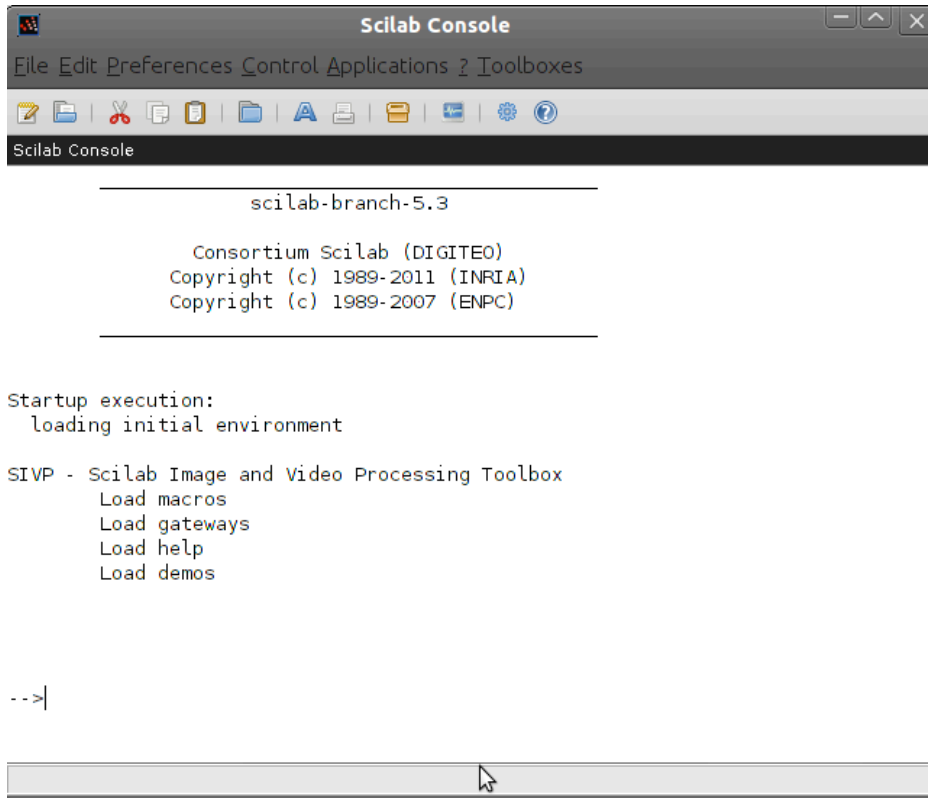


## Introducción al Scilab.

No cualquier cosa que escribamos en una computadora puede ser interpretado. Probablemente no nos responda nada si le preguntamos cuánto es  $1+1$ . Sin embargo, con un intérprete podemos hacer preguntas y dar órdenes que, escritas en el lenguaje adecuado, la computadora puede responder y realizar. El **Scilab** es un intérprete de un lenguaje de programación. Es muy parecido a una calculadora con mucha memoria y más facilidades.

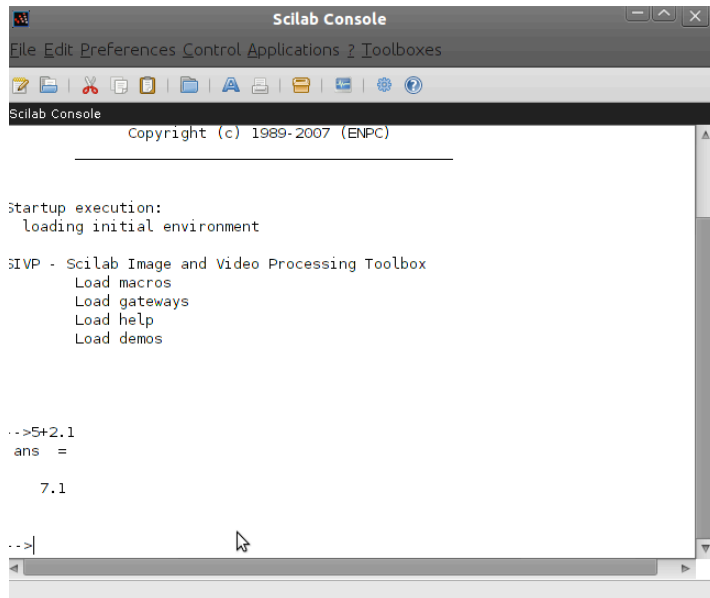


**Figura 1.** Consola de Scilab.

En la figura 1 se muestra la ventana que se abre cuando ejecutamos el programa Scilab. Llamaremos **consola** a esta ventana. Además de un mensaje donde se muestra la versión del programa (en este caso la 5.3), aparecen otros relacionados con los módulos del programa que se cargan durante el inicio. Por el momento no les prestaremos atención. Después del renglón que dice "Load demos" vemos un "prompt" o símbolo de sistema que en este caso es: --> . Una vez que aparece el símbolo de sistema, el programa está listo para recibir instrucciones.

Comencemos con una operación simple. Antes de hacerlo, debemos notar que, como todo lenguaje de programación, el idioma base es el inglés por lo que, tanto los nombres de las funciones conocidas como la notación que se usa para los números, corresponden al inglés. Esto significa, por ejemplo, que en lugar de escribir 1,5 (el número que se obtiene de dividir 3 por 2) se debe usar 1.5 o que la función trigonométrica "seno" se escribe sin en lugar de sen. Con esto en mente, supongamos que queremos sumar 5 y 2,1. Escribimos entonces en la consola: 5+2.1 y luego presionemos enter. Como se muestra en la figura 2, el programa responde con dos líneas la primera dice "ans =" (ans corresponde a la palabra en inglés

“answer”, es decir, respuesta) y la segunda el resultado de la operación “ 7.1”. Más abajo vuelve a aparecer el símbolo de sistema indicando que el programa está listo para seguir recibiendo instrucciones. Si queremos escribir números muy grandes o muy chicos deberemos recurrir a la notación “científica”. Por ejemplo, el número  $2 \cdot 10^{20}$  se escribe 2e20 o bien 2d20. Lo mismo sucede con los números muy chicos. El número  $2 \cdot 10^{-20}$  se escribe 2e-20 o bien 2d-20.



**Figura 2.** Utilización de la consola para efectuar cálculos sencillos.

Además de sumar podemos hacer otras operaciones entre números. Los siguientes ejemplos ilustran los símbolos que se usan en operaciones simples entre números:

Producto:  $2 \times 3 \rightarrow 2 * 3$ ; División:  $2 : 3 \rightarrow 2 / 3$ ; Potenciación:  $2^3 \rightarrow 2 ^ 3$ ; Raíz cuadrada:  $\sqrt{2} \rightarrow \text{sqrt}(2)$

El símbolo de la raíz cuadrada verifica lo que dijimos antes en relación al inglés: sqrt corresponde a “square root” (raíz cuadrada). Las operaciones se pueden combinar.

*Ejercicio.* Hacer los cálculos que se listan a continuación. Practique después hacer otros cálculos que involucren operaciones sencillos entre números.

$$5 \times (3^2 + 10:5)^{0.5} =$$

$$(\sqrt[3]{27} - \frac{5}{4}) =$$

$$\frac{5 \times 2^2}{4 - 3 \cdot 2} \times \left(14 - \frac{3}{5}\right)^4 =$$

Además de estas operaciones sencillas, el Scilab tiene definidas muchas funciones matemáticas conocidas, como las trigonométricas, el logaritmo (neperiano y en base 10) o la exponencial. En el caso de las trigonométricas, el argumento está siempre en radianes. Sus nombres son:

Trigonómicas: sin, cos, tan y sus inversas, asin, acos, atan.

Logaritmo y exponencial: log, log10, exp

Como en el caso de la raíz cuadrada, la notación es: sin(3) para calcular el seno del ángulo de 3 radianes.

*Ejercicio.* Hacer los cálculos que se listan a continuación. Practique después hacer otros cálculos que involucren el uso de funciones ya definidas en Scilab.

$$10 \ln(2) =$$

$$10 \log(10) =$$

$$e^2 =$$

$$\sin(0) =$$

$$\cos(3,14 \times 2^2) =$$

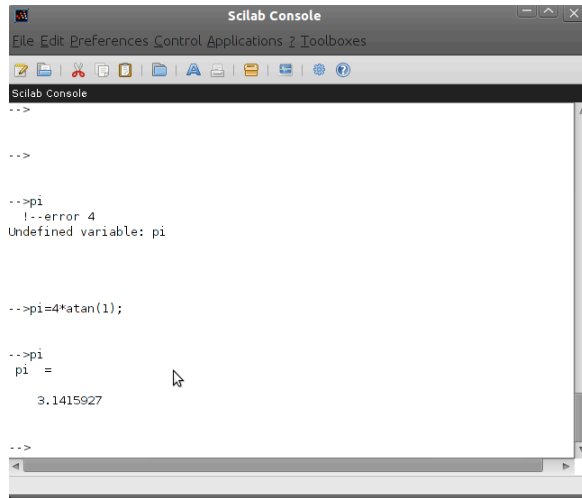
## Variables y asignación de valores.

En muchas ocasiones necesitamos almacenar el resultado de una operación para su uso posterior. En estos casos es útil crear una *variable* y asignarle a esta variable el número en cuestión (puede también asignarse a una variable otros tipos de datos pero nos concentraremos en números por el momento).

Supongamos por ejemplo que necesitamos el valor del número  $\pi$  porque efectuaremos cálculos trigonométricos. Si escribimos pi y apretamos enter el programa nos devuelve un mensaje de error porque la variable pi no ha sido definida con anterioridad. Tenemos entonces que indicarle a la computadora que pi para nosotros vale 3.1415927..... Existen muchas formas de hacerlo una de ellas es aprovechar el hecho de que el resultado del arcotangente de 1 es  $\pi/4$  y entonces podemos escribir

$$\text{pi} = 4 * \text{atan}(1);$$

Si lo hacemos y luego apretamos enter, habremos hecho dos cosas, primero creamos una variable que se llama  $\pi$  y segundo le hemos asignado a esa variable el valor del número  $\pi$ . Finalmente, nótese que escribimos un `;` al final de la igualdad (o *sentencia*). Al escribir el `;` el Scilab no nos devuelve ninguna respuesta inmediatamente. Si omitimos el `;` sí lo hace (probar introduciendo la definición de los dos modos). De todas maneras, en ambos casos, la variable  $\pi$  queda igualmente definida, lo que podemos comprobar si escribimos  $\pi$  y apretamos enter.



```
Scilab Console
File Edit Preferences Control Applications 2 Toolboxes

Scilab Console
-->

-->

-->pi
!--error 4
Undefined variable: pi

-->pi=4*atan(1);

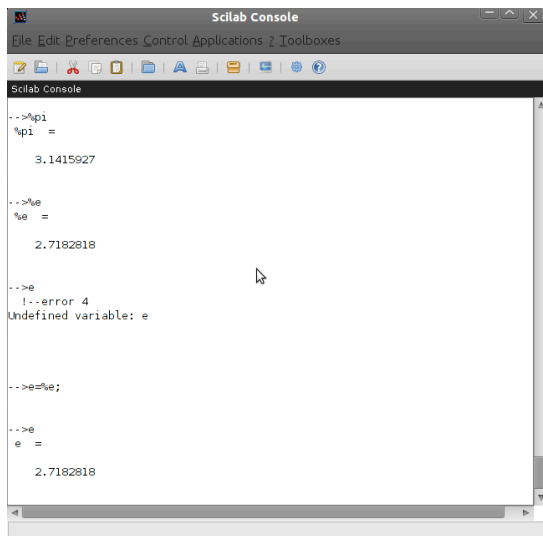
-->pi
pi =
    3.1415927

-->
```

**Figura 3.** La variable  $\pi$  no existe en el programa, necesitamos definirla y asignarle su valor. Para ello utilizamos la función arco tangente (`atan`) y asignamos el valor `4*atan(1)` a la variable  $\pi$ . Ahora la variable  $\pi$  contiene el valor del número  $\pi$ .

Por supuesto que la elección del nombre  $\pi$  para almacenar el valor del número  $\pi$  fue absolutamente arbitraria, y la asignación `x=4*atan(1);` es tan válida como la anterior. La única diferencia es que ahora el valor del número  $\pi$  se almacenó en una variable que se llama  $x$ .

Algunos números ya se encuentran definidos en Scilab. Por ejemplo el número  $\pi$  está guardado en una variable que se llama `%pi`, el número de Neper “ $e$ ” es `%e`. En la figura 4 se muestra otro ejemplo de asignación. Para evitar escribir el símbolo de porcentaje cada vez que nos referamos al número de Neper, podemos asignar su valor a una nueva variable, por ejemplo,  $e$ . Para ello utilizamos la instrucción `e=%e;`.



**Figura 4.** Algunos números ya están definidos.

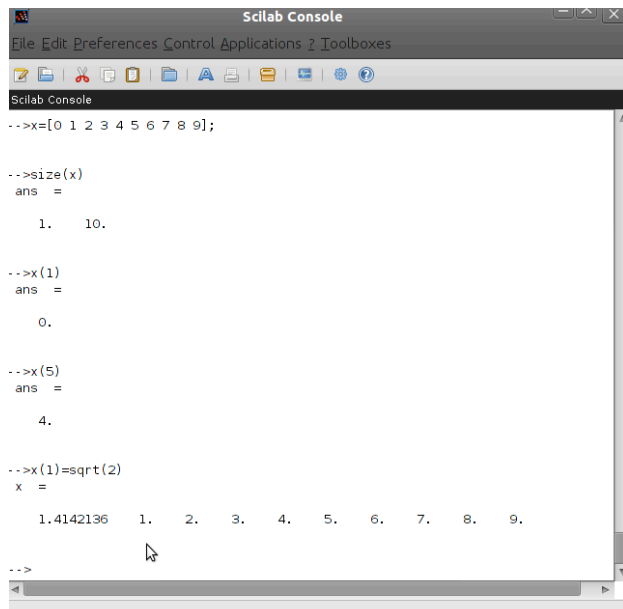
Cuando se asigna un valor a una variable a la que ya se le había asignado otro valor anteriormente, el programa olvida el valor anterior y se queda con el último asignado. Si queremos saber el valor que tiene una variable escribimos su nombre y apretamos enter.

*Ejercicio.* Efectúe cálculos utilizando el número de Neper y  $\pi$  y asegúrese de que son los números que usted conoce.

## Arreglos de números (vectores y matrices)

Hasta ahora hemos utilizado un solo tipo de datos: números. Los arreglos de números, a los que usualmente llamamos vectores, contienen muchos números agrupados bajo una misma variable.

Por ejemplo "x=[7 5 3 -2 23]" guarda en una variable "x" cinco números el "7", el "5", el "3", el "-2" y el "23". Escribamos "x=[7 5 3 -2 23];" y luego presionemos enter. Hemos creado una variable que se llama "x" que es un arreglo de cinco números (o un vector con cinco componentes). Si escribimos "x" el Scilab nos devuelve "7 5 3 -2 23". Si después hacemos x=[0 1 2 3 4 5 6 7 8 9]; y apretamos enter, ahora x corresponde a un arreglo o vector de 10 componentes (el programa "se olvida" del valor que le habíamos dado antes a x al volver a asignarle un valor). Para acceder a los elementos de un arreglo necesitamos el nombre del arreglo (en este caso x) y la posición del elemento. Ejemplo escribamos a continuación "x(1)" y presionemos enter. La respuesta es 0 que es el primer elemento del vector "x". "x(5)" enter, devuelve 4 porque es el quinto elemento del vector. La sentencia "x(1)=sqrt(2)" asigna el valor de raíz de 2 al primer elemento del vector.

The image shows a screenshot of the Scilab Console window. The window has a menu bar with 'File', 'Edit', 'Preferences', 'Control Applications', and 'Toolboxes'. Below the menu bar is a toolbar with various icons. The main area of the window displays the following commands and their outputs:

```
-->x=[0 1 2 3 4 5 6 7 8 9];

-->size(x)
ans =

    1.    10.

-->x(1)
ans =

    0.

-->x(5)
ans =

    4.

-->x(1)=sqrt(2)
x =

    1.4142136    1.    2.    3.    4.    5.    6.    7.    8.    9.

-->
```

**Figura 5.** Operaciones básicas con vectores.

Existen diversas maneras de generar un vector. Una es introduciendo a mano sus elementos como hicimos antes. Para ello sólo tenemos que escribir "nombre=[elem1 elem2 .....]" . Es decir nombre del vector, el signo = y entre corchetes (y separados por espacios) los elementos del vector. Cuando el vector que queremos definir sigue cierta regla hay formas más sencillas de definirlo y no necesitamos escribir todos los elementos. Si necesitamos un vector cuyo primer elemento sea el 0, cuyo último elemento sea el 1 y que tenga elementos equiespaciados (que difieran en el valor 0.1) hacemos: "x=0:0.1:1;" . Esta sentencia guarda en "x" el vector [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1]. La lógica es "inicio:paso:fin". Si no ponemos el paso, el Scilab usa un paso de 1. Por ejemplo la sentencia "x=-10:10" genera un vector de nombre x y que contiene los números entre el -10 y el 10 incrementándose de a uno.

*Ejercicio.* Defina un vector que comience en -3 y vaya hasta el 2003 con un paso de 5. ¿Cuántos elementos tiene el vector?

```

Scilab Console
File Edit Preferences Control Applications ? Toolboxes

-->x=-10:10
x =

      column 1 to 12
- 10. - 9. - 8. - 7. - 6. - 5. - 4. - 3. - 2. - 1.  0.  1.

      column 13 to 21
  2.  3.  4.  5.  6.  7.  8.  9. 10.

-->x=-1:0.1:1
x =

      column 1 to 10
- 1. - 0.9 - 0.8 - 0.7 - 0.6 - 0.5 - 0.4 - 0.3 - 0.2 - 0.1

      column 11 to 20
  0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

      column 21
  1.

```

**Figura 6.** Definiciones de arreglos de números.

Así como se definen vectores, también pueden definirse matrices. Por ejemplo, la sentencia `a=[1 2 3 ; 4 5 6]` define una matriz cuya primera fila es 1 2 3 y la segunda es 4 5 6.

Podemos calcular la **traspuesta** de una matriz o de un vector con el comando `'`. Por ejemplo, para `a=[1 2 3 ; 4 5 6]` (de 2x3), tenemos que

```

a' ↵
1.    4.
2.    5.
3.    6.

```

que es ahora una matriz de 3x2. De ahora en más usaremos el símbolo ↵ para indicar que debe presionarse la tecla enter.

El comando **length** calcula el tamaño de un vector. Por ejemplo, si `x=[1 2 4 4]` entonces `length(x)` ↵

```

4 ↵

```

El comando **size** calcula el tamaño de una matriz. Por ejemplo, si `A=[1 2 4;3 3 2; 2 2 1;5 6 7]` entonces,

```

size(A) ↵
4. 3. ↵

```

O sea, 4 filas y 3 columnas.

Si se calcula `size` de un vector devolverá por ejemplo

```

size(x) ↵
1.4. ↵

```

En este caso por ser un vector fila (una fila y cuatro columnas).

Si lo trasponemos devolverá,

```

size(x') ↵

```

4.1. ↴  
(4 filas y 1 columna).

El comando **zeros** crea una matriz o vector de ceros. Por ejemplo,

```
zeros(2,3) ↴  
0.    0.    0.  
0.    0.    0. ↴
```

Que es una matriz de 2x3 llena de ceros.

El comando **ones** crea una matriz o vector de unos. Por ejemplo,

```
ones(5,4) ↴  
1.    1.    1.    1.  
1.    1.    1.    1.  
1.    1.    1.    1.  
1.    1.    1.    1.  
1.    1.    1.    1. ↴
```

Que es una matriz de 5x4 llena de unos.

Observar que si queremos una matriz completa de otro número sólo tenemos que multiplicar esta matriz por ese número.

A veces uno quiere ver sólo una fila o una columna de una matriz ya existente. Por ejemplo si  $a = [1\ 3; 5\ 4]$  la matriz definida es:

```
1.    2.  
3.    4.  
y si hacemos
```

```
a(1,:)↴  
1.    3.↴
```

nos muestra sólo la primera fila de a.

Si hacemos,

```
a(:,2)↴  
2.  
4.↴
```

Nos muestra la segunda columna de a.

Usando el símbolo **:** para referirse a todas las filas o todas las columnas de una matriz podemos cambiar el valor de los elementos de una matriz o agregar filas o columnas. Por ejemplo,

```
a(:,3)=[1 2]'↴  
1.    2.    1.  
3.    4.    2.↴
```

le agrega una columna de elementos 1 y 2 a la matriz (observen que en este caso la dimensión del vector que agregamos tiene que ser igual a la de la cantidad de filas). Por otro lado, si ahora hacemos

```
a(:,1)=0↴
```



obtenemos:

```
0.    2.    1.
0.    4.    2.
```

es decir, hacemos todos los elementos de la columna 1 iguales a 0. Y si después hacemos

```
a(2,:)=3;
```

obtenemos:

```
0.    2.    1.
3.    3.    3.
```

es decir, hacemos todos los elementos de la fila 2 iguales a 3.

*Ejercicio.* Defina una matriz de 3x4 completa de unos. Agregue una columna de ceros y después una fila completa de 2. Calcule la dimensión de la nueva matriz.

Si queremos redefinir los valores de algunas filas (no de todas) o, equivalentemente, de algunas columnas, usamos el símbolo `:` intercalado entre el número de las filas o de las columnas que queremos redefinir. Por ejemplo, supongamos que la matriz `a` está dada por (piense cómo podría obtenerla a partir de alguna de las matrices definidas anteriormente):

```
0.    2.    1.
3.    3.    3.
2.    3.    4.
```

y hagamos `a(1:2,:)=6;`. Antes de mirar el resultado, piense qué espera obtener. Lo que se obtiene es una matriz donde todos los elementos de las filas 1 y 2 son iguales a 6.

## Operaciones básicas con vectores

Supongamos que queremos multiplicar cada componente del vector `[1 3 4 2 6]` con la componente correspondiente del vector `[8 54 3 2 8]`. Esto se hace así: `"[1 3 4 2 6].*[8 54 3 2 8]"`. De este modo se obtiene como resultado un nuevo vector de cinco componentes cada una de las cuales es el producto, componente a componente, de los dos vectores multiplicados. De la misma manera se procede para cualquier otra operación, teniendo cuidado de que los vectores tengan la misma cantidad de componentes. Por ejemplo, para dividir componente a componente se usa el símbolo `"/"`. Si queremos elevar al cuadrado todas las componentes de un vector usamos `".^2"`. Poder operar de esta forma es útil cuando los vectores tienen muchas componentes. Consideremos, por ejemplo, el vector definido en el último ejercicio y supongamos que queremos calcular el cuadrado de todas sus componentes. ¿Cómo lo haría? La sentencia `y = x.^2` da como resultado un vector con tantas componentes como `x`, cada una de las cuales es igual al cuadrado de las componentes de `x`. En el caso de vectores, la sentencia `y = x^2` da el mismo resultado. En el caso de matrices, sin embargo, es necesario usar la notación `".^2"` para elevar todas sus componentes al cuadrado. Para fijar ideas definamos el vector `x` como `x=[1 2 3 4]`.

Escribamos en la consola la siguiente secuencia:

```
x=[1 2 3 4];
```

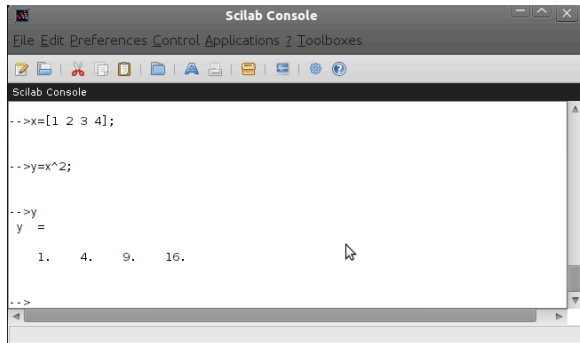
```
y=x^2;
```

```
y
```

`z=x.^2;` ↵

`z` ↵

Vemos que los dos nuevos vectores ("z" e "y") son iguales entre sí e iguales a [1 4 9 16].



```
Scilab Console
File Edit Preferences Control Applications ? Toolboxes

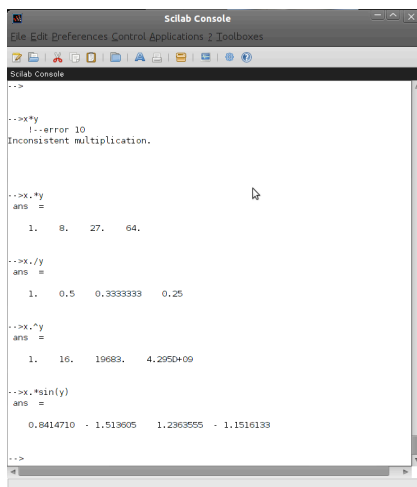
-->x=[1 2 3 4];

-->y=x^2;

-->y
y =
  1.    4.    9.   16.

-->
```

**Figure 7.** Operaciones con vectores



```
Scilab Console
File Edit Preferences Control Applications ? Toolboxes

-->

-->x*y
!--error 10
Inconsistent multiplication.

-->x.*y
ans =
  1.    8.   27.   64.

-->x./y
ans =
  1.    0.5  0.3333333  0.25

-->x.^y
ans =
  1.    16.  19683.  4.2950409

-->x.*sin(y)
ans =
  0.8414710 - 1.513605  1.2363555 - 1.1516133

-->
```

**Figure 8.** Operaciones con arreglos de números. Anteponiendo

un "." al símbolo de la operación, Scilab interpreta que la operación se debe efectuar elemento a elemento.

## Gráficos y arreglos.

Los vectores constituyen los elementos claves para hacer gráficos de una variable en función de otra. Por lo general tenemos un conjunto de datos "x" y otro conjunto de datos "y" relacionados entre sí. Por ejemplo definamos "x=-10:0.1:10;" e "y=x.^3;". Si queremos graficar "y" como función de "x" escribimos en la consola: `plot(x,y);` . Scilab abre entonces una nueva ventana donde hace el gráfico correspondiente y que se muestra en la Figura 9.

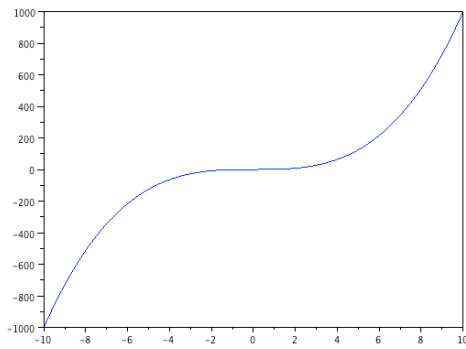


Figura 9.

El ejemplo de la Figura 10:

`x=-pi:0.01:pi;` ↵

`y=x.*sin(16*x);` ↵

`plot(x,y);` ↵

da lugar al gráfico de la Figura 11.

```
-->
-->
-->x=-pi:0.01:pi;
-->y=x.*sin(16*x);
-->plot(x,y)
```

Figura 10

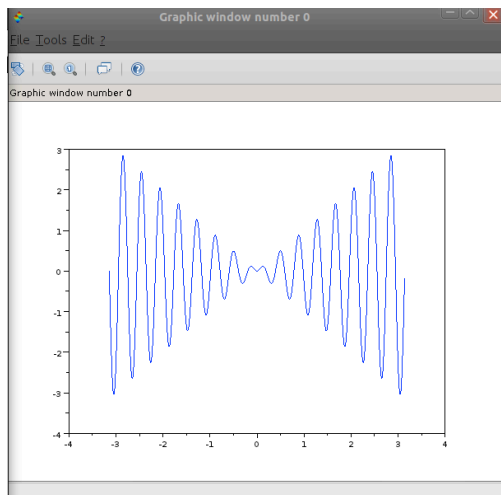


Figura 11

Se puede superponer un nuevo gráfico sobre esta última figura, por ejemplo, si escribimos `plot(x,x)` ; ↵ se dibuja la recta de pendiente 1 que pasa por el origen de coordenadas y si escribimos `plot(x,-x)` ; ↵ la de pendiente -1. Los comandos y el gráfico se observan en las figuras que siguen.

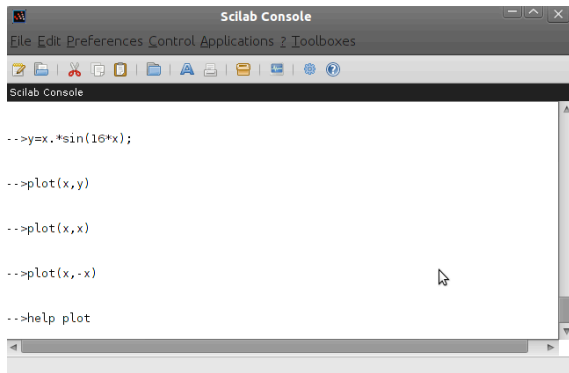


Figura 12

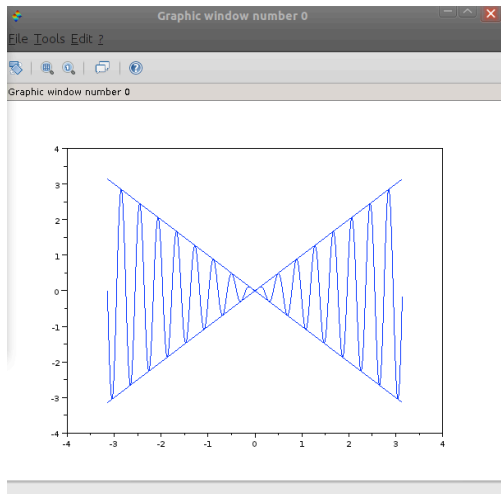


Figura 13

Si se quiere hacer un gráfico en una nueva ventana (es decir, sin superponer al gráfico anterior) escribimos

`figure;` ↵

en la consola y eso abre una nueva ventana. Cada nueva ventana de gráficos que se abre está identificada con un número (ver que en el borde de la ventana dice Graphic window number y el número que corresponda). El comando `plot` hace el gráfico en la última ventana abierta a menos que uno le indique algún otro número. Por ejemplo, si uno escribe:

`figure;` ↵

se abre una ventana (que se llamará Graphic window number 0 si no había ninguna otra previamente abierta). Si después escribimos

`plot(x,y);` ↵

el gráfico de “y” como función de “x” aparecerá en la ventana identificada con el número 0. Si luego hacemos nuevamente

`figure;` ↵

se abre una nueva ventana (Graphic window number 1) y al hacer

`plot(x,x^2);` ↵

el gráfico de  $x^2$  como función de x aparecerá en la ventana identificada con el número 1. Si inmediatamente después escribimos

`plot(x,10-x^2);` ↵

el gráfico de  $10-x^2$  como función de x aparecerá superpuesto al de  $x^2$  como función de x en la ventana identificada con el número 1. Y si escribimos

`figure(0);` ↵

```
plot(x,10-x^2) ; ↵
```

ahora el gráfico de  $10-x^2$  como función de  $x$  aparecerá superpuesto al de  $y$  como función de  $x$  en la ventana identificada con el número 0.

Las ventanas de figuras se pueden cerrar como cualquier ventana de windows. Los gráficos que aparecen allí se pueden guardar como una imagen (en distintos formatos) cliqueando en la opción File y eligiendo “Export to” en el desplegable correspondiente. Los contenidos de una ventana se pueden borrar (“limpiar”) sin borrar la ventana con el comando `clf`. En particular, `clf()`; (clear figure) limpia los contenidos de la última ventana abierta y `clf(1)`; los de la número 1.

Observen que en todos los casos en los gráficos aparecen líneas suaves. Esto es así porque, a menos que uno indique alguna opción distinta, el Scilab interpola entre los puntos del plano con abscisa igual a los valores de las componentes del vector  $x$  y ordenada igual a las del vector  $y$ . La interpolación se vuelve evidente para  $x$  con pocas componentes. Por ejemplo, tomen  $x=[1 \ 2 \ 4 \ 6]$  y grafiquen  $x^2$  como función de  $x$ .

A veces uno puede querer graficar sólo los puntos, sin interpolar por ninguna curva, puede querer cambiar de color, superponer curvas con colores elegidos previamente por uno. Todas esas son opciones al comando `plot`. No vamos a detallarlas todas (pueden consultar los ejemplos de la ayuda de Scilab). Acá listamos ejemplos para que prueben.

Primero definimos el vector cuyos puntos corresponden al eje de abscisas:

```
t=0:%pi/20:2*pi;
```

Después graficamos con distintas opciones (usando el color cyan):

Ejemplo 1:

```
clf();  
plot(t,sin(t),'cya+')
```

Ejemplo 2:

```
clf();  
plot(t,sin(t),'cyao')
```

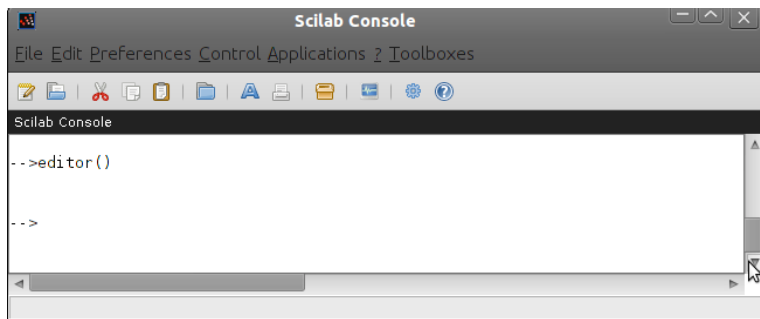
Ejemplo 3:

```
clf();  
plot(t,sin(t),'cyao-')  
-1->plot(t,sin(t),'cyao-')
```

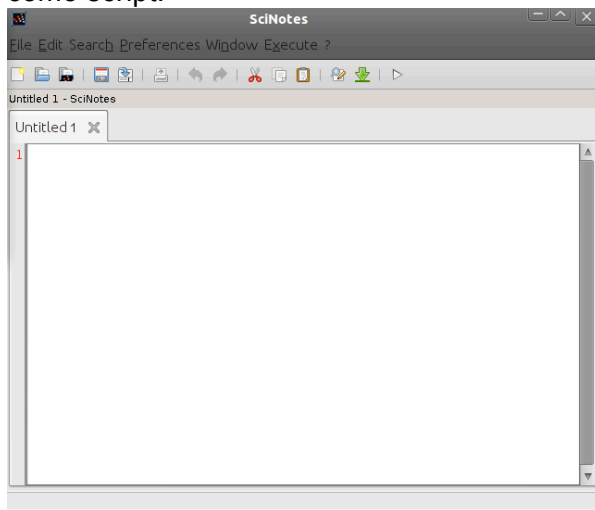
## Archivos de comandos (scripts)

Además de interpretar las instrucciones que uno ingresa en la consola, Scilab puede leerlas también de un archivo. Esto es muy útil porque nos permite escribir en un archivo las instrucciones que queremos ejecutar y después las podemos ejecutar todas las veces que queramos sin necesidad de escribirlas en la consola.

Para escribir el archivo con los comandos utilizaremos el editor que viene con Scilab para ello escribimos en la consola: `editor()` (también se puede abrir este editor abriendo el desplegable “Applications” y eligiendo la opción Scinotes).

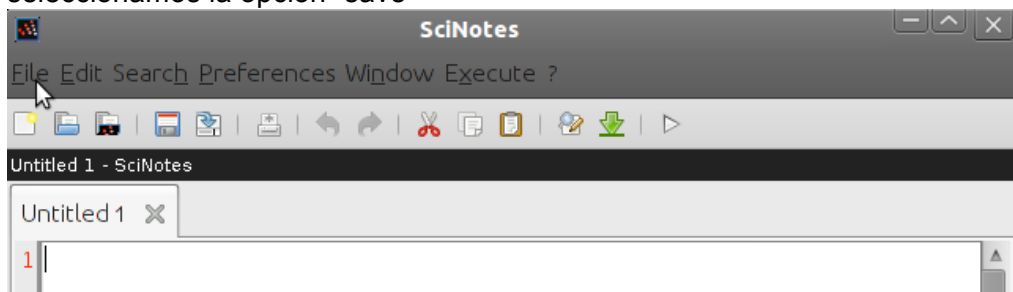


**Figure 13.** Cargamos el editor de texto para escribir las instrucciones y crear lo que se conoce como script.



**Figura 14.** Ventana del editor SciNotes que usamos para generar archivos de texto, en particular, los de comandos.

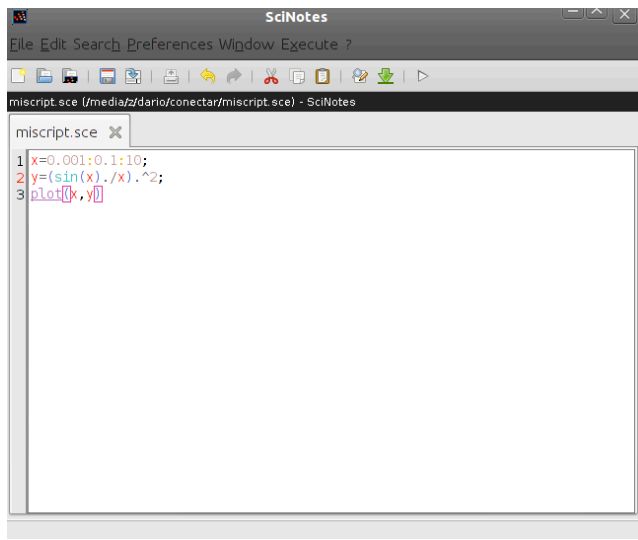
Lo primero que hacemos es guardar el archivo, para lo que vamos al menú File y allí seleccionamos la opción “save”



**Figura 15.** guardamos el archivo.

Le ponemos un nombre al archivo (por ejemplo, miscript.sce). El nombre tiene que terminar en .sce para que Scilab lo pueda identificar automáticamente como un archivo de comandos. En el archivo podemos introducir comandos e instrucciones como lo hicimos en la consola. La diferencia es que las instrucciones no se ejecutarán hasta tanto nosotros no hagamos algo desde la consola para que esto suceda. Por lo tanto podemos escribir varias líneas de instrucciones en el editor que serán luego ejecutadas en forma secuencial cuando se lo indiquemos al Scilab desde la consola.

En el ejemplo que se muestra en la Figura que sigue la secuencia de comandos involucra definir los vectores “x” e “y” y luego graficar “y” como función de “x”.



**Figura 16.** Ejemplo de un script.

Para que Scilab ejecute las instrucciones contenidas en el archivo se debe clicar en “Execute” y elegir alguna de las opciones del desplegable. Todas ellas ejecutan las instrucciones (la que dice “until caret” ejecuta todas las instrucciones hasta llegar a un símbolo particular). La diferencia de hacerlo con o sin eco radica en si al irse ejecutando las instrucciones se muestran resultados parciales en la consola o no. También se puede apretar un botón tipo “play” que significa Execute (sin eco). Cuando hacemos eso se ejecutan las instrucciones. En el ejemplo, todo finalizará con la generación del gráfico. Existe también la opción de elegir un subconjunto de instrucciones e indicar que se ejecuten sólo esas.

Si queremos editar un nuevo archivo, es posible utilizar la opción de nuevo archivo en el borde de la ventana del editor. Se abre entonces una nueva pestaña que corresponderá al nuevo archivo que queramos editar.

### *Ejercicio*

Repita algunos de los cálculos y figuras que hizo en los ejercicios anteriores pero ahora escribiendo las instrucciones en un archivo y ejecutándolo parcial o totalmente.

## **Funciones y loops.**

De la misma forma en que guardamos en un archivo un conjunto de instrucciones para ejecutar secuencialmente en algún momento posterior, es posible también definir funciones. Por ejemplo, abramos un nuevo archivo y escribamos lo siguiente:

```
function y=f(x)
y=sin(x)/x
endfunction
```

Guardamos después el archivo con algún nombre (por ejemplo mifuncion.sci (la extensión sci también es propia de los archivos que el Scilab reconoce). Al “ejecutar” la función (con alguno de los métodos descritos antes) lo que hacemos es que el Scilab reconozca, de ahora en más

(hasta que se nos ocurra redefinir el significado de “f”) a  $f(x)$  como la función definida en el archivo que guardamos. Por lo tanto, si en la consola hacemos:

```
f(1) ↓
```

obtenemos:

```
0.8414710
```

que es el resultado de hacer  $\sin(1)/1$ . También podemos graficar la función. Si la función puede aplicarse a un vector, hacer el gráfico es inmediato. En el ejemplo no lo es, ya que Scilab no entiende qué significa calcular el seno de un vector y dividirlo por el mismo vector. Si sabe hacerlo si escribimos las instrucciones en la definición de la función aclarando que en caso de tratarse de un vector el cálculo debe ser hecho componente a componente. Piense cómo debería modificar la definición de la función  $\sin(x)/x$  para poder aplicarla a un vector y que el resultado sea un vector de componentes iguales a hacer este cociente para cada una de las componentes por separado.

Acá damos la respuesta:

```
function y=f2(x)
y=sin(x)./x
endfunction
```

Ahora es posible definir un vector  $v=[0.1:0.1:1]$  y calcular el vector  $f2(v)$ . Por lo tanto, es posible también graficar  $f2(v)$  como función de  $v$ . Hágalo.

### *Ejercicio*

Defina una función cualquiera de una variable y luego gráfiquela para algún rango de la variable.

Usar la función  $f2$  en lugar de la función  $f$  no es la única opción que tenemos para graficar la función  $\sin(x)/x$  como función de  $x$  para valores de  $x$  dentro de un cierto rango. También podemos definir un vector  $x$ , aplicarle “a mano” la función  $f$  a cada una de sus componentes y guardar cada resultado en la componente de un vector  $y$ . Por ejemplo:

```
x=[0.1:0.1:1];
```

```
y=x;
```

```
y(1)=f(x(1));
```

```
y(2)=f(x(2));
```

```
y(3)=f(x(3));
```

y así sucesivamente con cada elemento de  $x$ . Si  $x$  tiene pocas componentes ésta pueda ser una opción, pero si tiene muchísimas no! Lo que sí podemos notar es que siempre escribimos casi la misma expresión, sólo cambiamos el número de la componente. Es decir, estamos haciendo algo recurrente. Es posible pedirle a Scilab que haga un cálculo recurrente sin escribir tanto. Esto se hace usando “loops” (o “bucles”). Por ejemplo, el loop que nos permite definir  $y(i)=f(x(i))$  para  $i$  entre 1 y el número de componentes de  $x$  es:

```
y=x;
```

```
n=length(x);
```

```
for i=1:n
```

```
y(i)=f(x(i));
```

```
end
```

En este ejemplo primero definimos un vector, “ $y$ ”, que tiene el mismo tamaño que  $x$  (al principio es igual a  $x$  componente a componente). Luego definimos una variable,  $n$ , igual al número de componentes de  $x$  (o de  $y$ ). Lo que sigue es el loop. Significa: para (for) todos los valores de  $i$  entre 1 y  $n$  ejecute la sentencia que sigue (empezando con  $i=1$  y terminando en  $i=n$ ). Cada vez



que Scilab ejecuta la igualdad para algún valor de  $i$  se olvida del valor anterior de la componente  $i$ -ésima de  $y$  (es decir, se sobre-escribe el valor de  $y(i)$ ). Compruebe que esto funcionó graficando “ $y$ ” como función de “ $x$ ”.

Los loops pueden ser usados para cualquier tipo de cálculo recurrente. Supongamos que queremos sumar  $1+1/2+1/4+1/8+1/16+\dots+1/2^{20}$ . Para esto podemos usar el siguiente loop:

```
suma=0;
for i=1:21
suma=suma+1/2^(i-1);
end
```

Acá estamos usando algo que no usamos hasta ahora: ¿qué significa hacer  $\text{suma}=\text{suma}+1/2^{(i-1)}$ ? Tomemos un ejemplo más sencillo. Supongamos que escribimos:  $\text{suma}=0$

Ahora la variable “suma” vale 0. Después escribimos

```
suma=suma+1
```

Bien, el valor de “suma” a la derecha de esta igualdad es 0 (es decir, el valor viejo de suma). Y el de la izquierda es  $0+1$ , es decir, 1. El de la izquierda es el nuevo valor. Repita esto varias veces y compruebe que el nuevo valor de suma se va incrementando en uno respecto del anterior. Esto indica que, cuando en el loop hacemos  $\text{suma}=\text{suma}+1/2^{(i-1)}$ ; a la derecha estamos tomando el valor que tenía suma hasta que sumamos el anterior valor de  $1/2^{(i-1)}$ , a ese valor le estamos sumando un nuevo valor de  $1/2^{(i-1)}$  y lo volvemos a guardar en la variable suma. El resultado al terminar el loop es haber sumado todos los términos de la forma  $1/2^{(i-1)}$  con  $i$  entre 1 y 20.

*Ejercicio* (para pensar y, si es posible, hacer en casa): Tiro oblicuo. Las coordenadas de una partícula que es arrojada cerca de la superficie de la Tierra con velocidad inicial de módulo  $v_0$  formando un ángulo  $\alpha$  con la horizontal están dadas por las siguientes funciones del tiempo:

$$x(t) = v_0 \cos(\alpha) t \quad (1)$$

$$y(t) = v_0 \sin(\alpha) t - \frac{g}{2} t^2 \quad (2)$$

donde  $x$  e  $y$  se refieren, respectivamente, a las coordenadas horizontal y vertical (a lo largo de la cual actúa la aceleración de la gravedad,  $g$ ) y donde se eligió el origen de coordenadas coincidiendo con la ubicación inicial de la partícula. Suponga que la mitad de la aceleración de

la gravedad se puede aproximar por  $\frac{g}{2} = 5 \frac{m}{s^2}$ . Grafique en figuras separadas:

- $x$  como función de  $t$  (para  $t$  entre 0 y algún valor mayor que 0)
- $y$  como función de  $t$  (para los mismos tiempos)
- $y$  como función de  $x$

para distintos valores de  $v_0$  y  $\alpha$ . Pensar cómo hacerlo.