UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA QUÍMICA

ENG07053 - TRABALHO DE DIPLOMAÇÃO EM ENGENHARIA QUÍMICA

Detecção de Falhas através de Estimadores de Estado

Autor: Adriano Vasconcellos Soares

Orientador: Jorge Otávio Trierweiler

Porto Alegre, julho de 16

Sumário

Agradecimentos

Aos meus pais, pelo apoio irrestrito, pelo amor e carinho. Minha maior felicidade é vê-los orgulhosos.

À minha irmã, Elisa, de quem sinto falta todos os dias.

Aos meus amigos que a engenharia me trouxe, João, Lucas, Leti, Cibele, Lemanski, Tadewald, Astro, Moraes, e tantos outros que, apesar de não citados, são tão importantes para mim.

Ao Professor Jorge Otávio Trierweiler, por compartilhar o conhecimento, pelos conselhos e amizade.

À tia Bete e ao tio Ademir, pelos jantares e por terem sido minha família aqui em Porto Alegre.

Resumo

A detecção de falhas em processos da indústria química é uma tarefa de fundamental importância. Ao detectarmos cedo uma anormalidade no sistema, uma ação corretiva se torna possível, evitando assim a obtenção de produtos fora de especificação, a quebra de equipamentos e até mesmo possíveis falhas de segurança. Dentre as técnicas de detecção e diagnóstico de falhas, as baseadas em modelos se apresentam como uma das mais relevantes, isto por permitirem a aplicação em plantas em estado transiente, em malha fechada e por reagirem bem a leves perturbações nas variáveis do processo. Com o âmbito de diminuir a incerteza quanto ao modelo desenvolvido do processo, se faz possível a utilização de estimadores de estado. Estes fazem uso tanto de dados provenientes de medições quanto informações geradas a partir de modelos para estimar o estado atual das variáveis de interesse. No presente trabalho, fez-se a simulação de um sistema de três tanques utilizando-se a linguagem Python. Dois diferentes estimadores de estado foram implementados, o filtro de Kalman estendido (EKF) e o filtro de Kalman estendido com restrições (CEKF). Uma nova metodologia de detecção de falhas utilizando CEKF foi proposta. Foi possível avaliar o ganho de robustez do algoritmo CEKF em relação ao EKF, mesmo para o sistema simples de três tanques. A metodologia proposta mostrou-se efetiva na detecção de três diferentes falhas no sistema de três tanques.

Lista de Figuras

Lista de Tabelas

1 Introdução

Há uma demanda crescente da sociedade por produtos de maior qualidade e por processos de produção mais limpos. Os processos industriais se tornaram altamente complexos e com maior nível de automatização e integração exatamente com objetivo de atender tal demanda. Este desenvolvimento carece de sistemas mais estáveis, confiáveis e seguros. Devido a isso, os campos de monitoramento de processos, diagnóstico de falhas e sistemas de controle tolerantes a falha têm ganhado maior atenção (DING, 2013).

A área de controle de processo avançou rapidamente nas últimas quatro décadas com o advento do controle computacional de processos complexos. Com o progresso em sistemas de controle distribuído e de controle preditivo baseado em modelos, os benefícios a vários segmentos da indústria química, petroquímica, de energia, do aço e tantas mais. Entretanto, uma tarefa de controle importantíssima ainda depende fundamentalmente da habilidade e conhecimento de operadores. Esta tarefa é responder a eventos anormais no processo, que envolve a detecção do evento, diagnóstico e causa de origem. (VENKATASUBRAMANIAN; RENGASWAMY; YIN, 2003)

Atualmente, a pesquisa e aplicação de sistemas de diagnóstico de falhas já se estendem além de processos usualmente considerados críticos em quesito de segurança (reatores nucleares, aeronaves, plantas químicas, etc.), e está presente em novas áreas de conhecimento como pilotagem autônoma e trens de alta velocidade. A detecção prévia de falhas para manutenção ajudam a evitar a parada (*shutdown*) de sistemas, quebras e até possíveis catástrofes, envolvendo danos materiais e fatalidades. (WITCZAK; APPROACHES, 2014)

A necessidade de avanços na área de diagnóstico de falhas em sistemas é evidente. Por meio de técnicas simples, se torna o possível o desenvolvimento de processos industriais menos suscetíveis a falhas humanas, aumentando assim a durabilidade de equipamentos e a segurança da planta como um todo. Por fim, obtendo também produtos de mais alta qualidade e evitando desperdícios na indústria.

O presente trabalho tem como objetivo utilizar técnicas de estimação de estados para o diagnóstico de falhas em processos da indústria química. Por meio da estimação dos estados, é possível fazer a utilização ótima de todas as informações disponíveis da planta a fim de descrevermos o estado atual do sistema. Isto é fundamental para a geração de ruídos que indicarão possíveis anormalidades no processo.

O processo em estudo é um sistema de três tanques simples, sendo este modelado e simulado em *Python*. Duas técnicas de estimação foram implementadas, o filtro de Kalman Estendido e filtro de Kalman estendido com restrições. A comparação destas é feita de forma a determinar a que melhor se adequa ao sistema. E, por fim, a técnica escolhida é utilizada para o diagnóstico de três diferentes falhas simuladas.

O trabalho está dividido como segue: no Capítulo 2 será mostrada uma breve fundamentação teórica de processo de diagnóstico de falhas e de estimação de estados. Serão expostas as principais técnicas encontradas na literatura de diagnóstico de falhas. Também são descritos os algoritmos de estimação de parâmetros por filtro de Kalman simples, filtro de Kalman estendido, estimação de horizonte móvel e filtro de Kalman estendido com restrições.

O sistema em estudo é detalhado no Capítulo 3. Os parâmetros utilizados para a simulação são descritos, além do processo de linearização do sistema, fundamental para a utilização de técnicas de estimação de estado. É descrita também a modelagem das falhas estudadas.

A metodologia utilizada é relatada no Capítulo 4. Detalhes sobre a ferramenta utilizada para a implementação são descritos. E, por fim, é evidenciado o processo de geração de sinal de falha e detecção.

No Capítulo 5 os resultados e discussões são apresentados. No primeiro momento são realizadas análises de forma a confirmar a implementação correta dos filtros, evidenciando assim o efeito de diferentes parâmetros na inicialização destes. A comparação do filtro de Kalman estendido com o filtro de Kalman estendido com restrições também é feita. Posteriormente, são expostos os resultados obtidos para o diagnóstico de falhas simuladas.

Finalmente, no Capítulo 6 são apresentadas as conclusões e sugestões para trabalhos futuros.

2 Revisão Bibliográfica

Inicialmente, nesta revisão bibliográfica se apresenta a visão geral da área e a seguir se discutem os estimadores de estado, o qual é o foco deste trabalho.

2.1 Técnicas de Detecção e Diagnóstico de Falhas

A detecção e diagnóstico de falhas (DDFs) têm se tornado uma disciplina cada vez mais difundida nas engenharias. Dentre suas principais vertentes (DING, 2013), é preciso citar:

- **Métodos baseados em sistemas redundantes:** consiste na recriação dos componentes do processo utilizando componentes de hardware idênticos (redundantes). A falha é detectada quando as saídas do processo são diferentes da do sistema em redundância.

- **Métodos baseados em processamento de sinais:** partindo da suposição que certos sinais carregam informações de possíveis falhas no sistema, o diagnóstico pode ser obtido através de processamento adequado do sinal. Sintomas típicos de falha são funções no domínio do tempo, como magnitude, valor médio, valores limite, tendências, ou no domínio da frequência, como linhas de espectro de frequência, densidade espectral, etc.

- **DDFs baseados em dados estatísticos:** a disponibilidade de um grande número de dados de processo, tanto históricos como coletados durante o processo, permitem a análise estatística destes. Esta é geralmente separada em duas fases: (i) treinamento, onde os dados históricos são apresentados como conhecimento prévio do processo e (ii) monitoramento online, onde as novas medidas são processadas no sistema de diagnóstico. Assim como o método de processamento de sinais, são ferramentas aplicadas basicamente a processos estáticos.

- **Métodos baseados em modelos:** consiste na aplicação de modelos analíticos que apresentam a descrição matemática da dinâmica do processo a sistemas de DDFs. A técnica consiste em duas partes: (i) geração do sinal de resíduo, ou seja, a diferença entre a variável medida e a fornecida pelo modelo e (ii) avaliação do resíduo e tomada de decisão.

O foco deste trabalho é em técnicas de DDFs baseadas em modelos. Engenheiros Químicos possuem o conhecimento necessário para a criação de modelos fenomenológicos de processos industriais, o que lhes concede uma vantagem fundamental na aplicação de tais técnicas na indústria.

Como já citado, técnicas baseadas em modelos consistem na detecção de falhas no processo, em atuadores ou sensores utilizando-se das dependências entre diferentes sinais medidos. Estas diferenças são expressas por modelos matemáticos do processo. Esta técnica apresenta diversas vantagens em relação a outros métodos, como: reagir bem a leves variações de características do processo com comportamentos abruptos ou insipientes no tempo; possibilitar o diagnóstico de falhas em atuadores, sensores e componentes do processo; ser aplicável em processos em estados transientes; detectar falhas em sistemas em malha fechada (ISERMANN, 2005).

A Figura 2.1 apresenta a estrutura básica do sistema de detecção de falhas. Baseado no sinal de entrada medido U e sinal de saída medido Y, o método de detecção gera o resíduo r, a estimação dos parâmetros Φ ou as estimativas do estado $\hat{z}$. Através da comparação com características do sistema em comportamento normal, variações são detectadas, levando a sintomas analíticos s.

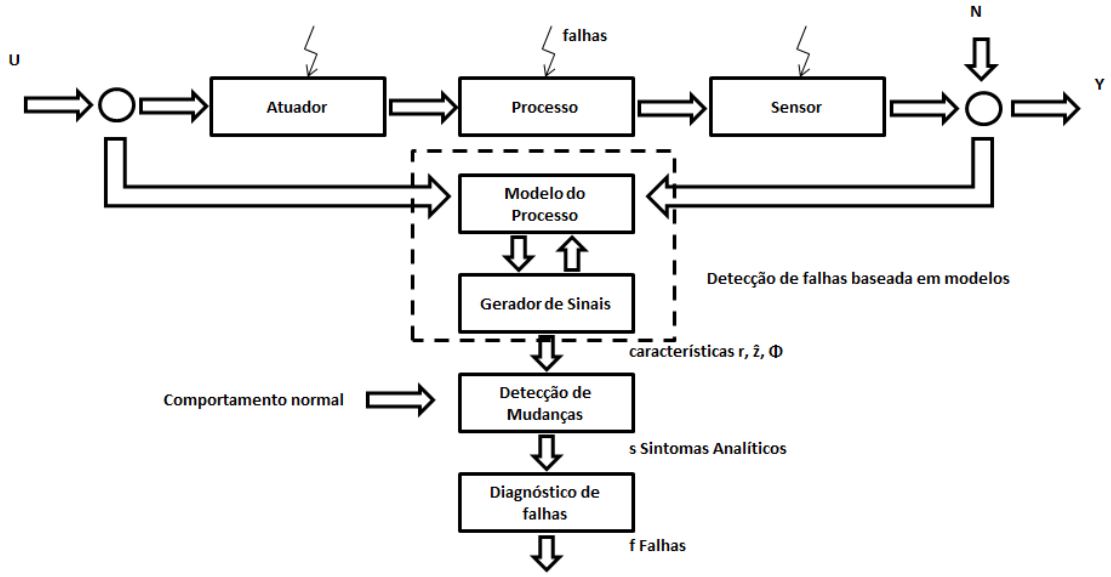

Figura 2.1: Esquema básico do processo de detecção de falhas baseado em modelos (Baseado em: ISERMANN, 2005.)

2.2 Redução da intensidade dos ruídos e condicionamento do sinal

A geração do resíduo se dá tipicamente pela comparação entre a as variáveis medidas do processo e as preditas pelo modelo matemático desenvolvido. Entretanto, para viabilizarmos tal comparação, é necessário levar em consideração a natureza ruidosa inerente a qualquer medida gerada por sensores. Independente do processo e aplicação, a fonte fundamental da informação obtida é a mesma e derivam de mensurações elétricas ruidosas de sensores mecânicos, óticos, magnéticos, etc. O ruído é tipicamente estatístico em natureza e, portanto, pode ser modelado como tal, levando assim a métodos estatísticos para abordarmos o problema.

O filtro de Kalman é o estimador de estado linear de variância mínima para sistemas de dinâmicas lineares com ruídos Gaussianos (RHODES, 1971). Ao utilizarmos uma ferramenta matemática para a estimação de estado visamos minimizar incertezas intrínsecas a três fontes básicas de informação: medidas de variáveis do sistema, modelo matemático representativo e estatísticas dos ruídos (SALAU, 2009).

Entretanto, a utilização de filtros de Kalman se limita a sistemas de dinâmicas lineares, o que muito raramente é encontrado em casos reais de engenharia. Para superar tais limitações, diversos filtros são encontrados na literatura, tais como o filtro de Kalman estendido (EKF, do inglês *Extended Kalman Filter*), o filtro de partículas (*Particle Filter*), UKF (*Unscented Kalman Filter*) entre outros. No presente trabalho foram estudados o filtro de Kalman Estendido e o filtro de Kalman estendido com restrições (CEKF, do inglês *Constrained Extended Kalman Filter*). Este último apresenta grande interesse para

sistemas reais, já que possibilita a restrição das variáveis estimadas a valores reais, o que permite em tese uma melhor convergência do filtro.

2.3 Equacionamento do Filtro de Kalman

Na indústria, os modelos matemáticos são desenvolvidos visando representar adequadamente o comportamento dos processos e equipamentos industriais. Através de testes empíricos, leis fundamentais ou conhecimento do sistema físico, estabelecem-se as relações entre variáveis de interesse, entradas e saídas do sistema. Desenvolvido o modelo, é possível a estudar a estrutura do sistema e modos de resposta, sendo necessário para isso observar o comportamento vigente do sistema através de sensores instalados no processo.

Entretanto, três razões fundamentais nos levam a concluir que o simples desenvolvimento de modelo e análise das variáveis medidas não são suficientes para realizar a análise do verdadeiro estado do sistema. Primeiro, nenhum modelo matemático é perfeito. O objetivo do modelo é representar as características dominantes e críticas ao sistema, portanto diversos efeitos são deixados consciente ou inconscientemente de fora deste. Outra imperfeição fundamental em qualquer modelo real é o fato de sistemas dinâmicos serem influenciados não apenas por entradas conhecidas (i.e., variáveis manipuladas e distúrbios medidos), mas também por distúrbios não medidos os quais não podem ser controlados ou mesmo eficientemente modelados. Por fim, no que tange aos sensores, além de normalmente não medirem todas as grandezas necessárias para descrever os modelos, ou seja, apenas uma parcela dos estados estão diretamente relacionados aos sinais medidos, apresentam ruídos de medição inerentes ao condicionamento e conversão dos sinais. Tudo isto ressalta a importância de se ter uma ferramenta de estimação de estados que utilize com eficiência as informações disponíveis do sistema estudado. Dentre as diversas ferramentas matemáticas usadas para a estimação estocástica de processos com mensurações ruidosas, uma das mais conhecidas e mais utilizadas é o filtro de Kalman.

O filtro de Kalman é essencialmente um conjunto de equações matemáticas que implementam um estimador do tipo preditor-corretor que é ótimo no sentido de minimizar covariância do erro estimado, quando condições necessárias são atendidas. Desde sua introdução, o filtro de Kalman tem sido sujeito de diversas pesquisas e aplicações, principalmente na área de navegação autônoma e assistida. Isto devido em grande parte pelos avanços na computação digital facilitaram a sua implementação, mas também devido a sua relativa simplicidade e robustez. Raramente as condições necessárias para o estado ótimo existem, e mesmo assim o filtro funciona bem em diversas aplicações independente da situação (WELCH; BISHOP, 2001).

Como algoritmo recursivo de processamento de dados, o filtro de Kalman é considerado ótimo. Ele processa todas as variáveis medidas, independente de sua precisão, para estimar o estado atual das variáveis de interesse, fazendo o uso do conhecimento do sistema e dinâmica dos componentes, a descrição estatística do ruído do sistema, dos erros de medida e incertezas na dinâmica do modelo, e qualquer informação disponível sobre as condições iniciais do sistema (MAYBECK, 1979).

2.3.1 O algoritmo do Filtro de Kalman discreto

O filtro de Kalman aborda o problema geral de tentar estimar o estado $\mathrm{x} \in \Re^{\mathrm{n}}$ de um processo discreto governado por um modelo linear descrito por (WELCH; BISHOP, 2001) e (LABBE JR, 2015):

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{2.1}$$

tendo como saídas as medidas $z \in \Re^m$ as quais se relacionam aos estados do sistema pela seguinte expressão:

$$z_k = Hx_k + v_k \tag{2.2}$$

As variáveis w_k e v_k representam o ruído de processo e medida, respectivamente. Elas são assumidas independentes uma da outra, brancas e de distribuição normal, tendo, portanto, ambas média zero e desvio padrão Q e R, para w_k e v_k respectivamente, tal que,

$$p(w) \sim N(0, Q) \tag{2.3}$$

$$p(v) \sim N(0, R) \tag{2.4}$$

A matriz A na equação (2.1) relaciona linearmente o estado no período de tempo prévio $\mathrm{k}-1$ com o estado no tempo corrente k, na ausência de um ruído de processo. A pode variar em cada período de tempo, mas aqui assumimos constante. A matriz B relaciona a entrada $u \in \Re^{\mathrm{l}}$ com os estados x. A matriz H na equação de mensuração relaciona o estado com a medida $\mathrm{z_k}$. H também pode variar com o tempo, mas assumimos constante, pois na formulação de Kalman considera que o sistema é linear invariante no tempo.

Definimos $\hat{\mathrm{x}}^{-} \in \Re^{\mathrm{n}}$ como a estimativa de estado *a priori* no passo k, dado o conhecimento do processo prévio ao estado k, e $\hat{\mathrm{x}} \in \Re^{\mathrm{n}}$ a estimativa *a posterirori* dada a medida $\mathrm{z_k}$. Podemos assim definir os erros *a priori* e *a posteriori* como:

$$e_k^- \equiv x_k - \hat{x}_k^- \tag{2.5}$$

$$e_k \equiv x_k - \hat{x}_k \tag{2.6}$$

As matrizes de covariância dos estados (P) *a priori* e *a posteriori* são dadas, respectivamente, por:

$$P_k^- = E[e_k^- e_k^{-T}] \tag{2.7}$$

$$P_k = E[e_k e_k{}^T] \tag{2.8}$$

O objetivo da do filtro é encontrar a melhor estimativa *a posteriori* possível através da combinação linear de uma estimativa *a priori* e uma diferença ponderada entra a medida atual e a medida predita, ou seja,

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{2.9}$$

A diferença $(\mathrm{z_k} - \mathrm{H}\hat{\mathrm{x}}_{\mathrm{k}}^{-})$ é chamada de erro de simulação ou resíduo.

A matriz nxm K é escolhida de forma a minimizar a matriz de covariância do erro a posteriori e é chamado de ganho de Kalman ou fator de mistura. A solução ótima para o problema proposto por Kalman para K é dado por:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^- \tag{2.10}$$

O filtro de Kalman produz estimativas do estado do processo usando uma espécie de controle de retroalimentação: a estimativa é feita e se obtém a resposta na forma de medidas (com ruído). Portanto, as equações do filtro de Kalman recaem em dois grupos: equações de atualização no tempo e atualização de medida. As equações de atualização no tempo são responsáveis por projetar no tempo à frente o estado corrente do processo e da covariância do erro para obter as estimativas *a priori* para o próximo passo. As equações de atualização de medidas são responsáveis pelo *feedback*, ou seja, incorporar uma nova medida na estimativa *a priori* para obter uma melhor estimativa *a posterirori*.

As equações de atualização no tempo podem ser vistas também como preditivas e as equações de atualização de medida como corretoras. Formando assim o algoritmo preditivo-corretor de solução de problemas numéricos.

As equações necessárias para o algoritmos são apresentadas abaixo.

Atualização no tempo (ou etapa de simulação):

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{2.11}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2.12}$$

Atualização da medida(ou etapa de correção):

$$K_k = P_k^- H^T (H P_k^- H^T + R)^- \tag{2.13}$$

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{2.14}$$

$$P_k = (I - K_k H)P_k^-. \tag{2.15}$$

2.4 Equacionamento do Filtro de Kalman Estendido (EKF)

O filtro de Kalman, como originalmente concebido, faz uso de equações lineares e é, portanto, de aplicabilidade limitada devido a não-linearidade encontrada na maioria de problemas reais. Como estamos tratando de dois modelos ao utilizarmos o algoritmo (modelo do processo e de medida), se fez necessário derivar do filtro original um algoritmo abrangente, capaz de lidar com a não-linearidade de um ou ambos modelos. A maneira mais óbvia de sobrepor tais limitações é através da linearização de ambos os modelos, e é exatamente essa a estratégia adotada no EKF.

O EKF lida, portanto, com a não-linearidade do sistema através da linearização deste no ponto da estimativa atual para que então o filtro linear seja usado. Esta foi uma das primeiras técnicas sugeridas para trabalhar com sistemas não lineares e continua uma das mais populares.

Começamos assumindo novamente que o processo possui um vetor de estado $x \in \Re^n$, mas agora descrito pelo seguinte modelo não linear:

$$\dot{x_k} = f(x_{k-1}, u_k) + w_{k-1} \tag{2.16}$$

com a medida $z \epsilon \Re^m$ tal que

$$z_k = h(x_k) + v_k \tag{2.17}$$

Na prática, não se sabe o valor exato das variáveis de ruído $\mathrm{w_k}$ e $\mathrm{v_k}$ em cada período de tempo. Entretanto, pode-se aproximar os vetores de estado e medida sem estes

$$x_k = f(x_{k-1}, u_k, 0) \tag{2.18}$$

$$z_k = h(x_k, 0) \tag{2.19}$$

É importante notar que a principal falha do algoritmo EKF é o fato de as distribuições das variáveis randômicas agora deixarem de ser normalmente distribuídas após as transformações não lineares. O EKF é simplesmente um estimador de estado *ad hoc* que apenas aproxima a regra de Bayes ótima por linearização (WELCH; BISHOP, 2001).

As equações governantes do processo se tornam então

$$x_k \approx \tilde{x}_k + A(x_k - \hat{x}_{k-1}) + W w_{k-1} \tag{2.20}$$

$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + V v_k \tag{2.21}$$

Onde $\mathrm{x_k}$ e $\mathrm{z_k}$ são os vetores de estado e medida atuais, $\tilde{\mathrm{x}}_\mathrm{k}$ e $\tilde{\mathrm{z}}_\mathrm{k}$ são aproximações dos vetores de estado e medida, e $\hat{\mathrm{x}}_\mathrm{k}$ e $\hat{\mathrm{z}}_\mathrm{k}$ são estimativas dos vetores de estado e medida.

A é a matriz jacobiana das derivadas parciais de f em respeito ao estado x, ou seja:

$$A_{ij} = \frac{\partial f_i}{\partial x_j}(\hat{x}_{k-1}, u_k, 0) \tag{2.22}$$

Já W é a matriz jacobiana de derivadas parciais de f em respeito a w, ou seja:

$$W_{ij} = \frac{\partial f_i}{\partial w_j}(\hat{x}_{k-1}, u_k, 0) \tag{2.23}$$

H é a matriz jacobiana de derivadas parciais de h em respeito a x:

$$H_{ij} = \frac{\partial h_i}{\partial x_j}(\tilde{x}_k, 0) \tag{2.24}$$

V é a matriz jacobiana de derivadas parciais de h em respeito a v:

$$\mathrm{V_{ij}} = \frac{\partial \mathrm{h_i}}{\partial \mathrm{v_j}}(\tilde{\mathrm{x}}_\mathrm{k}, 0) \tag{2.25}$$

Por fim, as equações necessárias para o algoritmos são apresentadas abaixo.

<u>Atualização no tempo (etapa de simulação)</u>

$$\hat{\mathrm{x}}_\mathrm{k}^- = \mathrm{f}(\hat{\mathrm{x}}_{\mathrm{k}-1}, \mathrm{Bu_k}) \tag{2.26}$$

$$\mathrm{P_k^-} = \mathrm{AP_{k-1}A^T} + \mathrm{W_k Q W_k^T} \tag{2.27}$$

Atualização da medida (etapa de correção)

$$K_k = P_k^- H^T (H P_k^- H^T + V_k R V_k^T)^- \tag{2.28}$$

$$\hat{x}_k = \hat{x}_k^- + K(z_k - h(\hat{x}_k^-)) \tag{2.29}$$

$$P_k = (I - K_k H) P_k^- . \tag{2.30}$$

2.5 Estimadores baseados em Horizonte Móvel

O EKF é um algoritmo computacionalmente eficiente, mas pode apresentar problemas de convergência mesmo em sistemas não-lineares simples. Além disto, estimações de estado dinâmicas devem frequentemente contar com restrições de variáveis de estado ou entradas do processo. Por exemplo, frações molares não tomam valores negativos e a soma é necessariamente igual a um. As estimativas geradas pelo EKF não possuem restrições, o que pode levar a instabilidade do filtro (UNGARALA; DOLENCE; LI, 2007). Os métodos de estimação de horizonte móvel (MHE, do inglês *Moving Horizon Estimation*) são uma alternativa para resolver os problemas apresentados pelo EKF.

O MHE é um algoritmo baseado em otimização. Ele trata o caso de estimação de estados como um problema de mínimos quadrados. A estimação é formulada como uma minimização de uma função quadrática ponderada das variáveis desconhecidas sujeitas à equação do modelo. De forma a não tornar o tamanho da otimização excessivamente grande, se escolhe um tamanho do horizonte na qual a otimização é realizada. O tamanho do horizonte oferece um *trade-off* entre precisão e eficiência computacional e é um parâmetro a mais para sintonizar (ROBERTSON; LEE, 1995).

O problema é primeiro modelado na forma de um conjunto de equações diferenciais não-lineares.

$$\dot{x} = f(x, u, t) + w(t) \tag{2.31}$$

$$x(0) = x_0 \tag{2.32}$$

$$y = h(x, t) + v(t) \tag{2.33}$$

Os sistemas são então discretizados. Para sistemas não-lineares, isto é feito através da integração do sistema no período de amostragem $t_k - t_{k-1}$.

$$x_k = x_k + \int_{k-1}^{k} (f(x(\tau), u(\tau), \tau) + w(\tau)) d\tau \tag{2.34}$$

$$x_k = f(x_k, u_k, t_k) + w_k \tag{2.35}$$

$$x(0) = x_0 \tag{2.36}$$

$$y_k = h(x_k, t_k) + v_k \tag{2.37}$$

A equação a seguir apresenta as equações de integração dinâmica envolvidas na formulação do MHE, para sistemas do tipo anterior

$$\hat{x}_{k-N|k} = \hat{x}_{k-N|k-1} + \hat{w}_{k-N-1|k} \tag{2.38}$$

$$\hat{x}_{k-N+1|k} = \hat{x}_{k-N|k} + \int_{k-N}^{k-N+1} f(x,u,\tau)d\tau + \hat{w}_{k-N|k}$$

$$\vdots$$

$$\hat{x}_{k|k} = \hat{x}_{k-1|k} + \int_{k-N+1}^{k} f(x,u,\tau)d\tau + \hat{w}_{k-1|k}$$

O problema de MHE é formulado no tempo k para um horizonte m, portanto, é formulado da seguinte maneira

$$\min_{\substack{w_{k-N-1|k},\dots,w_{k-1|k} \\ v_{k-N|k},\dots,v_{k|k}}} \Psi_k^N \quad \hat{w}_{k-N-1|k}^T \left(P_{k-N-1|k-1}\right)^{-1} \hat{w}_{k-N-1|k} \tag{2.39}$$

$$+ \sum_{j=k-N}^{k-1} \hat{w}_{j|k}^T (Q_{k-1})^{-1} \hat{w}_{j|k}$$

$$+ \sum_{j=k-N}^{k} \hat{v}_{j|k}^T (R_k)^{-1} \hat{v}_{j|k}$$

Sujeito a

$$\hat{x}_{k-N|k} = \hat{x}_{k-N|k-1} + \hat{w}_{k-N-1|k}$$

$$\hat{x}_{j+1|k} = f\left(x_{j|k}, u_j\right) + \hat{w}_{j|k} \qquad j = k-N, \dots, k-1$$

$$y_j = h\left(\hat{x}_{j|k}\right) + \hat{v}_{j|k} \qquad j = k-N, \dots, k$$

Estando as variáveis sujeitas às restrições de desigualdade na forma

$$x_{min} \leq \hat{x}_{j|k} \leq x_{max}$$

$$\hat{w}_{min} \leq \hat{w}_{j-1|k} \leq \hat{w}_{max} \qquad j = k-N, \dots, k$$

$$\hat{v}_{min} \leq \hat{v}_{j-1|k} \leq \hat{v}_{max} \qquad j = k-N, \dots, k$$

O problema de programação dinâmica não-linear apresentado pode ser resolvido de forma sequencial ou simultânea. Por exigir a resolução das igualdades a cada passo da otimização, com a integração numérica do sistema de equações, a forma sequencial demanda um grande esforço numérico e, consequentemente, um grande tempo computacional. No método simultâneo de solução, as restrições de igualdade são transformadas em restrições de igualdade algébrica via técnica de discretização, viabilizando então a resolução por algoritmos tradicionais de otimização quadrática.

A matriz de covariância pode ser obtida através da solução da equação da equação de Riccati, sendo que a solução é dada por (RAO; RAWLINGS; MAYNE, 2003):

$$P_{k|k}^N = \varphi_k P_{k-1|k-1}^N \varphi_k^T - \varphi_k P_{k-1|k-1}^N H_k^T \left[H_k P_{k-1|k-1}^N H_k^T + R_k\right]^{-1} H_k P_{k-1|k-1}^N \varphi_k^T + Q_k \tag{2.40}$$

A matriz φ é conhecida como matriz fundamental, e muitas vezes retratada como F. Está é obtida através da integração do modelo linearizado entre os espaços de tempo k e $\mathrm{k}-1$. A obtenção da matriz φ será retratada no próximo capítulo.

2.5.1 O Filtro de Kalman estendido com restrições (CEKF)

Ao estipularmos o horizonte do MHE como sendo igual a zero, obtemos o CEKF. Se as equações de medida são lineares, obtemos um problema de programação quadrática de fácil resolução que pode ser resolvido com baixo esforço computacional (GESTHUISEN; KLATT; ENGELL, 2001).

O problema de otimização se reduz então a

$$\min_{\widehat{\Theta}_{\mathrm{k|k}}} \Psi \quad \widehat{\Theta}_{\mathrm{k|k}}^{\mathrm{T}}\left(\mathrm{S}_{\mathrm{k|k}}\right)^{-1}\widehat{\Theta}_{\mathrm{k|k}} + \mathrm{d}^{\mathrm{T}}\widehat{\Theta}_{\mathrm{k|k}} \tag{2.41}$$

$$\widehat{\Theta}_{\mathrm{k|k}} = \begin{bmatrix} \mathrm{w}_{\mathrm{k-1|k}} \\ \mathrm{v}_{\mathrm{k|k}} \end{bmatrix}$$

$$\mathrm{S}_{\mathrm{k|k}} = \begin{bmatrix} \mathrm{P}_{\mathrm{k-1|k-1}} & 0 \\ 0 & \mathrm{R}_{\mathrm{k}} \end{bmatrix}$$

$$\mathrm{d} = 0$$

sujeito às restrições

$$[\mathrm{H} \quad \mathrm{I}]\widehat{\Theta}_{\mathrm{k|k}} = \mathrm{y}_{\mathrm{k}} - \mathrm{h}(\hat{\mathrm{x}}_{\mathrm{k|k-1}})$$

$$\mathrm{x}_{\min} \leq \hat{\mathrm{x}}_{\mathrm{k|k}} \leq \mathrm{x}_{\max}$$

$$\widehat{\mathrm{w}}_{\min} \leq \widehat{\mathrm{w}}_{\mathrm{k-1|k}} \leq \widehat{\mathrm{w}}_{\max}$$

$$\hat{\mathrm{v}}_{\min} \leq \hat{\mathrm{v}}_{\mathrm{k-1|k}} \leq \hat{\mathrm{v}}_{\max}$$

Dando-se finalmente a atualização da estimativa do estado no tempo k -por

$$\hat{\mathrm{x}}_{\mathrm{k|k}} = \hat{\mathrm{x}}_{\mathrm{k|k-1}} + \widehat{\mathrm{w}}_{\mathrm{k-1|k}} \tag{2.42}$$

$$\mathrm{y}_{\mathrm{k}} = \mathrm{h}(\hat{\mathrm{x}}_{\mathrm{k|k}}) + \hat{\mathrm{v}}_{\mathrm{k|k}} \tag{2.43}$$

Apesar de pertencer à família do MHE, o CEKF é considerado similar ao EKF, já que ambos os métodos utilizam um horizonte nulo para atualização das estimativas. Ambos apresentam o mesmo resultado na ausência de restrições e baixo ruído de processo (SALAU, 2009).

3 Metodologia para Detecção e Diagnóstico de Falhas

Neste TCC a metodologia empregada para DDFs consiste em empregar os estimadores de estado EKF e CEKF apresentados na revisão bibliográfica. Estes dois estimadores de estado são implementados em Python e o critério para detecção da falha está baseado em critério estatístico. A seguir se discute tanto a implementação quanto o critério estatístico utilizado.

3.1 Implementação em Python

O sistema foi implementado utilizando-se a linguagem de programação Python. O Python tem ganhado relevância na área de computação científica por dois fatores principais: possuir um tempo de implementação baixo (já que possui todo design focado em produtividade e facilidade de leitura de códigos) e ser uma plataforma sem custo (*Software* livre), facilitando assim o desenvolvimento de novas extensões e aumentando a acessibilidade. Devemos salientar que Python é uma linguagem interpretada, sendo assim consideravelmente mais lenta em relação a linguagens compiladas. Entretanto, diversos pacotes proporcionam uma interface entre o Python e bibliotecas compiladas e numericamente eficientes como por exemplo *numpy*, que oferece a interface para a biblioteca LAPACK de álgebra linear numérica (FANGOHR, 2015). Deve-se tomar cuidado, portanto, de sempre se fazer uso de bibliotecas compiladas nas partes de maior demanda computacional do código, ainda mais para simulações que, em geral, dependem de diversos métodos numéricos.

A simulação é realizada através da integração do sistema de equações lineares. Para isso, foi utilizado a função odeint do módulo intagrate do pacote *scipy*. A função faz uso do lsoda da biblioteca em FORTRAN ODEPACK. Lsoda automaticamente troca entre o método corretor-preditor (método de Adams) e o método de BDF (Backward differentiation formula), para sistemas não-rígidos e rígidos, respectivamente (HINDMARSH, 1983).

O filtro CEKF é resolvido a partir da solução de um problema de programação quadrática. Para a implementação deste, foram utilizadas as bibliotecas de otimização para Python *Openopt e CVXOPT*. A primeira serve apenas de interface para o solver contido na segunda.

3.2 Processo para Identificação de Falha no Sistema

Caso uma falha ocorra, a primeira tarefa é detectar o comportamento anormal do sistema e estimar o momento no qual a falha ocorreu. Um teste estatístico simples pode ser utilizado para a determinação do momento inicial da falha (PRAKASH; PATWARDHAN; NARASIMHAN, 2002). O teste estatístico é dado por:

$$\epsilon_k = \gamma_k^T V_k \gamma_k \tag{3.1}$$

sendo,

$$\gamma_k = y_k - h(\hat{x}_{k|k-1})$$

A variável γ_k é o resíduo no tempo k. V_k representa a matriz de covariância do erro, dada por:

$$V_k = HP_k^- H^T + R \tag{3.2}$$

O teste estatístico ϵ_k segue uma distribuição χ^2 central de n graus de liberdade. Para qualquer nível de significância escolhido, o critério para o teste pode ser escolhido desta distribuição. Surge a suspeita de falha no tempo t, portanto, caso o teste estatístico ϵ_k supera o critério limite do teste.

No presente trabalho, é considerado que o sistema apresenta falha quando o teste estatístico apresenta por três instantes de tempo seguidos um valor superior ao dado pela distribuição χ^2 com determinado nível de confiança (no caso foi utilizado nível de confiança de 99%).

4 Estudo de Caso

O sistema estudado consiste em três tanques unidos em sua base, formando um sistema de vasos comunicantes, conforme ilustrado na Figura 4.1. O tanque 1 está unido com o tanque 3 que, por sua vez, esta unido ao tanque 2. Tanto o tanque 1 quanto o tanque 2 apresentam entradas de vazão - Q_1e Q_2. O tanque 2 apresenta uma saída de de vazão Q_{20}. O sistema de três tanque é um sistema experimental bastante utilizado em laboratórios de controle, muito utilizado para o estudo de sistemas multivariáveis não-lineares, controle feedback e diagnóstico de falhas (HOU; XIONG; PATTON, 2005).

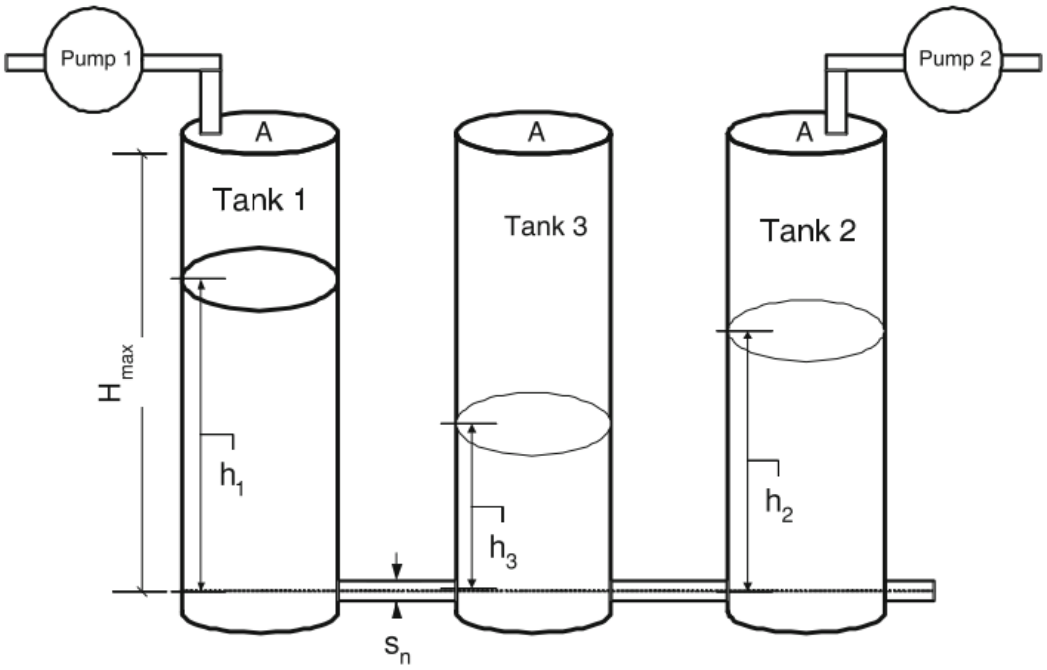

Figura 4.1: Representação esquemática do sistema de três tanques

4.1 Modelo Matemático

Analisando-se os fluxos de entrada e saída de massa de cada tanque e considerando-se a lei de Torricelli, o sistema pode ser modelado através das seguintes equações:

$$A_1 \frac{dh_1}{dt} = Q_1 - Q_{13} \tag{4.1}$$

$$A_2 \frac{dh_2}{dt} = Q_2 + Q_{32} - Q_{20} \tag{4.2}$$

$$A_3 \frac{dh_3}{dt} = Q_{13} - Q_{32} \tag{4.3}$$

Sendo,

$$Q_{13} = a_1 s_{13} sinal(h_1 - h_3)\sqrt{2g|h_1 - h_3|} \tag{4.4}$$

$$Q_{32} = a_3 s_{32} sinal(h_3 - h_2)\sqrt{2g|h_3 - h_2|} \tag{4.5}$$

$$Q_{20} = a_2 s_0 \sqrt{2gh_2} \tag{4.6}$$

Onde,

- Q_1, Q_2 são os fluxos de entrada (cm³/s);
- Q_{ij} são os fluxos volumétrico de líquido (cm³/s) do tanque i para o tanque j;
- $h_i(t)$, i = 1, 2, 3, são os níveis de água (cm) em cada tanque e variáveis medidas.

O restante das variáveis e respectivas unidades está descrito na Tabela 4.1.

Tabela 4.1: Parâmetros do sistema de três tanques

Parâmetro	**Símbolo**	**Valor**	**Unidade**
Área da seção transversal dos tanques	*Ai (i=1, 2, 3)*	154	cm²
Área da seção transversal dos tubos	*Si (i =0, 13, 32)*	0,5	cm²
Altura máxima dos tanques	*Himáx(i=1,2,3)*	62	cm
Máxima vazão de entrada	*Q1,2máx*	100	cm³/s
Coeficiente de vazão para o tubo 1	*a1*	0,46	
Coeficiente de vazão para o tubo 2	*a2*	0,6	
Coeficiente de vazão para o tubo 3	*a3*	0,45	

De forma a utilizarmos as técnicas de EKF e CEKF, é necessário descrevermos o sistema em sua forma espaço de estado. Para tanto, devemos tratar com a forma linearizada do sistema, que pode ser obtida através da linearização em um ponto de operação como apresentado a seguir:

$$\frac{dx}{dt} = Ax + Bu, \tag{4.7}$$

$$y = Cx \tag{4.8}$$

Sendo (4.7) o modelo do processo e (4.8) o modelo de mensuração das variáveis de estado. Para um dado ponto de operação:

$$x = \begin{bmatrix} h_1 - h_{1,0} \\ h_2 - h_{2,0} \\ h_3 - h_{3,0} \end{bmatrix} \tag{4.9}$$

$$u = \begin{bmatrix} Q_1 - Q_{1,0} \\ Q_2 - Q_{2,0} \end{bmatrix} \tag{4.10}$$

$$A = \frac{\partial f}{\partial h}\Big|_{h=h_0} \tag{4.11}$$

$$B = \begin{bmatrix} {}^{1}/_{A} & 0 \\ 0 & {}^{1}/_{A} \\ 0 & 0 \end{bmatrix} \tag{4.12}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

A matriz A é a jacobiana do modelo de processo e é dado por:

$$A = \begin{bmatrix} -\frac{a_1 s_{13} g}{A\sqrt{2g|h_{1,0}-h_{3,0}|}} & 0 & \frac{a_1 s_{13} g}{A\sqrt{2g|h_{3,0}-h_{1,0}|}} \\ 0 & -\frac{a_3 s_{32} g}{A\sqrt{2g|h_{2,0}-h_{3,0}|}} - \frac{a_2 s_0 g}{A\sqrt{2gh_{2,0}}} & \frac{a_3 s_{32} g}{A\sqrt{2g|h_{3,0}-h_{2,0}|}} \\ \frac{a_1 s_{13} g}{A\sqrt{2g|h_{1,0}-h_{3,0}|}} & \frac{a_3 s_{32} g}{A\sqrt{2g|h_{2,0}-h_{3,0}|}} & -\frac{a_3 s_{32} g}{A\sqrt{2g|h_{3,0}-h_{2,0}|}} \end{bmatrix} \tag{4.14}$$

A partir do modelo linear é possível encontrar recursivamente o estado x no momento t_k a partir do estado x no instante t_{k-1}:

$$x_k = F x_{k-1} + \Gamma_u u_k \tag{4.15}$$

A matriz F é conhecida como a matriz fundamental, e muitas vezes retratada também como φ. A discretização desta é fundamental para a aplicação de técnicas de filtro de Kalman, pois é utilizada na atualização da matriz de covariância do estado. Está é obtida através da integração do modelo linearizado entre os espaços de tempo k e $k-1$.

$$F = e^{A\Delta t} \tag{4.16}$$

A matriz exponencial $e^{A\Delta t}$ pode ser aproximada através de séries de potência de ordem n, como, por exemplo, a expansão em séries de Taylor:

$$e^{A\Delta t} = I + A\Delta t + \frac{(A\Delta t)^2}{2!} + \frac{(A\Delta t)^3}{3!} + \cdots \tag{4.17}$$

4.2 Descrição das Falhas Estudadas

Para o sistema de 3 tanques interligados, geralmente são propostos quatro tipos de falhas:

- Falha em componentes principais do sistema: no caso, tratam-se de vazamentos nos tanques, que podem ser descritos como uma saída adicional de vazão. Para o caso em estudo, estes são retratados da seguinte forma:

$$\theta_{A1}\sqrt{2gh_1}, \theta_{A2}\sqrt{2gh_2}, \theta_{A3}\sqrt{2gh_3} \tag{4.18}$$

onde θ_{A1}, θ_{A2}, θ_{A3} são desconhecidos e dependem do tamanho do vazamento.

- *Falha em demais componentes do sistema*: pode ocorrer o entupimento total ou parcial dos tubos que fazem a conexão entre tanques e tubo da corrente de saída,

causando desvios nas variáveis Q_{13}, Q_{32} e Q_{20}. As falhas foram modeladas da seguinte forma:

$$\theta_{A4} a_1 s_{13} sinal(h_1 - h_3)\sqrt{2g|h_1 - h_3|} \quad (4.19)$$

$$\theta_{A6} a_3 s_{32} sinal(h_3 - h_2)\sqrt{2g|h_3 - h_2|} \quad (4.20)$$

$$\theta_{A5} a_2 s_0 \sqrt{2gh_2} \quad (4.21)$$

sendo θ_{A4}, θ_{A5} e θ_{A6} $\epsilon[-1, 0]$e também desconhecidas

- *Falhas em sensores:* falhas aditivas denotadas por f_1, f_2e f_3.

- *Falha em atuadores*: falhas nas variáveis de entrada, no caso, das bombas que fornecem o fluxo mássico de entrada. São denotadas por f_4e f_5.

As falhas então são incluídas no sistema de três tanques descritos acima da seguinte forma.

$$\dot{x} = (A + \Delta A_F)x + Bu + E_f f \quad (4.22)$$

$$y = Cx + F_f f \quad (4.23)$$

sendo

$$\Delta A_F = \sum_{i=1}^{6} Ai\, \theta_{Ai} \quad (4.24)$$

$$f = \begin{bmatrix} f_1 \\ \vdots \\ f_5 \end{bmatrix} \quad (4.25)$$

$$E_f = [0 \quad B] \in \Re^{3X5} \quad (4.26)$$

$$F_f = [I_{3X3} \quad 0] \in \Re^{3X5} \quad (4.27)$$

As falhas modeladas como um termo aditivo na representação em estado-espaço - $E_f f$ e $F_f f$ - são denominadas falhas aditivas, enquanto as demais são chamadas de falhas multiplicativas. As últimas podem ocasionar em mudanças na dinâmica própria do sistema. (DING, 2013) (VILLEZ et al., 2011)

5 Resultados e Discussão

Neste capítulo, apresentaremos inicialmente testes e discussões da implementação do modelo em estudo e dos filtros, avaliando o comportamento do filtro frente a diferentes parâmetros para sua inicialização e breve comparação entre EKF e CEKF. Por fim, aplicamos o CEKF a sistemas simulados com três anormalidades distintas e analisamos sua aplicação na detecção de falhas para o sistema de três tanques. Toda a implementação do sistema e filtro, assim como código escrito para o desenvolvimento dos resultados se encontram no Apêndice A.

5.1 Testes da implementação dos filtros e discussão

De forma a testar a implementação dos filtros para o sistema em estudo, foi analisado tanto o comportamento das estimativas *a priori* (simuladas) e *a posteriori* (corrigidas pelos valores medidos) geradas pelo algoritmo, comparando-as aos valores medidos. Também foi analisado o efeito de parâmetros do filtro.

A Figura 5.1, além de ter a função de testar a implementação do filtro, torna mais evidente como o filtro, através da combinação do modelo do sistema e de medidas ruidosas, determina uma estimativa para o estado de interesse. O modelo em questão é o sistema de três tanques, sendo realçado o resultado para a estimativa do estado referente ao tanque 1. A cada instante k, o algoritmo produz uma projeção do estado para no instante $k+1$, dado o modelo do sistema, chamada aqui estimativa *a priori*. Através da combinação da estimativa *a priori* e da medida no mesmo instante, é gerada a estimativa *a posteriori*, levando-se em conta a incerteza que temos tanto da estimativa *a priori* (representada por P_k^-) quanto da medida (representada por R). Por isso, é possível observar que, ao iniciarmos o filtro, o valor medido ganha mais peso para a determinação do estado, já que o valor da matriz de covariância é alto. Com a diminuição dos valores na matriz de covariância dos estados conforme o filtro converge, diminui-se a incerteza quanto ao a estimativa do estado, ganhando esta assim mais relevância na estimativa do estado *a posteriori*.

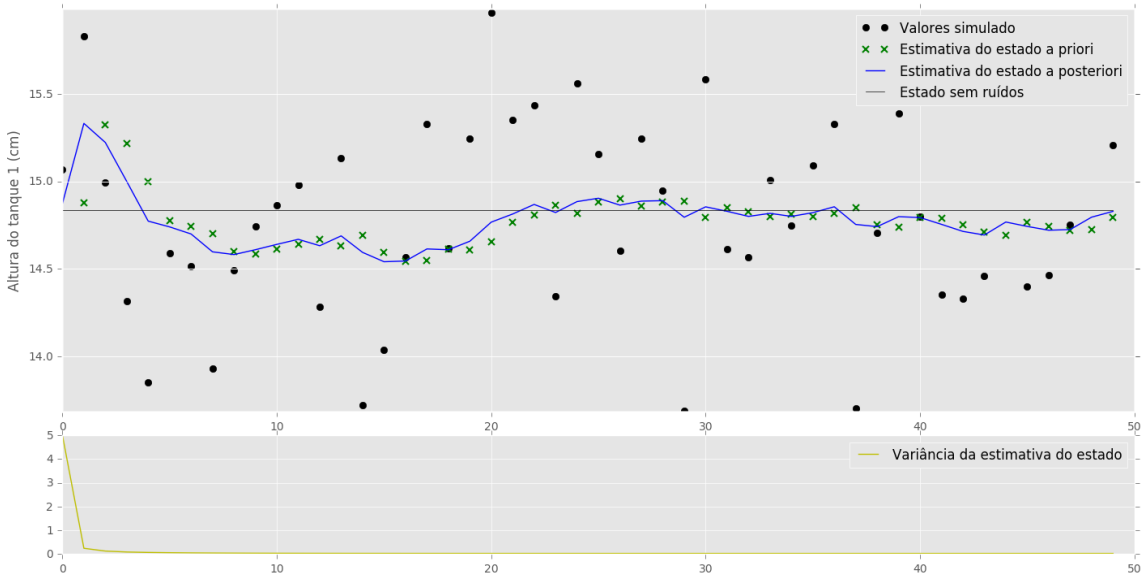

Figura 5.1: Estimação dos estados gerados por EKF e variância da estimativa dos estados (matriz P) para R0 = 0,25 e Q0 = 0,0025

A figura 5.2 foi também gerada utilizando-se o EKF. Nela, é possível observar a influência dos valores da matriz de covariância do modelo de medida (R0) e da matriz de covariância do modelo do sistema (Q0) para o funcionamento do filtro. Novamente confirmando que quanto maior a incerteza na medida, mais peso ganha para a estimativa do estado o estado anterior e a projeção *a posteriori* gerada pelo modelo. O contrário também é verdade, ou seja, maior a incerteza no modelo, maior o peso do valor da medida para a estimativa do estado. Os valores da matriz de covariância R0 e Q0 são determinantes, portanto, para o bom funcionamento do filtro. Estes valores são determinados a partir do conhecimento prévio da dinâmica do sistema, do modelo matemático e das incertezas envolvendo os sensores. Como no presente trabalho as variáveis são simuladas e o valor dos ruídos previamente conhecidos, não há dificuldade para determinarmos os valores de R0 e Q0, dificuldade esta a mais que se tem ao utilizarem-se valores reais.

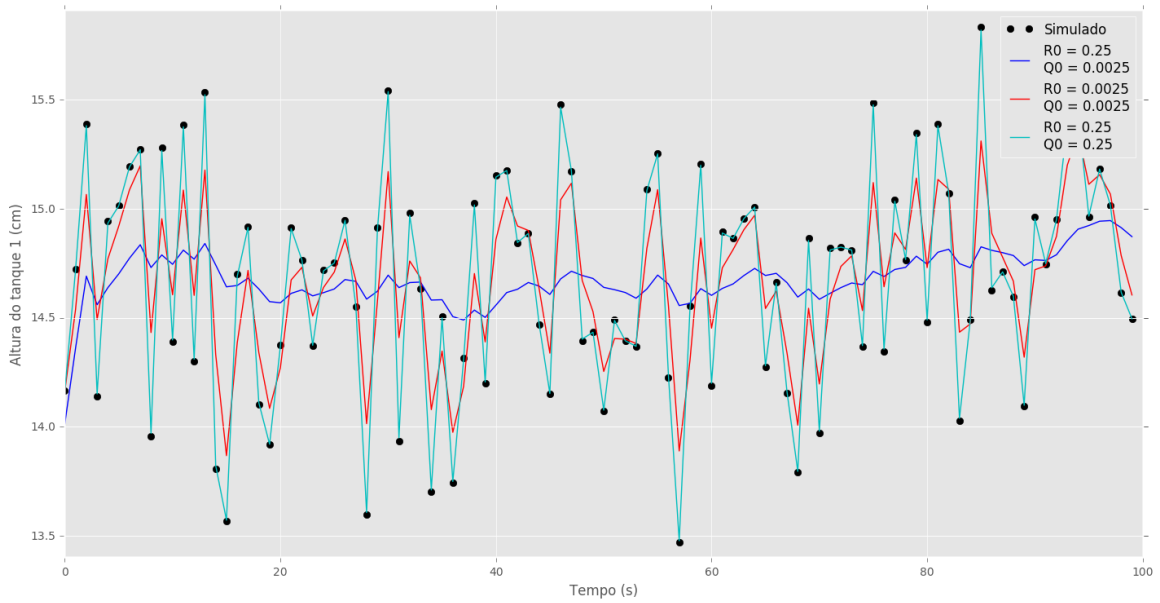

Figura 5.2: Influência de parâmetros no comportamento do filtro

5.2 Validação da distribuição de probabilidade do teste estatístico aplicado

Para averiguar se de fato a distribuição χ^2 com n graus de liberdade representa bem a distribuição de probabilidade do teste estatístico proposto, foi realizado uma análise de Monte Carlo da seguinte forma: dez simulações sem falha do sistema com horizonte de tempo (N) igual 10000 foram realizadas, o teste estatístico ϵ_k foi calculado a cada passo e, por fim, calculado o número de vezes no qual ϵ_k é menor que limites de confiança específicos. Os resultados podem ser observados na tabela 5.1. Podemos notar que a distribuição χ^2 de fato representou bem a distribuição de probabilidade do teste estatístico ϵ_k.

Os dados nos dão evidências que o teste estatístico pode de fato ser modelado por uma distribuição de probabilidade χ^2. Esta confirmação nos permite validar este como um teste pertinente para a determinação de falha do sistema.

Tabela 5.1: Teste de representatividade da distribuição χ^2

		Percentual abaixo dos valores críticos	
Simulação	**Iterações**	**Valor crítico = 11,34 (99%)**	**Valor crítico = 7,82 (95%)**
1	10000	98,97%	94,64%
2	10000	98,94%	94,55%
3	10000	98,93%	95,01%
4	10000	98,87%	94,87%
5	10000	99,07%	94,87%
6	10000	98,97%	94,93%
7	10000	98,81%	94,58%
8	10000	99,07%	94,67%
9	10000	99,01%	95,13%
10	10000	99,11%	94,82%

5.3 Vantagens do CEKF em relação ao EKF

É fato conhecido na literatura que o EKF possui problemas de convergência, mesmo em sistemas simples (ROMANENKO; CASTRO, 2004) (COYTE et al., 2015). Os problemas de convergência são decorrentes da utilização de modelos linearizados e da incapacidade de se levar em conta explicitamente as restrições de valores plausíveis para os estados durante o processo de estimação dos estados. A figura 5.3 demonstra um caso onde o EKF diverge para o sistema de três tanques. Para isto foram usados valores sabidamente esdrúxulos para inicialização do filtro. O valor de inicialização da matriz de covariância foi de 50000, ou seja, extremamente alto, e de inicialização das alturas dos tanques 1, 2 e 3 de 70, 3 e 2 cm, respectivamente. O filtro projeta um valor negativo para a altura do tanque dois e acaba divergindo. Ao lado direito, faz-se a comparação com o CEKF, dados os mesmos valores para a inicialização do sistema. Neste, dada à possibilidade de restringirmos valores, não permitindo assim valores negativos para as alturas dos tanques, o sistema não diverge, evidenciando assim o ganho de robustez do filtro ao usarmos o CEKF em detrimento do EKF.

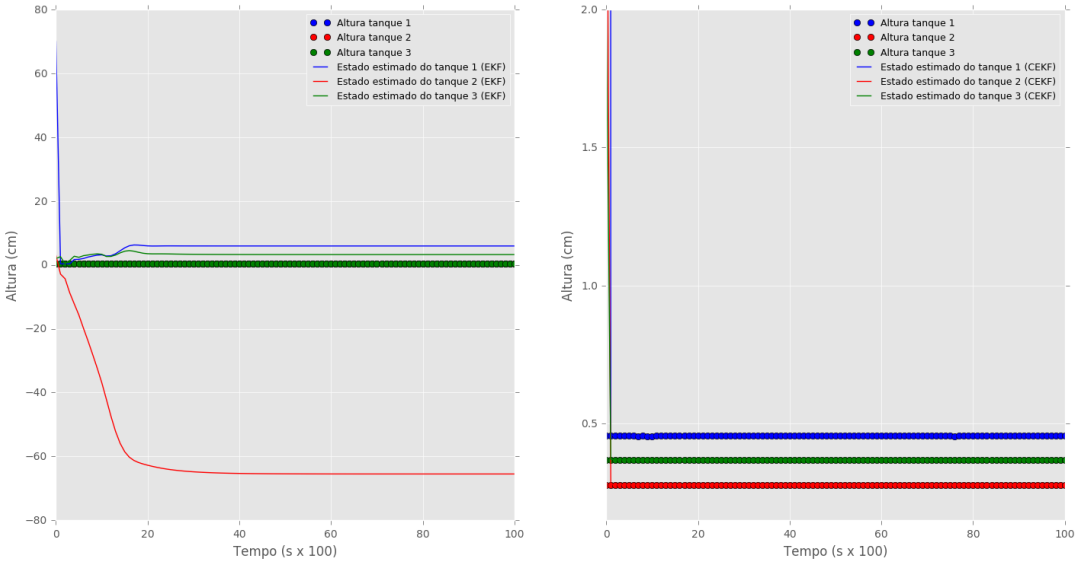

Figura 5.3: Comparação da convergência do EKF e CEKF

5.4 Identificação de falha

Para avaliarmos a capacidade do nosso sistema de realizar a identificação de falhas, o sistema foi simulado durante 600 segundos, com uma amostragem de $\Delta t = 1\ s$. Os parâmetros da simulação se encontram na Tabela 5.2. Foi realizado um distúrbio degrau na carga Q1 no instante t igual a 150 s. O objetivo de perturbarmos o sistema é avaliarmos se este será interpretado como uma falha. Os resultados de identificação de falhas foram obtidos utilizando-se CEKF apenas.

Tabela 5.2: Parâmetros utilizados na simulação

	Parâmetro	Valor	Unidade
	Horizonte de simulação	600	s
	Instante do degrau em Q1	150	s
	Valor do degrau em Q1	5	cm^3/s
	Vazão Q1 (t=0)	20	cm^3/s
	Vazão Q2 (t=0)	15	cm^3/s
	Desvio padrão da medida	0,1	cm
	Desvio padrão do processo	0,005	cm/s
	H1 (t=0)	11	cm
	H2 (t=0)	10	cm
	H3 (t=0)	9	cm
	Matriz de covariância R0	0,01	$(cm)^2$
	Matriz de covariância Q0	0,00025	$(cm/s)^2$
falha 1	Coeficiente de vazão do vazamento	0,15	
	Área da seção transversal do vazamento	0,5	cm^2
	Altura do vazamento	5	cm
falha 2	Desvio no sensor 1	3	cm
falha 3	Desvio na vazão Q1	5	cm^3/s

Para o presente trabalho não foram considerados desvios de modelos. O filtro trabalha sempre com um modelo perfeito do sistema. Ou seja, o modelo matemático usado para as simulações do sistema é o mesmo utilizado na etapa de simulação do filtro.

5.4.1 Resultado para falha 1 - Vazamento no tanque 1

No instante de tempo 250 s foi simulado um vazamento no tanque 1, sendo este vazamento consertado no tempo instante 450 s. Os parâmetros utilizados para simular o vazamento se encontram na tabela 5.2. O resultado da simulação e estados estimados podem ser vistos na figura 5.4. Podemos notar um aumento na altura do tanque a partir do instante 150 s devido ao distúrbio degrau na carga. A partir do instante 250 s, há uma queda repentina na altura devido ao vazamento, com impacto maior no tanque 1 obviamente. Pode-se notar visualmente a divergência do estado estimado com as medidas durante o período no qual o vazamento se encontra ativo. A falha é confirmada ao analisarmos o resultado dos resíduos produzidos pelo sistema. A partir da figura 5.5, é possível notar um aumento rápido no resíduo dos tanques, em especial do tanque 1. A relevância estatística do erro é confirmada pelo teste estatístico (parte inferior da figura). Através da metodologia utilizada, o primeiro sinal de falha no sistema foi obtido no instante t igual a 256 s. O sinal de falha não mais foi dado a partir do instante 473 s. O sistema respondeu, portanto, rapidamente a falha do sistema, tendo em vista que ele necessita obrigatoriamente de três medidas para confirmar a falha. Para acabar com o sinal de falha a resposta é mais lenta, já que o filtro demora mais tempo para convergir para o valor do estado, isto devido a sua matriz de covariância da estimativa do estado ser pequena, devido ao tempo que o filtro já se encontra ativo.

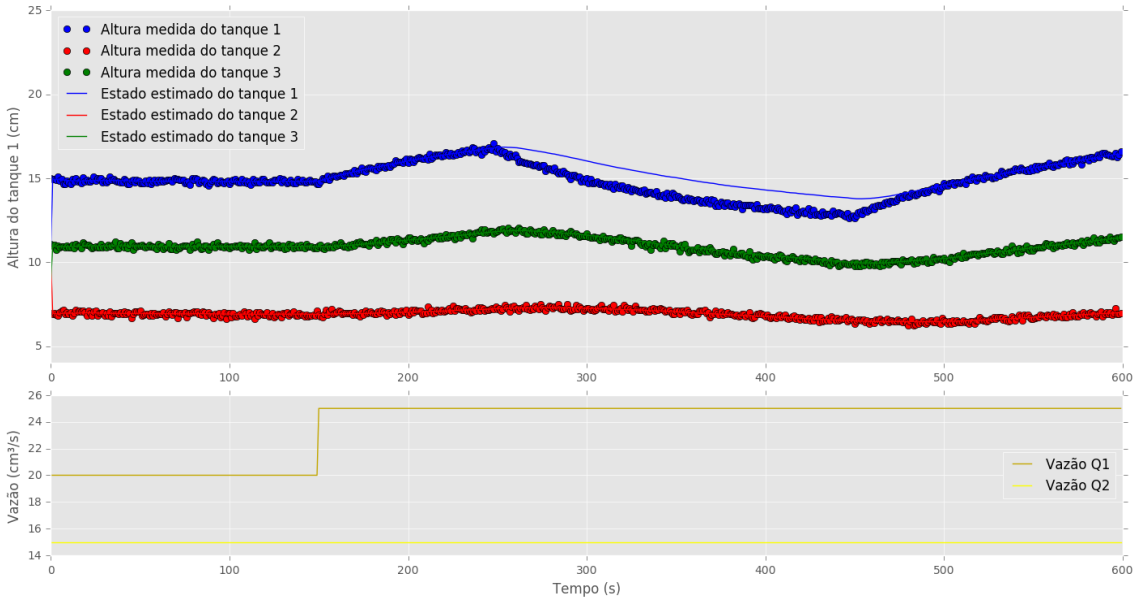

Figura 5.4: Simulação do sistema com ativação da falha 1

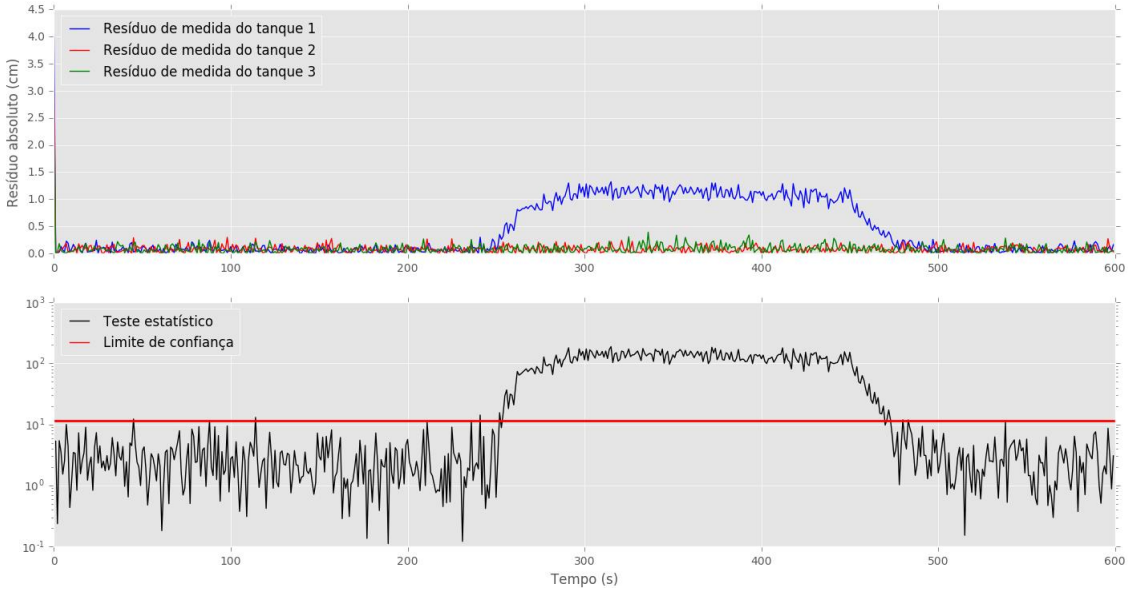

Figura 5.5: Resíduos gerados para a falha 1

5.4.2 Resultado para falha 2 - Desvio na medida gerada pelo sensor da altura do tanque 1

No instante de tempo 250 s foi simulado um desvio na medida gerada pelo sensor da altura do tanque 1, sendo este corrigido no tempo instante 450 s. O desvio é gerado somando-se um valor constante ao valor medido pelo sensor de altura do tanque 1. No caso, o valor somado foi de 3 centímetros. O resultado da simulação e estados estimados podem ser vistos na figura 5.6. Novamente, podemos notar um aumento na altura do tanque a partir do instante 150 s devido ao distúrbio degrau na carga. A partir do instante 250 s, há um aumento brusco no valor medido, devido ao desvio adicionado. Podemos notar que os demais valores medidos não sofrem impacto significativo, o que é esperado, já que esta falha não afeta o valor dos demais estados. Contanto, há efeito no estado estimado dos três tanques como esperado. A falha é confirmada ao analisarmos o resultado dos resíduos produzidos pelo sistema. A partir da figura 5.7, é possível notar um aumento rápido no resíduo dos tanques, em especial do tanque 1. O resíduo é muito pronunciado logo quando a falha é ativada, e diminui conforme o estado estimado converge para um novo valor. A relevância estatística do erro é confirmada pelo teste estatístico (parte inferior da figura). O primeiro sinal de falha no sistema foi obtido no instante t igual a 252 s, ou seja, no primeiro instante possível para a confirmação da falha assim que esta é ativada. O sinal de falha passou a não ser mais sinalizado a partir do instante 484 s. A demora para o término do sinal de falha se dá novamente pelo fato da matriz de covariância apresentar valores pequenos, fazendo com que o filtro tome mais tempo para convergir a novos patamares de estabilidade.

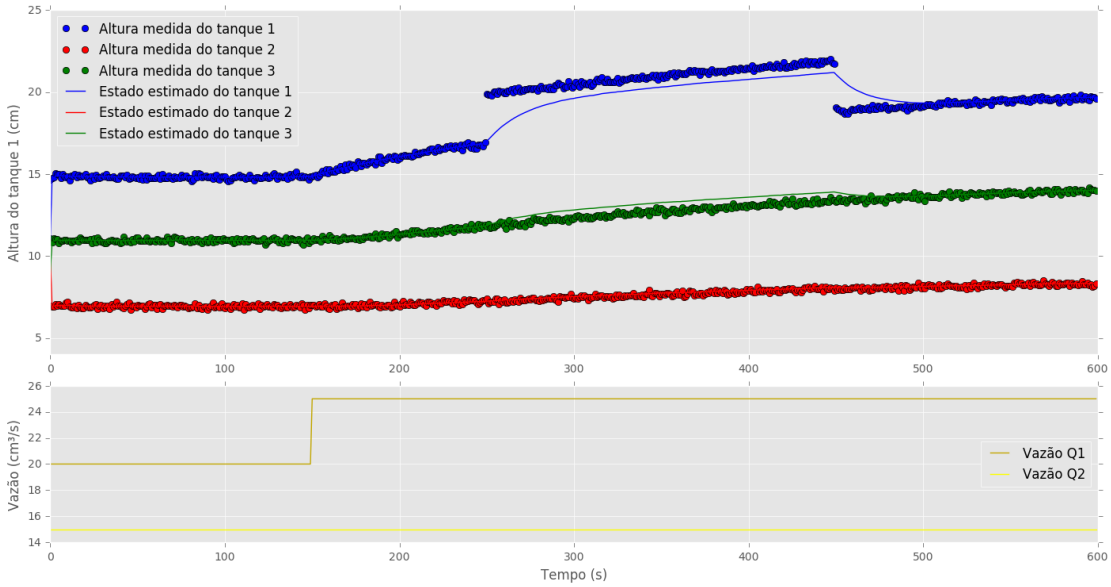

Figura 5.6: Simulação do sistema com ativação da falha 2

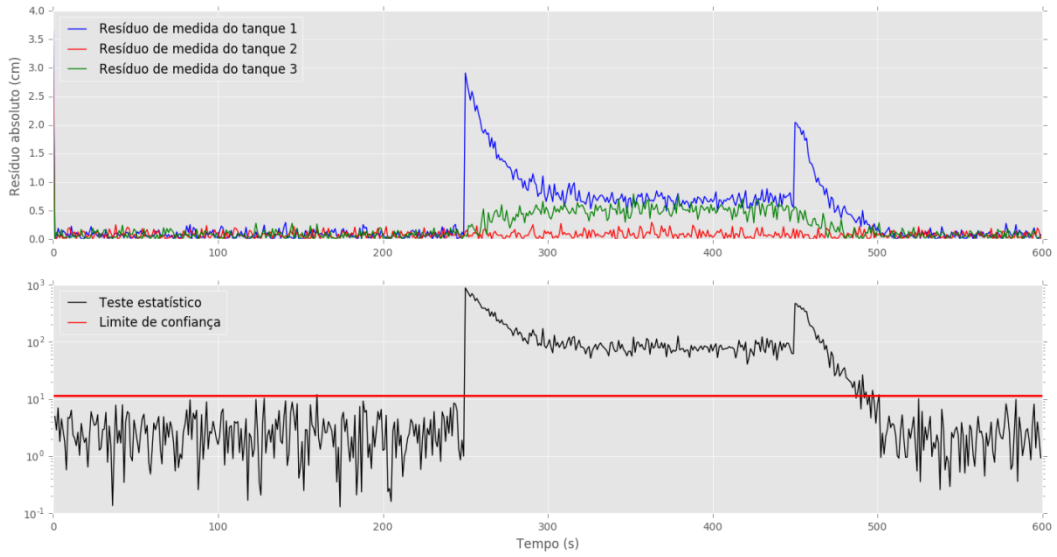

Figura 5.7: Resíduos gerados para a falha 2

5.4.3 Resultado para falha 3 – Falha na vazão de entrada Q1

No instante de tempo 250 s foi simulado um desvio na vazão de entrada Q, sendo este corrigido no tempo instante 450 s. O desvio é gerado somando-se um valor constante ao valor da variável Q1, representando assim uma falha de atuador no sistema. No caso, o valor somado foi de 5 cm³/s. O resultado da simulação e estados estimados podem ser vistos na figura 5.8. Novamente, podemos notar um aumento na altura do tanque a partir do instante 150 s devido a perturbação degrau na carga. A partir do instante 250 s, há um aumento da variável medida em relação a estimativa do estado. O aumento é muito mais sútil em comparação com a falha de número 1. A falha é confirmada ao analisarmos o resultado dos resíduos produzidos pelo sistema. A partir da figura 5.9, é possível notar um aumento no resíduo dos tanques, em especial do tanque 1. O primeiro sinal de falha no

sistema foi obtido no instante t igual a 269 s. O sinal de falha deixou de ser sinalizado a partir do instante 465 s.

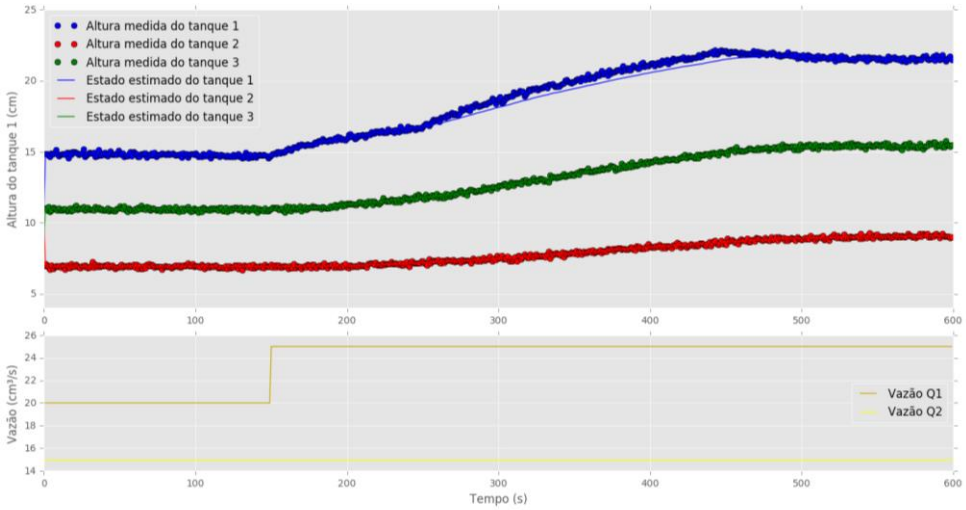

Figura 5.8: Simulação do sistema com ativação da falha 3

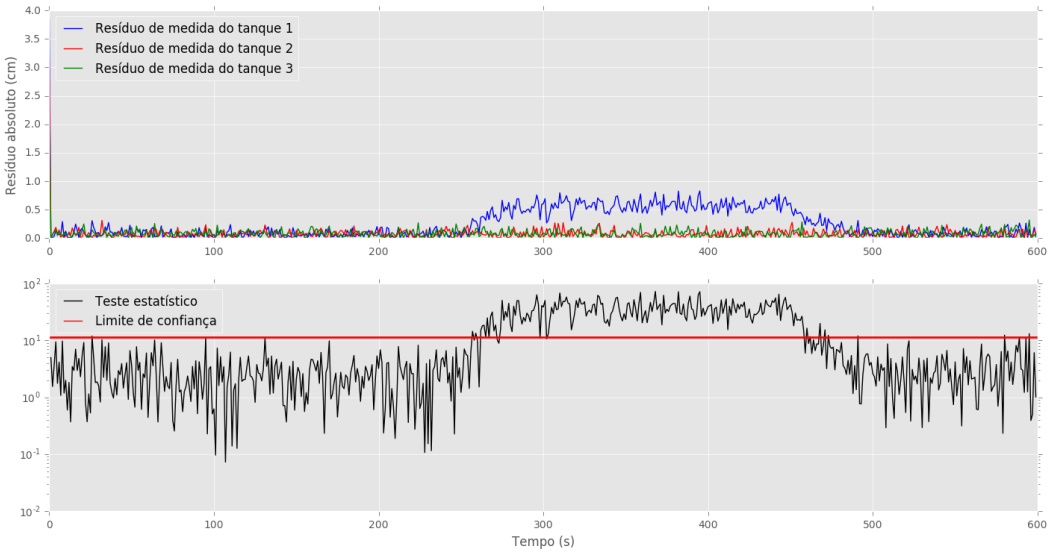

Figura 5.9: Resíduos gerados para a falha 3

6 Conclusões e Trabalhos Futuros

As técnicas de estimação de estado por meio de filtros de Kalman se encontram bem difundidas tanto na academia quanto na indústria. Uni-las a algoritmos de DDFs se torna, portanto, uma tarefa quase óbvia, já que assim reduzimos a intensidade de ruídos, inerentes a qualquer medição de um sistema real. A derivação mais comum do filtro de Kalman para sistemas não-estacionários, o EKF, apresenta problemas de divergência devido a linearização do sistema e impossibilidade de adicionarmos restrições, algo fundamental para aplicação em sistemas reais. O objetivo do trabalho foi a aplicação do CEKF a um sistema de três tanques, dado que este apresenta a possibilidade de adição de restrições, o que lhe confere uma maior robustez. A implementação se deu na linguagem Python. O sistema foi simulado, sendo três diferentes falhas impostas a este. A partir dos dados simulados, o processo de diagnóstico de falhas foi utilizado.

A maior robustez do CEKF em comparação ao EKF foi confirmada ao inicializarmos o filtro com valores esdrúxulos. O EKF mostrou que pode divergir mesmo em um sistema simples, tal como o estudado. A metodologia aplicada utilizando o CEKF como estimador de estado se mostrou efetiva e aplicável na detecção de falhas no sistema de três tanques simulado. Nas três diferentes falhas, a detecção se deu facilmente, sendo o teste estatístico utilizado efetivo como limite para o estado normal do sistema.

O trabalho, entretanto, é apenas a primeira evidência da possibilidade de usarmos o CEKF como ferramenta para DDF. Para trabalhos futuros, é sugerida primeiramente a análise das mesmas falhas estando o filtro submetido a um modelo com desvios. Este é um ponto crucial para a aplicação da metodologia a sistemas reais, tendo em vista a dificuldade que se há ao caracterizar um sistema através de um modelo matemático. Após esta primeira análise, a metodologia deve ser aplicada a dados reais de um sistema de três tanques, de forma a validarmos os resultados obtidos. Posteriormente, é necessária aplicação a sistemas mais complexos, como, por exemplo, o sistema *benchmark* Tenneessee Eastman. Também é sugerida a comparação do resultado obtido a partir do CEKF com outras técnicas de estimação de estado, como o MHE, UKF, *Particle Filter, etc.* Por fim, a metodologia deve ser testada a sistemas reais encontrados na indústria para validação final e aplicação na detecção de falhas.

Referências

COYTE, J. L. et al. A Drift Detecting Anti-Divergent EKF for Online Biodynamic Model Identification. **2015 IEEE 17th International Conference on High Performance Computing and Communications**, p. 1116–1121, ago. 2015.

DING, S. X. **Advances in Industrial Control Data-driven Design of Fault Diagnosis and Fault-tolerant Control Systems**. Advances in Industrial Control, 2013.

DTS 200 - Laboratory Setup - Three Tank System. **Amira GmbH**, 2002.

FANGOHR, H. Introduction to Python for Computational Science and Engineering (A beginner' s guide). 2015.

GESTHUISEN, R.; KLATT, K. Ý.; ENGELL, S. Þ. Optimization-Based State Estimation - A Comparative Study for the Batch Polycondesation of Polyethyleneterephthalate. **European Control Conference (ECC)**, n. 2, p. 1062–1067, 2001.

HINDMARSH, A. C. **ODEPACK, A Systematized Collection of ODE Solvers** MACS transactions on scientific computation, p. 55 - 64, 1983.

HOU, M.; XIONG, Y. S.; PATTON, R. J. Observing a three-tank system. **IEEE Transactions on Control Systems Technology**, v. 13, n. 3, p. 478–484, maio 2005.

ISERMANN, R. Model-based fault-detection and diagnosis – status and applications. **Annual Reviews in Control**, v. 29, n. 1, p. 71–85, jan. 2005.

LABBE JR, R. R. Kalman and Bayesian Filters in Python. 2015.

MAYBECK, P. S. Stochastic Models, Estimation and Control. **Academic Press**, v. 1, 1979.

PRAKASH, J.; PATWARDHAN, S. C.; NARASIMHAN, S. A Supervisory Approach to Fault-Tolerant Control of Linear Multivariable Systems. **Industrial & Engineering Chemistry Research**, v. 41, n. 9, p. 2270–2281, maio 2002.

RAO, C. V.; RAWLINGS, J. B.; MAYNE, D. Q. Constrained state estimation for nonlinear discrete-time systems: stability and moving horizon approximations. **IEEE Transactions on Automatic Control**, v. 48, n. 2, p. 246–258, fev. 2003.

RHODES, I. B. Tutorial Introduction to Estimation and Filtering. **IEEE Transactions on Automatic Control**, n. 6, p. 688–706, 1971.

ROBERTSON, D.; LEE, J. **A least squares formulation for state estimation**Journal os Process Control, , 1995.

ROMANENKO, A.; CASTRO, J. A. A. M. The unscented filter as an alternative to the EKF for nonlinear state estimation: a simulation case study. **Computers & Chemical Engineering**, v. 28, n. 3, p. 347–355, mar. 2004.

SALAU, N. P. G. Abordagem Sistemática para Construção e Sintonia de Estimadores de Estados Não-Linerares (Tese). **UFRGS, Porto Alegre**, 2009.

UNGARALA, S.; DOLENCE, E.; LI, K. Constrained Extended Kalman Filter For Nonlinear State Estimation. v. 2, p. 63–68, 2007.

VENKATASUBRAMANIAN, V.; RENGASWAMY, R.; YIN, K. A Review of Process Fault Detection and Diagnosis Part I: Quantitative Model-Based Methods. **Computers & Chemical Engineering,** v. 27, p. 293–311, 2003.

VILLEZ, K. et al. Kalman-based strategies for Fault Detection and Identification (FDI): Extensions and critical evaluation for a buffer tank system. **Computers & Chemical Engineering**, v. 35, n. 5, p. 806–816, maio 2011.

WELCH, G.; BISHOP, G. An Introduction to the Kalman Filter. p. 1–16, 2001.

WITCZAK, M.; APPROACHES, S. C. **Faut Diagnosis and Fault-Tolerant Control Strategies for Non-Linear Systems**. Springer, 2014.

APÊNDICE A – Implementação do Sistema e Filtros em Python

```
import numpy as np
from numpy import transpose as tp
from numpy.linalg import inv as iv
from numpy import dot as dot
import scipy.integrate as integ
import matplotlib.pyplot as plt
import time
from scipy import optimize
from scipy.optimize import fsolve
from scipy.linalg import expm
import cvxopt
cvxopt.solvers.options['show_progress'] = False
from scipy import optimize
from openopt import QP
from scipy import stats
from matplotlib import style
import pandas as pd
style.use('ggplot')
np.random.seed(0)
#style.available()


class three_tanks(object):
    def __init__(self, step = 1):

        self.a1 = 0.46
        self.a2 = 0.60
        self.a3 = 0.45
        self.A = 154     #cm2
        self.s0 = 0.5    #cm2
        self.s13 = 0.5   #cm2
        self.s23 = 0.5   #cm2
        self.g = 980     #cm/s2
        self.step = step



    def jacobian(self, x, step):

        h1 = x[0]
        h2 = x[1]
        h3 = x[2]

        a1 = self.a1
```

```
    a2 = self.a2
    a3 = self.a3
    A = self.A
    s0 = self.s0
    s13 = self.s13
    s23 = self.s23
    g = self.g

    A11 = -a1*s13*g/(A*(2*g*abs(h1-h3))**(1/2))
    A12 = 0
    A13 = a1*s13*g/(A*(2*g*abs(h1-h3))**(1/2))
    A21 = 0

    A22 = -a3*s23*g/(A*(2*g*abs(h2-h3))**(1/2))-a2*s0*g/(A*(2*g*abs(h2))**(1/2))
    A23 = a3*s23*g/(A*(2*g*abs(h2-h3))**(1/2))
    A31 = a1*s13*g/(A*(2*g*abs(h1-h3))**(1/2))
    A32 = a3*s23*g/(A*(2*g*abs(h2-h3))**(1/2))
    A33 = -a3*s23*g/(A*(2*g*abs(h2-h3))**(1/2))


    self.Amatrix = np.array([[A11, A12, A13],[A21, A22, A23], [A31, A32, A33]])

    self.phi = expm(self.Amatrix*step)


def diff_equation(self, y, t, Q1, Q2, Qsim, hole1 = 0):


    #State
    h1 = y[0]
    h2 = y[1]
    h3 = y[2]

    #Parameters


    a1 = self.a1
    a2 = self.a2
    a3 = self.a3
    A = self.A
    s0 = self.s0
    s13 = self.s13
    s23 = self.s23
    g = self.g


    ahole = 0.15 #cm2
    shole = 0.5
    hhole = 5
```

```
    #Flows

    Q13 = a1 * s13 * np.sign(h1 - h3) * (2 * g * abs(h1 - h3))**(1 / 2)
    Q32 = a3 * s23 * np.sign(h3 - h2) * (2 * g * abs(h3 - h2))**(1 / 2)
    Q20 = a2 * s0 * (2 * g * abs(h2)) ** (1 / 2)


    if h1 - hhole > 0 and hole1 == 1:
       Qhole1 = ahole * shole * (2 * g * abs(h1 - hhole))**(1 / 2)
    else:
       Qhole1 = 0


    dydt = np.zeros_like(y)

    dydt[0] = (Q1 - Q13 - Qhole1) / A
    dydt[1] = (Q2 + Q32 - Q20) / A
    dydt[2] = (Q13 - Q32) / A


    return dydt + Qsim


def steady_state(self):

    a1 = self.a1
    a2 = self.a2
    a3 = self.a3
    A = self.A
    s0 = self.s0
    s13 = self.s13
    s23 = self.s23
    g = self.g

    def f(h):
       h1ss, h2ss, h3ss = h
       nerro = 0.00001


       Q13ss = a1 * s13 * np.sign(h1ss - h3ss) * (2 * g * abs(h1ss - h3ss))**(1 / 2)
       Q32ss = a3 * s23 * np.sign(h3ss - h2ss) * (2 * g * abs(h3ss - h2ss))**(1 / 2)
       if h2ss <= 0:
          Q20ss = 0
       else:
          Q20ss = a2 * s0 *(2 * g * h2ss) ** (1 / 2)
       Q1 = self.Q1[0]
       Q2 = self.Q2[0]

       return (Q1 - Q13ss, Q2 + Q32ss - Q20ss, Q13ss - Q32ss)
```

```
        self.h1ss, self.h2ss, self.h3ss  =  fsolve(f, (2.0, 1.5, 0.5))


    def input_var(self, Q10, Q20, n, delta1 = 0, step1 = 0, delta2 = 0, step2 = 0):


        self.Q1 = [Q10]
        self.Q2 = [Q20]
        self.n = n


        for i in range(1, n):
            if delta1 == i:
                Q10 += step1
            if delta2 == i:
                Q20 += step2

            self.Q1.append(Q10)
            self.Q2.append(Q20)


    def simulate(self, y0, Qsim0, Rsim, falha = 0):

        step = self.step
        self.states = np.empty((self.n, 3))
        self.mesured = np.empty((self.n, 3))
        self.states[0, :] = y0
        self.mesured[0, :] = self.states[0, :] + np.random.normal(0, Rsim)
        hole = 0
        bias = np.array([0, 0, 0])
        bias2 = np.array([0, 0])

        for i in range(1, self.n):

            if falha == 0:
                # FALHA = 0, FURO NO TANQUE 1 DE T=250 ATÉ T = 450
                if i == 250:
                    hole = 1
                if i == 450:
                    hole = 0

            if falha == 1:
                # FALHA = 1, BIAS NO MEDIDOR DE ALTURA
                if i == 250:
                    bias = np.array([3, 0, 0])
                if i == 450:
                    bias = np.array([0, 0, 0])

            if falha == 2:
                # FALHA = 2, BIAS NA VAZAO DE ENTRADA
```

```
            if i == 250:
                bias2 = np.array([5, 0])
            if i == 450:
                bias2 = np.array([0, 0])


        Qsim = np.random.normal(0, Qsim0)
        result = integ.odeint(self.diff_equation, self.states[i - 1, :], [0, step],
args=(self.Q1[i] + bias2[0],self.Q2[i] + bias2[1],Qsim,hole))
        self.states[i, :] = result[1, :]
        aux = np.random.normal(0, Rsim)
        self.mesured[i, :] = self.states[i, :] + aux + bias
        #print(aux)


    def DEKF(self, y0, P0, Q, R):
        step = self.step
        Q = np.matrix(Q)
        R = np.matrix(R)
        self.K = np.empty((3,3,self.n))
        self.V = np.empty((3,3,self.n))
        self.state_ekf = np.empty((3, self.n + 1))
        self.state_ekf[:, 0] = y0
        self.state_k_1 = np.empty((3, self.n + 1))
        self.state_k_1[:, 0] = y0
        self.resid = np.empty((3, self.n))
        self.erro = np.empty((self.n))
        self.P = np.empty((3,3,self.n + 1))
        self.P[:,:,0] = P0
        Qsim = [0, 0, 0]

        for i in range(0, self.n):
            result = integ.odeint(self.diff_equation, self.state_ekf[:, i], [0, step],
args=(self.Q1[i],self.Q2[i],Qsim,0))
            self.state_ekf[:, i + 1] = result[1, :]
            self.state_k_1[:, i + 1] = result[1, :]
            self.jacobian(self.state_ekf[:, i], step)
            phi = np.matrix(self.phi)
            p_k_1 = np.matrix(self.P[:,:,i])
            self.P[:,:,i + 1] = np.array(phi * p_k_1 * phi.T + Q)
            self.V[:,:,i] = self.P[:,:,i + 1]
            p_k = np.matrix(self.P[:,:,i + 1])

            self.K[:,:,i] = np.array(p_k * iv(p_k + R))
            self.resid[:,i] = self.mesured[i,:]-self.state_ekf[:, i + 1]
            self.state_ekf[:, i + 1] = self.state_ekf[:, i + 1] + np.dot(self.K[:,:,i],
self.mesured[i,:]-self.state_ekf[:, i + 1])
            self.P[:,:,i + 1] = np.dot(np.eye(3)-self.K[:,:,i],self.P[:,:,i + 1])
            self.erro[i] = np.dot(np.dot(self.resid[:,i], np.linalg.inv(self.V[:,:,i])),self.resid[:,i])
```

```
    def CEKF(self, y0, P0, Q, R):

        step = self.step
        Q = np.matrix(Q)
        R = np.matrix(R)
        H = np.matrix(np.eye(3))
        Qsim = [0, 0, 0]
        self.state_cekf = np.empty((3, self.n))
        self.state_cekf[:, 0] = y0
        self.y_cekf = self.state_cekf.copy()
        self.c_P = np.empty((3,3,self.n + 1))
        self.c_P[:,:,0] = P0
        self.c_V = np.empty((3,3,self.n + 1))
        self.c_V[:,:,0] = P0
        self.c_resid = np.empty((3, self.n))
        self.c_erro = np.empty((self.n))


        h1max, h2max, h3max = [90] * 3
        h1min, h2min, h3min = [0] * 3
        y1max, y2max, y3max = [90] * 3
        y1min, y2min, y3min = [0] * 3



        for i in range(1, self.n):


            result = integ.odeint(self.diff_equation, self.state_cekf[:, i - 1], [0, step],
args=(self.Q1[i],self.Q2[i],Qsim,0))
            self.state_cekf[:, i] = result[1, :]
            print(self.state_cekf[:, i])

            self.jacobian(self.state_cekf[:, i], step)

            phi = np.matrix(self.phi)
            P_k_1 = np.matrix(self.c_P[:,:,i - 1])

            self.c_V[:,:,i] = H * P_k_1 * H.T + R
            self.c_P[:,:,i] = np.array(phi * P_k_1 * phi.T - phi * P_k_1 * H.T * iv(H * P_k_1 *
H.T + R) * H * P_k_1 * phi.T + Q)

            wmin = np.array([h1min, h2min, h3min]) - self.state_cekf[:, i]
            wmax = np.array([h1max, h2max, h3max]) - self.state_cekf[:, i]
            vmin = wmin
            vmax = wmax

            T1 = np.hstack((iv(np.array(P_k_1)), np.zeros((3,3))))
            T2 = np.hstack((np.zeros((3,3)), iv(np.array(R))))
```

```
            Ha = np.vstack((T1, T2))
            f = np.array([0.0001] * 6)

            lb = np.hstack((wmin, vmin))
            ub = np.hstack((wmax, vmax))

            Aeq = np.hstack((np.eye(3), np.eye(3)))
            beq = self.mesured[i,:] - self.state_cekf[:, i]

            sol = QP(Ha, f, A=np.eye(6), Aeq=Aeq, b=np.array([1000000] * 6) ,beq=beq,
lb=lb, ub=ub)
            sol.solve('cvxopt_qp')

            erro = self.mesured[i,:] - self.state_cekf[:, i]

            self.state_cekf[:, i] = self.state_cekf[:, i] + sol.xf[:3]
            self.y_cekf[:, i] = self.state_cekf[:, i] + sol.xf[3:]
            self.c_erro[i] = dot(erro, dot(iv(self.c_V[:,:,i]), erro))
```

Funcionamento do filtro

```
# Demonstrando o funcionamento do filtro

system = three_tanks()
Q10 = 20
Q20 = 15
system.input_var(Q10, Q20, n = 110, delta1 = 150, step1 = 5)
system.steady_state()
h0 = [system.h1ss, system.h2ss, system.h3ss]
Rsim = [0.5] * 3
Qsim = [0.05] * 3
system.simulate(h0, Qsim, Rsim, falha = 0)


system2 = three_tanks()
Q10 = 20
Q20 = 15
system2.input_var(Q10, Q20, n = 110, delta1 = 150, step1 = 5)
system2.steady_state()
h0 = [system2.h1ss, system2.h2ss, system2.h3ss]
Rsim = [0.0000000000001] * 3
Qsim = [0.0000000000001] * 3
system2.simulate(h0, Qsim, Rsim, falha = 0)

fig = plt.figure()
ax1 = plt.subplot2grid((4,1), (0, 0), rowspan = 3)
plt.ylabel('Altura do tanque 1 (cm)')
plt.xlabel('Tempo (s)')
```

```
ax2 = ax1.twinx()
ax3 = ax1.twinx()
ax4 = ax1.twinx()
ax2.axes.yaxis.set_ticklabels([])
ax3.axes.yaxis.set_ticklabels([])
ax4.axes.yaxis.set_ticklabels([])

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3)*0.5 ** 2
Q0=np.eye(3)*0.05 ** 2

system.DEKF(y0, P0, Q0, R0)

ultimo = 51
ax1.plot(system.mesured[:ultimo-1, 0], 'o', color = 'k')
ax2.plot(system.state_k_1[0, 1:ultimo], 'x', color = 'g', mew = 1.5)
ax3.plot(system.state_ekf[0, 1:ultimo], color = 'b')
ax4.plot(system2.mesured[:ultimo-1, 0], '-', color = 'k', lw = 0.5)

low = system.mesured[:ultimo - 1, 0].min() * 0.9995
high = system.mesured[:ultimo - 1, 0].max() * 1.0015

ax1.set_ylim([low, high])
ax2.set_ylim([low, high])
ax3.set_ylim([low, high])
ax4.set_ylim([low, high])

plt.plot([], [], 'o', color = 'k', label ='Valores simulado')
plt.plot([], [], 'x', color = 'g', label ='Estimativa do estado a priori', mew = 1.5)
plt.plot([], [], color = 'b', label ='Estimativa do estado a posteriori')
plt.plot([], [], color = 'k', lw = 0.5, label ='Estado sem ruídos')

plt.legend(loc = 1)
plt.legend(loc = 1).set_alpha(0.65)

ax4 = plt.subplot2grid((4,1), (3, 0))
ax4.plot(system.P[0, 0, :ultimo - 1], color = 'y')
ax4.fill_between(system.P[0, 0, :ultimo - 1], 0, facecolor = 'y', alpha = 0.8)
plt.plot([],[], color = 'y', label = 'Covariância da estimativa do estado')
plt.legend(loc='best')

plt.show()
```

Variação de Parâmetros do Filtro

```
style.use('ggplot')
```

```
system = three_tanks()
Q10 = 20
Q20 = 15
system.input_var(Q10, Q20, n = 110, delta1 = 150, step1 = 5)
system.steady_state()
h0 = [system.h1ss, system.h2ss, system.h3ss]
Rsim = [0.5] * 3
Qsim = [0.05] * 3
system.simulate(h0, Qsim, Rsim, falha = 0)


#Diferença de R0 e Q0

#plot initalization ----------------------------

fig = plt.figure()

ax0 = plt.subplot2grid((1,1), (0, 0))
plt.ylabel('Altura do tanque 1 (cm)')
plt.xlabel('Tempo (s)')
ax1 = ax0.twinx()
ax2 = ax0.twinx()
ax3 = ax0.twinx()
ax1.axes.yaxis.set_ticklabels([])
ax2.axes.yaxis.set_ticklabels([])
ax3.axes.yaxis.set_ticklabels([])

ultimo = 101


# system 1 ----------------------------------

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3)*0.5 ** 2
Q0=np.eye(3)*0.05 ** 2

system.DEKF(y0, P0, Q0, R0)

ax0.plot(system.mesured[:ultimo - 1, 0], 'o', color = 'k')
ax1.plot(system.state_ekf[0, 1:ultimo], 'b')

# system 2 ----------------------------------

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3)*0.05 ** 2
Q0=np.eye(3)*0.05 ** 2

system.DEKF(y0, P0, Q0, R0)
```

```
ax2.plot(system.state_ekf[0, 1:ultimo], color = 'r')


# system 3 ----------------------------------------

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3)*0.05 ** 2
Q0=np.eye(3)*0.5 ** 2

system.DEKF(y0, P0, Q0, R0)

ax3.plot(system.state_ekf[0, 1:ultimo], color = 'c')

#plot configurations ----------------------------------

low = system.mesured[:ultimo - 1, 0].min() * 0.995
high = system.mesured[:ultimo - 1, 0].max() * 1.005

ax0.set_ylim([low, high])
ax1.set_ylim([low, high])
ax2.set_ylim([low, high])
ax3.set_ylim([low, high])

plt.plot([], [], 'o', color = 'k', label = 'Simulado')
plt.plot([], [], color = 'b', label = 'R0 = 0.25\nQ0 = 0.0025')
plt.plot([], [], color = 'r', label = 'R0 = 0.0025\nQ0 = 0.0025')
plt.plot([], [], color = 'c', label = 'R0 = 0.25\nQ0 = 0.25')

plt.legend()
plt.legend(loc=1)


plt.show()
```

Testando a distribuição

```
# Teste distribuição

system = three_tanks()
Q10 = 20
Q20 = 15
system.input_var(Q10, Q20, n = 10000)
system.steady_state()
h0 = [system.h1ss, system.h2ss, system.h3ss]
Rsim = [0.1] * 3
Qsim = [0.001] * 3
```

```
system.simulate(h0, Qsim, Rsim, falha = 10)

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3) * Rsim[0] ** 2
Q0=np.eye(3) * Qsim[0] ** 2

system.CEKF(y0, P0, Q0, R0)

crit = stats.chi2.ppf(q = 0.99, df=3)
erros = system.c_erro[1:]
outlier = erros[erros>crit]
print('/n/n-----Limite de confiança 99%------/n')
print((1 - len(outlier)/len(erros))*100)

crit = stats.chi2.ppf(q = 0.95, df=3)
erros = system.c_erro[1:]
outlier = erros[erros>crit]
print('/n/n-----Limite de confiança 95%------/n')
print((1 - len(outlier)/len(erros))*100)
```

Comparação CEKF e EKF

```
# EKF contra CEKF

system = three_tanks(step = 100)
Q10 = 3
Q20 = 4

system.input_var(Q10, Q20, n = 101, delta1 = 150, step1 = 5)
system.steady_state()
h0 = [system.h1ss, system.h2ss, system.h3ss]
Rsim = [0.000005] * 3
Qsim = [0.000005] * 3
system.simulate(h0, Qsim, Rsim, falha = 0)

y0=[70, 3, 2]
P0=np.eye(3)*50000
R0=np.eye(3)*5
Q0=np.eye(3)*1

system.DEKF(y0, P0, Q0, R0)

fig = plt.figure()
ax1 = plt.subplot2grid((1,2), (0, 0))

ax1.plot(system.mesured[:, 0], 'o', color = 'b', mew = 0.5, label = 'Altura tanque 1')
ax1.plot(system.mesured[:, 1], 'o', color = 'r', mew = 0.5, label = 'Altura tanque 2')
ax1.plot(system.mesured[:, 2], 'o', color = 'g', mew = 0.5, label = 'Altura tanque 3')
```

```
    ax1.plot(system.state_ekf[0, 1:], color = 'b', mew = 0.5, label = 'Estado estimado do tanque 1 (EKF)')
    ax1.plot(system.state_ekf[1, 1:], color = 'r', mew = 0.5, label = 'Estado estimado do tanque 2 (EKF)')
    ax1.plot(system.state_ekf[2, 1:], color = 'g', mew = 0.5, label = 'Estado estimado do tanque 3 (EKF)')
    plt.xlabel('Tempo (s x 100)')
    plt.ylabel('Altura (cm)')
    plt.legend(loc=5, prop={'size':9})


    ax2 = plt.subplot2grid((1,2), (0, 1))

    system = three_tanks(step = 100)
    Q10 = 3
    Q20 = 4
    system.input_var(Q10, Q20, n = 101, delta1 = 150, step1 = 5)
    system.steady_state()
    h0 = [system.h1ss, system.h2ss, system.h3ss]
    Rsim = [0.000005] * 3
    Qsim = [0.000005] * 3
    system.simulate(h0, Qsim, Rsim, falha = 0)

    y0=[70, 3, 2]
    P0=np.eye(3)*50000
    R0=np.eye(3)*5
    Q0=np.eye(3)*1

    system.CEKF(y0, P0, Q0, R0)

    ax2.plot(system.mesured[:, 0], 'o', color = 'b', mew = 0.5, label = 'Altura tanque 1')
    ax2.plot(system.mesured[:, 1], 'o', color = 'r', mew = 0.5, label = 'Altura tanque 2')
    ax2.plot(system.mesured[:, 2], 'o', color = 'g', mew = 0.5, label = 'Altura tanque 3')
    ax2.plot(system.state_cekf[0, 1:], color = 'b', mew = 0.5, label = 'Estado estimado do tanque 1 (CEKF)')
    ax2.plot(system.state_cekf[1, 1:], color = 'r', mew = 0.5, label = 'Estado estimado do tanque 2 (CEKF)')
    ax2.plot(system.state_cekf[2, 1:], color = 'g', mew = 0.5, label = 'Estado estimado do tanque 3 (CEKF)')
    plt.xlabel('Tempo (s x 100)')
    plt.ylabel('Altura (cm)')

    low = 0.15
    high = 0.5
    ax2.set_ylim([low, high])

    plt.legend(loc=5, prop={'size':9})
    plt.show()
```

Simulação falha 1

```
# Filtro em estado ótimo com falha 0

system = three_tanks()
Q10 = 20
Q20 = 15
system.input_var(Q10, Q20, n = 600, delta1 = 150, step1 = 5)
system.steady_state()
h0 = [system.h1ss, system.h2ss, system.h3ss]
Rsim = [0.1] * 3
Qsim = [0.005] * 3
system.simulate(h0, Qsim, Rsim, falha = 0)

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3) * Rsim[0] ** 2
Q0=np.eye(3) * Qsim[0] ** 2

system.CEKF(y0, P0, Q0, R0)

fig = plt.figure()
ax1 = plt.subplot2grid((3,1), (0, 0), rowspan=2)
plt.ylabel('Altura do tanque 1 (cm)')

ax1.plot(system.mesured[:, 0], 'o', color = 'b', mew = 0.5, label = 'Altura medida do tanque 1')
ax1.plot(system.mesured[:, 1], 'o', color = 'r', mew = 0.5, label = 'Altura medida do tanque 2')
ax1.plot(system.mesured[:, 2], 'o', color = 'g', mew = 0.5, label = 'Altura medida do tanque 3')
ax1.plot(system.state_cekf[0, :], color = 'b', mew = 0.5, label = 'Estado estimado do tanque 1')
ax1.plot(system.state_cekf[1, :], color = 'r', mew = 0.5, label = 'Estado estimado do tanque 2')
ax1.plot(system.state_cekf[2, :], color = 'g', mew = 0.5, label = 'Estado estimado do tanque 3')
plt.legend(loc=2).set_alpha(0.6)
ax1.set_ylim([4, 25])

ax2 = plt.subplot2grid((3,1), (2, 0))
ax2.plot(system.Q1, color = '#c3a700', label = 'Vazão Q1')
ax2.plot(system.Q2, color = '#fdff00', label = 'Vazão Q2')
plt.ylabel('Vazão (cm³/s)')
plt.xlabel('Tempo (s)')
ax2.set_ylim([14, 26])
plt.legend(loc=5)

plt.show()
```

```
fig2 = plt.figure()
ax11 = plt.subplot2grid((2,1), (0, 0))
plt.ylabel('Resíduo absoluto (cm)')

ax11.plot(abs(system.mesured[:, 0] - system.state_cekf[0, :]), color = 'b', label = 'Resíduo de medida do tanque 1')
ax11.plot(abs(system.mesured[:, 1] - system.state_cekf[1, :]), color = 'r', label = 'Resíduo de medida do tanque 2')
ax11.plot(abs(system.mesured[:, 2] - system.state_cekf[2, :]), color = 'g', label = 'Resíduo de medida do tanque 3')

plt.legend(loc=2)

crit = stats.chi2.ppf(q = 0.99, df=3)

ax12 = plt.subplot2grid((2,1), (1, 0))
ax12.semilogy(system.c_erro[1:], color = 'k')
ax12.axhline(crit, color = 'r', linewidth = 2)
plt.plot([],[], color = 'k', label = 'Teste estatístico')
plt.plot([],[], color = 'r', label = 'Limite de confiança')
plt.legend(loc = 2)
plt.xlabel('Tempo (s)')

plt.show()
```

Simulação falha 2

```
# Filtro em estado ótimo com falha 1

system = three_tanks()
Q10 = 20
Q20 = 15
system.input_var(Q10, Q20, n = 600, delta1 = 150, step1 = 5)
system.steady_state()
h0 = [system.h1ss, system.h2ss, system.h3ss]
Rsim = [0.1] * 3
Qsim = [0.005] * 3
system.simulate(h0, Qsim, Rsim, falha = 1)

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3) * Rsim[0] ** 2
Q0=np.eye(3) * Qsim[0] ** 2

system.CEKF(y0, P0, Q0, R0)

fig = plt.figure()
```

```
    ax1 = plt.subplot2grid((3,1), (0, 0), rowspan=2)
    plt.ylabel('Altura do tanque 1 (cm)')

    ax1.plot(system.mesured[:, 0], 'o', color = 'b', mew = 0.5, label = 'Altura medida do
tanque 1')
    ax1.plot(system.mesured[:, 1], 'o', color = 'r', mew = 0.5, label = 'Altura medida do
tanque 2')
    ax1.plot(system.mesured[:, 2], 'o', color = 'g', mew = 0.5, label = 'Altura medida do
tanque 3')
    ax1.plot(system.state_cekf[0, :], color = 'b', mew = 0.5, label = 'Estado estimado do
tanque 1')
    ax1.plot(system.state_cekf[1, :], color = 'r', mew = 0.5, label = 'Estado estimado do
tanque 2')
    ax1.plot(system.state_cekf[2, :], color = 'g', mew = 0.5, label = 'Estado estimado do
tanque 3')
    plt.legend(loc=2).set_alpha(0.6)
    ax1.set_ylim([4, 25])

    ax2 = plt.subplot2grid((3,1), (2, 0))
    ax2.plot(system.Q1, color = '#c3a700', label = 'Vazão Q1')
    ax2.plot(system.Q2, color = '#fdff00', label = 'Vazão Q2')
    plt.ylabel('Vazão (cm³/s)')
    plt.xlabel('Tempo (s)')
    ax2.set_ylim([14, 26])
    plt.legend(loc=5)

    plt.show()

    fig2 = plt.figure()
    ax11 = plt.subplot2grid((2,1), (0, 0))
    plt.ylabel('Resíduo absoluto (cm)')

    ax11.plot(abs(system.mesured[:, 0] - system.state_cekf[0, :]), color = 'b', label =
'Resíduo de medida do tanque 1')
    ax11.plot(abs(system.mesured[:, 1] - system.state_cekf[1, :]), color = 'r', label =
'Resíduo de medida do tanque 2')
    ax11.plot(abs(system.mesured[:, 2] - system.state_cekf[2, :]), color = 'g', label =
'Resíduo de medida do tanque 3')

    plt.legend(loc=2)

    crit = stats.chi2.ppf(q = 0.99, df=3)

    ax12 = plt.subplot2grid((2,1), (1, 0))
    ax12.semilogy(system.c_erro[1:], color = 'k')
    ax12.axhline(crit, color = 'r', linewidth = 2)
    plt.plot([],[], color = 'k', label = 'Teste estatístico')
    plt.plot([],[], color = 'r', label = 'Limite de confiança')
    plt.legend(loc = 2)
    plt.xlabel('Tempo (s)')
```

```
plt.show()
```

Simulação falha 3

```
# Filtro em estado ótimo com falha 2

system = three_tanks()
Q10 = 20
Q20 = 15
system.input_var(Q10, Q20, n = 600, delta1 = 150, step1 = 5)
system.steady_state()
h0 = [system.h1ss, system.h2ss, system.h3ss]
Rsim = [0.1] * 3
Qsim = [0.005] * 3
system.simulate(h0, Qsim, Rsim, falha = 2)

y0=[11, 10, 9]
P0=np.eye(3)*5
R0=np.eye(3) * Rsim[0] ** 2
Q0=np.eye(3) * Qsim[0] ** 2

system.CEKF(y0, P0, Q0, R0)

fig = plt.figure()
ax1 = plt.subplot2grid((3,1), (0, 0), rowspan=2)
plt.ylabel('Altura do tanque 1 (cm)')

ax1.plot(system.mesured[:, 0], 'o', color = 'b', mew = 0.5, label = 'Altura medida do tanque 1')
ax1.plot(system.mesured[:, 1], 'o', color = 'r', mew = 0.5, label = 'Altura medida do tanque 2')
ax1.plot(system.mesured[:, 2], 'o', color = 'g', mew = 0.5, label = 'Altura medida do tanque 3')
ax1.plot(system.state_cekf[0, :], color = 'b', mew = 0.5, label = 'Estado estimado do tanque 1')
ax1.plot(system.state_cekf[1, :], color = 'r', mew = 0.5, label = 'Estado estimado do tanque 2')
ax1.plot(system.state_cekf[2, :], color = 'g', mew = 0.5, label = 'Estado estimado do tanque 3')
plt.legend(loc=2).set_alpha(0.6)
ax1.set_ylim([4, 25])

ax2 = plt.subplot2grid((3,1), (2, 0))
ax2.plot(system.Q1, color = '#c3a700', label = 'Vazão Q1')
ax2.plot(system.Q2, color = '#fdff00', label = 'Vazão Q2')
plt.ylabel('Vazão (cm³/s)')
```

```
    plt.xlabel('Tempo (s)')
    ax2.set_ylim([14, 26])
    plt.legend(loc=5)

    plt.show()

    fig2 = plt.figure()
    ax11 = plt.subplot2grid((2,1), (0, 0))
    plt.ylabel('Resíduo absoluto (cm)')

    ax11.plot(abs(system.mesured[:, 0] - system.state_cekf[0, :]), color = 'b', label =
'Resíduo de medida do tanque 1')
    ax11.plot(abs(system.mesured[:, 1] - system.state_cekf[1, :]), color = 'r', label =
'Resíduo de medida do tanque 2')
    ax11.plot(abs(system.mesured[:, 2] - system.state_cekf[2, :]), color = 'g', label =
'Resíduo de medida do tanque 3')

    plt.legend(loc=2)

    crit = stats.chi2.ppf(q = 0.99, df=3)

    ax12 = plt.subplot2grid((2,1), (1, 0))
    ax12.semilogy(system.c_erro[1:], color = 'k')
    ax12.axhline(crit, color = 'r', linewidth = 2)
    plt.plot([],[], color = 'k', label = 'Teste estatístico')
    plt.plot([],[], color = 'r', label = 'Limite de confiança')
    plt.legend(loc = 2)
    plt.xlabel('Tempo (s)')

    plt.show()
```