UNIVERSIDADE FEDERAL DE VIÇOSA
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

IGOR LOPES COSTA RAMALHO

DETERMINAÇÃO DA RESPOSTA DE UMA FUNÇÃO DE TRANSFERÊNCIA A ENTRADAS UNITÁRIAS UTILIZANDO LABFUN

VIÇOSA
2019

IGOR LOPES COSTA RAMALHO

DETERMINAÇÃO DA RESPOSTA DE UMA FUNÇÃO DE TRANSFERÊNCIA A ENTRADAS UNITÁRIAS UTILIZANDO LABFUN

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 – Monografia e Seminário – e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. André Gomes Tôrres.

VIÇOSA
2019

IGOR LOPES COSTA RAMALHO

DETERMINAÇÃO DA RESPOSTA DE UMA FUNÇÃO DE TRANSFERÊNCIA A ENTRADAS UNITÁRIAS UTILIZANDO LABFUN

Monografia apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e Tecnológicas da Universidade Federal de Viçosa, para a obtenção dos créditos da disciplina ELT 490 - Monografia e Seminário e cumprimento do requisito parcial para obtenção do grau de Bacharel em Engenharia Elétrica.

Aprovada em 25 de junho de 2019.

COMISSÃO EXAMINADORA

Prof. Dr. André Gomes Torres - Orientador
Universidade Federal de Viçosa

Prof. Dr. Denílson Eduardo Rodrigues - Membro
Universidade Federal de Viçosa

Profª.Dra. Kétia Soares Moreira - Membro
Universidade Federal de Viçosa

“Sonhe grande. Ter um sonho grande dá o mesmo trabalho de ter um sonho pequeno.”

(Jorge Paulo Lemann)

Dedico este trabalho a minha mãe, Maria Aparecida.

Agradecimentos

Primeiramente gostaria de agradecer a minha mãe, Maria Aparecida, pelo apoio incondicional e incentivo prestados não somente durante os anos de universidade, mas sim durante toda minha vida. Agradeço a minha irmã, Juliana, minha avó, Agripina, e aos familiares que acompanharam todos os passos dessa trajetória.

Ao meu orientador, André Gomes Tôrres, por todo o apoio e direcionamento dado ao longo da elaboração deste trabalho.

Aos meus amigos, antigos e aos novos que fiz durante os anos de graduação, por compartilharem momentos incríveis comigo.

A diElétrica e a todos os seus membros por todas as experiências e conhecimento adquirido. Conhecimentos esses, que sempre levarei comigo e que contribuíram de forma decisiva para minha formação.

E por fim, agradeço a Universidade Federal de Viçosa, seu corpo docente e administração que tão bem me receberam e que, através de suas diversas iniciativas, proporcionaram uma caminhada repleta de aprendizado, oportunidades e amizades.

Resumo

O presente trabalho mostra a implementação de um *software*, intitulado LabFUN - Laboratório de análise de funções de transferência, dentro da interface de programação Microsoft Visual Studio, em linguagem C#, para realizar a caracterização e determinação da curva de saída de um sistema de controle genérico modelado por uma função de transferência. Para efeito de análise, foram consideradas entradas unitária (impulso, degrau e rampa). Além disso, por se tratar de um dos primeiros trabalhos relacionados a programação orientada a objetos dentro do Departamento de Engenharia Elétrica da Universidade Federal de Viçosa, o corpo do texto possui uma forte presença de elementos de definição relacionados ao assunto e a temas correlatos.

Para realizar a construção do programa foram desenvolvidas equações matemáticas, baseadas nos conceitos de função de transferência e posteriormente foi realizada a readequação e implementação em ambiente de desenvolvimento. O resultado obtido foi uma versão executável do programa que agrega, em interface gráfica, os principais parâmetros para realizar a caracterização de um sistema estável em malha aberta e a exibição do gráfico da curva de saída para cada uma das entradas consideradas.

Palavras-chave: Programação orientada a objeto, Microsoft Visual Studio, C#, função de transferência e entradas unitárias.

Lista de variáveis e símbolos

G(s)	Função de transferência do sistema
X(s)	Transformada de Laplace da entrada do sistema
Y(s)	Transformada de Laplace da saída do sistema
$\mathcal{L}\{\}$	Operador da Transformada de Laplace
x(t)	Entrada do sistema no domínio do tempo
y(t)	Saída do sistema no domínio do tempo
A	Coeficiente de proporcionalidade de *G(s)*
B	Coeficiente quadrático do denominador de *G(s)*
C	Coeficiente linear do denominador de *G(s)*
D	Coeficiente independente do denominador de *G(s)*
$\delta(t)$	Impulso unitário no domínio do tempo
e	Número de Euler
$u(t)$	Degrau unitário no domínio do tempo
$r(t)$	Rampa unitária no domínio do tempo
ω_n	Frequência natural não amortecida
ξ	Coeficiente de amortecimento
K_1	Constante de redefinição de coeficiente linear
K_2	Constante de redefinição de coeficiente independente
$s_{1,2}$	Polos da função de transferência *G(s)*
σ	Parte real dos polos complexos
ω_d	Parte real dos polos complexos (Frequência natural amortecida)

Sumário

Lista de Figuras

1 Introdução

Um *software* consiste em uma sequência estruturada de instruções que, quando executadas, fornecem características, funções e o desempenho desejado. São estruturas de dados que possibilitam aos programas manipular informações adequadamente (PRESSMAN, 2011).

O mundo moderno não poderia existir sem o *software*. Infraestruturas e serviços nacionais são controlados por sistemas computacionais, e a maioria dos produtos elétricos inclui um computador e um software que o controla. A manufatura e a distribuição industriais são totalmente informatizadas, assim como o sistema financeiro. A área de entretenimento, incluindo a indústria da música, jogos de computador, cinema e televisão, faz uso intensivo de *software* (SOMMERVILLE, 2011).

Nos anos 60 nasceu a programação estruturada. Esse é o método estimulado por linguagens como C e Pascal. Usando-se linguagens estruturadas, foi possível, pela primeira vez, escrever programas moderadamente complexos de maneira razoavelmente fácil. Entretanto, com a programação estruturada, quando um projeto atinge um certo tamanho, ele torna-se incontrolável, pois a sua complexidade excede a capacidade dessa programação (UNIOESTE, 2014).

1.1 Programação orientada a objetos

O conceito predominante de programação antes de Programação orientada a objetos (POO) era a chamada programação procedural. Consistia basicamente em dividir a tarefa de programação em pequenos blocos de código chamados de procedimentos (*procedures*, em inglês), também conhecidos na época como sub-rotinas. Em todos os casos, o que se fazia, basicamente, era escrever um trecho de código que manipulasse os valores de algumas variáveis e desse algum tipo de retorno.

Primeiramente, expandiu-se a idéia de procedimento para a idéia de classe. Uma classe permite que vários procedimentos e dados sejam armazenados dentro dela. Os procedimentos passaram a chamar-se métodos e os dados passaram a chamar-se propriedades (LIMA; REIS,

2002). Constituem-se assim, as bases para a criação do que hoje conhecemos como Programação orientada a objetos.

1.1.1 Objetos e classes

Objetos são instâncias de classes, que determinam qual informação um objeto contém e como ele pode manipulá-la.

Um objeto é um elemento que representa, no domínio da solução, alguma entidade (abstrata ou concreta) do domínio de interesse do problema sob análise. Objetos similares são agrupados em classes. No paradigma de orientação a objetos, tudo pode ser potencialmente representado como um objeto. Sob o ponto de vista da POO, um objeto não é muito diferente de uma variável normal.

Uma classe é um gabarito para a definição de objetos. Através da definição de uma classe, descreve-se que propriedades — ou atributos — o objeto terá. Além da especificação de atributos, a definição de uma classe descreve também qual o comportamento de objetos da classe, ou seja, que funcionalidades podem ser aplicadas a objetos da classe (RICARTE, 2001).

1.1.2 Pilares da POO

Para se entender exatamente do que se trata a orientação a objetos, se tona necessário esclarecer quais são os requerimentos de uma linguagem para ser considerada nesse paradigma. Para isso, a linguagem precisa atender a quatro tópicos bastante importantes: Abstração, Encapsulamento, Herança e Polimorfismo (DEVMEDIA, 2014).

1.1.2.1 Abstração

A abstração consiste em um dos pontos mais importantes dentro de qualquer linguagem Orientada a Objetos. Como o foco é lidar com uma representação de um objeto real (o que dá nome ao paradigma), tem-se que imaginar o que esse objeto irá realizar dentro do sistema. São três pontos que devem ser levados em consideração nessa abstração: identidade, propriedades e métodos.

O primeiro ponto é a identidade e constitui a forma como o objeto é nomeado dentro do contexto de programação. A segunda parte diz respeito a características do objeto. No mundo

real qualquer objeto possui elementos que o definem. Dentro da programação orientada a objetos, essas características são nomeadas propriedades. Por fim, o terceiro ponto consiste em definir as ações que o objeto irá executar. Essas ações, ou eventos, são chamados métodos (DEVMEDIA, 2014).

1.1.2.2 Encapsulamento

Encapsulamento é o princípio de projeto pelo qual cada componente de um programa deve agregar toda a informação relevante para sua manipulação como uma unidade (uma cápsula). Aliado ao conceito de ocultamento de informação, é um poderoso mecanismo da programação orientada a objetos.

Na orientação a objetos, o uso da encapsulação e ocultamento da informação recomenda que a representação do estado de um objeto deve ser mantida oculta. Cada objeto deve ser manipulado exclusivamente através dos métodos públicos do objeto, dos quais apenas a assinatura deve ser revelada. O conjunto de assinaturas dos métodos públicos da classe constitui sua interface operacional.

Dessa forma, detalhes internos sobre a operação do objeto não são conhecidos, permitindo que o usuário do objeto trabalhe em um nível mais alto de abstração, sem preocupação com os detalhes internos da classe. Essa facilidade permite simplificar a construção de programas com funcionalidades complexas, tais como interfaces gráficas ou aplicações distribuídas (RICARTE, 2001).

1.1.2.3 Herança

O conceito de encapsular estrutura e comportamento em um tipo não é exclusivo da orientação a objetos; particularmente, a programação por tipos abstratos de dados segue esse mesmo conceito. O que torna a orientação a objetos única é o conceito de herança.

Herança é um mecanismo que permite que características comuns a diversas classes sejam fatoradas em uma classe base, ou superclasse. A partir de uma classe base, outras classes podem ser especificadas. Cada classe derivada ou subclasse apresenta as características (estrutura e métodos) da classe base e acrescenta a elas o que for definido de particularidade para ela (RICARTE, 2001).

1.1.2.4 Polimorfismo

Na natureza, vemos animais que são capazes de alterar sua forma conforme a necessidade, e é dessa ideia que vem o polimorfismo na orientação a objetos. Como sabemos, os objetos filhos herdam as características e ações de seus "ancestrais". Entretanto, em alguns casos, é necessário que as ações para um mesmo método sejam diferentes (DEVMEDIA, 2014).

Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse. Esse mecanismo é fundamental na programação orientada a objetos, permitindo definir funcionalidades que operem genericamente com objetos, abstraindo-se de seus detalhes particulares quando esses não forem necessários (RICARTE, 2001).

1.2 Programação multithread

Nas arquiteturas de sistemas operacionais mais antigas, o próprio processo constituía a unidade de execução. Nos sistemas operacionais mais modernos, já a partir do OS/2 e do Windows NT, a unidade de execução é uma *thread*, ou linha de execução. A *thread* constitui um subprocesso, ou processo de peso leve que visa dar maior grau de concorrência as atividades que um processo deve executar.

Todo processo possui pelo menos uma *thread*, denominada *thread* primária. O código da *thread* primária corresponde ao programa principal. Outras rotinas de um programa podem ser executadas como *threads* de forma concorrente ao programa principal, sendo criadas dinamicamente pela *thread* primária (SEIXAS FILHO; SZUSTER, 2003).

As *threads* possibilitam uma implementação relativamente simples do conceito de programação simultânea. Durante décadas, a simultaneidade foi possível, mas difícil. Softwares concorrentes eram difíceis de escrever, depurar e manter. Como resultado, muitos desenvolvedores escolheram o caminho mais fácil e evitaram a simultaneidade.

No entanto, com as bibliotecas e recursos de idiomas disponíveis para os modernos programas .NET, a simultaneidade é muito mais fácil. Atualmente, todo desenvolvedor pode (e deve) abraçar a simultaneidade (CLEARY, 2014).

1.3 Curva de resposta

A função de transferência de um sistema representado por uma equação diferencial linear invariante no tempo é definida como a relação entre a transformada de Laplace da saída (função de resposta – *response function*) e a transformada de Laplace da entrada (função de excitação – *driving function*), admitindo-se todas as condições iniciais nulas (OGATA, 2011). Assumindo *X(s)* e *Y(s)* como as transformadas de Laplace da entrada *x(t)* e da saída *y(t)*, respectivamente, de um sistema genérico como o representado na Figura 1 a função de transferência G(s) será dada por:

$$Função\ de\ transferência = G(s) = \frac{\mathcal{L}\{y(t)\}}{\mathcal{L}\{x(t)\}} = \frac{Y(s)}{X(s)} \quad (1)$$

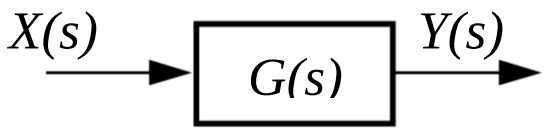

Figura 1 - Diagrama de blocos de um sistema genérico.

1.3.1 Resposta ao impulso

O impulso unitário $\delta(t)$, também chamado de função delta de Dirac, é definido pela expressão (2) (ZILL, 2016), sendo seu gráfico mostrado na Figura 2.

$$\delta(t) = \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases} \quad (2)$$

Realizando a transformada de Laplace, da expressão acima, tem-se:

$$\mathcal{L}\{\delta(t)\} = 1 \quad (3)$$

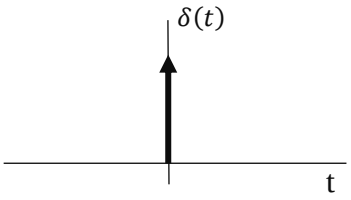

Figura 2 – Gráfico da função impulso unitário.

Considere a saída (resposta) de um sistema linear invariante no tempo a um impulso unitário de entrada quando as condições iniciais são nulas. Como a transformada de Laplace da função impulso unitário é igual a unidade, a transformada de Laplace da saída do sistema é dada pela equação:

$$Y(s) = G(s) \tag{4}$$

A função transferência e a função de resposta impulsiva de um sistema linear invariante no tempo contêm as mesmas informações sobre a dinâmica de um sistema. Dessa maneira, é possível obter informações completas sobre as características dinâmicas do sistema, por meio da excitação por um impulso de entrada e medindo a resposta (OGATA, 2011).

1.3.2 Resposta ao degrau

A função degrau frequentemente é usada quando operações de chaveamento sobre fontes DC estão envolvidas. Além disso, várias outras funções singulares podem ser dela deduzidas a partir de operações como integrações e derivações sucessivas (HIGUTI; KITANO, 2003). A função é definida pela expressão (5), sendo seu gráfico mostrado na Figura 3.

$$u(t) = \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases} \tag{5}$$

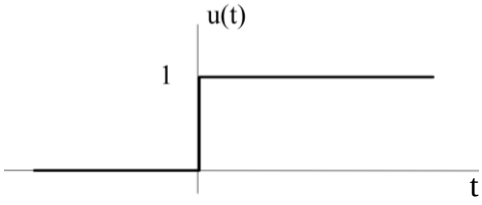

Figura 3 – Gráfico da função degrau unitário (HIGUTI; KITANO, 2003).

Considere agora, uma entrada no formato de degrau unitário para o mesmo tipo de sistema abordado no tópico anterior. Como a transformada de Laplace da função degrau unitário

é igual a 1/s, a transformada de Laplace da saída do sistema é dada pela expressão (OGATA, 2011):

$$Y(s) = \frac{G(s)}{s} \quad (6)$$

1.3.3 Resposta a rampa

A função rampa unitária $r(t)$ é definido pela expressão (7), sendo seu gráfico mostrado na Figura 4.

$$r(t) = \begin{cases} t, & t \geq 0 \\ 0, & t < 0 \end{cases} \quad (7)$$

Realizando a transformada de Laplace, da expressão acima, tem-se:

$$\mathcal{L}\{\delta(t)\} = \frac{1}{s^2} \quad (8)$$

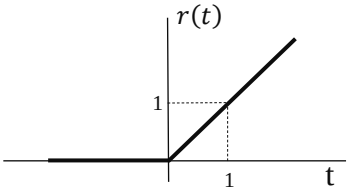

Figura 4 – Gráfico da função rampa unitária.

Considere, por fim, uma entrada no formato de rampa unitária para o mesmo tipo de sistema abordado nos tópicos anteriores. Como a transformada de Laplace da função rampa unitária é igual a $1/s^2$, a transformada de Laplace da saída do sistema é dada por (OGATA, 2011):

$$Y(s) = \frac{G(s)}{s^2} \quad (9)$$

1.4 Microsoft Visual Studio

O Microsoft Visual Studio é um ambiente de desenvolvimento integrado (IDE) da Microsoft Corporation para desenvolvimento de software especialmente dedicado ao .NET Framework e as linguagens C#, C++, JavaScript, Python, Visual Basic (VB), F# (F Sharp), TypeScript (TS) e R. Além do editor e do depurador padrão fornecidos pela maioria dos IDEs,

o Visual Studio inclui compiladores, ferramentas de preenchimento de código, designers gráficos e muitos outros recursos para facilitar o processo de desenvolvimento de software (MICROSOFT, 2019).

1.5 Linguagem de programação – C#

A linguagem C# (pronuncia-se C *Sharp*) faz parte de um conjunto de ferramentas oferecidas na plataforma .NET e surge como uma linguagem simples, robusta, orientada a objetos, fortemente tipada e altamente escalável a fim de permitir que uma mesma aplicação possa ser executada em diversos dispositivos de hardware, independentemente destes serem PCs, handhelds ou qualquer outro dispositivo móvel. Além do mais, a linguagem C# também tem como objetivo permitir o desenvolvimento de qualquer tipo de aplicação: *Web service*, aplicação Windows convencional, aplicações para serem executadas num *palmtop* ou *handheld*, aplicações para Internet, etc (LIMA; REIS, 2002).

1.6 MATLAB

MATLAB® é uma abreviatura de "*matrix laboratory*". O *software* é um produto da empresa norte americana MathWorks. Milhões de engenheiros e cientistas em todo o mundo usam o MATLAB® para analisar e projetar os sistemas e produtos que transformam nosso mundo. A linguagem de programação, baseada em matriz, é a maneira mais natural de expressar matemática computacional. O programa disponibiliza um conjunto de bibliotecas de funções matemáticas, geração de gráficos e manipulação de dados, oferecendo suporte a trabalhos relacionados a análise de dados, processamento de sinais, finanças quantitativa, gerenciamento de risco, robótica e sistemas de controle (MATHWORKS, 2019).

No presente trabalho, o MATLAB® é utilizado como ferramenta auxiliar na ilustração, por meio de gráficos, do modelo matemático desenvolvido e como parâmetro de referência para comparação de resultados.

1.7 Objetivo Geral

Este trabalho tem como objetivo principal implementar um programa de computador no Microsoft Visual Studio, em linguagem C#, para realizar a caracterização e definição da curva de resposta de sistemas de controle estáveis modelados por meio de uma função transferência.

Dado o objetivo geral, têm-se como objetivos específicos:

- Apresentar os principais conceitos referentes a programação orientada a objeto;
- Definir escopo do *software*, métodos, modelos e classes aplicáveis;
- Implementar o programa no Microsoft Visual Studio;
- Desenvolver uma interface gráfica de interação com usuário.

2 Metodologia

Para que o LabFUN – Laboratório de análise de funções de transferência, *software* alvo do presente trabalho, pudesse ser implementado se fez necessário realizar uma modelagem matemática e a sua posterior readequação em ambiente de desenvolvimento, sendo que esta tem como pré-requisitos iniciais respeitar as condições e limitações da linguagem escolhida (C#) e da própria IDE (Microsoft Visual Studio).

2.1 Modelagem matemática

A modelagem matemática da solução apresentada consiste em desenvolver um conjunto de equações genéricas que representem o comportamento de saída de um sistema a cada uma das entradas unitárias consideradas.

2.1.1 Função de primeira ordem

Seja um sistema de 1° ordem com Função de Transferência:

$$G(s) = \frac{Y(s)}{X(s)} = \frac{A}{Cs+D} \quad (10)$$

2.1.1.1 Resposta ao impulso

Para uma entrada no formato de impulso unitário:

$$x(t) = \delta(t) \Rightarrow X(s) = 1 \quad (11)$$

Assim, substituindo o valor de X(s) na equação (10):

$$Y(s) = \frac{A}{1.(Cs+D)} = \frac{A}{C}\frac{1}{\left(s+\frac{D}{C}\right)} \quad (12)$$

Aplicando a inversa de Laplace:

$$y(t) = \frac{A}{D}e^{-\frac{D}{C}t}, \text{ para } t \geq 0 \quad (13)$$

A curva de resposta para uma entrada impulso unitário, pode ser observada na Figura 5. Note que na figura T = D/C = D/A.

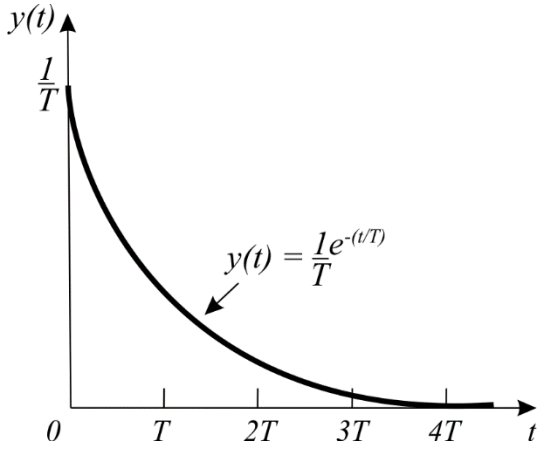

Figura 5 - Curva de resposta para entrada impulso unitário (OGATA, 2011).

2.1.1.2 Resposta ao degrau

Para uma entrada no formato de degrau unitário:

$$x(t) = u(t) \Rightarrow X(s) = \frac{1}{s} \quad (14)$$

Assim, substituindo o valor de X(s) na equação (10):

$$Y(s) = \frac{A}{s(Cs+D)} = \frac{A}{C}\frac{1}{s\left(s+\frac{D}{C}\right)} \quad (15)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{D}\left[1 - e^{-\frac{D}{C}t}\right], \quad \text{para t} \geq 0 \quad (16)$$

A curva de resposta para uma entrada degrau unitário, pode ser observada na Figura 6. Note que na figura a amplitude em estado estacionário é unitária e T= D/C.

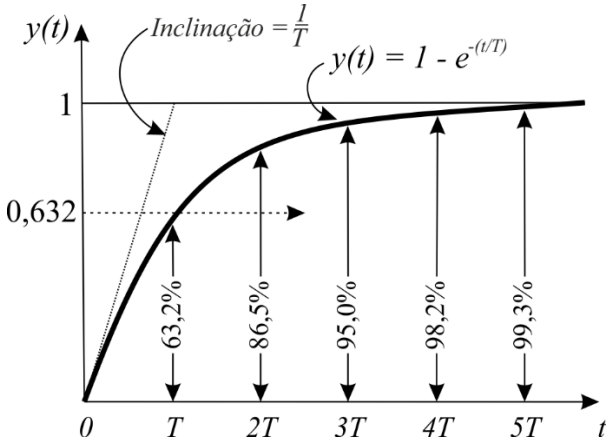

Figura 6 - Curva exponencial de resposta para entrada degrau unitário (OGATA, 2011).

2.1.1.3 Resposta a rampa

Para uma entrada no formato de rampa unitária:

$$x(t) = r(t) \Rightarrow X(s) = \frac{1}{s^2} \quad (17)$$

Assim, substituindo o valor de X(s) na equação (10):

$$Y(s) = \frac{A}{s^2(Cs+D)} = \frac{A}{C}\frac{1}{s^2\left(s+\frac{D}{C}\right)} \quad (18)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{D}\left[t - \frac{C}{D} + \frac{1}{D}e^{-\frac{D}{C}t}\right], \text{ para t} \geq 0 \quad (19)$$

A curva de resposta para uma entrada rampa unitária, pode ser observada na Figura 7 e assim como no item anterior T= D/C.

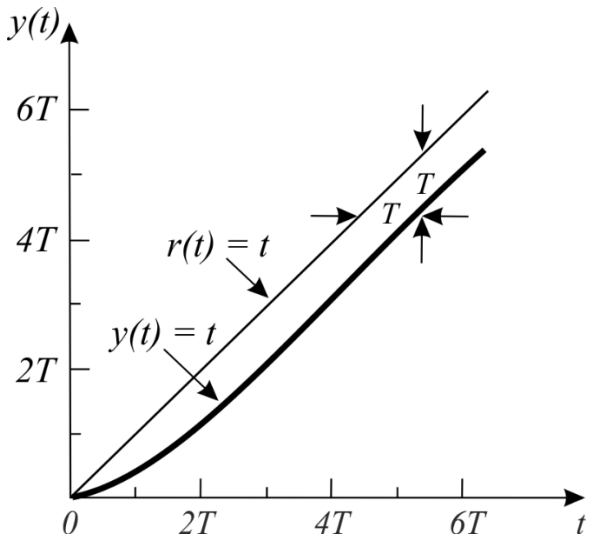

Figura 7 - Curva de resposta para entrada rampa unitária (OGATA, 2011).

2.1.2 Função de segunda ordem

Seja um sistema de 2° ordem com Função de Transferência:

$$G(s) = \frac{Y(s)}{X(s)} = \frac{A}{Bs^2+Cs+D} \quad (20)$$

Reescrevendo a equação (20):

$$G(s) = \frac{Y(s)}{X(s)} = \frac{A}{Bs^2+Cs+D} = \frac{A}{B}\frac{1}{s^2+K_1s+K_1} = \frac{A}{B\omega_n{}^2}\frac{\omega_n{}^2}{s^2+2\xi\omega_n s+\omega_n{}^2} \quad (21)$$

onde

$$K_1 = \frac{C}{B} = 2\xi\omega_n, \;\; K_2 = \frac{D}{B} = {\omega_n}^2 \;\; e \;\; \omega_n > 0 \qquad (22)$$

Analisando a localização dos polos do sistema:

$$s^2 + 2\xi\omega_n s + {\omega_n}^2 = 0 \qquad (23)$$

Calculando os polos da equação:

$$s_{1,2} = \frac{-2\xi\omega_n \pm \sqrt{4\xi^2{\omega_n}^2 - 4{\omega_n}^2}}{2} = \omega_n\left(-\xi \pm \sqrt{\xi^2 - 1}\right) \qquad (24)$$

Em todos os problemas de controle, o requisito fundamental a ser atendido é a estabilidade do sistema, o que se traduz pela necessidade de que os polos do sistema se situem no semi-plano esquerdo (S.P.E.). Esta região pode ser observada na Figura 8.

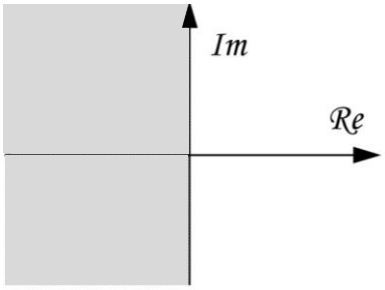

Figura 8 - Diagrama de localização de polos de um sistema estável.

Tendo em vista este fato, há três casos a considerar:

- $0 < \xi < 1 \Rightarrow \; s_1 \; e \; s_2$ são complexos conjugados (subamortecido)
- $\xi = 1 \Rightarrow \; s_1 = s_2$ são reais e iguais (criticamente amortecido)
- $\xi > 1 \Rightarrow \; s_1 \neq s_2$ são reais e distintos (superamortecido)

Às demais possibilidades quanto aos valores de ξ correspondem sempre a existência de sistemas instáveis (quando $\xi < 0$) ou sem amortecimento (quando $\xi = 0$).

2.1.2.1 Resposta ao impulso

Para uma entrada no formato de um impulso unitário, pela equação (11), tem-se que a transforma de Laplace, indicada por *X(s)*, da entrada é igual a um. Assim, substituindo o valor de *X(s)* na equação (21):

$$Y(s) = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{1.(s^2+2\xi\omega_n s+\omega_n{}^2)} \quad (25)$$

2.1.2.1.1 Primeiro caso: 0 < ξ < 1 – Sistema subamortecido

Nesse caso, os polos são complexos conjugados:

$$s_{1,2} = -\omega_n\xi \pm j.\,\omega_n\sqrt{1-\xi^2} = -\sigma \pm j\omega_d \quad (26)$$

onde: ω_n = frequência natural não amortecida
ω_d = frequência natural amortecida
ξ = coeficiente de amortecimento

Reescrevendo a equação (25) em função dos polos complexos:

$$Y(s) = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{s^2+2\xi\omega_n s+\omega_n{}^2} = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{(s+\sigma+j\omega_d)(s+\sigma-j\omega_d)} \quad (27)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{B\omega_n{}^2}\left[\frac{1}{\sqrt{1-\xi^2}} e^{-\xi\omega_n t}.\,sen\left(\omega_n\sqrt{1-\xi^2}.\,t\right)\right],\ \text{para t} \geq 0 \quad (28)$$

2.1.2.1.2 Segundo caso: ξ = 1 – Sistema criticamente amortecido

Nesse caso, os polos são reais, negativos e iguais:

$$s_{1,2} = -\omega_n < 0 \quad (29)$$

Reescrevendo a equação (25) em função dos polos:

$$Y(s) = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{s^2+2\xi\omega_n s+\omega_n{}^2} = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{(s+\omega_n)^2} \quad (30)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{B\omega_n{}^2}\omega_n{}^2 te^{-\omega_n t},\ \text{para t} \geq 0 \quad (31)$$

2.1.2.1.3 Terceiro caso: ξ > 1 – Sistema superamortecido

Nesse caso, os polos são reais, negativos e distintos:

$$s_{1,2} = \omega_n\left(-\xi \pm \sqrt{\xi^2-1}\right) < 0 \quad (32)$$

Reescrevendo a equação (25) em função dos polos:

$$Y(s) = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{s^2+2\xi\omega_n s+\omega_n{}^2} = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{(s-s_1)(s-s_2)} \quad (33)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{B\omega_n{}^2} \frac{\omega_n}{2\sqrt{\xi^2-1}} \left[e^{-\left(\xi-\sqrt{\xi^2-1}\right)\omega_n t} - e^{-\left(\xi+\sqrt{\xi^2-1}\right)\omega_n t} \right], \text{ para t} \geq 0 \qquad (34)$$

A curva de resposta para uma entrada impulso unitário obtida no MATLAB®, para diferentes valores de ξ pode ser observada na Figura 9. Para fins de plotagem foram utilizados os valores: A=10, B=1 e $\omega_n = 3$ rad/s.

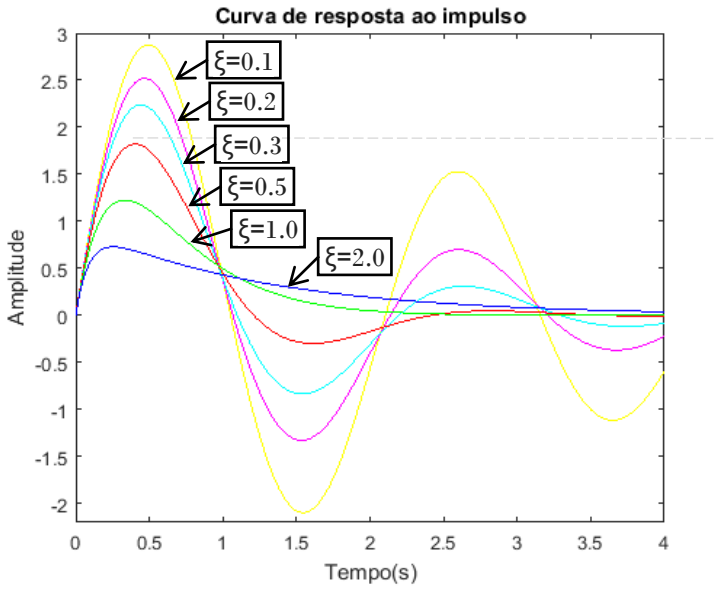

Figura 9 - Curva de resposta para entrada impulso unitário em função de ξ.

2.1.2.2 Resposta ao degrau

Para uma entrada no formato de um degrau unitário, pela equação (14), tem-se que a transforma de Laplace, indicada por *X(s)*, da entrada é igual a 1/s. Assim, substituindo o valor de *X(s)* na equação (21):

$$Y(s) = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{s(s^2+2\xi\omega_n s+\omega_n{}^2)} \qquad (35)$$

2.1.2.2.1 Primeiro caso: 0 < ξ < 1 – Sistema subamortecido

Nesse caso, os polos são complexos conjugados:

$$s_{1,2} = -\omega_n\xi \pm j.\omega_n\sqrt{1-\xi^2} = -\sigma \pm j\omega_d \qquad (36)$$

Reescrevendo a equação (35) em função dos polos:

$$Y(s) = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{s(s^2+2\xi\omega_n s+\omega_n{}^2)} = \frac{A}{B\omega_n{}^2} \frac{\omega_n{}^2}{s(s+\sigma+j\omega_d)(s+\sigma-j\omega_d)} \qquad (37)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{B{\omega_n}^2}\left[1 - \frac{1}{\sqrt{1-\xi^2}}e^{-\sigma t}.sen\left(\omega_d t + acos(\xi)\right)\right], \quad \text{para t} \geq 0 \qquad (38)$$

2.1.2.2.2 Segundo caso: $\xi = 1$ – Sistema criticamente amortecido

Nesse caso, os polos são reais, negativos e iguais:

$$s_{1,2} = -\omega_n < 0 \qquad (39)$$

Reescrevendo a equação (35) em função dos polos:

$$Y(s) = \frac{A}{B{\omega_n}^2}\,\frac{{\omega_n}^2}{s(s^2+2\xi\omega_n s+{\omega_n}^2)} = \frac{A}{B{\omega_n}^2}\,\frac{{\omega_n}^2}{s(s+\omega_n)^2} \qquad (40)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{B{\omega_n}^2}\,[1 - (1+\omega_n t)e^{-\omega_n t}], \text{ para t} \geq 0 \qquad (41)$$

2.1.2.2.3 Terceiro caso: $\xi > 1$ – Sistema superamortecido

Nesse caso, os polos são reais, negativos e distintos:

$$s_{1,2} = \omega_n\left(-\xi \pm \sqrt{\xi^2 - 1}\right) < 0 \qquad (42)$$

Reescrevendo a equação (35) em função dos polos:

$$Y(s) = \frac{A}{B{\omega_n}^2}\,\frac{{\omega_n}^2}{s(s^2+2\xi\omega_n s+{\omega_n}^2)} = \frac{A}{B{\omega_n}^2}\,\frac{{\omega_n}^2}{s(s-s_1)(s-s_2)} \qquad (43)$$

Decompondo em frações parciais e aplicando a inversa de Laplace:

$$y(t) = \frac{A}{B{\omega_n}^2}\left[1 + \frac{\omega_n}{2\sqrt{\xi^2-1}}\left(\frac{e^{s_2 t}}{s_2} - \frac{e^{s_1 t}}{s_1}\right)\right], \text{ para t} \geq 0 \qquad (44)$$

A curva de resposta para uma entrada degrau unitário obtida no MATLAB®, para diferentes valores de ξ pode ser observada na Figura 10. Para fins de plotagem foram utilizados os valores: A=10, B=1 e $\omega_n = 3$ rad/s.

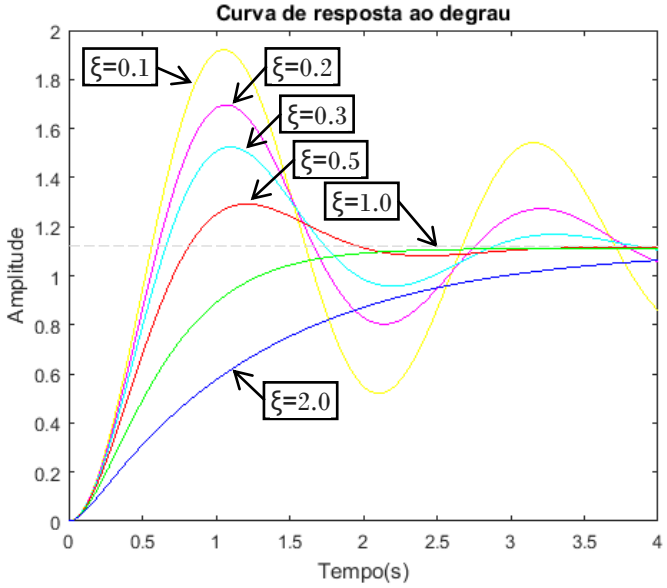

Figura 10 - Curva de resposta para entrada degrau unitária em função de ξ.

2.2 Estrutura do código

O código do software foi escrito em C# e foi estruturado em duas *threads*, sendo que os elementos do corpo do código se dividem em quatro grandes categorias: cálculo de parâmetros, controle de labels e botões, gráfico de resposta de saída e interface gráfica.

2.2.1 Definição de *threads*

Dentro do contexto do problema, a escolha de duas *threads* (a principal e uma secundária) foi feita tento em vista um melhor aproveitamento dos recursos computacionais de processamento e principalmente como método antitravamento.

A *thread* principal foi direcionada para a declaração e inicialização de variáveis globais, controle de labels e botões e instruções de chamada da *thread* secundária. Já a *thread* secundária é focada na realização dos cálculos e na plotagem do gráfico de reposta de saída.

2.2.2 Gráfico de resposta de saída

O gráfico de resposta de saída foi plotando tomando como base as equações de respostas de saída *y(t)* desenvolvidas na seção 2.1. Para definição em código foi utilizada a biblioteca de gráficos dinâmicos 2D *ZedGraph* (versão 5.1.7).

O *ZedGraph* é uma biblioteca de classes, controle de usuário e controle da *Web* para .net, escrito em C #, para desenhar gráficos de linha, barra e pizza 2D. Ele apresenta recursos de personalização completos e detalhados, mas a maioria das opções tem padrões para facilidade de uso. A biblioteca é adicionada através da opção gerenciar pacotes *Nuget* na janela do gerenciador de soluções, sendo que essa adição é vinculada ao a projeto através da criação de um novo elemento da caixa de ferramentas chamado *zedGraphControl.*

2.2.3 Interface gráfica

A interface gráfica do software foi construída dentro da janela de Design do Microsoft Visual Studio utilizando os elementos nativos da interface de programação e/ou das bibliotecas adicionadas. Foram utilizados os elementos *bottons (*botões), *labels* (linhas de texto), *textBox* (caixas de texto), *radioButton* (botões de seleção), *pictureBox* (caixa de imagem), *panels (*painéis de agregação) e *zedGraphControl* (área de plotagem da biblioteca *ZedGraph*). O *design* final da interface pode ser observado na Figura 11 e o mapeamento de elementos utilizados na construção do software na tabela 1.

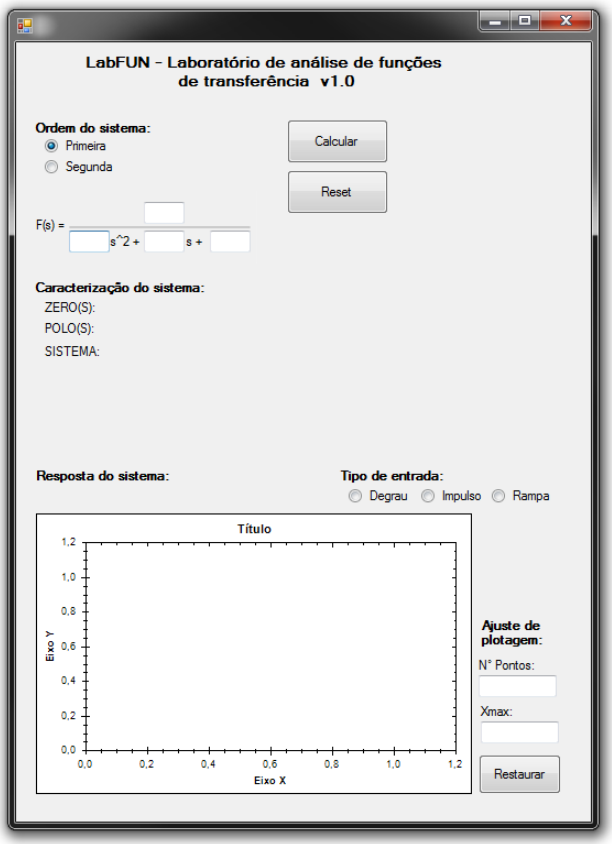

Figura 11 - *Design* da interface do usuário.

Tabela 1 – Mapeamento de elementos de construção do *software*.

Superclasse	**Classe**	**Identidade**	**Função**
Imput (entrada de informações)	Seleção	button1	Botão calcular
		button2	Botão reset
		button3	Botão restaurar escalas
		radioButton1	Botão de seleção: 1° ordem
		radioButton2	Botão de seleção: 2° ordem
		radioButton4	Botão de seleção: Degrau
		radioButton5	Botão de seleção: Impulso
		radioButton6	Botão de seleção: Rampa
	Textos e números	textBox2	Coeficiente de proporcionalidade
		textBox3	Coeficiente quadratico de F(s)
		textBox4	Coeficiente linear de F(s)
		textBox5	Coeficiente independente de F(s)
		textBox6	Deterniação do N° de pontos
		textBox7	Deterniação do X máximo
Output (exibição de informações)	Textos e números	label1	Titulo do programa
		label2	Indição de ordem
		label3	Indição de F(s)
		label5	Indicação s^2
		label4	Indicação de ajuste de plotagem
		label6	Indicação s+
		label7	Indicação de zeros
		label8	Indicação de polos
		label9	Indicação de sistema
		label10	Exibição de zeros
		label11	Exibição de polos
		label12	Exibição da caracterização do sistema
		label13	Indicação de seção
		label14	Indicação de frequência
		label15	Exibição de frequência
		label16	Indicação de coef. Amortecimento
		label17	Exibição de coef. Amortecimento
		label18	Indicação de seção
		label19	Indicação de seção
		label20	Indicação de tipo de entrada
		label21	Indicação de número de pontos
		label22	Indicação de x máximo
	Imagens	pictureBox1	Imagem de linha para função
		zedGraphControl1	Interface de plotagem

Superclasse	Classe	Identidade	Função
Output (exibição de informações)	Organização interna	panel1	Painel de ordem de função
		panel2	Painel de caracterização de sistema
		panel3	Painel de parâmetros
		panel4	Painel de resposta do sistema
		panel5	Painel de tipo de entrada

No que diz respeito as propriedades, foram preservadas a maioria das caractericas preestabelecidas. Os nomes de exibição foram alterados nos botões e labels, as dimensões nas caixas de texto e imagem, *zedgraph* e painéis. Além disso, algumas linhas de texto tiveram alterações nas suas propriedades relacionadas ao campo de fonte (tipo de fonte, tamanho e estilo).

Já os métodos, foram implementados através da manipulação dos do recurso e funções nativas do Visual Studio. Considerando o código implementado, existem diversas interações entre os objetos e métodos, porém pode-se destacar seis métodos: calcular, plotar gráfico, intertravemento de grau de função, intertravemento de função de entrada, ajuste de plotagem e resetar.

2.2.3.1 Método calcular

Método implementado pelo botão calcular. Este método está alocado na *thread* principal e tem como principal objetivo realizar os direcionamentos para realizar o cálculo dos valores de interesse, controle de variáveis auxiliares e exibição de parâmetros. Além disso, este método "chama" o método de plotagem de gráfico.

2.2.3.2 Método plotar gráfico

Este método pode ser considerado como o elemento central programa. Ele é o responsável efetivo por realizar os cálculos e por plotar o gráfico de resposta de saída. Este método interage diretamente com as caixas de texto de entrada de coeficientes, botões de seleção de ordem do sistema e de tipo de entrada (objetos de entrada de informação) e com o *zedGraphControl* (objeto de saída de informação).

2.2.3.3 Método intertravamento de grau de função

A escolha de grau da função é uma seleção pré-cálculo e como tal independe dos métodos de cálculo ou plotagem de gráfico e não possui interação direta com os demais objetos de entrada/saída de informação. O intertravamento de item se dá através da interação entre os

radioButton (botões de seleção) destinados a escolha de grau da função de transferência. Além da possibilidade de seleção mutualmente excludente implementada, a seleção de uma opção (primeiro ou segundo grau) altera a propriedade *visible* das linhas de texto (*label*) destinadas a indicar os parâmetros do sistema, sendo que essa opção somente se aplica a funções de transferência de segunda ordem.

2.2.3.4 Método intertravamento função de entrada

Esse segundo método de intertravamento atua de forma semelhante ao anterior. Existe a mesma questão de seleção mutualmente excludente, porém nesse caso também há uma interação direta com o *zedGraphControl*, que se caracteriza como um objeto de exibição de informação. A seleção do tipo de entrada é utilizada como um elemento de direcionamento de código no corpo do método de plotagem de gráfico e a alteração do tipo de entrada também é utilizada como um meio de "forçar a chamada" do método de plotagem. Essa característica foi implementada com o objetivo de proporcionar ao usuário a possibilidade de interação dinâmica com gráfico de saída, para os diferentes tipos de entrada consideradas.

2.2.3.5 Método ajuste de plotagem

O método interage com as caixas de texto de entrada de valores de referência e tem como objetivo proporcionar um possível reajuste para melhor visualização da curva de saída. A atualização é realizada de forma dinâmica a cada nova atualização de valores (pelo botão calcular ou pela alteração do tipo de sinal de entrada).

Como funcionalidade secundaria este método também apresenta uma segunda forma de chamada através do botão restaurar. Porém nesse caso, são restaurados os valores preestabelecidos para o número de pontos utilizados (20 pontos) e a variação desejada no eixo horizontal do gráfico (0 a 15). O ajuste da ordenada (eixo vertical) é realizado de forma automática pela função *zedGraphControl.RestoreScale* implementada dentro do método de plotagem de gráfico.

2.2.3.6 Método resetar

O método de reset é implementado pelo botão Reset e tem como função fazer com que o programa retorne a seu estágio inicial. O método interage com todos os objetos da tela de usuário e atua desfazendo as seleções de botões, limpando a superfície de plotagem e restaurando os valores do campo de ajuste de plotagem.

2.3 Interação com usuário

A interação do usuário com a interface se dá seguinte forma: inicialmente são solicitados como dados de entrada a seleção da ordem da função (primeiro ou segundo), os coeficientes da função de transferência e o tipo de entrada (degrau, impulso ou rampa). Ao pressionar o botão calcular o programa realiza os cálculos e exibe os polos, zeros, caracterização e parâmetros do sistema e o gráfico de resposta de saída.

Analisando interface, pode-se observar a presença dos campos de seleção do grau da função (primeiro ou segundo), caracterização do sistema em função dos polos, parâmetros do sistema (frequência não amortecida e coeficinte de amortecimento), seleção do tipo de entrada e o gráfico de resposta de saída.

Com o objetivo de otimizar o uso do programa e facilitar a visualização da curva de resposta, foram implementadas opções de ajuste de plotagem (no canto inferior direito da Figura 11), onde o usuário pode reajustar, caso seja necessário, o número de pontos e margem de variação da abscissa utilizados na construção do gráfico. Além disso, a interface também conta com um botão de *Reset* caso se deseje zerar o valor dos parâmetros de entrada e reiniciar a utilização do programa.

3 Resultados e Discussão

A partir da metodologia apresentada, foi desenvolvida uma interface de usuário contendo todos os elementos necessários para a avaliação preliminar do sistema em análise. A Figura 12 mostra a tela de usuário para uma função transferência de segundo grau com polos complexos.

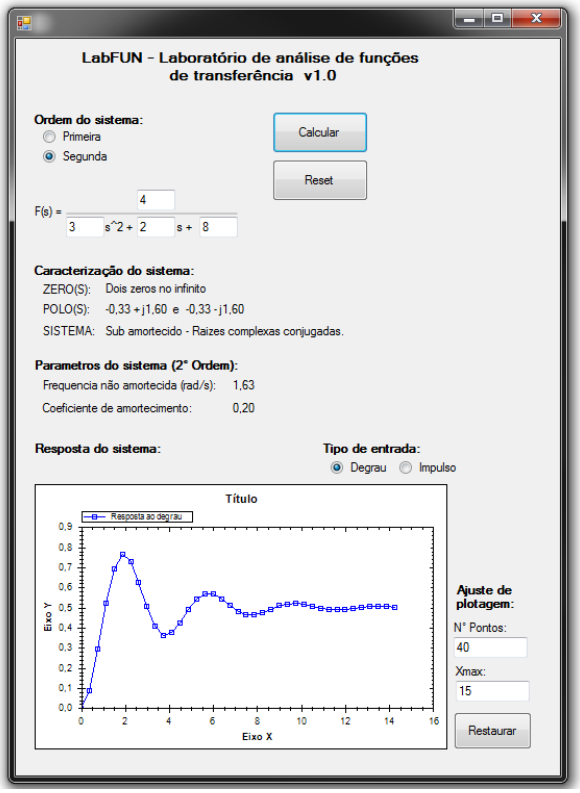

Figura 12 - Tela do usuário para uma função de transferência de segundo grau.

3.1 Análise de elementos no contexto da POO

Referenciando os itens utilizados na construção do software com os tópicos abordados relacionados a programação orientada a objetos, pode-se perceber que o programa desenvolvido apresenta uma evidente utilização do conceito de encapsulamento no que tange a restrição de itens manipuláveis pelo usuário, diponibilizando apenas a alteração de valores e/ou seleção entre opções predefinidas. Já a abstração se dá nos próprios elementos e em sua caracterização, dentro do corpo do código todos possuem uma identidade, propriedades e métodos bem definidos.

Os conceitos de herança e polimorfismo não são utilizados diretamente na construção do programa devido ao reduzido número de classes criadas. Porém, existem intrinsicamente na própria concepção da interface do Microsoft Visual Studio e sendo assim são utilizados de forma indireta. O uso da herança é caracterizada pelas diversas propriedades de uma classe base (superclasse) que são compartilhadas entre os elementos específicos utilizados (classes derivadas - elementos da janela caixa de ferramentas) e o polimorfismo pelas propriedades específicas de uma classe derivada (características da janela propriedades) que possuem um comportamento diferente, de forma nativa ou não, da superclasse que a originou.

3.2 Análise comparativa e parâmetros de desempenho

Comparando o cálculo de valores obtidos utilizando o LabFUN e o MATLAB para a mesma função considerada na Figura 12 obtêm-se um índice de correlação global igual a 0,9983 e um erro médio percentual de 1,98% para as 15 amostras avaliadas, sendo que para efeito de cálculo desses valores foram consideradas 16 casas decimais. A tabela 2 e o gráfico da Figura 13 apresentam os valores considerados na comparação.

Tabela 2 – Comparação numérica entre valores do LabFUN e do MATLAB

Amostra	**Tempo**	**Valor LabFUN**	**Valor MATLAB**	**Variação**
1	0	0,00	0	0
2	0,5	0,145860122625622	0,141649138970619	0,004211
3	1,0	0,447709519468422	0,435288922325577	0,012420
4	1,5	0,693131694149377	0,680388380189465	0,012743
5	2,0	0,761366308110577	0,759287575765171	0,002078
6	2,5	0,664618365797987	0,676794209742582	-0,012176
7	3,0	0,503565753725006	0,522890044044031	-0,019324
8	3,5	0,386179582785574	0,400339503839302	-0,014159
9	4,0	0,365492445334809	0,365965052620526	-0,000473
10	4,5	0,425620610050548	0,413202727865295	0,012418
11	5,0	0,510150474652243	0,493589731737757	0,016561
12	5,5	0,565062657928185	0,554626651416565	0,010436
13	6,0	0,568154332713770	0,569071313440797	-0,000917
14	6,5	0,532392228147980	0,542386097140734	-0,009994
15	7,0	0,488717759195267	0,500538350966085	-0,011821
Correlação = 0,9983 Erro percentual médio = 1,98%				

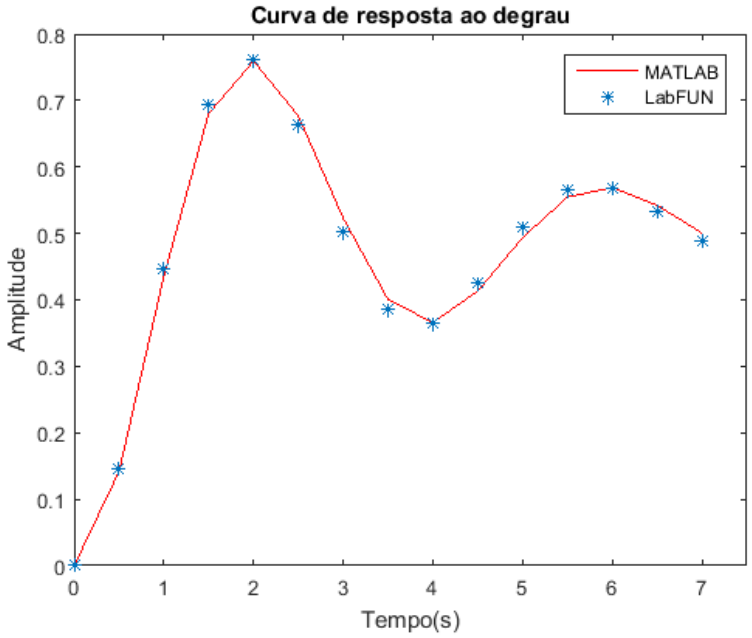

Figura 13 - Curva comparativa de entre valores do LabFUN e do MATLAB.

Com relação ao desempenho do programa durante sua execução. Considerando um processador Intel Core i5-2450M, 2.5 GHz, 64 bits com 4.0 GB de memória RAM, o programa (na versão executável - extensão .appref-ms) apresentou um consumo de memória de 10008 KB (0,238 % da memória física) e 8 *threads* em prioridade normal.

4 Conclusões

Em função dos objetivos apresentados, pode-se concluir que o presente trabalho cumpriu em sua integralidade os tópicos abordados. O software desenvolvido apresenta uma boa usabilidade, interface dinâmica e demanda um esforço computacional relativamente reduzido (0,238 % da memória física). Comparando os resultados obtidos utilizando o LabFUN e o MATLAB, software de referência, o coeficiente de correlação global foi igual a 0,9983, sendo observado um erro percentual médio de 1,98%.

O índice de correlação próximo a unidade comprova a efetividade dos resultados obtidos e legitima a escolha da abordagem de modelagem por transformada inversa de laplace em detrimento a implementação de aproximações via métodos numéricos. Para trabalhos futuros dentro da mesma temática, pode-se trabalhar no aumento de abrangência para funções de graus superiores e/ou instáveis em malha aberta.

No que diz respeito aos conceitos apresentados relacionados a programação orientada a objetos, *software* e correlatos, o trabalho se configura como uma possível base para introdução ao estudo do assunto, tendo como fator motivador a aplicação da POO e do Microsoft Visual Studio na resolução de um exemplo prático dentro do universo da engenharia elétrica. Além disso, o programa também apresenta a possibilidade de ser aplicado como ferramenta didática auxiliar no estudo de sistemas de controle.

A estrutura desenvolvida e os métodos aplicados, embora de forma introdutória, apresentam um grande potencial de aplicação e criam precedentes para a implementação de diversos tipos de aplicações semelhantes na análise de sistemas de controle, automação, circuitos elétricos/eletrônicos e processamento de sinais.

Referências Bibliográficas

PRESSMAN, Roger S. **Engenharia de software: uma abordagem profissional**. 7. ed. Porto Alegre: AMGH, 2011. Tradução de: Ariovaldo Griesi; revisão técnica: Reginaldo Arakaki, Júlio Arakaki, Renato Manzan de Andrade.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011. Tradução de: Ivan Bosnic e Kalinka G. de O. Gonçalves

UNIOESTE. **Introdução À Programação Orientada A Objetos em C++.** 2014. Disponível em: <http://www.inf.unioeste.br/~adair/ED/Apostilas/Introducao%20POO%20C++.pdf>. Acesso em: 15 maio 2019.

LIMA, Edwin; REIS, Eugênio. **C# e .Net para desenvolvedores.** Rio de Janeiro: Campus, 2002.

RICARTE, Ivan Luiz Marques. **Programação Orientada a Objetos: Uma Abordagem com Java**. São Paulo: Unicamp - Departamento de Engenharia de Computação e Automação Industrial, 2001.

DEVMEDIA. **Os 4 pilares da Programação Orientada a Objetos**. 2014. Disponível em: <https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>. Acesso em: 14 maio 2019.

SEIXAS FILHO, Constantino; SZUSTER, Marcelo. **Programação concorrente em ambiente Windows: uma visão de automação**. Belo Horizonte: Editora UFMG/Escola de Engenharia da UFMG, 2003.

CLEARY, Stephen. **Concurrency in C# Cookbook – Asynchronous, parallel, and multithreaded programming.** 1. ed. Sebastopol: O'Reilly Media, 2014.

OGATA, Katsuhiko. **Engenharia de controle moderno**. 5. ed. São Paulo: Person, 2011. Tradução de: Heloísa Coimbra de Souza.

ZILL, Dennis G. **Equações diferenciais com aplicações em modelagem.** 3. ed. São Paulo: Cengage Learning, 2016. Tradução: Márcio Koji Umezawa; revisão técnica Ricardo Miranda martins, Juliana Gaiba oliveira.

HIGUTI, Ricardo Tokio; KITANO, Cláudio. **Sinais e sistemas.** São Paulo: UNESP - Departamento de Engenharia Elétrica, 2003. Disponível em: <https://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/optoeletronica/sinais_e_sistemas.pdf>. Acesso em: 21 maio 2019.

MICROSOFT. **Documentos do Visual Studio: Linguagens**. Disponível em: <https://docs.microsoft.com/pt-br/visualstudio/?view=vs-2019#pivot=languages>. Acesso em: 16 maio 2019.

MICROSOFT. **Documentos do Visual Studio: Bem-vindo ao IDE do Visual Studio.** Disponível em: <https://docs.microsoft.com/pt-br/visualstudio/get-started/visual-studio-ide?view=vs-2019>. Acesso em: 16 maio 2019.

MATHWORKS. **Documentação do MATLAB**. Disponível em: < https://www.mathworks.com/help/matlab/index.html >. Acesso em: 20 maio 2019.

MATHWORKS. **Produtos MathWorks: MATLAB**. Disponível em: <https://www.mathworks.com/products/matlab.html?s_tid=hp_products_matlab>. Acesso em: 20 maio 2019.