENEGUEL

LEONARDO SIRINO

LOCALIZAÇÃO DE FONTES ACÚSTICAS EM CORPOS CILÍNDRICOS 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

_____

Orientador: Prof. Dr. Luciano Kiyoshi Araki

Departamento de Engenharia Mecânica - UFPR

_____

Prof. Me. Carlo Giuseppe Filippin

Departamento de Engenharia Mecânica - UFPR

Eng. Leonardo Meneghini Pires

Sistemas Mecânicos - Lactec

Curitiba

2018

AGRADECIMENTOS

Agradecemos ao nosso Professor Luciano Araki pelas orientações durante todo o desenvolvimento do trabalho**.**

Agradecemos ao Lactec pela disponibilização do espaço e dos equipamentos utilizados para a realização dos procedimentos experimentais.

Ao Me. Marco Aurelio Luzio e ao Eng. Nestor Carlos de Moura por compartilhar a enorme experiência que possuem com a técnica de emissão acústica, contribuindo em muito para a definição do tema e elaboração do trabalho

Ao Prof. Me. Carlo Giuseppe Filippin pelas orientações que vieram não somente durante a execução desse trabalho, mas também durante toda a graduação.

Ao Eng. Leonardo Meneghini Pires e ao Eng. Paulo Cristiano Moro pelo suporte durante as atividades experimentais e pelo compartilhamento de sua experiência com emissão acústica.

Ao Eng. Vagner Silvério, ao Samuel Kluthcovsky Cavalli e ao Paolo Romulo Cuevas pelo auxílio durante as atividades experimentais e pela amizade que sempre nos proporcionaram.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

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, utilizando uma aproximação da geodésica para o cálculo de distâncias entre pontos no tampo. A acurácia do método proposto é verificada através de uma série de testes numéricos e empíricos, e os resultados foram comparados aos do software comercial e aos métodos tradicionais de localização.

Palavras-chaves: Emissão acústica, Evolução diferencial, Localização, Vasos de pressão.

1. 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.

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 emissora do sinal.

Entretanto, devido à complexidade geométrica de elementos cilíndricos com tampos elipsoidais, como os vasos de pressão, as técnicas atuais de localização tradicionais empregam modelagens simplificadas dessas estruturas, geralmente planificando-as. 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 superiores à essa.

2. REVISÃO BIBLIOGRÁFICA

2.1. 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.

2.1.1. 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. Joseph 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.

2.1.2. MODOS DE PROPAGAÇÃO E VELOCIDADE DE ONDA

As ondas de EA, como qualquer onda mecânica, necessitam 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.

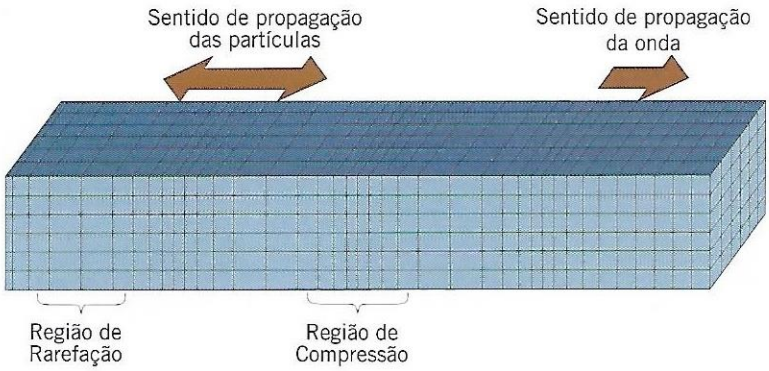


Figura 1 - Onda longitudinal

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.

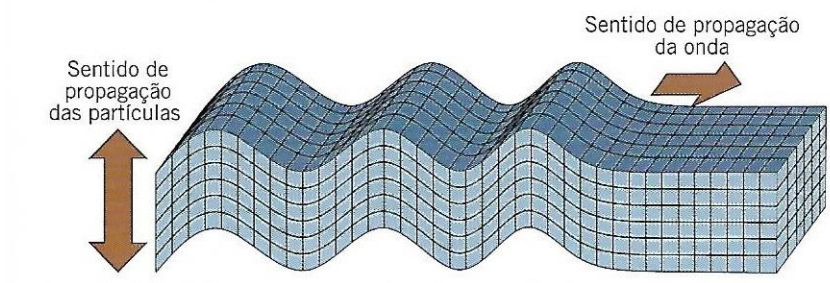


Figura 2 - Onda transversal

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, dentre elas as ondas de Rayleigh e as ondas de Lamb, apresentadas na Figura 3 e na Figura 4.

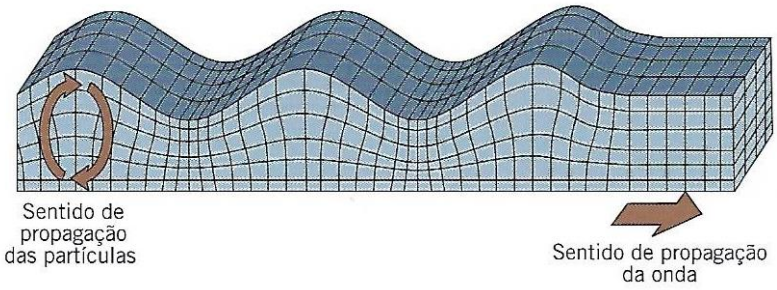


Figura 3 - Onda de Rayleigh

Fonte: Filippin, 2017

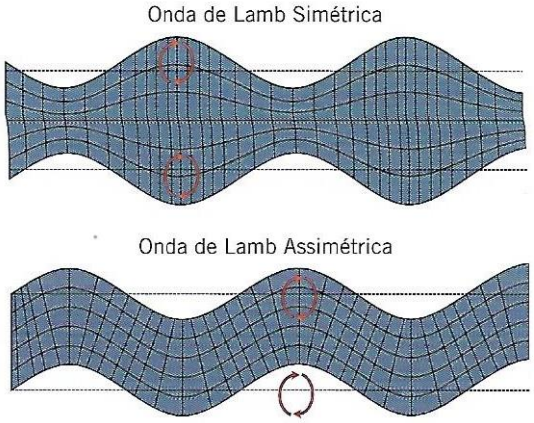


Figura 4 - Onda de Lamb

Fonte: Filippin, 2017

A velocidade de propagação da onda é dependente do material e do tipo de onda observado. Em um teste de EA diferentes tipos ondas podem ser identificados, dependendo do limiar de detecção usado e da geometria da estrutura analisada. Ao iniciar o teste é necessário então medir a velocidade da onda para o limiar que será usado durante todo o teste, a partir da quebra de grafite em uma linha de sensores, como apresentado na Figura 5. A quebra de grafite pode ser usada para produzir um sinal de EA porque gera onda semelhante ao crescimento de uma trinca.

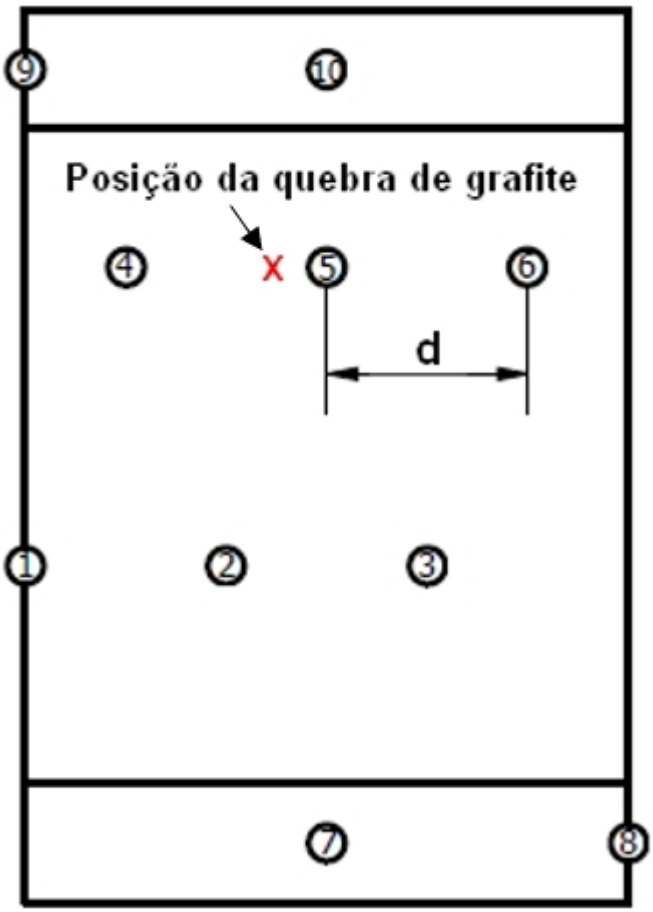


Figura 5 - Medição de velocidade da onda

A velocidade da onde pode ser determinada pela equação (2.1).

Fonte: Os autores

$$V = \frac{d}{\Delta t_{5,6}} \qquad (2.1)$$

2.1.3. 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 6. 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].

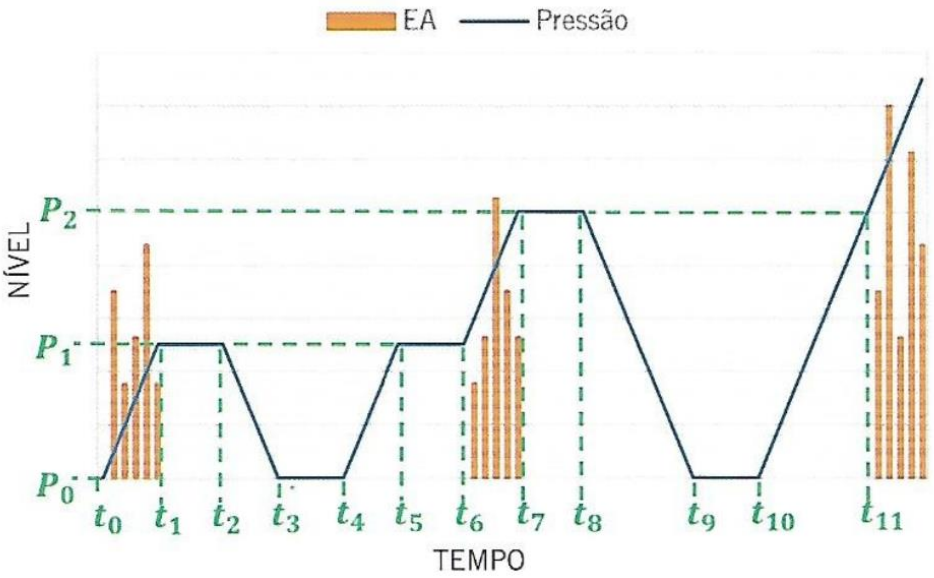


Figura 6 - Efeito Kaiser

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].

2.1.4. 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 7 são apresentados os componentes de um sensor de EA.

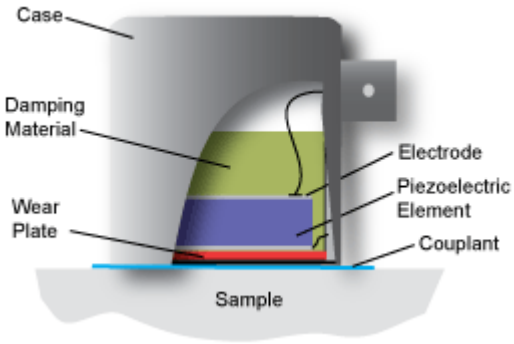


Figura 7 - Elementos de um sensor de EA

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 8.

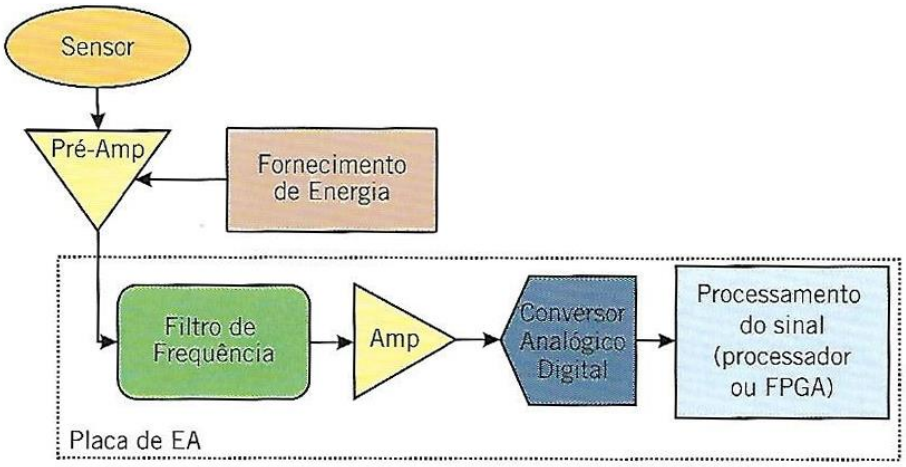


Figura 8 - Fluxograma do sinal de EA

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 fidedigna 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, onde cada placa de expansão de canais possui seu próprio FPGA.

2.1.5. 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, definem-se os *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 9 apresenta como esses parâmetros são observados em um sinal.

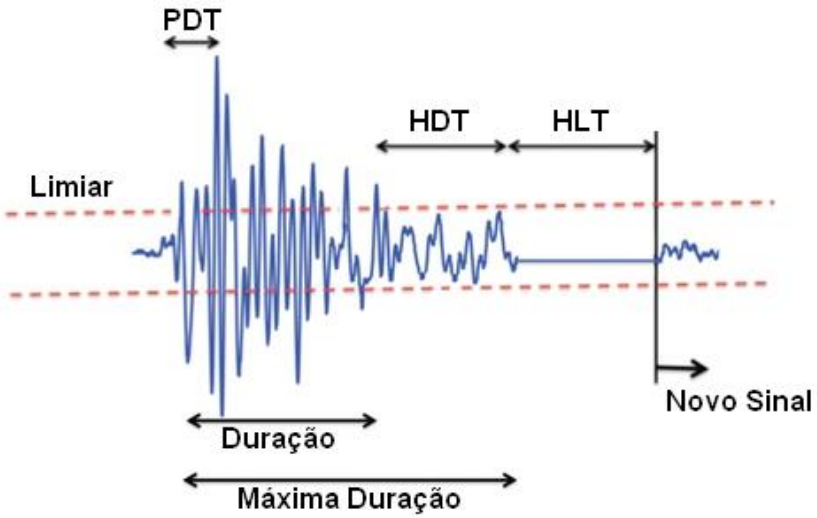


Figura 9 – Tempos de um sinal de EA

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 10 apresenta um exemplo de tela usada para o monitoramento de um ensaio.

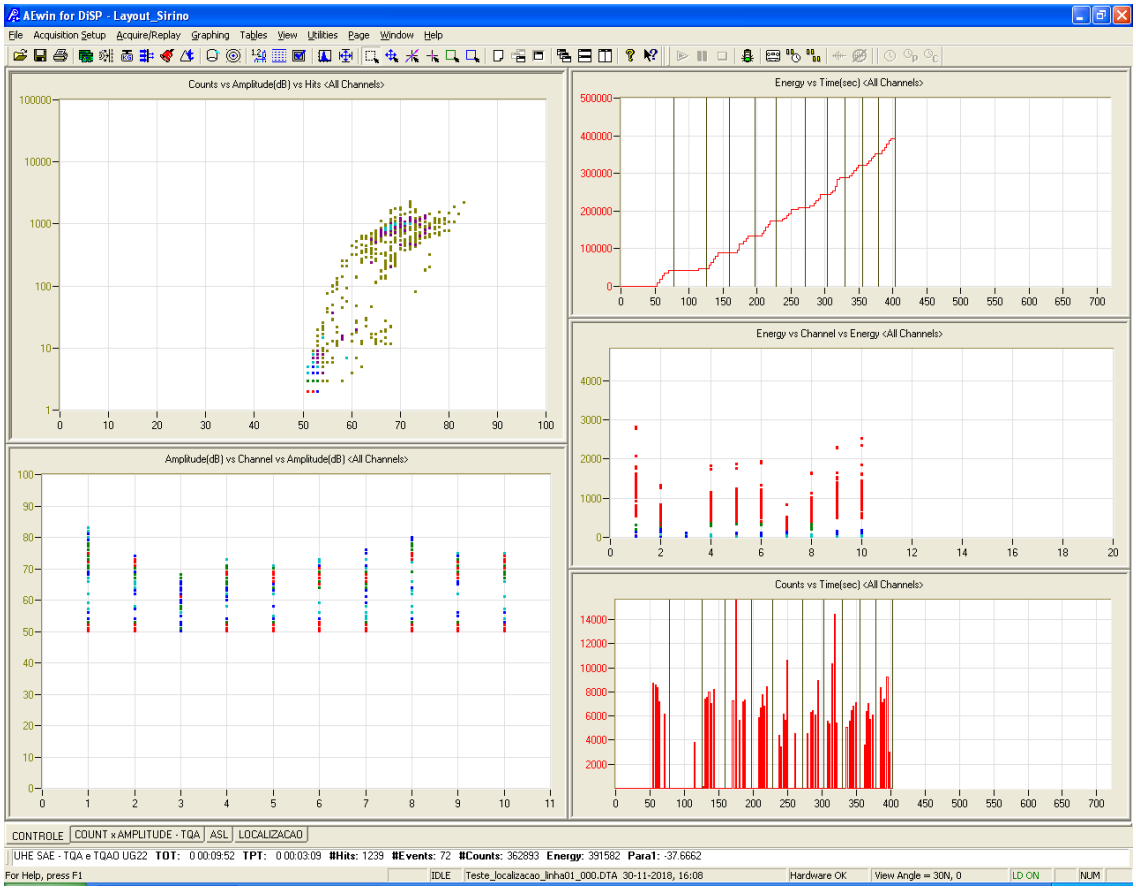


Figura 10 – Exemplo de tela de monitoramento para testes de emissão acústica

Fonte: Os autores

2.2. Vasos de pressão

Vasos de pressão são reservatórios de diferentes tipos, dimensões ou finalidades, projetados para resistir com segurança às pressões internas diferentes da pressão atmosférica ou às pressões externas. 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]. Essa 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 11) e conformadas para confecção do tampo (Figura 12). 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 11 - Processo de calandramento para confecção do corpo cilíndrico

Fonte: KNM Group Brasil, 2012

Figura 12 - 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: gotas frias, vazios, fendilhamento, ondulação, trincas, dobras, inclusões e segregações; da calandragem: espessura irregular e bolhas; da conformação: trincas, ondulações, 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, e antes da volta em funcionamento para vasos inativos por mais de 12 meses ou após o vaso ser instalado em outro local [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 13.

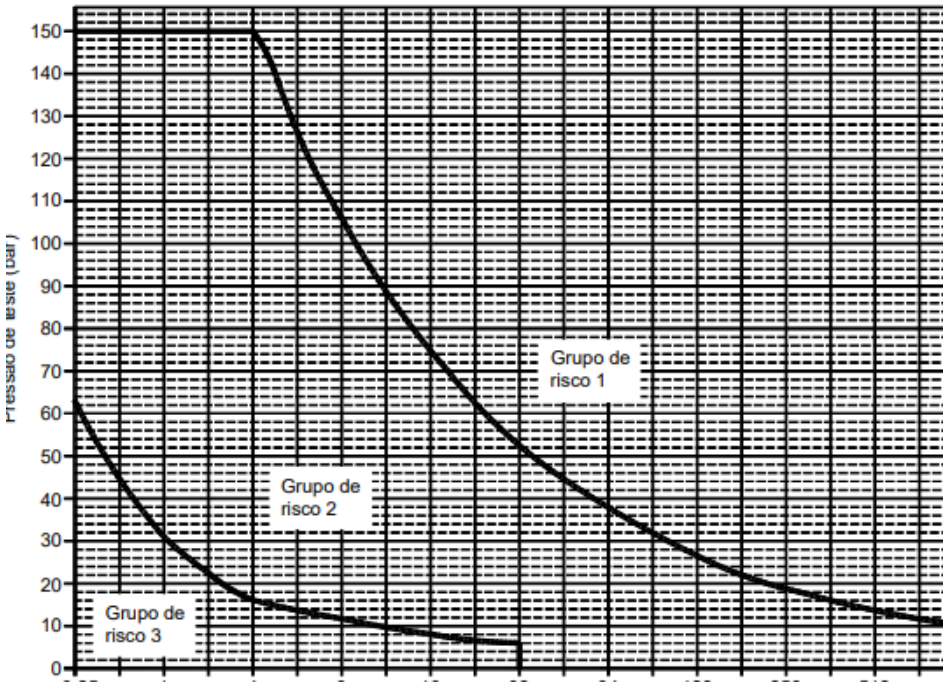


Figura 13 - Gráfico para definição dos grupos de risco para vasos de pressão

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 se 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 14, Figura 15 e Figura 16.

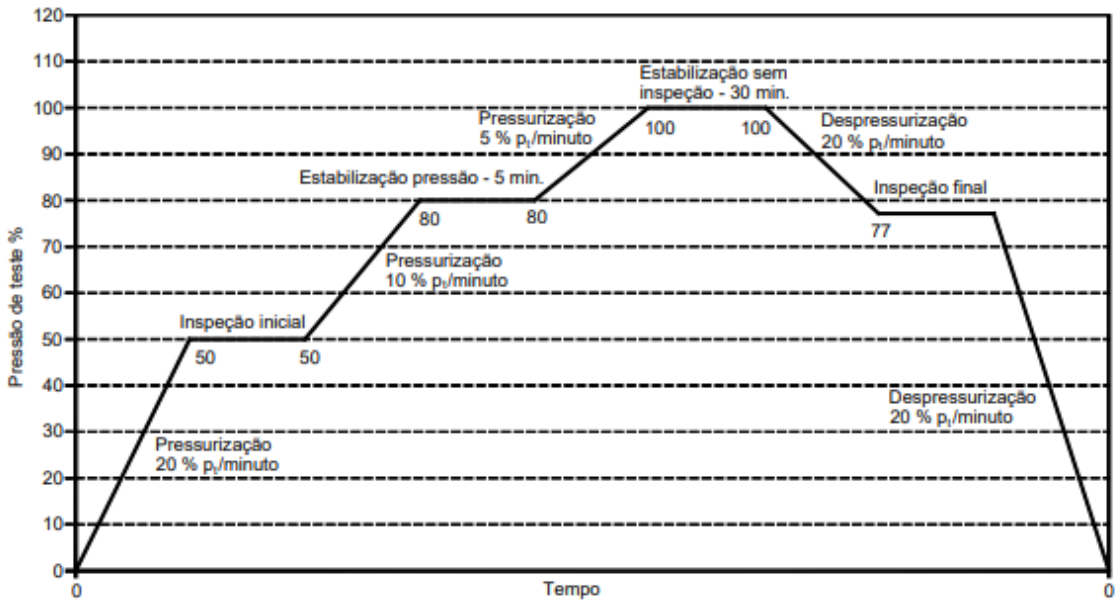


Figura 14 - Gráfico de teste hidrostático do grupo de risco 1

Fonte: N-2688

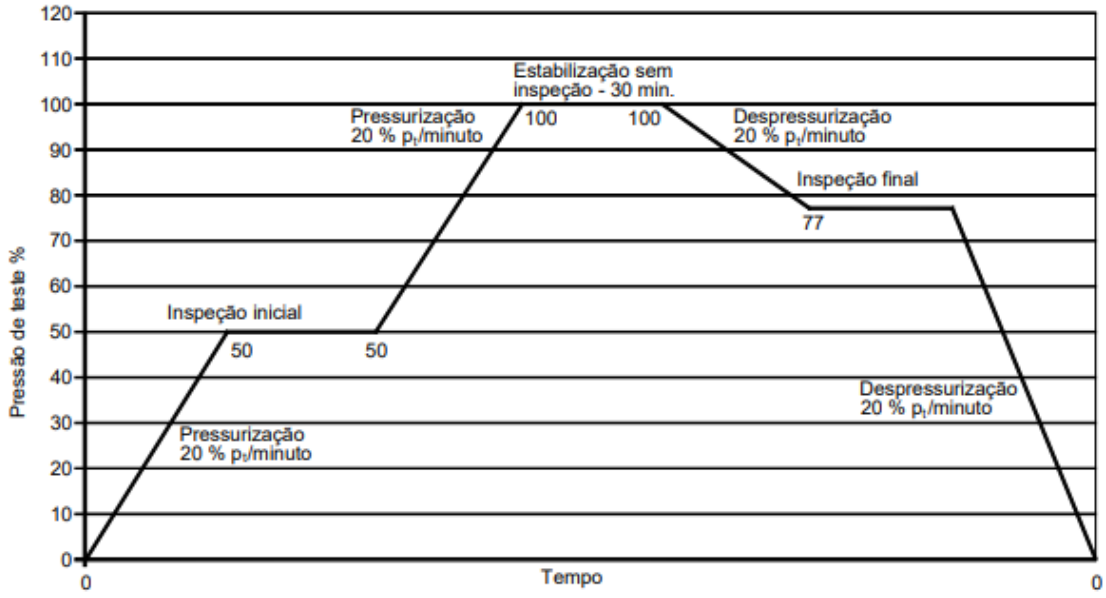


Figura 15 - Gráfico de teste hidrostático do grupo de risco 2

Fonte: N-2688

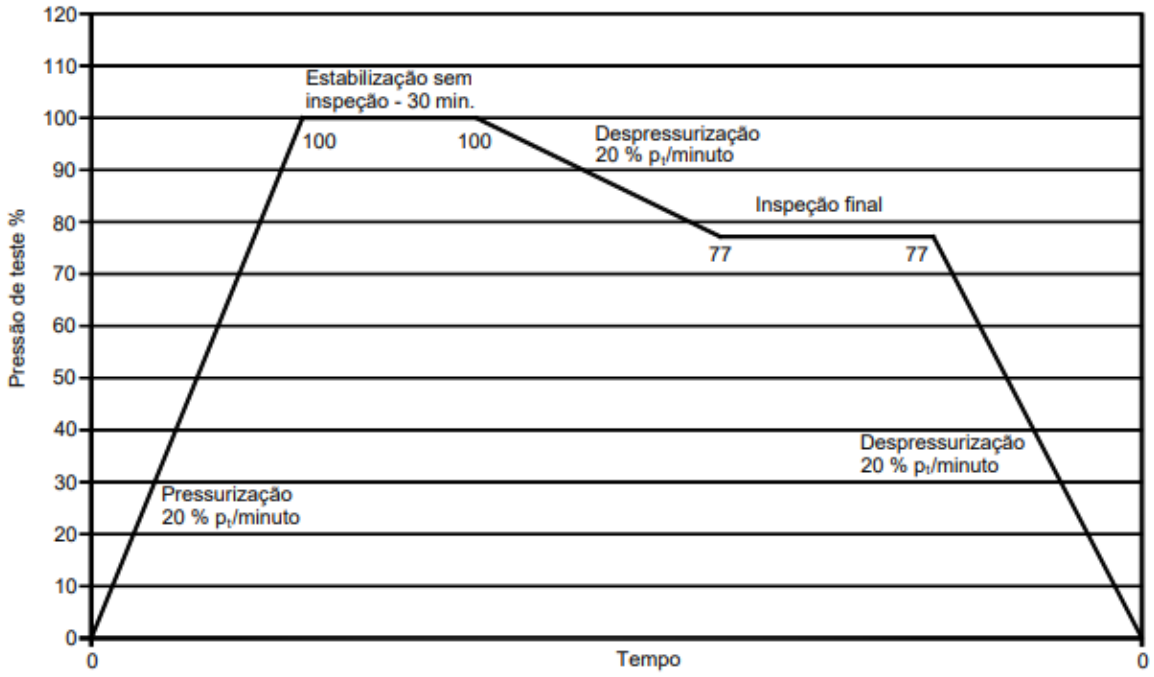


Figura 16 - Gráfico de teste hidrostático do grupo de risco 3

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 junto ao ensaio hidrostático para acompanhamento de defeitos subcríticos ou não visíveis a nível macroscópico. Ou ainda, pode ser aplicada em conjunto com o ensaio pneumático, em substituição ao ensaio hidrostático.

O crescimento de trincas é acompanhado da geração de vários sinais (*Hits*) e vazamentos levam a elevação do RMS do sinal [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].

2.3. 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 nesses sistemas utiliza 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 nas 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 de quaisquer duas hipérboles assim geradas ocorre sobre a fonte de emissão acústica, como é mostrado na Figura 17.

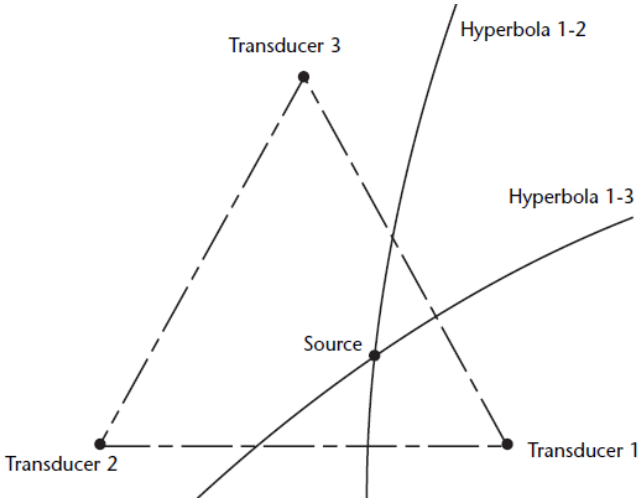


Figura 17 - Localização planar pelo método da hipérbole

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

Dado um sensor 1, um sensor 2 e uma fonte de emissão acústica com coordenadas (X_s, Y_s), tem-se r_1 a distância da fonte e o sensor 1, r_2 a distância entre a fonte e o sensor 2, θ o ângulo formado entre a linha que conecta o sensor 1 ao sensor 2 e a linha que conecta o sensor 1 e a fonte, D a distância entre os dois sensores e Z a distância da fonte à reta que conecta os dois sensores.

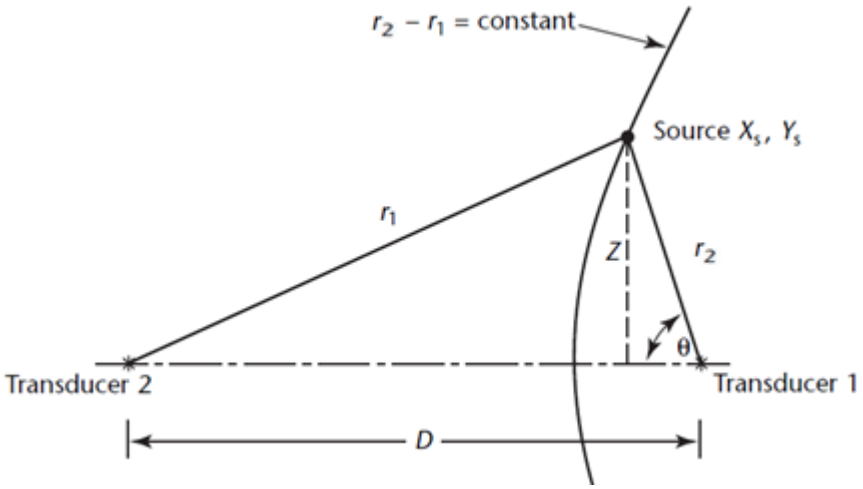


Figura 18 - Localização planar com dois sensores

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

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

$$\Delta t_{1,2} \cdot V = r_2 - r_1 \tag{2.2}$$

$$Z = r_2 \cdot \sin\theta \tag{2.3}$$

$$Z^2 = {r_1}^2 - (D - r_2 \cdot \cos\theta)^2 \tag{2.4}$$

Substituindo a equação (2.3) na (2.4):

$$(r_2 \cdot \sin\theta)^2 = {r_1}^2 - (D - r_2 \cdot \cos\theta)^2 \tag{2.5}$$

Simplificando a expressão (2.5) :

$${r_2}^2 = {r_1}^2 - D^2 + 2 \cdot r_2 \cdot D \cdot \cos\theta \tag{2.6}$$

Isolando r_1 na equação (2.2) e substituindo em (2.8) se chega a:

$$r_2 = \frac{1}{2} \cdot \frac{D^2 - \Delta t_{1,2}^2 \cdot V^2}{\Delta t_{1,2} \cdot V + D \cdot \cos\theta} \tag{2.7}$$

Inserindo um terceiro sensor, não alinhado aos dois anteriores, é possível obter outra equação semelhante a (2.9). 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).

$$\varepsilon = \sum_{i=1}^{n} (T_{Medido} - T_{Calculado})^2 \qquad (2.8)$$

$$\varepsilon = \sum_{i=1}^{n} [(T_0 + \Delta T_{Medido}) - (T_0 + \Delta T_{Calculado})]^2 \qquad (2.9)$$

$$\varepsilon = \sum_{i=1}^{n} (\Delta T_{Medido} - \Delta T_{Calculado})^2 \qquad (2.10)$$

Onde:

$$\Delta T_{Calculado} = V_{Som} \cdot \sqrt{(x_i - x)^2 + (y_i - y)^2} \qquad (2.11)$$

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.

2.4. 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 (s_{12}) e direção, ou azimute no ponto A (α_1), 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 (s_{12}). Esses e outros parâmetros cuja definição auxilia na resolução dos problemas, como longitude (ϕ) e diferença de longitude (λ), são apresentados na Figura 19.

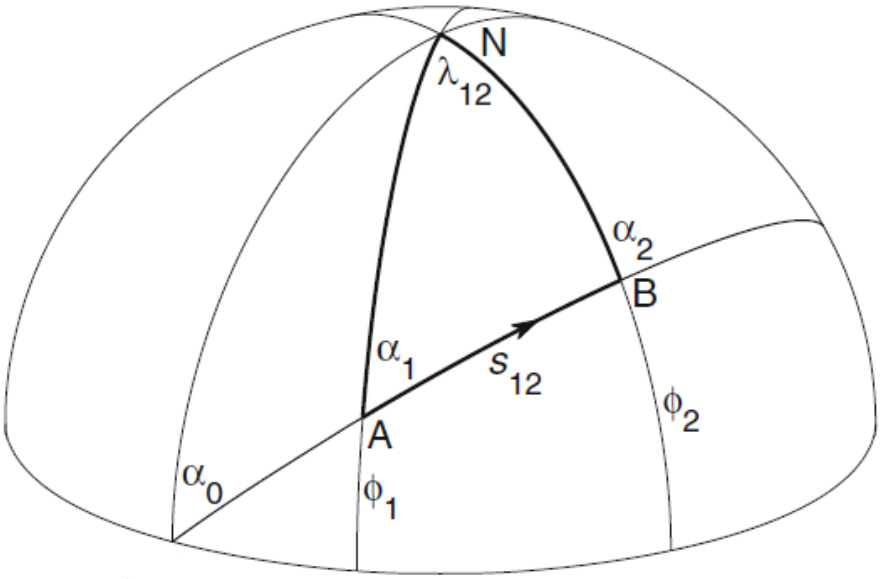


Figura 19 - Grandezas em um elipsoide de revolução

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. No referido trabalho, é definido uma esfera auxiliar para o cálculo da distância e obtém-se como resposta 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.

2.5. 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), 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.

2.5.1.1. Mutação

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

$$p_{novo} = p_{base} + F(p_1 - p_2) \tag{2.12}$$

Onde:

- p_{novo} é o novo PCS
- p_{base} é determinado o ponto base, onde será inserida a mutação
- F é denominado de coeficiente de ponderação da mutação, que determina a influência da mutação na formação de novos PCSs
- $p_1 \, e \, p_2$ 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.

2.5.1.2. Cruzamento

O cruzamento, também denominado de *crossover,* é aplicado na população após a mutação, com o objetivo de se aumentar a diversidade da população [9].

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 $p_i = \{x_1^i, x_2^i, \dots, x_n^i\}$

$$x_j^{r1} = \begin{cases} x_j^{r1} + F\left(x_j^{r2} - x_j^{r3}\right); \ se\ A < TR \\ x_j^{r1}; \ caso\ contrário \end{cases} \quad (2.13)$$

Onde:

- A é um valor gerado aleatoriamente
- TR é a taxa de recombinação para o cruzamento
- F é o fator de ponderação para o cruzamento
- x_j^{r2} e x_j^{r3} são as j-ésimas componentes de dois pontos escolhidos aleatoriamente

2.5.1.3. 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.

2.5.1.4. 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.

3. 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 tamanha 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.

3.1.1. 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 20 há uma representação dessas coordenadas em um modelo de vaso de pressão de tampo elipsoidal para um certo ponto P0.

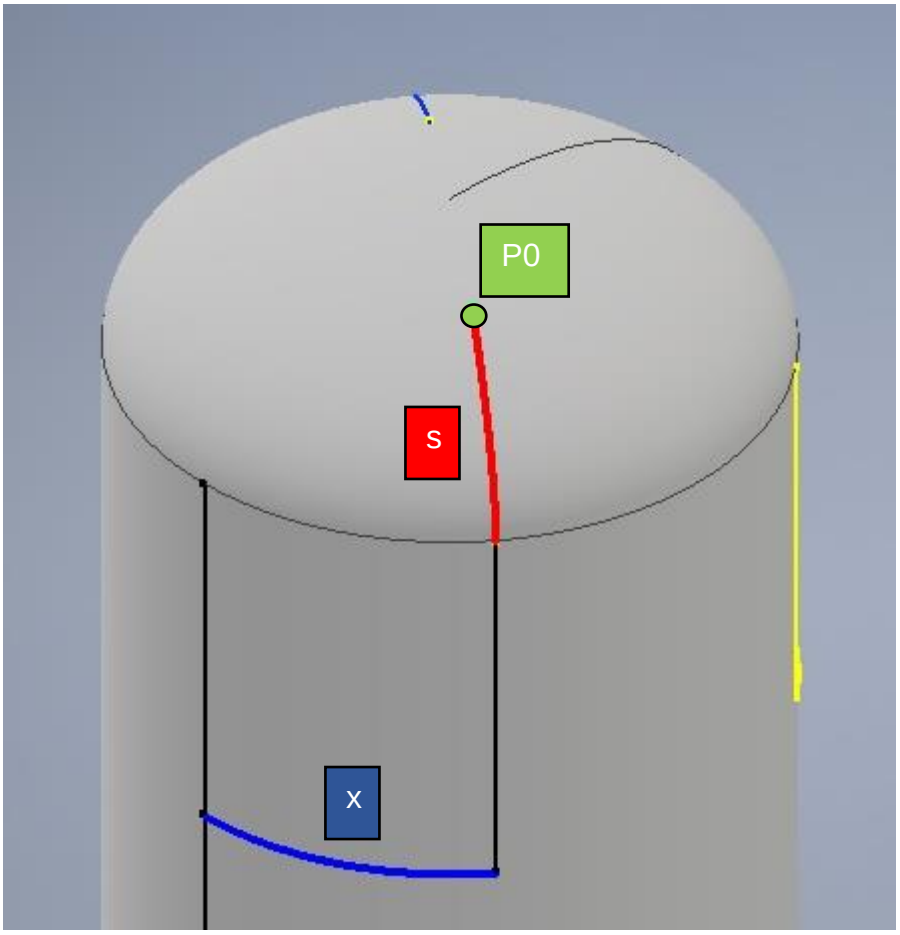


Figura 20 - Identificação das coordenadas x e s

Fonte: Os autores

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 21.

Essas coordenadas serão posteriormente utilizadas para se determinar o plano que contém a elipse que une os pontos.

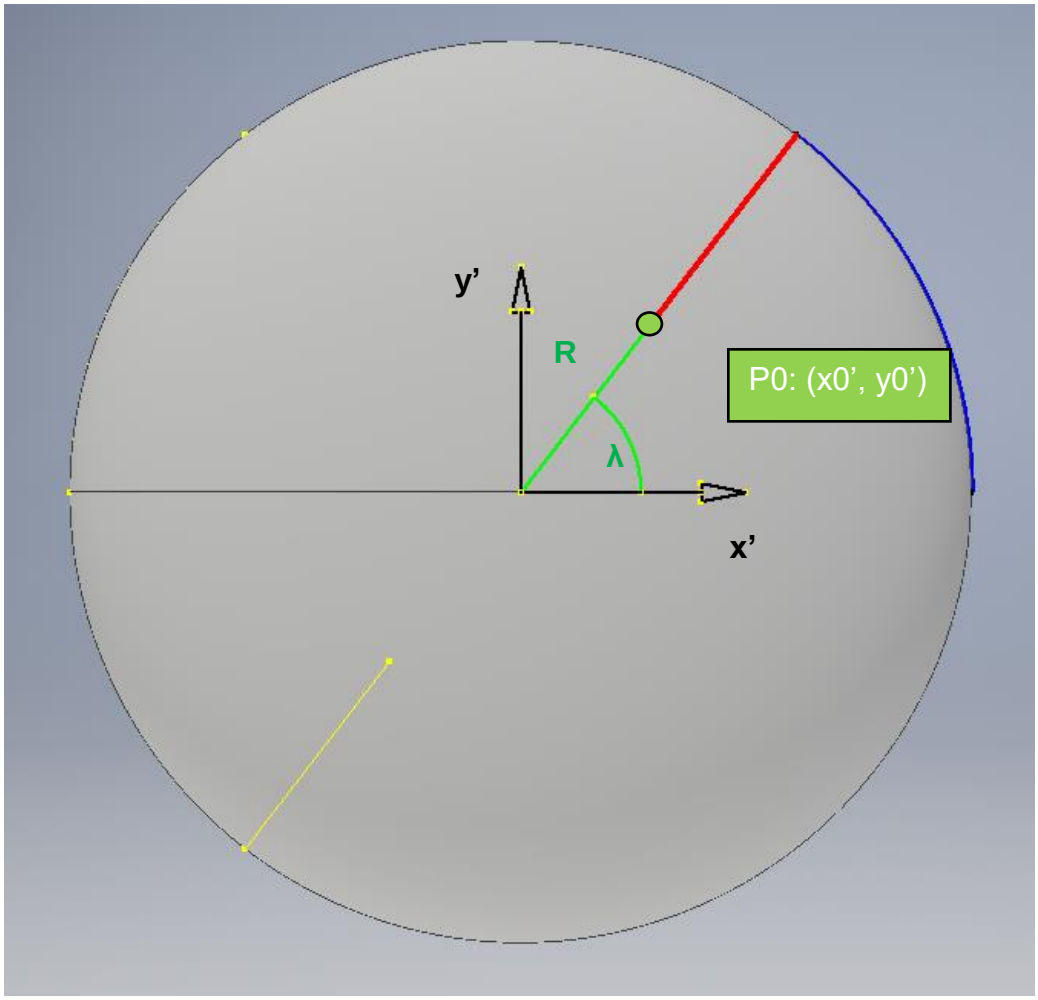


Figura 21 - Vista superior do tampo - coordenadas x0' e y0'

Fonte: Os autores

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)

$$\lambda = \frac{x}{\pi D} \qquad (3.1)$$

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 22 é apresentada a definição de uma secante da elipse.

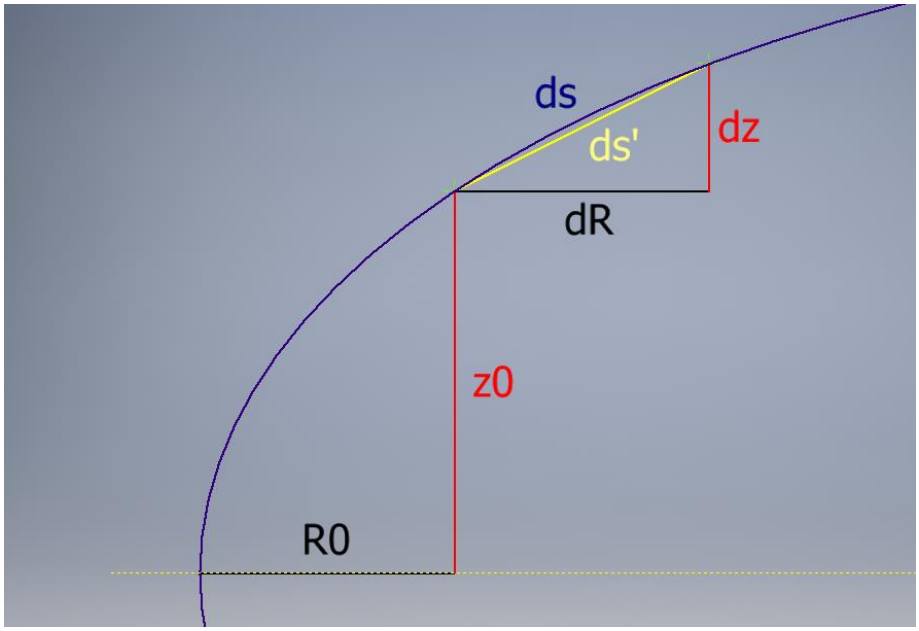


Figura 22 - Secante da elipse

Fonte: Os autores

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.

$$ds' = \sqrt{dR^2 + dz^2} \tag{3.2}$$

$$dz = af\sqrt{1 - \frac{R^2}{a^2}} - z_0 \tag{3.3}$$

Onde:

- a: raio do cilindro
- f: 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:

$$x_0' = R \cdot \cos(\lambda) \tag{3.4}$$

$$y_0' = R \cdot \text{sen}(\lambda) \tag{3.5}$$

$$z_0' = z \tag{3.6}$$

Onde os parâmetros x_0', y_0', R, λ são definidos na Figura 21 e z_0' é a altura do ponto relativo a base do tampo.

3.1.2. 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 usada 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 23.

$$d = \frac{x_2' \cdot y_1' - y_2' \cdot x_1'}{\sqrt{(y_2' - y_1')^2 - (x_2' - x_1')^2}} \tag{3.7}$$

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.

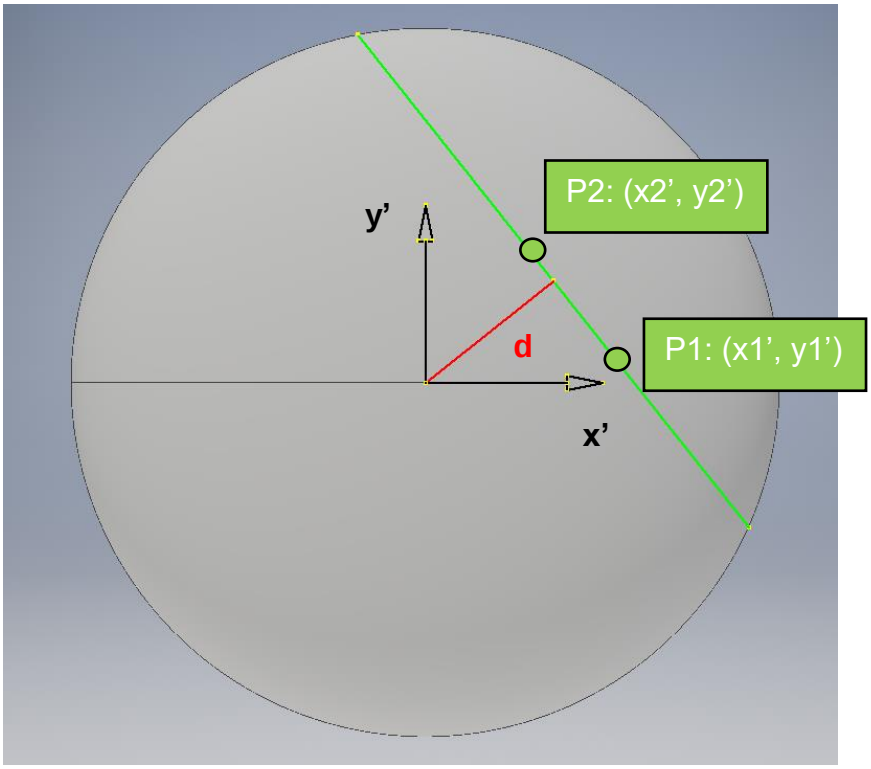


Figura 23 - Vista superior do plano de seccionamento

Fonte: Os autores

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:
$$x = d \tag{3.8}$$

Elipsoide:
$$\frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} = 1 \tag{3.9}$$

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

$$\frac{y^2}{a^2 \cdot \left(\frac{a^2 - d^2}{a^2}\right)} + \frac{z^2}{b^2 \cdot \left(\frac{a^2 - d^2}{a^2}\right)} = 1 \tag{3.10}$$

Portanto, os semieixos da elipse serão:

$$a' = a \cdot \sqrt{\left(\frac{a^2 - d^2}{a^2}\right)} \qquad (3.11)$$

$$b' = b \cdot \sqrt{\left(\frac{a^2 - d^2}{a^2}\right)} \qquad (3.12)$$

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

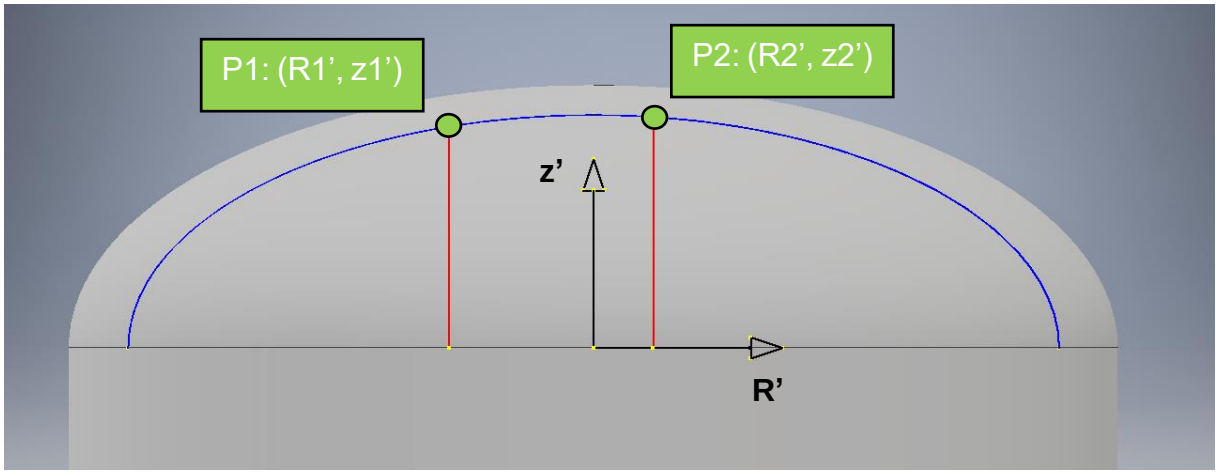


Figura 24 - Elipse auxiliar

Fonte: Os autores

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.

3.1.3. 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 25.

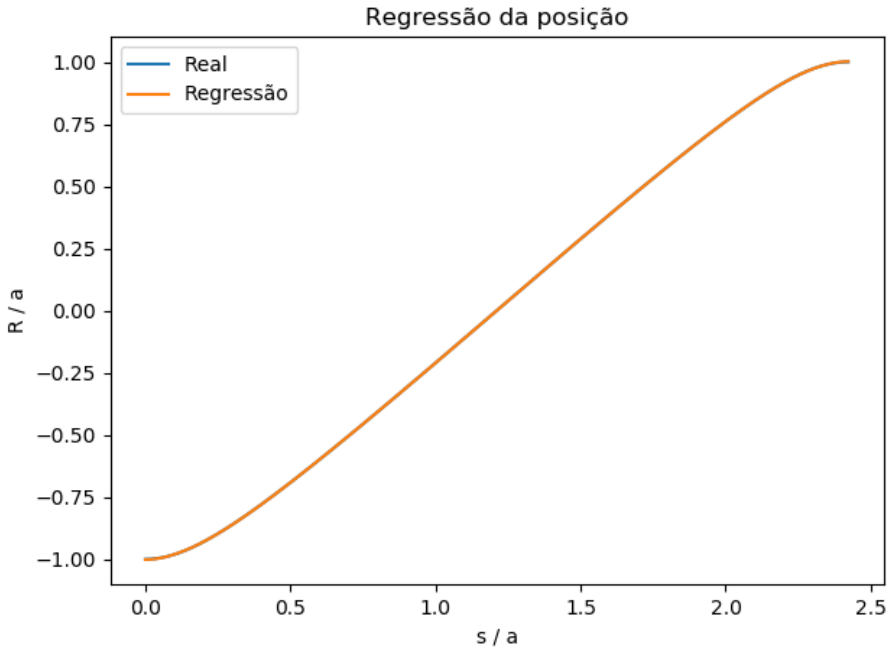


Figura 25 - Regressão Posição x Comprimento do arco

Fonte: Os autores

A função obtida foi:

$$\frac{R}{a} = -0.045 + 0.383 \cdot \frac{s}{a} - 1.367 \cdot \left(\frac{s}{a}\right)^2 + 2.662 \cdot \left(\frac{s}{a}\right)^3 + -3.102 \cdot \left(\frac{s}{a}\right)^4 + 2.247 \cdot \left(\frac{s}{a}\right)^5 + 0.012 \cdot \left(\frac{s}{a}\right)^6 - 1.001 \cdot \left(\frac{s}{a}\right)^7 \quad (3.13)$$

A regressão do arco pode ser observada na Figura 26.

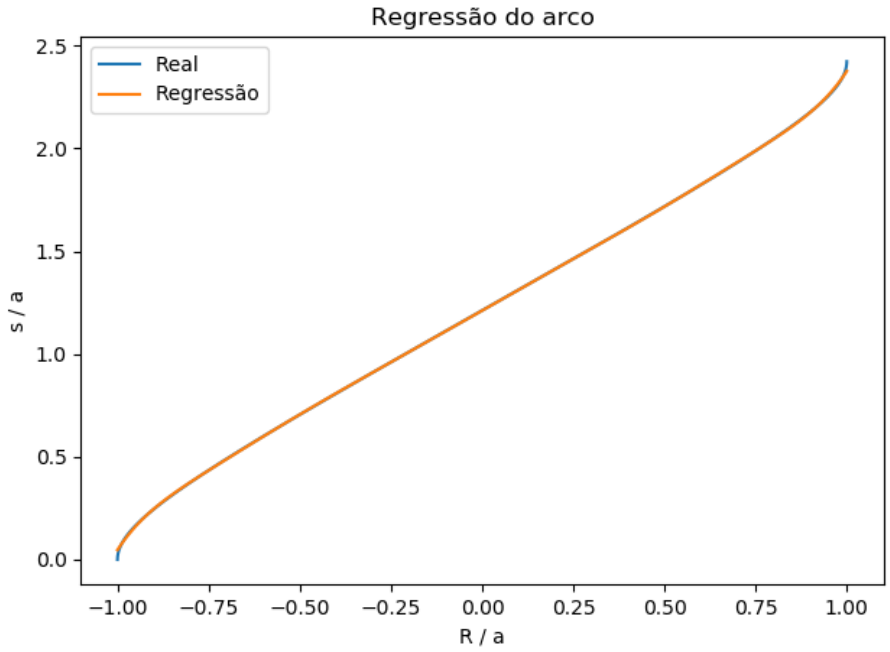


Figura 26 - Regressão Comprimento do arco x Posição

Fonte: Os autores

A função obtida foi:

$$\frac{s}{a} = 9.944 \cdot 10^{-1} - 1.672 \cdot \left(\frac{R}{a}\right)^{2} + 9.922 \cdot 10^{-1} \cdot \left(\frac{R}{a}\right)^{5} + -1.607 \cdot \left(\frac{R}{a}\right)^{7} \cdot 10^{-1} + 1.011 \cdot \left(\frac{R}{a}\right)^{9} + 1.211 \cdot \left(\frac{R}{a}\right)^{10} \quad (3.14)$$

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

3.1.4. 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 27 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.

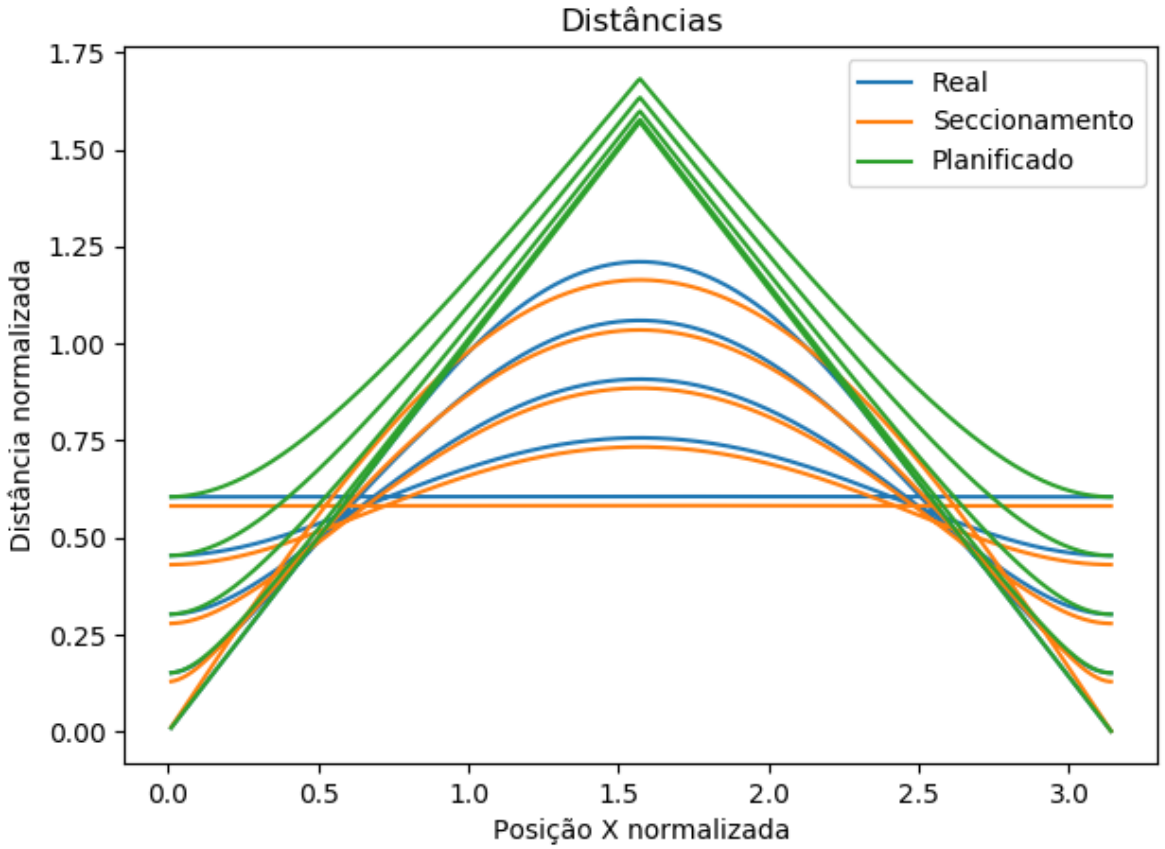


Figura 27 - Distâncias entre pontos no tampo para diferentes métodos de cálculo

Fonte: Os autores

Na Figura 28 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.

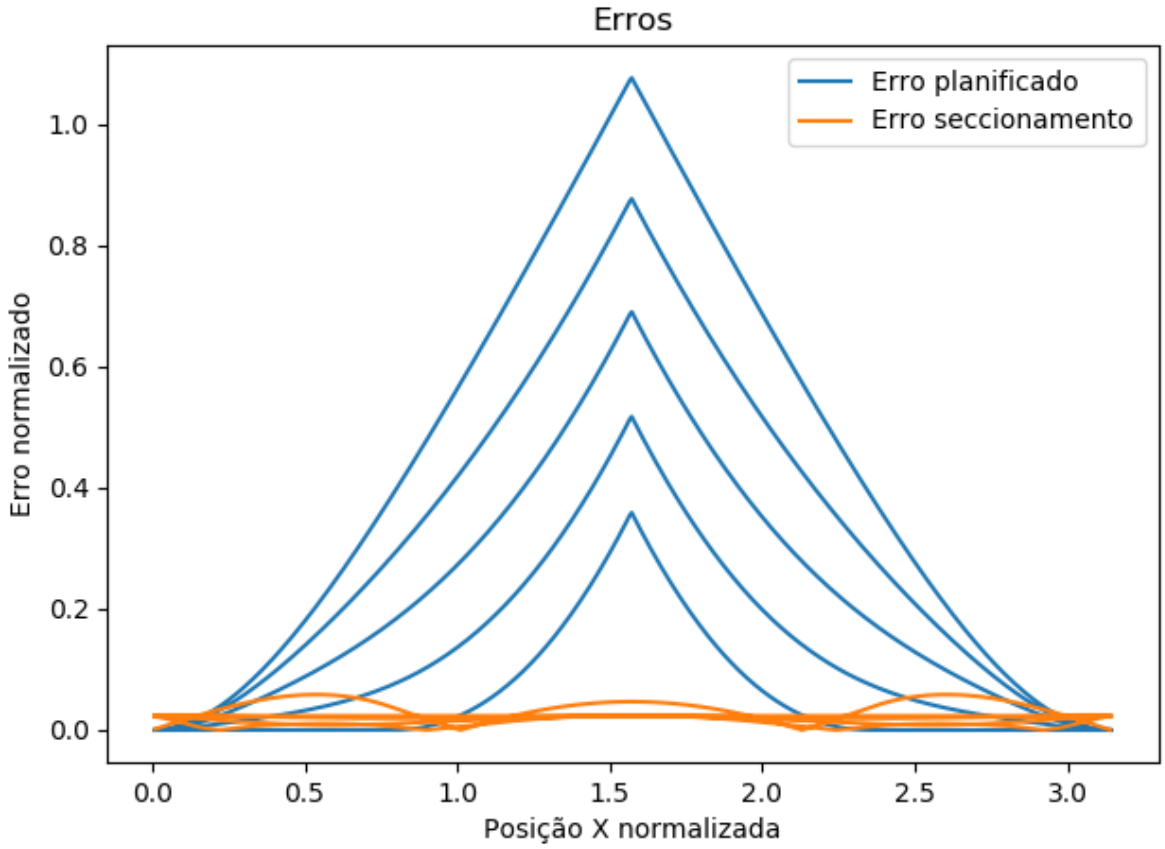


Figura 28 – Erro no cálculo da distância entre pontos no tampo para diferentes métodos

Fonte: Os autores

Na Figura 29 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.

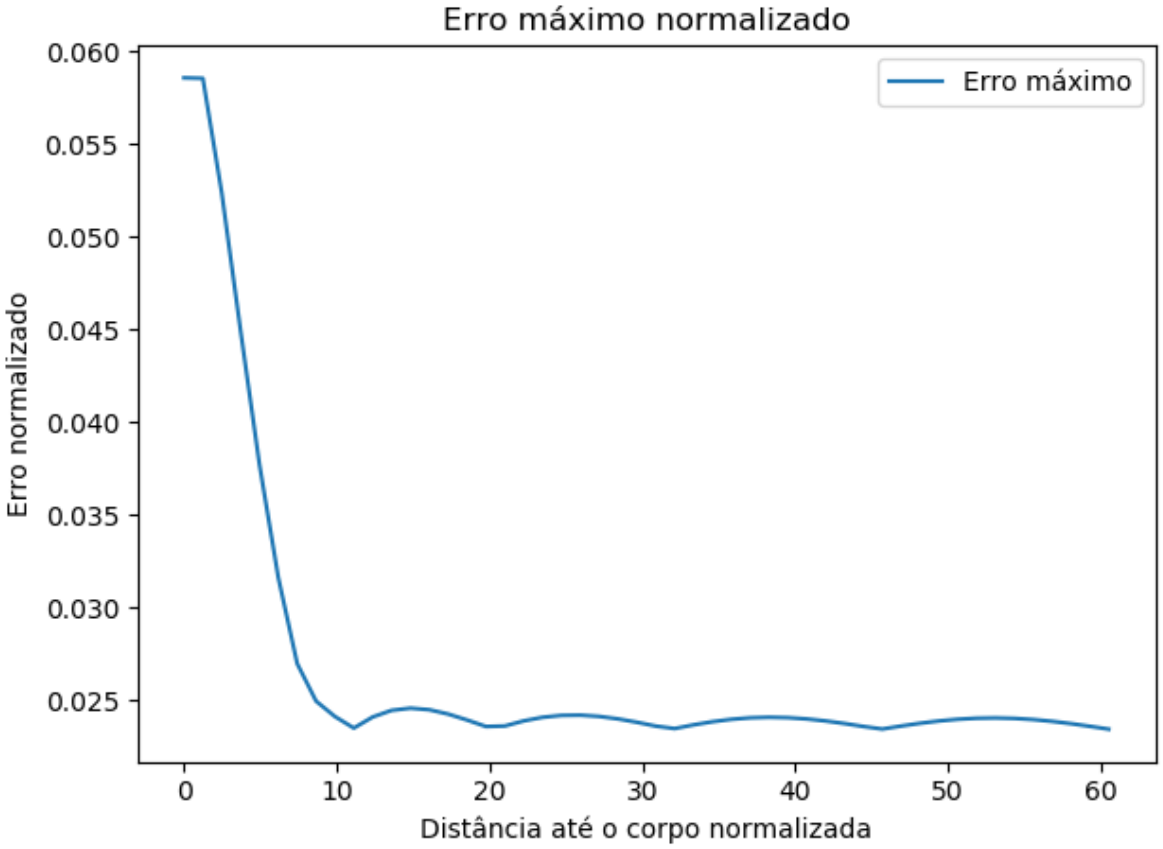


Figura 29 - Erro máximo do método de seccionamento

Fonte: Os autores

4. 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:

$$\boldsymbol{t} = \begin{Bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{Bmatrix} \tag{4.1}$$

Onde t_i é o tempo de chegada ao sensor i.

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

$$t_0 = min(\boldsymbol{t}) \tag{4.2}$$

$$\boldsymbol{dt} = \begin{Bmatrix} t_1 - t_0 \\ t_2 - t_0 \\ \vdots \\ t_n - t_0 \end{Bmatrix} \tag{4.3}$$

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

$$\boldsymbol{l} = \begin{Bmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{Bmatrix} \tag{4.4}$$

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 na 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.

$$\boldsymbol{t}^* = \begin{Bmatrix} l_1/v \\ l_2/v \\ \vdots \\ l_n/v \end{Bmatrix} = \begin{Bmatrix} t_1^* \\ t_2^* \\ \vdots \\ t_n^* \end{Bmatrix} \tag{4.5}$$

Onde v é velocidade do som na estrutura.

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 t_0 foi definido como o menor valor de tempo de chegada, ou seja, o tempo de chegada ao sensor mais próximo da fonte. O t_0^* 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:

$$\boldsymbol{dt}^* = \begin{Bmatrix} t_1^* - t_0^* \\ t_2^* - t_0^* \\ \vdots \\ t_n^* - t_0^* \end{Bmatrix} \tag{4.6}$$

O objetivo do algoritmo de minimização é determinar as coordenadas $x_p,\ y_p$ da fonte virtual que torne $\boldsymbol{dt}^*$ o mais próximo possível de $\boldsymbol{dt}$.

Para se determinar a proximidade de dois vetores existem diversas abordagens. No presente trabalho se optou por usar uma soma quadrática ponderada. Portanto, a proximidade dos vetores $\boldsymbol{dt}$ e $\boldsymbol{dt}^*$, que também será a função custo do algoritmo de minimização, foi definida como:

$$f = \sqrt{\frac{(\boldsymbol{dt}^* - \boldsymbol{dt})^2 \cdot \boldsymbol{w}}{A}} \tag{4.7}$$

Sendo:

$$\boldsymbol{w} = e^{\left(-b \cdot \frac{\boldsymbol{dt}}{A}\right)} \tag{4.8}$$

$$A = \frac{\sqrt{\left(\frac{\pi D}{2}\right)^2 + h^2}}{v} \tag{4.9}$$

Onde:

- $\boldsymbol{w}$: vetor de ponderação
- A: coeficiente de normalização
- b: ganho
- D: diâmetro do corpo cilíndrico
- h: altura do corpo cilíndrico

O vetor de ponderação tem por objetivo 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 possível entre dois pontos no corpo cilíndrico.

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

5. PROCEDIMENTO EXPERIMENTAL

5.1. 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 um vaso de pressão, estes pontos serão as coordenadas das fontes simuladas, as quais o método deve localizar a partir dos tempos de chegada nos sensores.

As dimensões do vaso foram as mesmas do vaso utilizado na análise empírica. Foram definidos 10 sensores em um padrão triangular, 6 deles no corpo cilíndrico e 2 em cada tampo elipsoidal. Conforme apresentado na Figura 30.

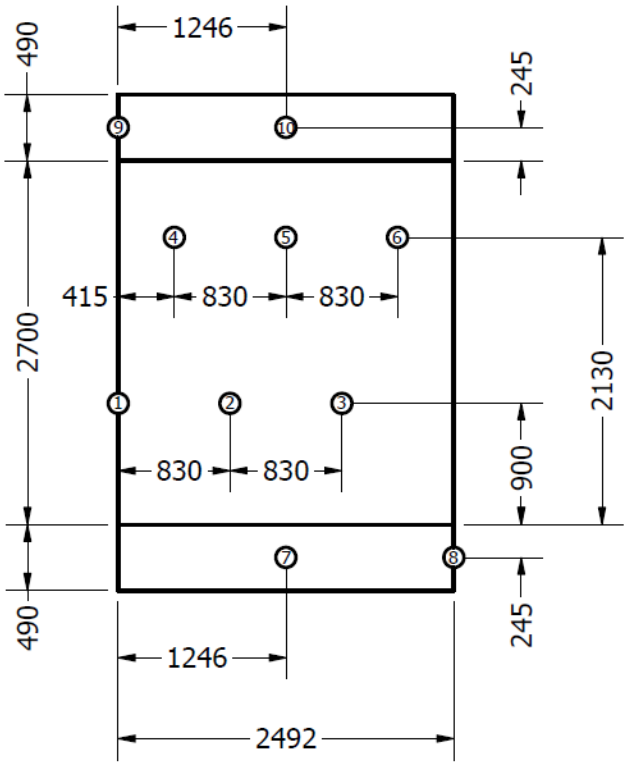


Figura 30 - Layout dos sensores no vaso de pressão

Fonte: Os autores

A velocidade de propagação da onda foi definida em 3,2 km/s, que é a velocidade da onda transversal em aço. Para cada fonte simulada, foram obtidas as distâncias para cada sensor com a biblioteca *Geographiclib*, dividindo essas distâncias pela velocidade, se chega aos tempos de chegada em cada sensor.

Esta informação dos tempos de chegada é a mesma informação é obtida durante a aplicação real da técnica. Esses valores foram então fornecidos ao algoritmo proposto, obtendo como retorno deste a posição estimada da fonte.

No segundo teste, admitiu-se a região de intersecção entre tampo e vaso como zona de interesse. Foram distribuídos 25 pontos nessa, e adotou-se a mesma posição dos 10 sensores anteriores. Definindo-se novamente a velocidade de propagação de onda igual a 3,2 km/s, obtendo-se as diferenças de tempo e resolvendo o problema de localização com os dois métodos: o de seccionamento proposto e método simplificado, onde o vaso é planificado.

No terceiro teste a zona de interesse foi uma linha vertical no centro do vaso, tendo seu início próximo ao vértice do tampo inferior e seu fim no meio do corpo cilíndrico. Os demais parâmetros foram mantidos para a execução deste terceiro teste.

5.2. Análise empírica

5.2.1. MATERIAIS

Para a realização da análise empírica, foi utilizado um vaso de pressão nas dependências do prédio LEME do instituto de pesquisa Lactec (Figura 31). Este vaso de pressão possui a função de acumulador de ar comprimido para um sistema de tratamento de superfície.

Durante a realização dos ensaios, certificou-se que o vaso não entraria em operação para assim garantir a segurança dos envolvidos e evitar interferências de ruídos inerentes à sua operação.

O layout de instalação dos sensores e as dimensões do vaso são apresentados na Figura 30. A distribuição dos sensores no corpo cilíndrico seguiu um padrão triangular, que é o recomendado para monitoramento de corpos cilíndricos [10]. Se optou por instalar 2 sensores em cada tampo, para desta forma obter localizações mais precisas das fontes próxima à interface corpo-tampo e no próprio tampo.

Figura 31 - Vaso de pressão utilizado na análise empírica

Fonte: Os autores

Todos os sensores utilizados foram do tipo R15I AST (Figura 32), do fabricante *Physical Acoustics*. Este sensor possui frequência de ressonância de 150 kHz, pré-amplificador integrado e funcionalidade de auto teste. Este sensor é mais comum para aplicação geral da técnica de EA [11].

Figura 32 - Sensor R15I AST

Fonte: https://www.physicalacoustics.com/by-product/sensors/R15I-AST-150-kHz-Integral-Preamp-AE-Sensor

Para aquisição dos sinais foi usada o sistema DiSP, também do fabricante *Physical Acoustics* (Figura 33). Cada placa padrão DiSP possui 4 canais para aquisição de sensores de EA. O sistema utilizado nos testes possuía 6 placas deste padrão, conferindo-lhe 24 canais de EA ao todo.

Figura 33 - Sistema DiSP

Fonte: Os autores

Os sensores foram fixados ao vaso de pressão com auxílio de porta sensor magnético desenvolvido pelo Lactec. Para garantir a integridade do sinal de EA, foi utilizada uma graxa acoplante entre o sensor e o vaso de pressão (Figura 34)

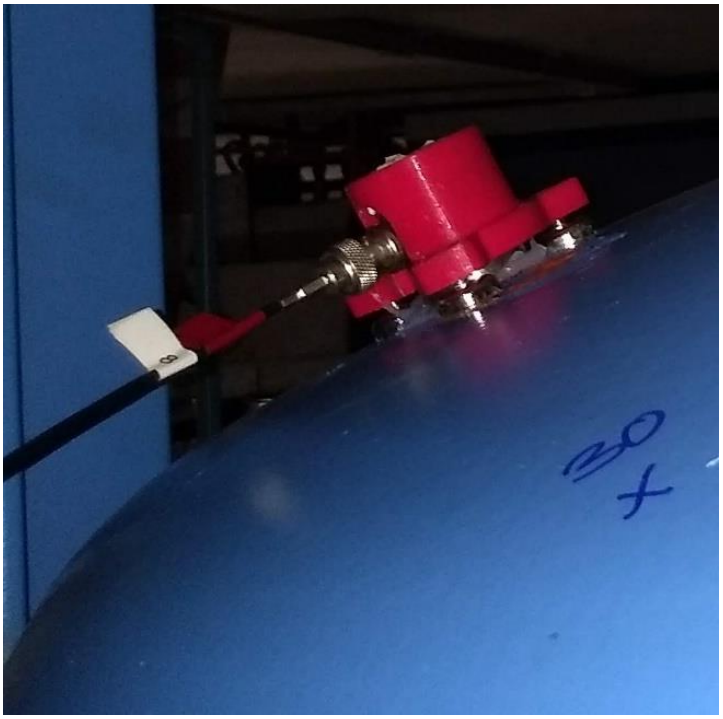


Figura 34 - Porta sensor e acoplante

Fonte: Os autores

5.2.1. VERIFICAÇÃO

O acoplamento foi verificado de acordo com um procedimento interno usado pelo Lactec. Este procedimento determina que devem ser quebradas 3 minas de grafite nas proximidades de cada sensor, sendo que todos os sinais gerados por estas quebras devem possuir amplitude maior que 90 dB, e para um mesmo sensor a variação máxima de amplitude permitida entre os sinais é de 3 dB.

Os resultados da verificação realizada no vaso de pressão são apresentados na Tabela 1.

Tabela 1 - Tabela de verificação do acoplamento dos sensores

Identificação	*1ª Medição*	*2 ª Medição*	*3 ª Medição*
Sensor 1	99	97	98
Sensor 2	94	94	93
Sensor 3	96	98	97
Sensor 7	92	90	90
Sensor 8	95	95	96

Por questões de segurança, os canais 4, 5, 6, 9 e 10 não foram verificados segundo o procedimento descrito acima, apenas se observou as amplitudes registradas nestes canais durante a verificação dos demais, como as amplitudes nesta situação sempre foram superiores a 80 dB, isto indica um bom acoplamento nestes canais também.

Além disso, os testes empíricos tiveram enfoque na porção inferior do vaso de pressão, portanto as informações mais importantes são as provenientes dos sensores 1, 2, 3, 7 e 8.

5.2.2. PROCEDIMENTO DE ENSAIO

Na primeira etapa da análise empírica, os sinais foram gerados através da funcionalidade de auto teste do sistema. Neste modo, o sensor atua como uma fonte, gerando ondas acústicas no material que serão recebidas pelos demais sensores.

O sistema de EA possui uma rotina de teste onde os canais são “pulsados” individualmente, sendo registrados os sinais captados pelos demais sensores. Para os testes do presente trabalho, foram gerados 10 pulsos em cada canal, cada pulso com duração de 20 µs e intervalo entre pulsos de 1000 ms.

O resultado desta rotina de auto teste são várias tabelas de resultados, cada tabela está associada à um canal atuando como fonte. Cada tabela apresenta o valor

médio dos 10 pulsos para uma série de parâmetros, dentre eles o tempo de chegada ao sensor, sendo este o valor utilizado pelo algoritmo para a localização da fonte.

```
                    ************************************
                    ****      Auto-Sensor Test      ****
                    ****  Wed Nov 07 13:48:50 2018  ****
                    ************************************


PROGRAM           AEwin for DiSP
THIS FILE         C:\Documents and Settings\USER\Desktop\TCC Sirino\teste.AST


PULSES OUTPUT     10
PULSE WIDTH       20 usec
PULSE INTERVAL    1000 msec

PULSER
 1
REC  #REC   DeltaT  AMP ENERGY  DURATION  COUNTS
  1    10       0   100   2267     19689    1259
  2    10     228    64    138      6715     230
  3    10     183    75    755     19558    1243
  4    10     268    66    866     21681    1421
  5    10     350    66    851     19924    1384
  6    10     440    62    337     13534     633
  7    10     316    75    759     18979    1174
  8    10     361    71    445     14782     739
  9    10     533    66    766     20907    1241
 10    10     824    66      6        37       3
```

Figura 35 - Exemplo de resultado de auto teste

Fonte: Os autores

A segunda parte da análise empírica consistiu na quebra de minas de grafite em diferentes pontos do vaso. Os pontos de geração de sinal foram separados em três linhas, numa mesma linha a coordenada y se mantinha constante e os pontos eram espaçados 250 mm na coordenada x, proporcionando 10 pontos por linha.

Para a linha 1, a coordenada y valia 0 mm e estavam contidos os pontos de 1 a 10; para a linha 2, a coordenada y valia 150 mm e estavam contidos os pontos de 11 a 20; para a linha 3, a coordenada y valia -150 mm e estavam contidos os pontos de 21 a 30.

Em cada ponto foram gerados no mínimo 5 eventos, ou seja, 5 quebras de minas de grafite que atingissem um número suficiente de sensores para se realizar a localização.

Na Figura 36 são apresentados alguns dos pontos de geração de sinal.

Figura 36 - Pontos de quebra de minas de grafite

Fonte: Os autores

6. RESULTADOS

6.1. Análise numérica

A primeira análise numérica resultou na Figura 37. Nesta figura são apresentadas as localizações obtidas pelo método de seccionamento proposto e pelo método planificado. Também são apresentadas as posições dos sensores da estrutura e as posições reais das fontes.

As linhas pretas tracejadas ligam a posição estimada da fonte com a posição real desta, assim pode se ter uma noção qualitativa do erro de cada método comparando-se os comprimentos destas linhas tracejadas.

Pode ser observado que no corpo cilíndrico, os dois métodos de localização geram resultados muito semelhantes, sendo ambos muito próximos à posição real da fonte. Já nos tampos elipsoidais, o método planificado gera resultados piores do que o método de seccionamento.

Ainda que o método de seccionamento tenha gerado resultados muito próximos da posição real da fonte para a maioria dos pontos, ainda existem alguns pontos em que este método falha, provavelmente devido à não convergência do procedimento iterativo.

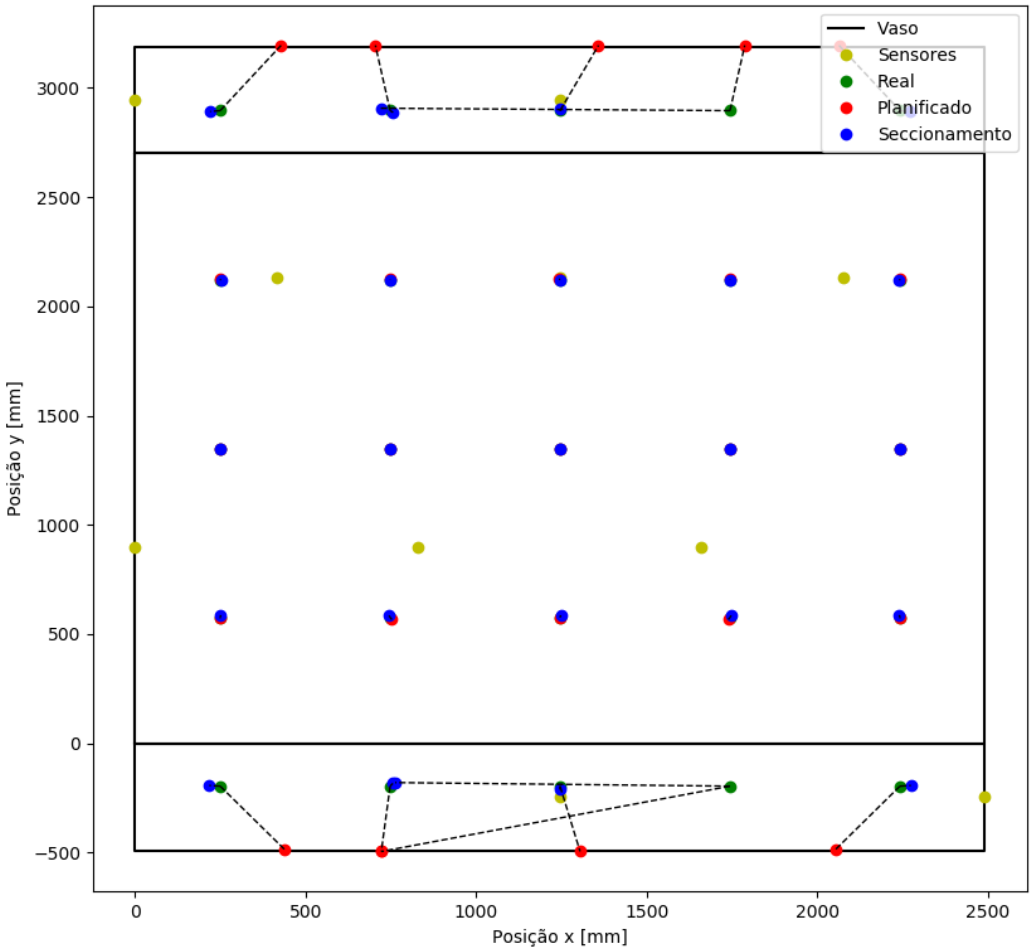


Figura 37 - Análise numérica - localização geral

Fonte: Os autores

O segundo teste resultou na Figura 38. Os resultados apresentados são muito semelhantes ao teste anterior, a única diferença são as posições das fontes de sinal e as linhas tracejadas nesta figura possuem a mesma cor do método ao qual estão relacionadas: linha tracejada vermelha está relacionada com o método planificado, linha tracejada azul está relacionada com o método de seccionamento.

Novamente pode-se observar melhor concordância do método de seccionamento.

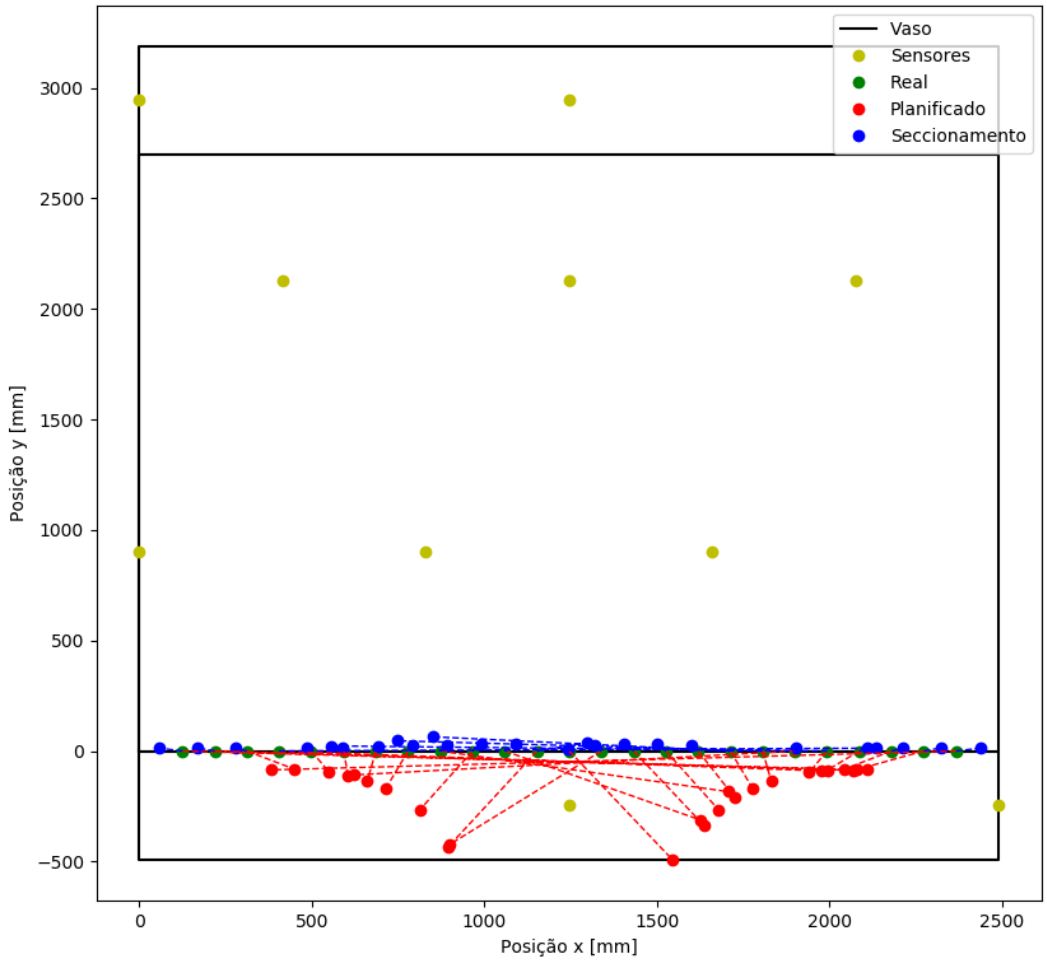


Figura 38 - Análise numérica - localização interface

Fonte: Os autores

Na Figura 39 são apresentados os desvios da localização, em função da posição x real da fonte, para os dois métodos. Este gráfico confirma que o método de seccionamento possui, no geral, melhor concordância, mas também pode se observar que existem alguns pontos em que o método falha, resultando em localizações muito distantes da fonte real.

Nesta figura os erros são apresentados como um percentual da diagonal do vaso, desta forma pode se fazer comparações com estruturas de tamanhos diferentes de forma mais direta.

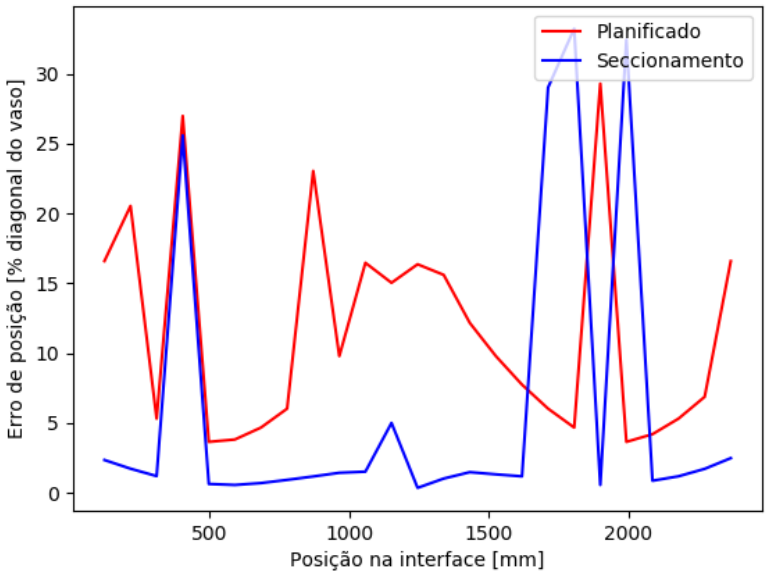


Figura 39 – Desvios da localização - localização interface

Fonte: Os autores

Por fim, o terceiro teste resultou na Figura 40. Nesta figura são apresentadas as mesmas informações do segundo teste.

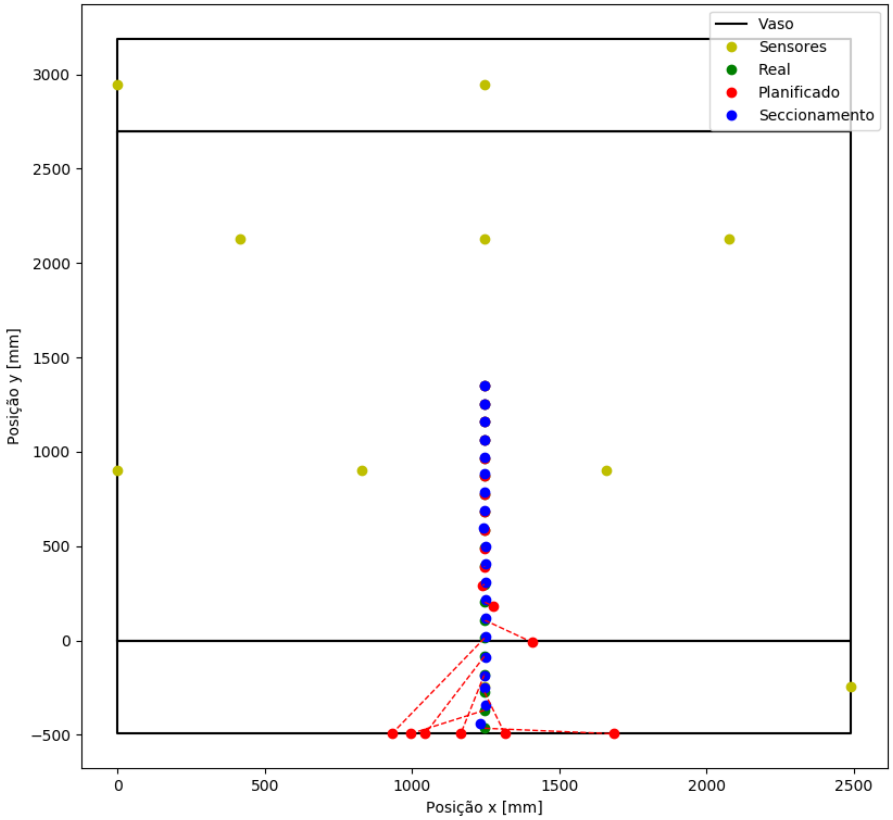


Figura 40 - Análise numérica - localização linha vertical

Fonte: Os autores

Na Figura 41 são apresentados os desvios dos métodos de localização em função da altura da posição real da fonte. Neste gráfico fica claro que o método de seccionamento se torna superior ao planificado quando a fonte se aproxima do tampo, o que era esperado, uma vez que o método planificado distorce apenas as distâncias nos tampos.

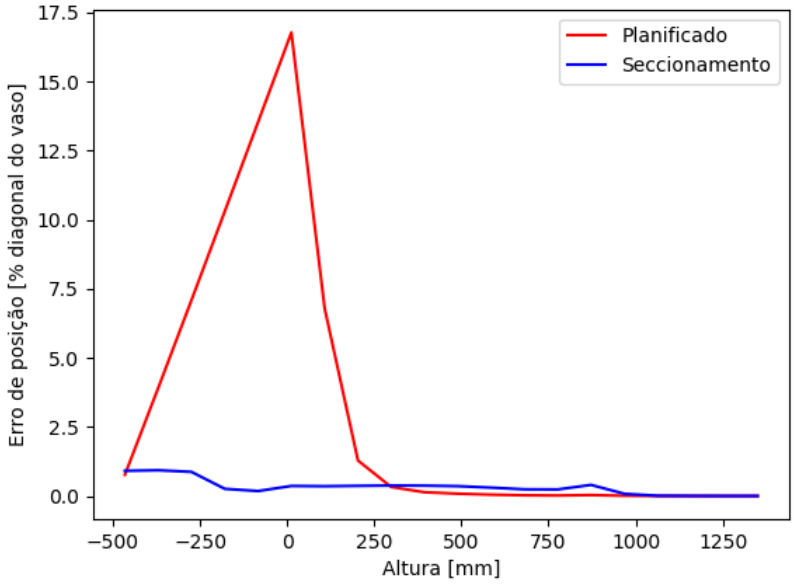


Figura 41 – Desvios da localização - localização linha vertical

Fonte: Os autores

6.1. Análise empírica

O primeiro teste da análise empírica gerou o arquivo de resultados do auto teste de sensores. Deste arquivo foram extraídos os tempos de chegada aos sensores, estes tempos foram então usados para tentar se localizar a fonte do sinal.

Na Figura 42 são apresentadas as posições reais das fontes (que correspondem as posições dos sensores), as localizações a partir do método planificado e as localizações a partir do método de seccionamento.

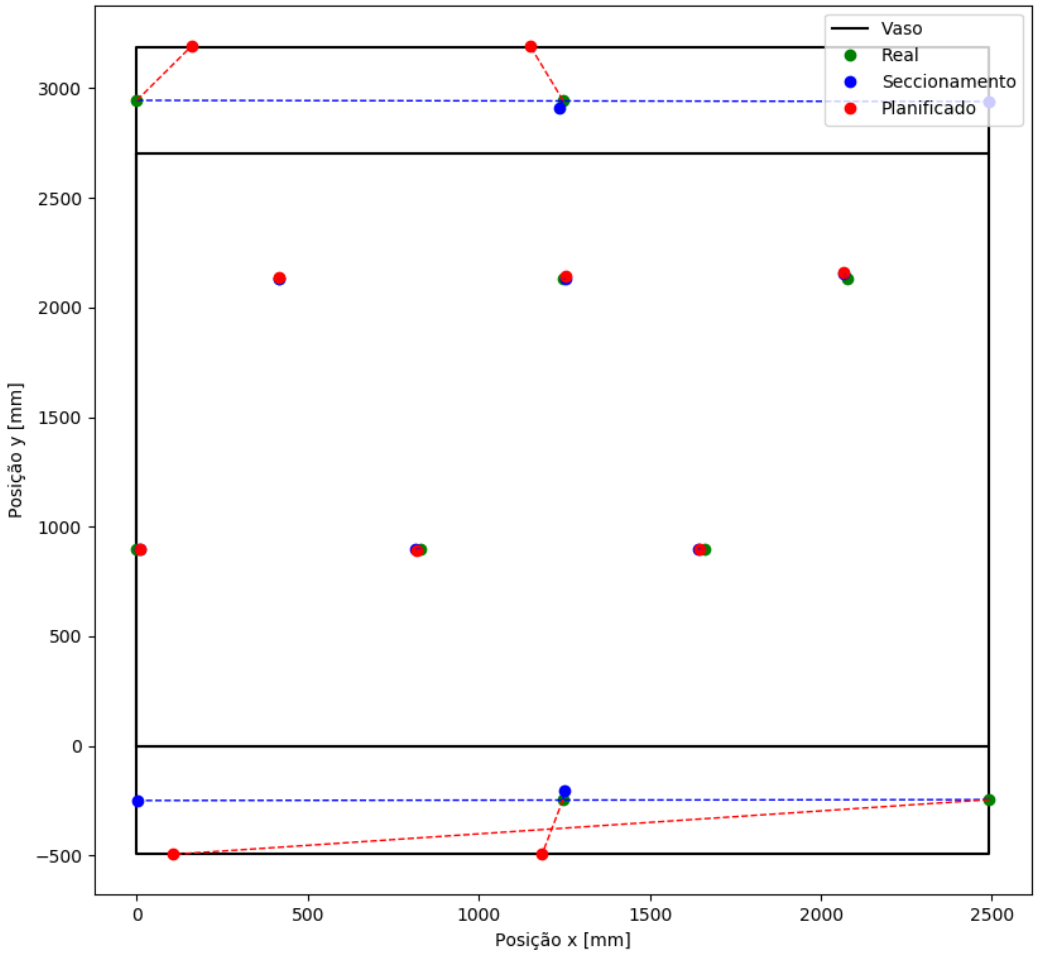


Figura 42 - Localização a partir do auto teste

Fonte: Os autores

Vale ressaltar que as linhas verticais que delimitam a vista planificada são coincidentes na geometria do vaso de pressão. Por este motivo em algumas situações o ponto localizado aparenta estar distante da posição real da fonte, enquanto, na verdade, está muito próximo.

Na Tabela 2 são apresentadas as localizações e os desvios em relação à posição real da fonte para cada método.

Tabela 2 - Desvio das localizações no auto teste

Ponto	Posição real		Planificado			Seccionamento		
	X [mm]	Y [mm]	X [mm]	Y [mm]	Erro [% diagonal]	X [mm]	Y [mm]	Erro [% diagonal]
1	0.00	900.00	9.90	899.63	0.33%	9.00	900.55	0.30%
2	830.67	900.00	816.52	891.36	0.56%	815.70	895.03	0.53%
3	1661.33	900.00	1646.15	900.00	0.51%	1640.66	900.05	0.70%
4	415.33	2130.00	417.23	2135.65	0.20%	415.95	2131.48	0.05%
5	1246.00	2130.00	1253.52	2141.10	0.45%	1252.84	2131.45	0.23%
6	2076.67	2130.00	2066.81	2158.98	1.03%	2068.09	2154.65	0.88%
7	1246.00	-245.00	1184.78	-494.90	8.08%	1251.27	-204.50	1.37%
8	2492.00	-245.00	104.38	-494.90	8.08%	1.43	-249.12	0.14%
9	0.00	2945.00	161.42	3194.90	8.09%	2492.76	2939.71	0.18%
10	1246.00	2945.00	1149.25	3194.90	8.08%	1235.48	2913.06	1.10%

Se observa que as localizações obtidas pelo método de seccionamento geralmente são melhores, sendo que para as fontes que estão nos tampos (ponto 7 a 10) a diferença dos dois métodos é bastante significativa.

Para o segundo teste empírico foram gerados 3 arquivos de dados, sendo um arquivo para cada linha de pontos de teste. Este arquivo foi gerado com o software comercial *AEWin,* que depois também foi o responsável por exportar os dados obtidos para um arquivo de texto que pudesse então ser processado pela rotina de localização.

A Figura 43 apresenta os resultados da localização para a linha 1. Nesta figura são apresentadas as localizações obtidas pelo software *AEWin* e as obtidas pelo método de seccionamento.

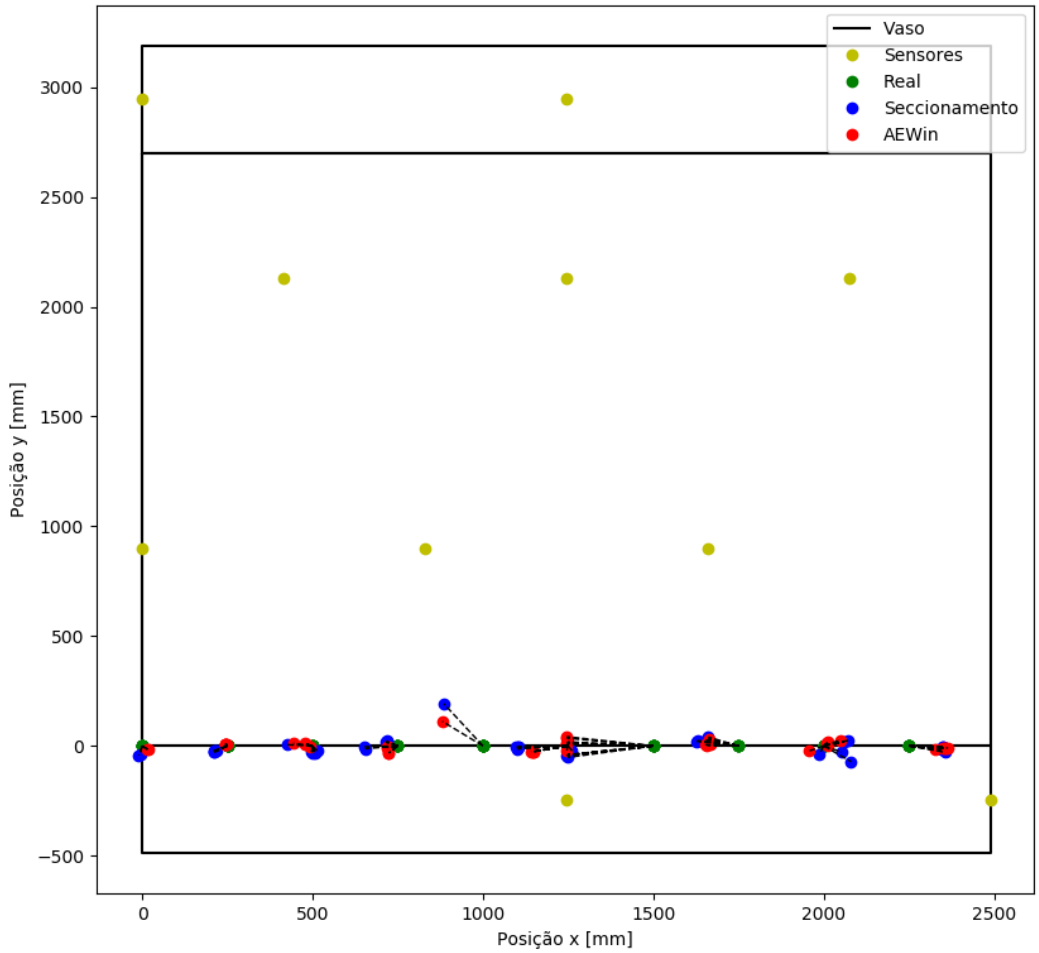


Figura 43 - Localizações linha 1

Fonte: Os autores

Na Figura 44 são apresentadas as mesmas informações da figura anterior, mas agora com detalhe na região de interesse.

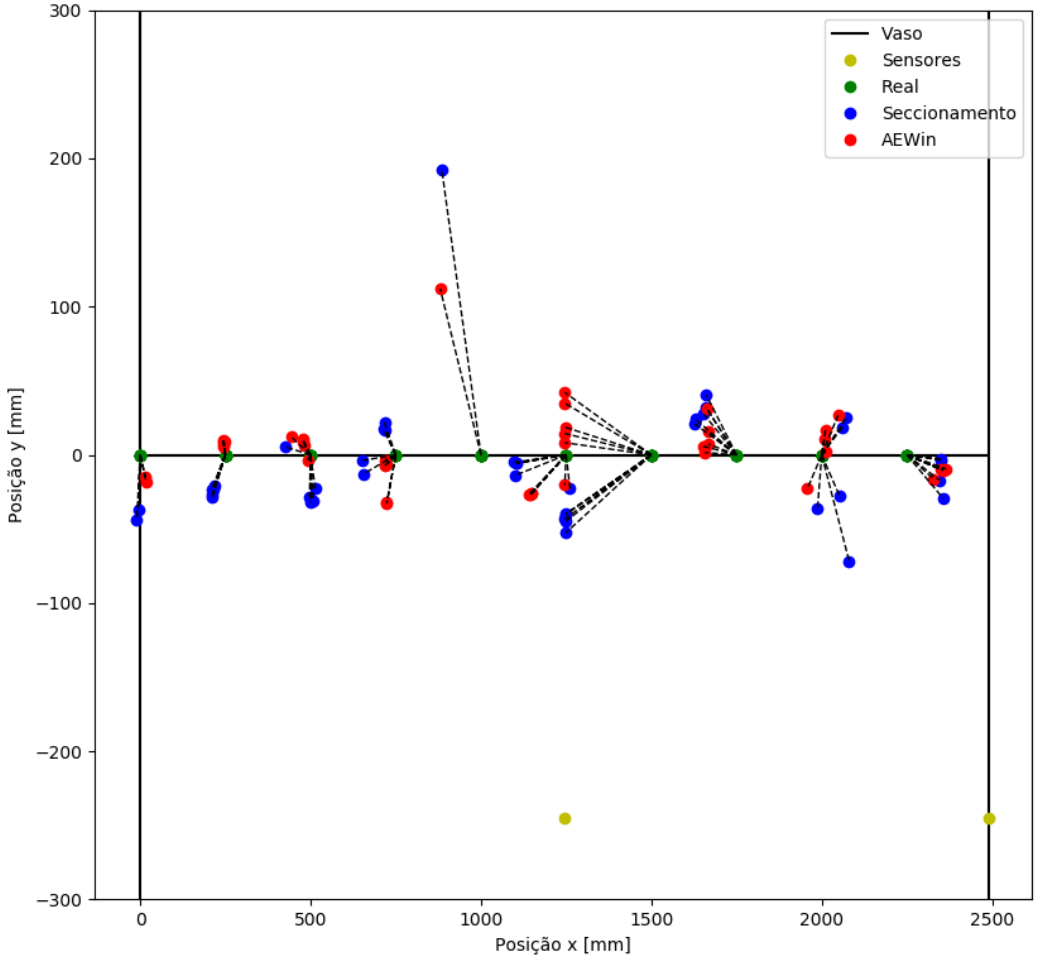


Figura 44 - Localizações linha 1 – detalhe

Fonte: Os autores

Na Figura 45 são apresentados os desvios da localização para os dois métodos. Como foram realizadas uma série de quebras de minas de grafite em cada ponto, obteve-se diferentes localizações para cada quebra; para representar essas várias localizações optou-se pelo gráfico de barras de erro apresentado abaixo.

Neste gráfico, o ponto central da barra representa a média do desvio da localização e a meia largura da barra representa o desvio padrão dos desvios de localização obtidos.

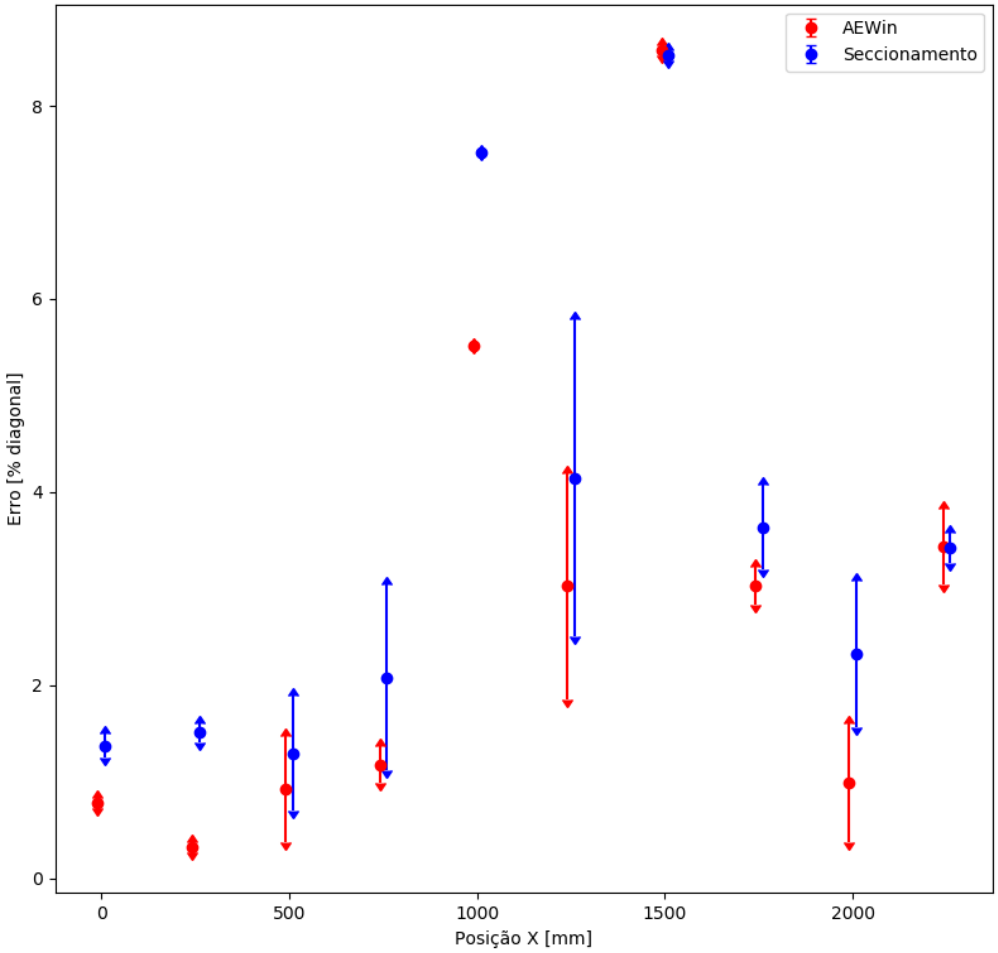


Figura 45 - Localizações linha 1 – desvios

Fonte: Os autores

Para a representação do gráfico acima, foram removidos os *outliers,* portanto os pontos que tiveram grandes desvios com os dois métodos não foram representados. Isso também fez com que em alguns gráficos na sequência, não existisse a barra de erro em algumas posições.

Na Figura 46 são apresentadas as localizações obtidas para a linha 2.

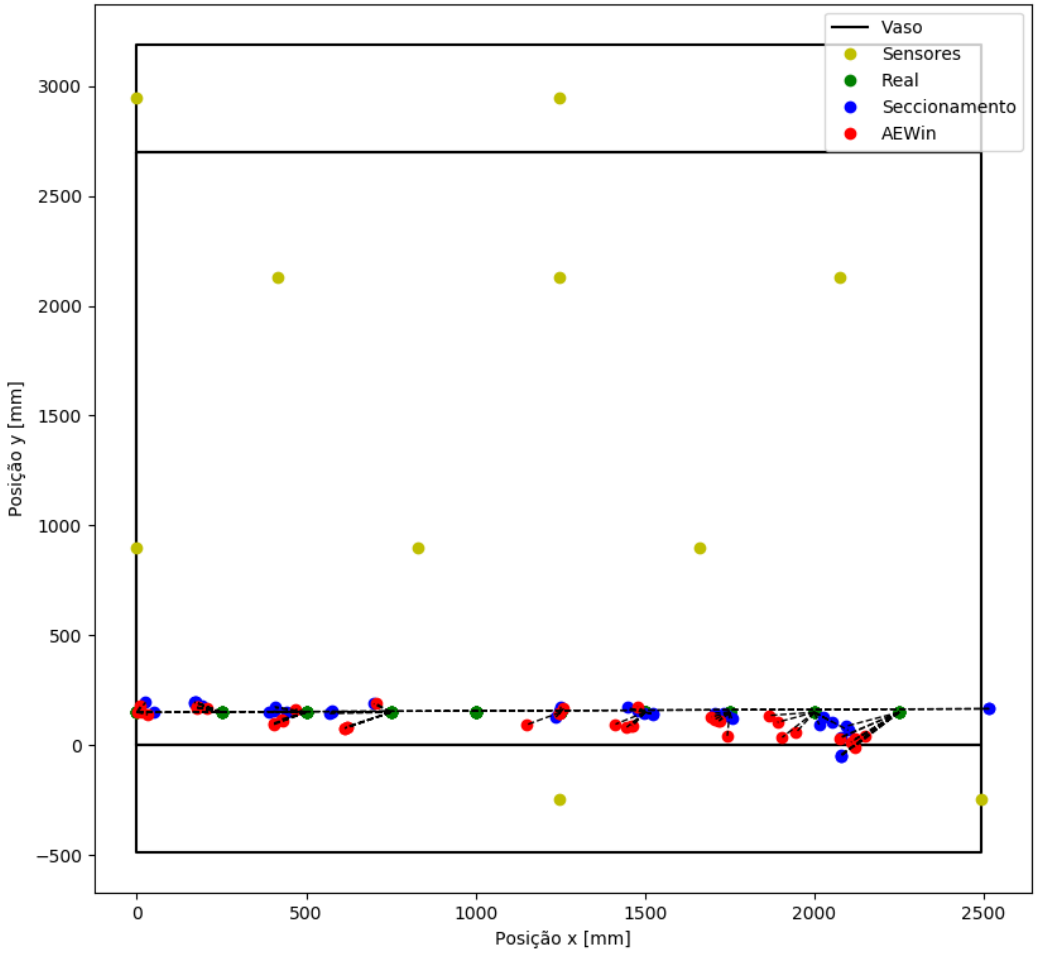


Figura 46 - Localizações linha 2

Fonte: Os autores

Na Figura 47 novamente são apresentadas as mesmas informações da localização, mas agora com detalhe na região de interesse.

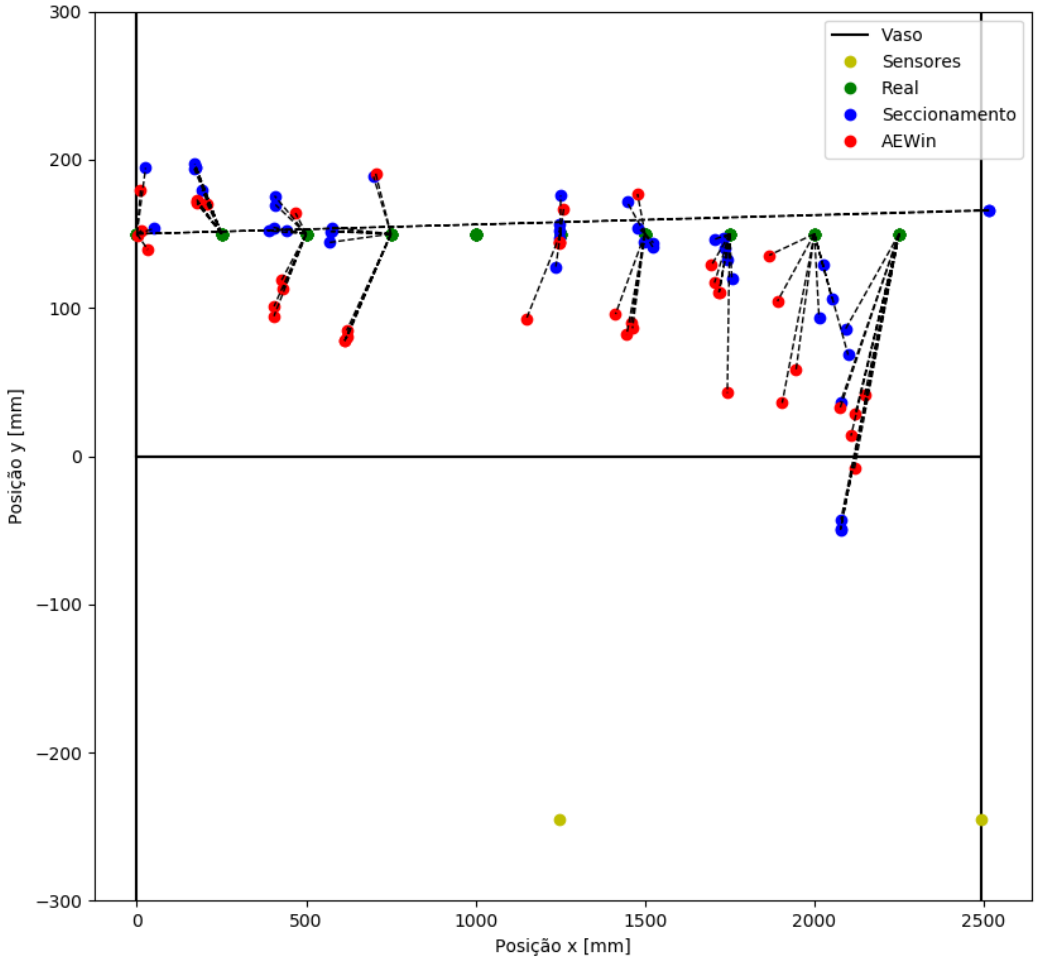


Figura 47 - Localizações linha 2 – detalhe

Fonte: Os autores

Na Figura 48 são apresentados os desvios de localização para a linha 2.

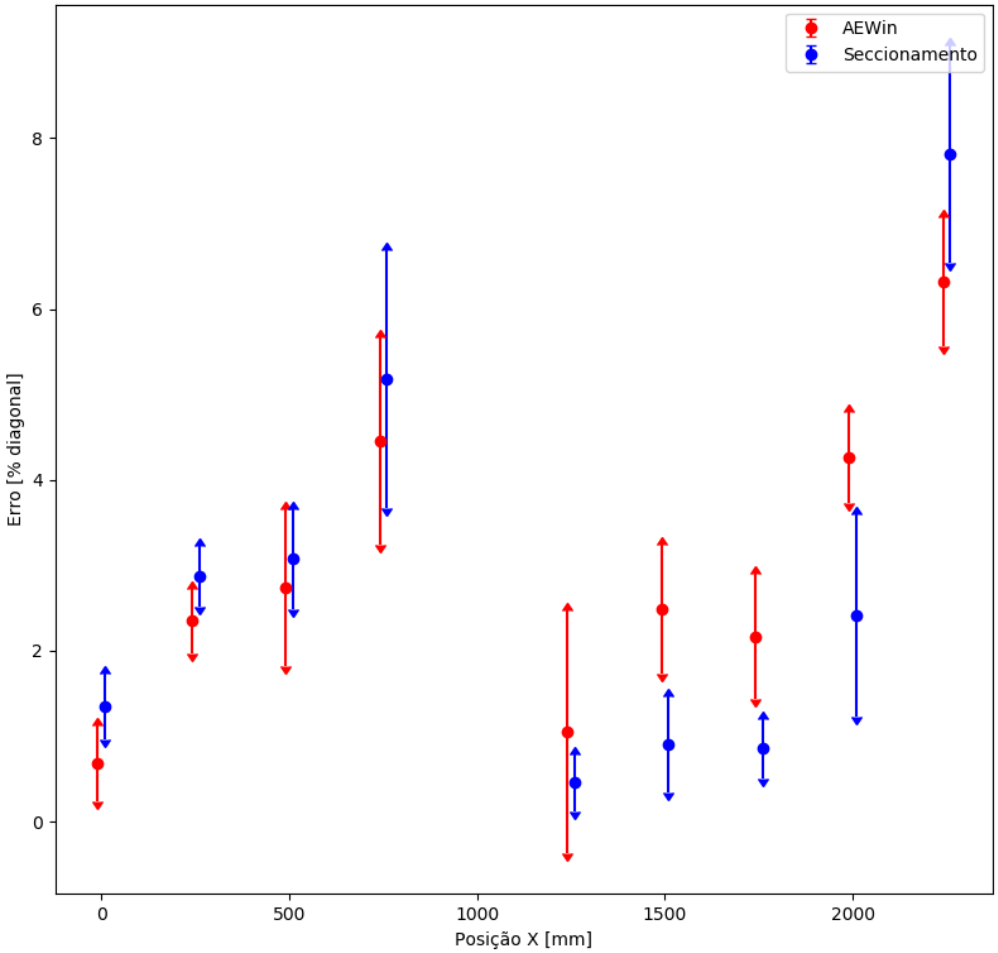


Figura 48 - Localizações linha 2 – desvios

Fonte: Os autores

Na Figura 49 são apresentadas as localizações obtidas para a linha 3.

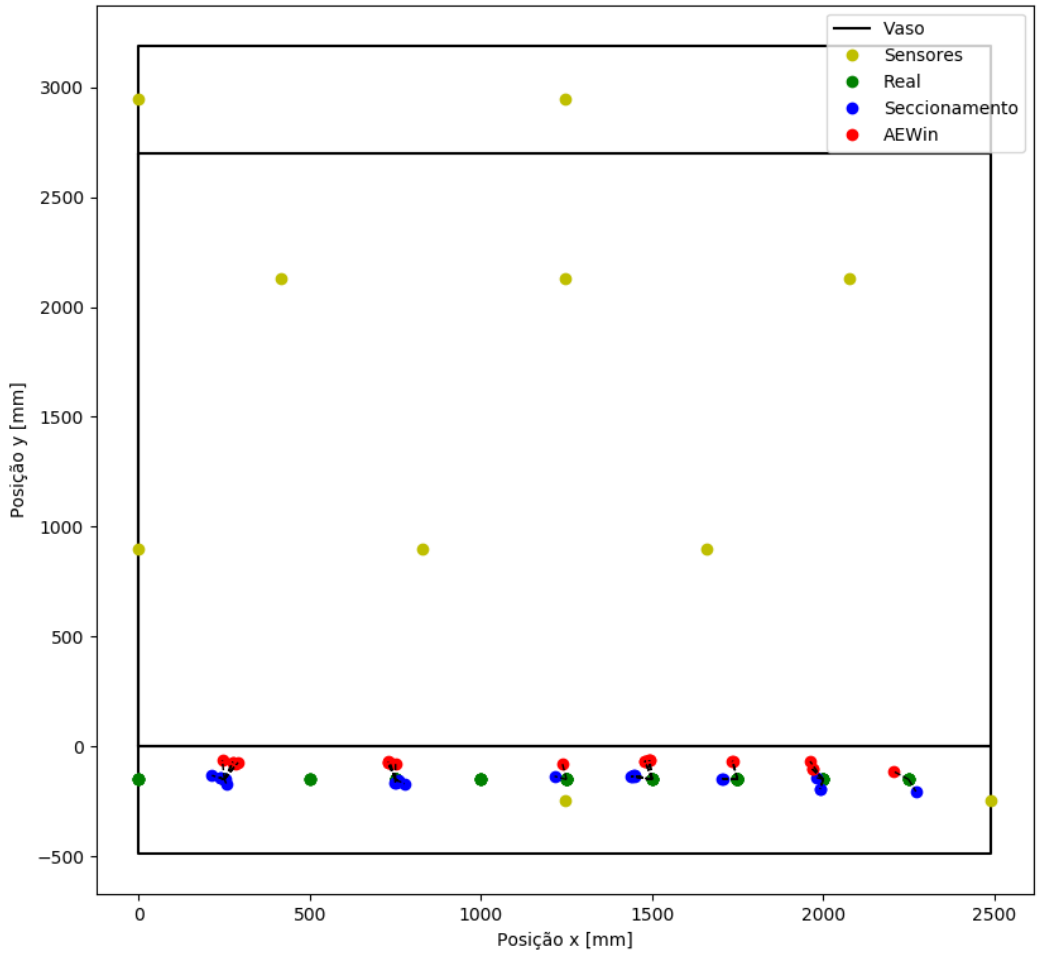


Figura 49 - Localizações linha 3

Fonte: Os autores

Na Figura 50 novamente são apresentadas as mesmas informações da localização, mas agora com detalhe na região de interesse.

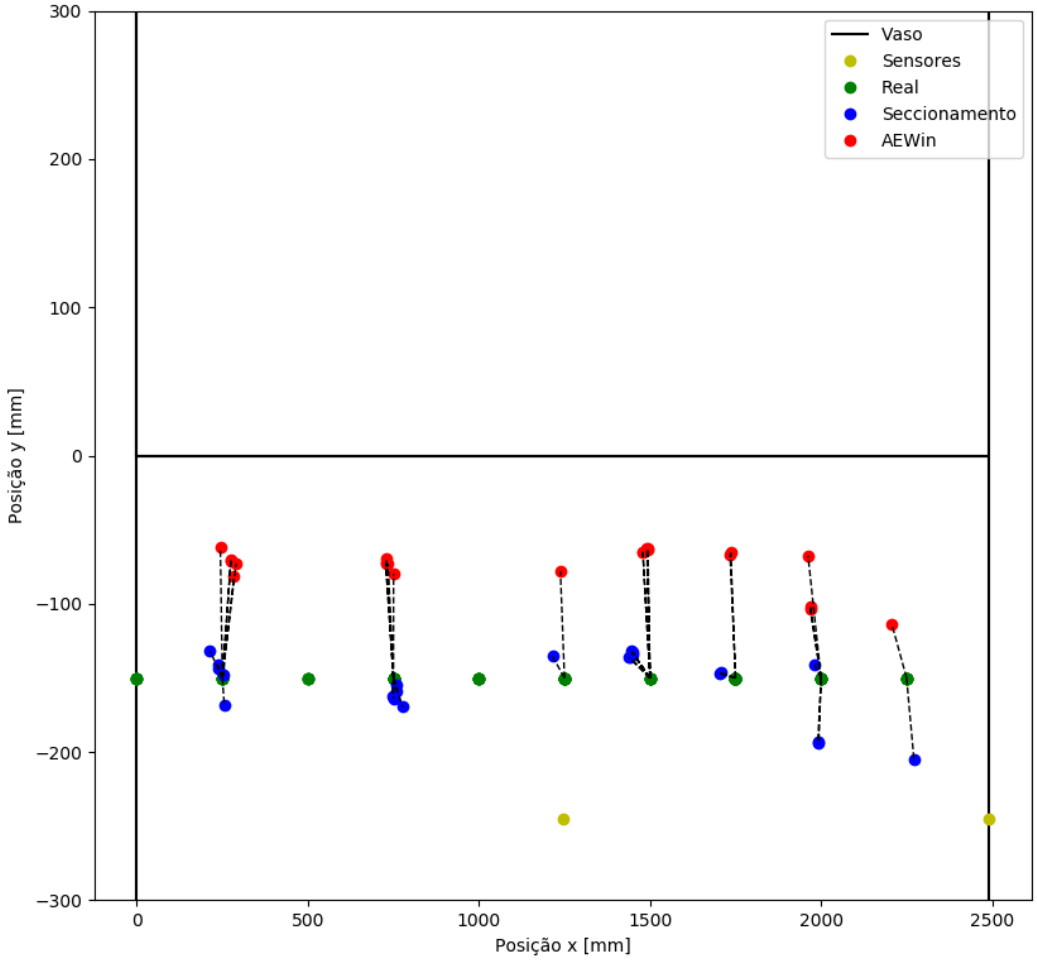


Figura 50 - Localizações linha 3 – detalhe

Fonte: Os autores

Na Figura 51 são apresentados os desvios de localização para a linha 3.

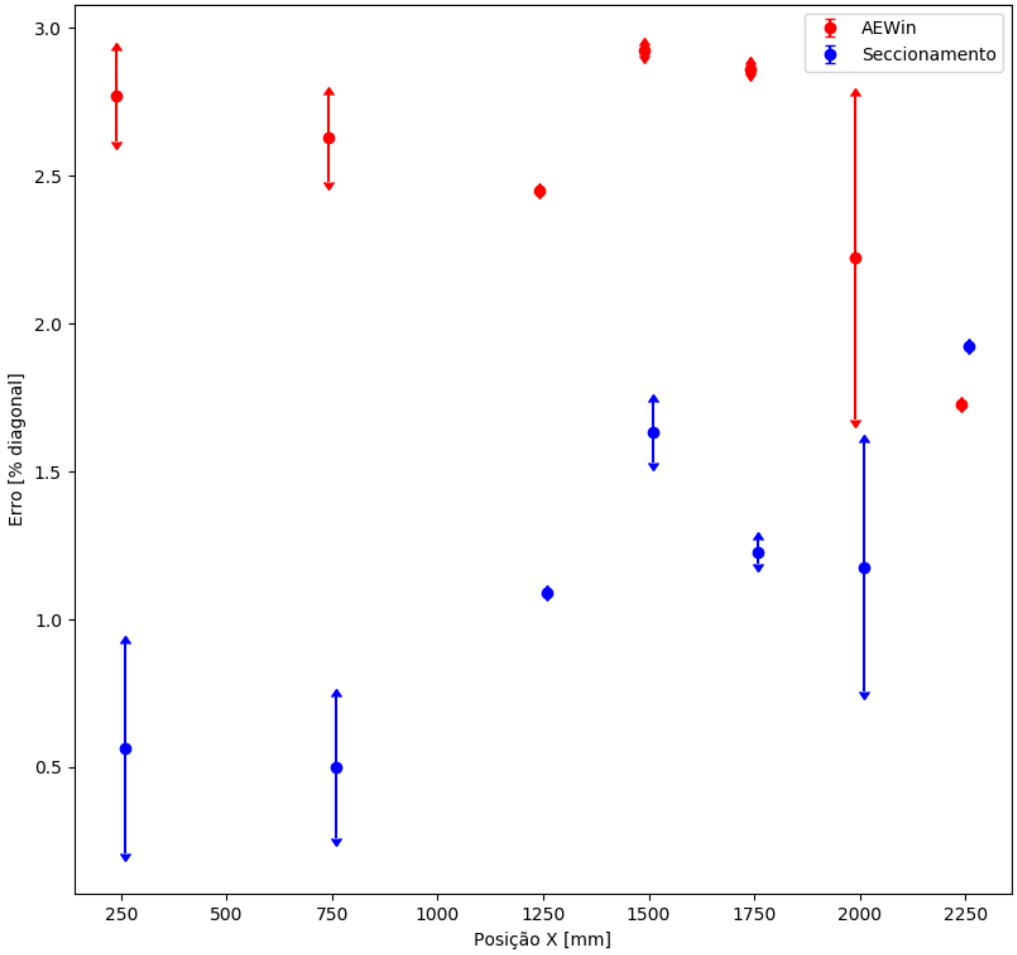


Figura 51 - Localizações linha 3 – desvios

Fonte: Os autores

7. CONCLUSÕES

Os resultados obtidos através do método do seccionamento se mostraram satisfatórios para toda a extensão da estrutura testada. No corpo cilíndrico a localização foi equivalente à do software comercial e à do método tradicional. Já no tampo o método proposto se mostrou mais preciso.

O tempo de execução é uma desvantagem do método, pois o cálculo de distâncias no tampo através de uma aproximação da geodésica, ainda que simplificada, é muito mais oneroso computacionalmente que o cálculo da distância euclidiana. Por isso não é possível, na configuração atual, empregar esse método para localização de defeitos em tempo real, mas é uma alternativa para uma análise posterior.

Para maior aplicabilidade da técnica, sugere-se que em trabalhos futuros seja otimizada a execução do código, tornando possível seu uso em tempo real. Uma alternativa é a seleção do algoritmo de localização com base em uma análise prévia dos tempos, ou seja, aplicando o método de seccionamento para cálculo de distâncias no tampo, onde seu tempo de execução é justificado pelo ganho de precisão.

A segunda parte dos testes empíricos revelou que o método proposto apresenta resultados inferiores ao método convencional na interface entre corpo e tampo, o que pode ser consequência da função de regressão apresentada na equação (3.14). Em trabalhos futuros pode ser estudada também a substituição dessa função por outra que represente melhor a curva obtida pelo método iterativo.

Por fim, para aplicação robusta do método do seccionamento em sistemas de emissão acústica é sugerida a realização de uma análise de sensibilidade, para se verificar a resposta desse em função de desvios nos tempos de chegada e parâmetros da geometria. É necessário também analisar a convergência do método para toda extensão da estrutura.

8. BIBLIOGRAFIA

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. PHYSICAL ACOUSTICS CORPORATION. **SAMOS AE SYSTEM USER'S MANUAL**. [S.l.]. 2005.
11. PHYSICAL Acoustics. Disponivel em: <https://www.physicalacoustics.com/by-product/sensors/R15I-AST-150-kHz-Integral-Preamp-AE-Sensor>. Acesso em: 3 dez. 2018.

12. 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>.

13. KAISER, J. **Untersuchung über das Auftreten von Geräuschen beim Zugversuch**. München. 1950.

14. KARNEY, C. F. F. Algorithms for geodesics, 26 jun. 2012. 13.

15. PRICE, K. V. An introduction to differential evolution. New Ideas in Optimization. [S.l.]: [s.n.], 1999. p. 79–108.

16. 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.

17. ANDREUCCI, R. **Ensaio por Ultra-Som**. São Paulo: US, 2003.

18. ENVIROCOUSTICS. Disponivel em: <http://www.envirocoustics.gr/products/acoustic_emission/ae_disp_eng.htm>. Acesso em: 3 dez. 2018.

APÊNDICE A - 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, y1, x2, 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 = 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, 'simple')
            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 = 1000
        polish = True

        """
        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))
            residue += 1E-10
```

```
            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 = 1500
        polish = False

        res = opt.differential_evolution(CalcResidue, bounds=bounds,
maxiter=maxiter, polish=polish, disp=False)

        # print(res)  # - Resultado da otimização

        # self.__printTimes()

        return res.get("x")
```