MODELADO, SIMULACION Y CONTROL DE SISTEMAS DINAMICOS

P.F.PULESTON y F.VALENCIAGA

Nota: Este apunte tiene por objetivo principal introducir al modelado, simulación y control de sistemas dinámicos empleando Matlab. Gran parte del material aquí presentado esta basado en los Control Tutorial for Matlab de la ***Universidad de Michigan****, Demos y Manuales de Matlab. Abundante información adicional y soluciones a muchos otros problemas vinculados al control de sistemas, procesamiento de señal y calculo matricial en general, puede encontrarse en dicha bibliografía.*

Parte I. Tutorial Básico de Matlab

Vectores

Matrices

Polinomios

Funciones

Archivos .m

Archivos .mat

Graficación

Impresión

Ayuda

Importante: Matlab es un programa de cálculo cuyos elementos básicos son arreglos. Sus variables son definidas en su forma más general como matrices.

VECTORES

Para crear un vector fila se utilizan corchetes dentro de los cuales los elementos deben estar separados por espacios o comas:

```
a = [1 2 3 4 5 6 9 8 7]
```

Matlab retorna:

```
a =
1  2  3  4  5  6  9  8  7
```

Nota de utilidad: si se desea que la variable creada no se muestre en pantalla, se debe ejecutar el comando con un ; final.

```
a = [1 2 3 4 5 6 9 8 7];
```

Para crear un vector con elementos entre n1 y n2, equiespaciados en d, el formato es V=[n1:d:n2]. Por ejemplo, para generar un vector de tiempo:

```
t = [0:2:20]
t =
0  2  4  6  8  10  12  14  16  18  20
```

Las operaciones entre escalares y vectores son muy sencillas, por ejemplo:

```
b = a + 2
b =
3  4  5  6  7  8  11  10  9
```

Cuando se realizan operaciones entre vectores deben respetarse las dimensiones:

```
c = a + b
c =
4 6  8  10  12  14  20  18  16
```

MATRICES

La creación de matrices en Matlab es similar a la de vectores (de hecho estos son un caso particular de matrices), sólo que cada fila de elementos debe separarse con un ***;*** o *return*:

```
B = [1 2 3 4; 5 6 7 8; 9 10 11 12]
B =
          1    2    3     4
          5    6    7     8
          9   10   11    12

B = [ 1  2  3  4
5  6  7  8
9 10 11 12]

B =
          1    2    3     4
          5    6    7     8
          9   10   11    12
```

Existen diversas formas de manipular matrices. Es posible tomar determinados componentes de la misma (sean elementos, filas o columnas) de la siguiente manera:

Elemento 1,4:

```
b14=B(1,4)

b14 =
     4
```

Fila 2:

```
fb2=B(2,:)

fb2 =
     5     6     7     8
```

Fragmento del vector fb2:

```
ffb2=fb2(2:4)
fb2=B(2,:)ffb2 =
     6     7     8
```

Se pueden obtener los valores máximos, mínimos, medios u ordenar los elementos de una matriz (*min, median, mean, sort*). Ejemplo:

```
max(fb2)            Mfb2=max(fb2)    [Mfb2,iMfb2]=max
ans = 8             Mfb2 = 8         (fb2)
                                     Mfb2 = 8
                                     iMfb2 = 4
```

Nótese que al no haberse indicado explícitamente en que variable guardar el resultado, este es asignado a la variable auxiliar ans *(esto es general para todas las operaciones).*

Obtención de la matriz traspuesta de B:

```
C = B'
C =
          1    5    9
          2    6   10
          3    7   11
          4    8   12
```

Es importante destacar que la multiplicación puede realizarse entre matrices o elemento a elemento. Ejemplo de multiplicación entre matrices:

```
D = B * C
D =
         30    70   110
         70   174   278
        110   278   446
```

```
D  = C * B
D =
        107   122   137   152
        122   140   158   176
        137   158   179   200
        152   176   200   224
```

Para realizar multiplicación (u otras operaciones en general) elemento a elemento (de matrices de igual dimensión), el operador de ser precedido por un punto:

```
E = [1 2;3 4]
F = [2 3;4 5]
G = E .* F

E =
          1   2
          3   4

F =
          2   3
          4   5

G =
2   6
12  20
```

Un ejemplo claro de cómo operar con matrices o con sus elementos es elevar una matriz cuadrada al cubo:

```
E^3
ans =
         37    54
         81   118
```

o cada uno de sus elementos al cubo:

```
E.^3
```

```
ans =
          1     8
         27    64
```

También se puede obtener en forma sencilla la inversa de una matriz:

```
X = inv(E)
X =
2.0000    1.0000
1.5000   -0.5000
```

y sus autovectores y autovalores:

```
[eiVe,eiva]=eig(E)

eiVe =
   -0.8246   -0.4160
    0.5658   -0.9094

eiva =
   -0.3723         0
         0    5.3723
```

Existe una función que permite obtener el polinomio característico de una matriz:

```
p = poly(E)
p =
1.0000   -5.0000   -2.0000
```

Y pueden calcularse las raíces del polinomio, verificando que coinciden con los autovalores de la matriz:

```
roots(p)

ans =
5.3723
0.3723
```

POLINOMIOS

En Matlab los polinomios son representados por vectores que tienen por elementos a los coeficientes del polinomio en orden descendente. Por ejemplo:

$$s^4 + 3 \cdot s^3 - 15 \cdot s^2 - 2 \cdot s + 9$$

en Matlab sería:

```
x = [1 3 -15 -2 9]
x =
1  3  -15  -2  9
```

Si alguna potencia no aparece en el polinomio deseado:

$$s^4 + 1$$

debe ser representado con ceros en los coeficientes correspondientes:

```
y = [1 0 0 0 1]
```

El polinomio puede ser evaluado en un dado punto a través de la función `polyval`. Por ejemplo, el polinomio anterior en s=2 es evaluado por:

```
z = polyval([1 0 0 0 1],2)
z =
17
```

Si en lugar de indicar un punto se desea evaluar un conjunto de puntos, basta con indicarlo con un vector cuyos elementos sean los puntos deseados.

La forma de obtención de raíces del polinomio ya fue indicada previamente. A modo de ejemplo:

$$s^4 + 3 \cdot s^3 - 15 \cdot s^2 - 2 \cdot s + 9$$

```
roots([1 3 -15 -2 9])
```

```
ans =
5.5745
2.5836
0.7951
0.7860
```

Dos polinomios pueden ser multiplicados a través de la convolución de sus coeficientes:

```
x = [1 2];
y = [1 4 8];
z = conv(x,y)

z =
1  6  16  16
```

Con la función *deconv* pueden dividirse dos polinomios, obteniéndose el resultado y el resto:

```
[xx, R] = deconv(z,y)
xx =
1  2
R =
0  0  0  0
```

Finalmente, cabe destacar dos funciones de gran utilidad:

- *polyfit(X,Y,N)*: dado un conjunto de puntos determinados por los vectores X, Y *plolyfit* da como resultado los coeficientes del polinomio aproximante, basandose en mínimos cuadrados.
- *interp(X,R)*: remuestrea la secuencia en el vector X a una tasa de muestreo R veces mayor, empleando interpolación pasabajo.

FUNCIONES

Funciones Estándar:

Para facilitar las operaciones, Matlab incluye muchas funciones estándar, tales como: sin, cos, log, exp, sqrt. Las mismas están generadas sobre la base de un bloque de código que realiza una tarea específica. Comúnmente, constantes tales como *pi* e *i* (*i*=*j*=sqrt(-1)) han sido también incorporadas a Matlab. Por ejemplo:

```
sin(pi/4)
ans =
0.7071
```

Para determinar el uso de una función basta con escribir: help [nombre función], en la ventana de comando.

Funciones Personalizadas:

Matlab permite crear funciones propias. Estas son subrutinas escritas en texto y guardadas como archivos file.m. Es importante destacar que para que al ejecutar una funcion personal, Matlab sepa donde buscarla, esta debe estar guardada en un subdirectorio que este incluido en el *path* de Matlab. La incorporación de este subdirectorio al *path* de Matlab, se realiza en el archivo *c:*\....*matlab**matlabbrc.m*

A las funciones se le pasan los valores de las variables como argumentos de entrada y retornan las variables de salida indicadas en la línea inicial de definición de la función. Es importante notar que las variables utilizadas en una función son locales. Si se desea incluir un texto de ayuda que aparezca cada vez que se ejecute help [nombre función] en la ventana de comando, basta con escribirlo como comentario a continuación de la línea definición de la función:

```
function [mean,stdev] = stat(x)
%Formato [mean,stdev] = stat(x).  Calcula la media y
%desviación estándard de
```

```
%un vector.
%Parametros de entrada: Vector x.
%Parametros de salida: mean (media) y stdev (desviación
%estándard).
%Función Personal P.F.Puleston 27/4/99.  Corregida
%25/5/99 incluyendose el Help.
n = length(x);
       mean = sum(x) / n;
      stdev = sqrt(sum((x - mean).^2)/n);
```

En este punto vale la pena comentar que para programar Matlab ofrece seis

MODIFICADORES DE FLUJO:

- *If*: junto con *else* y *elseif*, ejecuta un grupo de sentencias según una condición lógica.
- *Switch*, junto con case y o*therwise*, ejecuta diferentes grupos de sentencias dependiendo del valor de algunas condiciones lógicas.
- *While* ejecuta un grupo de sentencias un número indefinido de veces, dependiendo del valor de algunas condiciones lógicas
- *For* ejecuta un grupo de sentencias un número determinado de veces.
- *Try...catch* cambia el control de flujo si un error es detectado durante la ejecución.
- *Return* retorna la ejecución al punto en que se invocó la función.

Un ejemplo de modificadores de flujo es:

```
IF I == J
                A(I,J) = 2;
            ELSEIF ABS(I-J) == 1
               A(I,J) = -1;
```

```
ELSE
   A(I,J) = 0;
END
```

Notese la existencia de operadores de relación: ==, <, >, <=, >=, o ~=.

Otro ejemplo es:

```
[m,n]=size(x);
IF (~((m==1)|(n==1))|(m==1&n==1))
        error('La entrada debe ser un vector');
END
Y=sum(x)/length(x);
```

Notese la existencia de operadores lógicos: &, |, ~ y xor.

ARCHIVOS .M

Son conjuntos de comandos y segmentos de programa que, en forma similar a las funciones, se guardan en un archivo .m, y pueden ser ejecutados al invocarlos desde la ventana de comandos. En líneas generales operan en forma similar a las funciones, siendo su principal diferencia que comparten todas las variables del sistema, es decir sus variables son globales (no existen parámetros de entrada ni de salida).

ARCHIVOS .MAT

- Matlab permite salvar algunas variables o el *workspace* completo en archivos ad hoc con extensión *.mat*. Para ello es necesario emplear el comando:

  save fname X Y Z

  Esto guarda en el archivo *fname* (incluyendose en *fname* el *path* del subdirectorio) las variables X, Y y Z. Si no se indican las variables, salva todo el *workspace*. Si no se indica el *fname*, salva por defecto en un archivo

denominado *matlab.mat,* en el subdirectorio raíz de Matlab. También es opcional la posibilidad de que los datos sean guardados como ASCII.

– Para cargar los datos nuevamente en el workspace basta con ejecutar el comando:

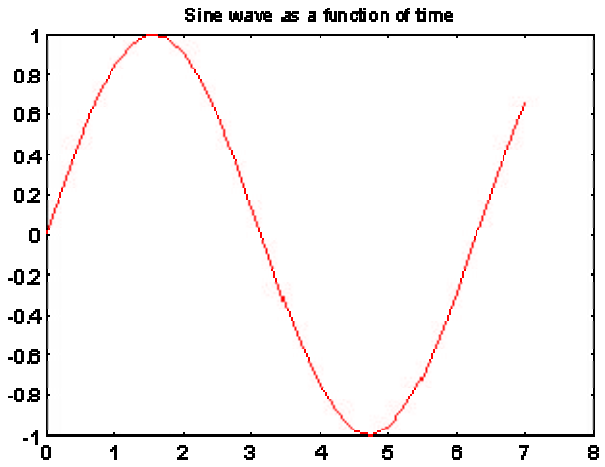
load fname

sin que sea necesario indicar ni la extensión ni el *path*, si *fname* fue salvado como .mat en un subdirectorio perteneciente al *path* de Matlab.

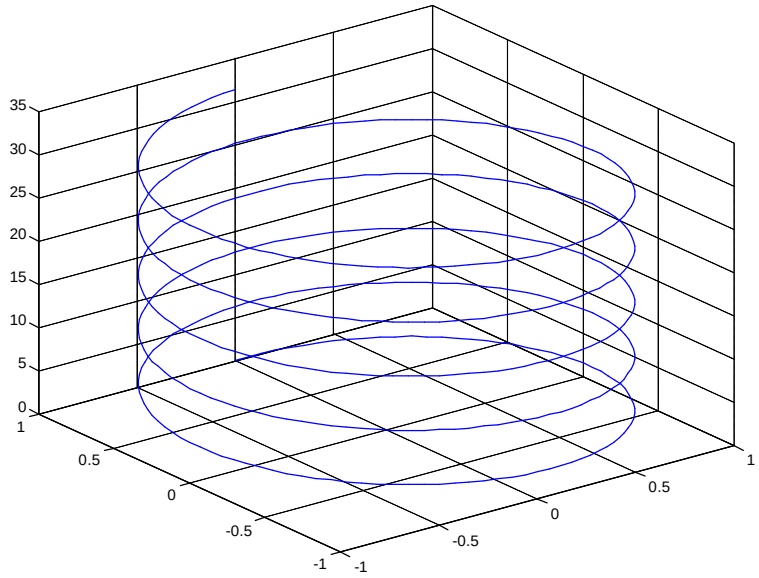
GRAFICACION

Graficar en Matlab es muy sencillo y presenta una variedad de alternativas. El comando que permite "mostrar" en una figura los elementos de una variable es *plot* (o su versión tridimensional *plot3*). Un ejemplo sencillo es:

```
t=0:0.25:7;
y = sin(t);
plot(t,y)
```



Un ejemplo de *plot3*:



Para ver las alternativas y posibilidades de graficación es recomendable recurrir al help o manual correspondiente. No obstante, es importante destacar la existencia (en esta última versión de Matlab) de un Editor de Propiedades Gráficas (ver en el menú de Archivo), que facilita la edición de figuras y el manejo de objetos gráficos. Por otra parte, las figuras pueden ser guardadas en archivos (ver el menú de Archivo de la figura) y llamadas posteriormente a los efectos de modificarlas. Otros comandos de interés vinculados a la visualización y edición de figuras son *zoom*, *plotedit* y *subplot*.

IMPRESION

Imprimir es muy sencillo, bastando con recurrir al Menú de Archivo para hacerlo. No obstante, resulta de utilidad en determinadas circunstancias el empleo del comando *print* (ver *help print*). Este permite salvar a archivo figuras para impresión en una diversidad de formatos.

AYUDA

Matlab tiene un muy buen *help on line* para cada comando. Este puede verse sobre el *workspace* ejecutando:

help `nombre comando`

Por otra parte, existe un completo menú de ayuda, incluyendo un índice con todos los comandos y ejemplos de simulación.