Práctica 3. Introducción a Scilab III

Objetivo. El objetivo de esta práctica es dar una introducción al uso de Scilab como un lenguaje de programación. Incluyendo programación mediante scripts y funciones.

Introducción.

En las dos prácticas anteriores hemos usado a Scilab como una calculadora que simplemente ejecuta los comandos conforme se van introduciendo manualmente en ventana de comandos. A partir de esta práctica veremos que Scilab también puede utilizarse como un poderoso lenguaje de programación.

Un programa en Scilab es simplemente un archivo que contiene comandos de Scilab que se ejecutarán en una cierta secuencia. Los archivos de comandos 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 3.1

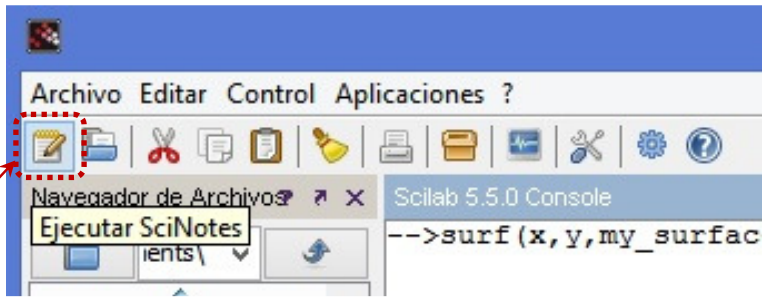

Figura 3.1.- Icono para ejecutar el editor SciNotes.

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')
```

O si se desea empezar a escribir una función nueva se teclea simplemente `edit`.

Control de flujo del programa.

Scilab posee instrucciones orientadas a controlar la secuencia (o flujo) de ejecución de las instrucciones de un programa. 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 o verdadero) 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 las prácticas anteriores 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 ($x \leq 4$) y ($x \leq 1$ ó $x \geq 3$) 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 ($x \leq 4$ y $x \leq 1$) ó ($x \geq 3$). 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*.

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(nombre_del_archivo,0)`, 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.
```

☞ Como se advirtió desde la introducción en la práctica No. 1, Los ciclos deben evitarse siempre que haya una función intrínseca 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
```

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)`.
- El número de variables de entrada enviadas a una función está disponibles dentro de la función en la variable `argn(2)`. El número de variables de salida solicitadas cuando una función se llama está disponible dentro de la función en la variable `argn(1)`. Opcionalmente se puede usar `argn(0)` para obtener ambos valores.
- El comando `mode(0)` provoca que se visualicen los resultados de los comandos que envían información a la consola (comandos `input`, `display` o comandos que no terminan con ;) en el código de la función. El comando `mode(-1)` deshabilita la visualización de resultados de cualquier comando que produzca una salida a la consola.
- 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`.

Como un ejemplo de función guardada en un archivo, abriremos el archivo `log10.sci` que corresponde a la función intrínseca de Scilab llamada `log10`, y que simplemente calcula el logaritmo de base 10 de un número. Para abrir el archivo podemos usar el comando `edit`: (los comentarios en español se han agregado al código original)

```
edit log10
```

```
// Scilab ( http://www.scilab.org/ ) - This file is part of Scilab
// Copyright (C) INRIA
// Copyright (C) DIGITEO - 2011 - Allan CORNET
//
// This file must be used under the terms of the CeCILL.
// This source file is licensed as described in the file COPYING, which
// you should have received as part of this distribution.  The terms
// are also available at
// http://www.cecill.info/licences/Licence_CeCILL_V2.1-en.txt
function y=log10(x)
    [lhs, rhs] = argn(0); // Obtiene número de variables de entrada (rhs)
                          // y de salida (lhs)
    if rhs == 0 then
        error(msprintf(gettext("%s: Wrong number of input argument(s): %d
expected.\n"),"log10", 1)); // Si se invocó sin argumentos, despliega un error
    end
    y = log(x) / log(10);//Número correcto de argumentos, calcula el logaritmo
endfunction
```

En la línea de comandos se invoca la función como sigue (por ejemplo) `y=log10(100)`

Ventajas y desventajas de funciones Vs. scripts.

Los scripts aún 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.

Sin embargo, 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. En adelante se preferirá programar solamente mediante funciones.

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

Ejemplo 3.9 : La siguiente función realiza la gráfica de un tren de pulsos rectangulares de amplitud 1 y periodo 1, es decir, grafica la función periódica cuya definición en el periodo fundamental $0 \le x \le 1$ es: $f(x)=\begin{cases}1 & si\ 0 \le x < 0.5 \\ 0 & si\ 0.5 \le x < 1\end{cases}$ y calcula el error de su aproximación por ciclo de su serie de Fourier $f_s(x)=0.5+\frac{2}{\pi}\left[\sin(\omega x)+\frac{1}{3}\sin(3\omega x)+...+\frac{1}{N}\sin(N\omega x)\right]$ donde $\omega = 2\pi$ es la frecuencia fundamental de $f(x)$ y donde N es el máximo número de armónicos deseado y debe de ser impar.

```
function err=fourier_aprox(n, N)
//  err=fourier_aprox(n,N) calcula el error de la aproximación por serie de
//  Fourier usando N armónicos (N impar) y n ciclos, con 10000 puntos por ciclo.
//  Para el tren de pulsos periódico de amplitud 1 y periodo 1
//  que vale 1 el primer medio periodo y 0 el segundo medio periodo
//
x=0:1e-4/n:n;    //Graficará n ciclos de f(x), con 10000 puntos por ciclo
w=2*%pi;  //frecuencia fundamental
f=1*(sin(w*x)>0); //Genera el tren de pulsos
clf; //limpia figura actual
plot(x,f,'k-','linewidth',3); xgrid; //Grafica el tren de pulsos
a=get("current_axes");
a.data_bounds=[0 n -0.2 1.2];
N=int(N); // se asegura que N sea entero
while (N/2)==int(N/2)
```

```
   N=N+1; //se asegura que N sea  impar
   disp(N,'N=')
end

//// A continuación evalúa la Serie de Fourier
fs=0.5;  //Inicializa sumatoria
for k=1:2:N
    fs=fs+(2/%pi/k)*sin(k*w*x);
end
plot(x,fs,'r-','linewidth',2);
title('Aproximación por Serie de Fourier','fontSize',5);
h=legend('exacto','aproximado');
h.font_size=3;
e=f-fs;  //Vector de diferencias entre f y fs
err=sqrt(e*e'); //raiz cuadrada de suma de cuadrados de elementos de e.
err= err/n;  //divide error entre numero de ciclos.
endfunction
```

Al ejecutar desde la línea de comandos como sigue, se obtiene la gráfica de la figura 3.2.

```
err=fourier_aprox(2,4) // a propósito se pone N par para que lo corrija
N=
    5.
err  =
    12.939235
```

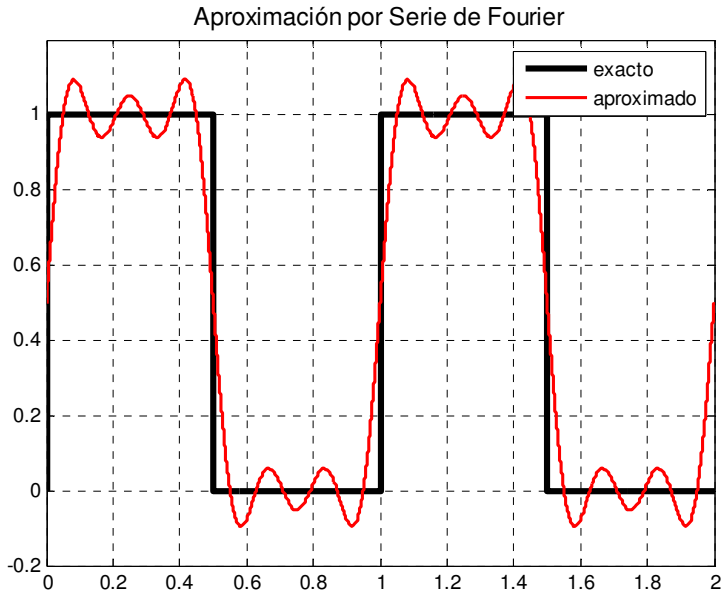

Figura 3.2. Aproximación del tren de pulsos obtenida al ejecutar `fourier_aprox(2,5)`.

Ejercicios:

1. Usando el hecho de que en Scilab una desigualdad como $2 \le x \le 3$ tiene un valor lógico que se convierte en valor numérico (0 o 1) al involucrarlo en cálculos aritméticos, genera sin usar ciclos la gráfica de la figura 3.3

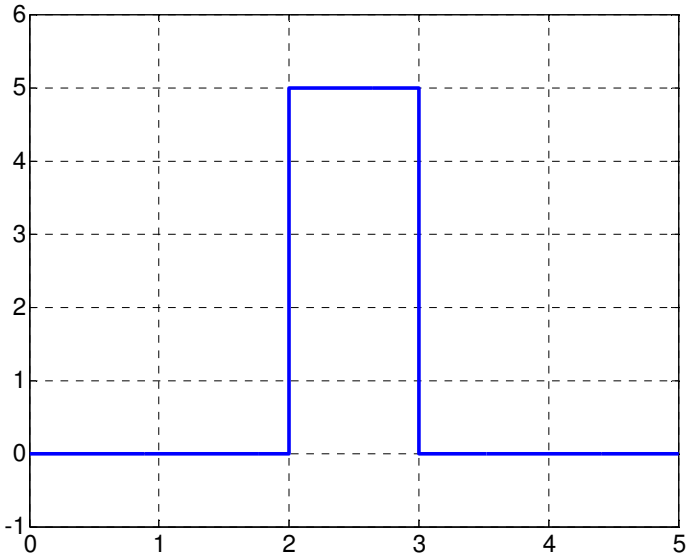

Figura 3.3

2. Modificar el código de la función `fourier_aprox` para que sólo grafique un ciclo del tren de pulsos y de su aproximación por serie de Fourier incrementando el número de armónicos N desde 1 hasta infinito (solo números impares). Esperando en cada valor de N a que el usuario presione la tecla [enter] para continuar (usar el comando: `input('presione [enter] para continuar');`

Desarrollo de la Práctica.

1. Probar todos los ejemplos propuestos por el profesor conforme los va explicando.
2. Realizar todos los ejercicios propuestos.
3. Contestar el cuestionario de evaluación de la práctica.

Reportar:

1. Investigar u obtener el desarrollo en serie de Fourier para la función triangular de periodo 1, dada en el periodo fundamental $0 \le x \le 1$ por: $f(x) = \begin{cases} 1-2x & si\ 0 \le x < 0.5 \\ 2x-1 & si\ 0.5 \le x < 1 \end{cases}$ y modificar la función `fourier_aprox(N)` para que grafique un ciclo de esta función y su aproximación en serie de Fourier con N armónicos.
2. Escribir la función `[m,b]=interpola(x1,y1,x2,y2)` para calcular la pendiente y la ordenada al origen de la recta que une los puntos de coordenadas (x1,y1) y (x2,y2) y que grafique los dos puntos (marcándolos con un signo '+') y la recta que los conecta.