

Práctica 0 de Álgebra

Introducción a Scilab

14 de febrero de 2014

Índice

1. Introducción	2
2. Introducción de datos en Scilab	2
3. Ayuda	6
4. Cómo guardar y recuperar sesiones	6
5. Operaciones con escalares	7
6. Operaciones con matrices	7
6.1. Operaciones básicas	7
6.2. Otras operaciones con matrices	9
6.3. Operaciones elemento a elemento	9
6.4. Tipos especiales de matrices	9
7. Manipulación interna de matrices	10
8. Polinomios	12
9. Estructuras de decisión y bucle con Scilab	13
9.1. El condicional if	13
9.2. El bucle for	14
9.3. El bucle while	15
10. Funciones con Scilab	15
11. Fin de sesión	16

1. Introducción

El programa Scilab trabaja fundamentalmente con matrices de coeficientes reales, complejos o booleanos. En esta práctica se pretende describir el funcionamiento básico del programa y todos los aspectos básicos sobre Scilab que los alumnos han de conocer para poder realizar las prácticas de Matemática Discreta y de Álgebra.

Scilab es un programa libre. En particular, se puede utilizar y distribuir libremente y consultar y modificar el código fuente en las condiciones de su licencia. El programa se puede descargar de <http://www.scilab.org>. Estas prácticas, con ligeras modificaciones, pueden realizarse con otros programas de sintaxis similar, por ejemplo sistemas libres, como Octave, o sistemas comerciales como Matlab.

Empezaremos indicando el funcionamiento básico del programa, indicando las diversas maneras de introducir matrices en Scilab y como manipularlas. A continuación describiremos como realizar con Scilab las principales operaciones matemáticas entre matrices. Terminaremos la práctica mostrando algunas de las estructuras de decisión y de bucles más habituales en Scilab.

El programa Scilab está disponible para diversas plataformas, entre las cuáles podemos destacar GNU/Linux, Microsoft Windows y MacOSX.

A lo largo de las diferentes prácticas iremos introduciendo más operadores, comandos y funciones básicas según las vayamos necesitando y describiremos con más detalle su funcionamiento. Únicamente pretendemos dar aquí los principales operadores, comandos y funciones básicos para poderlos buscar después con más facilidad.

Los ejemplos que se muestran están desarrollados con la versión 4.1.2 de Scilab bajo el sistema operativo GNU/Linux con un procesador Intel® Core™2 Duo E8400 a 3 GHz. La ejecución de Scilab en otras plataformas o sistemas operativos pueden dar resultados ligeramente diferentes, por lo que habrá que prestar atención.

En estos boletines utilizaremos los siguientes convenios tipográficos:

- Las instrucciones que se introducen desde el teclado aparecerán en un tipo de letra mecanográfico en negrita (por ejemplo, **entrada**).
- Los resultados o la salida de Scilab aparecerán en un tipo de letra mecanográfica de grosor mediano (por ejemplo, *salida*).
- Las expresiones genéricas en la descripción de los comandos de Scilab aparecerán con letra mecanográfica cursiva (por ejemplo, si aparece *nombre=expresión*, entenderemos que *nombre* y *expresión* se han de sustituir por las cadenas adecuadas en Scilab).

2. Introducción de datos en Scilab

El programa Scilab se puede ejecutar escribiendo en una línea de comandos

scilab

o, en un entorno gráfico de ventanas, haciendo doble clic (o clic sencillo) sobre el icono correspondiente o buscándolo en el menú adecuado. Aparece una ventana con el siguiente texto:

scilab-4.1.2

Copyright (c) 1989-2007
Consortium Scilab (INRIA, ENPC)

Startup execution:
loading initial environment

-->


Si se llama al programa Scilab con la opción `-nw`, el texto aparece en el terminal, sin ventana nueva. Esta opción es adecuada si no se necesitan capacidades gráficas o sólo se dispone de un terminal de texto.

La marca `-->` nos indica que podemos introducir comandos.

Los *vectores* se introducen escribiendo ordenadamente sus componentes separadas por una coma (,) o por un espacio en blanco y cerrando toda la expresión entre corchetes ([]). Los *escalares* se introducen directamente, sin necesidad de corchetes, y, en el caso de números decimales, separando la parte entera de la decimal con un punto (.).

Por ejemplo, el escalar $a = 2$ se introduce tecleando

a=2

y al pulsar la tecla  aparece

-->**a=2**

a =

2.

-->

El vector $\vec{v} = (1, -1, 3)$ se introduce con

-->**v=[1 -1 3]**

v =

1. - 1. 3.

o con


-->**v=[1,-1,3]**

v =

1. - 1. 3.

Nota: En algunas versiones del programa aparecen signos de exclamación alrededor de las matrices.

Una matriz de tipo $m \times n$ es una colección de $m \times n$ números organizados en m filas y n columnas. Si A es una matriz, $a_{i,j}$ denota el término que ocupa la fila i y columna j de la matriz y podemos escribir que $A = (a_{i,j})$.

En Scilab, los elementos de una matriz se introducen entre corchetes [], escribiendo ordenadamente los elementos de cada fila separados por comas o por espacios en blanco y separando las filas por punto y coma (;) o el carácter  (retorno de carro). No hace falta indicar previamente el tamaño de las matrices.

Por ejemplo, la matriz

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

se puede introducir en Scilab como

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

o como

**A=[1 2 3
4 5 6
7 8 9]**

o también como

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

Al introducir alguna de estas expresiones y pulsar  se obtiene

A =

1.	2.	3.
4.	5.	6.
7.	8.	9.

Ejemplo 1. Introducir la matriz A del ejemplo anterior. La pantalla de Scilab ha de quedar aproximadamente como se muestra en la figura 1.

Las instrucciones de Scilab tienen la forma

variable=expresión

o, simplemente,

expresión

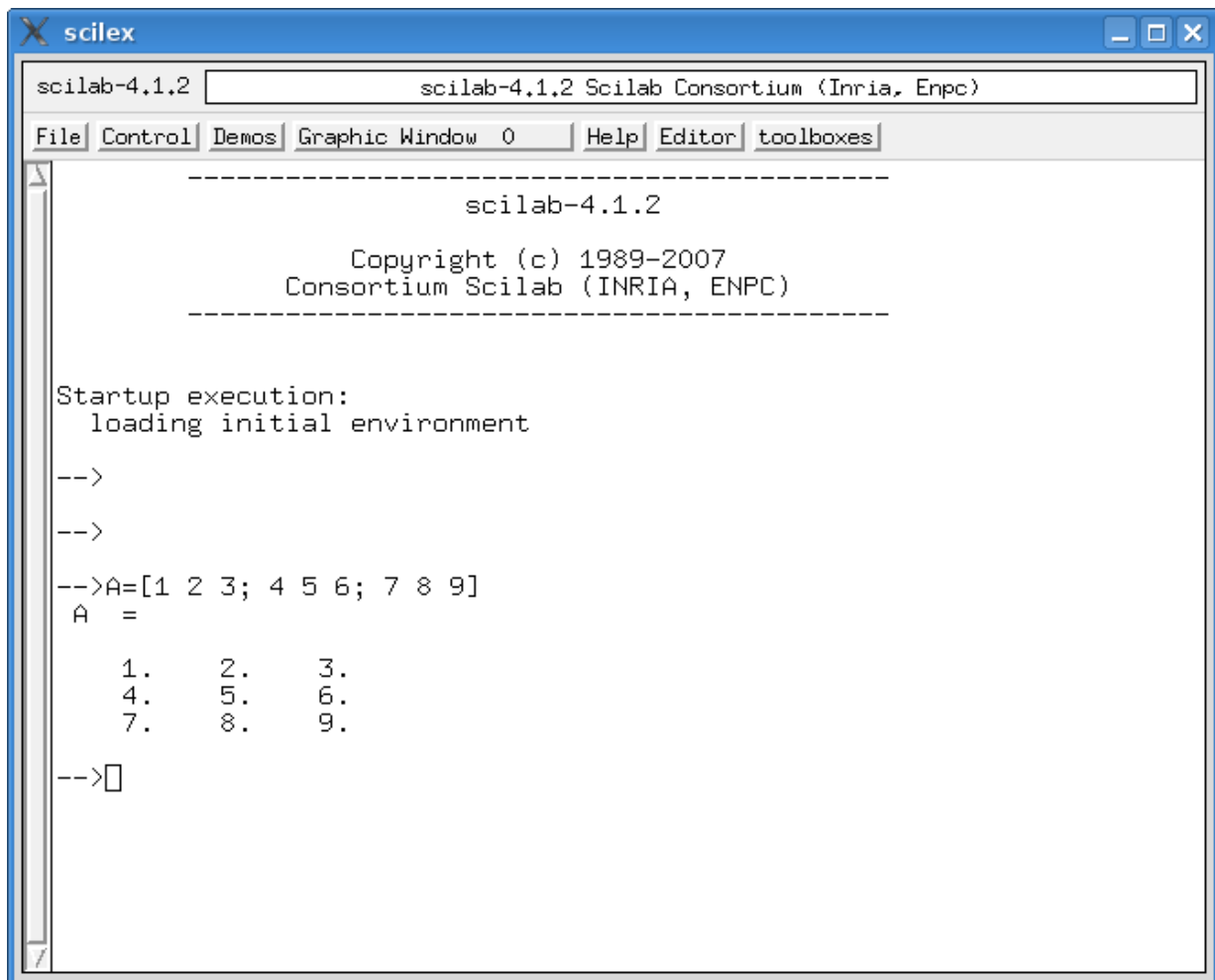



Figura 1: Introducción de una matriz a Scilab

Los nombres de las variables han de empezar por una letra, seguida de más letras o dígitos, hasta un máximo de 25 caracteres. Scilab no ejecuta las sentencias hasta que no se pulsa . Si una sentencia acaba con punto y coma (;), el resultado no se muestra en la pantalla, aunque el resultado se almacena en la memoria. Scilab distingue entre mayúsculas y minúsculas. Naturalmente, no se pueden utilizar otros comandos de Scilab como nombre de variables.

Es posible incluir diversas instrucciones en una misma línea separándolas por una coma (,), para mostrar los resultados, o por punto y coma (;), para no mostrarlos. De la misma forma, si una sentencia es muy larga y no cabe en una línea, se puede separar en diversas líneas poniendo al final de cada línea tres o más puntos (...).

Es posible recordar ordenes anteriores con las teclas de las flechas.

Si una expresión es asignada a una variable, ésta permanece almacenada como una variable de trabajo durante toda la sesión (o hasta que se le asigne otro valor o se borre con la instrucción **clear**, que borra todas las variables, o **clear** seguido del nombre de una o más variables para borrarlas). De esta forma, para recuperar esta expresión en cualquier momento,

será suficiente escribir el nombre de la variable. Si utilizamos la segunda forma de las instrucciones, el resultado se almacena en la variable `ans`. Eso nos permite recuperar el resultado de la operación inmediatamente anterior si no lo hemos asignado a otra variable.

Los comandos **who** y **whos** nos indican cuáles son las variables del entorno de trabajo.

Los comentarios se introducen precedidos de los caracteres `//` (dos barras) fuera de una cadena. Todo lo que haya detrás no se ejecuta.

3. Ayuda

Podemos encontrar ayuda sobre cualquiera de los comandos o funciones de Scilab escribiendo **help** o, si se usa Scilab en un entorno gráfico, pinchando con el ratón sobre el botón **Help**. También podemos encontrar ayuda sobre un comando específico tecleando

```
help comando
```

4. Cómo guardar y recuperar sesiones

Es posible guardar todas las variables del entorno de trabajo con el comando **save**. Por ejemplo,

```
-->save('fichero.dat')
```

guarda todas las variables en el fichero `fichero.dat`, mientras que

```
-->save('fichero.dat',A,b,c)
```

guarda las variables `A`, `b` y `c` en el fichero. Para recuperar el contenido de los ficheros, se utiliza el comando **load** con la sintaxis

```
-->load('fichero.dat','A','b','c')
```

Para guardar el contenido de una sesión de trabajo, se usa el comando **diary**:

```
-->diary('fichero.txt')
```

empieza la grabación de toda la sesión de trabajo en el fichero de texto `fichero.txt`. Para concluir la grabación utilizaremos

```
-->diary(0)
```

Si el fichero ya existe previamente se borrará. Por eso hay que prestar mucha atención al escribir el nombre del fichero. Una buena estrategia es incluir la fecha (y, si es necesario, la hora) en el nombre del fichero. Por ejemplo,

```
-->diary('md221010.txt');
```

5. Operaciones con escalares

Scilab es capaz de trabajar con escalares reales y complejos. Para trabajar con números decimales, la parte entera y la decimal se separan mediante un *punto* (.). Las operaciones suma, resta y multiplicación se hacen con los operadores **+**, **-** y *****, respectivamente. La potencia de un escalar a a un escalar p la escribiremos a^p . Las raíces cuadradas se pueden calcular también con el operador **sqrt**. Las raíces de otros índices se pueden representar como potencias de exponentes fraccionarios, como por ejemplo $2^{(1/3)}$ para $\sqrt[3]{2}$. Para la división hay dos operadores, **/** y ****. Por ejemplo, a/b quiere decir a dividido entre b , mientras que $a\b b$ quiere decir b dividido entre a .

La jerarquía de los operadores mencionados es la habitual: de mayor (el último en ejecución) a menor (el primero en ejecución) es la siguiente:

+ - * / \ ^

Para variar el orden de ejecución utilizaremos los paréntesis.

Ejemplo 2. Calculemos con Scilab el valor de la expresión $2x^3y + \sqrt{y^5z}$, con $x = 1,5$, $y = 3,7$, $z = 9$.

Primero introducimos las variables

```
-->x=1.5; y=3.7; z=9;
```

Seguidamente, introducimos la expresión:

```
-->2*x^3*y+sqrt(y^5*z)
ans =
```

```
103.97472
```

El resultado es 103,97472.

6. Operaciones con matrices

Seguidamente mostraremos como se ejecutan con Scilab las operaciones matriciales habituales y otras que serán útiles para algunas prácticas. Como los vectores no son más que matrices con una única fila, las operaciones que se describan serán también válidas para vectores.

6.1. Operaciones básicas

Suma y resta de matrices: Para sumar dos matrices utilizaremos el operador **+** y para restarlas el operador **-**. Las dos matrices han de tener el mismo tamaño (es decir, el mismo número de filas y de columnas). Si se pretende sumar o restar matrices que no tienen el mismo tamaño, Scilab nos indicará que hemos cometido un error.

Producto de un escalar por una matriz: Se realiza con el operador $*$.

Producto de matrices: Se realiza también con el operador $*$. El número de columnas de la primera matriz ha de coincidir con el número de filas de la segunda matriz. Si se pretende multiplicar dos matrices que no satisfacen esta condición, se obtiene un mensaje de error.

Potencia de una matriz: Para calcular el producto de una matriz A por ella misma n veces, utilizaremos el operador $^$. Por ejemplo, A^n . Si la matriz no es cuadrada, Scilab nos dará un mensaje de error.

Ejemplo 3. A continuación mostramos cálculos con matrices.

```
-->X=[1 2;3 -1]
X =

    1.    2.
    3.   -1.

-->Y=[0 1]
Y =

    0.    1.

-->X*Y
!--error 10
inconsistent multiplication

-->X+Y
!--error 8
inconsistent addition

-->X-Y
!--error 9
inconsistent subtraction

-->Y*X
ans =

    3.   -1.
```


6.2. Otras operaciones con matrices

Inversa de una matriz: Scilab calcula la inversa de una matriz cuadrada (si existe) con la función **inv()**. Si la matriz introducida no tiene inversa, aparecerá un mensaje de error. En las prácticas de Álgebra Lineal estudiaremos el funcionamiento detallado de esta función.

Traspuesta de una matriz: La matriz traspuesta de una matriz dada es la matriz obtenida intercambiando filas por columnas. Scilab la calcula con el operador apóstrofo (**'**), por ejemplo, **A'**.

6.3. Operaciones elemento a elemento

En algunas ocasiones puede ser útil realizar operaciones con matrices elemento a elemento que no corresponden a ninguna operación matemática entre matrices. Mostraremos a continuación alguna de estas operaciones.

Suma de un escalar a todos los elementos de una matriz: Se ejecuta poniendo el operador **+** entre la matriz y el escalar. Por ejemplo, **A+1** es la matriz que se obtiene cuando sumamos 1 a cada elemento de la matriz **A**.

Producto y cociente elemento a elemento: Para multiplicar elemento a elemento se usa el operador **«.*»**. Por ejemplo, **A.*B**. Hace falta que las matrices sean del mismo tamaño para poder hacer esta operación. Análogamente podemos calcular cocientes de matrices elemento a elemento con el operador **«./»**.¹

Potencia elemento a elemento: Para elevar a un número todos los elementos de una matriz se utiliza el operador **«. ^ »**. La matriz puede ser de cualquier tamaño y el exponente, cualquier número.

6.4. Tipos especiales de matrices

En Scilab hay implementadas unas cuantas matrices especiales. Por ejemplo, **zeros(m,n)** es la matriz nula de m filas y n columnas, **ones(m,n)** es la matriz de m filas y n columnas formada toda por unos y **eye(m,n)** es la matriz de tamaño m filas y n columnas que tiene unos en la diagonal y ceros en el resto de las posiciones (cuando $m=n$, es la matriz identidad de orden n , I_n). Análogamente, si *mat* es una matriz, **zeros(mat)**, **ones(mat)** y **eye(mat)** devuelven las matrices de las mismas dimensiones que *mat* formadas, respectivamente, por zeros, unos, y unos en la diagonal y ceros en el resto de posiciones.

¹Este último operador se puede utilizar, por ejemplo, para dividir un número entre todos los elementos de una matriz. En este caso, si el primer número es un entero, puede ser necesario separarlo del operador con un espacio (por ejemplo, **1 ./A**) o ponerle el punto decimal (**1. ./A**) porque **1./A** se interpretaría como el cociente entre el número real **1.** y la matriz **A**.

7. Manipulación interna de matrices

Supongamos que tenemos introducida una matriz A en nuestra sesión de Scilab y queremos cambiar el elemento que ocupa la fila i y la columna j por el valor *expresión*. Entonces utilizaremos la orden

$$A(i, j) = \text{expresión}$$

Esta acción modifica la matriz A . Por tanto, si no queremos perder la matriz A original podemos copiarla a otra matriz, haciendo por ejemplo

--> **B=A**

y efectuando después la modificación a la matriz **B**, por ejemplo,

--> **B(2,3)=10**

Si los índices son más grandes que el tamaño de la matriz, Scilab añade filas o columnas suficientes para acomodar los nuevos elementos. También conviene tener en cuenta que Scilab identifica las matrices 1×1 con los escalares.

Scilab permite introducir matrices definidas *por bloques*. Supongamos que A i B son matrices con el mismo número de filas, podemos construir la matriz

$$C = (A \ B)$$

obtenida poniendo las columnas de B a la derecha de las de A . Esta matriz se introduce en Scilab escribiendo

--> **[A, B]**

o

--> **[A B]**

Análogamente, si A y B son dos matrices con el mismo número de columnas, podemos construir la matriz

$$D = \begin{pmatrix} A \\ B \end{pmatrix}$$

poniendo las filas de B debajo de las filas de A . Esta matriz se introduciría en la forma

--> **[A; B]**

o como

--> **[A**
--> **B]**

De la misma manera, podemos añadir a una matriz A $m \times n$ una nueva fila escribiendo

```
-->[A; [a1 a2 ... an]]
```

y una nueva columna escribiendo

```
-->[A,[a1;a2; ...; am]]
```

Esta manera de trabajar por bloques es válida para cualquier número de matrices siempre que los números de filas y de columnas sean compatibles.

Ejemplo 4. Consideremos las matrices

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad B = \begin{pmatrix} 7 & 8 & 2 \\ 4 & 1 & 6 \\ 0 & 0 & 1 \end{pmatrix}.$$

```
-->A=[1 2 3
-->   4 5 6
-->   7 8 9]
A =
```

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

```
-->B=[7 8 2
-->   4 1 6
-->   0 0 1]
B =
```

```
7.    8.    2.
4.    1.    6.
0.    0.    1.
```

```
-->C=[A,B]
C =
```

```
1.    2.    3.    7.    8.    2.
4.    5.    6.    4.    1.    6.
7.    8.    9.    0.    0.    1.
```

```
-->D=[A;B]
D =
```

```
1.    2.    3.
4.    5.    6.
```

7.	8.	9.
7.	8.	2.
4.	1.	6.
0.	0.	1.

Una vez introducida una matriz *matriz*, es posible trabajar con submatrices extraídas de ella, usando expresiones como las siguientes:

- *matriz(i,j)* para el elemento que ocupa la posición fila *i* y columna *j* de *matriz*,
- *matriz(:,j)* para la columna *j* de *matriz*,
- *matriz(i,:)* para la fila *i* de *matriz*,
- *matriz(r:s,:)* para la submatriz formada por las filas entre la *r* y la *s* de *matriz*,
- *matriz(:,r:s)* para la submatriz formada por las columnas entre la *r* y la *s* de *matriz*,
- *matriz([r s],:)* para la submatriz formada por las filas *r* y *s* de *matriz*, y
- *matriz(:,[r s])* para la submatriz formada por las columnas *r* y *s* de *matriz*.

Observemos que estas expresiones también nos permiten hacer modificaciones en una matriz que ya ha sido introducida. Así, por ejemplo, si queremos modificar la fila *i* de *matriz*, podemos escribir *matriz(i,:)=nueva fila* y el programa nos devolverá la matriz *matriz* modificada. Hay que tener en cuenta que las dimensiones de la submatriz y la nueva matriz han de ser compatibles. Por otra parte, podemos asignar a todos los elementos de una submatriz un mismo elemento con la misma sintaxis.

Como antes, hay que tomar precauciones si no queremos perder la matriz inicial.

8. Polinomios

Para definir una variable polinómica en la indeterminada *s*, podemos definir la variable *s* en la forma

```
-->s=poly(0,"s")
s =
```

s

```
-->p=3+4*s+s^2
p =
```

3 + 4*s* + *s*²

Las raíces de un polinomio se calculan con la función **roots**. Para el polinomio **p** que hemos introducido tendríamos

```
-->roots(p)
ans =

- 1.
- 3.
```

Para evaluar un polinomio en un valor concreto utilizaremos la función **horner**. Por ejemplo

```
-->horner(p,2)
ans =

15.
```

En las prácticas de Álgebra Lineal trabajaremos con polinomios.

9. Estructuras de decisión y bucle con Scilab

Scilab es un lenguaje de programación en el que es posible utilizar instrucciones condicionales y ejecutar diversos bucles. Para referencia futura describimos algunas de estas instrucciones. Todas estas instrucciones acaban con la palabra reservada **end**.

9.1. El condicional **if**

La instrucción **if** se utiliza para evaluar una expresión lógica y ejecutar una serie de instrucciones si la expresión es verdadera.

La sintaxis es:

```
if condición then instrucciones
elseif condición then instrucciones
...
else instrucciones
end
```

donde las partes **elseif** y **else** son opcionales, y puede haber más de una cláusula **elseif**. La palabra **then** puede ser sustituida por un retorno de carro o una coma y *ha de encontrarse siempre en la misma línea que el correspondiente comando **if** o **elseif***.

Si la condición de la línea con **if** es verdadera, se ejecutan las instrucciones siguientes. En otro caso, se ejecuta la primera de las series de instrucciones **elseif** para las cuales la condición sea verdadera, y, si ninguna de ellas es verdadera, se ejecutan las instrucciones de la cláusula **else**.

Presentamos un ejemplo sencillo.

```
-->x=-4; if x<0 then r=-x,elseif x==0 then r=0,else r=x,end
r =
```

4.

```
-->x=5; if x<0 then r=-x,elseif x==0 then r=0,else r=x,end
r =
```

5.

Scilab también dispone del comando **select/case** que puede ser una alternativa útil cuando una expresión sólo puede tomar un pequeño número de valores interesantes. Para más información, teclear **help select** en Scilab.

9.2. El bucle for

El bucle for se utiliza para ejecutar unas instrucciones para todas las columnas de una matriz. Su sintaxis es

```
for variable=expresión do instrucción, ... instrucción, end
```

Aquí *expresión* suele ser una matriz y las órdenes se ejecutan para todas las columnas de la matriz. Los formatos habituales para esta expresión son

```
comienzo: paso : fin
```

o

```
comienzo: fin
```

Ejemplo 5. Con el siguiente ejemplo podemos escribir los cuadrados de los números impares entre 1 y 9:

```
-->for x=1:2:9 do [x,x^2],end
ans =
```

```
1.    1.
ans =
```

```
3.    9.
ans =
```

```
5.   25.
ans =
```

```

7.      49.
ans =

```

```

9.      81.

```

Con la siguiente instrucción calculamos la suma de los cuadrados de los 10 primeros números naturales. Usar punto y coma en vez de coma evita la escritura de todos los valores intermedios.

```

-->suma=0;for x=1:10 do suma=suma+x^2;end;suma
suma =

385.

```

9.3. El bucle **while**

Este bucle se utiliza para ejecutar unas instrucciones mientras una cierta expresión booleana sea verdadera. Su sintaxis es

```

while condición do instrucciones,...[,else instrucciones], end

```

La palabra **do** puede sustituirse por **then**, por una coma o por un salto de línea. La palabra **do** o **then** tienen que encontrarse en la misma línea que **while**. La cláusula **else** se ejecuta cuando *condición* deja de ser verdadera.

10. Funciones con Scilab

Ya hemos visto algunas funciones como **zeros** o **ones**, que permiten construir objetos de Scilab a partir de sus parámetros o argumentos. La sintaxis de una función de Scilab es

```

función(arg1, arg2...)

```

para dar un resultado, o

```

[var1, var2...]=función(arg1, arg2...)

```

para funciones que devuelven diversos resultados, asignados a las variables *var1*, *var2*...

Para definir una función en Scilab podemos usar las palabras clave **function** i **endfunction**. Su sintaxis es

```

function argssalida=nombrefunción(argsentrada)
    instrucciones
endfunction

```

donde *argsentrada* es una lista de identificadores de variables que se asignarán ordenadamente a los parámetros y *argssalida* es un identificador o una lista de identificadores de variables separados por comas i cerrados entre corchetes. La lista de instrucciones ha de contener asignaciones de estas variables para calcular el resultado.

Por ejemplo, la siguiente función calcula el cuadrado de un número dado:

```
-->function y=cuadrado(x)
-->y=x*x
-->endfunction

-->z=cuadrado(3)
z =

    9.
```

Como vemos, el argumento de salida es **y** y el de entrada es **x**. El valor de la función es el valor de **y** cuando acaba la definición de la función después de sustituir el parámetro **x** por el argumento utilizado.

La siguiente función admite dos parámetros, **x** e **y**, y devuelve dos resultados, que corresponden a la suma y a la diferencia de sus argumentos:

```
-->function [suma, dif]=sumadif(x,y)
-->    suma=x+y
-->    dif=x-y
-->endfunction

-->[m,n]=sumadif(5,3)
m =

    2.

n =

    8.
```

Observemos que la variable **m** queda asignada a la suma de los dos números y la variable **n** queda asignada a la diferencia entre los dos números.

Suele ser interesante almacenar una o varias funciones en un fichero de texto, con el fin de poderlas usar en diversas sesiones. Para ejecutar todas las instrucciones (no solo definiciones de funciones) de un fichero de texto *fichero.sci*, podemos usar el comando **exec**:

```
-->exec('fichero.sci')
```

La versión gráfica del programa Scilab incluye un editor de ficheros, al que se puede acceder desde el menú, que permite la ejecución del código que se seleccione.

11. Fin de sesión

Para terminar la sesión de Scilab se introduce la instrucción **exit**. Si se trabaja en un entorno de ventanas, también se puede salir cerrando la ventana de Scilab.