Lista 5

Luiz Georg 15/0041390

11 de novembro de 2021

Questão 1

Os dois filtros implementados usam uma planta comum. O diagrama de blocos utilizado para ambos é o mesmo, alternando apenas as funções utilizadas nas etapas de predição e atualização. As Figs. 1 to 3 mostram toda a parte independente do filtro de Kalman propriamente dito. As simulações foram realizadas utilizando as mesmas condições iniciais e os mesmos valores de ruídos para ambos os filtros. Para reproduzir as simulações (com nova condição inicial e ruído), basta navegar à pasta dos códigos anexos e executar o script `main.sce`. Para efeitos de visualização do trabalho, as simulações mostradas aqui foram escolhidas de um caso com grande erro inicial.

Funções da Planta (`funcoes_planta.sci`)

```
function [x_] = modeloPlanta(x, u)
    // Modelo contínuo da planta. Calcula a derivada do vetor de estados
    // a partir do vetor de estados e da entrada de controle.
    // Entrada:
    // x: vetor de estados
    // u: vetor de controle
    // Saída:
    // x_: derivada do vetor de estados

    x_(1, :) = x(2, :);
    x_(2, :) = - x(2, :) .* abs(x(2, :)) + u;
end


function [y] = modeloSaidas(x)
    // Modelo contínuo do sensor. Calcula as saídas do modelo a partir
    // do vetor de estados.
    // Entrada:
    // x: vetor de estados

    y = 10 * exp(-x(1, :));
end
```

Figura 1: Modelo do sistema no software xcos.

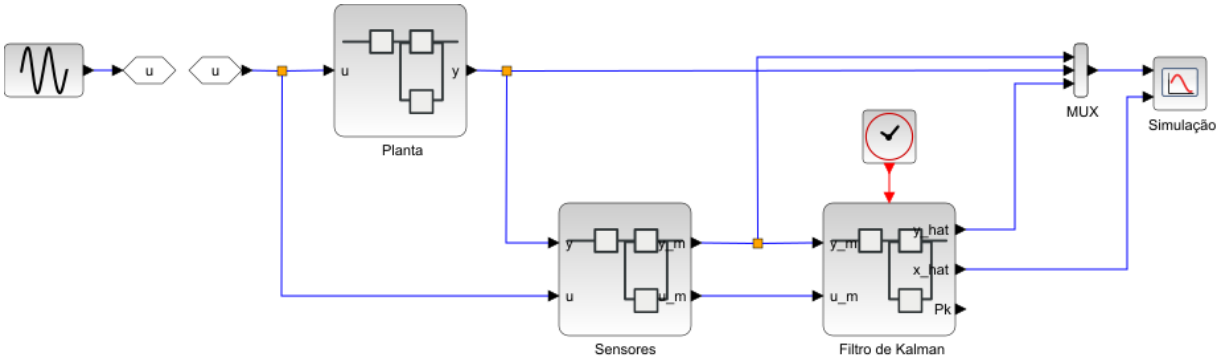


Figura 2: Superbloco *Planta*.

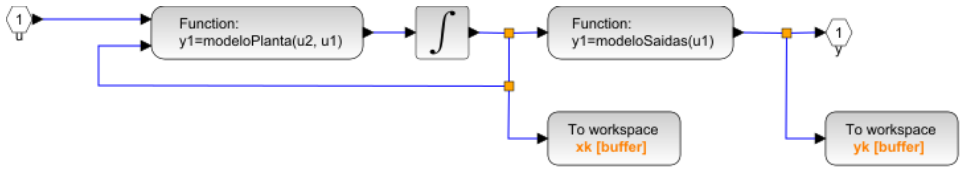
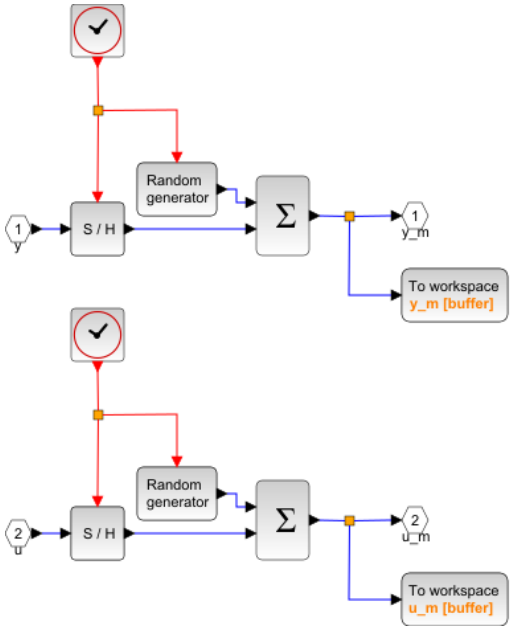
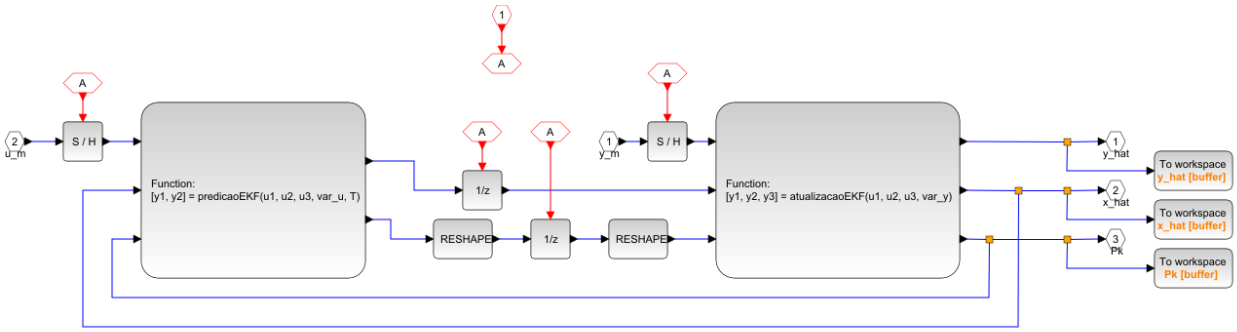


Figura 3: Superbloco *Sensores*.



Questão 2

Figura 4: Superbloco *Filtro de Kalman* para o Filtro de Kalman Extendido.



Funções de Linearização da planta (`funcoes_planta.sci`)

```
function [F] = JacobianoF(x, u, T)
    // Calcula o jacobiano da função de planta relativo ao vetor de estados.
    // Entrada:
    // x: vetor de estados
    // u: vetor de controle
    // T: período de amostragem

    F = [0, 1;
        0, -2 * abs(x(2))];
    F = eye(F) + (F * T);
```

```
end


function [G] = JacobianoG(x, u, T)
    // Calcula o jacobiano da função de planta relativo ao vetor de entrada.
    // Entrada:
    // x: vetor de estados
    // u: vetor de controle
    // T: período de amostragem

    G = [0;
         1];
    G = G * T;
end


function [H] = JacobianoH(x, u, T)
    // Calcula o jacobiano da função de saída relativo ao vetor de estados.
    // Entrada:
    // x: vetor de estados
    // u: vetor de controle
    // T: período de amostragem

    H = [-10 * exp(-x(1)), 0];
end
```

Funções do EKF (funcoes_EKF.sci)

```
function [x_inter, P_inter] = predicaoEKF(u, x_old, P_old, Q, T)
    // Efetua a etapa de predição do EKF. Calcula as matrizes jacobianas
    // do modelo e os valores estimados dos estados e da variância.
    // Entrada:
    // u: vetor de entrada
    // x_old: estado anterior
    // P_old: matriz de covariância do estado anterior
    // Q: matriz de covariância do ruído da entrada
    // T: período de amostragem
    // Saída:
    // x_inter: estado estimado
    // P_inter: matriz de covariância estimada

    F = JacobianoF(x_old, u, T);
    G = JacobianoG(x_old, u, T);

    x_inter = x_old + T * modeloPlanta(x_old, u);
    P_inter = F * P_old * F' + G * Q * G';
end


function [y_hat, x_hat, P_hat] = atualizacaoEKF(y_m, x_inter, P_inter, R)
    // Efetua a etapa de atualização do EKF. Calcula a matriz jacobiana
    // do sensor e os valores estimados de y e da variância.
    // Entrada:
    // y_m: valor do sensor
    // x_inter: estado estimado
    // P_inter: matriz de covariância do estado estimado
    // R: matriz de covariância do sensor
    // Saída:
    // y_hat: valor estimado da saída
    // x_hat: estado estimado atualizado
    // P_hat: matriz de covariância do estado estimado atualizado

    H = JacobianoH(x_inter);
```

```
    y_hat = modeloSaidas(x_inter);
    ey = y_m - y_hat;
    S = (H * P_inter * H') + R;
    K = P_inter * H' / S;
    x_hat = x_inter + (K * ey);
    I_KH = eye(P_inter) - K * H;
    P_hat = I_KH * P_inter * I_KH' + K * R * K';
end
```

Figura 5: Gráfico da potência da planta, potência medida, e potência estimada para o EKF.

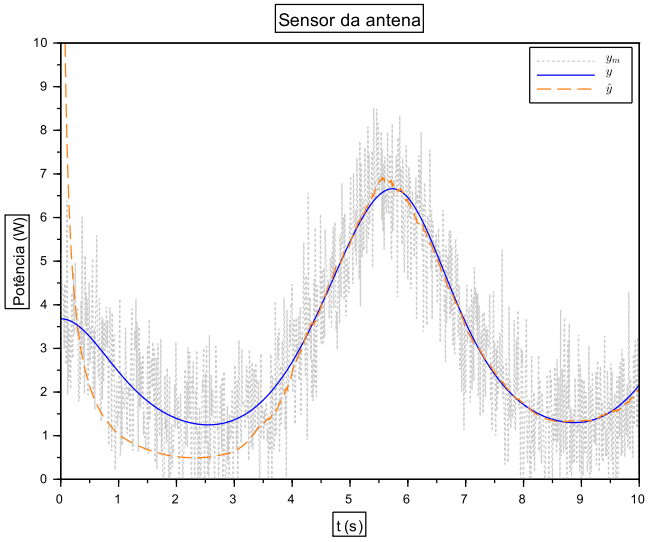


Figura 6: Gráfico do erro entre potência da planta e potência estimada para o EKF.

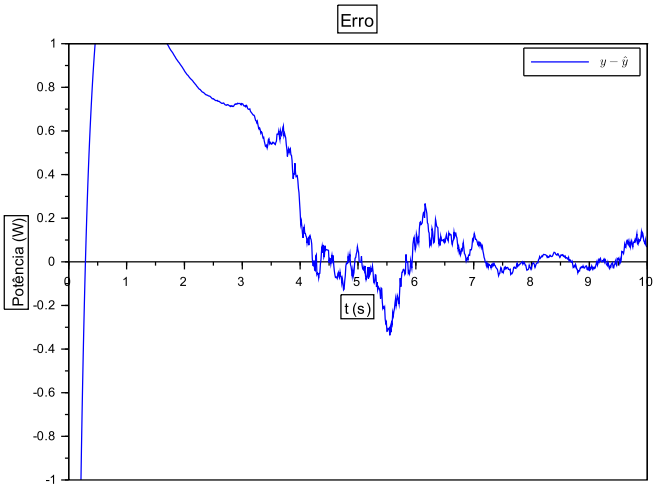


Figura 7: Gráfico dos estados da planta e estimados para o EKF.

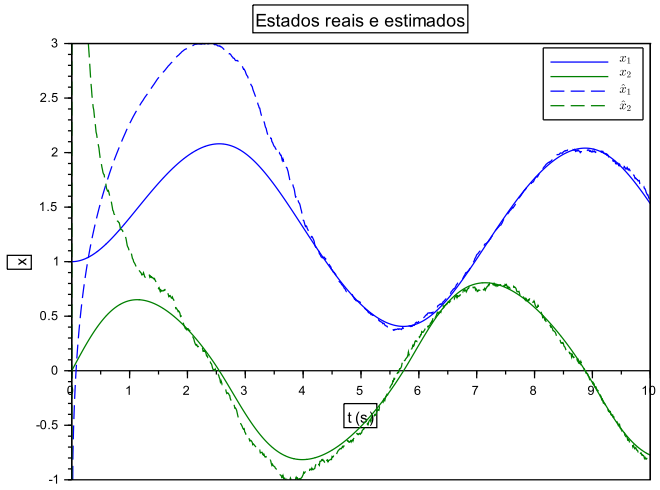


Figura 8: Gráfico da incerteza estimada para o EKF.

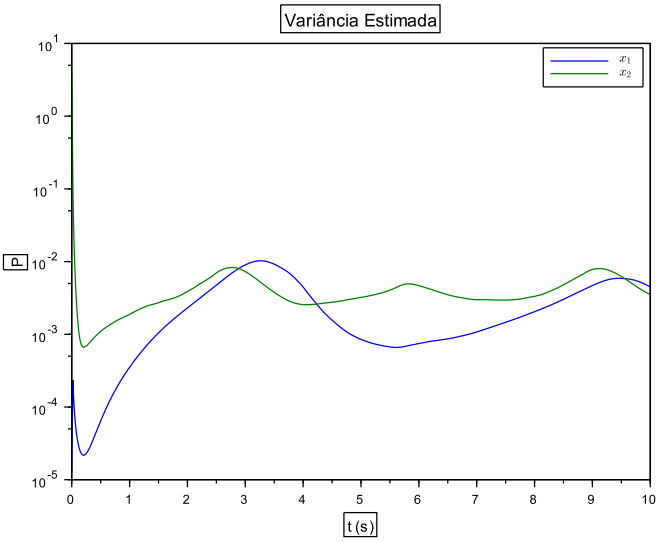
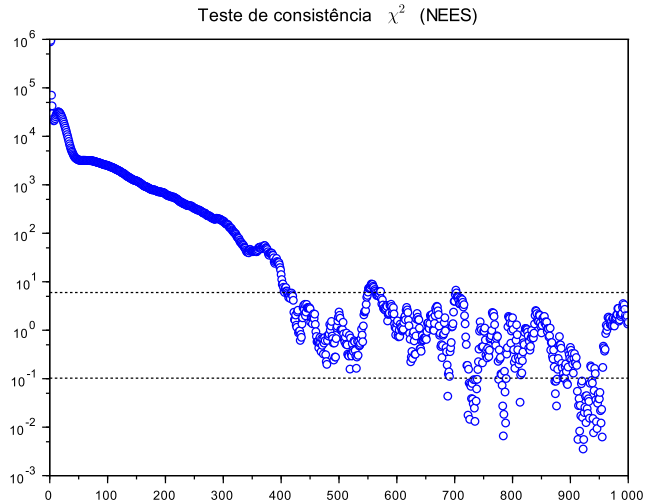
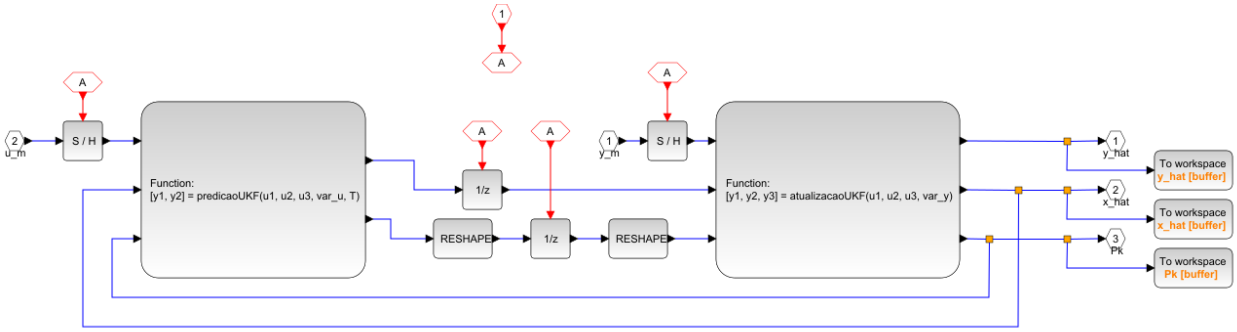


Figura 9: Gráfico da análise de consistência estatística χ^2 para o EKF.



Questão 3

Figura 10: Superbloco *Filtro de Kalman* para o Filtro de Kalman Unscented.



Funções da transformação Unscented (`funcoes_UKF.sci`)

```
function [mu_, cov_, X, Y, w] = unscentedTransform(mu, cov, fun)
    // Calcula a transformada Unscented de uma distribuição com uma
    // função não linear
    // Entrada:
    //  mu: vetor de médias
    //  cov: matriz de covariância
    //  fun: função não linear
    // Saída:
    //  mu_: vetor de médias após transformação
    //  cov_: matriz de covariância após transformação
    //  X: pontos sigma da distribuição original
    //  Y: pontos sigma da distribuição transformada
    //  w: pesos dos pontos

    [X, w] = sigPoints(mu, cov);
    Y = fun(X);
    [mu_, cov_] = meanCov(Y, w);
end


function Pxy = crossCov(X, Y, x, y, w)
    // Calcula a covariância cruzada entre duas distribuições sigma
    // Entrada:
    //  X: pontos sigma da primeira distribuição
    //  Y: pontos sigma da segunda distribuição
    //  w: pesos dos pontos sigma
    //  x: vetor de médias da primeira distribuição
    //  y: vetor de médias da segunda distribuição

    Pxy = (X - x(:, ones(w))) * ((Y - y(:, ones(w))) .* w(ones(y), :))';
end

function [mu, cov] = meanCov(X, w)
    // Calcula a média e a covariância de uma distribuição sigma
    // Entrada:
    //  X: pontos sigma da distribuição
    //  w: pesos dos pontos sigma
    // Saída:
    //  mu: vetor de médias
    //  cov: matriz de covariância

    mu = X * w';
    cov = X - mu(:, ones(w));
    cov = cov * (cov .* w(ones(mu), :))';
end
```

```
function [X, w] = sigPoints(mu, cov)
    // Calcula os pontos sigma a partir da média e covariância
    // Entrada:
    //  mu: vetor de médias
    //  cov: matriz de covariância
    // Saída:
    //  X: pontos sigma
    //  w: pesos dos pontos sigma

    n = length(mu);
    lambda = 3 - n;
    kappa = 3;
    KP = sqrt(3) * chol(cov)';
    X = [zeros(mu), KP, -KP];
    w = (1 / (2 * kappa))(:, ones(1, 2 * n + 1));
    w(1) = lambda / kappa;
    X = X + mu(:, ones(w));
end
```

Funções do UKF (funcoes_UKF.sci)

```
function [x_inter, P_inter] = predicaoUKF(u, x_old, P_old, Q, T)
    // Efetua a etapa de predição do UKF. Calcula as matrizes jacobianas
    // do modelo e os valores estimados dos estados e da variância.
    // Entrada:
    // u: vetor de entrada
    // x_old: estado anterior
    // P_old: matriz de covariância do estado anterior
    // Q: matriz de covariância do ruído da entrada
    // T: período de amostragem
    // Saída:
    // x_inter: estado estimado
    // P_inter: matriz de covariância estimada

    n = length(x_old);
    f = deff("x_ = @(x) x + T * modeloPlanta(x, u(:, ones(1, 2 * n + 1)))");
    G = JacobianoG(x_old, u, T);

    [x_inter, P_inter] = unscentedTransform(x_old, P_old, f);
    P_inter = P_inter + G * Q * G';
end


function [y_hat, x_new, P_new] = atualizacaoUKF(y_m, x_inter, P_inter, R)
    // Efetua a etapa de atualização do UKF. Calcula a matriz jacobiana
    // do sensor e os valores estimados de y e da variância.
    // Entrada:
    // y_m: valor do sensor
    // x_inter: estado estimado
    // P_inter: matriz de covariância do estado estimado
    // R: matriz de covariância do sensor
    // Saída:
    // y_hat: valor estimado do sensor
    // x_new: estado estimado atualizado
    // P_new: matriz de covariância do estado estimado atualizado

    h = deff("y = @(x) modeloSaidas(x)");

    [y_hat, S, X, Y, w] = unscentedTransform(x_inter, P_inter, h);
    S = S + R;
    Pxy = crossCov(X, Y, x_inter, y_hat, w);
```

```
    K = Pxy / S;
    x_new = x_inter + K * (y_m - y_hat);
    P_new = P_inter - K * Pxy';
end
```

Figura 11: Gráfico da potência da planta, potência medida, e potência estimada para o UKF.

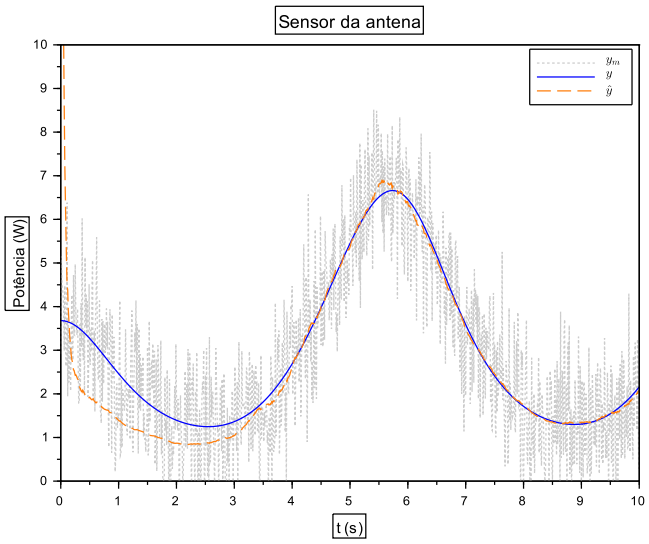


Figura 12: Gráfico do erro entre potência da planta e potência estimada para o UKF.

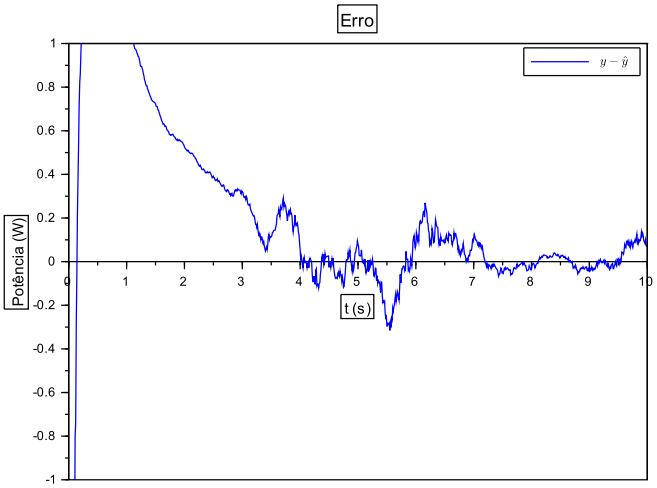


Figura 13: Gráfico dos estados da planta e estimados para o UKF.

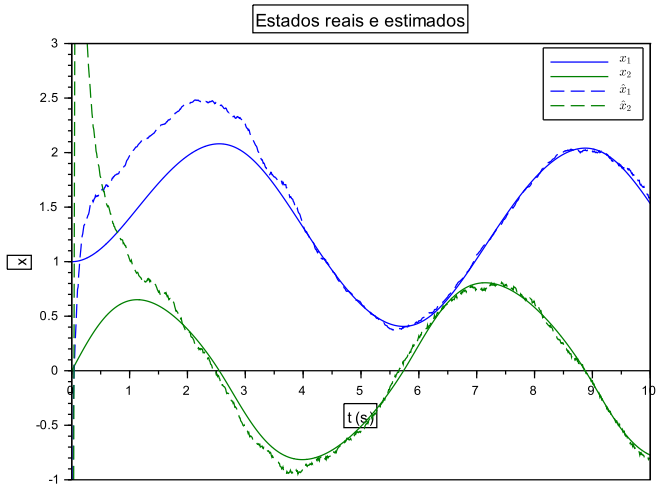


Figura 14: Gráfico da incerteza estimada para o UKF.

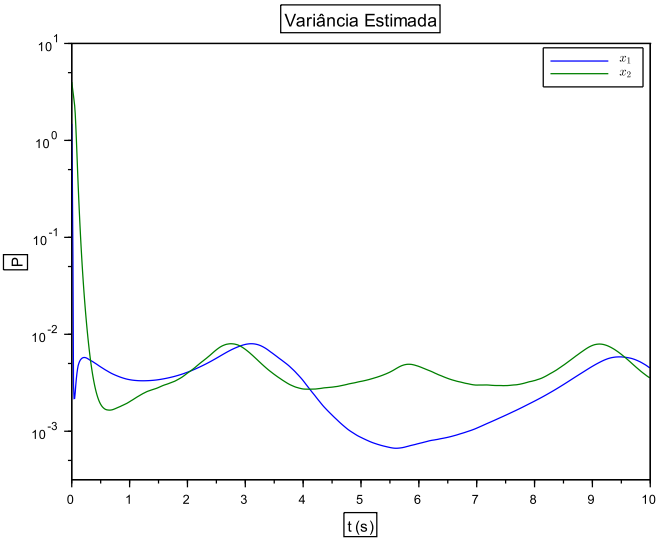


Figura 15: Gráfico da análise de consistência estatística χ^2 para o UKF.