Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica
EL7012 – Control inteligente de Sistemas

MANUAL DE USO: TOOLBOX PARA IDENTIFICACIÓN DE MODELO DIFUSO

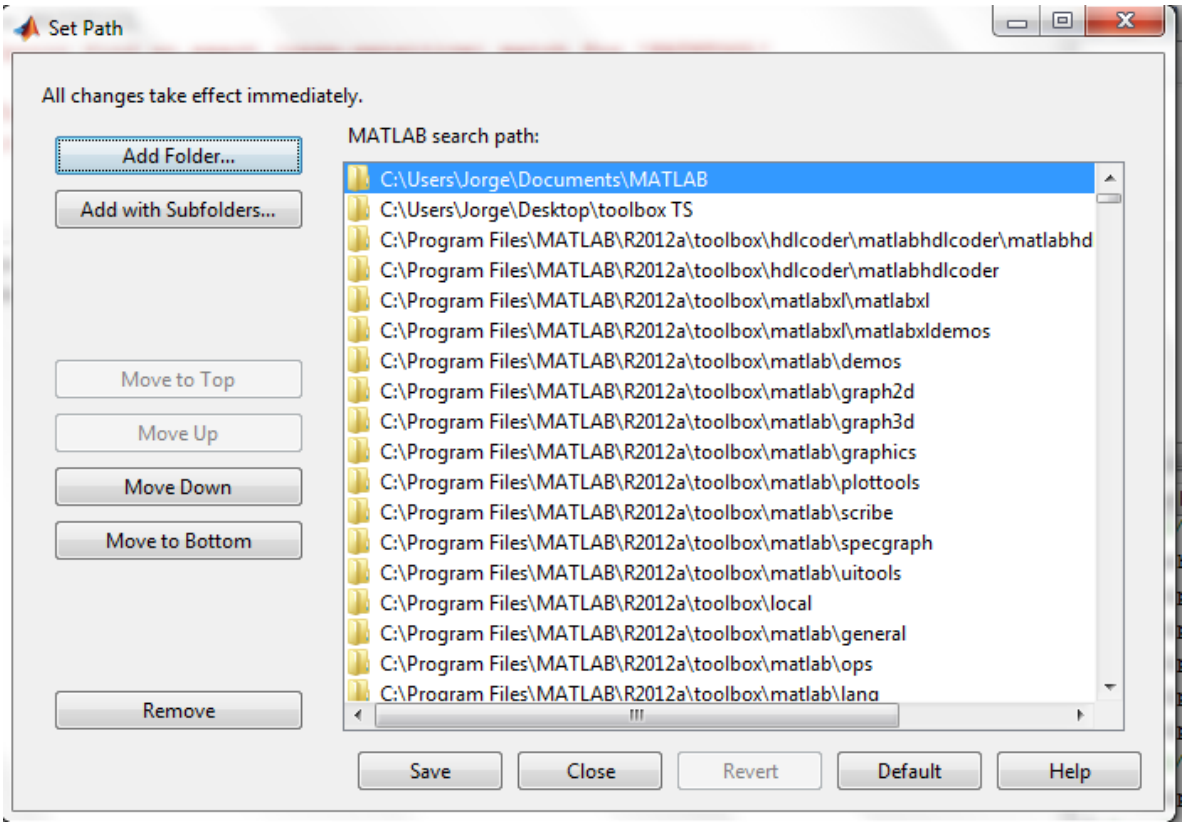
Jorge Collado

Contenido

1. Instalación

El primer paso corresponde a la instalación del toolbox, para esto se deben seguir los siguientes pasos:

1. Abrir el software MATLAB instalado en nuestro computador
2. En la ventana de comandos de MATLAB escribir “pathtool”, lo que desplegará una ventana como se muestra a continuación



3. En la ventana Set Path se usa la opción Add Folder, se busca y selecciona el toolbox, una vez hecho esto se presiona el botón Save y luego Close.

Si los pasos fueron seguidos correctamente el toolbox ya se encuentra instalado y podremos usar las funciones del toolbox sin tener que cambiar de directorio.

2. Descripción de funciones del toolbox

En esta sección se explicarán las principales funciones del toolbox, las funciones que no son detalladas no se usan por el usuario solo son utilizadas por las funciones principales que se describirán a continuación.

2.1. Funciones de modelación

2.1.1. takagisugeno1

Esta función corresponde a la función utilizada para obtener un modelo difuso TS a partir de datos de salida y de datos de entrada del modelo. Su sintaxis es la siguiente:

```
[model, result]=takagisugeno1(iden_y,iden_x,reglas,opcion)
```

La siguiente tabla explica a que corresponden las entradas y salidas de esta función:

Nombre	Entrada/Salida	Descripción
model	Salida	Corresponde a un conjunto de matrices que dan origen al modelo difuso entre ellos model.a, model.b y model.g que son necesarios para luego construir la salida del modleo difuso.-
result	Salida	
Iden_y	Entrada	Es la salida con la que se identifica el modelo difuso.
Iden_x	Entrada	Es la matriz cuyas columnas son las entradas con las que se identificará el modelo.
reglas	Entrada	Corresponde al número de clusters a utilizarse.
opcion	Entrada	Corresponde a la opción con la que se identificará el modelo utilizar “[2 2]”

2.1.2. ysim

Esta función construye la salida estimada a partir de un modelo difuso TS. Su sintaxis es la siguiente:

```
y=ysim(X,a,b,g)
```

La siguiente tabla explica a que corresponden las entradas y salidas de esta función:

Nombre	Entrada/Salida	Descripción
y	Salida	Corresponde a la salida predicha por el modelo difuso obtenido.
X	Entrada	Corresponde a la matriz cuyas columnas son las variables de entrada del modelo difuso obtenido
a	Entrada	Es una matriz que muestra el inverso de los anchos de las funciones de pertenencia para cada entrada del modelo y para cada regla.
b	Entrada	Es la matriz que muestra donde está centrada la función de pertenencia para cada entrada del modelo en cada regla.
g	Entrada	Corresponde a la matriz cuyas filas son los parámetros que acompañan a las entradas del modelo para cada regla

Funciones de identificación

2.1.3. clusters_optimo

Esta función es la encargada de encontrar el número de clusters óptimo de acuerdo a la matriz de entradas candidatas utilizada, para esto esta función grafica los errores de entrenamiento y de test en función del número de clusters en donde es posible sleccionar el número óptimo. Su sintaxis es la siguiente:

```
[errtest,errent]=clusters_optimo(ytest,yent,Xtest,Xent,max_clusters)
```

La siguiente tabla explica a que corresponden las entradas y salidas de esta función:

Nombre	Entrada/Salida	Descripción
errtest	Salida	Es un vector que contiene el error cuadrático medio de test para cada número de clusters.
errent	Salida	Es un vector que contiene el error cuadrático medio de entrenamiento para cada número de clusters.
ytest	Entrada	Corresponde al vector salida de nuestro conjunto de test
yent	Entrada	Corresponde al vector salida de nuestro conjunto de entrenamiento
Xtest	Entrada	Es la matriz cuyas columnas corresponden a las entradas candidatas para el modelo difuso utilizando el conjunto de test.
Xent	Entrada	Es la matriz cuyas columnas corresponden a las entradas candidatas para el modelo difuso utilizando el conjunto de entrenamiento.
max_clusters	Entrada	Es el número máximo de clusters al que se quiere llegar en el estudio de errores según cantidad de clusters.

2.1.4. sensibilidad

Esta función realiza un análisis de sensibilidad dada una cantidad de clusters y una matriz de entradas candidatas del modelo, devolviendo los índices para cada entrada candidata y cual debería eliminarse. Su sintaxis es la siguiente:

```
[p indice]=sensibilidad(yent,Xent,reglas)
```

La siguiente tabla explica a que corresponden las entradas y salidas de esta función:

Nombre	Entrada/Salida	Descripción
p	Salida	Es el número correspondiente a la columna de la matriz de variables de entrada candidatas que se debe eliminar según el análisis de sensibilidad.
indice	Salida	Es el vector que contiene los índices del análisis de sensibilidad para cada una de las variables de entrada candidatas para el modelo.
yent	Entrada	Corresponde al vector salida de nuestro conjunto de entrenamiento
Xent	Entrada	Es la matriz cuyas columnas corresponden a las entradas candidatas para el modelo difuso utilizando el conjunto de entrenamiento.
reglas	Entrada	Corresponde al número de clusters a utilizarse.

2.1.5. errortest

Esta función es utilizada para obtener el error de test dada una matriz candidata y un numero optimo de clusters asociado. Su sintaxis es la siguiente:

```
err=errortest(yent,Xent,ytest,Xtest,reglas)
```

La siguiente tabla explica a que corresponden las entradas y salidas de esta función:

Nombre	Entrada/Salida	Descripción
err	Salida	Es un vector que contiene el error cuadrático medio de test para cada número de clusters.
yent	Entrada	Corresponde al vector salida de nuestro conjunto de entrenamiento
Xent	Entrada	Es la matriz cuyas columnas corresponden a las entradas candidatas para el modelo difuso utilizando el conjunto de entrenamiento.
ytest	Entrada	Corresponde al vector salida de nuestro conjunto de test
Xtest	Entrada	Es la matriz cuyas columnas corresponden a las entradas candidatas para el modelo difuso utilizando el conjunto de test.
reglas	Entrada	Corresponde al número de clusters a utilizarse.

3. Ejemplo

En esta sección se dará un ejemplo de cómo realizar identificación de modelo difuso TS utilizando este toolbox.

Supongamos se tienen 5000 datos de una cierta salida $y(k)$ de un sistema, y 5000 datos de la entrada $u(k)$ de este sistema. El primer paso corresponde a dividir nuestros conjuntos de datos en conjunto de entrenamiento, de test y de validación. Por ejemplo el 50% de los datos se destinan para el entrenamiento mientras que para validación y test se deja el 25% de los datos para cada uno.

Para realizar identificación difusa es necesario seguir los siguientes pasos:

1. Definir conjunto de variables de entrada candidatas para el modelo difuso TS
2. Obtener el número de clústeres mediante los errores de test y entrenamiento
3. Calcular error de test del modelo difuso con las entradas candidatas y clústeres obtenidos
4. Calcular índices de sensibilidad de las entradas candidatas
5. Eliminar variable de menor índice de sensibilidad
6. Si el conjunto de entradas tiene más de una entrada volver al paso 2

Una vez definidos los conjuntos es necesario definir las entradas candidatas para el modelo difuso, supongamos que por nuestra experiencia construyendo modelos lineales para la misma problemática sabemos que utilizando 5 autorregresores para la salida y la entrada presente más 4 autorregresores como variables de entrada candidatas para el modelo, lo más probable es que el modelo final resulte con menos entradas de las iniciales. Construimos en matlab los autorregresores necesarios para $y_{ent}(k)$ y $u_{ent}(k)$ utilizando nuestros conjuntos de entrenamiento y a partir de estos nuestra matriz de variables de entrada candidatas para el modelo:

$$X_{ent} = (y_{ent}(k-1) \quad y_{ent}(k-2) \quad \cdots \quad y_{ent}(k-5) \quad u_{ent}(k) \quad \cdots \quad u_{ent}(k-4))$$

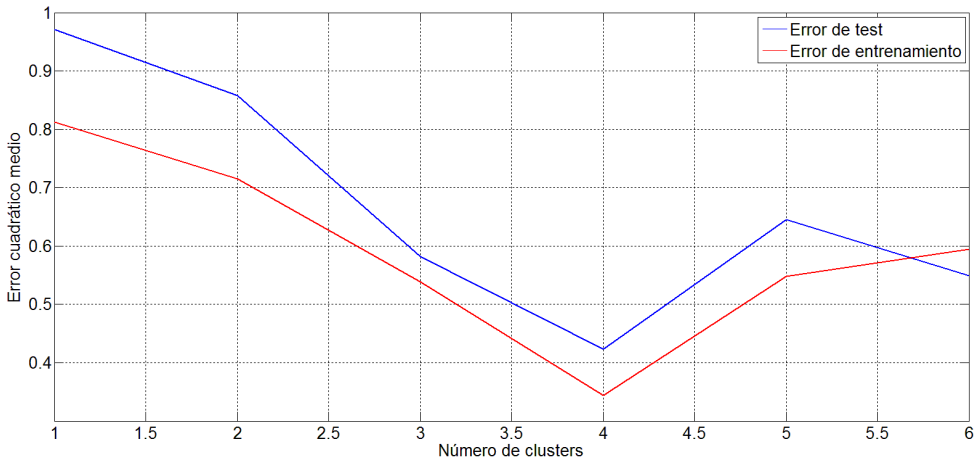
Donde cada una de las entradas es un vector de 2500x1 donde 2500 es el largo del conjunto de entrenamiento con lo que nuestra matriz queda de 2500x10. Hacemos lo mismo con los datos de test ya que será necesario para luego obtener el número óptimo de clusters:

$$X_{test} = (y_{test}(k-1) \quad y_{test}(k-2) \quad \cdots \quad y_{test}(k-5) \quad u_{test}(k) \quad \cdots \quad u_{test}(k-4))$$

Ahora es necesario obtener el número de clusters óptimo para nuestro X_{ent} para esto es necesario utilizar:

```
[errtest,errent]=clusters_optimo(ytest,yent,Xtest,Xent,max_clusters)
```

Utilizamos como entradas: $y_{test}(k)$ [1250x1] , $y_{ent}(k)$ [2500x1], X_{ent} [2500x10], X_{test} [1250x10] y como número máximo de clusters utilizaremos 6. Obteniéndose el siguiente resultado:



Se puede observar que para 4 clusteres el error de test es mínimo por lo tanto este es escogido como el número óptimo de clusters.

En esta parte es necesario caclular el error de test para nuestro número óptimo de clusters y las variables de entrada candidatas, para esto se utiliza:

```
err=errortest(yent,Xent,ytest,Xtest,reglas)
```

Utilizamos como entradas: $y_{test}(k)$ [1250x1] , $y_{ent}(k)$ [2500x1], X_{ent} [2500x10], X_{test} [1250x10] y como reglas el número óptimo de clusters obtenido 4.

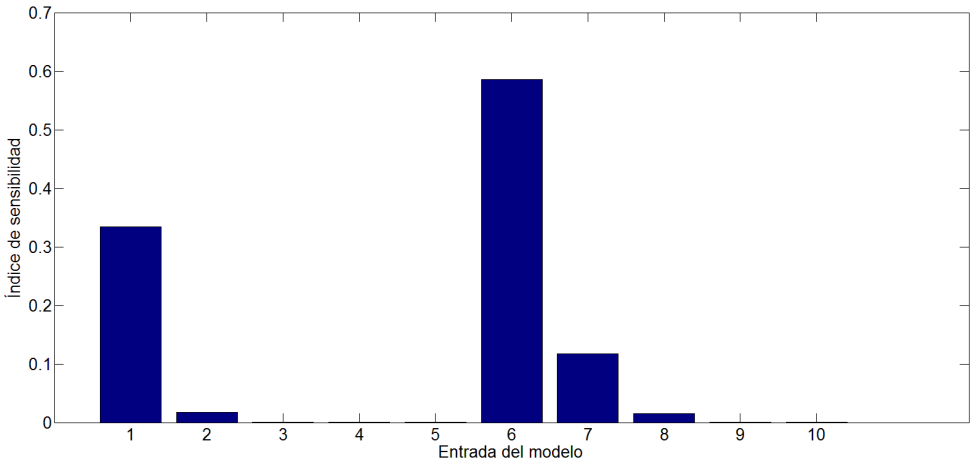
err = 0.4236

Este error debe ser alamcenado en un vector en cada iteración ya que será necesario al final-

Ahora es necesario calcular el índice de sensibilidad de las variables candidatas, para esto se utiliza:

```
[p indice]=sensibilidad(yent,Xent,reglas)
```

Utilizamos como entradas:, $y_{ent}(k)$ [2500x1], X_{ent} [2500x10] y como reglas el número óptimo de clusters, es decir, 4. Obteniendose el siguiente resultado:

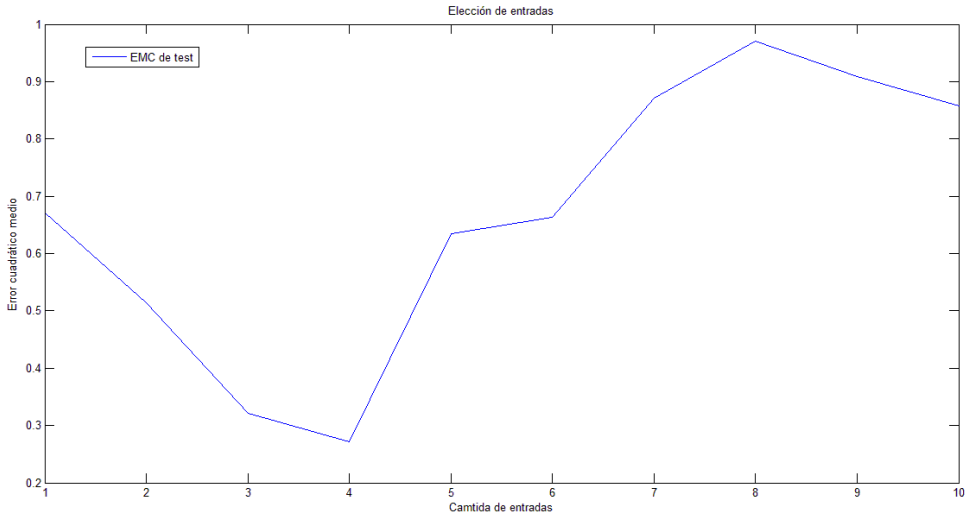


p=3

Por lo tanto se debe eliminar la entrada candidata correspondiente a la tercera columna de X_{ent}, que corresponde a $y\,(k-3)$

Con esto se sigue iterando hasta solo tener una entrada candidata.

Finalmente luego de realizar estos pasos para cada iteración se utiliza el error de test obtenido en cada iteración y se escoge el de menor error, nuestro modelo queda identificado con el número de clusters y las entradas del modelo asociadas a este error:



Acá se ve que el menor error de test corresponde a cuando el modelo contaba con 4 entradas y su número de clusters asociado. Así finalmente se obtiene que el modelo debe ser utilizando 2 clusters y de la forma:

$$\hat{y}(t) = f^{TS}(y(k-1), y(k-2), u\,(k), v\,(k-1))$$

Ahora se puede obtener nuestro modelo utilizando:

```
[model, result]=takagisugeno1(iden_y,iden_x,reglas,opcion)
```

Con nuestra salida de entrenamiento y la matriz de entrenamiento con las variables de entrada del modelo obtenidas, con reglas=2 y opción=[2 2].

Posteriormente se usa:

```
y=ysim(X,a,b,g)
```

Con X como la matriz de entrenamiento de variables de entrada obtenidas y con a, b y g obtenidos dentro de model del paso anterior, y obtenemos la salida de nuestro modelo final.