ANÁLISE DE TRANSITÓRIOS ENVOLVENDO DESLOCAMENTO DA FREQUÊNCIA.

Felipe de Alverga Feital Caseira

Projeto de Graduação apresentado ao Curso de Engenharia Elétrica da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Antonio Carlos Siqueira de Lima

RIO DE JANEIRO, RJ - BRASIL
FEVEREIRO DE 2012

ANÁLISE DE TRANSITÓRIOS ENVOLVENDO DESLOCAMENTO DA FREQUÊNCIA.

Felipe de Alverga Feital Caseira

PROJETO SUBMETIDO AO CORPO DOCENTE DO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA.

Aprovada por:

_____
Prof. Antonio Carlos Siqueira de Lima
(Orientador)

_____
Prof. Robson Francisco da Silva Dias

_____
Profa. Tatiana Mariano Lessa de Assis

RIO DE JANEIRO, RJ - BRASIL
FEVEREIRO DE 2012

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista.

Análise dos transitórios envolvendo o deslocamento da frequência.

Felipe de Alverga Feital Caseira

Fevereiro/2012

Orientador: Antonio Carlos Siqueira de Lima

Curso: Engenharia Elétrica

Esse projeto tem por objetivo apresentar o uso do deslocamento em frequência para o estudo de transitórios em redes elétricas simples. Para tanto, foi desenvolvida uma ferramenta de simulação utilizando uma linguagem de programação orientada à objeto e empregando um programa de cálculos matemáticos. A principal vantagem do emprego do deslocamento em frequência consiste no uso de passo de cálculos maiores aos comumente empregados. Um breve resumo da Transformada de Hilbert e de sinais analíticos no domínio do tempo complexo são apresentados.

Foram simulados diferentes circuitos, RL, RC, e um circuito considerando o chaveamento onde há apenas elementos concentrados. Os resultados empregando as duas formulações foram concordantes na maioria dos casos. Contudo, a representação de elementos chaveamentos apresentou diferenças significativas quanto a representação de chaves ideais.

Palavras-chave: Fasor Dinâmico, Transformada de Hilbert, Transitórios eletromagnéticos.

SUMÁRIO

1. Introdução 1
1.1. Objetivo 1
1.2. Descrição 2
2. Ferramentas Matemáticas Empregadas 3
2.1. Transformada de Hilbert 3
2.2. Fasores Girantes e Deslocamento em Frequência 4
3. Modelagem dos componentes usando o deslocamento da frequência 7
3.1. Modelagem dos elementos passivos. 7
3.1.1. Modelagem de uma indutância acoplada a uma resistência 7
3.1.2. Modelagem da capacitância 8
3.2. Modelagem das Fontes 8
3.3. Montagem da matriz de admitância nodal 9
3.3.1. Método do colapso de nós 9
3.3.2. Método nodal modificado 10
3.4. Estrutura do Programa em linguagem C++ 10
3.4.1. Aplicação da linguagem C++ na computação cientifica 10
3.5. Casos de Teste 17
3.5.1. Circuito RC 18
3.5.2. Circuito RL 19
3.5.3. Circuito RLC 19
3.5.4. Circuito com Chaveamento 20
3.5.5. Circuito com mutiplos nós e chaveamentos. 21
4. Conclusão 24
Apêndice A: Classes em C++ 25
Apêndice B Scripts em Matlab 35
Referências Bibliográficas 37

1. INTRODUÇÃO

As redes encontradas nos sistemas de transmissão e distribuição de energia elétrica são comumente analisadas sob alguns aspectos bastantes "rígidos". Há o estudo e análise das redes em regime permanente, onde a representação empregando sequência positiva é realizada no domínio da frequência empregando fasores. Há a avaliação do comportamento da rede face às pequenas ou grandes perturbações envolvendo apenas a frequência fundamental, conhecido como análise de transitórios eletromecânicos. E por fim, há os estudos de transitórios eletromagnéticos onde faz-se necessário abandonar a representação em redes de sequência e os sistemas tornam-se acoplados e multifásicos.

Até 1999, o sistema brasileiro de transmissão de energia em Alta-Tensão e Extra-Alta-Tensão, envolvendo níveis de tensão acima de 230 kV, era composto por dois grandes subsistemas, a saber, Norte-Nordeste e Sul-Sudeste. Com o aumento da interconexão das redes elétricas, em particular do sistema brasileiro, há uma necessidade maior de integração das ferramentas de análise. Ainda mais se levado o grau de severidade das possíveis intercorrências, como o recente blecaute de novembro de 2009, no qual faltou luz em 14 estados brasileiros e na maior parte do Paraguai. Se por um lado, a análise em regime permanente permite a representação de praticamente todo o sistema de transmissão, ela demanda um elevado grau de simplificação. Por outro lado, o estudo de transitórios eletromagnéticos demanda uma quantidade de dados maior, o que acarreta em limitação das dimensões das redes a serem analisadas.

Das ferramentas empregadas recentemente em sistemas de potencia, o uso de fasores girantes ou fasores dinâmicos se mostra bastante interessante, já tendo sido empregada na análise da rede sob condições de defeito assimétricos 0. Com o intuito de melhorar ainda o comportamento numérico dos sistemas de equações a serem resolvidos pode-se em conjunto com o fasor dinâmico se empregar o deslocamento em frequência, que consiste, conforme será mostrado mais adiante nesse documento, na modulação do sinal da portadora, nesse caso as tensões/corrente na frequência industrial.

1.1. OBJETIVO

O principal objetivo desse projeto é verificar o comportamento do deslocamento de frequência para a análise de transitórios em redes elétricas com passos de cálculo mais elevados que usualmente empregados. A princípio, o passo de cálculo deve ser pelo menos uma ordem de grandeza menor que a menor constante de tempo envolvida, porém

do ponto de vista prático, para transitórios de manobra, considera-se tipicamente um passo de cálculo da ordem de 50μs. Como se trata de um tema novo, antes da aplicação em configurações reais, consideram-se nesse documento apenas algumas configurações simples envolvendo apenas elementos concentrados. São considerados diversos passo de cálculo chegando a valores tipicamente encontrados em estudos de transitórios eletromecânicos, da ordem de alguns milésimos de segundos.

Um segundo objetivo deste trabalho é a comparação do desempenho computacional da metodologia usada para a representação do deslocamento em frequência considerando diferentes ferramentas como a linguagem de programação C++ e o MATLAB. Busca-se também demonstrar as vantagens do desenvolvimento empregando a programação orientada a objeto.

1.2. DESCRIÇÃO

Este trabalho está dividido em quatro capítulos contando com esse capítulo de introdução a saber:

- O Capítulo 2 apresenta uma breve descrição de fasores dinâmicos, transformada de Hilbert e deslocamento em frequência;
- O Capítulo 3 trata do desenvolvimento e funcionamento do programa, incluindo os casos testes.
- O Capítulo 4 traz as principais conclusões do presente projeto e sugestões para investigações futuras.

2. FERRAMENTAS MATEMÁTICAS EMPREGADAS

Conforme mencionado na introdução o presente capítulo tem por objetivo apresentar as informações principais do emprego de fasores girantes, Transformada de Hilbert e deslocamento em frequência. Embora sejam assuntos que mereçam de forma isolada um grande nível de detalhamento, o foco aqui é em produzir um conjunto básico de informações de forma a tornar o presente documento o mais autocontido possível.

2.1. TRANSFORMADA DE HILBERT

Antes da definição da Transformada de Hilbert é necessário estabelecer alguns conceitos fundamentais. O primeiro deles consiste na definição de sinais analíticos de tempo complexo, ou sinais exponenciais complexos 0. O termo consiste na tradução de "*analytic signal*" não confundir com funções analíticas ("*analytic functions*") 00. A formulação surgiu na área de processamento de sinais visando elaborar uma forma mais eficiente do cálculo da resposta em frequência e forma de onda no domínio de tempo de funções periódicas. Apresenta-se a seguir as definições necessárias para estabelecer um sinal analítico.

De acordo com a Teoria da Transformada de Fourier se f(t) é uma função real e F(ω) é a transformada de Fourier dada por (2.1)

$$F(\omega) = \mathcal{F}[f(t)] = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{2.1}$$

pode-se relacionar as respostas para frequências negativas e positivas por

$$F(-\omega) = F(\omega)^* \tag{2.2}$$

onde '*' significa conjugado complexo. Um sinal analítico de tempo complexo é definido como uma função cuja transformada para frequências negativas é nula, ou seja,

$$F_a(\omega) = \begin{cases} 2F(\omega) & \text{para} \quad \omega > 0 \\ F(0) & \text{para} \quad \omega = 0 \\ 0 & \text{para} \quad \omega < 0 \end{cases} \tag{2.3}$$

A expressão em (2.3) pode ser representada de forma um pouco mais compacta se a função degrau, u(.), for aplicada à frequência angular ω, conforme mostra a (2.4).

$$F_a(\omega) = F(\omega)\, 2\, u(\omega) \tag{2.4}$$

De acordo com a Teoria da Transformada de Fourier, multiplicação no domínio da frequência implica em convolução no domínio do tempo

$$f_a(t) = \mathcal{F}^{-1}[F(\omega)] * \mathcal{F}^{-1}[2u(\omega)] = \mathcal{F}^{-1}[F(\omega)] * \mathcal{F}^{-1}[1 + \text{sign}(\omega)] \tag{2.5}$$

$$f_a(t) = f(t) * \left[\delta(t) + j\frac{1}{\pi\, t}\right] = f(t) + j\left[f(t) * \frac{1}{\pi\, t}\right] = f(t) + j\hat{f}(t) \tag{2.6}$$

O termo $\hat{f}(t)$ corresponde a Transformada de Hilbert do sinal $f(t)$. Portanto do formalismo matemático é possível definir a Transformada de Hilbert de um sinal $f(t)$ conforme mostrado a seguir.

$$\hat{f}(t) = \mathcal{H}[f(t)] = \frac{1}{\pi}\int_{-\infty}^{\infty} \frac{f(t)}{t-\tau}\, d\tau \tag{2.7}$$

Similar ao que acontece nas integrais que ocorre na Transformada de Fourier, a integral em (2.7) pode não convergir e nesse caso não haveria Transformada de Hilbert. Alguns autores optam por definir a Transformada de Hilbert de uma forma um pouco distinta, conforme mostrado abaixo

$$\hat{f}(t) = \mathcal{H}[f(t)] = \frac{1}{\pi}\text{P.V.}\int_{-\infty}^{\infty} \frac{f(t)}{t-\tau}\, d\tau \tag{2.8}$$

no qual P.V. significa Valor Principal. Programas como o *Mathematica*, utilizam a definição de (2.8) 0. Já o MATLAB emprega uma versão alternativa baseada no uso da Transformada Rápida de Fourier (FFT) 00. Considere, por exemplo, o simples caso de uma excitação dada por,

$$f(t) = V_0 \cos(\omega_0 t) \tag{2.9}$$

cuja Transformada de Hilbert é definida por

$$\hat{f}(t) = V_0 \sin(\omega_0 t) \tag{2.10}$$

logo, o sinal analítico de tempo complexo é dado por

$$f(t) + j\hat{f}(t) = V_0 \cos(\omega_0 t) + V_0 \sin(\omega_0 t) = V_0 e^{j\omega_0 t} \tag{2.11}$$

2.2. FASORES GIRANTES E DESLOCAMENTO EM FREQUÊNCIA

A representação de um sinal periódico empregando fasores girantes consiste no emprego da decomposição do sinal em série de Fourier empregando exponenciais, conforme mostrado abaixo

$$f(t) = \sum_{k=-\infty}^{\infty} F(k\omega_0)\, e^{jk\omega_0 t} \tag{2.12}$$

no qual $\omega_0 = 2\pi/T$, sendo T o período fundamental da onda periódica, e $F(k\omega_0)$ corresponde aos componentes complexo que podem ser entendidos como fasores. Nos casos de estudo da rede em regime permanente k=1 e não há outros componentes harmônicos.

Sinais representativos de sistemas de energia elétrica quando representados por (2.12) apresentam, tipicamente, uma limitação de banda. Espera-se que tanto as tensões como as correntes que percorrem as redes elétricas possuam um número finito de harmônicos. Nesse caso, assumindo que a fundamental é um dos principais componentes do sinal representado por (2.12), é possível aumentar a robustez numérica do sinal através da demodulação do sinal. A demodulação consiste em deslocar o sinal original no plano complexo conforme mostrado abaixo.

$$f_a(t) = \left(f(t) + j\hat{f}(t)\right) e^{-j\omega_0 t} \tag{2.13}$$

A Fig.2.2.1 ilustra esquematicamente o efeito do demodulação para um sinal que apresenta um espectro centrado na fundamental. Após o deslocamento em frequência o sinal fica centrado em 0Hz.

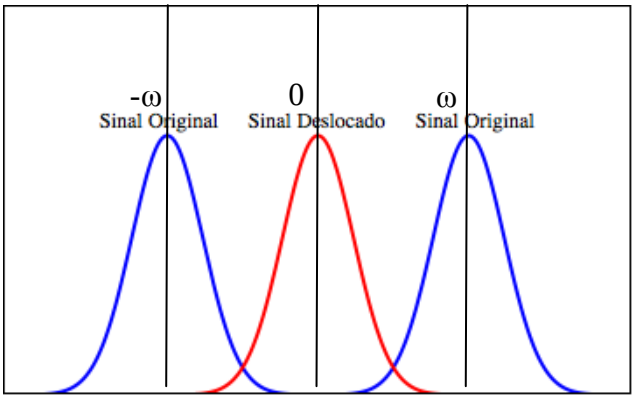


Figura 2.2-1 - Envelope no domínio da frequência de um sinal e o envelope após a aplicação da Transformada de Hilbert e o deslocamento de frequência.

Exemplo

Para ilustrar a aplicação de fasores girantes e da transformada de Hilbert apresenta-se a seguir um exemplo simples da energização de um circuito RLC por uma fonte do tipo cossenoidal. Os dados são os seguintes, R= 5Ω, L=0.25 H, C= 50μF, a fonte possui ângulo zero e uma frequência de 60Hz. A resposta analítica da corrente que alimenta o circuito concentrado é dada por

$i(t)$= -(0.0014517 -1j*0.0160036).*exp((-10.+1j*282.666).* t) + ... (2.14)
(0.0014517 +1j*0.0119609).*exp((0. -1j*376.991).* t) +...
(0.0014517 -1j*0.0119609).*exp((0. +1j*376.991).* t) + ...
(-0.0014517-1j*0.0160036).*exp((-10.-1j*282.666).* t);

A transformada de Hilbert é empregada via FFT para obter o sinal analítico. A Fig. 2.2.2 apresenta o resultado do cálculo da corrente original com a envoltória obtida através do absoluto do sinal analítico.

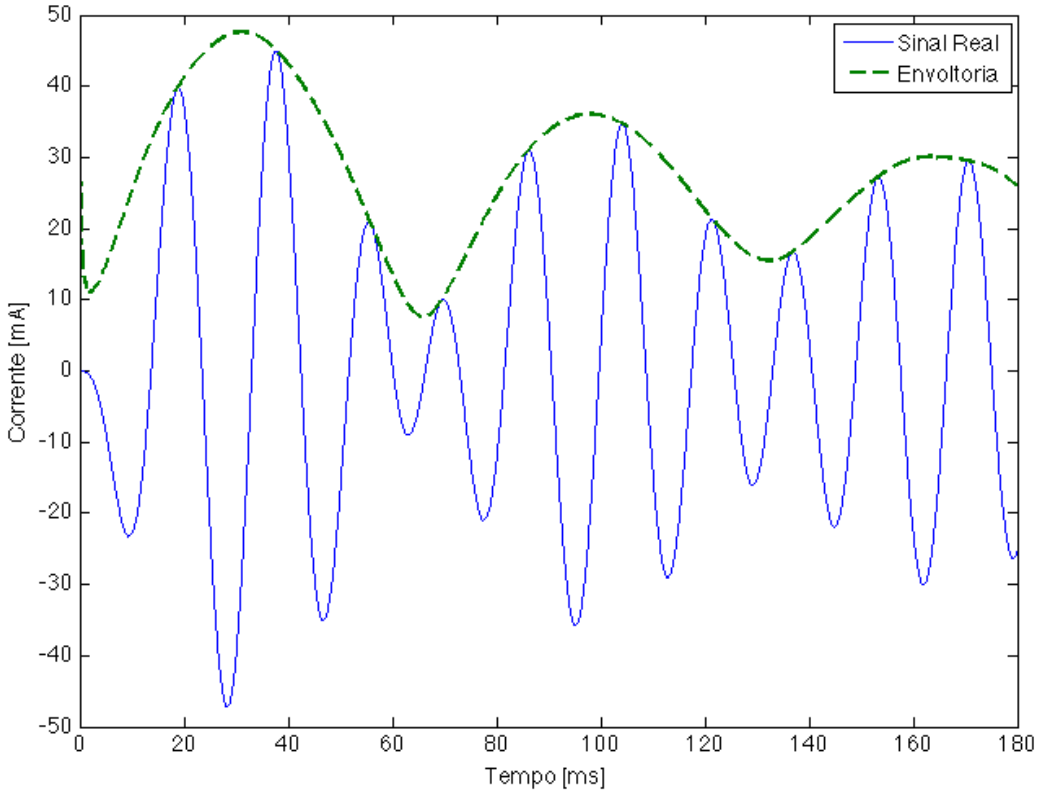


Figura 2.2-2 - Envelope no domínio da frequência de um sinal e o envelope após a aplicação da Transformada de Hilbert e o deslocamento de frequência.

Como os resultados da simulação é o sinal analítico do sistema, é possível para a análise dos dados usar o conceito de frequência instantânea do sinal. A frequência instantânea é a derivada no tempo do ângulo de fase de um sinal complexo.

$$\omega(t) = \frac{d\varphi}{dt} \tag{2.15}$$

O uso da frequência instantânea serve como indicador se a solução transiente atingiu o regime permanente ou se quase atingiu o regime permanente.

3. MODELAGEM DOS COMPONENTES USANDO O DESLOCAMENTO DA FREQUÊNCIA

Nesse capítulo será discutida a modelagem dos elementos e fontes. Modelagem esta que será apenas de elementos de parâmetros concentrados de rede e fontes do tipo cossenoidais. Também será discutida a montagem da matriz de admitância nodal e estruturação do programa em linguagem C++.

3.1. MODELAGEM DOS ELEMENTOS PASSIVOS.

Com o uso do fasor dinâmico é necessário remodelar os elementos passivos do sistema para poder ser feito o seu uso no desenvolvimento de uma ferramenta de análise dos transitórios eletromagnéticos

Para a remodelagem foi feita da seguinte forma. Criaram-se as equações diferenciais dos componentes no domínio do tempo coordenadas por fase. As variáveis no domínio do tempo foram transformadas em fasores dinâmicos. Por fim foi feita a discretização das equações utilizando o método de integração da regra do trapézio e criou-se um equivalente para o uso na ferramenta para a solução de transitórios magnéticos.

3.1.1. MODELAGEM DE UMA INDUTÂNCIA ACOPLADA A UMA RESISTÊNCIA

No domínio do tempo a equação diferencial que descreve a tensão de uma indutância acoplada a uma resistência em série é dada por

$$v(t) = R\, i(t) + L\frac{di(t)}{dt} \tag{3.1}$$

o circuito equivalente para o modelo EMTP é dado por $i(t) = g_{rl}\, v(t) - h_{rl}$ no qual

$g_{rl} = (R + \frac{2L}{\Delta t})^{-1}$

e

$h_{rl} = g_{rl}\left[\left(R - \frac{2L}{\Delta t}\right) g_{rl} - I\right] v(t - \Delta t) - g_{rl}\left(R - \frac{2L}{\Delta t}\right) h_{rl}(t - \Delta t)$ é os termos históricos

Transformando a equação 3.1 para o uso do fasor dinâmicos.

$$V(t) = RI(t) + \frac{Ldi(t)}{dt} + j\omega LI(t) \tag{3.2}$$

Usando a regra do trapézio para executar a discretização

$$I(t) = G_{rl}\, V(t) - H_{rl} \tag{3.3}$$

No qual Grl é a expressas da seguinte forma admitância equivalente e Hrl o termo de correntes históricas que são

$$G_{rl} = (R + \frac{2L}{\Delta t} + j\omega L)^{-1}$$

E

$$H_{rl} = G_{rl}\left[\left(R - \frac{2L}{\Delta t} + j\omega L\right)G_{rl} - I\right]V(t-\Delta t) - G_{rl}\left(R - \frac{2L}{\Delta t} + j\omega L\right)H_{rl}(t-\Delta t) \quad (3.4)$$

Dessa forma percebe-se que o modelo no deslocamento da frequência é similar ao do domínio do tempo.

3.1.2. MODELAGEM DA CAPACITÂNCIA

No domínio tempo a equação da corrente da capacitância é escrita da seguinte forma:

$$i(t) = C\frac{dv(t)}{dt} \quad (3.5)$$

Na forma do deslocamento da frequência esta pode ser reescrita

$$I(t) = C\frac{dV(t)}{dt} + j\omega CV(t) \quad (3.6)$$

usando a regra do trapézio para a integração a equação a diferenças encontrada é

$$I(t) = G_c V(t) - H_c(t) \quad (3.7)$$

Em que $G_c = \frac{2C}{\Delta t} + j\omega C$ e

$$H_c = \frac{4C}{\Delta t}V(t-\Delta t) - H_c(t-\Delta t) \quad (3.8)$$

O termo H_c é equivalente ao usado pelo EMTP no qual os termos são os do deslocamento da frequência.

3.2. MODELAGEM DAS FONTES

Serão modeladas apenas fontes cossenoidais de tensão é corrente. A fonte cossenoidal passa a ser somente

$$V(t) = V_0 e^{-j\omega t} \quad (3.9)$$

Para a modelagem de fontes com harmônicos a fonte o valor da fonte passa a possuir a componente somada ao valor da fonte modulada também.

$$V(t) = V_f e^{-j\omega t} + \sum V_h e^{-j\omega_h t} \quad (3.10)$$

Para poder simular os componentes com os harmônicos é necessário diminuir o passo de integração e simular os componentes das harmônicas de forma individual.

3.3. MONTAGEM DA MATRIZ DE ADMITÂNCIA NODAL

Usando a regra do trapézio, uma rede elétrica com suas equações diferenciais pode ser representada por uma serie de equações algébricas que podem ser interpretadas como admitâncias, tensões, correntes e termos históricos. As equações nodais do sistema podem ser escritas da seguinte forma

$$[G][V(t)] = [I(t)] + [H(t)] \tag{3.10}$$

Em que [G] é a matriz de admitância nodal, [V(t)] o vetor de tensões nodais, [I(t)] o vetor de correntes nodais e [H(t)] é o vetor dos termos históricos.

Para resolver o sistema da equação (3.10) foi desenvolvidos dois métodos de solução o método do colapso de nós e o método do nodal modificado.

3.3.1. MÉTODO DO COLAPSO DE NÓS

O sistema baseado no colapso de nós consiste que para cada chave que conecte dois nós, é eliminado um deles. No algoritmo desenvolvido para a montagem da matriz de admitância nodal a numeração dos nós deve seguir a seguinte ordem. Primeiro deve-se numerar todos os nós não conectados nem a fontes de tensão nem a chaves. Em seguida numerar os nós das chaves e por último numerar os nós das fontes de tensão.

Os nós contendo chaves são eliminados restando somente os que contiverem fontes e elementos. Dessa forma se resume ao sistema linear da equação 3.11. Na qual se pode dividir de acordo com a equação 3.12.

$$G_A\, V = I \tag{3.11}$$

$$\begin{vmatrix} G_{AA} & G_{AB} \\ G_{BA} & G_{BB} \end{vmatrix} \begin{vmatrix} V_A \\ V_B \end{vmatrix} = \begin{vmatrix} I_A \\ I_B \end{vmatrix} \tag{3.12}$$

$$G_{AA}\, V_A = I_A - G_{AB}\, V_B \tag{3.13}$$

Dessa maneira a solução do sistema passa a ser a equação 3.13 no qual V_A é o vetor de tensões desconhecidas, V_B o de tensões conhecidas, I_A o vetor de correntes e termos históricos e G_{AA} é a matriz de admitância nodal com tamanho igual ao número total de nós que não possuam chaves ou fontes de tensão.

3.3.2. MÉTODO NODAL MODIFICADO

Para o método do nodal modificado foi baseado em cima do trabalho do Alvarado[]. Nesse caso o sistema é resolvido através de um sistema linear do tipo A$\underline{x}$=$\underline{b}$ no qual a Matriz A é composta de quatro submatrizes Yn, Va, Sa e S0. A matriz Yn é a matriz de admitância nodal do sistema, a matriz Va é a matriz de incidência de fontes de tensão do sistema, a matriz Sa é a matriz de incidência de chaves do sistema e a matriz S0 é a matriz para anular as correntes nas chaves abertas.

$$\begin{vmatrix} Yn & Va^{t} & Sa^{t} \\ Va & 0 & 0 \\ Sa & 0 & S0 \end{vmatrix} \tag{3.3.2.5}$$

Com o vetor b sendo igual a:

$$\begin{vmatrix} In \\ Vs \\ 0 \end{vmatrix} \tag{3.6}$$

Em que In são as correntes nodais e Vs as tensões das fontes. O vetor x igual a:

$$\begin{vmatrix} In \\ Vs \\ Is \end{vmatrix} \tag{3.7}$$

Em que Vn são as tensões nodais Ivs as correntes nas fontes de tensão e Is a correntes nas chaves.

3.4. ESTRUTURA DO PROGRAMA EM LINGUAGEM C++

O programa foi desenvolvido a partir de simulações de casos de teste em linguagem de programação C++. Os resultados dos casos de teste foram comparados com os resultados obtidos destes simulados no programa Matlab.

Os desenvolvimentos feitos para os casos de teste serviram como base para programa à medida que eram desenvolvidos os casos de teste o ferramental numérico do programa, foi sendo testado e comparado.

3.4.1. APLICAÇÃO DA LINGUAGEM C++ NA COMPUTAÇÃO CIENTIFICA

A linguagem de programação C++ começou a ser desenvolvida em 1979, por Bjarne Stroustrup nos laboratórios Bell, como uma melhoria da linguagem C, sendo a primeira

publicação comercial da linguagem em 1985. Apesar da linguagem C ser uma das mais usadas no mundo ela possui limitações. Como toda vez que um programa começava a crescer demais (acima de 25.000 linhas de código) este se tornava extremamente complexo para sua total compreensão. O propósito do C++ era poder ultrapassar essa barreira sem se tornar extremamente complexo à compreensão do programa.

A maioria das melhorias implementadas foram relacionadas à programação orientada a objetos. Como uso de classes, um maior controle da memória alocada, a possibilidade da mesma função receber diferentes argumentos e o próprio programa definir como tratará os dados que é também chamado de sobre carregamento de funções (“function overloaded”).

Uma classe em C++ é um objeto instanciado dentro de um programa com múltiplas funções e objetos dentro dele. Os objetos dentro da classe podem ser outras classes, ponteiros ou variáveis. As funções da classe são métodos chamados pelo programa. Métodos estes que podem ser próprios da classe ou de algum objeto dentro da classe.

As funções e objetos dentro da classe podem existir de três formas, público, privado, ou protegido. Um elemento público pode ser chamado ou alterado por qualquer parte do programa enquanto este estiver instanciado. Um elemento privado só pode ser chamado ou alterado por um elemento dentro da classe ou que herde ou possua acesso aos elementos da classe. Um elemento protegido só pode ser chamado ou alterado por um elemento pertencente à classe.

Em 1994 começou um comitê conjunto da ANSI/ISO para a padronização do C++. Logo após a publicação do primeiro resumo do padrão foi criado, por Alexander Stepanov, a biblioteca padrão de modelos. Biblioteca essa que contemplava vários modelos de funções e logo foi adicionada ao padrão, causando um atraso na finalização da padronização. Hoje todos os principais compiladores aceitam o uso do padrão criado pela ANSI/ISO[10].

O C/C++ é uma linguagem de programação voltada para programadores, da forma que este oferece poucas restrições ao acesso da memória, uma quantidade pequena de palavras chave e uma estruturação em blocos. Os códigos em C/C++ são portáveis, ou seja, o código desenvolvido em uma máquina Linux possui uma grande possibilidade de funcionar numa máquina Windows sem precisar fazer alterações no código. Por causa de suas poucas restrições em relação ao acesso de memória é necessária uma atenção especial nos acessos das variáveis durante a execução do programa, pois um acesso errado pode causar a falha do programa ou mesmo do sistema operacional.

Essa grande liberdade faz com que seja uma excelente escolha para o uso na computação científica. Porém ainda não é muito utilizada, pois a principal linguagem de

computação científica ainda é o Fortran, que foi desenvolvido para esse fim, na década de 50. Essa resistência existe em parte por causa dos primeiros testes na década de 90 em que apontavam o Fortran como sendo mais rápido que o C++ e também por causa da larga biblioteca de álgebra linear já desenvolvida para o Fortran.

O desenvolvimento da biblioteca "f2c" que habilita o C++ a traduzir porções de código em Fortran e criação das bibliotecas de álgebra linear específicas para C++ como a LAPACK ajudou a aumentar seu uso da linguagem na computação científica.

O C/C++ é caracterizado como uma linguagem de programação de nível médio, ou seja, esta permite que se programe tanto em linguagem de máquina como Assembler, ou de forma estruturada como Pascal. C/C++ quase não faz checagem de erro em execução. Por exemplo, ele não checa se os limites dos vetores foram extrapolados durante uma iteração o que pode causar problemas durante a execução uma vez que ele pode acessar um endereço inválido ou em uso pelo sistema operacional.

Uma grande vantagem do C/C++ é o uso de bibliotecas que podem ser encadeadas. O encadeamento de bibliotecas é as funções serem declaradas em uma biblioteca estarem disponíveis para o uso numa outra biblioteca. Um exemplo disso é uma biblioteca de matemática no qual há uma de operações matemáticas (soma, subtração e etc.) outra de funções matemáticas (seno, exponencial e etc.) no qual a biblioteca de matemática possa chamar a biblioteca de funções ou de operações dependendo da necessidade. Com isso programa passa a ter um número reduzido de linhas uma vez que ele só chama as funções do arquivo principal. A biblioteca padrão de funções (Standard function library) é a principal biblioteca uma vez que esta possui um grande número de funções já escritas em C++.

Por ser uma linguagem orientada a objetos e as funções serem alocadas em bibliotecas o programa torna-se modular. Portanto, no desenvolvimento de novos programas existe a possibilidade de se compartilhar as bibliotecas e só utilizar as partes de interesse. Isso é válido, uma vez definida uma classe de objetos de fontes de tensão está pode possuir diferentes tipos de fontes (por exemplo, continua, alternada, e onda quadrada). No momento da criação do objeto na execução do programa é que se define o tipo da fonte e a partir dos objetos instanciados na classe definirem a função de saída da fonte.

Para modelos mais complexos como máquinas e linhas de transmissão também se pode aplicar este conceito. Além disso, as partes da modelagem dos elementos de maior complexidade que sejam comuns a outros elementos como indutâncias e capacitâncias. A classe de objetos pode possuir estes elementos como objetos instanciados dentro da classe.

Na modelagem de um sistema de potência, que é composto por linhas, máquinas, fontes e compensações, cria-se uma classe com estes objetos que foram modelados. Criam-se estes elementos à medida que forem necessários.

Com isso um único programa pode fazer o cálculo de um sistema simples ou de um complexo a partir dos parâmetros de entrada do programa. Dessa maneira podem-se simular diferentes sistemas a partir de um único programa.

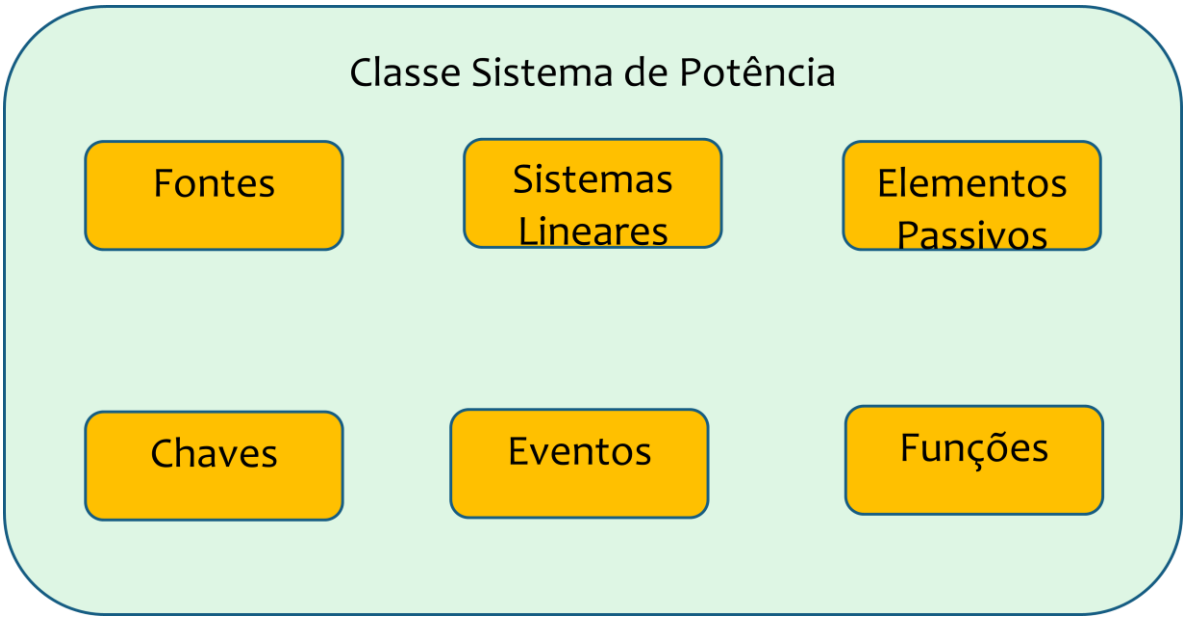


Figura 3.4.1-1 Encapsulamento das classes

Na figura 3.4.1.1 mostra como é encapsulada as classes no programa. A classe de Sistema de Potência possui dentro dela as classes referentes a fontes, solução de sistemas lineares, elementos passivos, chaves, eventos e as funções e elementos que pertencem somente à classe.

O desenvolvimento do programa foi iniciado através de simulações de circuitos simples. Para isso foram desenvolvidas as classes de elementos passivos e de fontes do sistema.

Os resultados foram exportados os valores para um arquivo de texto com o formato csv (" comma separated values" em português, valores separados por vírgulas). Este formato foi escolhido porque possibilita a criação de uma tabela com os resultados a partir de um arquivo de texto no qual a vírgula separa as células. Esse tipo de formato é facilmente importado pelo gnuplot ou mesmo ser tratado em um editor de planilhas como Microsoft Excel ou o Openoffice Calc.

Para a visualização gráfica dos resultados foi utilizado o gnuplot. O gnuplot é uma ferramenta de software livre com o código aberto que serve para realizar plotagens gráficas. Não foi feita uma integração direta dos resultados do programa com o gnuplot para serem feitos os gráficos. Para isso seria necessário um conhecimento de programação mais avançado e estava fora do escopo inicial do projeto.

A intenção inicial para o programa era o uso de alocação dinâmica da memória na maioria dos objetos relativos ao sistema. Com isso o programa estaria preparado para solucionar desde sistemas simples com poucos nós e elementos até sistemas mais complexos.

Isso se demonstrou muito mais complexo do que havia sido inicialmente pensado. Inicialmente se usou as funções básicas para alocação de memória. Porém à medida que eram instanciados objetos na memória, como a criação de matrizes e vetores para a solução, o programa perdia referência dos endereços de memória. Com isso os valores armazenados deixavam de estar corretos causando soluções numéricas divergentes.

Na pesquisa por soluções foi descoberto que seria necessária a criação de funções para alocação de memória para cada uma das classes criadas. Para implantar essa solução de forma a garantir o funcionamento do programa seriam necessárias técnicas de programação mais avançadas que não era o objetivo deste trabalho.

Com isso foi usado alocação estática de memória na qual foram definidos alguns parâmetros como sendo o máximo tamanho do sistema. Foi definido que o sistema possuiria no máximo dez elementos passivos, cinco fontes, cinco chaves e no máximo cinco chaveamentos ocorreriam.

Para a solução do sistema linear inicialmente ir-se-ia fazer uma integração com a biblioteca de funções JAMA do NIST (National Institute of Standards and Technology). Essa biblioteca possui as funcionalidades de álgebra linear já desenvolvidas para C++ similar a BLAS para o Fortran.

Por causa da restrição técnica do programa quanto ao uso de alocação dinâmica de memória a necessidade dessa integração diminuiu. Dessa forma foi utilizado para a solução de sistemas lineares o algoritmo de decomposição LU[5] adaptado para a utilização direta de números complexos.

Para as matrizes e vetores foi utilizado o container vector. O container vector é um modelo (template no original) de vetores no qual já possui diversas funções prontas para o uso. Este possui já definido o uso de alocação dinâmica, funções e determinados operadores como o operador igual que copia os elementos de um objeto para outro. Com isso há vantagens em relação ao uso dos arrays para vetores. Para o seu uso basta definir um tipo de

elemento (por exemplo, números inteiros, complexos e etc.) e na alocação de memória passa a ser feita internamente.

O acesso aos elementos da matriz é feito com a equação:

$$Posição=tam\_linha*coluna+linha \quad (3.xx)$$

Aonde o produto do tamanho da linha pelo o tamanho da coluna é o número total de elementos da matriz.

O uso de decomposição LU para solução se deve ao fato que para a maior parte do tempo de simulação a matriz não se altera. Dessa forma só é necessário recalcular caso haja alguma alteração na matriz e reduz o numero de operações para o calculo da solução.

Por fim foi desenvolvida a classe de chaves e de eventos que são utilizadas nos métodos de solução do sistema.

Independente do método usado, o programa é executado da seguinte forma: Inicialmente são lidos os arquivos de entrada do sistema e armazenados na memória para que possam ser usados durante a execução. É calculado o número de iterações que serão executadas no sistema. São inicializados os dados históricos do sistema. Monta-se a matriz de solução do sistema e esta é triangularizada.

Começam as iterações que possuem a seguinte estrutura. Inicialmente é checado se houve chaveamentos. Em caso positivo recalcula-se e a matriz e novamente é triangularizada. Calculam-se os valores instantâneos das fontes e monta-se o vetor de correntes. Soluciona-se o sistema linear e são atualizadas as correntes históricas dos elementos. Armazenam-se em arquivos individuais os valores das tensões e das correntes do sistema. Caso a iteração atual ainda seja menor do que a interação final um novo ciclo é começado.

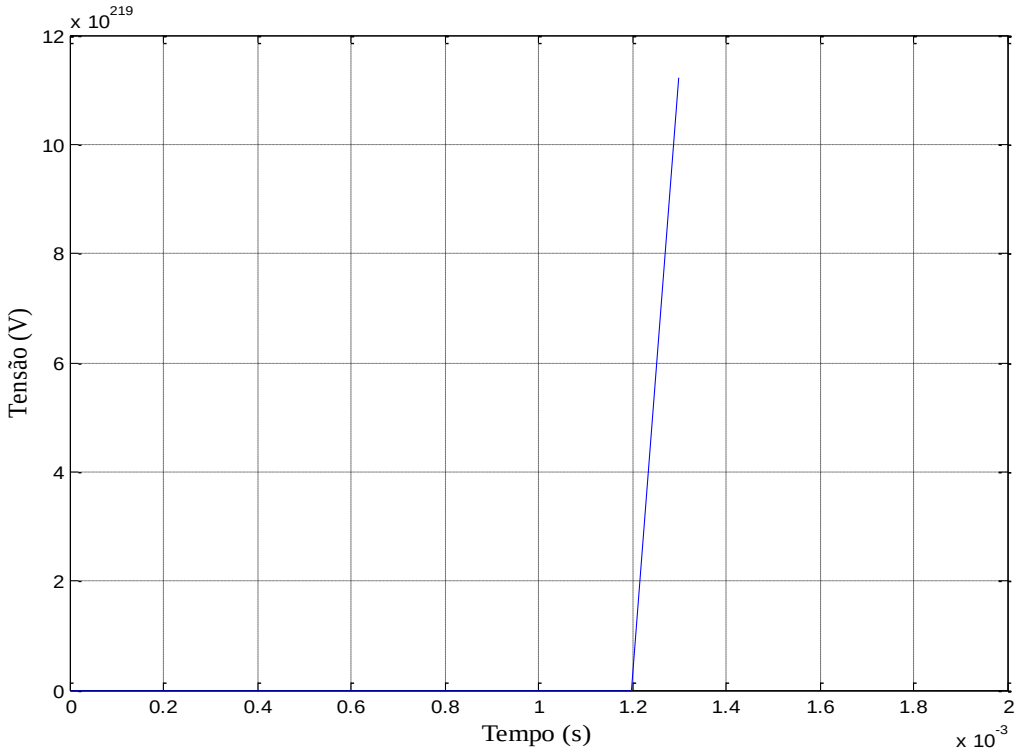
Ao termino das interações o sistema fecha os arquivos e retorna uma mensagem avisando o fim da execução do programa.

No programa final foram testados os dois tipos de solução. A solução pelo o método de colapso de nós funcionou de forma esperada sendo possível a extração dos dados.

Houve problemas com o método do nodal modificado. A solução divergia rapidamente causada pelo o pico inicial da solução. Por este motivo não foi incluído na versão final do programa. A figura 3.4.1.2 mostra o resultado da simulação por este método. Em poucas interações a solução diverge apesar de usar o mesmo passo do colapso de nós.

A figura 3.4.1.3 é uma visão macro do mapeamento do processo de execução do programa. Em quase todos eles existem subprocessos sendo executados, como checagem de erros do programa. Caso algum arquivo não seja lido por qualquer motivo o programa exibe uma mensagem de erro e finaliza o programa.

Esses subprocessos não foram mostrados abertamente para não ficar confusa a figura. Não caberia demonstrar cada um deles mesmo que individualmente uma vez que pouco acrescentaria no trabalho. A demonstração de um subprocesso já estaria quase no nível de programação.



3.4.1-2 - Simulação do nodal modificado divergindo

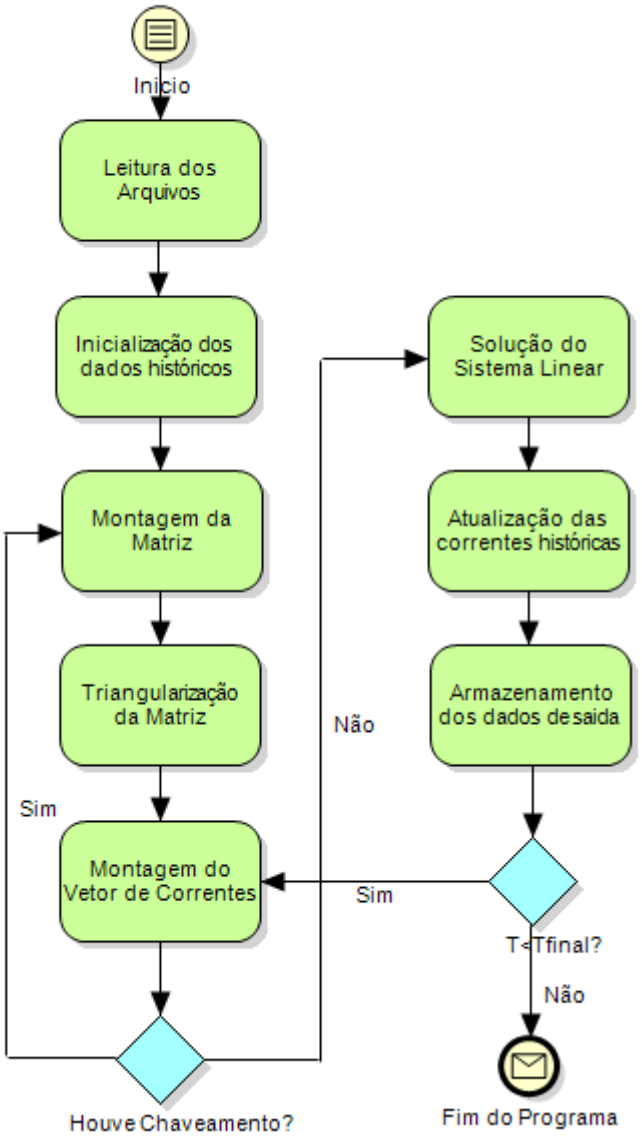


Figura 3.4.1-3 - Fluxo de funcionamento do programa

3.5. CASOS DE TESTE

Para poder melhor compreender o fasor dinâmico e o deslocamento da frequência foram simulados circuitos elétricos simples, como circuito RC, RL e RLC. Para esse desenvolvimento foi utilizado o script do Matlab para simular os elementos modelados. Essas simulações serviram de base para o desenvolvimento da biblioteca do programa principal. A partir dessas simulações foi possível analisar o comportamento numérico do fasor dinâmico e

qual seria a melhor forma de exportar os dados para tratamento posterior. Além de servir como base para análises iniciais.

3.5.1. CIRCUITO RC

Inicialmente foi simulado um circuito RC pela sua simplicidade de sua implantação. Este serviu para começar o desenvolvimento da classe de elementos passivos. Logo foi percebido que existem poucos elementos que possam ser privados na classe. Por esse motivo existiu a necessidade de um maior cuidado no desenvolvimento. Para que durante a execução não se perdesse ou modificasse informações vitais do objeto instanciado.

Análise similar também foi feita para a classe de fontes. Não foi feita distinção entre uma fonte de corrente e de tensão como função de retorno. Porém há variáveis de controle para cada tipo de fonte.

O circuito simulado foi um circuito com resistência de R=15 kΩ e capacitância de C=1μF e fonte de tensão de V=100 Volts.

Na figura 3.5.1.1 encontra-se o resultado da simulação do circuito RC. O envelope da tensão se comporta de maneira esperada percebendo o comportamento da onda. Foi utilizado um passo de 1ms para esse circuito

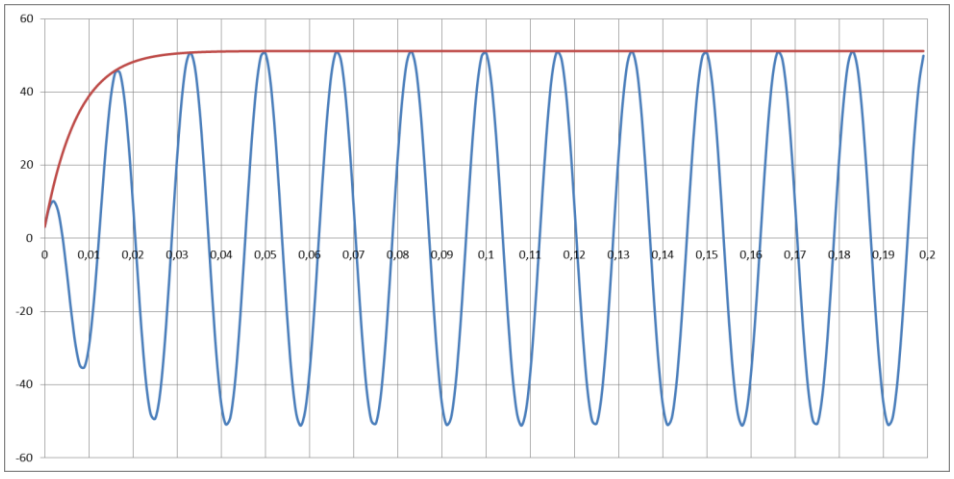


Figura 3.5.1-1 - Tensão no capacitor de um Circuito RC Serie

3.5.2. CIRCUITO RL

Em seguida foi desenvolvido o circuito RL. Para o termo de corrente histórica se for tratado somente o indutor este é diferente do representado pela equação 2.17. Porém para a construção da classe os elementos passivos foram utilizados os termos desta equação.

A classe de elementos passivos armazena as barras na qual o elemento está conectado, os valores da resistência, indutância, capacitância, a admitância, e o valor de corrente histórica do elemento. A admitância é calculada assim que os valores são inseridos. O cálculo da corrente histórica é uma função dentro da classe precisa possuir como argumentos a tensão e corrente do elemento.

O circuito simulado foi com resistência de R=30 Ω e indutância de L=300mH e fonte de tensão de V=200 Volts com passo de integração de 1 ms.

Na figura 3.5.2.1 podemos observar o comportamento do envelope consistente com a corrente no indutor.

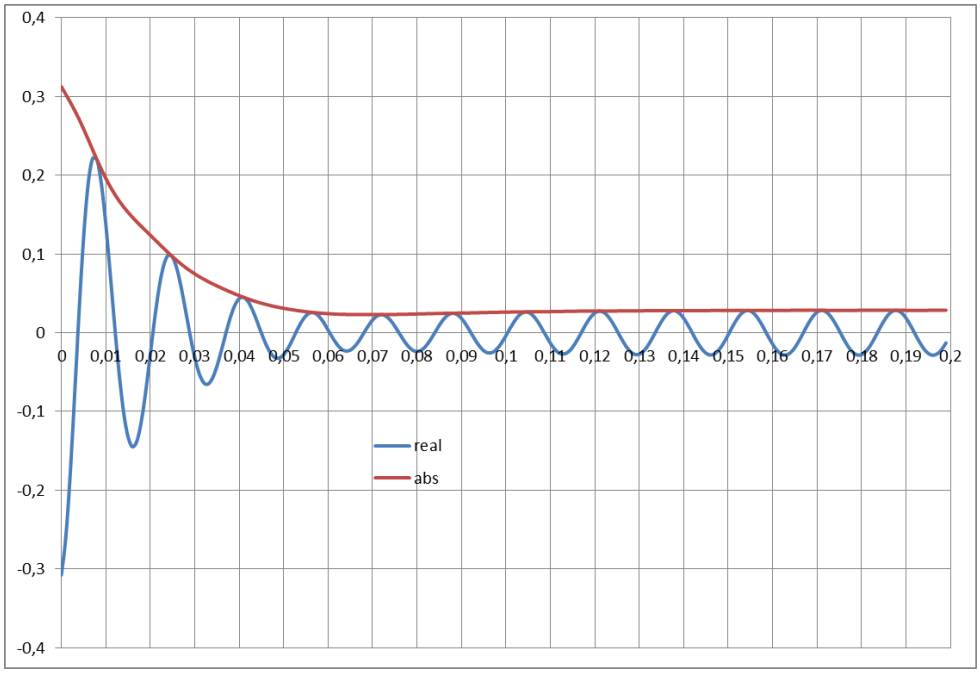


Figura 3.5.2-1 - Corrente do indutor no RL serie

3.5.3. CIRCUITO RLC

O circuito RLC serie serviu como teste para o uso da biblioteca de solução de sistemas lineares e comparação com os resultados obtidos com o Matlab. O circuito simulado possuía resistência de R=30 Ω, indutância de L=300mh e capacitância de C=50μF. Na figura 3.5.3.1

observa-se a tensão no capacitor. Não houve diferença significativa entre os resultados encontrados no Matlab e os implementados em C++.

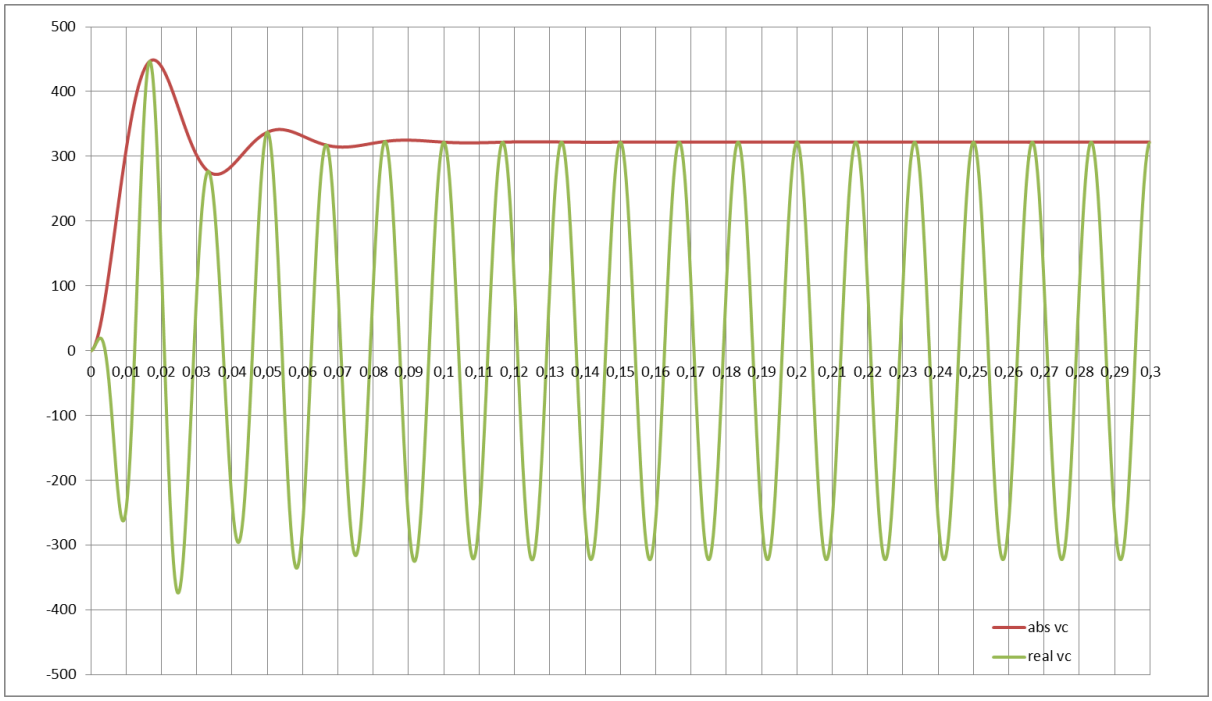


Figura 3.5.3-1 - Tensão no capacitor circuito RLC serie

3.5.4. CIRCUITO COM CHAVEAMENTO

No circuito da figura 3.5.4.1 serviu como para observar os efeitos dos chaveamentos nos circuitos. Esse circuito foi feito somente em C++ devido ao seu aumento de complexidade. A chave foi considerada ideal, ou seja ela não possui resistência. No caso estudado, o circuito é ligado com a chave S1 fechada. No instante de t=0,4 s a chave é aberta.

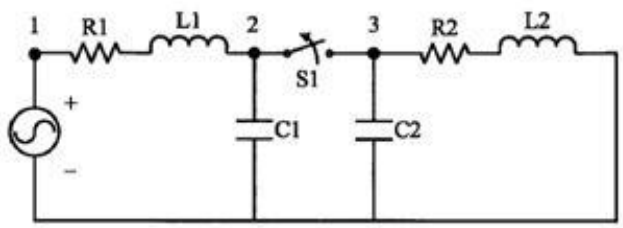


Figura 3.5.4-1 Circuito com chaveamento

Os parâmetros do circuito são os seguintes V_{rms} = 230 V, R1 = 3 Ω, L1 = 300mh, C1=20 µF, R2 = 50 Ω, L2 = 1 H e C2 =6 µF.

Na figura 3.5.4.2 fica clara a vantagem do uso do envelope para o seu uso por um operador de sistema uma vez que o comportamento da tensão na chave pode ser observado de forma mais clara do que através da ondas senoidais.

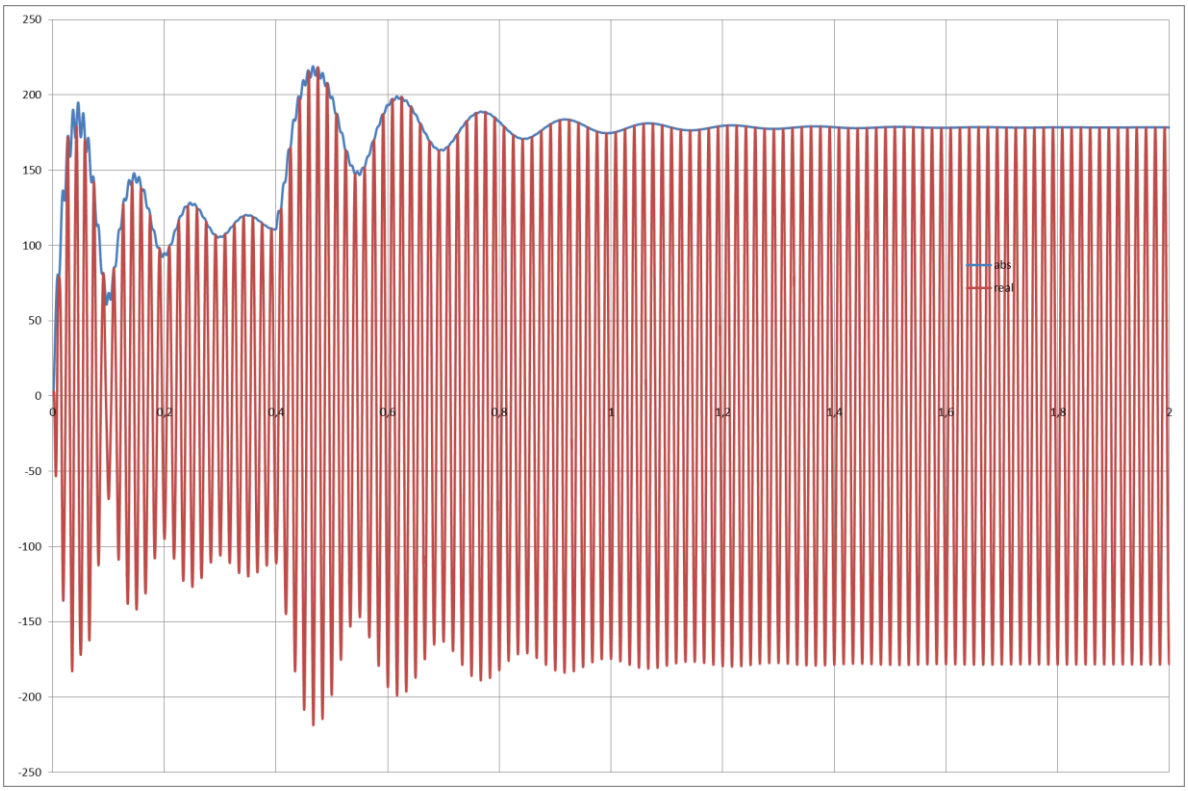


Figura 3.5.4-2 - Tensão na chave

3.5.5. CIRCUITO COM MÚLTIPLOS NÓS E CHAVEAMENTOS.

O circuito da figura 3.5.5.1 é a implementação completa do programa no qual há todos os módulos funcionando. O circuito inicialmente encontra-se com a chave entre 3 e 5 aberta e a chave entre 3 e 4 aberta doravante denominadas chave 2 e chave 1 respectivamente. As fontes são ligadas em t=0 No instante t=0,1s a chave 1 é aberta. No instante t=0,2 segundos a chave 1 e fechada. No instante t=0,3 a chave 2 é fechada e mantem-se assim até o final da simulação.

O passo utilizado foi de 0,5ms. A fonte de tensão teve o valor de V_{s1} = 1,05 V, a fonte de corrente I_{s1}= 1 A e a fonte de corrente I_{s2}= 1,5 A. Os valores do elementos passivos são indicados na figura.

Observa-se que no momento em que a chave 1 abre há uma grande oscilação no nó 1 devido a falta de elementos que atenuação.

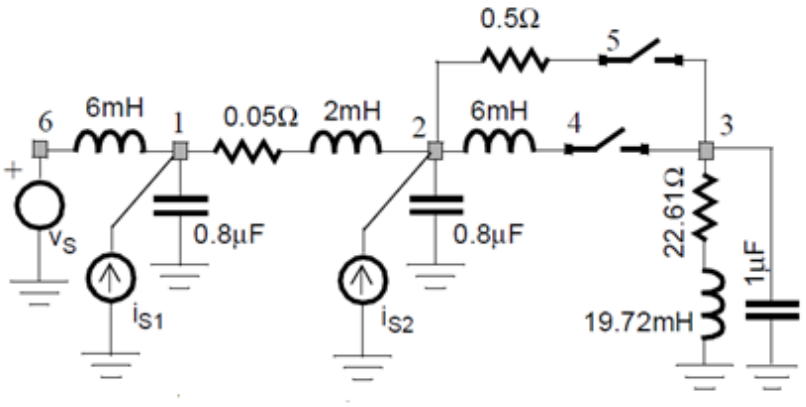


Figura 3.5.5-1 - Circuito de teste

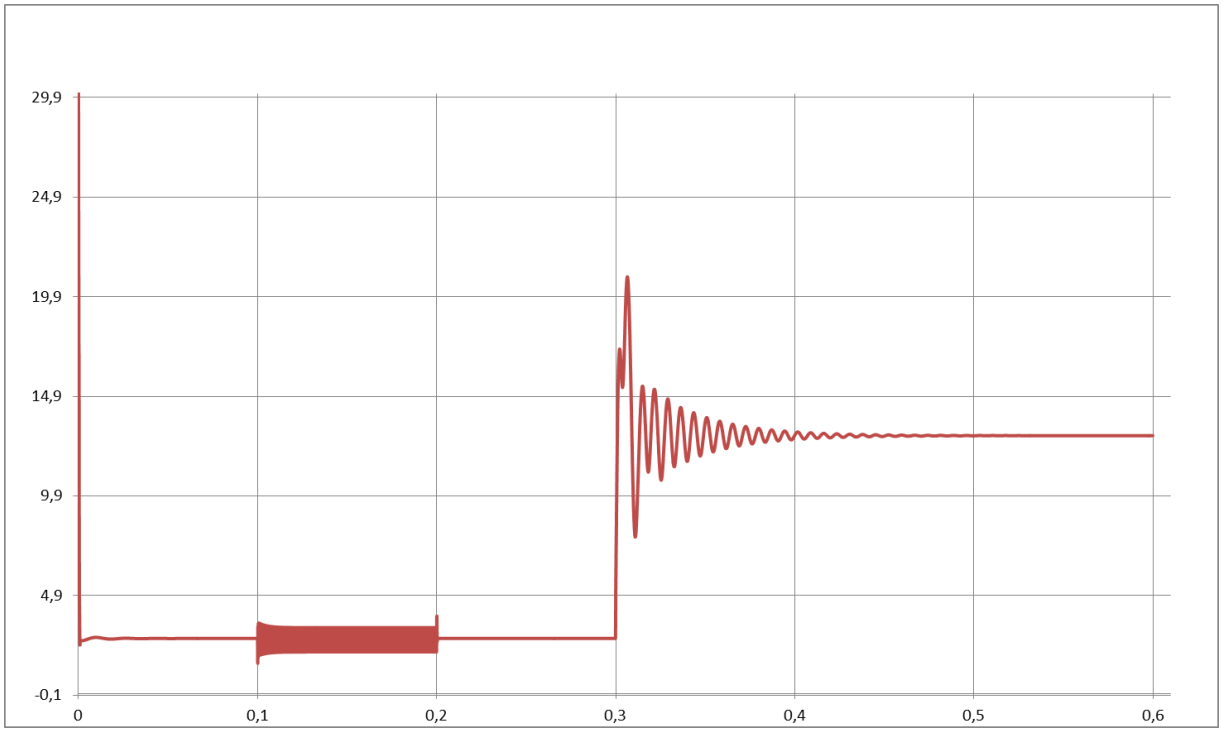


Figura 3.5.5-2 Tensão nó 1

Esse caso demonstra claramente as vantagens do uso do fasor dinâmico uma vez que consegue-se perceber o comportamento do circuito de maneira mais simples. Uma vez no qual se teria um grande número de ondas.

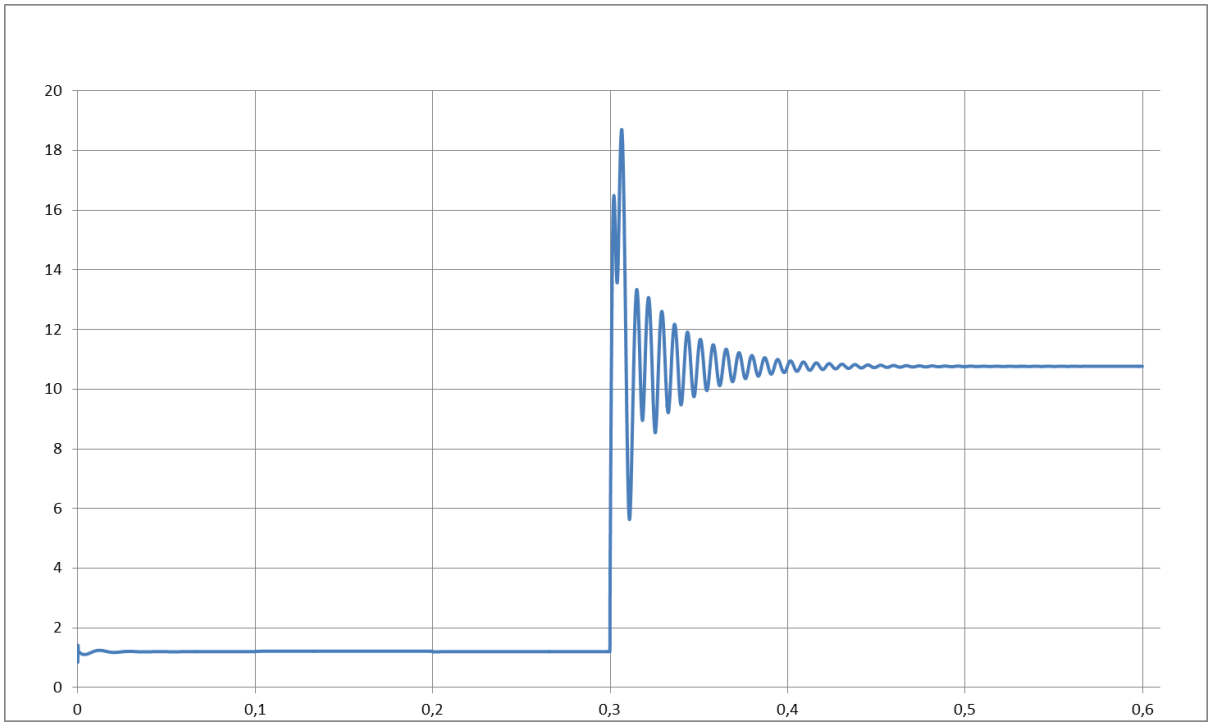


Figura 3.5.5-3 Tensão nó 2

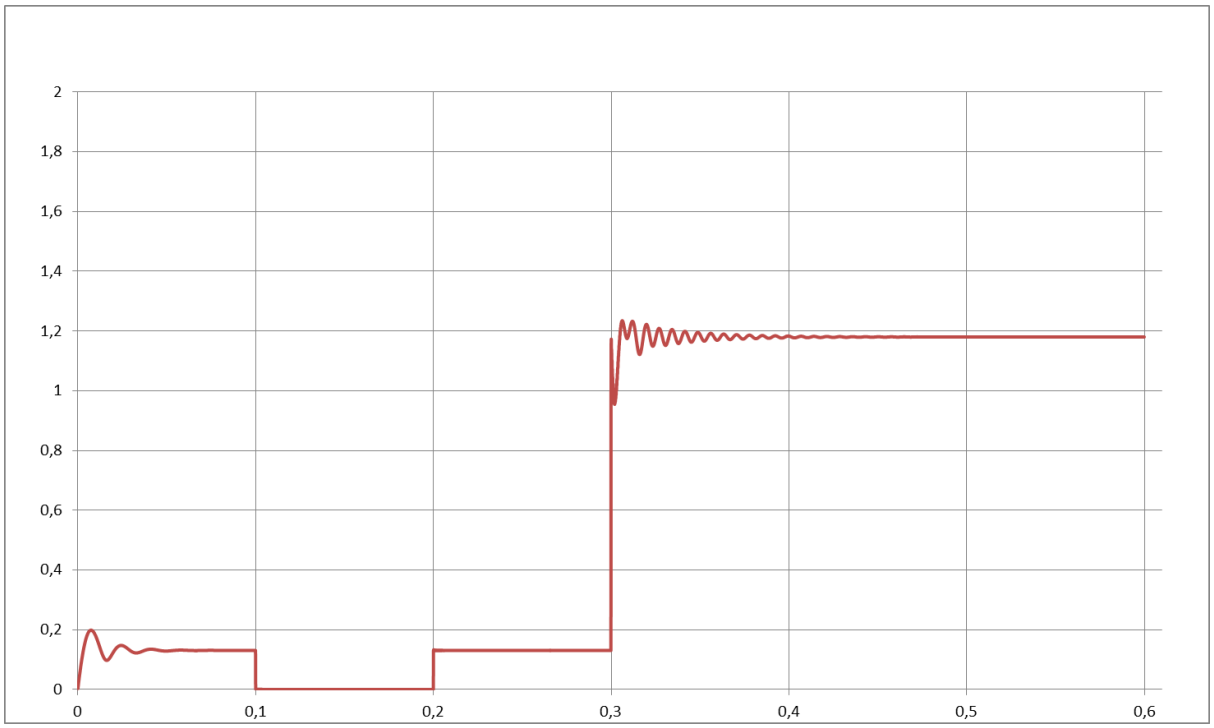


Figura 3.5.5-4 - Tensão nó 3

4. CONCLUSÃO

O principal objetivo para o uso do fasor dinâmico é a possibilidade de utilizar passos maiores de integração diminuindo o tempo total de simulação e fazer a sua integração como programa de simulação de transientes eletromagnéticos.

De uma forma geral a análise do deslocamento da frequência funcionou muito bem em todos os circuitos. Conseguiu-se usar passos maiores dos os usados comumente em programas de simulação de EMTP para a solução no domínio do tempo. A análise do deslocamento da frequência é completamente viável o seu uso como ferramenta de transitórios eletromagnéticos.

Apesar de se fazer uso de números complexos para a solução do sistema, aumentando o uso de memória e sendo necessária álgebra complexa não se transformou em um problema. O maior uso de memória pode ser compensado com alocação dinâmica da memória. O baixo número de interações comparado com o método do domínio no tempo compensa o aumento de memória demonstrando claras vantagens do método.

Mostra-se possível o desenvolvimento de uma ferramenta usando fasores dinâmicos para simulação de transientes em C++. Apesar de não ter conseguido programar com alocação dinâmica da memória, o seu uso é viável tornando assim mais eficiente o processo.

Para efeitos de estudo o programa conseguiu desempenhar a função para a qual ele foi projetado. Uma melhoria que poderia ser desenvolvida é a implementação de um motor gráfico para a extração de resultados como a ferramenta gnuplot.

O uso de passo variável traria grandes benefícios ao tempo de execução do programa. Durante os transitórios rápidos poderia se utilizar os passos já testados ou menores e depois aumentar o passo à medida que o sistema entra em regime permanente podendo se utilizar de passos de na ordem dos milissegundos. Por exemplo no caso executado nos primeiro 5 ms se utilizaria um passo menor e a medida que a solução caminha para o regime esse seria aumentado.

Sempre haverá melhorias por serem desenvolvidas para programas de computador. Por isso deve-se definir um escopo do será feito e atingir estes objetivos.

APÊNDICE A: CLASSES EM C++

Nesse apêndice está contido o código fonte das classes definidas em C++. Para facilitar o desenvolvimento foi definido que o tipo de variável complexa de tamanho Double foi criado uma definição de tipo chamada de dcomplex.

Também foi definido que a letra "J" maiúscula seria o número complexo.

A.1 Classes dos elementos Passivos.

```
typedef complex<double> dcomplex;
class passive_element {
    private:
    void calc_grl(void);
    void calc_gc(void);
    public:
    int busk, busm;
    void set_values(int,int,double,double,double);//contructor
    double r, l, c;
    dcomplex gc,grl;
    dcomplex hc,hl;
    void update_hc(dcomplex);
    void update_hl(dcomplex,dcomplex);
    dcomplex g_return(void);
};
void passive_element::set_values(int a, int b, double res, double ind,
double cap ){
    busk=a;
    busm=b;
    r=res;
    l=ind;
    c=cap;
    calc_gc();
    calc_grl();
    hc=0.;
    hl=0.;
};
void passive_element::calc_gc(void){
    gc=c*(2./step + J*ws);
};
void passive_element::calc_grl(void){
    dcomplex w;
    if ( (r==0) && (l==0) ) {
        grl=0.;
    }
    else if (l==0){
        grl=1/r;
    }
    else {
        w=( r + l*( 2./step + J*ws) );
        grl=1./w;
    };
};
```

```
void passive_element::update_hc(dcomplex vco){
    dcomplex hco;
    hco=hc;
    hc=4.*c*vco/step - hco;
    if ( c==0) hc=0;
};

void passive_element::update_hl(dcomplex vto,dcomplex ilo){
    dcomplex hlo;
    hlo=hl;
    hl=grl*((r-2.*l/step + J*ws*l )*grl - ilo )*vto - grl*(r-2.*l/step
+ J*ws*l)*hlo;
    if ( l==0 ) {
        hl=0.;
    };
};

dcomplex passive_element::g_return(){
    dcomplex g;
    g= grl+gc;
    return g;
};
```

A.2 Classe de fontes

```
// Fontes de Tensão tem valor são do tipo 0 e fontes de corrente são do
tipo 1
class source {
    private:
    double amp, phase, freq;
    public:
    int bus;
    bool s_type;
    dcomplex s_function(double);
    void set_values(int,bool,double,double,double);//constructor
};
dcomplex source::s_function(double t){
    dcomplex x;
    x=amp*exp(J*ws*t);
    return x;
};
void source::set_values( int bu, bool t, double a, double f, double
fr){
    amp=a;
    phase=f;
    s_type=t;
    freq=fr;
    bus=bu;
};
```

A.3 Classe de Chaves e Eventos

```
//chave do tipo normalmente aberto. aberto = 0 , fechado = 1
class sys_sw{
    public:
    int busk, busm;
    bool status;
```

```
    void switching(void);
    void set_values(int,int,bool);
    friend class events;
};
void sys_sw::set_values(int a,int b,bool c){
    busk=a;
    busm=b;
    status=c;
};
void sys_sw::switching(void){
        status= !status;
};
class events {
    private:
    bool sw_change;
    public:
    int k,m;
    double inst_sw;
    void set_time(int, int, double);
    bool change_status(sys_sw,int);
};
void events::set_time(int a, int b, double c){
    inst_sw=int(c/step);
    k=a;
    m=b;
    sw_change=1;
};
bool events::change_status(sys_sw x,int ti){
    bool a=0;
    if (ti==inst_sw) {
      if ( ( (x.busk==k) && (x.busm==m) )  ){
            x.switching();
            a=1;
      };
    };
    return a;
};
```

A.4 Classe de solução de sistema linear

```
class solver_lin_sis{
    int imax;
    vector<int> indx;
    int A_size;
    public:
    vector<dcomplex> LU_matrix;
    vector<dcomplex> x_vector;
    void decomp_lu(vector<dcomplex>,int);
    void solve_luxb(vector<dcomplex>);
    void show_lu(void);
};
void solver_lin_sis::decomp_lu(vector<dcomplex> A_matrix,int s_A){
    A_size=s_A;
    LU_matrix.assign(A_size*A_size,0.);
    LU_matrix=A_matrix;
    indx.assign(A_size,0);
    const double TINY=1.0e-40;
```

```
    int ii,jj,kk;
    double big,d;
    double vv[A_size];
    d=1.0;
    for (ii=0; ii<A_size; ii++) {
        double temp;
        big=0.0;
        for (jj=0; jj<A_size; jj++){
            temp = abs(LU_matrix[ii*A_size + jj]);
            if ( temp > big) big=temp;
        }
        vv[ii]=1.0/big;
    };
    for (kk=0; kk<A_size; kk++) {
        big=0.0;
        double temp;
        for (ii=kk; ii<A_size; ii++) {
            temp=vv[ii]*abs(LU_matrix[ii*A_size + kk]);
            if (temp > big) {
                big=temp;
                imax=ii;
            }
        }
        if (kk != imax) {
            for (jj=0; jj<A_size; jj++) {
                dcomplex temp;
                temp=LU_matrix[A_size*imax + jj];
                LU_matrix[imax*A_size+jj]=LU_matrix[kk*A_size+jj];
                LU_matrix[A_size*kk+jj]=temp;
            }
            d = -d;
           // cout << kk << " " << imax << "\n";
            vv[imax]=vv[kk];
        }
        indx[kk]=imax;
        for (ii=kk+1; ii<A_size; ii++) {
            dcomplex temp,temp1;
            temp=LU_matrix[ii*A_size+kk]/LU_matrix[kk*A_size+kk];
            LU_matrix[ii*A_size+kk]=temp;
            for (jj=kk+1; jj<A_size; jj++){
                temp1 = LU_matrix[ii*A_size+jj] -
temp*LU_matrix[kk*A_size+jj];
                LU_matrix[ii*A_size+jj]=temp1;
            }
        }
    };
};
void solver_lin_sis::solve_luxb(vector<dcomplex> b_vector){

    int i,ii=0,ip,j;
    dcomplex sum;
    x_vector=b_vector;
    for (i=0;i<A_size;i++) {
        ip=indx[i];
        sum=x_vector[ip];
        x_vector[ip]=x_vector[i];
        if (ii != 0) for (j=ii-1;j<i;j++) sum -=
LU_matrix[i*A_size+j]*x_vector[j];
```

```
        else if (sum != 0.0) ii=i+1;
        x_vector[i]=sum;
    };
    for (i=A_size-1;i>=0;i--) {
        sum=x_vector[i];
        for (j=i+1;j<A_size;j++) sum -=
LU_matrix[i*A_size+j]*x_vector[j];
        x_vector[i]=sum/LU_matrix[i*A_size+i];
    };
};
void solver_lin_sis::show_lu(void){
    cout << "\n"; // essa funcao ver a lu caso haja alguma necessidade
    for (int ii=0; ii<A_size; ii++){
        for (int jj=0; jj<A_size; jj++){
            cout << LU_matrix[ii*A_size +jj] << " ";
        };
        cout <<"\n";
    }
}
```

A.5 Classe sistema de Potência

```
#ifndef CLASSE_SISTEMA
#define CLASSE_SISTEMA
#include <iostream>
#include <fstream>
#include "solver.hpp"
#include <cstdlib>
#include "chave.h"
#include <string.h>

using namespace std;

class power_sys{
    private:
    passive_element elementos[10];
    source fontes[5];


    events eventos[10];
    int taman_data_ele, taman_data_sw,taman_data_eve;
    public:
    sys_sw chaves[5];
    solver_lin_sis sis_lin;
    int taman_gaa,taman_vsource,taman_data_source,taman_A;
    vector<dcomplex> matriz_A,vetor_I,vetor_V,matriz_gaa,matriz_gab;
    void read_elements(void);
    void read_sources(void);
    void read_switches(void);
    void read_events(void);
    void triangulize_g(void);
    void mount_gaa();
    void mount_gab();
    void mount_i(double);
    void update_h(double);
```

```
    void check_sw(double);
    void solve(void);
};

void power_sys::read_elements(void){
    ifstream ele("elementos.txt");
    if (!ele) cout << "Cannot open file elementos.txt\n";
    int a,b;
    double resis,induc,capac;
    taman_gaa=0;
    for (int ii=0; ele.good() && !ele.eof(); ii++){
        ele >> a >> b >> resis >> induc >> capac;
        elementos[ii].set_values(a,b,resis,induc,capac);
       // cout << "Linha"<< ii<< " - barra "<< a << " barra "<< b << "
R=" << resis <<" I="<< induc << " C="<< capac << "\n";
        taman_data_ele=ii;
        if(a>taman_gaa) {
            taman_gaa=a;
        };
        if (b>taman_gaa){
            taman_gaa=b;
        };
    };
    //cout <<":"<< taman_gaa << "\n";

    ele.close();
};

void power_sys::read_sources(void){
    ifstream sou("fontes.txt");
    if (!sou) cout << "Cannot open file fontes.txt\n";
    int a;
    double ampli, phase, fr;
    char ti[8];
    bool tipe_source;
    taman_vsource=0;
    taman_data_source=0;
    for (int ii=0; sou.good() && !sou.eof(); ii++){
        sou >> a >> ti >> ampli >> phase >> fr;
        tipe_source= strcmp(ti,"VOLTAGE");
        if (!tipe_source) taman_vsource++;
        fontes[ii].set_values(a,tipe_source,ampli,phase,fr);
        taman_data_source++;
    };
    sou.close();
};
void power_sys::read_switches(void){
    ifstream ch("chaves.txt");
    taman_data_sw=0;
    if (!ch) cout << "Cannot open file fontes.txt\n";
    for (int ii=0; ch.good()&& !ch.eof(); ii++){
        int a,b;
        char ti[8];
        bool status;
        ch >> a >> b >> ti;
        status= strcmp(ti,"ABERTO");
        chaves[ii].set_values(a,b,status);
        taman_data_sw++;
```

```
    };
    taman_data_sw--;
    ch.close();
};

void power_sys::read_events(void){
    ifstream eve("eventos.txt");
    taman_data_eve=0;
    for (int ii=0; eve.good() && !eve.eof() ; ii++ ){     // leitura
dos eventos
        int a,b;
        char ti[8];
        double inst;
        eve >> a >> b >> inst >> ti;
                eventos[ii].set_time(a,b,inst)
               taman_data_eve++;
    };
    taman_data_eve--;
    eve.close();
};


void power_sys::mount_gaa(void){
    int size1;
    taman_A=taman_gaa-taman_data_sw-taman_vsource;
    size1=taman_A*taman_A;
    matriz_gaa.assign(size1,0.);
    for (int ii=0; ii<taman_data_ele; ii++){
        int bk,bm,bs;
        bk=elementos[ii].busk-1;
        bm=elementos[ii].busm-1;
        bs=elementos[ii].busm;
        if (bm<taman_A){
            matriz_gaa[bk*taman_A+bk]=matriz_gaa[bk*taman_A+bk] +
elementos[ii].g_return();
            if (bs!=0){
                matriz_gaa[bm*taman_A+bm]=matriz_gaa[bm*taman_A+bm] +
elementos[ii].g_return();
                matriz_gaa[bk*taman_A+bm]=matriz_gaa[bk*taman_A+bm] -
elementos[ii].g_return();
                matriz_gaa[bm*taman_A+bk]=matriz_gaa[bm*taman_A+bk] -
elementos[ii].g_return();
            }

        };
}
for (int ii=0;ii<taman_data_ele; ii++){
        if (!fontes[ii].s_type){
            for (int jj=0; jj<taman_data_ele; jj++){
                if (fontes[ii].bus==elementos[jj].busm){
                    int bk;
                    bk=elementos[jj].busk;
                    matriz_gaa[bk*taman_A + bk]=matriz_gaa[bk*taman_A +
bk] + elementos[jj].g_return();
                }
            }
        }
```

```
    };

    for (int jj=0; jj<taman_data_ele; jj++){
        for (int ii=0; ii<taman_data_sw;ii++){
            int bk,bm,bs,bkk;
            bk=chaves[ii].busk;
            bm=chaves[ii].busm;
            if (elementos[jj].busm==bm){
                cout << "chaves";
                bs=elementos[jj].busk-1;
                if (chaves[ii].status){
                    bkk=bk-1;
                    matriz_gaa[bkk*taman_A +
bkk]=matriz_gaa[bkk*taman_A + bkk] + elementos[jj].g_return();
                    matriz_gaa[bs*taman_A + bkk]=matriz_gaa[bs*taman_A
+ bkk] - elementos[jj].g_return();
                    matriz_gaa[bkk*taman_A + bs]=matriz_gaa[bkk*taman_A
+ bs] - elementos[jj].g_return();
                }
                matriz_gaa[bs*taman_A + bs]=matriz_gaa[bs*taman_A + bs]
+ elementos[jj].g_return();
            }
        }

    };

};
void power_sys::mount_gab(void){
    int size1;
    size1=taman_vsource*taman_A;
    matriz_gab.assign(size1,0.);
    for (int ii=0;ii<taman_data_source;ii++){
        if (!fontes[ii].s_type){
            int bm;
            bm=fontes[ii].bus;
            for (int jj=0;jj<taman_data_ele;jj++){
                if(elementos[jj].busm==bm){
                    int bs,bk;
                    bk=elementos[jj].busk-1;
                    bs=bm-taman_A - taman_data_sw;
                    matriz_gab[bk*taman_A+bs]=
matriz_gab[bk*taman_A+bs] -elementos[jj].g_return();
                }
            }
        }
    }
};

void power_sys::mount_i(double ti){
    vetor_I.clear();
    vetor_I.assign(taman_A,0.);
    for (int ii=0; ii<taman_data_source; ii++){
        int jj=0;
        if (!fontes[ii].s_type){
            for (int kk=0; kk<taman_A; kk++){

vetor_I[kk]=vetor_I[kk]+matriz_gab[jj*taman_A+kk]*fontes[ii].s_function
(ti);
```

```
            };
            jj++;
        }
    };

    for (int ii=0; ii<taman_data_ele;ii++){
        for (int jj=0; jj<taman_A; jj++){
            if (elementos[ii].busk==jj+1){
                vetor_I[jj]=vetor_I[jj]+ elementos[ii].hc +
elementos[ii].hl;
            }
            if (elementos[ii].busm==jj+1){
                vetor_I[jj]=vetor_I[jj]+ elementos[ii].hc -
elementos[ii].hl;
            }
        }
    }
    for (int ii=0; ii<taman_data_source; ii++){
        if (fontes[ii].s_type){
            int bk=fontes[ii].bus;
            vetor_I[bk-1]=vetor_I[bk-1]+fontes[ii].s_function(ti);
        };
    };
    for (int ii=0; ii<taman_data_sw; jj++){
       if (chaves[ii].status){
           for (int jj=0; jj<taman_data_ele; jj++){
                if (elementos[jj].busm==chaves[ii].busm){
                      int bk,bs;
                      bk=chaves[ii].busk-1;
                      vetor_I[bk]=vetor_I[bk] - elementos[jj].hl -
elementos[jj].hc;
                }
           }
     }
    };

};
void power_sys::triangulize_g(void){
    sis_lin.decomp_lu(matriz_gaa,taman_A);
    matriz_gaa.clear();
};

void power_sys::update_h(double){
    for (int ii=0;ii<taman_data_ele; ii++){
        dcomplex v,ia,g;
        v=0.;
        int bk,bm;
        bk=elementos[ii].busk-1;
        bm=elementos[ii].busm-1;
        if (elementos[ii].busm<taman_A){
            if (elementos[ii].busm==0){
                  v=vetor_V[bk];
            } else { v=vetor_V[bk]-vetor_V[bm];
            }
        };
        for (int jj=0; jj<taman_data_sw; jj++){
            if (elementos[ii].busm==chaves[jj].busm){
                int bk,bs;
```

```
                bk=elementos[ii].busk;
                bs=chaves[jj].busk;
                if (chaves[jj].status ) v=vetor_V[bk]-vetor_V[bs];
            }
        for (int jj=0;jj<taman_data_sources; jj++ ){
            if (!fontes[jj].s_type){
                if (elementos[ii].busk=fontes[jj].bus)
v=fontes[jj].s_function(ti) - vetor_V;
            }
        };
        g=elementos[ii].g_return();

         ia=v*(g);
        if (elementos[ii].c!=0){
            elementos[ii].update_hc(v);
        } else{
            elementos[ii].update_hl(v,ia);
        };
    };
};


void power_sys::check_sw(int ti){
    bool juca;
    for (int ii=0;ii<taman_data_eve; ii++){
        if (eventos[ii].inst_sw==ti){
            for (int jj=0; jj<taman_data_sw; jj++){
                juca=eventos[ii].change_status(chaves[jj],ti);

            }
        }
    }
    if ( juca ) {
        mount_gaa();
        triangulize_g();
    };
};

void power_sys::solve(){
    sis_lin.solve_luxb(vetor_I);
    vetor_V=sis_lin.x_vector;
};
```

APÊNDICE B SCRIPTS EM MATLAB

B.1 Scripts de solução dos circuitos.

Os script de Matlab basicamente seguiam o mesmo principio todos havendo apenas alterações aonde fosse necessário. Nos casos das fontes com componentes harmônicas apenas foi inserido o termo referente a componente harmônica.

```
clear
v=325.27; % tensão de 230 Volts RMS
time=.3;
step=1e-3;
r=300;
l=.3;
c=0.00005;
simtime=time/step;
ws=2*pi*60;
hco=0;
hlo=0;
rl=r+(2/step + j*ws)*l; %cálculo dos parametros do sistema
gc=(2/step + j*ws)*c;
grl=1/rl;
hl=0; %inicialização dos termos históricos
hc=0;
for t=1:simtime
      vt=v*exp(j*ws*step*t);
      va= (-vt*grl - hl + hc)/(grl + gc);
      vl=(vt-va);
      il=vl*grl;
      hlo=hl;
      hco=hc;
      hl=grl*((r-2*l/step + j*ws*l)*grl - il)*vl - grl*(r-2*l/step +
j*ws*l)*hlo;
      hc=4/step*c*va - hco;
      vf.time(1,t)=t*step;
      vg(1,t)=vt;
      vcap(1,t)=va;
      ib(1,t)=il;
end
plot(vf.time,abs(ib));
xlabel('Tempo (s)')
ylabel('Tensão (V)')
grid
```

B.2 Script para cálculo da frequência instantânea.

Para o cálculo da frequência instantânea foi utilizado o seguinte script onde vt2 é o sinal complexo importado do programa, time é o tempo de simulação e step é o passo do programa.

```
phase_vt2=angle(vt2);
u_phase_vt2=unwrap(phase_vt2);
inst_freq= diff(u_phase)/(step*2*pi);
s_time=size(time) -1;
plot(time(1:s_time,inst_freq)
grid
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] A. M. Stankovic, T. Aydin, “Analysis of asymmetrical faults in Power Systems using Dynamic Phasors,” IEEE Trans. Power Systems, vol. 15, no. 3, Aug. 2000, pp. 1062-1068

[2] M. D. Ortigueira, “Processamento Digital de Sinais”, Fundação Calouste Gulbenkian, 2005

[3] A. Oppenheim, R. Schafer, J. Buck, “Discrete-time signal processing”, 2nd Ed., Prentice-Hall, 1999

[4] Wikipedia, “Analytic signal”, http:// http://en.wikipedia.org/wiki/Analytic_signal

[5] Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP. Numerical Recipes: The Art of Scientific Computing. 3ª Edição New York: Cambridge University Press. 2007 p. 48-56 ISBN 0521308119.

[6] R. Bracewell, “Fourier Transforms and its applications”, 3rd.Ed., Mac-Graw Hill, 2000

[7] E. Weisstein, “Hilbert Transform”, from MathWorld –A Wolfram Web Resource, http://mathworld.wolfram.com/HilbertTransform.html

[8] MathWorks, “Signal processing toolbox documentation,” disponível em http://www.mathworks.com/help/toolbox/signal/ref/hilbert.html

[9] S. Marple, “Computing the Discrete-Time “Analytic” Singal via FFT”, IEEE Trans. Signal Processing, vol. 47, no. 9, sept. 1999

[10] SCHILDT, Hebert – C++ The Complete Reference, 3ª edição - McGraw-Hill Osborne , 1998 ISBN-10: 0078824761

[11] ZHANG, Peng. Shifted Frequency Analysis for EMTP Simulation of Power System Dynamics. Tese (Doutorado em engenharia elétrica) – The University of Britsh Columbia. Vancouver: 2009.

[12] HENSCHEL, Sebastian. Analysis of electromagnetic and electromechanical power system transients with dynamic phasors. Tese (Doutorado em engenharia elétrica) – The University of Britsh Columbia. Vancouver: 1999.

[13] W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[1] GOLD, B., Oppenheim, A.V.,RADER, C. M. Theory and Implementation of Discrete Hilbert transform. Proc. Symp. Comput Process Commun. P.235-250 NY Polythecnic Press 1970.

[2] HENSCHEL, Sebastian. Analysis of electromagnetic and electromechanical power system transients with dynamic phasors. Tese (Doutorado em engenharia elétrica) – The University of Britsh Columbia. Vancouver: 1999.

[3] ZHANG, Peng. Shifted Frequency Analysis for EMTP Simulation of Power System Dynamics. Tese (Doutorado em engenharia elétrica) – The University of Britsh Columbia. Vancouver: 2009.

[4] SCHILDT, Hebert – C++ The Complete Reference, 3ª edição - McGraw-Hill Osborne , 1998 ISBN-10: 0078824761

[5] Press, WH; Teukolsky, SA; Vetterling, WT; Flannery, BP. Numerical Recipes: The Art of Scientific Computing. 3ª Edição New York: Cambridge University Press. 2007 p. 48-56 ISBN 0521308119.