Filtro de Kalman extendido: ejemplo péndulo

Presentación en vídeo: http://personales.upv.es/asala/YT/V/ekfml1.html y http://personales.upv.es/asala/YT/V/ekfml2.html

Este código ejecutó correctamente en Matlab ***R2019a***

Motivación y Objetivos:

El Filtro de Kalman obtenido a partir de la linealización alrededor de la trayectoria de estado estimado da lugar al filtro de Kalman extendido. El objetivo de este código es compararlo con el filtro de Kalman lineal en un modelo de un péndulo no lineal. Usaremos la implementación del mismo que provee Matlab en su comando **`extendedKalmanFilter`**.

Tabla de Contenidos

1.- Modelado

Modelo continuo de péndulo con dos puntos de equilibrio (estable $x_1 = 0$, inestable $x_1 = \pi$). Las ecuaciones de estado y salida son:

```
dxdt=@(x,u) [x(2); ... %deriv. posicion = velocidad
             -sin(x(1))-0.2*x(2)-0.25*x(2)^3+u];%deriv. velocidad = aceleracion
EcSalida=@(x,u) x(1); %medimos la posición
```

Discretizamos la ec. de estado por método de Euler (por su sencillez y porque otros métodos como ZOH, Tustin no aplican directamente a sistemas no-lineales), $x_{k+1} \approx x_k + T_s \cdot \frac{dx}{dt}$:

```
Ts=0.06;
EcEstadoDiscreto=@(x,u) x+Ts*dxdt(x,u);
```

La ecuación de salida, evidentemente, no se discretiza porque no hay dinámica.

Modelo con ruidos de proceso y medida:

Simularemos con `EcEstadoDiscreto(x,u)+w`, siendo w un ruido de proceso de distribución normal bidimensional con una cierta matriz de varianzas-covarianzas, y mediremos con `Ecsalida(x,u)+v`, siendo v un ruido de medida de distrib. normal.

1.1- Linealización

Obtengamos jacobianos y modelos linealizados mediante Symbolic toolbox:

```
x_s=sym('x',[2,1]);u_s=sym('u');
x_s
```

```
x_s =
```

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

```
u_s
```

```
u_s = u
```

```
EcEstado_simbolico=EcEstadoDiscreto(x_s,u_s)
```

```
EcEstado_simbolico =
```

$$\begin{pmatrix} x_1 + \frac{3\,x_2}{50} \\ -\frac{3\,{x_2}^3}{200} + \frac{247\,x_2}{250} + \frac{3\,u}{50} - \frac{3\,\sin(x_1)}{50} \end{pmatrix}$$

```
LinealizacionA_symbolic=jacobian(EcEstado_simbolico,x_s)
```

```
LinealizacionA_symbolic =
```

$$\begin{pmatrix} 1 & \frac{3}{50} \\ -\frac{3\,\cos(x_1)}{50} & \frac{247}{250} - \frac{9\,{x_2}^2}{200} \end{pmatrix}$$

Esta matriz $A(x)$, dependiente del estado, es la que utiliza el filtro de Kalman extendido. Vamos a convertirla en una función de Matlab (numérica) en vez de un objeto de la symbolic toolbox:

```
LinealizacionA_Compilada=matlabFunction(LinealizacionA_symbolic,'Vars',{x_s,u_s});
```

Comprobamos el funcionamiento, obteniendo la linealización en el origen (estable):

```
Aorigen=LinealizacionA_Compilada([0;0],0)
```

```
Aorigen = 2×2
    1.0000    0.0600
   -0.0600    0.9880
```

```
abs(eig(Aorigen))
```

```
ans = 2×1
    0.9958
    0.9958
```

y también la linealización en el equilibrio superior (inestable):

```
Aarriba=LinealizacionA_Compilada([pi;0],0)
```

```
Aarriba = 2×2
    1.0000    0.0600
    0.0600    0.9880
```

```
abs(eig(Aarriba))
```

```
ans = 2×1
    0.9337
    1.0543
```

El resto de matrices B,C,D tienen jacobiano constante (lineal en entradas, sensor lineal).

```
LinealizacionB=eval(jacobian(EcEstado_simbolico,u_s))%no depende de x, hacemos eval y p
```

```
LinealizacionB = 2×1
         0
    0.0600
```

```
LinealizacionC=eval(jacobian(EcSalida(x_s,u_s),x_s)) %no depende de x, hacemos eval y p
```

```
LinealizacionC = 1×2
     1     0
```

```
LinealizacionD=eval(jacobian(EcSalida(x_s,u_s),u_s))%no depende de x, hacemos eval y pu
```

```
LinealizacionD = 0
```

El modelo linealizado en el punto de equilibrio inferior (si queremos comparar con el observador puramente lineal), como función numérica de Matlab, lo escribiremos así:

```
EcEstadoDiscreto_LinealizadoOrigen=@(x,u) Aorigen*x+LinealizacionB*u;
```

2.-Simulación del filtro de Kalman extendido

Haremos distintos tipos de pruebas: observador lineal, filtro de Kalman extendido, y simulación no-lineal pero corrección con ganancia obtenida de un observador lineal.

```
tipomodelo='NoLineal';tipojacobiano='VarianteEnTiempo'; %no-lineal EKF
%tipomodelo='NoLineal';tipojacobiano='Constante'; %converge a ganancia observ. estacion
%tipomodelo='Lineal'; tipojacobiano='Constante'; %operamos con modelo linealizado en or

if(strcmp(tipomodelo,'NoLineal'))
    observador= extendedKalmanFilter(EcEstadoDiscreto,EcSalida);
else %lineal
    observador= extendedKalmanFilter(EcEstadoDiscreto_LinealizadoOrigen,EcSalida);
end
if(strcmp(tipojacobiano,'VarianteEnTiempo'))
    observador.StateTransitionJacobianFcn=LinealizacionA_Compilada;
```

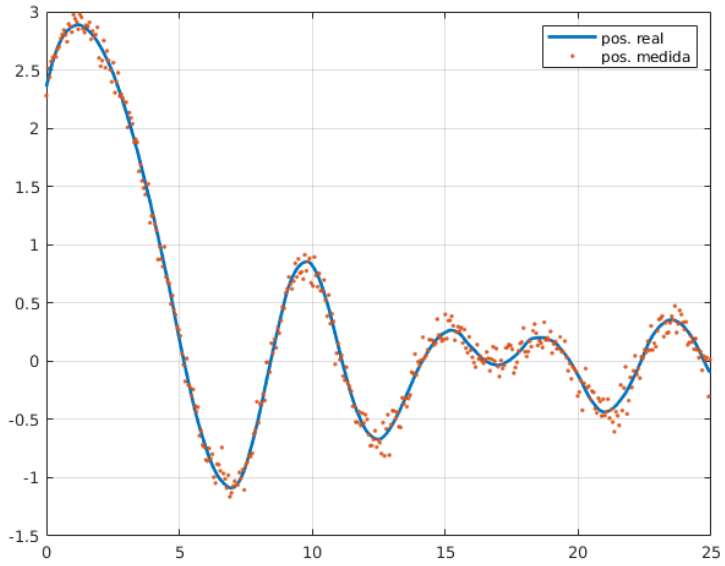
```
else %Jacobianos constantes, aproximados por linealizado
    observador.StateTransitionJacobianFcn=@(x,u) Aorigen;
end
observador.MeasurementJacobianFcn=@(x,u) LinealizacionC;

%parámetros de varianza de proceso, medida y estimado inicial:
observador.MeasurementNoise=0.08^2;
observador.ProcessNoise=diag([0.001 0.02].^2);
observador.StateCovariance=eye(2)*1.25^2;
rng(1234) %para poder comparar con distintos parámetros la misma serie de datos
xreal=[0.75*pi;1]; %condiciones iniciales proceso real
observador.State=[0.6*pi;0]; %condiciones iniciales observador
Tiempos=0:Ts:25;
uu=-0.25*sign(sin(1.4*Tiempos));
N=length(Tiempos);
gxr=zeros(N,2);grm=zeros(N,1);grstd=zeros(N,2);
for k=1:N

    medida=EcSalida(xreal,uu(k))+randn()*0.08;%Lectura de sensor con ruido real
    xestim(k,:)=correct(observador,medida,uu(k)); %corrector

    grstd(k,:)=sqrt(diag(observador.StateCovariance))';
    grm(k,:)=medida;
    grx(k,:)=xreal';


    xreal=EcEstadoDiscreto(xreal,uu(k))+diag([.001;.02])*randn(2,1); %simulación del es
    predict(observador,uu(k)); %estimación en bucle abierto de observador para siguient
end
```

2.1 Evaluación de resultados

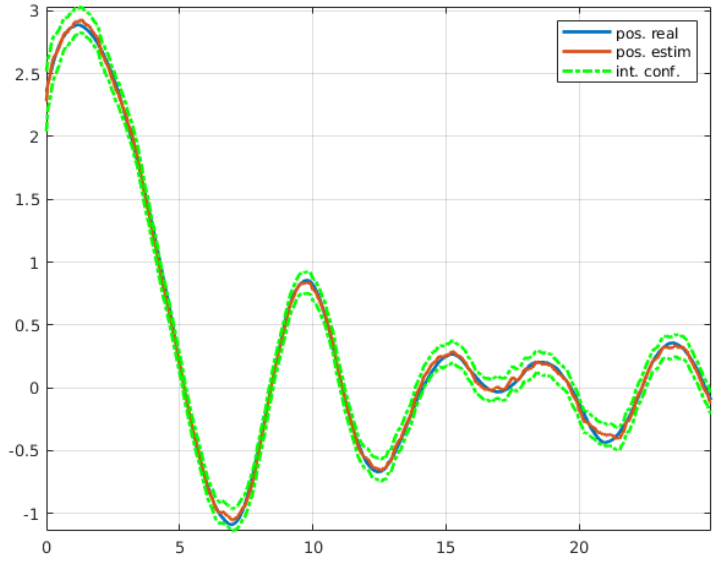
```
norm(grx-xestim,'fro') %Frobenius: raíz cuadrada de suma de cuadrados de todo
```

```
ans = 1.6812
```

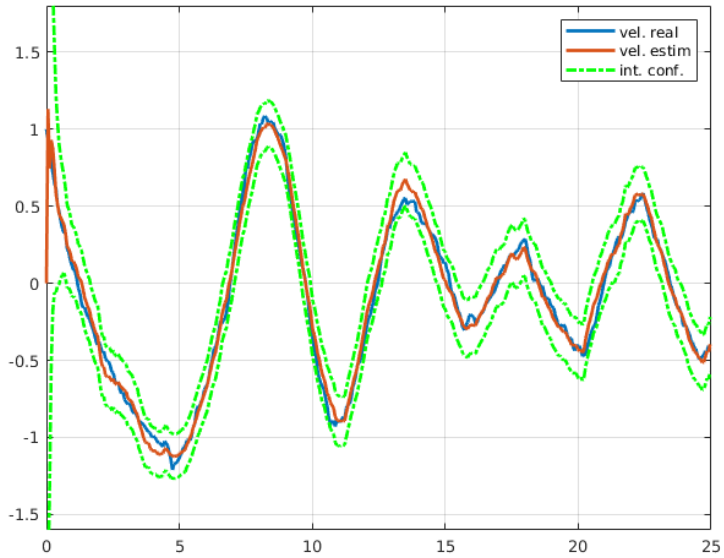
```
plot(Tiempos,grx(:,1),Tiempos,grm,'.','LineWidth',2)
legend('pos. real','pos. medida'), grid on
```




```
plot(Tiempos,[grx(:,1) xestim(:,1)],Tiempos, [xestim(:,1)-3*grstd(:,1) xestim(:,1)+3*gr
legend('pos. real','pos. estim','int. conf.'), grid on, axis tight
```




```
plot(Tiempos,[grx(:,2) xestim(:,2)],Tiempos, [xestim(:,2)-3*grstd(:,2) xestim(:,2)+3*gr
legend('vel. real','vel. estim','int. conf.'), grid on, ylim([-1.6 1.8])
```


vel. real
vel. estim
int. conf.
1.5
1
0.5
0
-0.5
-1
-1.5
0
5
10
15
20
25