

**MATHEUS DIAS REBESCHINI**

**ANÁLISE DE PROJETOS DE CONTROLADORES REALIZADOS EM  
PYTHON E MATLAB®**

**ILHA SOLTEIRA – SP**

**2022**

Matheus Dias Rebeschini

**ANÁLISE DE PROJETOS DE CONTROLADORES REALIZADOS EM  
PYTHON E MATLAB®**

Trabalho de graduação apresentado ao curso de Engenharia Elétrica, da Universidade Estadual Paulista como requisito para a obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Rodrigo Cardim

ILHA SOLTEIRA – SP

2022

FICHA CATALOGRÁFICA

Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

R291a            Rebeschini, Matheus Dias.  
                  Análise de projetos de controladores realizados em Python e Matlab /  
Matheus Dias Rebeschini. -- Ilha Solteira: [s.n.], 2022  
                  125 f. : il.

                  Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) --  
Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira,  
2022

                  Orientador: Rodrigo  
Cardim Inclui bibliografia


                  1. Teoria Controle Clássica. 2. Controle chaveado. 3. Desigualdade Matricial  
Linear. 4. MATLAB®. 5. Python.

  
Raiane da Silva Santos

Supervisora Técnica de Seção  
Seção Técnica de Referência, Atendimento ao usuário e Documentação  
Diretoria Técnica de Biblioteca e Documentação  
CRB/8 - 9999

## ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

Aos quatro dias do mês de fevereiro do ano de dois mil e vinte e dois, o discente *Matheus Dias Rebeschini*, matriculado sob o nº 161053361, tendo como banca examinadora o seu orientador, o *Prof. Dr. Rodrigo Cardim*, o *Prof. Dr. Carlos Antonio Alves* e o *Doutorando Igor Thiago Minari Ramos*, apresentou o Trabalho de Graduação intitulado "**Análise de Projetos de Controladores Realizados em Python e Matlab**" obtendo a nota 10,0 (Dez) e conceito Aprovado.




Prof. Dr. Rodrigo Cardim  
- orientador -



Matheus Dias Rebeschini  
- discente -



Prof. Dr. Carlos Antonio Alves  
- Membro da Banca -



Doutorando Igor Thiago Minari Ramos  
- Membro da Banca -

## **AGRADECIMENTOS**

Primeiramente gostaria de agradecer aos meus pais, que sempre me incentivaram a cursar a universidade, sempre me deram todo o respaldo possível e sempre acreditaram que eu conseguiria atingir todos os objetivos que foram traçados. Ao meu irmão que sempre esteve disposto a me auxiliar nos momentos mais conturbados. A minha namora que sempre esteve comigo nos momentos mais difíceis, compreendeu e teve paciência neste momento de transição da graduação.

À UNESP, todo o corpo docente, aos técnicos, todos extremamente qualificados, direção e administração por proporcionarem uma excelente formação de alto nível, formando profissionais qualificados e íntegros, formando não só profissionalmente como excepcionais pessoas.

Ao meu orientador Prof. Dr. Rodrigo Cardim, que acreditou no projeto me incentivou e me instruiu neste trabalho de graduação, sempre disposto a auxiliar no que estava em seu alcance. Ao Igor Minari, por me acompanhar em todo o processo de desenvolvimento deste trabalho, me abrindo caminhos, tirando dúvidas e estando sempre disposto para ajudar sempre que necessário.

À todas as amigas que construí durante o curso de graduação, pois sem eles eu sei que não teria chegado até o presente momento e sei que poderei contar com cada uma das amigas que fiz ao longo de toda a graduação para o reto da vida.

E para finalizar gostaria de agradecer, a todos que de alguma forma contribuíram para que este sonho de cursar Engenharia Elétrica fosse concretizado e que durante a minha vida profissional eu siga todos os ensinamentos recebidos durante o curso, repasse o conhecimento para todos a minha volta e incentive mais jovens a seguirem seus sonhos de cursarem uma universidade.

## RESUMO

Neste trabalho são apresentadas ferramentas para o projeto de controladores para plantas genéricas utilizando a teoria clássica de controle, através de um *script* desenvolvido utilizando a linguagem Python. O *script* foi desenvolvido considerando os principais parâmetros de especificação de projeto de controladores, como tempo de estabelecimento ( $t_e$ ), porcentagem de *overshoot* (P.O.), tempo de subida ( $t_s$ ) e tempo de pico ( $t_p$ ). Após realizado o projeto dos controladores será feito o gráfico do *Root Locus*, sendo que na sequência é apresentado cada parte do *script* que realiza esta tarefa acompanhado da apresentação dos resultados com os valores obtidos através do *script* em Python. Em sequência, será desenvolvido e comparado o projeto dos controladores utilizando o *software* proprietário MATLAB® e o *software* livre Python para a planta do sistema de Suspensão Ativa de Bancada, equipamento didático desenvolvido pela Quanser. Será apresentado a descrição do sistema através de espaço de estados para o desenvolvimento do projeto de controladores. O projeto dos controladores será baseado na metodologia proposta por Lyapunov, cujo objetivo é analisar uma “função de energia” relacionada com o vetor de estado do sistema. Considerando uma candidata a função de Lyapunov quadrática, as restrições no projeto de controladores podem ser reduzidas em desigualdades matriciais lineares (do inglês, *Linear Matrix Inequalities* – LMIs). É realizada a apresentação do *script* desenvolvido em Python capaz de resolver as LMIs e encontrar um ganho  $K$ .

**Palavras-chave:** Teoria Controle Clássica; Controle Chaveado; Desigualdade Matricial Linear; MATLAB®; Python.

## ABSTRACT

This work presents tools for the design of controllers for generic plants using the classical theory of control, through a script developed using the Python language. The script was developed considering the main controller design specification parameters, such as establishment time ( $t_e$ ), overshoot percentage ( $P.O.$ ), rise time ( $t_s$ ) and peak time ( $t_p$ ). After carrying out the controller's project, the Root Locus graph will be made, and in the sequence each part of the script that performs this task is presented, accompanied by the presentation of the results with the values obtained through the script in Python with the values found in the literature for the controller design. Next, the controllers design will be developed and compared using the proprietary MATLAB® software and the free Python software for the Active Banking Suspension system plant, didactic equipment developed by Quanser. The description of the system will be presented through the state space for the development of the controller's project. The controllers design will be based on the methodology proposed by Lyapunov, whose objective is to analyze an "energy function" related to the state vector of the system. Considering a candidate for a quadratic Lyapunov function, the constraints on controller design can be reduced to Linear Matrix Inequalities (LMIs). The script developed in Python capable of solving the LMIs and finding a K gain is presented.

Keywords: Classical Control Theory; Switched Control; Linear Matrix Inequalities; MATLAB®; Python.

## Lista de Figuras

Figura 1 – Exemplo diagrama de blocos de um levitador magnético. ....	17
Figura 2 – Representação sistema de primeira ordem. ....	18
Figura 3 – Função degrau. ....	19
Figura 4 – Resposta característica de um sistema de 1ª ordem a entrada degrau. ....	19
Figura 5 – Representação genérica de um sistema de 2ª ordem. ....	20
Figura 6 – Representação dos polos de sistema de 2ª ordem genérico no plano $s$ . ....	21
Figura 7 – Resposta de um sistema de segunda ordem a uma entrada degrau. ....	22
Figura 8 – Gráfico para Porcentagem de Overshoot $P.O. \leq 16\%$ . ....	25
Figura 9 – Par de curvas envoltórias de $y(t)$ . ....	26
Figura 10 – Gráfico para $te \leq 2,6s$ e critério de 1%. ....	27
Figura 11 – Gráfico com região que atende tempo de subida $ts \leq 1,8s$ . ....	28
Figura 12 – Gráfico referente $ts \leq 0,9s$ , $P.O. \leq 16\%$ , $te \leq 3s$ . ....	29
Figura 13 – Diagrama de blocos para um sistema realimentado. ....	30
Figura 14 – Diagrama de blocos para um sistema realimentado. ....	31
Figura 15 – Root-Locus para um sistema realimentado genérico. ....	32
Figura 16 – (a) Estado de Equilíbrio Estável e uma trajetória representativa; (b) Estado de Equilíbrio Estável Assintótico e uma trajetória representativa; (c) Estado de Equilíbrio Instável e uma trajetória representativa. ....	37
Figura 17 – Modelo de $\frac{1}{4}$ da suspensão ativa de um automóvel. ....	46
Figura 18 – Duplo sistema massa-mola-amortecedor usado para modelar o Sistema de Suspensão. ....	48
Figura 19 – Diagrama de corpo livre para a massa $M_s$ . ....	49
Figura 20 – Diagrama de corpo livre para a massa $M_u$ . ....	49
Figura 21 – Inserção das condições de projeto no <i>software</i> . ....	70
Figura 22 – Inserção dos polinômios do Numerador e Denominador. ....	70
Figura 23 – Parâmetros calculados pelo <i>software</i> . ....	71
Figura 24 – Root-Locus plotado no <i>software</i> , parâmetros informados pelo usuário. ....	71
Figura 25 – Root-Locus teórico referente a (4.1). ....	73
Figura 26 – Inserção das condições de projeto no <i>software</i> . ....	73
Figura 27 – Inserção dos polinômios do Numerador e Denominador. ....	74



Figura 28 – Parâmetros calculados pelo <i>software</i> .	74
Figura 29 – Root-Locus plotado via <i>software</i> e parâmetros fornecidos pelo usuário.	75
Figura 30 – Inserção das condições de projeto no <i>software</i> .	76
Figura 31 – Inserção dos polinômios do Numerador e Denominador.	77
Figura 32 – Parâmetros calculados pelo <i>software</i> .	78
Figura 33 – Root-Locus plotado pelo <i>software</i> com os parâmetros informados pelo usuário.	78
Figura 34 – Valores de entrada dos parâmetros, das funções do numerador e denominador.	80
Figura 35 – Parâmetros calculados pelo <i>software</i> .	80
Figura 36 – Root-Locus plotado pelo <i>software</i> com os parâmetros informados pelo usuário.	81
Figura 37 – Inserção dos novos valores de $Gs \times H(s)$ , mantendo-se os requisitos de projeto.	82
Figura 38 – Parâmetros calculados pelo <i>software</i> .	82
Figura 39 – Novo <i>Root-Locus</i> após a correção.	83
Figura 40 – Resultado da saída do sistema em malha aberta.	115
Figura 41 – Primeiro controlador único ganho MATLAB®.	116
Figura 42 – Sinal de controle controlador único ganho MATLAB®.	116
Figura 43 – Primeiro controlador único ganho Python.	117
Figura 44 – Sinal de controle controlador único ganho Python.	117
Figura 45 – Segundo controlador único ganho, incerteza na massa MATLAB®.	118
Figura 46 – Sinal de controle, único ganho, incerteza na massa MATLAB®.	118
Figura 47 – Segundo controlador único ganho com incerteza na massa Python.	119
Figura 48 – Sinal de controle, único ganho com incerteza na massa Python.	119

# SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>9</b>
<b>1.1 Objetivo .....</b>	<b>13</b>
<b>1.2 Estrutura do trabalho .....</b>	<b>14</b>
<b>2 FUNDAMENTAÇÃO TEÓRICA .....</b>	<b>16</b>
<b>2.1 Modelagem de um sistema.....</b>	<b>16</b>
<b>2.2 Especificação de projeto .....</b>	<b>18</b>
2.2.1 Tempo de pico ou instante de pico $tp$ .....	23
2.2.2 Porcentagem de Overshoot (P.O.) .....	24
2.2.3 Tempo de estabelecimento $te$ .....	25
2.2.4 Tempo de subida $ts$ .....	27
<b>2.3 Root Locus.....</b>	<b>29</b>
<b>2.4 Espaço de estados .....</b>	<b>34</b>
<b>2.5 Análise da estabilidade segundo Lyapunov .....</b>	<b>35</b>
<b>2.6 Realimentação de Estados .....</b>	<b>39</b>
<b>2.7 Taxa de decaimento .....</b>	<b>41</b>
<b>2.8 Parâmetros incertos para sistemas com realimentação de estados.....</b>	<b>41</b>
<b>2.9 Restrição no sinal de controle do atuador .....</b>	<b>42</b>
<b>2.10 Controlador Chaveado.....</b>	<b>44</b>
<b>2.11 Suspensão Ativa de Bancada .....</b>	<b>45</b>
<b>2.12 Modelo Matemático Suspensão Ativa de Bancada.....</b>	<b>47</b>
<b>2.13 Resolução LMIs com Matlab®: Yalmip, LMILab e Sedumi .....</b>	<b>50</b>
<b>2.14 Resolução de LMIs com Python: Picos e CVXOPT .....</b>	<b>51</b>
<b>3 DESENVOLVIMENTO .....</b>	<b>52</b>
<b>3.1 Primeiro projeto Root Locus.....</b>	<b>52</b>
<b>3.2 Apresentação e descrição de cada parte do código .....</b>	<b>52</b>

<b>3.3 Segundo projeto LMIs .....</b>	<b>61</b>
<b>3.4 Apresentação de cada parte do código .....</b>	<b>62</b>
<b>4 RESULTADOS E ANÁLISE.....</b>	<b>69</b>
<b>4.1 Primeira aplicação exemplo Root Locus .....</b>	<b>69</b>
<b>4.2 Segunda aplicação exemplo Root Locus.....</b>	<b>72</b>
<b>4.3 Terceira aplicação exemplo <i>Root Locus</i> .....</b>	<b>75</b>
<b>4.4 Terceira aplicação exemplo com correção do Root-Locus .....</b>	<b>79</b>
<b>4.5 Análise geral dos resultados dos projetos realizados utilizando o código desenvolvido</b>	<b>84</b>
<b>4.6 Projeto de Controladores via LMIs .....</b>	<b>84</b>
<b>4.7 Projeto controlador com único ganho K.....</b>	<b>85</b>
4.7.1 Projeto controlador MATLAB® .....	85
4.7.2 Projeto Python .....	88
<b>4.8 Projeto controlador com incerteza na massa total do veículo.....</b>	<b>91</b>
4.8.1 Projeto controlador MATLAB® .....	92
4.8.2 Projeto controlador Python.....	94
<b>4.9 Primeiro controlador chaveado considerando incerteza na massa do veículo .....</b>	<b>98</b>
4.9.1 Projeto controlador MATLAB® .....	99
4.9.2 Projeto controlador Python.....	103
<b>4.10 Controlador chaveado com redução no sinal de controle.....</b>	<b>106</b>
4.10.1 Projeto MATLAB®.....	106
4.10.2 Projeto Python .....	110
<b>4.11 Simulações .....</b>	<b>115</b>
4.11.1 Primeiro controlador único ganho $K$ , $\gamma = 7$ .....	115
4.11.2 Segundo controlador único ganho considerando incerteza na massa, $\gamma = 7$ .....	118
<b>5 CONSIDERAÇÕES FINAIS .....</b>	<b>121</b>
<b>REFERÊNCIAS .....</b>	<b>123</b>

## 1 INTRODUÇÃO

No mundo contemporâneo os termos que dominam as discussões em meios empresariais são as expressões: inovação tecnológica e redução de custos, visto que, estamos em um mercado que se caracteriza por uma acirrada competição, assíduo por mudanças rápidas, constantes, sempre na busca de novos produtos, processos e tecnologias (TABOADA, 2009).

De acordo com TABOADA, (2009) podemos assumir que a inovação tecnológica significa as competências desenvolvidas por uma instituição que proporcionou a criação de novos produtos, processos ou métodos que foram notados pelo mercado e aceitos como uma novidade.

Dentre todas as inovações que podem ser abordadas, pensamos principalmente na inovação de processos, que consiste na implementação de uma metodologia produtiva ou de entrega, carregando consigo um aspecto inovador podendo trazer melhorias para o processo, essa inovação pode ter pontos técnicos, equipamentos ou *softwares* que tragam valor para o produto final (TABOADA, 2009).

Neste sentido uma alternativa que vem ganhando força e sendo amplamente utilizada nos dias atuais é a utilização de *softwares* livres, no qual, vem sendo cada vez mais adotado por pessoas, empresas e governos, tendo como pontos atrativos serem gratuitos, oferecerem uma maior liberdade que os programas proprietários com relação a sua modificação, distribuição e utilização, tendo em vista que, disponibilizam sua base em código aberto (DE MATTOS, 2005).

Quando tratamos de um *software* livre temos que ter em mente que ele pode ser usado, copiado, estudado, modificado e redistribuído sem nenhum tipo de restrição, logo, além de poder utilizá-lo sem o pagamento de direitos autorais, o usuário tem acesso irrestrito ao código fonte, o que permite que qualquer usuário modifique e realize adaptações para viabilizar suas necessidades individuais (PETRUCELLI 2010).

São estas características que trazem os *softwares* livres para um contexto universitário e nos leva a crer que seria uma excelente opção para se colocar como uma possível ferramenta para fazer frente aos *softwares* privados utilizados no meio acadêmico.

Durante o curso de graduação, principalmente nas disciplinas na área de exatas, são utilizados *softwares* para realização de cálculos, simulações, comparação de resultados teóricos e experimentais. No curso de Engenharia Elétrica podemos citar alguns *softwares* utilizados durante a graduação: MATLAB<sup>®</sup> e Simulink<sup>®</sup>, OrCAD<sup>®</sup> PSpice<sup>®</sup>, Autodesk AutoCAD<sup>®</sup>,

Proteus<sup>®</sup>, entre outros, sendo que todos os citados são protegidos por licenças privativas com um alto valor de mercado.

A teoria de controle faz grande uso dos benefícios proporcionados pelos *softwares* matemáticos. Quando pensamos em controle podemos citar principalmente a teoria de controle clássico (também chamada de teoria de controle convencional), teoria de controle moderno e a teoria de controle robusto (OGATA, 2010).

A teoria do controle clássico tem sua base fundamentada nos métodos do lugar das raízes (*Root-Locus*) e de resposta em frequência, que são a essência da teoria clássica de controle. Estes métodos proporcionaram a obtenção de sistemas estáveis e que satisfazem uma série de condições de desempenho (OGATA, 2010).

Com a evolução dos sistemas modernos a serem controlados e a inserção de sistemas com várias entradas e saídas cada vez mais complexos, no qual requer um grande número de equações para encontrar uma solução faz surgir o apelo pela teoria de controle moderno, visto que, a teoria de controle clássica, que por sua vez estuda somente os casos de sistemas com apenas uma entrada e uma saída seja insuficiente para estes casos (OGATA, 2010).

É neste contexto que os *softwares* matemáticos se enquadram na teoria de controle. Com o advento da tecnologia e a disponibilidade de *softwares* matemáticos capazes de realizarem cálculos mais complexos com grande velocidade, o que acabou viabilizando a análise de sistemas complexos diretamente no domínio do tempo em sistemas de equações diferenciais com o emprego do método de variáveis de estado, criado para trabalhar com sistemas modernos complexos e rigorosos com relação a sua precisão.

A teoria de controle moderno apresentou uma simplificação quando se trata de projetos de sistemas de controle, visto que, tem como base um modelo do sistema de controle real, por outro lado, a estabilidade do sistema está atrelada ao erro entre o sistema real e o seu modelo, isto implica que caso ocorra uma mudança no sistema real que difere do sistema modelado, o sistema poderá ser instável (OGATA, 2010).

Para contornar o problema encontrado na teoria de controle moderna, a teoria de controle robusto vem sendo estudada, ela baseia-se em projetar um sistema, estabelecendo uma gama de possíveis incertezas e posteriormente projetar um controlador onde o erro do sistema esteja sempre dentro desta gama prevista anteriormente o que tornará o sistema estável (OGATA, 2010). A teoria de controle robusto quando analisada do ponto de vista matemático

é muito complexa, é nesse contexto onde os *softwares* matemáticos como o MATLAB<sup>®</sup> se destacam nas resoluções dos problemas de controle.

Como na teoria dos controladores robustos podemos ter várias incertezas, como por exemplo a inserção de distúrbios durante a aplicação, ou até mesmo uma falha na medição dos sensores, todos estes desvios que podem levar o sistema para a instabilidade são levados em consideração no projeto.

Outro fato relevante está na utilização da notação vetorial-matricial que simplifica muito a análise de projetos com muitas entradas e saídas, podendo ser então apenas um pouco mais complexo que o necessário para análise dos sistemas de equações diferenciais lineares de primeira ordem.

Uma das possibilidades de solução para controladores robustos é a utilização da teoria de controladores chaveados, onde seguindo uma lei de chaveamento o controlador pode assumir um conjunto de ganho que melhor atenda o sistema controlado em determinado momento.

Diversos trabalhos vêm sendo desenvolvidos sobre controladores robustos, estes estudos se justificam uma vez que os controladores chaveados tem uma melhor performance quando comparados com os controladores de único ganho, apresentando uma maior região de factibilidade (OLIVEIRA et al. 2017) e (SOUZA, 2013). No geral esses controladores podem obter um ganho pertencente a um conjunto de ganhos e regidos por uma lei de chaveamento que escolhe qual o ganho que minimiza a derivada da função de Lyapunov naquele instante específico.

Por mais que os controladores chaveados vem sendo amplamente estudados no meio acadêmico, no meio industrial os controladores PID são os mais utilizados em malhas de processos, por exemplo em plantas petroquímicas, celulose, laticínios, tratamento de água, metalúrgicas, entre outras (BAZANELLA, 2005).

Mesmo com o controlador PID desenvolvido com a teoria clássica de controle apresentando suas restrições com relação a solução ótima do problema, e mesmo com sua simplicidade na solução dos problemas estes controladores de estrutura simples e padronizada representam a imensa maioria das malhas de controle industriais em todo o mundo (mais de 90% de acordo com o autor ÅSTRÖM HÄGGLUND (1995). Como o controlador PID ainda é amplamente utilizado na indústria atualmente, abordaremos sobre ele neste trabalho.

O *software* amplamente utilizado na resolução das *Linear Matrix Inequalities* (LMIs) no projeto de controladores chaveados é o MATLAB<sup>®</sup> que é um *software* proprietário e tem seu custo de licença elevado.

Um *software* livre que vem chamando muita a atenção no meio acadêmico e industrial é a linguagem de programação Python, que pode ser uma segunda opção sobre a utilização do MATLAB<sup>®</sup> no projeto de controladores chaveados e no projeto de controladores utilizando a teoria clássica de controle.

Python tem como características ter uma sintaxe clara e concisa que proporciona a legibilidade do código-fonte, ou seja, trata-se de uma linguagem de altíssimo nível e bastante próxima do entendimento humano o que acarreta uma linguagem mais produtiva (BORGES, 2014).

A linguagem inclui diversas estruturas de alto nível (listas, dicionários, data/hora entre outras) dispõe de bibliotecas prontas como as bibliotecas científicas *Numpy* e *Scypy* entre outras. Permite a programação modular e funcional, além da programação orientada a objetos, é interpretado pela máquina virtual Python o que torna o seu código portátil, sendo assim, é possível compilar a aplicação em uma estação e executar em outros sistemas ou até realizar a execução do próprio código fonte (BORGES, 2014).

Python tem sua licença compatível com a *General Public License* (GPL) com menos restrições o que torna esta linguagem integrável em produtos proprietários. Tal linguagem é gerida pela *Python Software Foundation* (PSF) e foi criada em 1990 por Guido van Rossum, no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação localizado na Holanda (CWI) (BORGES, 2014).

Segundo Borges (2014) a linguagem é muito bem aceita nas empresas que prezam pela alta tecnologia como por exemplo:

- Google – em aplicações Web;
- Yahoo – em aplicações Web;
- Microsoft – em IronPython: Python para .NET;
- Nokia – disponível em linhas recentes de celulares;
- Disney – animações 3D;
- Youtube – Transmissões de vídeos entre outros.

O Python trabalha de forma isolada, entretanto, como se trata de uma linguagem é possível realizar integrações a outras linguagens, como por exemplo com Java, C, JavaScript, linguagens de marcação como HTML entre outras (DA SILVA, 2019).

Posto alguns atrativos, podemos pensar em utilizar Python como um possível concorrente ao *software* de domínio proprietário MATLAB<sup>®</sup>, visando a utilização de um *software* livre que oferece vários benefícios e se torna uma opção interessante para resolução de problemas de controle.

## 1.1 Objetivo

Este trabalho tem como principal objetivo verificar o comportamento da linguagem de programação Python na resolução de problemas de controle, principalmente na possibilidade de resolver LMIs (do inglês, *Linear Matrix Inequalities*) utilizando a linguagem de programação Python.

Inicialmente verificou-se a possibilidade da montagem de um programa utilizando a linguagem Python na resolução de problemas de projeto de controladores PID utilizando a teoria clássica de controle, projetando um controlador adequado utilizando o método do lugar das raízes (*Root-Locus*).

Posteriormente verificou-se o projeto de controlador chaveado para o sistema a ser controlado da suspensão ativa de bancada, fabricado pela QUANSER<sup>®</sup>, este projeto foi estruturado com a criação de um *script* na linguagem Python e um projeto semelhante no MATLAB<sup>®</sup>, sendo possível encontrar uma solução para as LMIs e encontrando um ou um conjunto de ganhos  $K$  para o projeto em Python. Após os *softwares* encontrarem uma solução factível, os resultados obtidos serão comparados.

No projeto do controlador chaveado foram utilizadas as LMIs (do inglês, *Linear Matrix Inequalities*) (BOYD et al., 1994). As soluções das restrições no projeto de controle descritas por LMIs podem ser encontradas utilizando o software MATLAB<sup>®</sup>, com a interface YALMIP (LÖFBERG, 2004) e o solver LMILab (GAHINET et al., 1994)

Quando utilizamos a linguagem de programação Python a resolução das LMIs ocorre de forma similar, entretanto, a interface utilizada é a biblioteca PICOS (SAGNOL;



STAHLBERG, 2021) que corresponde a interface YALMIP, e o *solver* utilizado será o CVXOP (*Python Software for Convex Optimization*) (SAGNOL; STAHLBERG, 2021).

## 1.2 Estrutura do trabalho

Neste trabalho foi abordado dois principais conceitos, sendo o primeiro o projeto de controladores utilizando a teoria clássica de controle em específico a plotagem do *Root Locus*, no qual implementou-se um projeto em Python visando obter de modo simples e rápido o gráfico do método do Lugar das Raízes para projeto de controladores.

O segundo tópico foi a construção e comparação de projetos de controladores chaveados de um sistema de controle para o conjunto de Suspensão Ativa de Bancada desenvolvido pela QUANSER®, no qual utilizou-se o MATLAB® e Python, e comparou-se o resultado de ambos visando evidenciar características que justifiquem a utilização do *software* livre Python para a resolução de LMIs.

Para tal foram apresentadas as teorias do controle clássico e de espaço de estados, também apresentou-se a metodologia utilizada no desenvolvimento do *script* em Python para ambos os casos.

Para uma apresentação adequada do tema e dos resultados obtidos, este trabalho está organizado da seguinte forma:

- Capítulo 2: Apresenta os conceitos fundamentais para a compreensão e desenvolvimento deste trabalho, a teoria clássica de controle abordando a análise de projetos por meio do *Root Locus*, representação de um sistema dinâmico através de espaço de estados, seguindo com o critério de estabilidade segundo Lyapunov onde as condições de estabilidade são dadas em termos de resolução de LMIs. São apresentadas a descrição da suspensão ativa de bancada desenvolvido pela QUANSER®, a modelagem do equipamento e sua descrição em espaço de estados. Para finalizar a parte teórica é apresentado o método de resolução de LMIs utilizando o MATLAB® e o método de resolução de LMIs utilizando Python.
- Capítulo 3: É feita uma descrição da solução de implementar um *script* em Python capaz de calcular os principais parâmetros de projeto da teoria clássica de controle e plotar o *Root Locus* da planta do projeto, na sequência é apresentado o *script* que foi desenvolvido para este escopo. Em seguida foi apresentado a segunda parte da ideia onde é descrito a resolução de LMIs utilizando Python, foi também apresentada cada parte do *script* utilizado.

- Capítulo 4: É apresentado exemplos de projetos onde é validado o funcionamento do *script* desenvolvido para plotagem do *Root Locus* e comparado com exemplos teóricos. Posteriormente é apresentado tanto o *script* quanto a comparação entre a resolução de LMIs utilizando o *software* MATLAB® com a resolução obtida utilizando o *software* livre Python. Contendo os resultados das implementações dos projetos de controle, bem como os comentários pertinentes de cada projeto.
- Capítulo 5: Apresenta uma conclusão geral sobre os resultados obtidos durante a realização deste trabalho e perspectivas futuras para melhorias e novos estudos.

## 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, são apresentados os principais conceitos utilizados no trabalho, que vão desde a definição de controladores Proporcional Integral Derivativo (PID) utilizando a teoria clássica de controle, as restrições de projeto, a sua metodologia de projeto, até suas aplicações na indústria e no ensino, visando expor todo o embasamento teórico utilizado para a elaboração do projeto. Por fim será apresentado toda a metodologia utilizada na elaboração dos projetos de controle com relação aos controladores chaveados projetados através das restrições descritas em LMIs, baseados em estudos já consolidados e disponíveis na literatura que formam uma base sólida para a execução deste trabalho.

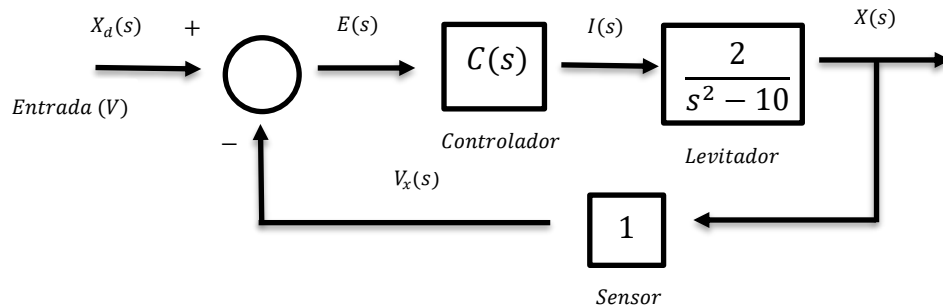
### 2.1 Modelagem de um sistema

Os modelos matemáticos quantitativos de sistemas físicos são utilizados para fins de projeto e análise de sistemas de controle, este modelo é definido como um conjunto de equações diferenciais ordinárias (EDOs), visto que tratamos de sistemas dinâmicos, logo as EDOs representam toda a dinâmica do sistema estudado com certa precisão. Se estas equações puderem ser linearizadas, podemos utilizar uma ferramenta extremamente importante para simplificar o método de solução que é a transformada de *Laplace*.

Em problemas de controle ótimo é vantajoso utilizar representações de espaço de estados, devido ao fato de trabalhar com várias entradas e várias saídas, entretanto, para uma análise da resposta transitória utilizando as simplificações da teoria *Root-Locus*, ou da resposta em frequência para um Sistema Linear e Invariante no Tempo (SLIT) com entradas e saídas únicas a representação pela função de transferência será mais conveniente do que outra representação (OGATA, 2010).

Após a obtenção de relações do tipo entrada-saída sob a forma de função transferência para componentes e subsistemas, podemos utilizar o artifício do diagrama de blocos, como exemplo apresentado na Figura 1 – Exemplo diagrama de blocos de um levitador magnético., para trazer simplicidade nas interconexões do sistema a ser controlado, tornando-se uma ferramenta facilitadora no projeto dos controladores (DORF, 2001).

Figura 1 – Exemplo diagrama de blocos de um levitador magnético.



Fonte: (ASSUNÇÃO; TEIXEIRA, 2018).

Na Figura 1 é apresentado um digrama de blocos que representa um sistema de controle de malha fechada de um levitador magnético. Neste diagrama nota-se que atuamos no sistema a partir da entrada  $X_d(s)$ , esta entrada passa por um comparador que realiza a diferença da entrada positiva com o sinal que está realimentado o sistema (entrada negativa do comparador), neste caso  $V_x(s)$ , se ainda existir a diferença entre os dois sinais o sinal  $E(s)$  será diferente de zero e existirá um sinal sobre o controlador  $C(s)$ , para que o mesmo atue no sistema de modo a diminuir o erro resultante do comparador através da realimentação.

Após a passagem pelo controlador, temos uma planta que caracteriza a modelagem matemática do sistema real a ser controlada, cada sistema possui uma dinâmica diferente, conseqüentemente possui sua equação característica, vale observar que as equações estão no plano complexo  $S$ , ou seja, aplicou-se uma transformada de *Laplace* na equação temporal para facilitar os cálculos no projeto do controlador.

Na saída da planta temos o sinal de saída  $X(s)$ , este sinal é colhido por um sensor que realimenta a entrada para fechar a malha, de tal forma que o erro seja levado a zero e o controlador deixa de atuar sobre o sinal  $E(s)$  até que um novo sinal seja aplicado à entrada do mesmo, tornando o sinal  $E(s)$  diferente de zero e fazendo o controlador atuar novamente.

A grande dificuldade está na parte de projetar o controlador  $C(s)$ , sendo necessário atender alguns critérios de projeto, e conseqüentemente, tornar o sistema de malha fechada um sistema estável que realmente controle a planta de forma adequada. A ferramenta que soluciona este problema é a utilização da técnica conhecida como Método do Lugar das Raízes (*Root-Locus*).

## 2.2 Especificação de projeto

Uma das vantagens de sistemas de controle com realimentação da entrada está na possibilidade de se ajustar o desempenho do sistema tanto para a parcela em regime transitório quanto para a parcela de regime permanente e quando se analisa e projeta sistemas de controle deve-se definir os valores dos índices de desempenho que garantam a eficiência do sistema de controle (DORF, 2001).

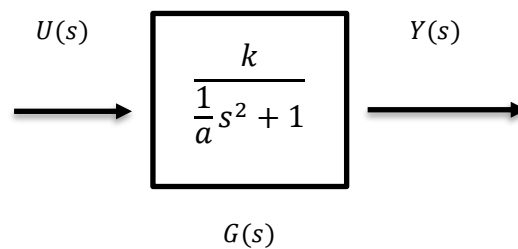
A resposta transitória é a parte da resposta que desaparece com o passar do tempo, já a resposta em regime permanente é a parte da resposta que ocorre, mesmo depois de ter passado muito tempo da aplicação de um sinal de entrada no sistema (DORF, 2001).

Com relação as especificações do projeto de sistemas de controle se incluem vários índices temporais para um comando de entrada especificada com uma determinada exatidão em regime permanente.

Os índices de desempenho nos sistemas de controle são analisados em função da resposta transitória do sistema quando submetido a uma entrada degrau, que possibilita a investigação de como um sistema responde quando tal entrada é aplicada, mais especificamente analisaremos o comportamento de sistemas de 1° e 2° ordem.

Analisando o sistema de primeira ordem podemos tirar algumas informações, de forma genérica um sistema de primeira ordem pode ser representado como mostrado na Figura 2.

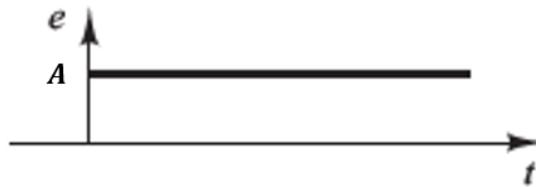
Figura 2 – Representação sistema de primeira ordem.



Adaptado de: (ASSUNÇÃO; TEIXEIRA, 2018).

Supondo que este sistema seja estável, onde  $a > 0$  e  $u(t)$  seja uma entrada do tipo degrau, onde  $u(t) = A$  para  $t \geq 0$ , como apresentado na Figura 3.

Figura 3 – Função degrau.



Fonte: (OGATA, 2010).

Sendo assim, a transformada de *Laplace* da função degrau é  $U(s) = \frac{A}{s}$  e conhecendo a saída genérica do sistema de primeira ordem podemos expressar por:

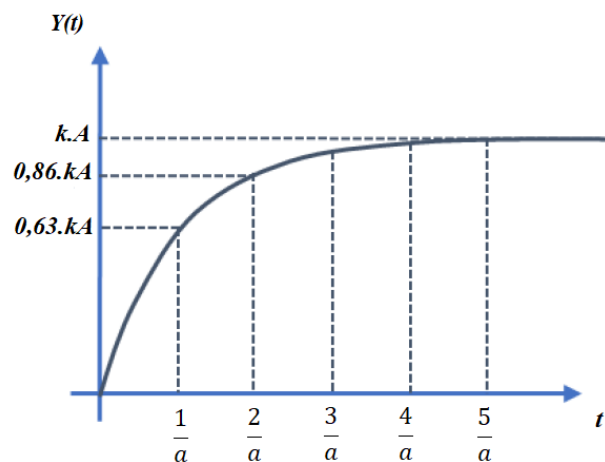
$$Y(s) = \frac{aK}{(s+a)} \times \frac{A}{s}. \quad (1)$$

Aplicando a transformada de *Laplace* inversa no sinal de saída mostrado em (1) temos:

$$y(t) = K \times A(1 - e^{-at}). \quad (2)$$

Em (2) está representada a resposta ao degrau para um sistema de primeira ordem genérico, e podemos fazer uma análise do resultado, para podermos separar o que se trata do período transitório da resposta e o que é regime permanente. Em (2), segue análise na Figura 4.

Figura 4 – Resposta característica de um sistema de 1ª ordem a entrada degrau.



Adaptado de: (ASSUNÇÃO; TEIXEIRA, 2018).

Observando a Figura 4, podemos notar que em  $t = \frac{1}{a}$  a saída  $y(t)$  atingiu 63% do seu valor de regime, já em  $t = \frac{2}{a}$  a saída  $y(t)$  atingiu 86% do seu valor de regime, em  $t = \frac{4}{a}$  a saída

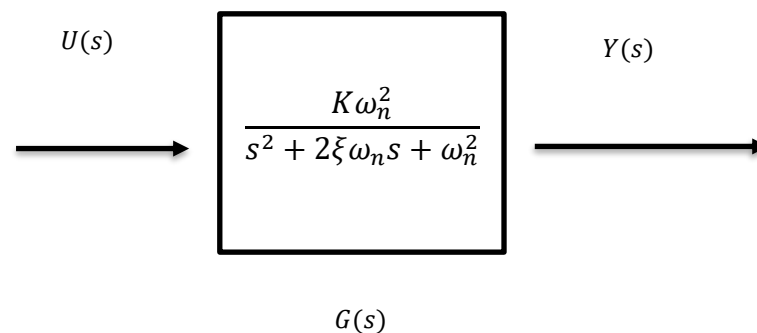
$y(t)$  atinge 98% do seu valor de regime, logo em termos práticos, considera-se que para  $t \geq \frac{4}{a}$  o sistema já está em regime permanente, portanto para  $t$  entre 0 e  $\frac{4}{a}$  temos o regime transitório, e para  $t \geq \frac{4}{a}$  temos o regime permanente.

Cada instante de tempo  $t = \frac{1}{a}$  é denominado constante de tempo do sistema, simbolizado por  $\tau$ , logo o tempo  $t = 4\tau$  é chamado de tempo de estabelecimento.

Antes de entrar nos critérios de especificação do projeto vamos analisar a resposta transitória de sistemas de segunda ordem e a partir da análise deste sistema tirar as expressões das especificações do projeto.

De forma análoga ao que foi realizado no sistema de primeira ordem, determinamos uma equação característica, para o sistema de segunda ordem genérico, que pode ser representando como mostrado na Figura 5.

Figura 5 – Representação genérica de um sistema de 2ª ordem.



Adaptado de: (ASSUNÇÃO; TEIXEIRA, 2018).

Onde  $K$  representa o ganho do controlador,  $\omega_n$  é a frequência natural não-amortecida e  $\xi$  é o coeficiente de amortecimento, em todos os casos será suposto que  $\omega_n > 0$  e  $\xi > 0$

O valor de  $\xi$  pode variar e gerar possibilidades diferentes de classificação, primeiro quando  $\xi = 1$  o sistema é criticamente amortecido, se  $\xi > 1$  o sistema é superamortecido e se  $\xi = 0$  se trata de um sistema não amortecido. A princípio vamos estudar o caso em que  $\xi$  está dentro do intervalo  $0 < \xi < 1$  que é o caso subamortecido.

Neste caso os polos de  $G(s)$  podem ser encontrados por meio das raízes do polinômio do denominador mostrado na Figura 5, sendo:

$$s^2 + 2\xi\omega_n s + \omega_n^2 = 0, \quad (3)$$

$$\Delta = b^2 - 4ac = 4\omega_n^2(\xi^2 - 1), \quad (4)$$

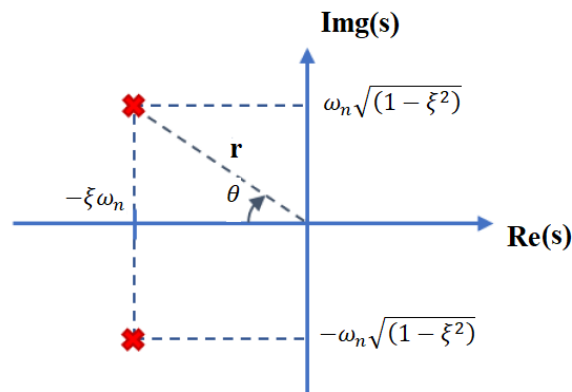
logo temos que os polos de  $G(s)$  são as raízes presentes em (3), sendo  $\xi^2 < 1$  então  $1 - \xi^2 > 0$  temos:

$$P_{1,2} = \frac{\left(2\xi\omega_n \pm \sqrt{4\omega_n^2(\xi^2 - 1)}\right)}{2} = -\xi\omega_n \pm \omega_n\sqrt{(1 - \xi^2)} \times \sqrt{-1}, \quad (5)$$

$$P_{1,2} = -\xi\omega_n \pm j\omega_n\sqrt{(1 - \xi^2)}. \quad (6)$$

Em (6) podemos montar a representação no plano  $s$  dos polos de um sistema de segunda ordem genérico, como está mostrado na Figura 6.

Figura 6 – Representação dos polos de sistema de 2ª ordem genérico no plano  $s$ .



Adaptado de: (ASSUNÇÃO; TEIXEIRA, 2018).

Ao analisarmos a Figura 6, podemos notar que:

$$r^2 = \left(\omega_n\sqrt{(1 - \xi^2)}\right)^2 + (\xi\omega_n)^2, \quad (7)$$

$$r = \omega_n, \quad (8)$$

$$\cos \theta = \frac{\xi\omega_n}{r}. \quad (9)$$

Substituindo (8) em (9), temos:

$$\cos \theta = \frac{\xi\omega_n}{\omega_n}, \quad (10)$$

$$\theta = \arccos \xi, \quad (11)$$



Do mesmo modo que foi realizado para a um sistema de primeira ordem, vamos analisar a resposta do sistema de segunda ordem mostrado na Figura 5 para uma entrada do tipo degrau unitário, sendo assim a entrada no domínio de *Laplace* será  $U(s) = \frac{1}{s}$ , logo teremos:

$$Y(s) = G(s) \times U(s) = \frac{K\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \times \frac{1}{s}, \quad (12)$$

aplicando a transformada de *Laplace* em (12), ficamos com:

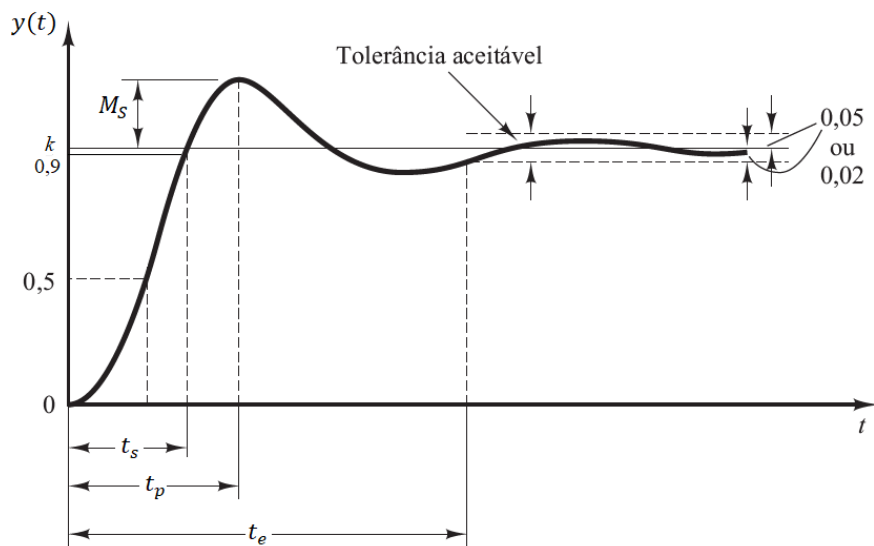
$$y(t) = K \left\{ 1 - e^{-\xi\omega_n t} \times [\cos(\omega_n\sqrt{1-\xi^2} \times t) + \frac{\xi\omega_n}{\omega_n\sqrt{1-\xi^2}} \times \text{sen}(\omega_n\sqrt{1-\xi^2} \times t)] \right\}, \quad (13)$$

como analisando o caso do sistema subamortecido onde  $0 < \xi < 1$  e para simplificar vamos definir  $\omega_d = \omega_n\sqrt{1-\xi^2}$  e  $\sigma = \xi\omega_n$ , logo (13) é simplificada para:

$$y(t) = K \left\{ 1 - e^{-\sigma t} \times [\cos(\omega_d \times t) + \frac{\sigma}{\omega_d} \times \text{sen}(\omega_d \times t)] \right\}. \quad (14)$$

Com a equação temporal genérica de um sistema de segunda ordem como mostrado em (14) podemos montar o gráfico da saída  $y(t)$  em função do tempo, este gráfico é mostrado na Figura 7.

Figura 7 – Resposta de um sistema de segunda ordem a uma entrada degrau.



Adaptado de: (OGATA, 2010).

No desempenho dos sistemas de controle normalmente as características são especificadas em função da resposta transitória a uma entrada degrau, visto que, é uma entrada

suficientemente brusca e fácil de ser gerada, e partindo da resposta do sistema a uma entrada degrau se torna possível calcular a resposta a qualquer tipo de sinal de entrada (OGATA, 2010).

Quando observamos a Figura 7 podemos notar que o sistema apresenta oscilações amortecidas antes de atingir o regime permanente e podemos tirar alguns parâmetros nos quais serão especificados no projeto e são apresentados a seguir:

- Tempo de pico ou instante de pico,  $t_p$ ;
- Máximo sobressinal ou “overshoot”,  $M_S$ ;
- Tempo de estabelecimento (ou de estabilização ou de acomodação ou de assentamento),  $t_e$ ;
- Tempo de subida,  $t_s$ .

### 2.2.1 Tempo de pico ou instante de pico $t_p$

O tempo de pico  $t_p$  é o tempo que é gasto para o sistema  $y(t)$  atingir o seu valor de máximo, a expressão para o tempo de pico é determinada no instante que  $y(t)$  é máximo, logo faremos  $\frac{d}{dt}y(t) = 0$  para encontrar o máximo da função  $y(t)$ .

Lembrando que a frequência natural não-amortecida  $\omega_n$  e o coeficiente de amortecimento  $\xi$ , são valores maiores que zero,  $\omega_n$  pode ser obtido tendo o tempo de estabelecimento,  $\xi$  obtido a partir a porcentagem de *overshoot* (*P. O.*), portanto torna-se possível realizar o cálculo do tempo de pico.

A seguir está a dedução do tempo de pico:

$$\frac{d}{dt}y(t) = k \left\{ \sigma e^{-\sigma t} \times [\cos(\omega_d \times t) + \frac{\sigma}{\omega_d} \times \text{sen}(\omega_d \times t)] - e^{-\sigma t} \times [-\omega_d \text{sen}(\omega_d \times t) + \sigma \times \text{cos}(\omega_d \times t)] \right\} = 0. \quad (15)$$

Distribuindo os termos  $\sigma e^{-\sigma t}$ ,  $-e^{-\sigma t}$  em (15), tem-se:

$$e^{-\sigma t} \times \left[ \frac{\sigma^2}{\omega_d} \text{sen}(\omega_d t) + \omega_d \text{sen}(\omega_d t) \right] = 0. \quad (16)$$

Sendo que  $-e^{-\sigma t} \neq 0$  para  $t > +\infty$  logo (16) torna-se:

$$\left( \frac{\sigma^2}{\omega_d} + \omega_d \right) \text{sen}(\omega_d t) = 0. \quad (17)$$

Para que (17) seja verdadeira  $\text{sen}(\omega_d t) = 0$  quando  $\omega_d = 0, \pi, 2\pi, 3\pi \dots$  logo se  $t = 0$  temos um ponto de mínimo da equação e o próximo zero que ocorre em  $t = \frac{\pi}{\omega_d}$  é o primeiro instante onde ocorre a derivada nula, sendo assim o primeiro instante de pico, como definido anteriormente  $\omega_d = \omega_n \sqrt{1 - \xi^2}$  ficamos com:

$$t_p = \frac{\pi}{\omega_n \sqrt{1 - \xi^2}}. \quad (18)$$

### 2.2.2 Porcentagem de Overshoot (P.O.)

Definimos a porcentagem de *overshoot* como sendo a porcentagem do quanto a resposta  $y(t)$  ultrapassa o valor de regime permanente, máximo sobressinal (MS) com relação ao valor de regime permanente da função  $y(t)$  sendo assim temos:

$$P. O. = \frac{MS}{K} \times 100 (\%). \quad (19)$$

Sendo MS o valor da diferença entre a função  $y(t)$  no instante  $t = t_p$  e o valor de regime permanente  $K$ , ou seja:

$$MS = y(t)|_{t=t_p} - K. \quad (20)$$

Substituindo  $t = t_p$  em (14), temos:

$$MS = -K + K \left\{ 1 - e^{-\sigma \left(\frac{\pi}{\omega_d}\right)} \times \left[ \cos \left( \omega_d \times \left(\frac{\pi}{\omega_d}\right) \right) + \frac{\sigma}{\omega_d} \times \text{sen} \left( \omega_d \times \left(\frac{\pi}{\omega_d}\right) \right) \right] \right\}, \quad (21)$$

$$MS = \left\{ K e^{-\sigma \left(\frac{\pi}{\omega_d}\right)} \right\}. \quad (22)$$

Substituindo (22) em (19), temos:

$$P. O. = e^{-\sigma \left(\frac{\pi}{\omega_d}\right)} \times 100 (\%), \quad (23)$$

sendo que  $\omega_d = \omega_n \sqrt{1 - \xi^2}$  e  $\sigma = \xi \omega_n$  ficamos com:

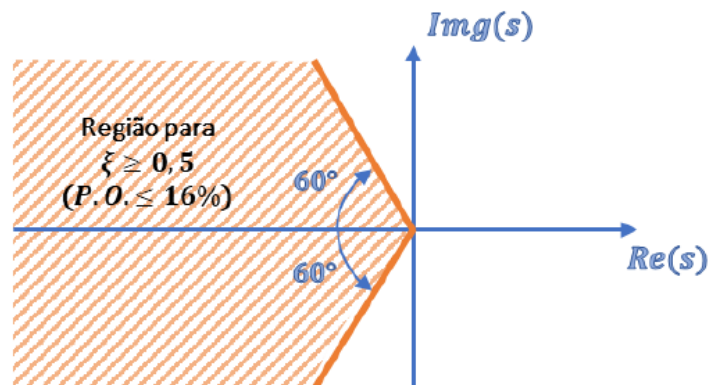
$$P. O. = e^{-\left(\frac{\xi \pi}{\sqrt{1 - \xi^2}}\right)} \times 100 (\%) \text{ para } 0 < \xi < 1. \quad (24)$$

Observa-se (24), que a porcentagem de *overshoot* depende exclusivamente de  $\xi$  e que quanto menor for  $\xi$  maior será a porcentagem de *overshoot* do sistema, logo concluímos que quanto maior for o coeficiente de amortecimento  $\xi$  menor será a oscilação da resposta.

A porcentagem de *overshoot* ( $P. O.$ ) é um parâmetro também de grande importância no projeto do controlador, pois tendo o valor da  $P. O.$  é possível calcular o valor de  $\xi$ , e tendo este

parâmetro podemos calcular o valor do ângulo  $\theta$  das retas que limitam a região onde os valores de  $K$  (ganho do controlador) podem estar para que atenda a  $P.O.$  dada no projeto, está angulação  $\theta$  pode ser calculada fazendo  $\theta = \arccos(\xi)$ .

Figura 8 – Gráfico para Porcentagem de Overshoot  $P.O. \leq 16\%$ .



Fonte: (OGATA, 2010).

Nota-se na Figura 8 que a região hachurada representa a área do *Root-Locus* que atende a especificação de projeto para uma porcentagem de *overshoot* menor que 16%, e as retas em laranja, representa o lugar que atende a especificação de projeto para porcentagem de *overshoot* igual a 16%.

### 2.2.3 Tempo de estabelecimento $t_e$

A função  $y(t)$  apresentada em (14) é interpretada como sendo um sinal oscilatório com uma amplitude máxima que decresce com o passar do tempo, este fato pode ser observado na dedução do tempo de pico, sendo assim, o tempo de estabelecimento é o tempo que a saída leva para atingir uma oscilação menor que um certo critério, porcentagem da saída que se aproxima do valor constante onde os pontos de derivada nula da função  $y(t)$  ocorreram para  $t_i = \frac{K_i \pi}{\omega_d}$  onde  $K_i = 0, 1, 2, 3 \dots$  logo temos:

$$y(t) = K \left\{ 1 - e^{-\sigma t_i} \times [\cos(\omega_d \times t_i) + \frac{\sigma}{\omega_d} \times \text{sen}(\omega_d \times t_i)] \right\}. \quad (25)$$

Observa-se que o termo:

$$\cos(\omega_d \times t_i) + \frac{\sigma}{\omega_d} \times \text{sen}(\omega_d \times t_i) = \frac{1, \text{ se } k_i \text{ for par}}{-1 \text{ se } k_i \text{ for impar}}, \quad (26)$$

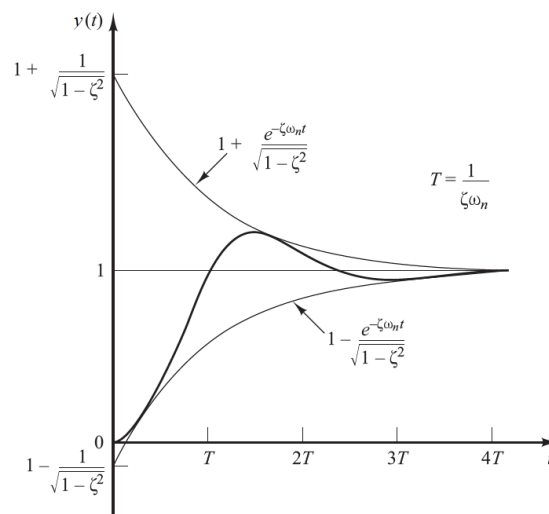
com a condição apresentada em (26), portanto (25) pode ser escrita como:

$$y(t) = K\{1 - e^{-\sigma t_i}\}, \quad (27)$$

$$y(t) = K\{1 + e^{-\sigma t_i}\}, \quad (28)$$

Isto implica em duas equações envoltórias exponenciais que limitam a resposta da função  $y(t)$  como apresentado na Figura 9.

Figura 9 – Par de curvas envoltórias de  $y(t)$ .



Fonte: (OGATA, 2010).

Portanto para determinarmos o tempo de estabelecimento  $t_e$  temos que igualar (27) e (28) ao valor assumido pela resposta  $y(t)$  que consideramos suficientemente pequeno para afirmar que o sistema entrou em regime permanente, sendo assim vamos assumir que um critério onde a saída  $y(t)$  está contida na faixa de  $\pm 1\%$  do valor de regime temos:

$$K \times (1 + 0.01) = K \times (1 + e^{-\sigma t_i}), \quad (29)$$

$$t_e = -\frac{\ln 0,01}{\sigma}. \quad (30)$$

Temos que  $\sigma = \omega_n \xi$ , então (30) fica:

$$t_e = -\frac{\ln 0,01}{\omega_n \xi} = \frac{4,6}{\omega_n \xi} \quad (\text{Critério } 1\%). \quad (31)$$

Podemos utilizar outros critérios como 2% ou 5%, logo (31) torna-se respectivamente:

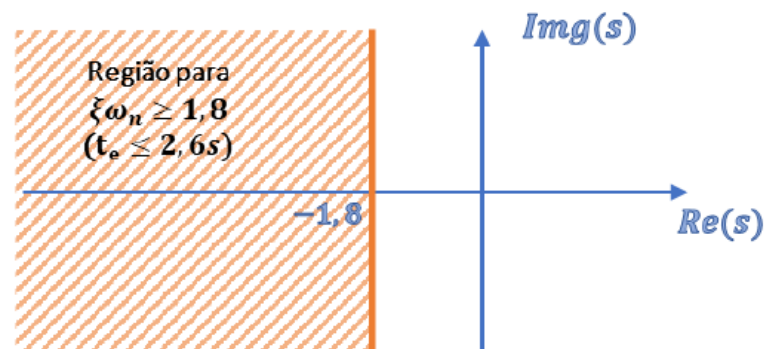
$$t_e = -\frac{\ln 0,02}{\omega_n \xi} = \frac{3,9}{\omega_n \xi} \quad (\text{Critério } 2\%), \quad (32)$$

$$t_e = -\frac{\ln 0,05}{\sigma \omega_n \xi} = \frac{3}{\omega_n \xi} \quad (\text{Critério } 5\%). \quad (33)$$

Nota-se que quanto maior for o coeficiente de amortecimento, menor será o tempo de estabelecimento.

O tempo de estabelecimento é um parâmetro de projeto muito importante, visto que, partindo do tempo de estabelecimento é possível se obter o valor de  $\omega_n \times \xi$ , que representa o ponto onde a reta que atende este tempo de estabelecimento se encontra no semiplano esquerdo do plano complexo, onde a reta intercepta o eixo real e vai de  $-\infty$  até  $+\infty$ . Caso o projeto exija tempos menores que um determinado valor, é necessário que o controlador assuma valores a esquerda desta reta, como pode ser observado no exemplo do *Root-Locus* da Figura 10. Figura 10 – Gráfico para  $t_e \leq 2,6s$  e critério de 1%, observa-se a região do *Root-Locus* que atende a especificação para um tempo de estabelecimento de 2,6 segundos com critério de 1%.

Figura 10 – Gráfico para  $t_e \leq 2,6s$  e critério de 1%.



Fonte: (ASSUNÇÃO; TEIXEIRA, 2018).

A área hachurada representa os lugares do *Root-Locus* que atende aos requisitos de projeto para quando o tempo de estabelecimento menor que 2,6 segundos, enquanto a reta vertical em laranja representa o lugar que atende o tempo de estabelecimento igual a 2,6 segundos. Portanto no projeto os polos do controlador devem estar na região hachurada para atender as especificações.

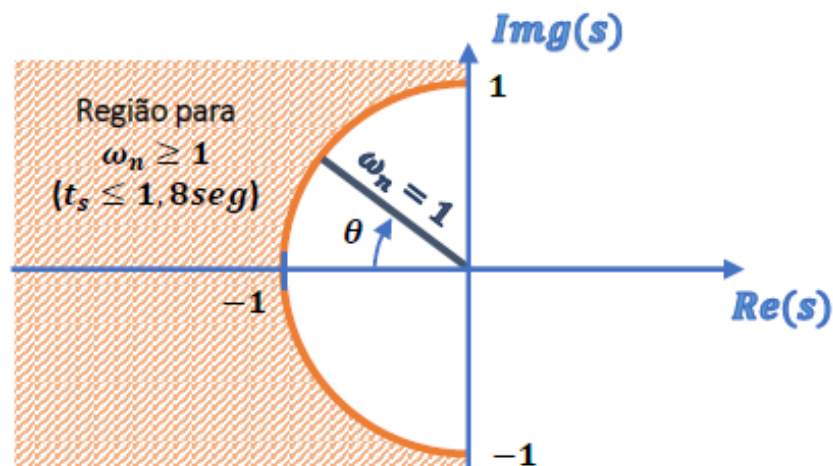
#### 2.2.4 Tempo de subida $t_s$

O tempo de subida  $t_s$  representa o tempo gasto para o sistema  $y(t)$  atingir 90% do seu valor de regime permanente, a equação para o tempo de subida também é obtida a partir de  $y(t)$  presente em (14), onde fazemos  $y(t) = 1$  e utilizando  $\xi = 0,5$  obtemos a seguinte expressão:

$$t_s = \frac{1,8}{\omega_n}. \quad (34)$$

No projeto utilizou-se (34) para o cálculo da frequência natural não-amortecida  $\omega_n$  referente ao tempo de subida que foi inserido pelo usuário. Quando especificamos um valor para  $t_s$ , o mesmo corresponde a um semicírculo de raio igual a  $\omega_n$  como pode ser visto na Figura 11.

Figura 11 – Gráfico com região que atende tempo de subida  $t_s \leq 1,8s$ .



Fonte: (ASSUNÇÃO; TEIXEIRA, 2018).

A região hachurada corresponde à área do *Root-Locus* que atende a solicitação de tempo de subida menor que 1,8s, já o círculo de raio  $\omega_n$  atende a especificação de projeto para um tempo de subida igual a 1,8s. Portanto no projeto os polos do controlador devem estar na região hachurada para atender as especificações.

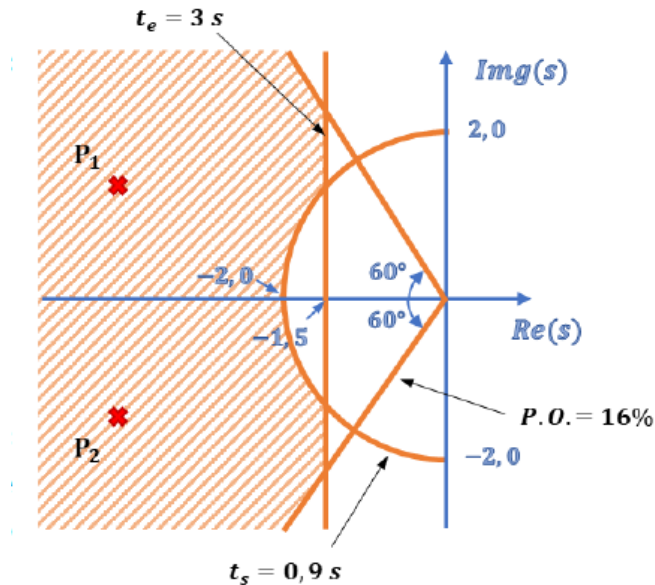
Podemos analisar todos estes critérios de projeto em um único gráfico e definir a região de operação que atenda a todas as exigências simultaneamente.

Por exemplo, dados os seguintes parâmetros de projeto:

- $t_s \leq 0,9s$  que resulta em  $\omega_n \geq 2$ ,
- $P.O. \leq 16\%$  que resulta em  $\xi \geq 0,5$ ,
- $t_e \leq 3s$  que resulta em  $\omega_n \xi \geq 1,5$ .

Obtemos a região que atende as três especificações ao mesmo tempo mostrado na Figura 12.

Figura 12 – Gráfico referente  $t_s \leq 0,9s$ ,  $P.O. \leq 16\%$ ,  $t_e \leq 3s$ .



Fonte: (ASSUNÇÃO; TEIXEIRA, 2018).

Podemos notar que os polos do controlador devem estar dentro da região hachurada para que atenda os três requisitos de projeto, caso os polos do controlador estejam fora da região mencionada, alguns dos critérios de projeto não serão atendidos, ou até mesmo os três critérios deixaram de ser contemplados no projeto.

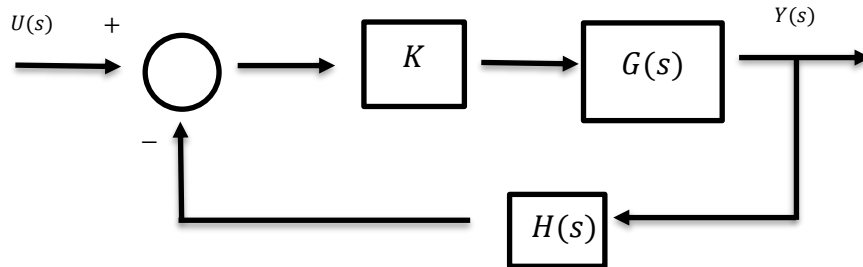
### 2.3 Root Locus

O método do lugar das raízes foi criado por Walter Richard Evans em 1953, tal método permite examinar as evoluções das raízes de uma equação quando um parâmetro do sistema, geralmente um ganho  $K$ , varia continuamente, e escolhendo de forma adequada o parâmetro que está sendo variado podemos fornecer a um dado sistema um comportamento dinâmico desejado, garantindo estabilidade, erro de regime permanente nulo entre outras características (ASSUNÇÃO; TEIXEIRA, 2018).

O método foi baseado em sistemas realimentados, como mostrado na Figura 13, podemos notar que a partir do diagrama pode-se encontrar a função de transferência de malha fechada  $\frac{Y(s)}{U(s)}$  e posteriormente, analisar a influência do ganho proporcional com relação ao lugar das raízes do polinômio.



Figura 13 – Diagrama de blocos para um sistema realimentado.



Adaptado de: (ASSUNÇÃO; TEIXEIRA, 2018).

Logo a função transferência de malha fechada é apresentada a seguir:

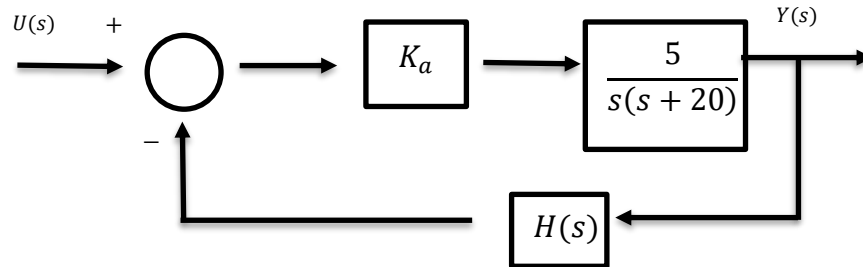
$$\frac{Y(s)}{U(s)} = \frac{K \times G(s)}{1 + K \times G(s) \times H(s)}. \quad (35)$$

O objetivo é ter regras simples para traçar o lugar geométrico formado pelas raízes do denominador  $1 + K \times G(s) \times H(s)$  quando o parâmetro  $K$  variar de 0 até  $+\infty$ , sem ter o conhecimento direto das raízes de malha fechada. Para tal vamos estudar (36):

$$1 + K \times G(s) \times H(s) = 0 \text{ para } 0 < K < +\infty. \quad (36)$$

Apresentamos a seguir um exemplo para demonstrar o modo como as raízes mudam de acordo com a variação do parâmetro  $K$ , e elucidar o funcionamento do método do Lugar das raízes, na Figura 14 é apresentado um diagrama de blocos para um sistema realimentado genérico.

Figura 14 – Diagrama de blocos para um sistema realimentado.



Adaptado de: (ASSUNÇÃO; TEIXEIRA, 2018).

De posse da função de transferência de malha fechada do sistema apresentado na Figura 14 encontramos (37) como segue:

$$\frac{Y(s)}{U(s)} = \frac{K_a \times \frac{5}{s(s+20)}}{1 + K_a \times 5s(s+20)} = \frac{5K_a}{s^2 + 20s + 5K_a}. \quad (37)$$

Partindo de (37) podemos encontrar os polos de  $\frac{Y(s)}{U(s)}$  como segue:

$$s_{1,2} = \frac{-20 \pm \sqrt{20^2 - 4 \times 5 \times K_a}}{2} = -10 \pm \sqrt{100 - 5 \times K_a}. \quad (38)$$

Variando-se o valor de  $K_a$  de 0 até  $\infty$ , é possível montar a Tabela 1 com os valores dos polos  $s_{1,2}$  em função de  $K_a$ .

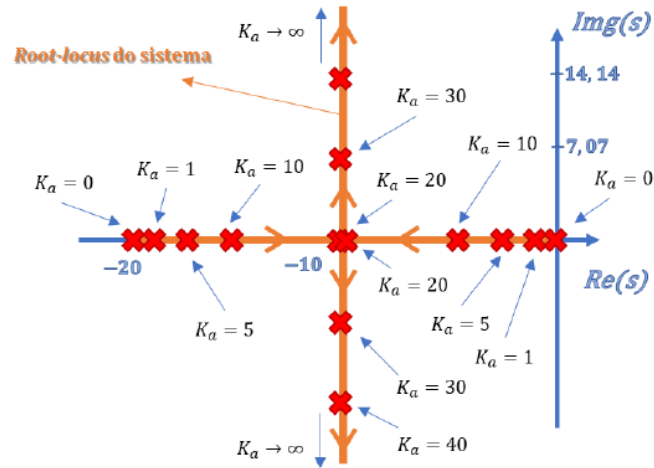
Tabela 1 – Variação dos Polos pela influência do ganho  $K_a$

$K_a$	$s_1$	$s_2$
0	-20	0
1	-19,75	-0,25
5	-18,66	-1,34
10	-17,07	-2,93
20	-10	-10
30	$-10 + j7,07$	$-10 - j7,07$
60	$-10 + j14,14$	$-10 - j14,14$
$+\infty$	$-10 + j\infty$	$-10 - j\infty$

Fonte: (ASSUNÇÃO; TEIXEIRA, 2018).

Diante dos valores obtidos na Tabela 1 podemos traçar o gráfico referente ao *Root-Locus* do nosso sistema genérico mostrado na Figura 14, e o resultado é apresentado Figura 15.

Figura 15 – Root-Locus para um sistema realimentado genérico.



Fonte: (ASSUNÇÃO; TEIXEIRA, 2018).

Observando a Figura 15 nota-se que conforme ocorre o aumento do ganho  $K_a$  as raízes do sistema aproximam-se até encontrar um ponto de partida neste caso  $-10$  e posteriormente cada uma das raízes irão para  $+\infty$  e  $-\infty$  respectivamente.

Walter Richard Evans propôs um método genérico para estes lugares geométricos, baseado em algumas regras simples, para conhecer o *Root-Locus* de um sistema sem a necessidade de determinar analiticamente o lugar das raízes do sistema (ASSUNÇÃO; TEIXEIRA, 2018).

A seguir listaremos algumas regras para a plotagem correta do *Root-Locus*:

Item N° 1 – Os ramos do *Root-Locus* começam nos polos de  $G(s) \times H(s)$ , nos quais  $K = 0$ . Os ramos terminam nos zeros de  $G(s) \times H(s)$ , inclusive zeros no infinito. O número de “zeros no infinito” é igual a  $N_{z\infty} = N_p - N_z$  sendo  $N_p$  o n° de polos de  $G(s) \times H(s)$  e  $N_z$  o n° de zeros de  $G(s) \times H(s)$  (DORF, 2001).

Regra N° 2 – As regiões do eixo real à esquerda de um número ímpar de polos mais zeros de  $K \times G(s) \times H(s)$  pertencem ao “*Root-Locus*” (DORF, 2001).

Item N° 3 – Quando  $k$  se aproxima de  $+\infty$ , os ramos do “*Root-Locus*” que tendem a infinito assintotam retas com inclinação  $\theta = 2i + N_p - N_z \times 180^\circ$ , para  $i=0, \pm 1, \pm 2, \dots, \pm (N_p - N_z - 1)$  sendo  $N_p$  o número de polos de  $G(s) \times H(s)$  e  $N_z$  o número de zeros de  $G(s) \times H(s)$  (DORF, 2001).

Item N° 4 – O ponto de partida das assíntotas é o centro de gravidade (C.G.) da configuração de polos e zeros de  $G(s) \times H(s)$ , ou seja: (DORF, 2001).

$$CG = \frac{\sum_{\text{polos}} - \sum_{\text{zeros}}}{N_p - N_z}. \quad (39)$$

Item N° 5 – Os pontos nos quais os ramos do “*Root-Locus*” deixam o eixo real são, ou entram no eixo real, são determinados a partir da relação: (DORF, 2001).

$$\frac{d}{ds} \left[ (G(s) \times H(s))^{-1} \right] = 0. \quad (40)$$

Item N° 6 – Duas raízes deixam ou entram no eixo real com ângulos  $\pm 90^\circ$  (DORF, 2001).

Item N° 7 – O “*Root-Locus*” é simétrico em relação ao eixo real (DORF, 2001).

Item N° 8 – O ganho  $K_p$  associado a um ponto  $P$  do “*Root-Locus*” pode ser obtido a partir da relação: (DORF, 2001).

$$K_p = \frac{1}{|G(s)H(s)|_{s=P}}. \quad (41)$$

Item N° 9: Os ângulos de saída de polos e chegada aos zeros, são determinados a partir da condição geral de ângulo (DORF, 2001).

Item N° 10 – O ponto onde o “*Root-Locus*” cruza o eixo imaginário é obtido fazendo-se  $s = j\omega$  na equação característica (DORF, 2001).

Item N° 11 – Se pelo menos dois ramos do “*Root-Locus*” vão para o infinito (ou seja, se existem ao menos 2 assíntotas), então a soma dos polos de malha fechada correspondentes a um mesmo valor de  $K$  é uma constante independente de  $K$  (DORF, 2001).

Sendo assim apresentadas todas as 11 regras para a determinação do “*Root-Locus*”. Por mais simples que seja implementar as regras do “*Root-Locus*”, leva-se um tempo considerável para realizar estes cálculos, calcular as derivadas, resolver o modulo, ângulo das assíntotas, ponto de partida e ponto de chegada, centro de gravidade e finalmente desenhar a região que representa o Lugar Geométrico das Raízes do sistema.

Considerando que na realização do projeto, será necessário projetar o controlador por tentativa e erro de acordo com o “*Root-Locus*” gerado, que de tal forma torna-se uma tarefa incomoda e pode acarretar eventuais erros de cálculos durante o processo.

Foi pensando em agilizar o processo de análise de controladores que implementou-se este *script*, de modo que o projetista fosse capaz de informar os critérios de projeto, informar o polinômio do numerador e denominador da função de malha fechada do sistema e obter como

resposta os parâmetros  $(\xi, \omega_n, e \xi \times \omega_n)$  referente as especificações fornecidas pelo usuário  $(P.O., t_e, Qual o Critério para o cálculo, t_s)$  e também obter o “*Root-Locus*” do seu sistema de controle em estudo, já com as regiões que atendem as especificações do projeto.

## 2.4 Espaço de estados

Até o presente momento, foi abordado a teoria de controle clássico que está fundamentada na relação de entrada-saída por meio de uma função de transferência. Entretanto em sistemas modernos mais complexos, certamente haverá múltiplas entradas e saídas, que podem ou não estar inter-relacionadas de maneira complexa ou não, logo a teoria clássica não contempla estes casos, para analisar estes sistemas mais complexos é de extrema importância reduzir a complexidade de suas expressões matemáticas e recorrer à *softwares* matemáticos para realizar os procedimentos necessários para análise de sistemas mais complexos (OGATA, 2010).

Sendo assim, a teoria de controle moderno é baseada na descrição de um sistema complexo de equações em termos de  $n$  equações diferenciais de primeira ordem que podem ser combinadas em equações diferenciais vetorial-matricial de primeira ordem, que acaba simplificando muito a representação deste sistema de equações (OGATA, 2010).

Quando temos o aumento do número de variáveis de estado, ou seja, um aumento no número de entradas e saídas, não aumenta a complexidade das equações, visto que, estamos utilizando a representação vetorial-matricial.

Visando trabalhar com o projeto de controlador para o módulo da suspensão ativa de bancada, onde trata-se de um sistema de controle com um grau mais elevado de especificação e precisão, onde exige uma modelagem da planta, um projeto de controlador que atenda todos os limitantes físicos e de performance, para tal caso foi realizado a descrição do sistema em espaço de estados no qual pode ser expresso como apresentado:

$$\dot{x}(t) = Ax(t) + Bu(t), \quad (42)$$

$$\dot{y}(t) = Cx(t) + Du(t). \quad (43)$$

Sendo a entrada de controle  $u(t) \in \mathbb{R}^m$ , os estados do sistema são dados por  $x(t) \in \mathbb{R}^n$ , o vetor de saída do sistema dado por  $y(t) \in \mathbb{R}^p$ ,  $A \in \mathbb{R}^{n \times n}$  a matriz de estados,  $B \in \mathbb{R}^{n \times m}$  a matriz de entrada,  $C \in \mathbb{R}^{p \times n}$  a matriz de saída e  $D \in \mathbb{R}^{p \times m}$  a matriz de alimentação direta.

Os sistemas apresentados são descritos por equações diferenciais ordinárias e são apresentadas como mostrado em (44) e (45).

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} + \begin{bmatrix} b_1(t) \\ b_2(t) \\ \vdots \\ b_n(t) \end{bmatrix} u(t), \quad (44)$$

$$y(t) = [c_1 \quad c_2 \quad \cdots \quad c_n] \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}. \quad (45)$$

## 2.5 Análise da estabilidade segundo Lyapunov

Na teoria de controle existem vários critérios para a análise da estabilidade de um sistema descritos na literatura, podemos citar o Critério de Nyquist, Critério de Routh Hurwitz e o método de Lyapunov.

No âmbito do estudo de sistemas de controle, temos como principal objetivo alcançar a estabilidade do sistema. O Método de Lyapunov para sistemas não lineares e/ou lineares no tempo, é um dos possíveis métodos que pode ser utilizado, sendo válido para sistemas de qualquer ordem.

A estabilidade segundo Lyapunov pode ser entendida como sendo um sistema inicial, onde neste sistema encontramos nossas condições iniciais que são as entradas do nosso sistema  $x(t)$ . Então Lyapunov definiu uma candidata a função de energia  $V$ , e está candidata a função energia depende da nossa entrada, logo, nossa função energia se torna  $V(x(t))$ .

Esta função energia  $V(x(t))$  já começa com uma energia inicial  $V(x(t)) > 0$  e para que o sistema seja estável é necessário que o sistema perca esta energia e atinja o ponto de equilíbrio ou seja  $V(x(t)) = 0$ . Para o sistema perca esta energia se faz necessário que a derivada da função energia seja negativa  $\dot{V}(x(t)) < 0$ , a derivada da função energia negativa, nos indica que o sistema está perdendo energia, caso essa perda de energia chegue a zero temos um sistema assintoticamente estável, agora se esta energia atingir um certo valor menor que seu valor inicial e permaneça neste determinado valor temos um sistema estável.

Logo se  $\dot{V}(x(t)) < 0$  temos um sistema assintoticamente estável e caso  $\dot{V}(x(t)) \leq 0$  o sistema é estável. Neste trabalho será considerado como candidata a função de Lyapunov  $V(x(t)) = x^T P x > 0$ .

Para que seja possível compreender melhor a análise de estabilidade segundo o método de Lyapunov vamos definir alguns conceitos.

Primeiramente vamos considerar um sistema que não tenha uma entrada forçada, ou seja,  $u(t) = 0$ , resultando (42) como mostrado a seguir:

$$\dot{x}(t) = Ax(t). \quad (46)$$

Vamos as definições:

Estado de Equilíbrio: Podemos dizer que um vetor constante  $x_e$  está em estado de equilíbrio do sistema apresentado em (46), caso  $\dot{x}(t) = Ax(t) = 0$ , ou seja, todas as derivadas das variáveis estudadas são nulas, logo as variáveis de estado não estão variando no tempo, então, o estado do sistema permanece em  $x_e$  e este é denominado estado de equilíbrio ou ponto de equilíbrio do sistema.

Estabilidade no Sentido de Lyapunov: Considere uma região de raio  $K$  definida em torno deste nosso estado de equilíbrio  $x_e$ , que é representada como segue:

$$\|x - x_e\| \leq K,$$

onde  $\|x - x_e\|$  é a norma euclidiana, que é definida por:

$$\|x - x_e\| = [(x_1 - x_{1e})^2 + (x_2 - x_{2e})^2 + \dots + (x_n - x_{ne})^2]^{\frac{1}{2}},$$

considere que  $S(\zeta)$  seja a região que consiste de todos os pontos sendo:

$$\|x_0 - x_e\| \leq \zeta,$$

onde  $x_0$  é o estado inicial e  $S(\zeta)$  a região que consiste em todos os pontos para os quais:

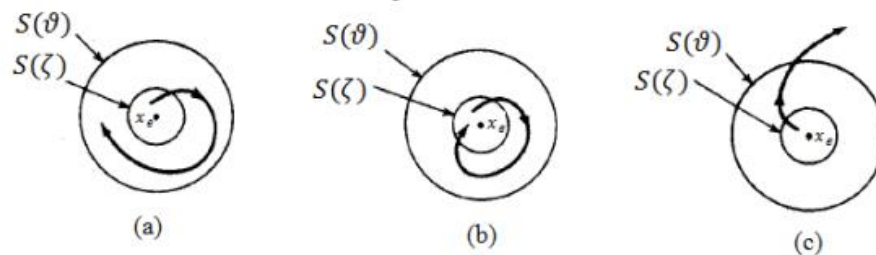
$$\|x - x_e\| \leq \vartheta, \forall t > 0.$$

Um estado de equilíbrio  $x_e$  do sistema mostrado em (46) é estável segundo Lyapunov se, correspondendo a cada  $S(\vartheta)$ , houver uma  $S(\zeta)$  tal que as trajetórias iniciadas em  $S(\zeta)$  não deixarem  $S(\vartheta)$  à medida que  $t$  aumenta indefinidamente.

Estabilidade Assintótica: um estado de equilíbrio  $x_e$  do sistema representado em (46) é dito assintoticamente estável se for estável no sentido de Lyapunov e se toda solução começando em  $S(\zeta)$  converge para  $x_e$  sem deixar  $S(\vartheta)$  à medida que  $t$  aumenta indefinidamente.

Instabilidade: um estado de equilíbrio  $x_e$  é dito instável se, para algum número real  $\vartheta > 0$  e qualquer número real  $\zeta > 0$ , não importando sua dimensão, sempre há um estado  $x_0$  em  $S(\zeta)$  tal que a trajetória começando neste ponto deixa a região  $S(\vartheta)$ .

Figura 16 – (a) Estado de Equilíbrio Estável e uma trajetória representativa; (b) Estado de Equilíbrio Estável Assintótico e uma trajetória representativa; (c) Estado de Equilíbrio Instável e uma trajetória representativa.



Fonte: Adaptado de (OGATA, 2010).

Vamos definir a seguir alguns conceitos adicionais como sinais de funções e forma quadrática de funções escalares que nos auxiliarão no estudo algébrico.

Função Escalar Definida Positiva: uma função escalar  $V(x)$  é dita definida positiva em uma região  $\Omega$  se  $V(x) > 0$  para todos os estados não nulos  $x$  na região  $\Omega$  e  $V(0) = 0$ .

Função Escalar Definida Negativa: uma função escalar  $V(x)$  é dita definida negativa em uma região  $\Omega$  se  $-V(x)$  é definida positiva.

Função Escalar Semidefinida Positiva: uma função escalar  $V(x)$  é dita semidefinida positiva se for positiva em quase todos os estados de uma região  $\Omega$ , exceto na origem e em outros estados de  $\Omega$  onde seu valor é zero.

Função Escalar Semidefinida Negativa: uma função escalar  $V(x)$  é dita semidefinida negativa em uma região  $\Omega$  se  $-V(x)$  é semidefinida positiva.

Função Escalar Indefinida: uma função escalar  $V(x)$  é dita indefinida se na região  $\Omega$  assume tanto valor positivo quanto negativo, independente da dimensão da região  $\Omega$ .

Forma Quadrática de Funções Escalares: as formas quadráticas ou também conhecidas como composta constituem de uma classe de funções escalares que desempenham um papel importante na análise de estabilidade, possibilitando formular inúmeros problemas de controle através de LMIs sendo que esta classe de função pode ser descrita por:



$$V(x) = x^T P x = [x_1 \ x_2 \ \dots \ x_n] \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \quad (47)$$

Onde  $x \in \mathcal{R}^n$  é um vetor e  $P \in \mathcal{R}^{n \times n}$  uma matriz real e simétrica.

Podemos verificar se a forma quadrática é ou não definida positiva utilizando o critério de Sylvester, que estabelece como condição necessária e suficiente para que a forma quadrática  $V(x)$  descrita em (47) seja definida positiva, que todos os menores principais sucessivos de  $P$  sejam positivos, ou seja,

$$P_{11} > 0, \det \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} > 0, \dots, \det \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{bmatrix} > 0,$$

ou ainda, basta verificar se todos os autovalores de  $P$  são positivos, caso sejam, pode concluir que  $P$  é dita simétrica e definida positiva.

Analisando a estabilidade segundo Lyapunov, considera-se o sistema linear proposto na em (46), sendo  $x(t)$  o vetor de estado e  $A$  uma matriz constante e não-singular, que implica em ( $\det(A) \neq 0$ ), dessa forma, o único estado de equilíbrio do sistema é a origem  $x_e = 0$ , logo definimos uma função de Lyapunov para esse sistema:

$$V(x(t)) = x(t)^T P x(t), \quad (48)$$

onde  $P \in \mathcal{R}^{n \times n}$  uma matriz simétrica e definida positiva. Tendo como derivada de  $V(x(t))$  em relação ao tempo:

$$\begin{aligned} \dot{V}(x(t)) &= \dot{x}(t)^T P x(t) + x(t)^T P \dot{x}(t) = (Ax(t))^T P x(t) + x(t)^T P (Ax(t)), \\ \dot{V}(x(t)) &= x(t)^T (A^T P + PA)x(t). \end{aligned} \quad (49)$$

Como  $V(x(t))$  foi escolhida sendo definida positiva (sempre positiva para  $x \neq 0$ ) para garantir a estabilidade assintótica é preciso garantir que  $\dot{V}(x(t))$  seja definida negativa:

$$\dot{V}(x(t)) = -x(t)^T Q x(t), \quad Q > 0, \quad (50)$$

ao compararmos (49) e (50):

$$-Q = (A^T P + PA),$$

ou

$$A^T P + PA = -Q, \text{ com } P = P^T > 0 \text{ e } Q > 0.$$

A partir dos resultados apresentados chegamos ao Teorema 1:

**Teorema 1** (BOYD et al., 1994): *considere o sistema descrito por  $\dot{x}(t) = Ax(t)$ , sendo  $x(t)$  o vetor de estado e  $A$  uma matriz não-singular ( $n \times n$ ) constante e com elementos reais. Uma condição necessária e suficiente para que o estado de equilíbrio  $x_e = 0$  seja assintoticamente estável é que exista uma matriz  $P$  simétrica definida positiva tal que:*

$$A^T P + PA < 0. \quad (51)$$

Quando consideramos o sistema linear apresentado em (46), onde  $A$  possui determinante não nulo, o que implica em um único estado de equilíbrio na origem, pode-se definir a candidata a função de Lyapunov como:

$$V(x(t)) = x(t)^T P x(t), \quad (52)$$

sendo  $P \in \mathbb{R}^{n \times n}$  uma matriz simétrica definida positiva, onde a candidata a função de Lyapunov do sistema  $V(x(t)) > 0$  que representa a energia do sistema, considerando que inicialmente o sistema possui energia positiva e visando atingir a estabilidade, tal energia deve ser decrescente, que é uma condição que pode ser constatada pela derivada de  $V(x(t))$  com relação ao tempo, que deverá ser definida negativa:

$$\dot{V}(x(t)) = \dot{x}(t)^T P x(t) + x(t)^T P A x(t), \quad (53)$$

substituindo (46) em (52) temos:

$$\dot{V}(x(t)) = (Ax(t))^T P x(t) + x(t)^T P A x(t), \quad (54)$$

simplificando (54), obtemos:

$$\dot{V}(x(t)) = x(t)^T (A^T P + PA) x(t). \quad (55)$$

Devido ao fato de  $V(x(t))$  ser definida positiva, este sistema só será estável se tiver comportamento decrescente. Então, é necessário que  $\dot{V}(x(t))$  seja definida negativa:

$$\dot{V}(x(t)) < 0, \quad (56)$$

para tal é necessário que:

$$(A^T P + PA) < 0. \quad (57)$$

## 2.6 Realimentação de Estados

Assumindo que o sistema apresentado em (42) seja um sistema linear invariante no tempo (SLIT) controlável. O projeto de controladores com realimentação de estados é realizado

de modo a encontrar uma matriz de ganho  $K \in \mathbb{R}^{n \times m}$ , sendo que este sistema seja assintoticamente estável quando realimentado com a entrada de controle apresentada a seguir:

$$u(t) = -Kx(t), \quad (58)$$

substituindo (58) em (42), temos:

$$\dot{x}(t) = Ax(t) - BKx(t), \quad (59)$$

simplificando temos:

$$\dot{x}(t) = (A - BK)x(t). \quad (60)$$

**Teorema 2** (BOYD et al., 1994): *considere o sistema descrito em (42). Caso exista uma matriz  $M \in \mathbb{R}^{m \times n}$  e uma matriz simétrica definida positiva  $X \in \mathbb{R}^{n \times n}$  tais que:*

$$XA^T - M^T B^T + XA - BM < 0, \quad (61)$$

*então a lei de controle apresentada em (58), sendo  $K = MX^{-1}$ , torna o sistema realimentado apresentado em (59) estável.*

**Prova:** substituindo (58) em (42) temos:

$$\dot{x}(t) = (A - BK)x(t). \quad (62)$$

A partir de (62) podemos inferir a nova matriz  $A$  do sistema, que será  $A_n = (A - BK)$ .

Substituindo (62) em (57), obtém-se a seguinte LMI:

$$(A - BK)^T P + P(A - BK) < 0, \quad (63)$$

$$A^T P - K^T B^T P + PA - PBK < 0, \quad (64)$$

$$P > 0 \rightarrow P = P^T. \quad (65)$$

Observando (63), temos uma *Bilinear Matrix Inequalities* (BMIs), portanto é necessário realizar manipulações de maneira a torná-la uma *Linear Matrix Inequalities* (LMIs).

Multiplicando as desigualdades apresentadas em (64) e (65) à esquerda e à direita por  $P^{-1}$ , fazendo  $X = P^{-1}$ ,  $M = KP^{-1}$  e  $M^T = P^{-1}K^T$  obtém-se:

$$XA^T - M^T B^T + XA - BM < 0, \quad (66)$$

$$X > 0 \rightarrow X = X^T. \quad (67)$$

A manipulação algébrica resultou na transformação da BMI obtida anteriormente em LMI, o que torna possível a resolução das LMI utilizando os *softwares* estudados (MATLAB® e Python) para projetar um ganho  $K$  que estabilize o sistema.

Sendo (66) e (67) factíveis é possível obter um controlador que estabiliza o sistema realimentado por:

$$K = MX^{-1}.$$

## 2.7 Taxa de decaimento

A taxa de decaimento tem relação com o tempo de estabilização do sistema, onde, quanto maior for a taxa de decaimento  $\gamma$ , menor será o tempo de estabilização do sistema.

Considerando o sistema controlado mostrado em (60), temos que a taxa de decaimento será definida como uma constante positiva  $\gamma$ , tal que:

$$\lim_{t \rightarrow \infty} e^{\gamma t} \|x(t)\| = 0, \quad (68)$$

e ainda se mantenha em todas as trajetórias  $x(t)$ ,  $t > 0$  (BOYD et al., 1994).

Devido a função quadrática de Lyapunov podemos determinar um limite inferior sobre a taxa de decaimento do sistema representado por (60) como apresentado:

$$V(x(t)) = x(t)^T P x(t), \quad (69)$$

$$\dot{V}(x(t)) \leq -2\gamma V(x(t)). \quad (70)$$

Para todas as trajetórias, tem-se o seguinte teorema:

**Teorema 3** (BOYD et al., 1994): *considere o sistema descrito por (60). Caso exista um escalar  $\gamma > 0$ , uma matriz  $M \in \mathbb{R}^{m \times n}$  e uma matriz simétrica definida positiva  $X \in \mathbb{R}^{n \times n}$  tais que*

$$XA^T - M^T B^T + AX - BM + 2\gamma X < 0, \quad (71)$$

$$X > 0, \quad (72)$$

*então a lei de controle apresentada em (60). sendo  $K = MX^{-1}$ , torna o sistema (60) estável e com taxa de decaimento igual a  $\gamma$ .*

## 2.8 Parâmetros incertos para sistemas com realimentação de estados

Quando se projeta um controlador, deve-se levar em conta parâmetros incertos que podem possuir características inerentes à planta estudada ou podem ser resultado de uma falha apresentada no atuador, sendo estes problemas tratados através do conceito de estabilidade quadrática adicionados as LMIs.

Seja um sistema sujeito a falhas:

$$\dot{x}(t) = A(\alpha)x(t) + B(\alpha)u(t), \quad (73)$$

sendo  $x(t) \in \mathbb{R}^n$  o vetor de estado do sistema,  $u(t) \in \mathbb{R}^m$  o sinal de controle e  $A(\alpha) = \alpha_i A_i$ ,  $B(\alpha) = \alpha_i B_i \in \mathbb{R}^{n \times m}$ , com  $\sum_{i=1}^r \alpha_i = 1$  e  $\alpha_i \geq 0$ ,  $i \in \{1, \dots, r\}$ .

A lei de controle representado em (58), torna o sistema:

$$\dot{x}(t) = (A(\alpha) - B(\alpha)K)x(t). \quad (74)$$

Assumindo a lei de controle apresentada em (60) e que o sistema está sujeito a uma taxa de decaimento  $\gamma > 0$ , tem-se o seguinte teorema:

**Teorema 4** (BOYD et al., 1994): *considere o sistema descrito por (74). Caso exista um escalar  $\gamma > 0$ , uma matriz  $M \in \mathbb{R}^{m \times n}$  e uma matriz simétrica definida positiva  $X \in \mathbb{R}^{n \times n}$  tais que:*

$$XA_i^T - M^T B_i^T + A_i X - B_i M + 2\gamma X < 0, \quad (75)$$

$$X > 0, \quad (76)$$

para todo  $i \in \{1, \dots, r\}$  a lei de controle descrita por (58), sendo  $K = MX^{-1}$ , torna o sistema descrito em (60) estável e com taxa de decaimento maior ou igual a  $\gamma$ .

**Prova:** Considere como candidata à função de Lyapunov (52), sua derivada em relação ao tempo dado por (53), a lei de controle representada em (58) e o sistema descrito por (73), obtém-se:

$$\begin{aligned} \dot{V}(x(t)) = & x(t)^T \left( \sum_{i=1}^r \alpha_i A_i - \sum_{i=1}^r \alpha_i B_i K \right)^T P x(t) \\ & + x(t)^T P \left( \sum_{i=1}^r \alpha_i A_i - \sum_{i=1}^r \alpha_i B_i K \right) x(t), \end{aligned} \quad (77)$$

$$\dot{V}(x(t)) = x(t)^T \left[ \left( \sum_{i=1}^r \alpha_i A_i - \sum_{i=1}^r \alpha_i B_i K \right)^T P + P \left( \sum_{i=1}^r \alpha_i A_i - \sum_{i=1}^r \alpha_i B_i K \right) \right] x(t). \quad (78)$$

Considerando a taxa de decaimento e substituindo (70) em (78), realizando as devidas simplificações temos:

$$\left( \sum_{i=1}^r \alpha_i A_i - \sum_{i=1}^r \alpha_i B_i K \right)^T P + P \left( \sum_{i=1}^r \alpha_i A_i - \sum_{i=1}^r \alpha_i B_i K \right) < 2\gamma P,$$

como  $\sum_{i=1}^r \alpha_i = 1$ , pode-se assumir:

$$(A_i - BK)^T P + P(A_i - B_i K) < 2\gamma P.$$

Multiplicando por  $P^{-1}$  pela esquerda e pela direita e substituindo  $X = P^{-1}$  e  $M = KM$ :

$$XA_i^T - M^T B_i^T + A_i X - B_i M + 2\gamma X < 0. \quad (79)$$

Sendo (79) factível para  $i \in \mathbb{K}_r$ , o controlador é dado por  $K = MX^{-1}$ .

## 2.9 Restrição no sinal de controle do atuador

Sempre que necessário realizar o projeto de um controlador, deve-se levar em consideração o valor do máximo sinal de controle no qual o atuador será exposto, visto que, em

muitos casos este sinal pode ser elevado o que poderia dificultar ou até mesmo acabar inviabilizando a implementação do controlador, portanto segue abaixo um teorema que propõe a redução do sinal de controle no qual o atuador será aplicado.

**Teorema 5** (TANAKA, 2001): *considerando que as condições apresentadas no Teorema 4 sejam satisfeitas, a restrição  $KK^T < \mu I_m$ , onde  $\mu$  é uma constante positiva e  $I_m$  a matriz identidade de ordem  $m$ , é atendida caso as LMIs:*

$$\begin{bmatrix} 1 & x^T(0) \\ x(0) & X \end{bmatrix} > 0 \text{ e } \begin{bmatrix} X & M^T \\ M & \mu \end{bmatrix} > 0, \quad (80)$$

Sejam satisfeitas, onde  $\mu = I_m * U$ , em conjunto com a LMI apresentada em (75).

**Prova:**

$$\begin{bmatrix} 1 & x^T(0) \\ x(0) & X \end{bmatrix} > 0, \quad (81)$$

$$1 - x^T(0)X^{-1}x(0) > 0 \rightarrow x^T(0)Px(0) < 1, \quad (82)$$

$$\begin{bmatrix} X & M^T \\ M & \mu \end{bmatrix} > 0, \quad (83)$$

$$X - \frac{1}{\mu}M^T M > 0 \rightarrow -X + \frac{1}{\mu}M^T M < 0, \quad (84)$$

multiplicando (84) pela direita e pela esquerda por  $P$  temos:

$$-PXP + \frac{1}{\mu}PM^T MP < 0, \quad (85)$$

sendo:

$$MP = KXP \rightarrow MP = K, \quad (86)$$

substituindo (86) em (85), ficamos com:

$$-P + \frac{1}{\mu}K^T K < 0, \quad (87)$$

multiplicando (87) pela esquerda por  $x^T$  e pela direita por  $x$ :

$$-x^T Px + \frac{1}{\mu}x^T K^T K x < 0, \quad (88)$$

$$\frac{1}{\mu}x^T K^T K x < x^T Px, \quad (89)$$

sendo  $\mu = Kx$  e  $\mu^T = K^T x^T$  e substituindo em (89):

$$\frac{1}{\mu}u^T u < x^T Px, \quad (90)$$

comparando (90) com (81), temos que:

$$\frac{1}{\mu} u^T u < x^T P x < x^T(0) P x(0) < 1, \quad (91)$$

Portanto

$$\frac{1}{\mu} u^T u < 1, \quad (92)$$

$$u^T u < \mu. \quad (93)$$

Logo a partir de (93), temos que o valor do maior sinal de controle será  $\sqrt{\mu}$ .

## 2.10 Controlador Chaveado

Apresentaremos a seguir a teoria sobre o projeto de controlador chaveado, onde tem-se o projeto de um controlador chaveado para Sistema Linear Invariante no Tempo SLIT.

Temos então que a função de Lyapunov do tipo quadrática representada por:

$$V(x(t)) = x(t)^T P x(t), \quad (94)$$

sendo que  $P = P^T > 0$ .

Quando seguimos a lei de controle chaveada temos como principal característica a minimização da derivada temporal da função de Lyapunov, por meio da seleção do ganho do controlador, ou seja, a partir de um argumento mínimos o controlador optará por um conjunto de ganhos ou por outro conjunto de ganhos que melhor atenda a solicitação do sistema. Esta seleção de ganhos do controlador pertence ao conjunto de ganhos  $K_j, j \in \mathbb{K}_s$  (BUZETTI, 2017).

Nesta lei de chaveamento são utilizadas matrizes auxiliares simétricas  $\bar{Q}_j, j \in \mathbb{K}_s$ .

Define-se o controlador chaveado da seguinte forma:

$$u(t) = -K_\sigma x(t), \quad (95)$$

$$\sigma(t) = \arg \min_{j \in \mathbb{K}} \{x^T(t) \bar{Q}_j x(t)\}. \quad (96)$$

Logo  $\sigma(t)$  tem o valor do índice  $j$  que resulta no menor valor de  $x^T \bar{Q}_j x(t)$  e no caso de dois ou mais  $\bar{Q}_j$  produzirem o menor valor de  $x^T \bar{Q}_j x(t)$ , então  $\sigma$  assumirá o menor valor do índice  $j$ .

**Teorema 6** (Souza et. al., 2013): *Suponha a existência de uma matriz simétrica positiva definida  $X \in \mathbb{R}^{n \times n}$ , matrizes simétricas  $Z_i, Q_j \in \mathbb{R}^{n \times n}$ , matrizes  $M_j \in \mathbb{R}^{m \times n}$  e um escalar  $\beta \geq 0$ , para todo  $i, j \in \mathbb{K}_s$  tais que:*

$$-B_i M_j - M_j^T B_i^T - Z_i - Q_j < 0, \quad (97)$$

$$X A_i^T + A_i X + Z_i + Q_i + 2\beta X < 0. \quad (98)$$

Então a lei de controle chaveada expressa em (96) torna o ponto de equilíbrio  $x_e = 0$  do sistema apresentado em (62) globalmente assintoticamente estável com uma taxa de decaimento maior ou igual a  $\beta$ , sendo  $P = X^{-1}$ ,  $\bar{Q}_j = X^{-1} Q_j X^{-1}$  e os ganhos do controlador dados por  $K_j = M_j X^{-1}$ .

## 2.11 Suspensão Ativa de Bancada

Como neste projeto o intuito foi verificar a possibilidade de projetar controladores chaveados utilizando a plataforma de programação Python, se fez necessário utilizar uma planta como caminho para verificar o desempenho da plataforma escolhida e compará-la com os resultados obtidos no MATLAB®.

Atualmente as montadoras de automóveis vem aumentando a tecnologia embarcada e melhorando o desempenho dos componentes do veículo. Um exemplo é a suspensão ativa dos automóveis que de modo geral utiliza sensores para a coleta de dados e essas informações identificam as condições da via e do veículo, para indicar os movimentos que devem ser empregas pelo conjunto da suspensão do carro de modo a armazenar, dissipar e introduzir energia no sistema, trazendo um maior conforto ao usuário.

Neste trabalho será utilizado o módulo da suspensão ativa de bancada, que é composto por uma estrutura que simula um quarto do conjunto de suspensão de um automóvel (QUANSER, 2018).

A planta escolhida para o desenvolvimento do trabalho foi o modelo da suspensão ativa de bancada, no qual a universidade dispõe de um módulo didático do sistema de suspensão ativa de bancada QUANSER®, que foi desenvolvido para simular um quarto de um veículo controlado por um mecanismo ativo de controle.

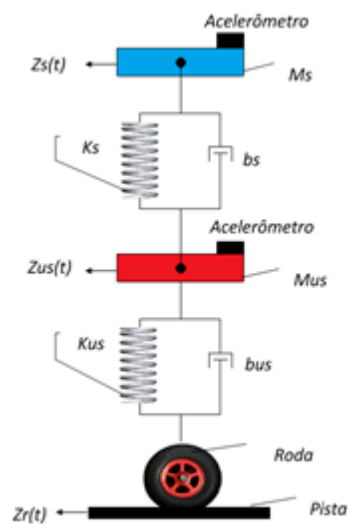
O módulo consiste em três placas sobrepostas, sendo intercaladas por molas e um atuador elétrico. A placa superior retrata o corpo do veículo e está separada da placa intermediária por duas molas, juntamente com as molas existe um motor de corrente contínua entre a placa superior e a intermediária, que é o componente ativo do sistema de suspensão ativa (D. R. de Oliveira, G. R. dos Santos, M. C. Teixeira, E. Assunção, R. Cardim 2018).



Também é disponibilizado na placa superior um acelerômetro que mede a aceleração desta placa com relação à planta da pista. As molas, que fazem a ligação entre a placa intermediária e a placa inferior, simulam a dinâmica do pneu. A placa inferior simula a pista que o carro percorre que é conectada a um motor de corrente contínua de resposta rápida, o que permite a simulação de vários perfis de pista quando é realizada uma implementação.

O objetivo principal deste sistema de controle é suavizar os impactos causados por possíveis imperfeições no trajeto que o automóvel percorre e garantir um conforto maior ao usuário do veículo. A seguir é apresentado um esquema referente ao módulo da suspensão ativa de bancada QUANSER®, com as respectivas constantes associadas ao sistema.

Figura 17 – Modelo de  $\frac{1}{4}$  da suspensão ativa de um automóvel.



Fonte: (BETETO, ASSUNÇÃO, TEIXEIRA, SILVA, BUZACHERO, CAUN, 2018).

A Tabela 2 apresenta os parâmetros mostrados na Figura 17 e que são utilizados no modelo matemático da dinâmica do sistema.

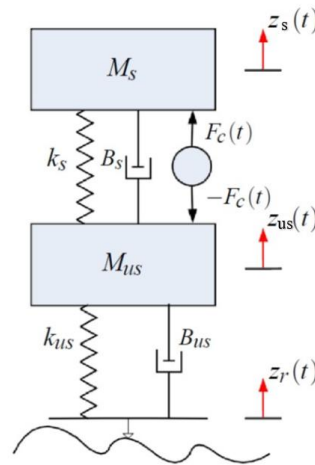
Tabela 2 – Parâmetros do modelo da suspensão ativa.

Parâmetro	Descrição	Valor
$M_s$	Representação $\frac{1}{4}$ da massa do veículo	2,45 kg
$M_{us}$	Representa as massas do conjunto pneu veículo	1 kg
$K_s$	Constante da mola que se encontra entre as duas placas	900 N/m
$b_s$	Constante do amortecedor que se encontra entre as duas placas	7,5 [N/m]s
$K_{us}$	Constante da mola que se encontra embaixo da placa intermediária	2500 N/m
$b_{us}$	Constante do amortecedor que se encontra embaixo da placa intermediária	5 [N/m]s
$Z_s(t)$	Varição no nível da carroceria do automóvel ao longo do tempo	-
$Z_{us}(t)$	Varição no nível dos pneus ao longo do tempo	-
$Z_r(t)$	Variações da pista ao longo do tempo	-

### 2.12 Modelo Matemático Suspensão Ativa de Bancada

Este sistema que representa  $\frac{1}{4}$  da suspensão ativa de um veículo pode ser modelado como um duplo sistema massa-mola-amortecedor. A sua representação em espaço de estados considera como entrada de controle a força realizada pelo atuador e com perturbação a derivada da superfície da estrada (QUANSER, 2010).

Figura 18 – Duplo sistema massa-mola-amortecedor usado para modelar o Sistema de Suspensão.



Fonte: Adaptado de (QUANSER, 2010A).

A Figura 18 é uma representação do sistema no qual é objetivo de estudo. As coordenadas generalizadas no canto esquerdo da Figura 18 representam:  $Z_s(t)$  o movimento do corpo do veículo,  $Z_{us}(t)$  o movimento da roda e  $Z_r(t)$  a superfície da pista.

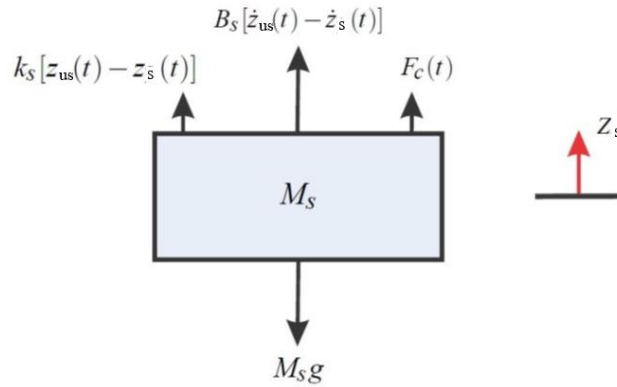
O modelo no espaço de estados desse sistema pode ser determinado através do diagrama de corpo livre de cada parte do veículo.

A seguir na Figura 19 é representado o diagrama de corpo livre do corpo do veículo  $M_s$ , para o movimento da coordenada generalizada  $Z_s$ , tem-se:

$$\ddot{Z}_s(t) = -g + \frac{F_c(t)}{M_s} + \frac{B_s \dot{Z}_{us}(t)}{M_s} - \frac{B_s \dot{Z}_s(t)}{M_s} + \frac{k_s(t) Z_{us}(t)}{M_s} - \frac{k_s(t) Z_s(t)}{M_s}, \quad (99)$$

sendo  $g$  a aceleração da gravidade.

Figura 19 – Diagrama de corpo livre para a massa  $M_s$ .

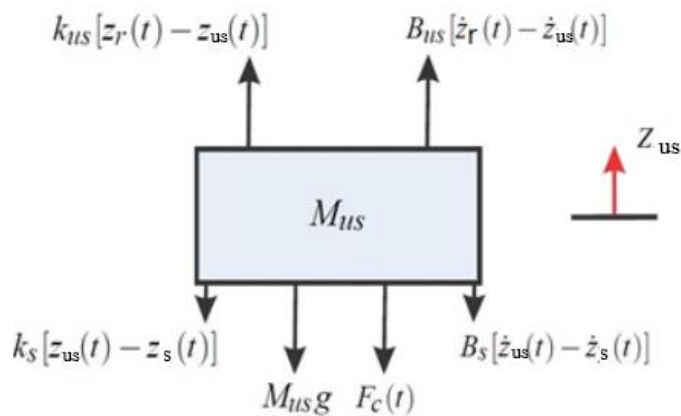


Fonte: Adaptado de (QUANSER, 2010A).

Já na Figura 20, temos o diagrama de corpo livre para a massa do pneu,  $M_{us}$ , com relação ao movimento  $Z_{us}(t)$ , temos:

$$\begin{aligned} \ddot{Z}_{us} = & -g - \frac{F_c(t)}{M_{us}} - \frac{(B_s + B_{us})\dot{Z}_{us}(t)}{M_{us}} + \frac{B_s\dot{Z}_s(t)}{M_{us}} + \frac{B_{us}\dot{Z}_r(t)}{M_{us}} \\ & - \frac{(k_{us}+k_s)Z_{us}(t)}{M_{us}} + \frac{k_s Z_s(t)}{M_{us}} + \frac{k_{us}Z_r(t)}{M_{us}}. \end{aligned} \quad (100)$$

Figura 20 – Diagrama de corpo livre para a massa  $M_{us}$ .



Fonte: Adaptado de (QUANSER, 2010A).

As coordenadas generalizadas do sistema foram escolhidas para a situação em que as molas  $k_s$  e  $k_{us}$  estão relaxadas, seria o caso em que  $Z_{us}(t) = Z_s = 0$ , entretanto, isto não ocorre

devido ao fato de que as molas nunca se encontram nessa condição devido ao peso das massas  $M_s$  e  $M_{us}$ . Logo a gravidade influencia no ponto de equilíbrio ( $Z_{us_e}(t)$ ,  $Z_{s_e}(t)$ ).

Os pontos de equilíbrio podem ser calculados lembrando-se de que no equilíbrio as derivadas de qualquer ordem de  $Z_{us}(t)$  e  $Z_s(t)$  são nulas, e partindo de (99) e (100), ficamos apenas com:

$$Z_{us_e}(t) = -\frac{(M_s+M_{us})g}{k_{us}}, \quad (101)$$

$$Z_{s_e}(t) = -\frac{M_s g}{k_s} - \frac{(M_s+M_{us})g}{k_{us}}. \quad (102)$$

Fazendo algumas mudanças de variáveis em (99) e (100), obtém-se:

$$\ddot{Z}_s(t) = \frac{1}{M_s} [F_c(t) + B_s \dot{Z}_{us}(t) - B_s \dot{Z}_s(t) + k_s(t)Z_{us}(t) - k_s(t)Z_s(t)], \quad (103)$$

$$\begin{aligned} \ddot{Z}_{us}(t) = \frac{1}{M_{us}} [-F_c(t) - (B_s + B_{us})\dot{Z}_{us}(t) + B_s \dot{Z}_{us}(t) + B_{us} \dot{Z}_r(t) \\ - (k_{us} + k_s(t))Z_{us}(t) + k_s Z_s(t) + k_{us} Z_r(t)], \end{aligned} \quad (104)$$

e, com estas alterações, a gravidade não aparece explicitamente nas equações de movimento do sistema.

Escolhendo como variáveis de estado  $x_1(t) = Z_s(t) - Z_{us}(t)$ , deflexão da suspensão;  $x_2(t) = \dot{Z}_s(t)$ , a velocidade vertical do corpo do veículo;  $x_3(t) = Z_{us}(t) - Z_r(t)$ , deflexão do pneu e  $x_4(t) = \dot{Z}_{us}(t)$  a velocidade vertical do conjunto roda. Com entrada  $u(t)$  a força exercida pelo controlador  $F_c(t)$  e como distúrbio  $d(t)$  a velocidade do perfil pista  $\dot{Z}_r(t)$ .

Ficamos com a seguinte representação no espaço de estados do sistema de suspensão ativa de bancada:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & -1 \\ -\frac{k_s}{M_s} - \frac{B_s}{M_s} & 0 & \frac{B_s}{M_s} & \\ 0 & 0 & 0 & 1 \\ \frac{k_s}{M_{us}} & \frac{B_s}{M_{us}} & \frac{K_{us}}{M_{us}} & -\frac{(B_s+B_{us})}{M_{us}} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M_s} \\ 0 \\ -\frac{1}{M_{us}} \end{bmatrix} u(t). \quad (105)$$

## 2.13 Resolução LMIs com Matlab®: Yalmip, LMILab e Sedumi

**Yalmip:** é um pacote (*toolbox*) que possibilita que problemas envolvendo programação matemática sejam representadas de maneira mais natural no MATLAB®, este pacote pode ser utilizado em várias situações como por exemplo: problemas de programação linear,

programação inteira e mista, problemas de programação semi-definida e em desigualdades matriciais bilineares.

Uma das grandes vantagens deste pacote é que ele apresenta suporte para vários *solvers*, o Yalmip permite resolver LMIs utilizando *solver* como o LMILab (*solver* do LMI *control toolbox*) ou também com Sedumi.

**LMILab:** é um *solver* de LMIs padrão do MATLAB®. Este *solver* diferente do Sedumi resolve apenas LMIs, assim, para projetos que desejam apenas resolver LMIs ele se torna preferível a princípio, visto que, ele atenderá as solicitações durante a solução das LMIs. O LMILab vem com o pacote *Robust Control Toolbox* do MATLAB. Neste trabalho o *solver* utilizado foi o LMILab.

## 2.14 Resolução de LMIs com Python: Picos e CVXOPT

PICOS é uma API (Application Programming Interface) Python amigável para vários solucionadores de programação cônica e inteira, muito parecido com YALMIP ou CVX no MATLAB®. Ele permite inserir um problema de otimização matemática como um modelo de alto nível, com suporte indolor para variáveis vetoriais e matriciais (complexas) e álgebra multidimensional. Seu modelo será transformado para a forma padrão compreendida por um solucionador apropriado que está disponível em tempo de execução. Isso torna seu aplicativo portátil, pois os usuários podem escolher entre vários solucionadores comerciais e de código aberto.

**CVXOPT** é um pacote de *software* livre para otimização convexa baseado na linguagem de programação Python. Ele pode ser usado com o interpretador Python interativo, na linha de comando, executando *scripts* Python, ou integrado em outro *software* por meio de módulos de extensão Python. Seu principal objetivo é tornar o desenvolvimento de *software* para aplicativos de otimização convexa simples, construindo na extensa biblioteca padrão do Python e nos pontos fortes do Python com uma linguagem de programação de alto nível. CVXOPT é o pacote do *solver* que corresponderia ao LMILab ou Sedumi no MATLAB®.

### 3 DESENVOLVIMENTO

O primeiro passo para a criação e/ou aprimoramento dos códigos, foi estudar novamente a teoria e as estruturas já presentes e estudados na disciplina de Controle Linear I e controle Linear II, para relembrar os conceitos teóricos e enquadrá-los no processo de construção dos códigos em Python.

#### 3.1 Primeiro projeto Root Locus

Durante a graduação tivemos contato com disciplina de Controle Linear I e também a disciplina optativa Computação Científica com Python, durante a disciplina de controle foram realizados vários projetos de controladores analógicos utilizando a técnica “*Root-Locus*” e buscando otimizar o método de projeto dos controladores, foi proposto a implementação de um *script* capaz de plotar o “*Root-Locus*” de um sistema qualquer, e juntamente com o “*Root-Locus*”, plotar a região que atende a solicitação de projeto especificada pelo usuário, trazendo uma ganho significativo na agilidade do projeto e evitando erros grosseiros de cálculo.

Para tal feito utilizou-se a plataforma Python para a realização do projeto.

#### 3.2 Apresentação e descrição de cada parte do código

Para a utilização do software se faz necessário a instalação do pacote *python-control* que nada mais é que um pacote onde ocorre a implementação de operações básicas para análise e projeto de sistemas de controle realimentados.

O pacote *python-control* faz uso do *NumPy* que se trata de uma biblioteca para a linguagem Python, adicionando suporte para matrizes e *arrays* multidimensionais grandes, junto com uma grande coleção de funções matemáticas de alto nível para operar tais *arrays* e *SciPy* que é uma biblioteca Python gratuita e de código aberto usada para computação científica e computação técnica. *SciPy* contém módulos para otimização, álgebra linear, integração, interpolação, funções especiais, *Fast Fourier transform* (FFT), processamento de sinais e imagens, solucionadores de *Ordinary differential equation* (ODE) e outras tarefas comuns na ciência e engenharia.

Segue processo de instalação:

Para instalar usando o pip

```
pip install slycot
pip install control
```

Para usuários anaconda se faz necessário a instalação no *Prompt*, abra o Anaconda

*Prompt* e instale:

```
conda install numpy scipy matplotlib
conda install -c python-control -c cyclus slycot control
```

Sendo que qualquer problema com a instalação pode ser verificado em:

<https://python-control.readthedocs.io/en/0.9.0/intro.html#installation>

Com o pacote *Python-Control* instalado vamos agora apresentar as bibliotecas utilizadas para a plena execução do *software*, garantindo que todas as funções utilizadas trabalhem de maneira adequada.

```
%matplotlib notebook
import control
import control.matlab as mt
import scipy
from IPython.display import clear_output
from fractions import Fraction as frac
import matplotlib.pyplot as plt
import numpy as np
import math as mat
```

Tendo em vista que o programa utilizaria alguns parâmetros globais e que seriam específicos como por exemplo a porcentagem de *overshoot*, o tempo de estabelecimento, o critério utilizado para o tempo de estabelecimento e o tempo de subida, criou-se uma classe com esses parâmetros, e dentro da classe de parâmetros é possível acessar qualquer uma das informações que eram fornecidas pelo usuário, destas citadas.

O bloco responsável pela criação da classe foi implementado como mostrado a seguir:

```
#Criando Classe sobre os parametros porcentagem de overshoot, tempo de estabelecimento, tempo de subida
ListaParametros=[]
class Parametros:
    def __init__(self, PO, Crit, te, ts):
        self.PO = PO          #porcentagem de overshoot
        self.Crit = Crit      #critério de análise da porcentagem do tempo de estabelecimento
        self.te = te         #tempo de estabelecimento
        self.ts = ts         #tempo de subida do sistema
        return
```

Também foi implementada uma função chamada “limite”, tal função permite que mesmo ocorrendo um erro no *software*, a mesma não deixaria que ocorresse uma interrupção



durante a execução, em vez de quebrar, ela retorna o erro para o usuário e continua compilando sem interromper o programa.

A seguir é apresentado o bloco de implementação referente a Função limite:

*#Função definida em aula pelo docente, e utilizada para fazer com que mesmo que aconteça um erro indesejado o programa não quebre.*

```
def limite (f):
    def aux(*x,**y):
        try:
            r = f(*x,**y)
            return r
        except Exception as e: #retorna o erro ao usuario mas não quebra o programa
            print (e)
    return aux
```

Também foi necessário utilizar uma função para converter os valores inseridos pelo usuário no formato de *string* para o formato *float*, visto que, os valores a serem utilizados nos cálculos são valores com utilização de ponto flutuante, sendo assim implementou-se a função que converte *string* para *float*.

*#função criada para converter os valores digitados na entrada do programa para float*

```
@limite
def converterfloat(x):
    flag = False
    while flag == False:
        try:
            ValorConvertido = float(frac(x))
            flag = True
            return ValorConvertido
        except:
            x=input('\033[1m' + '\n Por Favor, digite uma entrada válida!\n' + '\033[0m')
```

Outra função muito importante é chamada de “EntradaParametos”, esta função faz o controle dos dados que o usuário está inserindo referente aos parâmetros de projeto, porcentagem de *overshoot* (*P.O.*), tempo de subida ( $t_s$ ), tempo de estabelecimento ( $t_e$ ) e o critério utilizado para o cálculo de  $\xi \times \omega_n$  para o tempo de estabelecimento.

Nesta função também são verificados se os parâmetros digitados pelo usuário, são valores condizentes com a uma certa faixa de valores pré-determinados, não podendo passar desses valores, caso o usuário tente passar, o programa retorna ao usuário e pede novamente a inserção do dado que foi introduzido de forma incorreta.

**@limite**

```

#função utilizada para a aquisição dos parametro do projeto!
def EntradaParametros():
    print('\033[1m' + '\t \t Digite os valores de entradas pedidos. \n' + '\033[0m')
    controle = False #controle utilizado para verificação se o valor digitado foi realmente válido e dentro dos padro
es de aceitação definidos no programa
    while controle == False:
        PO = converterfloat(input('\033[1m' + 'Digite a Porcentagem de overshoot desejado em porcentagem %: ' + '\03
3[0m')) #faz a aquisição do valor da porcentagem de overshoot em na variavel PO
        if PO<=0 or PO>100 : #faz a verificação se a Porcentegem de Overshoot está na faixa entre 1 e 100%
            print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t Insira os valores em porcentagem no dominio'
                + 'de [1 a 100%]) \n' + '\033[0m') # caso não esteja informa ao usuario e pede novamente o valor
        else:
            controle = True
    controle = False
    while controle == False:
        Crit = converterfloat(input('\033[1m' + 'Digite o critério utilizado para o calculo do Tempo de estabelecimento '
+ '\033[0m'))#faz a aquisição do critério utilizado para o calculo na PO
        if Crit<=0 or Crit>100 : #faz a verificação se a Critério está na faixa entre 1 e 100%
            print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t Insira os valores de criterio em porcentagem'
                + 'de [1 - 100]) \n' + '\033[0m)# caso não esteja informa ao usuario e pede novamente o valor
        else:
            controle = True
    controle = False
    while controle == False:
        ts = converterfloat(input('\033[1m' + 'Digite o Tempo de subida: ' + '\033[0m'))#faz a aquisição do valor do te
mpo de subida do projeto
        if ts<=0 or ts>500 :#faz a verificação se o tempo de estabelecimento está na faixa de [0.0000001 - 500]
            print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t Insira os valores de tempo de subida dentro do '
                + 'intervalo de [0.0000001 - 500]) \n' + '\033[0m)# caso não esteja informa ao usuario e pede novamente
o valor
        else:
            controle = True
    controle = False
    while controle == False:
        te = converterfloat(input('\033[1m' + 'Digite o Tempo de estabelecimento desejado: ' + '\033[0m'))#faz a aquisi
ção do valor do tempo de estabelecimento do projeto
        if te<=0:
            print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t Digite apenas valores de tempo em segundo,'
                + 'valores positivos de tempo! \n' + '\033[0m)# caso não esteja informa ao usuario e pede novamente o v
alor
        else:

```

```

    controle = True
    controle = False
    ListaParametros.insert(0,Parametros(PO, Crit, te,ts)) # Insera os valores informados na lista de parametros.
    return

```

Em seguida foi necessário fazer a aquisição do polinômio do numerador da função  $G(s) \times H(s)$  informados pelo usuário, para fazer a aquisição destes dados foi necessário solicitar ao usuário que o mesmo informe o grau (ordem) do polinômio que será digitado, para posteriormente pegar os valores do coeficiente de cada termo referente ao seu grau, por exemplo:

Para um polinômio de grau 2 do tipo  $(s^2 + 3s + 5)$ , inicialmente o usuário informa que a ordem do polinômio é 2, e em seguida informa os coeficientes do polinômio neste exemplo [ 1, 3, 5 ], sempre do maior grau para o menor grau. A função que realizou este passo foi a função “Numerador”:

```

@limite
#função que faz a aquisição do polinomio do numerador da função de transferência.
def Numerador():
    print('\033[1m' + '\n \t \t Digite os valores do polinomio do numerador de G(s)*H(s) \n' + '\033[0m')
    controle = False #faz o controle da aquisição de dados
    while controle == False:
        print('\033[1m' + '\n \t \t O Grau do polinômio é dado pelo expoente de maior grau. '
            + '\n \t \t Por exemplo: s^2 -2s polinomio de grau 2 com coeficientes [1 -2 0] \n' + '\033[0m')
        print('\033[31m' + '\033[1m'+Atenção a Ordem (grau) do polinômio do numerador, obrigatoriamente tem que ser inferior\n '
            + ' a ordem (grau) do polinômio do denominador. ' + '\033[0m'+'\nPor exemplo: numerador [s+5] grau=1, denominador '
            + '[s^2 2s +1] grau=2 \n' + '\033[0m')
        #breve explicação de como os valores deverão ser inseridos pelo usuario
        numNum = converterfloat(input('\033[1m' + 'Digite a ordem (grau) do polinomio do numerador: ' + '\033[0m'))
        #faz a aquisição do Grau do polinomio que sera digitado.
        if numNum<0 or numNum>10 : # Verifica se o valor digitado para o grau do polinomio está dentro do intervalo de 0 a 10
            print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t ATENÇÃO A ORDEM (GRAU) DO POLINOMIO DEVE ESTAR '
                + '\nNA FAIXA ENTRE: [0 - 10]) \n' + '\033[0m')
        else:
            cont=numNum # atribui a um contador o valor do grau do polinomio informado pelo usuário
            while cont>=0: #enquanto o contador for maior ou igual a zero vai ficar pedindo os valores do coeficiente referente a cada grau do polinomio
                num = converterfloat(input('Digite o coeficiente correspondete á ordem %i: '%(float(cont))))# faz a aquisição do coeficiente do polinomio associado ao seu valor de "ordem"

```

```

if num<-1000 or num>1000 : #faz uma verificação para que nao seja utilizados valores astronomicos
    print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t Atenção os valores dos coeficientes dos p
olinomios '
        +'devem estar na faixa de: [-1000 até 1000]) \n' + '\033[0m')#caso o coeficiente esteja fora do interval
o avisa o usuário
else:
    ListaNumerador.append(num) #salva cada valor adquirido a posição na lista do numerador, p
osteriormente esta lista sera usada para plotar o grafico do Root-Locus
    cont=cont-1 #Decrementa o contador para poder perga o proximo coeficiente!
    controle = True #muda o controle para sair do while
return

```

O mesmo processo é repetido para o denominador, para fazer a aquisição dos coeficientes do denominador inicialmente foi necessário solicitar ao usuário que digite o grau do polinômio do denominador, vale aqui observar que o sistema não pode ter mais polos do que zeros, logo o grau do polinômio do denominador tem que ser no mínimo um grau superior ao polinômio do numerador.

Após o usuário inserir o grau do denominador o código exige que o usuário informe o coeficiente de cada ordem do polinômio, novamente do maior grau para o menor grau.

Veja um exemplo:

Para um polinômio no denominador de grau 3 do tipo  $(2s^3 + 8s^2 + s)$ , inicialmente o usuário informa que se trata de um polinômio de grau 3, e logo após informa os coeficientes do polinômio, seguindo a ordem sempre do maior grau para o menor, que neste caso resulta em [ 2, 8, 1, 0], nota-se que mesmo o coeficiente do termo  $s^0$  sendo zero, se faz necessário colocar o numeral zero no denominador para completar o polinômio. O código que implementou essa função é o “Denominador”.

@limite

*#função que faz a aquisição do polinomio do denominador da função de transferência.*

def Denominador():

```
print('\033[1m' + '\n \t \t Digite os valores do polinomio do denominador de G(s)*H(s) \n' + '\033[0m')
```

```
controle = False #faz o controle da aquisição de dados
```

```
while controle == False:
```

```
print('\033[1m' + '\n \t \t O Grau do polinômio é dado pelo expoente de maior grau. '
```

```
    +'\n \t \t Por exemplo: s^2 -2s polinomio de grau 2 com coeficientes [1 -2 0] \n' + '\033[0m')
```

```
print('\033[31m' + '\033[1m'+Atenção a Ordem (grau) do polinômio do numerador obrigatoriamente tem que s
er inferior '
```

```
    +' a ordem (grau) do polinômio do denominador. ' + '\033[0m'+\n Por exemplo: numerador [s+1], denomi
nador '
```

```

        + '[s^2 2s +1] \n' + '\033[0m')
        #novamente uma breve explicação de como os valores deverão ser inseridos pelo usuario
        numDen = converterfloat(input('\033[1m' + 'Digite a ordem (grau) do polinomio do denominador: ' + '\033[0m'
    )) #faz a aquisição do Grau do polinomio que sera digitado.
        if numDen<0 or numDen>10 :    # Verifica se o valor digitado para o grau do polinomio está dentro do inter
    valo de 0 a 10
            print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t ATENÇÃO A ORDEM (GRAU) DO POLINO
    MIO DEVE ESTAR '
                + 'NA FAIXA ENTRE: [0 - 10]) \n' + '\033[0m')
        else:
            cont=numDen    # atribui a um contador o valor do grau do polinomio informado pelo usuário
            while cont>=0:    #enquanto o contador for maior ou igual a zero vai ficar pedindo os valores do coeficient
    e referente a cada grau do polinomio
                den = converterfloat(input('Digite o coeficiente correspondete á ordem %i: %(float(cont))))
                if den<-1000 or den>1000 :
                    print('\033[31m' + '\033[1m'+\tObservação: ' + '\033[0m'+\t Atenção os valores dos coeficientes dos p
    olinomios '
                        + 'devem estar na faixa de: [-1000 até 1000]) \n' + '\033[0m')
                else:
                    ListaDenominador.append(den)    #salva cada valor adquirido a posição na lista do denominador, pos
    teriormente esta lista sera usada para plotar o grafico do Root-Locus
                    cont=cont-1    #Decrementa o contador para poder perga o proximo coeficiente!
                    controle = True    #muda o controle para sair do while
            return

```

Com os valores dos parâmetros e os coeficientes do numerador e do denominador adquiridos, foi necessário criar uma função para fazer os cálculos dos parâmetros necessários para a plotagem dos gráficos, foi essencial também definir a ordem dos parâmetros para que o gráfico do “Root-Locus” (Função “rl” no código) fosse traçado. Nesta função os parâmetros  $\xi$ ,  $\omega_n$ ,  $\xi \times \omega_n$ ,  $\theta$ , *coeficiente de reta (m)*, foram calculados para serem plotados no gráfico, foi definida a função de transferência do numerador e do denominador, nesta função também foram inseridos os valores calculados na “ListaParâmetrosCalculados”.

#### Função rl:

```

        #Criando uma função que faça o Calculo do Root Locus e dos parametros de projeto Porcentagem de Overshoot, C
    ritério da Porcentagem de Overshoot, tempo de subida, tempo de estabelecimento,  $\xi$ ,  $\omega_n$ ,  $\xi\omega_n$  e  $\theta$ 
        @limite
        def rl(num,den,PO,Crit,te):    #define os parametros que serão utilizados na função rl ( Root-
    Locus)
            E = np.sqrt((np.log(PO/100)*2)/((np.log(PO/100)*2)-np.pi*2))    #Função que calcula o valor de  $\xi$ 
            EWn = -np.log(Crit/100)/te    #Função que calcula o valor de  $\xi\omega_n$ 

```

```

Wn = 1.8/ListaParametros[0].ts          #Função que calcula o valor de  $\omega_n$ 
theta=np.arccos(E)                      #Função que faz o cálculo de  $\theta$  Theta
sys = mt.tf(num, den)                   #Define a Função de transferência
mt.rlocus(sys, PrintGain=True)          #Esboça o Root-Locus
plt.grid(True)
plt.title('Root-Locus')
plt.ylabel('Eixo Imaginário')
plt.xlabel('Eixo Real')

coeficienteReta = np.tan(np.arccos(E))   #calcula o coeficiente m da reta que sera plotada com a an
gulação theta
f = np.arange(-1000,0,0.01)             #Cria um vetor que vai de 0 a -1000 com passo 0.01
g = coeficienteReta*f                   #Multiplica o valor do coeficiente da reta pelo vetor
plt.plot(f,g,'y', label='Porcentagem de Overshoot') #plota a reta no quarto quadrante
plt.plot(f,abs(g),'y')                   #Plato a reta no terceiro quadrante
y = np.arange(-1000,1000,0.01)          #Cria um vetor que vai de -1000 até 1000 com passo 0.01
x = -EWn*np.ones(len(y))                #faz um produto de  $\zeta\omega_n$  vezes um vetor de uns do tamnho de y
plt.plot(x,y,'B', label='Tempo de Estabelecimento') # Plato a reta referente a  $\zeta\omega_n$ 
plt.plot(0,'w', label='Demais cores Root-Locus\nX-Polos O-Zeros') # plot usado para adicionar uma lege
nda
plt.legend(loc='best')

plt.show()
ListaParametrosCalculados.insert(0,Wn)   #insere o valor de  $\omega_n$  na posição 0 da ListaParametrosCa
lculados
ListaParametrosCalculados.insert(1,E)    #insere o valor de  $\zeta$  na posição 1 da ListaParametrosCalcu
lados
ListaParametrosCalculados.insert(2,EWn)  #insere o valor de  $\zeta\omega_n$  na posição 2 da ListaParametros
Calculados
ListaParametrosCalculados.insert(3,theta) #insere o valor de  $\theta$  Theta na posição 0 da ListaParametr
osCalculados
return

```

Definiu-se também uma função auxiliar chamada “Calculo”, que implementa os valores calculados novamente na “ListaParametrosCalculados”, Função “Calculo”:

```

#Função utilizada para fazer os calculos de projeto Porcentagem de Overshoot, Critério da Porcentagem de Overs
hoot, tempo de subida, tempo de estabelecimento,  $\zeta$ ,  $\omega_n$ ,  $\zeta\omega_n$  e  $\theta$ 
#função utilizada para apresentar os calculos separado do grafico, onde só se apresenta os parametros calculados
def Calculo(PO,Crit,te,ts):
    E = np.sqrt((np.log(PO/100)*2)/((np.log(PO/100)*2)-np.pi*2))
    EWn = -np.log(Crit/100)/te

```

```

Wn = 1.8/ListaParametros[0].ts
#theta=((360*(np.arccos(E)))/(2*np.pi))
theta=mat.acos(E)*360/(2*np.pi)
ListaParametrosCalculados.insert(0,Wn)
ListaParametrosCalculados.insert(1,E)
ListaParametrosCalculados.insert(2,EWn)
ListaParametrosCalculados.insert(3,theta)
Return

```

Por fim foi necessário organizar todas as informações em um menu de decisões para facilitar o acesso do usuário, esse menu foi feito utilizando um simples *while*, como pode ser observado no bloco do menu:

```

#Corpo do programa onde são gerenciados os menus principais
print('\033[1m+\n Software que calcula o Root-Locus de um Polinômio.'+'\033[0;0m \n')
controle=True #Controle do menu
while controle==True: #enquanto o controle for verdadeiro ele executa o menu
    controle2=True #mexi
    print('\033[1m+\n \t \t \v Menu de Opções:'+'\033[0;0m')
    print('\033[1m+ \t \v [E]' + '\033[0m" - "\033[4m+ 'Entrar no Software.' + '\033[0m'+\t'+\033[1m'
        + '[S]' + '\033[0m" - "\033[4m+ 'Sair Do Software.\n' + '\033[0m')
    menu = input ('Digite uma opção para entrada ou saída do software:') #entrada do menu 1
    if menu == 'E' or menu == 'e': #primeira decisão se sera entrada no software
        clear_output() #limpa as saidas
        EntradaParametros() #carrega os parametros colhidos na função entrada de Parametros
        ListaNumerador=[] #carrega os parametros colhidos na função Numerador
        ListaDenominador=[] #carrega os parametros colhidos na função Denominador
        ListaParametrosCalculados=[] #carrega os parametros colhidos na função Parametros Calculados
        Numerador()
        Denominador()
        while controle2==True: #Faz o controle do segundo menu
            print('\033[1m+\n \t \t \v Menu de Opções:'+'\033[0;0m')

            print('\033[1m+ \t \v [P]' + '\033[0m" - "\033[4m+ 'Parametros Do circuito' + '\033[0m'+\t'+\033[1m'
                + '[G]' + '\033[0m" - "\033[4m+ 'Plotar o Gráfico\n' + '\033[0m'+\033[1m+ \t \v [V]' + '\033[0m" - "\033
[4m+ 'Voltar ao Menu Anterior' + '\033[0m')
            MenuGrafParam = input ('Digite se você deseja conferir os parametros digitados e os parametros calculados,
'+ ou se quer vizualizar o grafico e finalizar o programa : \n \n') #entrada do menu 2
            if MenuGrafParam == 'P' or MenuGrafParam == 'p': #segunda decisão, se o usuario quer ver os parametros c
alculados
                clear_output()
                Calculo(ListaParametros[0].PO,ListaParametros[0].Crit,ListaParametros[0].te,ListaParametros[0].ts)
                print('Apresentamos a seguir oa parametros que foram calculados com os dados inseridos:')

```

```

        print('Para uma Porcentagem de Overshoot de %3.1f %%.'%(float(ListaParametros[0].PO)))
        print('Calculou-se um coeficiente de amortecimento \033[1m+' ξ'+\033[0;0m= %3.4f. \n'%(float(ListaP
arametrosCalculados[1])))
        print('Com o coeficiente de amortecimento \033[1m+' ξ'+\033[0;0m de %3.4f.'%(float(ListaParametros
Calculados[1])))
        print('Calculou-se o angulo θ entre o eixo real e a reta que define a região do potencial de overshoot \033[
1m+'θ'+\033[0;0m= %3.4f : \n'%(float(ListaParametrosCalculados[3])))
        print('Para um tempo de estabelecimento de %3.1f, e critério de estabelecimento de %3.1f,'%(float(Lista
Parametros[0].te),float(ListaParametros[0].Crit)))
        print('Calculou-se um coeficiente de amortecimento vezes frequência natural não-amortecida \033[1m+' ξ
ωn'+\033[0;0m= %3.4f. \n'%(float(ListaParametrosCalculados[2])))
        print('Para o tempo de subida estabelecido em %3.1f segundos'%(float(ListaParametros[0].ts)))
        print('Calculou-se uma frequência natural não amortecida \033[1m+' ωn'+\033[0;0m= %3.4f . \n'%(float
(ListaParametrosCalculados[0])))
        elif MenuGrafParam=='G' or MenuGrafParam=='g': #Ou se o usuário quer ver o 'Root-Locus'
            rl(ListaNumerador,ListaDenominador,ListaParametros[0].PO,ListaParametros[0].Crit,ListaParametros[0]
.te) #Função que plota o Root-Locus
            controle=False
            controle2=False
        elif MenuGrafParam=='V' or MenuGrafParam=='v': #Volta ao menu anterior
            controle2=False
            clear_output()
        else: #caso algum parametro seja digitado errado pede entrada valida
            clear_output()
            print('Voce digitou uma entrada invalida.\n \033[1m+'Por Favor Digite uma opção válida'+\033[0;0m ')
    elif menu=='S' or menu=='s': #se não sair do software
        controle=False
        clear_output()
    else: #caso algum parametro seja digitado errado pede entrada valida
        clear_output()
        print('Voce digitou uma entrada invalida.\n \033[1m+'Por Favor Digite uma opção válida'+\033[0;0m ')

```

### 3.3 Segundo projeto LMIs

Após execução do primeiro código em Python que possibilitou a análise e o projeto de controladores PID através da análise do *Root Locus*, surgiu a ideia de ampliar criando um novo código agora para controladores robustos, de tal forma que o código resolveria problemas de LMIs para o projeto de controladores chaveados.



O projeto de controladores chaveados que é feito resolvendo as LMIs da planta já foi amplamente estudado utilizando o *software* proprietário MATLAB®, e transcrever este código para um *software* de código aberto como Python seria de grande importância para o desenvolvimento, visto que, não dependemos mais de uma licença para utilização.

Porém o caminho para desenvolver o novo código que implementa as mesmas funções já conhecidas no MATLAB® em *software* aberto não é tão simples como parece. A nova plataforma por ser de código aberto trabalha com sistemas de bibliotecas abertas desenvolvidas em código livre que conversam entre si, temos várias bibliotecas livres desenvolvidas que conseguem se comunicar umas com as outras, e cabe ao programador verificar se esta comunicação é feita de maneira correta, também fica a cargo do programador estudar a linguagem de programação aceita por cada biblioteca, ou seja, a linguagem matemática utilizada na programação do MATLAB® não é a mesma linguagem de programação utilizada no desenvolvimento do código em Python, logo, é necessário aprender tudo sobre a nova linguagem de programação para conseguir utiliza-la para solucionar os problemas de engenharia.

### 3.4 Apresentação de cada parte do código

Para a utilização do *software* se faz necessário a instalação do pacote CVXOPT (*Python Software for Convex Optimization*) que nada mais é um pacote de *software* livre para otimização convexa baseado na linguagem de programação Python.

O pacote CVXOPT faz uso do *NumPy* que nada mais é do que uma biblioteca para a linguagem de programação Python, adicionando suporte para matrizes e *arrays* multidimensionais grandes, junto com uma vasta coleção de funções matemáticas de alto nível para operar tais *arrays*. Ademais, fez-se necessário o uso da biblioteca Picos que se trata de uma API Python amigável para vários solucionadores de programação cônica e inteira, muito parecido com YALMIP ou CVX no MATLAB®.

Para realizar a instalação do CVXOPT e da biblioteca PICOS basta seguir o processo descrito a seguir:

Para instalar usando o pip

```
pip install cvxopt
pip install picos
```

Para usuários anaconda se faz necessário a instalação no Prompt, abra o Anaconda *Prompt* e instale:

```
conda install -c conda-forge cvxopt
conda install -c picos picos
```

Com os pacotes devidamente instalados vamos agora apresentar as bibliotecas utilizadas para a plena execução do *software*, garantindo que todas as funções utilizadas sejam executadas de maneira adequada.

```
import cvxopt as cvx # cvxopt é o pacote do solver. ex. sedumi, lmilab
import picos as pic # picos é o pacote de interfaceamento com o solver. ex. yalmip
import numpy as np # numpy é um pacote que suporta operações com vetores e matrizes
from numpy import linalg as LA # linalg é um subpacote do numpy para algebra linear, # import (pacote) as (nome), faz-se a abreviação para utilizar o pacote
from picos import Constant
import time
```

Após a importação das bibliotecas e subpacotes necessários para a plena execução do programa, partimos para a montagem do código que nos apresentará como resultado um controlador chaveado capaz de estabilizar a planta.

Começamos definindo inicialmente a variável que o nosso *solver* irá resolver, neste caso utilizamos a biblioteca PICOS para atribuir a variável “lmi” como sendo a nossa variável a ser solucionada:

```
lmi = pic.Problem() # Definir a variável que vamos resolver
```

Na sequência devemos montar as matrizes  $A$  e  $B$  para que possamos apresentar o problema em espaço de estados como mostrado em (42), para o problema da suspensão ativa de bancada o equacionamento em espaço de estados está apresentado em (86) onde podemos extrair a matriz  $A$  e a matriz  $B$ .

Neste trabalho tanto a matriz  $A$  quanto a matriz  $B$  terão valores incertos na massa do carrinho o que nos força a inicialmente definir duas matrizes ( $A[0]$  e  $A[1]$ ) também ( $B[0]$  e  $B[1]$ ). Em Python, diferente do MATLAB® devemos declarar as variáveis  $A$  e  $B$  informando que está variável será uma variável *PICOS-Constant* e no caso da matriz  $A$  terá um tamanho de 16 termos e dimensão  $4 \times 4$ .

No caso da matriz  $B$  também será declarada como uma matriz *PICOS-Constant* com tamanho de 4 termos tendo dimensão  $2 \times 2$ .

Este processo de declaração é realizado também para o valor da massa do veículo ( $m_s$ ), para os valores incertos na entrada  $x_0, x_1, x_2, x_3$  e  $x_4$ , sendo que  $x_0[i] =$

$[x_1[c] \ x_2[v] \ x_3[b] \ x_4[n]]$ , sendo que cada termo  $x_1 \dots x_4$  pode assumir dois valores diferentes, logo a dimensão de  $x_0$  é de 16 matrizes.

Seguindo também foi definido as matrizes que faram parte da restrição:  $G_{2 \times 4}$ ,  $Q_{4 \times 4}$  e matriz  $Z_{4 \times 4}$ .

Todas estas definições de variáveis são definidas como mostrado a seguir:

```
A = [Constant("A[{}].format(i), range(i, i + 16), (4,4)) for i in range(2)]
B = [Constant("B[{}].format(i), range(i, i + 2), (1,2)) for i in range(2)]
ms = [Constant("ms[{}].format(i), range(i, i + 2), (1,2)) for i in range(2)]
x0 = [Constant("x0[{}].format(cont), range(cont, cont + 4), (1,4)) for cont in range(16)]
x1 = [Constant("x1[{}].format(i), range(i, i + 2), (1,2)) for i in range(2)]
x2 = [Constant("x2[{}].format(j), range(j, j + 2), (1,2)) for j in range(2)]
x3 = [Constant("x3[{}].format(k), range(k, k + 2), (1,2)) for k in range(2)]
x4 = [Constant("x4[{}].format(l), range(l, l + 2), (1,2)) for l in range(2)]
G = [Constant("G[{}].format(i), range(i, i + 4), (1,4)) for i in range(2)]
Q = [Constant("Q[{}].format(i), range(i, i + 16), (4,4)) for i in range(2)]
Z = [Constant("Z[{}].format(i), range(i, i + 16), (4,4)) for i in range(2)]
```

Posteriormente foi atribuído o valor das entradas incertas  $x_1 \dots x_4$  e os valores dos parâmetros que compõe o sistema da suspensão ativa de bancada e que preencheram as matrizes  $A$  e  $B$ . Esta atribuição de variáveis é realizada como mostrado a seguir:

```
x1[0]=-0.02
x1[1]=-0.02
x2[0]=-0.15
x2[1]=0.15
x3[0]=-0.02
x3[1]=0.02
x4[0]=-0.15
x4[1]=0.15
ms[0] = 1.455
ms[1] = 2.45
mus = 1
ks = 900
kus = 2500
bs = 7.5
bus = 5
beta=5
```

Com as matrizes devidamente declaradas na biblioteca PICOS, e as variáveis também devidamente atribuídas com seus valores é possível montarmos as matrizes  $A[0]$ ,  $A[1]$ ,  $B[0]$  e  $B[1]$ , para tal utilizamos a biblioteca *NumPy*, que nos possibilita montar as matrizes e trabalhar com as LMIs em sua forma matricial. A montagem das matrizes é realizada como apresentado a seguir:

```
A[0] = np.array([[0, 1, 0, -1], [-ks/ms[0], -bs/ms[0], 0, bs/ms[0]], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
A[1] = np.array([[0, 1, 0, -1], [-ks/ms[1], -bs/ms[1], 0, bs/ms[1]], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
B[0] = np.array([[0], [1/ms[0]], [0], [-1/mus]])
B[1] = np.array([[0], [1/ms[1]], [0], [-1/mus]])
```

Como nosso pacote de interfaceamento é a plataforma PICOS é necessário que as matrizes apresentadas anteriormente utilizando a biblioteca *NumPy* sejam convertidas para parâmetros PICOS, caso esta conversão de matrizes não seja realizada ao montarmos nossa matriz de restrições o *solver* não resolverá as LMIs, visto que, as matrizes não estão definidas como parâmetros que o *solver* consegue trabalhar, logo este passo de conversão é de extrema importância para a utilização do código em questão.

Está conversão foi um ponto que gerou muitas dúvidas durante a confecção do código, visto que, a biblioteca *NumPy* normalmente é aceita por todas as outras bibliotecas como ferramenta de trabalho com matrizes, porém, na solução de LMIs matrizes *NumPy* não são aceitas, sendo necessário realizar sua conversão para parâmetros PICOS.

```
# Transformando matrizes em parâmetro picos
A[0] = pic.new_param('A[0]', A[0] )
A[1] = pic.new_param('A[1]', A[1] )
B[0] = pic.new_param('B[0]', B[0] )
B[1] = pic.new_param('B[1]', B[1] )
```

Na sequência montou-se a matriz  $x_0$  de restrições na entrada, novamente foi montada uma matriz utilizando a biblioteca *NumPy*, posteriormente deverá ser convertida em parâmetro PICOS para que possa ser adicionada como restrição e solucionada pelo *solver*.

```
h=0
for c in range(2):
    for v in range(2):
        for b in range(2):
            for n in range(2): #Arrumar
                x0[h] = np.array([[x1[c]], [x2[v]], [x3[b]], [x4[n]]])
                h=h+1
```

Como vimos agora devemos converter as matrizes  $x_0[0] \dots x_0[15]$  em parâmetros PICOS como é feito a seguir:

```
x0[0] = pic.new_param('x0[0]', x0[0])
x0[1] = pic.new_param('x0[1]', x0[1])
x0[2] = pic.new_param('x0[2]', x0[2])
x0[3] = pic.new_param('x0[3]', x0[3])
x0[4] = pic.new_param('x0[4]', x0[4])
x0[5] = pic.new_param('x0[5]', x0[5])
x0[6] = pic.new_param('x0[6]', x0[6])
x0[7] = pic.new_param('x0[7]', x0[7])
x0[8] = pic.new_param('x0[8]', x0[8])
x0[9] = pic.new_param('x0[9]', x0[9])
x0[10] = pic.new_param('x0[10]', x0[10])
x0[11] = pic.new_param('x0[11]', x0[11])
x0[12] = pic.new_param('x0[12]', x0[12])
x0[13] = pic.new_param('x0[13]', x0[13])
x0[14] = pic.new_param('x0[14]', x0[14])
x0[15] = pic.new_param('x0[15]', x0[15])
```

Posteriormente criou-se as matrizes variáveis  $X_{4 \times 4}$ ,  $G_{2 \times 4}[0]$ ,  $G_{2 \times 4}[1]$ ,  $Q_{4 \times 4}[0]$ ,  $Q_{4 \times 4}[1]$ ,  $Z_{4 \times 4}[0]$ ,  $Z_{4 \times 4}[1]$  e  $U_{1 \times 1}$ .

```
# Definindo as variáveis (matriz variável P)
X = lmi.add_variable('X',(4,4),'symmetric') # definir as variáveis
G[0] = lmi.add_variable('G[0]',(2,4), 'continuous')
G[1] = lmi.add_variable('G[1]',(2,4), 'continuous')
Q[0] = lmi.add_variable('Q[0]',(4,4), 'hermitian')
Q[1] = lmi.add_variable('Q[1]',(4,4), 'hermitian')
Z[0] = lmi.add_variable('Z[0]',(4,4), 'hermitian')
Z[1] = lmi.add_variable('Z[1]',(4,4), 'hermitian')
U = lmi.add_variable('U',(1,1))
```

Em uma das restrições foi utilizada a matriz identidade, a biblioteca PICOS já disponibiliza uma matriz identidade no formato PICOS, não sendo necessário criar uma matriz *NumPy* e converter para parâmetro PICOS.

```
I = pic.I(2)
```

Feito isto podemos agora adicionar a nossa variável “lmi” as restrições do problema. Quem faz esta adição de restrições é a biblioteca PICOS, nossa biblioteca de interface, sendo assim pegaremos nossa variável “lmi = pic.problem()” e adicionaremos nela nossas restrições através do comando “*add\_constraint()*” como apresentado a seguir:

```
for i in range(2):
    lmi.add_constraint(((A[i]*X + X*A[i].H + Q[i] + Z[i] + 2*beta*X) << 0))
for j in range(2):
    lmi.add_constraint((-G[j].H*B[i].H - B[i]*G[j] - Q[j] - Z[i]) << 0)
    lmi.add_constraint((X & G[j].H) // (G[j] & U*I) >> 0)
for s in range(16):
    lmi.add_constraint(((1 & x0[s].H) // (x0[s] & X) >> 0)

lmi.add_constraint(X >> 0)
```

Com as restrições devidamente atribuídas, podemos agora apresentar as restrições como informação de saída, basta utilizar a função “*print*” como apresentado a seguir:

```
# Mostrar as restrições LMIs
print(lmi)
print('type: '+lmi.type)
print('status: '+lmi.status) # foi resolvido?
```

Na sequência utilizamos a biblioteca CVXOPT para resolver as LMIs definidas anteriormente e informar ao usuário se o resultado processado foi resolvido ou não dado o conjunto de restrições.

```
# Resolvendo as LMIs
sol = lmi.solve(solver='cvxopt',verbose=0)
print('status: '+lmi.status) #foi resolvido?
```

Após o *solver* resolver as LMIs com todas as restrições, é necessário converter as matrizes novamente para valores *NumPy* que são valores que podem ser imprimidos na tela e apresentados em sua forma matricial que conhecemos.

Depois de ter convertido as matrizes para a biblioteca *NumPy* novamente podemos utilizar o comando “*print*” para escrever os valores na tela de cada matriz como apresentado a seguir:

```
# Convertendo nossos parâmetros do solver em Arrays
print('Matriz Ótima P:')
X= np.array(X.value)
G[0]= np.array(G[0].value)
G[1]= np.array(G[1].value)
A[0]= np.array(A[0].value)
A[1]= np.array(A[1].value)
B[0]= np.array(B[0].value)
B[1]= np.array(B[1].value)
U= np.array(U.value)
```

```
print(X)
print("\n Os Autovalores de X são:")
print(LA.eigvals(X))
eigX = LA.eigvals(X)
```

Na sequência é realizada a conversão e escrita dos autovalores das restrições impostas como mostrado a seguir:

```
print("\nOs Autovalores de A1*X+X*A1.H-B1*G-G.H*B1.H são:")
print(LA.eigvals(A[0]@X+X@np.transpose(A[0]) - B[0]@G[0] - np.transpose(G[0])@np.transpose(B[0])))
eigLMI = LA.eigvals(A[0]@X+X@np.transpose(A[0]) - B[0]@G[0] - np.transpose(G[0])@np.transpose(B[0])) ##
Em numpy, A*B significa a multiplicação indexidada dos elementos
## O operador @ realiza a multiplicação matricial correta.
print("\nOs Autovalores de A2*X+X*A2.H-B2*G-G.H*B2.H são:") #errado mudou a restrição
print(LA.eigvals(A[1]@X+X@np.transpose(A[1]) - B[1]@G[1] - np.transpose(G[1])@np.transpose(B[1])))
eigLMI = LA.eigvals(A[1]@X+X@np.transpose(A[1]) - B[1]@G[1] - np.transpose(G[1])@np.transpose(B[1])) ##
Em numpy, A*B significa a multiplicação indexidada dos elementos
## O operador @ realiza a multiplicação matricial correta.
```

Na sequência verificamos se o sistema é factível e se é possível controlar e escrevemos na tela o resultado desta verificação:

```
if eigX.__gt__(0).all():
    if eigLMI.__lt__(0).all():
        print("\nSistema Factível. \nÉ possível controlar.")
    else: print("\nSistema Infactível")
```

Por fim convertemos o valor da matriz  $K$  que é o ganho do nosso controlador e escrevemos na tela o valor do nosso controlador chaveado  $K_0, K_1$ . Também convertemos o valor da matriz  $U$  e extraímos o valor da constante  $\mu$  para verificar qual *solver* encontrou o melhor valor na restrição da norma do controlador.

```

## Controlador
K = [Constant("K[{}]" .format(i),range(i, i + 2 ), (1,2)) for i in range(2)]
K[0] = G[0]@(LA.inv(X))
print("\nControlador K[0]: \n", K[0])
K[1] = G[1]@(LA.inv(X))
print("\nControlador K[1]: \n", K[1])
print("\nAutorvalor de A1-B1*K:\n", LA.eigvals(A[0]-B[0]@K))
print("\nAutorvalor de A2-B2*K:\n", LA.eigvals(A[1]-B[1]@K))
print("\n O valor de U é:")
print(U)
mi=np.sqrt(U)
print("\n O valor de mi é:")
print(mi)
#-----#
# objective value #
#-----#
#print( \nthe optimal value of this problem is:')
#print(lmi.obj_value())          #"print objective" would also work, because objective is valued
toc ()

```

## 4 RESULTADOS E ANÁLISE

Nesta seção apresentaremos os resultados obtidos com o código desenvolvido para o projeto de plotagem do *Root-Locus* utilizando o *software* de programação livre Python. Foram feitos inúmeros testes para verificar se o código estava realmente funcionando de forma adequada e como as expectativas foram atendidas serão apresentados a seguir alguns exemplos de aplicação do programa.

### 4.1 Primeira aplicação exemplo Root Locus

Para uma primeira aplicação implementaremos a função de transferência que foi apresentada na Seção 2.3, portanto em (37) verificou-se então a evolução das raízes do polinômio conforme variou-se o parâmetro  $K_a$ , o gráfico feito com os resultados analíticos pode ser observado na Figura 15.

Para utilizar o programa o parâmetro  $K_a$  será retirado de (37), visto que, é ele que estamos procurando encontrar com o “*Root-Locus*”.

Para função de transferência de malha fechada apresentada em (37) utilizou-se os seguintes parâmetros para projeto:

Porcentagem de *Overshoot*:

- $P.O. = 16\%$ ,

Critério para o cálculo do tempo de estabelecimento = 1%.

Tempo de subida:

- $t_s = 0,9$  segundos,

Tempo de subida:

- $t_e = 2,6$  segundos.

Para esses valores e utilizando em (24) obtemos um valor teórico de  $\xi = 0,5038$ , calculando o ângulo  $\theta = \arccos \xi$ , obtemos  $\theta = 59,74^\circ$ , utilizando (31), obtemos o valor de  $\xi \times \omega_n = 1,8$  e utilizando (34), obtemos o valor de  $\omega_n = 2$ . Esses valores são os valores teóricos.

Esses valores foram inseridos no projeto desenvolvido em Python e geraram os resultados apresentados na Figura 21, Figura 22, Figura 23 e Figura 24.



Figura 21 – Inserção das condições de projeto no *software*.

Digite os valores de entradas pedidos.

Digite a Porcentagem de overshoot desejado em porcentagem %: 16  
 Digite o critério utilizado para o calculo do Tempo de estabelecimento 1  
 Digite o Tempo de subida: 0.9  
 Digite o Tempo de estabelecimento desejado: 2.6

Fonte: próprio autor.

Figura 22 – Inserção dos polinômios do Numerador e Denominador.

Digite os valores do polinomio do numerador de  $G(s)H(s)$

**Cuidado:** A ordem do polinômio do numerador não pode ser maior que a ordem do polinômio do denominador, o que torna inviável a representação do Root-Locus)

O Grau do polinômio é dado pelo expoente de maior grau.  
 Por exemplo:  $s^2 - 2s$  polinomio de grau 2 com coeficientes [1 -2 0]

**Atenção a Ordem (grau) do polinômio do numerador, obrigatoriamente tem que ser inferior a ordem (grau) do polinômio do denominador.**

Por exemplo: numerador [s+5] grau=1, denominador [ $s^2 2s +1$ ] grau=2

Digite a ordem (grau) do polinomio do numerador: 0  
 Digite o coeficiente correspondete á ordem 0: 5

Digite os valores do polinomio do denominador de  $G(s)H(s)$

**Cuidado:** A ordem do polinômio do numerador não pode ser maior que a ordem do polinômio do denominador, o que torna inviável a representação do Root-Locus)

O Grau do polinômio é dado pelo expoente de maior grau.  
 Por exemplo:  $s^2 - 2s$  polinomio de grau 2 com coeficientes [1 -2 0]

**Atenção a Ordem (grau) do polinômio do numerador obrigatoriamente tem que ser inferior a ordem (grau) do polinômio do denominador.**

Por exemplo: numerador [s+1], denominador [ $s^2 2s +1$ ]

Digite a ordem (grau) do polinomio do denominador: 2  
 Digite o coeficiente correspondete á ordem 2: 1  
 Digite o coeficiente correspondete á ordem 1: 20  
 Digite o coeficiente correspondete á ordem 0: 5

Menu de Opções:

[P] - Parametros Do circuito [G] - Plotar o Gráfico  
 [V] - Voltar ao Menu Anterior

Fonte: próprio autor.

Figura 23 – Parâmetros calculados pelo *software*.

Apresentamos a seguir os parâmetros que foram calculados com os dados inseridos:

Para uma Porcentagem de Overshoot de 16.0 %.

Calculou-se um coeficiente de amortecimento  $\xi = 0.5039$ .

Com o coeficiente de amortecimento  $\xi$  de 0.5039.

Calculou-se o ângulo  $\theta$  entre o eixo real e a reta que define a região do potencial de overshoot  $\theta = 59.7438$  :

Para um tempo de estabelecimento de 2.6, e critério de estabelecimento de 1.0,

Calculou-se um coeficiente de amortecimento vezes frequência natural não-amortecida  $\xi\omega_n = 1.7712$ .

Para o tempo de subida estabelecido em 0.9 segundos

Calculou-se uma frequência natural não amortecida  $\omega_n = 2.0000$  .

Menu de Opções:

[P] - Parâmetros Do circuito [G] - Plotar o Gráfico

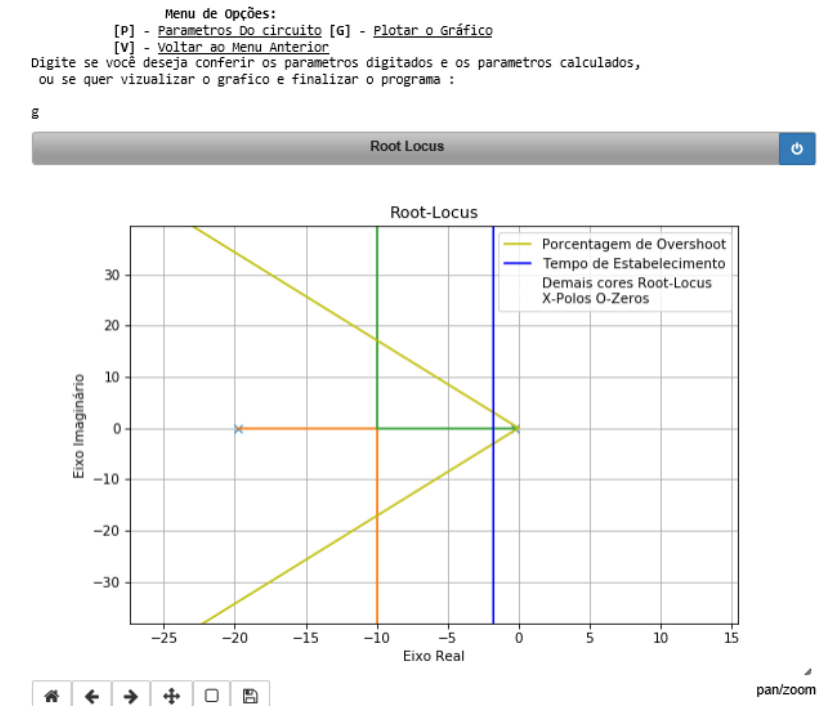
[V] - Voltar ao Menu Anterior

Digite se você deseja conferir os parâmetros digitados e os parâmetros calculados, ou se quer visualizar o gráfico e finalizar o programa :

Fonte: próprio autor.

Podemos notar que os valores apresentados pelo *software* são praticamente os mesmos calculados de forma teórica, se não os mesmos por meio de aproximações.

Figura 24 – Root-Locus plotado no *software*, parâmetros informados pelo usuário.



Fonte: próprio autor.

Podemos notar que as retas laranja e verde representam o *Root-Locus*, que é o mesmo Lugar das raízes mostrado na Seção 2.3 na Figura 15, pode-se notar também que foi implementado a reta em azul que representa o tempo de estabelecimento ( $t_e$ ) definido no projeto, e as retas em amarelo que representa a porcentagem de *overshoot* (*P.O.*) definidas no projeto.

Outro fator que nos auxilia é poder observar os polos marcados por um  $X$ , neste caso tem-se um polo na origem que resulta em erro de regime permanente nulo, e um polo em  $-20$ , é notório que não tem nenhum zero neste sistema.

Pelo gráfico fica fácil para o projetista verificar que tem uma região onde o *Root-Locus* se encontra a esquerda da reta azul referente ao tempo de estabelecimento e no interior das duas retas amarelas referente a porcentagem de overshoot de projeto, logo existe um ganho  $K_a$  que pode ser aplicado para garantir as especificações de projeto.

Pela observação do gráfico é possível afirmar que o sistema é estável pelo fato do *Root-Locus* se encontrar todo do lado negativo do eixo real, o que garante estabilidade.

## 4.2 Segunda aplicação exemplo Root Locus

Nesta segunda aplicação foram adotados os seguintes valores de projeto para o cálculo do controlador:

Porcentagem de *overshoot*:

- $P.O. = 10\%$

*Critério para o cálculo do tempo de estabelecimento* = 1%

Tempo de subida:

- $t_s = 2 \text{ segundos}$

Tempo de estabelecimento:

- $t_e = 3 \text{ segundos}$

A função  $G(s) \times H(s)$  que será testada é a função descrita por (106):

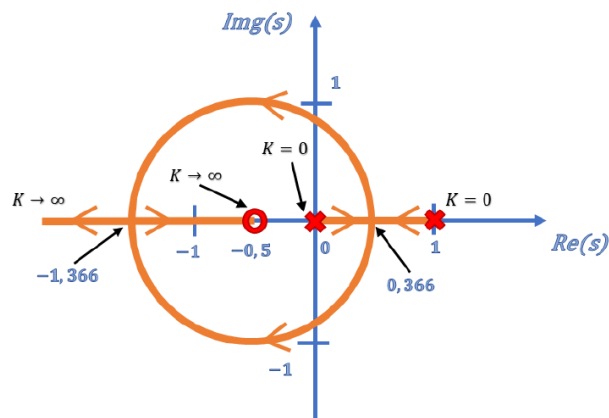
$$G(s) \times H(s) = \frac{K(s+0,5)}{s(s-1)} = \frac{K(s+0,5)}{s^2-s}. \quad (106)$$

Novamente para verificar a faixa de  $k$  o parâmetro será omitido e o numerador da função será  $(s + 5)$  para informa-lo no *software* deveremos informar o grau do polinômio neste caso 1, informar os coeficientes do polinômio na ordem do maior para o menor [1 0,5], para o

denominador também deverá ser informado o valor do grau do polinômio que para este caso vale 2, e em seguida informar o valor do coeficiente referente a cada termo do polinômio sempre seguindo a ordem do maior para o menor grau [1 -1].

Analisando a função é esperado que exista um polo em 0, outro polo em 1, e um zero em  $-0,5$ . Na Figura 25 é possível observar o gráfico referente apenas ao *Root-Locus* da função  $G(s) \times H(s)$ , mostrada em (4.1), a Figura 25 será usada como referência (teórica) para comparar a resposta obtida utilizando o código proposto.

Figura 25 – Root-Locus teórico referente a (4.1).



Fonte: próprio autor.

Na sequência é apresentado os resultados obtidos inserindo (4.1), juntamente com os parâmetros de projeto no código:

Figura 26 – Inserção das condições de projeto no *software*.

**Digite os valores de entradas pedidos.**

```

Digite a Porcentagem de overshoot desejado em porcentagem %: 10
Digite o critério utilizado para o calculo do Tempo de estabelecimento 1
Digite o Tempo de subida: 2
Digite o Tempo de estabelecimento desejado: 3

```

**Digite os valores do polinomio do numerador de  $G(s) \cdot H(s)$**

Fonte: próprio autor.

Figura 27 – Inserção dos polinômios do Numerador e Denominador.

Digite os valores do polinomio do numerador de  $G(s)H(s)$

**Cuidado:** A ordem do polinômio do numerador não pode ser maior que a ordem do polinômio do denominador, o que torna inviável a representação do Root-Locus)

O Grau do polinômio é dado pelo expoente de maior grau.  
Por exemplo:  $s^2 - 2s$  polinomio de grau 2 com coeficientes [1 -2 0]

**Atenção a Ordem (grau) do polinômio do numerador, obrigatoriamente tem que ser inferior a ordem (grau) do polinômio do denominador.**  
Por exemplo: numerador [s+5] grau=1, denominador [ $s^2 - 2s + 1$ ] grau=2

Digite a ordem (grau) do polinomio do numerador: 1  
Digite o coeficiente correspondete á ordem 1: 1  
Digite o coeficiente correspondete á ordem 0: 0.5

Digite os valores do polinomio do denominador de  $G(s)H(s)$

**Cuidado:** A ordem do polinômio do numerador não pode ser maior que a ordem do polinômio do denominador, o que torna inviável a representação do Root-Locus)

O Grau do polinômio é dado pelo expoente de maior grau.  
Por exemplo:  $s^2 - 2s$  polinomio de grau 2 com coeficientes [1 -2 0]

**Atenção a Ordem (grau) do polinômio do numerador obrigatoriamente tem que ser inferior a ordem (grau) do polinômio do denominador.**  
Por exemplo: numerador [s+1], denominador [ $s^2 - 2s + 1$ ]

Digite a ordem (grau) do polinomio do denominador: 2  
Digite o coeficiente correspondete á ordem 2: 1  
Digite o coeficiente correspondete á ordem 1: -1  
Digite o coeficiente correspondete á ordem 0: 0

Menu de Opções:  
[P] - Parametros Do circuito [G] - Plotar o Gráfico  
[V] - Voltar ao Menu Anterior

Digite se você deseja conferir os parametros digitados e os parametros calculados,

---

Fonte: próprio autor.

Figura 28 – Parâmetros calculados pelo *software*.

Apresentamos a seguir os parametros que foram calculados com os dados inseridos:  
Para uma Porcentagem de Overshoot de 10.0 %.  
Calculou-se um coeficiente de amortecimento  $\xi = 0.5912$ .

Com o coeficiente de amortecimento  $\xi$  de 0.5912.  
Calculou-se o angulo  $\theta$  entre o eixo real e a reta que define a região do potencial de overshoot  $\theta = 53.7610$  :

Para um tempo de estabelecimento de 3.0, e critério de estabelecimento de 1.0,  
Calculou-se um coeficiente de amortecimento vezes frequência natural não-amortecida  $\xi\omega_n = 1.5351$ .

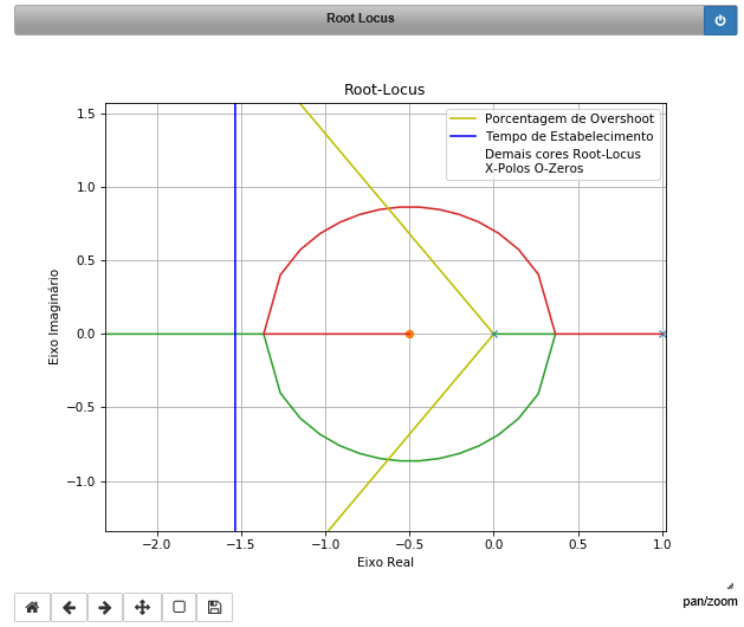
Para o tempo de subida estabelecido em 2.0 segundos  
Calculou-se uma frequência natural não amortecida  $\omega_n = 0.9000$  .

Menu de Opções:  
[P] - Parametros Do circuito [G] - Plotar o Gráfico  
[V] - Voltar ao Menu Anterior

Digite se você deseja conferir os parametros digitados e os parametros calculados, ou se quer visualizar o grafico e finalizar o programa :

Fonte: próprio autor.

Figura 29 – Root-Locus plotado via *software* e parâmetros fornecidos pelo usuário.



Fonte: próprio autor.

Podemos observar que o *Root-Locus* apresentado pelo *software* atendeu as expectativas quando comparamos a parte teórica da Figura 25 com o resultado do *software* apresentado na Figura 29 nota-se também dois polos em 0 e em 1, nota-se também o zero em  $-0,5$ .

Observando o *Root-Locus* (linha vermelha e linha verde) podemos notar que a região circular que contém os pares complexos conjugados para os valores possíveis de  $K$ , parte dela se encontra no interior das retas amarelas o que atende o valor de projeto referente à porcentagem de *overshoot* de 10%, porém a região circular do *Root-Locus* se encontra a direita da reta azul, o que não atende a região responsável pelo tempo de estabelecimento. Logo veremos um exemplo de um *Root-Locus* que será corrigido para atender as exigências de projeto, neste caso foi apenas um exemplo que não atende as especificações de projeto.

### 4.3 Terceira aplicação exemplo *Root Locus*

Testaremos agora o programa com uma função com polinômios de grau elevado. A função que será testada pode ser observada em (108).

$$G(s) \times H(s) = \frac{K(s+1)^2}{(s+8)(s+20)} \times \frac{1}{s(s^2+3,2s+3,56)}, \quad (107)$$

$$G(s) \times H(s) = \frac{K(s^2+2s+1)}{s^5+31,2s^4+253,16s^3+611,68s^2+569,6s} \quad (108)$$

Os valores de projeto que serão utilizados para este caso são:

Porcentagem de *Overshoot*:

- $P.O. = 14\%$

*Critério para o cálculo do tempo de estabelecimento* = 2%

Tempo de subida:

- $t_s = 0,3 \text{ segundos}$

Tempo de estabelecimento:

- $t_e = 0,5 \text{ segundos}$

Figura 30 – Inserção das condições de projeto no *software*.

**Digite os valores de entradas pedidos.**

```

Digite a Porcentagem de overshoot desejado em porcentagem %: 14
Digite o critério utilizado para o calculo do Tempo de estabelecimento 1
Digite o Tempo de subida: 0.3
Digite o Tempo de estabelecimento desejado: 0.5

```

Fonte: próprio autor.

Os valores inseridos nesta parte são referentes a especificação de projeto no qual o controlador deve atender.

Figura 31 – Inserção dos polinômios do Numerador e Denominador.

```

Digite os valores do polinomio do numerador de G(s)*H(s)

Cuidado: A ordem do polinômio do numerador não pode ser maior que a ordem do polinômio do denominador,
o que torna inviável a representação do Root-Locus)

O Grau do polinômio é dado pelo expoente de maior grau.
Por exemplo: s^2 -2s polinomio de grau 2 com coeficientes [1 -2 0]

Atenção a Ordem (grau) do polinômio do numerador, obrigatoriamente tem que ser inferior
a ordem (grau) do polinômio do denominador.
Por exemplo: numerador [s+5] grau=1, denominador [s^2 2s +1] grau=2

Digite a ordem (grau) do polinomio do numerador: 2
Digite o coeficiente correspondete á ordem 2: 1
Digite o coeficiente correspondete á ordem 1: 2
Digite o coeficiente correspondete á ordem 0: 1

Digite os valores do polinomio do denominador de G(s)*H(s)

Cuidado: A ordem do polinômio do numerador não pode ser maior que a ordem do polinômio do denominador,
o que torna inviável a representação do Root-Locus)

O Grau do polinômio é dado pelo expoente de maior grau.
Por exemplo: s^2 -2s polinomio de grau 2 com coeficientes [1 -2 0]

Atenção a Ordem (grau) do polinômio do numerador obrigatoriamente tem que ser inferior
a ordem (grau) do polinômio do denominador.
Por exemplo: numerador [s+1], denominador [s^2 2s +1]

Digite a ordem (grau) do polinomio do denominador: 5
Digite o coeficiente correspondete á ordem 5: 1
Digite o coeficiente correspondete á ordem 4: 31.2
Digite o coeficiente correspondete á ordem 3: 253.16
Digite o coeficiente correspondete á ordem 2: 611.68
Digite o coeficiente correspondete á ordem 1: 569.6
Digite o coeficiente correspondete á ordem 0: 0

```

Fonte: próprio autor.

Na Figura 31 foram inseridos os polinômios do numerador e denominador, sendo informado inicialmente o grau do polinômio, e em seguida os coeficientes correspondentes a cada grau do polinômio sempre seguindo da maior ordem para a menor ordem, neste caso o numerador de grau 2, se torna [1; 2; 1], e o denominador de grau 5 se torna [1; 31,2; 253,16; 611,68; 569,6].



Figura 32 – Parâmetros calculados pelo *software*.

Apresentamos a seguir os parâmetros que foram calculados com os dados inseridos:

Para uma Porcentagem de Overshoot de 14,0 %.  
Calculou-se um coeficiente de amortecimento  $\xi = 0.5305$ .

Com o coeficiente de amortecimento  $\xi$  de 0.5305.  
Calculou-se o ângulo  $\theta$  entre o eixo real e a reta que define a região do potencial de overshoot  $\theta = 57.9603$  :

Para um tempo de estabelecimento de 0.5, e critério de estabelecimento de 1.0,  
Calculou-se um coeficiente de amortecimento vezes frequência natural não-amortecida  $\xi\omega_n = 9.2103$ .

Para o tempo de subida estabelecido em 0.3 segundos  
Calculou-se uma frequência natural não amortecida  $\omega_n = 6.0000$  .

Menu de Opções:

[P] - Parâmetros Do circuito [G] - Plotar o Gráfico

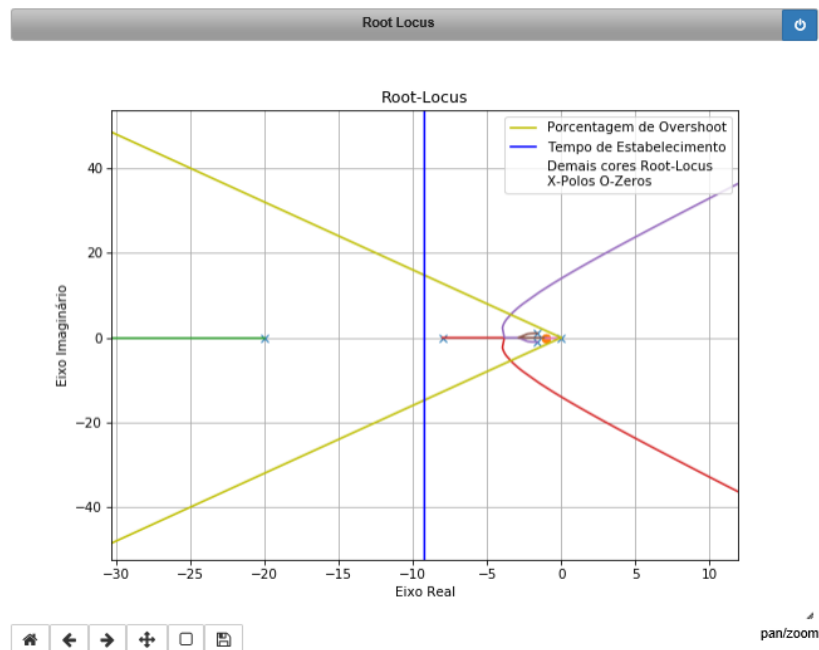
[V] - Voltar ao Menu Anterior

Digite se você deseja conferir os parâmetros digitados e os parâmetros calculados,  
ou se quer visualizar o gráfico e finalizar o programa :

Fonte: próprio autor.

Após toda a inserção dos dados foi pedido os valores dos parâmetros calculados com os coeficientes de projeto informados, e o *software* retornou o valor de  $\xi = 0,5305$ ,  $\omega_n = 6$ ,  $\theta = 57,9603^\circ$ ,  $\xi\omega_n = 9,2103$ .

Figura 33 – Root-Locus plotado pelo *software* com os parâmetros informados pelo usuário.



Fonte: próprio autor.

Com tudo informado ao *software* foi obtido o gráfico do “*Root-Locus*” juntamente com os requisitos de projeto informados pelo usuário, e o resultado é mostrado na Figura 33.

#### 4.4 Terceira aplicação exemplo com correção do Root-Locus

Neste exemplo é apresentado um caso em que o *Root-Locus* se encontra fora da região de projeto e o projetista realizará a correção após primeira análise.

Na aplicação adotaremos os seguintes valores de projeto:

Porcentagem de *Overshoot*:

- $P.O. = 70\%$

*Critério para o cálculo do tempo de estabelecimento* = 1%

Tempo de subida:

- $t_s = 2 \text{ segundos}$

Tempo de estabelecimento:

- $t_e = 3 \text{ segundos}$

A função  $G(s) \times H(s)$  inicialmente será a função apresentada em (109):

$$G(s) \times H(s) = \frac{K}{(s-2)(s-4)}. \quad (109)$$

Veremos agora os valores calculados a partir das especificações de projeto e o *Root-Locus* do referente a (109) de  $G(s) \times H(s)$ .

Figura 34 – Valores de entrada dos parâmetros, das funções do numerador e denominador.

Digite os valores de entradas pedidos.

Digite a Porcentagem de overshoot desejado em porcentagem %: 70  
 Digite o critério utilizado para o calculo do Tempo de estabelecimento 1  
 Digite o Tempo de subida: 2  
 Digite o Tempo de estabelecimento desejado: 3

Digite os valores do polinomio do numerador de  $G(s)H(s)$

O Grau do polinômio é dado pelo expoente de maior grau.  
 Por exemplo:  $s^2 - 2s$  polinomio de grau 2 com coeficientes [1 -2 0]

Atenção a Ordem (grau) do polinômio do numerador, obrigatoriamente tem que ser inferior a ordem (grau) do polinômio do denominador.  
 Por exemplo: numerador [s+5] grau=1, denominador [ $s^2 2s +1$ ] grau=2

Digite a ordem (grau) do polinomio do numerador: 0  
 Digite o coeficiente correspondete á ordem 0: 1

Digite os valores do polinomio do denominador de  $G(s)H(s)$

O Grau do polinômio é dado pelo expoente de maior grau.  
 Por exemplo:  $s^2 - 2s$  polinomio de grau 2 com coeficientes [1 -2 0]

Atenção a Ordem (grau) do polinômio do numerador obrigatoriamente tem que ser inferior a ordem (grau) do polinômio do denominador.  
 Por exemplo: numerador [s+1], denominador [ $s^2 2s +1$ ]

Digite a ordem (grau) do polinomio do denominador: 2  
 Digite o coeficiente correspondete á ordem 2: 1  
 Digite o coeficiente correspondete á ordem 1: -6  
 Digite o coeficiente correspondete á ordem 0: 8

Menu de Opções:  
 [P] - Parâmetros Do circuito [G] - Plotar o Gráfico  
 [V] - Voltar ao Menu Anterior

Digite se você deseja conferir os parametros digitados e os parametros calculados, ou se quer vizualizar o grafico e finalizar o programa :

Fonte: próprio autor.

Figura 35 – Parâmetros calculados pelo *software*.

Digite se você deseja conferir os parametros digitados e os parametros calculados, ou se quer vizualizar o grafico e finalizar o programa :

Apresentamos a seguir os parametros que foram calculados com os dados inseridos:  
 Para uma Porcentagem de Overshoot de 70.0 %.  
 Calculou-se um coeficiente de amortecimento  $\xi = 0.1128$ .

Com o coeficiente de amortecimento  $\xi$  de 0.1128.  
 Calculou-se o angulo  $\theta$  entre o eixo real e a reta que define a região do potencial de overshoot  $\theta = 83.5228$  :

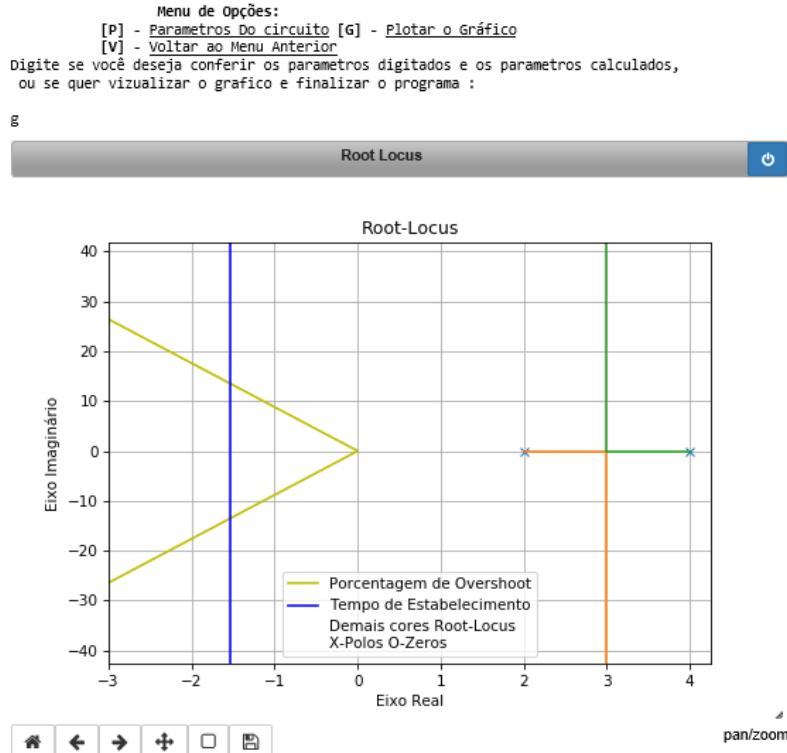
Para um tempo de estabelecimento de 3.0, e critério de estabelecimento de 1.0,  
 Calculou-se um coeficiente de amortecimento vezes frequência natural não-amortecida  $\xi\omega_n = 1.5351$ .

Para o tempo de subida estabelecido em 2.0 segundos  
 Calculou-se uma frequência natural não amortecida  $\omega_n = 0.9000$  .

Menu de Opções:  
 [P] - Parâmetros Do circuito [G] - Plotar o Gráfico  
 [V] - Voltar ao Menu Anterior

Fonte: próprio autor.

Figura 36 – Root-Locus plotado pelo *software* com os parâmetros informados pelo usuário.



Fonte: próprio autor.

Observando a Figura 36, nota-se que se trata de um sistema instável pelo fato do *Root-Locus* estar do lado positivo do eixo real, o que implica em um sistema instável, nota-se também que o *Root-Locus* não se encontra dentro da região de projeto logo não atende a região de projeto.

Para a correção do *Root-Locus* o projetista deve neste caso específico adicionar um zero no lado negativo do eixo real, porém quando se adiciona um zero também deve-se adicionar um polo, visto que o controlador não pode ter mais zeros do que polos devido ao fato de tornar inviável o projeto do mesmo.

Portanto como uma sugestão de projeto é podemos adicionar um zero em  $-3$  e um polo em  $-19$ , e a nova função  $G(s) \times H(s)$  torna-se:

$$G(s) \times H(s) = \frac{K(s+3)}{s+19} \times \frac{1}{(s-2)(s-4)}, \quad (110)$$

$$G(s) \times H(s) = \frac{K \times (s+3)}{s^3 + 13s^2 - 106s + 152}. \quad (110)$$

Adicionando novamente os valores ao projeto e plotando novamente o *Root-Locus* temos:

Figura 37 – Inserção dos novos valores de  $G(s) \times H(s)$ , mantendo-se os requisitos de projeto.

```

Digite os valores de entradas pedidos.

Digite a Porcentagem de overshoot desejado em porcentagem %: 70
Digite o critério utilizado para o calculo do Tempo de estabelecimento 1
Digite o Tempo de subida: 2
Digite o Tempo de estabelecimento desejado: 3

Digite os valores do polinomio do numerador de G(s)*H(s)

O Grau do polinômio é dado pelo expoente de maior grau.
Por exemplo: s^2 -2s polinomio de grau 2 com coeficientes [1 -2 0]

Atenção a Ordem (grau) do polinômio do numerador, obrigatoriamente tem que ser inferior
a ordem (grau) do polinômio do denominador.
Por exemplo: numerador [s+5] grau=1, denominador [s^2 2s +1] grau=2

Digite a ordem (grau) do polinomio do numerador: 1
Digite o coeficiente correspondete á ordem 1: 1
Digite o coeficiente correspondete á ordem 0: 3

Digite os valores do polinomio do denominador de G(s)*H(s)

O Grau do polinômio é dado pelo expoente de maior grau.
Por exemplo: s^2 -2s polinomio de grau 2 com coeficientes [1 -2 0]

Atenção a Ordem (grau) do polinômio do numerador obrigatoriamente tem que ser inferior
a ordem (grau) do polinômio do denominador.
Por exemplo: numerador [s+1], denominador [s^2 2s +1]

Digite a ordem (grau) do polinomio do denominador: 3
Digite o coeficiente correspondete á ordem 3: 1
Digite o coeficiente correspondete á ordem 2: 13
Digite o coeficiente correspondete á ordem 1: -106
Digite o coeficiente correspondete á ordem 0: 152

Menu de Opções:
[P] - Parametros Do circuito [G] - Plotar o Gráfico
[V] - Voltar ao Menu Anterior

Digite se você deseja conferir os parametros digitados e os parametros calculados,
ou se quer vizualizar o grafico e finalizar o programa :

```

Fonte: próprio autor.

Figura 38 – Parâmetros calculados pelo software.

```

Digite se você deseja conferir os parametros digitados e os parametros calculados,
ou se quer vizualizar o grafico e finalizar o programa :

[ ]

Apresentamos a seguir oa parametros que foram calculados com os dados inseridos:
Para uma Porcentagem de Overshoot de 70.0 %.
Calculou-se um coeficiente de amortecimento  $\xi = 0.1128$ .

Com o coeficiente de amortecimento  $\xi$  de 0.1128.
Calculou-se o angulo  $\theta$  entre o eixo real e a reta que define a região do potencial de overshoot  $\theta = 83.5228$  :

Para um tempo de estabelecimento de 3.0, e critério de estabelecimento de 1.0,
Calculou-se um coeficiente de amortecimento vezes frequência natural não-amortecida  $\xi\omega_n = 1.5351$ .

Para o tempo de subida estabelecido em 2.0 segundos
Calculou-se uma frequência natural não amortecida  $\omega_n = 0.9000$  .

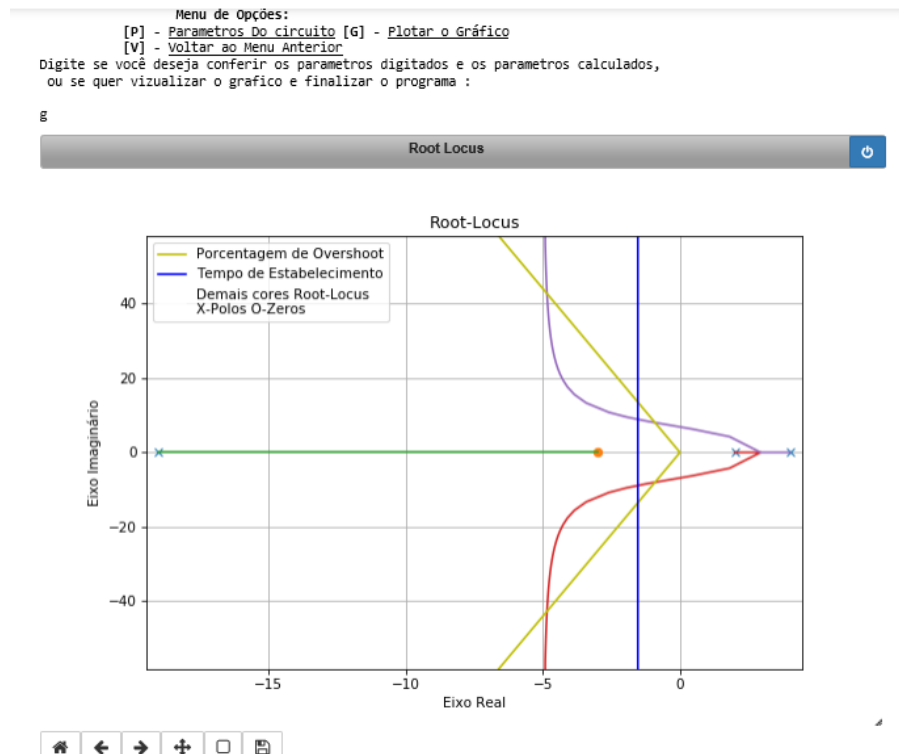
Menu de Opções:
[P] - Parametros Do circuito [G] - Plotar o Gráfico
[V] - Voltar ao Menu Anterior

```

Fonte: próprio autor.

Nota-se que os parâmetros de projeto se mantem os mesmos, visto que a única alteração ocorrerá no *Root-Locus* devido à mudança na função do numerador e no denominador de  $G(s) \times H(s)$ .

Figura 39 – Novo *Root-Locus* após a correção.



Fonte: próprio autor.

É notório que agora existam valores para  $K$  que tornam o *Root-Locus* estável, pois existe *Root-Locus* do lado negativo do eixo real. Observa-se que com a correção também existam valores para  $K$  tal que as especificações de projeto sejam atendidas, visto que, existe *Root-Locus* a esquerda da reta azul que representa o tempo de estabelecimento ( $t_e$ ) desejado, e também existe *Root-Locus* no interior das duas retas amarelas, que representam a porcentagem de *overshoot* ( $P.O.$ ), sendo assim o novo controlador é possível de implementar de modo a tornar o sistema estável e atendendo aos critérios de projeto.

Vale observar que foram escolhidos valores de projeto como porcentagem de *overshoot*, tempo de subida, e tempo de estabelecimento genéricos apenas para ilustrar uma certa aplicação de correção do *Root-Locus*.

#### 4.5 Análise geral dos resultados dos projetos realizados utilizando o código desenvolvido

Diante dos testes realizados com o *script* e durante os projetos de controladores analógicos, notou-se uma grande eficiência do *software* em realizar os cálculos do coeficiente de amortecimento  $\xi$ , da frequência natural não amortecida  $\omega_n$ , o valor de  $\xi\omega_n$ , o ângulo  $\theta$ , para os valores de projeto informados pelo usuário, estes valores são de extrema importância para o projetista conferir se seu projeto se classifica dentro das especificações.

Analisando o gráfico do *Root-Locus*, verificou-se que ele realmente representa a função informada pelo usuário de acordo com a variação de um parâmetro  $K$  e a inserção dos parâmetros de projeto juntamente com o gráfico do *Root-Locus*, permite ao projetista verificar se o projeto encontra-se dentro das regiões de especificação, caso seu projeto não esteja na região de especificação basta ele recalculá-lo de forma a trazer o *Root-Locus* para a região de projeto, e fazer uma nova verificação com os novos valores de projeto.

Sendo assim o *software* conseguiu atender as expectativas de ganhar agilidade e praticamente zerar os erros grosseiros no processo de cálculo dos parâmetros e análise do *Root Locus*, otimizando o tempo de projeto na hora de desenvolver seus controladores.

#### 4.6 Projeto de Controladores via LMIs

O principal intuito deste projeto, sempre foi verificar a factibilidade do *software* livre Python com relação aos projetos de controladores, utilizando técnicas clássicas de controle e técnicas de controle robusto.

Abordaremos agora um comparativo entre projetos realizados no MATLAB® e o mesmo projeto implementado utilizando Python. Inicialmente, realizou-se um projeto de ganho de realimentação visando garantir apenas a estabilidade do sistema, sem considerar restrições ou se preocupar com índices de desempenho.

Neste capítulo serão apresentados vários projetos de ganhos de realimentação, sempre com o intuito de manter a estabilidade e um bom desempenho do sistema de suspensão ativa de bancada.

A cada novo projeto foi inserido uma restrição ou se fez o uso de técnicas que melhoram o desempenho, buscando assim chegar a um controlador chaveado no qual a solução foi obtida

via MATLAB® e Python, sendo possível mostrar que Python é capaz de solucionar as LMIs e ser uma segunda opção segura e confiável com relação ao *software* proprietário MATLAB®.

#### 4.7 Projeto controlador com único ganho K

Inicialmente, realizou-se um projeto de ganho de realimentação no qual visamos garantir apenas a estabilidade do sistema. Com este projeto foi possível ter um *start* para verificar como seria o desenvolvimento de projetos que solucionam LMIs utilizando Python, e a partir deste primeiro projeto verificou-se de início quais seriam as dificuldades ao longo do trabalho.

Neste primeiro projeto após execução da simulação do controlador, verificou-se a necessidade de utilizar a restrição na taxa de decaimento, que neste primeiro projeto foi de  $\gamma = 7$  e fixar o valor máximo do sinal de controle em  $\mu = 1225$ , para não cair em uma faixa de operação onde o controlador não estabilizasse.

##### 4.7.1 Projeto controlador MATLAB®

O projeto de controladores via LMIs já é amplamente estudado e o *software* utilizado para resolução destes projetos normalmente é o MATLAB®, logo a seguir será apresentado o projeto do controlador com único ganho  $K$  que tem como base a lei de controle apresentado em (58).

Código Matlab® apresentado abaixo:

```
%parametros iniciais
clear
tic
ms = 2.45;
mus = 1;
ks = 900;
kus = 2500;
bs = 7.5;
bus = 5;
x1(1)=-0.02;
x1(2)=0.02;
x2(1)=-0.15;
x2(2)=0.15;
x3(1)=-0.02;
x3(2)=0.02;
x4(1)=-0.15;
x4(2)=0.15;
gama=7;
%matriz A e matriz B
```



```

A = [0 1 0 -1; -ks/ms -bs/ms 0 bs/ms; 0 0 0 1; ks/mus bs/mus -kus/mus -(bs+bus)/mus];
B = [0; 1/ms; 0; -1/mus];
%matriz x(0)
cont = 1;
for i = 1:1:2
    for j = 1:1:2
        for k = 1:1:2
            for l = 1:1:2
                x0{cont}=[x1(i) x2(j) x3(k) x4(l)];
                cont = cont + 1;
            end
        end
    end
end
end
%Definindo a matriz X que estará nas restrições (X*A' - M'*B' + A*X - B*M < 0)
X = sdpvar(4,4);
U = 35^2;
%Definindo a matriz M que estará nas restrições (X*A' - M'*B' + A*X - B*M < 0)
M = sdpvar(1,4);
%Criando as restrições para estabilização segundo as LMIs
Restr = (X > 0) + (X*A' - M'*B' + A*X - B*M + 2*gama*X < 0) + ([ X M'; M U*eye(1)] > 0);
for s = 1:16
    Restr = Restr + ([1, x0{s}; x0{s}', X] > 0);
end
%Configurando o Solver
opts = sdpsettings('solver','lmilab');
%Resolvendo as LMIs
sol = solvesdp(Restr,[],opts)
%fazendo a verificação se as LMIs encontradas pelo solvesdp satisfazem as
%restrições lmis
[r,d] = checkset(Restr); %A função checkset
%obtm os resíduos das restrições do problema de otimização 'Primal' e 'Dual'.
if sum(r<0) == 0
    disp('O Sistema pode ser controlado')
    X = double(X)%double converte a estrutura X (matriz variável do yalmip)
    %em uma matriz numérica
    M = double(M)
    K = M*inv(X)
    Autovalor = eig(A-B*K)
else
    disp('O sistema NÃO pode ser encontrado ')
end
end
toc

```

Após o desenvolvimento do *script* podemos executar o projeto e verificar os resultados

apresentados pelo *solver*, a seguir estão listados os resultados obtidos via MATLAB®:

```

>>VersaoFinalDeterminacaoGanhoKCorrigido
#####
You are using LMILAB. Please don't use LMILAB with YALMIP
https://yalmip.github.io/solver/lmilab/

Install a better SDP solver https://yalmip.github.io/allsolvers/

To get rid of this message, edit calllmilab.m
(but don't expect support when things do not work, YALMIP +
LMILAB => No support)
#####
sol =

```

struct with fields:

yalmipversion:

'20190425'

yalmiptime: 0.18832

solvertime: 0.18068

info: 'Successfully solved (LMILAB)'

problem: 0

O Sistema pode ser controlado

X =

0.10581	-0.27004	-0.092504	0.47075
-0.27004	5.2947	0.24317	-34.725
-0.092504	0.24317	0.083036	-0.58135
0.47075	-34.725	-0.58135	289.56

M =

-0.8863	33.7	0.33464	-24.39
---------	------	---------	--------

K =

115.57	42.521	42.653	4.9129
--------	--------	--------	--------

Autovalor =

-7.0012	+	58.721i
-7.0012	-	58.721i
-7.0007	+	15.726i
-7.0007	-	15.726i

Elapsed time is 0.718098 seconds.

>>

Neste primeiro projeto, o intuito foi realizar o projeto do controlador utilizando a teoria de Lyapunov e ter um primeiro projeto com um único ganho  $K$  solucionado pelas condições impostas pelas LMIs. Podemos notar que foi possível encontrar um controlador que estabiliza a planta e notamos também que a solução apresenta os autovalores formado por pares complexos conjugados sempre no lado esquerdo do eixo real no plano complexo.

### 4.7.2 Projeto Python

Durante a montagem do primeiro projeto em Python encontrou-se várias dificuldades, visto que, são poucos os trabalhos que mencionam a solução de LMIs utilizando Python, logo, foi necessário realizar uma busca e estudar as bibliotecas disponíveis para solucionar as LMIs utilizando a nova plataforma Python, neste processo foi necessário acessar as informações disponibilizadas pelas páginas de cada uma das bibliotecas PICOS (<https://picos-api.gitlab.io/picos/introduction.html>) e CVOXPT (<https://cvxopt.org/>) sendo de extrema importância para o desenvolvimento do trabalho, visto que em ambas as páginas apresentam exemplos de implementação de ambas as ferramentas.

Após realizar os estudos dos exemplos, foi necessário entender como seria feita a transcrição do código do MATLAB® para o código em Python, este foi um processo mais longo, visto que, foi necessário aprender um novo método de escrita de funções e compreender como as bibliotecas PICOS e CVXOPT se comunicam com a biblioteca de matrizes *NumPy*.

Entendido estes pontos foi possível montar o primeiro esboço para o controlador com único ganho em Python.

```

## Projeto do controlador para realimentação de estados para SLITs. em Python.
## Em Python, devemos declarar os pacotes que queremos utilizar
## Determinação do Ganho K
tic()
import cvxopt as cvx # cvxopt é o pacote do solver. ex. sedumi, lmilab
import picos as pic # picos é o pacote de interfaceamento com o solver. ex. yalmip
import numpy as np # numpy é um pacote que suporta operações com vetores e matrizes
from numpy import linalg as LA # linalg é um subpacote do numpy para algebra linear,
# import (pacote) as (nome), faz-se a abreviação para utilizar o pacote
from picos import Constant
tic()
epsilon = 1e-15 # O solver cvxopt possui uma tolerância de arredondamento
lmi = pic.Problem() # Definir a variável que vamos resolver
#Constantes do Sistema
ms = 2.45
mus = 1
ks = 900
kus = 2500
bs = 7.5
bus = 5
gama = 3
#Definindo as constantes(parâmetros conhecidos) (matriz A)
A = np.array([[0, 1, 0, -1], [-ks/ms, -bs/ms, 0, bs/ms], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
B = np.array([[0], [1/ms], [0], [-1/mus]])
# Transformando em parâmetro picos
A = pic.new_param('A', A)
B = pic.new_param('B', B)
x0 = [Constant("x0[{}]").format(cont), range(cont, cont + 4), (1,4)] for cont in range(16)]
x1 = [Constant("x1[{}]").format(i), range(i, i + 2), (1,2)] for i in range(2)]
x2 = [Constant("x2[{}]").format(j), range(j, j + 2), (1,2)] for j in range(2)]

```

```

x3 = [Constant("x3{0}".format(k), range(k, k + 2), (1,2)) for k in range(2)]
x4 = [Constant("x4{0}".format(l), range(l, l + 2), (1,2)) for l in range(2)]
x1[0]=-0.02
x1[1]=0.02
x2[0]=-0.15
x2[1]=0.15
x3[0]=-0.02
x3[1]=0.02
x4[0]=-0.15
x4[1]=0.15
h=0
for c in range(2):
    for v in range(2):
        for b in range(2):
            for n in range(2): #Arrumar
                x0[h] = np.array([[x1[c]], [x2[v]], [x3[b]], [x4[n]]])
                h=h+1
x0[0] = pic.new_param('x0[0]', x0[0])
x0[1] = pic.new_param('x0[1]', x0[1])
x0[2] = pic.new_param('x0[2]', x0[2])
x0[3] = pic.new_param('x0[3]', x0[3])
x0[4] = pic.new_param('x0[4]', x0[4])
x0[5] = pic.new_param('x0[5]', x0[5])
x0[6] = pic.new_param('x0[6]', x0[6])
x0[7] = pic.new_param('x0[7]', x0[7])
x0[8] = pic.new_param('x0[8]', x0[8])
x0[9] = pic.new_param('x0[9]', x0[9])
x0[10] = pic.new_param('x0[10]', x0[10])
x0[11] = pic.new_param('x0[11]', x0[11])
x0[12] = pic.new_param('x0[12]', x0[12])
x0[13] = pic.new_param('x0[13]', x0[13])
x0[14] = pic.new_param('x0[14]', x0[14])
x0[15] = pic.new_param('x0[15]', x0[15])
U = 1225
# Definindo as variáveis (matriz variável P)
X = lmi.add_variable('X',(4,4),'symmetric') # definir as variáveis
G = lmi.add_variable('G',(1,4))
I = pic.I(1)
# Definindo as Restrições LMI do problema
#Sem Epsilon
lmi.add_constraint((A*X +X*A.H -B*G -G.H*B.H +2*gama*X << 0))
lmi.add_constraint( ((X & G.H) // (G & U*I)) >> 0)
lmi.add_constraint(X >> 0)
for s in range(16):
    lmi.add_constraint(((1 & x0[s].H) // (x0[s] & X)) >> 0)
# Mostrar as restrições LMIs
print(lmi)
print('type: '+lmi.type)
print('status: '+lmi.status) # foi resolvido?
# Resolvendo as LMIs
sol = lmi.solve(solver='cvxopt',verbose=0)
print('status: '+lmi.status) #foi resolvido?
# Convertendo nossos parâmetros do solver em Arrays
print('Matriz Ótima P:')
X= np.array(X.value)
G= np.array(G.value)
A= np.array(A.value)
B= np.array(B.value)
print(X)
print("\n Os Autovalores de X são:")
print(LA.eigvals(X))
eigX = LA.eigvals(X)

```

```

print("\nOs Autovalores de A*X+X*A.H-B*G-G.H*B.H são:")
print(LA.eigvals(A @X+X@np.transpose(A) - B@G - np.transpose(G)@np.transpose(B)))
eigLMI = LA.eigvals(A @X+X@np.transpose(A) - B@G - np.transpose(G)@np.transpose(B)) ## Em numpy, A*B significa a
multiplicação indexada dos elementos
## O operador @ realiza a multiplicação matricial correta.
if eigX.__gt__(0).all():
    if eigLMI.__lt__(0).all():
        print("\nSistema Factível. \nÉ possível controlar.")
    else: print("\nSistema Infactível")
## Controlador
K = G@(LA.inv(X))
print("\nControlador K:", K)
print("\nAutorvalor de A-B*K:\n", LA.eigvals(A-B@K))
#-----#
# objective value #
#-----#
#print( \nthe optimal value of this problem is:')
#print(lmi.obj_value())          #"print objective" would also work, because objective is valued]
toc() # returns

```

Quando finalmente foi possível verificar cada uma das funções e após várias tentativas foi possível compilar o projeto em Python, os resultados obtidos são apresentados a seguir:

#### Feasibility Problem

find an assignment

for

1×4 real variable G

4×4 symmetric variable X

subject to

$A \cdot X + X \cdot A^T - B \cdot G - G^T \cdot B^T + 6 \cdot X \leq 0$

$[X, G^T; G, 1225] \geq 0$

$X \geq 0$

$[1, x0[0]^T; x0[0], X] \geq 0$

$[1, x0[1]^T; x0[1], X] \geq 0$

$[1, x0[2]^T; x0[2], X] \geq 0$

$[1, x0[3]^T; x0[3], X] \geq 0$

$[1, x0[4]^T; x0[4], X] \geq 0$

$[1, x0[5]^T; x0[5], X] \geq 0$

$[1, x0[6]^T; x0[6], X] \geq 0$

$[1, x0[7]^T; x0[7], X] \geq 0$

$[1, x0[8]^T; x0[8], X] \geq 0$

$[1, x0[9]^T; x0[9], X] \geq 0$

$[1, x0[10]^T; x0[10], X] \geq 0$

$[1, x0[11]^T; x0[11], X] \geq 0$

$[1, x0[12]^T; x0[12], X] \geq 0$

$[1, x0[13]^T; x0[13], X] \geq 0$

$[1, x0[14]^T; x0[14], X] \geq 0$

$[1, x0[15]^T; x0[15], X] \geq 0$

type: Feasibility Problem

status: unsolved

status: optimal

Matriz Ótima P:

$[[ 1.74578338e-02 -8.87263223e-02 -4.15606578e-03 2.08842062e-02]$

$[-8.87263223e-02 4.89158433e+00 3.60280456e-02 -1.21331864e+00]$

$[-4.15606578e-03 3.60280456e-02 8.08479643e-03 -6.86948311e-02]$

$[ 2.08842062e-02 -1.21331864e+00 -6.86948311e-02 2.19244719e+01]]$

Os Autovalores de X são:

$[2.20107301e+01 4.80737253e+00 6.35912339e-03 1.71370842e-02]$

Os Autovalores de  $A \cdot X + X \cdot A.H - B \cdot G - G.H \cdot B.H$  são:

$[-1.44527947e+02 -2.97446455e+01 -4.52822841e-02 -2.88115485e-01]$

Sistema Factível.  
É possível controlar.

Controlador K: [[-10.93691213 14.42134686 53.90665472 1.91815767]]

Autorvalor de A-B\*K:  
[-6.73207906+57.86605594j -6.73207906-57.86605594j  
-3.03258636+16.06600218j -3.03258636-16.06600218j]  
Elapsed time: 0.110611 seconds.

O *script* desenvolvido em Python foi capaz de encontrar uma solução para o problema de restrições via LMI e determinar um ganho  $K$  capaz de estabilizar a planta em questão, sendo assim o *software* livre atingiu a nossa primeira meta de ser capaz de solucionar as LMIs.

Ao compararmos os tempos de execução obtidos, notamos que o *script* executado no MATLAB® apresentou um tempo de 0,718098 segundos, enquanto com o *script* executado via Python, foram necessários apenas 0,110611 segundos. Ou seja, para este primeiro caso Python conseguiu encontrar uma solução para o problema aproximadamente 6,5 vezes mais rápido do que o MATLAB®, este fato, traz uma vantagem enorme em se utilizar Python em aplicações onde o tempo de execução do *script* seja muito longo, reduzindo consideravelmente o tempo de processamento.

Analisando os autovalores obtidos pelos *softwares* podemos notar que nos dois casos foram obtidos pares complexos conjugados localizados no lado esquerdo do eixo real.

Portanto podemos verificar inicialmente que a solução de LMIs no projeto de controladores utilizando Python é totalmente factível, sendo possível aumentar a complexidade de itens no qual exigirá um pouco mais da ferramenta estudada.

#### **4.8 Projeto controlador com incerteza na massa total do veículo**

Logo após o primeiro projeto ser executado, iniciou-se a próxima etapa no qual o objetivo foi ampliar um pouco a complexidade do projeto para entender melhor como seria esta transição do *software* MATLAB® para Python.

Neste segundo projeto, o ganho de realimentação foi projetado considerando novamente os critérios de estabilidade segundo Lyapunov, entretanto, agora foi adicionada mais uma incerteza ao projeto, neste caso em específico adicionou-se a incerteza na massa total do veículo, visto que, não se sabe ao certo a massa que o veículo terá ao receber passageiros, então

nesse projeto, adotamos como limite inferior da massa do veículo  $m_{s_1} = 1,455 \text{ kg}$  e como limite superior para a massa do veículo o valor de  $m_{s_2} = 2,45 \text{ kg}$ .

Ao considerar a massa total do veículo como incerto, garante-se que o controlador da suspensão ativa apresentará no mínimo um desempenho aceitável para diferentes quantidades de pesos de passageiros, que são situações possíveis no dia a dia.

Novamente considerou-se no projeto do controlador uma taxa de decaimento igual a  $\gamma = 5$

#### 4.8.1 Projeto controlador MATLAB®

Neste projeto utilizou-se as condições que estão dispostas no Teorema 4, o código desenvolvido no MATLAB® é apresentado a seguir:

```
tic
clear
clc
%parametros iniciais
ms1 = 1.455;
ms2 = 2.45;
mus = 1;
ks = 900;
kus = 2500;
bs = 7.5;
bus = 5;
x1(1)=-0.02;
x1(2)=0.02;
x2(1)=-0.15;
x2(2)=0.15;
x3(1)=-0.02;
x3(2)=0.02;
x4(1)=-0.15;
x4(2)=0.15;
gama=5;
%matriz A e matriz B
A1 = [0 1 0 -1; -ks/ms1 -bs/ms1 0 bs/ms1; 0 0 0 1; ks/mus bs/mus -kus/mus -(bs+bus)/mus];
A2 = [0 1 0 -1; -ks/ms2 -bs/ms2 0 bs/ms2; 0 0 0 1; ks/mus bs/mus -kus/mus -(bs+bus)/mus];
B1 = [0; 1/ms1; 0; -1/mus];
B2 = [0; 1/ms2; 0; -1/mus];
%matriz x(0)
cont = 1;
for i = 1:1:2
    for j = 1:1:2
        for k = 1:1:2
            for l = 1:1:2
                x0{cont}=[x1(i) x2(j) x3(k) x4(l)];
                cont = cont + 1;
            end
        end
    end
end
end
%Definindo a matriz X que estará nas restrições (X*A' - M'*B' + A*X - B*M < 0)
```

```

X = sdpvar(4,4);
U = 20^2;
%Definindo a matriz M que estará nas restrições (X*A' - M*B' + A*X - B*M < 0)
M = sdpvar(1,4);
%Criando as restrições para estabilização segundo as LMIs
Restr = (X > 0) + ((X*A1' - M*B1' + A1*X - B1*M < 0) + (X*A2' - M*B2' + A2*X - B2*M < 0) + ([ X M';
U*eye(1)] > 0));
for s = 1:16
    Restr = Restr + ([1, x0{s}; x0{s}', X] > 0);
end
%Configurando o Solver
opts = sdpsettings('solver','lmlab');
%Resolvendo as LMIs
sol = solvesdp(Restr,[],opts)
%fazendo a verificação se as LMIs encontradas pelo solvesdp satisfazem as
%restrições lmis
[r,d] = checkset(Restr); %A função checkset
%obtm os resíduos das restrições do problema de otimização 'Primal' e 'Dual'.
if sum(r<0) == 0
    disp('O Sistema pode ser controlado')
    X = double(X)%double converte a estrutura X (matriz variável do yalmip)
    %em uma matriz numérica
    M = double(M)
    K = M*inv(X)
    Autovalor_A1 = eig(A1-B1*K)
    Autovalor_A2 = eig(A2-B2*K)
else
    disp('O sistema NÃO pode ser encontrado ')
end
toc

```

Após adicionar a incerteza na massa total do veículo podemos verificar a seguir os resultados apresentados utilizando o *script* do MATLAB®.

```

sol =

struct with fields:

    yalmipversion:

    '20190425'
    yalmiptime: 0.080419
    solvertime: 0.051581
    info: 'Successfully solved (LMILAB)'
    problem: 0

```

O Sistema pode ser controlado

X =

0.084297	-0.31007	-0.056251	0.79759
-0.31007	11.556	0.19399	-23.932
-0.056251	0.19399	0.052264	-0.51664
0.79759	-23.932	-0.51664	159.5



M =

```
-0.71237          53.768          0.19922          13.442
```

K =

```
-1.8643          7.1592          -13.661          1.1236
```

Autovalor\_A1 =

```
-7.6854 + 59.096i
-7.6854 - 59.096i
-3.0403 + 20.622i
-3.0403 - 20.622i
```

Autovalor\_A2 =

```
-6.7868 + 58.725i
-6.7868 - 58.725i
-1.8931 + 16.083i
-1.8931 - 16.083i
```

Elapsed time is 0.330965 seconds.

>>

Novamente podemos notar que foi possível solucionar o projeto do controlador, obtemos o ganho  $K$ . Notamos novamente que os autovalores se encontram novamente no lado esquerdo do eixo real formado por pares complexos conjugados como esperado.

#### 4.8.2 Projeto controlador Python

Nesta segunda etapa, para adicionar a incerteza na massa do veículo não foram encontrados maiores problemas, sendo que, com o primeiro projeto já desenvolvido bastou adicionar o novo valor da massa do veículo, inserir mais uma matriz  $A$  e mais uma matriz  $B$ , e adicionar mais uma restrição, logo o código que descreve este processo é apresentado abaixo:

```
## Projeto do controlador para realimentação de estados para SLITs. em Python.
## Em Python, devemos declarar os pacotes que queremos utilizar
## Controlador Robusto Com Massa Variavel
tic()
import cvxopt as cvx # cvxopt é o pacote do solver. ex. sedumi, lmilab
import picos as pic # picos é o pacote de interfaceamento com o solver. ex. yalmip
import numpy as np # numpy é um pacote que suporta operações com vetores e matrizes
from numpy import linalg as LA # linalg é um subpacote do numpy para algebra linear,
# import (pacote) as (nome), faz-se a abreviação para utilizar o pacote
from picos import Constant
```

```

epsilon = 1e-15 # O solver cvxopt possui uma tolerância de arredondamento
lmi = pic.Problem() # Definir a variável que vamos resolver
#Constantes do Sistema
ms1 = 1.455;
ms2 = 2.45;
mus = 1;
ks = 900;
kus = 2500;
bs = 7.5;
bus = 5;
gama=5;
# Definindo as constantes(parâmetros conhecidos) (matriz A)
A1 = np.array([[0, 1, 0, -1], [-ks/ms1, -bs/ms1, 0, bs/ms1], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
A2 = np.array([[0, 1, 0, -1], [-ks/ms2, -bs/ms2, 0, bs/ms2], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
B1 = np.array([[0], [1/ms1], [0], [-1/mus]])
B2 = np.array([[0], [1/ms2], [0], [-1/mus]])
# Transformando em parâmetro picos
A1 = pic.new_param('A1', A1 )
A2 = pic.new_param('A2', A2 )
B1 = pic.new_param('B1', B1 )
B2 = pic.new_param('B2', B2 )
x0 = [Constant("x0{}".format(cont), range(cont, cont + 4), (1,4)) for cont in range(16)]
x1 = [Constant("x1{}".format(i), range(i, i + 2), (1,2)) for i in range(2)]
x2 = [Constant("x2{}".format(j), range(j, j + 2), (1,2)) for j in range(2)]
x3 = [Constant("x3{}".format(k), range(k, k + 2), (1,2)) for k in range(2)]
x4 = [Constant("x4{}".format(l), range(l, l + 2), (1,2)) for l in range(2)]
x1[0]=-0.02
x1[1]=0.02
x2[0]=-0.15
x2[1]=0.15
x3[0]=-0.02
x3[1]=0.02
x4[0]=-0.15
x4[1]=0.15
h=0
for c in range(2):
    for v in range(2):
        for b in range(2):
            for n in range(2): #Arrumar
                x0[h] = np.array([[x1[c]], [x2[v]], [x3[b]], [x4[n]]])
                h=h+1
#for y in range(16): #Arrumar
#    x0[y] = pic.new_param('x0[y]', x0[y])
x0[0] = pic.new_param('x0[0]', x0[0])
x0[1] = pic.new_param('x0[1]', x0[1])
x0[2] = pic.new_param('x0[2]', x0[2])
x0[3] = pic.new_param('x0[3]', x0[3])
x0[4] = pic.new_param('x0[4]', x0[4])
x0[5] = pic.new_param('x0[5]', x0[5])
x0[6] = pic.new_param('x0[6]', x0[6])
x0[7] = pic.new_param('x0[7]', x0[7])
x0[8] = pic.new_param('x0[8]', x0[8])
x0[9] = pic.new_param('x0[9]', x0[9])
x0[10] = pic.new_param('x0[10]', x0[10])
x0[11] = pic.new_param('x0[11]', x0[11])
x0[12] = pic.new_param('x0[12]', x0[12])
x0[13] = pic.new_param('x0[13]', x0[13])
x0[14] = pic.new_param('x0[14]', x0[14])
x0[15] = pic.new_param('x0[15]', x0[15])
U = 1225
# Definindo as variáveis (matriz variável P)
X = lmi.add_variable('X',(4,4),'symmetric') # definir as variáveis

```

```

G = lmi.add_variable('G',(1,4))
I = pic.I(1)
# Definindo as Restrições LMI do problema
# Com Epsilon
#lmi.add_constraint((A*X+X*A.H - B*G-G.H*B.H + epsilon << 0))
#lmi.add_constraint(X - epsilon >> 0)
# Sem Epsilon
lmi.add_constraint((A1*X+X*A1.H - B1*G-G.H*B1.H +2*gama*X << 0)) #Verificar necessidade do Epsilon
lmi.add_constraint((A2*X+X*A2.H - B2*G-G.H*B2.H +2*gama*X << 0))
lmi.add_constraint((X & G.H) // (G & U*I) >> 0)
lmi.add_constraint(X >> 0)
for s in range(16):
    lmi.add_constraint(((1 & x0[s].H) // (x0[s] & X)) >> 0)
# Mostrar as restrições LMIs
print(lmi)
print('type: '+lmi.type)
print('status: '+lmi.status) # foi resolvido?
# Resolvendo as LMIs
sol = lmi.solve(solver='cvxopt',verbose=0)
print('status: '+lmi.status) #foi resolvido?
#cvx_sol = sol.get('cvxopt_sol') # com get, é possível retomar alguns parâmetros que o solver encontrou
#print('\nResíduo primal: ', primal)
#tempo = sol.get('time')
#print('\nTempo de processamento: ',np.longlong(tempo))
# Convertendo nossos parâmetros do solver em Arrays
print('Matriz Ótima P:')
X= np.array(X.value)
G= np.array(G.value)
A1= np.array(A1.value)
A2= np.array(A2.value)
B1= np.array(B1.value)
B2= np.array(B2.value)

print(X)
print("\n Os Autovalores de X são:")
print(LA.eigvals(X))
eigX = LA.eigvals(X)
print("\nOs Autovalores de A1*X+X*A1.H-B1*G-G.H*B1.H são:")
print(LA.eigvals(A1@X+X@np.transpose(A1) - B1@G - np.transpose(G)@np.transpose(B1)))
eigLMI = LA.eigvals(A1@X+X@np.transpose(A1) - B1@G - np.transpose(G)@np.transpose(B1)) ## Em numpy, A*B significa a multiplicação indexada dos elementos
## O operador @ realiza a multiplicação matricial correta.
print("\nOs Autovalores de A2*X+X*A2.H-B2*G-G.H*B2.H são:")
print(LA.eigvals(A2@X+X@np.transpose(A2) - B2@G - np.transpose(G)@np.transpose(B2)))
eigLMI = LA.eigvals(A2@X+X@np.transpose(A2) - B2@G - np.transpose(G)@np.transpose(B2)) ## Em numpy, A*B significa a multiplicação indexada dos elementos
## O operador @ realiza a multiplicação matricial correta.
if eigX.__gt__(0).all():
    if eigLMI.__lt__(0).all():
        print("\nSistema Factível. \nÉ possível controlar.")
    else: print("\nSistema Infactível")
## Controlador
K = G@(LA.inv(X))
print("\nControlador K:', K)
print("\nAutorvalor de A1-B1*K:\n', LA.eigvals(A1-B1@K))
print("\nAutorvalor de A2-B2*K:\n', LA.eigvals(A2-B2@K))
#-----#
# objective value #
#-----#
#print('\nthe optimal value of this problem is:')
#print(lmi.obj_value()) # "print objective" would also work, because objective is valued
toc() # returns

```

Com o código devidamente transcrito e após realizar a execução obtemos o seguinte

resultado:

```

Feasibility Problem
find an assignment
for
  1x4 real variable G
  4x4 symmetric variable X
subject to
  A1·X + X·A1T - B1·G - GT·B1T + 10·X ≤ 0
  A2·X + X·A2T - B2·G - GT·B2T + 10·X ≤ 0
[X, GT; G, 1225] ≥ 0
X ≥ 0
[1, x0[0]T; x0[0], X] ≥ 0
[1, x0[1]T; x0[1], X] ≥ 0
[1, x0[2]T; x0[2], X] ≥ 0
[1, x0[3]T; x0[3], X] ≥ 0
[1, x0[4]T; x0[4], X] ≥ 0
[1, x0[5]T; x0[5], X] ≥ 0
[1, x0[6]T; x0[6], X] ≥ 0
[1, x0[7]T; x0[7], X] ≥ 0
[1, x0[8]T; x0[8], X] ≥ 0
[1, x0[9]T; x0[9], X] ≥ 0
[1, x0[10]T; x0[10], X] ≥ 0
[1, x0[11]T; x0[11], X] ≥ 0
[1, x0[12]T; x0[12], X] ≥ 0
[1, x0[13]T; x0[13], X] ≥ 0
[1, x0[14]T; x0[14], X] ≥ 0
[1, x0[15]T; x0[15], X] ≥ 0
type: Feasibility Problem
status: unsolved
status: optimal
Matriz Ótima P:
[[ 6.06263162e-03 -3.94148550e-02 -3.17180087e-03 2.02666239e-02]
 [-3.94148550e-02 1.10451254e+00 1.85297762e-02 -1.22609969e+00]
 [-3.17180087e-03 1.85297762e-02 3.20882890e-03 -2.52397835e-02]
 [ 2.02666239e-02 -1.22609969e+00 -2.52397835e-02 9.69726586e+00]]

Os Autovalores de X são:
[9.86893710e+00 9.34638403e-01 1.08038228e-03 6.39397457e-03]

Os Autovalores de A1·X+X·A1.H-B1·G-G.H·B1.H são:
[-1.29590849e+02 -1.48464092e+01 -1.47967008e-01 -1.18903435e-02]

Os Autovalores de A2·X+X·A2.H-B2·G-G.H·B2.H são:
[-1.22732235e+02 -1.27744603e+01 -1.12074685e-02 -9.33474939e-02]

Sistema Factível.
É possível controlar.

Controlador K: [[ 35.17286626 35.95400863 -18.15075191 3.179634 ]]

Autorvalor de A1-B1·K:
[-9.41750489+57.4376619j -9.41750489-57.4376619j
 -10.17532712+19.2552368j -10.17532712-19.2552368j]

Autorvalor de A2-B2·K:
[-7.4504714 +58.41109031j -7.4504714 -58.41109031j
 -6.07787663+15.43602006j -6.07787663-15.43602006j]
Elapsed time: 0.137632 seconds.

```

Neste segundo projeto, a maior dificuldade foi conseguir converter o projeto do MATLAB® para a plataforma Python, visto que, neste caso tivemos que trabalhar com duas matrizes  $A$  e duas matrizes  $B$  alterando um pouco a dinâmica com relação ao que foi realizado no projeto anterior.

Novamente Python foi capaz de encontrar uma solução dentro de um conjunto de possíveis soluções para o problema das LMIs, e conseqüentemente definir um ganho  $K$  para nosso controlador considerando a incerteza na massa do veículo.

Vale observar novamente a agilidade que Python encontra para solucionar o *script* levando apenas 0,137632 segundos, enquanto, no MATLAB® foram necessários 0,330965 segundos. Neste projeto Python conseguiu ser aproximadamente 2,5 vezes mais rápido que o MATLAB®, novamente trazendo um enorme atrativo para sua utilização, principalmente em projetos que demandem elevados tempos de processamento.

O que mostra novamente que é possível utilizar Python como uma segunda opção com relação ao uso do MATLAB®. Sendo assim todo o esforço empregado para a conversão de projeto do MATLAB® para Python retornou um resultado satisfatório atingindo um segundo objetivo.

Olhando para os autovalores apresentados nota-se que em ambos os *softwares* obtiveram pares complexos conjugados localizados no lado esquerdo do eixo real.

#### **4.9 Primeiro controlador chaveado considerando incerteza na massa do veículo**

Na sequência foi implementado o primeiro controlador chaveado considerando a incerteza na massa do veículo. Este foi de certo modo mais simples desenvolver no MATLAB® visto que, já se utilizava a ferramenta, por outro lado, no ambiente Python já foi um pouco mais trabalhoso devido a necessidade de compreender melhor o funcionamento da biblioteca PICOS em conjunto com a biblioteca *NumPy* no trato com as matrizes. Novamente foram adicionadas pequenas melhorias de um projeto para outro visando compreender melhor o modo como seria implementado o script em Python.

#### 4.9.1 Projeto controlador MATLAB®

Novamente como os projetos anteriores já traziam uma base, para o controle chaveado foi necessário apenas adicionar o número de controladores que seriam calculados, adicionar as novas matrizes que entraram na restrição, remodelar as restrições das LMIs. Diante de tal descrição, o projeto desenvolvido no MATLAB® é mostrado a seguir:

```

clear all
clc
tic
%parametros iniciais
ms1 = 1.455;
ms2 = 2.45;
mus = 1;
ks = 900;
kus = 2500;
bs = 7.5;
bus = 5;
nv=2;
%matriz A e matriz B
A{1} = [0 1 0 -1; -ks/ms1 -bs/ms1 0 bs/ms1; 0 0 0 1; ks/mus bs/mus -kus/mus -(bs+bus)/mus];
A{2} = [0 1 0 -1; -ks/ms2 -bs/ms2 0 bs/ms2; 0 0 0 1; ks/mus bs/mus -kus/mus -(bs+bus)/mus];
B{1} = [0; 1/ms1; 0; -1/mus];
B{2} = [0; 1/ms2; 0; -1/mus];
%Definindo a matriz X que estará nas restrições (X*A' - M'*B' + A*X - B*M < 0)
X = sdpvar(4,4,'symmetric');
%Definindo a matriz M que estará nas restrições (X*A' - M'*B' + A*X - B*M < 0)
for j=1:nv
    M{j} = sdpvar(1,4,'full','real');
    Q{j} = sdpvar(4,4,'full','real');
end
for i=1:nv
    Z{i} = sdpvar(4,4,'full','real');
end

%Criando as restrições para estabilização segundo as LMIs
LMIs = [];
LMIs = (X > 0)
for i=1:nv
    LMIs = LMIs + (X*A{i}' + Q{i} + A{i}*X +Z{i} < 0);
    for j=1:nv
        LMIs = LMIs + (-M{j}'*B{i}' -B{i}*M{j} -Q{j} -Z{i} <0);
    end
end
Restr = LMIs
%Configurando o Solver
opts = sdpsettings('solver','lmlab');
%Resolvendo as LMIs
sol = solvesdp(Restr,[],opts)
%fazendo a verificação se as LMIs encontradas pelo solvesdp satisfazem as
%restrições lmis
[r,d] = checkset(Restr); %A função checkset
%obtem os resíduos das restrições do problema de otimização 'Primal' e 'Dual'.
if sum(r<0) == 0
    disp('O Sistema pode ser controlado')
    X = double(X)%double converte a estrutura X (matriz variável do yalmip)

```

```

%em uma matriz numérica
for i=1:nv
    M{i} = double(M{i});
    K{i} = M{i}*inv(X);
    Q{i} = inv(X)*double(Q{i})*inv(X);
end
celldisp(K)
celldisp(Q)
for i=1:nv
    Autovalor_A{i} = eig(A{i}-B{i}*K{i});
end
celldisp(Autovalor_A)
else
    disp('O sistema NÃO pode ser encontrado ')
end
toc

```

Com o código devidamente elaborado, ao executá-lo obtemos o seguinte resultado:

```

+++++++
          ++++++
      |   ID|           Constraint|
          Coefficient range|
+++++++
          ++++++
|   #1|   Matrix inequality 4x4|           1 to 1|
+++++++
          ++++++
+++++++
|   ID|           Constraint|   Coefficient range|
+++++++
|   #1|   Matrix inequality 4x4|           1         to 1|
|   #2|   Element-wise inequality 16x1|       1 to       5000|
|   #3|   Element-wise inequality 16x1|   0.68729     to 2|
|   #4|   Element-wise inequality 16x1|   0.68729     to 2|
|   #5|   Element-wise inequality 16x1|       1 to       5000|
|   #6|   Element-wise inequality 16x1|   0.40816     to 2|
|   #7|   Element-wise inequality 16x1|   0.40816     to 2|
+++++++

Solver for LMI feasibility problems  $L(x) < R(x)$ 
This solver minimizes  $t$  subject to  $L(x) < R(x) + t*I$ 
The best value of  $t$  should be negative for feasibility
Iteration   :           Best value of  $t$  so far

* switching to QR
      1           0.172379
      2           0.029366
      3           0.021122
      4          -0.065584

Result: best value of  $t$ :          -0.065584

f-radius saturation:  0.088% of  $R =$ 

```

1.00e+09

#####

#####

You are using LMILAB. Please don't use LMILAB with YALMIP  
<https://yalmip.github.io/solver/lmilab/>

Install a better SDP solver  
<https://yalmip.github.io/allsolvers/>

To get rid of this message, edit calllmilab.m  
 (but don't expect support when things  
 do not work, YALMIP + LMILAB => No  
 support)

#####

sol =

struct with

fields:

yalmipversion

: '20190425'

yalmiptime: 0.68556

solvertime: 0.074443

info: 'Successfully

solved (LMILAB)' problem: 0

O Sistema pode ser controlado

X =

358.41	-228.73	0.57143	-148.63
-228.73	218.82	0.17334	94.04
0.57143	0.17334	0.39916	-0.76072
-148.63	94.04	-0.76072	228.78

K{1} =

-902.17	-10.337	1287.5	7.7911
---------	---------	--------	--------

K{2} =



```
-900.88      -8.6305      1457.3      8.4195
```

```
Q{1} =
```

```
0.84684      1.3496      445.67      -1.0033
1.7184      -2.5449      683.36      2.3606
61.428      31.778      -11082      34.984
-2.7242     -0.38315     2.7197      0.35872
```

```
Q{2} =
```

```
0.84397      1.3419      444.44      -0.99935
1.7107      -2.5602      680.95      2.3636
60.2        29.368      -11073      35.752
-2.7203     -0.38019     3.4878      0.36433
```

```
Autovalor_A{1} =
```

```
-3.3579 + 34.892i
-3.3579 - 34.892i
-0.65883 + 0i
4.6156 + 0i
```

```
Autovalor_A{2} =
```

```
-2.3603 + 32.244i
-2.3603 - 32.244i
-0.52764 + 0i
1.6291 + 0i
```

```
Elapsed time is 2.017078 seconds.
```

```
>>
```

Notamos que novamente foi encontrada uma solução e neste caso tivemos um tempo total de execução do *script* de 2,017078 segundos.

## 4.9.2 Projeto controlador Python

No projeto do primeiro controlador chaveado utilizando a plataforma de programação Python gerou um trabalho um pouco maior, sendo que, foi necessário criar mais variáveis, inserir mais matrizes e fazer novamente a declaração de matrizes na biblioteca PICOS, montar a matriz utilizando *NumPy*, converter está matriz para PICOS novamente para que o *solver* possa utilizá-la e após os cálculos converter a matriz para *NumPy* novamente para que possamos escrevê-la na tela.

Outra dificuldade foi entender como utilizar o laço de “*for*” juntamente com a biblioteca PICOS para adicionar novos parâmetros nas restrições das LMIs. Após sanar estas dúvidas o código do primeiro controlador chaveado foi escrito como segue:

```

## Projeto do controlador para realimentação de estados para SLITs. em Python.
## Em Python, devemos declarar os pacotes que queremos utilizar
## Controlador Robusto Com Massa Variavel
tic()
import cvxopt as cvx # cvxopt é o pacote do solver. ex. sedumi, lmlab
import picos as pic # picos é o pacote de interfaceamento com o solver. ex. yalmip
import numpy as np # numpy é um pacote que suporta operações com vetores e matrizes
from numpy import linalg as LA # linalg é um subpacote do numpy para algebra linear, # import (pac
ote) as (nome), faz-se a abreviação para utilizar o pacote
from picos import Constant
epsilon = 1e-15 # O solver cvxopt possui uma tolerância de arredondamento
lmi = pic.Problem() # Definir a variável que vamos resolver

A = [Constant("A[{}].format(i, range(i, i + 16), (4,4)) for i in range(2)]
B = [Constant("B[{}].format(i, range(i, i + 2), (1,2)) for i in range(2)]
ms = [Constant("ms[{}].format(i, range(i, i + 2), (1,2)) for i in range(2)]
#Constantes do Sistema
ms[0] = 1.455
ms[1] = 2.45
mus = 1;
ks = 900;
kus = 2500;
bs = 7.5;
bus = 5;
nv=2;
# Definindo as constantes(parâmetros conhecidos) (matriz A)
A[0] = np.array([[0, 1, 0, -1], [-ks/ms[0], -bs/ms[0], 0, bs/ms[0]], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
A[1] = np.array([[0, 1, 0, -1], [-ks/ms[1], -bs/ms[1], 0, bs/ms[1]], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
B[0] = np.array([[0], [1/ms[0]], [0], [-1/mus]])
B[1] = np.array([[0], [1/ms[1]], [0], [-1/mus]])
# Transformando em parâmetro picos
A[0] = pic.new_param('A[1]', A[0] )
A[1] = pic.new_param('A[2]', A[1] )
B[0] = pic.new_param('B[1]', B[0] )
B[1] = pic.new_param('B[2]', B[1] )
G = [Constant("G[{}].format(i, range(i, i + 4), (1,4)) for i in range(2)]
Q = [Constant("Q[{}].format(i, range(i, i + 16), (4,4)) for i in range(2)]
Z = [Constant("Z[{}].format(i, range(i, i + 16), (4,4)) for i in range(2)]
# Definindo as variáveis (matriz variável P)
X = lmi.add_variable('X',(4,4),'symmetric') # definir as variáveis

```

```

G[0] = lmi.add_variable('G[0]',(1,4), 'continuous')
G[1] = lmi.add_variable('G[1]',(1,4), 'continuous')
Q[0] = lmi.add_variable('Q[0]',(4,4), 'hermitian')
Q[1] = lmi.add_variable('Q[1]',(4,4), 'hermitian')
Z[0] = lmi.add_variable('Z[0]',(4,4), 'hermitian')
Z[1] = lmi.add_variable('Z[1]',(4,4), 'hermitian')
for i in range(2):
    lmi.add_constraint(( A[i]*X + X*A[i].H + Q[i] + Z[i] ) << 0)
    for j in range(2):
        lmi.add_constraint((-G[j].H*B[i].H - B[i]*G[j] - Q[j] - Z[i]) << 0)
lmi.add_constraint(X >> 0)
# Mostrar as restrições LMIs
print(lmi)
print('type: '+lmi.type)
print('status: '+lmi.status) # foi resolvido?
# Resolvendo as LMIs
sol = lmi.solve(solver='cvxopt',verbose=0)
print('status: '+lmi.status) #foi resolvido?

# Convertendo nossos parâmetros do solver em Arrays
print('Matriz Ótima P:')
X= np.array(X.value)
G[0]= np.array(G[0].value)
G[1]= np.array(G[1].value)
A[0]= np.array(A[0].value)
A[1]= np.array(A[1].value)
B[0]= np.array(B[0].value)
B[1]= np.array(B[1].value)
print(X)
print("\n Os Autovalores de X são:")
print(LA.eigvals(X))
eigX = LA.eigvals(X)
print("\nOs Autovalores de A1*X+X*A1.H-B1*G-G.H*B1.H são:")
print(LA.eigvals(A[0]@X+X@np.transpose(A[0]) - B[0]@G[0] - np.transpose(G[0])@np.transpose(B[0])))
eigLMI = LA.eigvals(A[0]@X+X@np.transpose(A[0]) - B[0]@G[0] - np.transpose(G[0])@np.transpose(B[0])) ## Em numpy
y, A*B significa a multiplicação indexidada dos elementos ## O operado
r @ realiza a multiplicação matricial correta.
print("\nOs Autovalores de A2*X+X*A2.H-B2*G-G.H*B2.H são:")
print(LA.eigvals(A[1]@X+X@np.transpose(A[1]) - B[1]@G[1] - np.transpose(G[1])@np.transpose(B[1])))
eigLMI = LA.eigvals(A[1]@X+X@np.transpose(A[1]) - B[1]@G[1] - np.transpose(G[1])@np.transpose(B[1])) ## Em numpy
y, A*B significa a multiplicação indexidada dos elementos ## O operado
r @ realiza a multiplicação matricial correta.
if eigX.__gt__(0).all():
    if eigLMI.__lt__(0).all():
        print("\nSistema Factível. \nÉ possível controlar.")
else: print("\nSistema Infactível")
    ## Controlador
K = [Constant("K[{}]" .format(i),range(i, i + 2 ), (1,2)) for i in range(2)]
K[0] = G[0]@(LA.inv(X))
print("\nControlador K[0]: \n', K[0])
K[1] = G[1]@(LA.inv(X))
print("\nControlador K[1]: \n', K[1])
print("\nAutorvalor de A1-B1*K:\n', LA.eigvals(A[0]-B[0]@K))
print("\nAutorvalor de A2-B2*K:\n', LA.eigvals(A[1]-B[1]@K))
toc()

```

Neste projeto foram encontradas maiores dificuldades, visto que, não foi tão simples encontrar como eram apresentadas as matrizes com subíndice, matrizes vetorizadas em *Numpy* e como seria feita a conversão destas matrizes que estavam escritas na biblioteca *Numpy* para

a biblioteca PICOS. E após superada as dificuldades buscando informações e exemplos nas páginas de informações de cada biblioteca e seguindo com a compilação do *script* gerou o resultado apresentado a seguir:

```
Complex Feasibility Problem
  find an assignment
  for
    4x4 hermitian variable Q[0], Q[1], Z[0], Z[1]
    1x4 real variable G[i]  $\forall i \in [0..1]$ 
    4x4 symmetric variable X
  subject to
    A[1] · X + X · A[1]T + Q[0] + Z[0]  $\preceq$  0
    -G[0]T · B[1]T - B[1] · G[0] - Q[0] - Z[0]  $\preceq$  0
    -G[1]T · B[1]T - B[1] · G[1] - Q[1] - Z[0]  $\preceq$  0
    A[2] · X + X · A[2]T + Q[1] + Z[1]  $\preceq$  0
    -G[0]T · B[2]T - B[2] · G[0] - Q[0] - Z[1]  $\preceq$  0
    -G[1]T · B[2]T - B[2] · G[1] - Q[1] - Z[1]  $\preceq$  0
    X  $\succeq$  0
type:    Complex Feasibility Problem
status:  unsolved
status:  optimal
Matriz Ótima P:
[[ 1.51367192e+00 -9.05190665e-01 -4.61339847e-03 -2.96687921e-02]
 [-9.05190665e-01  1.39876603e+00  1.12237277e-03  1.09980521e-01]
 [-4.61339847e-03  1.12237277e-03  5.76743329e-03 -2.88736546e-02]
 [-2.96687921e-02  1.09980521e-01 -2.88736546e-02  1.40748100e+01]]
```

Os Autovalores de X são:

```
[1.40759359e+01  2.36243769e+00  5.48950568e-01  5.69118669e-03]
```

Os Autovalores de  $A1 \cdot X + X \cdot A1.H - B1 \cdot G - G.H \cdot B1.H$  são:

```
[-0.05748186 -1.60541601 -2.11792021 -1.99502208]
```

Os Autovalores de  $A2 \cdot X + X \cdot A2.H - B2 \cdot G - G.H \cdot B2.H$  são:

```
[-1.45527368 -2.14356038 -1.95079062 -0.05733537]
```

Sistema Factível.

É possível controlar.

Controlador K[0]:

```
[[-896.53379065 -4.24696208 33.77206721 7.35167247]]
```

Controlador K[1]:

```
[[-895.69034855 -3.15261223 38.34777753 7.35424971]]
```

Autorvalor de  $A1 - B1 \cdot K$ :

```
[[-2.56033294+49.62557553j -2.56033294-49.62557553j
 -1.13171327 +1.06356168j -1.13171327 -1.06356168j]
 [-2.55201211+49.58713291j -2.55201211-49.58713291j
 -1.51481072 +0.84195091j -1.51481072 -0.84195091j]]
```

Autorvalor de  $A2 - B2 \cdot K$ :

```
[[-2.56599791+49.62694028j -2.56599791-49.62694028j
 -0.67205115 +0.99027667j -0.67205115 -0.99027667j]
 [-2.56056399+49.5887065j -2.56056399-49.5887065j
 -0.89953315 +0.98713034j -0.89953315 -0.98713034j]]
Elapsed time: 0.322744 seconds.
```

Mais uma vez o *software* em estudo conseguiu encontrar uma solução para as restrições das LMIs informadas, encontrando um ganho  $K[0]$  e  $K[1]$  possibilitando o uso para controle chaveado.

Novamente um ponto de atenção é para o tempo de execução do *script*, novamente Python se destaca fortemente levando apenas 0,322744 segundos para executar todo o código, enquanto o MATLAB® levou 2,017078 segundos, sendo assim, para este projeto Python conseguiu ser aproximadamente 6 vezes mais rápido que seu concorrente.

O controlador obtido no MATLAB® nos apresentou dois ganhos  $K[0]$  e  $K[1]$  próximos um do outro, o mesmo ocorre em Python.

Até o presente momento a plataforma Python superou as primeiras expectativas e cumpriu um bom papel no projeto dos controladores solucionando o conjunto de LMIs solicitadas pelo *solver* CVXOPT e a interface PICOS. Sendo assim novamente ele se apresenta como uma ferramenta capaz de substituir a utilização do MATLAB® caso necessário, bastando que o usuário se adeque aos padrões de programação exigidos por Python que são padrões inclusive mais brandos do que os exigidos pelo próprio MATLAB®

#### 4.10 Controlador chaveado com redução no sinal de controle.

Por fim o último projeto de controlador chaveado foi adicionada todas as restrições e todos os indicies de desempenho citados anteriormente em um único projeto, visando verificar a resposta do Python para todas estas situações e compará-lo com o que é apresentado pelo MATLAB®.

##### 4.10.1 Projeto MATLAB®

Novamente o primeiro projeto é executado no MATLAB® e serve como base para a conversão para o *script* do Python. A montagem do projeto no MATLAB® se deu como apresentado a seguir:

```
tic
clear all
```

```

clc
%parametros iniciais
beta = 2;
ms1 = 1.455;
ms2 = 2.45;
mus = 1;
ks = 900;
kus = 2500;
bs = 7.5;
bus = 5;
nv=2;
x1(1)=-0.02;
x1(2)=0.02;
x2(1)=-0.15;
x2(2)=0.15;
x3(1)=-0.02;
x3(2)=0.02;
x4(1)=-0.15;
x4(2)=0.15;
%matriz A e matriz B
A{1} = [0 1 0 -1; -ks/ms1 -bs/ms1 0 bs/ms1; 0 0 0 1; ks/mus bs/mus -kus/mus -(bs+bus)/mus];
A{2} = [0 1 0 -1; -ks/ms2 -bs/ms2 0 bs/ms2; 0 0 0 1; ks/mus bs/mus -kus/mus -(bs+bus)/mus];
B{1} = [0; 1/ms1; 0; -1/mus];
B{2} = [0; 1/ms2; 0; -1/mus];
%matriz x(0)
cont = 1;
for i = 1:1:2
    for j = 1:1:2
        for k = 1:1:2
            for l = 1:1:2
                x0{cont}=[x1(i) x2(j) x3(k) x4(l)];
                cont = cont + 1;
            end
        end
    end
end
%Definindo a matriz X que estará nas restrições (X*A' - M*B' + A*X - B*M < 0)
X = sdpvar(4,4,'symmetric');
U = sdpvar(1,1);
%Definindo a matriz M que estará nas restrições (X*A' - M*B' + A*X - B*M < 0)
for h=1:nv
    M{h} = sdpvar(1,4,'full','real');
    Q{h} = sdpvar(4,4,'full','real');
    Z{h} = sdpvar(4,4,'full','real');
end
%Criando as restrições para estabilização segundo as LMIs
LMIs = [];
LMIs = (X > 0)
for i=1:nv
    LMIs = LMIs + (X*A{i}' + Q{i} + A{i}*X +Z{i} + 2*beta*X < 0);
    for j=1:nv
        LMIs = LMIs + (-M{j}'*B{i}' -B{i}*M{j} -Q{j} -Z{i} < 0) + ([ X M{j}'; M{j} U*eye(1)] > 0);
        for s = 1:16
            LMIs = LMIs + ([1, x0{s}; x0{s}', X] > 0);
        end
    end
end
Restr = LMIs
%Configurando o Solver
opts = sdpsettings('solver','lmlab');
%Resolvendo as LMIs
sol = solvesdp(Restr,[U],opts) %quando adiciono a restrição [U] o solver não resolve!!!!!! <-----

```

```

%fazendo a verificação se as LMIs encontradas pelo solvesdp satisfazem as
%restrições lmis
[r,d] = checkset(Restr); %A função checkset
%obtem os resíduos das restrições do problema de otimização 'Primal' e 'Dual'.
if sum(r<0) == 0
    disp('O Sistema pode ser controlado')
    X = double(X)%double converte a estrutura X (matriz variável do yalmip) em uma matriz numérica
    U=double(U);
    mi=sqrt(U)
    for i=1:nv
        M{i} = double(M{i});
        K{i} = M{i}*inv(X);
        Q{i} = inv(X)*double(Q{i})*inv(X);
    end
    celldisp(K)
    celldisp(Q)
    for i=1:nv
        Autovalor_A{i} = eig(A{i}-B{i}*K{i});
    end
    celldisp(Autovalor_A)
else
    disp('O sistema NÃO pode ser encontrado ')
end
toc

```

Com o código devidamente elaborado, ao executá-lo obtemos o seguinte resultado para o controlador chaveado com restrição na entrada e redução da norma do controlador:

```

Result:  feasible solution of required
        accuracy best objective value:56.141249
        guaranteed relative accuracy:  9.50e-04
        f-radius saturation:  13.502% of R =  1.00e+09

#####
You are using LMILAB. Please don't use LMILAB with
YALMIP https://yalmip.github.io/solver/lmilab/

Install a better SDP solver
https://yalmip.github.io/allsolvers/

To get rid of this message, edit calllmilab.m
(but don't expect support when things do not
work, YALMIP + LMILAB => No support)
#####

sol =

struct with fields:

    yalmipversion:

        '20190425'
        yalmiptime: 0.9279
        solvertime: 0.6151
        info: 'Successfully solved'

```

(LMILAB) ' problem: 0

O Sistema pode ser

controlado X =

0.001472	-0.007281	-8.7867e-05	-0.004337
-0.007281	0.52021	0.0032239	-0.015825
-8.7867e-05	0.0032239	0.00073216	-0.0014644
-0.004337	-0.015825	-0.0014644	1.4501

mi =

7.4927

K{1} =

-0.0076391	7.724	28.91	4.1021
------------	-------	-------	--------

K{2} =

15.009	9.9578	13.703	2.1083
--------	--------	--------	--------

Q{1} =

-2.8227e+12	-1.0626e+11	1.0517e+13	6.2426e+08
-4.5697e+09	-1.8591e+09	2.8113e+11	2.2205e+08
6.6983e+12	2.0349e+11	-1.4588e+13	7.8694e+09
-1.3341e+10	-2.9026e+08	1.8925e+10	-2.5092e+07

Q{2} =

-2.8227e+12	-1.0626e+11	1.0517e+13	6.2426e+08
-4.5697e+09	-1.8591e+09	2.8113e+11	2.2205e+08
6.6983e+12	2.0349e+11	-1.4588e+13	7.8694e+09
-1.3341e+10	-2.9026e+08	1.8925e+10	-2.5092e+07

Autovalor\_A{1} =

-6.0285 + 59.028i



```

-6.0285 -      59.028i
-3.402 +      20.68i
-3.402 -      20.68i

Autovalor_A{2} =

-6.4277 +      58.66i
-6.4277 -      58.66i
-2.331 +      16.208i
-2.331 -      16.208i

```

```

Elapsed time is 3.536651 seconds.
>>

```

Notamos que neste último caso tivemos o ganho do controlador chaveado  $K[1]$  e  $K[2]$  bem próximo um do outro, notamos novamente que os autovalores são dados novamente por pares complexos conjugados localizados no lado esquerdo do eixo real e obtivemos um valor de  $\mu$  igual a 7,4927, ou seja, o maior valor que o controlador irá atingir será 7,4927.

#### 4.10.2 Projeto Python

Nesta última etapa a maior dificuldade foi utilizar o laço de repetição “for” juntamente com a biblioteca PICOS, esta dificuldade foi contornada utilizando

```

import cvxopt as cvx # cvxopt é o pacote do solver. ex. sedumi, lmlab
import picos as pic # picos é o pacote de interfaceamento com o solver. ex. yalmip
import numpy as np # numpy é um pacote que suporta operações com vetores e matrizes
from numpy import linalg as LA # linalg é um subpacote do numpy para algebra linear, # import (pac
ote) as (nome), faz-se a abreviação para utilizar o pacote
from picos import Constant
import time
def TicTocGenerator():
    # Generator that returns time differences
    ti = 0 # initial time
    tf = time.time() # final time
    while True:
        ti = tf
        tf = time.time()
        yield tf-ti # returns the time difference
TicToc = TicTocGenerator() # create an instance of the TicTocGen generator
# This will be the main function through which we define both tic() and toc()
def toc(tempBool=True):
    # Prints the time difference yielded by generator instance TicToc
    tempTimeInterval = next(TicToc)
    if tempBool:
        print( "Elapsed time: %f seconds.\n" %tempTimeInterval )
def tic():
    # Records a time in TicToc, marks the beginning of a time interval
    toc(False)
tic ()
lmi = pic.Problem() # Definir a variável que vamos resolver
A = [Constant("A[{}]").format(i), range(i, i + 16), (4,4)) for i in range(2)]

```

```

B = [Constant("B{0}".format(i), range(i, i + 2), (1,2)) for i in range(2)]
ms = [Constant("ms{0}".format(i), range(i, i + 2), (1,2)) for i in range(2)]
x0 = [Constant("x0{0}".format(cont), range(cont, cont + 4), (1,4)) for cont in range(16)]
x1 = [Constant("x1{0}".format(i), range(i, i + 2), (1,2)) for i in range(2)]
x2 = [Constant("x2{0}".format(j), range(j, j + 2), (1,2)) for j in range(2)]
x3 = [Constant("x3{0}".format(k), range(k, k + 2), (1,2)) for k in range(2)]
x4 = [Constant("x4{0}".format(l), range(l, l + 2), (1,2)) for l in range(2)]
x1[0]=-0.02
x1[1]=0.02
x2[0]=-0.15
x2[1]=0.15
x3[0]=-0.02
x3[1]=0.02
x4[0]=-0.15
x4[1]=0.15
G = [Constant("G{0}".format(i), range(i, i + 4), (1,4)) for i in range(2)]
Q = [Constant("Q{0}".format(i), range(i, i + 16), (4,4)) for i in range(2)]
Z = [Constant("Z{0}".format(i), range(i, i + 16), (4,4)) for i in range(2)]
ms[0] = 1.455
ms[1] = 2.45
mus = 1
ks = 900
kus = 2500
bs = 7.5
bus = 5
beta=5
A[0] = np.array([[0, 1, 0, -1], [-ks/ms[0], -bs/ms[0], 0, bs/ms[0]], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
A[1] = np.array([[0, 1, 0, -1], [-ks/ms[1], -bs/ms[1], 0, bs/ms[1]], [0, 0, 0, 1], [ks/mus, bs/mus, -kus/mus, -(bs+bus)/mus]])
B[0] = np.array([[0], [1/ms[0]], [0], [-1/mus]])
B[1] = np.array([[0], [1/ms[1]], [0], [-1/mus]])
#x0 = np.array([[0], [0], [0.03], [0]])
# Transformando em parâmetro picos
A[0] = pic.new_param('A[0]', A[0])
A[1] = pic.new_param('A[1]', A[1])
B[0] = pic.new_param('B[0]', B[0])
B[1] = pic.new_param('B[1]', B[1])
#x0 = pic.new_param('x0', x0)
h=0
for c in range(2):
    for v in range(2):
        for b in range(2):
            for n in range(2): #Arrumar
                x0[h] = np.array([[x1[c]], [x2[v]], [x3[b]], [x4[n]]])
                h=h+1
x0[0] = pic.new_param('x0[0]', x0[0])
x0[1] = pic.new_param('x0[1]', x0[1])
x0[2] = pic.new_param('x0[2]', x0[2])
x0[3] = pic.new_param('x0[3]', x0[3])
x0[4] = pic.new_param('x0[4]', x0[4])
x0[5] = pic.new_param('x0[5]', x0[5])
x0[6] = pic.new_param('x0[6]', x0[6])
x0[7] = pic.new_param('x0[7]', x0[7])
x0[8] = pic.new_param('x0[8]', x0[8])
x0[9] = pic.new_param('x0[9]', x0[9])
x0[10] = pic.new_param('x0[10]', x0[10])
x0[11] = pic.new_param('x0[11]', x0[11])
x0[12] = pic.new_param('x0[12]', x0[12])
x0[13] = pic.new_param('x0[13]', x0[13])
x0[14] = pic.new_param('x0[14]', x0[14])
x0[15] = pic.new_param('x0[15]', x0[15])
# Definindo as variáveis (matriz variável P)
X = lmi.add_variable('X',(4,4),'symmetric') # definir as variáveis

```

```

G[0] = lmi.add_variable('G[0]',(1,4), 'continuous')
G[1] = lmi.add_variable('G[1]',(1,4), 'continuous')
Q[0] = lmi.add_variable('Q[0]',(4,4), 'hermitian')
Q[1] = lmi.add_variable('Q[1]',(4,4), 'hermitian')
Z[0] = lmi.add_variable('Z[0]',(4,4), 'hermitian')
Z[1] = lmi.add_variable('Z[1]',(4,4), 'hermitian')
U = lmi.add_variable('U',(1,1))
I = pic.I(1)
Uprodut = U*I
for i in range(2):
    lmi.add_constraint(((A[i]*X + X*A[i].H + Q[i] + Z[i] + 2*beta*X) << 0))
    for j in range(2):
        lmi.add_constraint((-G[j].H*B[i].H - B[i]*G[j] - Q[j] - Z[i]) << 0)
        lmi.add_constraint((X & G[j].H) // (G[j] & U*I) >> 0)
        for s in range(16):
            lmi.add_constraint(((1 & x0[s].H) // (x0[s] & X)) >> 0)
lmi.add_constraint(X >> 0)
# Mostrar as restrições LMIs
#print(lmi)
print('type: '+lmi.type)
print('status: '+lmi.status) # foi resolvido?
# Resolvendo as LMIs
sol = lmi.solve(solver='cvxopt',verbose=0)
print('status: '+lmi.status) #foi resolvido?
# Convertendo nossos parâmetros do solver em Arrays
print('Matriz Ótima P:')
X= np.array(X.value)
G[0]= np.array(G[0].value)
G[1]= np.array(G[1].value)
A[0]= np.array(A[0].value)
A[1]= np.array(A[1].value)
B[0]= np.array(B[0].value)
B[1]= np.array(B[1].value)
U= np.array(U.value)
print(X)
print("\n Os Autovalores de X são:")
print(LA.eigvals(X))
eigX = LA.eigvals(X)
print("\nOs Autovalores de A1*X+X*A1.H-B1*G-G.H*B1.H são:") #errado mudou a restrição
print(LA.eigvals(A[0]@X+X@np.transpose(A[0]) - B[0]@G[0] - np.transpose(G[0])@np.transpose(B[0])))
eigLMI = LA.eigvals(A[0]@X+X@np.transpose(A[0]) - B[0]@G[0] - np.transpose(G[0])@np.transpose(B[0])) ## Em nump
y, A*B significa a multiplicação indexidada dos elementos ## O operado
r @ realiza a multiplicação matricial correta.
print("\nOs Autovalores de A2*X+X*A2.H-B2*G-G.H*B2.H são:") #errado mudou a restrição
print(LA.eigvals(A[1]@X+X@np.transpose(A[1]) - B[1]@G[1] - np.transpose(G[1])@np.transpose(B[1])))
eigLMI = LA.eigvals(A[1]@X+X@np.transpose(A[1]) - B[1]@G[1] - np.transpose(G[1])@np.transpose(B[1])) ## Em nump
y, A*B significa a multiplicação indexidada dos elementos ## O operado
r @ realiza a multiplicação matricial correta.
if eigX.__gt__(0).all():
    if eigLMI.__lt__(0).all():
        print("\nSistema Factível. \nÉ possível controlar.")
else: print("\nSistema Infactível")
## Controlador
K = [Constant("K[{}]").format(i),range(i, i + 2 ), (1,2)] for i in range(2)]
K[0] = G[0]@(LA.inv(X))
print("\nControlador K[0]: \n', K[0])
K[1] = G[1]@(LA.inv(X))
print("\nControlador K[1]: \n', K[1])
print("\nAutorvalor de A1-B1*K:\n', LA.eigvals(A[0]-B[0]@K))
print("\nAutorvalor de A2-B2*K:\n', LA.eigvals(A[1]-B[1]@K))
print("\n O valor de U é:")
print(U)

```

```

mi=np.sqrt(U)
print("\n O valor de mi é:")
print(mi)
#-----#
# objective value #
#-----#
#print('\nthe optimal value of this problem is:')
#print(lmi.obj_value())          #"print objective" would also work, because objective is valued
toc ()

```

Este sem dúvidas foi o projeto mais complexo para converter do MATLAB® para o Python, sendo que, estavam envolvidos todos os projetos anteriores em um único projeto, logo foi necessário utilizar todas as técnicas desenvolvidas anteriormente para criar este projeto.

Porém o fato de ter realizado os projetos anteriormente, começando do controlador mais básico e ir adicionando pequenas alterações, possibilitou chegar nesta etapa e concluir este *script* com sucesso.

Após realizar a estruturação do *script* e conseguir transcrever todo o equacionamento que é feito no MATLAB® para o Python, podemos executar o projeto e obter o seguinte resultado:

```

type:    Complex Feasibility Problem
status:  unsolved
status:  optimal
Matriz Ótima P:
[[ 9.36202805e-03 -5.56293350e-02 -5.17131849e-03  2.12586475e-02]
 [-5.56293350e-02  1.59689033e+00  2.87016029e-02 -2.04287001e+00]
 [-5.17131849e-03  2.87016029e-02  5.22053659e-03 -3.81940888e-02]
 [ 2.12586475e-02 -2.04287001e+00 -3.81940888e-02  1.62037297e+01]]

Os Autovalores de X são:
[1.64842150e+01  1.31906108e+00  1.60736960e-03  1.03190844e-02]

Os Autovalores de A1*X+X*A1.H-B1*G-G.H*B1.H são:
[-1.74058414e+02 -2.10533887e-01 -1.83780333e-02 -1.78819554e+01]

Os Autovalores de A2*X+X*A2.H-B2*G-G.H*B2.H são:
[-2.47543429e+02 -1.62562455e+01 -1.72386651e-02 -1.39290200e-01]

Sistema Factível.
É possível controlar.

Controlador K[0]:
[[-180.75764579   24.08366848  -58.09646643    4.23978154]]

Controlador K[1]:
[[ 77.9630852   35.04386706 -58.08164296   2.9463706 ]]

Autorvalor de A1-B1*K:
[[-7.15469319+56.81638596j -7.15469319-56.81638596j
 -7.82891037+17.76404834j -7.82891037-17.76404834j]

```

```
[-9.81499612+58.32212631j -9.81499612-58.32212631j
 -9.58170417+19.71273657j -9.58170417-19.71273657j]]
```

Autorvalor de  $A2-B2*K$ :

```
[[-5.86930439+57.16815991j -5.86930439-57.16815991j
 -4.70645147+14.14465562j -4.70645147-14.14465562j]
 [-7.69432252+59.20690826j -7.69432252-59.20690826j
 -5.76491403+15.70714179j -5.76491403-15.70714179j]]
```

O valor de  $U$  é:

```
1468.6153351341256
```

O valor de  $\mu$  é:

```
38.3225173381672
```

Elapsed time: 2.303605 seconds.

Novamente o primeiro objetivo foi alcançado Python conseguiu solucionar as LMIs propostas, determinar dois ganhos distintos para o controlador  $K[0]$  e  $K[1]$ , provando novamente ser uma opção para a solução de LMIs.

Notamos também que os autovalores estão enquadrados no lado esquerdo do plano complexo formado por pares complexo conjugados como esperado.

Outra constatação foi que a redução no máximo sinal de controle que será desenvolvido pelo controlador  $\mu$  obtida via MATLAB® foi de 7,4927, enquanto, via Python foi de 38,3225, observamos que a vantagem vai para o MATLAB® que conseguiu encontrar o menor valor de  $\mu$  sendo assim o controlador projetado via MATLAB® exigiria menos esforços do controlador.

Este projeto foi um primeiro trabalho abordando esta metodologia de resolução de LMIs utilizando o *software* livre Python e neste trabalho foi empregado apenas um *solver*, logo, levando isto em consideração podemos assumir que para um primeiro estudo Python se saiu muito bem no seu primeiro teste e deixou boas impressões que podem incentivar novos projetos que verifiquem a passibilidade de utilizar novos *solver* e obter indicies melhores que os apresentados pelo MATLAB®.

Em contrapartida Python atingiu novamente o menor tempo de execução do *script* para este caso em questão gastou 2,3036 segundo para executar todo o *script* enquanto o MATLAB® levou 3,5366 segundos, novamente está é uma vantagem significativa, visto que, em casos de projetos com elevados tempos de processamento, seria possível reduzir significativamente este tempo apenas utilizando Python.

## 4.11 Simulações

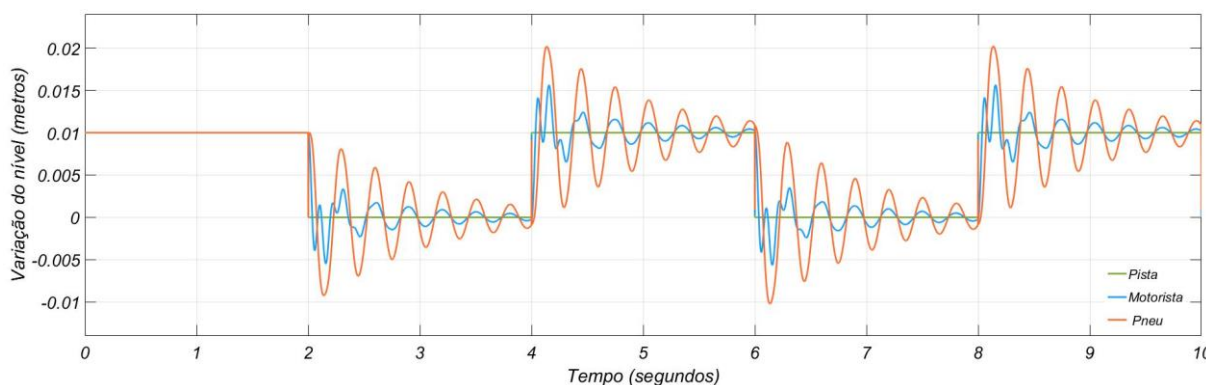
Após ter realizado a conversão dos projetos de controladores do MATLAB® para Python utilizando a teoria de Lyapunov onde o problema é descrito por meio de LMIs, e verificar que Python foi capaz de encontrar uma solução para o problema e determinar um ganho inclusive para projeto de controladores chaveados, fizemos algumas simulações para verificar a resposta do controlador projetado por cada um dos *softwares* em análise.

### 4.11.1 Primeiro controlador único ganho $K$ , $\gamma = 7$

Utilizando o Simulink® do MATLAB®, foi montado o projeto para simular a planta da suspensão ativa de bancada QUANSER® e a partir deste modelo simular o sistema em malha aberta, e na sequência com o controlador obtido via MATLAB® e via Python.

Sendo assim o primeiro gráfico é referente ao sistema em malha aberta, ou seja, sem nenhum controlador atuando.

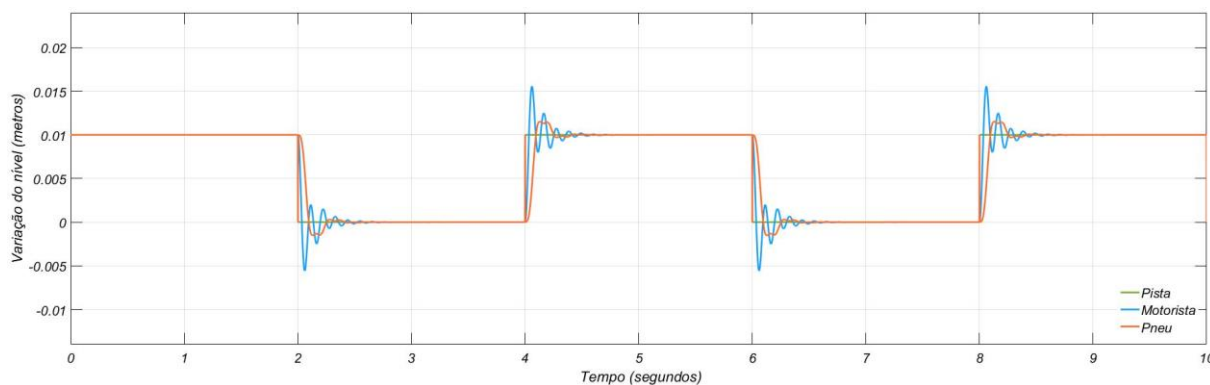
Figura 40 – Resultado da saída do sistema em malha aberta.



Fonte: próprio autor.

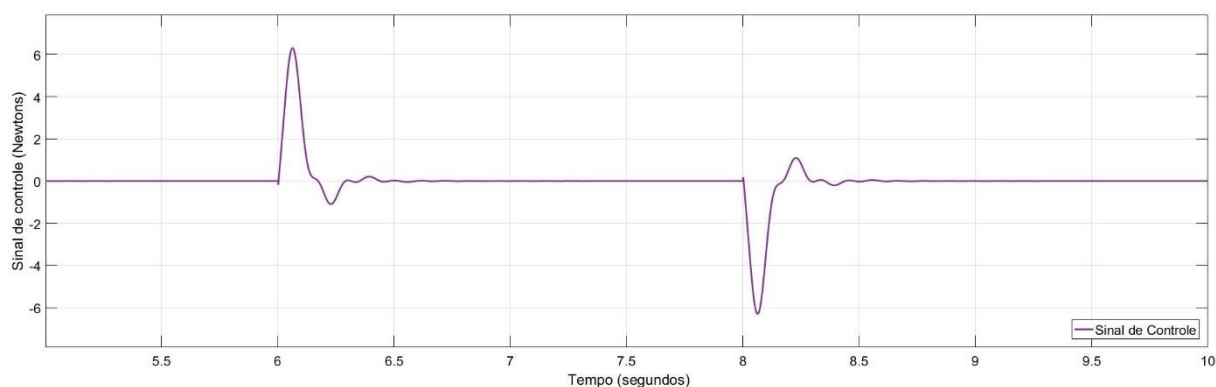
O próximo gráfico é referente a aplicação do controlador com único ganho projetado utilizando o MATLAB®.

Figura 41 – Primeiro controlador único ganho MATLAB®.



Fonte: próprio autor.

Figura 42 – Sinal de controle controlador único ganho MATLAB®.

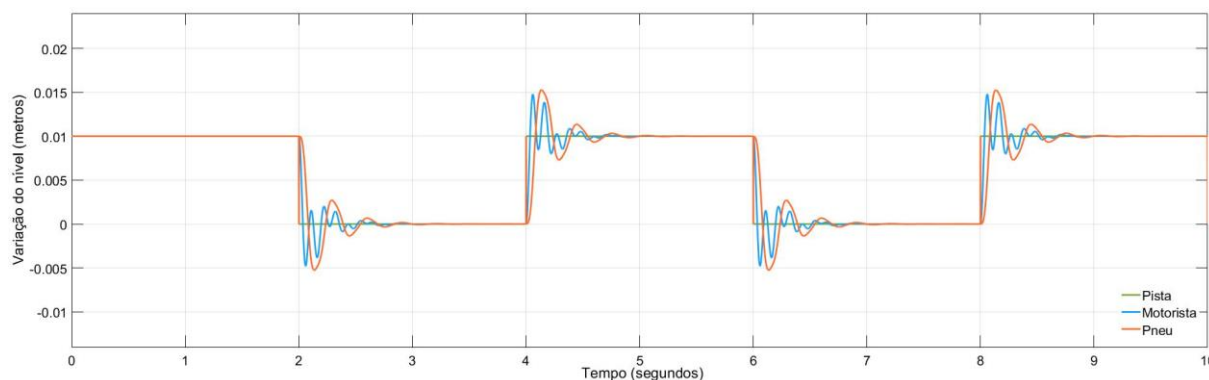


Fonte: próprio autor.

Analisando a Figura 41 podemos verificar que o controlador com único ganho foi capaz de estabilizar rapidamente as oscilações que chegam no motorista do veículo, logo, o controlador atingiu sua primeira função, estabilizar a planta em questão e melhorar o desempenho reduzindo as oscilações transferidas ao motorista. Já na Figura 42 está apresentado o sinal de controle gerado durante a simulação com o ganho  $K$  do controlador com único ganho via MATLAB®.

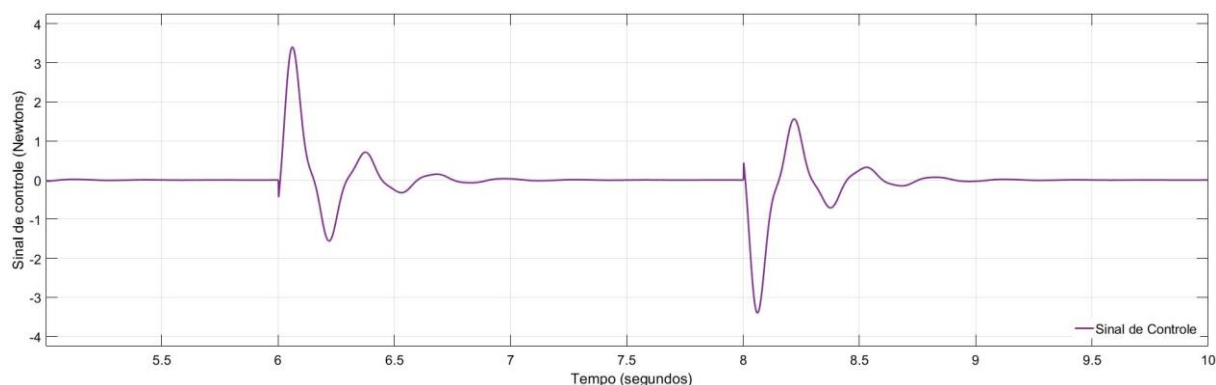
Na sequência foi realizado o mesmo procedimento, entretanto, agora foi utilizado o controlador projetado utilizando o *software* Python.

Figura 43 – Primeiro controlador único ganho Python.



Fonte: próprio autor.

Figura 44 – Sinal de controle controlador único ganho Python.



Fonte: próprio autor.

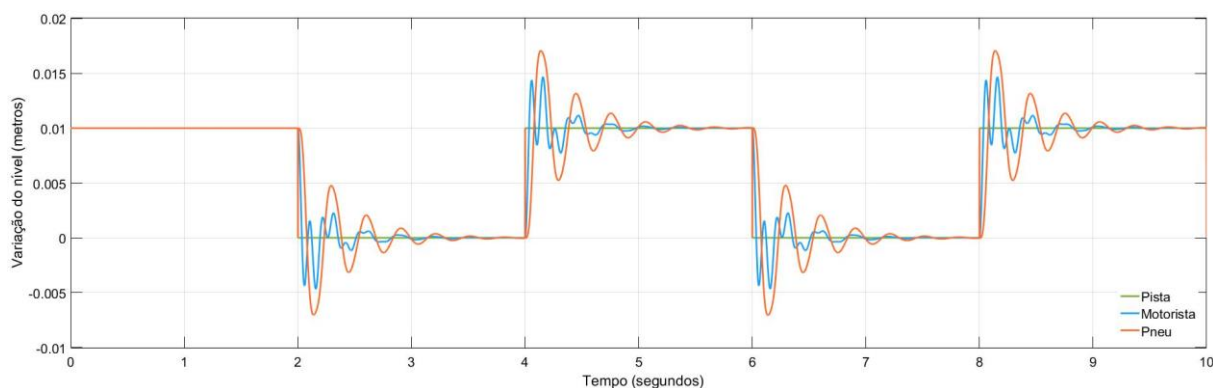
Quando observamos a Figura 43 podemos notar que o controlador obtido utilizando Python também foi capaz de atingir o primeiro objetivo que era estabilizar o sistema da suspensão ativa de bancada, e também conseguiu suavizar as oscilações sentidas pelo motorista, sendo assim, o *software* mostrou ser capaz de resolver os problemas das LMIs e encontrar um controlador que solucione o problema como observado. Na Figura 44 está apresentado o sinal de controle gerado para o controlador  $K$  obtido utilizando a plataforma Python.



#### 4.11.2 Segundo controlador único ganho considerando incerteza na massa, $\gamma = 7$

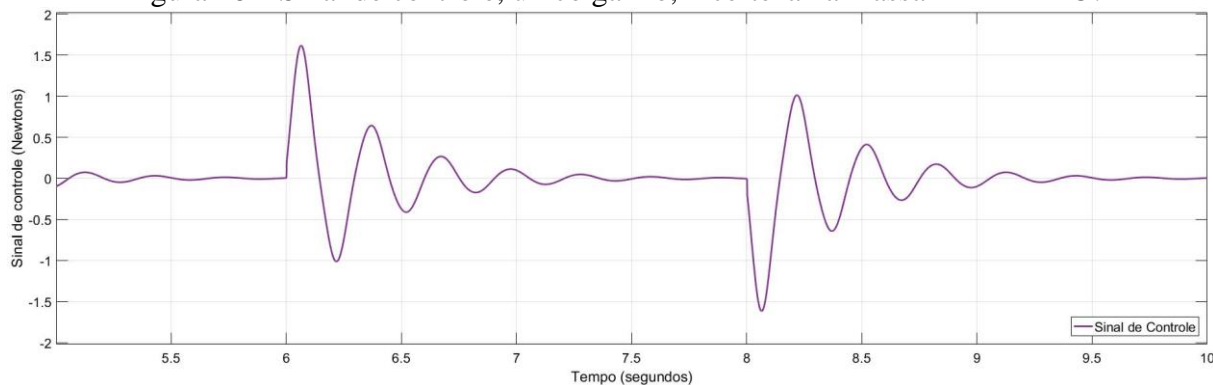
Agora vamos analisar o controlador projetado pelos dois *softwares*, consideramos novamente que o sinal de malha aberta é dado pela Figura 40. Vamos primeiro ao resultado do controlador apresentado pelo *software* MATLAB®.

Figura 45 – Segundo controlador único ganho, incerteza na massa MATLAB®.



Fonte: próprio autor.

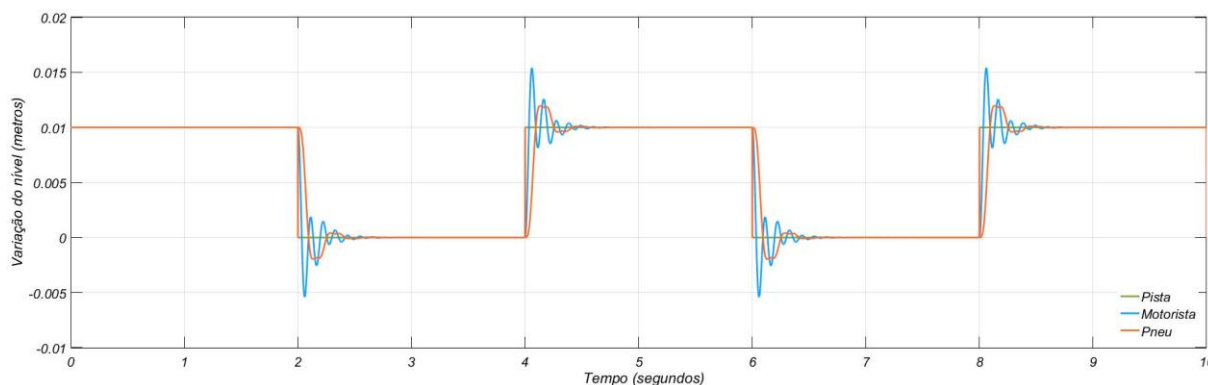
Figura 46 – Sinal de controle, único ganho, incerteza na massa MATLAB®.



Fonte: próprio autor.

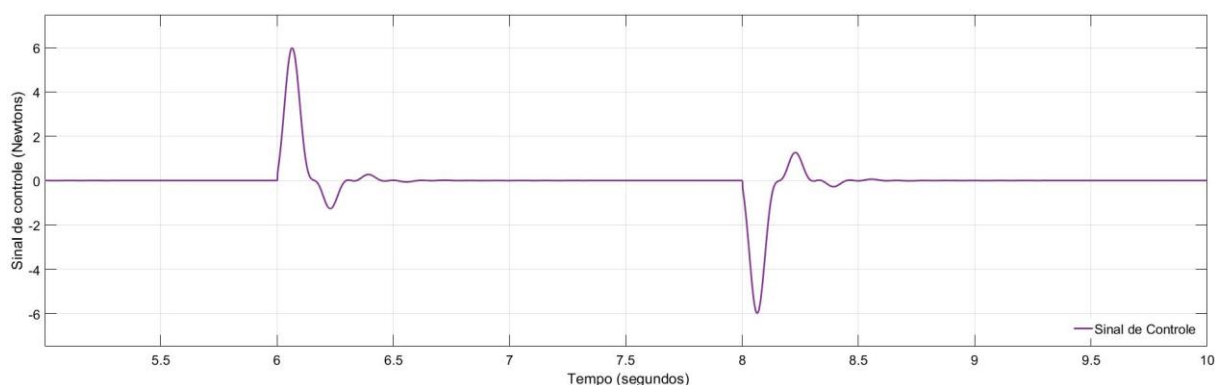
Seguindo com a análise podemos observar que novamente foi possível encontrar um controlador capaz de estabilizar o sistema e suavizar as oscilações que são passadas para o motorista do veículo.

Figura 47 – Segundo controlador único ganho com incerteza na massa Python.



Fonte: próprio autor.

Figura 48 – Sinal de controle, único ganho com incerteza na massa Python.



Fonte: próprio autor.

Através da Figura 47 podemos notar que novamente Python foi capaz de solucionar o problema das LMIs, comparando o resultado obtido com Python podemos observar que o controlador encontrado utilizando Python conseguiu estabilizar o sistema com aproximadamente 0,5 segundos enquanto o controlador projetado via MATLAB® levou quase 1,5 segundos para estabilizar o sistema como observado na Figura 45. Lembrando também que Python levou vantagem por executar o *script* em menor tempo quando comparado ao MATLAB®.

São apresentados também os sinais de controle para ambos os casos, na Figura 46 está exposto o sinal de controle obtido utilizando o ganho do controlador  $K$  para o segundo

controlador com restrição na massa do veículo utilizando o MATLAB® e na Figura 48 é apresentado o sinal de controle utilizado quando utilizado o ganho  $K$  obtido via Python.

## 5 CONSIDERAÇÕES FINAIS

Neste trabalho, apresentou-se uma alternativa para a utilização do *software* proprietário, no caso o MATLAB®, buscando validar um segundo *software* de código aberto, neste caso Python, no projeto de controladores, deixando mais uma opção de plataforma para utilização dos alunos que desejam projetar controladores utilizando a teoria de controle.

Diante disso, a pesquisa se deu em torno do estudo da plataforma livre visando solucionar projeto de controladores. O primeiro passo foi validar o uso da plataforma na teoria de controle clássico, e sem dúvida Python se mostrou capaz de calcular os parâmetros necessários durante as especificações de projetos e na plotagem do gráfico do *Root Locus*, sendo possível criar um *script* genérico onde o usuário entra com as informações da planta e os indices de desempenho do projeto e tem como retorno todos os parâmetros calculados e o gráfico plotado.

Sendo assim o primeiro teste em Python obteve um excelente êxito, visto que, todos os gráficos plotados foram de encontro com os resultados teóricos apresentados anteriormente, sendo assim, esta primeira validação foi realizada com êxito.

Seguindo com as intenções de utilização de Python como uma alternativa ao uso do MATLAB®, verificou-se a possibilidade de projetar controladores chaveados, utilizando a teoria de espaço de estados e a análise da estabilidade segundo Lyapunov, no qual os controladores são projetados com base em desigualdades matriciais lineares – LMIs.

Nesta segunda etapa o *software* em análise também não decepcionou, foi possível criar um *script* capaz de solucionar problemas envolvendo LMIs e encontrar um ganho capaz de estabilizar a planta da suspensão ativa de bancada.

Quando analisamos os resultados dos primeiros projetos realizados no MATLAB® com os primeiros projetos realizados em Python, notamos que Python foi capaz de solucionar as LMIs encontrando um controlador  $K$  e em todos os testes o tempo de execução do *script* em Python foi mais rápido quando comparado com o MATLAB®, este fato aponta uma vantagem na utilização da plataforma quando se tem um projeto que demanda muito tempo de processamento do *script* podendo reduzir significativamente o tempo de processamento para estes projetos.

Python só não conseguiu acompanhar o MATLAB® quando adicionamos a restrição no máximo sinal de controle, para este índice de desempenho o MATLAB® obteve um valor menor do que encontramos ao utilizar Python, porém nada que inviabilize o uso da plataforma.

Os próximos passo para este projeto seria a utilização de outros *solvers* disponíveis em Python para verificar se alcançamos valores de otimização próximos aos valores obtidos no MATLAB® o que tornaria Python totalmente equivalente ao *software* proprietário.

Um segundo passo seria fazer a implementação prática dos controladores obtidos utilizando Python no módulo didático da suspensão ativa de bancada fabricada pela QUASER® e verificar como se comporta cada um dos controladores projetados, aumentando assim a confiança e a validade no uso da plataforma.

Como possíveis trabalhos futuros temos a possibilidade de fazer um aplicativo onde o usuário interagem com o projeto, ou seja, pegar o *script* existente, criar uma interface gráfica, por exemplo diagrama de blocos onde o usuário insere os dados do projeto diretamente no diagrama de blocos e o *software* apresenta todos os parâmetros do projeto e também o gráfico do *Root Locus* tornando o mesmo mais atrativo para o usuário final podendo até incentivar os alunos com relação a utilização da programação em Python para o seu dia a dia na resolução de problemas.

## REFERÊNCIAS

- ÅSTRÖM, Karl Johan; HÄGGLUND, Tore. New tuning methods for PID controllers. In: **European Control Conference**, 1995. 1995.
- BAZANELLA, Alexandre Sanfelice; DA SILVA JUNIOR, João Manoel Gomes. **Sistemas de controle: Princípios e métodos de projeto**. UFRGS, 2005.
- BORGES, Luiz Eduardo. **Python para desenvolvedores: aborda Python 3.3**. Novatec Editora, 2014.
- BOYD, S., EL GHAOUI, L., FERON, E., & BALAKRISHNAN, V. (1994). Desigualdades matriciais lineares em teoria de sistemas e controle Sociedade de Matemática Industrial e Aplicada.
- BUZETTI, Ariel Starcke. **Projeto de controle robusto chaveado com falhas nos sensores**. 2017.
- DA SILVA, Rogério Oliveira; SILVA, Igor Rodrigues Sousa. Linguagem de Programação Python. **TECNOLOGIAS EM PROJEÇÃO**, v. 10, n. 1, p. 55-71, 2019.
- DE MATTOS, João Roberto Loureiro; DOS SANTOS GUIMARÃES, Leonam. **Gestão da tecnologia e inovação: uma abordagem prática**. Saraiva, 2005.
- DORF, R. C.; BISHOP, R. H. **Sistemas de Controle Moderno**; 8ª. Ed, LTC, Rio de Janeiro, 2001.
- D. R. de Oliveira, G. R. dos Santos, M. C. Teixeira, E. Assunção, R. Cardim, and U. N. L. Alves. **On switched control of discrete-time takagi-sugeno fuzzy systems with unknown membership functions**. In 2018 IEEE International Conference on Fuzzy Systems (FUZZIEEE), pages 1–8. IEEE, 2018.
- E. Assunção; M. C. Teixeira. **CONTROLE LINEAR Sistemas Contínuos e Discretos no Tempo**. 2018. 261 p. Vol 1.
- GAHINET, P., NEMIROVSKII, A., LAUB, AJ, & CHILALI, M. (1994, Dezembro). **A caixa de ferramentas de controle LMI**. In Proceedings of 1994 33rd IEEE Conference on Decision and Control (Vol. 3, pp. 2038-2041). IEEE.
- LÖFBERG, J. YALMIP: **A toolbox for modeling and optimization in MATLAB**. In: TAIPEI, TAIWAN. Proceedings of the CACSD Conference. [S.l.], 2004. v. 3.
- M. A. Beteto, E. Assunção, M. C. Teixeira, E. R. Silva, L. F. Buzachero, and R. P. Caun. **New design of robust lqr-state derivative controllers via lmis**. **IFAC-PapersOnLine**, 51(25): 422–427, 2018.
- OGATA, Katsuhiko. **Modern control engineering**. Prentice hall, 2010.

DE OLIVEIRA, DR, TEIXEIRA, MCM, ALVES, UNLT, DE SOUZA, WA, ASSUNÇÃO, E., & CARDIM, R. (2018). No projeto do controlador local comutado  $H_\infty$  para sistemas fuzzy T-S incertos sujeitos à saturação do atuador com funções de pertinência desconhecidas. **Conjuntos e Sistemas Fuzzy**, 344 , 1-26.

PETRUCCELLI, Ana Clara Faleiro et al. Vantagens e desvantagens do uso do software livre no mundo acadêmico e profissional. In: **Anais do Congresso Nacional Universidade, EAD e Software Livre 2010**.

QUANSER INNOVATE EDUCATE. **Active suspension LQG control using Quarc:** Instructor manual. [S.1]:[S.n], 2010. 48 p.

QUANSER INNOVATE EDUCATE. **Active suspension system:** User manual. [S.1]:[S.n], 2010. 23 p.

SAGNOL, Guillaume; STAHLBERG, Maximilian. **PICOS:** PICOS is a user friendly Python API to several conic and integer programming solvers. In: **PICOS : PICOS is a user friendly Python API to several conic and integer programming solvers.**, [S. 1.], 22 set. 2021. Disponível em: <https://picos-api.gitlab.io/picos/introduction.html#>. Acesso em: 20 out. 2021.

SOUZA, W. A. d. **Projeto de controladores robustos chaveados para sistemas não lineares descritos por modelos fuzzy Takagi-Sugeno**. Universidade Estadual Paulista (UNESP), 2013.

TABOADA, Carlos. **Gestão de tecnologia e inovação na logística**. IESDE BRASIL SA, 2009.

TANAKA, Kazuo; WANG, Hua O. **Projeto e análise de sistemas de controle fuzzy**. John Wiley & Sons Ltda, 2001.