1. Operaciones Básicas

1.1. Operaciones con matrices y vectores

Como se comentó en la introducción que hemos visto en el punto anterior, Scilab es un programa creado para trabajar con matrices, por lo tanto, este punto es probablemente el más importante y en el que mejor tenemos que aclararnos para empezar a trabajar. Tenemos muchas opciones para trabajar con ellas, podemos intercambiar matrices, permutarlas, invertirlas; Scilab es una herramienta de cálculo muy potente en lo que a matrices se refiere.

1.1.1. Introducción de matrices desde el teclado

Las matrices y vectores son variables del programa cuyos nombres podemos definir, siempre y cuando no utilicemos los caracteres que el programa tiene como caracteres prohibidos.

Para definir Scilab, se determinan el número de filas y de columnas en función del número de elementos que se proporcionan (o se utilizan). ***Las matrices se definen por filas***; los elementos de una misma fila están separados por ***blancos*** o ***comas***, mientras que las filas están separadas por pulsaciones ***intro*** o por caracteres ***punto y coma*** (;). Tomemos como ejemplo:

```
-->a=[1 2 1;3 4 2;5 3 1]
```

Cuya salida será:

```
a  =

!   1.0E+00    2.0E+00    1.0E+00 !
!   3.0E+00    4.0E+00    2.0E+00 !
!   5.0E+00    3.0E+00    1.0E+00 !
```

A partir de este momento la matriz **a** está disponible para hacer cualquier tipo de operación con ella (además de valores numéricos, en la definición de una matriz o vector se pueden utilizar expresiones y funciones matemáticas). Por ejemplo, una sencilla operación con **a** es hallar su ***matriz transpuesta***. En Scilab, el apóstrofo (') es el símbolo de *transposición matricial*. Para calcular **a'** (transpuesta de **a**) basta teclear lo siguiente (se añade a continuación la respuesta del programa):

```
-->a'
 ans  =

!   1.    3.    5. !
!   2.    4.    3. !
!   1.    2.    1. !
```

Como el resultado de la operación no ha sido asignado a ninguna otra matriz, Scilab utiliza un nombre de variable por defecto (***ans***, de *answer*), que contiene el resultado de la última operación. La variable ***ans*** puede ser utilizada como operando en la siguiente expresión que se introduzca. También podría haberse asignado el resultado a otra matriz llamada **b**.

Ahora vamos a definir una matriz **b** conjugada para hacer operaciones básicas con estas 2 matrices:

```
-->b=a'
 b  =

!   1.    3.    5. !
!   2.    4.    3. !
!   1.    2.    1. !
```

Comenzamos con las operaciones más básicas que podemos encontrar, la suma y la resta de matrices:

```
-->a+b
 ans  =

!   2.    5.    6. !
!   5.    8.    5. !
!   6.    5.    2. !

-->a-b
 ans  =

!   0.  - 1.  - 4. !
!   1.    0.  - 1. !
!   4.    1.    0. !
```

Si realizamos la multiplicación de matrices con el operando '*****' tendremos que tener cuidado con que el número de columnas de la primera matriz debe coincidir con el número de filas de la segunda:

```
-->a*b
 ans  =

!   6.     13.    12. !
!   13.    29.    29. !
!   12.    29.    35. !
```

También podemos utilizar una multiplicación elemento a elemento, que aunque no tiene demasiado sentido como multiplicación de matrices, si que es muy utilizable en el caso de que la matriz no sea más que un conjunto ordenado de valores.

```
-->a.*b
 ans  =

!   1.    6.    5. !
!   6.    16.   6. !
!   5.    6.    1. !
```

A continuación vamos a definir una nueva matriz **a** a partir de una función que genera valores aleatorios entre 0 y 1.

```
-->rand(2,2)
 ans  =

!   0.2113249    0.0002211 !
!   0.7560439    0.3303271 !
```

A partir de esta matriz **a** calculamos su inversa con el comando **inv(a):**

```
-->d=inv(a)  //es  necesario  que  la  matriz  sea  cuadrada  y  que  posea
determinante
 d  =

! - 2.     1.  - 4.9E-16 !
!   7.   - 4.    1.      !
! - 11.    7.  - 2.      !
```

Podemos comprobar multiplicando una por la otra que el cálculo es correcto:

```
-->a*d
 ans  =

!   1.    0.    0. !
!   0.    1.    0. !
!   0.    0.    1. !
```

Si los valores no son exactos podemos utilizar el comando **round()** ya que debido a los errores de aproximación en los cálculos podemos encontrar valores como `-4.9E-16` que representa un valor extremadamente pequeño.

```
-->d=round(d)
 d  =

! - 2.     1.    0. !
!   7.   - 4.    1. !
! - 11.    7.  - 2. !
```

Si queremos comentar las líneas de código que ejecutamos, a continuación de la operación podemos poner un comentario anteponiendo el carácter //

```
-->d=inv(a) //es necesario que la matriz sea cuadrada
```

De igual manera que se define una matriz podemos definir un vector:

```
-->x=[10 15 20] //vector fila
 x  =

!   10.    15.    20. !

-->y=[10;15;20]; //vector columna

-->z=[10,15,20]' //vector columna
 z  =

!   10. !
!   15. !
!   20. !
```

Podemos observar, podemos definir vectores fila y vectores columna, con sólo hacer la traspuesta del vector en la definición. También podemos definir un vector columna como si hicieramos una matriz de *1xn*

Como podemos ver, no hemos obtenido resultado tras realizar la operación; esto es debido a que hemos puesto un "**;**" al final de la línea de comando, esto hace que no salga por pantalla lo que hemos ejecutado, cosa que resulta muy útil cuando las matrices/vectores son de un número muy grande (100, 1000, …) y por lo tanto, difíciles de manejar visualmente.

En Scilab se accede a los elementos de un vector poniendo el índice entre paréntesis (por ejemplo *x(3)* ó *x(i)*). Los elementos de las matrices se acceden poniendo los dos índices entre paréntesis, separados por una coma (por ejemplo *A(1,2)* ó *A(i,j)*).

Las matrices ***se almacenan por columnas*** (aunque ***se introduzcan por filas***, como se ha dicho antes), y teniendo en cuenta esto puede accederse a cualquier elemento de una matriz con un sólo subíndice. Por ejemplo, si **A** es una matriz (3x3) se obtiene el mismo valor escribiendo *A(1,2)* que escribiendo *A(4)*.

1.1.2. Operaciones con matrices

Scilab puede operar con matrices por medio de ***operadores*** y por medio de ***funciones***. Se han visto ya los operadores *suma* (+), *producto* (*) y *traspuesta* ('), así como la función *invertir* ***inv( )***. Los operadores matriciales de SCILAB son los siguientes:

+	adición o suma
–	sustracción o resta
*	multiplicación
'	traspuesta
^	potenciación
\	división-izquierda
/	división-derecha
.*	producto elemento a elemento
./ y .\	división elemento a elemento
.^	elevar a una potencia elemento a elemento

Estos operadores se aplican también a las variables o valores escalares, aunque con algunas diferencias. Todos estos operadores son coherentes con las correspondientes operaciones matriciales: *no se puede por ejemplo sumar matrices que no sean del mismo tamaño*. Si los operadores no se usan de modo correcto se obtiene un mensaje de error.
Véase el siguiente ejemplo de tres ecuaciones formadas por una recta que no pasa por el origen y los dos ejes de coordenadas:

```
-->A=[1 2; 1 0; 0 1], b=[2 0 0]';
 A  =

!   1.    2. !
!   1.    0. !
!   0.    1. !

-->x=A\b, resto=A*x-b
 x  =

!   0.33333 !
!   0.66667 !
 resto  =

! - 0.33333 !
!   0.33333 !
!   0.66667 !
```

Vamos a ver como funcionan una serie de operadores:

```
-->a
 a  =

!   1.    2.    1. !
!   3.    4.    2. !
!   5.    3.    1. !

-->b
 b  =

!   1.    3.    5. !
!   2.    4.    3. !
!   1.    2.    1. !

-->a/b
 ans  =

!   0.    0.     1.  !
! - 2.    7.   - 9.  !
! - 7.    24.  - 36. !

-->a\b
 ans  =

!   0.  - 2.  - 7.  !
!   0.    7.    24. !
!   1.  - 9.  - 36. !
```

Si los operadores « / » y « \ » van precedidos de un "**.**" La operación se realiza elemento a elemento:

```
-->a./b
 ans  =

!   1.      0.66667    0.2      !
!   1.5     1.         0.66667  !
!   5.      1.5        1.       !

-->a.\b
 ans  =

!   1.         1.5        5.   !
!   0.66667    1.         1.5  !
!   0.2        0.66667    1.   !
```

1.1.3. Tipos de datos

Scilab trabaja siempre con el tipo Real de doble precisión, también tenemos la posibilidad de trabajar con *reales de simple precisión* (no hay operaciones implementadas), *enteros* y *booleanos*, pero no tienen buena parte de las operaciones matemáticas aunque sí lógicas. Este tipo de dato se guarda con un tamaño de 8Bytes, que tiene un tamaño de 15 cifras exactas. Además del tipo Real, podremos trabajar con strings, matrices, hipermatrices y estructuras más avanzadas.

1.1.3.1. Números reales de doble precisión

Los elementos constitutivos de vectores y matrices son números reales almacenados en 8 bytes (53 bytes para la mantisa y 11 para el exponente de 2; entre 15 y 16 cifras decimales equivalentes). Es importante saber cómo trabaja Scilab con estos números y los casos especiales que presentan. Scilab mantiene una forma especial para los *números muy grandes* (más grandes que los que es capaz de representar), que son considerados como ***infinito***. Por ejemplo, obsérvese cómo responde el programa al ejecutar el siguiente comando:

```
-->%inf
 %inf  =

   Inf
```

Así pues, para Scilab el ***infinito*** se representa como %***inf***. Scilab tiene también una representación especial para los resultados que no están definidos como números, que se representa como **Not a Number** (**%nan**):

```
-->%nan
 ans  =

   Nan
```

Los ***NaN*** se propagan al realizar con ellos cualquier operación aritmética, en el sentido de que, por ejemplo, cualquier número sumado a un ***NaN*** da otro ***NaN***. Scilab tiene esto en cuenta. Algo parecido sucede con los ***Inf***.

```
-->%nan*1
 ans  =

   Nan

-->%nan*%nan
 ans  =

   Nan

-->%nan*%inf
 ans  =

   Nan
```

Podemos encontrar 3 variables predefinidas por Scilab que nos dan los valores máximos y mínimos de este tipo de datos:

- ***eps*** devuelve la diferencia entre 1.0 y el número de coma flotante inmediatamente superior. Da una idea de la precisión o número de cifras almacenadas. En un PC, ***eps*** vale 2.2204e-016.
- ***realmin*** devuelve el número más pequeño con que se puede trabajar (2.2251e-308)
- ***realmax*** devuelve el número más grande con que se puede trabajar (1.7977e+308)

1.1.3.2. Números complejos (complex)

Muchas veces nos vamos a encontrar que el cálculo que necesitamos ejecutar nos lleva a tener que definir el cuerpo de los números complejos, dado que el cuerpo de los números reales no es suficiente, por ejemplo, para realizar transformadas de Fourier o Laplace. Para ello se define la variable compleja %**i**:

```
-->%i
 %i  =

    i
```

Como podemos ver, la definición se hace con el % como en todas las constantes prefijadas por el sistema , y que es necesario el uso del operador **"*"** para multiplicarlo por un escalar.

```
-->// j no se considera como variable imaginaria

-->a=[1+%i,2+3*%i;%i,%i + 4]
 a  =

!   1. + i      2. + 3.i !
!   i           4. + i   !

-->b=[1,2;0,4]+[1 3;1 1]*%i
 b  =

!   1. + i      2. + 3.i !
!   i           4. + i   !
```

Es importante advertir que el *operador de matriz transpuesta* (’), aplicado a matrices complejas, produce la matriz transpuesta conjugada. El operador punto y apóstrofo (**.’**) que calcula simplemente la matriz transpuesta.

```
-->a'
 ans  =

!   1. - i    - i        !
!   2. - 3.i    4. - i   !
-->a.'
 ans  =

!   1. + i      i        !
!   2. + 3.i    4. + i   !
```

1.1.3.3. Cadenas de caracteres

Para crear una cadena de caracteres (string) en Scilab podemos hacerlo de estos dos modos:

```
-->a='cadena de caracteres'
 a  =

 cadena de caracteres
-->a="cadena de caracteres"
 a  =

 cadena de caracteres
```

Debido a que para su uso es necesario un conocimiento previo de las funciones orientadas a matrices, postpondré otras explicaciones hasta que se hayan explicado estas, al igual que con los **strings**, se postpone la explicación de **hipermatrices, structs** y **cell arrays**.

Ya han aparecido algunos ejemplos de ***variables*** y ***expresiones*** matriciales. Ahora se va a tratar de generalizar un poco lo visto hasta ahora. Una ***variable*** es un nombre que se da a una entidad numérica, que puede ser una matriz, un vector o un escalar. El valor de esa variable, e incluso el tipo de entidad numérica que representa, puede cambiar a lo largo de una sesión de SCILAB 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*** (=).

Una expresión de SCILAB puede tener las dos formas siguientes: primero, asignando su resultado a una variable,

```
variable = expresión
```

y segundo evaluando simplemente el resultado del siguiente modo, `expresión` en cuyo caso el resultado se asigna automáticamente a una variable interna de SCILAB llamada ***ans*** (de *answer*) que almacena el último resultado obtenido. Se considera por

defecto que una expresión termina cuando se pulsa ***intro***. Si se desea que una expresión continúe en la línea siguiente, hay que introducir ***tres puntos*** (...) antes de pulsar ***intro***. También se pueden incluir varias expresiones en una misma línea separándolas por *comas* (,) o *puntos y comas* (;). Si una expresión ***termina en punto y coma*** (;) su resultado se calcula, pero no se escribe en pantalla. Esta posibilidad es muy interesante, tanto para evitar la escritura de resultados intermedios, como para evitar la impresión de grandes cantidades de números cuando se trabaja con matrices de gran tamaño.

A semejanza de **Fortran**, ***SCILAB distingue entre mayúsculas y minúsculas*** en los nombres de variables; además no hace falta declarar las variables que se vayan a utilizar. Esto hace que se deba tener especial cuidado con no utilizar nombres erróneos en las variables, porque no se recibirá ningún aviso del ordenador. Cuando se quiere tener una *relación de las variables* que se han utilizado en una sesión de trabajo se puede utilizar el comando ***who***. Existe otro comando llamado ***whos*** que proporciona además información sobre el tamaño, la cantidad de memoria ocupada y el carácter real o complejo de cada variable.

El comando ***clear*** tiene varias formas posibles:

- **clear** sin argumentos, ***clear*** elimina todas las variables creadas previamente (excepto las variables globales).
- **clear A, b** borra las variables indicadas.

1.1.3.4. Otras formas de definir matrices

SCILAB dispone de varias formas de definir matrices. El introducirlas por teclado sólo es práctico en casos de pequeño tamaño y cuando no hay que repetir esa operación muchas veces. Recuérdese que en SCILAB no hace falta definir el tamaño de una matriz. Las matrices toman tamaño al ser definidas y este tamaño puede ser modificado por el usuario mediante adición y/o borrado de filas y columnas. A continuación se van a ver otras formas más potentes y generales de definir y/o modificar matrices.

1.1.3.4.1 Tipos de matrices predefinidos

Existen en SCILAB varias funciones orientadas a definir con gran facilidad matrices de tipos particulares. Algunas de estas funciones son las siguientes:

- eye(2,2) forma la matriz unidad de tamaño (2x2)

```
-->eye(2,2)
 ans  =

!   1.    0. !
!   0.    1. !
```

- zeros(3,5) forma una matriz de *ceros* de tamaño (3x5)

```
-->zeros(3,5)
 ans  =

!   0.    0.    0.    0.    0. !
!   0.    0.    0.    0.    0. !
!   0.    0.    0.    0.    0. !
```

- ones(3) forma una matriz de *unos* de tamaño (3x3)

- ones(3,4) idem de tamaño (3x4)

```
-->ones(3,4)
 ans  =

!   1.    1.    1.    1. !
!   1.    1.    1.    1. !
!   1.    1.    1.    1. !
```

- linspace(x1,x2,n) genera un vector con **n** valores igualmente espaciados entre **x1** y **x2**

```
-->linspace(1,2,10)
 ans  =


         column 1 to 5

!   1.    1.1111111    1.2222222    1.3333333    1.4444444 !

         column  6 to 10

!   1.5555556    1.6666667    1.7777778    1.8888889    2. !

--> //es semejante a la definicion 1:1.125:10
```

- logspace(d1,d2,n) genera un vector con **n** valores espaciados logarítmicamente entre 10^d1 y 10^d2. Si d2 es **pi**, los puntos se generan entre 10^d1 y **pi.**
- rand(3,3) forma una matriz de números aleatorios entre 0 y 1, podemos dar diferentes posibilidades para cambiar la distribución estadística de la que partirá.
- Poly(a,x,"flag") : construye un polinomio que puede ser el conjunto de coeficientes de a, si por **flag** introducimos " coeff" y las raices del polinomio definido por a si introducimos "roots".

```
-->poly([1 2 1],'s',"roots")
 ans  =

               2   3
  - 2 + 5s - 4s + s
-->poly([1 2 1],'s',"coeff")
 ans  =

              2
    1 + 2s + s
```

1.1.3.4.2 Formación de una matriz a partir de otras

Scilab ofrece también la posibilidad de crear una matriz a partir de matrices previas ya definidas, por varios posibles caminos:

– recibiendo alguna de sus propiedades (como por ejemplo el tamaño),

– por composición de varias submatrices más pequeñas,

– modificándola de alguna forma.

A continuación se describen algunas de las funciones que crean una nueva matriz a partir de otra o de otras, comenzando por dos funciones auxiliares:

- [m,n]=size(A) devuelve el número de filas y de columnas de la matriz **A**. Si la matriz es cuadrada basta recoger el primer valor de retorno

```
-->a=[1 2 1;3 2 5;6 7 5];

-->[m,n]=size(a)

-->m = 3
 m  =

    3.
```

```
-->n = 3
 n  =

    3.
```

- n=length(x) calcula el número de elementos de un vector **x**

```
-->x=1:0.00001:2;

-->n=length(x)
 n  =

    100001.
```

- zeros(size(A)) forma una matriz de *ceros* del mismo tamaño que una matriz **A** previamente creada.

```
-->size(a)
 ans  =

!   3.    3. !

-->zeros(size(a))
 ans  =

!   0.    0. !
```

ones(size(A)) ídem con *unos*

- A=diag(x) forma una matriz diagonal **A** cuyos elementos diagonales son los elementos de un vector ya existente **x.**
- x=diag(A) forma un vector **x** a partir de los elementos de la diagonal de una matriz ya existente **A.**

```
-->diag(a)
 ans  =

!   1. !
!   2. !
!   5. !
```

- diag(diag(A)) crea una matriz diagonal a partir de la diagonal de la matriz **A.**

```
-->diag(diag(a))
 ans  =

!   1.    0.    0. !
!   0.    2.    0. !
!   0.    0.    5. !
```

- triu(A) forma una matriz triangular superior a partir de una matriz **A** (no tiene por qué ser cuadrada).

```
-->s=poly(0,'s');

-->triu([s,s;s,1])
 ans  =

!   s     s  !
!            !
!   0     1  !

-->triu([1/s,1/s;1/s,1])
 ans  =

!   1     1  !
!   -     -  !
!   z     z  !
!            !
!   0     1  !
!   -     -  !
!   1     1  !
```

- tril(A) ídem con una matriz triangular inferior.

Un caso especialmente interesante es el de crear una nueva matriz ***componiendo como submatrices*** otras matrices definidas previamente. A modo de ejemplo, vamos a realizar la matriz generadora de un código ortogonal con M=2:

```
-->a=[%f %f;%f %t];

-->a
 a  =

! F F !
! F T !
-->b=~a
 b  =

! T T !
! T F !
-->h2=[a a;a b]
 h2  =

! F F F F !
! F T F T !
! F F T T !
! F T T F !
```

1.1.3.4.3 Direccionamiento de vectores y matrices a partir de vectores

Los elementos de una matriz **a** pueden direccionarse a partir de los elementos de vectores:

```
-->a=rand(3,4)
 a  =

!   0.2113249    0.3303271    0.8497452    0.0683740 !
!   0.7560439    0.6653811    0.6857310    0.5608486 !
!   0.0002211    0.6283918    0.8782165    0.6623569 !

-->b=[1 3 6 11]
 b  =

!   1.    3.    6.    11. !

-->a(b)
 ans  =

!   0.2113249 !
!   0.0002211 !
!   0.6283918 !
!   0.5608486 !
```

Podemos ver que hemos obtenido las posiciones 1, 3, 6 y 11 de la matriz a, que debemos contar teniendo en cuenta que la matriz se recorre por columnas y no por filas.

Si queremos ver un elemento concreto de la matriz, podemos ejecutar lo siguiente:

```
-->a(3,2)
 ans  =

    0.6283918

-->a(6)
 ans  =

    0.6283918
```

Creamos esta nueva matriz **a** para que sea más fácil seguir los valores:

```
-->a=rand(4,4)*10
 a  =

!   7.2635068    2.3122372    3.0760907    3.616361  !
!   1.9851438    2.1646326    9.3296162    2.9222666 !
!   5.4425732    8.8338878    2.1460079    5.6642488 !
!   2.3207479    6.5251349    3.12642      4.826472  !
```

Ahora podemos ver las columnas 1, 2 y 3 de la 4ª fila

```
-->a(4,1:3)
 ans  =

!   2.3207479    6.5251349    3.12642 !
```

Calculamos la tercera fila

```
-->a(3,:)
 ans  =

!   5.4425732    8.8338878    2.1460079    5.6642488 !
```

Calculamos la tercera columna

```
-->a(:,3)
 ans  =

!   3.0760907 !
!   9.3296162 !
!   2.1460079 !
!   3.12642   !
```

1.1.3.4.4 Operador <<Dos Puntos>> (:)

Se trata de una de las formas de definir vectores y matrices más usado y más fácil de utilizar, dada la rápida visualización de la salida sin necesidad de ver el resultado:

```
-->x=1:1:10
 x  =

!   1.    2.    3.    4.    5.    6.    7.    8.    9.    10. !
```

En cierta forma se podría decir que el operador (:) representa un ***rango***: en este caso, los números enteros entre el 1 y el 10. Por defecto el incremento es 1, pero este operador puede también utilizarse con otros valores enteros y reales, positivos o negativos. En este caso el incremento va entre el valor inferior y el superior, pero podemos hacer que el incremento sea negativo, o que se haga con un incremento mayor o menor:

```
-->x=x(5:-1:1)
 x  =

!   6.    7.    8.    9.    10. !
```

1.1.3.4.5 Definición de matrices y vectores desde fichero

Scilab acepta el uso de scripts desde los que crear matrices, vectores, variables, etc; como si estuviéramos ejecutándolo desde la propia línea de comandos. Por ejemplo, si creamos el fichero **matriz_a.m** donde creamos una matriz a cualquiera, al ejecutarla en Scilab podremos ver que se crea como si estuvieramos generándola en el propio programa:

```
      //primer script
      a=randn(4)*10
      //fin del script
-->exec("/home/jose/eval_scilab/matriz")
a =

  -22.94025    6.55306   -2.80857   -1.33060
    5.12081    8.98726    1.81236    3.29938
   -0.69713    1.40557   25.56103    5.62650
  -17.50361   -0.63861    4.61985    1.21845
```

1.1.3.5. Operadores Relacionales

El lenguaje de programación de Scilab dispone de los siguientes operadores relacionales:

- < menor que
- > mayor que
- <= menor o igual que
- >= mayor o igual que
- == igual que
- ~= distinto que

Obsérvese que, salvo el último de ellos, coinciden con los correspondientes operadores relacionales de C. Sin embargo, ésta es una coincidencia más bien formal. En Scilab los operadores relacionales pueden aplicarse a vectores y matrices, y eso hace que

tengan un significado especial. Al igual que en C, si una comparación se cumple el resultado es *T* (***true***), mientras que si no se cumple es *F* (***false***). La diferencia con C está en que cuando los operadores relacionales de Scilab se aplican a dos matrices o vectores del mismo tamaño, *la comparación se realiza elemento a elemento*, y el resultado es otra matriz de unos y ceros del mismo tamaño, que recoge el resultado de cada comparación entre elementos.

```
-->a=1;b=2;

-->a<b
 ans  =

  T

-->a~=b
 ans  =

  T

-->a>=b
 ans  =

  F
```

1.1.3.6. Operadores Lógicos

Los operadores lógicos de Scilab son los siguientes:

- & and
- | or
- ~ negación lógica

Obsérvese que estos operadores lógicos tienen distinta notación que los correspondientes operadores de C (&&, || y !). Los operadores lógicos se combinan con los relacionales para poder comprobar el cumplimiento de condiciones múltiples.

```
-->a=%t;b=%f;

-->c=a&b
 c  =

  F

-->d=a|b
 d  =

  T
```