1.- Introducción a la programación en Scilab

1.- Introducción a Scilab

Scilab (Scientific Laboratory) es un paquete de herramientas de *software libre*, inspirado en **Matlab** y desarrollado por INRIA (Institut National de Recherche en Informatique et en Automatique) y la ENPC (École Nationale des Ponts et Chaussées) desde 1990. En 2003 fue creado el Scilab Consortium dentro de la fundación Digiteo y actualmente Scilab es desarrollado por Scilab Enterprises desde julio 2012.

Scilab fue diseñado para hacer cálculos numéricos, especialmente cálculos basados en matrices y álgebra lineal, análisis y visualización de datos y para facilitar la escritura de nuevos programas dentro de este tipo de objetivos. **Scilab** también ofrece la posibilidad de hacer algunos cálculos simbólicos como derivadas de funciones polinomiales y racionales. Posee cientos de funciones matemáticas y la posibilidad de integrar programas en los lenguajes más usados (Fortran, Java, C y C++). La ventaja de Scilab es que combina funciones matemáticas y gráficas con un poderoso lenguaje interpretado de alto nivel. Scilab viene con numerosas herramientas: gráficos 2D y 3D, animación, álgebra lineal, matrices dispersas, polinomios y funciones racionales, Simulación: programas de resolución de sistemas de ecuaciones diferenciales (explícitas e implícitas).

Iniciando Scilab.- Scilab para Windows puede iniciarse seleccionándolo desde el icono de Windows. Al ejecutarse Scilab crea una ventana como la mostrada en la figura 1.1. La ventana principal es la denominada **Consola** o **Ventana de Comandos** que muestra un mensaje inicial y termina con un apuntador consistente en los signos "-->" seguidos del cursor parpadeante indicando que Scilab está listo para recibir comandos en esa línea (Línea de comandos).

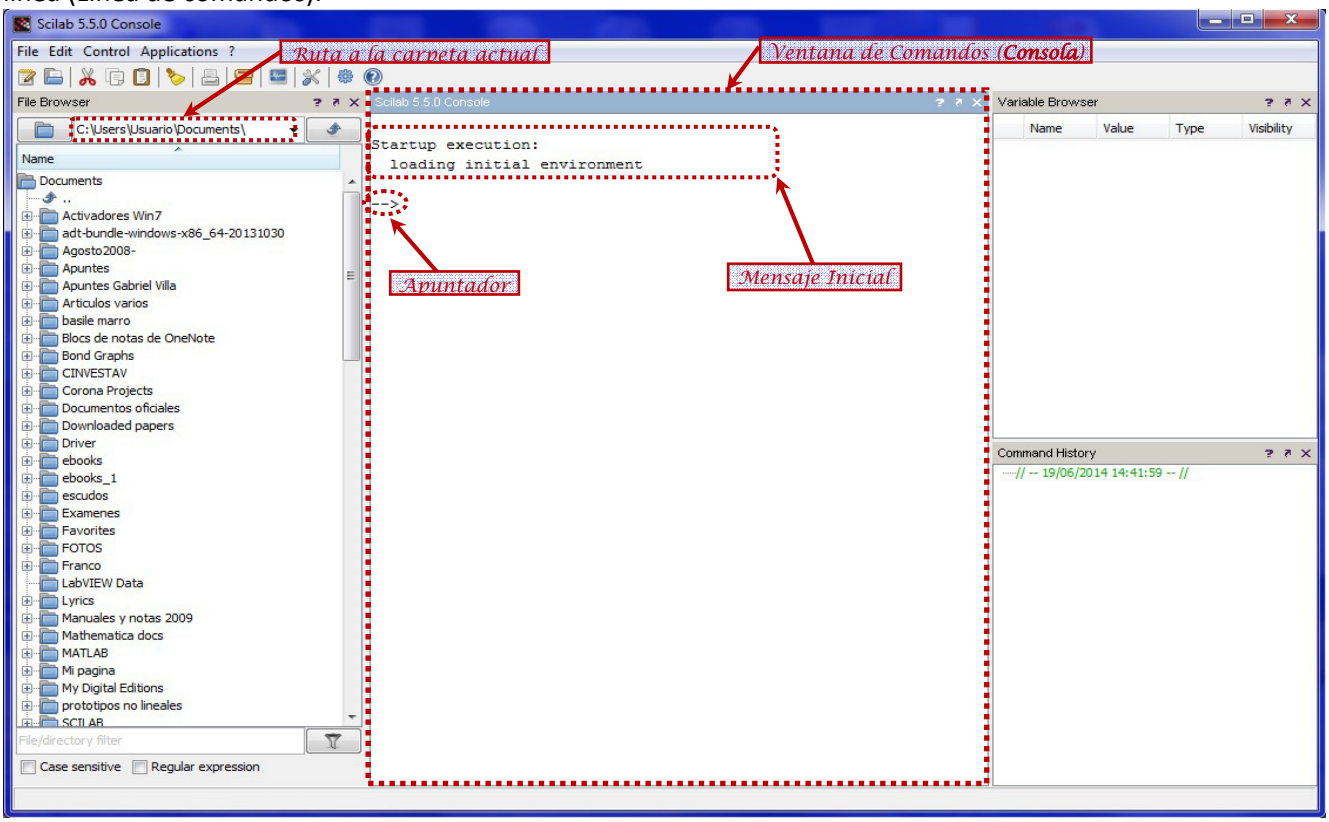

Figura 1.1.- Ventana principal y consola de Scilab

Además de la ventana de comandos, en la figura 1.1 se muestran también otras ventanas que aparecen por default la primera vez que se ejecuta Scilab:

- **File Browser.-** Explorador de archivos, visualiza los archivos guardados en la carpeta actual y permite navegar por las diferentes carptetas del sistema.
- **Variable Browser.-** Explorador de las variables que se van creando o las que se van importando.
- **Command History.-** Visualiza y permite re-ejecutar las instrucciones introducidas en la línea de comandos.

1.1.-USO DE SCILAB COMO UNA CALCULADORA.

La manera más sencilla de utilizar Scilab es introduciendo en la línea de comandos las operaciones que se desea realizar y el resultado se obtiene inmediatamente al terminar el comando con la tecla [enter] en la variable `ans`.

Ejemplo:

```
(4+8)/2
ans =
     6.
```

Ejemplo: lo mismo, pero usando una variable

```
x=4+8;
x/2
ans =
     6.
```

1.2.- OPERACIONES ARITMÉTICAS BÁSICAS:

De la misma forma que una calculadora, Scilab ofrece las siguientes operaciones aritméticas básicas:

Operación	Símbolo	Ejemplo	Precedencia
Suma	+	5 + 3	3
Resta	-	23 - 12	3
multiplicación	*	20.5 *1.5	2
división	/ o \	56/8 = 8\56	2
Potencia	^	5^2	1

1.2.1.- PRECEDENCIA DE LAS OPERACIONES BÁSICAS:

Las expresiones se evalúan de izquierda a derecha, con la operación de potencia teniendo la orden de precedencia más alta, seguida por la multiplicación y división que tienen ambas igual precedencia y seguidas finalmente por suma y resta que tienen ambas igual precedencia.

☞ Siempre se pueden emplear **paréntesis** para alterar esta precedencia de acuerdo al orden en que el usuario quiera evaluar una expresión.

Ejemplo: Para evaluar la expresión $\frac{8\times5}{7\times4}$ sin usar paréntesis se puede realizar como sigue:

```
8*5/7/4
ans =
    1.4285714
```

o también:

```
8/7*5/4
ans =
    1.4285714
```

1.3.- FORMATO DE VISUALIZACIÓN DE NÚMEROS Y SU REPRESENTACIÓN INTERNA.

La visualización de las cantidades numéricas se puede obtener en varios formatos. Los formatos de visualización de números no cambian la representación interna de un número, solamente su visualización. El formato por default es `format('v',10)`. Los formatos de visualización disponibles son:

Comando	Visualización de 1/6	Comentario
`format('v',n)`	`0.166...67`	Despliega n-2 dígitos (dos caracteres son para el punto y el signo). Por default n=10.
`Format('e',n)`	`1.66...67D-01`	Despliega n-6 dígitos (6 caracteres son para el punto, el signo, la letra D, el signo del exponente y dos dígitos de exponente). Por default n=10.

1.4.- CONSTANTES PREDEFINIDAS EN SCILAB.

Scilab proporciona valores predefinidos de algunas constantes que son usuales en los cálculos matemáticos. En la siguiente tabla se muestran las principales.

Variable	Valor predefinido	Comentario
`%i`	$\sqrt{-1}$	Unidad imaginaria (permite definir números complejos)
`%pi`	$\pi = 4\tan^{-1}(1)$	180^{o} en radianes
`%e`	`exp(1)`	Base de los logaritmos naturales
`%f, %F`	Valor lógico **Falso**	
`%t, %T`	Valor lógico **Verdadero**	
`%eps`	2^{-52}	Número positivo más pequeño sumable con un entero en Scilab
`%inf`	∞	Resultado Infinito, por ejemplo: 1/0
`%nan`	"No es un Número"	Operación no definida, por ejemplo: 0/0, inf-inf

Las constantes de la tabla anterior no pueden ser redefinidas.

Ejemplo: La siguiente expresión calcula el área de un círculo de radio 1.5:

```
%pi*1.5^2
ans=
   7.0685835
```

Ejemplo: se pueden hacer operaciones aritméticas con números complejos, por ejemplo

```
(1+%i)*(2-%i)
ans =
  3. + i
```

Ejemplo: Los números más pequeños que %eps se desprecian al sumarse con un entero:

```
format('v',20)
1+%eps
ans =
  1.00000000000000022
```

```
1+%eps/2
ans =
  1.
```

1.5.- MANEJO DE MATRICES Y VECTORES.

Las Matrices son el principal tipo de datos que maneja Scilab. La manera más sencilla de escribir una matriz es como una lista de elementos. Solamente hay que seguir unas convenciones básicas:

- Toda matriz es una arreglo rectangular de elementos y está organizada por:
  - renglones o filas (en forma horizontal)
  - columnas (en forma vertical).
- Los elementos de una fila se escriben separados con espacios o comas.
- Se Usa ; (punto y coma) para indicar el fin de cada renglón (fila).
- Se encierra la lista entera de elementos con paréntesis cuadrados, [ ].

Ejemplo: Para introducir la siguiente matriz de dos filas y dos columnas: $A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$:

```
 A=[ 1 2; 3 4]
A =
   1. 2.
   3. 4.
```

Una matriz de una sola fila o de una sola columna se denomina ***vector***:

Ejemplo: Introducir el vector fila $b=\begin{bmatrix}5 & 6\end{bmatrix}$ y el vector columna $c=\begin{bmatrix}7\\ 8\end{bmatrix}$

```
 b=[ 5 6]
b =
   5. 6.
 c=[ 7;8]
c =
   7.
   8.
```

Para evitar el despliegue de los resultados en la pantalla cada vez que se asigna una variable use el operador ";" al final de la expresión, para el ejemplo anterior:

```
 b=[ 5 6];
 c=[ 7;8];
```

1.5.1.- CONCATENANDO MATRICES:

Se pueden juntar dos matrices con el mismo número de columnas o con el mismo número de renglones para formar una matriz más grande:

Ejemplo:

```
 A=[1 2 3; 4 5 6;7 8 9];
 R=[10 11 12];
 B=[A ; R]          //Junta dos matrices de tres columnas
B =
     1.     2.     3.
     4.     5.     6.
     7.     8.     9.
    10.    11.    12.
 C=[10; 11; 12];
```

```
 B=[A C]            //Junta dos matrices de tres filas
B =
     1.     2.     3.    10.
     4.     5.     6.    11.
     7.     8.     9.    12.
```

1.5.2.- SUBMATRICES:

Si `A` es una matriz, la notación `A(i:j,m:n)` se refiere a la submatriz formada por los elementos desde la fila `i` hasta la fila `j` y desde la columna `m` hasta la columna `n`.

☞ Obsérvese que i, j, m, n sólo pueden ser números enteros mayores a cero y menores al tamaño correspondiente de la matriz.

Ejemplo: Si B es la matriz del ejemplo anterior

```
 C=B(2:3,2:3)
C =
     5.     6.
     8.     9.
```

1.5.3.- MANIPULACIÓN DE MATRICES ELEMENTO A ELEMENTO:

Si `A` es una matriz, la notación `A(i,j)` hace referencia al elemento individual de la fila `i` y la columna `j`. Esto permite leer o modificar cada elemento de matriz en base a su posición renglón columna especificados por los índices `i, j`

El símbolo dos puntos ":" permite referirse a un rango de valores enteros, lo cual puede ser usado para referirse a un rango de filas o de columnas.

Ejemplo: Generar todos los números enteros del 1 al 20, con incrementos de 2 en 2

```
x=1:2:10
x =
     1.     3.     5.     7.     9.
```

Si el incremento no se especifica, se toma por default 1:

```
 x=1:10
x =
     1.     2.     3.     4.     5.     6.     7.     8.     9.    10.
```

Usado como índice, la notación de dos puntos permite omitir no solamente el incremento, sino el inicio y el final:

Ejemplo: Partiendo del ejemplo anterior

```
 x(1:4) //los elementos del 1 al 4
ans =
     1.     2.     3.     4.
 x(:)  //los elementos desde el primero hasta el último en forma de columna
ans =
     1.
     2.
     3.
     4.
     5.
     6.
     7.
     8.
     9.
    10.
```

Es decir, el uso de dos puntos para la designación de filas y columnas implica, respectivamente, todas las filas o columnas;

Ejemplo: `A(:,3)` representa todas las filas en la columna tres, `A(2,:)` representa todas las columnas en la fila dos

```
 A=[1 2 3;4 5 6;7 8 9]
A =
     1.     2.     3.
     4.     5.     6.
     7.     8.     9.
 A(:,2)
ans =
     2.
     5.
     8.
 A(2,:)
ans =
     4.     5.     6.
```

☞ **Observaciones**:

- Usar sólo dos puntos, por ejemplo `A(:)`, reagrupa todos los elementos de una matriz en un vector columna.
- Colocar datos fuera del rango actual de una matriz rellena con ceros las zonas no especificadas, manteniendo la forma rectangular de la matriz.
- Asignar las filas o columnas de una matriz igual a la matriz vacía `[ ]` elimina estas filas o columnas.

Ejemplo: Prosiguiendo con la matriz del ejemplo anterior

```
 A(:)  // Convierte a columna
ans =
     1.
     4.
     7.
     2.
     5.
     8.
     3.
     6.
     9.
 A(2,5)=10 // Agrega un elemento fuera de rango
A =
     1.     2.     3.     0.     0.
     4.     5.     6.     0.    10.
     7.     8.     9.     0.     0.
 A(:,4:5)=[] // Elimina las columnas 4 y 5
A =
     1.     2.     3.
     4.     5.     6.
     7.     8.     9.
 B=A(:)'   // Convierte a renglón
B =
     1.     4.     7.     2.     5.     8.     3.     6.     9.
 C = (B>5)    // Genera un vector de valores lógicos
```

EJEMPLOS VARIOS:

```
 A=[1 2 3; 4 5 6; 7 8 9]   //Crea la matriz A
 A(3,3)=0                  //Cambia un elemento de la matriz
 A(2,6)=1;                 //Agrega un elemento fuera de rango
 A=[1 2 3; 4 5 6; 7 8 9]   //Vuelve a crear la matriz A
 B=A(3:-1:1,:)             //Crea la matriz B con las filas A en orden inverso
```

```
C=[A B(:,[1 3])]          //Añade la 1ª y 3ª columna de B a la derecha de A
B=A(1:2,2:3)              //Extrae submatriz de A
B=A(:)                    //Convierte A en vector columna
B=B'                      //transpone la columna para convertirla a fila
B(:,2)=[]                 //Elimina la segunda columna de B
B=B'                      //Transpone una matriz
A=B                       //Copia la matriz B en A
B(2,:)=[]                 //Elimina la segunda fila de B
x=-3:0.5:3                //Crea un vector desde -3 a 3 con incrementos de 0.5
```

1.5.4.- MATRICES CON ELEMENTOS COMPLEJOS.

Scilab permite el manejo de números complejos como ya se dijo, mediante el uso de la variable predefinida: `%i`=$\sqrt{-1}$.

Ejemplo: Para introducir la siguiente matriz con elementos complejos: $A=\begin{bmatrix} 1+i & 2-3i \\ 3-3i & 4+2i \end{bmatrix}$, se puede proceder como sigue:

```
A=[1+%i 2-%i*3; 3-%i*3 4+%i*2];
```

ó bien, separando en parte real y parte imaginaria:

```
A=[1 2; 3 4]+%i*[1 -3; -3 2];
```

1.5.5.- EL CONJUGADO Y EL TRASPUESTO DE UNA MATRIZ.

El operador comilla (') obtiene el traspuesto conjugado de una matriz, si se desea solamente trasponer la matriz se debe usar punto comilla (.') o bien, la función `conj`.

Ejemplo: Partiendo del ejemplo anterior

```
 A'    // Obtiene la matriz traspuesta y conjugada
ans =
   1. - i      3. + 3.i
   2. + 3.i    4. - 2.i
 A.'    // Obtiene la matriz traspuesta sin conjugar
ans =
   1. + i      3. - 3.i
   2. - 3.i    4. + 2.i
 conj(A')    // Misma operación que A.'
ans =
   1. + i      3. - 3.i
   2. - 3.i    4. + 2.i
```

1.5.6.- OPERACIONES ARITMÉTICAS CON MATRICES.

Las operaciones aritméticas básicas: suma, resta, multiplicación, e inclusive división se pueden realizar entre matrices de dimensiones compatibles mediante los operadores +, -, *, /.

Además de las operaciones aritméticas básicas entre matrices anteriores, Scilab dispone de **operaciones especiales elemento a elemento**, las cuales se enumeran en la siguiente tabla, en la cual se supone que `c` es un escalar arbitrario y las matrices `A` y `B` siguientes:

$$A=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ & & \dots & \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix},\ B=\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ & & \dots & \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix}$$

Operaciones elemento a elemento con matrices	
Suma con escalar `A+c = c+A`	$A+c=\begin{bmatrix} a_{11}+c & a_{12}+c & \dots & a_{1m}+c \\ a_{21}+c & a_{22}+c & \dots & a_{2m}+c \\ & & \dots & \\ a_{n1}+c & a_{n2}+c & \dots & a_{nm}+c \end{bmatrix}$

Multiplicación por escalar
`A*c = c*A =`
`A.*c = c.*A`
	$A*c=\begin{bmatrix} a_{11}*c & a_{12}*c & \dots & a_{1m}*c \\ a_{21}*c & a_{22}*c & \dots & a_{2m}*c \\ & & \dots & \\ a_{n1}*c & a_{n2}*c & \dots & a_{nm}*c \end{bmatrix}$

Multiplicación elemento a elemento `A.*B = B.*A`	$A.*B=\begin{bmatrix} a_{11}*b_{11} & a_{12}*b_{12} & \dots & a_{1m}*b_{1m} \\ a_{21}*b_{21} & a_{22}*b_{22} & \dots & a_{2m}*b_{2m} \\ & & \dots & \\ a_{n1}*b_{n1} & a_{n2}*b_{n2} & \dots & a_{nm}*b_{nm} \end{bmatrix}$
División elemento a elemento `A./B`	$A./B=\begin{bmatrix} a_{11}/b_{11} & a_{12}/b_{12} & \dots & a_{1m}/b_{1m} \\ a_{21}/b_{21} & a_{22}/b_{22} & \dots & a_{2m}/b_{2m} \\ & & \dots & \\ a_{n1}/b_{n1} & a_{n2}/b_{n2} & \dots & a_{nm}/b_{nm} \end{bmatrix}$

Elevación a Potencia elemento a elemento
`A.^c`
`c.^A`
`A.^B`
	$A.^c=\begin{bmatrix} a_{11}^{\,c} & a_{12}^{\,c} & \dots & a_{1m}^{\,c} \\ a_{21}^{\,c} & a_{22}^{\,c} & \dots & a_{2m}^{\,c} \\ & & \dots & \\ a_{n1}^{\,c} & a_{n2}^{\,c} & \dots & a_{nm}^{\,c} \end{bmatrix}$ $c.^A=\begin{bmatrix} c^{a_{11}} & c^{a_{12}} & \dots & c^{a_{1m}} \\ c^{a_{21}} & c^{a_{22}} & \dots & c^{a_{2m}} \\ & & \dots & \\ c^{a_{n1}} & c^{a_{n2}} & \dots & c^{a_{nm}} \end{bmatrix}$ $A.^B=\begin{bmatrix} a_{11}^{b_{11}} & a_{12}^{b_{12}} & \dots & a_{1m}^{b_{1m}} \\ a_{21}^{b_{21}} & a_{22}^{b_{22}} & \dots & a_{2m}^{b_{2m}} \\ & & \dots & \\ a_{n1}^{b_{n1}} & a_{n2}^{b_{n2}} & \dots & a_{nm}^{b_{nm}} \end{bmatrix}$

Ejemplo: Encontrar la solución del sistema de tres ecuaciones con tres incógnitas siguiente

$$x_1 + x_2 + x_3 = 1$$
$$x_2 + x_3 = 2$$
$$x_3 = 3$$

```
 A=[1 1 1;0 1 1;0 0 1];    //se define la matriz del sistema
 b=[1 2 3]';         //se define el vector columna de términos independientes
 x=A^-1 *b                 //se obtiene la solución
x =
    -1.
    -1.
     3.
 x=inv(A)*b                //inv(A) es lo mismo que A^-1
x =
    -1.
    -1.
     3.
```

Ejemplos varios: Continuando con el ejemplo anterior

```
 A^-1         // Obsérvese la diferencia del operador ^
 A.^-1        // y el operador .^
```

```
 A^3          // Calcula el producto matricial A*A*A
 A.^3         // Eleva al cubo cada elemento de A
 B=A*A'       // Obtiene una nueva matriz simétrica
 A*B          // Obsérvese la diferencia entre la multiplicación matricial *
 A.*B         // y la multiplicación elemento a elemento .*
```

Ejemplos de Errores comunes: Continuando con el ejemplo anterior se pueden intentar las siguientes operaciones que fallarán por no respetar las dimensiones de las matrices y su compatibilidad con las operaciones:

```
 b^2          // Intento de multiplicar un vector columna por si misma
Warning: Syntax "vector ^ scalar" is obsolete. It will be removed in Scilab 6.0.
  Use "vector .^ scalar" instead.
b*A           /// Intento de multiplicar matrices de dimensiones incompatibles
 !--error 10
Multiplicación inconsistente.Inner matrix dimensions must agree.

 A(3,4)       /// Intento de acceder a un elemento fuera de las dimensiones de A
     !--error 21
Índice inválido.
  A/0         /// Obsérvese que a diferencia de Matlab, dividir entre cero SI es ERROR
   !--error 27
División por cero...
```

1.5.7.- EL ESPACIO DE TRABAJO DE SCILAB:

Se denomina espacio de trabajo al conjunto de variables creadas por el usuario en cada sesión. El espacio de trabajo está vacío al inicio de cada sesión, o después de un comando `clear`. Se puede obtener información de las variables en el espacio de trabajo visualizando el navegador de variables (Variable Browser) como se muestra en la figura 1.3.

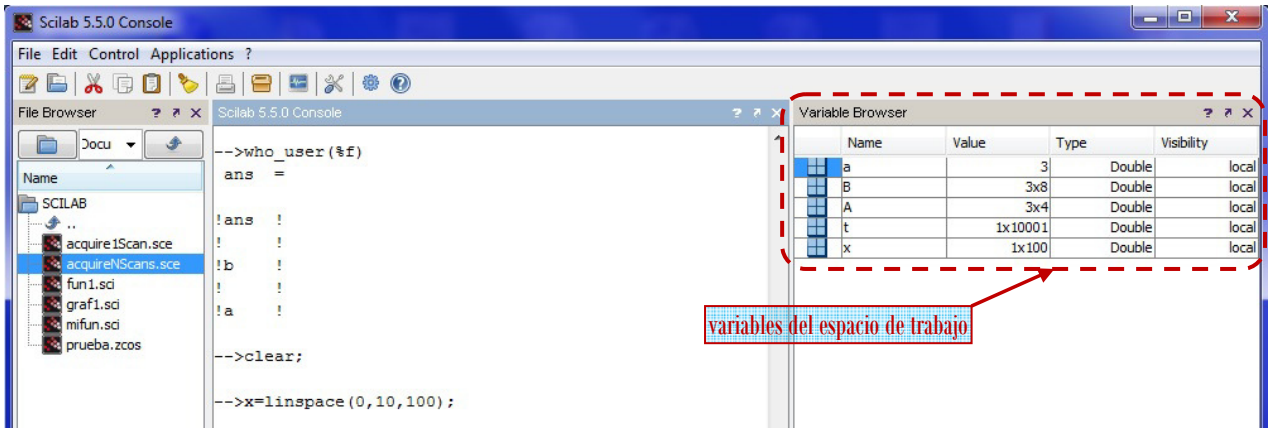

Figura 1.2.- navegador de variables mostrando el espacio de trabajo de Scilab.

O si se prefiere, se puede también utilizar el comando `who_user` o `who`; por ejemplo:

Ejemplo:

```
clear         // Limpia el espacio de trabajo
who_user      // obsérvese que el espacio de trabajo está vacío
ans  =
     []
A=rand(3,3)  // Genera una matriz al azar de 3 x 3
A =
```

```
    0.1454259    0.2165825    0.5196690
    0.4306040    0.9683006    0.8674864
    0.6140478    0.6637877    0.2877042
[n,m]=size(A)
n =
     3.
m =
     3.
who_user
User variables are:
 n         m         A         ans

 Using 20 elements out of 9990166
```

2.- PROGRAMACIÓN EN SCILAB.

En la sección anterior se ha usado a **Scilab** como una calculadora que simplemente ejecuta los comandos conforme se van introduciendo manualmente en la ventana de comandos. A continuación se verá que **Scilab** también puede utilizarse como un poderoso **lenguaje de programación**.

Programas en Scilab

Un programa en Scilab es simplemente un archivo que contiene instrucciones de Scilab que se ejecutarán en una cierta secuencia planeada. Los archivos de instrucciones de Scilab pueden llevar cualquier nombre o extensión, pero se prefiere usar las extensiones **`sce`** o **`sci`**. Hay dos tipos de programas en Scilab:

- **Scripts**: Son archivos que contienen comandos de Scilab y que operan sobre las variables del espacio de trabajo.
- **Funciones**: Son archivos que contienen comandos de Scilab, pero que operan sobre variables de entrada para producir variables de salida.

Los archivos-sci o archivos-sce se pueden generar y editar como cualquier archivo de texto, sin embargo, conviene editarlos con el editor integrado en el ambiente de desarrollo de Scilab, denominado **'SciNotes'**, ya que éste proporciona ayudas visuales, tales como resaltar con colores distintos los diferentes tipos de texto introducido (comandos, comentarios, variables, etc.) Se puede ejecutar este editor haciendo click sobre el icono de la barra de herramientas de Scilab mostrado en la figura 2.1

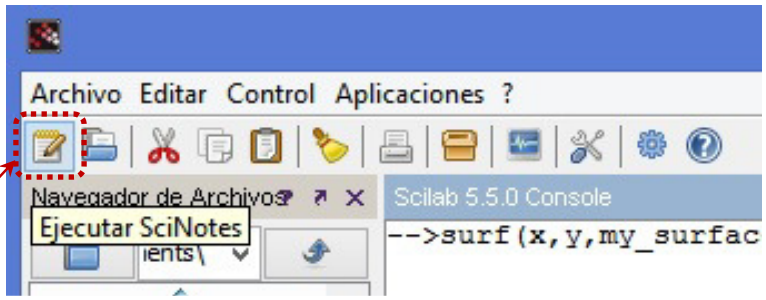

Figura 2.1.- Icono para ejecutar el editor SciNotes.

Otra manera de invocar el editor cuando ya se tiene almacenado un archivo es directamente tecleando en la ventana de comandos:

```
edit('nombre_de_archivo')
```

☞ Si se desea empezar a escribir una **función** nueva se teclea simplemente `edit`.

2.1.- Control de flujo del programa.

Un programa consta de dos tipos de instrucciones:

- Instrucciones secuenciales.
- Instrucciones de control del flujo de ejecución

2.1.1.- Instrucciones secuenciales.

Las instrucciones secuenciales son todas las instrucciones vistas hasta este punto que al ser incluidas en un programa se ejecutan una al terminar la otra en el orden en que fueron escritas, de acuerdo al diagrama de flujo de la figura 2.2.

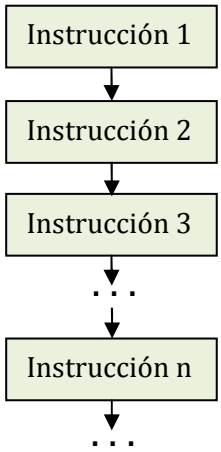

Figura 2.2.- Flujo de ejecución de instrucciones secuenciales.

Es decir, las instrucciones secuenciales no pueden alterar el orden o el número de veces de la ejecución de las instrucciones.

2.1.2.- Instrucciones de control del flujo ejecución del programa

Scilab posee instrucciones orientadas a controlar la secuencia (o flujo) de ejecución de las instrucciones de un programa. Estas son de dos tipos: **Instrucciones condicionales o instrucciones de repetición**.

Para realizar el control de flujo del programa Scilab tiene que evaluar **expresiones lógicas** para decidir entre diferentes caminos de ejecución de las instrucciones de un programa.

Expresiones lógicas.

Scilab permite construir expresiones de valor lógico falso (F) o verdadero (T) mediante diversas operaciones lógicas y relacionales (de comparación) que producen este tipo de resultados. Estas operaciones se agrupan en la siguiente tabla, en donde también se incluyen operadores aritméticos tratados en la sección anterior para aclarar su precedencia.

tipo	Operador		Descripción	Precedencia
Arimético	( )		Agrupación por paréntesis	1
	.'	'	Transpuesto y conjugado traspuesto	2
	. ^	^	Potencia y potencia matricial	
	+	−	Signo positivo y signo negativo	3
Lógico	~		Negación lógica	
	OR		OR lógico por renglón, columna o total	
	AND		AND lógico por renglón, columna o total	
Aritmético	.*	*	Multiplicación y multiplicación matricial	4
	./ ó .\	/ ó \	División elemento a elemento y matricial	
	+	−	Suma y resta	5
Especial	:		Operador dos puntos	6
Relacional	<	<=	Menor que y menor o igual que	7
	>	>=	Mayor que y mayor o igual que	
	==	~=	Igual a y diferente a	
Lógico	&		AND lógico elemento a elemento	8
	|		OR lógico elemento a elemento	9

- La **precedencia** define cual operación se realiza primero en una expresión que contiene varios de estos operadores. Nuevamente, una expresión siempre se evalúa de izquierda a derecha cuando las operaciones involucradas tienen la misma precedencia.
- Un valor aritmético es convertido a valor lógico cuando aparece como operador de una operación lógica, de acuerdo a la siguiente equivalencia:

Valor aritmético cero = valor lógico falso (F)
Valor aritmético diferente de cero= valor lógico verdadero (T)

Ejemplo 3.1. La expresión relacional $(1 \le x \le 3)$ ó $(5 \le x \le 7)$ en Scilab se escribe como: x>=1 & x<=3 | x>=5 & x<=7 . Obsérvese que no es necesario el uso de paréntesis ya que el operador & tiene mayor precedencia que el operador |. A continuación se muestra la evaluación de esta expresión para dos valores diferentes de x:

```
x=4; x>=1 & x<=3 | x>=5 & x<=7
ans =     F
x=6; x>=1 & x<=3 | x>=5 & x<=7
ans =     T
```

Ejemplo3.2. En cambio, la expresión relacional $(x \le 4)$ y $(x \le 1$ ó $x \ge 3)$ en Scilab se escribe como sigue: x<=4 & (x<=1|x>=3) y en este caso los paréntesis **sí** son necesarios, de lo contrario se estaría evaluando la expresión $(x \le 4$ y $x \le 1)$ ó $(x \ge 3)$. A continuación se muestra la evaluación de estas dos expresiones (con y sin paréntesis) para dos valores diferentes de x:

```
x=0; x<=4 & (x<=1|x>=3)
ans =     T
x=0; x<=4 & x<=1|x>=3
ans =     T
x=5; x<=4 & (x<=1|x>=3)
```

```
ans =     F
x=5; x<=4 & x<=1|x>=3
ans =     T
```

Las instrucciones de control de flujo forman bloques de instrucciones llamadas **estructuras**, las diferentes estructuras que se pueden crear en Scilab se resumen en la siguiente tabla.

Estructura de control de flujo	Descripción
for x = arreglo comandos **end**	Un ciclo o bucle `for` ejecuta repetidamente los comandos, asignando a la variable x una columna del arreglo en cada iteración, y repite desde la primera hasta la última columna del arreglo.
while expresión comandos **end**	Un ciclo o bucle `while` ejecuta los comandos mientras la expresión (lógica) tiene valor verdadero o distinto de cero.

if expresión
  comandos ejecutados si
  expresión es verdadera
else
  comandos ejecutados si
  expresión es falsa
end
	Una simple estructura `if-else-end` permite seleccionar entre dos bloques de comandos a ejecutar dependiendo del resultado de una expresión lógica.

if expresión1
  comandos ejecutados si la
  expresión1 es verdadera
elseif expresión2
  comandos ejecutados si la
  expresión2 es verdadera
elseif expresión3
...
else
  comandos ejecutados si todas las
  expresiones1,2,3,..., son falsas
end
	La estructura más general `if-elseif-end`. permite seleccionar entre la ejeccución de diferentes bloques de comandos dependiendo de los resultados de diferentes expresiones lógicas.

select expresión
  case valor1 **then**
  instrucciones_1
  case valor2 **then**
  instrucciones_2
  ...
  else
  otras_instrucciones
end
	La estructura `select-case-else-end`. permite seleccionar entre diferentes bloques de comandos dependiendo de los diferentes posibles valores de una sola expresión (no necesariamente lógica).

break	Termina la ejecución de bucles *for* y *while*.

2.1.3.- Programación mediante Scripts.

Los **scripts** y las **funciones** son archivos `.sci` o `.sce` y deben tener en cuenta los siguientes puntos:

- Estos archivos se crean en un editor de texto o en el editor de Scilab.
- El nombre de la función y el nombre del archivo deben de ser idénticos. Por ejemplo la función `linspace` se almacena en un archivo denominado `linspace.sci`.
- Para ejecutar un script o una función se deberá usar el comando **`exec`**, asegurándose primero de que el archivo esté guardado en el directorio actual.

- El comando `pwd` muestra la ruta del directorio actual y el comando `dir` muestra los archivos que contiene dicho directorio.
- El comando `cd nombre_de_directorio` permite cambiarse de directorio actual.

Como ya se mencionó, los scripts se ejecutan para procesar variables del espacio de trabajo, por lo tanto no tienen propiamente variables de entrada o de salida. En los siguientes ejemplos se ilustra como un script puede trabajar sobre una variable del espacio de trabajo, modificarla y producir un resultado.

Ejemplo 3.3: El siguiente script permite invertir el orden de las columnas del array x.

```
//Este script se guarda en el archivo invierte.sce
[m,n]=size(x);    //Obtiene n=número de columnas de x
y=zeros(m,n);     //Inicializa vector y del mismo tamaño que x
for i=1:n
     y(:,i)=x(:,n+1-i); //copia elementos de x en y en orden inverso
end
x=y    //reemplaza x por y
```

En la línea de comandos:

```
x=[1 2 3 4 5 6];
exec('invierte.sce',0)  //La opción 0 permite desplegar los valores
                        //previstos en el script.
x =
    6.    5.    4.    3.    2.    1.
exec('invierte.sce',0)
x  =
    1.    2.    3.    4.    5.    6.
```

☞ Los ciclos deben evitarse siempre que haya una función predefinida de Scilab equivalente que permita resolver el problema, ya que al ser Scilab un lenguaje interpretado el tiempo invertido en ejecutar un ciclo crecerá con el número de iteraciones que contiene.

Ejemplo 3.4: El ciclo del ejemplo anterior se puede reemplazar por una sola instrucción:

```
x=[1 2 3 4 5 6];
x(:,1:6)=x(:,6:-1:1) //Una sola instrucción
x =
     6     5     4     3     2     1
```

Para comparar la eficiencia de los dos métodos usamos los comandos **`tic`** y **`toc`** que permiten arrancar y parar la medición del tiempo de ejecución. Para notar mejor la diferencia usamos un vector de 100,000 elementos:

```
tic; x=[1:100000]; exec('invierte.sce',-1); toc  //Usando el ciclo for
ans  =
    0.203
tic; x=[1:100000];x(:,1:100000)=x(:,100000:-1:1); toc  //Sin usar el ciclo for
ans  =
    0.015
```

Ejemplo 3.5: El siguiente script localiza las coordenadas del punto máximo de la función $f(x)=(x-1)(x-2)(x-3)$ en el intervalo $1 \le x \le 3$:

```
//Este script se guarda en el archivo maxim.sce
x=1:0.0001:3;     //intervalo de búsqueda
n=length(x);
```

```
ymax=-%inf; //Inicializa valor máximo
for i=1:n
    y=(x(i)-1)*(x(i)-2)*(x(i)-3);   //evalúa la función en el valor x(i)
    if y>ymax
      ymax=y;     //Si encuentra un valor mayor, reemplaza ymax
    end
end
ymax  //Despliega resultado
```

En la línea de comandos:

```
exec('maxim.sce',0)
ans  =
    0.3849002
```

Ejemplo 3.6: El ciclo anterior se puede evitar de la siguiente manera:

```
x=1:0.0001:3;     //intervalo de búsqueda
y=(x-1).*(x-2).*(x-3);  //Obtiene un vector con los valores de la función
ymax=max(y) //Usa la función max de Scilab
ymax  =
    0.3849002
```

2.1.4.- Programación mediante Funciones.

Las **funciones** en Scilab se pueden definir de dos maneras:

- **Funciones definidas "*inline*"**, es decir, sin usar un archivo para almacenar el código de la función. Esto resulta práctico para funciones de pocas líneas.
- **Funciones definidas en archivos `.sci`**. Esta opción resulta la más adecuada cuando el código de la función requiere muchas líneas.

Ejemplo 3.7. Se puede definir una función *inline* para graficar una función medianamente complicada como sigue:

```
function y=mi_funcion(x), y=10*sin(x)./(x.^2+1), endfunction //define la función.
//Obsérvese que el nombre de x y de y son irrelevantes.
t=-10:0.1:10; //vector de tiempo
y=mi_funcion(t); //llama a la función para evaluarla en t.
y1=mi_funcion(t-4); //evalúa la función retardada 4 unidades de t.
plot(t,y,'b',t,y1,'r'); xgrid;
```

A diferencia de los scripts, **las funciones** de Scilab poseen las siguientes características:

- Todas las variables dentro de una función son locales a la función y por lo tanto se aíslan del espacio de trabajo de Scilab. Las únicas conexiones entre las variables locales de la función y el espacio de trabajo de Scilab son las variables de **entrada** y **salida**.
- Si una función cambia el valor de cualquier variable de entrada, los cambios aparecen sólo dentro de la función y no afectan a las variables en el espacio de trabajo de Scilab.
- Cuando una función tiene más de una variable de salida, éstas se encierran entre corchetes, por ejemplo `[n,m] = size(A)`.
- La función **`input`** detiene la ejecución y espera la entrada de un dato desde el teclado.
- El comando **`pause`** causa que se detenga la ejecución envía un nuevo apuntador, por ejemplo: `-1->`, el

cual indica que se tiene acceso a otro espacio de trabajo en donde son accesibles las variables de la función. Para regresar a la ejecución de la función y al espacio de trabajo original se deberá teclear el comando **`return`**.

Ventajas y desventajas de Funciones Vs. Scripts.
Normalmente se utilizan funciones y no scripts para hacer programas en Scilab, ya que al usar variables locales las funciones protegen las variables del espacio de trabajo y evitan alterar variables en forma accidental.

Sin embargo, los scripts se consideran ventajosos para definir arreglos de una gran cantidad de valores, especialmente cuando estos datos son resultados experimentales, se pueden capturar de uno en uno y guardarlos en un archivo externo como valores de una arreglo matricial en un script.

Ejemplo 3.8: el ejemplo del script que invierte las columnas de un arreglo matricial se puede escribir como una función de la siguiente manera:

```
function A=finv(B)
//  A=finv(B) produce una matriz A cuyas columnas son las mismas
//  que la matriz B, pero en orden invertido

[m,n]=size(B);        //Obtiene n=número de columnas de x
A=zeros(m,n);   //inicializa matriz A
A(:,1:n)=B(:,n:-1:1); //Invierte el orden de columnas de B y lo copia en A
endfunction
```

En la línea de comandos:

```
exec ('finv.sci',0) //Carga la función del archivo finv.sci
x=[1 2 3 4 5 6];
y=finv(x)
y =
     6.     5.     4.     3.     2.     1.
```

A lo largo del curso se realizarán diversas **funciones** y no scripts para implementar los diferentes métodos numéricos.