Laboratorio 2: Programación en SciLab

Ya vimos un poco de la sintaxis de SciLab. Ahora vamos a empezar a ver cómo usarlo para desarrollar cálculos largos.

1. Programación de Scripts

Miren la ventana de SciLab. Arriba a la derecha hay un cuadro desplegable donde indica un directorio (una carpeta). Ese es el directorio de trabajo de SciLab. Ahí se guardan los archivos y de ahí se leen. Vamos a cambiarlo.

Busquen un lugar adecuado y creen una carpeta para ustedes (pregúntenle al responsable de la sala de PCs dónde es un lugar adecuado para que no les borren los archivos).

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 vaya a `Aplicaciones >> Scinotes`. Lo primero que hacemos es guardar el archivo, para lo que vamos al menú Archivo y allí seleccionamos la opción "guardar". Le ponemos un nombre al archivo (por ejemplo, ejemplo1.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 archivo guardado escriba

```
x=1.95:.001:2.05; // Rango para la variable x
                  //x=x_inicial:longitud_del_paso:x_final
                  // esto es un comentario
// Polinomio en forma expandida
y1=(x.^9 -18*x.^8 + 144*x.^7- 672*x.^6 + 2016*x.^5 ...
- 4032*x.^4+ 5376*x.^3 - 4608*x.^2 + 2304*x - 512);
```

```
// Polinomio en forma anidada
y2=-512+(2304+(-4608+(5376+(-4032+(2016+(-672+(144+(-18+x) ...
.*x).*x).*x).*x).*x).*x).*x).*x;

// Polinomio factorizado
y3=(x-2).^9;

plot(x,y1,'--g',x,y2,'r',x,y3,'b');
legend('expandido','anidado','factorizado');
```

(ojo con las comillas, tienen que ser las verticales; las inclinadas para un lado o el otro no sirven).

Listo, para ejecutar el programa, hay que ir al menú `ejecutar`. Allí encontrará tres formas de ejecutarlo:

... archivo sin eco esta opción ejecuta el programa dando sólo los resultados del mismo en algún archivo en donde se decida guardar los resultados (veremos cómo hacer esto más adelante). Esta opción no es la más recomendable cuando se empieza a usar el programa ya que no es posible la visualización de posibles errores.

... archivo con eco esta opción ejecuta el programa y luego, en la Scilab-Console aparece el programa y el/los resultado/s, si es que no cortó antes con algún error en cuyo caso avisa en donde está el error.

...hasta el cursor, con eco esta opción ejecuta el programa sólo hasta donde esté parado el cursor. También, si selecciona una parte del programa, en lugar de **...hasta el cursor, con eco** aparece la opción **...selección, con eco** con la cual puede ejecutar pequeñas secciones del programa.

La ventaja de esto es que si queremos cambiar algo, por ejemplo el rango para las `x`, sólo hay que ir al archivo y cambiar la línea correspondiente, y no hace falta volver a ejecutar todos los comandos. Es importante, cuando se realice un cambio, volver a a guardar el archivo, porque al ejecutar en la ventana de comandos lo que se usa es lo último guardado, y no lo que esté escrito sin guardar.

1.1. Condicionales

Los programas que hagamos pueden ser un poco más inteligentes y tomar algunas decisiones. Para eso le tenemos que dar instrucciones del estilo ***Si*** *pasa esto* ***entonces*** *hacé esto* ***si no*** *hacé esto otro.*

Veamos un ejemplo. Vamos a calcular $\ln 2$ usando la función que trae

SciLab, y compararla con la aproximación

$$\ln 2 \approx \sum_{n=1}^{N} \frac{(-1)^{n-1}}{n}$$

Cree un archivo .sce y póngale el siguiente programa

```
tolerancia = 10^(-3);

Maquina = log(2) //Logaritmo calculado con la funcion de SciLab

// Logaritmo calculado con la sumatoria:
N=100;
n=1:N;  //es un vector con todos los indices de la sumatoria
Nosotros = sum( (-1).^(n-1) ./ n)
     //recuerden que .^ y ./ hacen la cuenta para cada elemento

if abs(Maquina - Nosotros)<tolerancia
    disp('Los resultados son parecidos');
else
    disp('los resultados son diferentes');
end
```

Pruebe cambiando los valores de `N` y de `tolerancia` para ver diferentes resultados posibles.

1.2. Bucles

Los bucles son partes del programa que se repiten muchas veces Hay dos clases de bucles: los bucles `for`, que se repiten un número fijo de veces, y los bucles `while` que se repiten dependiendo de alguna condición.

Veamos un ejemplo sencillo, para calcular el factorial de 100

```
F=1;
for i=1:100
     F=F*i;
end
F
```

Una cosa que parece difícil al principio, es que las variables que se usan dentro del bucle tienen que inicializarse antes con algún valor que no afecte al resultado (en este caso `F=1` porque no afecta a la multiplicación). Si no, en la primera vuelta se produciría un error, porque la computadora no sabría qué valor usar en la primera multiplicación.

Vamos a un ejemplo un poco más difícil. Calculemos los primeros 10 términos de la sucesión de Fibonacci.

```
Fviejo=0;
Factual=1;

for i=1:10
    Fnuevo = Fviejo + Factual //calculo el nuevo término
                              //ahora cambio, el nuevo pasa a ser actual,
                              //y el que era actual ahora es viejo.
    Fviejo = Factual;
    Factual = Fnuevo;
end

//observen que hay punto y coma en los renglones
//que no quiero ver el resultado, y no hay punto
//y coma en el resultado que sí quiero ver
```

La lista de valores para el índice puede ser una cosa distinta del típico `1:n`, por ejemplo, el siguiente programa calcula la suma $1^2+3^3+5^2+\cdots+51^2$.

```
impares = 1:2:51;
imparesAlCuadrado = impares .^ 2;

S=0;
for i=imparesAlCuadrado
    S=S+i;
end
S
```

Ejercicio Para la sucesión de Fibonacci, F_1, F_2,... calcule los cocientes F_{n+1}/F_n para n desde 1 hasta 50. Supuestamente este cociente tiende a $\frac{1+\sqrt{5}}{2}$. Los resultados obtenidos, ¿se parecen a eso?

Hay otro tipo de bucle, que se usa para los casos en los que no sé, a priori, cuántas veces va a haber que repetirlo. Se repite ***mientras*** se cumple una condición. Por ejempo

```
tol = 10^(-3); //tolerancia

S=0; //inicializo variables que se usan en el bucle
n=1;
while abs(S-log(2))>tol
    S = S + (-1)^(n-1) / n;
    n = n+1;
end
Repeticiones = n-1
ErrorAbsoluto = abs(S-log(2))
ErrorRelativo = ErrorAbsoluto / log(2)
```

Pruebe cambiando la tolerancia para ver cómo funciona. **Importante:** Si tarda mucho, la combinación de teclas `<Ctrl>+C` corta el programa o en el menú de Scilab-Console, vaya a Control, allí encontrará tres opciones: continuar, cancelar e interrumpir.

Si la tolerancia es demasiado chica, puede ser que el programa nunca termine (por culpa de los errores de redondeo), eso se arregla cambiando la condición del bucle por algo como

```
while abs(S-log(2))>tol & n<=10000
```

Expresiones lógicas

operador	descripción
`&`	Y
`|`	O
`~`	No
`<`	Menor
`>`	Mayor
`<=`	Menor o igual
`>=`	Mayor o igual
`==`	Igual (el símbolo simple `=` se usa para asignar valor a variables)
`~=`	Distinto
`or(a<b)`	algún elemento de `a` es menor que el correspondiente de `b`
`and(a<b)`	todos los elementos de `a` son menores que los correspondientes de `b`

2. Programación de funciones

La diferencia principal entre las funciones y los scripts es que en las funciones hay valores que se deciden al momento de usarlas. Por ejemplo, SciLab tiene programada una función para calcular logaritmos, que se llama `log`. Si quiero calcular el logaritmo de 2, no tengo que ir al programa ese y cambiar un número por un 2, sino que alcanza con escribir `log(2)`. La gracia es aprender, además de a usar las funciones que programaron otros, a programar nuestras propias funciones.

Algunas diferencias importantes entre las funciones y los scripts.

- Hay variables a las que no se les da un valor inicial, sino que que se les asignan valores cuando se llama a la función, poniendo los números correspondientes entre los paréntesis, se llaman *parámetros de entrada*
- Hay una o más variables que son el resultado de la función. Se llaman *parámetros de salida*
- El resto de las variables que se creen dentro de la función se borran automáticamente apenas la función termina, para que no ocupen lugar en el Workspace.

Para crear una función, hay que crear un archivo .sci o .sce, y en el primer renglón poner algo como

```
function resultado=estaesmifuncion(entrada1, entrada2, entrada3)
//poner la función que desea evaluar
endfunction
```

El nombre de la función es `estaesmifuncion`, los parámetros de entrada son `entrada1`, `entrada2` y `entrada3` y el parámetro de salida es `resultado`.

Si hacen falta más parámetros de salida, se pueden agregar entre corchetes, por ejemplo

```
function [r1,r2]=bhaskara(a,b,c)
disc = sqrt(b^2 - 4*a*c);
r1 = (-b-disc)/(2*a);
r2 = (-b+disc)/(2*a);
endfunction
```

Importante: No es necesario que el nombre del archivo sea el mismo nombre que el de la función como en MatLab. Si es muy importante que la ejecute pues al "ejecutar" la función (con alguno de los métodos descriptos antes) lo que hacemos es que el Scilab reconozca, de ahora en más (hasta que se nos ocurra redifinir el significado de "bhaskara") a bhaskara(a,b,c) como la función definida en el archivo que guardamos. Otra cosa importante es que, al usar el

```
endfunction
```

Ahora usamos esa función

```
>> bhaskara(1,-1,-1)
```

y el resultado es

```
ans =
   -0.6180
```

... nos dio una sola de las dos raices, y si se fijan en el Workspace, no están guardadas en memoria ni `r1` ni `r2`. Si queremos guardar los resultados en memoria (y verlos a los dos) tenemos que usar la función así:

```
>> [a,b]=bhaskara(1,-1,-1)
```

y el resultado será

```
a =
   -0.6180


b =
    1.6180
```

Y ahora sí, las dos variables quedan almacenadas en la memoria. Fíjense que no importa que yo llame `a` y `b` a los resultados en la ventana de comandos, y que al mismo tiempo dentro de la función `a` y `b` sean los nombres de dos variables. Dentro y fuera de la función son dos mundos distintos, y los nombres de un lado y del otro no tienen nada que ver.

Ejercicio ¿Se acuerdan de que en algunos casos una de las raíces se perdía por error de redondeo? En este caso fallan `[a,b]=bhaskara(1,10^9,1)` y `[a,b]=bhaskara(1,-10^9,1)`. Modifique la función para que use la fórmula adecuada para cada raíz (va a necesitar usar un `if` para controlar el signo del parámetro `b`)

Otro ejercicio ¿Qué hace la siguiente función?

```
function a=prueba(a,b)

a=round(abs(a)); b=round(abs(b));

if a==0
    error('no pueden ser ambos nulos');
else
    while b ~= 0
        r = a-fix(a./b).*b; %resto de dividir a/b
        a = b;
        b = r;
    end
end
endfunction
```

Pista: Lo vieron en álgebra, y lleva el nombre de Euclides.