**UNIVERSIDADE FEDERAL DO PARANÁ**

**ENGENHARIA MECÂNICA**

**JÉSSICA MENEGUEL**

**LEONARDO SIRINO**

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

**CURITIBA**

**2018**

**JÉSSICA MENEGUEL**

**LEONARDO SIRINO**

****

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

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

Orientador: Prof. Dr. Luciano Kiyoshi Araki

**CURITIBA**

**2018**

**TERMO DE APROVAÇÃO**

JÉSSICA MENEGUEL

LEONARDO SIRINO

LOCALIZAÇÃO DE FONTES ACÚSTICAS EM CORPOS CILINDRÍCOS DE EXTREMIDADES ELIPSOIDAIS

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

BANCA EXAMINADORA

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

Prof. Dr./Ms……..

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

Presidente da Banca

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

Prof. Dr./Ms……..

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

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

Prof. Dr./Ms……..

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

Curitiba

2018DEDICATÓRIA

(OPCIONAL). É a menção em que o autor presta homenagem ou dedica o trabalho a alguém. É colocada em folha distinta, logo após a folha de rosto, geralmente no fim da página no canto direito ou no final da página, justificado a direita e em negrito.

Exemplo 1:

***Ao meu sonho de um sistema diferente***

***Dedico***

Exemplo 2:

***Dedico este trabalho aos colegas de cooperativa que xxxxxxxxxxxxxxxxx.***

AGRADECIMENTOS

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

Exemplo 1:

A

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

***pelas orientações xxxxxxxxxxxxxxxx.***

***Joaquim da Silva***

***por xxxxxxxxxxxxxxxxxx.***

***Carmem Cristina***

***devido xxxxxxxxxxxxxxxxxxxxxx.***

***Exemplo 2:***

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

EPÍGRAFE

(Opcional) Folha onde o autor apresenta uma citação, seguida da indicação de autoria, relacionada com a matéria tratada no corpo do trabalho.

É a inscrição de um trecho em prosa ou composição poética que de certa forma embasou a construção do trabalho.

Exemplo:

***“Pouco conhecimento faz que as criaturas se sintam orgulhosas”.***

***Leonardo da Vinci***

LISTA DE ILUSTRAÇÕES

[Figura 1 - Elementos de um sensor de EA 14](#_Toc517908517)

[Figura 2 - Fluxograma do sinal de EA 14](#_Toc517908518)

[Figura 3 – Tempos de um sinal de EA 15](#_Toc517908519)

[Figura 4 – Alguma imagem relacionada aos parâmetros de um hit 16](#_Toc517908520)

[Figura 5 – Alguma imagem do programa com vários hits 16](#_Toc517908521)

[Figura 6 - Localização planar pelo método da hipérbole. T1, T2 e T3 são os tempos de chegada das ondas mecânicas nos sensores correspondentes 17](#_Toc517908522)

[Figura 7 - Localização planar com dois sensores 17](#_Toc517908523)

LISTA DE TABELAS

Nenhuma entrada de índice de ilustrações foi encontrada.

LISTA DE ABREVIATURAS E SIGLAS

|  |  |
| --- | --- |
| EA | Emissão acústica |
| END | Ensaio não destrutivo |
|  |  |
|  |  |
|  |  |
|  |  |

SUMÁRIO

[1. INTRODUÇÃO 11](#_Toc517908507)

[2. Revisão bibliográfica 13](#_Toc517908508)

[2.1. Emissão acústica 13](#_Toc517908509)

[2.1.1. A origem da técnica 13](#_Toc517908510)

[2.1.1. Equipamentos 13](#_Toc517908511)

[2.1.1. processamento do sinal de EA 15](#_Toc517908512)

[3. Localização 16](#_Toc517908513)

[4. Geodésica 18](#_Toc517908514)

[5. REFERÊNCIAS 20](#_Toc517908515)

RESUMO

A técnica de Emissão Acústica (EA) é um ensaio 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 poder monitorar uma grande região do equipamento em operação, com uso de poucos sensores, podendo ainda se determinar a posição de eventuais defeitos.

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 geometria da estrutura analisada. Quando se analisa 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, isso gera certa imprecisão nos resultados, principalmente para fontes sonoras nas extremidades.

O presente trabalho propõe uma alternativa para a técnica de localização em corpos cilíndricos com extremidades elipsoidais de modo que não haja distorções na geometria e forneça resultados mais precisos.

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

# INTRODUÇÃO

Emissão Acústica (EA) é uma técnica de ensaio não destrutivo (END) baseada na dec. 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 pois o ensaio interfere pouco na operação do equipamento e tem curta duração, e o reparo necessário devido aos eventuais defeitos encontrados é restringido a uma área limitada indicada nos resultados. Além disso, há economia relacionada a não necessidade de escavar de tubulações enterradas ou remover revestimentos quando na aplicação da técnica.

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

Uma aplicação usual de EA é o monitoramento, contínuo ou durante testes hidrostáticos, em vasos de pressão, que têm como principais causas de falhas trincas e vazamentos, defeitos os quais são detectáveis pela técnica. O tempo e custo de ações corretivas, como reparos por soldagem, é minimizado pela aplicação de ensaios de EA, já que a área na qual são empregados ensaios subsequentes, como ultrassom, e o próprio reparo da estrutura, se restringe àquela na qual foram localizadas fontes de EA, tornando a técnica muito atrativa para esta aplicação.

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

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

# Revisão bibliográfica

## Emissão acústica

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

### A origem da técnica

O primeiro registro do uso da técnica de EA data do século VIII pelo alquimista árabe Jabiribn Hayyan, quando 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 analisava apenas as fontes audíveis, esse tipo de relato continuou com Robert Anderson testando corpos de prova de alumínio além de seu ponto de escoamento; com Erich Scheil relatando ruído audível durante a formação de martensita no aço.

O começo da era moderna da técnica de EA teve início com um dos trabalhos mais importante até hoje, o trabalho de PhD de Joseph Kaiser, intitulado Investigação da ocorrência de ruído durante o ensaio de tração (*Untersuchung über das Auftreten von Geräuschen beim Zugversuch*), que registrou o primeiro relato do que hoje é conhecido como efeito Kaiser. Jopseh Kaiser observou que amostras que já haviam sido submetidas à 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 a tecnologia atual.

### Equipamentos

O uso moderno de EA não se limita as fontes audíveis, sensores piezelétricos são usados para captar ondes 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 1 são apresentados os componentes de um sensor de EA.

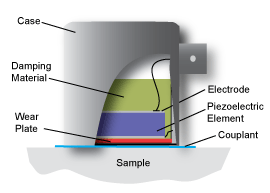


Figura - Elementos de um sensor de EA

Retirado de 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 à esta deformação, este 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 2.

FIGURA DO FLUXOGRAMA DE SINAL – LIVRO UEGA



Figura 2 - Fluxograma do sinal de EA

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, é enviado ao conversor analógico-digital (*Analog Digital Converter* – ADC). Este sinal, agora digitalizado, deve ser processado para que se retire as informações pertinentes, este processamento pode ser feito por um processador convencional, um circuito dedicado ou, mais frequentemente, um chip FPGA (*Field Programmable Gate Array*).

O processamento de sinais de EA é uma tarefa de grande custo computacional, já que as frequências de amostragem geralmente são elevadas (acima de 1 MHz) para que possa se registrar de maneira fidedignas sinais de EA com frequências bastante elevadas. Usar um processador convencional para esta tarefa pode limitar o número de canais de um sistema a um valor impraticável, por esse motivo se torna interessante o uso de FPGA’s. Os FPGA’s são circuitos integrados programáveis que permitem que as operações realizadas no sinal de EA sejam diretamente implementadas em hardware, fazendo com que tenha desempenho semelhante a de circuitos dedicados, mas ainda com a flexibilidade próxima a de um processador. Outra vantagem do uso de FPGA’s que 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.

### 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 usa-se as informações dos *hits.* Os *hits* são trechos do sinal de algum sensor que em algum momento ultrapassaram algum valor pré-determinado, denominado limiar de detecção. A duração desses *hits* é definida com base em alguns parâmetros, que também são definidos previamente. A Figura 3 apresenta como esses parâmetros são observados em um sinal.



Figura 3 – Tempos de um sinal de EA

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.



Figura 4 – Alguma imagem relacionada aos parâmetros de um hit

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



Figura 5 – Alguma imagem do programa com vários hits

## Localização

Nos softwares comerciais de Emissão Acústica 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 de EA empregam 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 por os diferentes sensores arranjados na estrutura. Uma interpretação geométrica do cálculo da localização planar são as hipérboles, que são definidas como curvas na quais é constante a diferença das distâncias de cada um dos seus pontos a dois pontos fixos ou focos. Logo, sobre uma hipérbole cujos focos são dois sensores i e j o valor da diferença entre os tempos de chegada, ti e tj, é constante. Quando é inserido mais um sensor no arranjo é possível traçar mais duas hipérboles. A intersecção das três hipérboles assim geradas ocorre sobre a fonte de emissão acústica, como é mostrado na Figura 6.



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

Tal correlação geométrica é demonstrada matematicamente na Figura 7.

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

Figura - Localização planar com dois sensores

A resolução simultânea das equações, ou seja, a intersecção entre duas hipérboles resulta na posição da fonte de EA.

A localização planar de um 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).

|  |  |  |
| --- | --- | --- |
|  |  | (.) |
|  |  | (.) |
|  |  | (.) |
|  |  | (.) |

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

## Geodésica

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

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.

Para o cálculo de distâncias nesses elementos foi usada a biblioteca para Python "*Geographiclib*". Pacotes dessa, o "*Geodesic*" e "*GeodesicLine*" são baseados na expansão de Taylor das integrais geodésicas válidas quando o achatamento f é pequeno. Já os pacotes *GeodesicExact* e *GeodesicLineExact* apresentam o calculo exato dessas integrais. Para achatamentos pequenos o cálculo aproximado é 2-3 mais rápido e 2-3 vezes mais acurado, já que é mais fácil de controlar erros round-off com solução em séries. Para aplicações em que o achatamento absoluto é maior que 0.02 deve-se usar as classes exatas.

# Métodos

Descrever método de localização no software comercial, como entrar com as coordenadas e tudo mais

## Método de seccionamento

O uso da biblioteca *Geographiclib* se mostrou inviável devido ao tempo de execução das rotinas, isso impossibilitaria o uso desta biblioteca para o cálculo de distâncias em tempo real.

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

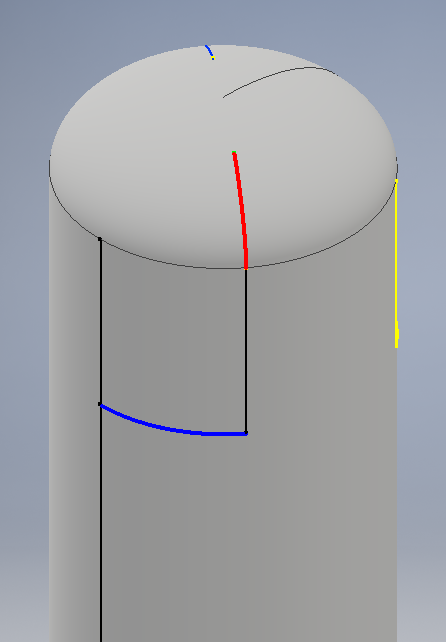
O método de seccionamento consiste em assumir que a geodésica entre dois pontos no elipsoide está contida num plano que seja paralelo ao eixo de revolução do elipsoide e que contenha os dois pontos. Esta abordagem, é descrita na sequência.

### Coordenadas auxiliares

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

Estes pontos devem ser determinados a partir das informações conhecidas dos pontos, as quais são a distância do ponto à 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, numa 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 8 há uma representação dessas coordenadas em um modelo de vaso de pressão de tampo elipsoidal para um certo ponto P0.



s

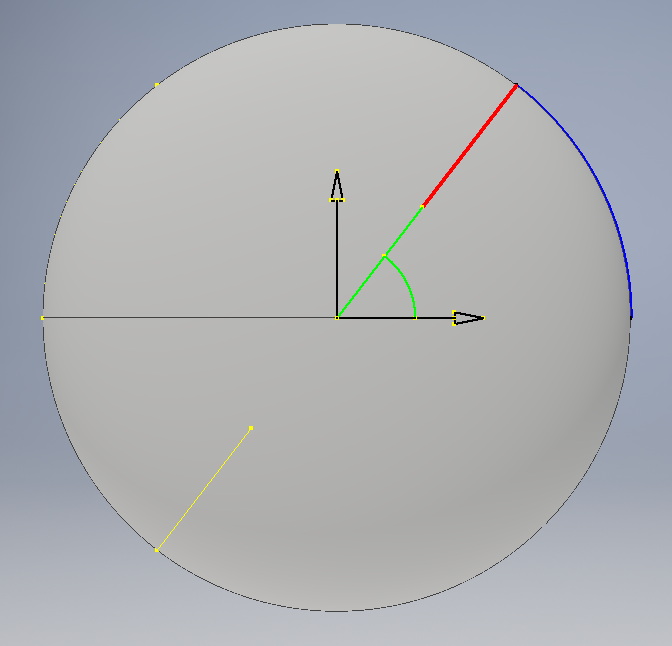
x

P0

Figura 8 - Identificação das coordenadas x e s

Quando o tampo é observado a partir de uma vista superior, o ponto P0 possuirá coordenadas diferentes, essas coordenadas serão denominadas x0’ e y0’, como pode ser observado na Figura 9.

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



**x’**

**y’**

P0: (x0’, y0’)

**λ**

**R**

Figura 9 - Vista superior do tampo - coordenadas x0' e y0'

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

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

|  |  |  |
| --- | --- | --- |
|  |  | (.) |

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 10 é apresentada a definição de uma secante da elipse.

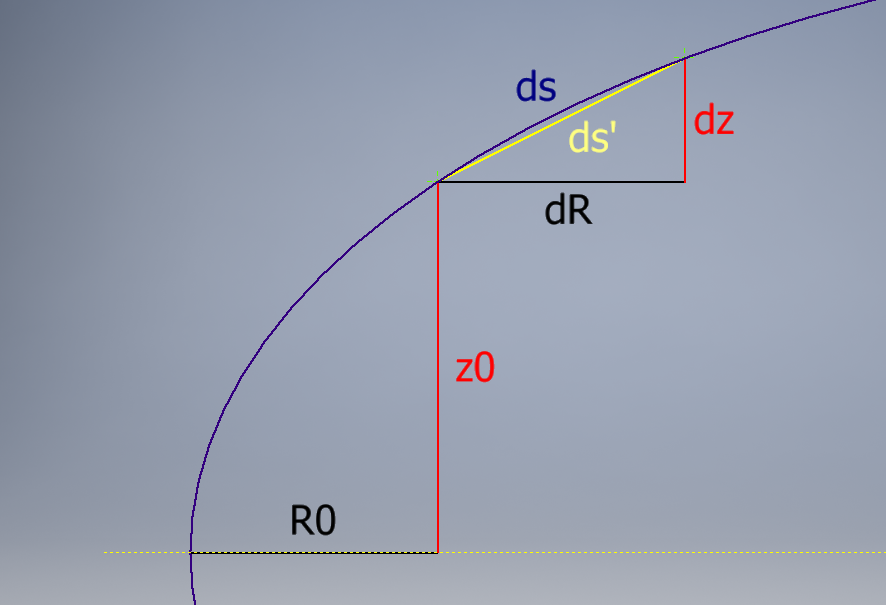


Figura 10 - Secante da elipse

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

|  |  |  |
| --- | --- | --- |
|  |  | (.) |

|  |  |  |
| --- | --- | --- |
|  |  | (.) |

Onde:

Definir essas grandezas na seção de geodésicas, definir todas as variáveis e as letras usadas para elas

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

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:

|  |  |  |
| --- | --- | --- |
|  |  | (.) |
|  |  | (.) |
|  |  | (.) |

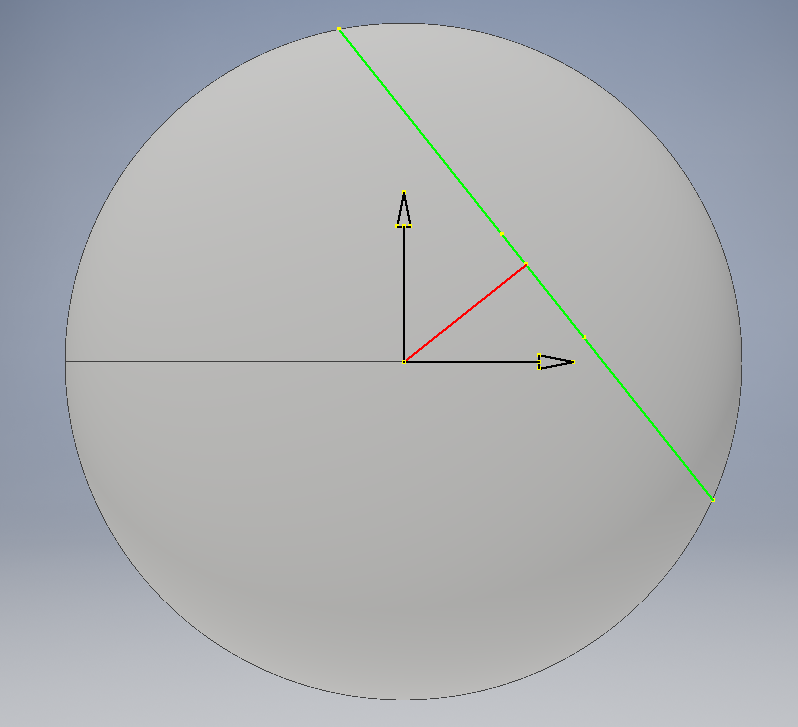
### Plano de seccionamento e elipse auxiliar

Com a informação das coordenadas auxiliares de dois pontos no elipsoide, pode se determinar a distância do plano que contém esses pontos até o eixo de revolução do elipsoide. Para isso, é observada a vista superior do elipsoide, onde o plano que contém os dois pontos é observado como uma reta e o problema se reduz a 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 (3.6).

|  |  |  |
| --- | --- | --- |
|  |  | (.) |

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



**x’**

**y’**

P1: (x1’, y1’)

P2: (x2’, y2’)

**d**

Figura - Vista superior do plano de seccionamento

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

|  |  |  |
| --- | --- | --- |
| Plano: |  | (.) |

|  |  |  |
| --- | --- | --- |
| Elipsoide: |  | (.) |
|  |  |  |

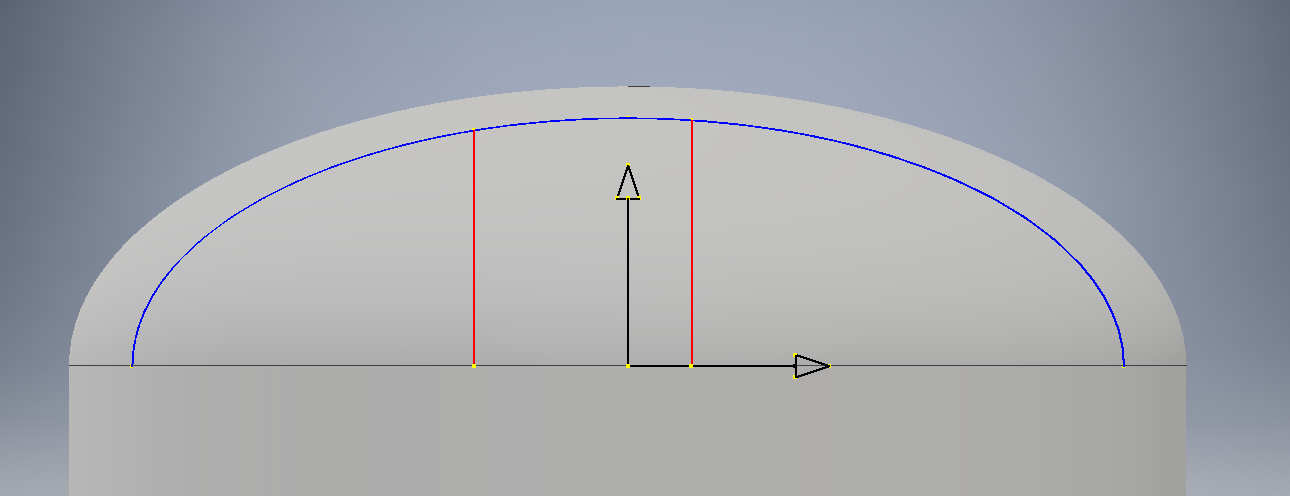
Combinando estas duas equações e realizando algumas manipulações algébricas se chega a:

|  |  |  |
| --- | --- | --- |
|  |  | (.) |

Portanto, os semi-eixos da elipse serão:

|  |  |  |
| --- | --- | --- |
|  |  | (.) |
|  |  | (.) |

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



P1: (R1’, z1’)

P2: (R2’, z2’)

**z’**

**R’**

Figura - Elipse auxiliar

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

Agora pode se calcular o comprimento do arco de elipse que liga os pontos P1 e P2, para isso é utilizado o procedimento descrito pelo pseudocódigo abaixo.

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

### aproximações

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

Para se contornar esta 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 foram obtidas para um valor de 0.5, que é o valor mais comum para tampos elípticos de vasos de pressão industriais.

A regressão da posição pode ser observada na Figura 13.

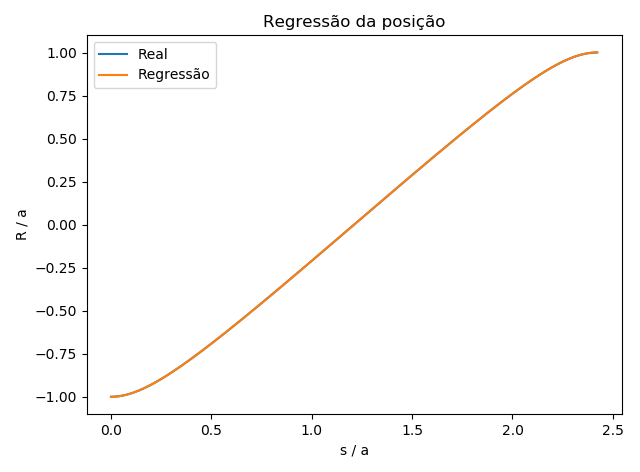


Figura - Regressão Posição x Comprimento do arco

A função encontrada foi:

|  |  |  |
| --- | --- | --- |
|  |  | (.) |

A regressão do arco pode ser observada na

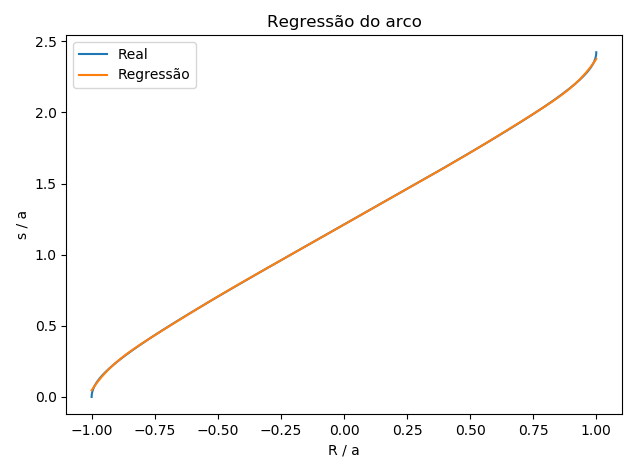


Figura - Regressão Comprimento do arco x Posição

A função encontrada foi:

|  |  |  |
| --- | --- | --- |
|  |  | (.) |

Com essas funções, não há a necessidade do procedimento incremental e a determinação da distância entre os pontos é muito mais rápida, conforme será mostrado na sequência do trabalho.

REFERÊNCIAS

Carlo Giuseppe Filippin, J. B. (2017). Emissão Acústica - Conceitos e Aplicações. Curitiba: Grafo Estúdio.

Kaiser, J. (15 de Fevereiro de 1950). Untersuchung über das Auftreten von Geräuschen beim Zugversuch. *Dr.-Ing. Dissertation, Fakultät für Maschinenwesen und Elektrotechnik der Technischen Universität München*. München.

National Science Foundation. (2001). *Introduction to Acoustic Emission Testing*. Retirado de NDT Resource Center: <https://www.nde-ed.org/EducationResources/CommunityCollege/Other%20Methods/AE/AE_Index.php>

ANEXO I: Código Python

import geographiclib.geodesic as geo

import matplotlib.pyplot as plt

import scipy.optimize as opt

import math as m

import numpy as np

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

self.Valid = True

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 SetValid(self, Valid):

"""Validade de um ponto para se calcular a distância.

Os clones horizontais de pontos no tampo não são válidos e portanto não se deve a calcular a ditância até eles.

Esses são mantidos para manter a equivalência entre os vetores de sensores e clones

Arguments:

Valid {[boolean]} -- [validade do ponto]

"""

self.Valid = Valid

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

if self.Valid:

Valido = "Valido"

else:

Valido = "Invalido"

text = Valido + "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.\_\_SensorListHClone = []

self.\_\_Xpath = []

self.\_\_Ypath = []

self.\_\_GenPlot = False

self.\_\_PointsonPlot = 100

self.\_\_SensorID = -1

self.\_\_tempSensorList = None

self.\_\_tempSensorListHClone = 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

def setCalcMode(self, mode):

self.CalcMode = mode

def setSectionMode(self, mode):

self.SectionMode = mode

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.\_\_DrawVessel()

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)

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

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

self.\_\_DrawVessel()

def SetVelocity(self, velocity):

"Definição da velocidade em mm/s"

self.veloc = velocity

def PrintAllSensors(self):

print("Sensores originais:")

for sensor in self.SensorList:

print(sensor)

print("\n Clones horizontais:")

for sensor in self.\_\_SensorListHClone:

print(sensor)

def \_\_DrawVessel(self):

self.\_\_VesselXPath = self.\_\_PlotRectangle(0, 0, self.diameter \* m.pi, -self.SemiPerimeter).get("xpath") + self.\_\_PlotRectangle(

0, 0, self.diameter \* m.pi, self.height).get("xpath") + self.\_\_PlotRectangle(0, self.height, self.diameter \* m.pi, self.SemiPerimeter).get("xpath")

self.\_\_VesselYPath = self.\_\_PlotRectangle(0, 0, self.diameter \* m.pi, -self.SemiPerimeter).get("ypath") + self.\_\_PlotRectangle(

0, 0, self.diameter \* m.pi, self.height).get("ypath") + self.\_\_PlotRectangle(0, self.height, self.diameter \* m.pi, self.SemiPerimeter).get("ypath")

def \_\_PlotRectangle(self, x0, y0, width, height):

xpath = [x0, x0 + width, x0 + width, x0, x0]

ypath = [y0, y0, y0 + height, y0 + height, y0]

return {"xpath": xpath, "ypath": ypath}

def \_\_sectionPos(self, s):

if self.SectionMode == "reg":

a = self.diameter / 2

f = self.f

s = s / a

pol = [-0.04520616, 0.38323073, -1.36785798, 2.66208137, -3.10204898, 2.24771868,

0.01275433, -1.00151578]

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 = [9.94406631e-01, -5.42331127e-13, -1.67276958e+00, 9.30333908e-13,

9.92271096e-01, -4.52365676e-13, -1.60763495e-01, 8.69627507e-14,

1.01106572e+00, 1.21105713e+00]

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):

x1 = point1.Xcap

y1 = point1.Ycap

x2 = point2.Xcap

y2 = point2.Ycap

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)

try:

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

except RuntimeWarning:

print("d: " + str(d))

return redF, d

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

"""

print("Coordenada auxiliar tampo X: " + str(Coords.Xcap))

print("Coordenada auxiliar tampo Y: " + str(Coords.Ycap))

print("Coordenada auxiliar tampo Z: " + str(Coords.Zcap))

"""

else:

print("Modo inválido")

Coords.SetOnCap(True)

else:

Coords.SetOnCap(False)

def \_\_PlotonCap(self, lat1, lat2, lon1, lon2, cap):

if self.CalcMode == 'geodesic':

smax = self.cap.Inverse(lat1=lat1, lat2=lat2,

lon1=lon1, lon2=lon2).get("s12")

Path = self.cap.InverseLine(

lat1=lat1, lat2=lat2, lon1=lon1, lon2=lon2)

lenghts = np.linspace(0, smax, self.\_\_PointsonPlot)

oldpathDiam = 0

for s in lenghts:

position = Path.Position(s12=s)

pathDiam = (position.get("lon2") + 180) \* \

self.diameter \* m.pi / 360

if abs(oldpathDiam - pathDiam) > self.diameter \* m.pi \* 0.8:

self.\_\_Ypath.append(m.nan)

self.\_\_Xpath.append(m.nan)

oldpathDiam = pathDiam

lataux = position.get("lat2")

Saux = self.cap.Inverse(

lat1=0, lon1=0, lat2=lataux, lon2=0).get("s12")

if cap == "sup":

Saux = Saux + self.height

else:

Saux = -Saux

self.\_\_Ypath.append(Saux)

self.\_\_Xpath.append(pathDiam)

elif self.CalcMode == 'section':

print("Não há plots no tampo para esse modo")

else:

print("Modo inválido")

def \_\_PlotonWall(self, x1, x2, y1, y2):

xpath = np.linspace(x1, x2, self.\_\_PointsonPlot)

xpath = xpath.tolist()

ypath = np.linspace(y1, y2, self.\_\_PointsonPlot)

ypath = ypath.tolist()

# Correção da coordenada X

aux = True

xcorrect = []

ycorrect = []

k = 0

for x in xpath:

if x > self.diameter \* m.pi:

x = x - self.diameter \* m.pi

if aux:

xcorrect.append(m.nan)

ycorrect.append(m.nan)

aux = False

elif x < 0:

x = x + self.diameter \* m.pi

if aux:

xcorrect.append(m.nan)

ycorrect.append(m.nan)

aux = False

xcorrect.append(x)

ycorrect.append(ypath[k])

k += 1

self.\_\_Xpath += xcorrect

self.\_\_Ypath += ycorrect

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

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

else: # Pontos em tampos opostos

if P1.Cap == "sup":

Psup = P1

Pinf = P2

else:

Psup = P2

Pinf = P1

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

# Distância entre um ponto no casco e outro no tampo

elif P1.OnCap ^ P2.OnCap:

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

else: # Distãncia entre pontos no casco

dist = m.sqrt((P1.Xcord - P2.Xcord) \*\*

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

if self.\_\_GenPlot:

self.\_\_PlotonWall(x1=P1.Xcord, x2=P2.Xcord,

y1=P1.Ycord, y2=P2.Ycord)

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")

if self.\_\_GenPlot:

self.\_\_PlotonCap(lat1=P1.Lat, lat2=P2.Lat,

lon1=P1.Lon, lon2=P2.Lon, cap=P1.Cap)

elif self.CalcMode == 'section':

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

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])

theta1 = np.arccos(np.dot(v1c, v12) /

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

theta2 = np.arccos(np.dot(v2c, v12) /

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

if theta1 > m.pi / 2:

u1 = -u1

if theta2 > m.pi / 2:

u2 = -u2

a = redF \* self.diameter / 2

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

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, Yaux, -2)

AuxGenPlot = self.\_\_GenPlot

self.\_\_GenPlot = False # Desabilitando temporariamente os gráficos para melhor desempenho

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)

self.\_\_AuxCoords(AuxPoint)

AuxPoint.SetOnCap(False)

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")

# print(FinalSearch) -- Resultado da minimização

dist = FinalSearch.get("fun")

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

AuxPoint.SetXcord(MinPos)

BestPoint = AuxPoint

self.\_\_GenPlot = AuxGenPlot

if self.\_\_GenPlot: # Plot do caminho que passa pelo casco e pelo tampo

self.\_\_AuxCoords(BestPoint)

BestPoint.SetOnCap(False)

# Plot do caminho no casco

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

BestPoint.SetOnCap(True)

# Plot do caminho no tampo

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

return dist

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

AuxPoint1 = VesselPoint(0, self.height, -2)

AuxPoint2 = VesselPoint(0, 0, -2)

AuxGenPlot = self.\_\_GenPlot

self.\_\_GenPlot = False # Desabilitando temporariamente os gráficos para melhor desempenho

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)

self.\_\_GenPlot = AuxGenPlot

if self.\_\_GenPlot: # Plot do caminho que passa pelo casco e pelo tampo

self.\_\_AuxCoords(AuxPoint1)

AuxPoint1.SetOnCap(True)

# Plot do caminho no tampo superior

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

AuxPoint1.SetOnCap(False)

AuxPoint2.SetOnCap(False)

# Plot do caminho no casco

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

AuxPoint2.SetOnCap(True)

# Plot do caminho no tampo inferior

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

return dist

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

SemiHeight = self.height / 2

Cond1 = (Source.Ycord > SemiHeight and Sensor.Ycord > SemiHeight) or (

Source.Ycord < SemiHeight and Sensor.Ycord < SemiHeight)

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

if Cond1 and Cond2:

if Source.Ycord > SemiHeight:

YAuxCord = self.height

else:

YAuxCord = 0

AuxPoint1 = VesselPoint(0, YAuxCord, -2)

AuxPoint2 = VesselPoint(0, YAuxCord, -2)

AuxGenPlot = self.\_\_GenPlot

self.\_\_GenPlot = False # Desabilitando temporariamente os gráficos para melhor desempenho

def CalcVerticalClone(Xaux):

"""Função auxiliar a distância entre dois pontos no casco, mas de forma que o caminho passe pelo tampo.

Arguments:

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

Returns:

[dist] -- [distância calculada]

"""

AuxPoint1.SetXcord(Xaux[0])

AuxPoint2.SetXcord(Xaux[1])

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)

totalDist = dist1 + dist2 + dist3

return totalDist

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

InitGuess = opt.brute(

CalcVerticalClone, (SearchRange, SearchRange), Ns=6)

FinalSearch = opt.minimize(

CalcVerticalClone, x0=InitGuess, method="BFGS")

# print(FinalSearch) -- Resultado da minimização

dist = FinalSearch.get("fun")

MinPos1 = FinalSearch.get("x")[0]

MinPos2 = FinalSearch.get("x")[1]

AuxPoint1.SetXcord(MinPos1)

AuxPoint2.SetXcord(MinPos2)

self.\_\_GenPlot = AuxGenPlot

if self.\_\_GenPlot: # Plot do caminho que passa pelo casco e pelo tampo

self.\_\_AuxCoords(AuxPoint1)

AuxPoint1.SetOnCap(False)

# Plot do caminho da fonte até o ponto auxiliar 1

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

AuxPoint1.SetOnCap(True)

AuxPoint2.SetOnCap(True)

# Plot do caminho entre pontos auxiliares

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

AuxPoint2.SetOnCap(False)

# Plot do ponto auxiliar 2 até o sensor

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

else:

dist = -1

return dist

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)

SensorHCloneCoords = VesselPoint(0, Ycord, ID)

if SensorCoords.Ycord >= 0 and SensorCoords.Ycord <= self.height: # Ponto no casco do vaso

# Sensor posicionado na porção direita do casco

if SensorCoords.Xcord >= self.diameter \* m.pi / 2:

XHClone = SensorCoords.Xcord - self.diameter \* m.pi

else:

# Sensor posicionado na porção esquerda do casco

XHClone = SensorCoords.Xcord + self.diameter \* m.pi

SensorHCloneCoords.SetXcord(XHClone)

else: # Ponto em um dos tampos do vaso

SensorHCloneCoords.SetXcord(m.nan)

SensorHCloneCoords.SetValid(False)

self.\_\_AuxCoords(SensorHCloneCoords)

self.\_\_SensorListHClone.append(SensorHCloneCoords)

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 FindFurthestPoint(self):

def CalcDistRemotePoint(x):

distances = self.calcAllDist(

SourceX=x[0], SourceY=x[1], GenPlot=False)

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 \_\_removeSensors(self, 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

ValidSensors = []

InvalidSensors = []

for sensor in self.\_\_SensorListHClone:

try:

IDs.index(sensor.ID)

ValidSensors.append(sensor)

except:

InvalidSensors.append(sensor)

self.\_\_SensorListHClone = ValidSensors

self.\_\_tempSensorListHClone = 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

temp = self.\_\_SensorListHClone + self.\_\_tempSensorListHClone

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

for sensor in temp:

self.\_\_SensorListHClone[sensor.ID] = sensor

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

self.\_\_removeSensors(IDs)

self.\_\_GenPlot = GenPlot

Source = VesselPoint(SourceX, SourceY, -1)

self.\_\_AuxCoords(Source)

MinDistances = []

if GenPlot:

plt.plot(self.\_\_VesselXPath, self.\_\_VesselYPath)

SensorX = []

SensorY = []

for sensor in self.SensorList:

SensorX.append(sensor.Xcord)

SensorY.append(sensor.Ycord)

SensorX.append(SourceX)

SensorY.append(SourceY)

plt.plot(SensorX, SensorY, ".")

i = -1

for sensor in self.SensorList:

i += 1

# Limpando o histórico do plot para cada sensor

self.\_\_Xpath = []

self.\_\_Ypath = []

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

self.\_\_Xpath.append(-10)

self.\_\_Ypath.append(m.nan)

HClloneSensor = self.\_\_SensorListHClone[i]

if HClloneSensor.Valid:

distHClone = self.\_\_calcDist(Source, HClloneSensor)

else:

distHClone = distDirect \* 10

self.\_\_Xpath.append(0)

self.\_\_Ypath.append(0)

self.\_\_Xpath.append(-10)

self.\_\_Ypath.append(m.nan)

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

if distVClone == -1:

distVClone = distDirect \* 10

self.\_\_Xpath.append(0)

self.\_\_Ypath.append(0)

self.\_\_Xpath.append(-10)

self.\_\_Ypath.append(m.nan)

Distances = [distDirect, distHClone, distVClone]

MinDistances.append(np.min(Distances))

if GenPlot:

MinDistIndex = Distances.index(np.min(Distances))

XAllPath = np.array(self.\_\_Xpath)

YAllPath = np.array(self.\_\_Ypath)

Xpaths = np.split(XAllPath, np.where(XAllPath == -10)[0])

Ypaths = np.split(YAllPath, np.where(XAllPath == -10)[0])

XBestPath = Xpaths[MinDistIndex]

XBestPath[0] = m.nan

YBestPath = Ypaths[MinDistIndex]

YBestPath[0] = m.nan

if MinDistIndex == 0:

path = "CD"

elif MinDistIndex == 1:

path = "CH"

else:

path = "CV"

print("Fonte: " + str(round(Source.Xcord, 1)) + " - " + str(round(Source.Ycord, 1)) + " -- Sensor: " + str(round(

sensor.Xcord, 1)) + " - " + str(round(sensor.Ycord, 1)) + " -- Distância: " + str(round(np.min(Distances), 3)) + " -- " + path)

plt.plot(XBestPath, YBestPath)

if GenPlot:

plt.show()

self.\_\_returnSensors()

return MinDistances

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

self.\_\_removeSensors(IDs)

distances = []

for (sensor, clone) in zip(self.SensorList, self.\_\_SensorListHClone):

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

if clone.Valid:

dist2 = np.sqrt((x - clone.Xcord)\*\*2 + (y - clone.Ycord)\*\*2)

else:

dist2 = 10 \* dist1

distances.append(np.min([dist1, dist2]))

self.\_\_returnSensors()

return distances

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

if exact:

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

else:

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

NPdist = np.array(distances)

NPdist += -np.min(NPdist)

times = NPdist / self.veloc

return times

def \_\_orderMembers(self, TimesToSensors):

IDs = []

for member in TimesToSensors:

(ID, time) = member

IDs.append(ID)

IDs.sort()

OrderedMembers = [0] \* len(IDs)

for member in TimesToSensors:

(ID, time) = member

i = IDs.index(ID)

OrderedMembers[i] = member

return OrderedMembers

def simpleLocation(self, TimesToSensors):

data = self.\_\_orderMembers(TimesToSensors)

IDs = []

MeasTimes = []

for member in data:

(ID, time) = member

IDs.append(ID)

MeasTimes.append(time)

MeasTimes = np.array(MeasTimes)

normalizer = 1 / (np.max(MeasTimes) + 1)

def CalcResidue(x):

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

tcalc = np.array(tcalc)

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

return residue

res = opt.minimize(CalcResidue, x0=[400, 500], method='BFGS')

# print("Localização simplificada:")

# print(res.get("x"))

return res.get("x")

def completeLocation(self, TimesToSensors):

data = self.\_\_orderMembers(TimesToSensors)

IDs = []

MeasTimes = []

for member in data:

(ID, time) = member

IDs.append(ID)

MeasTimes.append(time)

MeasTimes = np.array(MeasTimes)

normalizer = 1 / (np.max(MeasTimes) + 1)

def CalcResidue(x):

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

tcalc = np.array(tcalc)

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

return residue

x0 = self.simpleLocation(TimesToSensors)

res = opt.minimize(CalcResidue, x0=x0, method='L-BFGS-B', options={"gtol": 1E-5}, bounds=[

(-0.01 \* self.diameter \* m.pi, 1.01 \* self.diameter \* m.pi), (-1.01 \* self.SemiPerimeter, 1.01 \* (self.height + self.SemiPerimeter))])

"""

res = opt.differential\_evolution(CalcResidue, bounds = [(

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

"""

print(res) # - Resultado da otimização

return res.get("x")