SCILAB.

COMPUTACION NUMERICA BAJO LINUX Y WINDOWS

Andrés Jiménez Jiménez.
Titulado Superior de Apoyo a Investigación
Universidad de Cádiz.

Scilab es un software de calculo científico orientado a la computación numérica. Posee una extraordinaria versatilidad y capacidad para resolver problemas de matemática aplicada, física, ingeniería, procesamiento de señales y otras muchas aplicaciones. Su base la constituye un sofisticado interprete formado por cientos de rutinas de calculo matricial, análisis numérico y visualización gráfica. El programa está concebido como un software abierto, el usuario puede ampliarlo añadiendo sus propias primitivas o modificando las existentes. Emplea un entorno de ventanas amigable que recuerda mucho a los paquetes Matlab, Maple, Mathematica, etc.

Como obtenerlo e instalarlo.

Scilab está disponible para el entorno X11 en la mayor parte de las estaciones de trabajo Unix, Windows, y maquinas Linux. Está disponible tanto en *"http://www-rocq.inria.fr/scilab*" tanto en formato rpm como en código fuente, para los sistemas más conocidos Dec Alpha (OSF1 V4), Sun Sparc stations (Sun Solaris 5), HP9000 (HP-UX 10), SGI Mips (Irix 5.2), PC (Linux ELF, WINDOWS 95, 98 y NT4), Linux PPC.

Instalación bajo Uníx y Linux.

La instalación desde el código fuente es la más general ya que permite construir un fichero ejecutable adaptado específicamente a la máquina y al sistema operativo utilizado (sea cual sea). Se necesita aproximadamente 130 Mb de disco para desempaquetar e instalar el formato *tar.gz* y los compiladores de C (gcc) y Fortran (g77 o f2c). El procedimiento de instalación está descrito en un fichero texto denominado "*readme*" (o "*install*") que aparece al descomprimir el software.

La versión binaria exige un mínimo de 40 Mb (sin los fuentes). Dicha versión está linkada estáticamente y por tanto no requiere del compilador fortran. Si hemos bajado una distribución binaria como por ejemplo scilab-2.5-0.i386.rpm, para instalarla, bastaría hacer:

$ rpm -i scilab-2.5-0.i386.rpm

En ambos casos se requiere el entorno X Window (X11R4, X11R5 o X11R6) y una versión de tcl/tk igual o superior a la 8.0.

El programa se instala por defecto en "/usr/lib/scilab-2.5/". La variable de entorno denominada SCI contiene dicho directorio, $SCI cuelgan un conjunto de subdirectorios entre los que están:

bin/	Directorio con el código ejecutable (scilex) y algunos scripts para manejar e imprimir ficheros Postscript/Latex producidos por Scilab.
config/	Directorio con los makefiles necesarios para configurar distintos sistemas.
demos/	Directorio que contienen las demostraciones que pueden ser ejecutadas desde la opción "*demos*" del menú. El fichero "*alldems.dem*" permite añadir nuevas demostraciones a las ya existentes.
examples/	Directorio con ejemplos útiles para linkar Scilab con programas externos usando la orden "*intersci*" o empleando links dinámicos.
intersci/	Contiene los programas necesarios para añadir primitivas Fortran o C a Scilab.

libs/	Directorio que contiene las librerías de Scilab en código compilado.
macros/	Código fuente de las macros empleadas por el programa (ficheros *.sci), entre las que están las macros/util/edit y /util/manedit en las que se fija el editor por defecto. Esta asignacion puede cambiarse definiendo el nuevo editor en la variable *default_editor.*
util/	Rutinas útiles y ficheros ASCII.
X11_defaults/	Contiene el fichero de parámetros asociados con la apariencia del entorno de ventanas de Scilab (Xscilab). Además de los iconos del programa.
.scilab	Fichero de configuración personalizada que cada usuario puede situar en su $HOME

La shell que arranca el programa se llama "/usr/bin/scilab", en ella se redefine el PATH junto con nuevas variables de entorno "PRINTER" con la lista de posibles impresoras, "MANCHAPTER" localización de los manuales, DISPLAY, etc.

Instalación en Windows

En el caso de plataformas Windows 98, 2000 o XP, el procedimiento de instalación es el estandar: al ejecutar el "*Setup*", el software se instala automáticamente como cualquier programa Windows, apareciendo el icono de Scilab en el menú de inicio.

Manuales y ejemplos.

El aprendizaje de Scilab está facilitado por la completa documentación que aporta su pagina principal "*http://www-rocq.inria.fr/scilab/*". Esta dirección contiene además de las diferentes distribuciones del paquete, un manual de ayuda on-line, numerosos ejemplos prácticos y otros documentos en forma de FAQ's y HOWTO's. El manual on-line viene suministrado en los formatos PDF, Postscript o HTML y puede integrarse en el software como parte de la ayuda.

En la dirección "*ftp://ftp.inria.fr/INRIA/Projects/Meta2/Scilab/documentation/*" existe otros manuales específicos en formatos Postscript y LaTeX.

- *Introduction to Scilab*
- *Scicos: a Dynamic System Builder and Simulator*
- *LMITOOL: Linear Matrix Inequalities Optimization Toolbox Intersci: Automatically*
- *Interfacing C and Fortran Subroutines*
- *Signal Processing toolbox manua*
- *Scilab internal structure description*
- *Communication Toolbox Documentation*
- *Metanet User's Guide and Tutorial*

Modos de trabajo

Existen dos formas de trabajar con Scilab:

a) Modo interactivo: Ejecución de cualquier expresión, programa o función Scilab dentro del entorno. El modo interactivo puede arrancarse en modo ventana o en modo texto

- Modo ventana: *$ scilab*
- Modo texto: *$ scilab -nw*

b) Modo bash: Ejecución de un fichero sin necesidad de entrar en el entorno del programa.

$ echo "exec('ordenes.sce'); quit" | scilab -nw > resultados.out

Evidentemente es posible emplear la ordenes unix "*cron*" o "*at*" para que la ejecución se realice en un instante determinado.

El entorno. Primeros pasos

Una vez instalado el programa se arranca:

$ scilab

Ésta orden, abre un sencillo entorno como el que puede apreciarse en la (fig.1). En él aparece un menú superior que facilita las operaciones básicas con ficheros, demostraciones simples de comandos, ayuda, etc. Un primer contacto con Scilab que ofrece una idea clara de las capacidades de este programa, puede hacerse inspeccionando las demos del menú superior. El resto de la ventana, sirve para la introducción de comandos y presentación de resultados (la línea de comandos "-->" indica que el sistema está dispuesto para recibir una orden). La apariencia de las ventanas de Scilab puede ser alterada a nuestra voluntad. La forma directa de hacerlo es modificar el ficheros "*Xscilab"* contenidos en el directorio "*X11-defaults"*, aunque, una forma menos arriesgada, consiste en copiar en el fichero "$HOME/*.Xdefaults*" las líneas con las nuevas definiciones. Por ejemplo, si deseamos establecer una ventana cuadrada 500x500, con una barra de scroll color rojo bastaría hacer:

.Xdefaults

---

*Xscilab.color*Scrollbar.background:red*
*Xscilab*vpane.height: 500*
*Xscilab*vpane.width: 500*

Estudiemos como trabajar con el entorno.

Expresiones y funciones. Como en casi todos los programas las expresiones matemáticas elementales se construyen empleando los símbolos = (asignación), + (suma), - (sustracción), * (multiplicación) / (división), ^ (potenciación). Los paréntesis se emplean para organizar adecuadamente expresiones complejas. Una forma de decirle al programa que una instrucción demasiado larga no ha terminado y que continua en la siguiente línea es situar tres puntos al final de la línea. Las funciones matemáticas mas usuales *rand, max, min, cos, sin, sqrt, abs*, etc. pueden ser consultadas con las orden

```
--> apropos function
```

Las constantes matemáticas, ya implementadas por el paquete se representan comenzando con el símbolo %, las más importantes son: %pi es π, %e es e=2.718281.., %i es i = $(-1)^{1/2}$, %eps, etc. Podemos empezar ya a evaluar expresiones algebraicas escalares (con valores reales o complejos):

```
→ a= 5+2*%i;
→ b = 4-3*%i;
→ a*b
ans = 26. - 7.i
```

Las sentencias pueden acabar en punto y coma o no, según queramos que aparezca o no reflejado el resultado de comando. Esto es útil en muchos casos, para eliminar el innecesario "*echo*" en pantalla de sentencias como las asignaciones, gráficos, etc.

Matrices y vectores: Para crear una matriz basta dar sus valores (reales o complejos) entre corchetes separando por blancos los elementos de una misma fila y por punto y coma las distintas filas *v=[4 2 3 6; 1 0 3 2-%i].* Un elemento, una fila o una columna particular se especifica con la notación *v(<fila>,<columna>), v(:,<columna>), v(<fila>,:)* por ejemplo:

```
--> v=[4 2 3 6; 1 0 3 2-%i ]        // Definición de una matriz
 v  =
!  4.    2.    3.    6.      !
!  1.    0     3.    2. - i   !

--> v(1,3)                          // Elemento (1,3) de la matriz
 ans  = 3.

--> v(:,3)                          // Columna 3 de la matriz
 ans  =
!  3. !
!  3. !

--> v(1,:)                          // Fila 1 de la matriz
 ans  =
!  4.    2.    3.    6. !
```

Las operaciones como la suma la resta y la multiplicación de matrices utiliza los mismos operadores algebraicos (+, -, *). La multiplicación (*) es el producto matricial. Para realizar estas mismas operaciones pero elemento a elemento debemos utilizar los operadores (.* ./ .^). Por supuesto también existen las funciones propias del tratamiento matricial inversa "*inv()"*, determinante "*det()"*, descomposición singular "*svd()*", etc. La operación denominada transposición consiste en convertir las filas en columnas y las columnas en filas de un vector o matriz (v') se realiza situando el apóstrofe " ' " a la derecha del vector, por ejemplo *v'* .

```
--> v'
 ans  =
!  4.    1.      !
!  2.    0       !
!  3.    3.      !
!  6.    2. + i  !
```

Los vectores son un caso particular de matrices y se definen de la misma manera que estas. Un tipo de vector ampliamente utilizado como base de las definiciones de funciones y representaciones gráficas es el formado por una sucesión de valores igualmente espaciados sobre un intervalo. Para definirlo bastaría dar el primer valor del intervalo, el incremento numérico y el ultimo valor, por ejemplo:

```
--> v=5:-0.5:3
v  =
! 5.  4.5  4.  3.5  3. !
```

Existen también otros vectores útiles como el vector nulo formado por N ceros y el vector constante igual a 1 formado por N unos, esto se consigue con las sentencias*: zeros(1:N)* y *ones(1:N).*

Polinomios: Los polinomios pueden definirse de dos formas diferentes utilizando la primitiva *poly():*

a) Especificando sus raíces

```
--> p = poly( [1 2 ], 'x')
p = 2 - 3x + x^2
--> root(p)
ans  =  !   1. !
        !   2. !
```

b) Especificando sus coeficientes

```
--> q = poly( [1 2 ], 'x', 'c')
q = 1+ 2x
--> root(q)
ans = -0.5
```

Podemos observar el uso de que la función *root()* para calcular las raices reales o complejas del polinomio. Las operaciones algebraicas elementales suma, resta, producto y cociente se realizan con los operadores habituales *p+q, p-q, p*q, p/q*.

Ficheros de datos. La lectura y escritura de datos en ficheros adquiere una enorme importancia cuando el conjunto de valores es muy grande. Scilab dispone de un conjunto de sentencias que emulan a las ordenes de lectura y escritura del lenguaje C, tales como *printf, fprintf, scanf, fscanf,* etc., esto facilita el aprendizaje a los usuarios que poseen conocimientos de programación en C. Aparte de estas sentencias existe una manera fácil de escribir y leer de bloques de datos empleando las ordenes *write* y *read*

```
--> x=[1 2 %pi; %e 3 4];          // Define una matriz de datos
--> write('x.dat', x);            // Escribe la matriz de datos en x.dat
--> xnueva = read('x.dat', 2,3)   // Lee la matriz de datos especificando sus dimensiones.
xnueva=
! 1.      2.      3.1416  !
!2.7182 3.      4.      !
```

Los gráficos

La gama de gráficos científicos que Scilab puede realizar es enorme, comprende gráficos en dos y en tres dimensiones, en coordenadas cartesianas y en paramétricas, en escalas decimal, logarítmica y semilogarítmica, además de representaciones específicas para datos estadísticos, sistemas de control (Bode, Niquist, etc.), animaciones, etc. Veamos algunos ejemplos:

Graficos 2D El caso más sencillo de gráfico 2D es el de una función y=f(t), en el ejemplo siguiente se muestra como utilizar la función "*plot()"* para construir este gráfico.

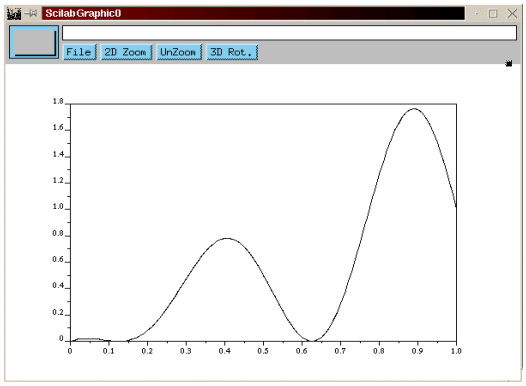
```
--> t=(0:0.01:1)';                          // Definición del rango de t.
--> ft = t.*( sin(2*%pi*t) -cos(2*%pi*t) ).^2;   // Definición de f(t)
--> plot2d( t, ft);                         // Ejecución del gráfico del tipo y = f(t)
```

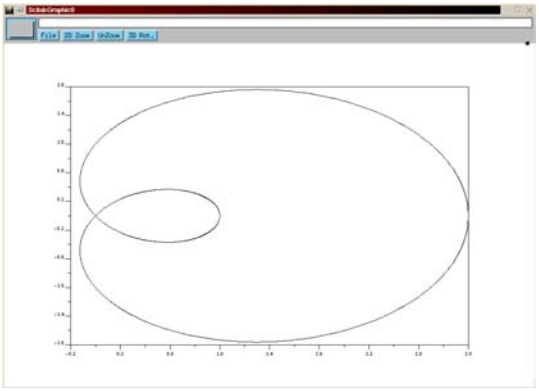
La misma función "*plot()"* sirve para construir gráficos paramétricos x=x(t), y=y(t) . En el código que sigue vemos como se hace

```
--> t=(-4:0.01:4);                  // Definición del rango de t.
--> xt = (2 * cos(t) +1) .* cos(t);  // Definicion de la funcion x(t).
--> yt = (2 * cos(t) +1) .* sin(t);  // Definicion de la funcion y(t).
--> plot2d( xt, yt );               // Ejecución del gráfico (x(t), y(t) )
```

Lo primero que podemos observar en las figuras es que el dibujo se realiza sobre una ventana gráfica. La ventana consta de un menú superior con cuatro botones:

-3D Rot.: Permite aplicar rotaciones al gráficos tridimensionales empleando el ratón.
-2D Zoom: Amplia un gráfico 2D. Este comando puede ser ejecutado recursivamente.
-UnZoom: Recupera el gráfico inicial.
-File: Abre un panel que permite imprimir el gráfico, exportarlo a un formato específico (Postscript, Postscript- LaTeX, Xfig), salvarlo como gráfico Scilab para luego cargarlo.





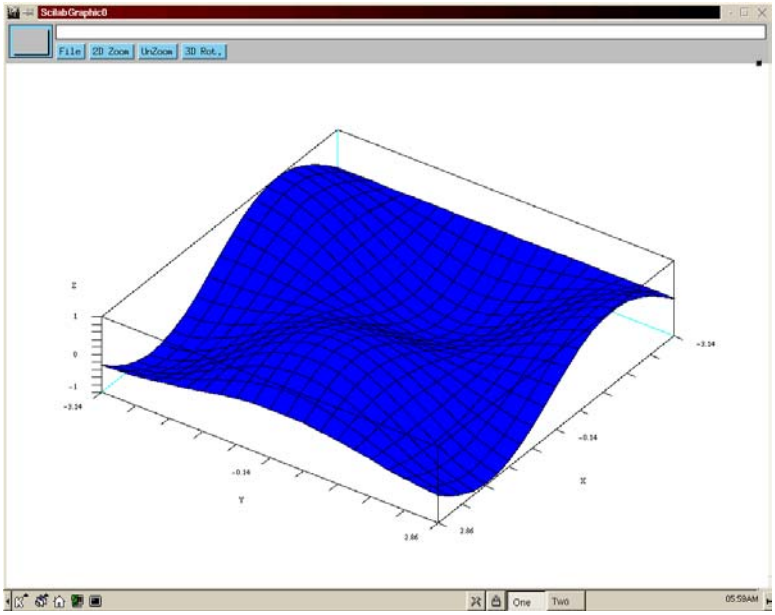
El sistema asigna a cada ventana que abre un número, de forma que pueden ser identificadas y manejarlas independientemente. Además, cada ventana tiene un contexto gráfico propio que puede ser alterado por el usuario mediante la orden *xset()*.De esta forma un mismo comando de dibujo puede dar resultados diferentes al ser ejecutados sobre diferentes ventanas. Para visualizar los valores del contexto asociado a la ventana activa podemos emplear la orden *xget()*.

Gráficos 3D. Se construyen de forma análoga con la orden "*plot3d()"*. Puede observarse que ahora hace falta especificar el rango de variación de dos parámetros y si se desea los ángulos de rotación de los ejes. La representación cartesiana de un gráfico *z=f(x,y)* sería:

```
--> x=-%pi:0.3:%pi;                  // Definición del rango de x.
--> y=-%pi:0.3:%pi;                  // Definición del rango de y.
--> plot3d( x, y, sin(x)'*cos(y), 35, 45 );  // Ejecución del gráfico (x, y, f(x,y) )
```

Es posible obtener una representación plana de un gráfico 3D mediante sus curvas de nivel, usando la función "*contour()"* en lugar del "*plot3d()"*, en este caso, se ha de especificar el número de niveles de contorno asociados.

```
--> x=-%pi:0.3:%pi;          // Definición del rango de x.
--> y=-%pi:0.3:%pi;          // Definición del rango de y.
--> contour( x, y, 10);      // Ejecución del gráfico (x(t), y(t) )
```



Por supuesto, Scilab puede emplear diferentes patrones de líneas, puntos, paletas de color, etc. Estas asignaciones definidas en el contexto gráfico antes mencionado se realizan con ordenes como: *xset, xsetech, title,* etc.

Lo que hemos visto es lo básico respecto a gráficos. La potencia y versatilidad de Scilab no se reduce a la representación de funciones en dos y tres dimensiones, también es capaz de representar conjuntos de datos en forma de nubes de puntos, modelos digitales, distribuciones estadísticas, animaciones etc.

La programación

Existen tres clases de programas que pueden ejecutarse en el entorno:

- Macros y funciones.
- Programas Scilab.
- Programas C o Fortran (Interfaces).

Estos programas se pueden construir desde el entorno, editándolos con la orden "*edit <nombre_programa>*". El editor de textos definido por defecto es el *emacs*.

Macros y funciones. Las macros y las funciones permiten realizar nuestras propias utilidades y reutilizarlas siempre que queramos. El procedimiento consiste en escribir con un editor de texto la función deseada (debe tener extensión *.sci*), cargarla desde el programa y finalmente ejecutarla. Por ejemplo:

```
--> edit mifuncion.sci
```

```
function [x,y]=mifuncion(a,b)
x=a+b
y=a-b
endfunction
```

```
--> getf('pathname/mifuncion.sci', 'c');    // Carga de la función
--> [x,y] = mifuncion(1,2)                  // Ejecución de la función
```

Un mismo fichero puede contener la definición de varias funciones, e incluso definir funciones sobrecargadas. Scilab maneja sus propias librerías de funciones, estas se sitúan en los subdirectorios de /usr/local/macros/. El usuario también puede organizar sus funciones en librerías mediante el comando "*lib*" .

Programas. Los programas Scilab también son ficheros de ordenes con extensión *.sci,* que se ejecutan con el comando

```
--> exec ordenes.sci
```

El programa busca el fichero "*ordenes.sci"* y lo ejecuta secuencialmente hasta que encuentre alguna sentencia de control de flujo. Las estructuras de control existentes y otras primitivas de programación pueden verse con la sentencia "*what*":

```
--> what
 Commands:
 if      else    for     while   end     select  case    quit    exit
 return  help    what    who     pause   clear   resume  then    do
 apropos abort   break   elseif
```

No vamos a analizar aquí con detalle las estructuras de programación, las ayudas en línea *help* y *apropos*, proporcionan información suficiente como para que no haya problemas a la hora de utilizarlas. Para los conocedores de la programación en C o Matlab, la programación Scilab no supondrá ningún problema, ya que la mayor parte de las sentencias emulan a las correspondientes en estos lenguajes.

Otro aspecto de la programación Scilab es el que relaciona los programas en Scilab con la programación en Matlab y Mapple (actualmente entre los más potentes y conocidos programas de computación numérica). Scilab ha previsto herramientas que permiten sustituir el código de los paquetes Matlab o Mapple por sus correspondientes funciones o primitivas en Scilab. Supongamos que queremos convertir un fichero Matlab denominado "*mifuncion.m*" a código Scilab solo hay que hacer

```
--> mfile2sci mifuncion.m
--> getf mifuncion.sci
```

La orden crea el fichero, "*mifuncion.sci*" generando el código traducido además de un segundo fichero "*mifuncion.cat*". En algunos casos la traducción es imperfecta o sencillamente no puede hacerse de ahí la presencia del segundo fichero que es un fichero de ayuda asociado a la traducciones imperfectas. No obstante, no cabe duda de que esta herramienta ahorra un importante trabajo a la hora de la migración de programas entre estos entornos.

Se necesita un segundo comando para cargar la función traducida en memoria preparándola para ser ejecutada ("*getf mifuncion.sci"*), a partir de este momento podría emplearse como cualquier otra función de Scilab.

Interfaces. Una subrrutina en código C o Fortran se puede utilizar como una función de Scilab, a este código se le denomina una *Interface*. Evidentemente esta posibilidad permite codificar programas muy exigentes en términos de complejidad o velocidad de calculo, además de la reutilización de código ya existente en estos lenguajes. El siguiente ejemplo ilustra como realizar esto con una función escrita en lenguaje C.

```
/* -- ------------------------------------------------------------------------
   -- Dado un vector (a) de tamaño  (n) se obtiene el vector
   -- tipificado (long unidad) si la cadena de caracteres ch='yes',
   -- en caso contrario se devuelve el mismo vector de entrada.
   -- ------------------------------------------------------------------------ */

#include <math.h>
#include <string.h>

int prueba(ch, n, a, c)
   char *ch;
   int *n;
   double *a, *c;
  {
  static int k;
  static double longitud;

  if (strcmp(ch, "yes") == 0){
     longitud = 0.0;
     for (k = 0; k < *n; ++k)
       longitud = longitud + a[k] * a[k];

        for (k = 0; k < *n; ++k)
          c[k] = a[k]/sqrt(longitud); }
  else {
    for (k = 0; k < *n; ++k)
          c[k] = a[k]; }

return(0); }
```

Una vez creado el programa "*prueba.c*", se usa la primitiva "*link()*" desde dentro de Scilab para crear un link dinámico (incremental) entre el código C y Scilab.

```
--> host('gcc -c prueba.c');          // Compila el programa.
--> link('./prueba.o','prueba','C');  // Linka el programa.
```

La sentencia "*call()*" ejecuta el programa, necesita como argumentos el nombre de la función a ejecutar, los argumentos de la función de entrada y de salida. Cada parámetro de entrada o salida además debe contener información de la posición que ocupa en la llamada y del tipo de dato que representa ('c' carácter o cadena, 'i' entero, 'd' double, 'r' real):

```
--> A = [1,2,3,4,5]; n=size(A,'*' );
--> c = call('prueba', ...     // Utiliza la función denominada "prueba".
          'yes',  1,  'c', ...     // El argumento 1 es la cadena 'yes'.
           n,     2,  'i',  ...     // El argumento 2 es el entero n.
           A,     3,  'd',  ...     // El argumento 3 es el array double A.
          'out',            ...    // La argumento 4 es un array de
          [1,n], 4, 'd' )        //  salida de dimensión n.
c = !   0.1348400   0.2696799   0.4045199   0.5393599   0.6741999   !
```

Existen muchos ejemplos ilustrativos en el directorio "*examples/link-examples*" además, la dirección "*http://www-rocq.inria.fr/scilab/doc/intro/*" contiene una guía completa del uso de las Interfaces.

Los toolboxes

Scilab dispone en la actualidad de un amplio abanico de librerías adicionales que amplían el software, estos programas denominados "*toolboxes*" cubren áreas específicas en los campos de la matemática, la ingeniería, simulación, etc. En la tabla que sigue se describe algunos de ellos:

Toolbox	Descripción
ANN	Análisis de Redes Neuronales.
EVOL	Algoritmos Evolutivos.
FABBRI	Manipulación de Imágenes.
FEM_Post	Detección de Fallos.
FISLAB	Inferencia en Lógica Borrosa.
FREEFEM	Elementos Finitos 2D.
FSQP	Procesos de Optimización.
HMM	Modelos de Markov.
LIPSOL	Programación Lineal.
Plotting library	Gráficos al estilo Matlab.

Los toolboxes están disponibles en "*http://www-rocq.inria.fr/scilab/contributions.html*". Su instalación es fácil, ya que generalmente se distribuyen en formato *tar.gz*, basta con descomprimirlo y posiblemente alterar el fichero .scilab para que cargue la nueva librería al entrar en el programa. Además el entorno también dispone de un entorno gráfico interactivo denominado SCICOS que permite el modelado y la simulación de sistemas dinámicos.

Conclusión.

Se ha intentado abordar la computación numérica presentando un software capaz de competir en calidad y eficiencia con aplicaciones comerciales de alto nivel como Mapple y Matlab, con las ventajas añadidas de ser multiplataforma y gratuito. Este documento ha procurado dar una visión de conjunto suficientemente amplia para iniciarse en el manejo del programa de manera que cualquier técnico o científico que comience a utilizarlo no tarde en descubrir las múltiples posibilidades de este entorno.