UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA DE QUÍMICA

Giovanna Musco Twardowski Pinto

CONTROLE PID COM GANHO PROGRAMADO DO REATOR DE VAN DE VUSSE

RIO DE JANEIRO
2023

Giovanna Musco Twardowski Pinto

CONTROLE PID COM GANHO PROGRAMADO DO REATOR DE VAN DE VUSSE

Trabalho de Conclusão de Curso apresentado à Escola de Química da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do grau de Engenheiro de Alimentos.

Orientador: Bruno Didier Olivier Capron

Rio de Janeiro

2023

CIP - Catalogação na Publicação

M659c

Musco Twardowski Pinto, Giovanna
CONTROLE PID COM GANHO PROGRAMADO DO REATOR DE VAN DE VUSSE / Giovanna Musco Twardowski Pinto. -- Rio de Janeiro, 2023.
121 f.

Orientador: Bruno Didier Olivier Capron.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Escola de Química, Bacharel em Engenharia de Alimentos, 2023.

1. projeto de plantas ou equipamentos industriais. 2. controle e instrumentação de proc. e equip. industriais. 3. estudo de modelagem e simulação. I. Didier Olivier Capron, Bruno, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

Giovanna Musco Twardowski Pinto

Controle PID com ganho programado do Reator de Van de Vusse

Trabalho de Conclusão de Curso apresentado à Escola de Química da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do grau de Engenheiro de Alimentos.

Aprovado em 28 de junho de 2023.

_____

Bruno Didier Olivier Capron, D.Sc., UFRJ

_____

Kese Pontes Freitas Alberton, D.Sc., UFRJ

_____

Ricardo Schmitz Ongaratto, D.Sc., UFRJ

Rio de Janeiro
2023

RESUMO

PINTO, Giovanna.

Controle PID com ganho programado do Reator de Van de Vusse. Rio de Janeiro, 2023. Trabalho de Conclusão de Curso (Graduação em Engenharia de Alimentos) - Escola de Química, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2023.

Na indústria química, além dos processos serem dinâmicos, muitos possuem dinâmica não linear. Além disso, um mesmo processo pode ser operado em diferentes pontos, seja quando há mudança de produtos, ou quando se busca a otimização em um ponto de operação economicamente ótimo. Assim, o controle tradicional, cujo controlador possui parâmetros fixos, nem sempre é eficiente para todos os pontos de operação possíveis. Neste trabalho, estudou-se a aplicação do controle adaptativo ao reator de Van de Vusse, que é um *benchmark* para design e controle de processos complexos. Para isto, primeiro um controlador PI foi desenvolvido, em torno de um ponto de referência, a partir do método de síntese direta. Utilizando a sintonia obtida como base, aplicou-se um método para adaptação da sintonia em função do ponto de operação, combinando duas metodologias: a sintonia automática e o Gain Scheduling. A ideia foi obter um controlador que fosse capaz de se adaptar a diferentes pontos de operação e distúrbios possíveis nesse processo. Simulações foram feitas em Python para avaliar o controlador obtido, que se mostrou eficiente para diferentes pontos de operação, apesar de apresentar respostas lentas e oscilatórias em pontos extremos de duas variáveis agendadas estudadas. Além disso, verificou-se também que o controle adaptativo desenvolvido foi mais eficiente do que o controlador fixo obtido na primeira etapa, ao redor de um ponto de referência.

Palavras-chave: Controlador PID. *Gain Scheduling*. Síntese Direta. Reator de Van de Vusse.

ABSTRACT

PINTO, Giovanna.

***Gain Scheduling* in PID Control of Van de Vusse Reactor**. Rio de Janeiro, 2023. Trabalho de Conclusão de Curso (Graduação em Engenharia de Alimentos) - Escola de Química, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2023.

The processes in the chemical industry are dynamic and they have non-linear dynamics. In addition, the same process can be operated at different points, when different products are used, or when an optimization is needed, seeking the optimal economic point. Thus, the traditional control, whose controller has fixed parameters, is not always efficient for all the possible operating points. This work studied the application of adaptative control in the Van de Vusse reactor, which is a benchmark for the design and control of complex processes. To achieve the objective of this work, the first step was to develop a PI controller, around a reference point, from the direct synthesis method. The automatic tuning method and the Gain Scheduling were used to adapt this tuning as a function of the operating point. The objective was to develop a controller that would be able to adapt to different points and possible disturbances in the process. Simulations were made in Python to evaluate the obtained controller, which was efficient for different operating points, despite presenting slow and oscillatory responses at extreme points of two scheduling variables. In addition, it was also verified that the developed adaptative control was more efficient than the traditional controller obtained in the first step of the study, around an initial point.

Keywords: PID Control. Gain Scheduling. Direct Synthesis Method. Van de Vusse Reactor.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CSTR	Reator de Tanque Agitado Contínuo
VC	Variáveis de Controle
VM	Variáveis Manipuladas
VD	Variáveis de Distúrbio
PID	Proporcional Integral Derivativo
PI	Proporcional Integral
P	Proporcional
DS	Síntese Direta

LISTA DE SÍMBOLOS

Y	Variável Controlada
U	Variável Manipulada
D	Variável Distúrbio
P	Saída do controlador
E	Sinal de erro
Y_m	Variável Y medida
Y_{sp}	*Set point*
$\tilde{Y}_{sp}$	*Set point* interno usado pelo controlador
Y_u	Mudança no Y por conta de U
Y_d	Mudança em Y por conta de D
G_c	Função de transferência do controlador
G_v	Função de transferência do elemento final de controle ou da válvula
G_p	Função de transferência do processo
G_d	Função de transferência do distúrbio
G_m	Função de transferência para o sensor e transmissor
K_m	Ganho do estado estacionário para G_m
$e(t)$	Função erro
$y_{SP}(t)$	*Set point* em sinal padrão
$y_m(t)$	Valor medido da variável de controle
$p(t)$	Saída do controlador
$\bar{p}$	Valor no estado estacionário
K_C	Ganho do controlador
τ_I	Tempo Integral
τ_D	Tempo Derivativo
K_I	Ganho Integral
K_D	Ganho Derivativo
τ_m	Constante de tempo do medidor
τ_v	Constante de tempo do elemento final de controle ou da válvula
K_v	Ganho do elemento final de controle ou da válvula
K_p	Ganho do processo

τ_p	Constante de tempo do processo
τ_c	Constante de tempo do controlador
θ	Representação do atraso no tempo ou tempo morto
τ_{dom}	Maior constante de tempo do processo
Δt	Tempo de amostragem
x	Vetor de estado
u	Vetor de entrada das variáveis manipuladas
d	Vetor de distúrbio
y	Vetor de saída das variáveis controladas medidas
$\frac{dx}{dt}$	Vetor da derivada no tempo de x
$Y(s)$	Vetor coluna da transformada de Laplace de $y(t)$
B	Ciclopentenol
A	Ciclopentadieno
D	Diciclopentadieno
C	Ciclopentanediol
c	Concentração molar (mol/L)
C_p	Capacidade calorífica $(kJ/(kg \cdot K)$
F	Vazão (L/h)
k_i	Razão de coeficiente
t	Tempo (h)
V_R	Volume do reator (L)
T	Temperatura $(°C)$
k_w	Coeficiente de transferência de calor $(kJ/(m^2 \cdot h \cdot K)$
A_R	Superfície do reator (m^2)
$\dot{Q_K}$	Taxa de refrigeração (kJ/h)
ΔH_R	Entalpia da reação (kJ/mol)
m_K	Massa do refrigerador (kg)
ρ	Densidade (kg/L)
K	Camisa do reator
0	Valor da corrente de entrada
E_i	Energia de Ativação

SUMÁRIO

1 INTRODUÇÃO

Na indústria química, os processos são dinâmicos e tipicamente apresentam comportamento dinâmico não linear, ou seja, suas características podem mudar com o tempo ou com as condições operacionais (SMITH; CORRIPIO, 1997). Além disso, um mesmo processo pode ser multipropósito, o que quer dizer que pode ser usado para gerar diferentes produtos. O ponto de operação também pode ser modificado visando otimizar o lucro da planta.

O principal objetivo do controle de processos é manter um processo nas suas condições operacionais desejadas, visando a segurança, eficiência, adequação às leis ambientais e lucratividade da planta, independente dos distúrbios existentes (SEBORG et al., 2011).

Por conta da dinâmica dos processos e dos diferentes pontos de operações possíveis, é claro que um controlador com uma única parametrização pode não funcionar bem em todas as condições existentes. Para contornar este problema, surgiu o controle adaptativo, que consiste em uma metodologia em que o comportamento do controlador pode ser modificado a fim de se adaptar à dinâmica do processo e aos distúrbios existentes (ÅSTRÖM; HÄGGLUND, 1995).

Existem dois métodos de controle adaptativo: o direto, para o qual os parâmetros do controlador são ajustados a partir de dados obtidos pelo processo em malha fechada; e o indireto, para o qual um modelo do processo é usado para parametrizar os controladores nos diferentes pontos de operação possíveis (ÅSTRÖM; HÄGGLUND, 1995).

O mecanismo de reação do reator de Van de Vusse em um CSTR se baseia na produção de ciclopentenol (B) a partir do ciclopentadieno (A). Os subprodutos desta reação são o diciclopentadieno (D) e o ciclopentanediol (C) (VAN DE VUSSE, 1964). Por se tratar de um sistema não linear, é um *benchmark* para design e controle de processos complexos, além de ser interessante para o estudo do controle adaptativo (KLATT; ENGELL, 1998).

O objetivo principal deste trabalho foi aplicar a metodologia do controle adaptativo para o reator de Van de Vusse, que possui uma dinâmica altamente não linear. A meta foi desenvolver um controlador com sintonia automática para todos os pontos de operação possíveis, considerando duas malhas de controle, ou seja, duas variáveis controladas. Para isto, combinou-se dois métodos do controle adaptativo: o método indireto de *Gain Scheduling* com o auxílio do método direto da sintonia automática. O *Gain Scheduling* é ideal para processos não lineares e a sintonia automática facilita muito o seu uso.

O trabalho é relevante por estudar um controle adaptativo para um caso pertinente à indústria de processos. A metodologia estudada pode ser usada para este ou outros processos

complexos, permitindo que um mesmo controlador seja capaz de se adaptar a diferentes pontos de operação da planta.

Para alcançar o objetivo principal, alguns objetivos secundários foram necessários. O primeiro dele foi desenvolver o controle para o Reator de Van de Vusse em torno de um ponto de referência. Para isto, as duas malhas de controle foram definidas, as funções de transferência dos componentes dessas malhas foram encontradas, com o auxílio de simulações em Python e com o uso do método da Síntese Direta. Depois disso, os controladores com parâmetros fixos encontrados foram avaliados, de forma individual e conjunta, a partir das respostas simuladas das malhas a distúrbios.

O segundo objetivo secundário foi projetar os controladores em torno de um ponto de operação genérico, a partir da linearização dos balanços do Reator de Van de Vusse e do cálculo da matriz de funções de transferência. Depois disso, um código em Python foi desenvolvido para encontrar a função de transferência do processo e do controlador para qualquer ponto de operação dentro das faixas das variáveis agendadas definidas para o *Gain Scheduling*. Alguns pontos de operação foram selecionados de forma aleatória e os controladores encontrados foram testados para dois distúrbios diferentes. Além disso, o controlador adaptativo obtido foi comparado com o controlador com parâmetros fixos obtidos através da Síntese Direta na primeira etapa da metodologia. Como resultado, o controlador adaptativo representou melhor desempenho que o controlador fixo.

O texto está organizado conforme descrito a seguir. Inicialmente, apresenta-se no Capítulo II uma revisão bibliográfica sobre os assuntos necessários para este trabalho. Primeiramente, apresenta-se o controle de processos, aborda-se sobre o controle feedback em malha fechada e, em seguida, sobre os tipos de controladores. Em seguida, apresentam-se as funções de transferência, a síntese direta e a teoria sobre a transformada Z e o controle digital. Descreve-se então os modelos em espaço de estados e matriz de função de transferência e, por fim, o controle adaptativo.

No Capítulo III, a metodologia do trabalho é apresentada, descrita em duas etapas: primeiro em função de um ponto de referência e depois em torno de um ponto genérico para enfim chegar ao controle adaptativo.

No Capítulo IV encontram-se os resultados obtidos, além de uma comparação e discussão sobre o que foi encontrado.

Por fim, no Capítulo V está a conclusão do trabalho, junto com sugestões para trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

2.1 CONTROLE DE PROCESSOS

A implementação de novas tecnologias nas plantas industriais cria, ao mesmo tempo, novos desafios e oportunidades para alcançar a boa performance nas plantas industriais. O aumento da competição, a preocupação com o meio ambiente e a mudança constante das condições econômicas são fatores que também influenciam nesse objetivo. Tornou-se impossível operar plantas modernas de forma segura e rentável, garantindo a qualidade e o cumprimento das leis ambientais sem a aplicação de sistemas computadorizados de controle de processos (SEBORG et al., 2011).

O principal objetivo do controle de processos é manter um processo nas suas condições operacionais desejadas, de forma segura e eficiente, obtendo os requisitos de qualidade do produto e ambientais (SEBORG et al., 2011). Ou seja, significa manter as variáveis do processo, como temperaturas, pressões, fluxos e composições nos valores desejados, independente dos distúrbios existentes (SMITH; CORRIPIO, 1997).

De acordo com SEBORG *et al.* (2011), processo é a conversão dos reagentes presentes na corrente de entrada para produtos usando operações químicas e físicas. Na prática, o termo processo é usado tanto para os processos operacionais como para os equipamentos de processo. Os três tipos de processos mais comuns são: contínuo, batelada e semi-batelada.

Os processos são naturalmente dinâmicos, o que quer dizer que mudanças sempre ocorrem. Assim, ações são necessárias para manter as variáveis do processo nos valores de especificação de projeto, garantindo assim a segurança, eficiência energética e material e qualidade e quantidade do produto (SMITH; CORRIPIO, 1997).

As três principais variáveis do processo que devem ser identificadas para o controle são as variáveis de controle, as variáveis manipuladas e as variáveis de distúrbio.

As variáveis de controle (VC) são aquelas que de fato são controladas. O valor desejado para essa variável é usualmente chamado *set point*.

As variáveis manipuladas (VM) são as que serão ajustadas para manter as variáveis de controle próximas do seu *set point*. Geralmente, são vazões e rotações de motores.

As variáveis de distúrbio (VD) afetam as variáveis controladas, mas não podem ser manipuladas. Costumam estar relacionadas às mudanças no ambiente operacional do processo, como por exemplo as condições das correntes de entrada ou a temperatura ambiente (SEBORG et al., 2011).

A definição dessas variáveis é um passo crítico para o desenvolvimento do controle e deve ser feita baseada no objetivo do controle.

Existem três componentes básicos em todos os sistemas de controle. São eles: o sensor / transmissor, o controlador, que é o cérebro do sistema de controle, e o elemento final de controle, geralmente válvulas. As três ações básicas de um controle são a medição, a decisão e a ação (SMITH; CORRIPIO, 1997).

O controle de processos pode ser feito a partir de duas estratégias. O primeiro método é o controle por *feedback*, que se baseia na medição da variável de controle para ajustar a variável manipulada. Neste caso, a variável de distúrbio não é medida. A principal vantagem desta estratégia é que a ação corretiva será tomada independente de qual for a variável de distúrbio. Mas, a sua desvantagem é que nenhuma ação é tomada até que o distúrbio altere o processo e a variável de controle não está mais no seu *set point* (SEBORG et al., 2011).

O segundo método é a estratégia de controle *feedforward*. Neste caso, a variável de distúrbio é medida e não a variável de controle. A vantagem deste método é que a ação corretiva é tomada antes do processo ser alterado e esta ação vai evitar que a variável controlada se distancie do *set point*. Já suas desvantagens são: a variável de distúrbio precisa ser medida e não existirá ações corretivas para variáveis de distúrbio que não sejam medidas (SEBORG et al., 2011).

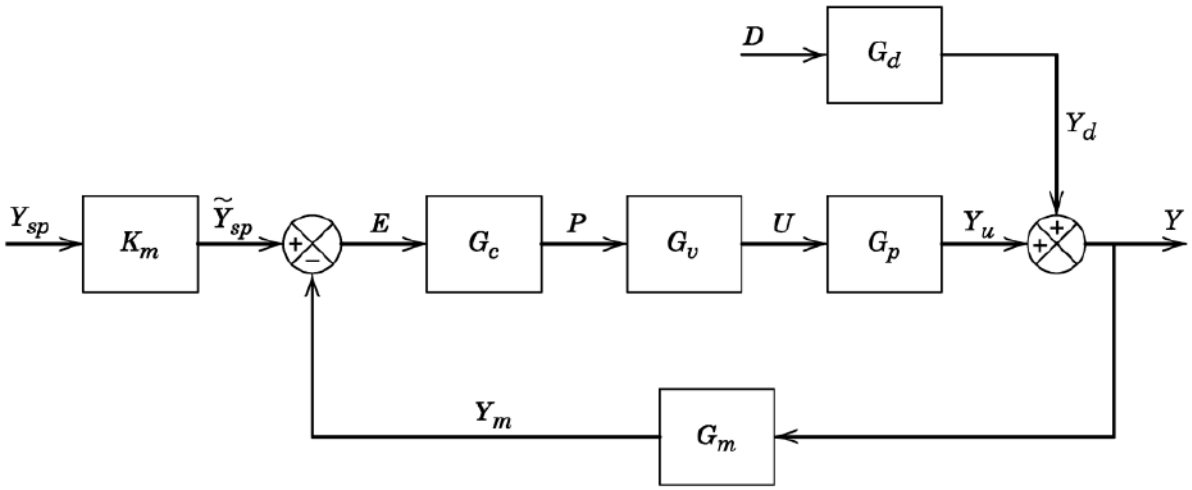
Uma estratégia que também é possível é a combinação das duas já citadas: sistema controle *feedforward-feedback*. Assim, o controle feedback corrigiria os distúrbios não medidos enquanto o *feedforward* evitaria alterações desnecessárias no processo (SEBORG et al., 2011). Na prática, a estratégia de controle feedback é sempre usada, enquanto a estratégia *feedforward* só é usada combinada com a primeira com o objetivo de compensar o efeito de distúrbios que impactam muito na variável controlada.

É importante ressaltar que as três ações do controle devem ocorrer em loop. Ou seja, em primeiro lugar, a medição deve ser feita. Essa informação deve ser transmitida para o controlador, na forma de um sinal padrão, que pode ser pneumático ou elétrico. O controlador é capaz de comparar essa variável medida com o valor do *set point* desejado e, a partir disso, tomar uma decisão. Por último, como consequência da decisão tomada pelo controlador, uma ação é tomada sobre o elemento final de controle, o que modifica o processo, alterando a variável controlada, que será medida mais uma vez. Desta forma, uma nova decisão será tomada, uma nova ação ocorrerá, alterando novamente o processo e assim por diante. Este tipo de controle é chamado de controle em malha fechada (SMITH; CORRIPIO, 1997).

2.2 PROCESSOS COM CONTROLE FEEDBACK EM MALHA FECHADA

A Figura 2.1 representa um diagrama de blocos geral representando um sistema de controle *feedback*. Este mesmo diagrama pode ser usado para representar muitos diagramas práticos de controle e pode ser adaptado como por exemplo pela adição de outros blocos quando existem também outros elementos no processo (SEBORG et al., 2011)

Figura 2.1 - Diagrama de blocos padrão de um sistema de controle por feedback.



Fonte: SEBORG et al., 2011

A notação usada na Figura 2.1 está representada na Figura 2.2 abaixo.

Figura 2.2 – Notação utilizada no diagrama de blocos.

Y	Variável Controlada	Y_u	Mudança no Y por conta de U
U	Variável Manipulada	Y_d	Mudança em Y por conta de D
D	Variável Distúrbio	G_c	Função de transferência do controlador
P	Saída do controlador	G_v	Função de transferência do elemento final de controle
E	Sinal de erro	G_p	Função de transferência do processo
Y_m	Variável Y medida	G_d	Função de transferência do distúrbio
Y_{sp}	Set point	G_m	Função de transferência para o sensor e transmissor
$\tilde{Y}_{sp}$	Set point interno usado pelo controlador	K_m	Ganho do estado estacionário para G_m

Fonte: SEBORG et al., 2011

Cada uma das variáveis descritas é a transformada de Laplace da variável de desvio da variável em questão em relação ao seu valor em regime estacionário. Já que o elemento final de controle geralmente é uma válvula, sua função de transferência se chama G_v. A função de transferência do processo G_p indica o efeito da variável manipulada na variável controlada. A função de transferência de distúrbio G_d representa o efeito do distúrbio na variável controlada (SEBORG et al., 2011).

O Y_{sp} e o D são sinais de entrada independentes do controle de processo porque eles não são afetados pelo controle. Para avaliar o controle, é preciso avaliar como o processo controlado reage a mudanças nessas entradas citadas (SEBORG et al., 2011).

2.3 TIPOS DE CONTROLADORES

O objetivo de um sistema de controle por feedback é garantir que o sistema em malha fechada tenha uma boa resposta, tanto para o estado estacionário quanto para o dinâmico. Para isso ocorrer, os seguintes critérios devem ser atendidos: o sistema deve ser estável; o efeito dos

distúrbios deve ser compensado; a resposta do controle deve fazer com que a variável controlada volte ao seu valor *set point* de forma rápida e suave e o sistema de controle deve ser robusto, ou seja, insensível às mudanças nas condições do processo e incertezas no modelo do processo. É difícil atingir todos esses critérios com um único controlador (SEBORG ET AL., 2011).

O algoritmo básico para o controle feedback é o PID, que é o mais usado na indústria e consiste na combinação de três modos: Proporcional, Integral, Derivativo. Essa junção faz com que o controlador seja flexível e possa ser usado em diferentes situações de controle (SMITH; CORRIPIO, 1997).

2.3.1 Modo Proprocional

No controle feedback, o objetivo é reduzir o erro a zero de acordo com o definido pela equação (2.1), onde $e(t)$ é o erro, $y_{SP}(t)$ é o set point (em sinal padrão) e $y_m(t)$ é o valor medido da variável de controle, o equivalente ao sinal proveniente do sensor / transmissor (SEBORG et al., 2011).

$$e(t) = y_{SP}(t) - y_m(t) \tag{2.1}$$

Para o controlador proporcional, a saída do controlador é proporcional ao sinal do erro, de acordo com a equação (2.2) onde $p(t)$ é a saída do controlador, $\bar{p}$ é o valor no estado estacionário e K_C é o ganho do controlador (SEBORG et al., 2011).

$$p(t) = \bar{p} + K_C e(t) \tag{2.2}$$

Os principais conceitos desse modo são: o ganho do controlador pode ser ajustado para fazer com que o a saída do controlador seja o quão sensível às variações se desejar ; o sinal do ganho também pode ser escolhido para fazer com que a saída do controlador aumente ou diminua conforme o erro crescer (SEBORG et al., 2011).

A principal desvantagem do modo puramente proporcional é a existência do erro residual do estado estacionário, também chamado de *offset*. Isto se vê quando ocorre uma mudança no valor do *set point*. Para controles onde *offsets* podem ser tolerados, este modo é interessante por causa da sua simplicidade (SEBORG et al., 2011). Além disso, o aumento do valor do ganho

proporcional garante uma resposta mais próxima do *set point*, ou seja diminui o *offset*. Mas, como consequência, a resposta fica mais oscilatória

2.3.2 Modo Integral

Para o controlador com modo integral, a saída do controlador depende da integral do sinal do erro ao longo do tempo, como mostra a equação (2.3), onde τ_I é o tempo integral.

$$p(t) = \bar{p} + \frac{1}{\tau_I}\int_0^t e(t^*)dt^* \tag{2.3}$$

A ação integral é muito usada pela sua vantagem de eliminar o *offset*. Isso sempre ocorre, com exceção de quando o distúrbio é capaz de provocar um erro muito grande a ponto de que a variação da variável manipulada não seja o suficiente para que a variável controlada volte ao seu valor *set point*. Mas, é importante citar que a ação integral normalmente é usada junto com a ação proporcional, criando um controlador proporcional - integral (PI), de acordo com a equação (2.4) (SEBORG et al., 2011).

$$p(t) = \bar{p} + K_C\left(e(t) + \frac{1}{\tau_I}\int_o^t e(t^*)dt^*\right) \tag{2.4}$$

A desvantagem do modo integral é a resposta oscilatória da variável controlada, o que reduz a estabilidade do controle por feedback. Existe um limite de oscilação que é tolerado para obter-se uma resposta rápida. Mas, para reduzir esse efeito indesejado, pode-se usar a ação derivativa (SEBORG ET AL., 2011).

2.3.3 Modo Derivativo

O modo derivativo antecipa o futuro comportamento do erro, considerando as suas variações anteriores. Desta forma, este modo tende a estabilizar o controle de processos e é usado para balancear a desestabilização provocada pelo modo integral (SEBORG ET AL., 2011).

A saída do controlador equivale ao valor do erro no estado estacionário enquanto ele for constante, ou seja, enquanto $\frac{de}{dt} = 0$, como mostra a equação (2.5), onde τ_D é o tempo

derivativo. Consequentemente, o modo derivativo nunca pode ser usado sozinho, precisa estar combinado com os modos proporcional ou proporcional - integral (SEBORG ET AL., 2011).

$$p(t) = \bar{p} + \tau_D \frac{de(t)}{dt} \tag{2.5}$$

Outra vantagem do modo derivativo é que ele melhora a resposta dinâmica do controlador. Mas, por outro lado, ele não pode ser utilizado quando a medição do processo é turbulenta, ou seja, quando existe grandes variações nas variáveis medidas. Neste caso, a ação derivativa aumentaria essa turbulência, afetando na segurança do processo (SEBORG ET AL., 2011).

2.3.4 Controle PID (Proporcional-Integral-Derivativo)

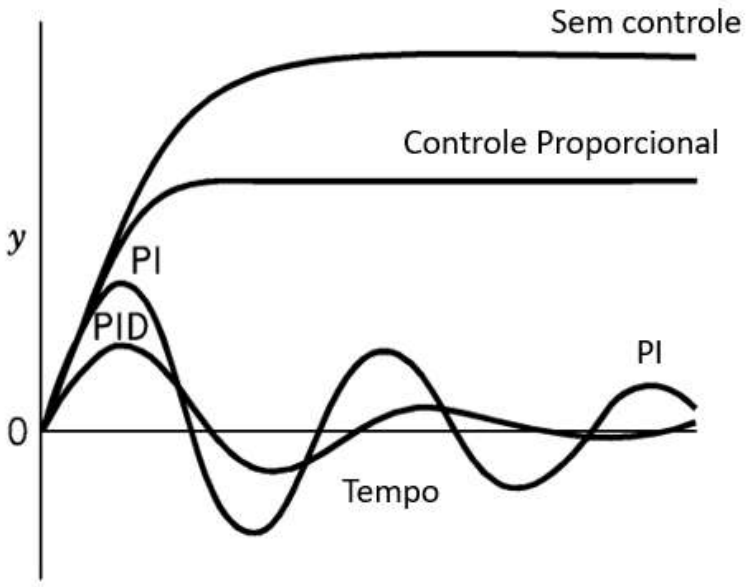
Controladores PI e PID são os mais usados para o controle de processos.

A equação (2.6) é a forma estendida que descreve o controlador PID. Nela, cada um dos três ganhos influencia somente o seu modo (SEBORG et al., 2011).

$$p(t) = \bar{p} + K_C e(t) + K_I \int_0^t e(t^*)dt^* + K_D \frac{de(t)}{dt} \tag{2.6}$$

A Figura 2.3 mostra o comportamento de processos com sistema de controle por feedback depois de uma variação de um degrau na variável distúrbio. Sem nenhum tipo de controle, é possível ver que o processo atinge lentamente um novo estado estacionário. Já o controle com o modo proporcional acelera essa resposta do processo e reduz o *offset*. A adição do modo integral, formando o controle PI elimina o *offset* mas deixa a resposta oscilatória. A adição do modo derivativo mostra que o controle PID reduz a oscilação e o tempo de resposta. Mas, é importante ressaltar que nem sempre os controles P, PI e PID resultarão em respostas oscilatórias já que a resposta depende dos valores das variáveis do controlador (K_C, τ_I e τ_D) e da dinâmica do processo (SEBORG et al., 2011).

Figura 2.3 - Resposta típica de processos com controle *feedback*.



Fonte: SEBORG et al., 2011

2.4 FUNÇÕES DE TRANSFERÊNCIA

Cada um dos blocos da Figura 2.1 representam funções de transferências que estão descritas nesta seção.

A função de transferência para o sensor e transmissor $G_m(s)$ pode ser descrita pela equação (2.7). K_m é o ganho estacionário e depende da variação das entrada e saída da variável que será medida, de acordo com a equação (2.8), sendo *span* a faixa de medição possível da variável medida. Já τ_m é a constate de tempo do medidor. A dinâmica do processo pode ser considerada desprezível ($\tau_m = 0$) quando a constante de tempo τ do processo é muito maior do que o próprio τ_m.

$$G_m(s) = \frac{K_m}{\tau_m s + 1} \tag{2.7}$$

$$K_m = \frac{\Delta\ variável\ medida}{span} \tag{2.8}$$

Existem seis formas diferentes para a função de transferência do controlador. A primeira, descrita na equação (2. 9) é referente ao controlador P. A equação (2.10) se refere ao controlador

integral e a equação (2.11) se refere ao controlador derivativo, que nunca é usado na prática. Já a equação (2.12) se refere ao controlador PI, a (2.13) ao controlador PD e a (2.14) ao controlador PID.

$$G_c(\mathrm{s}) = K_C \tag{2. 9}$$

$$G_c(s) = \frac{1}{\tau_I s} \tag{2.10}$$

$$G_c(s) = \tau_D s \tag{2.11}$$

$$G_c(s) = K_C * \left(1 + \frac{1}{\tau_I s}\right) \tag{2.12}$$

$$G_c(s) = K_C * (1 + \tau_D s\,) \tag{2.13}$$

$$G_c(s) = K_C \left(1 + \frac{1}{\tau_I s} + \tau_D s\right) \tag{2.14}$$

A função de transferência do elemento final de controle ou da válvula $G_v(s)$ pode ser descrita como uma função de transferência de primeira ordem, demonstrada na equação (2.15). O τ_v é definido de acordo com a válvula que é utilizada e deve ser idealmente igual a zero, o que representa uma dinâmica desprezível em relação à dinâmica do processo. Já K_v pode ser definido a partir de uma relação entre a saída e a entrada, ou seja, a variável manipulada e o sinal de saída do controlador. Em geral, esta relação não é linear, mas isto é algo que se busca. Quando a relação é linear, K_v pode ser calculado conforme descrito a equação (2.16).

$$G_v(s) = \frac{K_v}{\tau_v s + 1} \tag{2.15}$$

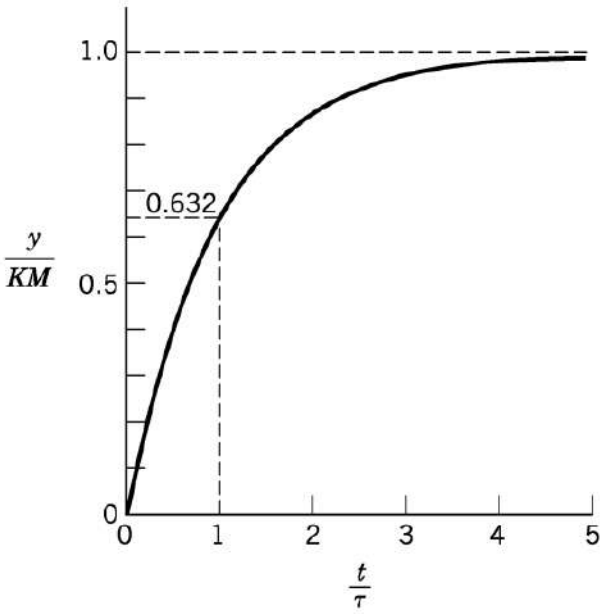
$$K_v = \frac{\Delta\ variável\ manipulada}{\Delta\ sinal\ de\ saída\ do\ controlador} \tag{2.16}$$

Uma forma de determinar a função de transferência do processo é avaliando sua resposta a diferentes inputs, como por exemplo uma função degrau, que é bastante comum em processos industriais.

A Figura 2.4 mostra a resposta de um processo de primeira ordem de constante de tempo τ a um degrau de magnitude M. O gráfico é adimensional, o tempo foi dividido pela constante de tempo do processo τ e a variação da saída Y dividida pelo valor do degrau (M) e pelo ganho do processo (K) (SEBORG ET AL., 2011).

É possível reparar que o processo não responde instantaneamente a uma mudança brusca na variável manipulada. Na verdade, o processo só atinge 63,2% do seu novo valor estacionário depois que se passa um intervalo de tempo igual a τ. Este valor só é de fato atingido quando $t \to \infty$, mas, pode ser aproximado quando $t \approx 5\tau$ (SEBORG ET AL., 2011).

Figura 2.4– Resposta no tempo de um processo de primeira ordem.



Fonte: SEBORG et al., 2011

Assim, a função de transferência de um processo de primeira ordem, $G_p(s)$, está descrita na equação (2.17). O ganho do processo pode ser obtido através da equação (2.18). Já a constante de tempo pode ser obtida a partir da premissa de que o processo está 63,2% completo no tempo τ_p.

$$G_p(s) = \frac{K_p}{\tau_p s + 1} \tag{2.17}$$

$$K_p = \frac{\Delta\ saÍda}{\Delta\ entrada} \tag{2.18}$$

2.5 SÍNTESE DIRETA

As configurações do controlador PID podem ser determinadas por diferentes técnicas. Neste trabalho, utilizou-se o a Síntese Direta (DS), que se baseia no modelo do processo e as funções de transferência para a malha fechada. Este método geralmente é usado para mudanças no *set point*, que levam a sintonias mais conservadoras, mas a função de transferência entre a VC e o distúrbio em malha fechada também poderia ser usada. Pode ser usado para processos que são estáveis em malha aberta e o controlador gerado pode ter estrutura PID ou PI (SEBORG et al., 2011).

O diagrama de blocos de um sistema de controle por feedback (Figura 2.1) deve ser o primeiro passo a ser realizado. A função de transferência do sistema em malha fechada para mudanças no *set point* é:

$$\frac{Y}{Y_{SP}} = \frac{K_m G_c G_v G_p}{1 + G_c G_v G_p G_m} \tag{2.19}$$

Para simplificar a equação (2.19), considera-se $G \triangleq G_v G_p G_m$. Além disso, também se considera que a dinâmica do medidor é desprezível, ou seja, $\tau_m = 0$, obtendo que $G_m = K_m$ e a equação (2.20).

$$\frac{Y}{Y_{SP}} = \frac{G_c G}{1 + G_c G} \tag{2.20}$$

Rearranjando a equação (2.19), obtêm-se a equação (2.21), que é a expressão para um controlador por feedback ideal. Mas, a função de transferência para a malha fechada (Y/Y_{SP}) não é conhecida *a priori* (SEBORG ET AL., 2011).

$$G_c = \frac{1}{G}\left(\frac{\frac{Y}{Y_{SP}}}{1-\frac{Y}{Y_{SP}}}\right) \tag{2.21}$$

Além disso, também é necessário fazer uma distinção entre o processo (G) e o modelo do processo ($\tilde{G}$), que é uma aproximação obtida pelo comportamento do processo. Assim, substitui-se o G por $\tilde{G}$ e o $\frac{Y}{Y_{SP}}$ por uma função de transferência desejada para a malha fechada $\left(\frac{Y}{Y_{SP}}\right)_d$, obtendo-se a equação (2.22) (SEBORG ET AL., 2011).

$$G_c = \frac{1}{\tilde{G}}\left[\frac{\left(\frac{Y}{Y_{SP}}\right)_d}{1-\left(\frac{Y}{Y_{SP}}\right)_d}\right] \tag{2.22}$$

Para um controle perfeito, deveríamos ter $\left(\frac{Y}{Y_{SP}}\right)_d = 1$, o que representa que a variável medida seria igual ao valor do set point, independente de mudanças e sem nenhum erro. Mas, esta situação não pode ser obtida pelo controle por feedback, já que o controlador só toma uma ação quando o erro for diferente de zero (SEBORG ET AL., 2011).

Processos com modelo de primeira ordem sem tempo morto são representados pela equação (2.23), onde τ_c é a constante de tempo desejada. Para este modelo, considera-se que o tempo de estabilização do processo é aproximadamente $5\tau_c$. Essas funções de transferência são consideradas para representar o comportamento desejado da resposta em malha fechada. Além disso, usar um ganho estacionário igual a 1 faz com que se obtenha o desejado $Y = Y_{SP}$ em regime estacionário (SEBORG ET AL., 2011).

$$\left(\frac{Y}{Y_{SP}}\right)_d = \frac{1}{\tau_c s + 1} \tag{2.23}$$

Assim, rearranjando as equações descritas, é possível obter a equação (2.24). Com o termo $\frac{1}{\tau_c s}$ é possível obter a ação integral no controle, o que elimina o *offset*. A escolha do τ_c

é capaz de deixar o controle mais agressivo (menor τ_c) ou menos agressivo e conservador (grande τ_c).

$$G_c = \frac{1}{\tilde{G}} \frac{1}{\tau_c s} \tag{2.24}$$

É importante ressaltar que a equação (2.24) não é adequada para modelos onde exista tempo morto. É possível usar o método da Síntese Direta nesse caso, mas outras considerações seriam necessárias, levando em conta o atraso no tempo representado por θ (SEBORG ET AL., 2011).

Existem algumas orientações para a escolha de τ_c. Para a maior parte dos processos, é aceitável pensar que $\tau_{dom} > \tau_c$, sendo τ_{dom} a maior constante de tempo do processo. A equação (2.25) significa que a resposta desejada para o sistema em malha fechada é três vezes mais rápido do que o em malha aberta (SEBORG ET AL., 2011).

$$\tau_c = \frac{\tau_{dom}}{3} \tag{2.25}$$

2.6 TRANSFORMADA Z E CONTROLE DIGITAL

O uso de computadores no controle de processos faz com que seja necessário converter o sinal analógico contínuo em um sinal discreto para que seja possível processar os dados. Desta forma, amostras devem ser retiradas do sinal e, então, deve-se calcular um sinal digital correspondente para cada uma dessas amostras. O intervalo de tempo entre cada uma das sucessivas amostras se chama tempo de amostragem (Δt) (SEBORG ET AL., 2011).

O tempo de amostragem deve ser pequeno o suficiente para que informações importantes não sejam perdidas. Ou seja, precisa-se evitar o *aliasing*, que é quando o sinal obtido pelas amostras representa uma função muito diferente da original. Além disso, também deve ser pequeno o suficiente para que a efetividade do processo seja mantida. (SEBORG ET AL., 2011).

É claro que quanto menor Δt for, mais a função discreta se aproxima do sinal contínuo. Mas, isso também representa uma grande quantidade de dados, aumentando o trabalho

computacional, o que limita o número de loops que a máquina conseguirá calcular. Desta forma, o cálculo de Δt deve levar em consideração a dinâmica do processo e o objetivo do controle (BABATUNDE; OGUNNAIKE; HARMON, 1996; SEBORG ET AL., 2011).

Não existem regras claras e definidas sobre como calcular Δt. A escolha mais prudente é aquela em que seja possível garantir que a informação da dinâmica do processo será mantida, mas que não sobrecarregue o computador. Seborg et al. (2011) cita uma orientação com base no trabalho de Åström e Wittenmark (1997), descrita na equação (2.26), sendo τ_{dom} a maior constante de tempo do processo (BABATUNDE; OGUNNAIKE; HARMON, 1996; SEBORG ET AL., 2011).

$$0{,}01 \leq \frac{\Delta t}{\tau_{dom}} \leq 0{,}05 \tag{2.26}$$

A orientação dada por BABATUNDE *et al*. (1996) diz que, para processos que possuam uma constante de tempo dominante, τ_{dom}, Δt deve ser calculado de acordo com a equação (2.27).

$$\Delta t \approx 0{,}2 * \tau_{dom} \tag{2.27}$$

Em sistemas que tenham mais de uma variável controlada, existirá uma constante de tempo dominante para cada uma das malhas de controle. Assim, no cálculo da equação (2.27), é importante utilizar a menor das duas constantes. Caso contrário, não seria possível capturar a dinâmica da malha com o menor τ_{dom}.

O design e a análise de sistemas digitais de controle é facilitado pelo uso da transformada z, que é, para o sistema discreto, análoga à transformada de Laplace para os sistemas contínuos (BABATUNDE; OGUNNAIKE; HARMON, 1996; SEBORG ET AL., 2011).

É possível passar do domínio de Laplace para o domínio z a partir da substituição descrita da equação (2.28), que é a substituição de diferenças retrógradas. Utilizando esta equação em um controlador PID, descrito em (2.14), obtêm-se a equação (2.29) (SEBORG ET AL., 2011).

$$s \cong \frac{(1 - z^{-1})}{\Delta t} \tag{2.28}$$

$$G_c(z) = \frac{K_c(a_0 + a_1 z^{-1} + a_2 z^{-2})}{1 - z^{-1}} \tag{2.29}$$

$$a_0 = 1 + \frac{\Delta t}{\tau_I} + \frac{\tau_D}{\Delta t} \tag{2.30}$$

$$a_1 = -\left(1 + \frac{2\tau_D}{\Delta t}\right) \tag{2.31}$$

$$a_2 = \frac{\tau_D}{\Delta t} \tag{2.32}$$

2.7 MODELOS DE ESPAÇO DE ESTADOS E MATRIZ DE FUNÇÕES DE TRANSFERÊNCIA

Os modelos de espaço de estados são muito úteis na representação de sistemas dinâmicos. As equações (2.33) e (2.34) descrevem um modelo em espaço de estado em que x é o vetor de estado, u é o vetor de entrada das variáveis manipuladas, d é o vetor de distúrbio e y é o vetor de saída das variáveis controladas medidas. Geralmente, x, u, d e y são funções do tempo. A derivada no tempo de x, $\frac{dx}{dt}$, também é um vetor. A, B, C e E são matrizes constantes (SEBORG ET AL., 2011).

$$\frac{dx}{dt} = Ax + Bu + Ed \tag{2.33}$$

$$y = Cx \tag{2.34}$$

Aplicando a transformada de Laplace a essas duas equações, obtêm-se uma expressão geral para a conversão de um modelo espaço de estados na sua matriz de funções de transferência, descrita nas equações abaixo. $Y(s)$ é um vetor coluna da transformada de Laplace de $y(t)$ e os outros vetores são obtidos de forma análoga (SEBORG ET AL., 2011).

$$sX(s) = AX(s) + BU(s) + ED(s) \tag{2.35}$$

$$Y(s) = CX(s) \tag{2.36}$$

Aplicando álgebra linear e rearranjando as equações, é possível obter a representação da função de transferência descrita na equação (2.37). A matriz da função de transferência do processo, $G_p(s)$, está descrita em (2.38) e a matriz da função de transferência do distúrbio, $G_d(s)$, está descrita em (2.39) (SEBORG ET AL., 2011).

$$Y(s) = G_p(s)\, U(s) + G_d(s)\, D(s) \tag{2.37}$$

$$G_p(s) \triangleq C\, [sI - A]^{-1}\, B \tag{2.38}$$

$$G_d(s) \triangleq C\, [sI - A]^{-1}\, E \tag{2.39}$$

As dimensões das matrizes dependem das dimensões de Y, U e D. É possível encontrar essas matrizes usando a função *ss2tf* no Python (SEBORG ET AL., 2011).

2.8 CONTROLE ADAPTATIVO

Alguns processos possuem características que mudam com o tempo ou com as condições operacionais. Para estes casos, um controlador com parâmetros fixos pode não ser eficiente. Por conta disso, desenvolveu-se o controle adaptativo, que se baseia no ajuste dos parâmetros do controlador para se adaptar à dinâmica do processo e aos distúrbios existentes. (ÅSTRÖM; HÄGGLUND, 1995).

O controle adaptativo contém diferentes técnicas que fornecem uma abordagem sistemática para ajustar o controlador de forma automática e em tempo real. O objetivo é atingir ou manter um nível de desempenho desejado no controle quando os parâmetros da planta são dinâmicos ou desconhecidos (LANDAU ET AL., 2011).

O desenvolvimento do controle adaptativo e dos seus métodos só ocorreu graças às teorias de estabilidade e ao controle moderno. A capacidade de manter o desempenho de um sistema

de controle em malha fechada mesmo com mudanças na dinâmica da planta vem motivando estudos sobre este tema (LUIZ ET AL., 1997).

A principal hipótese do controle adaptativo é que para qualquer ponto de operação da planta, existe um controlador com parâmetros fixos que seja capaz de obter o desempenho desejado, desde que esses parâmetros sejam escolhidos de forma apropriada. Isso enfatiza a importância do projeto do controlador e do conhecimento das informações sobre a estrutura da planta. O sistema de controle por feedback com parâmetros fixos é uma forma satisfatória para obter informações sobre a planta. Este modelo geralmente é desenvolvido ao redor de um ponto nominal (LANDAU ET AL., 2011).

Existem dois métodos de controle adaptativo: o direto e o indireto. No método direto, os parâmetros do controlador são ajustados a partir de dados obtidos pelo processo em malha fechada. Já nos métodos indiretos, os dados do processo são simulados offline e os parâmetros do controlador são então obtidos a partir do design do controle (ÅSTRÖM; HÄGGLUND, 1995).

Alguns exemplos de métodos indiretos são: Colocação Adaptativa de polos; Controle Preditivo Adaptativo Generalizado; Controle Adaptativo Linear Quadrático *e Gain Scheduling*, que foi o método utilizado neste trabalho (LANDAU ET AL., 2011).

2.8.1 Método Direto – Sintonia automática

A sintonia automática é um dos métodos de controle adaptativo direto onde o controlador é sintonizado automaticamente de acordo com a demanda do usuário (ÅSTRÖM; HÄGGLUND, 1995).

A sintonia automática de um controlador pode ser obtida a partir da combinação de métodos para determinar a dinâmica do processo com métodos que calculam os parâmetros de um controlador PID. Para isso, três passos são necessários: a ocorrência de um distúrbio que retire o processo do seu estado estacionário, a análise da resposta do sistema a esse distúrbio e o cálculo dos parâmetros do controlador (ÅSTRÖM; HÄGGLUND, 1995).

Controladores PID com capacidade de sintonia automática só começaram a estar disponíveis no mercado no começo dos anos 1980. Isto aconteceu graças ao desenvolvimento de tecnologias que tornaram isto possível a um preço acessível (ÅSTRÖM; HÄGGLUND, 1995).

2.8.2 Método Indireto – *Gain Scheduling*

O método de *Gain Scheduling* (Ganho Programado, traduzido para o português) é utilizado para casos de processos não lineares, processos que variam com o tempo ou processos em que o controle muda dependendo das condições operacionais. É um método eficaz para controlar sistemas cuja dinâmica muda com as condições operacionais (ÅSTRÖM; HÄGGLUND, 1995).

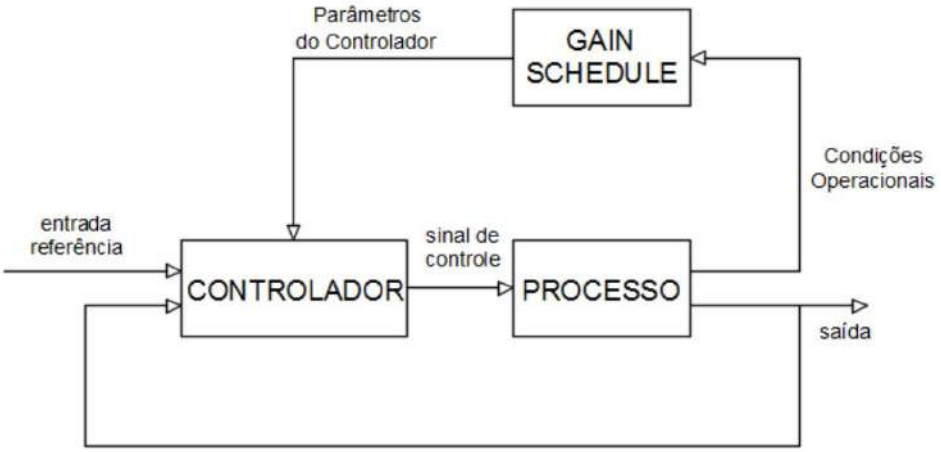
O *Gain Scheduling* começou a ser usado em projetos de pilotos automáticos para aeronave de alto desempenho em 1950 e esta foi uma das principais motivações para pesquisas. Os aviões se encaixam na ideia de processos que mudam com o tempo, já que operam com diferentes velocidades e em diferentes altitudes, com dinâmicas totalmente não linear e variável. Desta forma, para um dado ponto de operação, que é definido pela velocidade da aeronave e a altitude, sua dinâmica complexa pode ser aproximada por um modelo linear (SILVA, 2016).

Para usar esta técnica, é necessário medir algumas variáveis, chamadas de variáveis agendadas e que se relacionem bem com as mudanças na dinâmica do processo. O *Gain Scheduling* é uma alternativa interessante porque é capaz de rapidamente se adaptar às mudanças nas condições operacionais. Por outro lado, é difícil definir quais devem ser as variáveis agendadas (ÅSTRÖM; HÄGGLUND, 1995).

O mecanismo de adaptação neste caso é simplesmente consultar no computador uma matriz que tenha os parâmetros já calculados para cada um dos pontos de operação possíveis, dentro das faixas discretizadas das variáveis agendadas. Assim, ao alterar os parâmetros do controlador, é possível reduzir ou até eliminar o efeito das variações dos parâmetros no desempenho do controle (LANDAU ET AL., 2011).

A Figura 2.5 mostra um diagrama de blocos que representa um sistema em malha fechada com *Gain Scheduling*. Pode-se observar a existência de duas malhas: uma interna composta pelo processo e pelo controlador, e outra externa, onde os parâmetros do controlador são ajustados com base nas condições operacionais, o que garante a estabilidade e a efetividade do sistema. O bloco *Gain Schedule* representa uma matriz que possui valores pré-calculados dos parâmetros do controlador armazenados para as diferentes condições do processo (ÅSTRÖM; HÄGGLUND, 1995; SILVA, 2016).

Figura 2.5 - Sistema em malha fechada com controlador *Gain Scheduling*.



Fonte: SILVA, 2016

O desenvolvimento deste método de controle precisa de um grande esforço. O uso da sintonia automática pode facilitar muito o seu uso. Desta forma, o primeiro passo é a determinação das variáveis agendadas. Depois disso, um intervalo de variação destas variáveis é definido e quantificado em faixas discretas. Os parâmetros do controlador são determinados a partir da sintonia automática. Estes parâmetros calculados são armazenados em uma matriz e o processo é repetido até obter-se valores para todas as condições operacionais. Assim, é possível utilizar o *Gain Scheduling* em sistemas de controle computadorizados a partir da programação de uma matriz para armazenar e reprogramar os parâmetros do controlador (ÅSTRÖM; HÄGGLUND, 1995).

2.9 REATOR DE VAN DE VUSSE

Um modelo matemático para uma reação química complexa em um reator de tanque agitado contínuo (CSTR) foi introduzido como um *benchmark* para o design de processos não lineares. Dentre esses processos, existe o mecanismo de reação do reator de Van de Vusse (KLATT; ENGELL, 1998).

O mecanismo de reação apresentado por Van de Vusse se baseia na produção de ciclopentenol (B) a partir do ciclopentadieno (A) através da adição eletrofílica ácido catalisadora de água na solução diluída. Como subprodutos desta reação, obtêm-se o diciclopentadieno (D), produzido a partir da reação de Diels-Alder, e o ciclopentanediol (C),

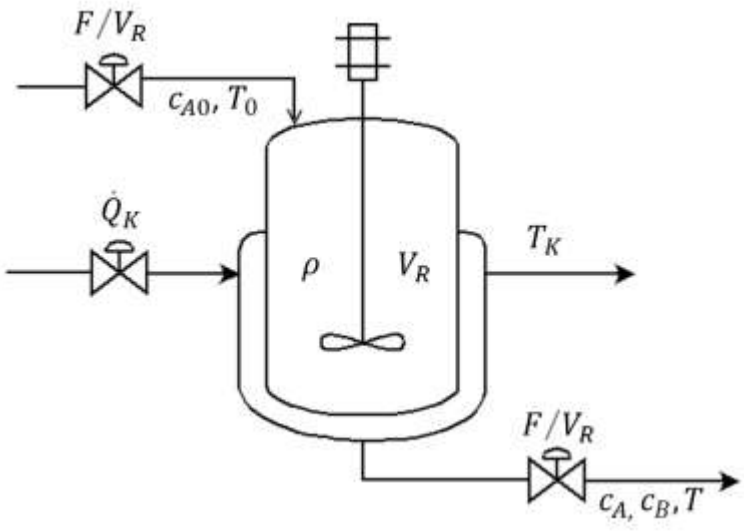
através da adição de outra molécula de água ao produto. Estas reações estão descritas no esquema reacional abaixo (VAN DE VUSSE, 1964).

$$A \overset{k_1}{\rightarrow} B \overset{k_2}{\rightarrow} C \tag{2.40}$$

$$A + A \overset{k_3}{\rightarrow} D \tag{2.41}$$

A Figura 2.6 mostra o modelo reacional do CSTR desta reação. A corrente de entrada do tanque só contém o reagente A em baixas concentrações (c_{A0}), o tempo de residência é considerado ideal para que a reação ocorra conforme o desejado e a mistura é perfeita, ou seja, a densidade é constante (FREITAS; BEDOR; CAPRON, 2020).

Figura 2.6 - Representação esquemática do reator de Van de Vusse.



Fonte: FREITAS; BEDOR; CAPRON, 2020

A nomenclatura utilizada para descrever os componentes do sistema, além dos parâmetros cinéticos, as constantes e variáveis físicas do processo está descrita na Figura 2.7.

Figura 2.7 – Nomenclatura utilizada na descrição do Reator de Van de Vusse.

Variáveis e constantes		Índices	
c	Concentração molar	A	Ciclopentadieno
C_p	Capacidade calorífica	B	Ciclopentanol
F	Vazão	K	Camisa do reator
k_i	Razão de coeficiente	0	Valor da corrente de entrada
t	Tempo		
V_R	Volume de reator		
T	Temperatura		
k_w	Coeficiente de transferência de calor		
A_R	Superficie do reator		
$\dot{Q}_k$	Taxa de refrigeração		
ΔH_R	Entalpia da reação		
m_K	Massa do refrigerador		
ρ	Densidade		

Fonte: FREITAS; BEDOR; CAPRON, 2020

As equações de balanço do reator de Van de Vusse levando em conta as simplificações já citadas estão descritas a seguir (KLATT; ENGELL, 1998; VOJTESEK; DOSTAL, 2010).

$$\frac{dc_A}{dt} = \frac{F}{V_R}(c_{A0} - c_A) - k_1 c_A - k_3 c_A^2 \tag{2.42}$$

$$\frac{dc_B}{dt} = -\frac{F}{V_R} c_B + k_1 c_A - k_2 c_B \tag{2.43}$$

$$\frac{dT}{dt} = \frac{F}{V_R}(T_0 - T) + \frac{k_w A_R}{\rho C_p V_R}(T_K - T) - \frac{k_1 c_A \Delta H_R^{AB} + k_2 c_B \Delta H_R^{BC} + k_3 c_A^2 \Delta H_R^{AD}}{\rho C_p} \tag{2.44}$$

$$\frac{dT_K}{dt} = \frac{1}{m_K C_{pK}}[\dot{Q}_K + k_w A_R (T - T_K)] \tag{2.45}$$

Os valores dos parâmetros cinéticos da reação e as propriedades do reator e da camisa que foram considerados neste trabalho estão descritos nas Tabela 2.1 e Tabela 2.2.

Tabela 2.1- Parâmetros cinéticos da reação.

Reação	$k_i = k_{i0} e^{(E_i/T)}$			Entalpia de Reação $\Delta H_R(^{kJ}/_{mol})$
	k_{i0}	Unidade de k_{i0}	Energia de Ativação E_i(K)	
$A \overset{k_1}{\rightarrow} B$	$1{,}287 \times 10^{12}$	h^{-1}	$-9758{,}3$	4,2
$B \overset{k_2}{\rightarrow} C$	$1{,}287 \times 10^{12}$	h^{-1}	$-9758{,}3$	-11,0
$A + A \overset{k_3}{\rightarrow} D$	$9{,}043 \times 10^{9}$	$L/(mol \cdot h)$	-8560	-41,85

Fontes: KLATT; ENGELL, 1998

Tabela 2.2 - Propriedades físico-químicas e dimensão do reator.

Parâmetro	Valor	Unidade
ρ	0,9342	kg/L
C_p	3,01	$kJ/(kg \cdot K)$
k_w	4032	$kJ/(m^2 \cdot h \cdot K)$
A_R	0,215	m^2
V_R	10	L
m_K	5,0	kg
C_{pK}	2,0	$kJ/(kg \cdot K)$

Fontes:, KLATT; ENGELL, 1998

Existem muitos estudos de técnicas de controle baseadas no reator com cinética de Van de Vusse por conta da sua característica não linear. Dentre estes trabalhos, existe o de Vojtesek e Dostal (2010) que estudou o controle adaptativo de um reator CSTR com a reação de Van de Vusse a partir de simulações no computador. Eles foram capazes de obter um bom controle baseado em modelos polinomiais e lineares quadráticos.

Freitas et al. (2020) estudou a identificação do reator de Van de Vusse, com base em redes neuronais, concluindo que o modelo é muito sensível em relação a escolha do tempo de amostragem e a quantidade de dados utilizada.

O estudo de Ho et al. (2014) se baseou em dois *benchmarks*, sendo um deles o reator CSTR de Van de Vusse isotérmico. Eles propuseram um controlador preditivo generalizado com dupla adaptação capaz de modelar e sintonizar de forma síncrona e em tempo real. Os autores obtiveram um controlador com boa performance, representando melhorias ao esquema abordado em trabalhos anteriores. Além disso, também foram capazes de sugerir para novos estudos um novo esquema que poderia ser capaz de lidar com sistemas multivariáveis com interações em loop.

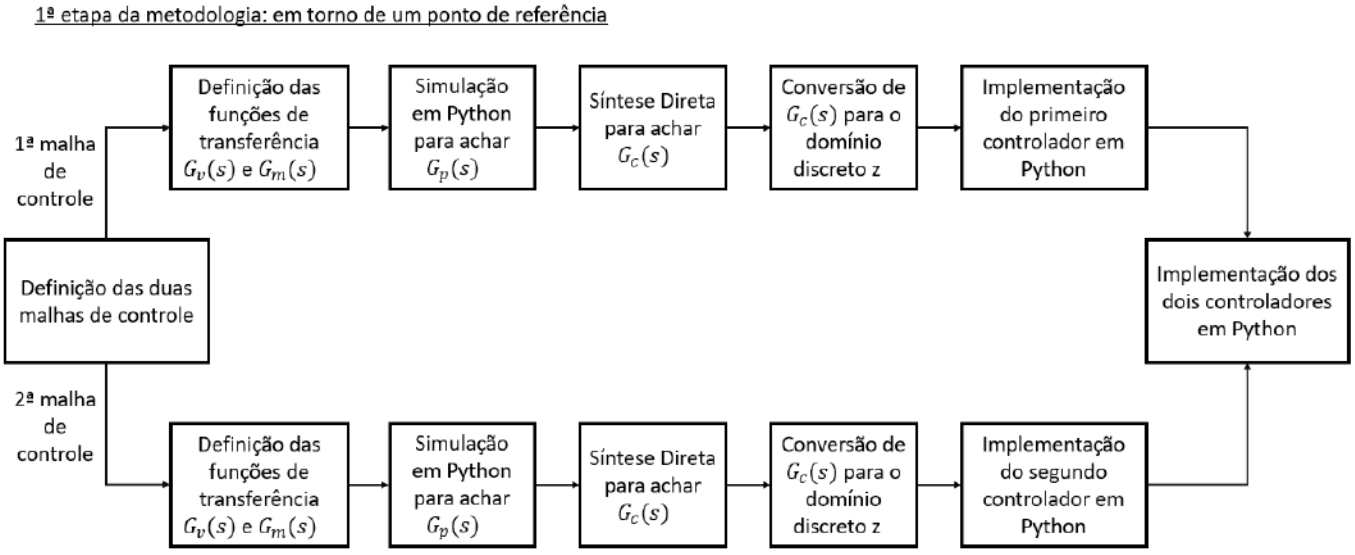
3 METODOLOGIA

3.1 FUNÇÕES DE TRANSFERÊNCIA E IMPLEMENTAÇÃO NO PYTHON EM FUNÇÃO DE UM PONTO DE REFERÊNCIA

Neste trabalho, duas malhas de controle foram consideradas. A primeira malha de controle teve como variável controlada c_B e como variável manipulada F. Como o produto de interesse no reator de Van de Vusse é B, é natural querer controlar sua concentração. Já a variável controlada da segunda malha de controle foi T, a partir da manipulação de $\dot{Q_K}$. A temperatura do reator possui grande relação com a produção de B e, por conta disso, ela também é uma variável interessante para controlar.

O procedimento desta primeira etapa da metodologia está descrito na Figura 3.1. Depois de definir as duas malhas de controle, o mesmo procedimento foi feito da mesma forma para as duas malhas. Primeiro, definiu-se $G_v(s)$ e $G_m(s)$ com base em estudos já realizados no reator de Van de Vusse. Com essas funções de transferência, realizou-se uma simulação em Python para ver como o processo reagiria a um distúrbio. Para isto, considerou-se o ponto de operação, cujos valores estão descritos na Tabela 3.1. Duas simulações foram feitas, primeiro um degrau positivo e depois um degrau negativo, ambos de mesma amplitude, na variável manipulada. A partir destas simulações, foi possível determinar o $G_p(s)$.

Figura 3.1 – Fluxograma da primeira etapa da metodologia.



Fonte: Elaboração própria

Tabela 3.1 – Pontos considerados como estado estacionário inicial nas simulações.

Parâmetro	Valor
F_{SS}	$1000\ L$
T_{SS}	$110\ °C$
$c_{A\ ss}$	$5{,}12\ mol/L$
$\dot{Q_{K}}_{SS}$	$-4250\ KJ/h$

Fonte: FREITAS; BEDOR; CAPRON, 2020

Depois disso, $G_c(s)$ foi determinada a partir do método de síntese direta, que foi convertido para o domínio z discreto. Após isso, foi possível implementar as funções de transferência dos controladores no Python, inicialmente separados e depois juntos.

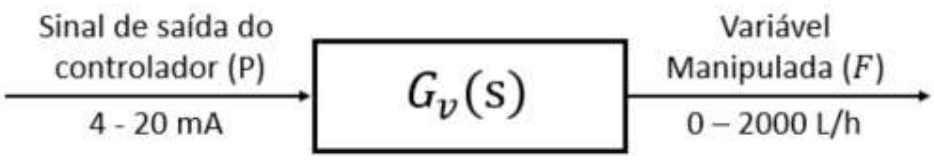
3.1.1 Malha de controle da concentração de B

Como já descrito, a variável controlada da primeira malha de controle foi c_B e a variável manipulada F.

O primeiro passo foi definir a função de transferência da válvula, $G_v(s)$, descrita pela equação (2.15). Para isso, era preciso definir K_v e τ_v.

A fórmula para determinar K_v está descrita em (2.16). Para fazer este cálculo foi preciso definir que um controlador elétrico seria considerado, ou seja, o sinal de saída do controlador seria em mA, podendo variar entre 4 e 20 mA. Além disso, foi necessário definir qual seria a variação possível de F. Para isto, considerou-se que a vazão poderia variar entre 0 e duas vezes o valor de F no estado estacionário, 1000 L/h. As variações definidas e a função de transferência da válvula estão representadas na Figura 3.2.

Figura 3.2– Diagrama de blocos de $G_v(s)$ da primeira malha de controle.



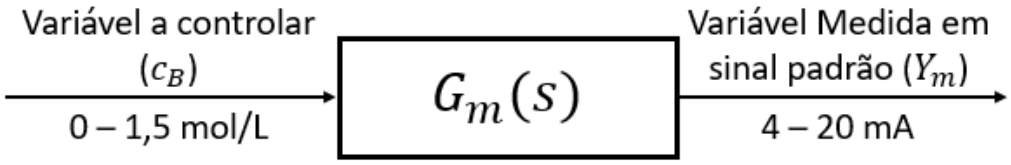
Fonte: Elaboração própria

A dinâmica de válvulas de controle tende a ser relativamente rápida considerando a dinâmica do processo em si. Assim, aproximar por uma função de transferência de primeira ordem, como descrito na equação (2.15), onde $\tau_v \ll \tau_p$ é uma consideração razoável para o estudo do controle de um processo (SEBORG ET AL., 2011). Desta forma, pode-se considerar que a dinâmica da válvula é desprezível, ou seja, τ_v é igual a zero.

O segundo passo foi definir a função de transferência para o sensor e transmissor, $G_m(s)$, também chamada de função de transferência do medidor, que foi descrita pela equação (2.7). Para isso, era preciso definir K_m e τ_m.

A fórmula para achar K_m está descrita em (2.8). Definiu-se que o transmissor também era elétrico, ou seja, o seu sinal é padrão e varia entre 4 e 20 mA. Com base em Cassol et al. (2018), considerou-se que o c_B variava entre 0 e 1,5 mol/L. Estas variações e a função de transferência do medidor estão representadas na Figura 3.3.

Figura 3.3– Diagrama de blocos de $G_m(s)$ da primeira malha de controle.



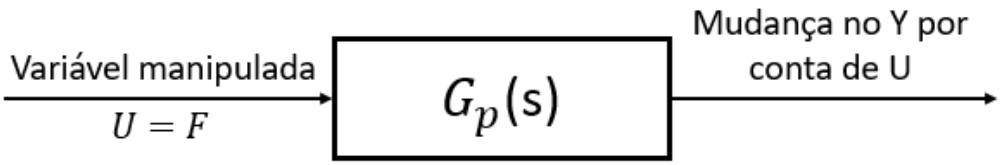
Fonte: Elaboração própria

Grande parte dos sensores e transmissores utilizados em controle respondem e medem de forma rápida em relação à dinâmica dos processos. Assim, a dinâmica do medidor pode ser considerada desprezível (SEBORG ET AL., 2011). Considerando que c_B pode ser medida por absorção ultravioleta, a constante de tempo na análise está na faixa de segundos, o que a torna significativamente menor do que a constante de tempo da dinâmica do reator. (TRIERWEILER, 1997). Assim, considerou-se que a dinâmica do medidor é desprezível e τ_m é igual a zero.

Com estas considerações, é possível obter as duas funções de transferência e a partir disso, realizou-se duas simulações em Python com o objetivo de se calcular G_p. É importante ressaltar que a simulação parte de um ponto de referência de estado estacionário cujos valores estão descritos na Tabela 3.1. Na primeira simulação, aplicou-se um degrau positivo de $100\ L/h$ e na segunda, um degrau negativo de $-100\ L/h$, ambos distúrbios na vazão e em $t = 0{,}001h$.

A Figura 3.4 mostra o diagrama de blocos de $G_p(s)$, deixando claro como um distúrbio na variável manipulada pode interferir na variável controlada c_B.

Figura 3.4 – Diagrama de blocos de $G_p(s)$ da primeira malha de controle.



Fonte: Elaboração própria

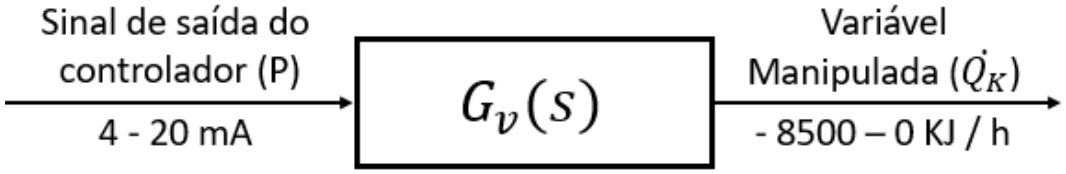
A função de transferência $G_p(s)$ foi calculada a partir da equação (2.17), K_p pela equação (2.18) e a constante de tempo pode ser obtida a partir da premissa de que a resposta atinge 63,2% do novo valor em regime estacionário em tempo τ_p horas.

3.1.2 Malha de controle da temperatura

A variável manipulada da segunda malha de controle foi a temperatura do reator T e a variável manipulada a taxa de refrigeração $\dot{Q_K}$. O passo a passo foi o mesmo realizado para a primeira malha.

Para definir a função de transferência da válvula $G_v(s)$, partiu-se das mesmas premissas de uma válvula com dinâmica desprezível, ou seja, τ_v igual a zero. Para o cálculo de K_v, a equação (2.16) foi utilizada, considerando-se de novo um controlador elétrico. A faixa de variação de $\dot{Q_K}$ foi definida como entre -8500 e 0 KJ/h, com base no estudo de Klatt e Engell, 1998. As variações definidas e a função de transferência da válvula estão representadas na Figura 3.5.

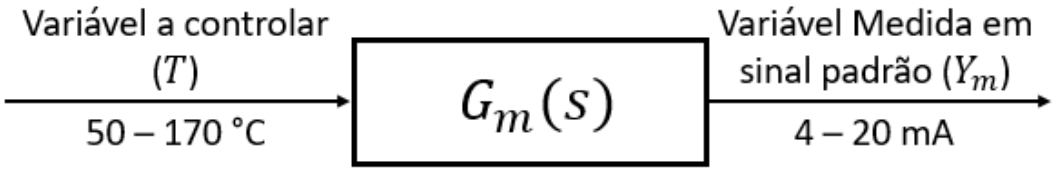
Figura 3.5– Diagrama de blocos de $G_v(s)$ da segunda malha de controle.



Fonte: Elaboração própria

Para definir a função de transferência do medidor $G_m(s)$, considerou-se dinâmica desprezível, então τ_m igual a zero. Já o ganho K_m foi calculado pela equação (2.8). O medidor considerado também foi elétrico e T variava entre 50 e 170°C, com base em Freitas, Bedor e Capron, 2020. Estas variações e a função de transferência do medidor estão representadas na Figura 3.6.

Figura 3.6 – Diagrama de blocos de $G_m(s)$ da segunda malha de controle.

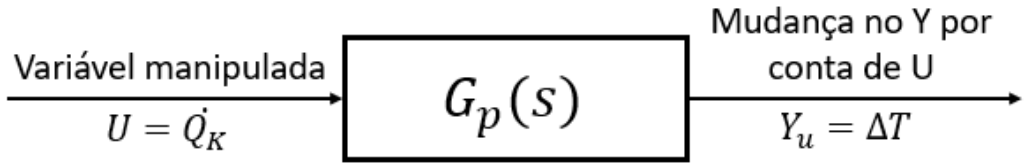


Fonte: Elaboração própria

Para calcular $G_p(s)$, duas simulações em Python foram feitas. Na primeira, aplicou-se um degrau positivo de +1000 KJ/h e na segunda, um degrau negativo de -1000 KJ/h, ambos distúrbios em $\dot{Q_K}$ e em $t = 0{,}001h$. Mais uma vez, a simulação partiu de um ponto de referência de estado estacionário cujos valores estão descritos na Tabela 3.1.

A Figura 3.7 mostra o diagrama de blocos de $G_p(s)$, mostrando como o distúrbio na variável manipulada pode interferir na variável controlada.

Figura 3.7– Diagrama de blocos de $G_p(s)$ da segunda malha de controle.



Fonte: Elaboração própria

A função de transferência do processo $G_p(s)$ da segunda malha foi obtida da mesma forma descrita para a primeira malha, de acordo com a equação (2.17). O ganho foi calculado pela equação (2.18) e a constante de tempo a partir da premissa de que a resposta atinge 63,2% do novo valor em regime estacionário em τ_p horas.

3.1.3 Cálculo do tempo de amostragem

Para converter o sinal analógico contínuo em um sinal discreto, no domínio z, foi preciso definir o tempo entre cada uma das sucessivas amostras, ou seja, o tempo de amostragem (Δt). Para isto, a orientação de BARBATUNDE *et al*., descrita na equação (2.27), foi levada em consideração.

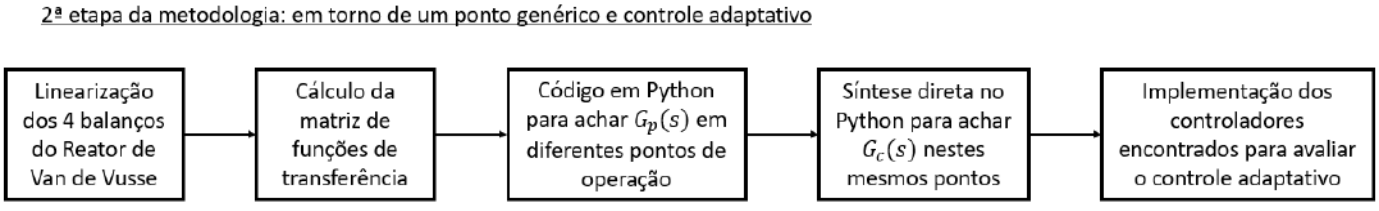
Neste trabalho, duas malhas de controle foram estudadas, então, existem duas constantes de tempo dominantes, uma para cada malha. Desta forma, no cálculo do tempo de amostragem, a menor das duas constantes foi considerada para que fosse possível capturar a dinâmica de ambas as malhas.

3.2 CÁLCULO DE $G_p(s)$ E $G_c(s)$ EM TORNO DE UM PONTO GENÉRICO

Em uma planta industrial real, várias condições de operação são possíveis para um determinado processo. Por exemplo, a concentração desejada do produto pode mudar, algum distúrbio pode alterar a temperatura, dentre outras opções possíveis. A cada mudança dessas, um novo estado estacionário é atingido, o que representa um ponto de operação diferente. A válvula e o medidor não irão mudar, logo, $G_v(s)$ e $G_m(s)$ vão continuar os mesmos, já calculados no tópico 3.1. Mas, o $G_p(s)$ não é constante para todos os pontos de operação, o que consequentemente altera o $G_c(s)$. Isso significa que, após mudanças no ponto de operação, o $G_p(s)$ inicialmente calculado não é mais capaz de descrever aquele processo e o $G_c(s)$ pode não conseguir mais realizar um controle eficaz.

Por conta disso, pensou-se em uma forma de automatizar o cálculo de $G_p(s)$ e $G_c(s)$, conseguindo um controle adaptativo para qualquer ponto de operação do processo estudado. Esta etapa da metodologia está esquematizada na Figura 3.8.

Figura 3.8– Fluxograma da segunda etapa da metodologia.



Fonte: Elaboração própria

O primeiro passo foi linearizar em função de um ponto de operação genérico os 4 balanços que descrevem a reação de Van de Vusse, descritos nas equações 2.28 a 2.31. O segundo passo foi escrever esses balanços em um modelo de matriz de funções de transferências. Depois, essas informações foram colocadas em um código no Python capaz de extrair as funções de transferência desejadas para achar o $G_p(s)$ de cada uma das malhas analisadas. Com esse $G_p(s)$, seria possível realizar a síntese direta para calcular $G_c(s)$.

O resultado obtido foi funções de transferência de 4ª ordem, o que inviabilizou o passo da síntese direta. Assim, uma das maiores dificuldades encontradas foi a automatização da parametrização do controlador para qualquer ponto de operação. O processo da reação estudada não possui uma função de transferência de primeira ordem, o que dificultou a sua representação genérica e a obtenção do controlador a partir da síntese direta.

Assumiu-se que mudanças no ponto de operação quase não provocam variação na constante de tempo τ do processo, mas influenciam muito no ganho K. Além disso, ao analisar os gráficos obtidos na simulação em Python para achar $G_p(s)$ na primeira etapa da metodologia (Figura 4.1, Figura 4.2, Figura 4.4 e Figura 4.5), observou-se que a resposta do processo é muito similar à resposta de uma função de primeira ordem (Figura 2.4). Com base nisso, considerou-se que $G_p(s)$ é sempre uma função de transferência de primeira ordem, com τ_p constante para todos os pontos de operação e K_p variando.

Levando em conta essa consideração, os passos descritos foram realizados e um código em Python foi desenvolvido para calcular um K_p aproximado a uma função de transferência de primeira ordem. Além disso, o cálculo da síntese direta também foi automatizado e incluído no programa.

3.2.1 Cálculo da matriz de funções de transferência para um ponto genérico

A obtenção da matriz de funções de transferência ao redor de um ponto de operação genérico foi automatizado. Para isso, o primeiro passo foi linearizar os quatros balanços que descrevem a reação de Van de Vusse descritos nas equações (2.42), (2.43), (2.44) e (2.45), através da expansão em série de Taylor truncada na primeira ordem ao redor do ponto de operação. O segundo passo foi calcular o balanço em estado estacionário. Por fim, o terceiro passo foi fazer a diferença entre o balanço em regime transiente linearizado e o balanço em regime estacionário. Esse passo a passo foi feito para todos os quatro balanços, obtendo quatro

equações com variáveis de estado que depois foram utilizadas para montar a matriz, de acordo com o descrito na seção 2.7.

As variáveis de estado do sistema da reação de Van de Vusse são c_A, c_B, T e T_K. Logo, x, que é o vetor das variáveis de estado, está descrito na equação (3.1), já considerando as variáveis de desvio obtidas no passo anterior.

$$x = \begin{bmatrix} c'_A \\ c'_B \\ T' \\ T'_K \end{bmatrix} \tag{3.1}$$

O vetor das variáveis manipuladas u está descrito na equação (3.2) e o vetor das variáveis medidas controladas y está em (3.3). Nenhum distúrbio foi considerado, então d é igual a zero.

$$u = \begin{bmatrix} F \\ \dot{Q}_k \end{bmatrix} \tag{3.2}$$

$$y = \begin{bmatrix} c_B \\ T \end{bmatrix} \tag{3.3}$$

A partir das equações acima, é possível chegar na equação (3.4). Os balanços de $^{dx}/_{dt}$ já foram encontrados e assim é possível calcular as matrizes A_{4X4} e $B_{4\,X\,2}$, cujo desenvolvimento está descrito abaixo.

$$\frac{dx}{dt} = \begin{bmatrix} \frac{dc_A}{dt} \\ \frac{dc_B}{dt} \\ \frac{dT}{dt} \\ \frac{dT_K}{dt} \end{bmatrix} = A_{4\,X\,4} \begin{bmatrix} c'_A \\ c'_B \\ T' \\ T'_K \end{bmatrix} + B_{4\,X\,2} \begin{bmatrix} F \\ \dot{Q}_k \end{bmatrix} + E * 0 \tag{3.4}$$

$$\frac{dc_A}{dt} = a_{11}c'_A + a_{12}c'_B + a_{13}T' + a_{14}T'_K + b_{11}F' + b_{12}\dot{Q}'_K \tag{3.5}$$

$$\frac{dc_B}{dt} = a_{21}c'_A + a_{22}c'_B + a_{23}T' + a_{24}T'_K + b_{21}F' + b_{22}\dot{Q}'_K \tag{3.6}$$

$$\frac{dT}{dt} = a_{31}c'_A + a_{32}c'_B + a_{33}T' + a_{34}T'_K + b_{31}F' + b_{32}\dot{Q}'_K \tag{3.7}$$

$$\frac{dT_K}{dt} = a_{41}c'_A + a_{42}c'_B + a_{43}T' + a_{44}T'_K + b_{41}F' + b_{42}\dot{Q}'_K \tag{3.8}$$

$$A_{4\,X\,4} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{3.9}$$

$$a_{11} = \left[-\frac{\bar{F}}{V_R} - \overline{k_1} - 2\overline{k_3}\overline{c_A} \right] \tag{3.10}$$

$$a_{12} = 0 \tag{3.11}$$

$$a_{13} = \left[\frac{\overline{k_1}\ \mathrm{E}_1\ \overline{c_A} + \overline{k_3}\ \mathrm{E}_3\ \overline{c_A}^2}{\bar{T}^2} \right] \tag{3.12}$$

$$a_{14} = 0 \tag{3.13}$$

$$a_{21} = \left[\overline{k_1} \right] \tag{3.14}$$

$$a_{22} = \left[-\frac{\bar{F}}{V_R} - \overline{k_2} \right] \tag{3.15}$$

$$a_{23} = \left[\frac{-\overline{k_1}\ E_1\ \overline{c_A} + \overline{k_2}\ E_2\ \overline{c_B}}{\overline{T^2}} \right] \tag{3.16}$$

$$a_{24} = 0 \tag{3.17}$$

$$a_{31} = \left[\frac{-\overline{k_1}\,\Delta H_R^{AB} - 2\,\overline{k_3}\,\overline{c_A}\,\Delta H_R^{AD}}{\rho\, C_p}\right] \tag{3.18}$$

$$a_{32} = \left[-\frac{\overline{k_2}\Delta H_R^{BC}}{\rho\, C_p}\right] \tag{3.19}$$

$$a_{33} = \left[-\frac{\bar{F}}{V_R} - \frac{k_w\, A_R}{\rho\, C_p\, V_R} + \frac{\overline{k_1}\, E_1\, \overline{c_A}\, \Delta H_R^{AB} + \overline{k_2}\, E_2\, \overline{c_B}\, \Delta H_R^{BC} + \overline{k_3}\, E_3\, \overline{c_A^2}\, \Delta H_R^{AD}}{\rho\, C_p\, \bar{T}^2}\right] \tag{3.20}$$

$$a_{34} = \left[\frac{k_w A_R}{\rho\, C_p\, V_R}\right] \tag{3.21}$$

$$a_{41} = 0 \tag{3.22}$$

$$a_{42} = 0 \tag{3.23}$$

$$a_{43} = \left[\frac{k_w\, A_R}{m_K\, C_{p\,K}}\right] \tag{3.24}$$

$$a_{44} = \left[-\frac{k_w\, A_R}{m_K\, C_{p\,K}}\right] \tag{3.25}$$

$$B_{4\,X\,2} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix} \tag{3.26}$$

$$b_{11} = \left[\frac{c_{A0} - \overline{c_A}}{V_R}\right] \tag{3.27}$$

$$b_{12} = 0 \tag{3.28}$$

$$b_{21} = \left[-\frac{\overline{c_B}}{V_R}\right] \tag{3.29}$$

$$b_{22} = 0 \tag{3.30}$$

$$b_{31} = \left[\frac{T_0 - \bar{T}}{V_R}\right] \tag{3.31}$$

$$b_{32} = 0 \tag{3.32}$$

$$b_{41} = 0 \tag{3.33}$$

$$b_{42} = \left[\frac{1}{m_K C_{p\,K}}\right] \tag{3.34}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.35}$$

3.3 CONTROLE ADAPTATIVO

O controle adaptativo é uma boa escolha para o controle do reator de Van de Vusse, que é um benchmark de processos não lineares. Neste trabalho, utilizou-se o método indireto de *Gain Scheduling* com o auxílio da sintonia automática.

O primeiro passo foi definir que as variáveis agendadas seriam: F, T_0, $c_{A\,0}$ e $\dot{Q}_K$. As suas faixas de variação estão descritas na Tabela 3.2 abaixo. As variações de F, T_0 e $c_{A\,0}$ foram definidas com base em FREITAS, BEDOR e CAPRON (2020), enquanto a variação de $\dot{Q}_K$ foi baseada em Trierweiler, 1997.

Tabela 3.2 – Faixas de variação das variáveis agendadas para o *Gain Scheduling*.

$0 \leq F \leq 2000\ L/h$
$100 \leq T_0 \leq 150\ °C$
$4 \leq c_{A\,0} \leq 6\ mol/L$
$-8500 \leq \dot{Q_K} \leq 0\ KJ/h$

Fonte: Elaboração própria

Estas variáveis agendadas se relacionam bem com as mudanças possíveis na dinâmica do processo. Ou seja, mudanças nessas variáveis dentro destas faixas de variação pré-definidas são distúrbios possíveis para este processo. Por isso, foi preciso calcular os parâmetros do controlador que se adequassem a diferentes pontos dentro destas faixas, garantindo um controle eficiente, independente do distúrbio que ocorresse.

Um código em Python foi programado. Nele, houve a quantificação em faixas discretas das faixas de variação das variáveis agendadas já definidas. Para cada uma destas faixas quantizadas, uma sintonia automática foi feita para calcular os parâmetros do controlador. Esta etapa foi feita a partir da automatização já descrita, com o uso das matrizes com as funções de transferência e da síntese direta automatizada. Os parâmetros do controlador calculados foram armazenados em uma matriz e este processo foi feito até obter-se valores para todas as condições operacionais consideradas. Os códigos em Python desenvolvidos neste trabalho estão nos Apêndice A e B.

4 RESULTADOS

4.1 FUNÇÕES DE TRANSFERÊNCIA E IMPLEMENTAÇÃO NO PYTHON EM FUNÇÃO DE UM PONTO DE REFERÊNCIA

4.1.1 Malha de controle da concentração de B

O cálculo das funções de $G_v(s)$ e de $G_m(s)$ da primeira malha de controle foi descrito na seção 3.1.1 e os resultados obtidos estão descritos na Tabela 4.1.

Tabela 4.1 – Funções de transferência obtidas para a primeira malha de controle.

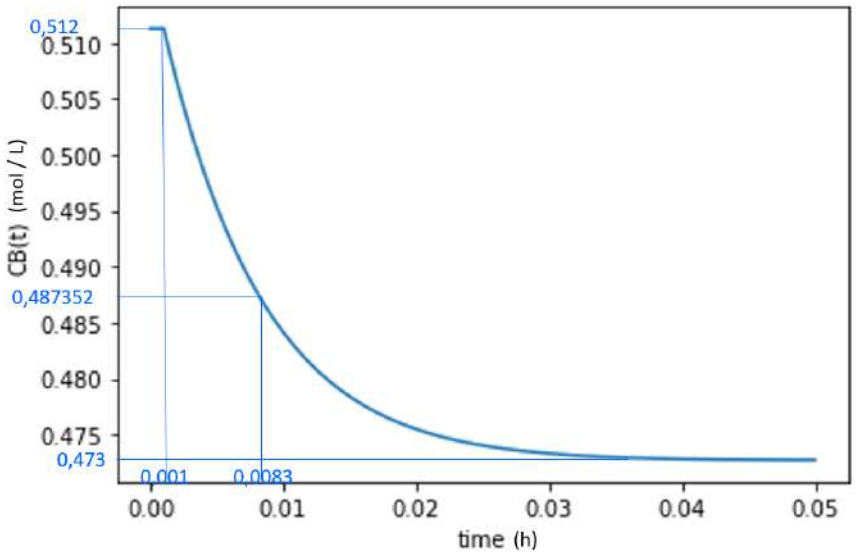
$G_{v\ 1ª\ malha}(s) = 125$	$G_{m\ 1ª\ malha}(s) = 10{,}667$

Fonte: Elaboração própria

Com as funções de transferência calculadas, foi possível realizar duas simulações em Python para calcular $G_p(s)$, como já descrito na metodologia.

A Figura 4.1 mostra o gráfico da variação de c_B em função do tempo obtido para a primeira simulação, com um degrau de 100 L/h. O novo c_B estacionário, ou seja, o seu valor quando $t \to \infty$ é 0,473 mol/L. Assim, a equação (4.1) mostra o valor do ganho do processo.

Figura 4.1 – c_B x t com degrau positivo para a primeira malha de controle.



Fonte: Elaboração própria

$$K_{p\ 1\ 1ª\ malha} = \frac{0{,}473 - 0{,}512}{1100 - 1000} = \frac{0{,}039}{100} = -3{,}9 * 10^{-4} mol/L^2 \quad (4.1)$$

Sabe-se que o sistema leva um tempo igual a τ_p depois da ocorrência do distúrbio para atingir 63,2% do novo valor estacionário, que equivale a $0{,}487352\ mol/L$. Na Figura 4.1 é possível ver que esse valor acontece quando o tempo é igual a $0{,}0083h$. Como o degrau aconteceu em $t = 0{,}001h$, a constante de tempo do processo corresponde a diferença entre esses valores, de acordo com a equação (4.2).

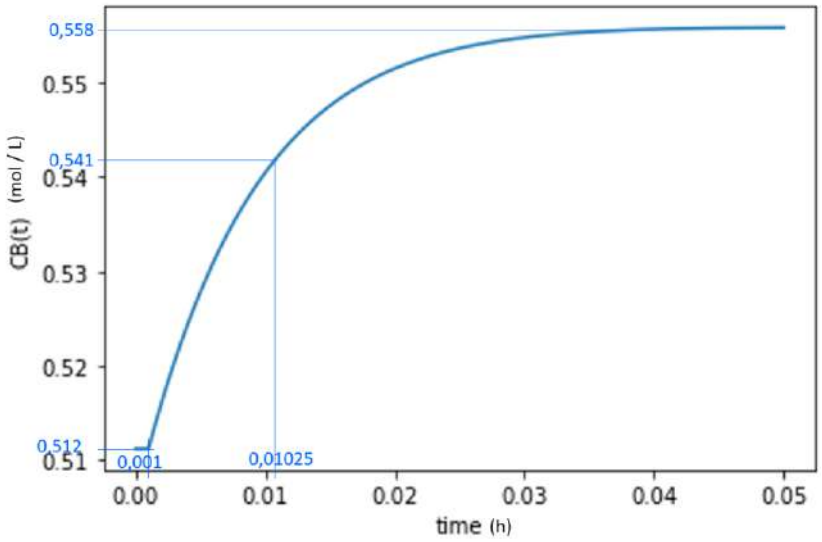
$$\tau_{p\ 1\ 1ª\ malha} = 0{,}0083 - 0{,}001 = 0{,}0073h \tag{4.2}$$

Assim, obteve-se o $G_{p\ 1}(s)$ referente à primeira simulação, indicado na equação (4.3).

$$G_{p\ 1\ 1ª\ malha}(s) = \frac{-3{,}9 * 10^{-4}}{0{,}0073s + 1} \tag{4.3}$$

O mesmo desenvolvimento foi feito para a simulação onde aplicou-se um degrau negativo de $-100\ L/h$ no mesmo $t\ = 0{,}001h$. O gráfico obtido está na Figura 4.2.

Figura 4.2– $c_B\ x\ t$ para com degrau negativo para a primeira malha de controle.



Fonte: Elaboração própria

O novo valor do estado estacionário de c_B é 0,558. Assim, o valor do ganho obtido para a segunda simulação está descrito na equação (4.4).

$$K_{p\ 2\ 1^{a}\ malha} = \frac{0{,}558 - 0{,}512}{900 - 1000} = \frac{0{,}046}{-100} = -4{,}6 * 10^{-4} mol/L^2 \quad (4.4)$$

Para o cálculo de τ_p calculou-se que 63,2% deste novo estado estacionário equivale a $0{,}541\ mol/L$, o que ocorre em $0{,}01025\ h$. Descontando o tempo antes do distúrbio acontecer, temos o valor da constante de tempo descrita na equação (4.5).

$$\tau_{p\ 2\ 1^{a}\ malha} = 0{,}01025 - 0{,}001 = 0{,}00925h \quad (4.5)$$

Assim, obteve-se o $G_{p\ 2}$, referente à segunda simulação, indicado na equação *(4.6)*.

$$G_{p\ 2\ 1^{a}\ malha}(s) = \frac{-4{,}6 * 10^{-4}}{0{,}00925s + 1} \quad (4.6)$$

Os parâmetros da função de transferência do processo foram calculados a partir das médias dos valores de K_p e τ_p obtidos, de acordo com as equações (4.7) e (4.8), obtendo-se o descrito na equação (4.9).

$$K_{p\ 1^{a}\ malha} = \frac{K_{p\ 1\ 1^{a}\ malha} + K_{p\ 2\ 1^{a}\ malha}}{2} = \frac{[(-3{,}9) + (-4{,}6)] * 10^{-4}}{2} = -4{,}25 * 10^{-4}\ mol/L^2 \quad (4.7)$$

$$\tau_{p\ 1^{a}\ malha} = \frac{\tau_{p1\ 1^{a}\ malha} + \tau_{p\ 2\ 1^{a}\ malha}}{2} = \frac{0{,}0073 + 0{,}00925}{2} = 0{,}008275\ h \quad (4.8)$$

$$G_{p\ 1^{a}\ malha}(s) = \frac{-4{,}25 * 10^{-4}}{0{,}008275s + 1} \quad (4.9)$$

Com os valores das funções de transferência calculados ($G_v\,(s), G_m(s)$ e $G_p(s)$), foi possível aplicar o método de Síntese Direta, de acordo com a equação (2.24) para calcular $G_c(s)$. O valor da constante de tempo do controlador τ_c foi calculado a partir da equação (2.25),

sendo $\tau_{dom} = 0{,}008275\ h$. Por fim, obteve-se o $G_c(s)$ descrito em (4.10), referente a um controlador do tipo PI.

$$G_{c\ 1^a\ malha}(s) = -5{,}2937 - 639{,}81 * \frac{1}{s} \tag{4.10}$$

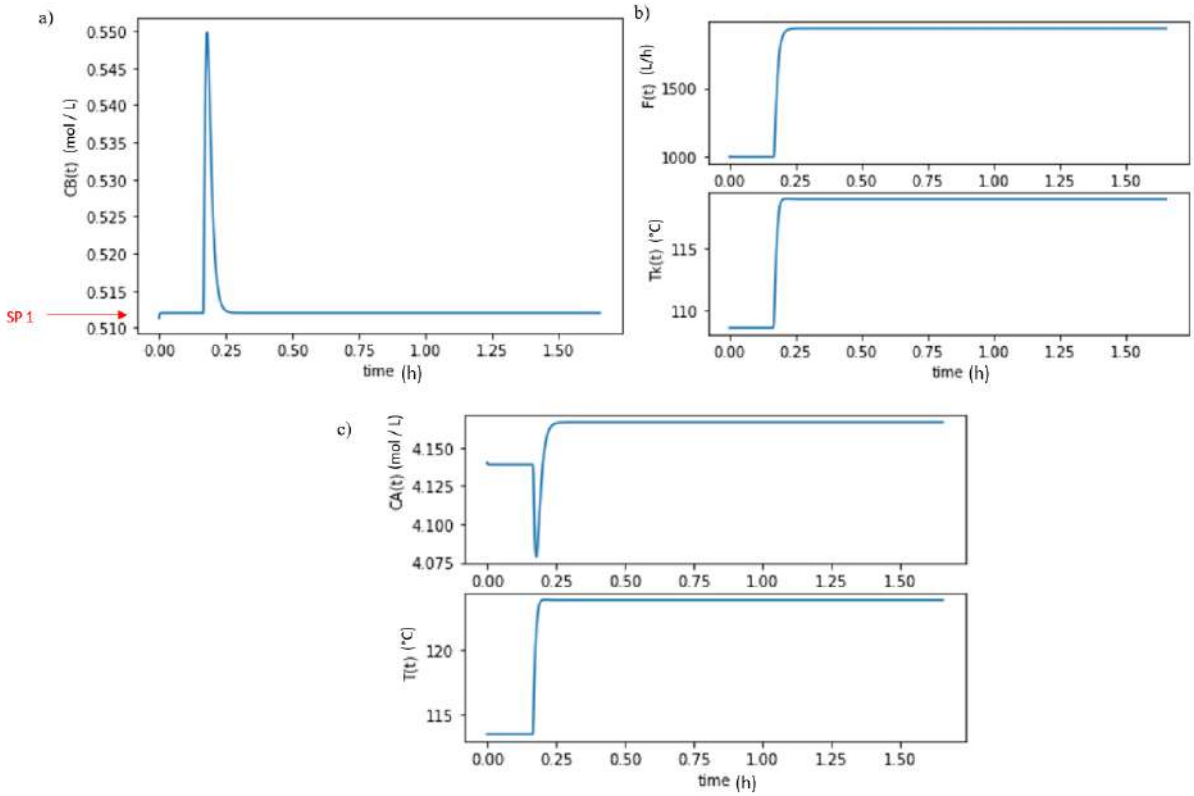
A função de transferência $G_c(s)$ foi passada para o domínio z a partir da substituição de diferenças retrógradas descrita na equação (2.26). O Δt foi calculado pela equação (2.27).

$$\Delta t \approx 0{,}2 * 0{,}008275 = 0{,}001655\ h \tag{4.11}$$

$$G_{c\ 1^a\ malha}(z) = -5{,}2937 * \left(\frac{1{,}2 - z^{-1}}{1 - z^{-1}}\right) \tag{4.12}$$

A equação obtida de $G_c(z)$ da primeira malha foi implementada no Python e os gráficos da Figura 4.3 foram plotados para avaliar o desempenho do controlador a um degrau de $20°C$ em T_0 em $t = 0{,}15h$. A variável controlada desta malha é o c_B, representada na Figura 4.3 (a), é possível perceber que o controlador é eficiente pois é capaz de fazer com que c_B retorne ao seu valor *set point*, representando na figura por SP 1, de forma rápida e sem oscilações.

Figura 4.3 –Simulação do controlador da primeira malha de controle em Python.



Fonte: Elaboração própria

4.1.2 Malha de controle da temperatura

O cálculo das funções de $G_v(s)$ e de $G_m(s)$ da segunda malha de controle foram descritos na seção 3.1.2 e os resultados obtidos estão descritos na Tabela 4.2.

Tabela 4.2 – Funções de transferência obtidas para a segunda malha de controle.

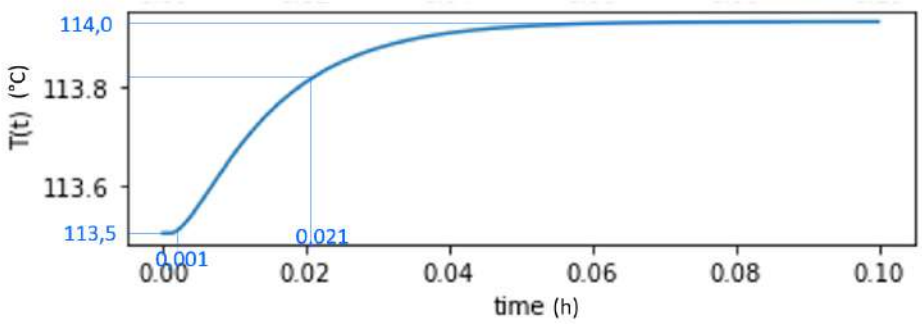
$G_{v\ 2ª\ malha}(s) = 531{,}25$	$G_{m\ 2ª\ malha}(s) = 0{,}133$

Fonte: Elaboração própria

O próximo passo foi a simulação em Python para calcular $G_p(s)$.

A Figura 4.4 mostra o gráfico da variação de T em função do tempo obtido para a primeira simulação, com um degrau de +1000 KJ/h. O novo $\dot{Q}_K$ estacionário, quando $t \to \infty$ é 114,0 °C. O ganho do processo está descrito na equação (4.13).

Figura 4.4 – $\dot{Q}_K$ x t com degrau positivo para a segunda malha de controle.



Fonte: Elaboração própria

$$K_{p\,1\,2^{a}\,malha} = \frac{114{,}0 - 113{,}5}{-3250 - (-4250)} = 0{,}0005\ °C/KJ/h \tag{4.13}$$

O novo valor do estado estacionário é $114{,}0\ °C$ e 63,2% deste valor equivale a $113{,}82\ °C$. Na Figura 4.4 é possível ver que esse valor acontece quando o tempo é igual a $0{,}021h$. Descontando o tempo em que o degrau aconteceu, obtêm-se o $\tau_{p\,1}$ descrito na equação (4.14).

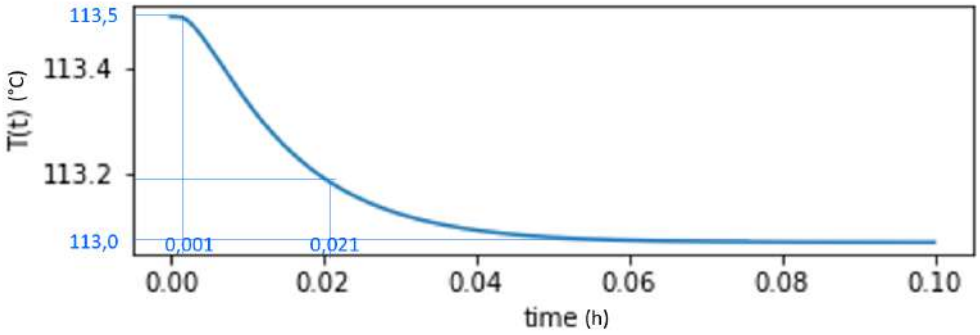
$$\tau_{p\,1\,2^{a}\,malha} = 0{,}021 - 0{,}001 = 0{,}020\ h \tag{4.14}$$

Assim, obteve-se o $G_{p\,1}\,(s)$, referente à primeira simulação da segunda malha, indicado na equação (4.15).

$$G_{p\,1\,2^{a}\,malha}(s) = \frac{0{,}0005}{0{,}02s + 1} \tag{4.15}$$

O mesmo desenvolvimento foi feito para a segunda simulação, onde aplicou-se um degrau negativo de -1000 KJ/h no mesmo $t\ = 0{,}001h$. O gráfico obtido está na Figura 4.5.

Figura 4.5 – $\dot{Q}_K$ x t com degrau negativo para a segunda malha de controle.



Fonte: Elaboração própria

O novo valor do estado estacionário de T é $113{,}0\,°C$. Assim, o ganho obtido para a segunda simulação está descrito na equação (4.16).

$$K_{p\ 2\ 2ª\ malha} = \frac{113{,}0 - 113{,}5}{-5250 - (-4250)} = 0{,}0005\ °C/KJ/h \tag{4.16}$$

Para o cálculo de τ_p calculou-se que 63,2% deste novo estado estacionário equivale a $113{,}18°C$, o que ocorre em $0{,}021\ h$. Descontando o tempo antes do distúrbio acontecer, temos o valor da constante de tempo descrita na equação (4.17).

$$\tau_{p\ 2\ 2ª\ malha} = 0{,}021 - 0{,}001 = 0{,}020h \tag{4.17}$$

Assim, obteve-se o $G_{p\ 2}(s)$, referente à segunda simulação da segunda malha de controle, indicado na equação (4.18).

$$G_{p\ 2\ 2ª\ malha}(s) = \frac{0{,}0005}{0{,}020s + 1} \tag{4.18}$$

O $G_p(s)$ foi calculado a partir das médias dos valores de K_p e τ_p obtidos. Como estes valores são iguais, obteve-se o descrito na equação (4.19).

$$G_{p\ 2ª\ malha}(s) = \frac{0{,}0005}{0{,}020s + 1} \tag{4.19}$$

Com os valores das funções de transferência calculados ($G_v(s), G_m(s)$ e $G_p(s)$), foi possível aplicar o método de Síntese Direta, de acordo com a equação (2.24) para calcular $G_c(s)$. O valor da constante de tempo do controlador τ_c foi calculado a partir da equação (2.25), sendo $\tau_{dom} = 0{,}020\ h$, que foi a maior constante de tempo encontrada para essa malha de controle. Por fim, obteve-se o $G_c(s)$ descrito em (4.20), referente a um controlador do tipo PI.

$$G_{c_{2ª\ malha}}(s) = 84{,}693 + 4235{,}232 * \frac{1}{s} \tag{4.20}$$

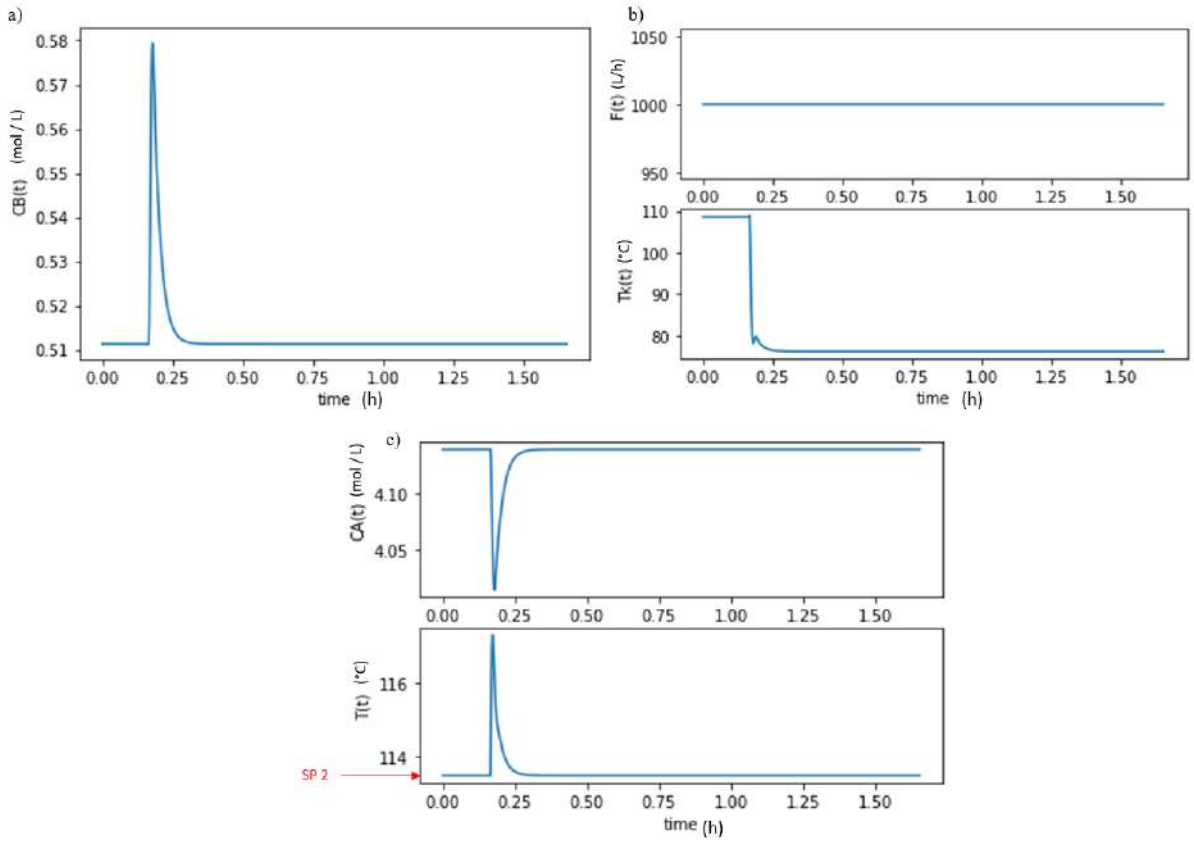
A função de transferência $G_c(s)$ foi passada para o domínio z a partir da substituição de diferenças retrógradas descrita na equação (2.26). O Δt foi calculado pela equação (2.27). Neste cálculo, utilizou-se $\tau_{dom} = 0{,}008275\ h$, que foi o mesmo utilizado para este mesmo cálculo na malha anterior. A menor constante de tempo dominante entre as duas malhas foi considerada para conseguir capturar a dinâmica da malha com a menor constante de tempo dominante.

$$\Delta t \approx 0{,}2 * 0{,}008275 = 0{,}001655\ h \tag{4.21}$$

$$G_{c\ 2ª\ malha}(z) = 84{,}693 * \left(\frac{1{,}083 - z^{-1}}{1 - z^{-1}}\right) \tag{4.22}$$

A equação obtida de $G_c(z)$ da segunda malha também foi implementada no Python e os gráficos da Figura 4.6 foram plotados para avaliar o desempenho do controlador a um mesmo distúrbio degrau de $20°C$ em T_0 em $t = 0{,}15\ h$. A variável controlada desta malha é a T, representada Figura 4.6 (c), é possível perceber que o controlador também é eficiente pois faz com que T retorne ao seu valor *set point*, representado no gráfico por SP 2, de forma rápida e sem oscilações.

Figura 4.6 – Simulação do controlador da segunda malha de controle em Python.

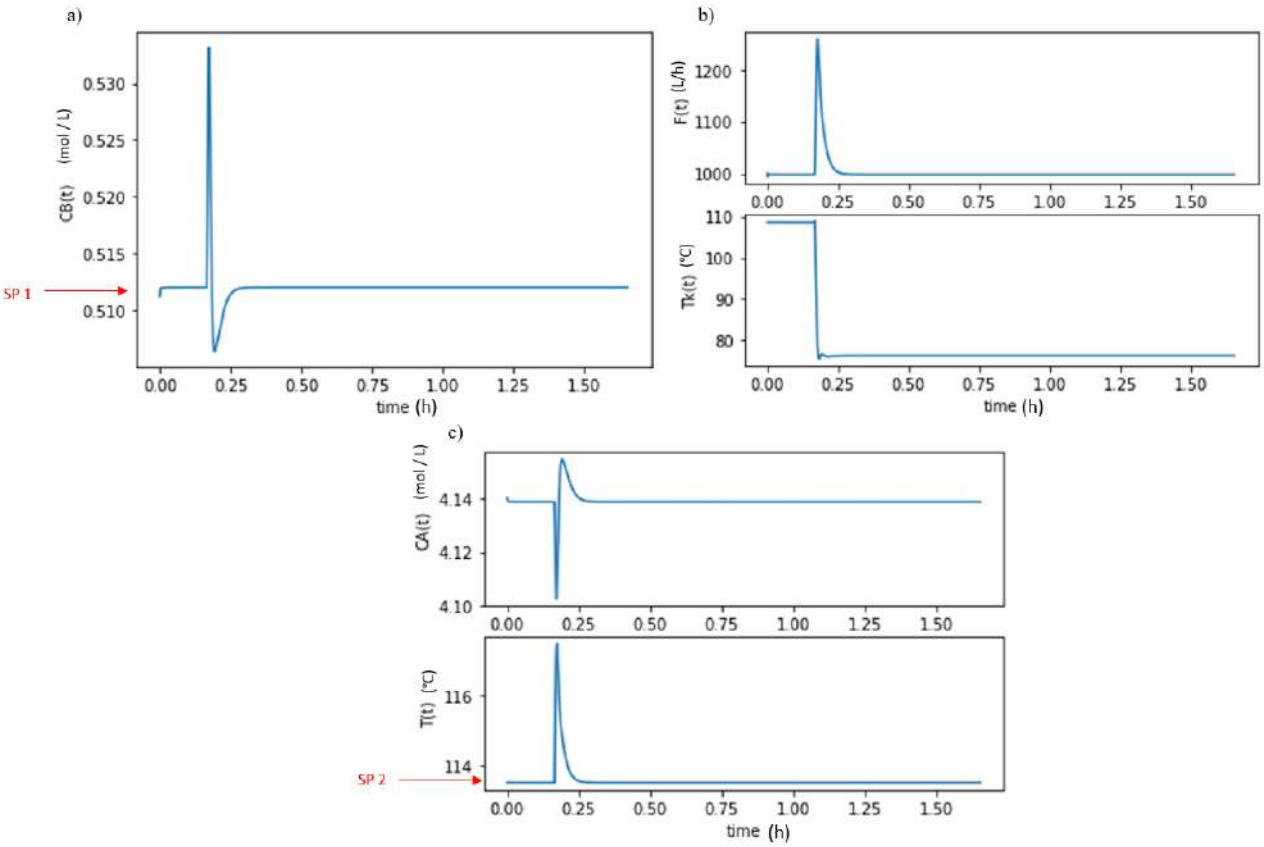


Fonte: Elaboração própria

4.1.3 Simulação em Python dos dois controladores

Os dois controladores foram implementados em Python e o mesmo distúrbio foi feito para avaliar se eles conseguiam atuar em conjunto. Os gráficos obtidos estão na Figura 4.7. O controle continuou sendo efetivo já que tanto c_B quanto T retornaram rapidamente aos seus valores iniciais desejados, representados no gráfico respectivamente por SP 1 e SP 2.

Figura 4.7– Simulação dos dois controladores em Python.



Fonte: Elaboração própria

4.2 AUTOMATIZAÇÃO NO CÁLCULO DE $G_p(s)$ E $G_c(s)$

4.2.1 Linearização dos balanços

Os balanços em função de variáveis de desvio estão descritos nas equações (4.23) a (4.26).

$$\frac{dc'_A}{dt} = \left[\frac{c_{A0} - \overline{c_A}}{V_R}\right] \mathrm{F}' + \left[-\frac{\bar{F}}{V_R} - \overline{k_1} - 2\overline{k_3}\overline{c_A}\right] c'_A + \left[\overline{k_1}\,\frac{E_1}{\bar{T}^2}\,\overline{c_A} + \overline{k_3}\,\frac{E_3}{\bar{T}^2}\,\overline{c_A}^2\right] \mathrm{T}' \qquad (4.23)$$

$$\frac{dc'_B}{dt} = \left[-\frac{\overline{c_B}}{V_R}\right] F' + \left[-\frac{\bar{F}}{V_R} - \overline{k_2}\right] c'_B + \left[\overline{k_1}\right] c'_A + \left[-\overline{k_1}\,\frac{E_1}{\bar{T}^2}\,\overline{c_A} + \overline{k_2}\,\frac{E_2}{\bar{T}^2}\,\overline{c_B}\right] T' \qquad (4.24)$$

$$\frac{dT'}{dt} = \left[\frac{T_0 - \bar{T}}{V_R}\right] F' + \left[-\frac{\bar{F}}{V_R} - \frac{k_w\, A_R}{\rho\, C_p\, V_R} + \frac{\overline{k_1}\, E_1\, \overline{c_A}\, \Delta H_R^{AB} + \overline{k_2}\, E_2\, \overline{c_B}\, \Delta H_R^{BC} + \overline{k_3}\, E_3\, \overline{c_A^2}\, \Delta H_R^{AD}}{\rho\, C_p\, \bar{T}^2}\right] T' + \left[\frac{k_w A_R}{\rho\, C_p\, V_R}\right] T_K' + \left[\frac{-\overline{k_1}\, \Delta H_R^{AB} - 2\, \overline{k_3}\, \overline{c_A}\, \Delta H_R^{AD}}{\rho\, C_p}\right] c_A' + \left[-\frac{\overline{k_2} \Delta H_R^{BC}}{\rho\, C_p}\right] c_B' \quad (4.25)$$

$$\frac{dT_K'}{dt} = \left[\frac{1}{m_K C_{p\,K}}\right] \dot{Q}_K' + \left[\frac{k_w\, A_R}{m_K\, C_{p\,K}}\right] T' + \left[-\frac{k_w\, A_R}{m_K\, C_{p\,K}}\right] T_K' \quad (4.26)$$

4.2.2 Síntese Direta no Python

A função de transferência do controlador $G_c(s)$ foi calculada a partir da Síntese Direta, cujas fórmulas foram descritas na seção 2.5 e estão resumidas abaixo.

$$G_c(s) = \frac{1}{G_v(s)\, G_p(s)\, G_m(s)} \frac{1}{\tau_c s} \quad (4.27)$$

Conforme já discutido, $G_v(s)$ e $G_m(s)$ são constantes para cada uma das malhas e $G_p(s)$ muda conforme o ponto de operação. Considerando τ_p constante, é possível escrever as equações (4.27) em função somente de K_p, que é calculado de forma automática no código em Python a partir do uso das matrizes descritas na seção anterior.

4.2.2.1 Malha de controle da concentração de B

Os parâmetros fixos da primeira malha de controle estão descritos na Tabela 4.3 abaixo.

Tabela 4.3– Parâmetros fixos da primeira malha de controle.

$G_{v\ 1ª\ malha}(s) = 125$	$G_{m\ 1ª\ malha}(s) = 10{,}667$	$\tau_{p\ 1ª\ malha} = 0{,}008275\ h$

Fonte: Elaboração própria

Manipulando a equação (4.27) e usando esses parâmetros fixos, obtêm-se a função de transferência do controlador PI da primeira malha de controle, em função de $K_{p\ 1ª\ malha\ de\ controle}$, descrita na equação (4.28).

$$G_{c\ 1ª\ malha\ de\ controle}(s) = \frac{0{,}00225}{K_{p\ 1ª\ malha\ de\ controle}} + \frac{0{,}272}{K_{p\ 1ª\ malha\ de\ controle}} * \frac{1}{s} \qquad (4.28)$$

4.2.2.2 Malha de controle da temperatura

Os parâmetros fixos da segunda malha de controle estão descritos na Tabela 4.4 abaixo.

Tabela 4.4 –Parâmetros fixos da segunda malha de controle.

$G_{v\ 2ª\ malha}(s) = 531{,}25$	$G_{m\ 2ª\ malha}(s) = 0{,}133$	$\tau_{p\ 2ª\ malha} = 0{,}020\ h$

Fonte: Elaboração própria

Manipulando a equação (4.27) e usando esses parâmetros fixos, obtêm-se a função de transferência do controlador PI da segunda malha de controle, em função de $K_{p\ 2ª\ malha\ de\ controle}$, descrita na equação (4.29).

$$G_{c\ 2ª\ malha\ de\ controle}(s) = \frac{0{,}0424}{K_{p\ 2ª\ malha\ de\ controle}} + \frac{2{,}119}{K_{p\ 2ª\ malha\ de\ controle}} * \frac{1}{s} \qquad (4.29)$$

4.3 CONTROLE ADAPTATIVO

O código em Python contém todas as informações: a matriz com os balanços automatizados, a síntese direta e a matriz com os parâmetros dos controladores das duas malhas armazenados.

As faixas de variação das variáveis agendadas são descritas na Tabela 3.2. Cada uma destas faixas foram discretizadas em 10 intervalos, de acordo com o descrito na Tabela 4.5.

Tabela 4.5 – Discretização das variáveis agendadas para o *Gain Scheduling*.

$F\ (L/h)$	$T_0\ (°C)$	$c_{A\,0}\ (mol/L)$	$\dot{Q_K}\ (KJ/h)$
$0 \leq F \leq 200$	$100 \leq T_0 \leq 105$	$4 \leq c_{A\,0} \leq 4{,}2$	$-8500 \leq \dot{Q_K} \leq -7650$
$200 \leq F \leq 400$	$105 \leq T_0 \leq 110$	$4{,}2 \leq c_{A\,0} \leq 4{,}4$	$-7650 \leq \dot{Q_K} \leq -6800$
$400 \leq F \leq 600$	$110 \leq T_0 \leq 115$	$4{,}4 \leq c_{A\,0} \leq 4{,}6$	$-6800 \leq \dot{Q_K} \leq -5950$
$600 \leq F \leq 800$	$115 \leq T_0 \leq 120$	$4{,}6 \leq c_{A\,0} \leq 4{,}8$	$-5950 \leq \dot{Q_K} \leq -5100$
$800 \leq F \leq 1000$	$120 \leq T_0 \leq 125$	$4{,}8 \leq c_{A\,0} \leq 5{,}0$	$-5100 \leq \dot{Q_K} \leq -4250$
$1000 \leq F \leq 1200$	$125 \leq T_0 \leq 130$	$5{,}0 \leq c_{A\,0} \leq 5{,}2$	$-4250 \leq \dot{Q_K} \leq -3400$
$1200 \leq F \leq 1400$	$130 \leq T_0 \leq 135$	$5{,}2 \leq c_{A\,0} \leq 5{,}4$	$-3400 \leq \dot{Q_K} \leq -2550$
$1400 \leq F \leq 1600$	$135 \leq T_0 \leq 140$	$5{,}4 \leq c_{A\,0} \leq 5{,}6$	$-2550 \leq \dot{Q_K} \leq -1700$
$1600 \leq F \leq 1800$	$140 \leq T_0 \leq 145$	$5{,}6 \leq c_{A\,0} \leq 5{,}8$	$-1700 \leq \dot{Q_K} \leq -850$
$1800 \leq F \leq 2000$	$145 \leq T_0 \leq 150$	$5{,}8 \leq c_{A\,0} \leq 6{,}0$	$-850 \leq \dot{Q_K} \leq 0$

Fonte: Elaboração própria

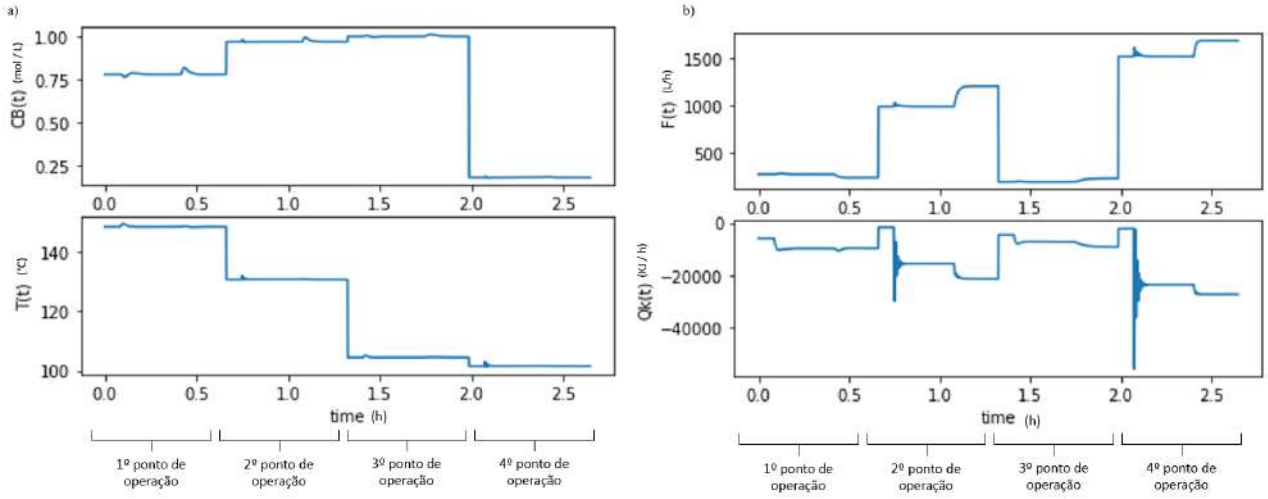
Para ver se o controle estava sendo efetivo, os gráficos das Figura 4.8 foram plotados. Ambos os gráficos possuem 4 simulações, com 4 mudanças do ponto operacional do processo. Em cada uma dessa simulações, os valores das variáveis agendadas foram escolhidos de forma aleatória a partir de uma função que garante que todos os pontos dentro da faixa de variação possível possuem a mesma probabilidade de serem selecionados. Os valores das variáveis agendadas de cada simulação estão descritos na Tabela 4.6. A cada uma dessas mudanças, muda-se os parâmetros do controlador, definidos de acordo com a matriz do *Gain Scheduling*, e muda-se também o valor desejado para a variável controlada. Além disso, dois distúrbios são aplicados em cada um dos diferentes pontos. Primeiro, aplica-se um degrau positivo de $5°C$ em T_0 e, depois, aplica-se um degrau positivo de $0{,}5\ mol/L$ em $c_{A\,0}$. O objetivo destes dois distúrbios é ver se os controladores são realmente eficazes nas diferentes condições operacionais. Cada uma das simulações possui 400 passos, o que representa um tempo de 0,625h. O primeiro distúrbio acontece no passo número 50 e o segundo distúrbio no passo 250. O tempo em que o degrau é dado equivale ao número de passos vezes o tempo de amostragem, ou seja, o primeiro distúrbio acontece em $t = 0{,}078\ h$ e o segundo em $t = 0{,}39\ h$.

Tabela 4.6 –Variáveis agendadas de cada uma das 4 simulações.

1ª simulação	2ª simulação	3ª simulação	4ª simulação
$F = 268{,}73\ L$	$F = 990{,}87\ L$	$F = 187{,}72\ L$	$F = 1524{,}56\ L$
$T_0 = 142{,}37\ °C$	$T_0 = 122{,}48\ °C$	$T_0 = 101{,}42\ °C$	$T_0 = 100{,}11\ °C$
$c_{A\,0} = 5{,}53\ mol/L$	$c_{A\,0} = 5{,}30\ mol/L$	$c_{A\,0} = 5{,}67\ mol/L$	$c_{A\,0} = 4{,}89\ mol/L$
$\dot{Q}_K = -5959{,}45\ KJ/h$	$\dot{Q}_K = -1690{,}21\ KJ/h$	$\dot{Q}_K = -4537{,}86\ KJ/h$	$\dot{Q}_K = -2227{,}68 KJ/h$

Fonte: Elaboração própria

Figura 4.8– Simulações de 4 pontos de operação diferentes com aplicação de degrau de distúrbio em T_0 e $c_{A\,0}$.



Fonte: Elaboração própria

Os dois gráficos da Figura 4.8 (a) mostram as duas variáveis controladas, c_B que é a variável controlada da primeira malha e T que é a da segunda malha. Os dois gráficos da Figura 4.8 (b) representam as duas variáveis manipuladas, F é a da primeira malha e $\dot{Q}_k$ é a da segunda. Os pontos de operação estão especificados na imagem. É possível ver que o controle foi efetivo para todos os pontos analisados já que mesmo depois dos dois distúrbios, o controlador fez com que as variáveis controladas retornassem rapidamente ao seu valor desejado inicial, sem muita oscilação.

4.3.1 CONTROLE ADAPTATIVO X FIXO

Depois de ver que o controle funcionou nos quatro pontos de operação escolhidos de forma aleatória e com os dois distúrbios analisados, o próximo objetivo foi comparar o controlador adaptativo desenvolvido com um controlador fixo.

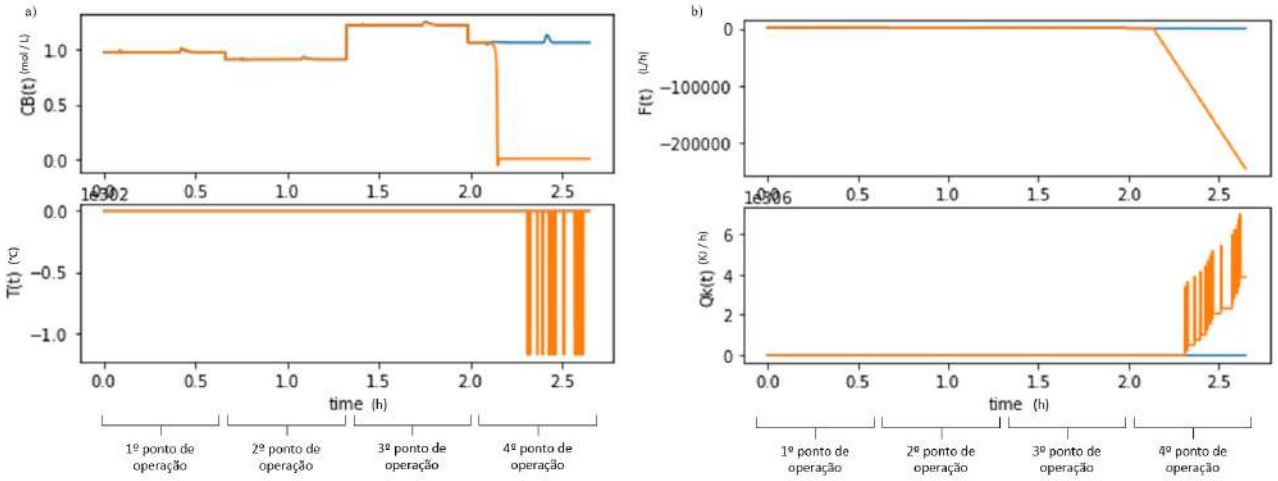
Os controladores fixos de cada uma das malhas utilizados foram aqueles que foram calculados no tópico 4.1, em torno de um ponto de referência. Esta comparação foi feita para quatro pontos de operação diferentes, que foram definidos da mesma forma já descrita no tópico anterior, ou seja, escolhendo de forma aleatória um valor dentro da faixa de variação de cada uma das variáveis agendadas. Os mesmos distúrbios foram aplicados: um degrau positivo de $5°C$ em T_0 e, depois, um degrau positivo de $0{,}5\ mol/L$ em $c_{A\,0}$. O resultado desta comparação está na Figura 4.9 e os valores das variáveis agendadas está descrito na Tabela 4.7.

Tabela 4.7 –Variáveis agendadas da simulação comparativa.

1ª simulação	2ª simulação	3ª simulação	4ª simulação
$F = 1688{,}84\ L$	$F = 1022{,}55\ L$	$F = 953{,}20\ L$	$F = 563{,}68\ L$
$T_0 = 137{,}90\ °C$	$T_0 = 120{,}25\ °C$	$T_0 = 129{,}17°C$	$T_0 = 137{,}79\ °C$
$c_{A\,0} = 4{,}84\ mol/L$	$c_{A\,0} = 5.57\ mol/L$	$c_{A\,0} = 5{,}82\ mol/L$	$c_{A\,0} = 5{,}24\ mol/L$
$\dot{Q_K} = -5928{,}67 KJ/h$	$\dot{Q_K} = -5573{,}50 KJ/h$	$\dot{Q_K} = -3962{,}51 KJ/h$	$\dot{Q_K} = -5995{,}95 KJ/h$

Fonte: Elaboração própria

Figura 4.9– Simulações comparativa entre o controlador adaptativo (azul) e o controlador fixo (laranja).



Fonte: Elaboração própria

Na Figura 4.9 em laranja temos o controle realizado pelo controlador fixo e em azul o controlador adaptativo. É possível perceber que o controlador fixo realizou um bom controle para os três primeiros pontos de operação, assim como o controlador adaptativo. Mas, ao olhar para o quarto ponto de operação, é possível ver que o controlador fixo não foi capaz de manter a variável controlada no seu valor *set point*, enquanto o controlador adaptativo foi eficiente.

Desta forma, pode-se concluir que o controle adaptativo desenvolvido no trabalho funciona bem para diferentes distúrbios e diferentes pontos de operação, além de ser uma boa solução para a reação de Van de Vusse.

4.3.2 CONTROLE ADAPTATIVO EM PONTOS EXTREMOS

Por fim, o último objetivo foi ver se o controlador adaptativo desenvolvido também funcionava bem em pontos extremos de F e $\dot{Q_K}$. Para isto, 8 novas simulações foram feitas, aplicando os mesmos distúrbios já descritos em T_0 e c_{A_0}. Primeiro, valores de F extremos foram testados, primeiro um próximo ao limite inferior e outro próximo ao limite superior, mantendo os valores de T, c_A e $\dot{Q_K}$ nos seus valores de estado estacionário, descritos Tabela 3.1. Depois, o mesmo processo foi feito, mas avaliando valores extremos de $\dot{Q_K}$, mantendo os outros nos seus valores de estado estacionário. Por fim, avaliou-se também a combinação de valores extremos de F e de $\dot{Q_K}$.

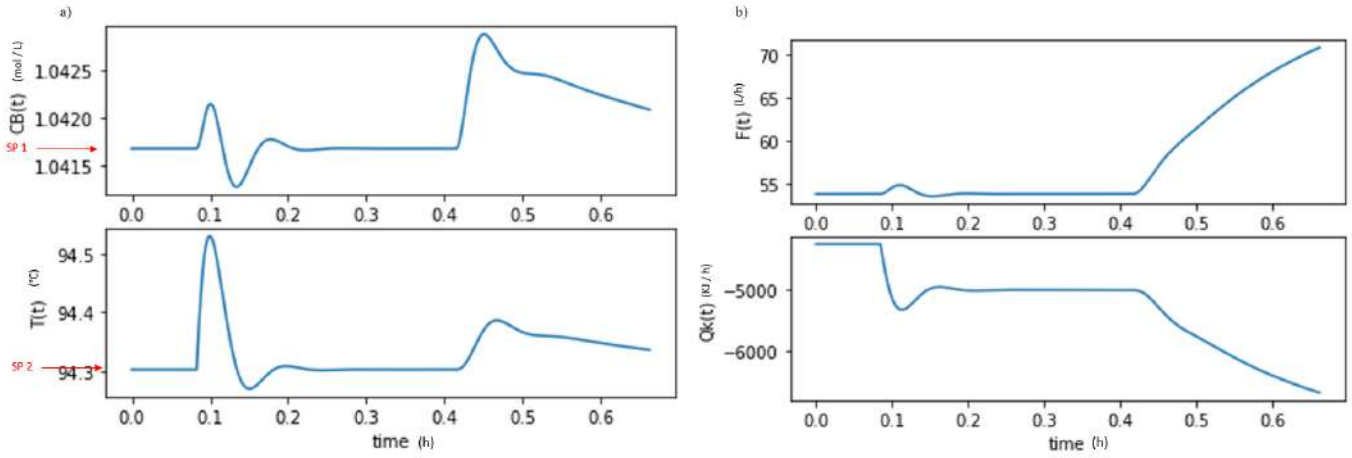
Todos os valores utilizados estão descritos na Tabela 4.8. Para definir os valores próximos aos limites, a mesma função no Python foi usada para escolher algum valor aleatoriamente dentro do intervalo definido. Para os valores próximos ao limite inferior de F, definiu-se a faixa de variação $0 \leq F \leq 400\ L$, enquanto para o limite superior definiu-se $1600 \leq F \leq 2000\ L$. Para o limite inferior de $\dot{Q_K}$ definiu-se $-8500 \leq \dot{Q_K} \leq -6800\ KJ/h$ e para o limite superior $-1700 \leq \dot{Q_K} \leq 0\ KJ/h$.

Tabela 4.8 – Variáveis agendadas das simulações com variações extremas de F e $\dot{Q}_K$.

1ª simulação	2ª simulação	3ª simulação	4ª simulação
$F_{\substack{limite \\ inferior}} = 53{,}75\ L$	$F_{\substack{limite \\ superior}} = 1653{,}75$ L	$F_{SS} = 1000$ L	$F_{SS} = 1000$ L
$T_{SS} = 110\ °C$	$T_{SS} = 110\ °C$	$T_{SS} = 110\ °C$	$T_{SS} = 110\ °C$
$c_{A\,SS} = 5{,}12\ mol/L$	$c_{A\,SS} = 5{,}12\ mol/L$	$c_{A\,SS} = 5{,}12\ mol/L$	$c_{A\,SS} = 5{,}12\ mol/L$
$\dot{Q}_{K\,SS} = -4250\ KJ/h$	$\dot{Q}_{K\,SS} = -4250\ KJ/h$	$\dot{Q}_{K\ \substack{limite \\ inferior}} = -8271{,}58\ KJ/h$	$\dot{Q}_{K\ \substack{limite \\ suprior}} = -1471{,}58\ KJ/h$
5ª simulação	**6ª simulação**	**7ª simulação**	**8ª simulação**
$F_{\substack{limite \\ inferior}} = 382{,}41\ L$	$F_{\substack{limite \\ superior}} = 1982{,}41$ L	$F_{\substack{limite \\ inferior}} = 382{,}41$ L	$F_{\substack{limite \\ superior}} = 1982{,}41$ L
$T_{SS} = 110\ °C$	$T_{SS} = 110\ °C$	$T_{SS} = 110\ °C$	$T_{SS} = 110\ °C$
$c_{A\,SS} = 5{,}12\ mol/L$	$c_{A\,SS} = 5{,}12\ mol/L$	$c_{A\,SS} = 5{,}12\ mol/L$	$c_{A\,SS} = 5{,}12\ mol/L$
$\dot{Q}_{K\ \substack{limite \\ inferior}} = -6888{,}69\ KJ/h$	$\dot{Q}_{K\ \substack{limite \\ inferior}} = -6888{,}69\ KJ/h$	$\dot{Q}_{K\ \substack{limite \\ superior}} = -88{,}69\ KJ/h$	$\dot{Q}_{K\ \substack{limte \\ superior}} = -88{,}69\ KJ/h$

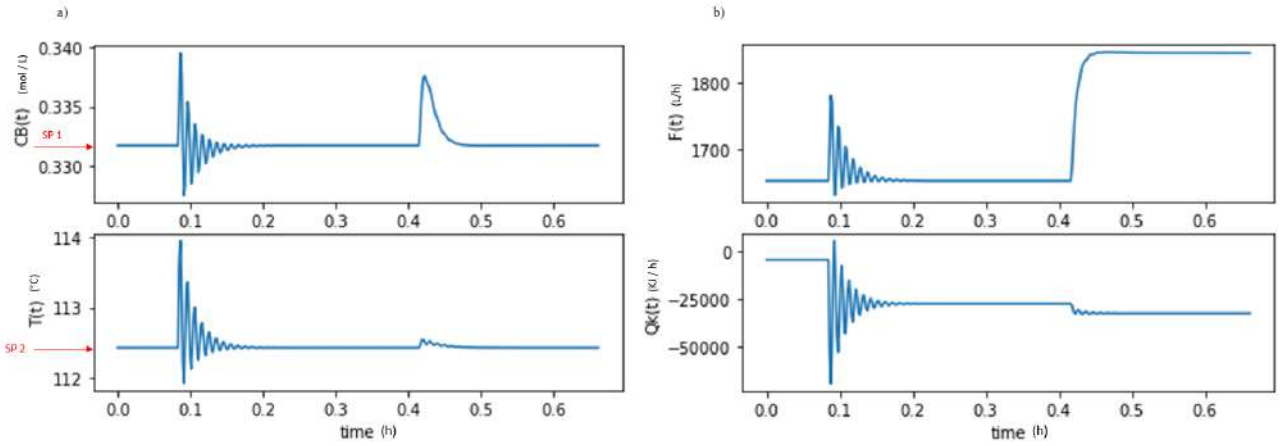
Fonte: Elaboração própria

Figura 4.10 – 1ª Simulação de variações extremas: F dentro do seu limite inferior.



Fonte: Elaboração própria

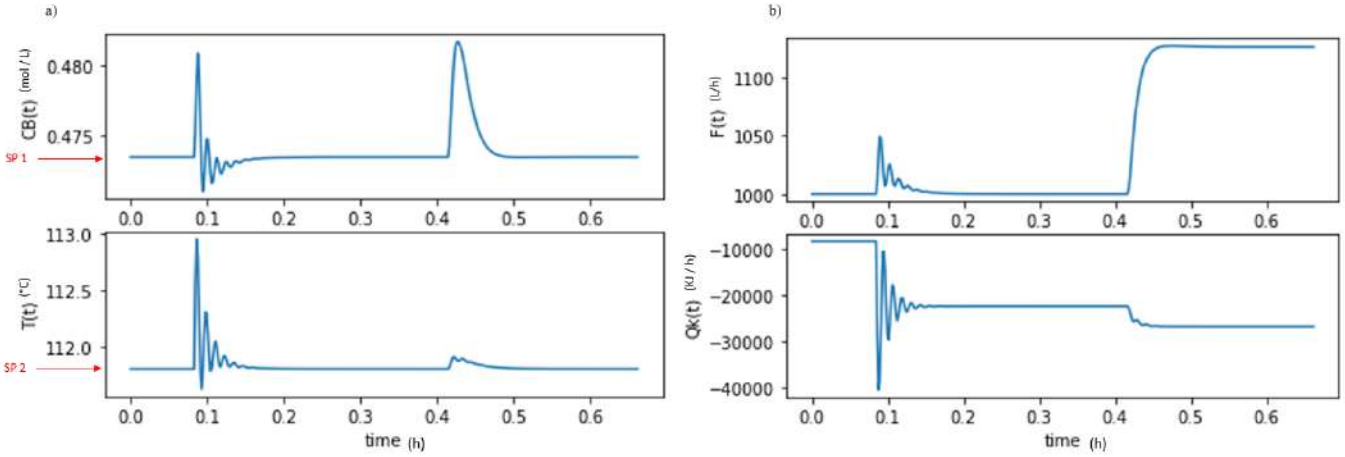
Figura 4.11 – 2ª Simulação de variações extremas: F dentro do seu limite superior.



Fonte: Elaboração própria

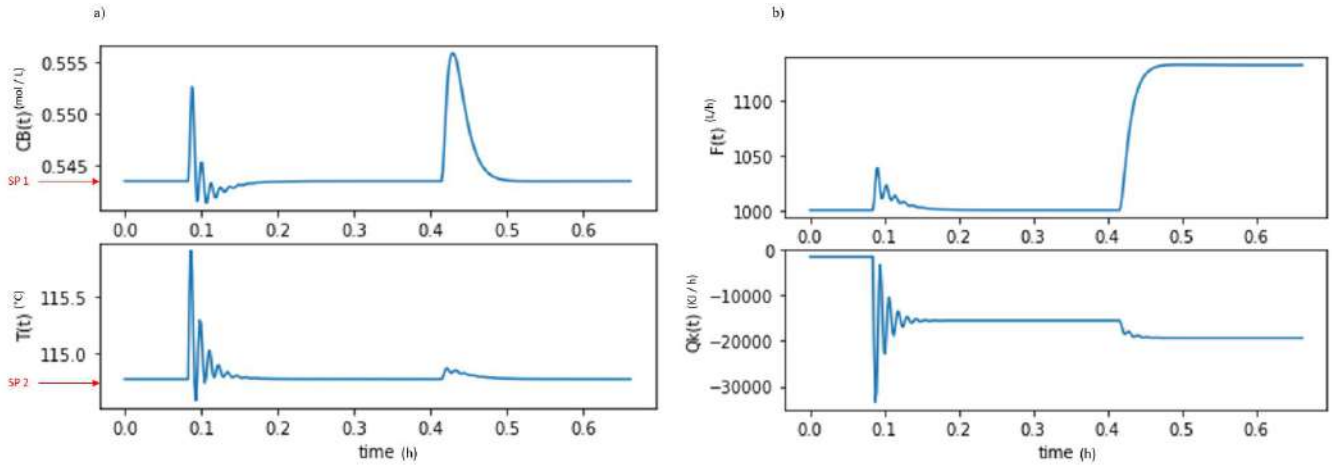
Analisando primeiro a variação de F dentro de seus valores extremos, é possível ver que o controle obtido para o limite superior foi melhor do que o obtido para o limite inferior. Na Figura 4.11, é possível ver que, apesar da resposta ter sido oscilatória, o controlador foi capaz de obter o *set point* para as duas variáveis controladas nos dois distúrbios analisados. Já na Figura 4.10, que mostra o limite inferior de F, percebe-se que o controlador foi capaz de retornar as variáveis controladas ao seu valor desejado após o primeiro distúrbio, de $+\ 5°C$ em T_0, mas não obteve uma resposta adequada para o segundo distúrbio, $+0{,}5\ mol/L$ em $c_{A\,0}$. O controle é capaz de estabilizar para os dois *set points*, mas, de forma bem mais lenta que não aparece no período de tempo analisado para o segundo distúrbio. Esse resultado parece mostrar que a hipótese de que as constantes de tempo mudavam pouco com o ponto de operação pode não ser válida.

Figura 4.12 – 3ª Simulação de variações extremas: $\dot{Q}_K$ dentro do seu limite inferior.



Fonte: Elaboração própria

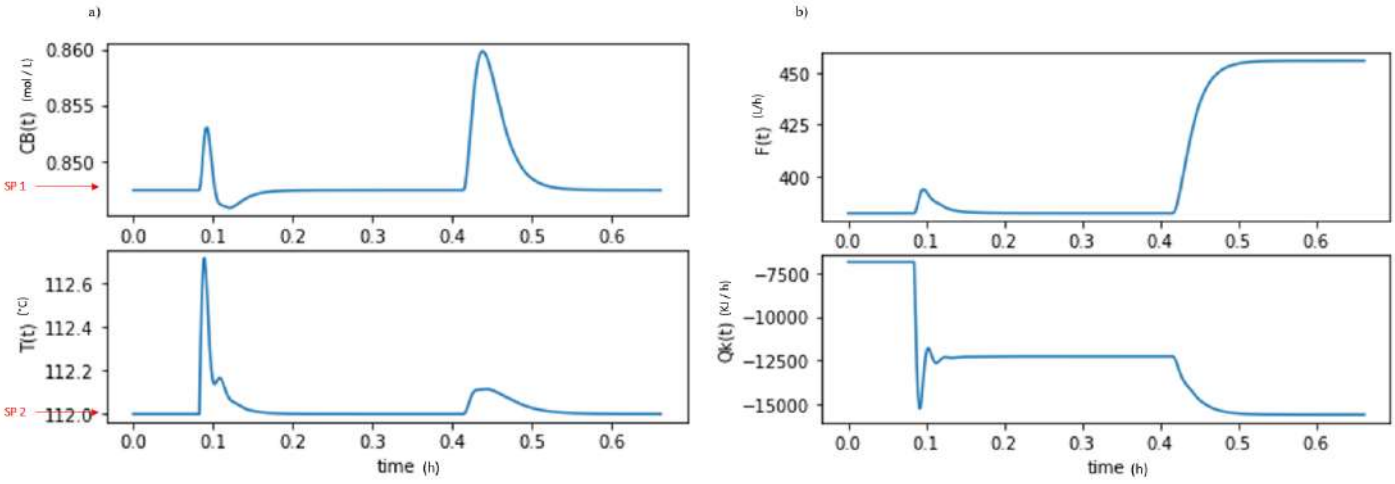
Figura 4.13– 4ª Simulação de variações extremas: $\dot{Q_K}$ dentro do seu limite superior.



Fonte: Elaboração própria

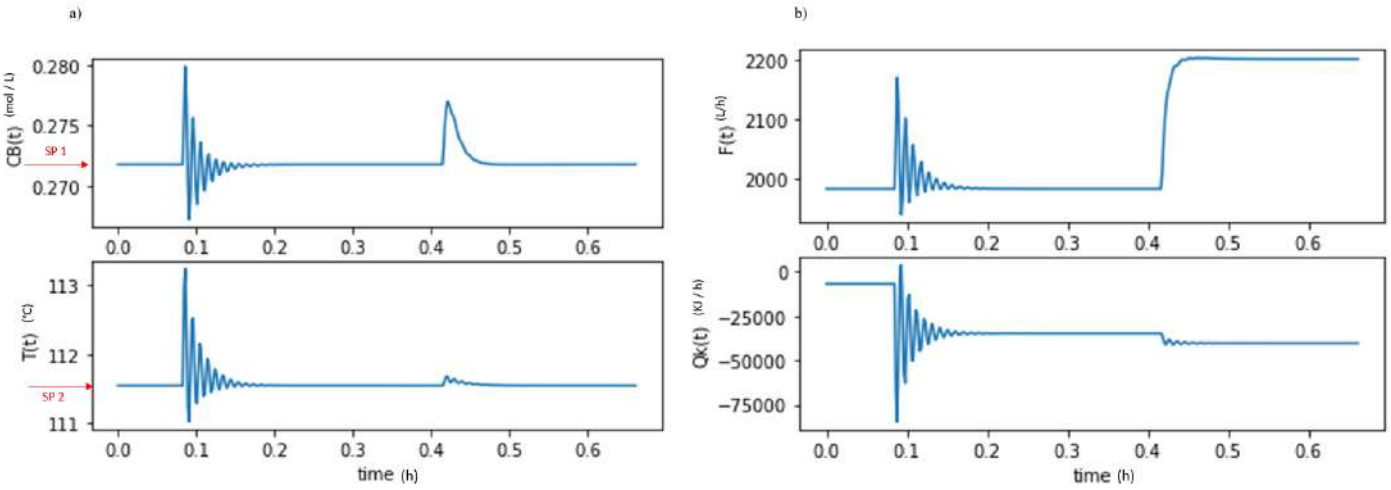
Analisando a variação de $\dot{Q_K}$ dentro de seus valores extremos, é possível ver que o controle obtido foi bastante parecido em ambos os limites: a resposta foi mais oscilatória do que o adequado, mas o controlador conseguiu retornar os valores das variáveis controladas para o seu valor desejado. Além disso, percebe-se que a resposta foi mais oscilatória para o primeiro distúrbio, em T_0, do que para o segundo, em $c_{A\,0}$.

Figura 4.14– 5ª Simulação de variações extremas: F e $\dot{Q_K}$ dentro dos seus limites inferiores.



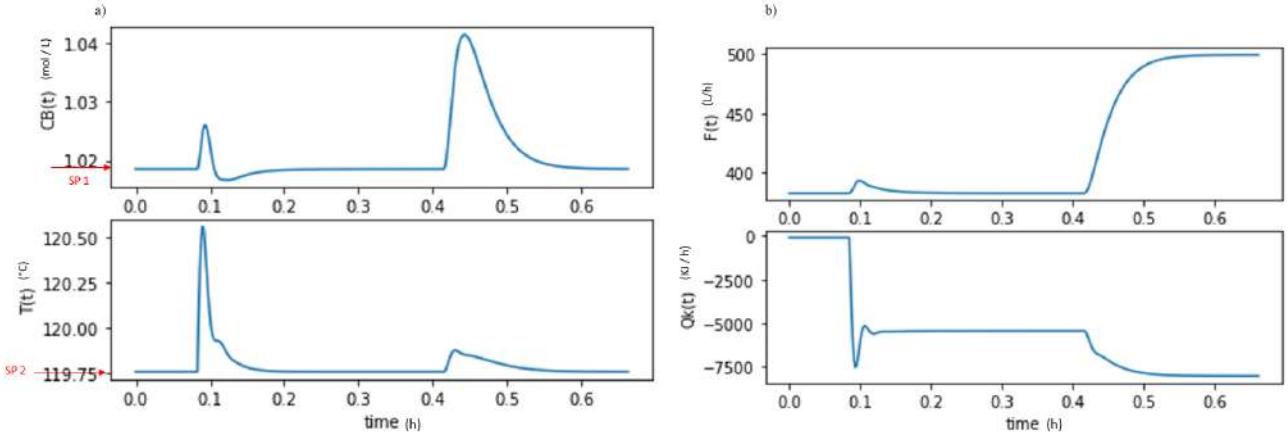
Fonte: Elaboração própria

Figura 4.15 – 6ª Simulação de variações extremas: F dentro do seu limite superior e $\dot{Q_K}$ dentro do seu limite inferior.



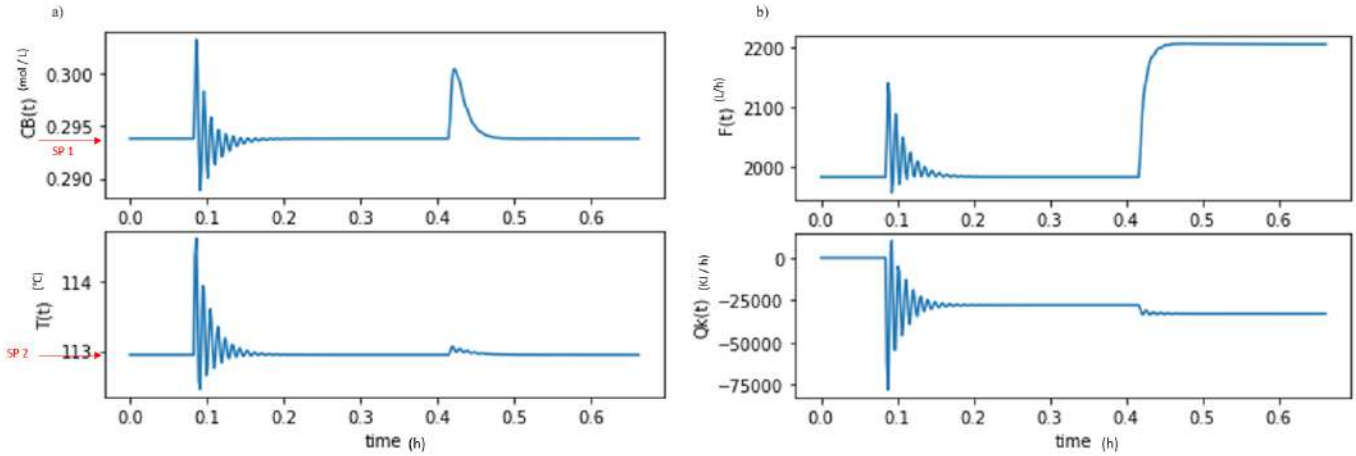
Fonte: Elaboração própria

Figura 4.16 – 7ª Simulação de variações extremas: F dentro do seu limite inferior e $\dot{Q_K}$ dentro do seu limite superior.



Fonte: Elaboração própria

Figura 4.17– 8ª Simulação de variações extremas: F e $\dot{Q_K}$ dentro dos seu limites superiores.



Fonte: Elaboração própria

Analisando a combinação dos extremos de F e $\dot{Q_K}$, é possível ver que os controles obtidos estão mais oscilatórios que o ideal, mas ainda conseguem fazer com que as variáveis controladas retornem ao seu valor desejado.

Neste trabalho, a Síntese Direta foi utilizada para obter os parâmetros do controlador, obtendo um controlador do tipo PI. Para que este método fosse utilizado, a função de transferência do processo G_p foi aproximada por uma função de transferência de primeira ordem, aonde somente o K_p mudava com os diferentes pontos de operação, ou seja, considerou-se τ_p constante. Esta aproximação foi capaz de obter um controlador que funcionou bem em grande parte dos pontos, mas que se funcionava melhor para pontos intermediários de F e $\dot{Q_K}$.

5 CONCLUSÕES

O presente trabalho teve como objetivo principal aplicar a metodologia do controle adaptativo para o reator de Van de Vusse, que é um benchmark para o design e controle de processos complexos pela sua dinâmica altamente não linear e por ser um processo cuja função de transferência não é de primeira ordem.

Depois de algumas simulações, descritas no Capítulo 4, conclui-se que o controlador obtido foi efetivo para diferentes pontos de operação e dois distúrbios diferentes, ou seja, ele foi capaz de fazer com que ambas as variáveis controladas retornassem aos seus valores desejados.

Além disso, comparando o controlador adaptativo encontrado e o controlador fixo também obtido neste trabalho, concluiu-se que o controlador adaptativo teve boa performance em mais pontos de operação do que o fixo.

Em relação às simulações feitas em pontos extremos de F e de $\dot{Q}_k$, percebeu-se que o controlador adaptativo obtido também foi capaz de fazer com que as variáveis controladas retornassem ao seus valores de *set point*, mas apresentou uma resposta mais oscilatória ou lenta do que o ideal.

Por fim, conclui-se que o controle adaptativo desenvolvido neste trabalho foi sim efetivo e satisfatório, mas funciona melhor quando as variáveis agendadas F e de $\dot{Q}_K$ encontram-se em valores intermediários e não tão próximos dos seus limites.

6 TRABALHOS FUTUROS

Neste trabalho, a função de transferência do processo G_p foi aproximada como uma função de transferência de primeira ordem para possibilitar o uso da Síntese Direta. Esta consideração foi feita com base na análise da resposta do processo a um degrau, que foi muito parecida com a resposta obtida para processos de primeira ordem. O controlador do tipo PI obtido pela Síntese Direta funcionou bem em grande parte dos pontos. Mas, uma sugestão para próximos trabalhos é o uso de outro método de sintonia que seja capaz de obter também um modo derivativo. Como o método integral diminui a estabilidade do sistema e tende a produzir respostas oscilatórias, o uso de uma função derivativa poderia ser capaz de compensar esta tendência desestabilizadora e reduzir a oscilação. Assim, o controle seria mais eficiente em todos os pontos de operação, inclusive nos pontos extremos aonde foi possível ver uma resposta mais oscilatória do que o desejado.

A constante de tempo do processo também foi considerada constante, ou seja, somente K_p variava nos diferentes pontos de operação. Ao fazer esta hipótese, informações sobre a dinâmica do processo foram perdidas, o que pode ter relação com as respostas lentas obtidas nos pontos extremos de vazão e taxa de refrigeração. Desta forma, outra sugestão de melhoria do trabalho é a estimação de τ_p de forma automática para todos os pontos de operação. Assim, seria possível calcular esta variável para todos os pontos de operação e a aproximação de que τ_p é constante não seria necessária. Ao ter mais informações sobre a dinâmica do processo, pode ser possível obter respostas mais rápidas.

Uma das considerações do trabalho foi a descrita na equação (2.25), que diz que τ_c é um terço de e τ_{dom}, o que significa dizer que a resposta do sistema em malha fechada é três vezes mais rápida do que a resposta em malha aberta. Para trabalhos futuros, isto pode ser investigado, fazendo uma análise de sensibilidade deste parâmetro. Sabe-se que valores mais altos de e τ_c resultam em um controle menos agressivo, o que pode ser interessante para melhorar o controle nos pontos extremos onde observou-se uma resposta oscilatória.

7 REFERÊNCIAS

ÅSTRÖM, K. J.; HÄGGLUND, T. **Pid Controllers: Theory, Design and Tuning**. 2nd UK ed. edição ed. Research Triangle Park, N.C: ISA, 1995.

BABATUNDE, A. O.; OGUNNAIKE, W.; HARMON, W. R. Process Dynamics, Modelling, and Control. Babatunde A. Ogunnaike, W. HArmon Ray. Oxford University Press, Oxford 1995, 1260 Seotem. Zahlr. Abb. und Tab., £ 65.-, ISBN 0-19-509119-1. **Chemie Ingenieur Technik**, v. 68, n. 6, p. 730–730, 1996.

FREITAS, G. C.; BEDOR, N. B. A.; CAPRON, B. D. O. Estudo de identificação de um reator de Van de Vusse com Redes Neuronais Feedforward. **Estudo de identificação de um reator de Van de Vusse com Redes Neuronais Feedforward**, 2020.

KLATT, K.-U.; ENGELL, S. Gain-scheduling trajectory control of a continuous stirred tank reactor. **Computers & Chemical Engineering**, v. 22, n. 4, p. 491–502, 1 jan. 1998.

LANDAU, I. D. et al. **Adaptive Control: Algorithms, Analysis and Applications**. London: Springer London, 2011.

LUIZ, C. et al. Controle Adaptativo Versus Controle Fuzzy: Um Estudo de Caso em um Processo de Nivel. v. 8, 1 maio 1997.

SEBORG, D. E. et al. **Process Dynamics and Control**. 3 rd ed. Courier Westford: John Wiley & Sons, Inc., 2011.

SILVA, C. A. A. O MÉTODO GAIN SCHEDULING NO CONTROLE DA PRESSÃO NA PERFURAÇÃO DE POÇOS DE PETRÓLEO. 2016.

SMITH, C. A.; CORRIPIO, A. B. **Principles and practice of automatic process control**. 2nd ed ed. New York: J. Wiley, 1997.

TRIERWEILER, J. O. **A systematic approach to control structure design**. Als Ms. gedr ed. Aachen: Shaker, 1997.

VAN DE VUSSE, J. G. Plug-flow type reactor versus tank reactor. **Chemical Engineering Science**, v. 19, n. 12, p. 994–996, 1 dez. 1964.

VOJTESEK, J.; DOSTAL, P. Adaptative Control of Chemical Reactor. 2010.

APÊNDICE A – Códigos utilizados para a 1ª etapa da Metodologia: em torno de um ponto de referência

1. Encontrando $G_{p\ 1^{a}\ malha}$

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import math

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    V= 10 # L
    k10= 1.287E12 # h-1
    k20= 1.287E12 # h-1
    k30= 9.043E9 # L/molA.h
    mE1R= -9758.3 # K
    mE2R= -9758.3 # K
    mE3R= -8560.0 # K
    mdeltaHAB= -4.20 # kJ/molA
    mdeltaHBC= 11.0 # kJ/molB
    mdeltaHAD= 41.85 # kJ/molA
    rho= 0.9342 # kg/L
    Cp= 3.01 # kJ/kg.K
    Kw= 4032.0 # kJ/h.K.m2
    Ar= 0.215 # m2
    mk= 5.0 # kg
    Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
        1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 ) )*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T) +Kw*Ar/(rho*Cp*V)*(Tk-T),
        Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
```

```
    return dydt
def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 e deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
    # pontos no tempo
    t = np.linspace(0,deltaT)

    # resolvendo a ODE
    s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
    s_next = s_list[-1,:]

    return s_next

# Exemplo de simulação
# Condições iniciais
NB_STEPS = 500 #número de passos da simulação
F = 1000
Tin = 110
CAin = 5.1
Qk = -4250
s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
print(s)
deltaT = 0.0001 # h

# Iniciação da memória
Flist = [F]
CAlist = [s[0]]
CBlist = [s[1]]
Tlist = [s[2]]
Tklist = [s[3]]

for j in range(NB_STEPS):

    # degrau positivo em F
    if j == 10:
        F = 1100

    # degrau negativo em F
    #if j == 10:
      # F = 900
```

```
    s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

    # Atualização da memória
    Flist.append(F)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT, NB_STEPS + 1)

plt.figure(1)
plt.plot(t,CBlist)
plt.xlabel('time')
plt.ylabel('CB(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
plt.xlabel('time')
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Tklist)
plt.xlabel('time')
plt.ylabel('Tk(t)')

plt.figure(3)
plt.subplot(211)
plt.plot(t,CAlist)
plt.xlabel('time')
plt.ylabel('CA(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.xlabel('time')
plt.ylabel('T(t)')

plt.show()
```

2. Simulando $G_{c\ 1ª\ malha}$

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import math

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    V= 10 # L
    k10= 1.287E12 # h-1
    k20= 1.287E12 # h-1
    k30= 9.043E9 # L/molA.h
    mE1R= -9758.3 # K
    mE2R= -9758.3 # K
    mE3R= -8560.0 # K
    mdeltaHAB= -4.20 # kJ/molA
    mdeltaHBC= 11.0 # kJ/molB
    mdeltaHAD= 41.85 # kJ/molA
    rho= 0.9342 # kg/L
    Cp= 3.01 # kJ/kg.K
    Kw= 4032.0 # kJ/h.K.m2
    Ar= 0.215 # m2
    mk= 5.0 # kg
    Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
        1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 ) )*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T) +Kw*Ar/(rho*Cp*V)*(Tk-T),
        Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
    return dydt
```

```
def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 e deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
    # pontos no tempo
    t = np.linspace(0,deltaT)

    # resolvendo a ODE
    s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
    s_next = s_list[-1,:]

    return s_next

# Exemplo de simulação
# Condições iniciais
NB_STEPS = 1000 #número de passos da simulação
F = 1000
Tin = 110
CAin = 5.1
Qk = -4250
s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
print(s)
deltaT = 0.001655 # h

#setpoint para Cb
CB_SP=0.512

#parâmetros do modelo
Km = 10.667
Kv = 125

#parâmetros do controlador PI
Kc=-5.2937
a0=1.2

# Iniciação da memória
Flist = [F]
CAlist = [s[0]]
CBlist = [s[1]]
Tlist = [s[2]]
Tklist = [s[3]]
```

```
e_ant = 0 %Km*(CB_SP-s[1])
p_ant = 0

for j in range(NB_STEPS):

    #degrau em Tin
    if j == 100:
        Tin = 120

    #e(t)
    e_atual=Km*(CB_SP-CBlist[-1])

    #sinal de controle
    p_atual = p_ant + Kc*a0*e_atual - Kc*e_ant
    deltaF=Kv*(p_atual-p_ant)
    F=Flist[-1]+deltaF
    e_ant=e_atual
    p_ant=p_atual

    s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

    # atualização da memória
    Flist.append(F)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT, NB_STEPS + 1)

plt.figure(1)
plt.plot(t,CBlist)
plt.xlabel('time')
plt.ylabel('CB(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
```

```
plt.xlabel('time')
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Tklist)
plt.xlabel('time')
plt.ylabel('Tk(t)')

plt.figure(3)
plt.subplot(211)
plt.plot(t,CAlist)
plt.xlabel('time')
plt.ylabel('CA(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.xlabel('time')
plt.ylabel('T(t)')

plt.show()
```

3. Encontrando $G_{p\ 2^{a}\ malha}$

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import math

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    V= 10 # L
    k10= 1.287E12 # h-1
    k20= 1.287E12 # h-1
    k30= 9.043E9 # L/molA.h
    mE1R= -9758.3 # K
    mE2R= -9758.3 # K
    mE3R= -8560.0 # K
    mdeltaHAB= -4.20 # kJ/molA
```

```
    mdeltaHBC= 11.0 # kJ/molB
    mdeltaHAD= 41.85 # kJ/molA
    rho= 0.9342 # kg/L
    Cp= 3.01 # kJ/kg.K
    Kw= 4032.0 # kJ/h.K.m2
    Ar= 0.215 # m2
    mk= 5.0 # kg
    Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
        1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 ) )*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T) +Kw*Ar/(rho*Cp*V)*(Tk-T),
        Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
    return dydt

def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 e deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
    # pontos no tempo
    t = np.linspace(0,deltaT)

    # resolvendo a ODE
    s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
    s_next = s_list[-1,:]

    return s_next

# Exemplo de simulação
# Condições iniciais
NB_STEPS = 1000 #número de passos da simulação
F = 1000
Tin = 110
CAin = 5.1
Qk = -4250
s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
print(s)
deltaT = 0.0001 # h
```

```
# Inicialização da memória
Flist = [F]
CAlist = [s[0]]
CBlist = [s[1]]
Tlist = [s[2]]
Tklist = [s[3]]

for j in range(NB_STEPS):
    #degrau positivo em Qk
    #if j == 10:
      #Qk = -3250

     #degrau negativo em Qk
    if j == 10:
       Qk = -5250
    s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

    # atualização da memória
    Flist.append(F)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT, NB_STEPS + 1)

plt.figure(1)
plt.plot(t,CBlist)
plt.xlabel('time')
plt.ylabel('CB(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
plt.xlabel('time')
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Tklist)
```

```
plt.xlabel('time')
plt.ylabel('Tk(t)')

plt.figure(3)
plt.subplot(211)
plt.plot(t,CAlist)
plt.xlabel('time')
plt.ylabel('CA(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.xlabel('time')
plt.ylabel('T(t)')

plt.show()
```

4. Simulando $G_{c\ 2ª\ malha}$

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import math

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    V= 10 # L
    k10= 1.287E12 # h-1
    k20= 1.287E12 # h-1
    k30= 9.043E9 # L/molA.h
    mE1R= -9758.3 # K
    mE2R= -9758.3 # K
    mE3R= -8560.0 # K
    mdeltaHAB= -4.20 # kJ/molA
    mdeltaHBC= 11.0 # kJ/molB
    mdeltaHAD= 41.85 # kJ/molA
    rho= 0.9342 # kg/L
    Cp= 3.01 # kJ/kg.K
```

```
    Kw= 4032.0 # kJ/h.K.m2
    Ar= 0.215 # m2
    mk= 5.0 # kg
    Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
        1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 )
)*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T)
+Kw*Ar/(rho*Cp*V)*(Tk-T),
        Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
    return dydt
def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 e deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
    # pontos no tempo
    t = np.linspace(0,deltaT)

    # resolvendo a ODE
    s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
    s_next = s_list[-1,:]

    return s_next

# Exemplo de simulação
# Condições iniciais
NB_STEPS = 1000 #número de passos da simulação
F = 1000
Tin = 110
CAin = 5.1
Qk = -4250
s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estados iniciais fixos
print(s)
deltaT = 0.001655 # h

#setpoint para T
T_SP=113.50 #°C

#parâmetros do modelo
```

```
Km = 0.1333
Kv = - 531.25

#parâmetros do controlador PI
Kc=-84.8
a0=1.083

# Inicialização da memória
Flist = [F]
Qklist = [Qk]
CAlist = [s[0]]
CBlist = [s[1]]
Tlist = [s[2]]
Tklist = [s[3]]

e_ant = 0 %Km*(T_SP-s[1])
p_ant = 0

for j in range(NB_STEPS):
    #degrau em Tin
    if j == 100:
        Tin = 120

    # degrau em F
   # if j == 10:
    #    F = 1100

    #e(t)
    e_atual=Km*(T_SP-Tlist[-1])

    #sinal de controle
    p_atual = p_ant + Kc*a0*e_atual - Kc*e_ant
    deltaQk=Kv*(p_atual-p_ant)
    Qk=Qklist[-1]+deltaQk
    e_ant=e_atual
    p_ant=p_atual

    s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

    # Atualização da memória
```

```
    Flist.append(F)
    Qklist.append(Qk)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT, NB_STEPS + 1)

plt.figure(1)
plt.plot(t,CBlist)
plt.xlabel('time')
plt.ylabel('CB(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
plt.xlabel('time')
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Tklist)
plt.xlabel('time')
plt.ylabel('Tk(t)')

plt.figure(3)
plt.subplot(211)
plt.plot(t,CAlist)
plt.xlabel('time')
plt.ylabel('CA(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.xlabel('time')
plt.ylabel('T(t)')

plt.show()
```

5. Simulando as duas malhas de controle juntas

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
```

```
from scipy.integrate import odeint
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
import math

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    V= 10 # L
    k10= 1.287E12 # h-1
    k20= 1.287E12 # h-1
    k30= 9.043E9 # L/molA.h
    mE1R= -9758.3 # K
    mE2R= -9758.3 # K
    mE3R= -8560.0 # K
    mdeltaHAB= -4.20 # kJ/molA
    mdeltaHBC= 11.0 # kJ/molB
    mdeltaHAD= 41.85 # kJ/molA
    rho= 0.9342 # kg/L
    Cp= 3.01 # kJ/kg.K
    Kw= 4032.0 # kJ/h.K.m2
    Ar= 0.215 # m2
    mk= 5.0 # kg
    Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
        1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 ) )*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T) +Kw*Ar/(rho*Cp*V)*(Tk-T),
        Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
    return dydt

def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 and deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
    # pontos no tempo
    t = np.linspace(0,deltaT)
```

```
    # resolvendo a ODE
    s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
    s_next = s_list[-1,:]
    return s_next

# Exemplo de simulação
# Condições iniciais
NB_STEPS = 1000 #número de passos da simulação
F = 1000
Tin = 110
CAin = 5.1
Qk = -4250
s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
print(s)
deltaT = 0.001655 # h

#MALHA DE CONTROLE 1- VC = CB , VM = F
#setpoint para Cb
CB_SP=0.512

#parâmetros do modelo
Km1 = 10.667
Kv1 = 125

#parâmetros do controlador PI
Kc1=-5.2937
a01=1.2

#MALHA DE CONTROLE 2- VC = T , VM = QK
#setpoint para T
T_SP=113.50 #°C

#parâmetros do modelo
Km2 = 0.1333
Kv2 = -531.25

#parâmetros do controlador PI
Kc2=-84.8
a02=1.083
```

```
# Inicialização da memória
Flist = [F]
Qklist = [Qk]
CAlist = [s[0]]
CBlist = [s[1]]
Tlist = [s[2]]
Tklist = [s[3]]

e_ant1 = 0 %Km1*(CB_SP-s[1])
p_ant1 = 0

e_ant2 = 0 %Km2*(T_SP-s[1])
p_ant2 = 0

for j in range(NB_STEPS):

    #step em Tin
    if j == 100:
        Tin = 120

    # step em F
   # if j == 10:
    #    F = 1100

    #e(t)
    e_atual1=Km1*(CB_SP-CBlist[-1])
    e_atual2=Km2*(T_SP-Tlist[-1])

    #sinal de controle
    #CONTROLADOR 1
    p_atual1 = p_ant1 + Kc1*a01*e_atual1 - Kc1*e_ant1
    deltaF=Kv1*(p_atual1-p_ant1)
    F=Flist[-1]+deltaF
    e_ant1=e_atual1
    p_ant1=p_atual1

    #CONTROLADOR 2
    p_atual2 = p_ant2 + Kc2*a02*e_atual2 - Kc2*e_ant2
    deltaQk=Kv2*(p_atual2-p_ant2)
```

```
    Qk=Qklist[-1]+deltaQk
    e_ant2=e_atual2
    p_ant2=p_atual2
    s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

    # atualização da memória
    Flist.append(F)
    Qklist.append(Qk)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT, NB_STEPS + 1)

plt.figure(1)
plt.plot(t,CBlist)
plt.xlabel('time')
plt.ylabel('CB(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
plt.xlabel('time')
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Tklist)
plt.xlabel('time')
plt.ylabel('Tk(t)')

plt.figure(3)
plt.subplot(211)
plt.plot(t,CAlist)
plt.xlabel('time')
plt.ylabel('CA(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.xlabel('time')
plt.ylabel('T(t)')
```

```
plt.show()
```

APÊNDICE B – Códigos utilizados para a 2ª etapa da Metodologia: em torno de um ponto genérico e controle adaptativo

1. **Simulando o controlador adaptativo**

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve
from scipy.signal import ss2tf
import matplotlib.pyplot as plt
import random

#parâmetros fixos
V= 10 # L
k10= 1.287E12 # h-1
k20= 1.287E12 # h-1
k30= 9.043E9 # L/molA.h
mE1R= -9758.3 # K
mE2R= -9758.3 # K
mE3R= -8560.0 # K
mdeltaHAB= -4.20 # kJ/molA
mdeltaHBC= 11.0 # kJ/molB
mdeltaHAD= 41.85 # kJ/molA
rho= 0.9342 # kg/L
Cp= 3.01 # kJ/kg.K
Kw= 4032.0 # kJ/h.K.m2
Ar= 0.215 # m2
mk= 5.0 # kg
Cpk = 2.0 # kJ/kg.K

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    # V= 10 # L
    # k10= 1.287E12 # h-1
    # k20= 1.287E12 # h-1
    # k30= 9.043E9 # L/molA.h
    # mE1R= -9758.3 # K
    # mE2R= -9758.3 # K
    # mE3R= -8560.0 # K
    # mdeltaHAB= -4.20 # kJ/molA
    # mdeltaHBC= 11.0 # kJ/molB
```

```
    # mdeltaHAD= 41.85 # kJ/molA
    # rho= 0.9342 # kg/L
    # Cp= 3.01 # kJ/kg.K
    # Kw= 4032.0 # kJ/h.K.m2
    # Ar= 0.215 # m2
    # mk= 5.0 # kg
    # Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
        1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 )
)*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T)
+Kw*Ar/(rho*Cp*V)*(Tk-T),
        Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
    return dydt
def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 e deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
    # pontos no tempo
    t = np.linspace(0,deltaT)

    # resolvendo a ODE
    s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
    s_next = s_list[-1,:]
    return s_next

# Exemplo de simulação
# Condições iniciais
#NB_STEPS = 1000 #número de passos da simulação
# F = 1000
# Tin = 110
# CAin = 5.1
# Qk = -4250

#Vetores de valores de operação
num_pontos=10 #numero de pontos por variável

F_op = np.linspace(0,2000,num_pontos+1)+(2000-0)/num_pontos/2
F_op=F_op[:-1]
```

```
print(F_op)

Qk_op = np.linspace(-8000,0,num_pontos+1)+(0- (-8000))/num_pontos/2
Qk_op=Qk_op[:-1]
print(Qk_op)

Cain_op = np.linspace(4,6,num_pontos+1)+(6-4)/num_pontos/2
Cain_op=Cain_op[:-1]
print(Cain_op)

Tin_op = np.linspace(100,150,num_pontos+1)+(150-100)/num_pontos/2
Tin_op=Tin_op[:-1]
print(Tin_op)

indexes = np.linspace(0,num_pontos-1,num_pontos,dtype=int)
print(indexes)

# estrutura para armazenar os valores de Kc1 para cada ponto de operação
kc1_mem=np.zeros((num_pontos, num_pontos, num_pontos, num_pontos))
# estrutura para armazenar os valores de Kc2 para cada ponto de operação
kc2_mem=np.zeros((num_pontos, num_pontos, num_pontos, num_pontos))
#contador
cont=0

for i in indexes:
    for j in indexes:
        for k in indexes:
            for l in indexes:
                F=F_op[i]
                Qk=Qk_op[j]
                CAin=Cain_op[k]
                Tin=Tin_op[l]
                # definição do ponto de operação
                s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
                # print(s)
                CA_ss=s[0]
                CB_ss=s[1]
                T_ss=s[2]
                Tk_ss=s[3]
                F_ss=F
```

```
T_in=Tin

k1_ss = k10*np.exp( mE1R/( T_ss + 273.15))
k2_ss = k20*np.exp( mE2R/( T_ss + 273.15))
k3_ss = k30*np.exp( mE3R/( T_ss + 273.15))

#Definindo os componentes das matrizes
a_11 = -((F_ss/(V) + k1_ss + 2*k3_ss*CA_ss))
a_13 = ((k1_ss*mE1R*CA_ss + k3_ss *mE3R* CA_ss**2)/(T_ss**2))
a_21 = k1_ss
a_22 = (-F_ss/V)-k2_ss
a_23 = ((k2_ss*mE2R*CB_ss-k1_ss*mE1R*CA_ss)/(T_ss**2))
a_31 = -((k1_ss*mdeltaHAB + 2*k3_ss*CA_ss*mdeltaHAD)/(rho*Cp))
a_32 = - ((k2_ss*mdeltaHBC)/(rho*Cp))
a_33 = - ((F_ss/V)+((Kw*Ar)/(rho*Cp*V))
       +((-CA_ss*mdeltaHAB*k1_ss*mE1R - CB_ss*mdeltaHBC*k2_ss*mE2R -

 CA_ss**2*mdeltaHAD*k3_ss*mE3R)/(rho*Cp*T_ss**2)))

a_34 = ((Kw*Ar)/(rho*Cp*V))
a_43 = ((Kw*Ar)/(mk*Cpk))
a_44 = -((Kw*Ar)/(mk*Cpk))

b_11 = ((CAin - CA_ss)/V)
b_21 = -CB_ss/V
b_31 = ((T_in-T_ss)/V)
b_42 = (1/(mk*Cpk))

#Definindo as matrizes

a = [[a_11,0,a_13,0],[a_21,a_22,a_23,0],[a_31,a_32,a_33,a_34],[0,0,a_43,a_44]]

b = [[b_11,0],[b_21,0],[b_31,0],[0,b_42]]

c = [[0,1,0,0],[0,0,1,0]]

d = [[0,0],[0,0]]

# K_Cb_F
H0=ss2tf(a, b, c, d,0)
```

```
            K_Cb_F=H0[0][0][-1]/H0[1][-1]
            Kc1=0.00225/K_Cb_F

            # K_T_Qk
            H1=ss2tf(a, b, c, d,1)
            K_T_Qk=H1[0][1][-1]/H1[1][-1]
            Kc2=-0.0424/K_T_Qk

            kc1_mem[i][j][k][l]=Kc1
            kc2_mem[i][j][k][l]=Kc2

            cont+=1
            #print(cont)

# Exemplo de simulação
# Inicialização da memória
Flist = []
Qklist = []
CAlist = []
CBlist = []
Tlist = []
Tklist = []

# Condições iniciais
NB_STEPS = 400 #número de passos da simulação
num_simulações = 4
cont_sim = 1
random.seed(1)

while cont_sim <= num_simulações:

   #seleção aleatoria das condições de operação
   F = random.uniform(0,2000)
   Tin = random.uniform(100,150)
   CAin = random.uniform(4,6)
   Qk = random.uniform(-8000,0)
   print(F,Tin,CAin,Qk)

   #Definição dos kc1 e kc2 correspondentes
```

```
Kc1=kc1_mem[np.argmin(abs(F_op-F))][np.argmin(abs(Qk_op-Qk))][np.argmin(abs(Cain_op-CAin))][np.argmin(abs(Tin_op-Tin))]
Kc2=kc2_mem[np.argmin(abs(F_op-F))][np.argmin(abs(Qk_op-Qk))][np.argmin(abs(Cain_op-CAin))][np.argmin(abs(Tin_op-Tin))]
print(Kc1)
print(Kc2)

s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
print(s)
deltaT = 0.001655 # h

#MALHA DE CONTROLE 1- VC = CB , VM = F
#setpoint para Cb
CB_SP=s[1]

#parâmetros do modelo
Km1 = 10.667
Kv1 = 125

#parâmetros do controlador PI
# Kc1=0.00225/K_Cb_F
a01=1.2

#MALHA DE CONTROLE 2- VC = T , VM = QK
#setpoint para T
T_SP=s[2] #°C

#parâmetros do modelo
Km2 = 0.1333
Kv2 = -531.25

#parâmetros do controlador PI
# Kc2=-0.0424/K_T_Qk
a02=1.083

e_ant1 = 0 %Km1*(CB_SP-s[1])
p_ant1 = 0

e_ant2 = 0 %Km2*(T_SP-s[2])
p_ant2 = 0
```

```
for j in range(NB_STEPS):

    # atualização da memória
    Flist.append(F)
    Qklist.append(Qk)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

    #degrau em Tin
    if j == 50:
        Tin = Tin + 5
    if j == 250:
        CAin = CAin + 0.5

    #e(t)
    e_atual1=Km1*(CB_SP-CBlist[-1])
    e_atual2=Km2*(T_SP-Tlist[-1])

    #sinal de controle
    #CONTROLADOR 1
    p_atual1 = p_ant1 + Kc1*a01*e_atual1 - Kc1*e_ant1
    deltaF=Kv1*(p_atual1-p_ant1)
    F=Flist[-1]+deltaF
    e_ant1=e_atual1
    p_ant1=p_atual1

    #CONTROLADOR 2
    p_atual2 = p_ant2 + Kc2*a02*e_atual2 - Kc2*e_ant2
    deltaQk=Kv2*(p_atual2-p_ant2)
    Qk=Qklist[-1]+deltaQk
    e_ant2=e_atual2
    p_ant2=p_atual2


    s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

cont_sim += 1
```

```
# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT*num_simulações, NB_STEPS*num_simulações)

plt.figure(1)
plt.subplot(211)
plt.plot(t,CBlist)
plt.xlabel('time')
plt.ylabel('CB(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.xlabel('time')
plt.ylabel('T(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
plt.xlabel('time')
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Qklist)
plt.xlabel('time')
plt.ylabel('Qk(t)')

plt.show()
```

2. Controlador Adaptativo X Controlador Fixo

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve
from scipy.signal import ss2tf
import matplotlib.pyplot as plt
import random

#parâmetros fixos
V= 10 # L
k10= 1.287E12 # h-1
k20= 1.287E12 # h-1
```

```
k30= 9.043E9 # L/molA.h
mE1R= -9758.3 # K
mE2R= -9758.3 # K
mE3R= -8560.0 # K
mdeltaHAB= -4.20 # kJ/molA
mdeltaHBC= 11.0 # kJ/molB
mdeltaHAD= 41.85 # kJ/molA
rho= 0.9342 # kg/L
Cp= 3.01 # kJ/kg.K
Kw= 4032.0 # kJ/h.K.m2
Ar= 0.215 # m2
mk= 5.0 # kg
Cpk = 2.0 # kJ/kg.K

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    # V= 10 # L
    # k10= 1.287E12 # h-1
    # k20= 1.287E12 # h-1
    # k30= 9.043E9 # L/molA.h
    # mE1R= -9758.3 # K
    # mE2R= -9758.3 # K
    # mE3R= -8560.0 # K
    # mdeltaHAB= -4.20 # kJ/molA
    # mdeltaHBC= 11.0 # kJ/molB
    # mdeltaHAD= 41.85 # kJ/molA
    # rho= 0.9342 # kg/L
    # Cp= 3.01 # kJ/kg.K
    # Kw= 4032.0 # kJ/h.K.m2
    # Ar= 0.215 # m2
    # mk= 5.0 # kg
    # Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
```

```
    1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 ) )*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T) +Kw*Ar/(rho*Cp*V)*(Tk-T),
    Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
  return dydt
def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 e deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
  # pontos do tempo
  t = np.linspace(0,deltaT)

  # resolvendo a ODE
  s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
  s_next = s_list[-1,:]

  return s_next

# Cálculo de Kc1 e Kc2 para o ponto de operação de referencia
F = 1000
Tin = 110
CAin = 5.1
Qk = -4250

# definição do ponto de operação
s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
# print(s)
CA_ss=s[0]
CB_ss=s[1]
T_ss=s[2]
Tk_ss=s[3]
F_ss=F
T_in=Tin

k1_ss = k10*np.exp( mE1R/( T_ss + 273.15))
k2_ss = k20*np.exp( mE2R/( T_ss + 273.15))
k3_ss = k30*np.exp( mE3R/( T_ss + 273.15))

#Definindo os componentes das matrizes
a_11 = -((F_ss/(V) + k1_ss + 2*k3_ss*CA_ss))
a_13 = ((k1_ss*mE1R*CA_ss + k3_ss *mE3R* CA_ss**2)/(T_ss**2))
```

```
a_21 = k1_ss
a_22 = (-F_ss/V)-k2_ss
a_23 = ((k2_ss*mE2R*CB_ss-k1_ss*mE1R*CA_ss)/(T_ss**2))
a_31 = -((k1_ss*mdeltaHAB + 2*k3_ss*CA_ss*mdeltaHAD)/(rho*Cp))
a_32 = - ((k2_ss*mdeltaHBC)/(rho*Cp))
a_33 = - ((F_ss/V)+((Kw*Ar)/(rho*Cp*V))
       +((-CA_ss*mdeltaHAB*k1_ss*mE1R - CB_ss*mdeltaHBC*k2_ss*mE2R -

 CA_ss**2*mdeltaHAD*k3_ss*mE3R)/(rho*Cp*T_ss**2)))

a_34 = ((Kw*Ar)/(rho*Cp*V))
a_43 = ((Kw*Ar)/(mk*Cpk))
a_44 = -((Kw*Ar)/(mk*Cpk))

b_11 = ((CAin - CA_ss)/V)
b_21 = -CB_ss/V
b_31 = ((T_in-T_ss)/V)
b_42 = (1/(mk*Cpk))

#Definindo as matrizes
a = [[a_11,0,a_13,0],[a_21,a_22,a_23,0],[a_31,a_32,a_33,a_34],[0,0,a_43,a_44]]

b = [[b_11,0],[b_21,0],[b_31,0],[0,b_42]]

c = [[0,1,0,0],[0,0,1,0]]

d = [[0,0],[0,0]]

# K_Cb_F
H0=ss2tf(a, b, c, d,0)
K_Cb_F=H0[0][0][-1]/H0[1][-1]
Kc1_ref=0.00225/K_Cb_F

# K_T_Qk
H1=ss2tf(a, b, c, d,1)
K_T_Qk=H1[0][1][-1]/H1[1][-1]
Kc2_ref=-0.0424/K_T_Qk
# print(Kc1_ref)
# print(Kc2_ref)
```

```
# Calculo de Kc1 e Kc2 para todos os pontos de operação
#Vetores de valores de operação
num_pontos=10 #numero de pontos por variável

F_op = np.linspace(0,2000,num_pontos+1)+(2000-0)/num_pontos/2
F_op=F_op[:-1]
#print(F_op)

Qk_op = np.linspace(-8000,0,num_pontos+1)+(0- (-8000))/num_pontos/2
Qk_op=Qk_op[:-1]
#print(Qk_op)

Cain_op = np.linspace(4,6,num_pontos+1)+(6-4)/num_pontos/2
Cain_op=Cain_op[:-1]
#print(Cain_op)

Tin_op = np.linspace(100,150,num_pontos+1)+(150-100)/num_pontos/2
Tin_op=Tin_op[:-1]
#print(Tin_op)

indexes = np.linspace(0,num_pontos-1,num_pontos,dtype=int)
#print(indexes)

# estrutura para armazenar os valores de Kc1 para cada ponto de operação
kc1_mem=np.zeros((num_pontos, num_pontos, num_pontos, num_pontos))
# estrutura para armazenar os valores de Kc2 para cada ponto de operação
kc2_mem=np.zeros((num_pontos, num_pontos, num_pontos, num_pontos))
#contador
cont=0

for i in indexes:
   for j in indexes:
      for k in indexes:
         for l in indexes:
            F=F_op[i]
            Qk=Qk_op[j]
            CAin=Cain_op[k]
            Tin=Tin_op[l]
            # definição do ponto de operação
            s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # fixed initial states
```

```
# print(s)
CA_ss=s[0]
CB_ss=s[1]
T_ss=s[2]
Tk_ss=s[3]
F_ss=F
T_in=Tin

k1_ss = k10*np.exp( mE1R/( T_ss + 273.15))
k2_ss = k20*np.exp( mE2R/( T_ss + 273.15))
k3_ss = k30*np.exp( mE3R/( T_ss + 273.15))

#Definindo os componentes das matrizes
a_11 = -((F_ss/(V) + k1_ss + 2*k3_ss*CA_ss))
a_13 = ((k1_ss*mE1R*CA_ss + k3_ss *mE3R* CA_ss**2)/(T_ss**2))
a_21 = k1_ss
a_22 = (-F_ss/V)-k2_ss
a_23 = ((k2_ss*mE2R*CB_ss-k1_ss*mE1R*CA_ss)/(T_ss**2))
a_31 = -((k1_ss*mdeltaHAB + 2*k3_ss*CA_ss*mdeltaHAD)/(rho*Cp))
a_32 = - ((k2_ss*mdeltaHBC)/(rho*Cp))
a_33 = - ((F_ss/V)+((Kw*Ar)/(rho*Cp*V))
       +((-CA_ss*mdeltaHAB*k1_ss*mE1R - CB_ss*mdeltaHBC*k2_ss*mE2R -

 CA_ss**2*mdeltaHAD*k3_ss*mE3R)/(rho*Cp*T_ss**2)))

a_34 = ((Kw*Ar)/(rho*Cp*V))
a_43 = ((Kw*Ar)/(mk*Cpk))
a_44 = -((Kw*Ar)/(mk*Cpk))

b_11 = ((CAin - CA_ss)/V)
b_21 = -CB_ss/V
b_31 = ((T_in-T_ss)/V)
b_42 = (1/(mk*Cpk))


#Definindo as matrizes


a = [[a_11,0,a_13,0],[a_21,a_22,a_23,0],[a_31,a_32,a_33,a_34],[0,0,a_43,a_44]]
```

```
            b = [[b_11,0],[b_21,0],[b_31,0],[0,b_42]]

            c = [[0,1,0,0],[0,0,1,0]]

            d = [[0,0],[0,0]]

            # K_Cb_F
            H0=ss2tf(a, b, c, d,0)
            K_Cb_F=H0[0][0][-1]/H0[1][-1]
            Kc1=0.00225/K_Cb_F

            # K_T_Qk
            H1=ss2tf(a, b, c, d,1)
            K_T_Qk=H1[0][1][-1]/H1[1][-1]
            Kc2=-0.0424/K_T_Qk

            kc1_mem[i][j][k][l]=Kc1
            kc2_mem[i][j][k][l]=Kc2

            cont+=1
            #print(cont)

# Exemplo de simulação

# Inicialização da memória
Flist = []
Qklist = []
CAlist = []
CBlist = []
Tlist = []
Tklist = []

# Inicialização da memória
Flist_ref = []
Qklist_ref = []
CAlist_ref = []
CBlist_ref = []
Tlist_ref = []
Tklist_ref = []
```

```
# Condições iniciais
NB_STEPS = 400 #número de passos da simulação
num_simulações = 4
cont_sim = 1
random.seed(0)

while cont_sim <= num_simulações:

   #seleção aleatoria das condições de operação
   F = random.uniform(0,2000)
   Tin = random.uniform(100,150)
   CAin = random.uniform(4,6)
   Qk = random.uniform(-8000,0)
   print(F,Tin,CAin,Qk)

   F_init=F
   Tin_init = Tin
   CAin_init = CAin
   Qk_init=Qk

   #Definição dos kc1 e kc2 correspondentes
   Kc1=kc1_mem[np.argmin(abs(F_op-F))][np.argmin(abs(Qk_op-Qk))][np.argmin(abs(Cain_op-
CAin))][np.argmin(abs(Tin_op-Tin))]
   Kc2=kc2_mem[np.argmin(abs(F_op-F))][np.argmin(abs(Qk_op-Qk))][np.argmin(abs(Cain_op-
CAin))][np.argmin(abs(Tin_op-Tin))]
   #print(Kc1)
   #print(Kc2)

   s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
   #print(s)
   deltaT = 0.001655 # h
   s_init = s

   #MALHA DE CONTROLE 1- VC = CB , VM = F
   #setpoint para Cb
   CB_SP=s[1]

   #parâmetros do modelo
   Km1 = 10.667
   Kv1 = 125
```

```
#parâmetros do controlador PI
# Kc1=0.00225/K_Cb_F
a01=1.2

#MALHA DE CONTROLE 2- VC = T , VM = QK

#setpoint para T
T_SP=s[2] #°C

#parâmetros do modelo
Km2 = 0.1333
Kv2 = -531.25

#parâmetros do controlador PI
# Kc2=-0.0424/K_T_Qk
a02=1.083

e_ant1 = 0 #Km1*(CB_SP-s[1])
p_ant1 = 0

e_ant2 = 0 #Km2*(T_SP-s[2])
p_ant2 = 0

for j in range(NB_STEPS):

    # atualização da memória
    Flist.append(F)
    Qklist.append(Qk)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

    #degrau em Tin
    if j == 50:
        Tin = Tin + 5
    if j == 250:
        CAin = CAin + 0.5
```

```
    #e(t)
    e_atual1=Km1*(CB_SP-CBlist[-1])
    e_atual2=Km2*(T_SP-Tlist[-1])

    #sinal de controle
    #CONTROLADOR 1
    p_atual1 = p_ant1 + Kc1*a01*e_atual1 - Kc1*e_ant1
    deltaF=Kv1*(p_atual1-p_ant1)
    F=Flist[-1]+deltaF
    e_ant1=e_atual1
    p_ant1=p_atual1

    #CONTROLADOR 2
    p_atual2 = p_ant2 + Kc2*a02*e_atual2 - Kc2*e_ant2
    deltaQk=Kv2*(p_atual2-p_ant2)
    Qk=Qklist[-1]+deltaQk
    e_ant2=e_atual2
    p_ant2=p_atual2

    s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

CAin = CAin_init
Tin=Tin_init

F=F_init
Qk = Qk_init
s=s_init

e_ant1 = 0 #Km1*(CB_SP-s[1])
p_ant1 = 0

e_ant2 = 0 #Km2*(T_SP-s[2])
p_ant2 = 0

for j in range(NB_STEPS):


    Flist_ref.append(F)
    Qklist_ref.append(Qk)
    Tklist_ref.append(s[3])
```

```
        CAlist_ref.append(s[0])
        CBlist_ref.append(s[1])
        Tlist_ref.append(s[2])

        #degrau emTin
        if j == 50:
           Tin = Tin + 5
        if j == 250:
           CAin = CAin + 0.5

        #e(t)
        e_atual1=Km1*(CB_SP-CBlist_ref[-1])
        e_atual2=Km2*(T_SP-Tlist_ref[-1])

        #sinal de controle
        #CONTROLADOR 1
        p_atual1 = p_ant1 + Kc1_ref*a01*e_atual1 - Kc1_ref*e_ant1
        deltaF=Kv1*(p_atual1-p_ant1)
        F=Flist_ref[-1]+deltaF
        e_ant1=e_atual1
        p_ant1=p_atual1

        #CONTROLADOR 2
        p_atual2 = p_ant2 + Kc2_ref*a02*e_atual2 - Kc2_ref*e_ant2
        deltaQk=Kv2*(p_atual2-p_ant2)
        Qk=Qklist_ref[-1]+deltaQk
        e_ant2=e_atual2
        p_ant2=p_atual2

        s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

    cont_sim += 1

# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT*num_simulações, NB_STEPS*num_simulações)

plt.figure(1)
plt.subplot(211)
plt.plot(t,CBlist)
plt.plot(t,CBlist_ref)
```

```
plt.xlabel('time')
plt.ylabel('CB(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.plot(t,Tlist_ref)
plt.xlabel('time')
plt.ylabel('T(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
plt.plot(t,Flist_ref)
plt.xlabel('time')
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Qklist)
plt.plot(t,Qklist_ref)
plt.xlabel('time')
plt.ylabel('Qk(t)')

plt.show()
```

3. Variações extremas de F e $\dot{Q}_K$

```
#Funções simulando o comportamento do reator de Van De Vusse
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve
from scipy.signal import ss2tf
import matplotlib.pyplot as plt
import random

#parâmetros fixos
V= 10 # L
k10= 1.287E12 # h-1
k20= 1.287E12 # h-1
k30= 9.043E9 # L/molA.h
mE1R= -9758.3 # K
mE2R= -9758.3 # K
mE3R= -8560.0 # K
```

```
mdeltaHAB= -4.20 # kJ/molA
mdeltaHBC= 11.0 # kJ/molB
mdeltaHAD= 41.85 # kJ/molA
rho= 0.9342 # kg/L
Cp= 3.01 # kJ/kg.K
Kw= 4032.0 # kJ/h.K.m2
Ar= 0.215 # m2
mk= 5.0 # kg
Cpk = 2.0 # kJ/kg.K

# função que retorna dy/dt
def modelVdV(y,t,F,Qk,Tin,CAin): #Tk em °C, Tin em °C, F em L/h, CAin em molA/L
    CA, CB, T, Tk= y
    #parâmetros fixos
    # V= 10 # L
    # k10= 1.287E12 # h-1
    # k20= 1.287E12 # h-1
    # k30= 9.043E9 # L/molA.h
    # mE1R= -9758.3 # K
    # mE2R= -9758.3 # K
    # mE3R= -8560.0 # K
    # mdeltaHAB= -4.20 # kJ/molA
    # mdeltaHBC= 11.0 # kJ/molB
    # mdeltaHAD= 41.85 # kJ/molA
    # rho= 0.9342 # kg/L
    # Cp= 3.01 # kJ/kg.K
    # Kw= 4032.0 # kJ/h.K.m2
    # Ar= 0.215 # m2
    # mk= 5.0 # kg
    # Cpk = 2.0 # kJ/kg.K

    #equações diferenciais
    dydt= [(F/V)*(CAin-CA) - k10*np.exp( mE1R/( T+273.15 ) )*CA - k30*np.exp( mE3R/(T+273.15) )*CA**2,
        -(F/V)*CB      +k10*np.exp( mE1R/( T+273.15 ) )*CA - k20*np.exp( mE2R/(T+273.15) )*CB,
        1/rho/Cp*( k10*np.exp( mE1R/( T+273.15 ) )*CA*mdeltaHAB + k20*np.exp( mE2R/( T+273.15 ) )*CB*mdeltaHBC + k30*np.exp( mE3R/( T+273.15 ) )*(CA**2)*mdeltaHAD) + F/V*(Tin-T) +Kw*Ar/(rho*Cp*V)*(Tk-T),
        Qk/mk*Cpk + (Kw*Ar/mk*Cpk)*(T - Tk)]
    return dydt
```

```
def stepVdV(deltaT,s,F,Qk,Tin,CAin): #integra o modelo entre 0 e deltaT

#s vetor com as condições iniciais (Ca Cb T Tk)
    # pontos no tempo
    t = np.linspace(0,deltaT)

    # resolvendo a ODE
    s_list = odeint(modelVdV,s,t,args=(F,Qk,Tin,CAin))
    s_next = s_list[-1,:]

    return s_next

# Exemplo de simulação
# Condições iniciais
#NB_STEPS = 1000 #número de passos da simulação
# F = 1000
# Tin = 110
# CAin = 5.1
# Qk = -4250

#Vetores de valores de operação
num_pontos=10 #numero de pontos por variável

F_op = np.linspace(0,2000,num_pontos+1)+(2000-0)/num_pontos/2
F_op=F_op[:-1]
print(F_op)

Qk_op = np.linspace(-8000,0,num_pontos+1)+(0- (-8000))/num_pontos/2
Qk_op=Qk_op[:-1]
print(Qk_op)

Cain_op = np.linspace(4,6,num_pontos+1)+(6-4)/num_pontos/2
Cain_op=Cain_op[:-1]
print(Cain_op)

Tin_op = np.linspace(100,150,num_pontos+1)+(150-100)/num_pontos/2
Tin_op=Tin_op[:-1]
print(Tin_op)

indexes = np.linspace(0,num_pontos-1,num_pontos,dtype=int)
```

```
print(indexes)

# estrutura para armazenar os valores de Kc1 para cada ponto de operação
kc1_mem=np.zeros((num_pontos, num_pontos, num_pontos, num_pontos))
# estrutura para armazenar os valores de Kc2 para cada ponto de operação
kc2_mem=np.zeros((num_pontos, num_pontos, num_pontos, num_pontos))
#contador
cont=0

for i in indexes:
    for j in indexes:
        for k in indexes:
            for l in indexes:
                F=F_op[i]
                Qk=Qk_op[j]
                CAin=Cain_op[k]
                Tin=Tin_op[l]
                # definição do ponto de operação
                s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
                # print(s)
                CA_ss=s[0]
                CB_ss=s[1]
                T_ss=s[2]
                Tk_ss=s[3]
                F_ss=F
                T_in=Tin

                k1_ss = k10*np.exp( mE1R/( T_ss + 273.15))
                k2_ss = k20*np.exp( mE2R/( T_ss + 273.15))
                k3_ss = k30*np.exp( mE3R/( T_ss + 273.15))

                #Definindo os componentes das matrizes
                a_11 = -((F_ss/(V) + k1_ss + 2*k3_ss*CA_ss))
                a_13 = ((k1_ss*mE1R*CA_ss + k3_ss *mE3R* CA_ss**2)/(T_ss**2))
                a_21 = k1_ss
                a_22 = (-F_ss/V)-k2_ss
                a_23 = ((k2_ss*mE2R*CB_ss-k1_ss*mE1R*CA_ss)/(T_ss**2))
                a_31 = -((k1_ss*mdeltaHAB + 2*k3_ss*CA_ss*mdeltaHAD)/(rho*Cp))
                a_32 = - ((k2_ss*mdeltaHBC)/(rho*Cp))
                a_33 = - ((F_ss/V)+((Kw*Ar)/(rho*Cp*V))
```

```
        +((-CA_ss*mdeltaHAB*k1_ss*mE1R - CB_ss*mdeltaHBC*k2_ss*mE2R -

 CA_ss**2*mdeltaHAD*k3_ss*mE3R)/(rho*Cp*T_ss**2)))

a_34 = ((Kw*Ar)/(rho*Cp*V))
a_43 = ((Kw*Ar)/(mk*Cpk))
a_44 = -((Kw*Ar)/(mk*Cpk))

b_11 = ((CAin - CA_ss)/V)
b_21 = -CB_ss/V
b_31 = ((T_in-T_ss)/V)
b_42 = (1/(mk*Cpk))

#Definindo as matrizes

a = [[a_11,0,a_13,0],[a_21,a_22,a_23,0],[a_31,a_32,a_33,a_34],[0,0,a_43,a_44]]

b = [[b_11,0],[b_21,0],[b_31,0],[0,b_42]]

c = [[0,1,0,0],[0,0,1,0]]

d = [[0,0],[0,0]]

# K_Cb_F
H0=ss2tf(a, b, c, d,0)
K_Cb_F=H0[0][0][-1]/H0[1][-1]
Kc1=0.00225/K_Cb_F

# K_T_Qk
H1=ss2tf(a, b, c, d,1)
K_T_Qk=H1[0][1][-1]/H1[1][-1]
Kc2=-0.0424/K_T_Qk

kc1_mem[i][j][k][l]=Kc1
kc2_mem[i][j][k][l]=Kc2

cont+=1
#print(cont)

# Exemplo de simulação
```

```
# Inicialização da memória
Flist = []
Qklist = []
CAlist = []
CBlist = []
Tlist = []
Tklist = []

# Condições iniciais
NB_STEPS = 400 #número de passos da simulação
#num_simulações = 4
num_simulações = 1
cont_sim = 1
#random.seed(1)
random.seed(2)

while cont_sim <= num_simulações:

   #seleção aleatoria das condições de operação
#F
   #variaçao total de F
   #F = random.uniform(0,2000)

   #limite inferior de F
   #F = random.uniform(0,400)

   #limite superior de F
   F = random.uniform(1600,2000)

   #estado estacionario de F
   #F = 1000

#Tin
   #variação total de Tin
   #Tin = random.uniform(100,150)

   #estado estacionário de Tin
   Tin = 110

#Cain
```

```
    #variação total de Cain
    #CAin = random.uniform(4,6)

    #estado estacionário de Cain
    CAin = 5.12

#Qk
    #variação total de Qk
    #Qk = random.uniform(-8500,0)

    #limite inferior de Qk
    #Qk = random.uniform(-8500,-6800)

    #limite superior de Qk
    Qk = random.uniform(-1700,0)

    #estado estacionario de Qk
    #Qk = - 4250

    print(F,Tin,CAin,Qk)

    #Definição dos kc1 e kc2 correspondentes
    Kc1=kc1_mem[np.argmin(abs(F_op-F))][np.argmin(abs(Qk_op-Qk))][np.argmin(abs(Cain_op-
CAin))][np.argmin(abs(Tin_op-Tin))]
    Kc2=kc2_mem[np.argmin(abs(F_op-F))][np.argmin(abs(Qk_op-Qk))][np.argmin(abs(Cain_op-
CAin))][np.argmin(abs(Tin_op-Tin))]
    print(Kc1)
    print(Kc2)

    s = fsolve(modelVdV,[5.1, 0, 100, 86],args=(0,F,Qk,Tin,CAin)) # estado inicial fixo
    print(s)
    deltaT = 0.001655 # h

    #MALHA DE CONTROLE 1- VC = CB , VM = F
    #setpoint para Cb
    CB_SP=s[1]

    #parâmetros do modelo
    Km1 = 10.667
    Kv1 = 125
```

```
#parâmetros do controlador PI
# Kc1=0.00225/K_Cb_F
a01=1.2

#MALHA DE CONTROLE 2- VC = T , VM = QK
#setpoint para T
T_SP=s[2] #°C

#parâmetros do modelo
Km2 = 0.1333
Kv2 = -531.25

#parâmetros do controlador PI
# Kc2=-0.0424/K_T_Qk
a02=1.083

e_ant1 = 0 %Km1*(CB_SP-s[1])
p_ant1 = 0

e_ant2 = 0 %Km2*(T_SP-s[2])
p_ant2 = 0

for j in range(NB_STEPS):

    # atualização da memória
    Flist.append(F)
    Qklist.append(Qk)
    Tklist.append(s[3])
    CAlist.append(s[0])
    CBlist.append(s[1])
    Tlist.append(s[2])

    #degrau em Tin
    if j == 50:
        Tin = Tin + 5
    if j == 250:
        CAin = CAin + 0.5

    #e(t)
```

```
        e_atual1=Km1*(CB_SP-CBlist[-1])
        e_atual2=Km2*(T_SP-Tlist[-1])

        #sinal de controle
        #CONTROLADOR 1
        p_atual1 = p_ant1 + Kc1*a01*e_atual1 - Kc1*e_ant1
        deltaF=Kv1*(p_atual1-p_ant1)
        F=Flist[-1]+deltaF
        e_ant1=e_atual1
        p_ant1=p_atual1

        #CONTROLADOR 2
        p_atual2 = p_ant2 + Kc2*a02*e_atual2 - Kc2*e_ant2
        deltaQk=Kv2*(p_atual2-p_ant2)
        Qk=Qklist[-1]+deltaQk
        e_ant2=e_atual2
        p_ant2=p_atual2


        s= stepVdV(deltaT,s,F,Qk,Tin,CAin)

    cont_sim += 1

# mostrando os resultados
t=np.linspace(0,NB_STEPS*deltaT*num_simulações, NB_STEPS*num_simulações)

plt.figure(1)
plt.subplot(211)
plt.plot(t,CBlist)
plt.xlabel('time')
plt.ylabel('CB(t)')
plt.subplot(212)
plt.plot(t,Tlist)
plt.xlabel('time')
plt.ylabel('T(t)')

plt.figure(2)
plt.subplot(211)
plt.plot(t,Flist)
plt.xlabel('time')
```

```
plt.ylabel('F(t)')
plt.subplot(212)
plt.plot(t,Qklist)
plt.xlabel('time')
plt.ylabel('Qk(t)')

plt.show()
```