CAPITULO 1

INTRODUCCION A SCILAB

1.1. INTRODUCCION AL SOFTWARE LIBRE

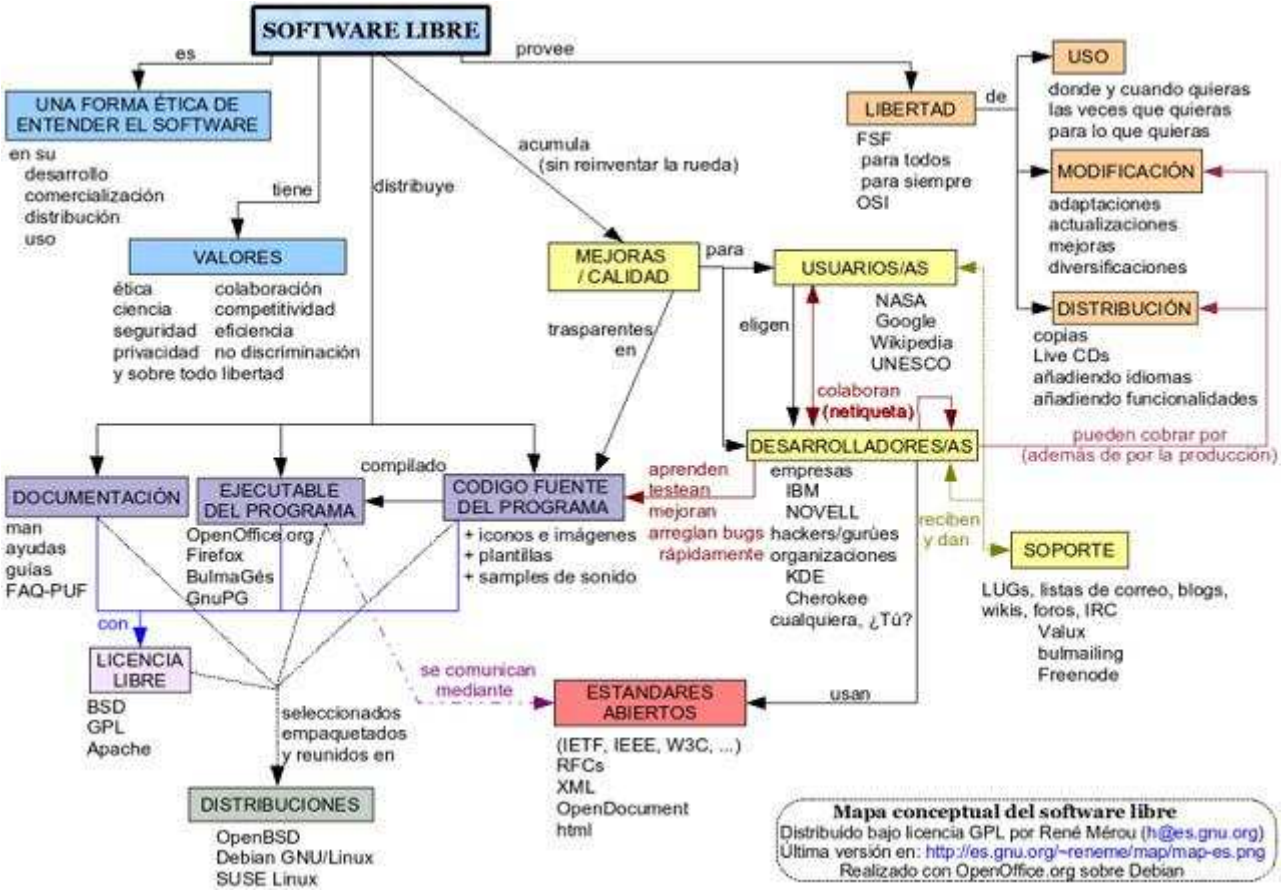

Fig.1.1.Mapa conceptual del software libre.

En esta introducción trataremos de explicar que es el software libre y conceptos importantes para la comprensión del mismo [2] [3]. (Ver fig.1.1).

En 1985 Richard Stallman fundó la Free Software Foundation (**FSF**) con el fin de conseguir fondos para el desarrollo y la protección del software libre, y sentó los fundamentos éticos del mismo, con documentos como "*The GNU Manifesto*" y "*Why Software Should Not Have Owners*".

El término software libre se refiere a libertad, tal como fue concebido por Richard Stallman en su definición. En concreto se habla de cuatro libertades:

- Libertad para ejecutar el programa en cualquier sitio y con cualquier propósito (privado, educativo, público, comercial,…).

- Libertad para estudiarlo y adaptarlo a nuestras necesidades. Esto exige el acceso al código fuente.

- Libertad de redistribución, de modo que se nos permita colaborar con vecinos y amigos.

- Y libertad para mejorar el programa y publicar las mejoras. Esto también exige el acceso al código fuente.

Estas libertades están protegidas de acuerdo con la legalidad vigente por medio de una licencia. En ella se plasman las libertades, pero también son posibles restricciones compatibles, como por ejemplo dar crédito a los autores originales si redistribuimos.

Por otro lado, vemos que no estamos hablando de software gratuito, y el software libre se puede vender si se desea. Pero debido a la libertad de redistribución, cualquiera puede redistribuirlo sin pedir dinero a cambio ni permiso a nadie, lo que hace prácticamente imposible obtener dinero de ello.

Más adelante, en 1998, se creó la Open Source Initiative (**OSI**), que decidió adoptar el término "*open source software"* (software de fuente abierta) como una marca para introducir el software libre en el mundo comercial, tratando de evitar así la ambigüedad que en inglés supone el término "*free software"* (que significa tanto libre como gratis).

Son dos por tanto las principales motivaciones para el desarrollo de software libre:

- La motivación ética, abanderada por la FSF, y partidaria del apelativo "*free"*, que argumenta que el software es conocimiento y debe poderse difundir sin trabas.

- Y la motivación pragmática, abanderada por la OSI, partidaria del apelativo "*open source"*, que argumenta ventajas técnicas y económicas.

La principal diferencia entre los términos "*open source"* y "*free"* es que éste último tiene en cuenta los aspectos éticos y filosóficos de la libertad, mientras que el primero sólo los aspectos técnicos.

En un intento por aunar los mencionados términos que se refieren a conceptos semejantes, se está extendiendo el uso de la palabra "*FLOSS"* (Free-Libre-Open Source Software).

Aparte de las dos grandes motivaciones comentadas, la gente que trabaja en software libre puede hacerlo por muchas otras razones como la simple diversión.

En definitiva, estrictamente hablando, lo que diferencia al software libre del resto del software es un aspecto legal: la licencia antes mencionada. Se trata, en palabras de uso común, de un contrato entre el autor o propietario de los derechos y los usuarios, que estipula lo que éstos últimos pueden hacer con su obra y en qué condiciones.

Para el software libre existen distintos tipos de licencias que garantizan las cuatro libertades anteriormente citadas [3] [4]:

- La Licencia Pública General, GPL
- La licencia tipo Berkeley Software Distribution, tipo BSD
- Y la licencia tipo Mozilla Public License, tipo MPL

La **GPL** es con diferencia la licencia más popular y conocida de todas las licencias del mundo del software libre y su autoría corresponde a la FSF. Está pensada para asegurar la libertad del código en todo momento, ya que un programa publicado y licenciado bajo sus condiciones nunca podrá ser hecho propietario. Es más, ni ese programa ni modificaciones del mismo pueden ser publicadas con una licencia diferente a la propia GPL. Los partidarios de las licencias tipo BSD ven en esta cláusula un recorte de la libertad, mientras que sus seguidores ven en ello una forma de asegurarse que ese software siempre va a ser libre.

Está permitido realizar modificaciones sin restricciones, aunque, como ya hemos dicho, sólo se pueda integrar código licenciado bajo GPL con otro código que se encuentre bajo una licencia idéntica o compatible. Esto ha venido a llamarse el *efecto viral* de la GPL, ya que el código publicado una vez con esas condiciones nunca puede cambiar de condiciones.

En líneas básicas, la licencia GPL permite la redistribución binaria y la de los códigos fuentes, aunque, en el caso de que redistribuya de manera binaria, obliga a que también se pueda acceder a los códigos fuentes.

La **BSD** la única obligación que exige es la de dar crédito a los autores, mientras que permite tanto la redistribución binaria, como la de los códigos fuentes, sin obligar a ninguna de las dos en ningún caso. Asimismo, da permiso para realizar modificaciones y ser integradas con otros programas casi sin restricciones.

La licencia BSD es ciertamente muy popular, como se puede ver a partir del hecho de que existen varias licencias de condiciones similares: XWindow, Tcl/Tk, Apache,…, que se han venido a llamar licencias tipo BSD.

La **MPL** [5] por otro lado se utiliza en gran cantidad de software libre de uso cotidiano.

La licencia MPL no es tan excesivamente permisiva como las licencias tipo BSD pero se considera cercana a ellas. En realidad, fue la primera licencia nueva después de muchos años, que se encargaba de algunos puntos que no fueron tenidos en cuenta por las licencias BSD y GPL.

1.2. SCILAB Y OTROS PROGRAMAS DE SOFTWARE LIBRE

Entre las herramientas de software libre para cálculo numérico podemos encontrar las siguientes como principales: Octave, Scilab, Scipy y RLab. Todas ellas con un gran parecido al clásico Matlab [6] [7]. Como sabemos, **Matlab** [8] es el software por excelencia en cuanto a cálculo numérico, sin embargo, tiene una gran desventaja: es demasiado caro. Nuestra alternativa por tanto puede ser el software libre anteriormente citado.

Octave [9] es prácticamente un clon de Matlab, por lo que programas escritos en Matlab pueden pasarse casi de forma directa a Octave. La primera versión alpha apareció en 1993 y actualmente se encuentra en la versión 2.1.73 que se distribuye con licencia GPL. Su principal desarrollador es John W. Eaton.

Scilab [10] es el software sobre el que se trabajará en este proyecto. Es un paquete de software científico para computación numérica que provee un entorno de trabajo abierto y potente para aplicaciones científicas y de ingeniería. Ha sido desarrollado desde 1990 por investigadores pertenecientes al **INRIA** (Institut National de Recherche en Informatique et Automatique) y al **ENPC** (École Nationale des Ponts et Chaussées). Su distribución comenzó en 1994 con sus programas fuente puestos a disposición de todos en Internet, y actualmente se utiliza en el mundo entero en la industria, la investigación y la educación. Desde Mayo de 2003, es mantenido y desarrollado por **Scilab Consortium**, que engloba a centros de investigación y empresas, en concreto veintitrés son los miembros que lo forman. La versión actual es la 4.1 y aunque su licencia no es exactamente GPL, las diferencias sólo afectarían a los usos comerciales de versiones de Scilab modificadas por el usuario. La sintaxis de Scilab es también similar a la de Matlab, tal vez menos compatible que Octave, pero con más funcionalidades.

Scipy [11] por otro lado, es un proyecto más joven, comenzó su desarrollo en 2001 y está basado en el lenguaje de programación Python. Sus orígenes se remontan al paquete con extensiones numéricas para Python denominado Numeric. Posteriormente apareció Numarray, con la intención de construir un paquete más flexible y de limpiar el código, aunque resultó ser más lento para cálculos matriciales en pocas dimensiones. En el año 2005, el principal impulsor de Scipy, Travis Oliphant, quiso unificar ambos paquetes en un único paquete que integrase sus respectivas ventajas, éste se denominó Numpy, pudiéndose considerar como el núcleo de Scipy. Scipy en sí mismo, se concibe actualmente como una extensión de las funcionalidades de Numpy. La versión actual es la 0.5.2 para Scipy, y la 1.0.1 de Numpy. Scipy está patrocinado por una empresa, **Enthought** (Scientific Computing Solutions) y se ofrece con

licencia BSD. Es el programa más diferente de Matlab de los anteriormente citados.

Rlab [12], finalmente, según su autor no intenta ser una copia de Matlab sino que por el contrario, intenta tomar sus mejores características y proporcionar una sintaxis y semántica mejoradas.

En mayo de 2006 el departamento de Ingeniería Electrónica y Comunicaciones de la Universidad de Zaragoza realizó un estudio comparativo [7] de las herramientas de software: Octave, Scilab, Scipy y Matlab, en el que se puntuó a todas ellas en diferentes aspectos. Este estudio se realizó con las versiones 2.1.72 para Octave, 3.1.1 para Scilab, 0.4.6 para Scipy y 6.0 para Matlab, y algunas de las conclusiones que pueden resultar más interesantes para el usuario y que no han sido mencionadas todavía se resumen a continuación:

- Todas las herramientas presentan las funcionalidades básicas que permiten trabajar con matrices. Asimismo se puede trabajar con funciones, encontrar ceros y mínimos. Los polinomios también existen en todos los programas, así como funciones elementales de procesado de señal como la convolución, correlación, transformada de Fourier,…. Se puntuó por igual a todas las herramientas.

- En cuanto a las funcionalidades avanzadas, en Octave es necesario descargar paquetes adicionales para obtenerlas, y entre librerías avanzadas y paquetes podemos encontrar 62, un número que se considera elevado. Scilab posee hasta 87 paquetes, incluyendo contribuciones de usuarios. Scipy junto con Numpy tiene 24 paquetes, quedando por tanto por debajo de los demás.

- En Octave podemos dibujar gráficos en 2D y 3D y también editar imágenes. Con Scilab también podemos editar gráficos, imágenes,…. Por su parte, Scipy ha decidió recientemente no incluir directamente los gráficos, y dejar que esta funcionalidad se apoye en otras herramientas. En este apartado Scipy se colocó por debajo del resto de herramientas.

- En cuanto a fallos o cuelgues durante la ejecución de programas Matlab y Octave apenas tienen. Para Scilab, no se concretó nada. Y para el caso de Scipy al ser un proyecto más joven se concluyó que tenía más fallos, aunque no eran críticos en ningún caso.

- En cuanto a rapidez Matlab y Scipy se sitúan por encima de Octave y Scilab.

- También el estudio evaluó la información a disposición del usuario para el uso de los programas. Matlab es de sobra conocido. Octave tiene buenos manuales de introducción y existen numerosas páginas Web aunque se echaron en falta ejemplos aclaratorios en la ayuda del programa. En Scilab la ayuda del programa es buena y fácil de usar, pero no se encontraron muchos manuales ni páginas Web en castellano. Scipy se consideró el peor de todos, ya que la ayuda del programa no es muy explícita.

- Octave es muy fácil de usar si se conoce previamente Matlab. En Scilab, la línea de comandos es algo más peculiar. Respecto a Scipy, su uso es el más diferente de Matlab.

- En cuanto a desarrollo y madurez todas las herramientas están activas y no hay cambios dramáticos de una versión a otra. En las tres herramientas de software libre, podemos encontrar versiones no estables (no probadas todavía), pero con los últimos cambios que los desarrolladores han incluido. Esto permite a los usuarios acceder a las funcionalidades más nuevas. Quizás, Scipy entrega versiones con demasiada frecuencia y por ello se colocó ligeramente por debajo de las demás en el estudio.

- Las versiones en Windows se instalan sin problemas. En el caso de utilizar Linux la instalación puede complicarse en algunos casos. En este apartado se colocó a Octave el primero y a Scilab el último.

En conclusión, en lo que se refiere a Scilab se concluyó que se trata de un entorno muy completo, con comandos bastante parecidos a MatLab pero que ha creado sus propias interfaces gráficas, su propio lenguaje,…, la valoración fue muy positiva.

1.3. COMO CONTRIBUIR A SCILAB

Como ya dijimos, a partir de Mayo de 2003, Scilab pasó a ser mantenido por **Scilab Consortium** cuyos objetivos principales hoy en día son:

- Organizar la cooperación e intercambio entre los **desarrolladores** de Scilab, con vistas a incorporar dentro del programa los últimos avances científicos en el área de computación numérica.

- Y organizar la cooperación e intercambio entre **usuarios** de Scilab de forma que el programa pueda ser utilizado en forma efectiva en cualquier ámbito.

Desde el punto de vista del usuario, Scilab presenta algunas ventajas que conviene recalcar en este punto, aunque muchas ya se puedan suponer por lo comentado hasta el momento:

- Disponibilidad de la última versión vía Internet. Scilab está disponible de forma gratuita la Web oficial [10] donde puede encontrarse también numerosa documentación así como muchas otras obras y colaboraciones. Normalmente, todos los materiales pueden ser utilizados gratuitamente, manteniendo solamente los créditos y referencias correspondientes a los autores.

- El programa puede ser utilizado, copiado y distribuido de forma legal.

- Los resultados obtenidos pueden ser divulgados sin restricción.

- Se tiene acceso al código fuente.

- Y la certeza de estar participando de una comunidad cuyo principal objetivo es la difusión sin restricciones del conocimiento.

Al igual que podemos descargar contribuciones de la Web oficial [10], manuales por ejemplo, también podemos subirlas. Como usuario, hay muchos modos de contribuir a Scilab, aportando funciones propias, cajas de herramientas (toolboxes),….

Las condiciones e instrucciones a seguir en caso de querer contribuir podemos encontrarlas en [13]. Resumiendo, para contribuir habrá que acceder a [14], ventana de la fig.3.1, rellenar los campos, y subir la aportación in situ o mencionando el sitio Web de donde puede ser descargada.

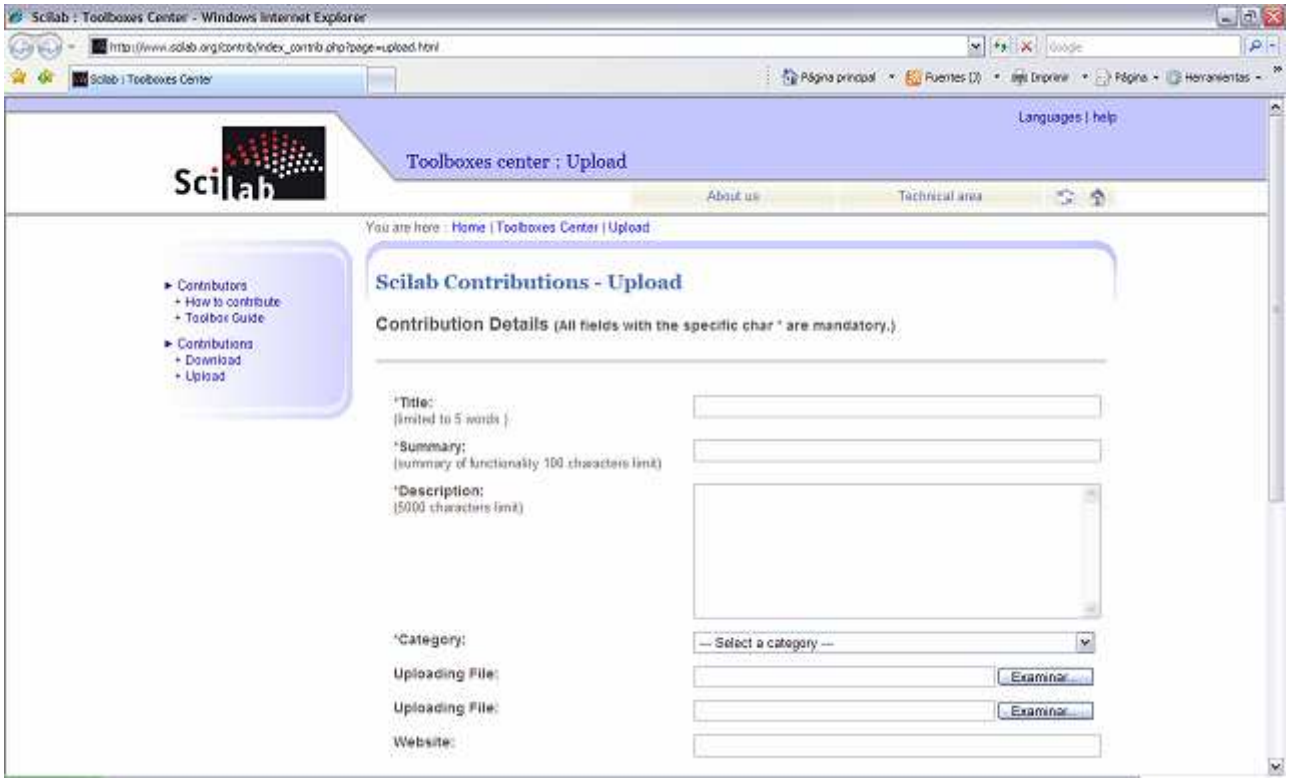

Fig.3.1. Scilab Upload

El campo **Sumario** deberá contener una breve descripción de la contribución y mencionar las exigencias de instalación.

El campo de **Descripción** deberá mencionar a los autores de la contribución, y si la contribución está basada en bibliotecas/programas externos sus autores también deben ser nombrados.

Es muy importante no olvidar mencionar las restricciones de la licencia y de los derechos de autor siempre que existan. Y también hay que asegurarse de que se tiene el derecho a redistribuir librerías y programas incluidos en nuestra contribución.

Finalmente debe proporcionarse una dirección de correo electrónico válida para que el equipo de Scilab pueda ponerse en contacto con el contribuyente.

Toda contribución que se suba podrá ser difundida posteriormente por el grupo Scilab.

Además, Scilab Consorcium organiza concursos con el fin de incentivar las contribuciones al software. El último anunciado en la Web oficial [10] fue: *2007 SCILAB WORKSHOP IN CHINA*, concurso organizado junto con entidades Chinas y dirigido a personal académico, estudiantes de universidad e instituto

de investigación, ingenieros de organizaciones industriales y de negocio, y otros profesionales. Las bases, premios y demás detalles de este concurso se especifican en [15].

1.4. INSTALACIÓN Y PUESTA A PUNTO DE SCILAB

Como ya es sabido en la Web pueden encontrarse manuales, también FAQ´s (frequent asked questions), diferencias con Matlab,… y por supuesto podemos descargar Scilab desde ella. Su posterior instalación en el Sistema Operativo WINDOWS será sencilla:

Paso 1: Descargar el programa. La Web oficial tendrá el aspecto de la fig.4.1.

Fig.4.1. Aspecto pág. Oficial de Scilab.

Solo tenemos que pinchar sobre **Download Scilab 4.1** y en las siguientes ventanas, fig.4.2 y fig.4.3, elegir **Guardar** y donde queremos guardar el archivo de instalación (En **Mis documentos** por ejemplo).

Fig.4.2. Ventana de descarga.

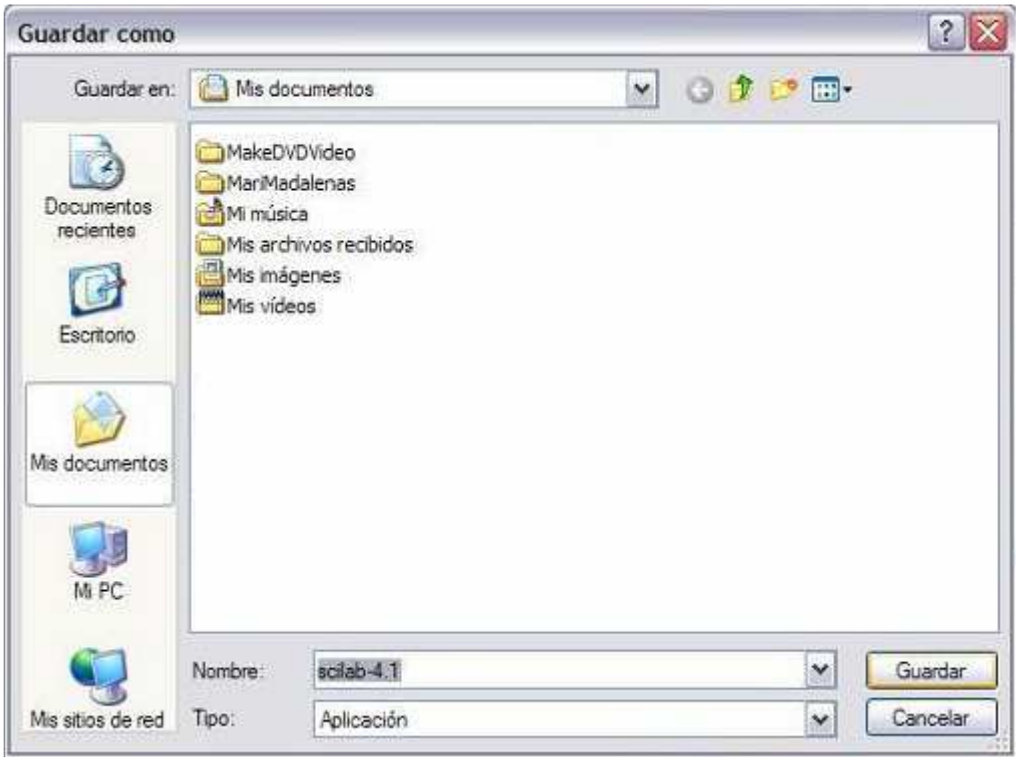

Fig.4.3. Dónde descargar

Solo quedará esperar, fig.4.4, y en unos minutos tendremos guardado Scilab.

Fig.4.4. Ventana de espera

Paso 2: Instalación. Si hemos seguido el paso 1 correctamente encontraremos ahora en **Mis documentos** el icono de la fig.4.5.

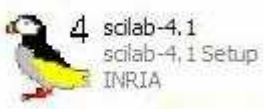

Fig.4.5. Icono de instalación.

Al hacer doble clic sobre él aparecerá una ventana como la de la fig.4.6. Seleccionaremos el idioma.

Fig.4.6. Ventana de selección de idioma.

A continuación saltará la ventana de bienvenida, la de la fig.4.7, donde pinchamos **Next**.

Fig.4.7. Ventana de bienvenida.

Ahora vendrá la ventana que hace referencia al acuerdo de licencia, ventana de la fig.4.8, y para continuar damos clic en el campo **I accept the agreement** y después en el botón **Next** otra vez.

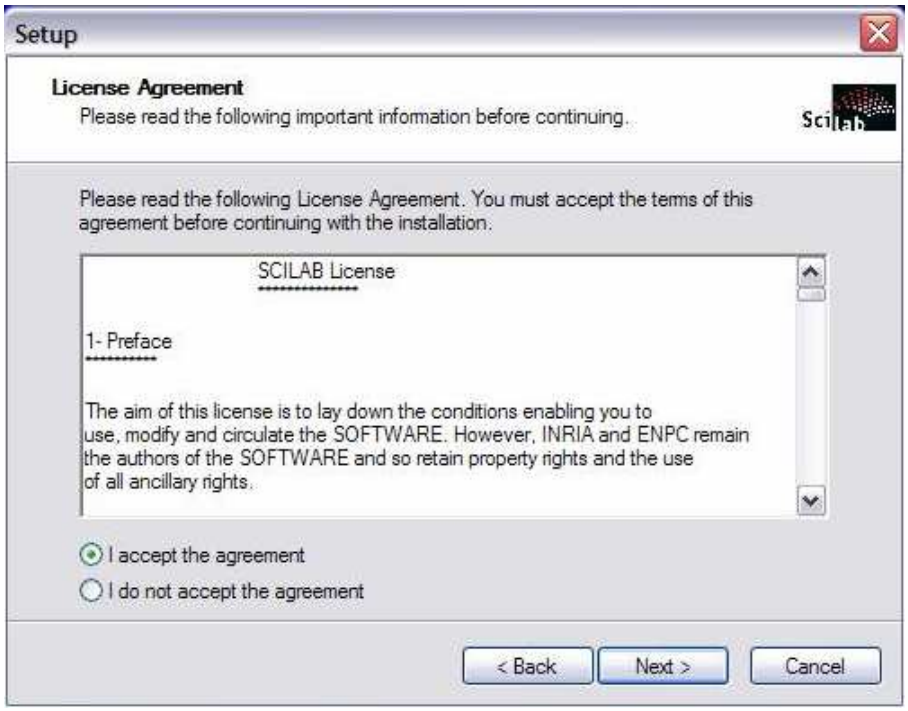

Fig.4.8. Acuerdo de licencia.

La siguiente ventana será la de la fig.4.9, donde se seleccionará el directorio o ruta en la que Scilab quedará instalado. Podemos seleccionar la ruta que queramos o usar la ruta predeterminada C:\Archivos de programa\scilab-4.1, para continuar damos clic en **Next**.

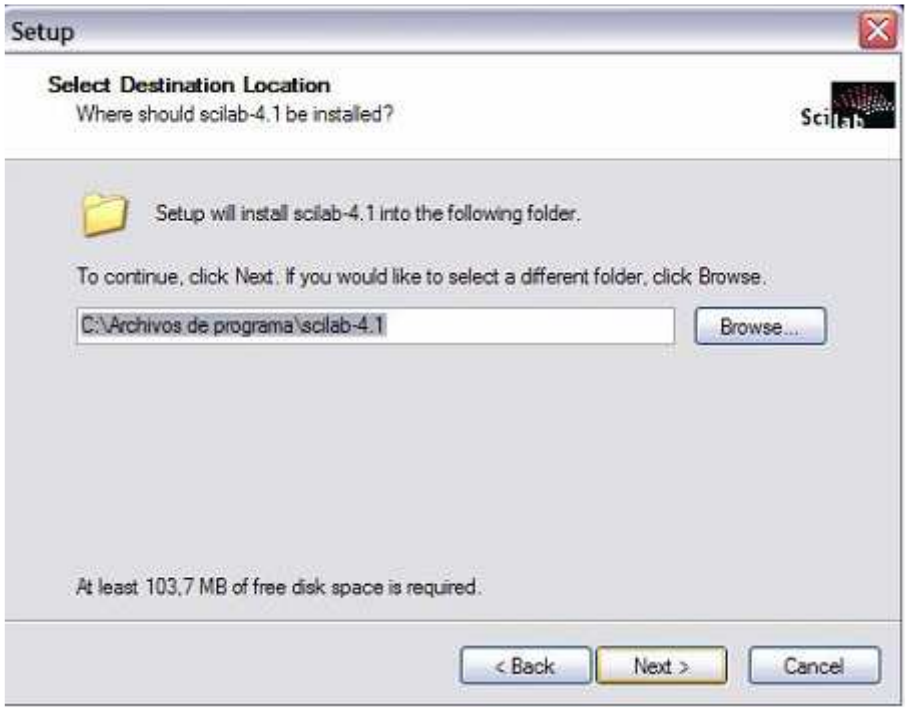

Fig.4.9. Destino de instalación.

Hecho lo anterior, la siguiente ventana es la de la fig.4.10, en la que elegiremos los componentes que queremos incluir en nuestra instalación. Si elegimos la instalación por defecto no marcaremos ningún campo y pulsaremos **Next**.

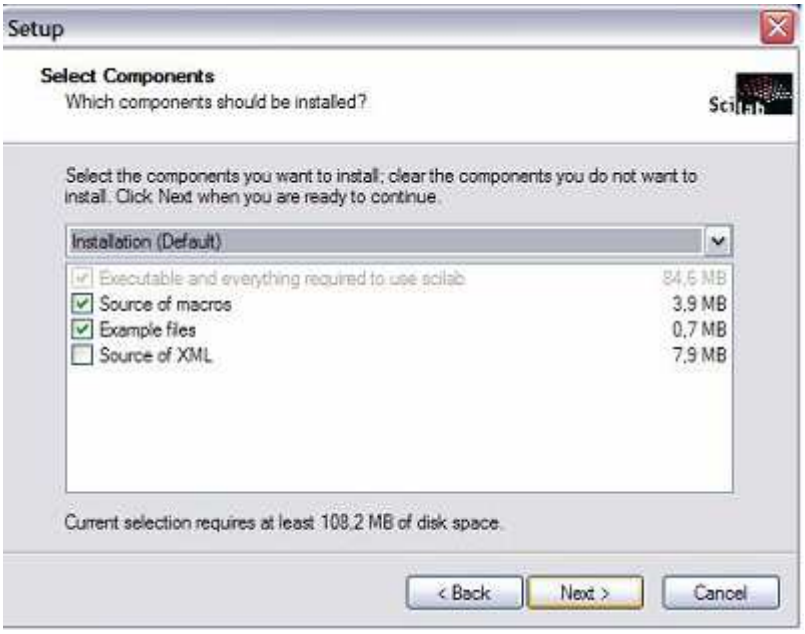

Fig.4.10. Selección de componentes.

En las dos ventanas siguientes pulsamos **Next** también, y con ello estaremos creando un acceso directo al programa en el escritorio.

La posterior ventana, fig.4.11, nos mostrará un resumen de todo lo que hemos seleccionado anteriormente. Pincharemos en **Install** si todo es correcto y esperaremos unos minutos, fig.4.12.

Fig.4.11. Ventana resumen.

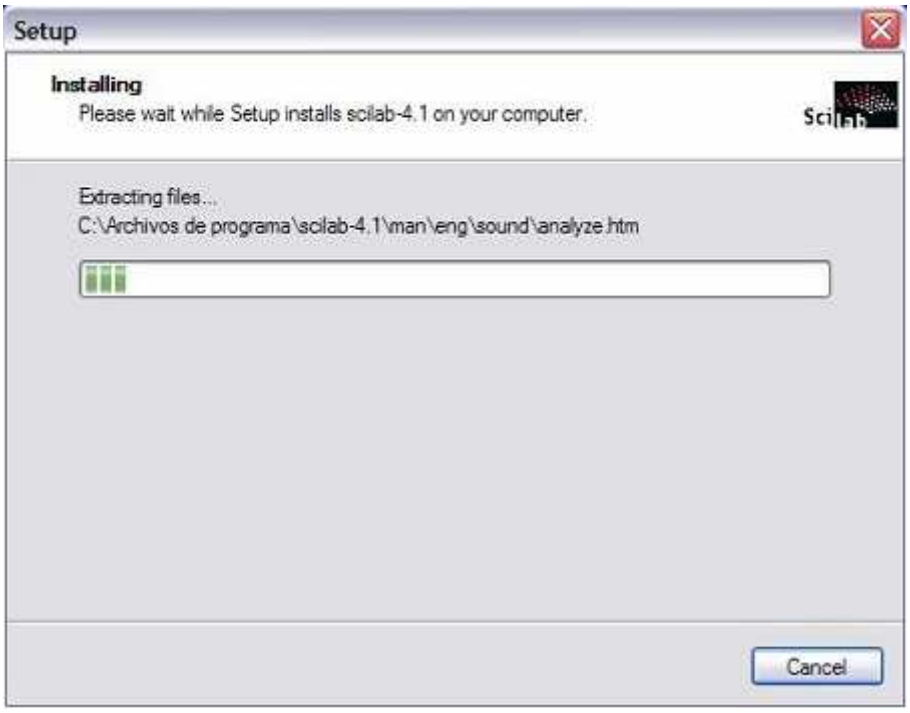

Fig.4.12. Ventana de espera.

(La siguiente ventana es de información y pincharemos **Next)**.

Finalmente, la ventana que aparecerá será la de la fig.4.13. Damos **Finish** y el programa queda así perfectamente instalado.

4.13. Instalación completada.

1.5. BREVE MANUAL DE SCILAB

Scilab [9] contiene centenares de funciones matemáticas y la posibilidad de añadir interactivamente al sistema programas escritos en los lenguajes más usados (Fortran, C, C++,…). Tiene estructuras de datos sofisticadas (como listas, polinomios, funciones racionales, sistemas lineales…), un intérprete y un lenguaje de programación de alto nivel. Scilab fue concebido para ser un sistema abierto donde el usuario pueda definir nuevos tipos de datos y operaciones sobre estos tipos de datos.

El sistema provee numerosas herramientas:

- Gráficos 2-D y 3-D.
- Álgebra lineal.
- Polinomios y funciones racionales.
- …

En este apartado se intentará confeccionar un pequeño manual para el manejo básico de las herramientas y operaciones de Scilab, que lejos de ser exhaustivo podrá ser muy útil.

1.5.1. Primeros pasos con Scilab

Al activar Scilab aparece por pantalla la ventana de la fig.5.1.1

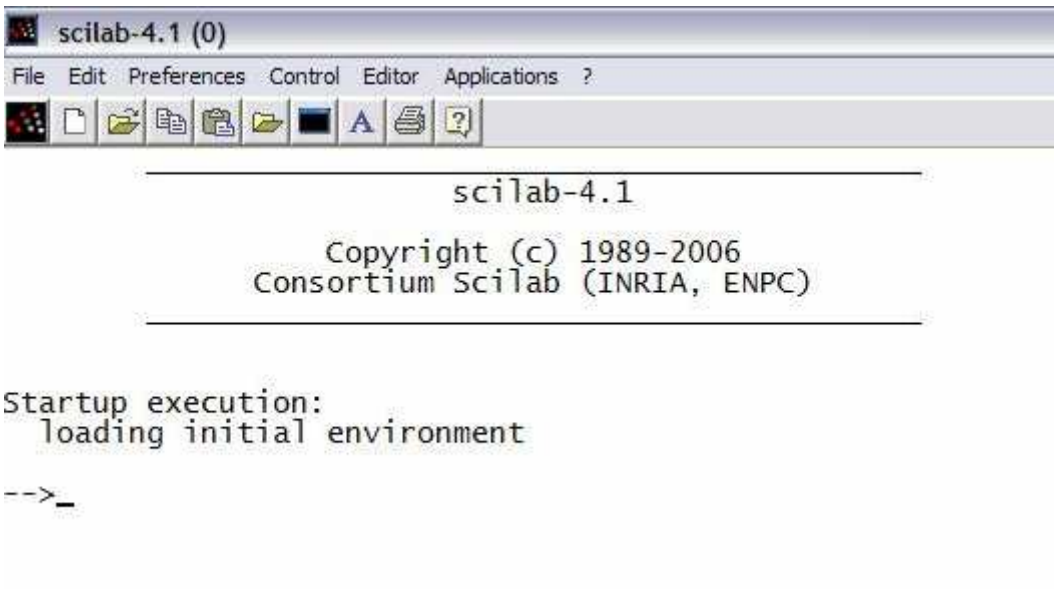

Fig.5.1.1.Entorno Scilab.

Ya podemos escribir al frente de → las ordenes de Scilab y activarlas pulsando Intro.

Si escribimos y activamos

→t=2.5

Scilab crea la variable t y le asigna el valor 2.5. Además, muestra esta asignación por pantalla, lo que no ocurrirá si acabamos las órdenes en punto y coma. La orden tendrá efecto en ambos casos.

Es importante saber que Scilab distingue entre mayúsculas y minúsculas.

Si se da la orden

→t = 2*t

se utiliza el antiguo valor de t para la operación, pero después de la orden, t queda valiendo 5.

También es posible, por medio de las flechas hacia arriba y hacia abajo, buscar una orden anterior para editarla y activarla. A veces necesitaremos repetir órdenes.

Siguiendo con las órdenes, en una misma línea de Scilab puede haber varias. Estas deben estar separadas por coma o por punto y coma. Por ejemplo:

→t1 = 2 , t2 = 3 ; t3 = t2-t1

En las ordenes de Scilab los espacios en blanco entre signos no son indispensables, simplemente sirven para facilitar la lectura.

Por otro lado, los símbolos

+ - * /

sirven para las 4 operaciones aritméticas básicas, y el signo **-** además, para indicar el inverso aditivo.

Para elevar a una potencia tenemos dos opciones

^ **

Para la raíz

sqrt ( )

Scilab utiliza para agrupar los paréntesis redondos

( )

También tiene predefinidas algunas constantes especiales cuyos nombres están precedidos del signo %. Para los valores e, π, $\sqrt{-1}$, sus nombres son: **%e**, **%pi**, **%i.**

Scilab tiene predefinidas muchas funciones matemáticas. Algunas muy elementales y comunes como: **abs** (valor absoluto), **acos** (arcocoseno), **acosh** (arcocoseno hiperbólico),….Cómo se usan lo vemos con un ejemplo:

→i = -5
→j= abs (i)

Para conocer más funciones, pues hay muchas, y más información sobre ellas, existen manuales muy completos en la Web [10].

También, para tener información mas detallada sobre alguna función en concreto, puede usarse la ayuda del programa. Basta con escribir **help** y a continuación el nombre de la función, por ejemplo:

→help abs

Si no se conoce el nombre de la función, pero se desea buscar sobre un tema, se debe utilizar **apropos**. Por ejemplo:

→apropos matriz

1.5.2. Complejos

Scilab maneja de manera sencilla los números complejos. Vemos con ejemplos dos formas de definir un mismo complejo:

→a = 3 + 4*%i
→a = 3 + sqrt (-4)

Cualquiera de las dos formas es válida y definen a = 3 + 4i.

Para las operaciones con números complejos (suma, resta, multiplicación,...) usamos los mismos símbolos ya comentados.

Las funciones **real**, **imag** y **conj** permiten obtener la parte real, la parte imaginaria y el conjugado de un complejo respectivamente. Si se utiliza la función **abs** con un complejo, se obtiene la magnitud o módulo del mismo.

1.5.3. Polinomios

Un polinomio se puede definir por sus coeficientes o por sus raíces con la orden **poly**. Si escribo

→p = poly ([1 2 3 4], "x", "coeff")

defino en la variable p el polinomio $1 + 2x + 3x^2 + 4x^3$.Y si escribo

→q = poly ([1 2 4], "x", "roots")

defino en la variable q el polinomio $-8+14x-7x^2+x^3$, cuyas raíces son exactamente 1, 2 y 4.

Escribir

→q = poly ([1 2 4], "x")

produce exactamente el mismo resultado que el ejemplo anterior, o sea, "roots" es el tipo de definición por defecto.

La doble comilla " puede ser reemplazada por la comilla sencilla ' .Y más aún, se puede reemplazar 'coeff' por 'c' y 'roots' por 'r'.

Hay muchas funciones aplicables a polinomios. Vemos tres: **roots**, **coeff** y **horner**.

La función roots calcula las raíces de un polinomio. Por ejemplo, con

→roots (q)

obtendremos como resutado 1, 2 y 4.

La función coeff hace referencia a los coeficientes de los polinomios. Por ejemplo, con

→k = coeff (q, 2)

se asigna a la variable k el valor -7, el coeficiente de x^2 en el polinomio q. Y con

→c = coeff (q)

se asignará a c todos los coeficientes de q. Siendo c, por tanto, un vector.

La orden **horner** se usa para evaluar un polinomio p en un valor t (un número u otro polinomio). Ejemplo:

→t = 1
→horner (q, t)

El resultado, si todo va bien, debe darnos 0 en este caso.

Por último decir que con los polinomios se pueden hacer sumas, multiplicaciones, restas,… siempre que sean polinomios en la misma variable.

→v = p + q + p^2 – 3*p*q

1.5.4. Matrices y vectores

¿Cómo definir una matriz? Una de las formas (hay varias) es la siguiente:

→a = [1 2 3 ; 4 5 6 ; 7 8 9]

Con esto hemos definido una matriz cuadrada 3x3 llamada **a**. De forma equivalente podrían definirse un vector fila **u** y un vector columna **v**:

→u = [1 2 3]
→v = [4 ; 5 ; 6]

Scilab también permite crear algunos tipos especiales de matrices:

- **ones** (2,4) crea una matriz de unos de tamaño 2x4.
- **zeros** (4,2) crea una matriz de ceros de tamaño 4x2.

- **rand** (15,30) crea una matriz aleatoria de tamaño 15x30.

- **eye** (6,6) crea la matriz identidad de orden 6x6

También, a veces es útil crear vectores con elementos igualmente espaciados, como el vector **w**:

→w = [1 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6]

Esto también puede hacerse así

→w= 1:0.2:2.6

Ahora veremos, a través de ejemplos, algunas de las cosas que podemos hacer con la matriz **a** y los vectores **u**, **v** y **w**. Esto podrá hacerse extensible a cualquier otra matriz o vectores de características similares. Entonces:

- Si escribo

  →a(2,3)

  obtendré el elemento de **a** correspondiente a la intersección de la fila 2 y la columna 3. En el caso de los vectores podré escribir u(3) en lugar de u(1,3) para obtener el tercer elemento de **u**, y v(2) en lugar de v(2,1) para obtener el segundo elemento de **v**.

- Con la orden

  →a(3,:)

  obtendré la tercera fila de **a** entera.

- Con

  →a(:,2)

  obtengo la segunda columna de la **a**.

- Con

  →a(:)

el resultado es un vector columna de tamaño 9x1, que corresponde a colocar la primera columna de **a**, seguida la segunda, y luego de la tercera.

- Con

  →u(:)

  convertimos el vector fila **u** en un vector columna.

- La orden

  →w(2) = []

  suprime la segunda componente de w y desplaza el resto. De manera análoga

  →a(: ,2) = []

  suprime la segunda columna.

Por medio de ***** se puede hacer el producto entre un número y una matriz. Los signos **+** y **-** permiten hacer sumas y restas entre matrices del mismo tamaño. Cuando el producto de matrices es posible (número de columnas de la primera matriz igual al número de filas de la segunda), este se indica también por medio de ***** .Para matrices (y vectores) del mismo tamaño se puede hacer la multiplicación elemento a elemento utilizando **.*** .De manera análoga se podrá hacer también la división elemento a elemento con **./** y la elevación a una potencia de todos los elementos con **.^**. Vemos algunos ejemplos sencillos:

→a*a
→a.*a
→a.^3

No hay que confundir este último caso con

→a^3

que seria equivalente a

→a*a*a

Las dos órdenes siguientes son equivalentes también:

→sqrt (a)
→a.^(1/2)

Y son diferentes a:

→sqrtm (a)
→a^(1/2)

Funciones como sqrt, sin, cos,… que apliquemos directamente a una matriz se aplicarán a cada uno de sus elementos. Hay que tener cuidado con esto.

La transposición de una matriz (o de un vector) se indica por medio de comilla. Obtener la transpuesta de **a** es tan simple como escribir:

→a'

Es importante advertir que el operador de matriz transpuesta (**'**), aplicado a matrices complejas, produce la matriz transpuesta conjugada. El operador punto y comilla (**.'**) será el que calcule simplemente la matriz transpuesta.

Por otro lado hay que comentar que en Scilab hay numerosas funciones para el manejo de matrices. Algunas de las más elementales pueden ser las siguientes:

- rank(b), que calcula el rango de una matriz cualquiera b.
- det(c), que calcula el determinante de una matriz cuadrada cualquiera c.
- inv(d) inversa de una matriz cuadrada e invertible cualquiera d.
- rref(b) matriz escalonada equivalente a b reducida por filas.

En este apartado ya sólo quedaría comentar como resolvemos en Scilab un sistema de ecuaciones lineales del tipo Ax = b, donde se conocen la matriz A y el vector columna b.

→A = [1 2 3 ; 2 -1 5 ; -4 3 2]
→b = [1 ; 2 ; 3]

Si A es una matriz cuadrada e invertible, como es el caso, el sistema tiene, teóricamente, una única solución y se puede resolver por una de las dos ordenes siguientes. La primera conlleva el cálculo de la inversa y la segunda usa un algoritmo eficiente de Scilab para hallar la solución:

→x1 = inv(A)*b
→x2 = A\b

Teóricamente el resultado debe ser el mismo.

Fuera del caso de solución única hay otros dos casos: el caso de sistema inconsistente (sin solución) y el caso de muchas soluciones (número infinito de soluciones). Si el sistema es inconsistente, entonces se desea encontrar una seudosolución, la solución por mínimos cuadrados. En este caso la orden

→x = A\b

encuentra una de estas seudosoluciones, y si las columnas de A son linealmente independientes, esta seudosolución es única.

1.5.5. Funciones

Scilab viene con un editor llamado Scipad. Se activa mediante la opción **Editor** de la barra de menú de Scilab. Aquí podremos escribir nuestras funciones, más de una en un archivo si queremos. Las guardaremos también donde queramos, pulsando **File→Save as** en la barra de menú de Scipad, siempre con extensión .sci.

El esquema general de una función es el siguiente:

```
function [res1, res2,...] = nombre función (par1, par2,...)

    ...
    ...

endfunction
```

El significado de res1 es resultado 1 o parámetro de salida 1. El significado de par2 es parámetro de entrada 2. Cuando la función tiene un único resultado podemos omitir los corchetes.

¿Cómo usamos nuestras funciones? Lo vemos con un par de ejemplos muy simples.

Ejemplo 1. Escribimos en Scipad lo siguiente y guardamos como ya se indicó, con el nombre prueba.sci por ejemplo.

```
//-------------------------------------------------------------------------------------------------
// (Con doble barra se escriben los comentarios) ---------------------------------------

        function y = prueba (x)

                y=2*x;

         endfunction

//-------------------------------------------------------------------------------------------------
```

Ahora habrá que cargar el archivo si queremos usarlo en Scilab. Lo mas sencillo es usar la barra de menú de Scilab: **File→Exec** y elegimos el archivo que queremos cargar. En caso de haber algún error en nuestra función se nos mostrará inmediatamente por pantalla.

Una vez cargado el archivo, las funciones se pueden utilizar como las otras funciones de Scilab. Por ejemplo, en este caso sería válida la siguiente orden:

```
→prueba (5)
```

Ejemplo 2. Escribimos en Scipad y guardamos y cargamos como en el ejemplo anterior.

```
//-------------------------------------------------------------------------------------------------

        function [y, z] = otraprueba (x1, x2)

                y = 2*x1;
                z = 3*x2;

        endfunction

//-------------------------------------------------------------------------------------------------
```

Una orden válida en este caso podría ser por ejemplo:

→[a b] = otraprueba (3,3)

A la hora de escribir funciones más complejas es necesario conocer las principales estructuras de control de flujo en Scilab. De forma resumida hablamos de:

- if

```
if   condición
     ...
else
     ...
end
```

- for

```
for var = lim1:incr:lim2
     ...
end
```

- select case

```
select  variable
     case valor1
          ...
     case valor2
          ...
```

else

...

end

- while

while *condición*

...

...

end

La orden **break** permite la salida forzada de un bucle for o de un bucle while.

La orden **return** permite salir de una función antes de llegar a la última orden.

Otra orden que sirve para interrumpir una función, en este caso interrumpiendo la evaluación, es **abort**.

También son muy importantes los operadores lógicos:

<	menor
<=	menor o igual
>	mayor
>=	mayor o igual
==	igual
~=	diferente
<>	diferente
&	y
|	o
~	no

Finalmente, mencionar la orden **printf**, que es similar a la orden del mismo nombre en lenguaje C, solo que aquí usa comillas simples en lugar de dobles. Ejemplo:

→a=2; b=3;
→printf (' a = %1d b = %1d \n', a, b)

Otras órdenes para escritura en Scilab son: **fprintf**, **print**, **write**,…

1.5.6. Gráficas

Usaremos la orden **plot2d** para conseguir gráficas en dos dimensiones. Lo vemos con un ejemplo:

→a = -2; b = 3;
→x = a: 0.05: b;
→plot2d (x, sin(x))

Scilab Graphic nos muestra la fig.5.6.1:

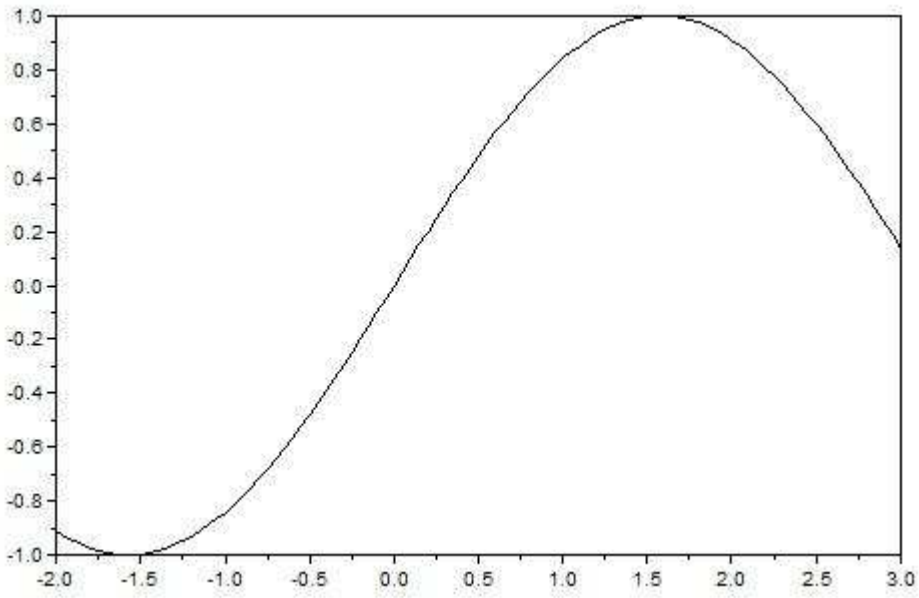

Fig.5.6.1.Resultado plot2d.

Si a continuación damos la orden

→plot2d (x, cos(x))

las gráficas del seno y el coseno aparecen superpuesta. Si no queremos que esto ocurra habrá que dar antes la orden

→xbasc ()

En la misma figura, si queremos, podemos pintar varias funciones con una sola orden. Pero en este caso, es imprescindible que los datos de plot2d aparezcan como vectores columna para que todo funcione bien. Si seguimos con el ejemplo visto, la orden ahora sería:

→plot2d (x', [sin (x') cos (x')])

Obtendremos en la misma figura la gráfica del seno y el coseno (las mismas que antes obtuvimos por separado). Scilab usa colores diferentes para distinguir entre gráficas. De igual modo podemos pintar tres, cuatro, cinco…gráficas sobre la misma figura.

Por otro lado, para conseguir gráficas tridimensionales usaremos **plot3d**. Vemos un ejemplo:

→x = [0:0.3:2*%pi];
→plot3d (x, x, sin(x')*cos(x))

Scilab Graphic nos muestra la fig.5.6.2:

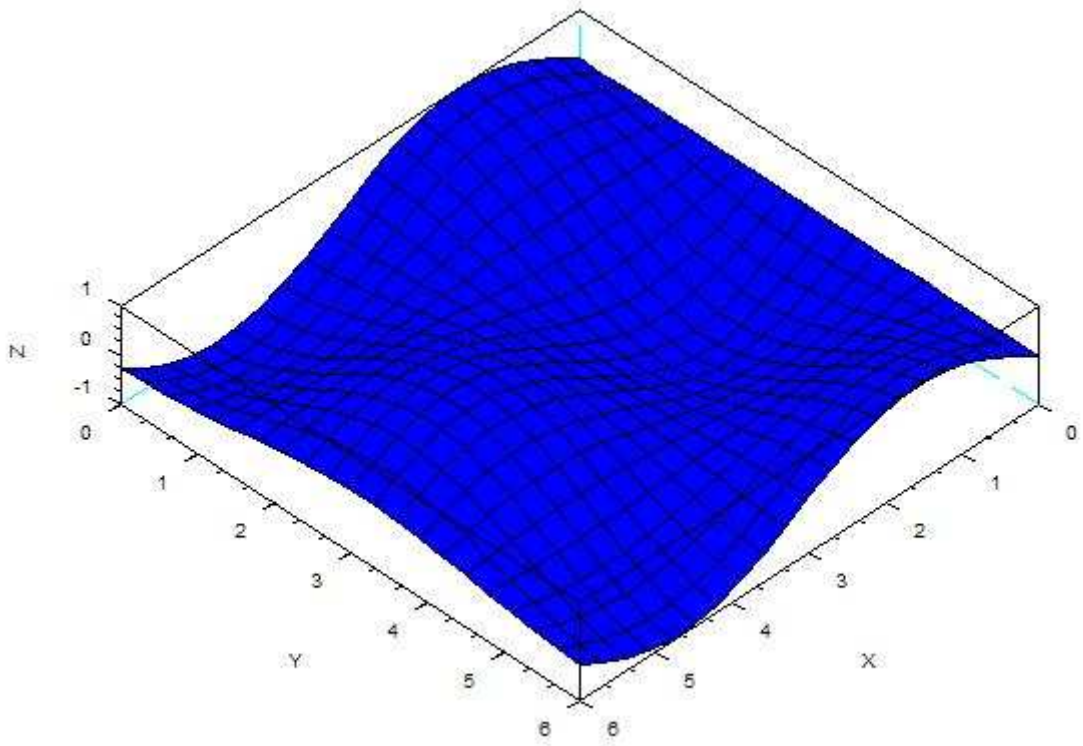

Fig.5.6.2.Resultado tridimensional.

1.5.7. Integración y derivación

Para calcular la integral de una función cualquiera entre dos valores de la abscisa x usaremos **integrate**, es lo más sencillo. Ejemplo:

→integrate ('sin(x)','x',0,%pi)

Hay otras funciones, que también pueden ser interesantes, como: **intg**, **inttrap**,…

Para derivar usamos **derivat**. Ejemplo:

→p=poly ([0],"x");

→derivat (1/p)

También puede ser interesante consultar **derivative**.

Hasta aquí este breve manual, que nos servirá de apoyo a la hora de realizar nuestro proyecto con Scilab.