CENTRO UNIVERSITÁRIO DE BARRA MANSA

PRÓ-REITORIA ACADÊMICA

ENGENHARIA ELÉTRICA

Aluno: PEDRO HENRIQUE NASCIMENTO VIEIRA

Orientador: LEONARDO DE CARVALHO VIDAL

LOCALIZAÇÃO DE FALTAS FASE-TERRA EM LINHAS DE TRANSMISSÃO UTILIZANDO REDE NEURAL TREINADA COM DADOS SIMULADOS COM O EMTP-ATP

BARRA MANSA

2016

PEDRO HENRIQUE NASCIMENTO VIEIRA

LOCALIZAÇÃO DE FALTAS FASE-TERRA EM LINHAS DE TRANSMISSÃO UTILIZANDO REDE NEURAL TREINADA COM DADOS SIMULADOS COM O EMTP-ATP

Monografia apresentada ao Curso de Engenharia Elétrica do Centro Universitário de Barra Mansa – UBM Cicuta, como requisito parcial para a obtenção do título de Engenheiro Elétrico, sob a orientação do Prof. Leonardo de Carvalho Vidal.

BARRA MANSA

2016

Agradecimentos

À minha família, que sempre me apoiou e deu suporte desde a infância.

Aos professores do UBM, que me ensinaram e orientaram quando necessário.

Agradeço meus amigos e amigas também pelo apoio e companheirismo em todos os momentos.

Ao Sci-Hub e o livre compartilhamento de informação e conhecimento que ele promove, sem o qual este trabalho seria impossível.

Resumo

VIEIRA, Pedro Henrique Nascimento. Localização de faltas fase-terra em linhas de transmissão utilizando Rede Neural treinada com dados simulados com o EMTP-ATP. 2016. 72 p. Monografia (Graduação em Engenharia Elétrica) - Centro Universitário de Barra Mansa, Barra Mansa, RJ.

O sistema de transmissão de energia elétrica, devido a sua extensão, está sujeito a faltas. Quando uma falta permanete ocorre, a localização dela é crucial para o rápido restaelecimento do sistema. Para localizar o ponto de ocorrência de um curto-circuito fase-terra em uma linha de transmissão hipotética, este trabalho apresenta um algoritimo e código de computador para sistematicamente simular faltas fase-terra em uma linha de transmissão com o EMTP-ATP e treinar uma Rede Neural com tais dados simulados. Para que se tenha um volume grande de dados para treinar a Rede Neural, códigos de computador na linguagem de programação Python foram desenvolvidos de forma a automatizar a criação de arquivos de entrada do EMTP-ATP e a aquisição dos resultados deles. A Rede Neural treinada apresentou um bom desempenho quando testada com os mesmos dados usados para treino dela; porém testes com dados de uma linha de transmissão real durante faltas devem ser feitos.

Palavras-Chave: EMTP-ATP, faltas fase-terra, linhas de transmissão, Python, Rede Neural, simulação, transitórios.

Abstract

VIEIRA, Pedro Henrique Nascimento. Localization of phase-ground faults in transmission lines using Neural Network trained with simulated data from the EMTP-ATP. 2016. 72 p. Monograph (Electrical Engineering) - Centro Universitário de Barra Mansa, Barra Mansa, RJ.

The electrical energy transmission system is subject to faults due to its extension. When a fault occurs, its localization crucial for a fast restoration of the system. To locate the point where a phase-ground short-circuit occurred in an hypothetical transmission line, this work presents an algorithm and computer code to systematically simulate phase-ground faults in a transmission line with the EMTP-ATP and train a Neural Network with those simulated data. In order to a great volume of data be avaible for trainning the Neural Network, computer code written in the Python programming language were developed to automate the creation of input archives of the EMTP-ATP and their results' acquisition. The trained Neural Network has shown good performance when tested with the same data used in its trainning; though tests with data from a real transmission line during faults should be made.

Keywords: EMTP-ATP, Neural Network, phase-ground faults, Python, simulation, transients, transmission lines.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Motivação

Em nossa sociedade há uma dependência da energia elétrica e sua confiabilidade. Para verificar isto basta observar quantos equipamentos elétricos existem atualmente nas residências e indústrias.

A interrupção do fornecimento de energia elétrica, além de ser inconveniente, pode ter consequências graves, e.g. um hospital que possui equipamentos elétricos necessários à manutenção da vida. Portanto há um contínuo esforço para melhorar a confiabilidade dos sistemas de alimentação elétrica, para prevenir falhas e, caso ocorram, saná-las rapidamente.

Como definido por Dallbello et al. (2007), faltas elétricas em sistemas de potência são curtos-circuitos que podem ocorrer devido a descargas atmosféricas, rompimento de cabos ou torres, entre outros.

As faltas provenientes de rompimento de cabos e torres, por exemplo, são permanentes, pois requerem a intervenção de uma equipe de manutenção para corrigi-las. Quando elas ocorrem, geralmente, a equipe de manutenção percorre a extensão da linha até encontrar o defeito, que pode passar por locais de difícil acesso. Isto faz que o reestabelecimento do sistema demore, causando transtornos aos consumidores.

1.2 Proposta do Trabalho

Para amenizar esse problema dos sistemas de potência, propõe-se neste trabalho criar um algoritmo para localizar as faltas permanentes a partir dos dados de tensão e corrente na subestação receptora.

Todos os códigos de computador utilizados neste trabalho serão feitos na linguagem de programação Python.

1.2.1 Objetivo Geral

Far-se-á a simulação computacional de uma linha de transmissão conectando uma subestação emissora e outra receptora.

O efeito pelicular será levado em consideração na simulação devido aos componentes de alta frequência que existem durante os transitórios causados pela falta.

1.2.2 Objetivo Específico

Através do processamento, por decomposição em séries de Fourier, dos sinais de tensão e corrente na subestação receptora, far-se-á uma inferência para determinar o ponto e as fases envolvidas na falta.

1.3 Delimitações

1.3.1 Sistema de Potência Simulado

O sistema será condensado, tanto no lado emissor quanto receptor, em seu equivalente Thévenin por fase. É um sistema trifásica com tensão de operação de 230 kV, 60 Hz, em ambos os lados (emissor e receptor). O lado receptor e emissor tem, relação X/R (aproximadamente) de 9 e 226, respectivamente. A potência de curto ciruito é 1170 MVA para o lado receptor, e 4680 MVA para o lado emissor.

1.3.2 Linha de Transmissão Considerada

A linha de transmissão considerada é trifásica, um condutor por fase, com eixo de simetria vertical. Considerar-se-á que os parâmetros, para uma dada frequência, não variam durante toda a extensão da linha. Será usado um modelo de linha que inclua o efeito pelicular. A resistividade do solo é considerada 100 $\Omega \times m$.

2 Fundamentação Teórica

2.1 Linhas de Transmissão

2.1.1 Propagação de Ondas Eletromagnéticas

Pelo desenvolvimento das equações de Maxwell do eletromagnetismo, **Fontana2013** conclui que existem ondas de tensão e corrente se propagando nas linhas de transmissão.

Fontana2013 ainda considera a linha como circuitos distribuídos ao longo de sua distância. A partir disto, emprega-se a teoria de circuitos elétricos para análise da tensão e corrente na linha considerando-a seções de circuito π de comprimento $\triangle z$, conforme Figura 2.1.

As equações das ondas de tensão e corrente na linha, também chamadas de ondas viajantes, são:

$$V(z,t) = V(z+\triangle z,t) + R\,\triangle z\,I(z,t) + L\,\triangle z\,\frac{\partial I(z,t)}{\partial t} \tag{2.1}$$

$$I(z,t) = I(z+\triangle z,t) + G\,\triangle z\,V(z+\triangle z,t) + C\,\triangle z\,\frac{\partial V(z+\triangle z,t)}{\partial t} \tag{2.2}$$

Dividindo ambas as equações acima por $\triangle z$, fazendo-o tender a zero, e rearranjando os elementos, obtém-se:

$$\frac{\partial V(z,t)}{\partial z} = RI(z,t) + L\frac{\partial I(z,t)}{\partial t} \tag{2.3}$$

$$\frac{\partial I(z,t)}{\partial z} = GV(z,t) + C\frac{\partial V(z,t)}{\partial t} \tag{2.4}$$

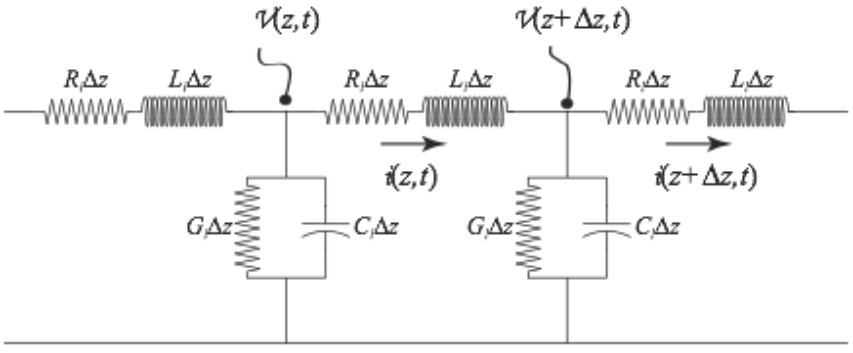


Figura 2.1: Representação de uma linha de transmissão por circuitos em cascata (**Fontana2013**).

2.1.2 Solução geral das equações da linha monofásica

Seguindo o mesmo princípio que **Fontana2013**, Andrade, Leite e Leão (2013) deduzem a solução das equações diferenciais parciais para o caso geral de uma linha de transmissão monofásica utilizando condições de contorno (tensão e corrente em um terminal). Permitindo, assim, o estudo dos transitórios eletromagnéticos na linha sendo a única simplificação assumir os parâmetros da linha como lineares.

A solução para a tensão e a corrente em um ponto x da linha, no tempo t, tem forma:

$$\begin{bmatrix} V(x,t) \\ I(x,t) \end{bmatrix} = \begin{bmatrix} A_1 & B_1 & A_2 & B_2 \\ C_1 & D_1 & C_2 & D_2 \end{bmatrix} \begin{bmatrix} V(0,t_1) \\ I(0,t_1) \\ V(0,t_2) \\ I(0,t_2) \end{bmatrix} \tag{2.5}$$

$V(0,t_1)$ e $I(0,t_1)$ denotam, respectivamente, tensão e corrente em um ponto inicial da linha no tempo t_1. A escolha de t_1 e t_2 influencia na precisão do resultado, que também é afetado pela posição onde os resultados são desejados, e pela velocidade da onda. Recomenda-se fazer:

$$t - t_1 = t_2 - t = \frac{x}{v_t} \tag{2.6}$$

A velocidade da onda é determinada pela indutância e capacitância da linha:

$$v_t = \frac{1}{\sqrt{LC}} \tag{2.7}$$

As constantes que aparecem na equação 2.5 são:

$$A_1 = \frac{-e^{-q} - 1}{2\sinh(q)} \cosh(x\gamma_{RG}) \tag{2.8}$$

$$B_1 = \frac{e^{-q} + \frac{LG}{RC}}{2\sinh(q)} Z_{RG} \sinh(x\gamma_{RG}) \tag{2.9}$$

$$C_1 = \frac{e^{-q} + \frac{LG}{RC}}{2\sinh(q)} \frac{\sinh(x\gamma_{RG})}{Z_{RG}} \tag{2.10}$$

$$D_1 = A_1 \tag{2.11}$$

$$A_2 = \frac{e^{q} - 1}{2\sinh(q)} \cosh(x\gamma_{RG}) \tag{2.12}$$

$$B_2 = \frac{-e^q + \frac{LG}{RC}}{2\sinh(q)} Z_{RG} \sinh(x\gamma_{RG}) \tag{2.13}$$

$$C_2 = \frac{-e^q + \frac{LG}{RC}}{2\sinh(q)} \frac{\sinh(x\gamma_{RG})}{Z_{RG}} \tag{2.14}$$

$$D_2 = A_2 \tag{2.15}$$

Sendo:

$$q = \left(\frac{R}{Z_{LC}} + GZ_{LC}\right) x \tag{2.16}$$

$$Z_{RG} = \sqrt{\frac{R}{G}} \tag{2.17}$$

$$Z_{LC} = \sqrt{\frac{L}{C}} \tag{2.18}$$

Andrade, Leite e Leão (2013) chamam γ_{RG} de coeficiente de propagação. As constantes Z_{RG} e Z_{LC} de impedâncias características. Z_{RG} referente à dissipação de energia na linha; e Z_{LC} referente ao armazenamento de energia nos campos elétrico e magnético no entorno da linha.

As constantes R, L, C, G são os parâmetros distribuídos (por unidade de comprimento) da linha; em ordem: resistência, indutância, capacitância, e condutância.

2.1.3 Transposição

A transposição de uma linha de transmissão é uma técnica no qual a posição física dos condutores é trocada em intervalos. Ela tem por objetivo diminuir a indutância da linha e, assim, suas perdas (Tavares, 1998).

2.1.4 Aproximação por circuitos π em cascata

Uma simplificação que é comumente encontrada é a representação da linha por circuitos π, como o da Figura 2.1, de comprimento z. Nela, os parâmetros distribuídos da linha são concentrados em cada segmento π equivalente.

Dommel (1986) evidencia na Figura 2.2 que, para simulação de transitórios, quanto mais circuitos em cascata, mais acurada, mas não mais precisa, a resposta quando comparada com a solução do circuito por ondas viajantes.

A Figura 2.2 mostra a tensão no terminal de recepção de uma linha monofásica se uma tensão de $10\ Vcc$ é conectada ao terminal de envio em $t = 0\ s$. Dados da

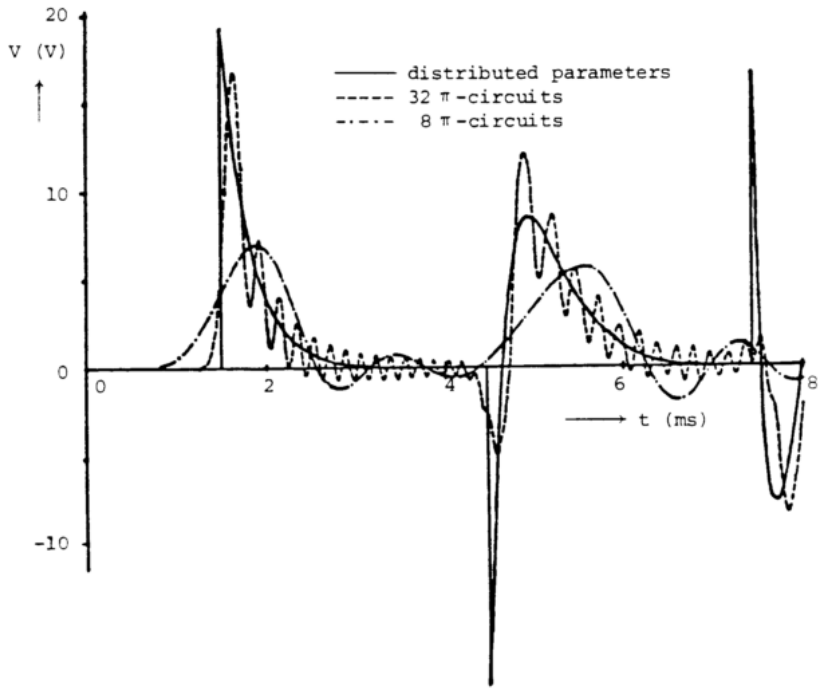


Figura 2.2: Tensão no terminal de recepção de uma linha monofásica (Dommel, 1986, p. 59).

linha: $R = 0.0376\ \Omega/milha$, $L = 1.52\ mH/milha$, $C = 14.3\ nF/milha$, $comprimento = 320\ milhas$, terminal de recepção com indutância shunt de $100\ mH$.

Pela Figura 2.2 e as afirmações de Paz (2005): fazendo o comprimento $\triangle z$ de cada circuito π tender a dz, a representação da linha por circuitos em cascata se aproxima de uma linha real.

Há de se notar, porém, que a aproximação por circuitos π em cascata, mesmo usando um número grande destes, apresenta uma resposta mais oscilatória que o modelo a parâmetros distribuídos. Por isto métodos se supressão dessas oscilações devem ser usados (**Marti1989b**).

2.1.5 Modelo de Bergeron

O modelo a parâmetros distribuídos de Bergeron, baseado no método das características para solução de equações diferenciais parciais hiperbólicas, transforma um circuito com elementos diferenciais (capacitores e indutores) em um circuito equivalente de Norton.

Testes conduzidos por Andrade, Leite e Leão (2013) utilizando linhas reais demonstram que o erro do modelo de Bergeron torna-se mais significativo quanto maior o comprimento da linha.

Dommel (1969) usa este método, originalmente aplicado a problemas de hidráulica, combinado com o método de integração trapezoidal para simular linhas de transmissão sem perdas representada por parâmetros distribuídos. As perdas da linha

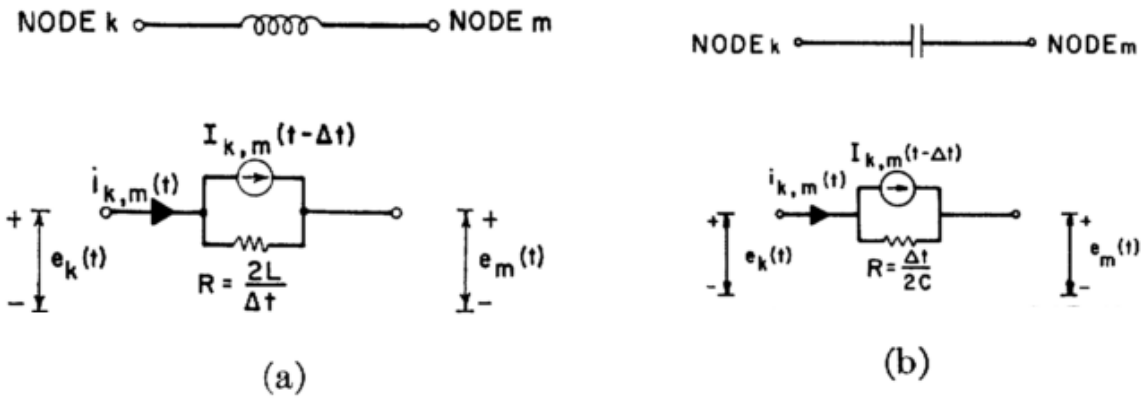


Figura 2.3: Circuito Norton equivalente por integração trapezoidal: (a) indutor, (B) capacitor. Adaptado de (Dommel, 1986).

são representadas por resistências concentradas nos seus terminais.

O algoritmo para solucionar um sistema no modelo de Bergeron é simples, rápido e numericamente estável (Dommel, 1969).

2.1.6 Dedução do modelo de Bergeron

2.1.6.1 Circuitos equivalentes de Norton para elementos armazenadores de energia

Fazendo a integração trapezoidal para encontrar uma expressão analítica para a corrente saindo de (ou entrando em) um terminal, Dommel (1969) demonstra que indutores e capacitores podem ser expressos por um circuito equivalente de Norton. Este é o mesmo processo demonstrado por Sauer e Pai (2006). A Figura 2.3 sintetiza isto utilizando um passo de integração de $\triangle t$.

O valor da fonte de corrente é determinado pela equação:

$$I_{k,m}(t - \triangle t) = \frac{V_{k,m}(t - \triangle t)}{R} + i_{k,m}(t - \triangle t) \tag{2.19}$$

O erro de truncamento desta aproximação, que é útil para resolver um sistema elétrico por análise nodal iterativamente, obtida a partir da técnica de integração trapezoidal, é de $(\triangle t)^3$ Dommel (1969).

2.1.6.2 Linha de transmissão sem perdas

Dommel (1969) deduz as equações de uma linha de transmissão sem perdas, i.e., condutância e resistência nulas:

$$\frac{-\partial V}{\partial x} = L\frac{-\partial I}{\partial t} \tag{2.20}$$

$$\frac{-\partial I}{\partial x} = C\frac{-\partial V}{\partial t} \tag{2.21}$$

Cuja solução, geralmente atribuída a d'Alembert, tem forma:

$$I(x,t) = f_1(x - vt) + f_2(x + vt) \tag{2.22}$$

$$V(x,t) = Z_0 f_1(x - vt) - Z_0 f_2(x + vt) \tag{2.23}$$

Sendo f_1 e f_2 funções arbitrárias, interpretadas fisicamente como onda progressiva e onda retrógrada, respectivamente Dommel (1969). O termo Z_0 é a impedância de surto, também chamada de impedância característica Paz (2005). A velocidade de propagação da onda é denotada por v. Elas são dadas por:

$$Z_0 = \sqrt{\frac{L}{C}} \tag{2.24}$$

$$v = \sqrt{\frac{1}{LC}} \tag{2.25}$$

Combinando as equações acima é possível obter:

$$V(x,t) + Z_0 I(x,t) = 2Z_0 f_1(x - vt) \tag{2.26}$$

$$V(x,t) - Z_0 I(x,t) = -2Z_0 f_2(x + vt) \tag{2.27}$$

Segundo Dommel (1969), a expressão $(V(x,t) + Z_0 I(x - vt))$ é constante quando $(x - vt)$ é constante. O mesmo é válido para $(V(x,t) - Z_0 I(x,t))$ e $(x + vt)$. Fazendo-se uma analogia: um observador viajando junto com uma das ondas com velocidade v enxerga sempre o mesmo valor, pois numa linha sem perdas não há atenuação da onda. Essas equações podem ser representadas como um circuito equivalente, conforme Figura 2.4.

Seja τ o tempo que a onda com velocidade v leva para chegar ao terminal m da linha, após sair do terminal k. Então o valor de $(V + Z_0 I)$ no terminal k, no tempo

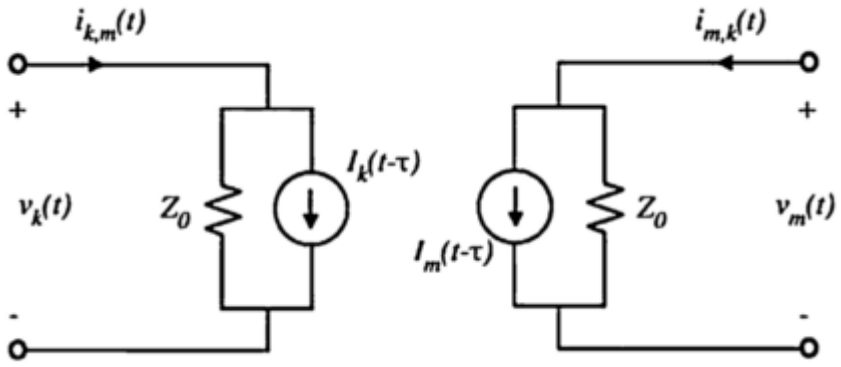


Figura 2.4: Circuito equivalente da linha no modelo de Bergeron (Caballero, Costa e Kurokawa, 2014, p. 15).

$(t - \tau)$, será o mesmo valor no terminal m, no tempo t.

Os terminais m e k não possuem uma conexão topológica. O efeito de um no outro é capturado indiretamente pelas fontes de corrente I_k e I_m.

$$I_k(t-\tau) = \frac{-1}{Z_0} V_m(t-\tau) - i_{m,k}(t-\tau) \tag{2.28}$$

$$I_m(t-\tau) = \frac{-1}{Z_0} V_k(t-\tau) - i_{k,m}(t-\tau) \tag{2.29}$$

2.1.6.3 Representação das perdas

Negligenciando a condutância shunt da linha $(G = 0)$, pode-se capturar o efeito das perdas colocando-se resistências concentradas nos terminais da linha. Cada uma delas igual à metade da resistência série total da linha (Caballero, Costa e Kurokawa, 2014).

Outra forma de se capturar o efeito das perdas negligenciando a condutância shunt é colocando resistências concentradas nos terminais da linha igual a um quarto da resistência total; e metade da resistência total no meio da linha, como feito por Dommel (1969). Nesta perspectiva as variáveis da Figura 2.4 tornam-se:

$$Z_0 = \sqrt{\frac{L}{C}} + \frac{R}{4} \tag{2.30}$$

$$h = \frac{Z_0 - \frac{R}{4}}{Z_0 + \frac{R}{4}} \tag{2.31}$$

$$I_k(t-\tau) = \left(\frac{-1}{Z_0}V_m(t-\tau) - i_{m,k}(t-\tau)\right)\left(\frac{1+h}{2}\right) + \left(\frac{-1}{Z_0}V_k(t-\tau) - i_{k,m}(t-\tau)\right)\left(\frac{1-h}{2}\right) \tag{2.32}$$

$$I_m(t-\tau) = \left(\frac{-1}{Z_0}V_k(t-\tau) - i_{k,m}(t-\tau)\right)\left(\frac{1+h}{2}\right) + \left(\frac{-1}{Z_0}V_m(t-\tau) - i_{m,k}(t-\tau)\right)\left(\frac{1-h}{2}\right) \tag{2.33}$$

O modelo de Bergeron assim deduzido, porém, não considera o efeito da frequência (efeito pelicular) sobre os parâmetros longitudinais da linha (indutância e resistência).

2.2 Efeito Pelicular

Devido à capacitância em derivação (shunt) e da indutância longitudinal, quanto maior a frequência da corrente na linha, i.e., quanto mais rápido ela varia, menos a corrente penetra no condutor. Em frequências elevadas somente as camadas mais externas do condutor estão, de fato, conduzindo corrente; esta sendo negligível na seção mais interna do condutor. A este efeito dá-se o nome de Efeito Pelicular. (Adami, 2008)

> Define-se profundidade de penetração δ ou profundidade pelicular, como a distância percorrida pela onda no interior do bom condutor, de modo que sua amplitude decresça a $e^{-1} = 37\%$ de seu valor junto à superfície. (Adami, 2008, p. 173).

$$\delta = \frac{1}{\sqrt{f\mu\sigma\pi}} \tag{2.34}$$

Onde:

- f é a frequência da onda eletromagnética;
- μ é a permeabilidade magnética do condutor;
- σ é a permeabilidade magnética do condutor.

Assim sendo, quanto maior a frequência da onda, menor é a profundidade de penetração da corrente no condutor, i.e., menor a área de condução e, com isto, maior

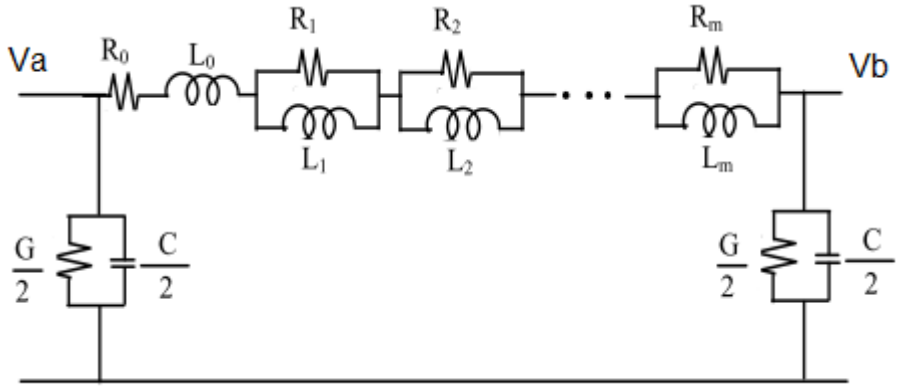


Figura 2.5: Inserção do efeito da freqüência nos circuitos que representam a linha (Kurokawa et al., 2007, p. 340).

a resistência elétrica. Isto fará, também, que o fluxo magnético concatenado na parte interna do condutor seja menor, reduzindo a indutância.

Os elementos transversais (condutância e capacitância), como afirma Tavares (1998), não são significativamente dependentes da frequência, sendo-o apenas da posição relativa dos condutores.

2.3 Modelo de linha considerando o efeito pelicular

Devido ao efeito pelicular, deve-se modelar a linha de tal forma que a influência da frequência durante os transitórios seja levada em consideração.

2.3.1 Blocos RL

Kurokawa et al. (2007) e Tavares (1998) consideram o efeito da frequência nos parâmetros longitudinais, i.e., resistência e indutância, das linhas de transmissão pela inserção de elementos RL paralelos, tantos quanto necessário, em cada segmento do circuito, conforme ilustra a Figura 2.5.

A inconveniência deste método, porém, é o grande número de equações diferenciais resultantes que devem ser resolvidas para fazer a simulação. Uma linha aproximada por 200 circuitos π, com 5 blocos RL, resulta em um sistema com mais de 1000 equações diferenciais (Caballero, Costa e Kurokawa, 2014).

Para contornar este problema, Caballero, Costa e Kurokawa (2014) propuseram um ajuste similar ao modelo de linha de Bergeron. Nele, as resistências concentradas nos terminais da linha são substituídas pelos blocos RL. Quanto a esta adaptação, não foi encontrada nenhuma comparação da acurácia da simulação com a resposta de uma linha real durante transitórios.

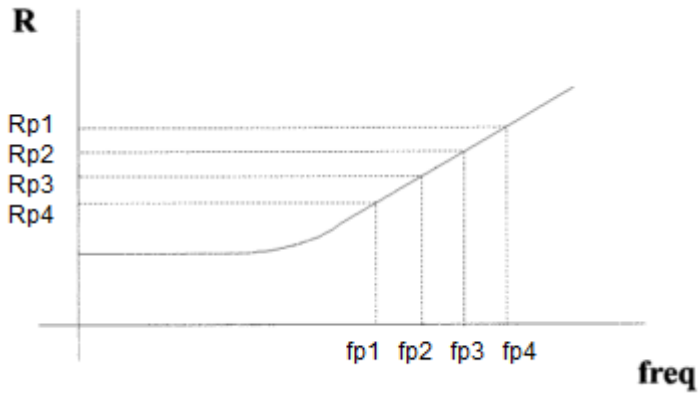


Figura 2.6: Resistência de um modo em função da frequência. Adaptado de (Tavares, 1998, p. 135).

2.3.1.1 Valores de R_m e L_m.

A quantidade de elementos RL paralelos é arbitrada, como sugerido por Tavares (1998) de forma a representar a forma que os parâmetros da linha variam em cada década de frequência.

Tavares (1998) propõe plotar a resistência em função da frequência, Figura 2.6, e dividi-la em m décadas iguais.

Faz-se:

$$R_1 = R_{p2} - R_{p1} \tag{2.35}$$

"Ou seja, a resistência em paralelo corresponde ao acréscimo de resistência em cada intervalo de frequência."(Tavares, 1998, p. 135).

$$R_1 = \sqrt{f_{p1} f_{p2}} \tag{2.36}$$

$$L_1 = \frac{R_1}{\omega_1} \tag{2.37}$$

> Isto significa que a indutância paralela corresponde àquela associada a ressonância do circuito para uma frequência que seria a média geométrica das frequências que definem o intervalo f_{p1} a f_{p2}. Os demais circuitos paralelos são calculados de forma análoga. (Tavares, 1998, p. 135).

Para os parâmetros série:

$$R_0 = R_{DC} \tag{2.38}$$

$$L_0 = L_{f \to 0} \tag{2.39}$$

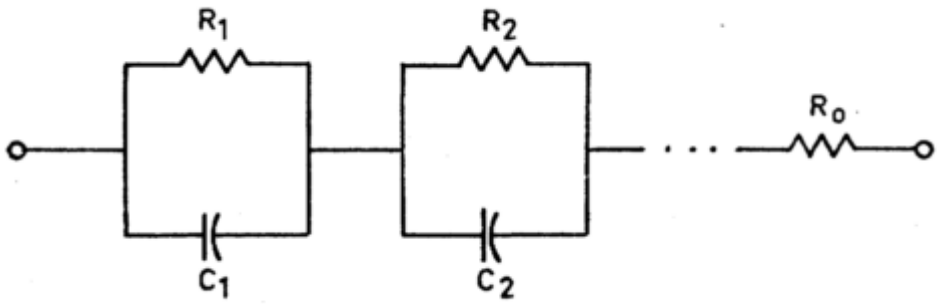


Figura 2.7: Circuito que sintetiza da impedância característica dependente da frequência no modelo de Bergeron (Marti, 1982, p. 150).

2.3.2 Síntese do efeito da frequência no modelo de Bergeron

Similar a Caballero, Costa e Kurokawa (2014), Marti (1981) propõe a substituição da impedância característica nos terminais da linha (no modelo de Bergeron) por blocos RC paralelo, figura 2.7. Ela foi incorporada ao programa EMTP inicialmente feito por Dommel (1969), que determina automaticamente os valores de cada R e C, e também o número destes.

$$Z_0 = \frac{N(s)}{D(s)} = H\frac{(s+z_1)(s+z_2)\dots(s+z_n)}{(s+p_1)(s+p_2)\dots(s+p_n)} \tag{2.40}$$

$$Z_0 = k_0 + \frac{k_1}{s+p_1} + \frac{k_2}{s+p_2} + \dots + \frac{k_n}{s+p_n} \tag{2.41}$$

$$R_0 = k_0 \tag{2.42}$$

$$R_i = \frac{k_i}{p_i} \tag{2.43}$$

$$C_i = \frac{1}{k_i} \tag{2.44}$$

A impedância característica Z_0 é calculada em função da frequência ω e aproximada por funções racionais no plano complexo (Dommel, 1986; Marti, 1981).

Marti (1982) conclui reconhecendo que o modelo apresenta bons resultados para linhas aéreas transpostas e balanceadas.

2.4 Simulação Computacional

2.4.1 Decomposição modal linhas de transmissão polifásicas

Considerando os valores de tensão e corrente como fasores em uma determinada frequência (regime harmônico, domínio da frequência), como feito por Paz (2005), as equações da linha polifásica podem ser escritas como:

$$\frac{-dV_f}{dx} = Z_f I_f \tag{2.45}$$

$$\frac{-dI_f}{dx} = Y_f V_f \tag{2.46}$$

Sendo:

- V_f o vetor das tensões de fase;
- I_f o vetor das correntes de fase;
- Z_f a matriz das impedâncias de fase;
- Y_f a matriz das admitâncias de fase.

As matrizes Z_f e Y_f são simétricas e cheias devido ao acoplamento dos parâmetros entre os condutores.

Tanto Paz (2005) quanto Tavares (1998) contornam este problema através de uma transformação apropriada (uma mudança de base), chamada transformação modal. Ela permite considerar o sistema original de f fases acopladas como um sistema de m modos desacoplados; que são resolvidos como se fossem linhas monofásicas. Depois se faz a transformação inversa para obter as respostas no domínio de fase. As equações que descrevem esse processo são as seguintes:

$$I_f = T_i I_m \tag{2.47}$$

$$I_m = T_i^{-1} I_f \tag{2.48}$$

$$V_f = T_v V_m \tag{2.49}$$

$$V_m = T_v^{-1} V_f \tag{2.50}$$

$$\frac{-dV_m}{dx} = T_v^{-1} Z_f T_i I_m \tag{2.51}$$

$$\frac{-dI_m}{dx} = T_i^{-1} Y_f T_v V_f \tag{2.52}$$

Definem-se as matrizes T_v e T_i de tal forma que as matrizes:

$$Z_m = T_v^{-1} Z_f T_i \tag{2.53}$$

$$Y_m = T_i^{-1} Y_f T_v \tag{2.54}$$

Sejam matrizes diagonais, facilitando a resolução das equações do sistema.

Este recurso, de transformar fases em modos, pode ser utilizado para simulação de transitórios, como o faz Dommel (1969). Portando não é restrito somente a simulações em regime permanente.

2.4.2 ATP-EMTP

O EMTP é um programa de computador feito para simulação de transitórios eletromagnéticos por Dommel (1969). A versão para microcomputadores desenvolvida a partir dele é chamada de EMTP-ATP. Atualmente ele pode ser usado através do ATPDraw, um pré-processador gráfico que facilita a criação dos arquivos para simulação, os *cards* (Prikler e Høidalen, 2009).

2.4.2.1 Simulação por análise nodal

O algoritmo usado por Dommel (1969) no EMTP considera o sistema linear, após substituir todos os elementos da rede sendo simulada por seus equivalentes Norton (fonte de corrente em paralelo com resistor), chegando à equação matricial linear da rede utilizando análise nodal. É o mesmo processo detalhado por Branin Jr. (1967):

$$\begin{bmatrix} Y_{AA} & Y_{AB} \\ Y_{BA} & Y_{BB} \end{bmatrix} \begin{bmatrix} V_A(t) \\ V_B(t) \end{bmatrix} = \begin{bmatrix} i_A(t) \\ i_B(t) \end{bmatrix} - \begin{bmatrix} I_A \\ I_B \end{bmatrix} \tag{2.55}$$

Onde $V_A(t)$ é o vetor de tensões desconhecidas nos nós do sistema. As matrizes de admitância são constantes para um passo de integração $\triangle t$ também constante. O vetor I_A representa as fontes de correntes conectadas aos nós. O vetor $i_A(t)$ representa as correntes injetadas nos nós.

Dommel (1969) apresenta formas de lidar com não linearidades sem alterar significativamente o modelo, não abordadas nesta monografia.

2.5 Liguangem de Programação Python

Python é uma linguagem de programação orientada a objetos que não é muito popular entre engenheiros, mas que, para a comunidade de programadores em geral, é mais popular que Fortran Kiusalaas (2005).

Os programas em Python não são compilados em linguagem de máquina, mas são rodados em interpretadores que traduzem o programa para a máquina linha a linha durante a execução, assim como o MATLAB. Ele possui as seguintes vantagens (Kiusalaas, 2005):

- É open-source, que significa que é gratuito e que qualquer um pode editar seu código-fonte;
- Está disponível para os principais sistemas operacionais (Linux, Unix, Windows, MacOS, etc.);
- É mais fácil de aprender e produz código mais legível que outras linguagens;
- Python e suas extensões são fáceis de instalar.

Uma característica importante dos códigos em Python, apontada por Kiusalaas (2005), é que os loops, sub-rotinas, condicionais, etc. não possuem uma declaração de fim (end). Ao invés disso o corpo do bloco desses códigos é definido pela endentação; e por isso ela é parte integral da sintaxe em Python.

Outro fato importante, também apontado por Kiusalaas (2005), é que Python, assim como MATLAB, diferencia maiúsculas de minúsculas.

Além disto, Python possui um pacote chamado F2PY (faz parte do SciPy: Jones, Oliphant e Peterson (2001–)), que possui uma interface que permite que códigos escritos em FORTRAN ou C sejam invocados pelo interpretador Python, e vice-versa (ver também Cython: Dalcin et al. (2010)). Este interfaceamento é útil para reaproveitar códigos escritos em FORTRAN, C e C++.

2.5.1 Módulos usados

Para implementar redes neurais usou-se o Scikit-Learn: Pedregosa et al. (2011), versão 0.18, que contém a implementação de rede neural *Multilayer Perceptron* para classificação e regressão.

Tambem usou-se o numPy: Walt, Colbert e Varoquaux (2011) e SciPy: Jones, Oliphant e Peterson (2001–) pelas rotinas de cálculo numérico que apresentam.

2.6 Aprendizado Estatístico

Aprendizado estatístico é a disciplina cujo objetivo é aprender a partir de dados. Com um conjunto de dados, chamado dados de treinamento, procura-se, a partir deles, criar um modelo que permita fazer previsões sobre dados ainda não observados (Hastie, Tibshirani e Friedman, 2009). A dedução do modelo, o aprendizado, pode ser supervisionado ou não.

Quando a dedução do modelo é feita por um algoritmo, geralmente um programa de computador, dá-se o nome de aprendizado de máquina.

Aprendizado supervisionado, de acordo com Hastie, Tibshirani e Friedman (2009), é aquele no qual há presença de resultados (saída) para um conjunto de características (entrada), para guiar o processo de aprendizado.

Aprendizado não supervisionado é aquele no qual não há resultados, e sim apenas um conjunto de características. A tarefa deste aprendizado é determinar como os dados estão aglomerados ou organizados, i.e., determinar padrões nos dados. (Hastie, Tibshirani e Friedman, 2009).

2.6.1 Métodos de aprendizado supervisionado

Esta forma de aprendizado busca prever o(s) valor(es) da saída para um dado de entrada.

2.6.1.1 Regressão Linear e Quadrados Mínimos.

Seja $X^T = (X_1, X_2, \ldots, X_p)$, a saída Y é prevista por regressão linear pelo modelo:

$$\hat{Y} = X^T \hat{\beta} \tag{2.56}$$

Onde $\hat{\beta}$ denota um vetor de coeficientes.

Este método de previsão, ilustrado na figura 2.8 é, segundo Hastie, Tibshirani e Friedman (2009), um dos mais populares. Busca-se determinar os coeficientes (β_i) de forma que o somatório dos quadrados residuais seja mínimo. Este ajuste é chamado de Quadrados Mínimos e seu valor é dado pela equação:

$$SRQ(\beta) = (y - X\beta)^T (y - X\beta) \tag{2.57}$$

Fazendo a regressão local, em subconjuntos dos dados, é possível ajustar melhor o modelo. A figura 2.8 exemplifica uma regressão linear: valores acima da reta são classificados como laranja, e abaixo como azul.

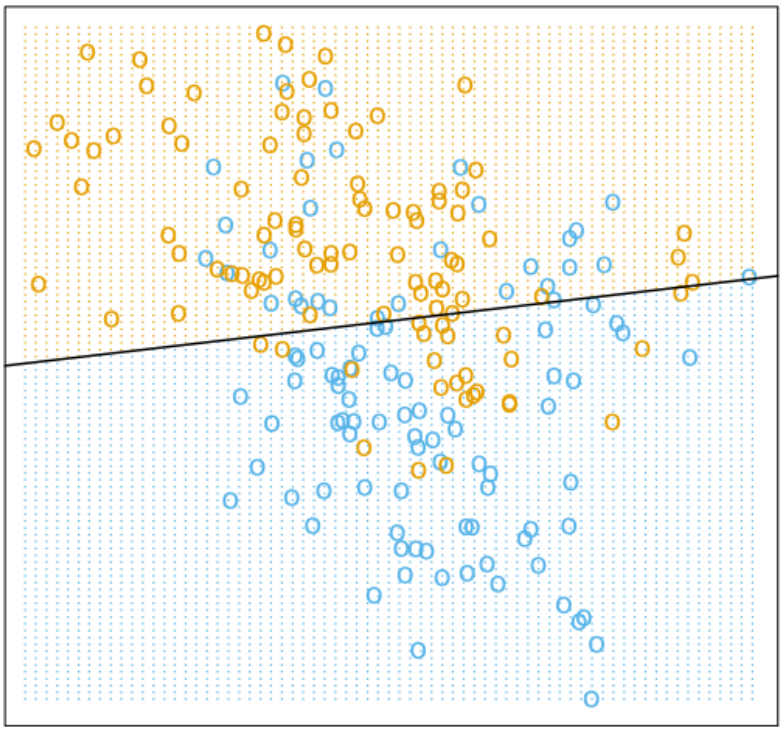


Figura 2.8: Regressão Linear de um conjunto de dados com dois valores de saída: azul ou laranja (Hastie, Tibshirani e Friedman, 2009, p. 13).

2.6.1.2 *Nearest-Neighbor* (Vizinhos mais próximos)

O método dos k Vizinhos Mais Próximos, chamado de agora em diante como $k - Nearest$ usa a média dos k pontos mais próximos do valor x sendo previsto para determinar a saída (Hastie, Tibshirani e Friedman, 2009).

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in V_k(x)} y_i \tag{2.58}$$

Onde $V_k(x)$ denota a vizinhança de x, que implica a existência de uma métrica entre os pontos. O resultado deste método aplicado aos dados da figura 2.8 está ilustrado na figura 2.9.

Hastie, Tibshirani e Friedman (2009) alertam que este método tende a ser instável. Para aprimorá-lo pode-se:

- Usar pesos menores para pontos mais longínquos, ao invés de 0 ou 1 (dentro ou fora da vizinhança), conhecido como método *kernel*;

- Ajustar os pesos, em conjuntos com muitas variáveis, para enfatizar alguns mais que os outros.

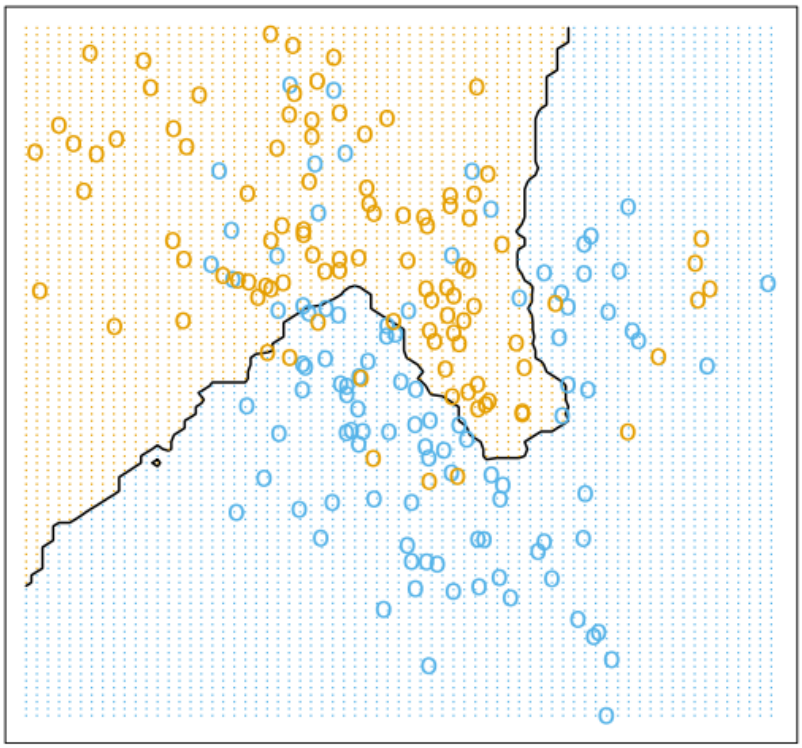


Figura 2.9: Classificação $15 - Nearest$ do mesmo conjunto de dados que a figura 2.8 (Hastie, Tibshirani e Friedman, 2009, p. 15).

2.6.1.3 Redes neurais

Os algoritmos que são classificados como Redes Neurais buscam uma função não linear a partir de combinações lineares das entradas (Hastie, Tibshirani e Friedman, 2009). Embora Haykin (1994) demonstre que o tópico sobre redes neurais seja diverso e multidisciplinar, a discussão que se segue, feita por Hastie, Tibshirani e Friedman (2009), é sobre um algoritimo que, segundo os mesmos, é mais comum de ser encontrado e usado. O algoritmo apresentado se assemelha ao *Multilayer Perceptrons* apresentado por Haykin (1994).

Rede Neural pode ser vista como uma regressão (ou classificação) feita em dois estágios. A figura 2.10 e as equações seguintes sumarizam isto:

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad (m = 1, \ldots, M) \tag{2.59}$$

$$T_k = (\beta_{0k} + \beta_k^T Z), \quad (k = 1, \ldots, K) \tag{2.60}$$

$$f_k(X) = g_k(T), \quad (k = 1, \ldots, K) \tag{2.61}$$

Segundo Hastie, Tibshirani e Friedman (2009) a função $\sigma(v)$ é uma função de

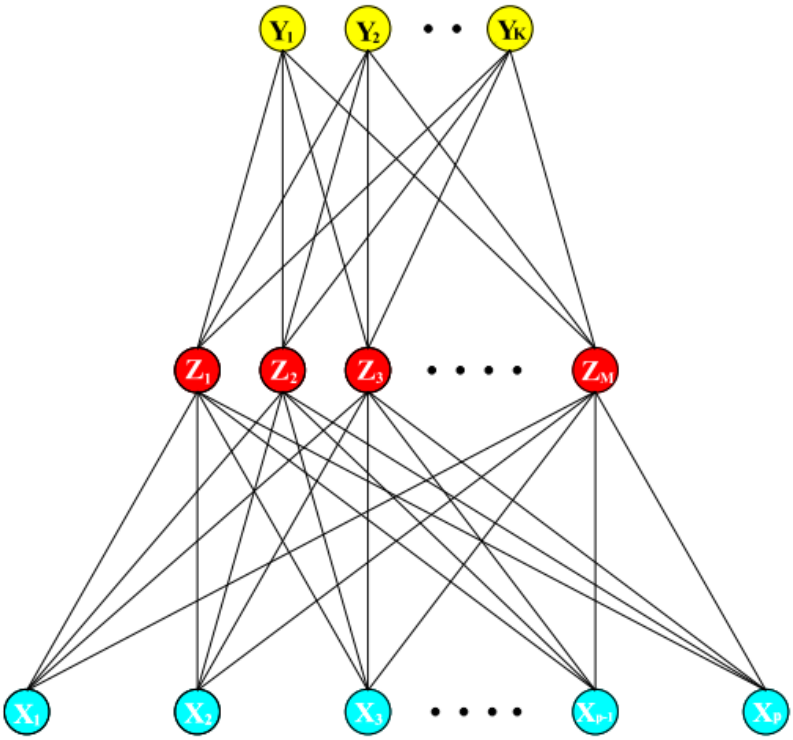


Figura 2.10: Esquema de uma rede neural (Hastie, Tibshirani e Friedman, 2009, p. 393).

ativação dos neurônios que aproxima uma função degrau, geralmente igual a:

$$\sigma(v) = \frac{1}{1 + e^{-v}} \tag{2.62}$$

E a função de saída:

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}} \tag{2.63}$$

Na figura 2.10 cada uma das entradas X são alteradas pelas funções Z (chamada de camada oculta) que, juntas, produzem as saídas Y.

Os parâmetros desconhecidos, denominados pesos, formam o conjunto θ.

$$\{\alpha_{0m}, \alpha_m; m = 1, \ldots, M\} \quad M(p+1) \text{ pesos} \tag{2.64}$$

$$\{\beta_{0k}, \beta_k; k = 1, \ldots, M\} \quad K(M+1) \text{ pesos} \tag{2.65}$$

A função erro para regressão:

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2 \tag{2.66}$$

E para classificação:

$$R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i) \tag{2.67}$$

Com classificador $G(x) = \text{argmax}_k f_k(x)$.

É comum a função $R(\theta)$ ser minimizada localmente por gradiente descendente. O Mínimo global é indesejado por ser indício de *overfitting* Hastie, Tibshirani e Friedman (2009).

Hastie, Tibshirani e Friedman (2009) dizem ser melhor normalizar as entradas de forma que tenham média 0 e variância 1; e que os pesos iniciais tenham valores aleatórios próximos a zero (tipicamente $[-0.7, +0.7]$). Além disso, o número de neurônios e camadas destes (ocultas) costuma ser guiada por experimentação.

A normalização pode ser feita pela equação:

$$\hat{x}_i = \frac{x_i - x_{\text{média}}}{x_{\text{máx}} - x_{\text{mín}}} \tag{2.68}$$

Computacionalmente, Hastie, Tibshirani e Friedman (2009) dizem que treinar uma rede neural requer $O(N_p M L)$ operações, para N observações, M unidades ocultas e L épocas de treinamento.

2.6.2 A maldição da dimensionalidade

Quando tenta-se construir modelos preditivos em espaços com grande dimensão (número de variáveis), a quantidade de dados necessários para que se tenha um modelo estatísticamente relevante cresce exponencialmente. Este fenômeno é chamado por Hastie, Tibshirani e Friedman (2009) de Maldição da Dimensionalidade (*dimensionality curse*). Isto está associado ao crescimento do volume do espaço, que pode ser encarado como uma medida de esparcidade dos dados; i.e., os dados serão considerados esparsos se eles ocuparem uma fração pequena do volume do espaço.

2.6.3 Erro e complexidade do modelo

É desejável que o modelo preditivo tenha um erro mínimo. É importante que o modelo não descreva bem demais os dados de treinamento. A figura 2.11 ilustra esse fenômeno, chamado por Hastie, Tibshirani e Friedman (2009) de *overfitting* quando a complexidade do modelo é alta e *underfitting*, quando ela é baixa.

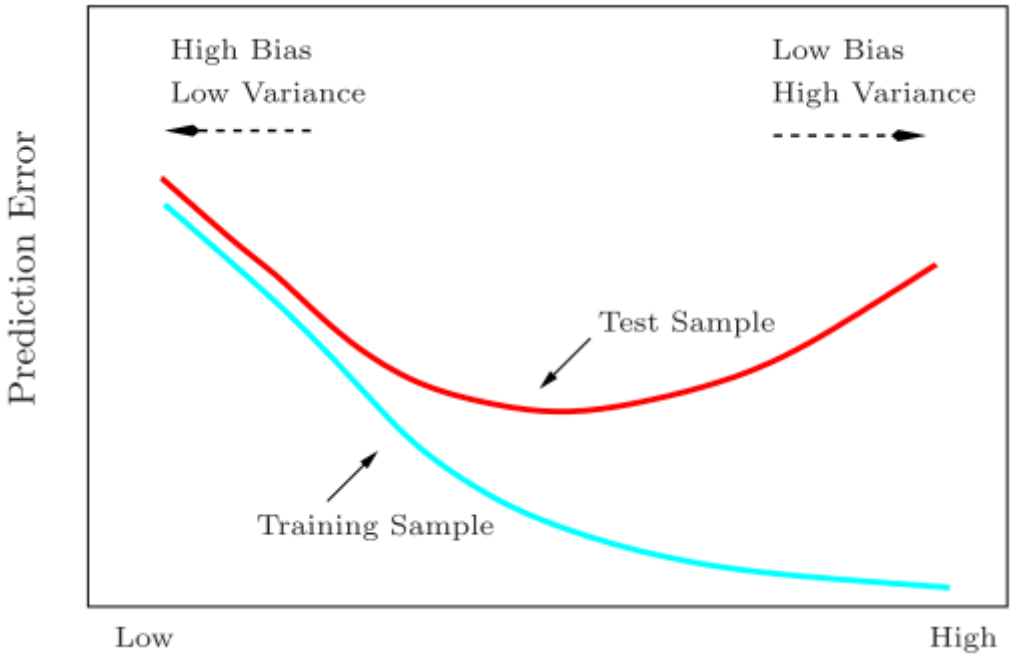


Figura 2.11: Relação do erro do modelo e a complexidade dele (Hastie, Tibshirani e Friedman, 2009, p. 38).

2.6.4 Métodos de aprendizado não supervisionado

No aprendizado não supervisionado, por definição, não há uma variável y que possa ser considerada saída, ou resposta, de um conjunto de variáveis (entradas) x_i. É exatamente por isto que Hastie, Tibshirani e Friedman (2009) argumentam que é difícil verificar qual método de estimação é melhor ou mais apropriado. Pois, por não haver uma variável de saída, não e possível estimar o erro.

2.6.4.1 Regras de associação.

É um método que tenta encontrar grupos de varíaveis $(x_i, \ldots, x_n)$ que aparecem mais frequentemente. É mais comum ser aplicado a dados binários, i.e. variáveis que representam a presença ou não de um atributo (Hastie, Tibshirani e Friedman, 2009).

2.6.4.2 Análise de aglomeração.

Também chamado por Hastie, Tibshirani e Friedman (2009) de segmentação de dados, é um método que busca agrupar os dados, e também os grupos propriamente, em estruturas hierárquicas. Este metodo requer uma noção de grau de similaridade entre objetos individuais.

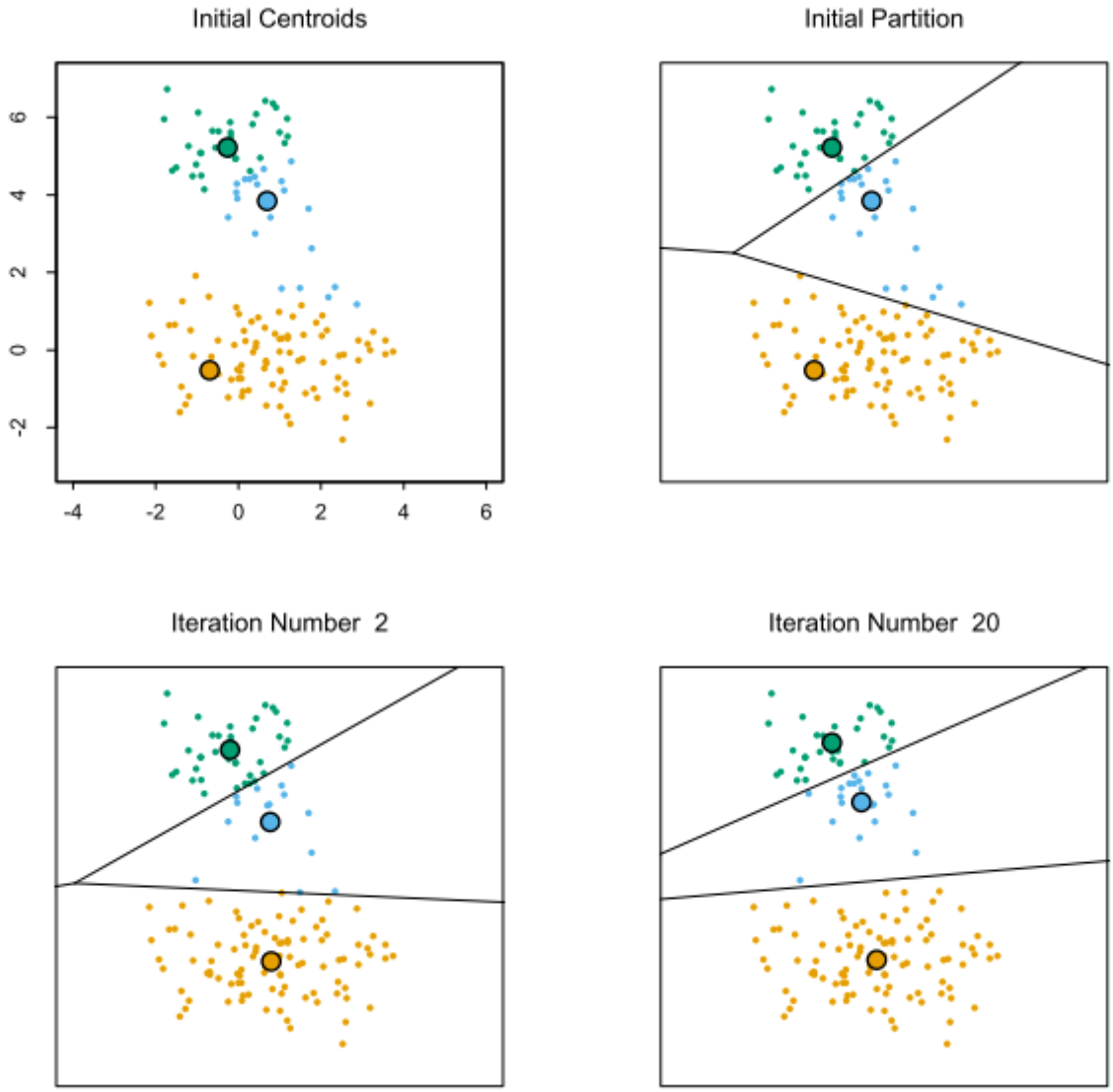


Figura 2.12: Exemplo de aglomeração 3-mean (Hastie, Tibshirani e Friedman, 2009, p. 511).

2.6.4.3 K-médias

Um modelo de análise de aglometação é o chamado *K-means clustering* (aglomeração de K-médias). Nele os dados são agrupados em K grupos usando a média geométrica das similaridades das variáveis. Hastie, Tibshirani e Friedman (2009) dizem que os algoritmos mais comuns tem como entrada uma matriz de dissimilaridades que, geralmente, é o quadrado da distância euclidiana entre os pontos.

O algoritmo que implementa K-médias faz K aglomerações achando mínimo local da variância de cada grupo em respeito à média deles. Cara cada conjunto de médias, cada observação é atribuída ao grupo cuja média é mais próxima do seu valor. O algoritmo termina quando atinge um mínimo local e, por isso, o resultado pode ser subótimo (Hastie, Tibshirani e Friedman, 2009). A figura 2.12 mostra um conjunto de dados aglomerados por esse algoritmo e o resultado dele após 20 iterações.

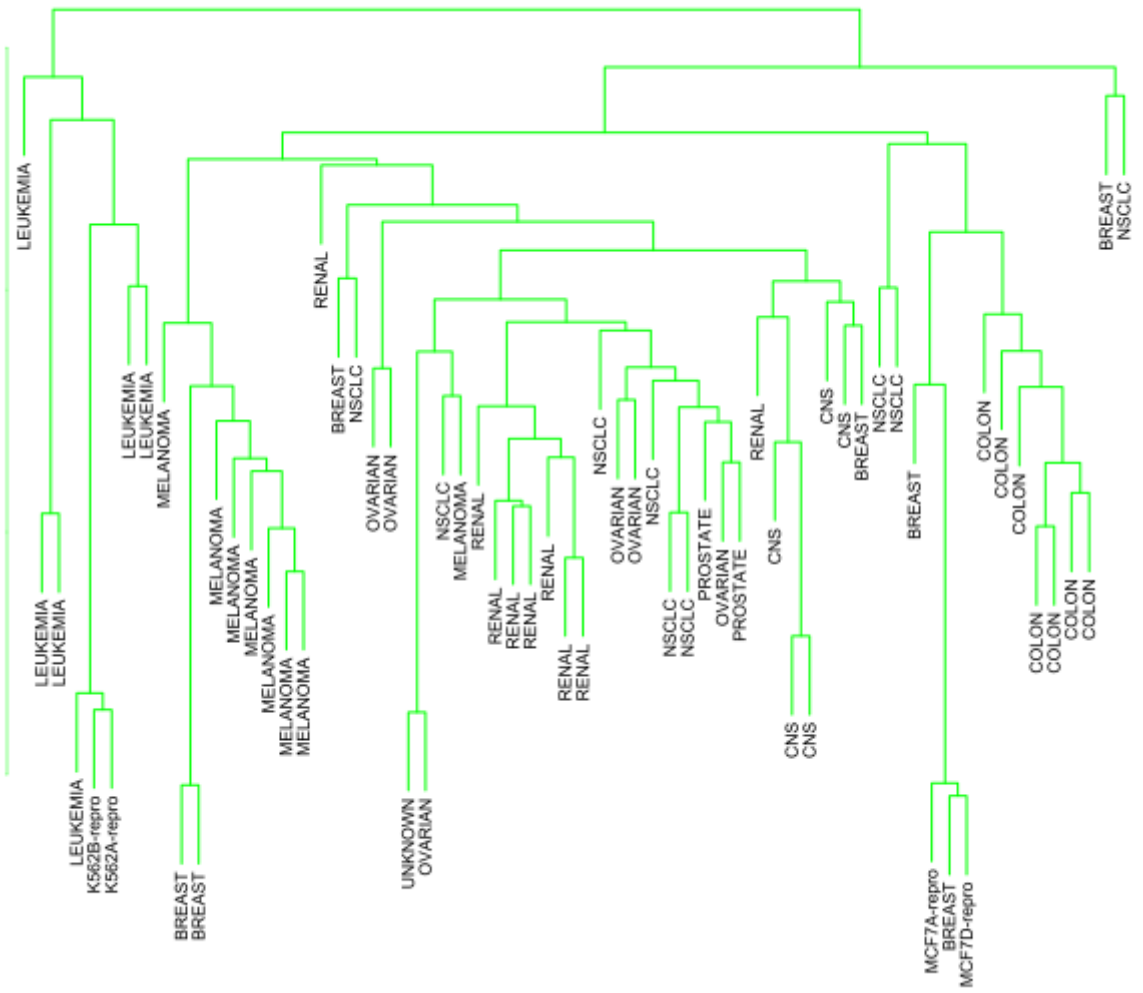


Figura 2.13: Exemplo de dendograma (Hastie, Tibshirani e Friedman, 2009, p. 522).

2.6.4.4 Aglomeração hierárquica

Na aglomeração hierárquica os grupos vão sendo formado em hierarquias, onde cada grupo é aglomerado ao outro mais similar. Duas estratégias podem ser usadas: aglomerativa (*bottom-up*) ou divisiva (*top-down*) (Hastie, Tibshirani e Friedman, 2009).

Este tipo de aglomeração produz o dendograma (figura 2.13), onde a altura dos nódulos é proporcional a dissimilaridade do grupo. No nível mais baixo estão as observações individuais. É devido a este gráfico que, segundo Hastie, Tibshirani e Friedman (2009), este método é popular.

2.6.4.5 Aglomeração espectral

Métodos de aglomeração como a hierárquica e K-média usam uma métrica elíptica ou esférica (Hastie, Tibshirani e Friedman, 2009). Quando os grupos são não-convexos, tais métodos virão a falhar. A aglomeração espectral foi desenvolvida para

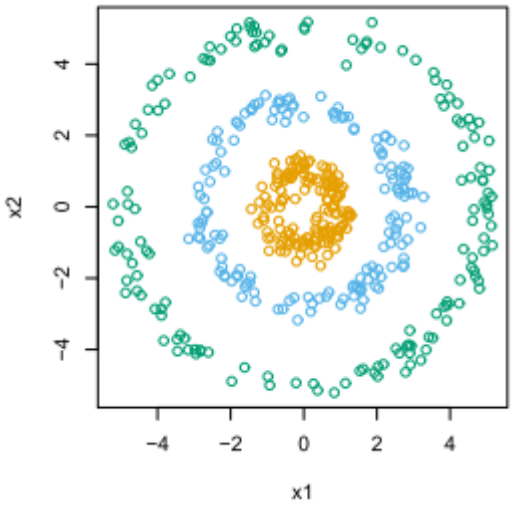


Figura 2.14: Exemplo de aglomeração espectral (Hastie, Tibshirani e Friedman, 2009, p. 546).

este tipo de situação. A figura 2.14 ilustra uma aglomeração espectral.

2.7 Análise e Processamento de Sinais

2.7.1 Série de Fourier

A expansão em série de Fourier de uma função periódica é a decomposição da função em termos das suas componentes de várias frequências (Sampaio, Ferreira e Brandão, 2006).

Em outras palavras, uma função periódica pode ser expressa como um somatório de funções senoidais de diferentes frequências, fase e amplitude. As figuras 2.15, 2.16 e 2.17 ilustram esse conceito. Nelas, as componentes de frequência são obtidas a partir da transformada de Fourier. O código usado para gerá-las encontra-se no apêndice A.

Quando o sinal com o qual estamos trabalhando for não periódico, Sampaio, Ferreira e Brandão (2006) e Prandoni e Vetterli (2008) sugerem expressá-lo como uma soma contínua (integral) de sinais exponenciais.

2.7.2 Sinal discreto

O armazenamento de dados em um computador (sinais digitais) impõe uma restrição: não é possível ter valores contínuos no tempo, isto é, os valores são discretos. Isto porque armazenar os valores continuamente exigir-se-ia infinitos pontos, i.e., uma memória infinita. Assim sendo, sinais digitais são discretos.

Segundo Prandoni e Vetterli (2008), um sinal discreto é uma sequência de

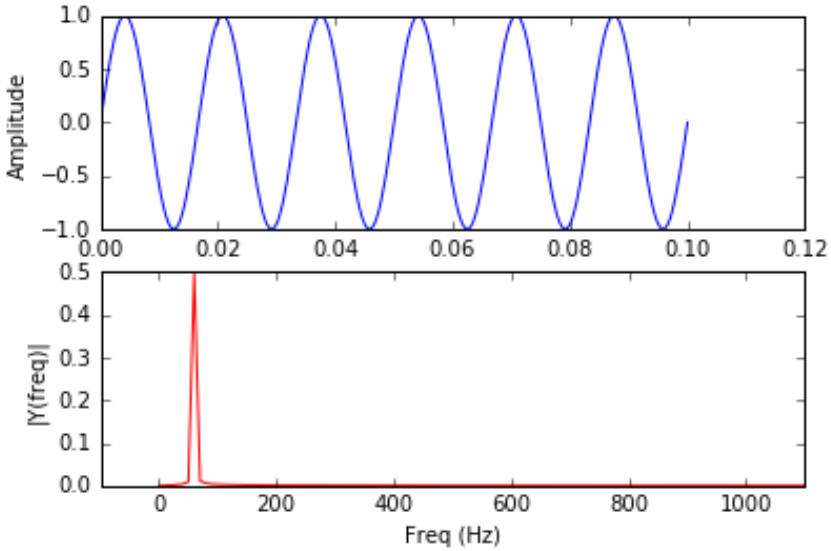


Figura 2.15: $f(t) = \sin(2\pi 60t)$

valores complexos, e.g.

$$x(n) = e^{j\frac{\pi}{20}n} \qquad n = 1, 2, 3, \ldots, N \tag{2.69}$$

2.7.3 Sinais como vetores

Seja $x(n)$ um sinal com N amostras. Prandoni e Vetterli (2008) sugerem visualizar tal sinal como um vetor com N elementos e, assim sendo, definido em um espaço euclidiano (complexo) de dimensão N, i.e., $\mathbb{C}^N$.

2.7.3.1 Base de Fourier em $\mathbb{C}^N$

A base de um espaço vetorial é um conjunto de vetores linearmente independente tal que qualquer vetor do espaço é uma combinação linear da base.

Prandoni e Vetterli (2008) definem a base de Fourier de um espaço o conjunto de vetores W^k, para o qual

$$W_N^{kn} = e^{-j\frac{\pi}{N}nk} \tag{2.70}$$

é o k-ésimo veor da amostra n. k corresponde a uma determinada frequência (rad/amostra).

2.7.4 Transformada de Fourier

Tanto Sampaio, Ferreira e Brandão (2006) quanto Prandoni e Vetterli (2008) relacionam uma função no domínio da frequência e do tempo através das seguintes

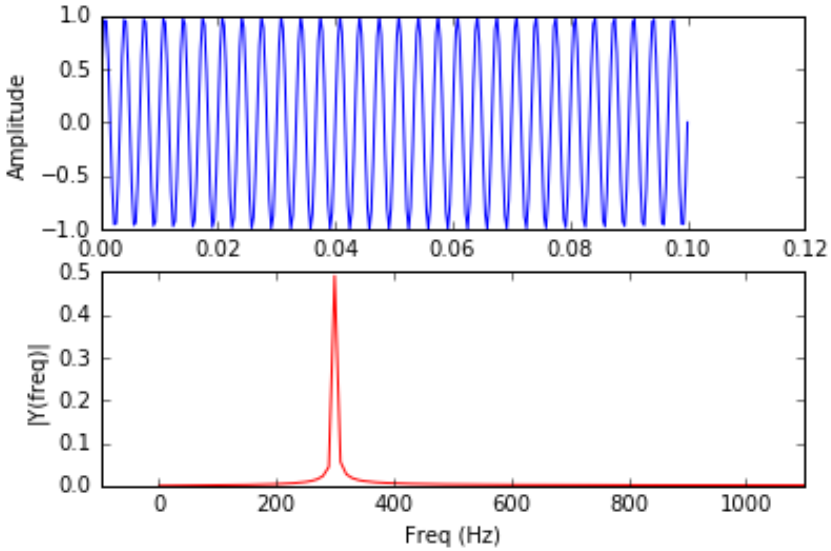


Figura 2.16: $f(t) = \sin(2\pi 300t)$

equações:

$$F(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-j\omega t}dt \tag{2.71}$$

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)e^{j\omega t}d\omega \tag{2.72}$$

Segundo Sampaio, Ferreira e Brandão (2006, p. 27), se a função $f(t)$ pertence ao espaço $L^1(\mathbb{R})$, então a função transformada $F(\omega)$ existe. Se a função $f(t)$ pertence ao espaço $L^2(\mathbb{R})$, então a função transformada $F(\omega)$ existe e $F(\omega) \in L^2(\mathbb{R})$.[a]

2.7.4.1 Transformada discreta Fourier

A transformada de Fourier de um sinal discreto tem a forma Prandoni e Vetterli (2008):

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \tag{2.73}$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-kn} \tag{2.74}$$

Onde $k = \omega$ é uma dada frequência discreta. E N é o comprimento da amostra.

Como demonstram Prandoni e Vetterli (2008), a transformada discreta de Fourier pode ser vista como uma mudança de base em um espaço vetorial. As transfor-

[a] Os espaçoes L^p são chamados espaços de Lebesgue (Zakon, 1977); interprete como o espaço de funções p-integráveis.

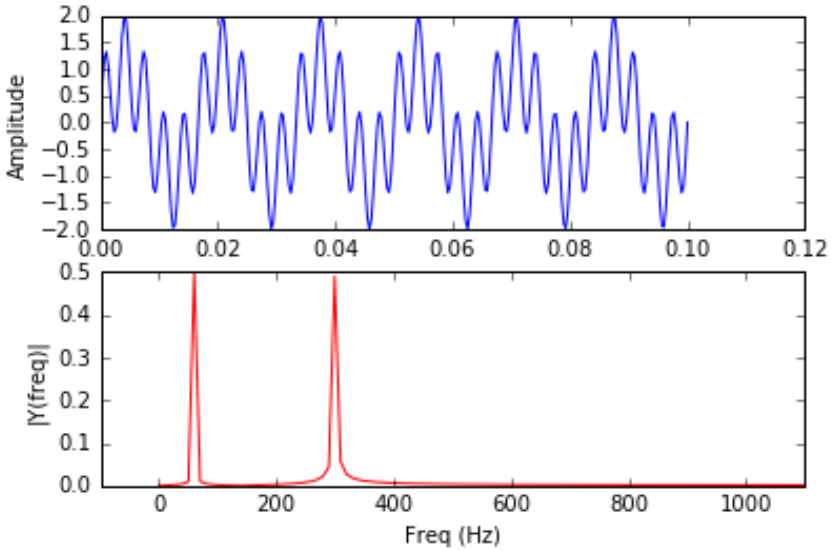


Figura 2.17: $f(t) = \sin(2\pi 60t) + \sin(2\pi 300t)$

madas acima em forma matricial são:

$$X_k = W x_n \tag{2.75}$$

$$x_n = W^H X_k \tag{2.76}$$

Na qual W é uma matriz quadrada de ordem N formada pelos vetores da base de Fourier. E W^H é a matriz transpota conjugada de W.

2.7.5 O Teorema da Amostragem

Segundo **Johnson2013**, o teorema que se segue é atribuído a Harold Nyquist, e depois revivido por Claude Shannon quando os computadores se tornaram públicos diz que é possível converter uma função do tempo em uma sequência de números e, depois, converter a sequência de volta na função do tempo sem erro. O teorema a seguir é chamado por Sampaio, Ferreira e Brandão (2006) de Teorema da Amostragem Uniforme:

> Um sinal limitado em faixa, i.e., que não tem nenhuma componente espectral acima da frequência Ω_N rad/s, é determinado univocamente por suas amostras, $x(n) = x_c\{nT\}$ com $(n = 0, +1, -1)$ ... tomadas a intervalos uniformes menores do que $\frac{\pi}{\Omega_N}$ segundos. Isto é, o período de amostragem deve satisfazer $T < \frac{\pi}{\Omega_N}$. A freqüência de amostragem $\Omega = \frac{2\pi}{T}$ deve, então, satisfazer $\Omega > 2\Omega_N$. Chamamos o número $2\Omega_N$ de frequência de Nyquist. (Sampaio, Ferreira e Brandão, 2006, p. 49)

Segundo **Johnson2013**, respeitadas as condições acima, é possível recuperar o sinal original por um filtro passa-baixa com frequência de corte igual a $\frac{2\pi}{\Omega_N}$ sem

introduzir erro, i.e., *aliasing* ou mascaramento.

Johnson2013 diz que a Frequência de Nyquist também é conhecida como Frequência de Amostragem de Shannon.

Nem sempre uma taxa de amostragem maior é melhor. Isto porque mais amostras significam, também, mais erros.

2.7.5.1 Erro de quantização.

Como afirma **Johnson2013**, a conversão, por um computador, de um sinal analógico (contínuo no tempo) para um digital introduz erro, pois o valor do sinal discreto deve ser múltiplo de um valor inteiro.

Devido à memória finita dos computadores, não é possível representar todos os valores (amplitude) que um sinal possa ter. Assim sendo, os valores armazenados em computadores constituem uma aproximação. Sampaio, Ferreira e Brandão (2006) chama a diferença entre o valor da amostra quantizada e do valor do sinal de erro de quantização. Segundo os mesmos, este erro geralmente não é conhecido, e um modelo estatístico deve ser usado.

"A representação estatística de erros de quantização é baseada nas seguintes considerações" (Sampaio, Ferreira e Brandão, 2006, p. 59).

1. O erro de quantização $e(n)$, é uma sequência de amostras de um processo aleatório estacionário.

2. A sequência de erro $e(n)$ não é correlata com a sequência $x(n)$ do sinal original.

3. As variáveis aleatórias do processo de erro não são correlatas, isto é, o erro é um processo de ruído branco.

4. A distribuição de probabilidade do processo de erro é uniforme.

3 Desenvolvimento

3.1 Simulação com o ATP-EMTP

Percebeu-se que o ATPDraw, quando executa uma simulação, compila o circuito e então invoca alguns comandos no prompt de comando do Windows. Como demonstra a figura 3.1. Conhecendo-os, é possível criar um código para executar simulações sucessivas de forma automática, em um volume impraticável para qualquer humano.

Os comandos que permitem tal execução automática são:

```
COPY C:\ATPdraw\STARTUP STARTUP
```

```
C:\ATPdraw\tpbig1 'arquivo-com-cards.atp' > 'arquivo-resultado.lis'
```

Eles intruem o computador a abrir o arquivo-com-cards.atp com o programa tpbig1 que é o ATP-EMTP propriamente) e salvar no arquivo-resultados.lis.

Para realizar as simulações dos curto-circuitos na linha de transmissão sendo simulada, criaram-se, no ATPDraw, arquivos que serviram de base. Eles são: *template* e seções da linha de transmissão, e um circuito que modela o sistema de potência sendo simulado. A partir deles, criou-se um código para replicá-lo alterando alguns valores chave, como ponto que ocorre o curto-circuito e valor da resistência de falta.

3.1.1 *Template* da linha simulada

Criou-se um arquivo com uma *LCC Template* com dados conforme figuras 3.2 e 3.3. A linha utiliza o modelo JMarti, tem 1 km, não transposta, efeito pelicular incluso. Ela é trifásica, um condutor por fase, e dois cabos guarda (para-raios). Deu-se a ela o título lttcc. Compilou-se o modelo gerando o arquivo lttcc.lib e exportou-o, gerando o arquivo lttcc.alc.

Conforme figura 3.4, mais nove seções LCC foram criadas, utilizando o template. Elas foram nomeadas L1 a L10, cada uma sendo 1 km mais longo que a antecessora. Assim, L8 significa uma seção de linha de 8 km. Os nós de cada seção foram nomeados, e.g., XL08.

O circuito da figura 3.4 (linhasL1L10.atp) foi então compilado, gerando os arquivos lttcc.lib, e lttcc-L1.lib a lttcc-L10.lib.

A figura 3.5 mostra parte do arquivo compilado que representa o circuito. É possível identificar como o ATPDraw nomeou (automaticamente) cada seção da linha através do nome dos nós de cada seção. Dependendo da forma que o circuito for

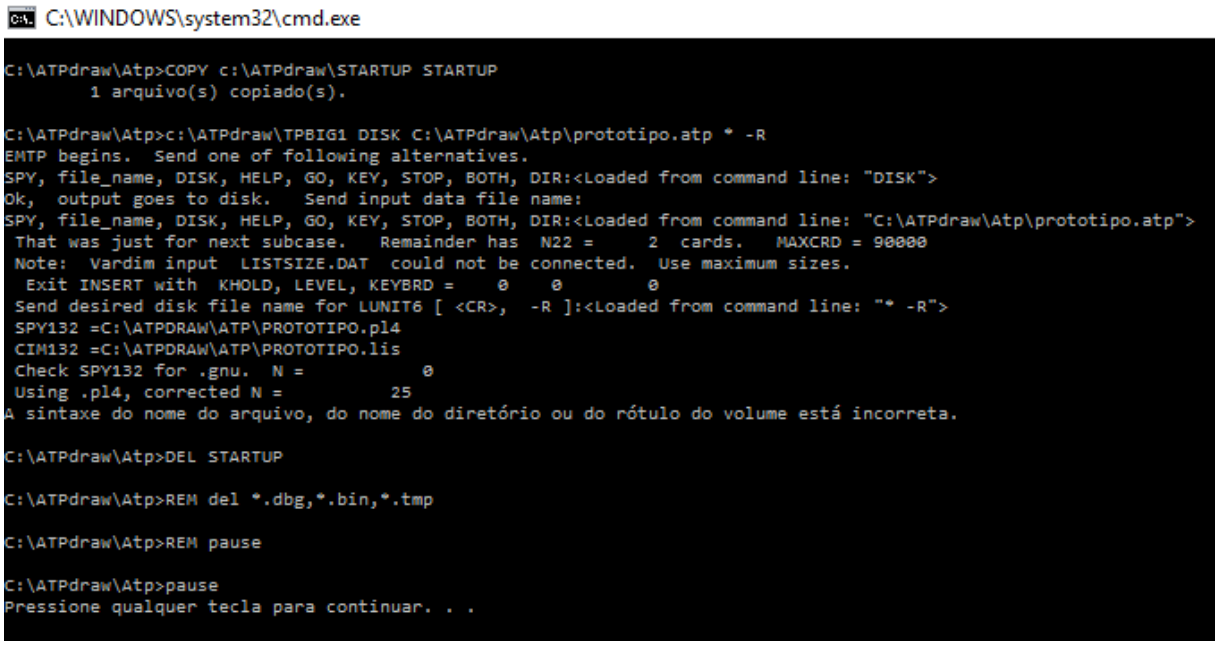


Figura 3.1: Prompt de comando quando o EMTP-ATP é executado pelo ATPDraw.

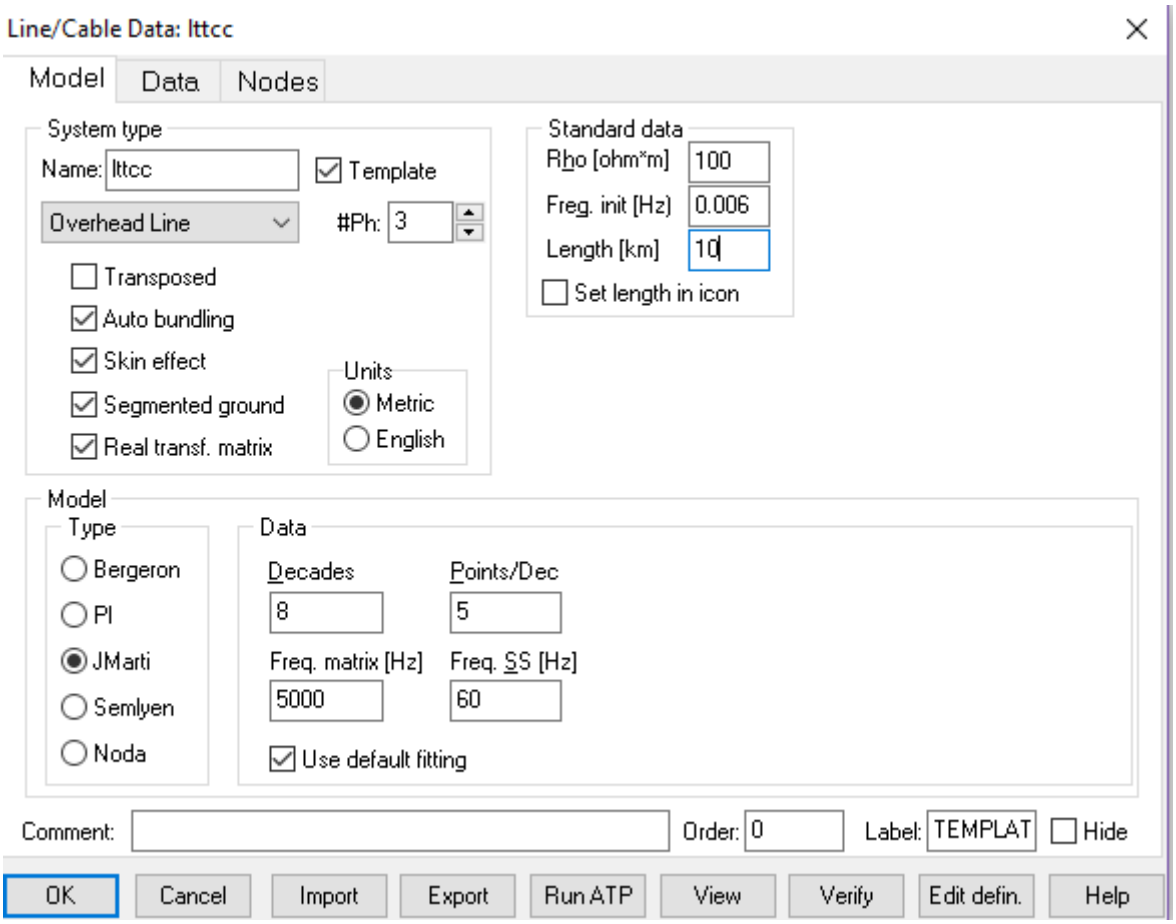
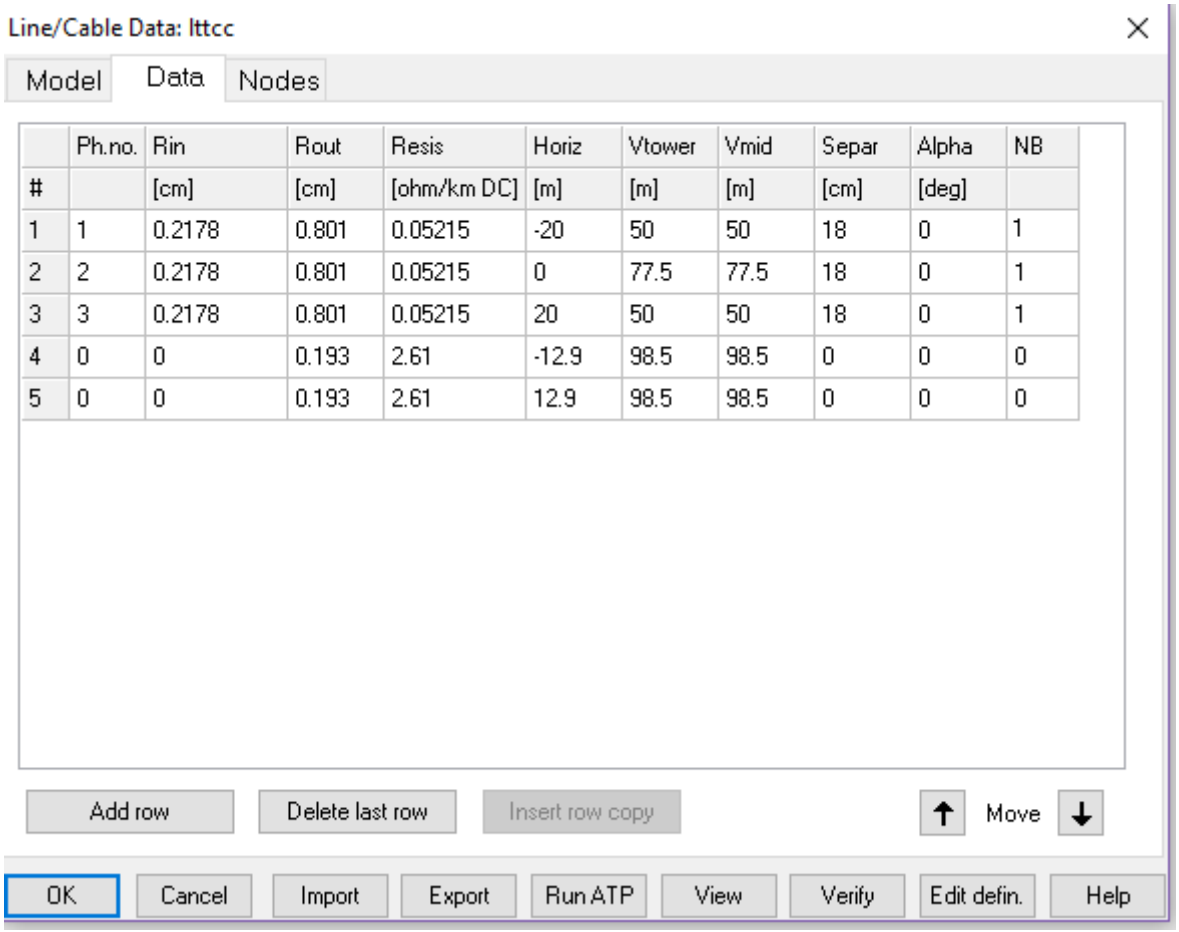


Figura 3.2: Modelo de linha usando JMarti.



#	Ph.no.	Rin [cm]	Rout [cm]	Resis [ohm/km DC]	Horiz [m]	Vtower [m]	Vmid [m]	Separ [cm]	Alpha [deg]	NB
1	1	0.2178	0.801	0.05215	-20	50	50	18	0	1
2	2	0.2178	0.801	0.05215	0	77.5	77.5	18	0	1
3	3	0.2178	0.801	0.05215	20	50	50	18	0	1
4	0	0	0.193	2.61	-12.9	98.5	98.5	0	0	0
5	0	0	0.193	2.61	12.9	98.5	98.5	0	0	0

Figura 3.3: Informações geométricas da linha.

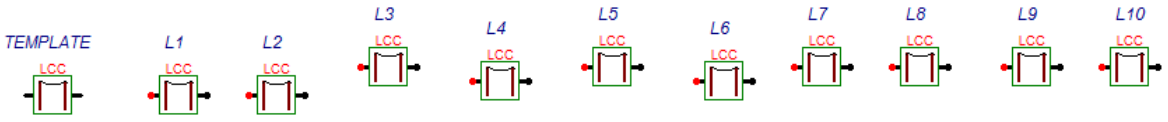


Figura 3.4: Circuito para compilar as seções da linha: de 1 a 10 km.

```
$INCLUDE, C:\ATPdraw\Atp\lttcc_L1.lib, XL01A#, XL01B#, XL01C#, X0001A $$
  , X0001B, X0001C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L2.lib, XL02A#, XL02B#, XL02C#, X0002A $$
  , X0002B, X0002C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L3.lib, XL03A#, XL03B#, XL03C#, X0003A $$
  , X0003B, X0003C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L4.lib, XL04A#, XL04B#, XL04C#, X0004A $$
  , X0004B, X0004C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L5.lib, XL05A#, XL05B#, XL05C#, X0005A $$
  , X0005B, X0005C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L6.lib, XL06A#, XL06B#, XL06C#, X0006A $$
  , X0006B, X0006C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L7.lib, XL07A#, XL07B#, XL07C#, X0007A $$
  , X0007B, X0007C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L8.lib, XL08A#, XL08B#, XL08C#, X0008A $$
  , X0008B, X0008C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L9.lib, XL09A#, XL09B#, XL09C#, X0009A $$
  , X0009B, X0009C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL10A#, XL10B#, XL10C#, X0010A $$
  , X0010B, X0010C
```

Figura 3.5: Parte do *card* que gerou as seções de 1 a 10 km.

desenhado, o ATP pode, por exemplo, dar o nome lttcc-L5.lib à seção de linha L2. Isto parece estar correlacionado com a ordem com que cada seção foi criada.

É importante que os arquivos compilados pelo circuito da figura não sejam sobrescritos pelo ATP.

3.1.2 Modelo do circuito base

A linha de transmissão simulada com o EMTP-ATP foi aproximada por várias seções de linha. O objetivo é facilitar a manipulação do arquivo que a representa de forma a mudar o ponto de falta.

O circuito foi montado de acordo com a figura 3.6. O lado receptor (à esquerda) e emissor (à direita) são equivalentes Thévenin por fase. Supõe-se que uma seção de linha de 100 km gerará o mesmo resultado que 10 seções de 10 km.

O lado receptor é considerado o ponto zero (0º km), onde estão os medidores de tensão e corrente.

Consiste de uma linha de 100 km, com transposição a cada 20 km. Cada segmento do modelo padrão tem 10 km. A ordem das transposições (esquerda para a direita) é:

1. ABC-BCA
2. ABC-CAB
3. ABC-CBA

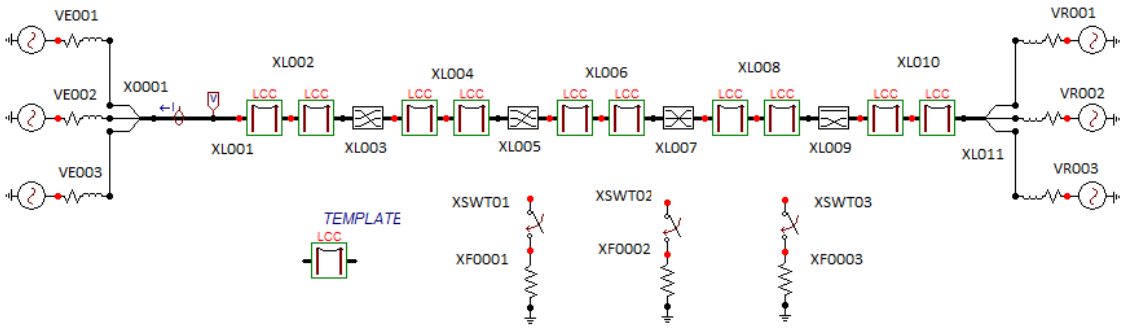


Figura 3.6: Circuito base desenhado no ATPDraw.

Tabela 3.1: Valores dos equivalentes Thévenin do circuito base.

Fase	Tensão (kV)	Impedância (Ω, para 60 Hz)
Emissor A	$241.5 \angle 0^o$	$0.050 + j11.309$
Emissor B	$241.5 \angle 120^o$	$0.050 + j11.309$
Emissor C	$241.5 \angle 240^o$	$0.050 + j11.309$
Receptor A	$225.5 \angle 0^o$	$6.000 + j37.700$
Receptor B	$225.9 \angle 122^o$	$5.000 + j45.239$
Receptor C	$225.2 \angle 241^o$	$5.000 + j52.779$

4. ABC-ACB

Os nós foram nomeados para permitir a identificação deles. O *card* gerado pelo ATPDraw após a compilação encontra-se no apêndice F.

A ideia é criar um programa que, a partir desse *card* base, gerará outros a serem simulados mudando a conexão dos nós, valores dos elementos, etc. E então executará o EMTP-ATP de forma automática para cada.

A tabela 3.1 contém os valores dos equivalentes Thévenin do circuito base. A frequência do sistema é 60 Hz.

3.1.3 Situações a serem simuladas

A linha simulada possui uma torre a cada quilômetro inteiro. Todas as faltas simuladas acontecem em uma torre; para representar faltas decorridas do rompimento do dielétrico dos isoladores. Nenhuma situação de rompimento de cabo foi simulada.

Criou-se um código (em Python, vide apêndice B e C) para variar o ponto de conexão da chave na linha, e também os comprimentos das seções da linha de forma a se obter 99 arquivos, cada um correspondente a um ponto de falta (1º ao 99º km). O ponto de referência (0º km) é o nó de medição no ramo receptor (à esquerda na figura 3.6).

Por exemplo: para fazer um curto-circuito no 3º km, troca-se a primeira seção

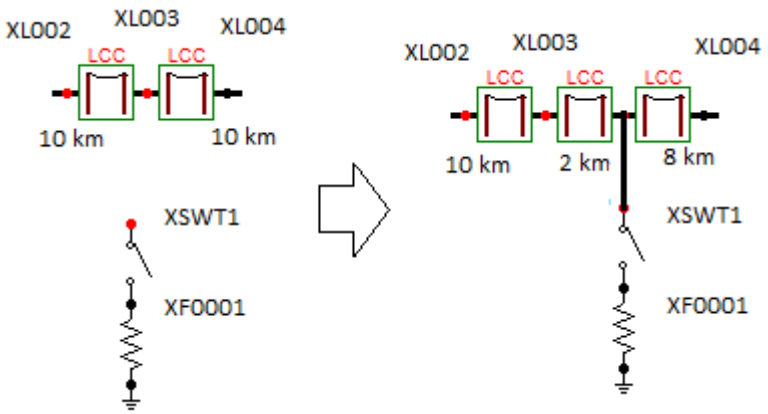


Figura 3.7: Inserção de curto-circuito fase-terra em um ponto.

de linha por duas: uma de 3 km e outra de 7 km, nomeando o nó que conecta as duas de XSWT0, o nó que conecta à chave. A figura 3.7 ilustra esse conceito:

No caso de o ponto do curto-circuito for um múltiplo inteiro de 10, então o nome do nó XSWT0 é substituído pelo XL00N correspondente ao ponto da falta, e não há alteração das seções da linha.

Todos curto-circuitos foram para terra: fase-terra, fase-fase-terra, e fase-fase-fase-terra. Variou-se o ponto (km) da falta para cada uma dessas situações. Todas as situações com resistência de falta de 500 $m\Omega$. Uma função para variar a resitência de falta tenha sido criada (B), contudo ela não foi usada.

3.1.3.1 Configuração da simulação.

As chaves que causam o curto-circuito fecham em $t = 0.05s$ em todas as situações simuladas. Uma função para variar este valor não foi criada.

Simulation:

- Passo de tempo $\triangle t = 1\mu s$;
- Tempo máximo $t_{max} = 1s$;
- *Time-domain simulation*.

Output:

- *Print. freq. = 1*;
- *Plot freq. = 1*;
- *Plotted output*.

3.1.3.2 Observação sobre o nome das barras

As simulações com os arquivos alterados pelos algoritmos do apêndice B e C estavam dando erro. Eles pararam quando todos os nós (barras) receberam um nome com seis caracteres.

Assim, o arquivo que tinha barras com nome XSWT1 dava erro ao simular. Quando ele foi renomeado para XSWT01 o erro parou.

3.1.3.3 Suposições.

Os equivalentes Thévenin do sistema de potência são insensíveis à condição de falta, i.e., não se alteram durante a ocorrência dela.

3.2 Localização do ponto de falta por rede neural

Julgou-se a Rede Neural como o algoritimo mais apropriado para se fazer o aprendizado de máquina a fim de localizar as faltas. O código para formatar os dados e treinar uma rede neural com os mesmos encontram-se no apêndice E.

3.2.1 Transformação dos dados

Os dados da simulação são lidos e amostrados na frequência de 500 Hz, a fim de simular um medidor digital. É feita a transformada de Fourier dos dados (tensão e corrente nas três fases). A parte real e imaginára dos resultados são separadas e concatenadas em uma lista ordenada (*array*) da forma:

$$[\quad Re(V_A(\omega)), Re(V_B(\omega)), Re(V_C(\omega)), Re(I_A(\omega)), Re(I_B(\omega)), Re(I_C(\omega)),$$

$$Im(V_A(\omega)), Im(V_B(\omega)), Im(V_C(\omega)), Im(I_A(\omega)), Im(I_B(\omega)), Im(I_C(\omega))\quad]$$

Depois os dados são normalizados de forma a terem média 0 e variância 1, fazendo-se para cada um dos valores:

$$\hat{x}_i = \frac{(x_i - x_{\text{média}})}{\sigma} \tag{3.1}$$

Onde σ é o desvio padrão dos dados. A média e o desvio padrão usados na equação 3.1 foram calculados a partir de todos os valores transformados, sem distinção entre corrente ou tensão. Em outras palavras, após a transformação de Fourier de todos os dados e a separação da parte real da imaginária, a média e o desvio padrão foram calculados utilizando todos esses valores, com todos os dados simulados (foram calculados apenas uma vez).

3.2.2 Rede Neural

Dois tipos de rede neural foram treinadas com as mesmas configurações: classificadora e regressora.

Configurações:

- 100 camadas ocultas;
- ativação logistica dos neurônios (função sigmoid);
- método de resolução adam (uma variação do gradiente descendente, segundo Pedregosa et al. (2011));
- taxa de aprendizado constante igual a 0.001;

A rede do tipo classificadora foi treinada com todos os dados simulados.

A rede regressora foi treinada com apenas uma parcela dos dados simulados, sendo outra parcela reservada para teste. Essa separação dos dados foi feita de forma pseudo-aleatória; os *seed*[a] usados estão presentes no código do apênice E.

[a]Geração de números pseudo-aleatórios é deterministica, mas de forma a resultar numa distribuição gaussiana deles. Utilizar um mesmo *seed* nos códigos de computador permite obter sempre os mesmos números. Isto é útil para reprodutibilidade.

4 Resultados e Discussão

4.1 Simulações com o ATP-EMTP

As figuras 4.1 até 4.14 mostram o resultado das simulações descritas na seção 3.1 para uma falta no km 83, para todas as combinações de fases faltosas. Todos os valores referem-se a tensão e corrente no início da linha (visto do lado receptor). As figuras foram geradas com o código no apêndice D.

As figuras 4.15 até 4.20 mostram as frequências presentes durante uma falta com as fases AB-terra. Gráficos obtidos a partir da transformada de fourier dos dados simulados após eles terem sidos amostrados numa frequência de 500 Hz, para simular um medidor. O módulo das frequências está em escala logarítimica.

4.2 Treinamento da Rede Neural

4.2.1 Classificadora

Após o treinamento da rede neural com todos os dados simulados, estes mesmos dados foram usados para testá-la. Das 693 combinações de fases faltosas com ponto de falta, 618 foram previstos corretamente; equivalente a 89.18 % (o *score*) da rede, porcentagem de acertos.

Pela inspeção da tabela 4.1, é possível perceber que os erros foram em relação às fases faltosas e, quando errou o ponto da falta (sete erros), este erro não foi maior que 1 km.

4.2.2 Regressora

Cada vez que uma rede neural do tipo regressora foi criada, treinada e testada, os resultados dos testes variaram de forma consistente. O erro absoluto, em km, foi calculado como:

$$e = |previsto - correto| \tag{4.1}$$

Em uma situação que reserva 15.44 % dos dados para teste, a rede apresentou um *score* de 94.21 % (tabela 4.2 e figura 4.21). Em outra situação, reservando 13.56 % dos dados para teste, a rede apresentou *score* de 87.00 % (tabela 4.3 e figura 4.22). Numa terceira situação, com reserva de 12.41 % dos dados para teste, os *score* da rede ficaram em torno de 57.02 % (tabela 4.4 e figura 4.23).

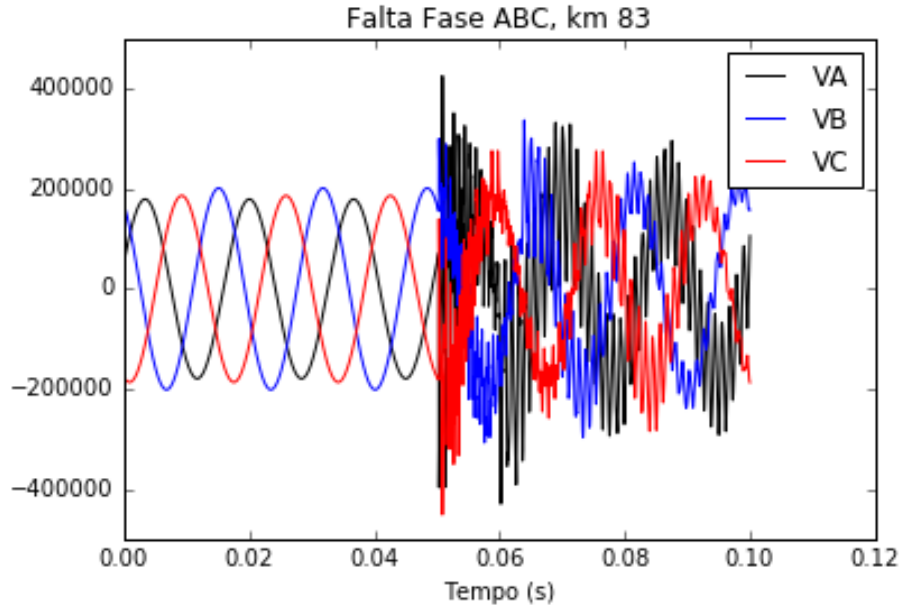


Figura 4.1: Tensões quando simulada uma falta nas fases ABC, km 83.

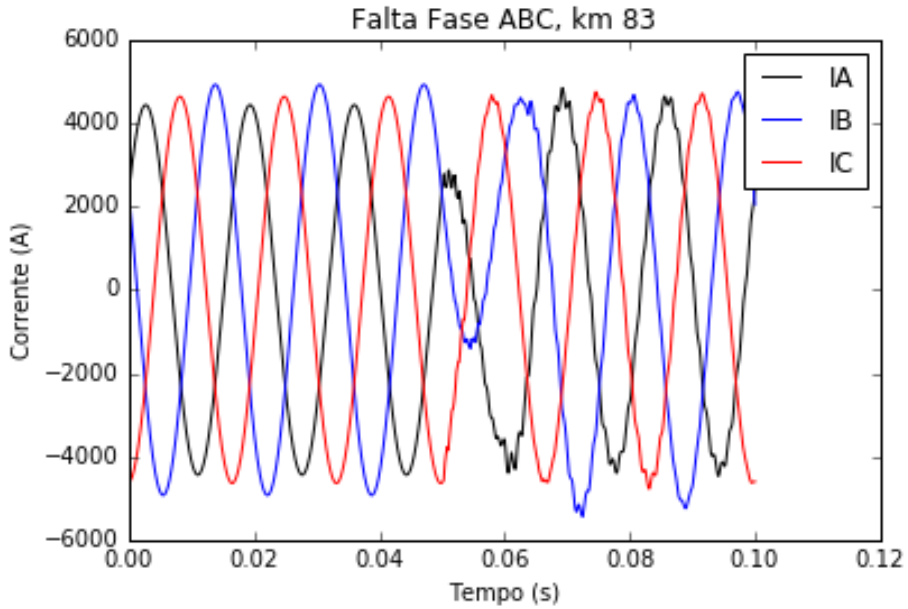


Figura 4.2: Correntes quando simulada uma falta nas fases ABC, km 83.

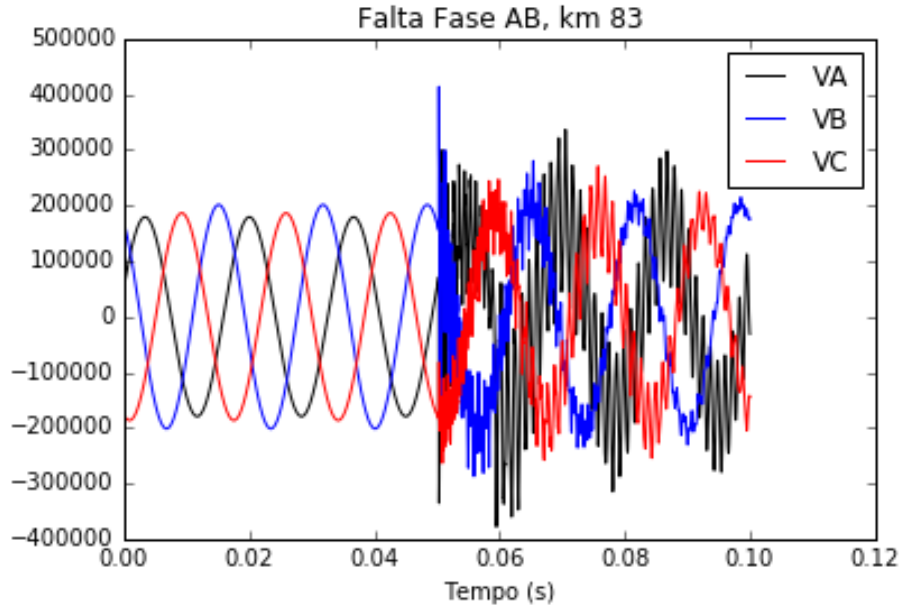


Figura 4.3: Tensões quando simulada uma falta nas fases AB, km 83.

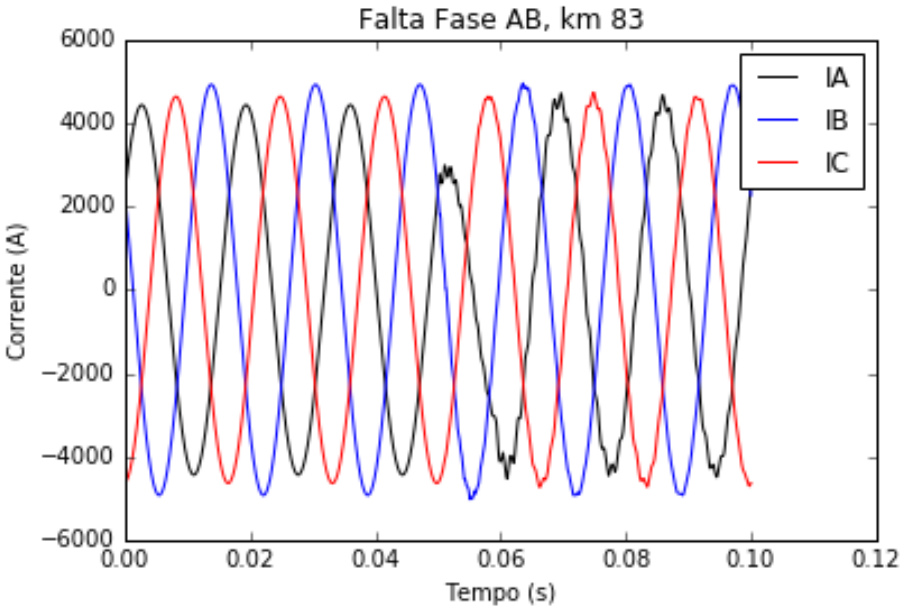


Figura 4.4: Correntes quando simulada uma falta nas fases AB, km 83.

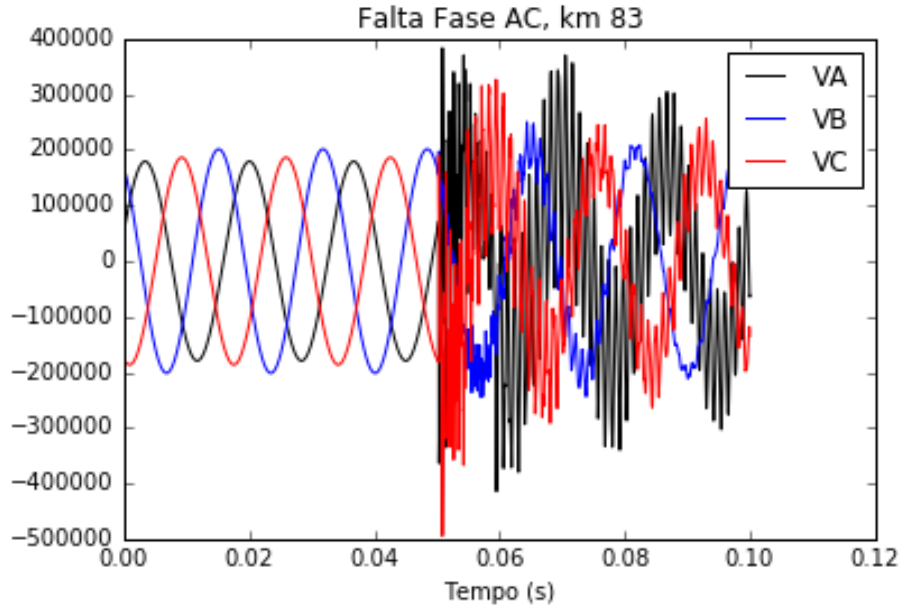


Figura 4.5: Tensões quando simulada uma falta nas fases AC, km 83.

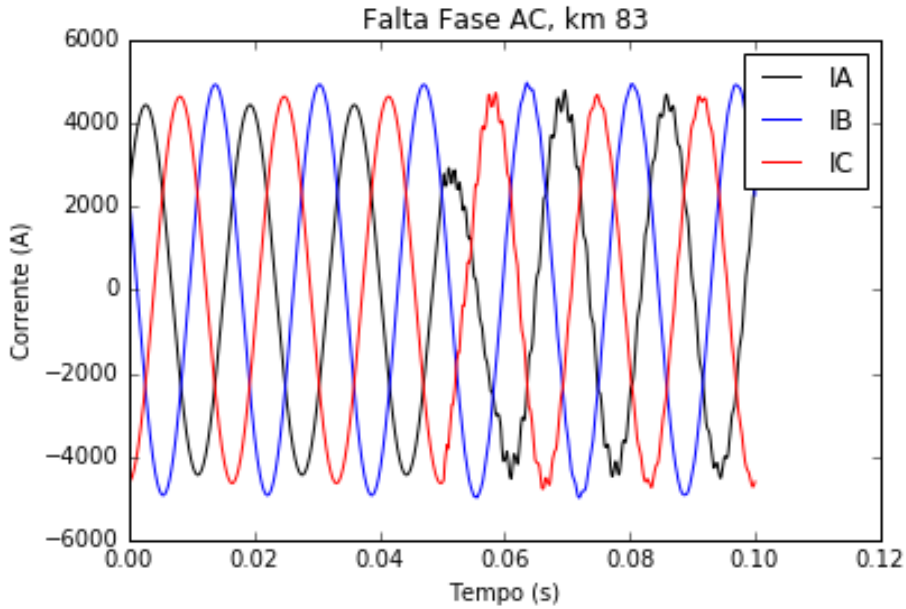


Figura 4.6: Correntes quando simulada uma falta nas fases AC, km 83.

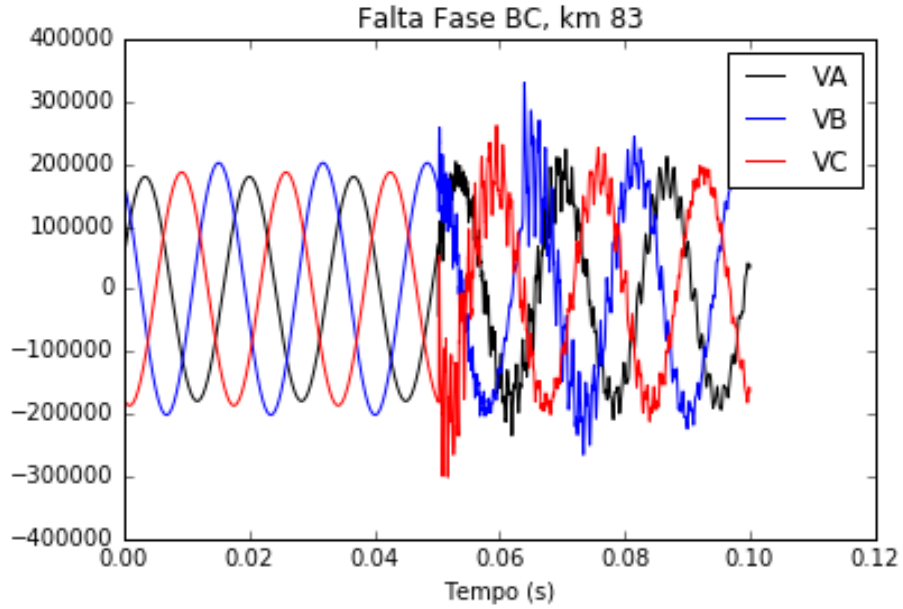


Figura 4.7: Tensões quando simulada uma falta nas fases BC, km 83.

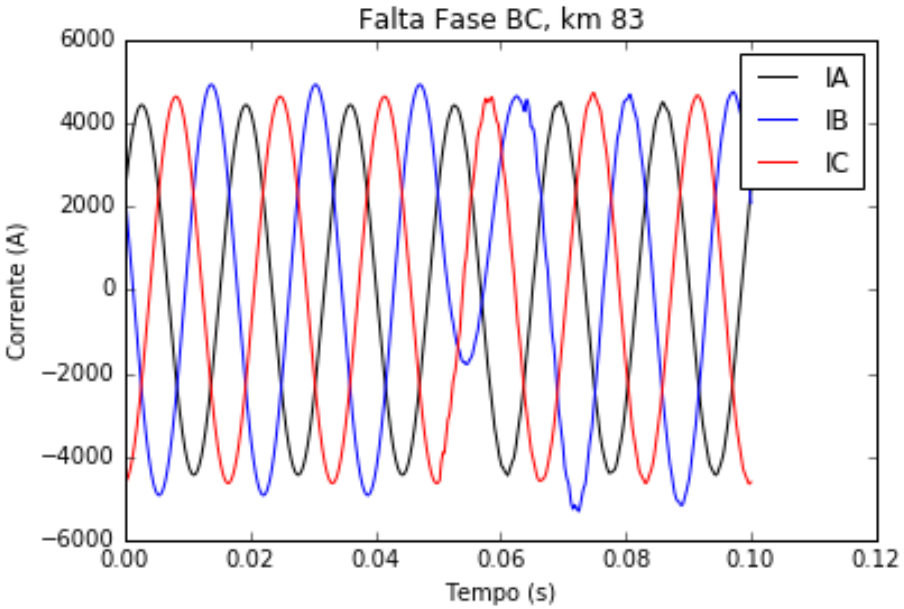


Figura 4.8: Correntes quando simulada uma falta nas fases BC, km 83.

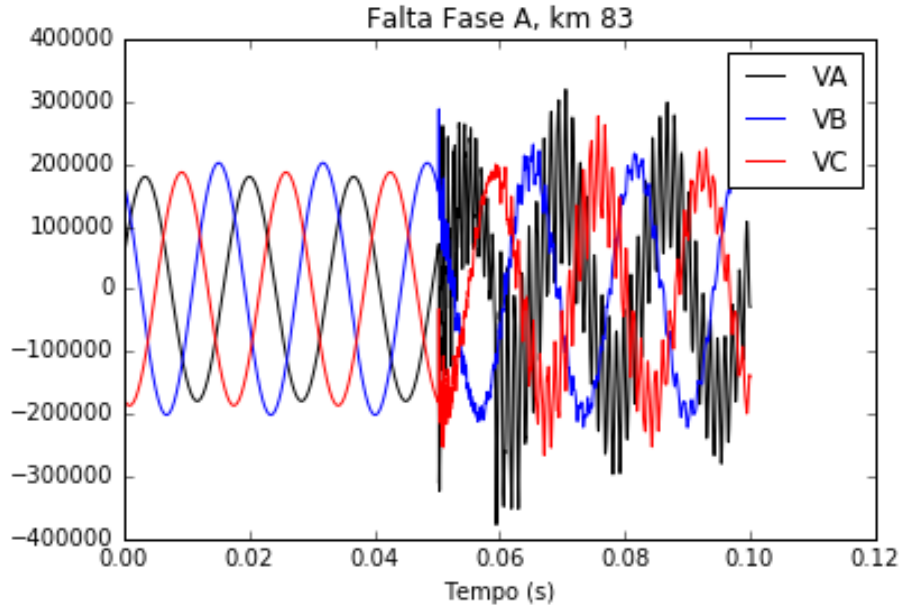


Figura 4.9: Tensões quando simulada uma falta na fase A, km 83.

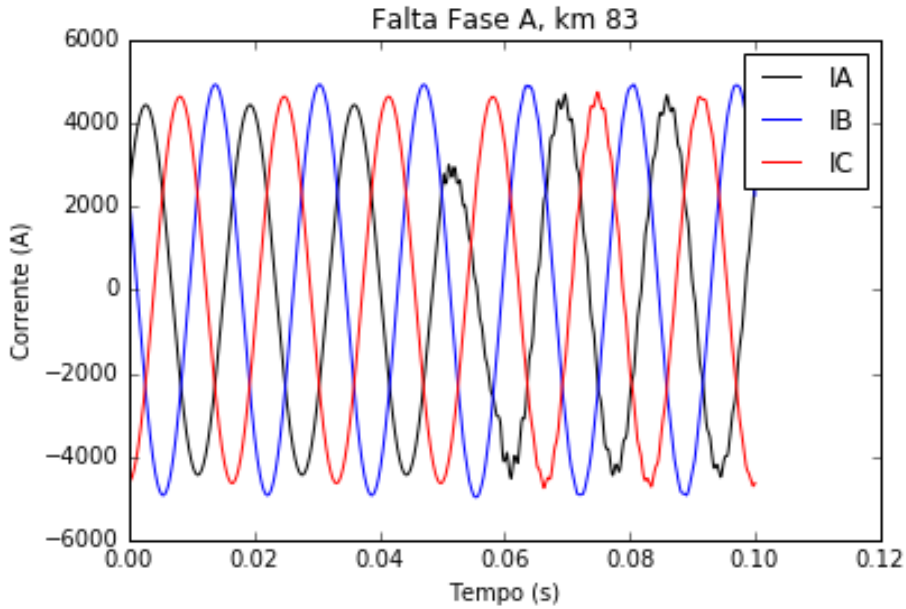


Figura 4.10: Correntes quando simulada uma falta na fase A, km 83.

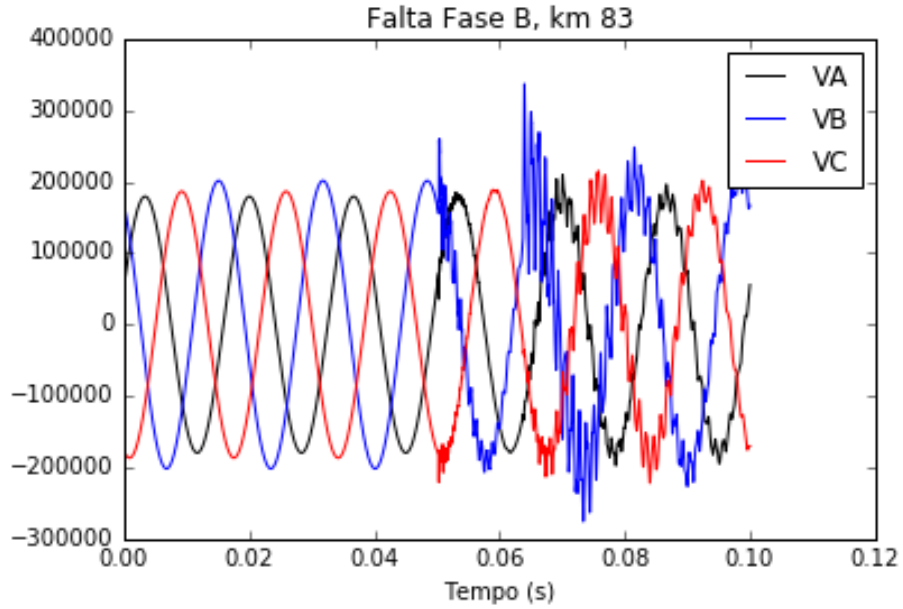


Figura 4.11: Tensões quando simulada uma falta na fase B, km 83.

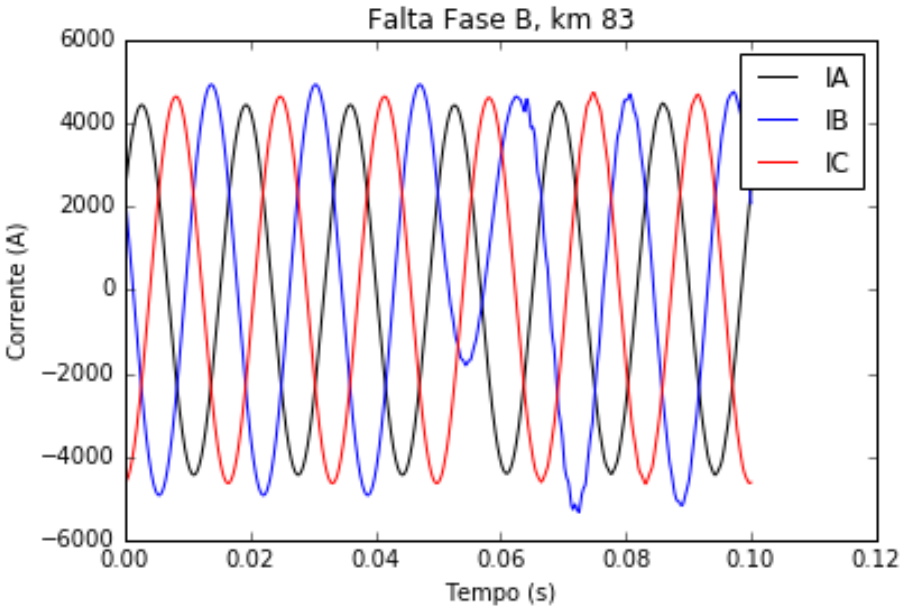


Figura 4.12: Correntes quando simulada uma falta na fase B, km 83.

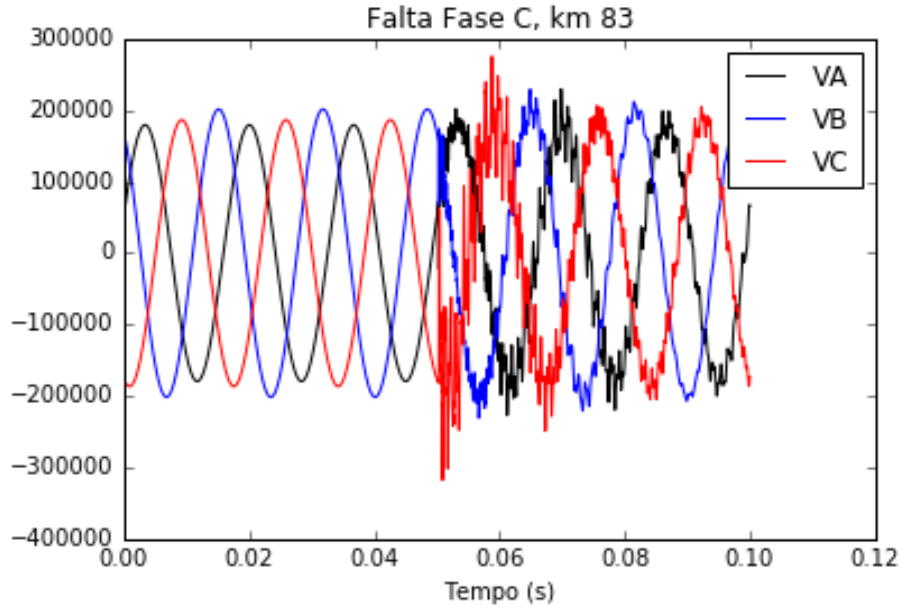


Figura 4.13: Tensões quando simulada uma falta na fase C, km 83.

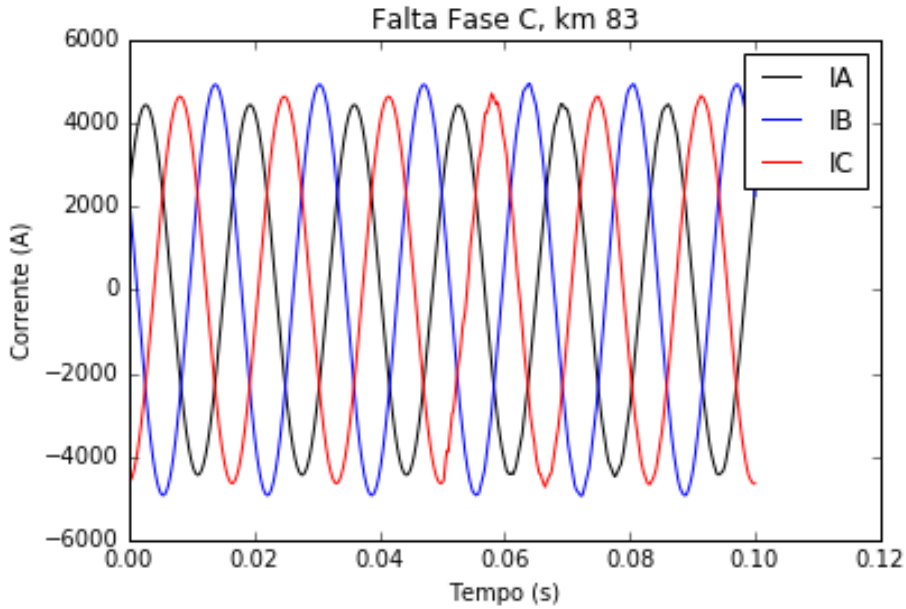


Figura 4.14: Correntes quando simulada uma falta na fase C, km 83.

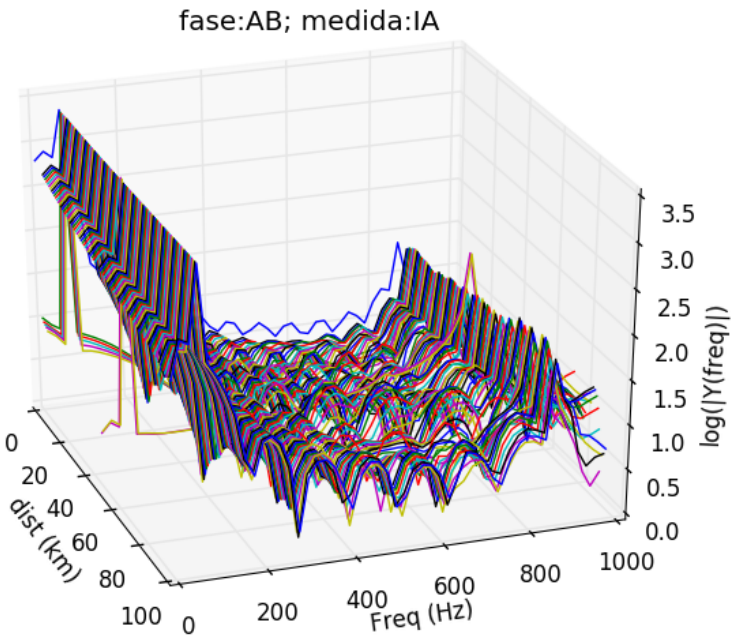


Figura 4.15: Frequências presentes na corrente da fase A. Fases faltosas: AB.

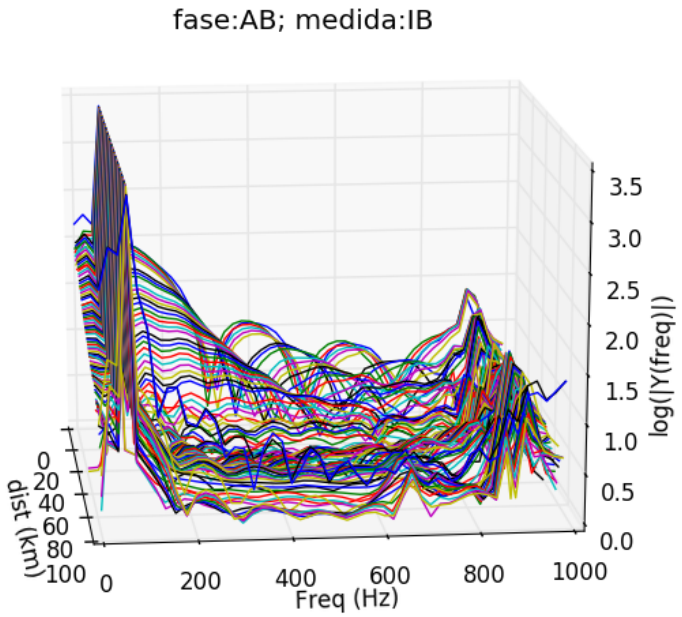


Figura 4.16: Frequências presentes na corrente da fase B. Fases faltosas: AB.

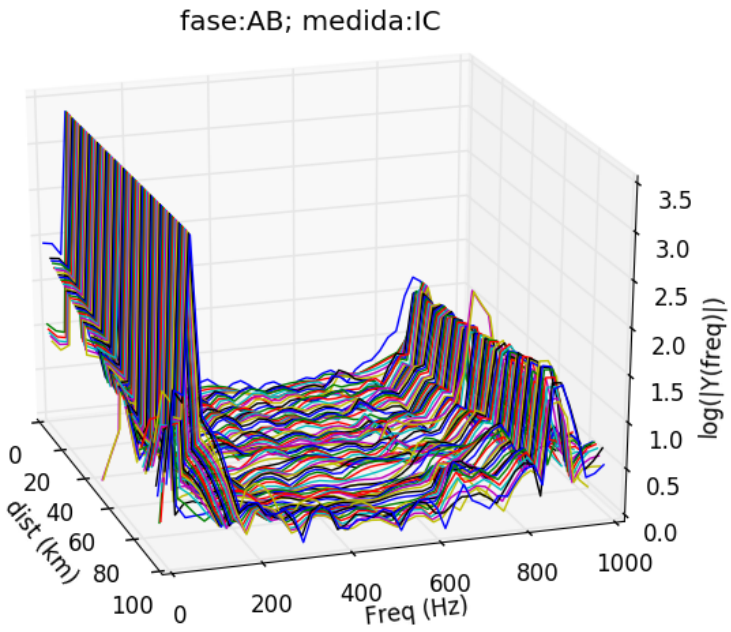


Figura 4.17: Frequências presentes na corrente da fase C. Fases faltosas: AB.

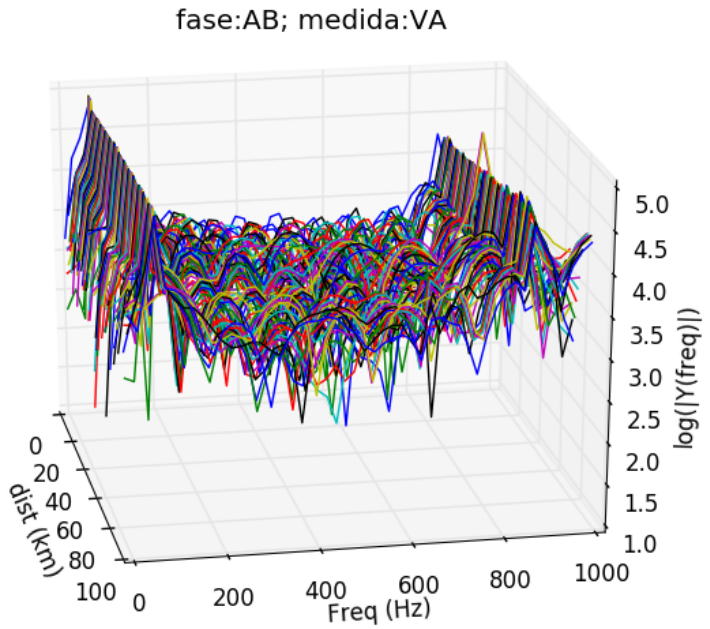


Figura 4.18: Frequências presentes na tensão da fase A. Fases faltosas: AB.

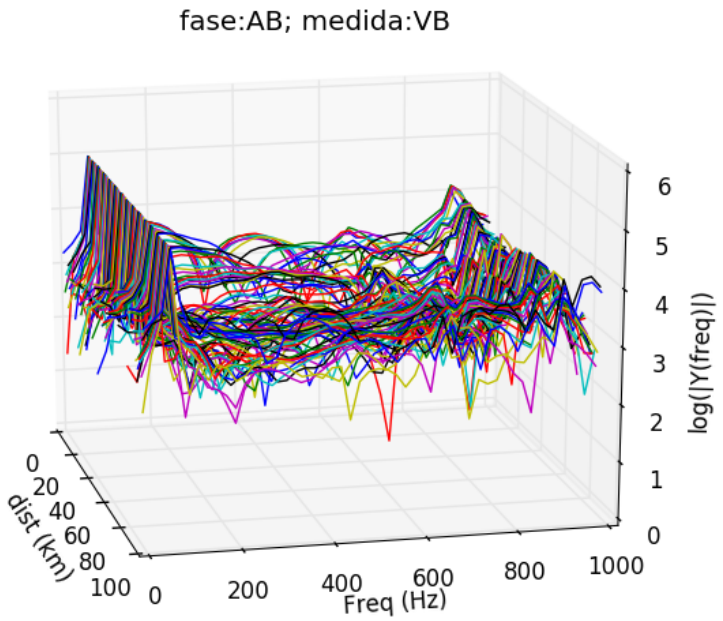


Figura 4.19: Frequências presentes na tensão da fase B. Fases faltosas: AB.

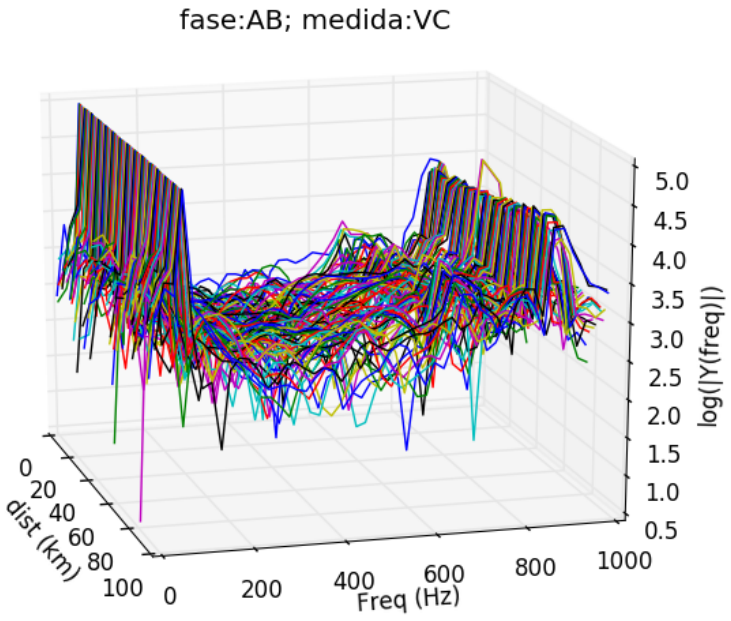


Figura 4.20: Frequências presentes na tensão da fase C. Fases faltosas: AB.

Tabela 4.1: Resultados previsto erroneamente e valor esperado correspondente.

previsto	correto
FFTB42	FFTB43
FFTB46	FFTB47
FFTAB50	FFTBC50
FFTAB30	FFTAC30
FFTA48	FFTAB48
FFTABC56	FFTAC56
FFTB52	FFTB53
FFTAC80	FFTABC80
FFTB58	FFTB57
FFTAC58	FFTABC58
FFTAB60	FFTAC60
FFTAC10	FFTA10
FFTABC57	FFTAC57
FFTC49	FFTC48
FFTB47	FFTB48
FFTAB70	FFTBC70
FFTA51	FFTAB51
FFTC64	FFTBC64
FFTB41	FFTB42
FFTBC90	FFTB90
FFTAB20	FFTA20
FFTABC40	FFTA40
FFTBC65	FFTC65
FFTAB49	FFTA49
FFTC66	FFTBC66

Tabela 4.2: Dados descritivos do erro absoluto da rede neural regressora com *score* 0.9421.

	(km)
Média	3.96298
Desvio padrão	5.68621
Mediana	2.01826
Máximo	35.00962
Mínimo	0.01242

Tabela 4.3: Dados descritivos do erro absoluto da rede neural regressora com *score* 0.8700.

	(km)
Média	4.55468
Desvio padrão	8.88146
Mediana	1.98168
Máximo	60.08489
Mínimo	0.01073

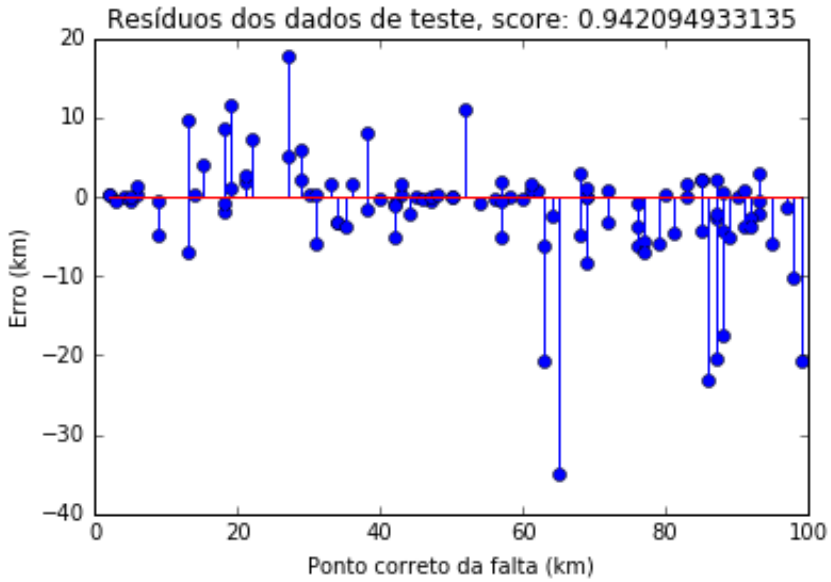


Figura 4.21: Gráfico dos resíduos para rede neural regressora com *score* 0.9421.

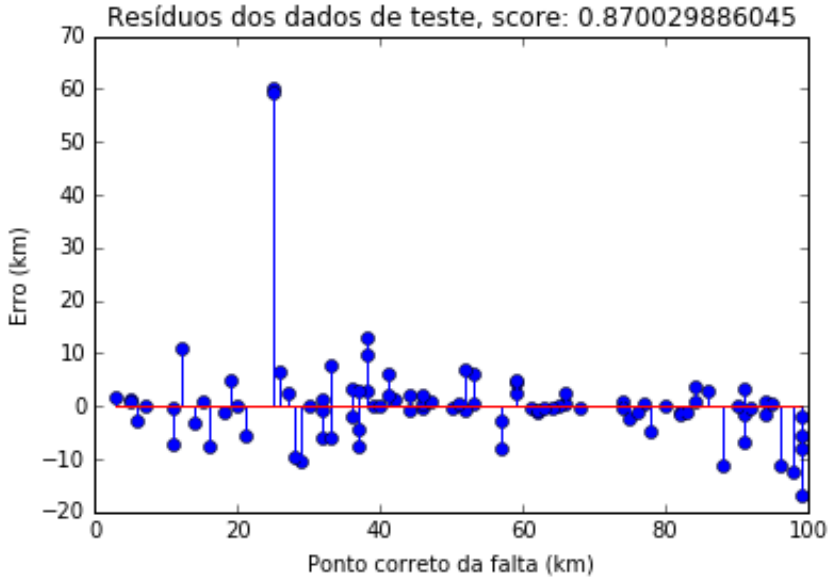


Figura 4.22: Gráfico dos resíduos para rede neural regressora com *score* 0.8700.

Tabela 4.4: Dados descritivos do erro absoluto da rede neural regressora com *score* 0.5702.

	(km)
Média	14.15198
Desvio padrão	12.18204
Mediana	9.78856
Máximo	45.42522
Mínimo	0.21980

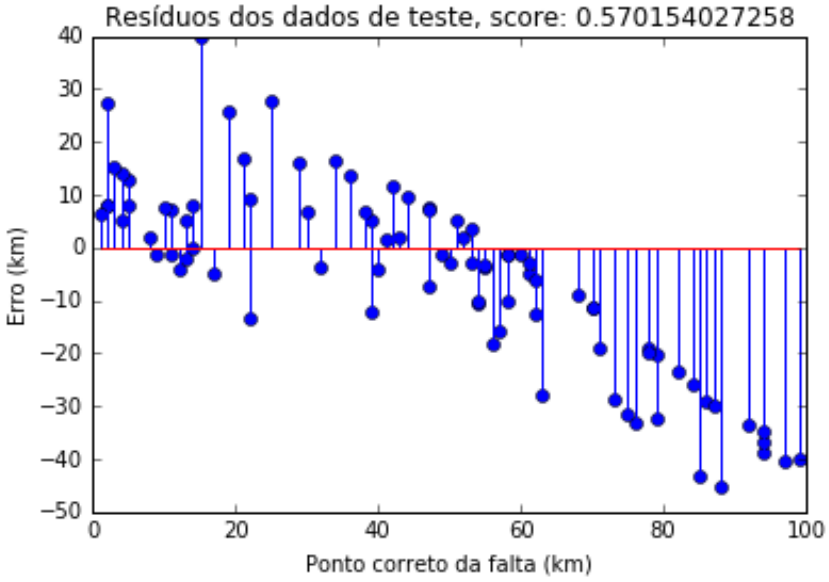


Figura 4.23: Gráfico dos resíduos para rede neural regressora com *score* 0.5702.

5 Conclusão

Todos os arquivos com os códigos presentes nos apêndices estão disponíveis no repositório pessoal do autor no GitHub: `https://github.com/pedrohnv/MonografiaVieiraUBM`. No mesmo repositório estão os arquivos básicos criados no ATPDraw, dos quais os outros podem ser gerados com os códigos.

5.1 Simulação

O esperado para as tensões e correntes durante as faltas é que, nas fases envolvidas na falta, a tensão diminuísse e a corrente aumentasse, ambas de forma significativa. E estes novos valores deveriam se sustentar até o fim da simulação, pois a falta não é extinta.

Os resultados das tensões e correntes das simulações, demonstrados na seção 4.1, não estão como o esperado. O contrário se observa em alguns casos: a corrente diminui. Não se sabe se houve um erro na simulação por instabilidade numérica, ou se os valores usados na simulação não condizem com os de um sistema real. Nenhum esforço foi feito para garantir que tais valores fossem plausíveis. Eles foram escolhidos de forma arbitrária.

Recomenda-se para trabalhos futuros garantir que a simulação tenha valores condizentes aos de um sistema real.

5.1.1 Circuito base pra simulação com EMTP-ATP

O circuito base feito no ATPDraw (figura 3.6) serviu ao seu propósito de permitir simulações sistemáticas das faltas em todo o comprimento da linha.

A criação das seções de linha de transmissão (de 1 a 10 km) foi feita de forma manual. Trabalhos futuros se beneficiariam se um código de computador fosse criado para que um número arbitrário de seções fosse criada de forma automática (e.g. 1 a 200 km).

5.1.2 Situações simuladas

Nenhuma das simulações mudou o ângulo de incidência da falta, i.e., as faltas sempre acontecem no mesmo instante de tempo, com os mesmos valores de tensão e corrente. A simulação sistemática com outros ângulos de incidência deve ser feita em trabalhos futuros para que a rede neural tenha um treinamento melhor. Uma forma de se fazer isso é variando o instante do tempo no qual as chaves que conectam a linha à terra fecham.

5.2 Rede Neural

É necessário adquirir dados de uma linha real durante curto-circuitos para colocar a rede neural à prova (checar se não houve *overfitting*). O aprimoramento dos dados usados para treino, i.e. simular faltas ocorrendo em diferentes instantes de tempo, em diferentes situações de carga, com outras resistências de falta e pontos de ocorrência desta, é deixado para trabalhos futuros.

Também para trabalhos futuros, é recomendado que treine redes neurais com outras configurações para comparar os resultados entre elas.

Nenhum esforço foi feito para que a rede neural saiba diferenciar entre a operação normal da linha e uma situação de falta. Esta parte do treinamento é algo a ser feito em outra pesquisa.

5.2.1 Classificadora

A rede neural classificadora treinada apresentou um bom resultado no teste; que é esperado, pois os dados usados para teste foram os mesmos usados para o treino dela.

Como a rede treinada é do tipo classificadora, ela foi capaz de, além de localizar o ponto de falta, classificá-la quanto às fases envolvidas.

5.2.2 Regressora

À primeira vista, comparando os gráficos do erro absoluto nas figuras 4.21 e 4.22, tem-se a impressão que a rede neural com *score* de 87.00 % teve um desempenho melhor quanto aos erro absolutos. Porém, analisando as tabelas 4.2 e 4.3, percebe-se que a rede com *score* 94.21 %, além de uma taxa de acerto maior, apresentou dados descritivos do erro melhores (média, desvio padrão, máximo).

A impressão citada acima ocorre devido à diferença de escala dos eixos verticais das figuras 4.21 e 4.22. Devido ao exposto anteriormente, pode-se concluir que a rede com *score* maior teve o melhor desempenho. Contudo, ela apresentou média próxima a 4 km, o que é considerado alto em uma situação prática.

Bibliografia

Adami, J. F. (2008). “Detecção e Identificação de Arcos de Contorno em Cadeias de Isoladores de Linhas de Transmissão Utilizando Técnicas de Processamento de Sinais.” Tese de doutorado. Universidade Federal de Itajubá, p. 213.

Andrade, L. de, H. Leite e M. T. P. Leão (2013). *Time-domain distributed parameters transmission line model for transient analysis*. Em: *Progress In Electromagnetics Research B* 53.April, pp. 25–46.

Branin Jr., F. H. (1967). *Computer methods of network analysis*. Em: *Proceedings of the IEEE* 55.11, pp. 1787–1801. ISSN: 0018-9219. DOI: 10.1109/PROC.1967.6010.

Caballero, P. T., E. C. M. Costa e S. Kurokawa (2014). *Fitting the frequency-dependent parameters in the Bergeron line model*. Em: *Electric Power Systems Research* 117, pp. 14–20. ISSN: 03787796. DOI: 10.1016/j.epsr.2014.07.023. URL: http://dx.doi.org/10.1016/j.epsr.2014.07.023.

Dalcin, L. et al. (2010). *Cython: The Best of Both Worlds*. Em: *Computing in Science and Engineering* 13.undefined, pp. 31–39. ISSN: 1521-9615. DOI: doi.ieeecomputersociety.org/10.1109/MCSE.2010.118.

Dallbello, A. C. et al. (2007). *Análise de Sinais Provocados por Defeitos em Linhas de Transmissão Utilizando Técnicas de Telecomunicação e Processamento de Sinais*. Em: *IV Congresso de inovação tecnológica em energia elétrica*, pp. 1–8. URL: http://www2.aneel.gov.br/biblioteca/citenel2007/.

Dommel, H. (1969). *Digital Computer Solution of Electromagnetic Transients in Single- and Multiphase Networks*. Em: *IEEE Transactions on Power Apparatus and Systems* PAS-88.4, pp. 388–399. ISSN: 0018-9510. DOI: 10.1109/TPAS.1969.292459.

— (1986). *Electromagnetic Transients Program Theory Book*. Bonneville Power Administration, p. 483.

Hastie, T., R. Tibshirani e J. Friedman (2009). *The Elements of Statistical Learning*. 2nd. Springer Series in Statistics. New York, NY: Springer New York, p. 745. ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7. arXiv: 1010.3003. URL: http://www.springerlink.com/index/10.1007/b94608http://link.springer.com/10.1007/978-0-387-84858-7.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. 2nd. Delhi, India: Pearson Education, p. 823. ISBN: 81-7808-300-0.

Jones, E., T. Oliphant, P. Peterson et al. (2001–). *SciPy: Open source scientific tools for Python*. [Online; accessed 2016-10-16]. URL: http://www.scipy.org/.

Kiusalaas, J. (2005). *Numerical Methods in Engineering with Python*. New York: Cambridge University Press, p. 424. ISBN: 978-0-511-12810-3. URL: www.cambridge.org/9780521852876.

Kurokawa, S. et al. (2007). *Representação de linhas de transmissão por meio de variáveis de estado levando em consideração o efeito da frequencia sobre os parâmetros longitudinais*. Em: *Revista Controle & Automação* 18.3, pp. 337 – 347. ISSN: 01031759.

Marti, J. R. (1982). *Accurate Modeling of Frequency-Dependent Transmission Lines in Electromagnetic Transient Simulations*. Em: *IEEE Transactions on Power Apparatus Systems* PAS-101.1, pp. 147–157. DOI: 10.14288/1.0095571.

Marti, J. R. (1981). "The problem of frequency dependence in transmission line modelling". Tese de doutorado. Vancouver: University of British Columbia, p. 200. DOI: 10.14288/1.0095571. URL: https://open.library.ubc.ca/cIRcle/collections/ubctheses/831/items/1.0095571.

Paz, M. d. A. (2005). "Modelo Reduzido de Linhas de Transmissão para Transitórios Eletromagnéticos - Aplicação de Propriedades Complexas". Tese de doutorado. Universidade Estadual de Campinas, p. 145.

Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python*. Em: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Prandoni, P. e M. Vetterli (2008). *Signal processing for communications*. 1st. Italy: EPFL Press, p. 371. ISBN: 978-2-940222-20-9.

Prikler, L. e H. K. Høidalen (2009). *ATPDraw Users' Manual*. 1.0. 1.

Sampaio, R., E. L. C. Ferreira e A. d. S. Brandão (2006). *Análise e processamento de sinais*. Em: *XXIX CNMAC*. Vol. 22. São Carlos: Sociedade Brasileira de Matemática Aplicada e Computacional, p. 130. ISBN: 8586883263.

Sauer, P. W. e M. A Pai (2006). *Power system dynamics and stability*. Urbana: The University of Illinois at Urbana-Champaign, p. 349. ISBN: 1588746739 9781588746733.

Tavares, M. C. D. (1998). "Modelo de Linha de Transmissao Polifásica utilizando Quase-Modos". Tese de doutorado. Universidade Estadual de Campinas, p. 274.

Walt, S. v. d., S. C. Colbert e G. Varoquaux (2011). *The NumPy Array: A Structure for Efficient Numerical Computation*. Em: *Computing in Science and Engineering* 13.2, pp. 22–30. URL: http://scitation.aip.org/content/aip/journal/cise/13/2/10.1109/MCSE.2011.37.

Zakon, E. (1977). *Mathematical Analysis: Volume II*. Ed. por B. J. Lucier e T. Zakin. 2009ª ed. West Lafayette, Indiana, USA: The Trillia Group, p. 425. ISBN: 978-1-931705-03-8. URL: http://www.trillia.com/zakon-analysisII.html.

Apêndices

A Exemplo de uso da Transformada de Fourier

```
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 11 11:53:45 2016

@author: pedro
"""
import numpy as np
import matplotlib.pyplot as plt


def componentesFrequenciaSinal(y, fs):
    n = len(y) # length of the signal
    k = np.arange(n)
    T = n/fs
    frq = k/T # two sides frequency range
    frq = frq[range(int(n/2))] # one side frequency range

    Y = np.fft.fft(y)/n # fft computing and normalization
    Y = Y[range(int(n/2))]
    return frq, abs(Y)


def plotarSinalTransformado(t, y, fs):
    frq, Y = componentesFrequenciaSinal(y, fs)

    fig, ax = plt.subplots(2, 1)
    ax[0].plot(t,y)
    ax[0].set_xlabel('Time')
    ax[0].set_ylabel('Amplitude')
    ax[1].plot(frq, Y, 'r') # plotting the spectrum
    ax[1].set_xlabel('Freq␣(Hz)')
    ax[1].set_ylabel('|Y(freq)|')
    plt.xlim([-100,1100])
    plt.show()


t = np.linspace(0,0.1,300)
fs = 1/(t[1] - t[0]) # freq. amostragem

y1 = np.sin(2 * np.pi * 60 * t)
y2 = np.sin(2 * np.pi * 300 * t)

plotarSinalTransformado(t, y1, fs)
plotarSinalTransformado(t, y2, fs)
plotarSinalTransformado(t, y1+y2, fs)
```

B Funções para criar e simular arquivos ATP-EMTP

```
# -*- coding: utf-8 -*-
"""
Funções auxilio TCC.

Ambiente: Windows 10, 64-bit

@autor: Pedro Henrique Nascimento Vieira, 2016
"""
import re
import numpy as np
import scipy as sp

def criarBase(base, novo):
    '''
    Copiar o arquivo base e "setar" todas as seções da linha de transmissão
    para seções de 10 km.
    '''
    f_n = open(novo, 'x')

    with open(base, 'r') as f:
        for linha in f:
            for n in range(1,11):
                pt = re.compile("lttcc_L"+str(n)+".lib")
                linha = re.sub(pt, "lttcc_L10.lib", linha)

            f_n.write(linha)

    f_n.flush()
    f_n.close()


def simular(arquivo):
    from subprocess import call
    '''Arquivo refere-se ao caminho no computador que ele se encontra
    (junto com a extensão .atp).'''

    comando = ('C:\\ATPdraw\\tpbig1 '+ arquivo + ' > ' + arquivo[:-4] + 'Res.lis')

    # envia comandos para o command prompt
    call("COPY C:\\ATPdraw\\STARTUP STARTUP", shell = True)
    call(comando, shell = True)


def extrairResultados(arq):
    '''Lê um arquivo .lis e extrai o output colocando em outro arquivo.'''
    f_n = open(arq + "resultado.txt", 'x')
    f_v = open(arq, 'r')

    procurandoInicio = True
    n = 0

    # procurar inicio dos resultados no arquivo;
    # ler resultados até encontrar linha com "Suspended simulation"
    for linha in f_v:
        if procurandoInicio and "EMTP output variables follow." in linha:
            procurandoInicio = False

        elif not procurandoInicio:
            if 'Step' in linha and 'Time' in linha:
```

```
            # split depois join na string para sumir com os espaços extras
            f_n.write("␣".join( linha.split() ) + "\n")

        elif n > 6:
            if ("Suspended␣simulation" in linha) or ("Final␣time␣step" in linha):
                break

            elif "switch" in linha:
                pass

            else:
                res = linha.split()
                try:
                    res.remove("SPY:")
                except ValueError:
                    pass

                f_n.write("␣".join( res ) + "\n")

        else:
            n += 1

n = 0
# pular as linhas de suspensão até encontrar o final
for linha in f_v:
    if n > 6:
        if ("Suspended␣simulation" in linha):
            n = 0

        elif ("Final␣time␣step" in linha):
            break

        elif "switch" in linha:
                pass

        else:
            res = linha.split()
            try:
                res.remove("SPY:")
            except ValueError:
                pass

            f_n.write("␣".join( res ) + "\n")

    else:
        n += 1

n = 0
# pegar último resultado, uma linha única que vem depois do final
for linha in f_v:
    if n > 0:
        res = linha.split()

        try:
            res.remove("SPY:")
        except ValueError:
            pass

        f_n.write("␣".join( res ) + "\n")
        break

    else:
        n += 1

f_v.close()
f_n.flush()
```

```
    f_n.close()


def lerResultados(arq, extraido = False):
    '''Lê um arquivo com os resultados extraídos e, com ele, cria um  data frame.'''
    import pandas as pd

    if extraido:
        df = pd.read_csv(arq, sep = "␣")
    else:
        df =  pd.read_csv(arq + "resultado.txt", sep = "␣")

    nomes = list(df)[:2]

    v = ['XL001', 'V']
    # identificar as colunas a partir do nome das barras.
    # não há garantia que o ATP colocará os valores na mesma ordem sempre.
    for nome in list(df)[2:5]:
        if nome == v[0] + 'A':
            nome = v[1] + 'A'

        elif nome == v[0] + 'B':
            nome = v[1] + 'B'

        elif nome == v[0] + 'C':
            nome = v[1] + 'C'

        else:
            raise ValueError("Identificação␣inesperada␣para␣a␣barra")

        nomes += [nome]

    i = ['XL001', 'I']
    for nome in list(df)[5:8]:
        if nome == i[0] + 'A.1':
            nome = i[1] + 'A'

        elif nome == i[0] + 'B.1':
            nome = i[1] + 'B'

        elif nome == i[0] + 'C.1':
            nome = i[1] + 'C'

        else:
            raise ValueError("Identificação␣inesperada␣para␣a␣barra")

        nomes += [nome]

    df.columns = nomes

    return df


def definirBarras(km):
    '''Função auxiliar. Determina a barra à montante e à jusante do km
    especificado.'''
    ponto = km/100
    x = 0.10

    # determinar a porcentagem do ponto
    while ponto > x:
        x += 0.10

    # dar nome às barras a partir da porcentagem
    if x < 0.80:
```

```
        barra1 = "XL00" + str( round(x * 10) )
        barra2 = "XL00" + str( round((x + 0.10) * 10) )

    elif x < 0.9:
        barra1 = "XL009"
        barra2 = "XL010"

    else:
        barra1 = "XL010"
        barra2 = "XL011"

    return barra1, barra2


def definirSecoes(km):
    '''Função auxiliar. Determina o comprimento da primeira e segunda seção no
    qual dividir uma seção de 10 km.'''
    secao1 = km

    # extrai a unidade de 'km' de forma a ter uma seção menor que 10 km
    while secao1 > 10:
        # se o 'km' for grande, este loop pode demorar muito
        # considerar o uso de regex...
        secao1 -= 10

    secao2 = int(10 - secao1)
    secao1 = int(secao1)

    return secao1, secao2


def mudarResistenciaFalta(linhaTexto, fase, Rnovo, Rvelho = 0.5):
    '''Função auxiliar, muda o valor do resistor de falta.'''
    dicionario = {'A':'1', 'B':'2', 'C':'3'}

    if ("XF000" + dicionario.get(fase)) in linhaTexto and ("XSWT0"  +
                                        dicionario.get(fase)) not in linhaTexto:
        return re.sub(".5", str( float(Rnovo) ), linhaTexto)
    else:
        return linhaTexto


def replicarMudarResistencia(R, fase, titulo,
                    base = "C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp"):
    '''
    Cria um arquivo novo, mudando a resistência de falta das fases especificadas,
    baseado em um arquivo pré-existente.

    Fases é uma string com o nome das fases (ABC) pra colocar a falta; a ordem
    não importa.
    '''

    nomeNovo = "C:\\ATPdraw\\ATP\\TCC\\" + titulo + ".atp"
    novo = open(nomeNovo, 'x')
    velho = open(base, 'r')

    for linha in velho:
        lin = mudarResistenciaFalta(linha, R, fase)
        novo.write(lin)


    novo.flush()
    novo.close()
```

```
def inserirFaltaFaseTerra(linhasTexto, barra1, barra2, secao1, secao2,
                          faseA, faseB, faseC):
    '''Função auxiliar. Substitui o nome das barras de forma a fazer a falta
    fase terra.'''
    lin1 = ''
    lin2 = ''

    dicionario = {'A':'1', 'B':'2', 'C':'3'}

    # função auxiliar para trocar o nome das barras das linhas de texto
    def foo(substituir, fase, linha1, linha2):
        z = dicionario.get(fase)

        if substituir:
            linha1 = re.sub(barra2 + fase, "XSWT0" + z, linha1)
            linha2 = re.sub(barra1 + fase, "XSWT0" + z, linha2)
        else:
            linha1 = re.sub(barra2 + fase, "XLINT" + fase, linha1)
            linha2 = re.sub(barra1 + fase, "XLINT" + fase, linha2)

        return linha1, linha2


    repl1 = "lttcc_L" + str(secao1) + ".lib"
    repl2 = "lttcc_L" + str(secao2) + ".lib"

    for l in linhasTexto:
        linha1 = re.sub("lttcc_L10.lib", repl1, l)
        linha2 = re.sub("lttcc_L10.lib", repl2, l)

        linha1, linha2 = foo(faseA, "A", linha1, linha2)
        linha1, linha2 = foo(faseB, "B", linha1, linha2)
        linha1, linha2 = foo(faseC, "C", linha1, linha2)

        lin1 += linha1
        lin2 += linha2

    return lin1, lin2


def faltaFaseTerra10(km, titulo, fases = 'ABC',
                     base = "C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp"):

    barra = definirBarras(km)[1]

    nomeNovo = "C:\\ATPdraw\\ATP\\TCC\\" + titulo + str(km) + ".atp"
    novo = open(nomeNovo, 'x')

    with open(base) as arq:
        for linha in arq:
            if "XSWT0" in linha:
                if 'A' in fases:
                    lin = re.sub("XSWT01", barra + 'A', linha)

                elif 'B' in fases:
                    lin = re.sub("XSWT02", barra + 'B', linha)

                elif 'C' in fases:
                    lin = re.sub("XSWT03", barra + 'C', linha)

            else:
                lin = linha

            novo.write(lin)
```

```
    novo.flush()
    novo.close()


def replicarFaltaTerra(km, titulo, fases = 'ABC',
                       base = "C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp"):
    '''
    Cria um arquivo novo com uma falta fase-terra com as fases especificadas,
    baseado em um arquivo pré-existente.

    Fases é uma string com o nome das fases (ABC) pra colocar a falta; a ordem
    não importa.
    '''
    if km % 10 == 0:
        return faltaFaseTerra10(km, titulo, fases = 'ABC',
                       base = "C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp")

    barra1, barra2 = definirBarras(km)

    secao1, secao2 = definirSecoes(km)

    def faltaNaFase(f):
        if f in fases:
            return True
        else:
            return False

    falta = [None, None, None]
    t = 0

    for f in 'ABC':
        falta[t] = faltaNaFase(f)
        t+=1

    with open(base, 'r') as arq:
        # possível leak de memoria se o arquivo for grande
        todasLinhas = arq.readlines()

    nomeNovo = "C:\\ATPdraw\\ATP\\TCC\\" + titulo + str(km) + ".atp"
    novo = open(nomeNovo, 'x')

    pular = False

    for n in range( len(todasLinhas) ):
        linha = todasLinhas[n]

        if pular:
            # pula essa linha de texto
            pular = False

        elif ("$INCLUDE" in linha) and (barra1 in linha) and (barra2 in linha):

            linhasTexto = todasLinhas[n : n+2]

            # cria duas linhas de texto para inserir no arquivo novo
            lin1, lin2 = inserirFaltaFaseTerra(linhasTexto, barra1, barra2,
                                secao1, secao2, falta[0], falta[1], falta[2])

            pular = True

            novo.write(lin1)
            novo.write(lin2)
```

```
        else:
            novo.write(linha)


    novo.flush()
    novo.close()


def replicar99(titulo, fases = 'ABC',
                    base = "C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp"):
    '''
    Cria falta fase-terra em todos os pontos (1 a 99) da linha usando o arquivo
    base especificado.
    '''
    for km in range(1,100):
        replicarFaltaTerra(km, titulo, fases, base)


def amostrarMedidor(dados, taxaAmostragem = 500, periodoDados = 1e-6):
    '''

    Simula um medidor digital extraindo valores de dados de entrada.

    ____________
    Parâmetros:
        dados, dataframe de dados do qual extrair os valores, considerando
    que tenha um período igualmente espaçado entre valores;
        periodoDados = 1e-6, em segundos;
        taxaAmostragem = 500, em Hz

    Se (1/periodoDados/taxaAmostragem) não resultar em um valor inteiro,
    arredondamento é feito; pode levar a erros.

    __________
    Retorna:
        os valores capturados pelo medidor
    '''
    valoresAmostrados = dados.loc[dados['Step'] == 1]
    intervalo = round(1/periodoDados/taxaAmostragem)
    n = 0

    for valor in dados.Step:
        if n == intervalo:
            n = 1
            valoresAmostrados = valoresAmostrados.append(
                                            dados.loc[dados['Step'] == valor])

        else:
            n += 1

    return valoresAmostrados


def pasta(f): return "C:\\ATPdraw\\ATP\\TCC\\FFT" + f + "0.5"

def arquivo(f, km): return pasta(f) + "\\FFT" + f + str(km) + "Res.lis"

def arquivoResultado(f, km): return arquivo(f, km) + "resultado.txt"

combinacoesFases = ['ABC', 'AB', 'AC', 'BC', 'A', 'B', 'C']

def extrairTodos():
    for fase in combinacoesFases:
```

```
            for km in range(1,100):
                extrairResultados(arquivo(fase, km))

def lerTodosArquivos(extraidos = True):
    '''apos arquivos organizados manualmente;'''
    if not extraidos: #para não re-extrair
        extrairTodos()

    dados = {}
    for fase in combinacoesFases:
        for km in range(1,100):
            dados.update({"FFT" + fase + str(km) : lerResultados(
                                                arquivoResultado(fase, km), True)})

    return dados


def lerTodosArquivosFourier(extraidos=True):
    if not extraidos: #para não re-extrair
        extrairTodos()

    medidas = ['VA', 'VB', 'VC', 'IA', 'IB', 'IC']
    dados = {}
    for fase in combinacoesFases:
        for km in range(1,100):
            valores = lerResultados(arquivoResultado(fase, km), True)
            valores = amostrarMedidor(valores)
            valoresFourier = np.array([])
            for v in medidas:
                valoresFourier = np.hstack((valoresFourier, sp.fft(valores.get(v))))

            dados.update({"FFT" + fase + str(km) : valoresFourier})

    return dados #va,vb,vc,ia,ib,ic
```

C Código de execução

```
# -*- coding: utf-8 -*-
"""
Codigo procedual executado para fazer o TCC com o EMTP-ATP. Ele é
encarregado de criar e executar as simulações.

Ambiente: Windows 10, 64-bit

@autor: Pedro Henrique Nascimento Vieira, 2016
"""
import funcoesATP as fatp

# compilou-se (criou) o arquivo prototipo.atp, movidos para a pasta Base.
# a partir dele, criado o novo arquivo base
try:
    fatp.criarBase(base = "C:\\ATPdraw\\ATP\\TCC\\Base\\prototipo.atp",
            novo = "C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp")

    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp")
    fatp.extrairResultados("C:\\ATPdraw\\ATP\\TCC\\Base\\cktBaseRes.lis")
except FileExistsError:
    pass

base = "C:\\ATPdraw\\ATP\\TCC\\Base\\cktBase.atp"

# gerar arquivos com falta fase terra.
fatp.replicar99("FFTA", fases = 'A', base = base)
# Simular cada um dos arquivos para cirar os arquivos .lis
for km in range(1,100):
    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\FFTA" + str(km)+ ".atp")


# A cada simulação os arquivos foram movidos manualmente para pastas separadas
fatp.replicar99("FFTB", fases = 'B', base = base)
for km in range(1,100):
    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\FFTB" + str(km)+ ".atp")


fatp.replicar99("FFTC", fases = 'C', base = base)
for km in range(1,100):
    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\FFTC" + str(km)+ ".atp")


fatp.replicar99("FFTAB", fases = 'AB', base = base)
for km in range(1,100):
    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\FFTAB" + str(km)+ ".atp")


fatp.replicar99("FFTAC", fases = 'AC', base = base)
for km in range(1,100):
    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\FFTAC" + str(km)+ ".atp")


fatp.replicar99("FFTBC", fases = 'BC', base = base)
for km in range(1,100):
    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\FFTBC" + str(km)+ ".atp")


fatp.replicar99("FFTABC", fases = 'ABC', base = base)
for km in range(1,100):
    fatp.simular("C:\\ATPdraw\\ATP\\TCC\\FFTABC" + str(km)+ ".atp")
```

D Análise exploratória da simulação

```
# -*- coding: utf-8 -*-
"""
Análise exploratoria das simulações

@author: pedro
"""
import funcoesATP as fatp
import matplotlib.pyplot as plt
import random

dados = fatp.lerTodosArquivos()

combinacoesFases = ['ABC', 'AB', 'AC', 'BC', 'A', 'B', 'C']

pastaSalvarFiguras = ("C:\\Users\\pedro\\Documents\\Estudos\\Pesquisas\\Monografia\\"
                      "figuras\\analiseExploratoria")

def detalharSalvarGrafico(fasesFaltosas, km, titulo=1):
    plt.title("Falta Fase " + fasesFaltosas + ", km " + str(km))
    plt.xlabel("Tempo (s)")

    if titulo == 1:
        plt.ylabel("Tensão (V)")
        m = "V"
    else:
        plt.ylabel("Corrente (A)")
        m = "I"

    plt.legend([m+"A", m+"B", m+"C"])

    plt.savefig(pastaSalvarFiguras + "\\FFT" + fasesFaltosas + str(km) +
                "-" + m + ".png")
    plt.show()


def plotTensoesCorrentes(fasesFaltosas, kms = [20,50,80]):
    for x in kms:
        plt.plot(dados.get("FFT" + fasesFaltosas + str(x)).Time,
                 dados.get("FFT" + fasesFaltosas + str(x)).VA, 'k')

        plt.plot(dados.get("FFT" + fasesFaltosas + str(x)).Time,
                 dados.get("FFT" + fasesFaltosas + str(x)).VB, 'b')

        plt.plot(dados.get("FFT" + fasesFaltosas + str(x)).Time,
                 dados.get("FFT" + fasesFaltosas + str(x)).VC, 'r')

        detalharSalvarGrafico(fasesFaltosas, x)


        plt.plot(dados.get("FFT" + fasesFaltosas + str(x)).Time,
                 dados.get("FFT" + fasesFaltosas + str(x)).IA, 'k')

        plt.plot(dados.get("FFT" + fasesFaltosas + str(x)).Time,
                 dados.get("FFT" + fasesFaltosas + str(x)).IB, 'b')

        plt.plot(dados.get("FFT" + fasesFaltosas + str(x)).Time,
                 dados.get("FFT" + fasesFaltosas + str(x)).IC, 'r')

        detalharSalvarGrafico(fasesFaltosas, x, 0)


random.seed(4564841458)
```

```
km = [int(random.random()*100)]

for f in combinacoesFases:
    plotTensoesCorrentes(f, km)
```

E Código para treinar a rede neural

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 17 10:55:39 2016

Codigo para treinar a rede neural com os resultados simulados no ATP.

@author: Pedro Henrique Nascimento Vieira, 2016
"""
import matplotlib.pyplot as plt
import re
import random
import funcoesATP as fatp
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier, MLPRegressor
import warnings
warnings.filterwarnings('ignore') # many deprecation warnings

# shape = (n_amostras, n_features)
dados = fatp.lerTodosArquivosFourier()

# formatar dados e padronizar valores para variância 1 e média 0
# valores reais são separados dos imaginários, e concatenados horizontalmente
# depois a normalização é feita
arrayKeys = []
for situacao in dados.keys():
    arrayKeys += [situacao]

    novosValores = np.hstack((dados.get(situacao).real, dados.get(situacao).imag))
    try:
        arrayValores = np.vstack((arrayValores, novosValores))
    except NameError:
        arrayValores = np.array(novosValores)

valoresPadronizados = (arrayValores - arrayValores.mean())/arrayValores.std()

arrayKeys = np.array(arrayKeys)

# rede neural classificadora
cerebro = MLPClassifier(activation='logistic', max_iter=10000)
cerebro.fit(valoresPadronizados, arrayKeys)

# testar valores e quantificar quantos erros e acertos
acertos = 0
resultadosErrados = {} #key: resultado previsto, valor: resultado correto
for n in range(valoresPadronizados.shape[0]):
    if cerebro.predict(valoresPadronizados[n]) == arrayKeys[n]:
        acertos += 1
    else:
        resultadosErrados.update({cerebro.predict(valoresPadronizados[n])[0] :
            arrayKeys[n]})

resultadosErrados = pd.DataFrame.from_dict(resultadosErrados, orient='index')
resultadosErrados.columns = ["previsto/correto"]

print("Total de acertos:", acertos)
print("Acertos:", acertos/valoresPadronizados.shape[0]*100, "%")
resultadosErrados

############ Regressor
arrayPontos = np.array([])
```

```
for s in arrayKeys: #extrair o ponto da falta
    arrayPontos = np.append(arrayPontos, float(re.findall('\d+', s)[0]))


#random.seed(234947238) # Reserva 15.44 % para teste; gera um score de ~94 %
#random.seed(544628) # Reserva 13.56 % para teste; gera um score de ~87 %
random.seed(444) # Reserva 12.41 % para teste; gera um score de ~66 %

amostrasTreino = np.array([])
pontosTreino = np.array([])
amostrasTeste = np.array([])
pontosTeste = np.array([])

a = 0
for n in range(arrayPontos.size):
    if random.gauss(1.05, 1) > 0:
        try:
            amostrasTreino = np.vstack((amostrasTreino, valoresPadronizados[n]))
        except ValueError:
            amostrasTreino = valoresPadronizados[n]

        pontosTreino = np.append(pontosTreino, arrayPontos[n])

    else:
        a += 1
        try:
            amostrasTeste = np.vstack((amostrasTeste, valoresPadronizados[n]))
        except ValueError:
            amostrasTeste = valoresPadronizados[n]

        pontosTeste = np.append(pontosTeste, arrayPontos[n])


print("Dados␣reservados␣para␣Teste:", a/arrayPontos.size*100, "%")

cerebroRegressor = MLPRegressor(activation='logistic', max_iter=5000)
cerebroRegressor.fit(amostrasTreino, pontosTreino)

score = cerebroRegressor.score(amostrasTeste, pontosTeste)
print("Acertos:", score, "%")

# residuals teste
previstos = np.array([])
corretos = np.array([])
for n in range(amostrasTeste.shape[0]):
    previstos = np.append(previstos,
                          cerebroRegressor.predict(amostrasTeste[n]))

    corretos = np.append(corretos, pontosTeste[n])

plt.stem(corretos, previstos-corretos)
plt.ylabel("Erro␣(km)")
plt.xlabel("Ponto␣correto␣da␣falta␣(km)")
plt.title("Resíduos␣dos␣dados␣de␣teste,␣score:␣" + str(score))
plt.show()

print("Média:", np.abs(previstos-corretos).mean())
print("Desvio␣Padrão:", np.abs(previstos-corretos).std())
print("Mediana:", np.median(np.abs(previstos-corretos)))
print("Máximo:", np.abs(previstos-corretos).max())
print("Mínimo:", np.abs(previstos-corretos).min())
```

F *Card* do circuito base compilado pelo ATP-Draw

```
BEGIN NEW DATA CASE
C ————————————————————
C Generated by ATPDRAW outubro, terça-feira 4, 2016
C A Bonneville Power Administration program
C by H. K. Høidalen at SEfAS/NTNU - NORWAY 1994-2015
C ————————————————————
$DUMMY, XYZ000
C dT >< Tmax >< Xopt >< Copt ><Epsiln>
1.E-6 .1
500 1 1 1 1 0 0 1 0
C 1 2 3 4 5 6 7 8
C 345678901234567890123456789012345678901234567890123456789012345678901234567890
/BRANCH
C < n1 >< n2 ><ref1><ref2>< R >< L >< C >
C < n1 >< n2 ><ref1><ref2>< R >< A >< B ><Leng><><>0
VR002 XL011B .05 30. 0
VR001 XL011A .05 30. 0
VR003 XL011C .05 30. 0
VE002 X0001C 5. 120. 0
VE003 X0001A 5. 140. 0
VE001 X0001B 6. 100. 0
XF0001 .5 0
XF0003 .5 0
XF0002 .5 0
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL001B, XL001C, XL001A, XL002B $$
, XL002C, XL002A
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL002B, XL002C, XL002A, XL003B $$
, XL003C, XL003A
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL003C, XL003A, XL003B, XL004C $$
, XL004A, XL004B
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL004C, XL004A, XL004B, XL005C $$
, XL005A, XL005B
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL005B, XL005C, XL005A, XL006B $$
, XL006C, XL006A
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL006B, XL006C, XL006A, XL007B $$
, XL007C, XL007A
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL007A, XL007C, XL007B, XL008A $$
, XL008C, XL008B
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL008A, XL008C, XL008B, XL009A $$
, XL009C, XL009B
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL009A, XL009B, XL009C, XL010A $$
, XL010B, XL010C
$INCLUDE, C:\ATPdraw\Atp\lttcc_L10.lib, XL010A, XL010B, XL010C, XL011A $$
, XL011B, XL011C
/SWITCH
C < n 1>< n 2>< Tclose ><Top/Tde >< Ie ><Vf/CLOP >< type >
XL001BX0001B MEASURING 1
XL001CX0001C MEASURING 1
XL001AX0001A MEASURING 1
XSWT01XF0001 .05 -1. 0
XSWT03XF0003 .05 -1. 0
XSWT02XF0002 .05 -1. 0
/SOURCE
C < n 1><>< Ampl. >< Freq. ><Phase/T0>< A1 >< T1 >< TSTART >< TSTOP >
14VR003 341532.575 60. -120. -1. 100.
14VR001 341532.575 60. -1. 100.
14VR002 341532.575 60. 120. -1. 100.
14VE001 318905.158 60. -1. 100.
14VE003 318480.894 60. -119. -1. 100.
14VE002 319470.844 60. 122. -1. 100.
/OUTPUT
XL001BXL001CXL001A
BLANK BRANCH
BLANK SWITCH
```

```
BLANK SOURCE
BLANK OUTPUT
BLANK PLOT
BEGIN NEW DATA CASE

BLANK
```