UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO ACADÊMICO DO AGRESTE
NÚCLEO DE TECNOLOGIA
CURSO DE ENGENHARIA CIVIL

ANDERSON FERREIRA ALVES

DESENVOLVIMENTO DE UM SISTEMA COMPUTACIONAL WEB PARA PROJETO BASEADO EM CONFIABILIDADE DE DUTOS SUJEITOS À MÚLTIPLOS DEFEITOS DE CORROSÃO

Caruaru
2019

ANDERSON FERREIRA ALVES

DESENVOLVIMENTO DE UM SISTEMA COMPUTACIONAL WEB PARA PROJETO BASEADO EM CONFIABILIDADE DE DUTOS SUJEITOS À MÚLTIPLOS DEFEITOS DE CORROSÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Civil da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Bacharel em Engenharia Civil.

Área de concentração: Estruturas.

Orientador: Profª. Drª. Juliana von Schmalz Torres.

Caruaru

2019

Catalogação na fonte:
Bibliotecária – Simone Xavier - CRB/4 - 1242

A474d Alves, Anderson Ferreira.
Desenvolvimento de um sistema computacional web para projeto baseado em confiabilidade de dutos sujeitos à múltiplos defeitos de corrosão. / Anderson Ferreira Alves. – 2019.
117 f. il. : 30 cm.

Orientadora: Juliana von Schmalz Torres.
Monografia (Trabalho de Conclusão de Curso) – Universidade Federal de Pernambuco, CAA, Engenharia Civil, 2019.
Inclui Referências.

1. Dutos e tubulações. 2. Confiabilidade (Engenharia). 3. Monte Carlo, Método de. 4. Redes neurais (Computação). 5. Android (Recurso eletrônico). 6. Compiladores (Programas de computador). I. Torres, Juliana von Schmalz (Orientadora). II. Título.

CDD 620 (23. ed.) UFPE (CAA 2019-065)

ANDERSON FERREIRA ALVES

DESENVOLVIMENTO DE UM SISTEMA COMPUTACIONAL WEB PARA PROJETO BASEADO EM CONFIABILIDADE DE DUTOS SUJEITOS À MÚLTIPLOS DEFEITOS DE CORROSÃO

Trabalho de Conclusão de Curso a ser apresentado ao Curso de Engenharia Civil do Centro Acadêmico do Agreste - CAA, da Universidade Federal de Pernambuco - UFPE, em cumprimento às exigências para a obtenção do grau de Bacharel em Engenharia Civil.

_____

Anderson Ferreira Alves

BANCA EXAMINADORA

A banca examinadora composta pelos professores abaixo, considera o candidato ANDERSON FERREIRA ALVES aprovado com nota_____.

Profª. Drª. Juliana von Schmalz Torres:_____
Universidade Federal de Pernambuco – UFPE (Orientadora)

Profª. Drª. Mariana Fernandes dos Santos Villela:_____
Universidade Federal de Pernambuco – UFPE (Avaliador)

Prof. Dr. Alessandro Romário Echevarria Antunes:_____
Universidade Federal de Pernambuco – UFPE (Avaliador)

Prof. Dr. Elder Alpes de Vasconcelos:_____
Universidade Federal de Pernambuco – UFPE (Coordenador da disciplina de TCC)

Caruaru, ____ de junho de 2019.

Dedico aos meus tios Lurdes e Valfrido, aos meus primos Edvaldo e Walter, e ao meu irmão Fabiano por me apoiarem, ajudarem e acreditarem em mim em todos os momentos.

AGRADECIMENTOS

Primeiramente agradeço à Deus pelo dom da vida e por tudo que tenho e o que conquistei até hoje, agradeço também à minha família, especialmente minha tia Lurdes e meu tio Valfrido, aos meus primos Edvaldo e Walter, e a meu irmão Fabiano, por sempre me proporcionarem uma infraestrutura familiar que me permitiu continuar com os meus estudos, além é claro de todo o apoio e ajuda quando eu mais precisava

Meus agradecimentos aos meus amigos e colegas da universidade que sempre estiveram dispostos a compartilhar conhecimentos e sanar dúvidas contribuindo desta forma para meu crescimento pessoal, além é claro dos momentos de descontração. Em especial agradeço a Kennendh, Felipe, Bruno, Matheus, Ayane, Hellen, Isabela, João Lucas, Kevin, Letônio, Claudiano, Alisson, Cláudio, Miguel e Daniel. Além de todos os outros que não foram mencionados, mas que tiveram grande importância ao longo da graduação.

Agradeço a todo o corpo docente da Universidade Federal de Pernambuco, que constituído por uma equipe competente de profissionais permitiu um bom aprendizado e base sólida dos principais conceitos da Engenharia Civil.

E por último, mas não menos importante, agradeço incondicionalmente à minha orientadora de TCC, Juliana, que sempre esteve disposta a escutar minhas ideias, e sempre estimulando aplicações do meu interesse por programação, além é claro pelas várias vezes em que me ajudou sanando dúvidas e abrindo novos horizontes com novas sugestões de pesquisas.

“Projetos Conjuntos tem mais chances de sucesso quando se beneficiam de ambos os lados”.

(Eurípedes)

RESUMO

O sistema dutoviário é um dos mais eficientes no transporte de cargas como petróleo e gás, isso se deve ao baixo custo operacional se comparado à outras metodologias e pela grande eficiência devido ao fato de operar de forma contínua. Porém esse sistema é suscetível à corrosão, que por sua vez diminui a capacidade resistente do duto, o que pode gerar ruptura acarretando diversos prejuízos. A corrosão gera defeitos ao longo do duto, estes defeitos podem interagir gerando uma perda ainda maior na resistência do mesmo. Por meio da análise de confiabilidade estrutural, é possível levar em consideração as incertezas das variáveis do problema, proporcionando um projeto mais confiável e econômico. No presente trabalho, é desenvolvido um sistema web para análise e projeto (cálculo da espessura ótima) utilizando confiabilidade estrutural em dutos sujeitos à múltiplos defeitos de corrosão. Este sistema é composto por um aplicativo Android e um *webservice* (aplicação web) que se comunica com o software Matlab, para que os métodos de confiabilidade sejam executados remotamente. Um ambiente em três dimensões para representar o duto 3d e seu defeitos foi implementado na Engine Unity, onde são criados jogos e ambientes tridimensionais, e integrado ao aplicativo desenvolvido neste trabalho, afim de validar a representação do modelo dos dados de forma visual. Além disso, no presente trabalho, foi criada uma linguagem de programação para que o sistema suportasse a escrita e resolução de outros tipos de problemas de confiabilidade estrutural. Tal como, especificar outra variável de projeto que não seja a espessura, ou até mesmo problemas que não tenham relação com dutos, como o projeto de vigas. A análise de confiabilidade foi feita utilizando-se dos métodos FORM, Monte Carlo, Monte Carlo com Esperança Condicionada, e Monte Carlo com Redes Neurais. Utilizou-se de formulações empíricas obtidas da norma BS7910(2005) para o cálculo da pressão de falha. A evolução do processo corrosivo ao longo do tempo foi estudada considerando o modelo linear de corrosão proposto por Ahammed.

Palavras-chave: Dutos. Corrosão. Confiabilidade Estrutural. Monte Carlo. FORM. Android.

ABSTRACT

The pipeline system is one of the most efficient in the transport of cargoes such as oil and gas, this is due to the low operational cost compared to other methodologies and the great efficiency due to the fact of operating continuously. However, this system is susceptible to corrosion, which in turn reduces the resistant capacity of the pipeline, which can generate rupture causing several damages. Corrosion causes defects along the duct, these defects can interact generating an even greater loss in the resistance of the same. Through the structural reliability analysis, it is possible to take into account the uncertainties of the problem variables, providing a more reliable and economical project. In the present paper, a web system is developed for analysis and design (calculation of the optimal thickness) using structural reliability in pipeline subject to multiple corrosion defects. This system consists of an Android application and a webservice (web application) that communicates with Matlab software so that reliability methods are executed remotely. A 3D environment to represent the 3d pipeline and its defects was implemented in Engine Unity, where games and three-dimensional environments are created, and integrated with the application developed in this paper, in order to validate the representation of the data model in a visual way. In addition, in the present paper, a programming language was created for the system to support the writing and resolution of other types of structural reliability problems. Like, specify another design variable other than thickness, or even problems that are unrelated to pipelines, such as beam design. The reliability analysis was done using the FORM, Monte Carlo, Monte Carlo with Conditioned Hope, and Monte Carlo with Neural Networks methods. Empirical formulations obtained from BS7910 (2005) were used to calculate the failure pressure. The evolution of the corrosive process over time was studied considering the linear corrosion model proposed by Ahammed.

Keyword: Pipeline. Corrosion. Reliability Structural. Monte Carlo. FORM. Android.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ASME	American Society of Mechanical Engineers
CPU	Central Processing Unit
EDT	Event Dispatching Thread
FORM	First Order Reliability Method
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
ILI	In-line Inspection
JSON	JavaScript Object Notation
MC	Monte Carlo
MCEC	Monte Carlo com Esperança Condicionada
MCRN	Monte Carlo com Redes Neurais
MVVM	Model-View-ViewModel
NNT	Neural Network Toolbox
REST	Representational State Transfer
ROA	Resource Oriented Architecture
SDK	Software Development Kit
SOAP	Simple Object Access Protocol
SORM	Second-Order Reliability Method
URI	Uniform Resource Identifiers
XML	Extensible Markup Language

LISTA DE SÍMBOLOS

σ_{rup}	Pressão de ruptura do duto
σ_{flow}	Tensão de fluxo
f_R	Fator de redução
α	Fator empírico relacionado a geometria da área corroída
A	Área da seção corroída
A_0	Área original da seção corroída
M	Fator de folias
l	Comprimento do defeito de corrosão
t	Espessura da parede do duto
d	Profundidade máxima do defeito
σ_{circ}	Pressão circunferencial no duto
P	Variável de projeto
D	Diâmetro externo do duto
P_{falha}	Pressão de falha ou de ruptura do duto
Φ_{circ}	Espaçamento circunferencial entre defeitos de corrosão
s	Espaçamento longitudinal entre defeitos de corrosão
l_{eq}	Comprimento equivalente de um conjunto de defeitos interagentes
n_d	Número de defeito de um grupo de defeitos interagente
l_i	Comprimento do i-ésimo defeito do grupo de defeitos interagente
s_i	i-ésimo espaçamento de um grupo de defeitos interagente alinhados longitudinalmente
d_{eq}	Profundidade equivalente de um conjunto de defeitos interagentes
d_i	Profundidade do i-ésimo defeito do grupo de defeitos interagente
d_0	Profundidade do defeito na última inspeção
R_d	Taxa de corrosão radial
T	Tempo da inspeção atual
T_0	Tempo da última inspeção
l_0	Comprimento do defeito na última inspeção
R_l	Taxa de corrosão longitudinal
$G(U)$	Função de falha

U	Vetor de variáveis aleatórias
pf	Probabilidade de falha
$f_U(U)$	Função da distribuição de probabilidade conjunta das variáveis aleatórias U
R	Resistencia ou pressão de falha do duto
S	Solicitação ou pressão atuante no duto
V	Vetor de variáveis aleatórias no espaço normais padrão estatisticamente Independentes (média = 0 e desvio padrão = 1).
β	Índice de confiabilidade
g(V)	Função de falha no espaço reduzido
Γ	Matriz inversa da matriz triangular inferior obtida da decomposição de Choleski da matriz dos coeficientes de correlação das variáveis aleatórias U
σ_{DP}	Matriz diagonal contendo os desvios padrões das variáveis aleatórias
m	Vetor com as medias das variáveis aleatórias
J	Jacobiano
L	matriz triangular inferior obtida da decomposição de Choleski da matriz dos coeficientes de correlação das variáveis aleatórias U
L_{ij}	Elemento da linha i e coluna j da matriz L
p	Número de variáveis aleatórias do problema
ρ^E_{ij}	Coeficiente de correlação equivalente entre as variáveis U_i e U_j.
ρ_{ij}	Coeficiente de correlação entre as variáveis U_i e U_j.
ϕ^{-1}	Inverso da função cumulativa normal padrão.
∇g(V)	Gradiente da função de falha no espaço reduzido
I_i	Fator de importância
α_i	Componente do vetor normal a superfície de falha no ponto de projeto correspondente a variável aleatória i.
N	Número de simulações no método de monte Carlo
X	Variável aleatória estatisticamente independente das outras e que possui maior dispersão.
F_X	Função de distribuição acumulada da variável X
y	Saída de um neurônio artificial

w_{ji}	Peso sináptico da sinapse i pertencente ao neurônio j
x_i	Entrada correspondente a sinapse i.
φ	Função ativação
v	Combinador linear do vetor de pesos sinápticos e do vetor de entradas
e_j	Erro gerado pelo neurônio j
d_j	Saída do conjunto de treinamento correspondente ao neurônio j
y_j	Saída do neurônio j
ε	Erro total gerado por uma saída da rede
w_{ji}	Peso sináptico da sinapse i pertencente ao neurônio j
η	Taxa aprendizado da rede neural
δ_j	Gradiente local
β^{alvo}	Índice de confiabilidade alvo
P^k	Valor da variável de projeto na iteração k

SUMÁRIO

1 INTRODUÇÃO

Dutos são construídos de acordo com normas internacionais de segurança para o transporte de substâncias explica Toro (2014). O transporte dutoviário tem se mostrado eficiente como meio para movimentação de líquidos como petróleo e gás natural. Esse fato não é por acaso, afinal este tipo de transporte permite uma grande movimentação de massas, operação contínua, transporte para longas distância, não requer embalagens e apresenta um gasto muito inferior se comparado ao transporte destes recursos utilizando meios de transporte rodoviários e ferroviários por exemplo (FILHO, 2018).

Mas uma coisa que também é certa, é que quando este tipo de transporte apresenta falhas, como uma tubulação rompida, os prejuízos são desastrosos. Primeiramente tem-se os danos financeiros pelo fato de ter perdido uma quantidade considerável de material como óleo diesel ou petróleo no vazamento. Existe também, as indenizações pelos impactos ambientais que podem poluir e destruir vários ecossistemas aquáticos, e por último temos os danos às vidas humanas.

Muitos dos acidentes envolvendo dutos estão relacionados a tubulação corroída, já que as tubulações são de ferro ou aço. O problema é que a rede de dutos, muitas vezes é enterrada e também é muito extensa o que acaba dificultando a inspeção em todos os locais, além é claro dos reparos quando necessários. Uma das metodologias adotadas é fazer a análise de confiabilidade estrutural em dutos existentes ou o projeto em dutos que serão implantados, de tal forma que se programe com exatidão o momento certo de inspecionar cada tubulação. Isto por sua vez previne acidentes e gera economia de recursos.

Diante destas premissas, e também o fato da popularização dos dispositivos móveis como *smartphones* e *tablets*, e do acesso fácil à internet, principalmente a móvel, no presente trabalho foi implementada uma arquitetura composta por um aplicativo *Android* que recebe as informações do duto e de seus defeitos de corrosão e envia esses dados pela internet para um servidor, proposto neste trabalho, que possui acesso as rotinas do *software* Matlab. O servidor com muito mais poder de processamento que o dispositivo móvel, resolve o problema mais rápido e devolve a resposta para o aplicativo. Desta forma, as análises podem ser feitas no próprio local de inspeção, agilizando a tomada de decisão.

Além disso, uma linguagem de programação foi desenvolvida, a fim de facilitar a resolução numérica de problemas de confiabilidade estrutural, diminuindo as linhas de código. Desta forma, apenas os detalhes essenciais do problema como variáveis estatísticas, paramétricas e de projeto, além da definição da função de falha, são necessários, utilizando assim poucas linhas de programação. A linguagem é compilada para a linguagem do *software* Matlab, chamando desta forma rotinas programadas no mesmo. Isso por sua vez, permite uma rápida adaptação do sistema, como por exemplo adotar o diâmetro como variável de projeto ou mesmo definir a função de falha em função de outras normas, além da definição padrão referente à norma BS7910 (2005).

1.1 JUSTIFICATIVA

O transporte dutoviário tem se mostrado um sistema bastante eficiente para o transporte de produtos como petróleo e gás natural. Algumas de suas vantagens incluem transporte de grandes quantidades de carga, operação 24 horas diárias por longas distâncias, fácil implementação, bem como baixo custo operacional de transporte e energia (INSTITUTO BRASIL LOGÍSTICO, 2018).

No Brasil, este tipo de transporte ainda é pequeno, representando apenas cerca de 3% do transporte destes produtos, enquanto que apenas o transporte rodoviário representa 65%, e o ferroviário 19,5%, os demais juntos representam menos que 15% (FILHO, 2018). Porém este tipo de transporte é adequado por reduzir custos e por dificultar ações de roubo. Apesar de todas estas vantagens, as tubulações utilizadas nestes sistemas sofrem ação da umidade e seu material constituinte acaba corroendo. Este efeito de corrosão diminui a espessura da seção do duto, o que por sua vez acaba diminuindo sua capacidade resistente. Afim de se evitar um colapso, que poderia provocar prejuízos milionários e desastres ambientais de grandes proporções, são feitas inspeções e substituição de dutos.

O problema é que, se essa inspeção não é feita no tempo adequado, o mesmo vale para substituição do duto, acaba gerando gastos desnecessários. O presente trabalho contempla o projeto da espessura ótima de um duto, de tal forma que se tenha o máximo de aproveitamento do duto ao longo do tempo, em termos de vida útil, e que serviços de manutenção e troca sejam feitos quando realmente for necessário.

1.2 MOTIVAÇÃO

O interesse no assunto surgiu durante as atividades do projeto de iniciação cientifica (FACEPE) “Projeto baseado em confiabilidade de dutos sujeitos à múltiplos defeitos de corrosão”, orientado pela professora doutora Juliana V. S. Torres no período de agosto de 2016 a julho de 2017. No projeto de pesquisa foi projetada a espessura ótima de dutos sujeitos à múltiplos defeitos de corrosão de dimensões iguais alinhados longitudinalmente e igualmente espaçados utilizando o método FORM e as formulações empíricas para a pressão de falha proposta pela norma BS7910 (2005).

Dando continuidade a esse projeto, decidiu-se focar na construção de um aplicativo na plataforma Android, no estudo de redes neurais artificiais, na construção de uma infraestrutura em rede para envio dos dados para um servidor remoto, responsável por resolver problemas de análise e projeto baseado em confiabilidade, e no desenvolvimento de uma linguagem de programação para problemas estruturais de confiabilidade.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

- ✓ Implementar um sistema computacional web para resolução de problemas de confiabilidade estrutural focados em dutos sujeitos à múltiplos defeitos de corrosão.

1.3.2 Objetivos Específicos

- ✓ Implementar os métodos de confiabilidade estrutural (FORM, Monte Carlo, Monte Carlo com Esperança Condicionada e Monte Carlo com Redes Neurais) em dutos sujeitos à múltiplos defeitos de mesmas dimensões e igualmente espaçados alinhados longitudinalmente no *software* Matlab;
- ✓ Implementar o *webservice* em Java, responsável por expor as rotinas de confiabilidade estrutural implementadas no Matlab para um aplicativo Android remoto;

- ✓ Implementar um aplicativo Android para a captação dos dados do problema e envio para o servidor remoto;
- ✓ Implementar a representação tridimensional do duto e dos seus defeitos na Engine Unity e integra-la com o aplicativo Android;
- ✓ Implementar um compilador para a linguagem AnderScript, idealizada pelo autor do presente trabalho, para resolver problemas de confiabilidade de forma mais simplificada, e integrar este módulo com o aplicativo, a fim de expandir as possibilidades do sistema web.

2 REVISÃO BIBLIOGRÁFICA

Em 1994, Sagrilo apresentou o desenvolvimento de uma ferramenta computacional para a análise de confiabilidade estrutural em grandes estruturas, por meio dos métodos analíticos FORM e SORM, utilizando uma técnica adaptativa de superfície de resposta para o cálculo dos gradientes da função de falha.

Em 1998, Ahammed propõe um modelo de corrosão linear a longo prazo para dutos, o estudo chega a analisar a exposição à corrosão de um duto em um período de sessenta anos. Ahammed adotou uma abordagem probabilística e as variáveis aleatórias adotadas (profundidade do defeito, diâmetro do duto, comprimento do defeito, pressão interna no duto, taxa de corrosão radial, taxa de corrosão longitudinal, tensão última do material e espessura da parede do duto) possuem distribuições normais e não-normais.

Em 2004, Barbosa utilizou uma rede neural treinada para diminuir o número de simulações no método de Monte Carlo, com o objetivo de reduzir o custo computacional na análise de confiabilidade. Neste trabalho são utilizadas redes multilayer Perceptron juntamente com o método de Monte Carlo, utilizando 3 metodologias diferentes. Na primeira é utilizada a técnica de esperança condicionada, a segunda representa a função de falha de forma implícita, e na última a rede neural é utilizada em todo o processo de análise de confiabilidade.

Em 2008, Verzenhassi desenvolve um programa computacional em Fortran para encontra o coeficiente de segurança parcial ótimo que minimiza o custo esperado total de sistemas estruturais. Tal programa está acoplado a um programa de confiabilidade estrutural desenvolvido pela EESC/USP e a um programa comercial de

análise por elementos finitos. O trabalho inclui alguns estudos de casos como análise de uma torre de telefonia sujeita a cargas de vento. Verzenhassi encontra relações entre a confiabilidade ótima da estrutura e as consequências, bem como os custos de falha.

Já em 2009, Torres traz uma análise de dutos com corrosão, aplicando o sistema de acoplamento de programas computacionais para a análise de confiabilidade. Analisando dutos com defeitos isolados e com múltiplos defeitos. Demonstra uma metodologia para verificação da segurança e dimensionamento baseado em confiabilidade de tais estruturas. Define-se a função de falha em termos da pressão interna aplicada e da pressão de falha, sendo esta calculada considerando modelos empíricos, o Método dos Elementos Finitos (MEF), e o método de superfície resposta. Para este último utiliza aproximação do tipo ajuste de dados via Krigagem. Apresenta o dimensionamento baseado em confiabilidade do duto, buscando garantir a conservação do nível de segurança por um determinado período de tempo.

Em 2011, Xu e Cheng analisaram a confiabilidade de dutos com defeitos de corrosão através de um modelo de elementos finitos comparando diferentes teores de aço. A pressão de falha para efeito de comparação foi obtida por três modelos empíricos (B31G, B31G Modificada e DNV). Foi observado que a pressão de falha é reduzida com o aumento da profundidade do defeito e com a diminuição do teor de aço.

Em 2014, Toro avalia a precisão de alguns modelos empíricos para a determinação da pressão de falha de dutos sujeitos à corrosão. Os métodos avaliados foram: ASME B31G, ASME B31G modificado, DNV RP F101 e PCORRC. O estudo foi feito a partir de base de dados sobre dutos reais e artificiais da literatura somando mais de 400 resultados de ensaios de ruptura. Toro desenvolve o conceito de variável do modelo, responsável por quantificar o erro acumulado pelos diversos modelos. Por meio da aplicação da variável “erro do modelo” foram feitas análises de confiabilidade utilizando o método FORM, obtendo como saída o índice de confiabilidade e a probabilidade de falha. O estudo analisou a evolução da pressão de falha à medida que se aumentava a profundidade do defeito, além disso foram identificadas as variáveis mais importantes do problema por meio de medida de sensibilidade.

Ainda em 2014, Gomes e Beck propõem um novo modelo de crescimento de corrosão em dutos enterrados por meio de polinômios de caos. A espessura ótima do duto, o tempo da primeira inspeção e o tempo entre as inspeções sucessivas são

consideradas como variáveis de projeto. Gomes e Beck se concentram em minimizar os custos totais esperados na vida útil, que incluem custos de construção, inspeções e reparo, além dos custos esperados com possíveis falhas. Os números esperados de falhas, reparos e substituições são avaliados por uma análise probabilística usando amostragem hipercubo latina.

Em 2015, Wan, Yajima, et al. descrevem uma abordagem de agrupamento de defeitos de corrosão, baseado no campo aleatório de Markov, para extrair potenciais segmentos homogêneos a partir de uma grande extensão do duto de uma estrutura de dutos com propriedades heterogêneas no solo. Um exemplo envolvendo um intervalo de duto de 110 km é empregado para ilustrar a implementação da abordagem de clustering. Por fim Wan, Yajima, et al. concluem que o processo de propagação da corrosão externa em um gasoduto enterrado depende da sua posição ao longo do duto e está altamente relacionada ao ambiente do solo.

Em 2016, Leira, Naess e Naess abordam a análise de confiabilidade de dutos de corrosão considerando também os efeitos do sistema na análise de confiabilidade. A análise é realizada pela utilização de métodos de simulação de Monte Carlo aprimorados, pois segundo o estudo, são muito mais eficientes para a quantificação da confiabilidade do sistema no caso de múltiplos componentes com níveis de correlação arbitrários. Os exemplos que são estudados compreendem sistemas com defeitos de corrosão, independentes e correlacionados.

Em 2017, Tee e Pesinis avaliam a confiabilidade dependente do tempo, de um gás natural subterrâneo corroído, pertencente a um sistema de oleoduto ao longo de sua vida útil. A análise de confiabilidade é baseada em segmentos, em oposição à baseada em defeitos e o segmento de duto é examinado em relação à corrosão externa por perda de metal. O processo não-homogêneo de Poisson e um modelo de lei de potência empírica são empregados para geração de defeitos de corrosão ao longo do tempo e para o crescimento dos defeitos, respectivamente. A probabilidade de falha dependente do tempo é avaliada empregando a função de estado limite para falhas sob pressão interna. A Pressão interna é modelado usando um método baseado em processos de ondas quadradas de Poisson. A seguir, o estudo usa a análise supracitada, em conjunto com um modelo heurístico, a fim de investigar a influência de reparos imperfeitos na previsão da confiabilidade do sistema de dutos.

Ainda em 2017, Zhou, Xiang e Hong empregam os processos gaussianos gama e inverso para modelar o crescimento dos defeitos, enquanto a dependência entre os

crescimentos de diferentes defeitos é caracterizada usando a abordagem de cópula gaussiana e soma de processo estocástico. No estudo conclui-se que o crescimento de defeitos na análise de confiabilidade, é adequado para o crescimento de corrosão relativamente lento que é típico para tubulações enterradas.

Ainda em 2017, o comportamento de falha de dutos com defeitos de corrosão interagentes foi estudado por Xu, Li, et al. que utilizaram um método de elementos finitos para este fim, e então uma solução foi proposta para prever a pressão usando uma rede neural artificial.

Ainda em 2017, Zemati, Chelloudj e Amirat apresentam uma metodologia com o objetivo de contribuir para a avaliação da vida remanescente de dutos usando análise de confiabilidade, a fim de correlacionar o comprimento e a profundidade dos defeitos de corrosão. Os dois parâmetros através de um parâmetro comum, o índice de confiabilidade, são investigados através de quatro modelos de engenharia bem estabelecidos; Irwin, SINTAP, ASME B31G e ASME B31G Modificado. Em seguida, os resultados fornecidos por cada um dos quatro modelos são coordenados, afim de auxiliar na tomada de decisão para fornecer respostas realistas para substituir e/ou reparar um duto sujeito à pressão interna.

Em 2019, Amaya-Gómez, Sánchez-Silva, et al. fazem uma revisão das funções reconhecida de estado limite para oleodutos corroídos, discutindo suas suposições e aplicabilidades. O estudo enfoca na pressão de falha em dutos de petróleo e gás. Portanto, uma comparação completa é apresentada com base em critérios de falha, dimensões de defeito aceitáveis, probabilidade de falha e previsão de erros com base em testes de ruptura experimentais e numéricos. O objetivo do estudo consiste em avaliar o nível de conservadorismo de cada modelo simplificado, dependendo da tenacidade do material e da taxa de corrosão, afim de selecionar modelos de confiabilidade em dutos corroídos para futuras estratégias de intervenção.

Ainda em 2019, Sun e Cheng desenvolvem um modelo tridimensional para investigar a interação mecânico-eletroquímica de múltiplos defeitos de corrosão alinhados longitudinalmente em uma tubulação de aço X46 enterrada. Uma técnica de acoplamento de campo multi-física foi empregada para derivar as distribuições de tensão, deformação, potencial de corrosão e densidade de corrente anódica nos defeitos. Sun e Cheng explicam que para múltiplos defeitos de corrosão, existe um espaçamento crítico, abaixo do qual existe uma interação entre eles. Concluem que o espaçamento máximo de interação aumenta à medida que o comprimento do defeito

aumenta. Além disso, à medida que o espaçamento do defeito diminui, há uma forte interação entre eles, resultando em uma alta tensão plástica nos defeitos. Por fim, Sun e Cheng comentam que a interação entre múltiplos defeitos de corrosão, existe não apenas no campo de tensão mecânica, mas também no campo de corrosão eletroquímica. E que um aumento do comprimento do defeito aumenta a tensão local nos defeitos, deslocando o potencial de corrosão negativamente, e consequentemente aumentando a densidade da corrente anódica tanto no defeito quanto na área adjacente.

No mesmo ano, Mishra, Keshavarzzadeh e Noshadravan apresentam uma nova abordagem para o gerenciamento da vida útil baseado em confiabilidade de dutos enterrados sujeitos a corrosão. Primeiro, um modelo probabilístico para a evolução temporal do crescimento da corrosão é construído a partir de dados disponíveis usando o formalismo do polinômio de caos. O modelo é usado para propagação sistemática da incerteza subjacente nas funções do estado limite e da confiabilidade na vida útil. Em seguida, é proposta uma estratégia de otimização computacionalmente eficiente e precisa usando polinômios substituto a fim de resolver a otimização estocástica associada ao gerenciamento da vida útil de tubulações enterradas. O método proposto facilita a otimização do agendamento de manutenção para alcançar o custo mínimo esperado da vida útil.

Ainda em 2019, Wang, Yajima e Castaneda estabelecem um modelo de crescimento de corrosão estocástico para estruturas de tubulações subterrâneas. O modelo foi desenvolvido tendo em mente que a evolução dos danos causados pela corrosão localizada é dependente do tempo, após 3 estágios: estágio de ativação (nucleação), estágio de crescimento (propagação), e estágio de estado estacionário (passivação). A correlação temporal da evolução do defeito pode ser bem representada por um processo de ponte Brownian geométrico. Duas aplicações são ilustradas: a primeira visa prever a evolução da função de densidade de probabilidade da profundidade do defeito de corrosão, e a segunda avalia a confiabilidade de uma estrutura de dutos.

Por fim ainda em 2019, Seghier, Keshtegar, et al. avaliam a probabilidade de falha de tubulações corroídas de aço X60. Para este problema, a função de desempenho de falha por corrosão é desenvolvida usando um modelo M5Tree baseado em calibração com banco de dados de testes de falhas reais. A análise estatística dos dados do relatório de ILI é realizada para melhor modelagem da

geometria dos defeitos de corrosão (ou seja, comprimento e profundidade dos defeitos), onde diferentes distribuições de probabilidade (Normal, Lognormal, Frechet, Gumbel, Weibull) foram testados. Além disso, o efeito da geometria dos defeitos na probabilidade de falha foi investigado para essas diferentes distribuições de probabilidade. Em seguida, a influência das distribuições na análise de confiabilidade também foi ilustrada. Os resultados indicaram que aumentos na profundidade dos defeitos reduzem fortemente os níveis de segurança do problema, e que a falta de seleção de distribuições nos defeitos poderia levar a resultados conservadores.

3 REFERENCIAL TEÓRICO

Nas próximas sessões serão apresentados alguns conceitos que foram utilizados no presente trabalho, para a implementação do sistema computacional web desenvolvido. A seção 3.1 descreve o cálculo da pressão de falha em duto contendo defeitos de corrosão. Na seção 3.2 é apresentada a confiabilidade estrutural, bem como dois métodos de confiabilidade bastante conhecidos na literatura: o método FORM e o método de Monte Carlo. Em seguida a seção 3.3 é apresentada uma revisão do método de Newton-Raphson, utilizado para encontrar a raiz de uma função, utilizado neste trabalho para encontrar o valor da variável de projeto (espessura do duto).

A seção 3.4 descreve os fundamentos da teoria sobre redes neurais, muito utilizadas para aproximar funções continuas de qualquer ordem. As seções 3.5 e 3.6 descrevem os fundamentos e funcionamento da plataforma Android, para criação de aplicativos para dispositivos móveis, e dos *webservices* RESTful, utilizados para fornecer acesso a recursos remotos, conforme será discutido posteriormente. Em seguida, temos a seção 3.7, que apresenta a teoria para a criação e compiladores, utilizados para converter um programa em uma linguagem de programação para outra linguagem, sendo aplico no presente trabalho, para converter código da linguagem AnderScript, crida no presente trabalho, para a linguagem do software Matlab.

Por último, na seção 3.8, é apresentado o conceito de thread, que permitem a existência de computação paralela dentro de programas, sendo utilizada no presente trabalho para permitir que múltiplos usuários tenham acesso simultâneo ás rotinas do Matlab de forma remota.

3.1 CÁLCULO DA PRESSÃO DE FALHA

Os dutos operam transportando fluidos, portanto estão submetidos à pressão interna. Caso esta pressão, ultrapasse uma pressão limite, chamado de pressão de falha, a tubulação rompe, gerando vários prejuízos. As formulações empíricas para o cálculo da pressão de falha devem levar em consideração as propriedades mecânicas e geométricas do duto, bem como a geometria dos defeitos de corrosão (TORO, 2014). Segundo Toro (2014) os modelos semi-empíricos são baseados na mecânica da fratura e em ensaios experimentais. A formulação básica dos modelos é baseada nos critérios da equação NG-18 Surface Flaw Equation. Basicamente a NG-18 descreve uma relação entre a tensão de fluxo no duto pressurizado e o comprimento do defeito. Esta relação está representada na equação (1):

$$\sigma_{rup} = \sigma_{flow} * f_R \tag{1}$$

O fator de redução f_R é definido pela equação (2):

$$f_R = \frac{1 - \alpha * \frac{A}{A_0}}{1 - \alpha * \left(\frac{A}{A_0}\right) * M^{-1}} \tag{2}$$

A área corroída original A_0 = l*t, leva em consideração toda a espessura do duto, enquanto que a área corroída é A = l*d, considera apenas a profundidade do defeito. A tensão circunferência é obtida pela relação da pressão interna do duto, pela área da seção transversal da parede, a equação (3) ilustra essa relação.

$$\sigma_{circ} = \frac{PD}{2t} \tag{3}$$

No estado limite considera-se que a pressão de falha é igual a pressão circunferencial, desta forma a pressão de falha pode ser expressa pela equação (4).

$$\sigma_{rup} = \frac{P_{falha} D}{2t} \tag{4}$$

Substituindo a equação (4) na equação (1) e isolando P_{falha}, obtém-se a equação (5):

$$P_{falha} = \frac{2t\sigma_{flow}}{D} f_R \tag{5}$$

Substituindo a equação (2) na equação (5) determina-se a pressão de falha do duto para um defeito de corrosão, expressa na equação (6):

$$P_{falha} = \frac{2t\sigma_{flow}}{D} * \frac{1 - \alpha * \frac{d}{t}}{1 - \alpha * \left(\frac{d}{t}\right) * M^{-1}} \tag{6}$$

O fator de dilatação M é um parâmetro adimensional proporcional à relação entre o comprimento pela espessura do tubo e do diâmetro externo. Analisando a equação (6), pode-se observar que realmente a pressão de falha depende das propriedades mecânicas do duto, e da geometria deste e dos defeitos de corrosão.

As diferentes normas surgem variando os valores de α e do fator de dilatação M. Estas normas incluem: B31G, B31G modificada, PCORRC ou Battelle, DNV RP F101 e BS7910.

3.1.1 Norma BS7910 e Defeitos Interagentes

Como dito anteriormente, as diferentes normas sugerem valores diferentes para α, e expressões diferentes para o fator de dilatação M. Para a norma BS7910, as expressões para o cálculo da pressão de falha e para o fator de dilatação, estão representadas respectivamente nas equações (7) e (8), (BS7910, 2005):

$$P = \frac{2t\sigma_{flow}}{D - t} * \frac{1 - \frac{d}{t}}{1 - \frac{d}{t} * \frac{1}{M}} \tag{7}$$

$$M = \left(1 + 0.31\left(\frac{l}{\sqrt{D.t}}\right)^2\right)^{\frac{1}{2}} \quad (8)$$

Porém, em problemas reais, existem múltiplos defeitos de corrosão em um duto. Este conjunto de defeitos de corrosão, tem potencial para gerar uma pressão de falha maior que qualquer um dos defeitos isolados. Isso ocorre pelo fato de suas linhas de influência se sobreporem aumentando ainda mais sua influência, o que diminui ainda mais a resistência do duto.

A norma BS7910 (2005) permite calcular pressão de falha do duto, para múltiplos defeitos interagentes, utilizando a mesma expressão para o cálculo da pressão de falha para um único defeito, porém deve-se calcular as características do defeito equivalente para este conjunto de defeitos interagentes. Desta forma são calculados a profundidade e o comprimento equivalentes como dados de entrada para a equação (7).

Para que dois defeitos adjacentes interajam entre si, é necessário que os mesmos cumpram os seguintes critérios, que estão representados graficamente na Figura 1:

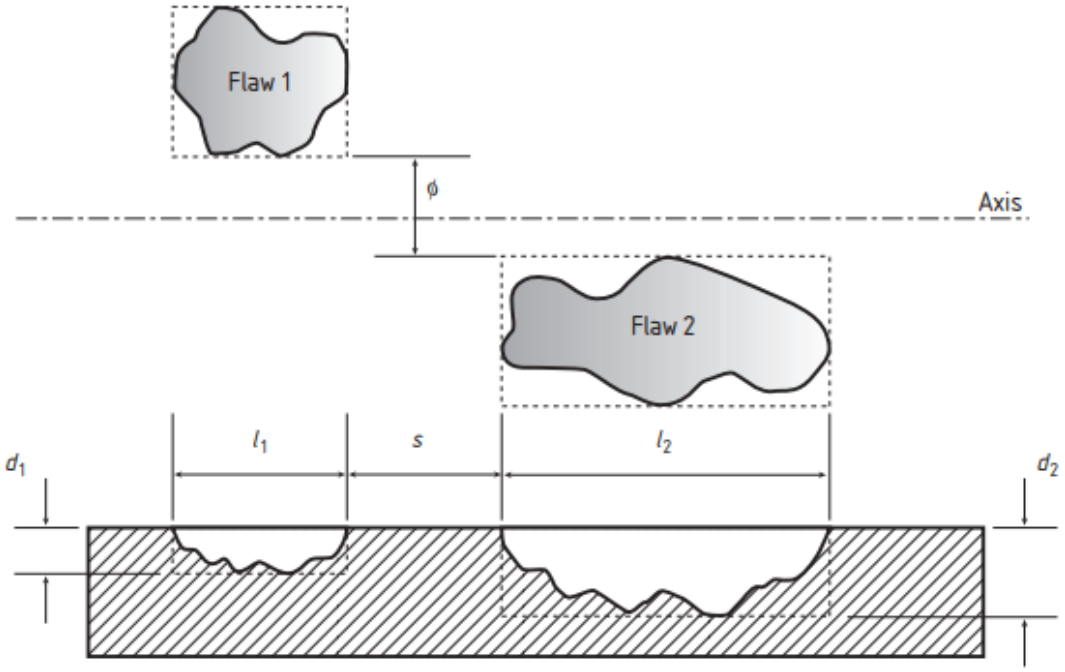
a) Ambos os defeitos devem possuir profundidade máxima de no mínimo 20% da espessura do duto;
b) O espaçamento circunferencial φ_{circ}, entre os defeitos não pode exceder o valor dado pela equação (9):

$$\varphi_{circ} < 360\frac{3}{\pi}\sqrt{\frac{t}{D}} \quad (9)$$

c) O espaçamento longitudinal s, entre os defeitos não pode exceder o valor dado pela equação (10):

$$s < 2\sqrt{Dt} \quad (10)$$

Figura 1 – Dimensões associadas à interação entre falhas

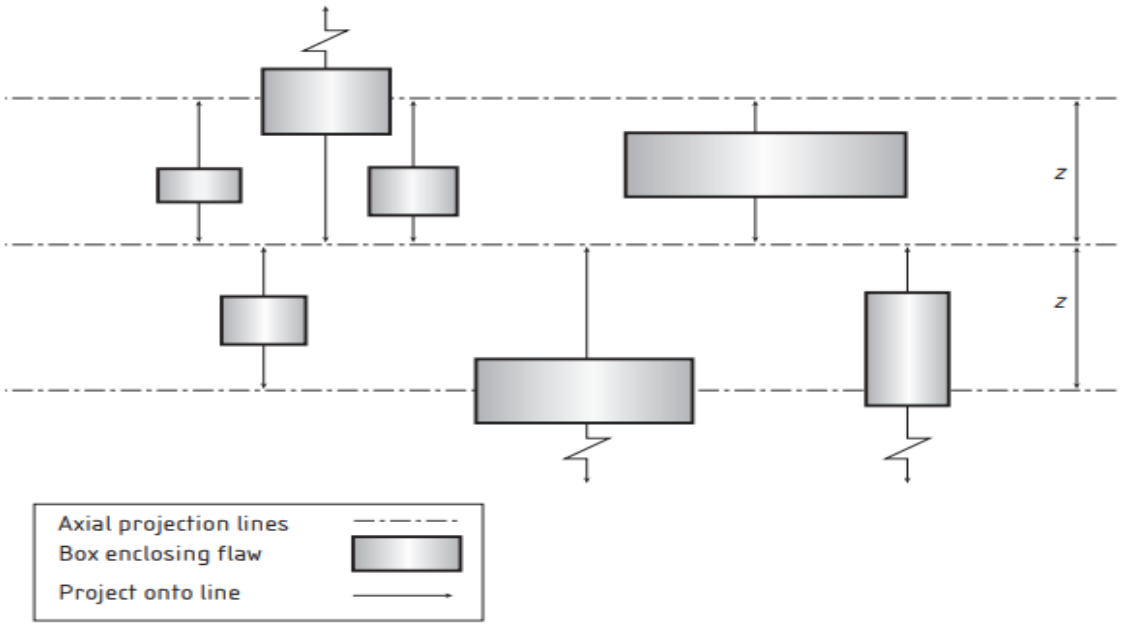


FONTE: (BS7910, 2005)

De acordo com a norma (BS7910, 2005), os procedimentos para estimar a pressão de falha, para múltiplos defeitos interagentes, são os seguintes:

1) Para as regiões onde há perda geral de metal, ou seja, menos de 10% da espessura do duto, deve ser considerada a própria espessura do duto para o cálculo da pressão de falha;
2) A seção corroída seve ser dividida em seções longitudinais de comprimento mínimo de $5\sqrt{Dt}$, com sobreposição mínima de $2.5\sqrt{Dt}$. Os passos 3) a 10) devem ser repetidos para todas essas seções, afim de avaliar todas as possíveis interações;
3) Construir uma série de linhas de projeções longitudinais ao longo da circunferência do duto com espaçamento circunferencial dado pela equação (9), conforme representado na Figura 2;

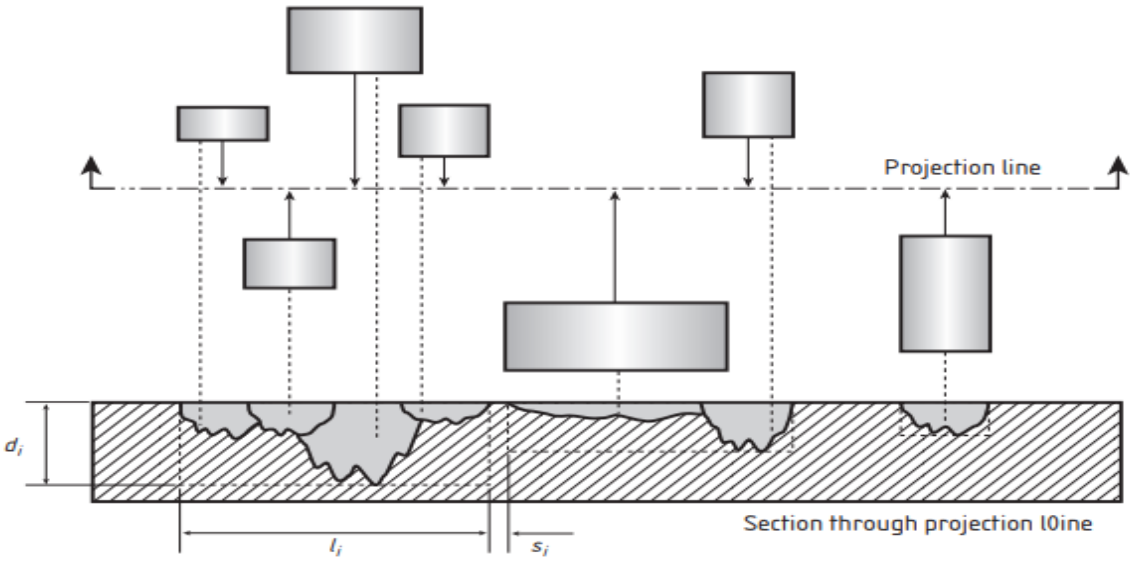
Figura 2 – Projeção circunferencial entre falhas interagentes



FONTE: (BS7910, 2005)

4) Para cada linha de projeção considerar o intervalo +-φ_{circ}, projetar nessa linha todos os defeitos que estiverem dentro do intervalo;
5) Onde as falhas se sobrepõem, as mesmas devem ser combinadas para formarem uma única falha de comprimento igual ao comprimento combinado e profundidade igual à profundidade máxima, conforme ilustrado na Figura 3;

Figura 3 – Projeção e sobreposição de falhas interagentes

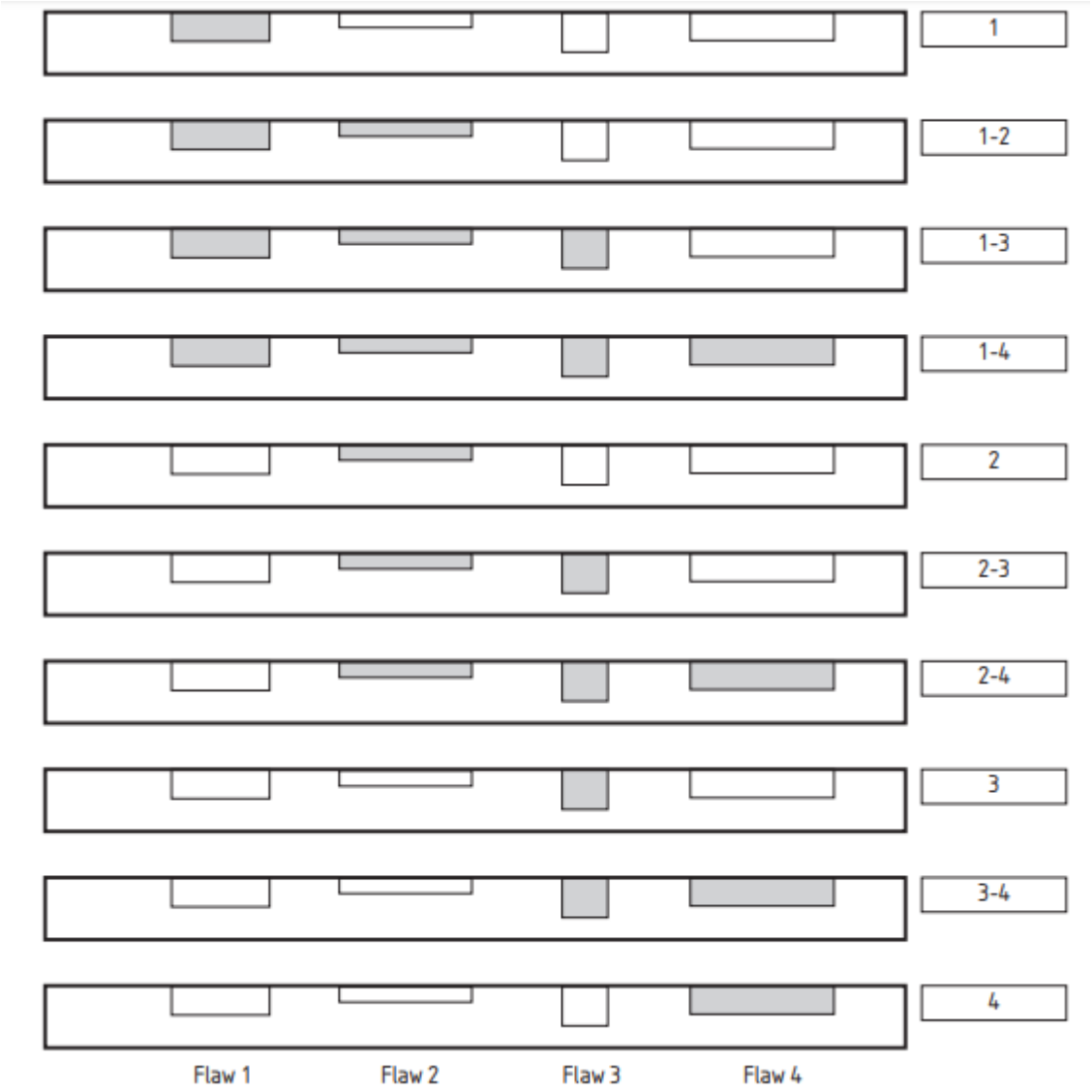


FONTE: (BS7910, 2005)

6) Tratar cada falha ou falha combinada como uma única falha, usar essas falhas resultantes e calcular a pressão de falha para cada uma delas;
7) Calcular o comprimento combinados de todas as combinações de falha possíveis usando a equação (11), conforme demonstrado na Figura 4;

$$l_{eq} = l_{n_d} + \sum_{i=1}^{i=n_d-1} (l_i + s_i) \qquad (11)$$

Figura 4 – Exemplo de agrupamentos de falhas adjacentes



FONTE: (BS7910, 2005)

8) Calcular a profundidade equivalente de todas as combinações possíveis usando a equação (12);

$$d_{eq} = \frac{\sum_{i=1}^{i=n_d} d_i l_i}{l_{eq}} \quad (12)$$

9) Calcular a pressão de falha usando a equação (7) para cada uma das combinações considerando o comprimento e profundidade equivalentes para cada combinação;

10) A pressão de falha da linha de projeção atual é tomada como a menor das pressões individuais e das combinações.

A pressão de trabalho segura é tomada como sendo a menor de todas as projeções circunferências.

3.1.2 Modelo linear de corrosão

A corrosão é uma espécie de processo decorrente da degradação de um material, na maioria das vezes metal, por uma interação química ou eletroquímica com o meio no qual está inserido (GENTIL, 1996). Desta forma as condições climáticas da região, as características do solo, caso o duto seja enterrado, possuem forte influência na velocidade com que o processo corrosivo se desenvolve com o tempo. Para Toro (2014, p. 21) “A corrosão é um dos principais mecanismos de falha em dutos enterrados, tornando necessária a reparação ou até a substituição de trechos de dutos”.

O problema é que a maioria das tubulações é feita de ferro ou aço e por isso estão sujeitas a deterioração por corrosão. O processo corrosivo apresenta-se por meio de falhas denominadas defeitos de corrosão. Estes defeitos, crescem com o tempo fazendo com que a espessura do duto diminua, acarretando maiores chances de a tubulação romper, o que pode causar danos econômicos, ambientais e humanos.

Os dutos são projetados ou analisados levando em consideração o tempo da última inspeção e o tempo atual (inspeção futura na simulação), ou seja, existe um período de tempo em que a corrosão muda seus parâmetros. Na verdade, o projeto de dutos é feito visando uma data futura de inspeção, e o duto tem que resistir até

esta data limite. Isto é feito para minimizar o número de inspeções, já que as mesmas demandam o gasto de recursos financeiros, que por sua vez envolvem mão-de-obra e equipamentos.

Para se programar adequadamente o número de inspeções, bem como prever possíveis falhas na tubulação, é preciso conhecer como o processo corrosivo evolui ao longo do tempo. Ahammed (1998) propôs um modelo linear de corrosão ao longo do tempo, o que facilita sua implementação. Neste modelo a taxa de corrosão é constante, e os crescimentos da profundidade e comprimento do defeito, são lineares. As expressões para os crescimentos do comprimento e profundidade do defeito, estão representadas respectivamente nas equações (13) e (14).

$$d = d_0 + R_D(T - T_0) \tag{13}$$

$$l = l_0 + R_L(T - T_0) \tag{14}$$

3.2 ANÁLISE DE CONFIABILIDADE ESTRUTURAL

Desde o seu surgimento, a computação tem evoluído muito rapidamente, isso permitiu que recursos como computação paralela, computação em nuvem, sistemas distribuídos, enormes bases de dados e o desenvolvimento da inteligência artificial se tornassem cada vez mais acessíveis ao usuário. Além disso, o desenvolvimento em software é cada vez mais crescente, novos métodos numéricos são formulados, novas arquiteturas são adotadas, plataformas inteiras são criadas e softwares robustos estão cada vez mais acessíveis.

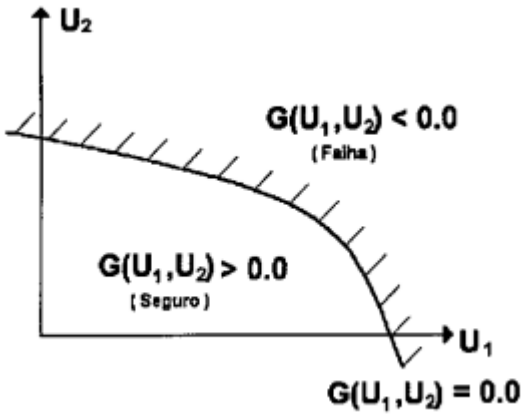
Com todos estes recursos em mãos, executar uma análise ou projeto de uma estrutura inteira pode demorar apenas alguns minutos. Apesar desta facilidade com o fator tempo, e consequentemente a possibilidade de fazer várias simulações antes de sua construção, a estrutura pode sim apresentar probabilidade de vir ao colapso, mesmo em um bom projeto. Isso ocorre porque as variáveis do problema sejam elas propriedades mecânicas, geométricas ou dados sobre as forças atuantes na estrutura, apresentam imprecisões em suas medidas. Estas imprecisões se propagam tornando o modelo mais distante da realidade.

Para mensurar essa chance de colapso da estrutura, existe a confiabilidade estrutural que considera que nenhuma estrutura é completamente segura, todas apresentam ainda que pequena uma probabilidade de vir a colapsar (SAGRILO, 1994). Para diminuir e mensurar esta probabilidade de falha, a confiabilidade estrutural utiliza de informações estatísticas como média, desvio padrão e covariância, além de conceitos da estatística como variáveis aleatórias e distribuição de probabilidade.

“A confiabilidade estrutural é uma ferramenta adicional que permite ao engenheiro estrutural quantificar as incertezas nas variáveis do seu projeto e auxilia-lo na tomada de decisões com mais segurança” (SAGRILO, 1994). Através da análise de confiabilidade é possível por exemplo calcular a probabilidade de falha de uma tubulação de petróleo e através disto, programar de forma otimizada a inspeção, reparo e troca de material, o que por sua vez diminui os custos da companhia envolvida.

A confiabilidade estrutural se baseia em uma função de estado limite último, conhecida como função de falha G(U), sendo que U = (U_1, U_2,...,U_n) é o conjunto das variáveis aleatórias do problema. A superfície de falha ocorre onde G(U) = 0, e a mesma divide o domínio seguro G(U) > 0 do domínio de falha G(U) < 0. Um exemplo desta superfície é ilustrado na Figura 5.

Figura 5 – Exemplificação da função de falha



FONTE: (SAGRILO, 1994)

Para a confiabilidade estrutural, o importante é o cálculo da probabilidade de falha pf, isto é, a probabilidade de a função de falha assumir valores que estejam

dentro do domínio de falha. Esta probabilidade é expressa de acordo com a equação (15):

$$pf = P[G(U) \leq 0] \tag{15}$$

Conforme Sagrilo (1994) demonstra, a probabilidade de falha pode ser reescrita em função da distribuição de probabilidade conjunta das variáveis aleatórias do problema, e integrada sobre todo o domínio de falha, conforme representado na equação (16):

$$pf = \int_F fu(U)\, du \tag{16}$$

A função de falha, para problemas de dutos, é geralmente descrita de acordo com a equação (17):

$$G(U) = R - S \tag{17}$$

A resistência R neste caso é a pressão de falha do duto, discutida anteriormente na seção 3.1.1, e a solicitação S, é a pressão interna aplicada ao duto.

3.2.1 FORM

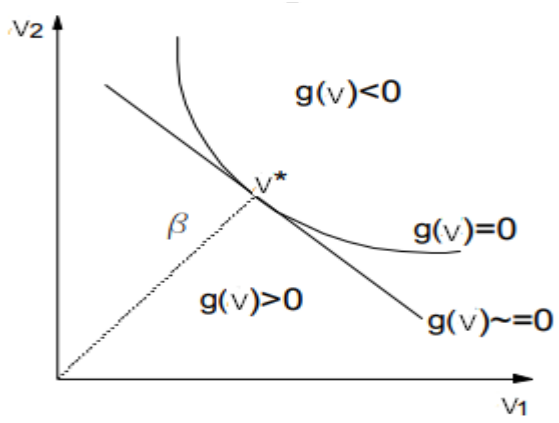
O FORM é um método de confiabilidade para o cálculo da probabilidade de falha que evita o cálculo da integral definida na equação (16). Para isso, o FORM converte as variáveis randômicas U do problema, para variáveis V normais padrão estatisticamente independentes (média = 0 e desvio padrão = 1).

Conforme aponta Sagrilo (1994), no método FORM, a superfície de falha G(V)= 0 é aproximada de forma linear (hiperplano), no ponto de maior densidade local de probabilidade, que corresponde ao ponto de projeto V*, que por sua vez é o ponto mais próximo da origem. O valor da distância deste ponto até a origem é chamado de índice de confiabilidade, e está definido na equação (18):

$$\beta = |V^*| \tag{18}$$

Na Figura 6 é exemplificada a representação gráfica do método FORM, onde podemos observar que ocorre uma aproximação linear através de um plano no ponto mais próximo da origem:

Figura 6 – Representação gráfica do método FORM



FONTE: Adaptado de Barbosa (2004)

3.2.1.1 Transformação de variáveis

A transformação de variáveis, envolve a eliminação da correlação entre as variáveis aleatórias e o cálculo das variáveis normais equivalentes. Representando assim, um mapeamento do espaço de projeto U para o espaço normal padrão V (TORO, 2014).

Para o caso onde U contém somente variáveis normais e estas por sua vez possuírem correlação entre si (ou não), um vetor V de variáveis normais padrão estatiscamente independentes pode ser obtido conforme a equação (19) demonstra:

$$\begin{aligned} V &= \Gamma{\sigma_{DP}}^{-1}(U - m) \\ &= J(U - m) \end{aligned} \tag{19}$$

Onde Γ é igual à L^{-1}, onde L é uma matriz triangular inferior obtida da decomposição de Choleski da matriz dos coeficientes de correlação de U, sendo expressa pela equação (20):

$$L = \begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{12} & L_{22} & 0 & 0 \\ . & . & . & . \\ L_{1p} & L_{2p} & . & L_{pp} \end{bmatrix} \tag{20}$$

Tomando p como sendo o número de variáveis aleatórias da transformação, a matriz triangular L pode ser obtida a partir das expressões da equação

(21):

$$\begin{aligned} L_{11} &= 1{,}0 \\ L_{i1} &= \rho_{i1} \qquad i = 1, \dots, n \\ L_{ik} &= \frac{1}{L_{kk}} \left(r_{ik} - \sum_{j=1}^{k-1} L_{ij} L_{kj} \right) \qquad 1 < k < i \\ L_{ii} &= \sqrt{1 - \textstyle\sum_{j=1}^{i-1} L_{ij}^2} \qquad i > 1 \end{aligned} \tag{21}$$

Para os casos onde as variáveis não são normais é necessário realizar uma transformação em normal equivalente para utilizar a equação (19). Adotando-se duas variáveis U_i e U_j com distribuições de probabilidade quaisquer e dependentes entre si, cuja dependência é definida pelo coeficiente de correlação ρ_{ij}, pode-se definir o coeficiente de correlação equivalente entre as duas distribuições normais equivalentes utilizando a equação (22):

$$\rho_{ij}^{\mathrm{E}} = \mathrm{F}\, \rho_{ij} \tag{22}$$

O fator F depende apenas de ρ_{ij} e dos coeficientes de variação das variáveis U_i e U_j. Kiureghian e Liu (1986) apresentam para uma variedade de expressões analíticas para o fator F.

Obtidas as normais equivalentes das variáveis U e as suas correlações equivalentes, o próximo passo consiste em utilizar a equação (19) para a obtenção das variáveis normais padrões estaticamente independentes V, da mesmo forma que é feita para variáveis normais.

Para o caso onde a função de densidade de probabilidades conjunta $f_u(U)$ é conhecida, a transformação de Rosenblatt é a mais recomendada na conversão das

variáveis U em V (SAGRILO, 1994). Essa conversão é ilustrada nas expressões da equação (23):

$$\begin{aligned} V_1 &= \phi^{-1}[F_{U1}(U_1)] \\ V_2 &= \phi^{-1}[F_{U2}(U_2/U_1)] \\ &\vdots \\ V_n &= \phi^{-1}\left[F_{Un}\left(\frac{U_n}{U_1 U_2 \dots U_n}\right)\right] \end{aligned} \tag{23}$$

Onde;

$F_{Ui}\left(\frac{U_i}{U_1 U_2 \dots U_{i-1}}\right)$ é a função cumulativa de probabilidade da variável U_i condicionada a valores conhecidos das variáveis $U_1, U_2, U_3, \dots, U_{i-1}$;

ϕ^{-1} é o inverso da função cumulativa normal padrão.

3.2.1.2 Busca ao ponto de projeto

Um dos passos fundamentais para o cálculo da probabilidade de falha pelo método FORM, é o de encontrar o ponto U* sobre a superfície de falha mais próxima à origem. Para encontrar o ponto de projeto no espaço das variáveis reduzidas, desenvolve-se uma expressão iterativa com as condições expressa na equação (24):

$$\begin{aligned} &\text{Minimize } |V| \\ &\text{Sujeito a } g(V) = 0 \end{aligned} \tag{24}$$

O método mais usado para resolver este problema de otimização é o método proposto por Hasofer e Lind (1974) e melhorado por por Rackwitz e Fiessler (1978). Este algoritmo é mais conhecido como HL-RF e é está representado na expressão iterativa da equação (25):

$$V^{k+1} = \frac{1}{|\nabla g(V^k)|^2}[\nabla g(V^k)^T V - g(V^k)]\nabla g(V^k)^T \tag{25}$$

Onde a função de falha no espaço reduzido, e o gradiente da função de falha no espaço reduzido, são dados respectivamente pelas equações (26) e (27), (SAGRILO, 1994):

$$g(V^k) = G(U^k) \tag{26}$$

$$\nabla g(V^k) = (J^{-1})^T \nabla G(U^k) \tag{27}$$

O algoritmo converge quando $\left|\mathrm{V}^{\mathrm{k+1}}\right| - \left|\mathrm{V}^{\mathrm{k}}\right| / \left|\mathrm{V}^{\mathrm{k+1}}\right| \leq \mathrm{Tol}$(tolerância). Sendo $\mathrm{V}^{\mathrm{k+1}}$ o novo ponto, e V^{k} o ponto atual.

3.2.1.3 Fator de importância

Além do índice de confiabilidade o método FORM pode retornar outras informações uteis. Segundo Sagrilo (1994), um dos mais importantes resultados obtidos através dos métodos analíticos são as medidas de sensibilidade relacionadas ao índice de confiabilidade em relação a variação dos parâmetros que definem a função de falha.

Um destes fatores é o fator de importância que define a importância relativa de uma variável aleatória na análise em questão (SAGRILO, 1994). Para uma variável aleatória i do problema, o fator de importância é dado pela equação (28):

$$I_i = \alpha_i^2 \tag{28}$$

Onde α_i é a componente do vetor normal à superfície de falha no ponto de projeto V, correspondente a variável i. O fator α_i é dado pela equação (29):

$$\alpha_i = \frac{\Delta g(V)_i}{|\Delta g(V)|} \tag{29}$$

3.2.2 Monte Carlo

A técnica de Monte Carlo é uma das mais simples para avaliar a probabilidade de falha de uma estrutura. Basicamente consiste em gerar um número bastante grande de amostras aleatórias para simular um experimento. A partir das distribuições de probabilidade das variáveis, é gerado um conjunto de amostras independentes. Para cada amostra gerada a função de falha é avaliada G(U), caso ela esteja no domínio de falha G(U) < 0 ela é contada.

Por fim a probabilidade de falha é obtida pela divisão da contagem de amostras no domínio de falha pelo número total de amostras. Conforme apontado por Sagrilo(1994) a obtenção da probabilidade de falha expressa na equação (16) pode ser aproximada pela equação (30), desde que o número de amostras seja elevado:

$$pf = \frac{1}{N}\sum_{i=1}^{N} I\{G(U) \leq 0\} \tag{30}$$

Apesar de ser um método que apresenta boas aproximações e ser de fácil implementação, este método requer um número bastante elevado de simulações, é por isso que basicamente este método é utilizado para validar novos métodos (TORO, 2014). Afim de reduzir o número de simulações do método de Monte Carlo foram criados vários métodos derivados do mesmo, o que fazem basicamente é adotar uma técnica de redução da variância.

Estas técnicas de redução da variância permitem que sejam geradas amostras na região mais representativas, ou seja, com menos amostras é possível uma representação satisfatória do modelo. Algumas destas técnicas incluem o Hipercubo Latino, amostragem por importância e esperança condicionada.

3.2.2.1 Esperança Condicionada

Segundo (BARBOSA, 2004) “O objetivo desta técnica é reduzir o espaço amostral das variáveis para a obtenção da probabilidade de falha, caracterizando o problema de forma condicional”. O método consiste em escolher uma variável aleatória X, que seja estatisticamente independente das outras e que possua maior

dispersão, isto é, que contribui com maior intensidade para a probabilidade de falha. Em seguida, deve-se expressar a variável X em termos das outras, através da relação G(U) = 0.

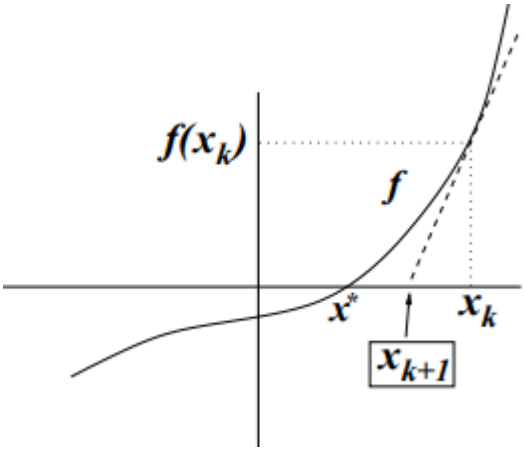
Reformulando a equação (16) em termos da função de distribuição acumulada F da variável X, e aplicando a técnica de Monte Carlo, Barbosa (2004) obteve a expressão para o cálculo da probabilidade de falha utilizando o método de Monte Carlo com Esperança Condicionada, conforme ilustrado na equação (31):

$$pf = \frac{1}{N}\sum_{i=1}^{N} F_X(X) \tag{31}$$

3.3 NEWTON-RAPHSON

O método de Newton-Raphson é conhecido pela sua rápida convergência e se baseia em calcular o próximo ponto da curva utilizando o ponto atual e a tangente da curva nesse ponto. Desta forma é obtida a interseção da tangente com o eixo das abcissas e calculado o próximo ponto da curva, o método prossegue até que a diferença entre o ponto atual e o próximo seja menor que uma tolerância previamente estabelecida ou que o número de iterações máxima seja excedido. Uma esquematização do método é demonstrada na Figura 7.

Figura 7 – Representação gráfica do método de Newton-Raphson



FONTE: (ASANO e COLLI, 2009)

Matematicamente o processo iterativo é montando usando a regra de iteração expressa pela equação (32), conforme demonstrado por Asano e Colli (2009):

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{32}$$

3.4 REDES NEURAIS

Um dos grandes objetivos da humanidade sempre foi a construção de máquinas inteligentes que pudessem operar e executar tarefas complexas sem a intervenção humana. Afim de alcançar este objetivo, o homem começou a observar como ele mesmo pensa, afim de entender os mecanismos que geram as ideias, assim como os pensamentos (RUSSEL e NORVIG, 2013).

Este sonho começou a se tornar cada vez mais tangível, a partir do século passado, onde foram desenvolvidos vários modelos de inteligência artificial. A partir da segunda década deste século, houve um aumento significativo nas bases de dados no mundo, além do barateamento de recursos como computação paralelas e do próprio computador desde anos anteriores, isto tudo proporcionou a ascensão das redes neurais. As redes neurais, permitem que tarefas antes realizadas apenas por humanos, fossem realizadas também por computadores (HAYKIN, 2008).

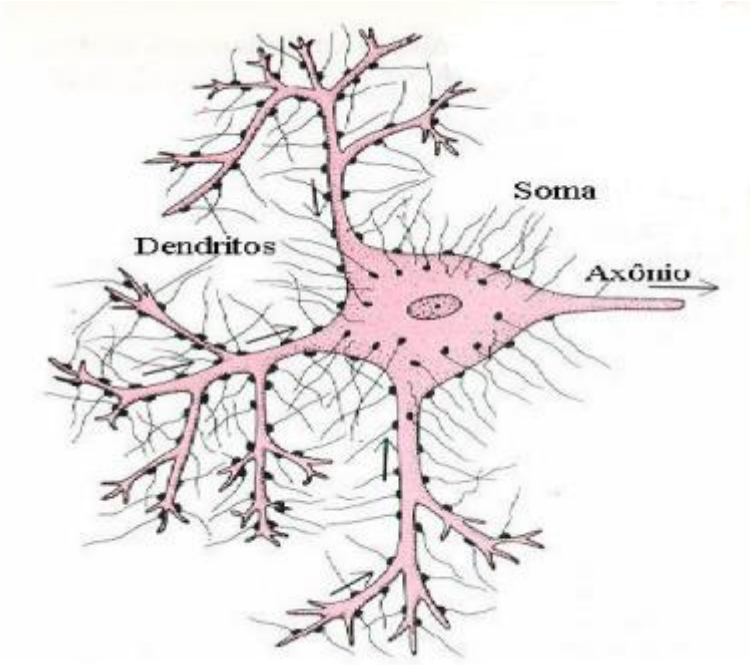
Redes neurais conseguem, a partir de um grande conjunto de dados de treinamento, extrair conceitos abstratos e relevante de tal forma que possa generalizar e extrapolar resultados. Segundo Haykin (2008) as principais características de uma rede neural são:

- ✓ Tolerância a falhas, mesmo que parte da rede apresente problemas, a rede ainda é capaz de apresentar boas aproximações, pelo fato da informação da rede estar distribuída;
- ✓ Generalização, permite que a rede gere saídas boas para entradas desconhecidas, que não pertencem ao conjunto de treinamento;
- ✓ Capacidade de aprendizagem, capacidade de aprender com os dados, isto é, extrair os conceitos implicitamente contidos nos dados;
- ✓ Habilidade de aproximação, desde que os dados sejam representativos do processo ou problema, a rede é capaz de aproximar funções continuas de ordem qualquer.

3.4.1 Estrutura das redes neurais

O cérebro humano é formado por cerca de 10 bilhões de neurônios (RUSSEL e NORVIG, 2013). Esta rede densa de neurônios possui diversos tipos de conexões responsáveis por transmitir impulsos nervosos, responsáveis pelo pensamento, reflexos, ações e as lembranças. Apesar de toda a complexidade desta rede, ela é formada por unidades simples, chamadas de neurônios. Os neurônios são a unidade básica de processamento, eles recebem e propagam o sinal do impulso elétrico para um ou mais neurônios. É possível visualizar uma ilustração de um neurônio típico na Figura 8.

Figura 8 – Estrutura de um neurônio biológico



FONTE: (BARBOSA, 2004)

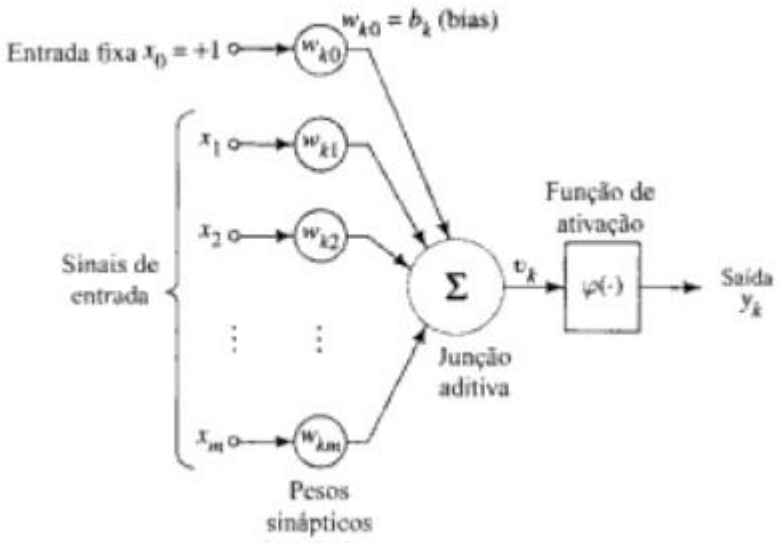
Conforme observa-se na Figura 8, os principais componentes de um neurônio são, (HAYKIN, 2008):

- ✓ Os dendritos, que são extensões na forma de filamentos responsáveis por receber o impulso de outros neurônios;
- ✓ A soma, ou corpo do neurônio, responsável por combinar os estímulos coletados de outros neurônios;

- ✓ O axônio, responsável por transmitir os estímulos para os neurônios ligados a ele.

As redes neurais artificiais se baseiam nesse comportamento da rede neural humana. Da mesma forma, são usadas unidades de processamento menores chamadas de neurônios artificiais, responsáveis por receberem entradas processadas de outros neurônios, combina-las e repassarem a saída gerada para outros neurônios (HAYKIN, 2008). Na Figura 9 é ilustrado os componentes de um neurônio artificial, sendo que na prática, este neurônio é implementação como uma rotina.

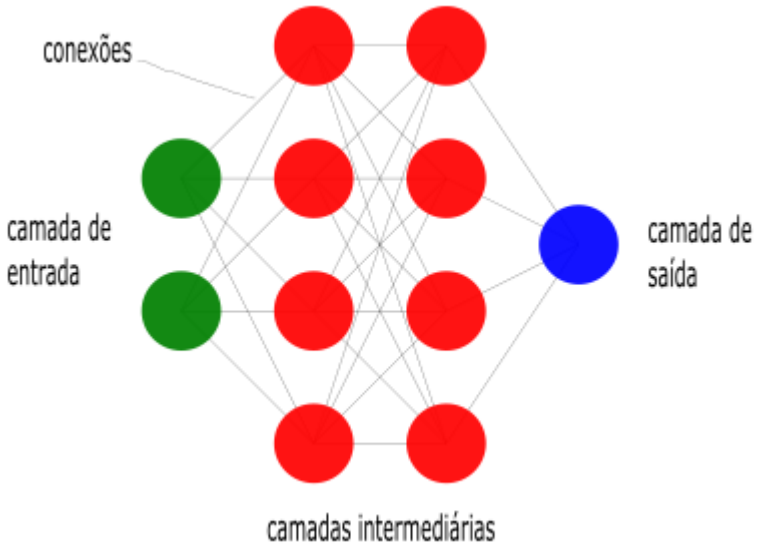
Figura 9 – Estrutura de um neurônio artificial



FONTE: (HAYKIN, 2008)

As redes neurais artificiais são organizadas em camadas de neurônios: camada de entrada, uma ou mais camadas ocultas e uma camada de saída. O número de neurônios nas camadas de entrada e saída, é função do tamanho do vetor de entrada e saída, respectivamente, enquanto que a quantidade de camadas intermediarias, bem como o número de neurônios em cada camada é determinado por meio de experimentos ou por recomendações da literatura. Uma exemplificação de uma rede neural com uma camada de entrada com 2 neurônios, 2 camadas intermediarias com 4 neurônios cada e uma camada de saída com apenas 1 neurônios, é ilustrada na Figura 10.

Figura 10 – Rede neural com 3 camadas ocultas



FONTE: (AUTOR, 2019)

Cada conexão em uma rede neurais possui um peso, que é um valor numérico responsável por ponderar uma determinada entrada (saída de outro neurônio) em um neurônio. Conforme aponta Barbosa (2004) na formação de uma rede neural 3 etapas devem ser seguidas:

- ✓ A determinação da arquitetura da rede, que define a forma como as camadas estão interligadas e como enviam e recebem dados umas das outras;
- ✓ A determinação da função de ativação, que define a forma como um neurônio processa o sinal de saída;
- ✓ A determinação do método de aprendizagem, responsável por ajustar os pesos das ligações da rede, fazendo com que a rede extraia o padrão dos dados.

3.4.2 Função de ativação

Como dito anteriormente um neurônio é uma unidade de processamento que recebe várias entradas e as combina gerando uma saída, que repassa para um ou mais neurônios. Matematicamente a saída de um neurônio é expressa pela equação (33).

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right) = f(v) \tag{33}$$

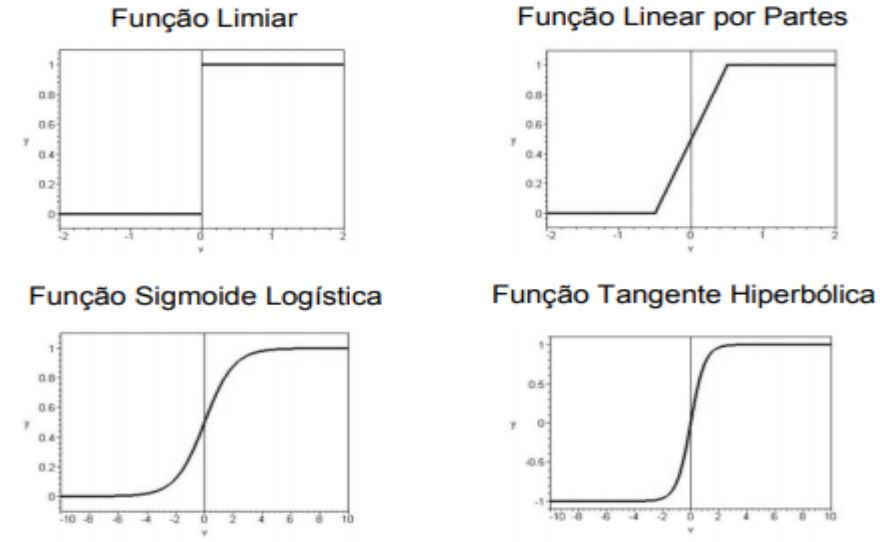
Onde w_0 é o valor de bias do neurônio, cuja a entrada correspondente é a constante 1. Existem vários tipos de função de ativação que são específicas para determinadas classes de problemas, e até mesmo para determinadas camadas (entrada, oculta ou saída). As principais funções de ativação utilizadas, segundo Quiles (2004), estão representadas matematicamente e graficamente, nas Figura 11 e Figura 12 respectivamente:

Figura 11 - Tipos de Funções de Ativação

Tipo de Função	Função
Limiar	$f(v) = \begin{cases} 1 \; se \; v \geq 0 \\ 0 \; se \; v < 0 \end{cases}$
Linear por Partes	$f(v) = \begin{cases} 1 \; se \; v \geq \frac{1}{2} \\ v \; se \; -\frac{1}{2} < v < \frac{1}{2} \\ 0 \; se \; v \leq -\frac{1}{2} \end{cases}$
Sigmoide Logística	$f(v) = (1 + \exp(-av))^{-1}$
Tangente Hiperbólica	$f(v) = \tanh(v)$

FONTE: Adaptado de Quiles (2004)

Figura 12 - Representação Gráfica dos Tipos de Funções de Ativação



FONTE: (QUILES, 2004)

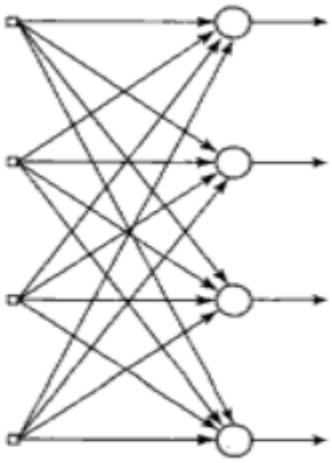
3.4.3 Arquitetura de redes neurais

A arquitetura das redes neurais está relacionada a forma como os neurônios se ligam com neurônios de outras camadas. Conforme Haykin (2008) explica, conhecer a arquitetura da rede é importante, pois influencia no tipo de treinamento escolhido. Além de influenciar no treinamento, a arquitetura da rede define a classe de problemas para o qual ela é mais adequada.

Conforme Haykin (2008) expõe existem basicamente 3 arquiteturas de redes neurais:

- ✓ Redes Single-Layer Feedforward: é uma rede bastante simples, basicamente existem apenas a camada de saída e a camada de entrada, ou seja, não existe nenhuma camada oculta. O fluxo do sinal é sempre progressivo, ou seja, não existe realimentação das camadas, este tipo de rede é exemplificado na Figura 13.

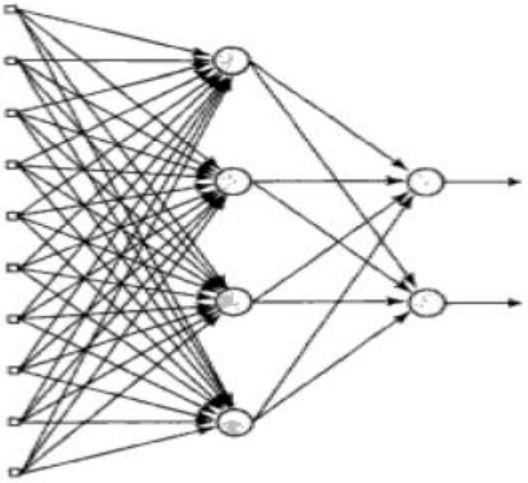
Figura 13 – Rede alimentada adiante com uma única camada



FONTE: (HAYKIN, 2008)

- ✓ Redes Multilayer Feedforward: neste caso a rede possui uma ou mais camadas ocultas e o fluxo de sinal também é progressivo. A rede pode ser densamente conectada quando todos os neurônios de uma camada estão conectados a todos os neurônios da camada seguinte, ou pode ser parcialmente conectada, quando algumas destas conexões estão ausentes. Este tipo de rede é exemplificado na Figura 14.

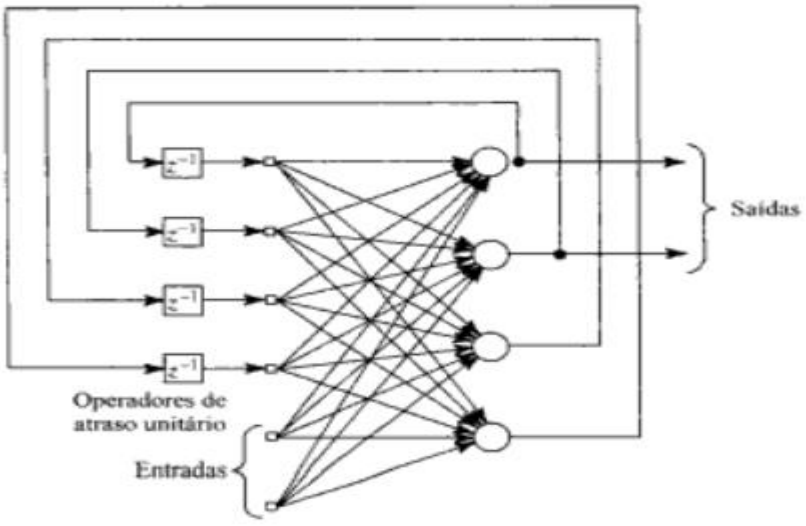
Figura 14 – Rede alimentada adiante totalmente conectada com 1 camada oculta



FONTE: (HAYKIN, 2008)

- ✓ Redes Recorrentes: este tipo de rede não é progressivo, ou seja, há pelo um loop de realimentação na rede. Isto significa que a saída de um neurônio pode voltar para o mesmo como entrada uma ou mais vezes. Este tipo de arquitetura é muito utilizado em problema em que a rede precise manter “memória” ao longo do tempo, algumas aplicações incluem prever a colisão de veículos através de filmagens, e completar uma frase de um texto de forma coerente e de acordo com o contexto. Este tipo de rede é exemplificado na Figura 15.

Figura 15 – Rede recorrente sem camadas ocultas



FONTE: (HAYKIN, 2008)

3.4.4 Treinamento de redes neurais

O treinamento de uma rede neural é o passo mais importante para a validação da mesma. Durante o treinamento são apresentados conjuntos de dados para que a rede aprenda o padrão ou regra associada aos dados, para que dessa forma forneça boas interpolações ou extrapolações baseadas no que aprendeu. Este processo de aprendizado, é gradual e por meio de um processo interativo, onde os pesos sinápticos são ajustados para que forneçam saídas mais parecidas com a realidade.

Haykin (2008) expõe a existência de três tipos de aprendizado:

- ✓ Aprendizado supervisionado: o usuário dispõe de conjuntos de dados com pares de entrada e saída, desta forma a rede pode comparar a saída gerada com a saída real, e desta forma definir e calcular uma métrica de erro para que possa ajustar o valor dos pesos sinápticos. Alguns algoritmos utilizados para este tipo de treinamento incluem a regra delta, o backpropagation e Levenberg-Marquardt.
- ✓ Aprendizado não-supervisionado: neste tipo de treinamento existem apenas os dados de entrada, não há dados de saída. A rede aprende a criar representações internas de características especificas, ou seja, cria classes, por isso esse tipo de treinamento é realizado em redes criadas para problemas de classificação. Os principais algoritmos de treinamento incluem o hebbiano e o aprendizado por competição.
- ✓ Aprendizado por reforço: o aprendizado ocorre devido a interações contínuas entre o agente (rede) e o ambiente. O agente realiza determinadas ações e obtém do ambiente um sinal de recompensa, indicando o quão bem aquela ação foi naquele momento. O objetivo do agente é maximizar a recompensa, de tal forma que aprenda uma política de ações ótimas, que por sua vez permite ao agente tomar as melhores ações possíveis. Desta forma, os dados de entrada e saída são obtidos a partir da própria interação do agente com o ambiente.

3.4.5 Algoritmo backpropagation

O algoritmo Backpropagation é o mais utilizado para aprendizado de redes neurais, atualmente o método evoluiu bastante e possui diversas variações. Este

algoritmo se baseia em duas etapas na rede: a propagação da entrada, e a retropropagação do erro (HAYKIN, 2008). Segundo Haykin (2008) na primeira etapa os dados de entrada são submetidos aos neurônios de entrada da rede, que se propagam camada por camada ao longo da rede até gerar uma saída. Neste passo, os pesos sinápticos das conexões dos neurônios são mantidos fixos.

Na segunda etapa, a saída gerada é subtraída da resposta real, gerando um sinal de erro que é propagado de forma inversa ao longo da rede, isto é, começando da camada de saída em direção às camadas ocultas. A medida que o sinal de erro passa pelas camadas, os pesos sinápticos das ligações dos neurônios são ajustados, afim de que a saída da rede se torne mais parecida com a resposta real.

O sinal de erro do neurônio j na interação n é calculado a partir da equação (34):

$$e_j(n) = d_j(n) - y_j(n) \tag{34}$$

O erro total gerado por uma saída da rede é dado pela equação (35):

$$\varepsilon(n) = \frac{1}{2}\sum_{j\epsilon C} e_j^2 \tag{35}$$

Onde o C denota o conjunto dos neurônios da camada de saída da rede. A correção dos pesos sinápticos é dada pela equação (36):

$$\Delta w_{ji}(n) = \eta\delta_j(n)y_i(n) \tag{36}$$

Dependendo de onde o neurônio está localizado, o termo $\delta_j(n)$ é diferente. Portanto temos dois casos, (HAYKIN, 2008):

Caso 1: O neurônio está localizado na camada de saída

. Neste caso calcula-se diretamente o sinal de erro usando-se a equação (34), e o termo $\delta_j(n)$ é calculado pela equação (37):

$$\delta_j(n) = e_j(n)\varphi'_j\left(v_j(n)\right) \quad (37)$$

Caso 2: O neurônio está localizado em uma camada oculta

. Neste caso, o sinal de erro do neurônio é calculado recursivamente em termos dos sinais de erros dos neurônios, aos quais este neurônio está conectado, conforme demonstrado na equação (38) para o cálculo do termo $\delta_j(n)$, onde o índice k denota os neurônios conectados ao neurônio j.

$$\delta_j(n) = \varphi'_j\left(v_j(n)\right)\sum_k \delta_k(n)\, w_{kj}(n) \quad (38)$$

3.5 ANDROID

3.5.1 Plataforma Android

Segundo Google (2019), “o Android é um sistema Linux multiusuário em que cada aplicativo é um usuário diferente”. Como cada aplicativo é um usuário, cada aplicativo possui suas próprias permissões para acessar recursos do sistema, tais recursos incluem acesso a componentes de hardware, internet, arquivos, outras fontes de dados, etc. além disso, cada aplicativo Android é executado em uma máquina virtual Java separada, garantindo desta forma o isolamento do código do aplicativo. A linguagem padrão de programação da plataforma Android é o Java.

Como Google (2019) defende, os aplicativos Android, são criados partir da combinação de componentes distintos que possuem seus próprios ciclos de vida, e que podem ser invocados independentes. Isso cria um modelo em que o aplicativo possui múltiplos pontos de entrada, diferentemente da maioria das outras plataformas em que existe um único ponto de entrada, normalmente uma função ou método “main”. Isso permite que o aplicativo seja iniciado a partir de qualquer componente, melhorando desta forma a experiência do usuário.

Outra característica importante dos aplicativos Android, é que os mesmos podem definir diversos recursos XML, para que se adaptem aos diversos tipos de

configurações. Alguns exemplos de configurações incluem a posição da tela (retrato ou paisagem), e o idioma atual (GOOGLE, 2019).

3.5.2 Estrutura das Aplicações Android

Como dito anteriormente os aplicativos Android possuem múltiplos pontos de entrada, justamente pelo fato de ser formado pela justaposição de componentes que desempenharem funções especificas dentro da aplicação. O sistema Android reconhece 4 tipos de componentes distintos, (MEDNIEKS, DORNIN, *et al.*, 2013):

- ✓ Atividades: representam uma tela da aplicação. Geralmente cada aplicativo possui várias telas, que por sua vez possuem vários elementos gráficos chamados de *View's*. Cada *View* controla uma porção da tela e possui manipuladores de eventos associados a si, como por eventos ouvintes para quando ocorrer toques na tela, deslizar do dedo, etc;
- ✓ Serviços: são componentes que são executados em background, ou seja, não possuem interface gráfica. Eles são usados para o processamento de tarefas longas como downloads, entre outras tarefas que não possam ser executadas na *thread*, ponteiro responsável por ler e executar o código do programa, principal do aplicativo, com o risco de travar a aplicação e a mesma ser fechada pelo sistema operacional.
- ✓ Provedores de Conteúdo: conforme dito anteriormente cada aplicativo possui suas permissões para acessar recursos. Cada aplicativo, possui arquivos privados em que nenhuma outra aplicação pode ter acesso, porém em muitos casos é preciso que uma aplicação compartilhe dados com outras, como por exemplo a lista telefônica do aplicativo de ligação. Os provedores de conteúdo, permitem que os dados, arquivos, registros no banco de dados *SQLite*, entre outras fontes de dados, possam ser acessados de forma segura e controlado por outros aplicativos.
- ✓ Receptores de transmissão: são componentes que respondem a anúncios feitos pelo sistema ou por outros aplicativos. Desta forma, eventos como bateria fraca, termino de downloads de certos arquivos, podem ser detectados por aplicativos, para que os mesmos possam responder com um aviso ao usuário, ou qualquer outra ação.

Além destes componentes temos as intenções (*Intents*) que servem para iniciar componentes, além de transmitir dados de um componente para outro. Desta forma um componente ao iniciado, pode, através das informações recebidas, decidir que tipo de ação executar.

Qualquer componente criado no aplicativo e que deva ser utilizado pelo mesmo, deve ser declarado no seu arquivo de manifesto. Este arquivo controla as principais configurações do aplicativo, como os componentes da aplicação, as permissões, a versão mínima do Android para executar o aplicativo, entre muitas outras configurações.

Afim de permitir um maior reaproveitamento de componentes, na versão 3.0 do sistema Android foi criado o componente fragmento, que não é reconhecido pelo sistema, apenas pelas telas, ou atividades (MEDNIEKS, DORNIN, *et al.*, 2013). Seguindo esta filosofia, a tela é modularizada, onde porções da tela é controlada por um fragmento, sendo que este fica responsável por comportar as *View's*, bem como despachar os eventos de GUI para as mesmas. A atividade então torna-se apenas uma gerente de fragmentos, deixando todo o trabalho pesado para os mesmos. Este modelo permite criar telas como uma composição de fragmentos, que podem fazer tarefas complexas e abrigar diversos *View's* organizadas nos mais diversos layouts.

3.5.3 Ciclo de vida dos componentes

Um conceito muito importante quando se fala em aplicações Android, é o ciclo de vida dos componentes. Tanto o aplicativo quanto os componentes, que o formam, possuem início, meio e fim. Quando determinadas condições acontecem, o Sistema Operacional modifica o estado do ciclo de vida dos componentes, ativando eventos, que por sua vez consistem em chamadas de funções especiais dentro da aplicação.

As atividades, possuem um ciclo de vida que está associado basicamente com a visibilidade da tela atual. Na Tabela 1, é possível visualizar os principais métodos do ciclo de vida de uma atividade, (MEDNIEKS, DORNIN, *et al.*, 2013):

:

Tabela 1 - Métodos de ciclo de vida de uma atividade

Método	Descrição
onCreate	Chamado quando a atividade é criada, nesse método é definido o layout da tela bem como as configurações iniciais
onRestart	Chamado quando a tela se torna visível para o usuário
onResume	Chamado quando a tela está apta para interagir com o usuário
onPause	Chamado quando a tela não está mais interativa, isso pode acontecer quando outra tela estiver no topo da pilha da aplicação
onStop	Chamado quando a tela não está mais visível ao usuário
onDestroy	Chamado quando a tela é encerrada, pode ocorrer quando a tela é retirada do topo da pilha ou quando o aplicativo é encerrado

FONTE: (AUTOR, 2019)

O conhecimento do ciclo de vida de uma aplicação Android é essencial para um bom desempenho da mesma. Conforme Google (2019) aconselha, uma aplicação deve ceder recursos para o sistema Android, quando por exemplo a tela atual não estiver visível, diminuindo desta forma o uso da CPU. Isto por sua vez implica na diminuição do gasto de energia, aumentando a performance da aplicação, ou seja, a aplicação só usa recursos quando realmente precisa, quando não é mais necessário ela os libera.

3.6 WEBSERVICE RESTFUL

“Os Serviços Web permitem que sistemas heterogéneos comuniquem entre si por meio de trocas de mensagens” (NGOLO, 2009). Os sistemas neste caso, podem ser de diversas plataformas como sistemas operacionais e linguagens de programação diferentes. Além disso, a reutilização do serviço web para diversos clientes é outro fator importante, pois clientes como aplicativos Android, aplicações desktop e aplicações web podem consumir o mesmo webservice.

Para que possam ser executados, os webservices precisam de um contêiner de servidor, que gerencia detalhes de baixo nível como pool de conexões,

mapeamento da URI, configuração do protocolo HTTP, etc. No mercado existem vários containers disponíveis de forma livre, Impacta (2019) indica alguns como: Apache Tomcat, JBoss, Glassfish, Jetty e IIS.

3.6.1 REST

"Um dos motivos pelo qual a arquitetura REST está a aumentar de popularidade é a sua simplicidade e facilidade de uso bem como o uso extensivo de tecnologias Web nativas como o HTTP" (NGOLO, 2009). O acesso ao recursos e serviços ofertados pelo webservice é feito por meio da URI do serviço, o que torna o REST mais simples que outras arquiteturas como o SOAP, onde devem ser especificadas várias camadas do serviço, bem como a configuração de várias propriedades em arquivos de configuração.

O princípio fundamental do REST, segundo Goncalves (2011) é a existência de recursos e de endereços associados a estes recursos. Recursos neste caso, podem ser um documento, um registro em um banco de dados ou o resultado de um algoritmo. Os endereços se referem a uma URI, para que o recurso seja encontrado na internet.

O REST como mencionado anteriormente se baseia na existência de recursos, e isto introduz uma arquitetura orientada a serviços (ROA). A ROA por sua vez se baseia em quatro pilares, conforme aponta Goncalves (2011) :

- ✓ Endereçabilidade: para que uma aplicação seja endereçável, a mesma deve expor seus recursos por meio de URI. A especificação errada do endereço do recurso torna impossível o acesso ao mesmo;
- ✓ Ausência de estados: cada requisição HTTP está isolada, ou seja, não depende da requisição seguinte ou da anterior. A requisição carrega consigo toda a informação de que o servidor necessita para acessar o recurso ou executar uma tarefa;
- ✓ Conectividade: os recursos devem estar tão conectados quanto possível. Isto permite que o usuário descubra todas as ações e recursos por meio de uma única URI base;
- ✓ Interface Uniforme: O protocolo HTTP e as suas primitivas fornecem uma interface ao REST, mas o que torna essa interface uniforme é o fato de todos os serviços usarem a interface da mesma forma.

3.6.2 Métodos HTTP

Conforme Ngolo (2009) aponta, o HTTP é o protocolo dominante na internet por estar presente em todas as plataformas e ser utilizado em quase todas as requisições. Usando o protocolo HTTP é possível ter acesso aos mais variados recursos como vídeos, imagens, páginas da internet, resultados de operações, etc. Quando se faz uma requisição HTTP, é necessária a especificação da URI do recurso ou serviço bem como a ação relacionada ao recurso ou serviço. Goncalves (2011) comenta que a ação é especificada por meio das primitivas ou verbos HTTP. Os verbos mais relevantes estão listados na Tabela 2.

Tabela 2 - Verbos HTTP

Método	Descrição
GET	Recupera um recurso
PUT	Atualiza um recurso
POST	Envia um recurso para o servidor
DELETE	Apaga um recurso

FONTE: (AUTOR, 2019)

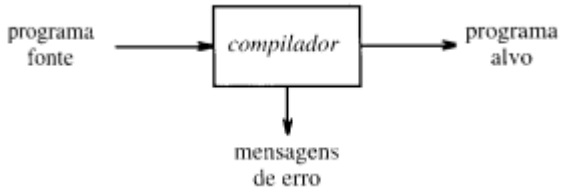
Em uma requisição HTTP em um serviço RESTful, os dados são transferidos por meio de mensagens. Ngolo (2009) indica que dentre os principais tipos de formatos de mensagens incluem JSON e XML. Atualmente o JSON é o mais utilizado por sua simplicidade e por estruturar a informação de forma muito mais compacta que o XML, o que torna o envio e recebimento das mensagens muito mais rápidos.

3.7 COMPILAÇÃO

Atualmente existe diversos programas aplicativos nas mais diversas áreas, estes programas contam com as mais diversas GUI, que por sua vez facilita a vida do usuário por emular funções complexas em uma representação gráfica amigável e fácil de usar. Apesar de todas estas facilidades, as interfaces gráficas podem não resolver todos os problemas, e caso conseguissem seria bastante complexa sua implementação quanto sua usabilidade quebrando assim o seu princípio básico: a facilidade.

Uma solução é a criação de uma linguagem de domínio especifico, para um determinado tipo de problema. Uma linguagem expande as possibilidades, pois com algumas linhas de código é possível resolver tarefas complexas. A especificação de uma linguagem envolve a criação de um compilador. Um compilador é um programa que converte um programa escrito em uma linguagem fonte para um programa na linguagem alvo (AHO, SETHI e ULLMAN, 1995), esta definição é ilustrada na Figura 16.

Figura 16 - Um Compilador



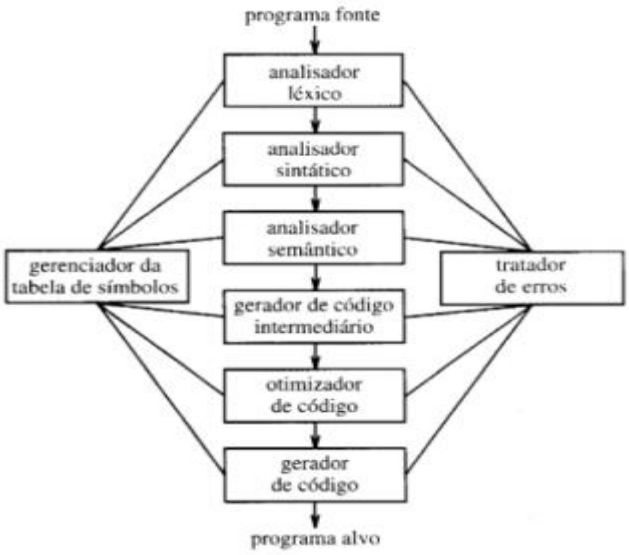
FONTE: (AHO, SETHI e ULLMAN, 1995)

É comum linguagens de alto nível tais como C e C++ serem compiladas para linguagens de baixo nível, tal como o código Assembly. O processo de compilação existe para facilitar a programação. Um exemplo é variedade de linguagens de alto nível que fornecem uma maneira muito mais fácil de se programar do que as de baixo nível, que estão muito próximas do código binário.

Conforme aponta Aho, Sethi e Ullman (1995), a compilação apresenta duas partes: a análise e a síntese. Na parte da análise, o programa fonte é dividido em suas partes constituintes, e é criada uma representação intermediária do mesmo. Na parte de síntese, a representação intermediária é sintetizada juntamente com seus atributos para gerar o programa na linguagem alvo, que geralmente é o código de máquina.

O compilador opera em fases, que converte o programa para diversas representações diferentes ao longo de todo o processo de compilação. Na Figura 17 é possível visualizar todas as seis fases do processo de compilação.

Figura 17 – Etapas da compilação



FONTE: (AHO, SETHI e ULLMAN, 1995)

As fases de analise léxica, sintática e semântica fazem parte da análise do compilador, enquanto que a geração de código intermediário, otimização de código e geração de código final fazem parte da síntese. Associado à estas fases do compilador, temos a tabela de símbolos, responsável por guardar atributos dos componentes do programa, e o verificador de erros, responsável por apresentar um relatório sobre os diversos erros que surgem durante todo o processo de compilação (AHO, SETHI e ULLMAN, 1995).

3.7.1 Analise léxica

A análise léxica é a primeira fase do compilador e envolve a leitura dos caracteres do programa fonte e geração de tokens para o analisador sintático. Um token é um identificador retornado por um conjunto de cadeias de caracteres que obedece a uma regra ou padrão (AHO, SETHI e ULLMAN, 1995), sendo classificados como:

- ✓ Palavras-clave: palavras que devem aparecer literalmente na linguagem sem variação alguma.
- ✓ Identificadores: palavras que obedecem a um padrão, porém podem assumir diversos valores, são utilizados para nomear entidades em um programa como funções, variáveis, classes, constantes, etc.

- ✓ Símbolos especiais: símbolos que não podem parecer em identificadores ou palavras chave, são usados para compor expressões aritméticas e logicas, comando de atribuição, etc.
- ✓ Constantes: podem ser valores reais, inteiros, booleanos, caracteres ou literais.
- ✓ Comentários: cadeias de caracteres com o único propósito de documentação.

Além disso, o analisador léxico tem como função eliminar espaços em branco e comentários, além de detectar e informar os erros léxicos.

3.7.2 Expressões e definições regulares

Um ponto muito importante com relação à analisadores léxicos é a especificação dos tokens. Uma notação muito utilizada para esse fim são as expressões regulares. Na prática o que ocorre é a utilização de uma linguagem de especificação de analisadores léxico, baseada na notação de expressões regulares, para criar de forma automatizada um analisador léxico.

Conforme Aho, Sethi e Ullman (1995) apontam, as expressões regulares são formadas a partir de expressões regulares mais simples. Onde cada expressão regular descreve um conjunto de cadeias de caracteres, sem precisar listas todas as cadeias do conjunto. Conforme Aho, Sethi e Ullman (1995), as principais regras para a construção de expressões regulares são:

- ✓ Alternância: representada por uma barra invertida (|), serve para separar alternativas;
- ✓ Agrupamento: representado pelos parênteses, serve para definir o escopo ou a procedência da cadeia;
- ✓ Repetição: está relacionado a um quantificador após um token ou agrupamento, e indica o número de vezes que o elemento precedente pode ocorrer.

Complementar ao conceito de expressões regulares, temos o conceito de definições regulares, que segundo tal basicamente consistem em nomear expressões regulares, para que estas possam ser usadas como se fosses símbolos, facilitando desta forma a legibilidade e a escrita de outras expressões regulares em função destas.

3.7.3 Análise sintática

O analisador sintático é responsável por validar a sintaxe da linguagem, para isso ele agrupa os tokens fornecidos pelo analisador léxico, e verifica se uma determinada cadeia pode ser gerada pela gramatica da linguagem (AHO, SETHI e ULLMAN, 1995). Para isso, o analisador sintático constrói a arvore sintática correspondente do programa. Além disso, o analisador sintático tem como tarefa a detecção de erros sintático do programa, ou seja, informar sobre as construções que não obedecem a gramática da linguagem (AHO, SETHI e ULLMAN, 1995).

3.7.4 Gramatica livre de contexto

Conforme Aho, Sethi e Ullman (1995) explicam, uma gramática descreve a estrutura hierárquica das construções de uma linguagem de programação. Já uma gramatica livre de contexto possui a mesma tarefa, porém não é sensível as características semânticas da linguagem.

As gramaticas livres de contexto são formadas por 4 elementos (AHO, SETHI e ULLMAN, 1995):

- ✓ Terminais: são os símbolos básicos pelos quais se constroem as cadeias, Neste caso os terminais são os próprios tokens;
- ✓ Não-terminais: definem conjuntos de cadeias que auxiliam na especificação da linguagem, além disso, por serem os nós interiores das arvores sintática, impõe uma ordem hierárquica na linguagem, que por sua vez é a base para a análise sintática e a própria tradução;
- ✓ Símbolo de partida: é o não-terminal cujo conjunto de cadeias é a própria linguagem definida pela gramatica, ou seja, ele é o nó raiz de qualquer programa escrito nessa linguagem;
- ✓ Produções: especificam a forma pela qual os terminais e não-terminais de uma gramática podem se combinar para formar as cadeias da linguagem. Cada produção consiste em um não-terminal seguido por uma seta, seguido por uma cadeia de não-terminais e/ou terminais.

Para exemplificar um pouco estes conceitos considere a gramatica definida na Figura 18.

Figura 18 – Gramática para as quatro operações aritméticas

```
expressao -> expressao + termo | expressao - termo | termo;
termo -> termo * fator | termo / fator | fator;
fator -> DIGITO | (expressao);
```

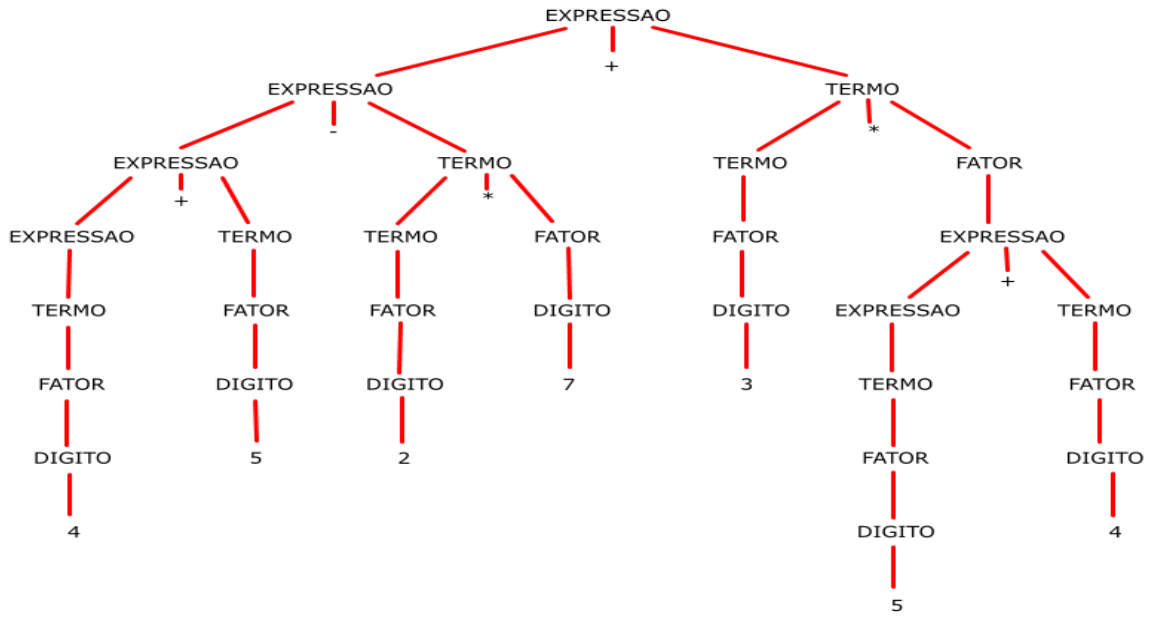
FONTE: (AUTOR, 2019)

Na Figura 18, o token “DIGITO” denota o conjunto dos dígitos de 0 a 9. O não terminal “expressao” é o símbolo de partida, pois ele é a raiz, ou seja, através dele toda a gramática se desenvolve. O token “DIGITO” é um terminal, enquanto que “expressão”, “termo” e “fator” são os não-terminais, pois os mesmos são nós internos da árvore, cuja função é facilitar a escrita da gramatica em termos dos terminais. Por fim cada uma das linhas da Figura 18 representa uma produção, pois temos um não terminal seguido por uma seta, seguido por uma cadeia de terminais e/ou não terminais.

3.7.5 Arvores gramaticais

As arvores gramaticais mostram, como o símbolo de partida deriva uma cadeia de construção da linguagem. Na Figura 19, pode-se observar a árvore gramatical para a expressão 4 + 5 – 2 * 7 + 3 * (5 + 4) conforme a gramática da Figura 18:

Figura 19 – Árvore gramatical para uma expressão matemática



FONTE: (AUTOR, 2019)

Uma arvore gramatical representa o programa de forma hierárquica, e possui as seguintes características (AHO, SETHI e ULLMAN, 1995):

1) A raiz é rotulada pelo símbolo de partida;
2) Cada nó é rotulado por um token ou uma produção vazia;
3) Cada nó interior é rotulado como um não terminal;
4) Se A é um nó interior e x_1, x_2,..., x_n são rótulos dos filhos daquele nó, então A-> x_1,x_2,...,x_n é uma produção.

3.7.6 Analise semântica

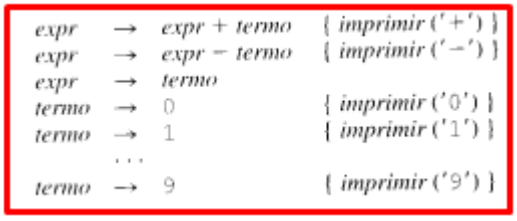
Como visto anteriormente a sintaxe de uma linguagem de programação é geralmente descrita por uma gramatica livre de contexto. A questão é que todo programa possui um contexto associado com as partes constituintes do programa.

Chamamos isto de semântica do programa, e consiste na utilização da árvore gramatical para: identificar operadores e operandos das expressões, reconhecer erros semânticos, fazer verificações de compatibilidade de tipo, analisar o escopo das variáveis, fazer verificações de correspondência entre parâmetros, etc. (AHO, SETHI e ULLMAN, 1995). Alguns exemplos de erros semânticos incluem: utilizar uma variável que não foi declarada ou fora do seu escopo, e fazer atribuição onde os lados direito e esquerdo possuem tipos incompatíveis.

Como estamos falando de gramatica livre de contexto, a validação semântica do programa não está incluída na especificação da gramática, portanto deve ser realizada de forma separada. Durante a análise sintática, os atributos dos tokens e produções são salvos na tabela de símbolos, sendo que posteriormente estes dados são consultados para verificar a validade semântica do programa. Este processo é dirigido pela sintaxe, ou seja, para cada regra sintática da gramática (produções) é associada uma ação semântica representada por uma chamada de rotina, cuja função é salvar e consultar atributos na tabela de símbolos ou em uma representação da arvore sintática (AHO, SETHI e ULLMAN, 1995).

Na Figura 20, é representada uma gramatica simples para criar expressões que envolvem soma e subtração, pode-se observar que cada produção possui uma ação semântica associada, que neste caso é a impressão de algum caractere.

Figura 20 - Exemplo de ação semântica incorporada à uma produção



FONTE: (AHO, SETHI e ULLMAN, 1995)

3.7.7 Geração de código intermediário

Uma representação intermediária é um código para uma máquina abstrata que deve ser fácil de produzir e traduzir no programa objeto (AHO, SETHI e ULLMAN, 1995). O mais conhecido código intermediário é o código dos 3 endereços utilizado quando a linguagem alvo é algum Assembly. No código dos 3 endereços, existem no máximo 3 variáveis envolvidas em um comando, uma no lado esquerdo da atribuição, e duas no lado direito separadas por algum operador.

3.7.8 Otimização de código

A otimização de código consiste em uma melhoria no código intermediário para que o código final seja mais rápido em sua execução. Conforme Aho, Sethi e Ullman (1995) apontam, o ideal seria que os compiladores produzissem código tão bom quanto o código escrito à mão, porém isto só ocorre em caso limitados e com uma certa dificuldade.

De forma macroscópica existem 2 formas de otimizações em compiladores: as dependentes da máquina-alvo como a alocação de registradores, e as que não dependem da máquina-alvo, por isso podem ser aproveitadas em diversas plataformas. Existe um consenso popular que diz que a maioria dos programas gastam 90% do seu tempo de execução em 10% do seu código (AHO, SETHI e ULLMAN, 1995). Com base em informações como essa, os compiladores procuram otimizar as regiões mais críticas do programa, que são definidas muitas vezes por meio de estatísticas a respeito de outros programas-fonte.

3.7.9 Geração de código final

Essa consiste na fase final do compilador, onde finalmente é gerado o código para a linguagem alvo. Geralmente, o código é gerado para uma linguagem de baixo nível como *assembly* ou código de máquina. Porém, também é muito comum a conversão de uma linguagem de alto nível para outra também de alto nível, é o que ocorre com muitas linguagens de script, como por exemplo a Engine Unity, onde são programados jogos e ambientes 3d. A programação no Unity é feita na linguagem C#, que por sua vez é compilada para linguagem especificas dependendo da plataforma alvo, se for para Android, por exemplo, é compilado para código Java. Neste caso, a linguagem C# tem uma característica multiplataforma, onde existe um gerador de código final especifico para cada plataforma.

3.8 THREADS E CONCORRÊNCIA

Inerente a todo sistema computacional temos o conceito de Thread. Tanenbaum (2012) esclarece que thread em um programa é um ponteiro que lê as instruções do código e as executa sequencialmente. O que acontece é que muitas vezes os programas ou aplicativos precisas executar tarefas de forma não sequencial, por exemplo, o usuário pode querer continuar interagindo com a interface gráfica de uma planilha eletrônica, enquanto que a mesma está salvando os dados da planilha, ou até mesmo procurando atualizações.

Para resolver este tipo de problema é necessária a criação de várias Threads dentro do programa, uma para cada tarefa não sequencial, o que pode aumentar a performance da aplicação, adicionando um grau de paralelismo real, o que diminui o tempo de computação.

3.8.1 Benefícios

Conforme Goetz, Peierls, et al. (2008) descrevem, o uso de threads facilita a modelagem de tarefas e ações, transformando tarefas assíncronas em um conjunto de tarefas sequenciais. Permite também converter um código complicado em um código linear mais simples de escrever. Os principais casos de uso de threads são, (GOETZ, PEIERLS, *et al.*, 2008):

- ✓ Explorar vários processadores: programas com múltiplas *threads* são executados simultaneamente em vários processadores, aumentando a performance;
- ✓ Simplicidade da modelagem: gerenciar tarefas assíncronas é complicado pois envolve a manipulação de recursos de baixo nível. Um fluxo de trabalho assíncrono e complicado pode ser decomposto em vários fluxos de trabalho mais simples e síncronos, cada um executando em um thread separado, interagindo apenas uns com os outros em pontos de sincronização específicos;
- ✓ Interfaces de usuário mais responsivas: Atualmente, os kits de GUI utilizam o modelo EDT. Neste modelo a thread principal do programa é responsável por responder aos eventos da GUI. Em um acionamento de botão, por exemplo, os manipuladores de eventos são chamados na *thread* principal. Se a tarefa for curta, não irá gerar “congelamentos” na GUI. Porém, para os casos em que as tarefas são longas, a thread principal pode iniciar outra *thread* para executar o trabalho. Após esta terminar a tarefa, a mesma devolve para a thread principal o resultado ou informações relacionadas à tarefa, a thread principal então atualiza a tela para refletir a conclusão ou andamento da tarefa. Este modelo, portanto impede o comuns travamentos de tela, proporcionando uma melhor experiência do usuário.

Além disso, Tanenbaum (2012) indica outra aplicação para as *threads*, que seria o processamento massivo de dados. Segundo esta proposta, uma *thread* ficaria responsável por receber os dados, uma segunda thread por processa-los, e uma terceira por enviar os dados processados. Desta forma, todas as 3 tarefas estarão sendo executadas ao mesmo tempo. Além disso, o uso de threads é muito usado em computação numérica para acelera-la, um exemplo é a construção de uma matriz com milhares de linhas e colunas, onde várias *threads* podem construir um pedaço da matriz separadamente, e ao final do processo juntar as contribuições para formar a matriz inteira.

3.8.2 Thread Safety

Apesar de todos os seus benefícios, as *threads* devem ser usadas com cuidado, pois programas *multithread* podem adicionar novos tipos de erros ou falhas, que não existem em um ambiente com uma única thread. Além disso, o uso excessivo

de threads pode ter o efeito contrário no desempenho, uma vez que a troca de contexto entre as várias *threads* dentro da aplicação, consome uma fatia considerável do tempo de execução do processo.

Conforme Goetz, Peierls, et al. (2008) discutem, o código thread safety consiste em gerenciar o acesso ao estado mutável compartilhado. Este estado, consiste nas variáveis mutáveis do programa que estejam acessível por múltiplas threads. Portanto tornar um código seguro consiste em proteger os dados do acesso simultâneo não controlado. Um código seguro garante que o acesso ao estado mutável de qualquer variável seja feito de forma sincronizada, evitando dessa forma, corrupção dos dados.

Goetz, Peierls, et al. (2008) comentam ainda, que a sincronização de threads deve ser feitas em pontos específicos do código, chamados de regiões críticas. É nas regiões críticas, que ocorre o acesso a recursos compartilhados, e portanto deve ser feito de forma sincronizada, isto é, uma única thread por vez pode estar em uma região critica. Sincronizar threads em suas regiões críticas impedem problemas como condições de corrida. Para exemplificar este problema considere a classe Java definida na Figura 21.

Figura 21 – Definição da Classe java Estadio

```
public class Estadio {

    private int pessoaCount;

    public void addPessoa() {
        pessoaCount++;
    }

}
```

FONTE: (AUTOR, 2019)

Conforme ilustrado na Figura 21 a classe Estadio possui um único atributo que representa o número de pessoa no estádio, e um método responsável por incrementar este número. Suponha que haja uma instancia dessa classe compartilhada por duas ou mais threads, e que todas estão a todo momento recebendo novas pessoas, portanto incrementando o valor do atributo “pessoaCount”. Como sabemos, o computador opera por meio de processos e a CPU muda a execução de um processo para outro em um tempo muito curto, de tal forma, que em um único segundo vários

processos podem ter sido executados várias vezes, isso nos dá a sensação de paralelismo, sendo este paradigma conhecido como Multiprogramação (TANENBAUM, 2012). O mesmo acontece com as *threads* em um processo, a CPU escalona a execução de uma thread para outra.

O problema é que a execução da única instrução do método “addPessoa” não é uma operação atômica. A verdade é que ela é uma forma compacta para três operações: ler o valor da variável, incrementar o valor e escrever o novo valor. Desta forma, uma *thread* A pode invocar o método “addPessoa” e então, ler o valor da variável e incrementa-lo, mas na hora de escrever o novo valor, a CPU decide suspender a execução desta thread para executar uma thread B. A thread B por sua vez, pode fazer seu trabalho normalmente e chegar a incrementar o valor, mas quando a CPU voltar para a thread A, a mesma vai gravar aquele valor que havia calculado. Podemos observar aqui, que um incremento foi perdido, gerando uma corrupção nos dados, isso considerando apenas 2 threads. A situação pode se tornar muito mais problemática com mais threads.

O problema destes erros é que são difíceis de detectar e de difícil reprodução, na verdade um sistema inteiro pode funcionar tranquilamente por um bom tempo, mas uma hora o erro de sincronização pode aparecer e quebrar todo o sistema e a validade de seus dados.

4 METODOLOGIA

No presente trabalho, para o problema de análise de confiabilidade estrutural, são utilizados os seguintes métodos: Monte Carlo (MC), Monte Carlo com Esperança Condicionada (MCEC), FORM e Monte Carlo com Redes Neurais (MCRN). Em cada um desses métodos é abordado o problema de dutos considerando defeitos de dimensões iguais alinhados longitudinalmente e igualmente espaçados.

Estes métodos são utilizados para a análise de confiabilidade estrutural. Para o projeto baseado em confiabilidade de dutos, o método de Newton-Raphson é aplicado. Este método é associado aos métodos de confiabilidade, a fim de encontrar o ponto de projeto sobre a superfície de falha, obtendo dessa forma a espessura de projeto, para que o duto seja dimensionado para suportar as solicitações que lhe são impostas.

Como dito anteriormente, o presente trabalho consiste na implementação de um sistema computacional para o projeto baseado em confiabilidade de dutos sujeitos a múltiplos defeitos de corrosão. Para isso, é implementado um aplicativo Android com interface gráfica nativa, com integração com a Engine Unity para o gerenciamento da cena tridimensional, responsável por representar o duto e seus defeitos.

A abordagem adotada para resolver os problemas de confiabilidade, foi a de integrar o aplicativo Android com um servidor remoto executando um *WebService*, implementado no presente trabalho, a fim de expor funções de cálculo para os usuários por meio do protocolo HTTP, o principal meio de envio de informação na internet, usando o padrão RESTful. Desta forma a maior carga de computação ficou por conta servidor, que pode ser qualquer computador que execute o webservice implementado no presente trabalho e que tenha conexão com a internet, o que permite uma resposta mais rápida e unificada. Para que o servidor fosse capaz de executar os cálculos matemáticos de confiabilidade estrutural, aplicando os métodos de confiabilidade tal como o FORM, foi utilizado o software MATLAB. Desta forma o cliente remoto acessa indiretamente o Matlab para executar sua computação numérica.

Tendo em vistas, que o sistema, composto pelo aplicativo e o webservice, possui limitações como qualquer outro, e que o usuário poderia estar interessado em resolver um problema de dutos usando uma outra variável de projeto, ou até mesmo um problema qualquer de confiabilidade estrutural, que não tenha relação com dutos, foi desenvolvido um editor de código no aplicativo juntamente com uma linguagem de programação nomeada de “AnderScript”.

Esta linguagem permite uma sintaxe mais clara do problema fazendo com que o usuário se preocupe apenas com variáveis de projeto, variáveis aleatórias e a definição da função de falha. Enquanto que a manipulação destas expressões é feita por rotinas previamente programadas no Matlab, ou seja, o código do usuário é compilado em código que pode ser interpretado pelo Matlab e executado como qualquer outra função.

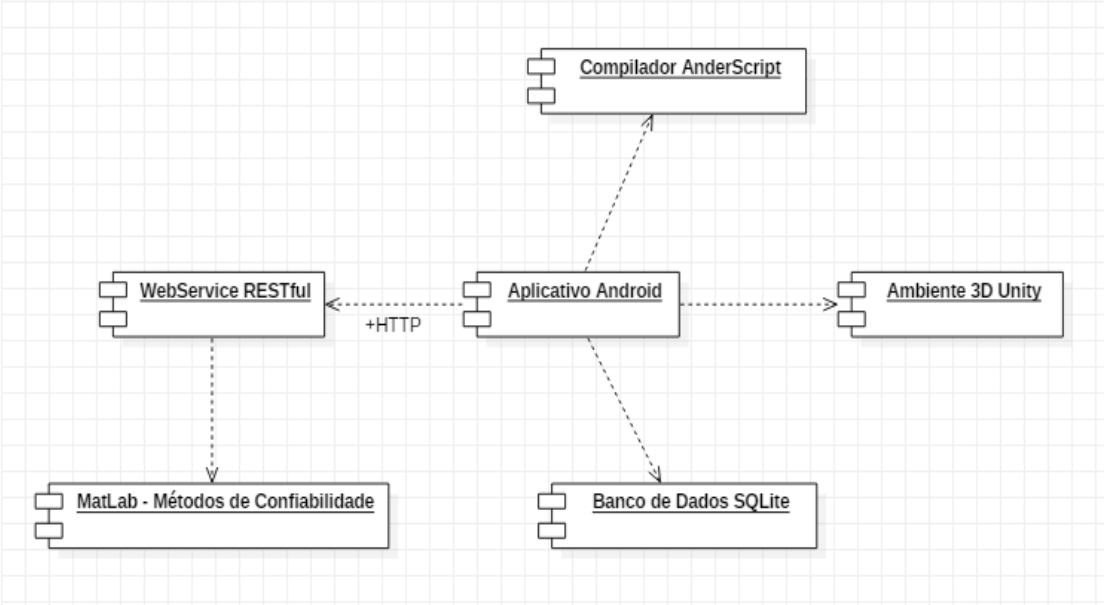
Os passos para a implementação do sistema serão apresentados na seguinte ordem:

1. Implementação dos métodos de confiabilidade estrutural no MatLab;
2. Implementação e Interação com o *WebService*;
3. Implementação do aplicativo Android;

4. Integração com a Engine Unity;
5. Desenvolvimento e acoplamento da linguagem de programação.

Cada um desses passos é representado por um componente diferente, onde um componente neste caso é tratado como um bloco independente dos outros que realiza tarefas exclusivas, tais como gerenciar a cena tridimensional (componente do Unity), e receber requisições de usuários e devolver uma resposta (*WebService*). O sistema completo bem como a relação entre esses diversos componentes pode ser visualizado na Figura 22.

Figura 22 – Diagrama de componentes do sistema



FONTE: (AUTOR, 2019)

Observando a Figura 22, nota-se que o aplicativo Android é depende de todos os outros componentes, o *WebService* por sua vez depende do acesso às rotinas de confiabilidade executadas no *software* MatLab. A dependência com o componente de banco de dados, expressa apenas a forma como o aplicativo persiste os dados no dispositivo, onde as informações sobre os dutos e seus respectivos defeitos de corrosão são especificadas utilizando a GUI do aplicativo, e salvas por meio de um banco de dados SQLite.

4.1 COMPONENTE DE CONFIABILIDADE

Como dito anteriormente, os métodos de confiabilidade foram implementados no presente trabalho através do *software* MatLab e chamados pelo *WebService*. Para facilitar o desenvolvimento foram criados arquivos de funções no MatLab para cada método de confiabilidade utilizado (FORM, Monte Carlo, Monte Carlo com Esperança Condicionada e Monte Carlo com Redes Neurais), tanto para a análise de confiabilidade quanto para o projeto baseado em confiabilidade, e colocados todos juntos no mesmo diretório, a fim de se evitar trocar o diretório de trabalho para cada nova chamada de funções de arquivos Matlab.

4.1.1 Cálculo da pressão de falha

Para o presente trabalho, o cálculo da pressão de falha foi realizado por expressões empíricas, utilizando a norma BS7910 (2005). Esta norma aproxima o defeito pelo paralelepípedo envolvente conforme visto anteriormente na Figura 1, facilitando a medição das dimensões do defeito. Como visto anteriormente, o cálculo da pressão de falha envolve apenas as características para um único defeito, porém em situações reais os dutos podem apresentar diversos defeitos de corrosão em quaisquer posicionamentos ao longo do duto

No presente trabalho, considerou-se dutos com múltiplos defeitos de corrosão com dimensões iguais alinhados longitudinalmente e igualmente espaços. Como visto anteriormente na seção 3.1.1, a norma BS7910 permite converter um conjunto de defeitos interagente em um único defeito equivalente, possibilitando a utilização da equação (7), para um único defeito, para o defeito equivalente referente a um grupo de defeitos interagentes.

4.1.2 Análise de confiabilidade estrutural

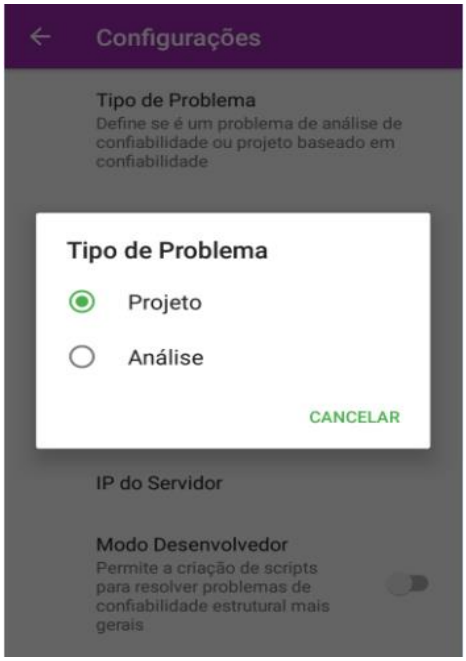
Como dito anteriormente foi desenvolvido um aplicativo para análise e projeto de dutos submetidos à corrosão, cujos parâmetros (tipo de problema, IP do servidor, método de confiabilidade e modo de desenvolvimento), podem ser configurados em uma tela de configurações no aplicativo, conforme ilustrado na Figura 23. O tipo de problema (análise ou projeto baseado em confiabilidade) pode ser configurado, assim

como o método de confiabilidade utilizado conforme ilustrado nas Figura 24 e Figura 25 respectivamente. Desta forma, o usuário pode escolher um dos seguintes métodos de confiabilidade: FORM, Monte Carlo, Monte Carlo com Esperança Condicionada e Monte Carlo com Redes Neurais.

Figura 23 – Tela de Configurações do aplicativo

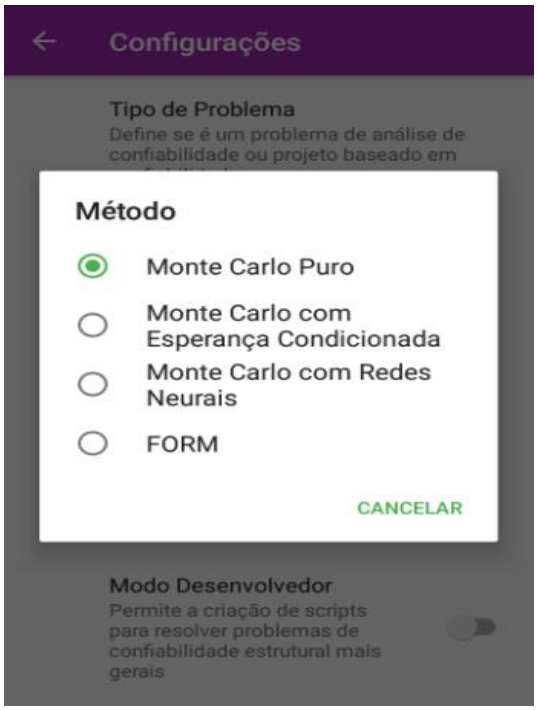
FONTE: (AUTOR, 2019)

Figura 24 – Opção “tipo do problema”



FONTE: (AUTOR, 2019)

Figura 25 – Escolha do método de confiabilidade



FONTE: (AUTOR, 2019)

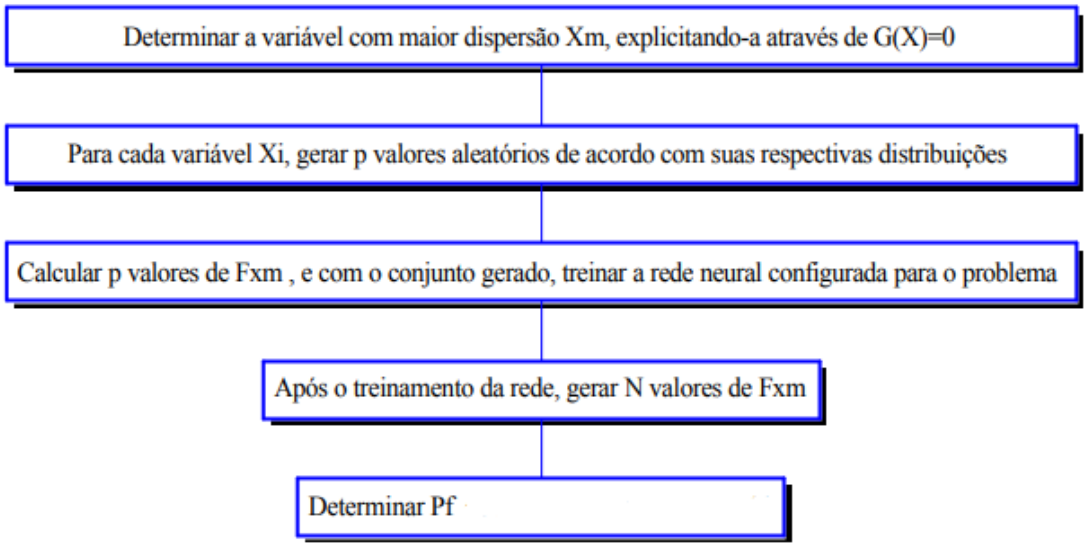
Para cada um desses métodos, foram aplicadas algumas funções no Matlab que recebem como entrada as informações do duto e de seus defeitos, dados sobre das geometrias do duto e dos defeitos e do material do duto, retornando o índice de confiabilidade do duto. Os procedimentos para a implementação dos métodos FORM, Monte Carlo e Monte Carlo com Esperança Condicionada foram explicados anteriormente nas seções 3.2.1 e 3.2.2.

Já a implementação do método de Monte Carlo com Redes Neurais foi realizada conforme a metodologia apresentada por Barbosa (2004). Essa metodologia propõe que primeiramente, o método de Monte Carlo deve utilizar a esperança condicionada como técnica de redução da variância. Assim como no método de Monte Carlo clássico, são geradas amostras aleatórias para as variáveis estatísticas do problema, porém em um número bem menor. Para cada conjunto de amostras, é calculado o valor da variável aleatória X de maior dispersão, variável que contribui com maior significância na probabilidade de falha, que neste caso é a pressão interna aplicada ao duto, bem como o valor da função de distribuição acumulada F_{xm} para esta variável, conforme discutido anteriormente na seção 3.2.2.1.

Este conjunto de amostras r_i, juntamente com os valores da função de distribuição acumulada F_{xm} para a variável X, calculados para cada amostra, são

então submetidos como um conjunto de treinamento [r_i, F_{xm}] para a rede neural. Uma vez que a rede está treinada, são geradas novas amostra aleatórias, mas em vez de calcular novos valores de distribuição acumulada para a variável X, estes valores são gerados pela rede neural, e por fim aplicada a equação (31). Estes passos são resumidos no fluxograma ilustrado na Figura 26.

Figura 26 – Fluxograma algoritmo Monte Carlo com Redes Neurais



FONTE: (BARBOSA, 2004)

A rede neural adotada, é uma rede Multilayer Feedforward densamente conectada, possui 2 camadas ocultas, cada uma com 7 neurônios, e a camada de entrada possui 4 neurônios, correspondentes as quatro variáveis aleatórias utilizadas no presente trabalho (profundidade do defeito, pressão interna, taxa de corrosão radial e espessura do duto). A rede neural foi implementada utilizando a caixa de ferramentas para redes neurais do Matlab (NNT).

Para o método de Monte Carlo foram adotas 50000 simulações, para o método de Monte Carlo com Esperança Condicionada 10000 simulações, enquanto que o método de Monte Carlo com Redes Neurais 8000 simulações.

4.1.3 Projeto baseado em confiabilidade estrutural

O aplicativo desenvolvido, possui a opção de projeto baseado em confiabilidade para que seja calculado a espessura de projeto do duto, esta opção é ilustrada na Figura 24. Os mesmos métodos para a análise de confiabilidade são também utilizados no projeto baseado em confiabilidade. Este constitui um processo de duplo laço, no qual o Newton-Raphson, que faz a busca da variável de projeto num processo iterativo, faz chamadas aos métodos de confiabilidade, tais como o FORM, que também é um método iterativo.

Assim como foi feito para a análise de confiabilidade, para o projeto baseado em confiabilidade foram criados arquivos na linguagem da plataforma Matlab para cada método de confiabilidade. Desta forma, os dados de entradas são as informações do duto e dos defeitos, valores da média e o coeficiente de variação para cada variável aleatória, e os valores das variáveis paramétricas, além do índice de confiabilidade alvo especificado pelo usuário, sendo como resultado de saída neste caso, a variável de projeto espessura do duto.

O algoritmo para obtenção do ponto de projeto pode ser descrito pelos seguintes passos, que consistem na aplicação do método de Newton-Raphson:

1º - Definição do valor alvo para o índice de confiabilidade (β^{alvo}), ou seja, qual o grau de confiabilidade que a estrutura deve possuir, sendo este índice especificado de acordo com definições da JCSS (2000);

2º - Definir um do ponto de partida para a variável de projeto, no presente trabalho, espessura do duto;

3º - Início da iteração do método de duplo laço, Newton-Rapson juntamente com métodos de confiabilidade;

4º - Início do método de confiabilidade utilizado, definindo as variáveis aleatórias, para determinar o índice de confiabilidade $\beta(U, \mathrm{P}^{\mathrm{k}})$, associado às variáveis aleatórias U e à variável de projeto da iteração corrente P^{k};

5º - Determinação da função matemática $g(U, \mathrm{P}^{\mathrm{k}})$ para solução do problema com restrição de verificação de segurança definida pela equação (39);

$$g(U, \mathrm{P}^{\mathrm{k}}) = \beta(U, \mathrm{P}^{\mathrm{k}}) - \beta^{\mathrm{alvo}} = 0 \tag{39}$$

6º - Determinação do gradiente da função g em $(\mathrm{P}^{\mathrm{k}})$;

7º - Determinação do novo ponto pelo método de Newton-Raphson, neste caso obtêm-se um novo valor para a espessura do duto segundo a equação (40);

$$\mathrm{P}^{\mathrm{k+1}} = \mathrm{P}^{\mathrm{k}} - \frac{g\left(U, \mathrm{P}^{\mathrm{k}}\right)}{\nabla g(U, \mathrm{P}^{\mathrm{k}})} \tag{40}$$

Onde $\nabla g\left(U, \mathrm{P}^{\mathrm{k}}\right)$ é o gradiente da função $g\left(U, \mathrm{P}^{\mathrm{k}}\right)$, podendo ser obtido através da equação (41);

$$\nabla g\left(U, \mathrm{P}^{\mathrm{k}}\right) = \frac{\beta\left(U, \mathrm{P}^{\mathrm{k}} + \Delta \mathrm{P}^{\mathrm{k}}\right) - \beta\left(U, \mathrm{P}^{\mathrm{k}}\right)}{\Delta \mathrm{P}^{\mathrm{k}}} \tag{41}$$

8º - Verifica-se o critério de convergência adotado. Se a convergência é atingida encerra-se o algoritmo, caso contrário retorna-se ao 2º passo utilizando como ponto de partida a espessura do duto $\mathrm{P}^{\mathrm{k+1}}$. Adota-se para a tolerância um valor de 0,0001.

4.2 COMPONENTE DO WEBSERVICE

O webservice implementado neste trabalho, foi feito utilizando a biblioteca JAX-WS, da linguagem java. Esta biblioteca é utilizada para a criação de webservices RESTful na plataforma java. E o contêiner de servidor adotado foi o Glassfish.

Afim de que as rotinas do MatLab fossem acessíveis para o aplicativo Android, foi construído um *webservice* RESTful que funcionasse como *Wrapper,* uma rotina envolvente*,* para as rotinas Matlab, isto é, para cada rotina MatLab existe uma rotina no *webservice* que recebe os dados do aplicativo e chama a rotina correspondente do MatLab para obter o resultado da análise ou projeto baseado em confiabilidade, e retornar para o cliente.

O problema que surge é que o Matlab é um programa que foi criado tendo em mente que apenas um usuário acesse cada sessão do Matlab por vez, ou seja, caso uma *thread,* atividade não sequencial*,* de uma requisição no *webservice* coloque uma tarefa para ser executada no Matlab, e logo após outra *thread* também coloque uma

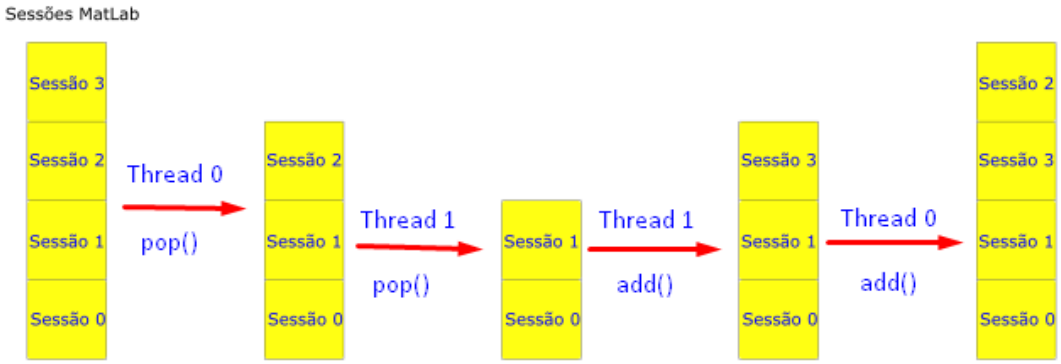
tarefa para ser executada, haverá corrupção nos dados de tal forma que nenhuma das duas obterá o resultado correto, pois ambas estarão alterando as mesmas variáveis presentes na sessão do Matlab. Porém fazer com que uma única *thread* tenha acesso as funções do Matlab, faz com que um único usuário seja atendido por vez, impossibilitando que múltiplos usuários consumam o *webservice* simultaneamente.

A solução encontrada é inicializar um determinado número de instâncias do MatLab, sendo que cada *thread* teria acesso a uma única instancia por vez que estivesse livre. Essa solução é viável, porém um problema deve ser resolvido: o acesso às instâncias do Matlab, uma execução do *software* Matlab, deve ser feito de forma sincronizada, afim de que uma *thread* não cause corrupção nos resultados de outras threads.

As conexões com o Matlab são feitas no webservice através da classe java “MatlabEngine” que recupera as sessões compartilhadas do MatLab presentes no computador. A fim de evitar os problemas de concorrência descritos na seção 3.8.2, foi adotada a seguinte abordagem: as conexões com o Matlab foram todas armazenadas em uma pilha sincronizada, estrutura de dados onde o último elemento inserido é o primeiro a ser retirado e apenas uma única thread por vez pode inserir e retirar elementos, de tal forma que tentativas de retirar e colocar elementos nela, seja feito de forma sincronizada, isto é, apenas uma única thread tem acesso a pilha por vez.

Desta forma, quando uma *thread* está prestes a se conectar a uma sessão do Matlab, ela acessa de forma sincronizada a pilha, caso haja conexões disponíveis, a thread retira uma conexão da pilha e a usa por meio de uma operação “pop” (retirar do topo da pilha). Em seguida quando terminar seu trabalho, ela insere novamente aquela conexão na pilha por meio de uma operação “add” (adicionar ao topo da pilha). Desta forma, não há risco de duas threads acessarem a mesma sessão do MatLab, o que poderia gerar corrupção nos dados. Caso todas as conexões estejam ocupadas, a thread devolve uma mensagem de erro, para ser exibida ao usuário no aplicativo, alertando para tentar novamente em algum momento posterior. A Figura 27 exemplifica este procedimento de acesso a pilha sincronizada de sessões do Matlab, onde no exemplo em questão duas threads acessam a pilha de sessões Matlab contendo quatro sessões diferentes.

Figura 27 – Pilha de sessões Matlab sincronizada com as *threads* do *webservice*



FONTE: (AUTOR, 2019)

Como dito anteriormente o aplicativo acessa as funções do *webservice* através do protocolo HTTP. Na Figura 28 estão representadas as assinaturas, definição da função, de alguns dos métodos do webservice visíveis ao aplicativo.

Figura 28 – Cabeçalhos de alguns métodos de confiabilidade do webservice

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("PMCDI/post")
public String PMCDI(String content) {

@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("PMCECDI/post")
public String PMCECDI(String content)

@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("PMCRNDI/post")
public String PMCRNDI(String content)

@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("PFORMDI/post")
public String PFORMDI(String content)
```

FONTE: (AUTOR, 2019)

Como pode-se observar em todos os métodos ilustrados na Figura 28 é transmitido uma *string* como parâmetro, que é uma *string* no formato JSON, que contém todos os dados da requisição de forma estruturada. Isto é especificado por

meio da anotação “@Consumes”, que define o formato em que os dados são recebidos pelo método do *webservice*. Da mesma forma, quando o método do *webservice* obtém o resultado da rotina do Matlab, o método cria uma *string* em formato JSON, especificada por meio da anotação “@Produces”, que define o formato dos dados de saída, contendo o resultado e a retorna para o usuário.

4.3 COMPONENTE DO APLICATIVO

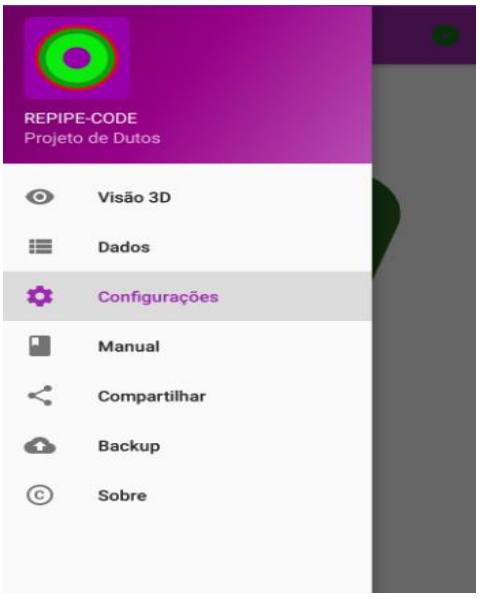
4.3.1 Interface Gráfica do aplicativo e Material design

Outro conceito importante quando se projeta uma interface gráfica, é justamente como construí-la. O kit de desenvolvimento de software (SDK) do Google fornece vários controles padrões para criação de interfaces gráficas no seu ambiente de desenvolvimento integrado (IDE) padrão, o Android Studio. Porém mesmo com estes controles, é necessário algo que guie no projeto de interfaces. Para isso o Google fornece uma especificação conhecida como material design, que se baseia em maximizar a experiência do usuário, tornando-a mais intuitiva, eficiente e produtiva.

Conforme (RALLO, 2019) “O Material Design tem como objetivo sintetizar os conceitos clássicos de um bom design com a inovação e possibilidades trazidas com a tecnologia e a ciência”. Os princípios básicos do Material Design abrangem vários tópicos como sombras, sobreposição de elementos gráficos, movimentação (animações), cor e iconografia (desenho dos ícones do aplicativo). O presente trabalho visou seguir os princípios básicos do Material Design, a fim de proporcionar uma melhor experiência de usuário, através de uma GUI moderna e intuitiva.

O projeto da GUI do aplicativo visou facilitar a captação de informações do duto e de seus defeitos de forma simplificada. Conforme podemos observar na Figura 29 o aplicativo possui algumas opções em sua “gaveta”, menu lateral do aplicativo:

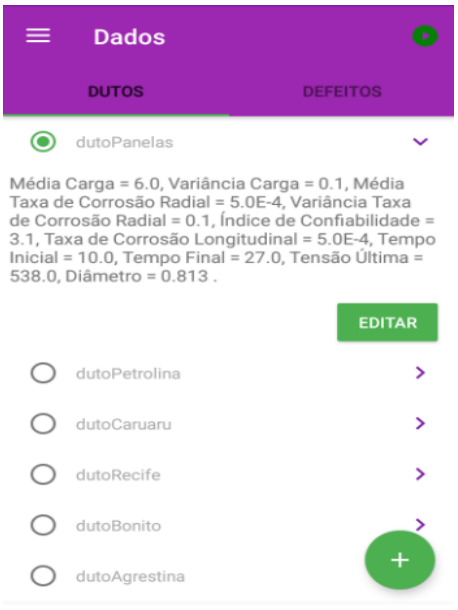
Figura 29 – “Gaveta” do aplicativo



FONTE: (AUTOR, 2019)

A opção “Dados” no aplicativo é a mais importante do aplicativo, pois permite captar os dados do problema. É onde ocorre a gravação das informações dos dutos e de seus defeitos. Uma vez que vários dutos podem ser avaliados ao longo de um determinado período, optou-se por permitir o armazenamento das informações de vários dutos no aplicativo, e seu acesso é feito através de uma lista ilustrada na Figura 30.

Figura 30 – Lista de dutos cadastrados



FONTE: (AUTOR, 2019)

Ao se criar um novo duto através do ícone representado pelo sinal “+”, conforme mostrado na Figura 30, deve-se preencher as informações da tela seguinte representada na Figura 31.

Figura 31 – Tela de adicionar/alterar um duto

FONTE: (AUTOR, 2019)

Similar à lista de dutos, que permite o acesso às informações dos dutos cadastrados no aplicativo, existe a lista de defeitos para cada duto, que permite acessar as informações dos defeitos pertencentes a um duto registrado. Na aba “Defeitos” da Figura 30 é exibida uma lista dos defeitos do duto selecionado, cujo botão está preenchido com verde na lista de dutos. Da mesma forma a criação de novos defeitos para o duto selecionado é feita através do ícone com o sinal “+”, e com o preenchimento das informações da tela seguinte representada na Figura 32, no que diz respeito às variáveis aleatórias e paramétricas do defeito.

Figura 32 – Tela de adicionar/alterar defeitos iguais alinhado longitudinalmente

FONTE: (AUTOR, 2019)

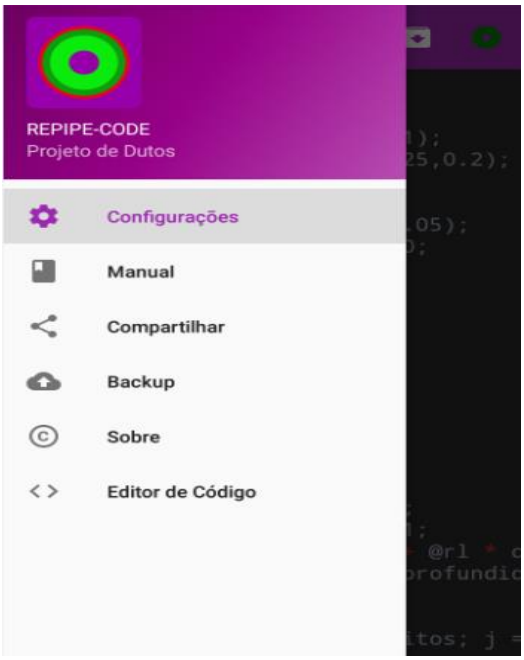
A opção “Visão 3d” permite visualizar de forma tridimensional a representação do duto atualmente selecionado juntamente com seus defeitos de corrosão. Na Figura 33 é exemplificado um duto com 5 defeitos de mesmas dimensões alinhados longitudinalmente ao longo do duto.

Figura 33 – Representação 3d de 5 defeitos iguais alinhados longitudinalmente

FONTE: (AUTOR, 2019)

Como dito anteriormente, o aplicativo possui um compilador para a linguagem “AnderScript”, desenvolvida neste trabalho para resolver problemas de confiabilidade estrutural. Esta opção de se programar no aplicativo se torna visível ao ativar o modo desenvolvedor nas configurações conforme foi ilustrado anteriormente na Figura 23. Após esta opção ser ativada, a “gaveta” do aplicativo, um menu lateral, é alterada conforme ilustra a Figura 34.

Figura 34 – Gaveta do aplicativo no modo desenvolvedor



FONTE: (AUTOR, 2019)

Esta modificação da gaveta do aplicativo acontece por questões de semântica para refletir a mudança na forma de especificar os dados e a função de falha, que neste caso é feita pelo usuário. A linguagem de programação AnderScript foi feita para resolver problemas de confiabilidade estrutural de forma geral, ou seja, não é especifico para problemas de corrosão de dutos. Esta nova gaveta retira as opções “Visão 3d” e “Dados”, e adiciona a opção “Editor de Código” para que se possa programar na linguagem “AnderScript”.

Desta forma, os dados do problema são especificados via código, bem como a definição da função de falha e as variáveis do problema. Para executar um problema de confiabilidade, é necessário apenas clicar no botão localizado na extremidade superior direita da tela, presente nas telas “Dados”, “Visão 3d” e “Editor de Código”.

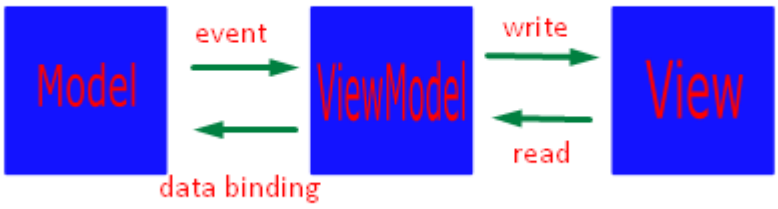
Em seguida as informações do problema são enviadas para o *WebService*, que por sua vez ao terminar o processamento da tarefa, retorna o resultado para o aplicativo.

4.3.2 Padrão de projeto de software MVVM

Atualmente temos vários “Design Patterns”, ou padrões de projeto, que são soluções amplamente aceitas e eficientes para problemas recorrentes no desenvolvimento de *software*. Os padrões de projeto modelam a criação, estruturação e comportamento de instancias de componentes nos programas, favorecendo desta forma a reutilização de *software*, a legibilidade, fácil manutenção e aumento considerável da produtividade no projeto de *software*.

Um deste padrões é o MVVM, que modela a forma como se dá a relação entre a visão (Interface gráfica) e o modelo (os dados e as regras do aplicativo). Segundo Nunes (2017) o padrão MVVM define basicamente 3 componentes: a *View*, o *Model*, e o *ViewModel*, estes 3 componentes podem ser visualizados na Figura 35.

Figura 35 – Representação gráfica do padrão MVVM



FONTE: (AUTOR, 2019)

Abaixo está uma explicação detalhada sobre cada um destes componentes:

- ✓ *Model*: Implementação do modelo de domínio da aplicação que inclui o modelo de dados, regras de negócio e validações de lógica.
- ✓ *View*: Entidade responsável por definir a estrutura, layout e aparência do que será exibido na tela
- ✓ *ViewModel*: Ele age como intermediário entre a *View* e o *Model*, e é o responsável por manusear o *Model* para ser utilizado pela *View*. Ele utiliza o

databinding, técnica que mantém os dados da aplicação sincronizados com a interface gráfica*,* para notificar mudanças aos observadores (*View*).

O padrão MVVM por sua vez, utiliza o padrão observador, para que uma mudança nos dados seja notificada automaticamente para todos os componentes que estejam interessados naquele dado. Desta forma, qualquer acesso aos dados na aplicação, obtém-se sempre a versão mais atualizada. No presente trabalho, isso possibilitou que as mudanças nas informações de dutos e de defeitos de corrosão, fossem instantaneamente representadas na GUI do aplicativo, bem como na visualização tridimensional.

4.4 COMPONENTE DO AMBIENTE TRIDIMENSIONAL

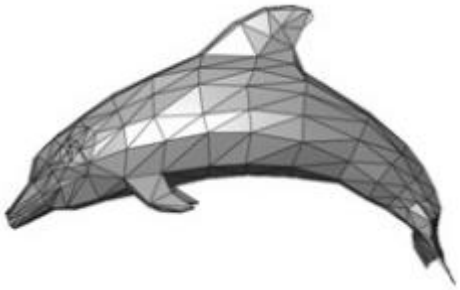
O Unity é uma *engine*, programa para simplificar o desenvolvimento de aplicações gráficas, para construção de jogos e simulações em ambientes tridimensionais. No presente trabalho, o ambiente tridimensional do aplicativo foi implementado como um projeto no *software* Unity separado do projeto do aplicativo, no software Android Studio. O passo seguinte foi exportar o projeto no software Unity como uma biblioteca Java e integra-la ao projeto do aplicativo. Apesar de ser importado para o aplicativo, a interface de comunicação entre as duas tecnologias não é tão trivial, conforme será discutido a seguir.

O ambiente 3d integrado ao aplicativo possui rotinas e gerenciamento próprio, como a renderização do ambiente tridimensional e o gerenciamento das entidades 3d. Ou seja, ele é uma “caixa preta” que possui funcionamento predefinido, portanto não é facilmente modificável. A solução é criar no ambiente 3d rotinas que recebem um único argumento na forma de uma *String*. Toda informação passada do aplicativo para o ambiente 3d deve ser textual. Desta forma, quando o usuário cria um novo duto ou defeito no aplicativo, este recolhe estas informações e as converte em um único texto, por fim as envia para o método correspondente do ambiente 3d. Este por sua vez extrai as informações do texto, e atualiza as malhas do duto e dos defeitos de corrosão.

4.4.1 Construção das malhas

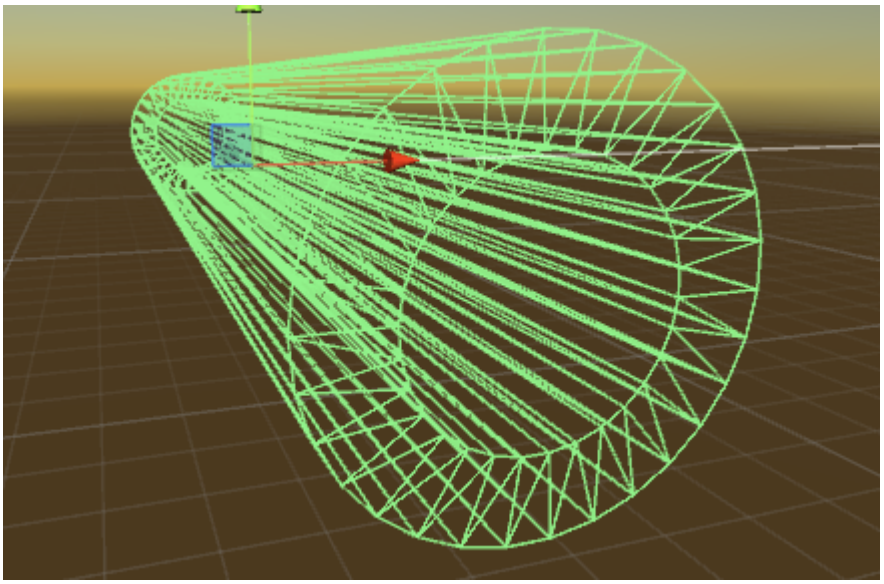
A construção das malhas, dutos e defeitos, foi feita utilizando elementos triangulares. Em computação gráfica, qualquer objeto tridimensional pode ser modelado utilizando essa primitiva gráfica conforme ilustra a Figura 36. A representação do duto em termos de triângulos por sua vez é ilustrada na Figura 37.

Figura 36 – Malha Gráfica



FONTE: (WIKIMEDIA FOUNDATION, 2018)

Figura 37 – Triângulos formadores do cilindro vazado



FONTE: (AUTOR, 2019)

Para se construir uma malha de elementos tridimensionais, deve se especificar os vértices desta malha por meio de um vetor, bem como as conexões destes vértices responsáveis por formar a malha (UNITY TECHNOLOGIES, 2019). Neste vetor de valores reais, cada 3 valores correspondem um vértice, onde estes 3 valores são as

coordenadas x, y e z deste vértice. A Figura 38 exemplifica a estrutura desse vetor entre as linhas 26 e 35.

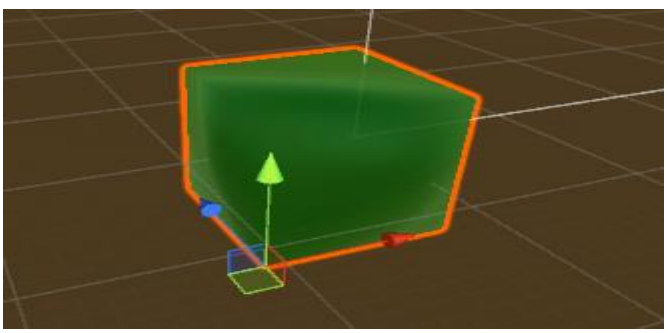
Figura 38 – Função responsável por criar um cubo na engine Unity

```
22
23      public void createCubo(float cubeX, float cubeY, float cubeZ)
24      {
25          Vector3[] vertices = new Vector3[8];
26
27          vertices[0] = new Vector3(0, 0, 0);
28          vertices[1] = new Vector3(cubeX, 0, 0);
29          vertices[2] = new Vector3(0, cubeY, 0);
30          vertices[3] = new Vector3(0, 0, cubeZ);
31          vertices[4] = new Vector3(cubeX, 0, cubeZ);
32          vertices[5] = new Vector3(cubeX, cubeY, 0);
33          vertices[6] = new Vector3(0, cubeY, cubeZ);
34          vertices[7] = new Vector3(cubeX, cubeY, cubeZ);
35
36          int[] triangulos = new int[] {
37              0, 2, 1, 1, 2, 5,
38
39              3, 0, 1, 1, 4, 3,
40
41              0, 3, 2, 2, 3, 6,
42
43              1, 5, 4, 5, 7, 4,
44
45              6, 3, 4, 6, 4, 7,
46
47              6, 5, 2, 7, 5, 6
48          };
49
50          cuboMesh.Clear();
51          cuboMesh.vertices = vertices;
52          cuboMesh.triangles = triangulos;
53          cuboMesh.RecalculateNormals();
54
55      }
```

FONTE: (AUTOR, 2019)

Além de especificar os vértices da malha, deve-se especificar como estes vértices formam a malha. Afinal existem infinitas possibilidades de ligação entre os vértices, que por sua vez originam infinitas configurações de malhas. A especificação da malha é feita por um segundo vetor, o vetor de índices, onde cada 3 números correspondem um triângulo. Os valores deste segundo vetor são os índices dos vértices que forma os triângulos (UNITY TECHNOLOGIES, 2019). O código da Figura 38 ao ser executado cria o cubo representado na Figura 39:

Figura 39 – Cubo gerado via código da Figura 38



FONTE: (AUTOR, 2019)

Na Figura 38 podemos observar que entre a linha 27 e a linha 34 ocorre as definições dos vértices do cubo, cujo tamanho do vetor “vertices” é de 8 (cubo). Já o vetor “triângulos” possui dimensão de 36, pois um cubo possui 6 lados, cada lado é um retângulo que pode ser representado por 2 triângulos, e cada triangulo é especificado por meio de 3 índices no vetor de vértices, logo 6x2x3 resulta em 36 índices para especificar todos os triângulos do cubo. A especificação dos triângulos é feita entre as linhas 36 e 48.

Como observado anteriormente na Figura 33 os defeitos foram criados como “manchas” sobre a malha do duto, ao invés de buracos na malha do duto, isto foi feito para que as malhas dos defeitos fossem independentes da malha do duto. De fato, as malhas dos defeitos possuem um pequeno espaçamento em relação à malha do duto, pois caso ficassem sobrepostas, as duas malhas provocariam efeitos estranhos na renderização.

4.5 COMPONENTE DO COMPILADOR

Como dito anteriormente, o compilador do aplicativo foi desenvolvido para estender as capacidades do sistema. O analisador léxico da linguagem foi implementado utilizando expressões e definições regulares, conforme explicado anteriormente na seção 3.7.2, através da biblioteca java chamada JFlex. Já o analisador sintático foi definido por meio de gramatica livre de contexto utilizando a biblioteca Java CUP. Não houve otimização de código, tornando o código mais rápido conforme discutido na seção 3.7.8, pois nem mesmo teve geração de código intermediário, discutido brevemente na seção 3.7.7, a compilação converte diretamente o código “AnderScript” para código MatLab.

4.5.1 Sintaxe da linguagem AnderScript

A abordagem adotada visou simplificar a escrita de problemas de confiabilidade estrutural para um único elemento estrutural, sem correlação entre as variáveis aleatórias do problema, e com apenas uma única variável de projeto. Na Tabela 3 são mostradas as palavras-chaves da linguagem, responsáveis por especificar expressões, funções, blocos de código e comandos.

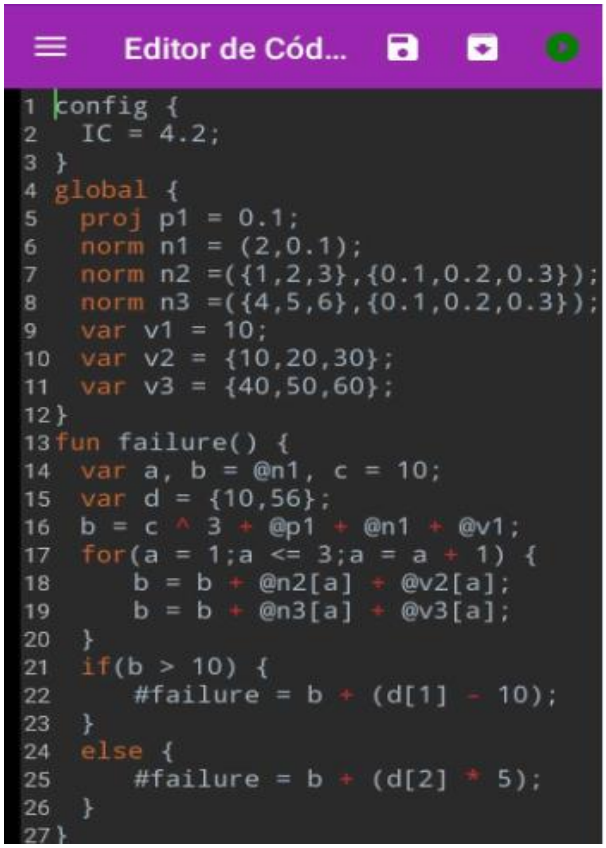
Tabela 3 - Palvras-Chave da linguagem AnderScript

config	global	fun
proj	norm	var
if	else	for

FONTE: (AUTOR, 2019)

Pode-se observar que são poucas palavras-chave se comparadas a uma linguagem convencional como a linguagem C. Na Figura 40 pode-se visualizar um exemplo de código escrito na linguagem AnderScript.

Figura 40 – Código exemplo na linguagem AnderScript



FONTE: (AUTOR, 2019)

Conforme pode-se observar, temos os blocos “config” e “global”, e a função “failure”. O bloco “config” (linhas 1 a 3 da Figura 40) serve apenas para configurar as variáveis do sistema. Em sua versão atual, o AnderScript possui apenas a variável de Índice de confiabilidade alvo (IC) como variável do sistema, que tem como padrão o valor 3,1. Como as variáveis do sistema possuem valores padrão, o bloco “config” é

opcional no código. O bloco global (linhas 4 a 12 da Figura 40) é responsável por definir e inicializar as variáveis aleatória e paramétricas do problema. Variáveis aleatórias com distribuição normal são definidas com a palavra-chave “norm” (linha 6 da Figura 40) e são iniciadas com os valores da média e desvio padrão da variável. Já as variáveis paramétricas são definidas utilizando “var” (linha 9 da Figura 40), e por fim as variáveis de projeto são definidas utilizando a palavra-chave “proj” (linha 5 Figura 40) e em sua inicialização recebem como entrada o valor do ponto de partida da variável de projeto. Atualmente, o compilador só permite a definição de apenas uma variável de projeto. As palavras-chaves “norm” e “var”, podem ser usadas para criar vetores de variáveis aleatórias e paramétricas, respectivamente, conforme ilustrado nas linhas 7 a 8 e 10 a 11 da Figura 40.

Na função “failure” (linhas 13 a 27 da Figura 40) é onde ocorre a definição da função de falha, que pode ser expressa em função das variáveis do bloco “global” e das variáveis de escopo local, corpo da função. Toda definição de função que tiver um retorno deve especifica-lo por meio de uma variável especial nomeada como “#NomeDaFunção”. No caso da função “failure”, essa variável se chama “#failure” (linhas 22 e 25 da Figura 40). Além disso, a variável de retorno e as variáveis definidas no escopo global, possuem usos restritos, afim de evitar diversos situações desnecessárias descritas a seguir.

As variáveis do bloco “global” nunca podem ser usadas no lado esquerdo de expressões de atribuição, com isso seus valores nunca são mudados diretamente pelo usuário. Caso seja uma variável paramétrica, permanecerá com os mesmos valores da sua criação, caso seja uma variável aleatória ou de projeto seu valor será gerenciado e atualizado pelo tempo de execução do algoritmo FORM, de acordo com seu tipo de distribuição no caso de variáveis aleatórias, e com o ponto de partida no caso de variáveis de projeto. No caso da variável de retorno, a mesma só pode figurar no lado esquerdo das atribuições, ou seja, só pode receber valores e não fazer parte de uma expressão.

Variáveis locais são definidas usando a palavra-chave “var”. Também podem ser definidos vetores com esta mesma palavra-chave. Os únicos comandos disponíveis na linguagem são “if”, “if-else” e “for”. Além disso, variáveis do escopo global são acessíveis de dentro das funções colocando-se um “@” antes de seu nome. Por exemplo a variável “n1” contida na Figura 40 (linha 6), a mesma é acessada na função “failure” usando “@n1” conforme pode-se observar na linha 16 da Figura 40.

Isso foi definido para impedir que variáveis de mesmo nome no escopo local, declaradas dentro de funções, ocultasse variáveis do bloco “global”.

Os operadores da linguagem englobam as 4 operações básicas (+, -, *, /), além do operador de exponenciação(^), os operadores relacionais (<, >, <=, >=, == e !=) e lógicos(|| e &&).

4.5.2 Compilação para o MatLab

O código aqui denominado de AnderScript, linguagem implementada pelo autor do presente trabalho, permite uma escrita mais fácil para problemas de confiabilidade estrutural, uma vez que o usuário precisa apenas especificar as variáveis do problema e a definição da função de falha. Detalhes como atualização das variáveis aleatórias, o método de confiabilidade utilizado, transformação das variáveis para o espaço reduzido e busca ao ponto de projeto são abstraídos do usuário. Para que tudo isso funcione o código convertido para a linguagem do Matlab segue algumas convenções explicadas posteriormente. Na Figura 41 é mostrado a compilação do código da Figura 40.

Figura 41 – Código compilado para o MatLab

```
1   a=0
2   b=ALEATORIA_(1)
3   c=10.0
4   d=[10.0, 56.0]
5   b=c^3.0+var_project+ALEATORIA_(1)+vetor_values(1)
6   a=1.0
7   while (vpa(subs(a,ALEATORIA_,U_'),6)<=3.0)
8   b=b+ALEATORIA_(round(a+1))+vetor_values(round(a+1))
9   b=b+ALEATORIA_(round(a+4))+vetor_values(round(a+4))
10  a=vpa(subs(a,ALEATORIA_,U_'),6)+1.0
11  end
12  if (vpa(subs(b,ALEATORIA_,U_'),6)>10.0)
13  GU_=b+(d(1.0)-10.0)
14  else
15  GU_=b+(d(2.0)*5.0)
16  end
```

FONTE: (AUTOR, 2019)

Na Figura 42 é observado o trecho de código inicial, da função Matlab responsável por realizar a análise de confiabilidade estrutural para problemas que usam o compilador AnderScript (linguagem elaborada no presente trabalho).

Figura 42 – Trecho de código da função Matlab que dá suporte ao AnderScript

```
function [BETA_SINAL_] = BSPConfCodigo(var_project,vetor_media, vetor_desvio_padrao, vetor_values, string_codigo, DELTA)
    NUMIT_=5; % NÚMERO TOTAL DE ITERAÇÕES DO FORM
    U_= vetor_media'; %VETOR U TERÁ VALORES NUMERICOS
    tam_=length(U_);%TAM REPRESENTA A QUANTIDADE DE VARIÁVEIS ALEATÓRIAS
    ALEATORIA_ = sym('U_', [1 tam_]);
```

FONTE: (AUTOR, 2019)

Na definição da função mostrada na Figura 42, o parâmetro “var_project” recebe o ponto de partida da variável de projeto. O parâmetro “vetor_media” recebe todas as médias definidas nas variáveis “norm”, o parâmetro “vetor_desvio_padrao” recebe todos os desvios padrões das variáveis “norm”, o parâmetro “vetor_values” recebe os valores das variáveis paramétricas definidas no bloco “global” e por fim o parâmetro “string_codigo” recebe o código Matlab compilado na forma textual.

O código Matlab na forma textual pode ser executado utilizando o comando “eval” do MatLab. Um exemplo é a execução do comando “for” na forma textual conforme é ilustrado na Figura 43, onde é ilustrada a siada do comando, que consiste na impressão do valor da variável “i” em cada iteração.

Figura 43 – Execução de comando na forma textual

```
>> comando = 'for i=1:3 i \n end ';
>> eval(char(sprintf(comando)))

i =

     1


i =

     2


i =

     3
```

FONTE: (AUTOR, 2019)

Como o código é compilado para ser executado no método FORM, é necessária a criação de variáveis simbólicas do MatLab, uma para cada variável aleatória. As referências às variáveis do bloco “global” e à variável de retorno, variável

que representa o retorno (resultado) de uma rotina, são convertidas para referências às variáveis definidas na função da Figura 42, anteriormente citada, e seguem as seguintes regras:

- ✓ Nas expressões condicionais dos comandos “if”, “if-else” e “for”, além das expressões de inicialização e incremento do comando "for", as variáveis definidas no bloco “global” são convertidas para valores reais, conforme ilustrado na linha 17 da Figura 40, que é compilada para linha 7 da Figura 41. Em todos os outros casos são usados os valores simbólicos. Isso é feito porque operações de comparação por exemplo exigem os valores reais e não simbólicos, conforme ilustrado na linha 18 da Figura 40, que é compilada para linha 8 da Figura 41;
- ✓ As variáveis aleatórias são convertidas para índices específicos do vetor “ALEATORIA_” que representa um vetor de variáveis simbólicas, conforme ilustrado na linha 14 da Figura 40, onde a variável “b” é compilada para linha 2 da Figura 41;
- ✓ A variável de retorno da função “failure” é convertida para a variável “GU_” que é a representação simbólica da função de falha, conforme ilustrado na linha 22 da Figura 40 que é compilada para linha 13 da Figura 41;
- ✓ Nas expressões que necessitam de valores reais, como dito anteriormente, as referências às variáveis aleatórias são convertidas para índices da variável “U_” que representa o vetor do valor real da variável aleatória na iteração atual;
- ✓ Nas expressões que necessitam de valores reais, as referências às variáveis locais que consistam em expressões que contenham variáveis aleatórias, são convertidas em “vpa(subs(“VariavelLocal”,ALEATORIA_,U_'),6)”, para assim obter o valor real associado a variável, conforme ilustrado na linha 21 da Figura 40 que é compilada para linha 12 da Figura 41;
- ✓ As variáveis paramétricas definidas no bloco “global” são convertidas para referências às posições da variável “vetor_values conforme ilustrado na linha 16 da Figura 40 que é compilada para linha 5 da Figura 41;
- ✓ As variáveis previamente definidas na função da Figura 42 criada no Matlab, possuem o sufixo “_”, e as variáveis da linguagem AnderScript só aceitam caracteres alfanuméricos. Desta forma, não ocorre o risco que o usuário oculte uma variável do tempo de execução.

As variáveis locais quando são convertidas do código AnderScript para o código MatLab permanecem com o mesmo nome. Além disso, todas as funções do MatLab estão acessíveis no código AnderScript. Por fim, a linguagem trabalha apenas com variáveis ou vetores, a definição e o uso de matrizes não são suportados. Em sua versão atual, a linguagem AnderScript aceita a definição de uma função apenas, a função “failure”.

4.5.3 Recuperação de Erros

O compilador AnderScript possui algumas formas de recuperação de erros, a fim de se evitar que certos problemas ocorram durante a execução no servidor. Os seguintes erros são sinalizados em tempo de compilação:

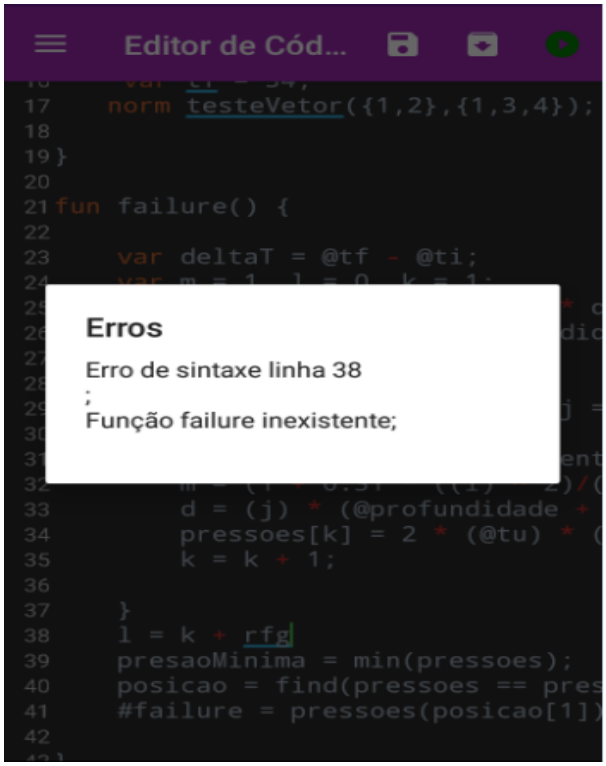
- ✓ Criar mais de uma variável de projeto;
- ✓ Criar uma ou mais variáveis no bloco global com o mesmo nome;
- ✓ Criar uma ou mais variáveis locais na mesma função com o mesmo nome;
- ✓ Erros de sintaxe;
- ✓ Na criação de vetores de variáveis aleatórias, especificar um vetor com valores de media com tamanho diferente do vetor de desvios padrões;
- ✓ Referenciar uma variável não declarada ou inexistente;
- ✓ Tentar mudar o valor de uma variável do bloco “global”;
- ✓ Não definir a função principal “failure”, função onde ocorre a definição da função de falha, sem argumentos.
- ✓ Criar 2 ou mais funções com o mesmo nome,

Quando o erro é encontrado, o aviso é exibido no aplicativo e o envio dos dados ao servidor não acontece até que o usuário corrija todos os erros do código. As Figura 44 e Figura 45 exemplificam algumas possíveis mensagens de erros.

Figura 44 – Exemplificação de possíveis erros no editor de código

FONTE: (AUTOR, 2019)

Figura 45 - Exemplificação de possíveis erros no editor de código



FONTE: (AUTOR, 2019)

4.6 PARÂMETROS ADOTADOS PARA O DUTO

Nesse trabalho, será utilizado para a avaliação de confiabilidade o duto corroído apresentado na literatura por AHAMMED (1998). Sendo, que esses valores, que estão apresentados na Tabela 4, correspondem aos dados da primeira inspeção da corrosão, que ocorreu em um tempo T0 = 10 anos. Vale salientar, que o duto escolhido foi adaptado ao problema. Isso porque, no problema original não possui a informação da variável aleatória distância entre defeitos, de forma que teve de ser arbitrado um valor e uma distribuição de probabilidade para essa variável (ROCHA, 2016).

A princípio foram consideradas nove variáveis aleatórias conforme ilustrado na Tabela 4.

Tabela 4 – Variáveis aleatórias

Variável	Distribuição	Média	Coef. Variação
Profundidade do defeito(d)	Normal	3 mm	0,1
Pressão interna(P_s)	Normal	5 MPa	0,1
Taxa de corrosão radial(R_d)	Normal	0,1 mm/ano	0,2
Espessura do duto(t)	Normal	10 mm	0,05
Tensão última do material(σ_u)	Lognormal	538 MPa	0,067
Diâmetro do duto(D)	Normal	600 mm	0,03
Comprimento do defeito(l)	Normal	200 mm	0,05
Taxa de corrosão longitudinal(R_l)	Normal	0,1 mm/ano	0,2
Distância entre defeitos(s)	Normal	25 mm	0,05

FONTE: Adaptado de Ahammed (1998)

O primeiro passo será realizar a análise de sensibilidade das variáveis aleatórias do problema por meio do método FORM, a fim de determinar as que mais contribuem para a quantificação da probabilidade de falha. Isso por sua vez vai diminuir o número de variáveis aleatórias do problema, já que as de menor fator de importância serão transformadas em variáveis paramétricas. O número reduzido de variáveis aleatórias no problema, por sua vez, aumenta a performance dos métodos de confiabilidade, pois no caso do FORM, o tamanho das matrizes no método, ficam

menores. No caso dos métodos baseados no Monte Carlo o tamanho de cada amostra diminui. Por último no caso do treinamento da rede neural, o número de entradas na rede diminui tornando o treinamento da mesma mais rápido.

5 RESULTADOS E DISCUSSÃO

Os algoritmos FORM e Monte Carlo foram adaptados (TORRES, 2009) para o caso em estudo. O modelo empírico de pressão de falha proposto pela norma BS-7910 (2005) foi implementado para o caso de múltiplos defeitos de dimensões iguais alinhados longitudinalmente. Também foram calculados os fatores de importância das variáveis aleatórias envolvidas, por meio do algoritmo FORM.

5.1 ANÁLISE DOS FATORES DE IMPORTÂNCIA

Para os dados fornecidos pela Tabela 4, e considerando apenas 2 defeitos interagentes, foram calculados os fatores de importância utilizando a equação (28) na execução do método FORM para diversos tempos de inspeção atual, conforme ilustrado na Tabela 5:

Tabela 5 – Fatores de importância para as variáveis aleatórias

Variável	T=20 anos	T=30anos	T=40anos	T=50anos
Profundidade do defeito	0,1325	0,1242	0,1015	0,07843
Pressão interna	0,1613	0,1151	0,0785	0,0546
Taxa de corrosão radial	0,0589	0,2208	0,4062	0,5577
Espessura do duto	0,5163	0,4582	0,3634	0,2762
Tensão última do material	0,1096	0,0670	0,0404	0,0253
Diâmetro do duto	0,0180	0,0107	0,0063	0,0040
Comprimento do defeito	0,0033	0,0036	0,0032	0,0026
Taxa de corrosão longitudinal	0,0000	0,0000	0,0000	0,0000
Distância entre defeitos	0,00002	0,0003	0,0004	0,0003

FONTE: (AUTOR, 2019)

Observando a Tabela 5, nota-se que as variáveis profundidade do defeito, pressão interna, taxa de corrosão radial e espessura do duto, apresentam os maiores valores nos períodos entre 20 e 50 anos para a inspeção atual. Apesar de a tensão última do material, apresentar um valor do fator de importância relativamente alto, com o passar do tempo, o seu valor decai bastante se tornando insignificante. A taxa de corrosão longitudinal por sua vez apresentou um valor muito baixo. Porém nos resultados, a precisão adotada foi de quatro casas decimais, por isso o valor está representado como sendo nulo para a taxa de corrosão longitudinal

Desta forma, no presente trabalho foram adotadas como variáveis aleatórias: profundidade do defeito, pressão interna, taxa de corrosão radial e espessura do duto.

5.2 ANÁLISE DE CONFIABILIDADE COM A GUI DO APLICATIVO

Foram feitas análises de confiabilidade para os quatro métodos estudados (FORM, Monte Carlo, Monte Carlo com Esperança Condicionada e Monte Carlo com Redes Neurais) considerando dois defeitos interagentes, afim de avaliar o duto descrito por Ahammed (1998). Na Tabela 6 estão resumidos os resultados dos índices de confiabilidade em função do tempo da última atual:

Tabela 6 – Índice de Confiabilidade em função do tempo atual de inspeção

Inspeção Atual (anos)	**20**	**30**	**40**	**50**
FORM	6,1299	4,2983	2,6539	1,3403
MC	Infinito	4,1075	2,6462	1,3409
MCEC	7,9088	4,2952	2,6509	1,3446
MCRN	6,6558	4,3446	2,6156	1,3396

FONTE: (AUTOR, 2019)

Conforme pode-se observar na Tabela 6, o índice de confiabilidade a tende a diminuir com o passar do tempo. De fato, à medida que o tempo passa o processo de corrosão tem maior ação na estrutura, diminuindo a sua capacidade resistente, acarretando desta forma, uma diminuição na segurança da estrutura. Isto por sua vez diminui o índice de confiabilidade. Todos os quatro métodos apresentaram resultados muito próximos nos tempos 30 a 50 anos, porém no tempo 20 anos houve uma certa

diferença. De fato, para o tempo de 20 anos, o índice de confiabilidade retornado pelos métodos, equivale a uma probabilidade de falha muito baixa (5.8167e-10 para o método FORM), tão baixa que o método de Monte Carlo chegou a retornar um índice de confiabilidade infinito.

Outra informação útil tirada da Tabela 6 é com relação a vida útil do duto analisado. Adotando-se um índice de confiabilidade alvo para o duto, é possível estimar o tempo de vida remanescente o mesmo. Considerando um índice de confiabilidade alvo de 3.1, o tempo de vida útil do duto está entre 40 e 30 anos. Algumas analise foram feitas no intervalo de 30 a 40 anos, cujos resultados estão ilustrados na Tabela 7.

Tabela 7 – Índice de confiabilidade no intervalo de 30 a 40

Inspeção Atual (anos)	**35**	**36**	**37**	**38**
FORM	3.4379	3.2745	3.1143	2.9575

FONTE: (AUTOR, 2019)

O valor mais próximo do índice de confiabilidade alvo é 3,1143 que corresponde ao tempo de inspeção atual de 37 anos. Como o tempo da última inspeção é de 10 anos, logo a vida remanescente do duto é de 27 anos (37 – 10 anos).

Em seguida, foi feita a análise de confiabilidade para os métodos citados anteriormente. Desta vez em relação ao número de defeitos, fixando o tempo da atual inspeção em 35 anos. Na Tabela 8, estão representados estes resultados:

Tabela 8 – Índice de confiabilidade em função do número de defeitos

Nº Defeitos	**2**	**3**	**4**	**5**
FORM	3,4379	3,3053	3,2397	3,2012
MC	3,4600	3,2794	3,2534	3,1747
MCEC	3,5409	3,2991	3,2331	3,1571
MCRN	3,4221	3,2822	3,1458	3,0940

FONTE: (AUTOR, 2019)

Conforme observa-se na Tabela 8, o índice de confiabilidade a tende diminuir à medida que o número de defeitos aumenta. Como citado anteriormente, neste trabalho, à medida que o número de defeitos interagente aumenta, também aumenta o efeito de interação entre os defeitos de corrosão, causando desta forma uma redução ainda mais significativa da resistência do duto, se comparada ao efeito causado por defeitos isolados. Além disso, ambos os quatro métodos apresentam resultados muito próximos, enquanto que o Monte Carlo com Redes Neurais apresenta os menores valores se comparado aos outros, o que caracteriza uma abordagem mais conservadora, favorecendo desta forma a segurança.

5.3 PROJETO DE CONFIABILIDADE COM A GUI DO APLICATIVO

Como citado anteriormente, os mesmos métodos de análise de confiabilidade foram também utilizados no projeto baseado em confiabilidade. Para que isso fosse possível, como visto anteriormente na seção 4.1.3, estes métodos são chamados pelo método de Newton-Raphson, afim de se calcular o valor da variável de projeto.

Primeiramente foi projetada a espessura do duto variado o número de defeitos interagentes. Neste caso, adotou-se os valores de 37 anos para a inspeção atual, e o valor de 3,1 para o índice de confiabilidade alvo. Estes resultados estão representados na Tabela 9.

Tabela 9 – Espessura ótima em função do número de defeitos

Nº Defeitos	**2**	**3**	**4**	**5**
FORM	9,4286	9,5318	9,5838	9,6147
MC	9,2832	9,3205	9,3608	9,4456
MCEC	9,2628	9,3500	9,3992	9,4567
MCRN	9,2370	9,3279	9,3494	9,4355

FONTE: (AUTOR, 2019)

Como discutido na análise de confiabilidade, o aumento no número de defeitos interagentes, diminui o índice de confiabilidade da estrutura. No projeto baseado em confiabilidade, isto é refletido no aumento da espessura, pois para se manter um índice de confiabilidade alvo fixo, ao aumentar o número de defeitos diminui a segurança da estrutura. Para combater essa diminuição do índice de confiabilidade, o

método de confiabilidade utilizado, tende a aumenta a espessura do duto, o que por sua vez aumenta a capacidade resistente do mesmo.

Na Tabela 9, é possível perceber que realmente em todos os 4 métodos utilizados, a espessura é incrementada à medida que o número de defeitos interagentes aumenta. Porém, apesar de próximos, os valores da espessura para os 4 métodos apresentam divergências em termos de precisão. Enquanto que, o método FORM apresenta os maiores valores de espessura, portanto apresentando uma abordagem mais conservadora, já que está a favor da segurança, os outros métodos, relacionados ao método de Monte Carlo, apresentam resultados um pouco inferiores, porém parecidos entre si.

Em seguida, foi feito o projeto baseado em confiabilidade variando o índice de confiabilidade alvo. Para este problema, foram fixados o tempo da inspeção atual como sendo 35 anos, e o número de defeitos como sendo igual a 3. Os resultados estão listados na Tabela 10.

Tabela 10 – Espessura ótima em função do índice de confiabilidade alvo (B_{alvo})

B_{alvo}	3,1	3,3	3,7
FORM	9,2687	9,3820	9,6085
MC	9,3613	9,4483	9,7318
MCEC	9,3636	9,3660	9,3734
MCRN	9,3176	9,4661	NaN

FONTE: (AUTOR, 2019)

Conforme representado na Tabela 10, em todos os métodos utilizados, à medida que o índice de confiabilidade aumenta, a espessura do duto também aumenta. De fato, um índice de confiabilidade maior fornece uma maior segurança da estrutura, e para aumentar esta segurança é preciso aumentar a espessura do duto como esperado. Observando-se os dados contidos na Tabela 10, novamente percebe-se que os métodos FORM e Monte Carlo apresentaram resultados consistentes, pois a espessura do duto cresce em função do número de defeitos, apesar de não tão próximos.

Em contrapartida, o método de Monte Carlo com Esperança Condicionada apresentou pouca variabilidade em função do aumento do índice de confiabilidade

alvo, enquanto que o método de Monte Carlo com Redes Neurais para um aumento do índice de confiabilidade alvo, apresentou sérias dificuldades para encontrar a variável de projeto, a ponto de não retornar nenhum resultado válido, conforme ilustra o resultado “NaN” que significa “não é número”, ou seja, nenhuma saída é gerada.

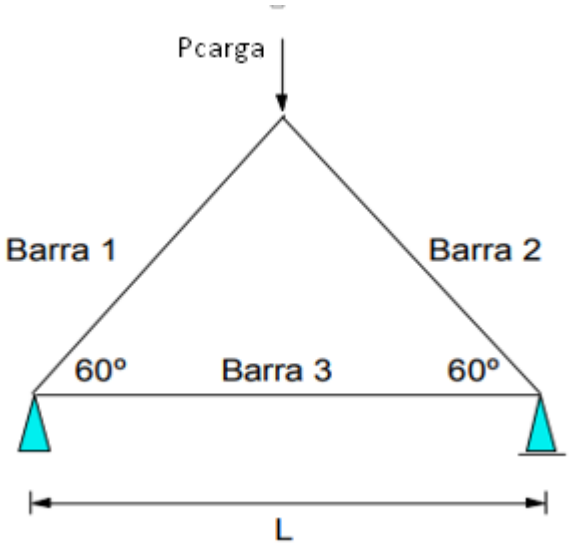
5.4 ANÁLISE DE CONFIABILIDADE USANDO ANDERSCRIPT

Para validar a análise de confiabilidade utilizando a linguagem AnderScript, foram utilizados dois exemplos apresentados por Barbosa (2004). O primeiro problema está relacionado a uma treliça isostática e o outro à uma viga engastada com carregamento distribuído. Ambos exemplos, servem para ilustrar que a linguagem AnderScript é adaptável a problemas de confiabilidade que não estejam relacionados a apenas dutos.

5.4.1 Problema 1: Treliça Isostática

Este exemplo proposto por Barbosa (2004) consiste em calcular a probabilidade de falhas para cada barra da treliça ilustrada na Figura 46. As variáveis aleatórias são a carga P_{carga} aplicada na treliça, e a Resistencia à compressão R das barras. As informações estatísticas destas variáveis estão representadas na Tabela 11:

Figura 46 – Treliça isostática do Problema 1



FONTE: (BARBOSA, 2004)

Tabela 11 – Variáveis aleatórias do Problema 1 (treliça isostática)

Variável	Distribuição	Média	Desvio Padrão
Carga (P_{carga})	Normal	14	1,25
Resistencia (R)	Normal	11	1,5

FONTE: Adaptado de Barbosa (2004)

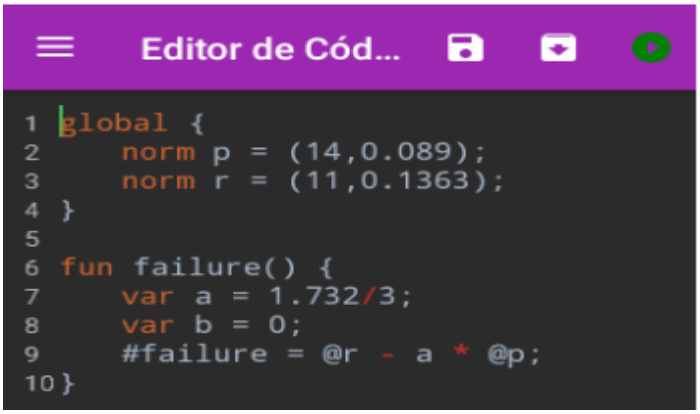
A função de falha é definida como a diferença entre resistência à compressão da barra pela solicitação axial na barra. Resolvendo a treliça da Figura 46 pelo o método dos nós, obtém-se os valores de esforços normais de $P_{carga}\sqrt{3}/3$ para as barras 1 e 2, e de $P_{carga}\sqrt{3}/6$ para a barra 3. Desta forma as funções de falha para as barras 1 e 2, e para a barra 3, podem ser representadas pelas equações (42) e (43), respectivamente:

$$G(U) = R - \frac{P_{carga}\sqrt{3}}{3} \tag{42}$$

$$G(U) = R - \frac{P_{carga}\sqrt{3}}{6} \tag{43}$$

Este problema é escrito através do aplicativo utilizando a linguagem “AnderScript” para as barras 1 e 2, e 3, conforme ilustrado nas Figura 47 e Figura 48 respectivamente:

Figura 47 - Código AnderScript para as barras 1 e 2 do Problema 1



FONTE: (AUTOR, 2019)

Figura 48 - Código AnderScript a barra 3 para do Problema 1

FONTE: (AUTOR, 2019)

Como pode-se observar nas Figura 47 e Figura 48, a variável “a” é equivalente ao termo $\sqrt{3}/3$ na equação (42), e ao termo $\sqrt{3}/6$ na equação (43). Além disso, como dito anteriormente, a definição de variáveis aleatórias na linguagem AnderScript requer a média e o coeficiente de variação de cada variável aleatória. Ao observar a Tabela 11, nota-se que estão representados apenas a média e o desvio padrão das variáveis aleatórias, não está ilustrado o coeficiente de variação necessário para especificar variáveis aleatórias na linguagem AnderScript. Conforme Barbosa (2004), o coeficiente de variação é obtido da relação entre o desvio padrão e a média. Essa relação gerou os valores de coeficiente de variação de 0.089 e 0.1363 apresentados nas Figura 47 e Figura 48. A comparação dos resultados está ilustrada na Tabela 12:

Tabela 12 - Comparação dos resultados para o Problema 1

Barra	**1 e 2**	**3**
Barbosa (2004)	0,039848	2,24e-6
AnderScript	0,039687	2,24e-6

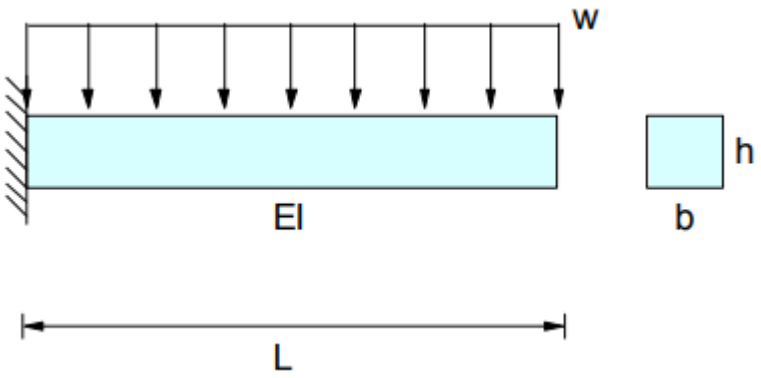
FONTE: (AUTOR, 2019)

Conforme pode se observar na Tabela 12, os resultados são bastante próximos, validando desta forma o algoritmo da linguagem AnderScript. Um desempenho muito bom, em termos de precisão numérica, se for levado em conta que foram programadas apenas algumas poucas linhas de código para resolver este problema.

5.4.2 Problema 2: Viga em balanço com carregamento distribuído

Barbosa (2004) propõe outro exemplo, desta vez para o cálculo do índice de confiabilidade da viga engastada representada na Figura 49. A estrutura é uma viga em balaço com um carregamento distribuído e seção retangular.

Figura 49 – Viga em balanço com comportamento linear elástico do Problema 2



FONTE: (BARBOSA, 2004)

A função de falha neste caso está relacionada à flecha transversal da extremidade da viga. O deslocamento não deve exceder o limite de serviço L/325, onde L é o comprimento da viga. A função de falha para este caso é representada pela (44):

$$G(U) = \frac{L_{viga}}{325} - \frac{wbL^4}{8EI} \tag{44}$$

Onde w é o carregamento distribuído, E é o módulo de elasticidade, I é o momento de inércia da seção da viga, b é largura da viga e h é a altura da viga. Barbosa (2004) considera o comprimento L com o valor de 6m, e o modulo de elasticidade E com o valor de $2{,}6.10^4$MPa. Aplicando estes valores, bem como o momento de inercia para seções retangulares ($I = bh^3/12$), obtém-se a equação (45) em função de w e h, cujas informações estatísticas estão representadas na Tabela 13:

$$G(U) = 18.461538 - 74769.2307\frac{w}{h^3} \tag{45}$$

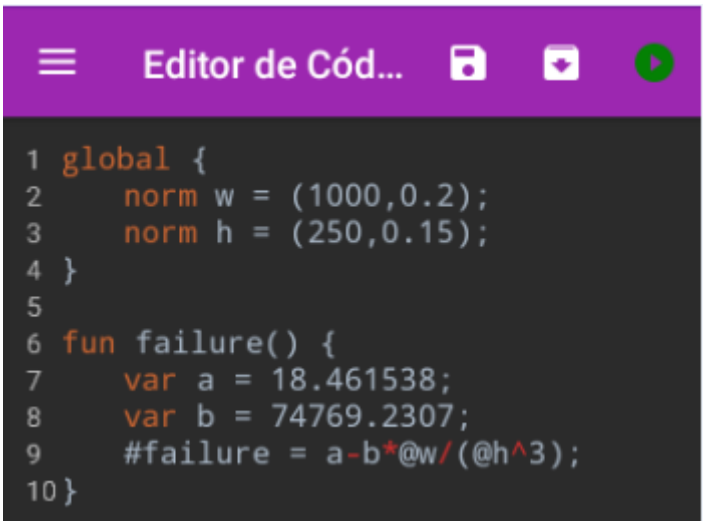
Tabela 13 – Variáveis aleatórias do Problema 2 (viga engastada)

Variável	Distribuição	Média	Coef. Variação
w	Normal	1000 N/m²	0,2
h	Normal	250mm	0,15

FONTE: Adaptado de Barbosa (2004)

A escrita deste problema na linguagem AnderScript está ilustrada no código da Figura 50:

Figura 50 – Código AnderScript para o Problema 2



FONTE: (AUTOR, 2019)

A comparação dos resultados obtidos por Barbosa (2004) e os obtidos pelo código da Figura 50 é ilustrada na Tabela 14.

Tabela 14 – Comparação dos resultados para o Problema 2

Barbosa	AnderScript
2.341	2.331

FONTE: (AUTOR, 2019)

A precisão dos resultados, valida o uso da linguagem AnderScript, que permitiu a resolução do problema com apenas 10 linhas de código conforme ilustrado na Figura 50.

5.5 PROJETO DE CONFIABILIDADE USANDO ANDERSCRIPT

Por fim, foi realizado o projeto de um duto utilizando a linguagem AnderScript. Como dito anteriormente a linguagem serve para resolver problemas de confiabilidade estrutural de um único componente estrutural, sem correlação entre as variáveis aleatórias, e com apenas uma única variável de projeto. Uma das propostas é resolver problemas de dutos com características diferentes, como por exemplo utilizar outra norma para o cálculo da pressão de falha, ou a escolha de outra variável de projeto que não seja a espessura, tal como a espessura.

Para este problema, foi projetado um duto com as mesmas características da Tabela 4, sendo que a variável de projeto neste caso é o diâmetro. A Figura 51 ilustra o código AnderScript para o cálculo do diâmetro de projeto.

Figura 51 – Código AnderScript para encontrar o diâmetro de projeto

```
1  global {
2      proj diametro = 0.1;
3      norm espessura = (0.01,0.05);
4      norm profundidade = (0.003,0.1);
5      norm carga = (5,0.10);
6      norm rd = (0.0001,0.2);
7      var distancia = 0.025;
8      var comprimento = 0.200;
9      var nDefeitos = 5;
10     var rl = 0.0001;
11     var tu = 538;
12     var ti = 10.0;
13     var tf = 35.0;
14 }
15
16 fun failure() {
17     var deltaT = @tf - @ti;
18     var m = 1, l = 0, k = 1;
19     var l0 = @comprimento + @rl * deltaT;
20     var j, d, pressoes = @profundidade, presaoMinima;
21     var posicao;
22     if(@profundidade < 0.2 * @espessura && @distancia > 2  * (@diametro * @espessura)^( 1 / 2)) {
23         l = l0;
24         m = (1 + 0.31 * ((l) ^ 2)/(@diametro * @espessura))^(1 / 2);
25         d = (@profundidade + @rd * deltaT);
26         pressoes[1] = 2 * (@tu) * (@espessura / (@diametro - @espessura)) * (1 - (d / @espessura)) / (1 - ((d) / (@espessura * m)));
27         #failure = pressoes[1] - @carga ;
28     }
29     else {
30         for(j = 1; j <= @nDefeitos; j = j + 1) {
31             l = (j - 1) * ((@comprimento + @rl * deltaT) + @distancia) + l0;
32             m = (1 + 0.31 * ((l) ^ 2)/(@diametro * @espessura))^(1 / 2);
33             d = (j) * (@profundidade + @rd * deltaT) * (@comprimento + @rl * deltaT) / l;
34             pressoes[k] = 2 * (@tu) * (@espessura / (@diametro - @espessura)) * (1 - (d / @espessura)) / (1 - ((d) / (@espessura * m)));
35             k = k + 1;
36         }
37         presaoMinima = min(pressoes);
38         posicao = find(pressoes == presaoMinima);
39         #failure = pressoes(posicao[1]) - @carga ;
40     }
41 }
```

FONTE: (AUTOR, 2019)

Pode-se observar na Figura 51, nas linhas 2 a 13 ocorre a definição das variáveis aleatórias e paramétricas do problema. Na linha 22 ocorre a verificação da

interação entre os defeitos, conforme mostrado anteriormente nos critérios a) e c) para interação entre defeitos na seção 3.1.1. Em seguida da linha 23 a linha 27, ocorre a definição da função de falha para um único defeito, este trecho é executado caso não haja interação entre os defeitos. Havendo interação entre os defeitos, o código entre as linhas 31 e 34 calcula a pressão de falha para cada combinação de defeitos, sendo que na linha 37 a menor pressão de falha é selecionada e utilizada na linha 39 para o cálculo da função de falha. O código para a combinação entre os defeitos (linhas 31 a 34) foi simplificado, se comparado à geração dos grupos de defeitos interagentes comentados anteriormente na seção 3.1.1. Isso porque, os defeitos neste caso possuem dimensões iguais e são igualmente espaçados alinhados longitudinalmente.

A execução do código da Figura 51 gera como saída o valor do diâmetro de 617,51mm. Conforme observado nos exemplos anteriores, este valor consiste em um diâmetro dentro dos limites aceitáveis, portanto consiste em um bom resultado do método.

6 CONSIDERAÇÕES FINAIS

O uso da confiabilidade estrutural é uma ferramenta de auxílio ao engenheiro que permite a consideração das incertezas associadas aos parâmetros inerentes a todo projeto. Como toda estrutura possui probabilidade não nula de vir ao colapso, o uso da análise de confiabilidade permite quantificar este risco e tomar medidas que o diminua o máximo possível acidentes, visando impedir o colapso da estrutura que por sua vez pode causar danos econômicos, ambientais e humanos.

O uso adequado da análise de confiabilidade por sua vez permite uma grande economia para a companhia envolvida, pois permite minimizar a troca de material, bem como a frequência dos reparos e inspeções. Uma vez que estas operações demandam muitos gastos, relacionados a equipamentos, mão-de-obra e materiais, otimizar este processo evita custos desnecessários.

Dentro da confiabilidade estrutural, o uso de formulações empíricas se comparados às metodologias numéricas como elementos finitos, possui a vantagem de ter um tempo de processamento menor, além é claro de ser mais fácil de se implementar.

Ao fazer a análise de confiabilidade, pode-se notar que o índice de confiabilidade do duto analisado diminui em função do aumento do tempo de exposição à corrosão. Isso ocorre pelo fato de que as consequências da corrosão, diminuição da espessura do duto, aumentam com o passar do tempo, diminuindo desta forma a vida útil do duto. Além disso, foi observado que o índice de confiabilidade também depende do número de defeitos interagentes, de fato, à medida que o número de defeitos aumenta, o índice de confiabilidade diminui. Isto se deve ao fato de que os múltiplos defeitos podem sobrepor suas áreas de influência, causando um efeito ainda maior sobre a estrutura em termos de diminuição da capacidade resistente do duto.

Com relação ao projeto baseado em confiabilidade, pode-se notar que o valor da variável de projeto (espessura) aumenta à medida que o número de defeitos aumenta. O aumento do número de defeitos diminui a resistência do duto, e a forma de compensar isso é aumentar sua espessura, para que o mesmo possa ter um acréscimo em sua resistência, e desta forma atender ao índice de confiabilidade alvo. Pode-se notar também, que o índice de confiabilidade alvo possui influência na espessura final do duto, pois ao se aumentar o índice de confiabilidade alvo, a

espessura do duto também tende a aumentar, em resposta a necessidade de manter um nível de segurança ainda maior.

Na análise de confiabilidade, os 4 métodos analisados, tiveram resultados muito parecidos. Neste caso, o único método que possui desvantagens significativas é o método de Monte Carlo clássico, devido ao seu alto custo computacional. Porém, o cenário muda no projeto baseado em confiabilidade, onde apenas os métodos FORM, e Monte Carlo tiveram resultados satisfatórios, com relação ao aumento da espessura do duto com o aumento no número de defeitos. Portanto, considerando os 4 métodos analisados neste trabalho, conclui-se que na análise de confiabilidade pode ser feita satisfatoriamente com os métodos de FORM, Monte Carlo com Esperança Condicionada e Monte Carlo com Redes Neurais. Enquanto que o projeto baseado em confiabilidade possui o maior desempenho se for feito utilizando o método FORM, pelo mesmo apresentar resultados precisos, em relação ao comportamento real, e pouco custo computacional.

A metodologia proposta neste trabalho, de que cada thread do webservice tenha acesso a uma instância exclusiva do Matlab, possui pontos positivos e negativos. Com relação aos pontos positivos, está a fácil implementação e implantação do sistema. Além de que, para um baixo número de conexões simultâneas o sistema (*webservice* e o aplicativo) atende adequadamente ao seu propósito de realizar a computação dos métodos de confiabilidade em um tempo curto. Em contrapartida, precisar de uma instância do Matlab para cada thread ativa, pode sobrecarregar o servidor para um número grande de clientes. Além disso, é bastante provável que não haja muitos serviços de hospedagem (“aluguel” de estrutura computacional para abrigar aplicações web) disponíveis no mercado que tenham o Matlab instalado ou que permitam sua instalação. Desta forma, a implantação deste sistema em uma estrutura não proprietária pode ser um problema.

Uma possível solução é o uso do MatlabServer que permite criar webservices diretamente utilizando a estrutura do Matlab. Isto permite que as funções do Matlab sejam acessadas simultaneamente, sem que precisem ser criadas seções para cada requisição. Uma outra solução é usar a plataforma Python, a mesma possui módulos para desenvolvimento de *WebServices*, os mais recentes algoritmos em redes neurais e inteligência artificial, além de matemática simbólica igualmente ao Matlab. Além disso, o Python possui muitos serviços de hospedagem, o que facilita ainda mais sua implantação em um servidor remoto.

A linguagem AnderScript apresenta uma forma mais fácil de se escrever problemas de confiabilidade que contenham apenas um único componente estrutural, sem correlação entre as variáveis aleatórias e com apenas uma única variável de projeto. A compilação e execução são feitas para o método FORM, que conforme comentado anteriormente, apresenta bons resultados em termos de precisão numérica e fidelidade ao comportamento real, tanto para a fase de análise em confiabilidade, quanto para projeto baseado em confiabilidade, o que justifica os bons resultados em termos de precisão numérica obtidos anteriormente neste trabalho ao se escrever e resolver problemas nesta linguagem.

Além disso, o uso do aplicativo para captar as informações relativas ao duto e seus defeitos de corrosão, apresentou um método fácil para a entrada dos dados. Uma das principais vantagens, foi a possibilidade de armazenar informações de diversos defeitos, e a possibilidade de poder exportar e compartilhar estar informações na forma de um arquivo no formato “.txt”. Por fim, a integração com o ambiente tridimensional desenvolvido na Engine Unity, apresentou a representação 3d do duto e seus defeitos de corrosão de forma satisfatória, com poucas linhas de programação sem grandes consequências negativas na performance do aplicativo.

7 TRABALHOS FUTUROS

Como sugestões de trabalhos futuros tem-se:

- ✓ Fazer análise e projeto baseado em confiabilidade de dutos sujeitos a múltiplos defeitos de corrosão desalinhados;
- ✓ Fazer análise e projeto baseado em confiabilidade utilizando modelos numéricos para o cálculo da pressão de falha;
- ✓ Utilização de modelo não-linear de corrosão;
- ✓ Melhorar a linguagem AnderScript para suportar a definição de mais de uma variável de projeto, e a definição de outros tipos de distribuição de probabilidade, que não seja apenas distribuição normal, para as variáveis aleatórias;
- ✓ Implementar o sistema proposto no presente trabalho na plataforma Python.

REFERENCIAS

AHAMMED, M. Probabilistic estimation of remaining life of a pipeline in the presence of active defects. **International Journal of Pressure Vessels and Piping**, Vol. 75, 1998. Pages 325 - 329.

AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compiladores Princípios, Técnicas e Ferramentas**. Rio de Janeiro: LTC-Livro Técnicos e Científicos Editora S.A., 1995.

AMAYA-GÓMEZ, R. et al. Reliability assessments of corroded pipelines based on internal pressure – A review. **Engineering Failure Analysis**, v. 98, p. 190-214, 2019.

ASANO, C. H.; COLLI, E. **Cálculo Numérico - Fundamentos e Aplicações**. Instituto Militar de Engenharia - Universidade de São Paulo. São Paulo. 2009.

BARBOSA, A. H. **Análise de Confiabilidade Estrutural Utilizando o Método de Monte Carlo e Redes Neurais**. Dissertação (Mestrado em Engenharia Civil) - Universidade Federal de Ouro Preto. Ouro Preto. 2004.

BS7910. **Guide on Methods for Assessing the Acceptability of Flaws in Metallic Structures - Annex G: The Assessment of Corrosion in Pipes and Pressure Vessels**. British Standard. [S.l.]. 2005.

FILHO, R. D. P. O transporte por dutos ainda é incipiente no Brasil. **Revista Adnormas**, 2018. Disponivel em: <https://revistaadnormas.com.br/2018/10/09/o-transporte-por-dutos-ainda-e-incipiente-no-brasil/>. Acesso em: 27 maio 2019.

GENTIL, V. **Corrosão**. 3ª. ed. Rio de Janeiro: LTC - Livros Técnicos e Científicos Editora S. A., 1996.

GOETZ, B. et al. **Java Concurrency in Pratice**. 2ª. ed. Nova Jersey: Addison-Wesley, 2008.

GOMES, W. J. S.; BECK, A. T. Optimal inspection and design of onshore pipelines under external corrosion process. **Structural Safety**, v. 47, p. 48-58, 2014.

GONCALVES, A. **Introdução à Plataforma Java EE 6 com GlassFish 3**. 2ª. ed. Rio de Janeiro: Editora Ciência Moderna Ltda., v. Paulo André P. Marques, 2011.

GOOGLE. App fundamentals. **Android Developers**, 2019. Disponivel em: <https://developer.android.com/guide?hl=pt-BR>. Acesso em: 20 maio 2019.

HASOFER, A. M.; LIND, N. C. Exact and Invariant Second-Moment Code Format. **Journal of Engineering Mechanics (ASME)**, v. 100, p. 111-121, 1974.

HAYKIN, S. **Redes Neurais Princípios e prática**. 2ª. ed. São Paulo: ARTMED Editora S.A., 2008.

IMPACTA. 9 servidores de aplicação úteis para desenvolvedores. **Blog Impacta**, 2019. Disponivel em: <https://www.impacta.com.br/blog/2017/08/02/7-servidores-de-aplicacao-desenvolvedores/>. Acesso em: 27 maio 2019.

INSTITUTO BRASIL LOGÍSTICO. IBLOG. **MODAL DUTOVIÁRIO (DUTOS, CLASSIFICAÇÃO, VANTAGENS E DESVANTAGENS.)**, 2018. Disponivel em: <https://institutobrasillogistico.com.br/2018/01/28/modal-dutoviario-dutos-classificacao-vantagens-e-desvantagens/>. Acesso em: 27 maio 2019.

JCSS. **PROBABILISTIC MODEL CODE, Part 1 - BASIS OF DESIGN**. Joint Committee on Structural Safety. [S.l.]. 2000.

KIUREGHIAN, A. D.; LIU, P. L. Structural Reliability Under Incomplete Probability Information. **Journal of Engineering Mechanics (ASCE)**, v. 112, 1986.

LEIRA, B. J.; NÆSS, A.; NÆSS, O. E. B. Reliability analysis of corroding pipelines by enhanced Monte Carlo simulation. **International Journal of Pressure Vessels and Piping**, v. 144, p. 11-17, 2016.

MEDNIEKS, Z. et al. **Programando o Android**. 2º. ed. São Pulo: Novatec Editora Ltda, 2013.

MISHRA, M.; KESHAVARZZADEH, V.; NOSHADRAVAN, A. Reliability-based lifecycle management for corroding pipelines. **Structural Safety**, v. 76, p. 1-14, 2019.

NGOLO, M. A. **Arquitetura Orientada a Serviços REST para Laboratórios Remotos**. Dissertação (Mestre em Engenharia Eletrotécnica e de Computadores) - Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa. Lisboa. 2009.

NUNES, F. Android MVC x MVP x MVVM qual Pattern utilizar—Parte 1. **Medium**, 2017. Disponivel em: <https://medium.com/@FilipeFNunes/android-mvc-x-mvp-x-mvvm-qual-pattern-utilizar-parte-1-3defc5c89afd>. Acesso em: 27 maio 2019.

QUILES, M. G. **Sistema de Visão Baseado em Redes Neurais para o**. Dissertação (Mestrado em CIência da Computação e Matemática Computacional) - Universidade de São Paulo. São Paulo. 2004.

RACKWITZ, R.; FIESSLER, B. Structural Reliability Under Combined Randon Load Sequence. **Computer and Structures**, v. 9, p. 489-494, 1978.

RALLO, R. Material Design: aprenda tudo sobre o design do Google! **Rock Content**, 2019. Disponivel em: <https://rockcontent.com/blog/material-design/>. Acesso em: 27 maio 2019.

ROCHA, A. C. F. **ESTUDO DE CONFIABILIDADE DE DUTOS SUJEITOS A CORROSÃO**. Trabalho de Conclusão de Curso (Graduação em Engenharia Civil) - Universidade Federal de Pernambuco. Caruaru. 2016.

RUSSEL, S.; NORVIG, P. **Inteligência Artificial**. 3ª. ed. Rio de Janeiro: Elsevier Editora Ltda., 2013.

SAGRILO, L. V. **Análise de Confiabilidade Estrutural Utilizando os Métodos análiticos FORM E SORM**. Tese (Doutorado em Ciências em Engenharia Civil) - Universidade Federal do Rio de Janeiro. Rio de Janeiro. 1994.

SEGHIER, M. E. A. B. et al. Reliability analysis based on hybrid algorithm of M5 model tree and Monte Carlo simulation for corroded pipelines: Case of study X60 Steel grade pipes. **Engineering Failure Analysis**, v. 97, p. 793-803, 2019.

SUN, J.; CHENG, Y. F. Modelling of mechano-electrochemical interaction of multiple longitudinally aligned corrosion defects on oil/gas pipelines. **Engineering Structures**, v. 190, p. 9-19, 2019.

TANENBAUM, S. **Sistemas Operacionais Modernos**. 3º. ed. São Paulo: Pearson Education do Brasil Ltda., 2012.

TEE, K. F.; PESINIS, K. Reliability prediction for corroding natural gas pipelines. **Tunnelling and Underground Space Technology**, v. 65, p. 91-105, 2017.

TORO, J. N. **Pressão de Ruptura de Dutos contendo Defeitos de Corrosão**. Dissertação (Mestre em Engenharia de Estruturas) - Universidade de São Paulo. São Carlos. 2014.

TORRES, J. S. **Uma Metodologia para Verificação da Segurança e Dimensionamento Ótimo de Dutos com Defeitos Causados por Corrosão**. Tese (Doutorado em Estruturas) - Universidade Federal de Pernambuco. Recife. 2009.

UNITY TECHNOLOGIES. Example - Creating a Quad. **Unity Documentation**, 2019. Disponivel em: <https://docs.unity3d.com/Manual/Example-CreatingaBillboardPlane.html>. Acesso em: 20 Maio 2019.

VERZENHASSI, C. C. **Otimização de risco estrutural baseada em confiabilidade**. Dissertação (Mestrado em Engenharia de Estruturas) - Escola de Engenharia de São Carlos. São Carlos. 2008.

WANG, H. et al. A clustering approach for assessing external corrosion in a buried pipeline based on hidden Markov random field model. **Structural Safety**, v. 56, p. 18-29, 2015.

WANG, H.; YAJIMA, A.; CASTANEDA, H. A stochastic defect growth model for reliability assessment of corroded underground pipelines. **Process Safety and Environmental Protection**, v. 123, p. 179-189, 2019.

WIKIMEDIA FOUNDATION. Polygon mesh. **Wikipedia**, 2018. Disponivel em: <https://en.wikipedia.org/wiki/Polygon_mesh>. Acesso em: 05 jun. 2019.

XU, L. Y.; CHENG, Y. F. Reliability and Failure Pressure Prediction of Various Grades of Pipeline Steel in the Presence of Corrosion Defects and Pre-Strain. **International Journal of Pressure Vessels and Piping**, 2011.

XU, W.-Z. et al. Corroded pipeline failure analysis using artificial neural network scheme. **Advances in Engineering Software**, v. 112, p. 255-266, 2017.

ZELMATI, D.; GHELLOUDJ, O.; AMIRAT, A. Correlation between defect depth and defect length through a reliability index when evaluating of the remaining life of steel pipeline under corrosion and crack defects. **Engineering Failure Analysis**, v. 79, p. 171-185, 2017.

ZHOU, W.; XIANG, W.; HONG, H. P. Sensitivity of system reliability of corroding pipelines to modeling of stochastic growth of corrosion defects. **Reliability Engineering & System Safety**, v. 167, p. 428-438, 2017.