Ecuaciones en Derivadas Parciales y Análisis Numérico

Prácticas.

Capítulo 1. Matrices en Matlab

1.1 El entorno de trabajo

El programa Matlab ofrece un entorno interactivo donde podemos ejecutar comandos y examinar los resultados sobre la marcha, de modo distinto a un lenguaje de programación como C que nos obliga a escribir todo el programa, compilarlo y ejecutarlo. En otras palabras, *Matlab es interactivo*. Ésto lo hace más apropiado para aprender o para explorar un problema desconocido.

Además, ofrece algunas facilidades como una vista de las variables y sus valores actuales. Se puede activar esta vista seleccionando el menú 'Window/Workspace' (probablemente ya esté activa). A continuación, escribimos un comando de asignación de una variable:

```
theta=pi/2
```

Observamos que en la vista del workspace aparece una variable theta, que contiene el valor que acabamos de introducir. Puedes eliminar la variable ejecutando:

```
clear theta
```

o seleccionando las variables que quieres eliminar en el workspace, seguido de click derecho y *eliminar*. Otra utilidad que te resultará muy útil es la ayuda del programa, que puedes abrir en una ventana separada activando el menú “Ayuda de Matlab”, o en la misma terminal, escribiendo la palabra clave **`help`** seguida del nombre del comando que quieres aprender a usar:

```
help plot      %ayuda en la terminal (los comentarios comienzan por %)
doc plot       %ayuda en ventana separada
```

1.2 Matrices

En Matlab, el valor de una variable, pero también el tipo de entidad que representa, puede cambiar a lo largo de una sesión de Matlab o a lo largo de la ejecución de un programa. La forma más normal de cambiar el valor de una variable es colocándola a la izquierda del operador de asignación (=), es decir, *variable = expresión.* También podemos escribir la expresión sin más en la línea de comandos si no queremos guardar el resultado. Por defecto, después de ejecutar cualquier comando, Matlab nos muestra el resultado por pantalla. Si no queremos que lo haga, hay que poner un punto y coma (**;**) al final de la expresión.

El tipo de datos más usado en Matlab (prácticamente el único que vamos a usar en esta asignatura), es la matriz de datos numéricos de coma flotante. Una constante numérica es de hecho una matriz 1x1 y un vector de datos numéricos de longitud N es una matriz de dimensiones **`Nx1`** (vector fila) ó **`1xN`** (vector columna). Aunque en muchos sitios se puede usar indistintamente un vector fila o columna, son objetos distintos, y deberemos tener mucho cuidado en no confundirlos.

Para empezar, veamos las dos sintaxis más habituales para introducir de forma manual una matriz A de dimensión (**3x3**):

```
A=[1 2 3; 4 5 6; 7 8 10]
A=[1, 2, 3; 4, 5, 6; 7, 8, 9]
```

Para leer y cambiar el valor de una entrada de la matriz, escribimos entre paréntesis los números de fila y de columna:

```
A(2,3)          %devuelve 6
A(2,3)=0
```

Creamos otra matriz **B** igual a la traspuesta conjugada de **A**:

```
B=A'
```

Ahora ya están definidas las matrices **A** y **B**, y es posible seguir operando con ellas. Por ejemplo, hacemos el producto de **B** por **A**:

```
B*A
```

Del mismo modo, es posible definir un vector fila **X** o un vector columna **Y** en la forma siguiente:

```
X=[10 20 30]              % vector fila
Y=[11; 12; 13]            % vector columna
```

En ambos casos es posible acceder a los elementos de un vector poniendo entre paréntesis un sólo número:

```
X(1)+Y(2)                 %devuelve 22
```

Sin embargo, su comportamiento con respecto a la suma o producto de matrices es muy distinto. Podemos percibir la diferencia entre uno y otro al pedir sus dimensiones como matrices:

```
size(X)
size(Y)
X+Y             %error
X+Y'            %ok
```

En muchos cálculos matriciales los datos y/ó los resultados no son reales sino complejos. Matlab trabaja sin ninguna dificultad con números complejos. Para ver como se representan por defecto los números complejos, ejecutamos los siguientes comandos:

```
a=sqrt(-4)
a =
    0 + 2.0000i

3 + 4j
ans =
    3.0000 + 4.0000i
```

El número imaginario se puede representar tanto con la **i** como con la **j**.

Operaciones

Los operadores matriciales de Matlab son los siguientes:

```
A'        %matriz traspuesta (en realidad, traspuesta conjugada: A.' es
          %la traspuesta sin conjugar)
A^k       %potencia
inv(A)    %matriz inversa
A+B       %adición o suma
A-B       %sustracción o resta
A*B       %multiplicación
A\B       %división a la izquierda (igual a inv(A)*B)
A/B       %división a la derecha (igual a A*inv(B))
A\Y       %división a la izquierda (igual a inv(A)*Y, pero más eficiente)

length(X)      %longitud de un vector
sum(X)         %suma de los elementos de un vector
norm(X)        %norma euclidea del vector

size(A)        %dimensiones de una matriz
eye(n)         %matriz identidad nxn
zeros(n,m)     %matriz de dimensiones nxm rellena con ceros
ones(n,m)      %matriz de dimensiones nxm rellena con unos
diag(X)        %matriz con ceros fuera de la diagonal y los elementos del
               %vector X en la diagonal principal
diag(X,k)      %matriz con los elementos del vector X en una diagonal
               %distinta de la diagonal principal (k suele ser 1 o -1)
```

En general, para poder sumar o restar dos matrices, sus dimensiones deben ser iguales. Para poder aplicar una multiplicación o división a dos matrices cuadradas, las dimensiones deben casar según las reglas de álgebra lineal. Sin embargo, siempre se pueden efectuar estas operaciones si una de las matrices es un escalar, es cuyo caso la operación se realiza elemento a elemento. Comprobamos lo anterior con la matriz **A** 3x3 y el vector fila **X** 1x3 definidos antes:

```
A+X       %error
A-B       %ok
X*A       %ok
A*X       %error
3*A       %ok
A/3       %ok
3/A       %error
```

Siempre existe también la posibilidad de aplicar elemento a elemento los operadores ***, ^, /** y ****. Para ello basta poner un punto (**.**) delante del operador. Observa la diferencia entre los dos comandos siguientes:

```
A*B       %producto de matrices
A.*B      %producto elemento a elemento (si tienen las mismas dimensiones)
```

O también:

```
[1 2 3 4]^2    %error
[1 2 3 4].^2   %devuelve [1 4 9 16]
```

Una función escalar (como **sin**, o **exp**) puede actuar sobre matrices, y lo hace elemento a elemento. Ej:

```
sin([0 pi/3 pi/2 pi])
```

Usando el operador dos puntos (**:**) podemos crear vectores fila con valores consecutivos, o espaciados una cantidad fija:

```
x=1:10
x=1:1.5:10
```

```
x=12:-1:10
```

Un ejemplo típico de todo lo anterior:

```
x=[0:pi/50:8*pi]';
y=sin(x);
z=cos(x);
plot(x,y);     %el comando plot dibuja grafos de funciones y trayectorias
               %en el plano
plot(y,z);
plot3(x,y,z); %el comando plot3 dibuja trayectorias en el espacio
               %puedes probar otros tipos de gráficas haciendo click
               %derecho sobre una variable del workspace
```

Submatrices

Vamos a estudiar cómo extraer elementos y rangos de elementos de entre los elementos de una matriz. Construyamos una matriz que contiene un cuadrado "mágico" de orden 6:

```
M=magic(6)
```

Utilizando los dos puntos (**:**) podemos seleccionar los elementos de una submatriz de **M**:

```
M(6, 1:4)      %leer los cuatro primeros elementos de la sexta fila
M(6, 3:end)    %leer los elementos de la sexta fila a partir del tercero
M(6, :)        %leer todos los elementos de la sexta fila
```

Calculamos las sumas de los elementos de la segunda fila y la tercera columna:

```
sum(M(2,:))
sum(M(:,3))
```

Es posible asignar un rango entero de valores de M con un sólo comando (las dimensiones a ambos lados deben coincidir). Utilizamos el comando **ones** para conseguir poner doses en las filas 1 y 2 de la matriz M.

```
M(1:2,:)=2*ones(2,6)
```

Como vemos hay muchas formas de definir matrices. También podemos combinar la notación de corchetes con los métodos anteriores.

```
D=[X, 2*X, 3*X]
E=[X; 2*X; 3*X]
F=[zeros(6,1) M]
G=[zeros(1,6); M]
```

1.3 Sentencias de control

Sentencia if

En su forma más simple, la sentencia **if** se escribe en la forma siguiente (la indentación es opcional pero muy recomendable):

```
if condición
```

```
    sentencias
end
```

Existe también la bifurcación múltiple, que tiene la forma:

```
if condición1
    bloque1
elseif condición2
    bloque2
else % opción por defecto
    bloque3
end
```

Sentencia for

La sentencia **for** repite un bucle un número predeterminado de veces. La sentencia **for** de Matlab es muy diferente y no tiene la generalidad de la sentencia **for** de C. La siguiente construcción ejecuta sentencias con valores de **k** desde **1** hasta **n** (una variable definida con anterioridad), variando de uno en uno (la indentación es opcional pero muy recomendable).

```
for k=1:n
    sentencias
end
```

En el siguiente ejemplo el bucle se ejecuta por primera vez con **k=n**, y luego **k** va disminuyendo cada vez **0.2** hasta ser meno que **1**, en cuyo caso el bucle se termina:

```
for k=n:-0.2:1
    sentencias
end
```

En el siguiente ejemplo se presenta una estructura correspondiente a dos bucles anidados. La variable **l** es la que variará más rápidamente (por cada valor de **k**, **l** toma todos sus posibles valores):

```
for k=1:m
    for l=1:n
         sentencias
    end
end
```

Sentencia while

La estructura del bucle **while** es muy similar a la de C. Su sintaxis es la siguiente

```
while condición
    sentencias
end
```

Aquí, condición puede ser una expresión vectorial o matricial. Las sentencias se siguen ejecutando mientras haya elementos distintos de cero en condición, es decir, mientras haya algún o algunos elementos ciertos. El bucle se termina cuando todos los elementos de condición son falsos (es decir, cero). Ejemplo:

```
epsilon = 1;
while (1+epsilon) > 1
```

```
    epsilon = epsilon/2;
end
epsilon        %problema: ?que representa epsilon?
```

1.4 Funciones matemáticas elementales que operan de modo escalar

Estas funciones, que comprenden las funciones matemáticas trascendentales y otras funciones básicas, actúan sobre cada elemento de la matriz como si se tratase de un escalar. Se aplican de la misma forma a escalares, vectores y matrices. Algunas de las funciones de este grupo son las siguientes:

```
sin(x)         %seno
cos(x)         %coseno
tan(x)         %tangente

asin(x)        %arco seno
acos(x)        %arco coseno
atan(x)        %arco tangente (devuelve un ángulo entre -pi/2 y +pi/2)
atan2(s,c)     %arco tangente (devuelve un ángulo entre -pi y +pi); se
               %le pasan 2 argumentos, proporcionales al seno y al coseno

sinh(x)        %seno hiperbólico
cosh(x)        %coseno hiperbólico
tanh(x)        %tangente hiperbólica
asinh(x)       %arco seno hiperbólico
acosh(x)       %arco coseno hiperbólico
atanh(x)       %arco tangente hiperbólica

log(x)         %logaritmo natural
log10(x)       %logaritmo decimal
exp(x)         %función exponencial
power(x,y)     %x elevado a la potencia y
sqrt(x)        %raíz cuadrada

rem(x)         %resto de la división entera (2 argumentos, que no tienen
               %que ser enteros)
round(x)       %redondeo hacia el entero más próximo
fix(x)         %redondea hacia el entero más próximo a 0

sign(x)        %devuelve -1 si x < 0, 0 si x = 0 y 1 si x > 0. Aplicada a
               %un número complejo, devuelve un vector unitario en la misma
               %dirección
real(x)        %partes reales
imag(x)        %partes imaginarias
abs(x)         %valores absolutos
angle(x)       %ángulos de fase
```