Capítulo 1

Introducción a Octave

1.1 Resumen

Estas notas corresponden a un curso de grado sobre la utilización de Octave (o Matlab), principalmente orientado a la resolución de ecuaciones diferenciales. Está basado fundamentalmente en el manual de Octave escrito por el creador de Octave John W. Eaton (<jwe@bevo.che.wisc.edu>). La utilidad de estas notas es en parte que son una traducción al castellano, pero además, a diferencia de áquel este no pretende ser un manual de referencia sino que es apto para ser un curso de grado de introducción al lenguaje de Octave.

1.2 Introducción

Octave es un lenguaje de alto nivel, diseñado para hacer cálculos numéricos. Provee una interfase con línea de comando para resolver problemas lineales y noineales y otros experimentos numéricos numéricamente. Puede ser usado también en forma de procesamiento tipo *batch*.

En la mayoría de los sistemas, la forma de invocar a Octave es con el comando del shell `'octave'`. Octave muestra un mensaje inicial inicando que esta listo para aceptar instrucciones. A partir de allí se pueden escribir comandos inmediatamente.

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

Los siguientes capítulos describen todas las funcionalidades de Octave en detalle, pero antes de eso, puede ser muy útil dar una muestra de sus posibilidades.

Si eres nuevo en el uso de Octave, es recomendable que trates de reproducir estos ejemplos usando Octave. Las líneas marcadas como `'octave>'` son líneas que tu debes escribir, terminandolas con un "retorno de carro" (la tecla `'Enter'` en la PC). Octave responderá con un resultado o un gráfico.

1.2.1 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:

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

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

```
octave> b = rand (3, 2);
octave>
```

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:

```
octave> b
b =
  0.70335  0.88008
  0.26807  0.79486
  0.94203  0.24523
octave>
```

1.2.2 Aritmética con matrices

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

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

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

```
octave> a * b
ans =
   2.8555   2.1654
  10.9866   8.5763
  46.8020  36.4707
octave>
```

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

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

1.2.3 Resolución de sistemas de ecuaciones

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

```
octave> a \ b
warning: matrix singular to machine precision, rcond = 2.00564e-18
ans =
   1.38217   1.47409
  -1.16316  -1.25136
   0.21901   0.22272
octave>
```

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

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

1.2.4 Integrando ecuaciones diferenciales

Octave tiene funciones internas para resolver ecuaciones diferenciales nolineales de la forma:

```
dx/dt= f (x, t)
```

con la condición inicial:

```
x(t = t0) = x0
```

Para que Octave integre ecuaciones de esta forma, debes primero escribir una función `'f(x,t)'`. Esto puede ser hecho directamente en la línea de comando. Por ejemplo, los comandos siguientes definen el miembro derecho de un sistema de dos ecuaciones diferenciales nolineales de sumo interés. Nótese que mientras estás escribiendo la función, Octave responde con un prompt diferente, lo cual indica que está esperando para completar tu entrada de la misma:

```
octave> function xdot = f (x, t)
>
> r = 0.25;
> k = 1.4;
> a = 1.5;
> b = 0.16;
```

```
> c = 0.9;
> d = 0.8;
>
> 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);
>
> endfunction
```

Dada la condición inicial:

```
octave> x0 = [1; 2];
```

y el conjunto de instantes temporales como un vector columna (nótese que el primer elemento del vector corresponde a la condición inicial dada arriba):

```
octave> t = linspace (0, 50, 200)';
```

es fácil integrar el sistema:

```
octave> x = lsode ("f", x0, t);
```

La función `'lsode'` usa el *"Livermore Solver for Ordinary Differential Equations",* described in A. C. Hindmarsh, *"ODEPACK, a Systematized Collection of ODE Solvers"*, in: *Scientific Computing,* R. S. Stepleman et al. (Eds.), North-Holland, Amsterdam, 1983, pages 55-64.

1.2.5 Obtención de salidas gráficas

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

```
octave> plot (t, x)
```

Si estás usando el sistema X Window, Octave creará automaticamente una ventana separada para mostrar el gráfico.

1.2.6 Recuperación de comandos

En el prompt de Octave puedes recuperar, editar y reemitir comandos previos usando comandos al estilo de Emacs o Vi.

1.2.7 Usando la ayuda

Octave 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, debes primero conocer el nombre del comando que quieres 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. Puedes obtener más ayuda sobre cualquiera de los items listados incluyendo simplemente el nombre del item como argumento. Por ejemplo:

```
help plot
```

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

1.3 Tipos de datos

Todas las versiones de Octave 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 especilizados escribiendo las rutinas apropiadas en C++ (esta es una ventaja de Octave de ser "free-software" es decir que tenemos acceso a los fuentes). En algunos sistemas no es necesario recompilar Octave si es posible hacer uso de la opción de linkedición dinámica de vínculos.

1.3.1 Objetos Numéricos

Los objetos numéricos internos de Octave 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 o submatrices usando un sistema muy poderoso de indexación. (Esto lo veremos más adelante).

1.3.2 Cadenas de caracteres (strings)

Una cadena de caracteres en Octave puede ser entrada delimitando la cadena con apóstrofes o comillas:

```
octave> a="parrot"
a = parrot
octave> b='parrot'
b = parrot
octave>
```

La notación con apóstrofes es compatible con Matlab mientras 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, Octave 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.

Los caracteres especiales que no pueden ser introducidos directamente (por ejemplo las comillas) deben ser "escapeados" ("escaped"):

```
octave> a="Esto es \"interesante\""
a = Esto es "interesante"
octave> b="directorio en DOS c:\\PEDRO\\OCTAVE\\MACROS"
b = directorio en DOS c:\PEDRO\OCTAVE\MACROS
octave>
```

1.3.3 Estructuras de datos

Esta opción 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>
```

1.4 Tipos de datos numéricos

Estas 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.

1.4.1 Matrices

Las matrices son entradas entre corcheter por filas. Los elementos en la misma fila van separados por comas y las filas van separadas por punto y coma:

```
octave> a = [1, 2; 3, 4]
a =
  1  2
  3  4
octave>
```

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

```
octave> [a , a]
ans =
  1  2  1  2
  3  4  3  4
octave> [a ; a]
ans =
  1  2
  3  4
  1  2
  3  4
octave> [a ; [1 1] ]
ans =
  1  2
  3  4
  1  1
octave> [a , [1 1] ]
error: number of rows must match
octave>
```

1.4.2 Tamaño de los objetos

Las funciones ‘`columns()`’, ‘`rows()`’, ‘`length()`’ y ‘`size()`’ permiten inspeccionar las diferentes dimensiones de las matrices. ‘`size()`’ retorna un vector de dos componentes con el número de columnas y de filas, mientras que ‘`columns()`’ y ‘`rows()`’ retornan cada uno de esos valores por separado. ‘`length()`’ retorna el maximo de ambas dimensiones: su uso está orientado a vectores ya sean horizontales o verticales.

```
octave> a=rand(3,5)
a =
  0.700263  0.029678  0.858449  0.836040  0.205443
  0.814676  0.162003  0.361927  0.325249  0.740022
  0.223069  0.722034  0.221037  0.202889  0.606426
octave> columns(a)
ans = 5
octave> rows(a)
ans = 3
octave> size(a)
ans =
```

```
  3  5
octave> b=rand(5,1);
octave> length(b)
ans = 5
octave> length(a)
ans = 5
octave> length(b)
ans = 5
octave> length(b')
ans = 5
octave> columns(b)
ans = 1
octave> columns(b')
ans = 5
octave>
```

Además, existen funciones básicas que retornan valores Booleanos (1=true, 0=false): `'is_empty(A)'`, `'is_vector (A)'`, `'is_scalar (A)'`, `'is_square (X)'`, `'is_symmetric (X, TOL)'`.

1.4.3 Rangos

Un "rango" es una forma conveniente de construir vectores con elementos espaciados uniformemente. El rango esta definido por el valor inicial, incremento (opcional) y valor final separados por `':'`. Si el incremento no está se asume 1.

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

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 es necesario para ahorrar memoria. Por ejemplo si queremos hacer un lazo (esto se 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 8Mb 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()'`.

1.5 Expresiones

Las expresiones son la unidad básica con la cual se arman las sentencias en Octave. 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.

1.5.1 Expresiones de índices

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

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

En general la expresión `'a(i1,i2)'` retorna los valores de la submatriz de `'a'` conteniendo las filas cuyos índices estn 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 `'i2'`. Por ejemplo:

```
octave> a(1,[2 1])
ans =
  2  1
octave>
```

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 3 veces es el llamado *"truco de Tom"* (*"Tom's trick"*)

```
octave> a=rand(3,1)
a =
  0.456580
  0.080549
  0.202045
octave> a(:,[1 1 1])
ans =
  0.456580  0.456580  0.456580
  0.080549  0.080549  0.080549
  0.202045  0.202045  0.202045
octave>
```

Indexamiento "cero-uno"

Octave acepta un tipo especial de indexamiento llamado *"cero-uno"* (*"zero-one indexing"*). Esto es una extensión de Octave (es decir que en este sentido no es compatible con Matlab). La idea es que si el vector de índices está formado sólo por unos y ceros entonces podemos extraer ciertas columnas (o filas) con un vector de índices que tenga 1 en aquellas columnas que queremos extraer y 0 en las otras. Por ejemplo:

```
octave> a(:,[0 1])
ans =
  2
  4
octave>
```

extrae la segunda columna de `'a'`. Cuando todos los índices son 1, esta modalidad de indexación entra en conflicto con la tradicional. Por ejemplo `'a(:,[1 1])'` puede interpretarse como:

- una matriz con la primera columna de `'a'` si lo interpretamos en la forma de indexación *tradicional*
- `'a'`, si lo interpretamos como indexación "cero-uno".

Para evitar esta ambigüedad podemos indicar explícitamente a Octave cual de las dos formas de indexación debe emplear en caso de haber conflicto. Esto se hace a través de una *variable interna* llamada `'prefer_zero_one_indexing'`. Puesto a 1 indica que en caso de conflicto use indexación "cero-uno" y si esta puesto a 0 que utilice la convencional. Notar que este es el caso del *"truco de Tom"* mencionado, pero en ese caso no hay conflicto ya que en general la dimensión que se expande es originariamente 1.

```
octave> a
a =
  1  2
  3  4
octave> prefer_zero_one_indexing=1;
octave> a(1,[1 1])
ans =
  1  2
octave> prefer_zero_one_indexing=0;
octave> a(1,[1 1])
ans =
  1  1
octave>
```

Existen toda una serie de variables como ésta que permiten configurar a gusto el comportamiento de Octave en varias situaciones.

Indexación Fortran

Si la variable `'do_fortran_indexing'` es puesta a 1, entonces una matriz puede ser indexada por un sólo índice, como es usual en Fortran. Recordar que en ese caso, los elementos son recorridos por columna, es decir `'a(1)=a(1,1), a(2)=a(2,1),'` ... `'a(rows(a)+1)=a(1,2)'` etc... Por ejemplo:

```
octave> do_fortran_indexing =1
do_fortran_indexing = 1
octave> a(:)
ans =
  1
  2
  3
  4
  5
  6
  7
  8
  9
octave> a=rand(2)
a =
  0.48575  0.46822
  0.57453  0.50091
octave> a(:)
ans =
  0.48575
  0.57453
  0.46822
  0.50091
octave>
```

1.5.2 Llamadas a funciones

Una *"función"* es el equivalente en Fortran de una rutina. 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 Octave viene con una librería de tales 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 ()               # ningun argumento
```

cada función "espera" un número de argumentos, por ejemplo `'sqrt()'` espera un solo agumento:

```
sqrt(ARGUMENTO)
```

Múltiples argumentos de entrada y salida

Algunas de las funciones pueden esperar un número variable de argumentos. Esto es muy usado por ejemplo cuando hay parámetros optativos, como tolerancias. En el siguiente ejemplo generamos una matriz random *"casi-simétrica"* (con un pequeña componente aleatoria `'0(1e-3)'`). `'is_symmetric'` retorna un valor `'0'` (Boolean *"false"*)lo cual indica que la matriz no es simétrica con la tolerancia interna (`'0(1e-16)'`) mientras que si le indicamos una tolerancia de `'1e-2'` a través de un segundo parámetro, entonces `'is_symmetric'` sí retorna 3 (la dimension de la matriz) que corresponde a un valor lógico *"true"*.

```
octave> a=rand(3);
octave> a=a+a'+1e-3*rand(3)
a =
  1.4658882  1.2932925  1.1064879
  1.2933913  1.8328507  0.2221298
  1.1065347  0.2216927  0.0087181
octave> is_symmetric(a)
ans = 0
octave> is_symmetric(a,1e-2)
ans = 3
octave>
```

Una llamada a función también puede retornar múltiples valores, por ejemplo la función `'eig'` retorna la descomposición normal de una matriz en forma de la matriz diagonal de autovalores y la matriz de autovectores:

```
octave> a=[2 1; 1 2]
a =
  2  1
  1  2
octave> eig(a)
ans =
  1
  3
octave> [v,d]=eig(a)
v =
  -0.70711   0.70711
   0.70711   0.70711
d =
  1  0
  0  3
octave>
```

Nótese 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 de Octave podemos saber cuantos argumentos se están requiriendo y dependiendo de esto retornar los valores apropiados.

Llamadas por valor

El mecanismo de paso de argumentos en Octave 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 8Mb de memoria y al pasarlo como argumento a `'f()'` la copia interna ocupará otros 8Mb. Sin embargo Octave es lo sufuecientemente astuto como para crear la copia `sólo 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;
  endif
endfunction
```

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)'`, o `'gamma (n+1)'`.

1.5.3 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 a derecha.
- `'X ^ Y'` o `'X ** Y'` potencia de matrices.
- `'X .^ Y'` o `'X .** Y'` operador de potencia elemento a elemento.
- `'X''` transpuesta conjugada de `'X'`
- `'X.''` transpuesta de `'X'`

Resolución de sistemas

Sea resolver el sistema de ecuaciones `'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 non invierte la matriz `'a'` sino que la factoriza tipo `'L*U'` y luego elimina. Esto es de tener en cuenta especialmente para matrices grandes.

```
octave> a=[2 1; 1 2]; b=[1;0];
octave> x=inv(a)*b
x =
   0.66667
  -0.33333
octave> x=a\b
x =
   0.66667
  -0.33333
octave>
```

Operaciones "elemento a elemento"

Tal vez uno de los elementos más útiles de 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.

```
octave> a=rand(20,1); b=rand(20,1);
octave> p=sum(a.*b)
p = 5.1900
octave>
```

El uso de estas operaciones no sólo significa una notación más compacta sino que es mucho más eficiente. Por ejemplo, si `'a'` y `'b'` son matrices de 500x500, entonces la siguiente operación es equivalente a la versión compacta `'b=a.^0.5'`. Sin embargo la opción con lazos `'for'` tarda unas 70 veces m'as lo que la versión compacta.

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

1.5.4 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 myor 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 compración se hace siempre elemento a alemento:

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

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

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

Puede evitarse el uso de indexación "cero-uno" utilizando la función `'find()'`.

1.5.5 Operaciones lógicas

Existen los siguientes operadores lógicos:

- `'X & Y'` operador lógico "AND"
- `'X | Y'` operador lógico "OR"
- `'!', '~X'` operador lógico "NOT"

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

Existen las versiones *"corto-circuitadas"* de estos operadores `'X && Y'` y `'X || Y'`. La idea es que el segundo argumento es evaluado sólo si el primero cumple con la condición. Por ejemplo, la expresión `'a>0 && b=sqrt(a)'` sólo evalúa la expresión `'b=sqrt(a)'` si `'a>0'`. Esto puede hacerse también con la construcción `'if-endif'`, pero es más largo.

```
octave> a=rand
a = 0.29860
octave> a>0 && b=sqrt(a) ; b
b = 0.54644
octave> a=-rand
a = -0.25817
octave> a>0 && b=sqrt(a) ; b
b = 0.54644
octave>
```

1.5.6 Asignaciones

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

```
octave> foo = 1
foo = 1
octave> 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.

```
octave> a(:,1)=[]
a =
  7  9
  2  3
  6  4
octave> a(1,1)=[]
a =
```

```
  7  9
  2  3
  6  4
octave>
```

1.5.7 Operadores de incremento

Estos operadores incrementan el valor de una variable en la más o menos la 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'`

Por ejemplo:

```
octave> x=5; x++
ans = 5
octave> x
x = 6
octave> x=5; ++x
x = 6
octave> x
x = 6
octave>
```

1.5.8 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 cortocircuitados `'||'`, `'&&'`
- operadores lógicos `'|'`, `'&'`
- operadores de relacion `'<'`, `'<='`, `'=='`, `'>='`, `'>'`, `'!'`, `'~='`, `'<>'`
- operador "dos puntos" (rango) `':'`

- suma y resta ‘+’, ‘-’
- multiplicación y división ‘*’, ‘/’, ‘\’, ‘.\’, ‘.*’, ‘./’
- transpuesta ‘’’, ‘.’’
- operadores unarios ‘+’, ‘-’, ‘++’, ‘--’, ‘!, ‘~’
- exponenciación ‘^’, ‘**’, ‘.^’, ‘.**’

1.6 Sentencias de control de flujo

Las sentencias de control de flujo como ‘if’, ‘while’ aceptan ‘end’ como sentencia de finalización, igual que Matlab. Eisten además sentencias de finalización específicas como ‘endif’, ‘endwhile’. Ambas posibilidades son equivalentes pero el uso de la versión “larga” facilita la detección y diagnóstico de errores por el “parser”.

1.6.1 Sentencia ‘if’

```
if (CONDITION)
  THEN-BODY
elseif (CONDITION)
  ELSEIF-BODY
else
  ELSE-BODY
endif
```

La parte ‘elseif’ y ‘else’ son opcionales. Puede haber varias secciones ‘elseif’ pero sólo 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");
endif
```

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

```
octave> x
x =
  3  5  1
  3  2  7
  3  8  4
```

```
octave> if x>2 ; disp("verdad") ; endif
octave> if x>0 ; disp("verdad") ; endif
verdad
octave>
```

Esto también se puede lograr con la función `'all()'`. para vectores fila o columna `'all(X)'` da 1 o 0 si todos los elementos son "verdaderos" (diferentes de 0). 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.

```
octave> x
x =
  3  5  1
  3  2  7
  3  8  4
octave> all(x==3)
ans =
  1  0  0
octave> all(x>1)
ans =
  1  1  0
octave> any(x==1)
ans =
  0  0  1
octave> any(x>7)
ans =
  0  1  0
octave>
```

1.6.2 Sentencia 'while'

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

```
while (CONDITION)
  BODY
endwhile
```

Por ejemplo el ejemplo siguiente calcula el factorial de un entero `'n'`:

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

1.6.3 Sentencia 'for'

Esta es la sentencia más conveniente cuando se deben contar iteraciones en un lazo. Corresponde a la sentencia `'do'` de Fortran y la forma general es:

```
for VAR = EXPRESSION
  BODY
endfor
```

En general `'EXPRESSION'` puede ser una matriz y `'VAR'` va tomando dentro del lazo cada uno de las columnas de `'EXPRESSION''`. Por ejemplo:

```
octave> for k=[3 5 4 7 6]
> k
> endfor
k = 3
k = 5
k = 4
k = 7
k = 6
octave>
```

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:

```
octave> a
a =
   3   2   1   8   1
   6   6  10   5   4
octave> for col=a
> col'
> endfor
ans =
  3  6
ans =
  2  6
ans =
  1  10
ans =
  8  5
ans =
  1  4
octave>
```

1.6.4 La sentencia 'break'

Esta sentencia permite "salir" de un lazo `'while'` o `'for'`, el siguiente ejemplo encuentra el mínimo divisor de un entero y también identifica si el número es primo

```
num = 103;
div = 2;
while (div*div <= num)
  if (rem (num, div) == 0)
    break;
  endif
  div++;
endwhile
if (rem (num, div) == 0)
  printf ("Smallest divisor of %d is %d\n", num, div)
else
  printf ("%d is prime\n", num);
endif
```

1.6.5 La sentencia 'continue'

Es similar a `'break'` pero pasa a la siguiente iteración del lazo. El siguiente ejemplo muestra por pantalla sólo los elementos elementos pares de un vector `'vec'`:

```
vec = round (rand (1, 10) * 100);

# imprime solo los elementos pares

for x = vec
  if (rem (x, 2) ! 0)
    continue;
  endif
  printf ("%d\n", x);
endfor
```

1.6.6 Continuación de lineas

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= variable_demasiado_larga + ...
>    otra_variable_demasiado_larga \
>    + 1
```

1.7 Funciones y Scripts

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

```
octave> file
```

Existen dos tipos de archivos `'.m'`, los "scripts" y las "funciones". Los scripts son simplemente listas de sentencias, al ser llamado es como si las lista de sentencias fuera incluido en el prompt de Octave. Por ejemplo si escribimos el siguiente archivo `'program.m'` (`'$'` es el prompt del shell de Unix):

```
$ cat program.m
disp(" Sea la matriz: ")

A=rand(5);

disp(" y el miembro derecho: ")

b=rand(5,1)

disp(" la solucion del sistema lineal \"A x = b\" es: ")

x=A\b

$
```

y lo invocamos en el prompt de Octave, obtenemos:

```
octave.bin> program
 Sea la matriz:
A =

  0.0585678  0.4895890  0.2189970  0.3789282  0.0070595
  0.9847305  0.8799793  0.5523543  0.3834434  0.9909419
  0.9980757  0.6343816  0.5049118  0.9492567  0.3068205
  0.6248775  0.1056383  0.8699436  0.8245406  0.6066207
  0.0062269  0.1845125  0.9168088  0.1129778  0.6668941

 y el miembro derecho:
b =

  0.83350
  0.72816
  0.73784
  0.57180
  0.37908
```

```
 la solucion del sistema lineal "A x = b" es:
x =

  -1.27390
   0.94521
  -0.62628
   1.52004
   0.92228

octave>
```

Un script puede llamar a otro script y así siguiendo.

Las funciones son más parecidas al concepto de "subrutina" en Fortran. Primero, pasan ciertos valores de entrada y de salida, los cuales dentro de la función toman nombres distintos. Las demás variables "externas" quedan aisladas de las "internas". Por ejemplo si definimos la siguiente función que calcula el promedio de los elementos de un vector:

```
function retval = avg (v)
  retval = sum (v) / length (v);
endfunction
```

y la llamamos de la siguiente manera en Octave:

```
octave> vec=rand(1,5)
vec =

  0.61434  0.26735  0.15900  0.77773  0.85297

octave> avg(vec)
ans = 0.53428
octave> v="string"
v = string
octave> avg(vec)
ans = 0.53428
octave> v="string"
v = string
octave>
```

Vemos que la variable que "afuera" es llamada `'vec'` adentro de la función es llamada `'v'`. Además, podemos usar afuera una variable llamada `'v'` sin que entre "en colisión" con la interna.