UNA INTRODUCCIÓN A MATLAB

Este apunte corresponde al un curso de grado sobre la utilización de Matlab, principalmente orientado a la aplicación de este sistema en la resolución de ecuaciones diferenciales. Está basado fundamentalmente en el curso que dictaron el Dr. Fernando Basombrío y el Dr. Mario Storti durante la primera mitad de 1997 en el IB. Otra fuente fundamental para estos apuntes es el manual de Octave escrito por su creador: John W. Eaton (email: jwe@bewo.che.wsic.edu). La utilidad de estas notas reside en que resumen la profundidad del curso dictado por Basombrío y Storti, sin llegar a ser un manual de referencia como el de Eaton, tornándolas aptas para quienes se introducen en la utilización del lenguaje de Matlab u Octave.

Pablo A. Ramírez

CONTENIDO

1.

1. Introducción.

Matlab es un lenguaje de alto nivel, diseñado para hacer cálculos numéricos. Provee una interfase con línea de comando para resolver problemas lineales y no lineales y otros experimentos en forma numérica. Puede ser usado también en forma de procesamiento tipo *batch.*

En la mayoría de los sistemas, la forma de invocar a Matlab es con el comando del shell 'matlab'. Matlab muestra un mensaje inicial indicando que está listo para aceptar instrucciones y haciendo algunas sugerencias de comandos para iniciar. A partir de allí se pueden escribir comandos inmediatamente.

```
Commands to get started: intro, demo, help help

Commands for more information: help, whatsnew, info, subscribe

»
```

Si ocurre algún tipo de problema, se puede interrumpir la tarea que está realizando Matlab con 'Control – C' (usualmente escrito como 'C – c' para abreviar). Para salir de Matlab simplemente se debe escribir 'exit' en el prompt. En aquellos sistemas que soportan control de tareas (como Linux y la mayoría de los sistemas Unix, VMS, etc...) se puede suspender Matlab (u Octave según sea el caso) enviando una señal 'SIGSTP' (usualmente 'C–z').

Los siguientes capítulos describen solo una parte de las funcionalidades de Matlab, aunque resultan ser una guía muy útil para dar una muestra de sus posibilidades.

Si se es nuevo en el uso de Matlab, es recomendable que uno trate de reproducir los ejemplos que se muestran. Las líneas marcadas como '>>' son líneas que se deben escribir terminándolas con un retorno de carro (tecla 'enter' de la PC). Matlab responderá con un resultado o un gráfico.

1.1 Cómo crear una matriz.

Para crear una matriz y guardarla en una variable de manera que se pueda hacer referencia a ella más tarde, basta con escribir:

```
» a = [ 1, 1, 2; 3, 5, 8; 13, 21, 34]

a =

    1     1     2
    3     5     8
   13    21    34

»
```

Matlab responde imprimiendo (en la pantalla) la matriz en columnas alineadas. Terminar un comando con punto y coma indica a Matlab que no muestre el resultado. Por ejemplo:

```
» b = rand(3, 2);

»
```

creará una matriz de 3 filas y 2 columnas con cada elemento puesto a un valor aleatorio ("random") entre cero y uno.

Para mostrar el valor de una variable, simplemente se debe escribir el nombre de la variable. Por ejemplo, para mostrar el valor guardado en la matriz 'b', se debe tipear el comando:

```
» b

b =

    0.5194    0.0535
```

```
    0.8310    0.5297
    0.0346    0.6711
»
```

1.2 Aritmética con matrices.

Matlab posee una notación especial para efectuar aritmética matricial. Por ejemplo, para multiplicar la matriz 'a' por un escalar:

```
» 2 * a
ans =
     2     2     4
     6    10    16
    26    42    68
»
```

Para multiplicar dos matrices 'a' y 'b' , se debe escribir el comando:

```
» a*b
ans =
    1.4195    1.9255
    5.9897    8.1781
   25.3781   34.6378
»
```

'ans =' indica "answer" (respuesta). Para formar el producto matricial 'transpuesta (a) * a', escribir el comando:

```
» a' * a
ans =
       179       289       468
       289       467       756
       468       756      1224
»
```

1.3 Resolución de sistemas de ecuaciones lineales.

Para resolver el sistema de ecuaciones lineales 'ax = b', es conveniente usar el operador de división a la izquierda ' \ ':

```
» a \ b
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.005637e-018
ans =
  1.0e+015 *
    4.7955    1.8900
    4.7955    1.8900
```

```
  -4.7955   -1.8900
»
```

Esto ese conceptualmente equivalente a 'inv (a) * b' , pero evita el cálculo de la inversa de la matriz directamente.

Si la matriz de coeficientes es singular, Matlab emitirá un mensaje de advertencia y calculará una solución en el sentido de norma mínima.

1.4 Integrando ecuaciones diferenciales

Matlab tiene funciones para resolver ecuaciones diferenciales no lineales de la forma:

$$\frac{dx}{dt} = f(x,t)$$

con la condición inicial:

$$x(t = t_0) = x_0$$

Para que Matlab integre ecuaciones de esta forma, se debe escribir primero una función' $f(x,t)$'. Esto puede ser hecho directamente escribiendo una secuencia de comandos de Matlab en un archivo cuyo nombre define el nombre de una nueva función, necesariamente la extensión de estos archivos será '.m'. Por ejemplo, los comandos siguientes definen el miembro derecho de un sistema de dos ecuaciones diferenciales no lineales de sumo interés. Para escribir la nueva función se puede utilizar cualquier editor, por ejemplo, la siguiente función fue editada con el "*Bloc de Notas*" que puede encontrarse en cualquier sistema operativo Windows (puaj!):

```
function xdot = f(t, x)

r = 0.25;
k = 1.4;
a = 1.5;
b = 0.16;
c = 0.9;
d = 0.8;
xdot = [0; 0];

xdot(1) = r*x(1) * (1-x(1)/k) - a*x(1) * x(2) / (1 + b*x(1));
xdot(2) = c*a*x(1)*x(2)/(1 +  b*x(1)) - d*x(2);

end
```

dada la condición inicial

```
» x0 = [1; 2];
```

es fácil integrar el sistema usando la función ode23:

```
» [T,X] = ode23('f',0, 200, x0);
```

La función ode23 utiliza adaptivamente los esquemas de Runge – Kutta de orden 2 y 3 respectivamente.

1.5 Obtención de salidas gráficas.

Para mostrar la solución del ejemplo previo gráficamente, use el comando:

```
» plot (t, x)
```

Si se usa el sistema X Window, Matlab creará automáticamente una ventana separada para mostrar el gráfico.

1.6 Recuperación de comandos.

En el prompt de Matlab se puede recuperar, editar y reemitir comandos previos usando comandos al estilo de Emacs o Vi.

1.7 Usando la ayuda.

Matlab tiene facilidades de help abundantes. La misma documentación está disponible en forma impresa y también está disponible en forma interactiva, ya que ambas formas de documentación han sido creadas a partir del mismo archivo.

Para obtener ayuda, se debe conocer primero el nombre del comando que se quiere usar, lo cual no es siempre obvio. Un buen lugar para empezar es tipear directamente 'help'. Esto mostrará todos los operadores, palabras reservadas, funciones, variables internas y funciones de archivo. Se puede obtener más ayuda sobre cualquiera de los ítems listados incluyendo simplemente el nombre del ítem como argumento. Por ejemplo:

```
» help plot
```

mostrará el texto de ayuda para la función 'plot'.

2. Tipos de datos.

Todas las versiones de Matlab incluyen un cierto número de tipos de datos internos incluyendo matrices y escalares complejos y reales, y un tipo de dato llamado "data structure" (como el "record" de Fortran'90 o el "struct" de C. Es también posible definir tipos de datos especializados escribiendo las rutinas apropiadas en C++ (esta es una ventaja que tiene Octave al ser "free-software" es decir que se tiene acceso a las fuentes). En algunos sistemas no es necesario recompilar Octave si es posible hacer uso de la opción de linkedición dinámica de vínculos.

2.1 Objetos Numéricos.

Los objetos numéricos internos de Matlab incluyen escalares y matrices reales y complejos. Todos los objetos numéricos son representados internamente como números de punto flotante de doble precisión (¡¡incluso los enteros!!).

Las matrices pueden ser de cualquier orden y pueden ser cambiadas de tamaño y forma dinámicamente. Es muy simple extraer filas individuales, columnas y submatrices usando un sistema muy poderoso de indexación. (Esto se verá más adelante).

2.2 Cadenas de caracteres (strings).

Una cadena de caracteres en Matlab puede ser entrada delimitando la cadena con apóstrofes:

```
» a='parrot'
a =
parrot
```

```
» b='parrot'
b =
parrot
»
```

En Octave puede utilizarse indistintamente comillas o apóstrofes. La notación con apóstrofes es compatible con Matlab, pero se presta a confusión con el operador de transposición de matrices. Por eso es preferible usar comillas (esto NO es compatible con Matlab).

Internamente, Matlab guarda las cadenas como matrices cuyos elementos son caracteres. Todas las operaciones de indexación que funcionan con matrices pueden ser usadas con las cadenas para indicar subcadenas.

En Octave los caracteres especiales que no puede ser introducidos directamente (por ejemplo las comillas) debe se “escapeados” (“escaped”).

2.3 Estructuras de datos.

Esta es una opción que (a nuestro conocimiento) no existe en Matlab pero si en Octave. Permite agrupar en un objeto a objetos de diferente tipo y en forma recursiva:

```
octave> x.a = 1;
octave> x.b = [1, 2; 3, 4];
octave> x.c = “string”;
octave> x.d.a = “first d part”;
octave> x.d.b = rand(2);
octave> x
x =
{
        a = 1
        b =
                1   2
                3   4
        c = string
        d =
                {
                        a = first d part
                        b =
                                0.34585   0.68540
                                0.47757   0.58387
                }
}
octave> x.d.b(2,2)
ans = 0.58387
octave>
```

3. Tipos de datos numéricos.

Estos son algunos ejemplos de constantes numéricas con valores reales. todas tienen el mismo valor:

105

1.05e+2

1050e-1

Para valores complejos:

3 + 4i

3.0 + 4.0i

0.3e1 + 40e-1i

'i' denota la unidad imaginaria.

3.1 Matrices.

Las matrices son entradas entre corchetes por filas. Los elementos en la misma fila va separados por comas o simplemente por espacios y las filas van separadas por punto y coma.

```
» a = [1, 2; 3, 4]
a =
    1    2
    3    4
»
```

Los elementos que intervienen en una matriz pueden ser expresiones arbitrarias mientras que las dimensiones tengan sentido cuando se combinan entre si:

```
» a = [1, 2; 3, 4]
a =
    1    2
    3    4
» [ a, a]
ans =
    1    2    1    2
    3    4    3    4
» [a; a]
ans =
    1    2
    3    4
    1    2
    3    4
» [a; [1 1]]
ans =
```

```
     1     2
     3     4
     1     1
» [a, [1 1]]
??? All matrices on a row in the bracketed expression must have the
 same number of rows.
»
```

3.2 Tamaño de los objetos.

Las 'length()' y 'size()' permiten inspeccionar las diferentes dimensiones de las matrices. La función 'size()' retorna un vector con el número de filas y columnas. La función 'length()' retorna el máximo de ambas dimensiones: su uso está orientado a vectores ya sean fila o columnas.

```
» a = rand(3,5)
a =
    0.4175    0.9304    0.0920    0.7012    0.2625
    0.6868    0.8462    0.6539    0.9103    0.0475
    0.5890    0.5269    0.4160    0.7622    0.7361
» size(a,1)
ans =
     3
» size(a,2)
ans =
     5
» size(a)
ans =
     3     5
» b = rand(5,1)
b =
    0.3282
    0.6326
    0.7564
    0.9910
    0.3653
» length(b)
ans =
     5
» length(a)
ans =
```

```
    5
» length(b')
ans =
    5
» size(b,1)
ans =
    5
» size(b,2)
ans =
    1
»
```

En Octave, además existen funciones básicas que retornan valores Booleanos (1 = true, 0 = false): ‘is_empty(a)’, ‘is_scalar(a)’, ‘is_square(a)’, ‘is_symetric(a, tol)’.

3.3 Rangos.

Un “rango” es una forma conveniente de construir vectores con elementos espaciados uniformemente. El rango está definido por el valor inicial, incremento (opcional) y valor final separados por el operador ‘:’. Si el incremento no está se asume por defecto igual a 1.

```
» a = 1:0.5:3
a =
   1.0000   1.5000   2.0000   2.5000   3.0000
» a = 5:10
a =
    5    6    7    8    9   10
»
```

Los rangos son muy utilizados para controlar los valores que toman los índices en los lazos. Sin embargo el rango no es convertido explícitamente a vector hasta que esto no es necesario para ahorrar memoria. Por ejemplo, si queremos hacer un lazo (esto se va a ver después) de 1 a 1000000 y lo ponemos de la forma 1:1000000, esto generaría en principio un vector de longitud 1000000 de reales doble precisión, lo cual consume 8 Mb de memoria RAM.

Otro punto a tener en cuenta en cuanto a los rangos es que (debido a errores de redondeo) el punto final puede no estar dentro del vector generado. en el caso de que esto sea absolutamente necesario debe usarse en su lugar la función ‘linspace()’.

4. Expresiones.

Las expresiones son la unidad básica con la cual se arma las sentencias en Matlab. Una expresión da un valor al ser evaluada, el cual se puede imprimir, validar (en el sentido lógico), guardar en una variable, pasar a una función, o asignar su valor a una variable con un operador de asignación.

4.1 Expresiones de índices.

Una expresión indicial permite referenciar o extraer parte de los elementos de una matriz o vector. Dada la matriz:

```
» a = [1, 2; 3, 4]
a =
    1    2
    3    4
» a(1,[1 2])
ans =
    1    2
»
```

En general ‘a(i1,i2)’ retorna los valores de la submatriz de ‘a‘ conteniendo las filas cuyos índices están en ‘i1’ y columnas en ‘i2’ en el ejemplo previo podemos reemplazar ‘a(1, [1 2])’ por ‘a(1, 1:2)’. Una forma equivalente muy útil y compacta es ‘a(1, :)’. En esta expresión ‘:’ quiere decir: *todos los valores que toma el índice correspondiente*. Como el ‘:’ está en el índice de filas y la matriz tiene dos columnas ‘:’ es equivalente a ‘[1 2]’.

Las filas o columnas aparecen en el orden en que aparecen sus índices en ’i1’ o ‘y2’. Por ejemplo:

```
» a(1 , [2 1])
ans =
    2    1
»
```

De esta forma se puede extender vectores a matrices. Por ejemplo, sea ‘a‘ un vector columna, entonces una forma de obtener una matriz que contenga al vector ‘a‘ repetido tres veces es el llamado “truco de Tom”

```
» a = rand(3,1)
a =
   0.2470
   0.9826
   0.7227
» a(: , [1 1 1])
ans =
   0.2470   0.2470   0.2470
   0.9826   0.9826   0.9826
   0.7227   0.7227   0.7227
»
```

4.2 Llamadas a funciones.

Una “función” es el equivalente en Fortran de una rutina, en C su equivalente también son las funciones. Existen una serie de funciones “internas” lo cual quiere decir que son accesibles desde cualquier programa. Por ejemplo la función ‘sqrt()’ que calcula la raíz cuadrada es una de ellas. Además, el usuario puede definir sus propias funciones editando un archivo de texto y escribiendo una serie de sentencias. De hecho Matlab viene con una librería de funciones que, como no son tan requeridas no se han introducido como internas.

La forma de llamar a una función es a través de una "llamada de función":

```
sqrt(x^2 + y^2)        #un argumento
ones(n,m)              #dos argumentos
rand()                 #ningún argumento
```

cada función espera un número de argumentos, por ejemplo 'sqrt()' espera solo un argumento.

Múltiples argumentos de entrada y salida.

Algunas de las funciones pueden esperar un número variable de argumentos o puede retornar múltiples valores, por ejemplo al función 'eig' retorna la descomposición normal de una matriz diagonal de autovalores y la matriz de autovectores:

```
» a =[2 1; 1 2]
a =
    2    1
    1    2
» eig(a)
ans =
    1
    3
» [v, d] = eig(a)
v =
   0.7071    0.7071
  -0.7071    0.7071
d =
    1    0
    0    3
»
```

Notar que en la primera llamada no hay miembro izquierdo en la asignación y el valor retornado es un vector con los autovalores. Al escribir una función en Matlab podemos saber cuantos argumentos están requiriendo y dependiendo de esto retornar los valores apropiados.

Llamadas por valor

El mecanismo de paso de argumentos en Matlab es "por valor", en contraposición con Fortran donde los valores se pasan *por referencia.* Esto significa que en realidad la función ve internamente una copia de la variable. Esto evita tener que hacer copias internas de la variable para evitar que su valor fuera de la rutina sea modificado accidentalmente. También permite pasar constantes como argumentos incluso en el caso en que la función va a modificar los valores internamente. Esto parecería representar un desperdicio de memoria ya que en el siguiente caso 'x' ocupa 8 Mb de memoria y al pasarlo como argumento a 'f()' la copia interna ocupará otros 8 Mb de memoria. Sin embargo Matlab y Octave son lo suficientemente astutos como para crear la copia solo si la variable va a ser modificada internamente.

```
x = rand(1000);
f(x);
```

Llamada recursiva

Salvo en casos especiales, se puede llamar a funciones recursivamente. Por ejemplo, se puede calcular el factorial de un entero de la siguiente manera:

```
function retval = fact( n )
if( n > 0 )
        retval = n * fact( n - 1 );
else
        retval = 1;
end
end
```

En general es ineficiente hacer esto ya que cada vez que la función es llamada se guarda una copia de todas las variables de la función. Una forma mucho más eficiente es usar 'prod(1:n)'.

4.3 Operaciones aritméticas.

Para matrices 'X' e 'Y' podemos hacer las siguientes operaciones:

- 'X + Y', 'X - Y' suma y resta de matrices. Las dimensiones deben coincidir.
- 'X * Y' Multiplicación de matrices. La dimensión interna debe coincidir, es decir las dimensiones deben ser 'n x m' y 'm x p'.
- 'X .* Y', 'X ./ Y' multiplicación y división elemento a elemento.
- 'X \ Y', 'X / Y' división a izquierda y derecha.
- 'X ^ k' potencia de matrices.
- 'X .^ k' 'X .^ Y' potencia de matriz elemento a elemento.
- 'X'' transpuesta conjugada de X.
- 'X.'' transpuesta de X.

Resolución de sistemas de ecuaciones lineales.

Sea resolver el sistema de ecuaciones lineales 'a * x = b'. La forma más evidente de hacerlo es usando la función 'inv()' que retorna la inversa de la matriz. El uso del operador '\' es mucho más eficiente ya que no invierte la matriz 'a' sino que la factoriza tipo 'L*U' y luego elimina. Esto es de tener en cuenta especialmente para matrices grandes.

```
» a = [ 2 1; 1 2 ]; b = [1; 0];
» x = inv(a)*b
x =
   0.6667
  -0.3333
» x = a\b
x =
   0.6667
  -0.3333
»
```

Operaciones elemento a elemento

Tal vez uno de los elementos más útiles de Matlab y Octave es el uso de los operadores "elemento a elemento". Si 'a = b .* c' esto es equivalente a un doble lazo en los "i j" sobre la operación: 'a(i,j) = b(i,j) * c(i,j)'. Por ejemplo podemos calcular el producto escalar de dos vectores de la siguiente forma. La función 'sum()' retorna la suma de los elementos del vector argumento.

```
» a = rand(20,1); b = rand(20,1);
» p = sum(a.*b)
p =
   5.9702
»
```

El uso de estas operaciones no sólo significa una notación más compacta que es mucho más eficiente. Por ejemplo, si 'a' y 'b' son matrices de 500 x 500, entonces la siguiente operación es equivalente a la versión compacta de 'b = a.^0.5'. Sin embargo la opción con lazos 'for' tarda unas 70 veces más que la versión compacta.

```
for k = 1:500
        for l = 1:500
                b(k,l) = a(k,l)^0.5;
        end
end
```

4.4 Operadores de comparación

- 'X < Y' Verdadero si 'X' es menor que 'Y'
- 'X <= Y' Verdadero si 'X' es menor que 'Y'
- 'X == Y' Verdadero si 'X' es igual a 'Y'
- 'X >= Y' Verdadero si 'X' es mayor o igual que 'Y'
- 'X >Y' Verdadero si 'X' es mayor que 'Y'
- 'X != Y' , 'X ~= Y', 'X <> Y' Verdadero si 'X' no es igual a 'Y'

Todos los operadores de comparación retornan 1 si el resultado es verdadero y 0 si es falso. La comparación se hace siempre elemento a elemento:

```
» a
a =
    1    1    2
    4    0    4
    4    4    0
» a>2
ans =
    0    0    0
    1    0    1
    1    1    0
» a>=2
```

```
ans =
    0    0    1
    1    0    1
    1    1    0
» a==0
ans =
    0    0    0
    0    1    0
    0    0    1
»
```

Combinado con las expresiones de indexación “cero-uno” explicadas anteriormente esto permite extraer convenientemente columnas y filas de una matriz. En el ejemplo siguiente se usa esta combinación para extraer primero todos los elementos del vector ‘a’ que son menores o iguales que 3 y después el complemento.

Puede evitarse el uso de indexación “ cero-uno” utilizando la función ‘find ()’.

4.5 Operadores lógicas.

Existen los siguientes operadores lógicos:

- ‘X & Y’ operador lógico “AND”
- ‘X Y’ operador lógico “OR”
- ‘!X’ , ‘~X’ operador lógico “NOT”

Un elemento 0 es interpretado como falso y en caso contrario como verdadero.

```
» a
a =
    4    4    2    4    4    4    3    3    5    1
» a <= 3
ans =
    0    0    1    0    0    0    1    1    0    1
» a(a <= 3)
ans =
    2    3    3    1
» a(a > 3)
ans =
    4    4    4    4    4    5
»
```

4.6 Asignaciones.

Son expresiones que guardan un nuevo valor en una variable. Las variables no tienen un tipo definido por ejemplo:

```
» foo = 1
foo =
   1
» foo = 'bar'
foo =
bar
»
```

Asignar una matriz vacía '[ ]' equivale muchas veces a eliminar esa fila o columna. Sin embargo los elementos eliminados deben ser tales que la matriz resultante sea rectangular, sino los resultados son impredecibles.

```
» a
a =
   7   9   3
   2   3   4
   6   4   7
» a(:,1) = []
a =
   9   3
   3   4
   4   7
» a(1,1) = []
??? In an assignment A(matrix,matrix) = B, the number of rows in B
and the number of elements in the A row index matrix must be the same.
»
```

4.7 Operadores de incremento

Estos operadores solo existen en Octave. Ellos incrementan el valor de una variable en más o menos una unidad. Existen versiones "pre" y "post" dependiendo de si la operación se produce antes o después de retornar el resultado. Son muy utilizados en lazos y son más eficientes que la operación 'x=x+1'.

- '++x' pre-incremento en 1 de 'x'
- '--x' pre-incremento en -1 de 'x'
- 'x++' post-incremento en 1 de 'x'
- 'x--' post-incremento en -1 de 'x'

4.8 Precedencia de los operadores

La precedencia de los operadores indica cuales son las operaciones que se realizan primero al evaluar una expresión. Por ejemplo, '-x^2' se reduce a '-(x^2)' ya que el operador '^' tiene precedencia sobre '-'.

- operadores de fin de sentencia ';', ','
- asignación '='
- operadores lógicos '|', '&'
- operadores de relación '<', '<=', '==', etc.
- operador dos puntos (rango) ':'
- suma y resta '+', '-'
- multiplicación y división '*', '/'
- transpuesta ' ' '
- operadores unitarios '-', '+'
- exponenciación '^'

5. Sentencias de control de flujo.

Las sentencias de control de flujo como 'if', 'while' aceptan 'end' como sentencia de finalización. En Octave existen sentencias específicas como 'endif', 'endwhile'. Ambas posibilidades son equivalentes en Octave pero el uso de la versión larga facilita la detección y diagnóstico de errores por el "parser".

5.1 Sentencia 'if'.

```
if(CONDICION)
        if SENTENCIA
elseif(CONDICION)
        elseif SENTENCIA
else
        else SENTENCIA
end
```

La parte 'elseif' y 'else' son opcionales. Puede haber varias secciones 'elseif' pero solo una 'else'. Notar que 'elseif' va todo junto, no debe ir separado como en 'else if'.

Ejemplo:

```
if( rem(x,2) == 0 )
        printf( "x es par\n" );
elseif( rem(x,3) == 0 )
        printf( "x es impar y divisible por 3\n" );
else
        printf( "x es impar pero no divisible por 3\n" );
end
```

Para verificar condiciones sobre matrices es importante saber que si una matriz aparece en una condición esta es tomada como verdad si todos los elementos son verdad:

```
» x
x =
    3    5    1
    3    2    7
    3    8    4
» if x>2 ; disp('verdad'); end
» if x>0 ; disp('verdad'); end
verdad
»
```

Esto también se puede lograr con la función 'all()' para vectores fila o columna 'all(x)' d 1 o 0 si todos los elementos son 'verdaderos' (diferentes de cero). Para una matriz 'X' de 'n x m' la expresión 'Y = all(X)' retorna un vector de '1 x m' donde 'Y(k) = all(X(:,k))'. Una función similar es 'any(X)' que retorna 1 si alguno de los elementos es verdadero.

5.2 Sentencia 'while'

Es el operador más simple de repetición. Repite un cierto grupo de sentencias hasta que se cumpla una cierta condición:

```
while(CONDICION)
        SENTENCIA
end
```

El ejemplo siguiente calcula el factorial de un entero 'n':

```
p = 1;
while( n > 1 )
p = p * n--;
end
```

5.3 Sentencia 'for'

Esta es la sentencia más conveniente cuando se deben contar iteraciones en un lazo:

```
for( var = EXPRESION )
        SENTENCIA
end
```

En general 'EXPRESION' puede ser una matriz y 'var' va tomando dentro del lazo cada una de las columnas de 'EXPRESION'. Por ejemplo:

```
» for k = [3 5 4 7 6]
k
end
k =
    3
```

```
k =
    5
k =
    4
k =
    7
k =
    6
»
```

El uso más frecuente es para hacer un lazo sobre un índice que se incrementa constantemente en cada iteración, esto se puede hacer en forma muy compacta con un "rango": 'for k=1:n'. En una versión más sofisticada la variable va tomando como valor cada una de las columnas de la expresión.

5.4 La sentencia 'break'

Esta sentencia permite "salir" de un lazo "while" o "for".

5.5 La sentencia 'continue'

Es similar a "break" pero pasa a la siguiente iteración del lazo.

5.6 Continuación de líneas

El operador de continuación de líneas es '...'. Octave soporta la continuación de líneas de Matlab y también la forma más usual en la mayoría de los lenguajes de script de Unix ('\')

```
» x = varable_demasiado_larga + ...
otra_variable_demasiado_larga + ...
1
```

6. Funciones y Scripts

Cuando un cierto grupo de sentencias es muy usado puede incluirse en un archivo con extensión 'file.m' y ser llamado desde Matlab u Octave por su nombre:

```
>> file
```

Existen dos tipos de archivo '.m', los "scripts" y las "funciones". Los scripts son simplemente listas de sentencias, al ser llamado es como si la lista de sentencias fuera incluido en el prompt de Matlab.

Un script puede llamar a otro script y así siguiendo.

Las funciones primero pasan valores de entrada y de salida, los cuales dentro de la función toman nombres distintos. las demás variables externas quedan aisladas de las internas.