INTRODUCCIÓN A MATLAB

Moisés Cordeiro Costas

Índice

INTRODUCCIÓN A MATLAB. PRIMEROS PASOS

El uso de Help

Para obtener ayuda sobre cualquier comando de MATLAB se puede emplear:

>> *help orden* muestra en la pantalla información sobre el comando *orden*

>> *helpwin orden* hace lo mismo que el comando anterior, pero mostrando la información en una ventana que permite navegar para obtener información adicional.

Comandos básicos

>> *ver* muestra la versión, el código de licencia y los toolboxes disponibles

>> *who* (o >> *whos*) lista de todas las variables predefinidas (pueden verse también en el *workspace*)

>> *save archivo* guarda todas las variables definidas en la sesión de trabajo (guarda en binario en un archivo con extensión .mat)

>> *save archivo a b* guarda las variables a y b

>> *load archivo* carga las variables guardadas en el archivo

>> *clear x y* borra las variables x e y

>> *clear* (o >> *clear all*) borra todas las variables de la sesión de trabajo

>> *path* muestra el directorio donde se va a trabajar (también aparece en la ventana en el *current directory*)

>> *quit* salir de MATLAB

Recomendaciones básicas

- MATLAB distingue entre mayúsculas y minúsculas

- Trabajar siempre en un path conocido

- El nombre de una variable puede tener como mucho 31 caracteres

- El nombre de una variable debe comenzar obligatoriamente por una letra. Puede contener letras, números y el guion de subrayado; no se permiten espacios en blanco (escoger una metodología y emplear en todo el código)

- No es aconsejable nombrar variables mediante expresiones que tienen un significado específico en MATLAB.

- Hay algunas variables que, por defecto, tienen un valor asignado:

a) *pi* contiene el valor de π

b) *i* o *j* representan a la unidad imaginaria ($\sqrt{-1}$)

c) *inf* representa el valor infinito

d) *NaN* (Not a Number), representa una expresión indeterminada

Operaciones aritméticas con MATLAB

Los símbolos que utiliza MATLAB para realizar los cálculos aritméticos son los siguientes:

Suma	Resta	Multiplicación	División	Potenciación
+	-	*	/	^

El orden de prioridad de las operaciones es la misma que en las calculadoras. Se pueden emplear paréntesis para alterar esta ordenación.

Variables

Para almacenar los resultados de los cálculos, se emplean **variables**. MATLAB no requiere ni declarar ni dimensionar variables (aunque es recomendable)

- las variables se auto-declaran al inicializarlas

- la memoria se reasigna dinámicamente

Las variables pueden ser:

- *numéricas* aquellas que tienen un valor numérico asignado (x = 2)

- *simbólicas* NO tienen un valor numérico asignado. Para declararlas se emplea el comando *syms* (*syms x y*)

- *cadena de caracteres* (*cadena = 'hola')*

Para conocer el valor que tiene una variable basta con escribir su nombre en la ventana de comandos

Signos de puntuación, comentarios y movimiento del cursor

- En una misma línea se pueden definir varias variables separadas por (,) o por (;). La diferencia está en que (;) suprime la impresión por pantalla (eco)

- Empleando (…) se puede continuar una línea

- MATLAB ignora lo que se introduce a la derecha del signo %, lo cual permite introducir comentarios

- Movimientos del cursor: para recuperar comandos, MATLAB emplea las flechas del terclado

↑ recupera la línea previa ↓ recupera la línea siguiente

← mueve el cursor hacia la izquierda → mueve el cursor hacia la derecha

Nota: apretando la tecla esc se borra el comando escrito

Funciones predefinidas

MATLAB incorpora una serie de funciones intrínsecas que se corresponden con las funciones matemáticas más empleadas. Algunos ejemplos de ellas son:

- *abs(a)* calcula el valor absoluto de a

- *sqrt(a)* calcula la raíz cuadrada de a

- *sin(a)*, *cos(a)*, *tan(a)* calcula el seno, el coseno y la tangente de a, respectivamente

- *log(a)* calcula el logaritmo neperiano de a

- *exp(a)* calcula e^a

Formatos numéricos

Cuando el resultado de un cálculo es un número entero, MATLAB lo presenta en la pantalla como un entero simple que tenga menos de 10 cifras. Si tiene más de 10 cifras o es un número decimal, puede expresarlo en alguno de los formatos siguientes con las órdenes que se especifican:

>> *format short* 3 cifras parte entera y 4 decimales

>> *format short e* 1 cifra parte entera, 4 decimales y 3 en el exponente

>> *format long* 2 cifras parte entera y 14 decimales

>> *format long e* 1 cifra parte entera, 15 decimales y 3 en el exponente

>> *format rat* escribe en formato racional

Por defecto MATLAB emplea el formato “short”, que se puede cambiar con las órdenes señaladas previamente. En cualquier caso, el formato elegido solo afecta a la visualización en la pantalla, no a la precisión en los cálculos.

ESTRUCTURAS BÁSICAS DE DATOS

Vectores

Los vectores se introducen escribiendo, entre corchetes, cada una de sus componentes separadas por un espacio en blanco o una coma:

>> v = [0 1 -22 3 -5 8 pi] % vector fila

>> v = [0; 1; -22; 3; -5; 8; pi] % vector columna

Una vez definido un vector se puede acceder a sus componentes para conocer su valor, utilizarlo o modificarlo:

- *v(i)* devuelve la componente i-ésima del vector *v*. La primera componente siempre es 1

- *v(i1:h:i2)* devuelve las componentes de *v* desde *i1* hasta *i2* con incremento *h*

- *v([i,j,k])* devuelve las componentes *i, j, k* de *v*

MATLAB permite generar vectores sin necesidad de introducir explícitamente sus componentes, de diferentes maneras:

- *v = [x1:h:xn]* genera el vector (*x1, x1 + h, x1 +2h, …, x1 + kh*), donde *k* es el mayor entero tal que *x1 + kh ≤ xn*. En este caso, los corchetes son opcionales

- *v = [x1:xn]* hace lo mismo que *[x1:1:xn]*

- *v = linespace(a, b, n)* genera un vector de *n* componentes, espaciadas uniformemente comenzando en *a* y terminado en *b*.

Matrices

Se introducen por filas, separando cada fila de la siguiente por medio de un punto y coma, y los distintos elementos de una misma fila por espacios en blanco o comas (tamaño máximo 524228 elementos)

Una vez definida una matriz se puede acceder a sus elementos para conocer sus valores, utilizarlos o modificarlos:

- *A(i,j)* devuelve el elemento *(i,j)* de la matriz *A*

- *A(i1:h:i2, j1:k:j2)* devuelve la submatriz de *A* formada por las filas desde la *i1 hasta* i2 con incremento *h* y las columnas de la *j1* hasta la *j2* con incremento *h*

- *A(i,:)* devuelve el elemento *i* de todas las columnas, es decir, la fila *i*

- *A(i1:h:i2, :)* devuelve la submatriz de A formada por las filas desde la *i1* hasta la *i2* con incremento *h*

Inicialización de matrices

- *ones(n)* genera una matriz cuadrada de orden nxn formada por unos

- *ones(m,n)* genera una matriz mxn formada por unos

- *zeros(n)*, *zeros(m,n)* funciona al igual que las anteriores pero la matriz está formada por ceros

- *eye(n)* genera la matriz identidad nxn

- *eye(m,n)* genera una matriz de orden mxn con unos en la diagonal principal y ceros en el resto

- *rand(m,n)* genera una matriz de orden mxn con números aleatorios

Con *diag* podemos construir a partir de un vector una matriz diagonal

>> diag([1 2], 1)

El comando blkdiag permite construir matrices diagonales por bloques

>> blkdiag(1, [1 2; 3 4], 5)

Inicialización de matrices dispersas

Las matrices que tienen un gran número de elementos nulos se las conoce cómo matrices dispersas o huecas. Cuando necesitamos trabajar con este tipo de matrices, no es razonable introducir en el ordenador todas sus entradas, pues de esta manera no sólo nos veríamos obligados a escribir todos sus elementos, sino que, además, estaríamos desperdiciando una considerable cantidad de memoria.

Con el comando *a = sparse(i, j, c, m, n)* donde:

- *m* indica el número de filas de la matriz

- *n* indica el número de columnas de la matriz

- *c* es un vector que contiene a los elementos no nulos de la matriz

- *i, j* son los vectores de número enteros que indican la posición de cada elemento del vector *c*. A los vectores *i* y *j* se los conoce con el nombre de punteros y tienen la misma dimensión que el vector *c*, que coincide con el número de elementos de la matriz no nulos.

La función *sparse* también admite la forma simplificada

>> A = sparse(i, j, c)

en la que se toma como *m* el máximo valor de las componentes de *i* y como *n* el máximo entre las de *j*

Con el comando *s = sparse(m,n)*; se genera una matriz dispersa de tamaño nxn

Si X es una matriz, *s = sparse(X)* transforma la matriz en dispersa

- Las matrices dispersas ahorran memoria y los cálculos son más eficientes

- Todas las operaciones de matrices funcionan con matrices dispersas

- Una operación de matriz dispersa con 'llena' transforma la matriz en llena

- Si se vuelve ineficiente, MATLAB transforma la matriz sparse en llena

- Si se quiere solamente reservar espacio para una matriz dispersa sin inicializar se utiliza también el comando *spalloc*

>> A = spalloc(m,n, cont)

este comando genera con nombre *A* una matriz de tamaño mxn con una estimación de *cont* elementos distintos de cero

Si deseamos visualizar la matriz *A* llena, es decir, en la forma tradicional, podemos usar el comando *full*

>> full(A)

Para visualizar gráficamente con MATLAB los elementos no nulos de una matriz podemos utilizar el comando *spy(A)*. Este comando genera una figura en la que se marca con un punto cada elemento no nulo de la matriz

Operaciones con vectores y matrices

- *A + B* suma *A* y *B*

- *A – B* resta *A* y *B*

- *A*B* multiplica *A* por *B* (producto matricial)

- *λ*A* multiplica por *λ* todos los elementos de *A*

- *A^n* eleva la matriz *A* al número entero *n*

- *A.'* genera la traspuesta de *A*

- *A'* proporciona la conjugada transpuesta de *A*. Nótese que, si *A* es real, *A.'* y *A'* coinciden

- *λ + A* suma *λ* a cada elemento de la matriz *A*

- *A.*B* multiplica *A* por *B* elemento a elemento, es decir, define la matriz donde cada elemento (*i, j*) es *aij bij* (elemento a elemento, mismo tamaño)

- *A.^λ* eleva cada elemento de *A* a *λ*

- *A.^B* eleva cada elemento de *A* al correspondiente de *B*

Si *u* y *v* son dos vectores con el mismo número de coordenadas:

- *dot(u, v)* calcula el producto escalar

- *cross(u, v)* calcula el producto vectorial, es necesario que *u* y *v* tengan tres coordenadas

- *c = sum(v)* suma los elementos del vector

- *norm(v, p); norm(v)* calcula la norma de un vector *p = 1, 2, …, inf*. Por defecto *p = 2* (norma 2 → norma euclídea)

- *setdiff(u,v)* devuelve los valores de *u* que no están en *v*

- *bord = sort(v)*, *[bord, p] = sort(v)* ordena el vector. Si tiene un argumento de salida, devuelve el vector ordenado. Si tiene dos, devuelve el vector ordenado y la posición que ocupaban en el vector inicial los elementos ordenados

- *length(v)* devuelve la longitud del vector

- *diag(v)* construye una matriz con los términos de *v* en la diagonal principal

- *A([i, j], :) = A([j, i], :)* intercambia las filas *i* y *j*

Otras operaciones con matrices:

Si A es una matriz mxn

- *b = sum(A)* da como resultado un vector fila que es la suma de los elementos de cada columna

- *inv(A), det(A), rank(A)* inversa, determinante y rango de *A*

- *diag(A)* devuelve un vector con los términos diagonales de la matriz *A*

- *size(A)* devuelve el tamaño de la matriz *A* (número de filas y de columnas)

Con vectores o con matrices pueden realizarse las mismas operaciones elementales que son escalares

- Operaciones trigonométricas: *sin, cos, tan, asin, acos, atan…*

- Operaciones de redondeo *floor, ceil, round, fix*

>> *floor* redondea hacia *-inf*

>> *ceil* redondea hacia *+inf*

>> *round* redondea hacia el entero más próximo

>> *fix* redondea hacia 0

- Operaciones modulares: *rem, mod* devuelven el resto después de la división

- Operaciones exponenciales: *exp, log, log2, log10, sqrt*

Gráficos

Se pueden crear múltiples gráficos 2D y 3D. Habitualmente se suelen utilizar los gráficos 2D. La forma más simple es el comando *plot*. Para eso, debemos almacenar en un vector las abscisas y en otro las ordenadas.

Tenemos distintas posibilidades para la elección del tipo de rasgo y color

color	trazo
y → amarillo	. → puntos
m → magenta	o → círculos
r → rojo	x → signo x
g → verde	+ → signo +
b → azul	* → signo *
w → blanco	: → línea punteada
k → negro	-- → línea discontinua

Retoques gráficos

Permiten añadir al dibujo alguna información sobre la función que representa

- *xlabel('texto')* e *ylabel('texto')* incorporan texto al eje de abscisas y ordenadas respectivamente

- *title('texto')* pone el título 'texto' en la parte superior del dibujo

- *gtext('texto')* permite poner un texto en la ventana gráfica utilizando el puntero del ratón

- *grid* hace una cuadrícula

- *axis([x1 x2 y1 y2])* pinta en el rango [x1, x2]x[y1, y2]

El comando subplot

El comando *subplot(m, n, p)* divide la ventana gráfica en una matriz de mxn subventanas y coloca el gráfico activo en la subventana p-ésima, contando de izquierda a derecha y de arriba abajo

MATLAB COMO LENGUAJE DE PROGRAMACIÓN

Apertura y cierre de ficheros

>> *fid = fopen('fichero')* Abre el fichero ya existente. Devuelve un número entero no negativo que identifica al fichero abierto. Si el identificador toma el valor -1 significa que el fichero no se ha abierto correctamente

>> *fid = fopen('fichero', 'permiso')* Abre el fichero con los siguientes permisos:

- r → Lectura (read). Se aplica a un fichero ya existente (por defecto)

- r+ → Lectura y escritura. Se aplica a un fichero ya existente

- w → Escritura (w). Si no existe un fichero con el nombre indicado se crea; si ya existe, se borra y lo abre vacío

- w+ → Lectura y escritura. Si no existe un fichero con el nombre indicado, se crea; si ya existe, se borra y lo abre vacío

- a → Escritura (add). Si no existe un fichero con el nombre indicado se crea, si ya existe, añade información

- a+ → Lectura y escritura. Si no existe un fichero con el nombre indicado, se crea; si ya existe, añade información

>> *frewind(fid)* revovina un fichero con identificador fid

>> *fclose(fid)* cierra el fichero identificado con fid; devuelve un 0 si el proceso fue realizado de manera exitosa y -1, si no

Lectura/introducción de datos

Se realiza de dos maneras distintas, por teclado (input) o por lectura de ficheros con formato (fscanf)

1. *input* permite introducir un dato por teclado

2. *fscanf* permite leer datos con formato de un fichero abierto con *fopen*

- *[A, count] = fscanf(fid, 'formato')* lee los datos del fichero abierto con identificador fid y los guarda en la matriz A (en columna) con el formato especificado en formato. El número de elementos leídos queda registrado en count

- *[A, count] = fscanf(fid, 'formato', tamaño el)* lee los datos especificados en el parámetro opcional tamaño el del fichero abierto con identificador fid y los guarda en la matriz A (en columna) con el formato especificado en formato. El número de elementos leídos queda registrado en count. Si no se especifica el tamaño el A es un vector columna.

El argumento formato tanto de las órdenes de escritura como lectura consta de una cadena formada por caracteres de escape (precedidos por \) y por caracteres de conversión (precedidos del carácter %)

1. Caracteres de escape

- *n* paso a una nueva línea

- *t* tabulación horizontal

- *b* vuelve hacia atrás un solo carácter (backspace)

2. Caracteres de conversión

- *%d* enteros en sistema decimal

- *%f* reales de punto fijo

- *%e* reales de punto flotante

- *%s* cadena de caracteres

Para trabajar con enteros se utilizan caracteres de conversión de la forma %md, donde m es el número de cifras del entero. Como por ejemplo %5d indica a un entero decimal con cinco cifras

Para trabajar con números reales se utilizan los caracteres de conversión de la manera %m.n*, donde m es el número de espacios ocupados por el número real, incluidos el punto decimal y el signo si los hubiera y n es el número de decimales de dicho número y * es uno de los caracteres de conversión f o e. Como por ejemplo: %9.3f permite representar un número real que tiene 9 cifras en total: 3 cifras decimales, el punto decimal y 5 cifras enteras

Para trabajar con cadenas de caracteres utilizan los caracteres de conversión de la forma %ms, donde m es el número total de caracteres de la cadena que se desea representar.

Escritura de datos

La escritura de datos se realiza de dos maneras distintas, en pantalla o en ficheros con formato (disp, fprintf)

1. disp permite escribir en pantalla mensajes alfanuméricos

- *disp('cadena de caracteres')* muestra por pantalla el mensaje cadena de caracteres

- *disp(x)* muestra por pantalla el valor de la variable x

2. fprintf permite escribir datos con formato especificado en un fichero previamente abierto con *fopen*

- *fprintf(fid, 'formato', A, …)* escribe los elementos de la matriz A en un fichero abierto con identificador fid con el formato especificado en formato

- *fprintf('formato', A, …)* lo mismo que el anterior comando pero por pantalla

Operadores racionales y lógicos

En MATLAB se utilizan los operadores relacionadores:

Operador	<	<=	>	>=	==	~=
Descripción	Menor	Menor o igual	Mayor	Mayor o igual	Igual	distinto

Los operadores racionales permiten construir expresiones lógicas de la siguiente manera

expresión1 OpR expresión2

Donde Por es un operador racional y expresion1 y expresion2 son números, matrices (de igual dimensión) o cadenas de caracteres. Las expresiones lógicas toman el valor 1 cuando son verdaderas y 0 cuando son falsas

Operadores lógicos

Permiten construir una expresión lógica a partir de otras dadas

Operador	&	|	~
Designación	Conjunción	Disyunción	Negación

A partir de dos expresiones lógicas p y q, los operadores lógicos &, | permiten definir las expresiones lógicas

p & q, p | q

Prioridad de operadores

En una misma expresión pueden aparecer los tres tipos de operadores estudiados: aritméticos, relacionales y lógicos. El orden de prioridad de estos, es decir, el orden en el que se ejecutan las operaciones viene dado por la siguiente tabla leída de arriba hacia abajo:

Prioridad de operadores en MATLAB
^ .^
* / \ .* ./
+ - ~
> < >= <= == ~=
& | xor

Estas órdenes de prioridad pueden modificarse, y es aconsejable hacerlo para mayor claridad, utilizando paréntesis

Control de flujo

Un programa es un conjunto de órdenes relacionadas entre sí. En MATLAB las órdenes que forman un programa suelen escribirse en un fichero .m

Para crear programas complejos son necesarias estructuras de control de flujo:

- Estructuras repetitivas: Permiten que la ejecución de un conjunto de órdenes pueda repetirse varias veces. En MATLAB se consiguen con los bucles *for-end* y *while-end*

- Estructuras alternativas: Hacen que se ejecute un conjunto u otro de órdenes, según se verifique o no cierta condición. En MATLAB se consigue con instrucciones *if-elseif-else-end*

Bucles for-end

Hacen que un grupo de órdenes pueda repetirse un número determinado de veces. Su sintaxis es la siguiente:

for i = x

conjunto de órdenes

end

donde *i* es una variable contador y *x* un vector

Al llegar al comando *for* la variable *i* toma como valor la primera coordenada del vector *x* y ejecuta el *conjunto de órdenes*. A continuación, *i* toma como valor a la segunda coordenada de *x*y vuelve a ejecutar el *conjunto de órdenes*. El bucle se repite tantas veces como coordenadas tiene el vector *x*

Observación: Para facilitar la lectura de los programas conviene identar las líneas de manera que la escritura refleje la estructura del programa

Observación: Dentro de un *conjunto de órdenes* que se efectúan en un bucle *for* puede haber otro/s bucle/s *for*, y, en tal caso, se dice que los bucles están anidados

Bucles while-end

Sintaxis:

while expresión lógica

    conjunto de órdenes

end

Hacen que un *conjunto de órdenes* se ejecute mientras una *expresión lógica* sea verdadera

Lo que distingue a los bucles *for* de los bucles *while* es que en los primeros el *conjunto de órdenes* se ejecuta un número prefijado de veces, mientras que en los segundos, el *conjunto de órdenes* se ejecuta un número de veces desconocido a priori.

Estructuras if-elseif-else-end

En ocasiones se quiere ejecutar un conjunto de órdenes sólo en caso de que se verifique cierta condición. La estructura es la siguiente (no tienen que darse todos los pasos siempre):

if expresión lógica

    conjunto de órdenes

elseif expresión lógica

    conjunto de órdenes

else

    conjunto de órdenes

end

Otras órdenes

Otras órdenes interesantes para construir programas son las siguientes:

- *keyword* transfiere el control al teclado hasta que se escribe return (tecla enter)

- *return* cuando se utiliza en combinación con *keyboard*, devuelve el control al programa. Dentro de un programa interrumpe su ejecución

- *break* interrumpe la ejecución de un bucle. Si hay varios bucles aninados, la orden *break* interrumpe el buble más interno que la contiene. Suele usarse dentro de un condicional *if-end*

- *error('mensaje')* muestra por pantalla el mensaje y finaliza la ejecución del fichero .m Suele usarse dentro de un condicional *if-end* para mostrar mensajes de error

- *tic* ponen a cero el reloj MATLAB

- *toc* muestra el tiempo de reloj

Ficheros .m de función

MATLAB dispone de un gran número de funciones predefinidas que pueden ser llamadas escribiendo su nombre (sin, lu, eigs…) El usuario tiene la posibilidad de definir otras utilizando ficheros .m especiales, a los que denominamos ficheros m de función

Para definir una función se escribe un fichero .m en el que el nombre de este tiene que COINCIDIR con el de la función).

function [variables de salida] = nombre de la función (variables de entrada)