0. PROLOGO

Este trabajo de la asignatura **TRATAMIENTO DIGITAL DE SEÑALES** del **Programa de Doctorado AUTOMÁTICA E INFORMÁTICA INDUSTRIAL** a parte de mostrarnos el tratamiento de señales con ***Scilab*** de **INRIA**, quiere ser una muestra del uso que se puede dar a **INTERNET** como herramienta de trabajo para el científico e investigador. Todo el software utilizado en este trabajo ha sido obtenido de distribución gratuita en **INTERNET**, tanto el software científico ***Scilab*** , el capturador de imágenes de pantalla **CaptureEze97**, el navegador **Netscape**, el descompresor de ficheros **Winzip**, el antivirus **Anyware**, como el lector **Gsview** para visualizar los manuales de *Scilab* que se encuentran en formato PS.

Este manual trata de mostrar una primera aproximación con el entorno *Scilab* y su Toolbox de Tratamiento de Señales . El lector obtendrá una visión general de *Scilab* y comparará este entorno con *Matlab* y se dará cuenta que cuenta con una potentísima herramienta al alcance de su mano.

1. INTRODUCCIÓN

1.a. ¿ Qué es *Scilab* ?

Scilab se ha desarrollado en INRIA para aplicaciones de Sistemas de Control y Tratamiento de Señales

Scilab está hecho en tres partes distintas: un interprete, librerías de funciones (procedimientos *Scilab*) y librerías con rutinas en *Fortran* y *C*.

Una ventaja de *Scilab* es que utiliza la sintaxis de *Matlab* para el manejo de MATRICES NUMÉRICAS, además de poder utilizar objetos más complejos. Por ejemplo, la gente que se dedica al control puede querer manejar matrices con racionales, polinomios o funciones de transferencia. Esto es muy fácil de realizar con *Scilab* mediante una representación simbólica de objetos matemáticos complicados, tales como funciones de

Resumiendo podemos decir que *Scilab* es un paquete de software científico para el cálculo numérico con un entorno muy amigable para el usuario. *Scilab* presenta las siguientes características :

- multivariable, etc...).
- Interprete sofisticado y lenguaje de programación con la sintaxis tipo *MATLAB*.
- Cientos de funciones matemáticas desarrolladas (el usuario puede ampliarlas).
- Fantásticos gráficos (2D, 3D y con animación).
- Muchas librerías desarrolladas (las llamadas toolboxes en *MATLAB*):
  - Algebra lineal
  - Control (control clásico, LQG, H_∞)
  - Paquete para optimización LMI (Matrices con inecuaciones lineales).
  - Optimización (diferenciable y no diferenciable).

- Scicos (entorno interactivo para la modelización y simulación con sistemas híbridos)

- Metanet (análisis de redes y optimización).

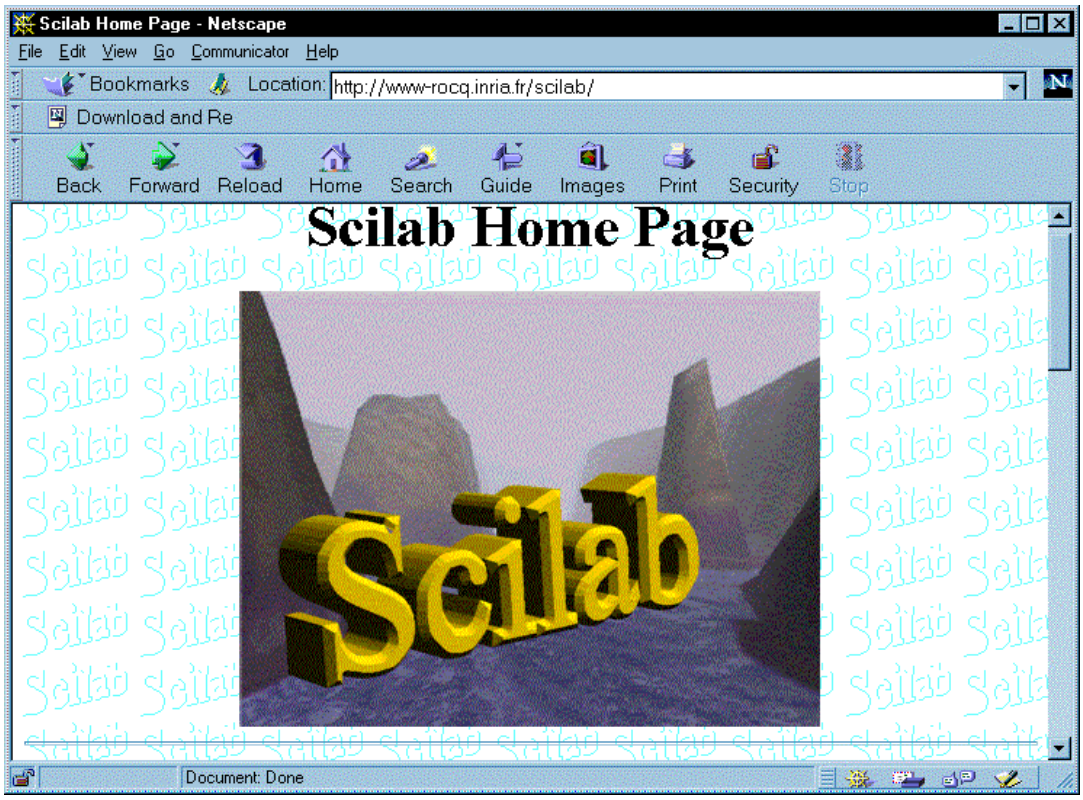
Existe un grupo de News **comp.soft-sys.math.scilab** donde se pueden intercambiar experiencias , ideas y programas desarrollados en *Scilab*. También para cualquier duda o sugerencia existe el *e-mail* Claude.Gomez@inria.fr cuya respuesta suele ser de un día para otro.

1.b. Requerimientos del Sistema para su instalación.

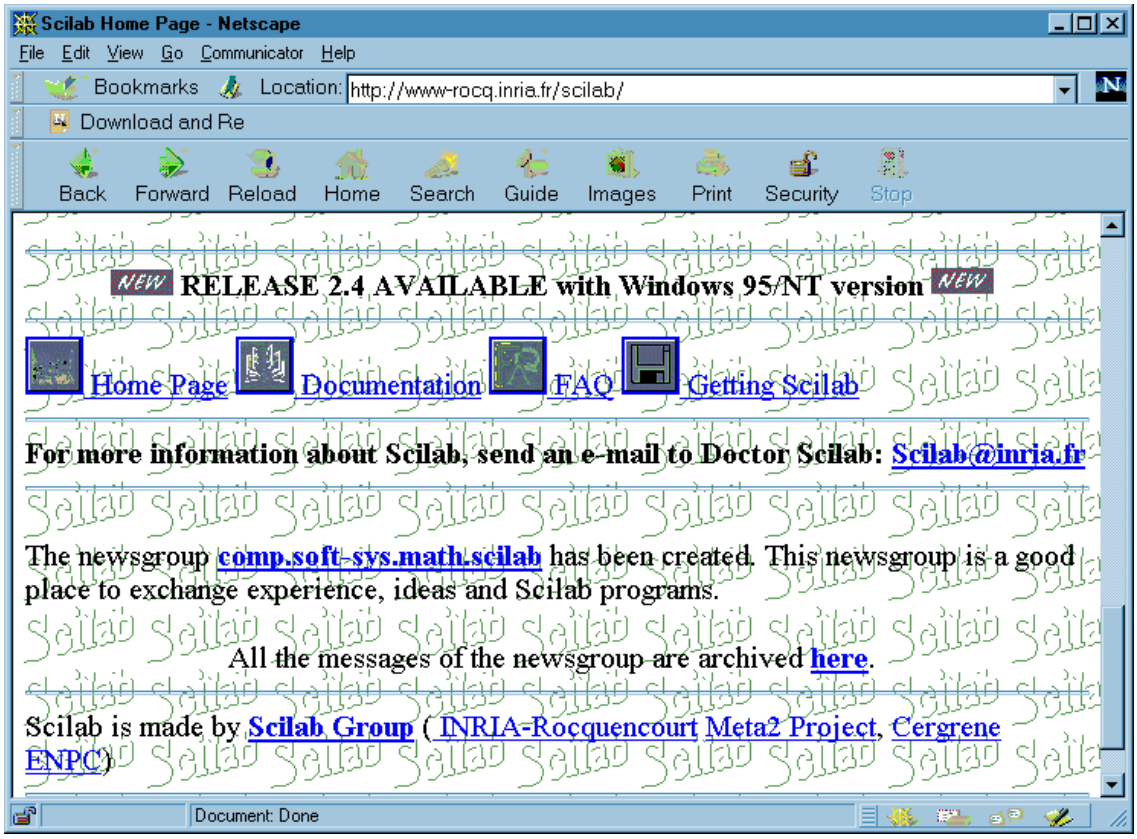
Los requerimientos de instalación de la versión fuente necesita como máximo 75 Mb de memoria en el disco duro para descomprimir el fichero e instalar la aplicación. Puede ser instalada perfectamente en un PC con S.O. Windows 95 ó 98, Linux, Unix-Xwindows; también sobre estaciones de trabajo Sun Sparc (Sun OS 4.1.3.), Sun Sparc (Sun Solaris 2.3), HP 9000 (HP-UX 9.01), IBM-RS6000 (AIX 3.2), etc.

1.c. Instalación de *Scilab* para Windows NT/95

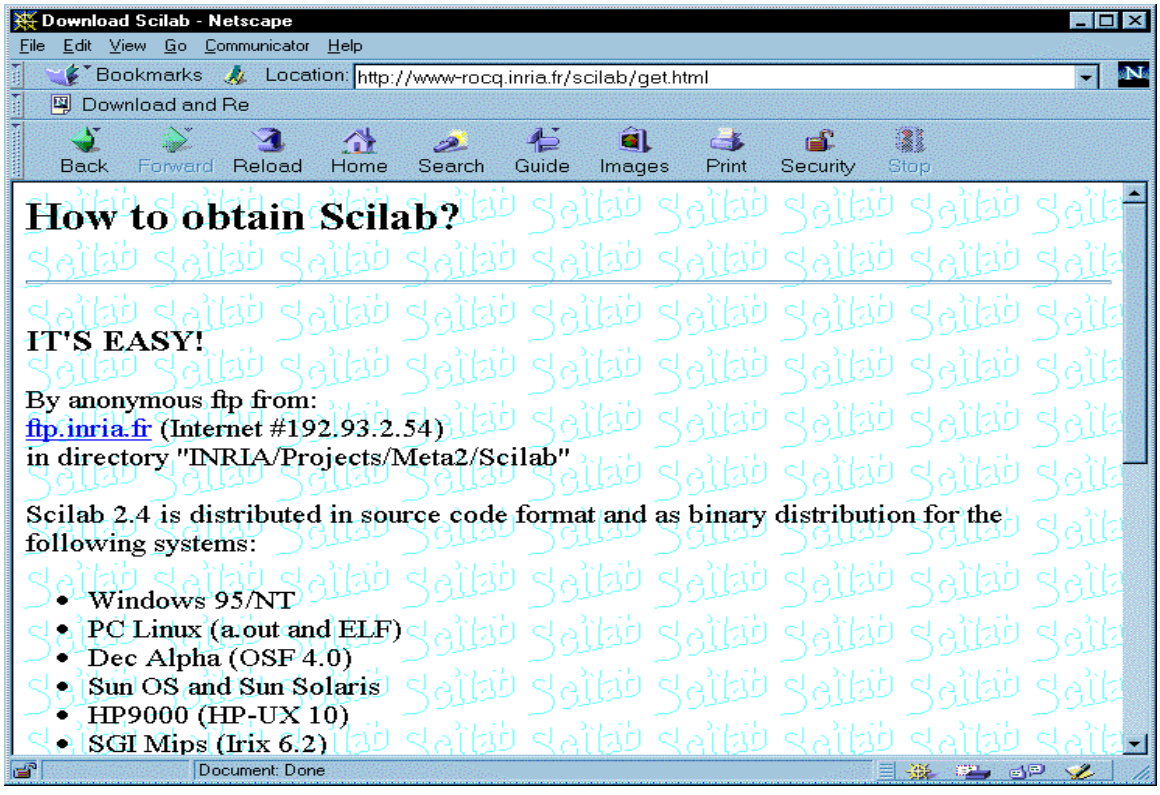
Scilab para Windows 95. Lo primero que hacemos es conectarnos a la página WEB de *Scilab* (**http://www-rocq.inria.fr**) apareciendo la siguiente página :



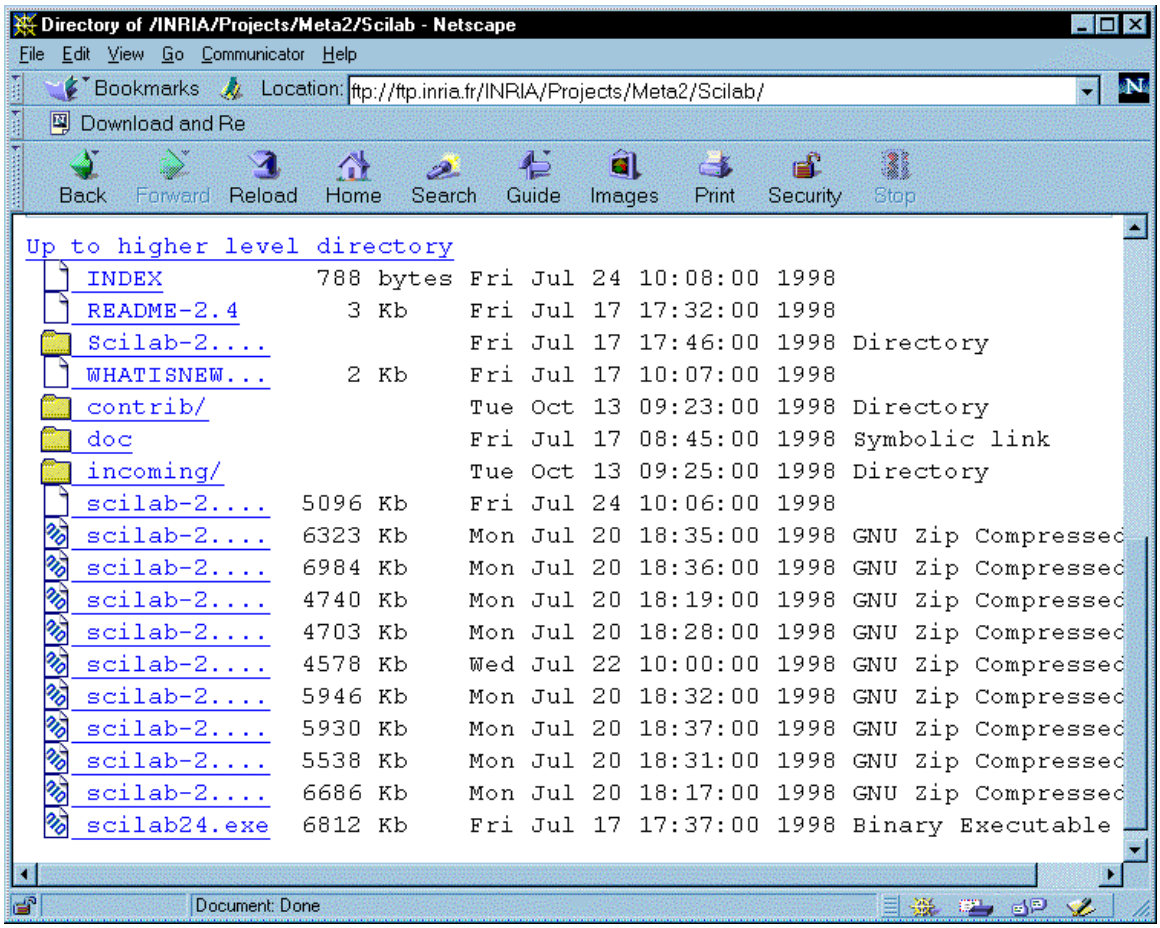
A continuación me muevo en la página y hago click en **Getting Scilab** (ver siguiente figura) :



Esta acción me llevará a la siguiente página WEB, que me va indicando paso a paso como obtener vía FTP *Scilab* para el sistema operativo que me interese. Es importante decir que INRIA va actualizando con frecuencia la versión, corrigiendo errores e implementando nuevas funciones.



Hago click en **ftp.inria.fr** y pasaré a la siguiente página que como podemos observar presenta estructura de árbol de directorios. Es aconsejable leer los archivos **INDEX** y **README**, que me dan información del contenido de los directorios y de los ficheros. Haciendo click en **UP TO HIGHER LEVEL DIRECTORY**, subo a un nivel superior en la estructura de directorios.

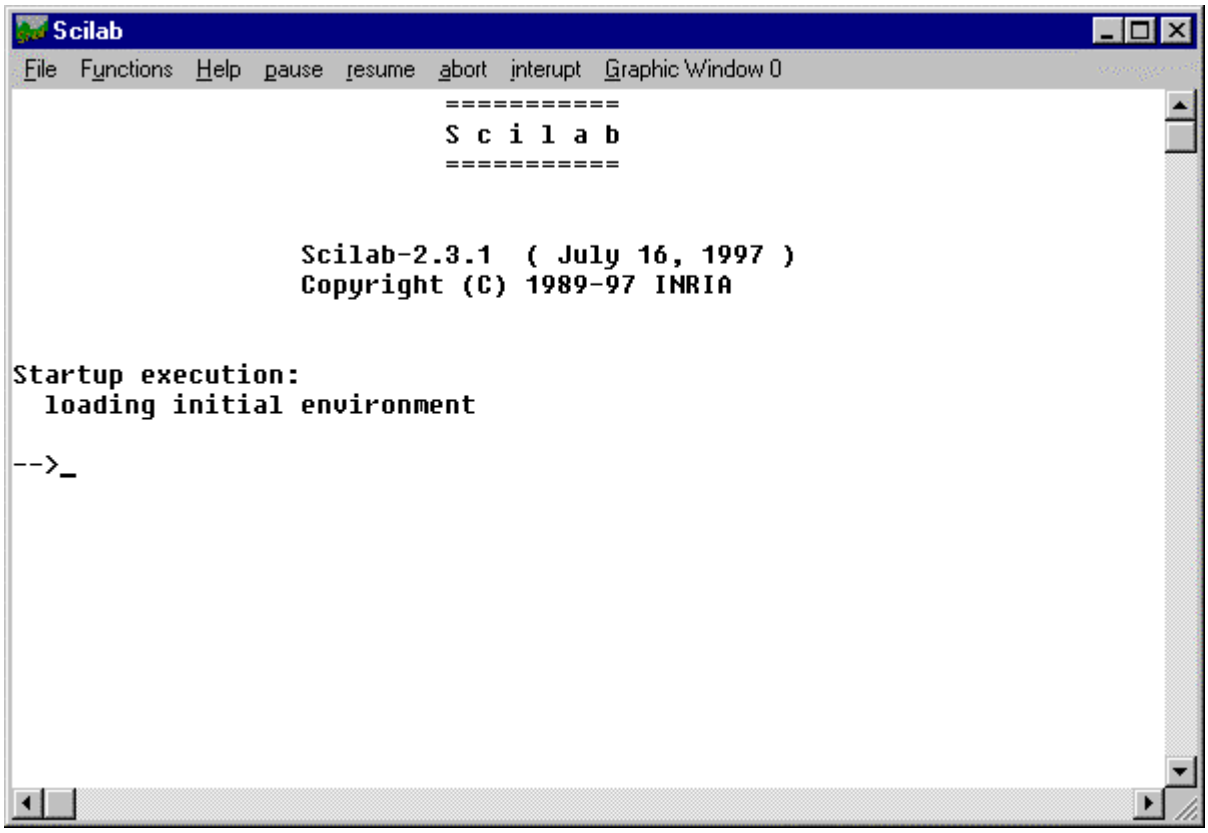


Una vez aquí debo centrarme en descargar en mi disco duro el fichero **scilab24.exe** Binary Executable.Una vez descargado el fichero **scilab24.exe** en mi disco duro, debe tener un tamaño de 6,65 MB, hago doble click sobre él y se instalará automáticamente. Durante la instalación me irán apareciendo ventanas de dialogo para que elija el directorio de instalación y si quiero que aparezca en **programas** dentro de **inicio.**

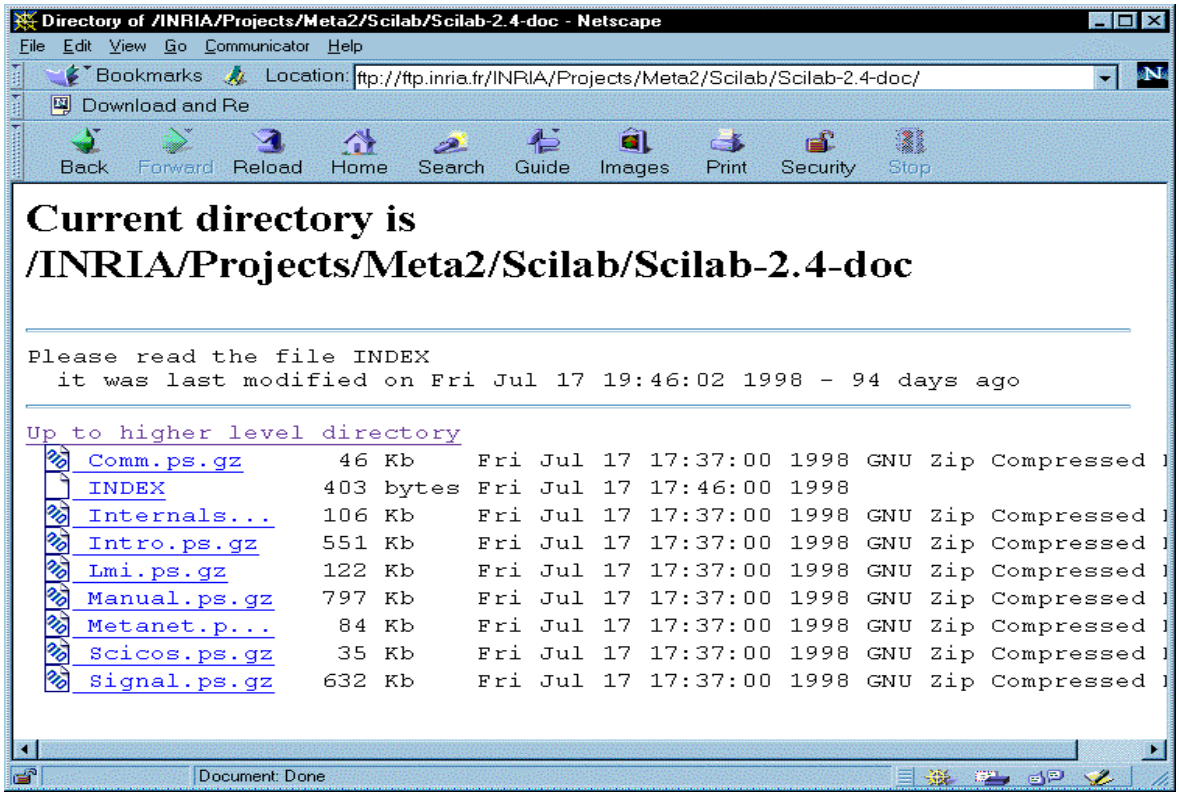
Si he elegido el directorio raíz para su instalación me creará el directorio **C:\scilab-24**, es importante reseñar que la versión 2.4 con respecto a la anterior la 2.3.1, incluye la utilidad de desinstalación de mi disco duro de *Scilab*. Para ejecutar *Scilab* debo hacer doble click sobre **C:\scilab-24\bin\runscilab.exe** o desde el botón de **inicio** en **programas**.

(***nota " para quién tenga la versión Scilab 2.3.1 "*** : la versión 2.3.1 necesitaba instalar la aplicación **Cygwin gnu-win32** para ello debía conseguir el fichero **usertools.exe** en la página Web **http://www.cygnus.com/misc/gnu-win32**. Es importante reseñar que **usertools.exe** me instala la aplicación **Cygwin gnu-win32** necesaria para que funcione en mi PC esta versión de *Scilab* para Windows NT y 95).

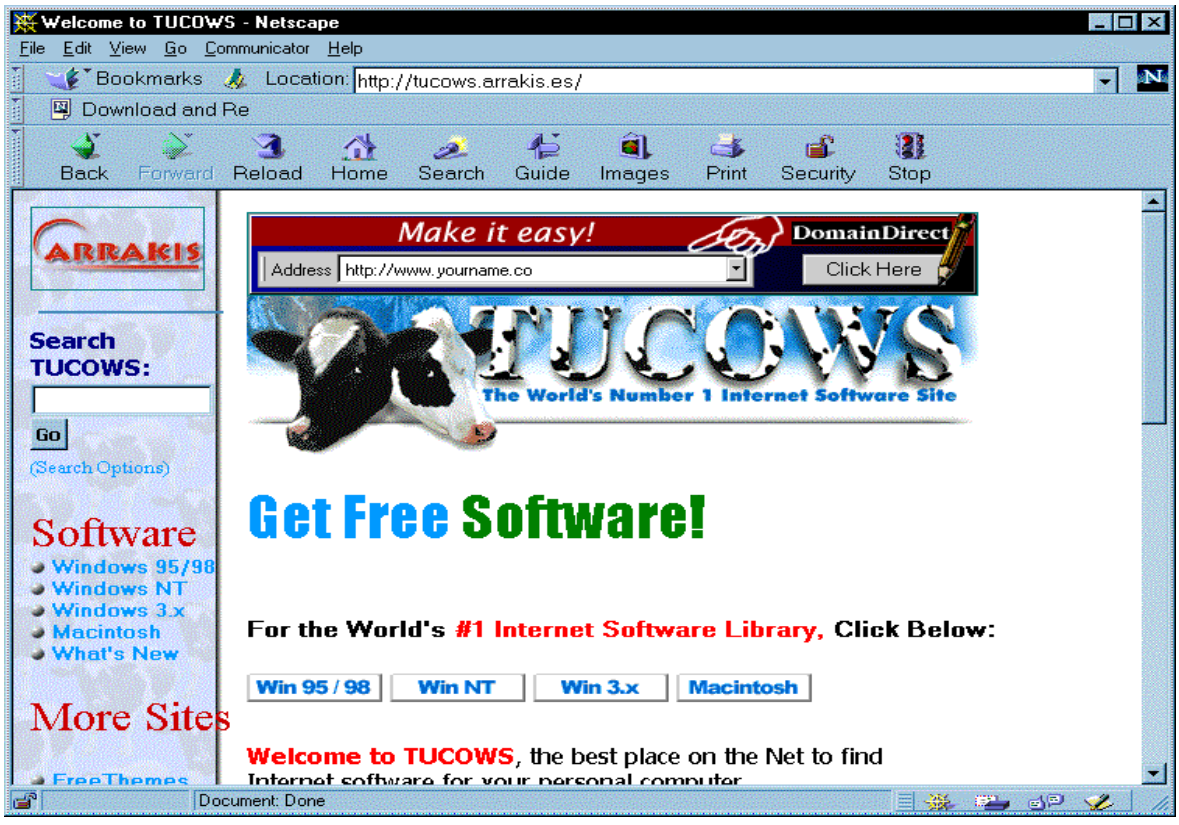
La **apariencia del entorno *Scilab*** una vez ejecutado es el siguiente :



Estupendo ya tenemos instalado *Scilab*, pero necesitamos los manuales donde se encuentra toda la documentación, por lo cual nos conectamos vía Internet a *Scilab* y otra vez a **ftp.inria.fr** . En el directorio **\doc** se encuentra toda la documentación en **ficheros postscript comprimidos** como podemos ver en la figura siguiente :

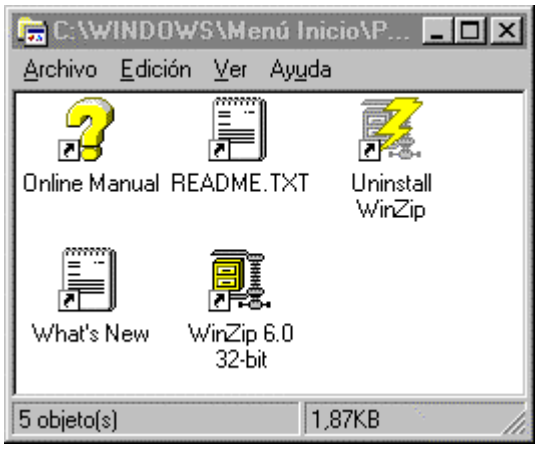


Descargo en mi disco haciendo doble click sobre ellos, pero una vez obtenidos se me plantea el problema de cómo descomprimirlos y leerlos. El problema está solucionado conectándome a la página **http://tucows.arrakis.es** :



Desde esta página puedo descargar software gratuito para Windows 95/98, Windows 3.xx , Windows NT y Macintosh, instalando en mi PC las siguientes aplicaciones :

- ***WINZIP*** que permite descomprimir ficheros :



- ***GSVIEW*** que permite visualizar ficheros *Postscript* :

Además no está de más instalar un **antivirus** ya que estoy descargando e instalando muchos ficheros de *Internet* que podrían estar contaminados.

- ***ANYWARE*** que es un antivirus :

- ***CAPTUREEZE97*** que es un capturador de imágenes de pantalla, para realizar este manual:

Scilab.

Scilab está dividido en los siguientes directorios :

El directorio principal scilab-xxx contiene los ficheros **scilab.star** (fichero de arranque), el **notice.tex** (fichero del copyright) y el **configure** (configuración). Además como podemos ver en la figura de arriba los subdirectorios más significativos son los siguientes :

- **bin** que es el directorio de los ficheros ejecutables. El código ejecutable de *Scilab* se encuentra allí. Este directorio también contiene los scripts que administran los ficheros que se generan con *Scilab* y la impresión de los mismos.

- **demos** es el directorio donde se encuentran todos los ejemplos demostraciones. El fichero alldems.dem permite añadir una nueva demostración para que pueda ser ejecutada haciendo click sobre el botón **Demos**. Es muy útil consultar **Demos** ya que los ejemplos incluidos pueden inspirar al nuevo usuario.

- **examples** contiene ejemplos muy útiles de cómo puedo conectar *Scilab* con programas externos, utilizando un enlace dinámico (dynamic link) o **intersci**.

- **man** es el directorio que contiene la *ayuda on-line* de *Scilab* y que está subdividido en submanuales.

- **imp** es el directorio de las rutinas de gestión de la impresión de ficheros *Postscript*.

- **maple** es el directorio que contiene el código fuente de las funciones *Maple*, las cuales permiten la inserción de objetos *Maple* en las funciones *Scilab*. Para que sea eficiente, la transferencia está hecha con *Fortran* dinámicamente enlazado a *Scilab*.

- **routines** es el directorio que contiene el código fuente de todas las rutinas numéricas.

- **Intersci** contiene el programa que permite la construcción de la interfaz necesaria para añadir nuevo *Fortan* o *C* a *Scilab*.

- **scripts** es el directorio que contiene todo el código fuente de los ficheros *script* de la *Shell*.

1.e. Botones de control de *Scilab*.

La ventana de *Scilab* presenta los siguientes botones de control :

- **pause** interrumpe la ejecución de *Scilab* y lo pone en modo de pausa.

- **resume** permite continuar con la ejecución de un programa después de una pausa, generada como un comando dentro de una función, por el botón pause o por control+C.

- **abort** aborta totalmente la ejecución total de un programa y retorna el prompt al nivel superior.

- **Help** invoca la ayuda on-line (presenta un menú)

- **Graphic Window** puedo seleccionar la activación de una ventana gráfica.

- **File y Functions** no son botones con una sola función, si no que presentan menús para ejecutar una función, definir una nueva, archivar un fichero, cargar un fichero en el entorno *Scilab*, etc.

2. PRINCIPALES DIFERENCIAS CON *MATLAB*.

Antes de empezar con nuestra primera sesión con *Scilab*, vamos a decirle a los usuarios de *Matlab* que ambos tienen muchas similitudes. Pero existen unas diferencias que se deben conocer antes de esta primera sesión, de tal forma que se puedan ir asimilando según se avanza en los ejemplos de este manual.

2.a. Funciones.

Las funciones en Scilab no son ficheros m (m-files) como en *Matlab*. Una o varias funciones pueden ser definidas en un único **fichero .sci** (ej. **mifichero.sci**), que son los ficheros de ejecución por lotes de Scilab. El nombre del fichero no tiene porqué ser necesariamente el de la función. El nombre de la función o funciones viene dado por :

```
function [y]=fct1(x)
...
function [y]=fct2(x)
...
```

Scilab. Se debe ejecutar el comando **getf("mifichero.sci","c")** antes de usarlas. Las funciones también pueden ser definidas *on-line* o dentro de otra función mediante el comando **deff**.

En *Scilab* para ejecutar un fichero por lotes (script file) debo usar **exec("nombrefichero")** , mientras que en *Matlab* se ejecuta directamente con el nombre de fichero (ej. mifichero.m se ejecuta con mifichero si estoy en el directorio donde se encuentra grabado).

2.b. Lineas de comentario.

En *Scilab* empiezan con //, mientras que en *Matlab* empiezan con **%**.

2.c. Variables.

En *Scilab* las variables predefinidas llevan el prefijo **%** (ej. %i, %aux, ...).

2.d. Cadenas (strings).

En *Scilab* son consideradas como matrices cadena de 1 por 1. Cada entrada de una matriz cadena tiene su propia longitud.

2.e. Variables Booleanas.

Las variables booleanas en *Scilab* son **%T** y **%F**, mientras que en *Matlab* son **0** y **1**.

2.f. Polinomios.

En *Scilab* los polinomios y matrices polinómicas son definidas por la función **poly** , mientras que en *Matlab* son considerados como vectores de coeficientes.

2.g. Matrices vacías.

[ ]+1 retorna 1 en *Scilab*, mientras que en *Matlab* retorna **[ ]**.

2.h. Gráficos.

Excepto para las funciones **plot** y **mesh**, *Scilab* y *Matlab* no son compatibles.

2.i. Scicos (*Scilab*) y Simulink (*Matlab*).

Ninguno de los dos simuladores son compatibles.

2.j. Ficheros Binarios.

Los ficheros salvados en *Matlab* como binarios no pueden ser cargados en *Scilab*. Pero existe una utilidad para convertirlos, esta se encuentra en la página Web en **ftp.inria.fr** dentro del directorio **contrib/matlab_interface**.

2.k. Variables Locales y Globales.

En *Scilab* si una variable no está definida en una función (ej. no está entre los parámetros de entrada), entonces automáticamente es considerada como una variable global. El valor actual de la variable es usado dentro de la función. Las funciones pueden ser llamadas con menos parámetros de entrada o salida. Aquí tenemos un ejemplo :

```
function [y1,y2]=f(x1,x2)
y1=x1+x2
y2=x1-x2

-->[y1,y2]=f(1,1)
y2  =
  0.
y1  =
    2.

-->f(1,1)
ans  =
  2.

-->f(1)
y1=x1+x2; y2=x1-x2
 !--error     4
undefined variable : x2
at line       2 of function f

-->x2=1;

-->[y1,y2]=f(1)
 y2  =
 0.
y1  =
  2.

-->f(1)
 ans  =

2.
```

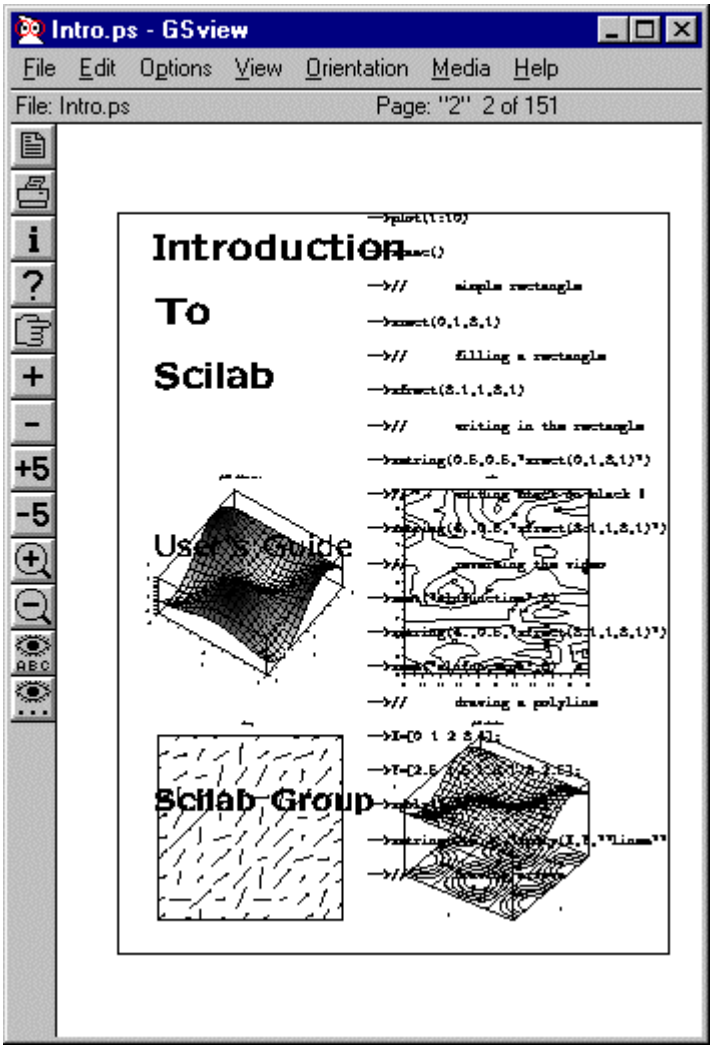
2.I. Lista de comandos equivalentes.

La mayoría de las funciones implementadas en *Scilab* y *Matlab* son idénticas. Algunas de ellas son ligeramente diferentes en su sintaxis. A continuación se muestra un resumen comandos con diferente sintaxis y su equivalencia:

MATLAB	Scilab
all	and
any	or
balance	balanc
clc	CTRL-L
clock	unix('date')
computer	unix_g('machine')
cputime	timer
delete	unix('rm file')
dir	unix_g('ls')
echo	mode
eig	spec or bdiag
eval	evstr
exist	exists + type
fclose	file('close')
feof	
ferror	
feval	evstr and strcat
filter	rtitr
finite	(x < %inf)
fopen	file('open')
fread	read
fseek	file
ftell	
fwrite	writeb
global	
home	
isglobal	
isinf	== %inf
isnan	==
isstr	type + ==
isstr	type + ==
keyboard	pause + resume
lasterr	
lookfor	apropos
more	lines
pack	stacksize
pause	halt
qz	gspec+gschur
randn	rand
rem	modulo
setstr	code2str
strcmp	==
uicontrol	
uimenu	getvalue
unix	unix_g
version	
which	whereis
nargin	[nargout,nargin]=argn(0)
nargout	

3. PRIMERA SESIÓN CON *SCILAB*.

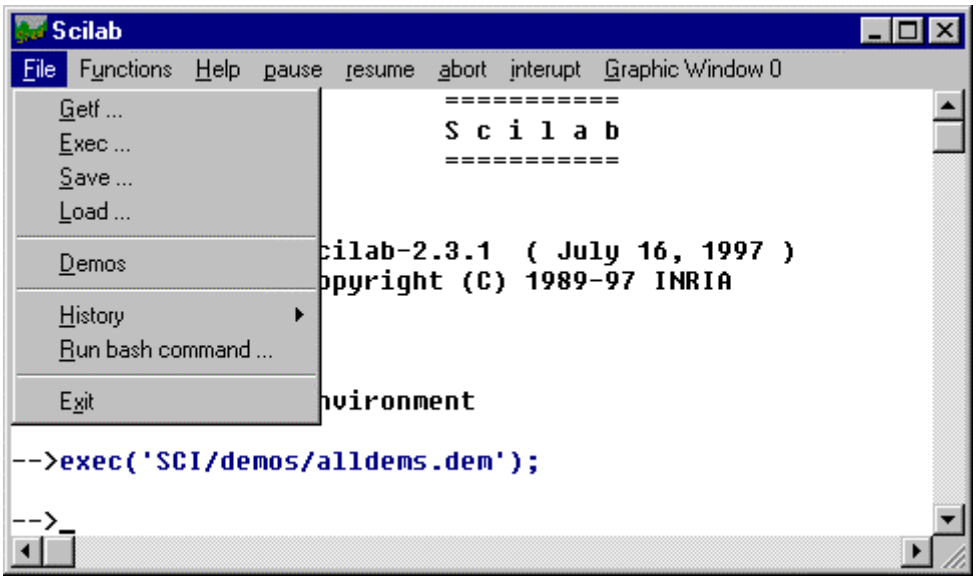
En esta sesión con *Scilab* no se profundizará en la programación con *Scilab* , ya que este manual trata de dar una idea general del entorno, para pasar al Tratamiento de Señales con *Scilab* que es lo que nos ocupa en esta asignatura. A los lectores que quieran profundizar más en la programación les remito al manual **INTRODUCTION TO SCILAB User's Guide** , que se encuentra en el directorio **\doc** en ftp.inria.fr dentro del fichero **Intro.ps.gz**. Este fichero es un **Zip** que al descomprimirlo con **WinZip** obtendré el **Intro.ps** que ya podré visualizar con **Gsview** :



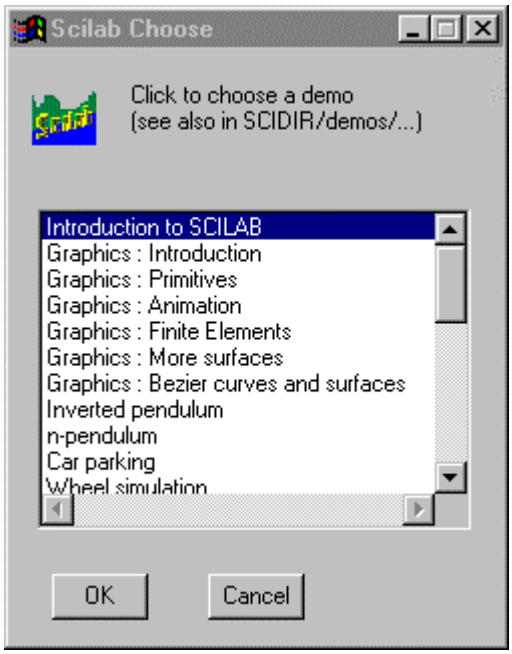
3.a. Uso de la Ayuda y Demos.

La apariencia de *Scilab* nada más ejecutado aparece en la figura de la página 5, antes de explicar ningún comando, veremos como se utiliza la **ayuda** y las **demos**.

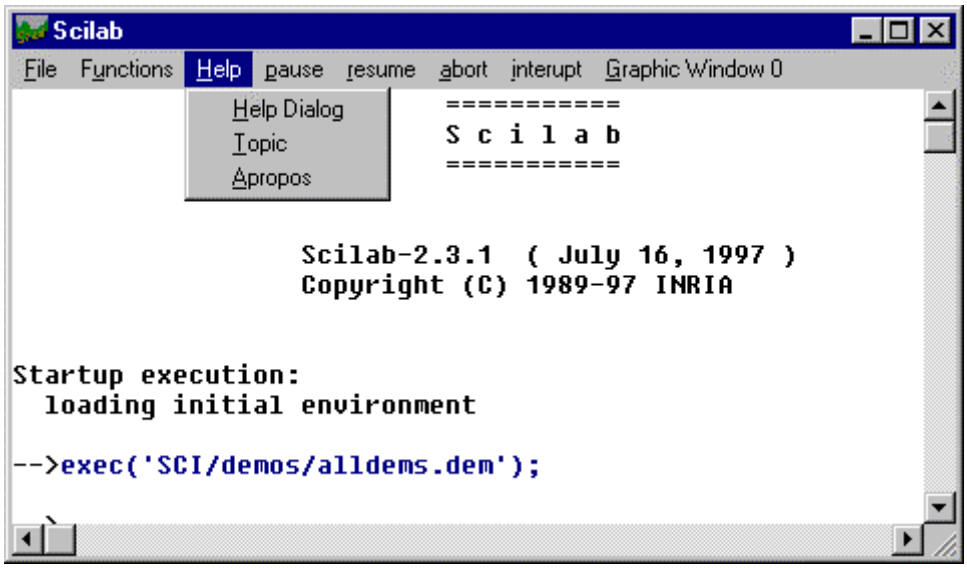
A continuación despliego el menú **File** de la ventana *Scilab* viendo que aparece la opción Demos:



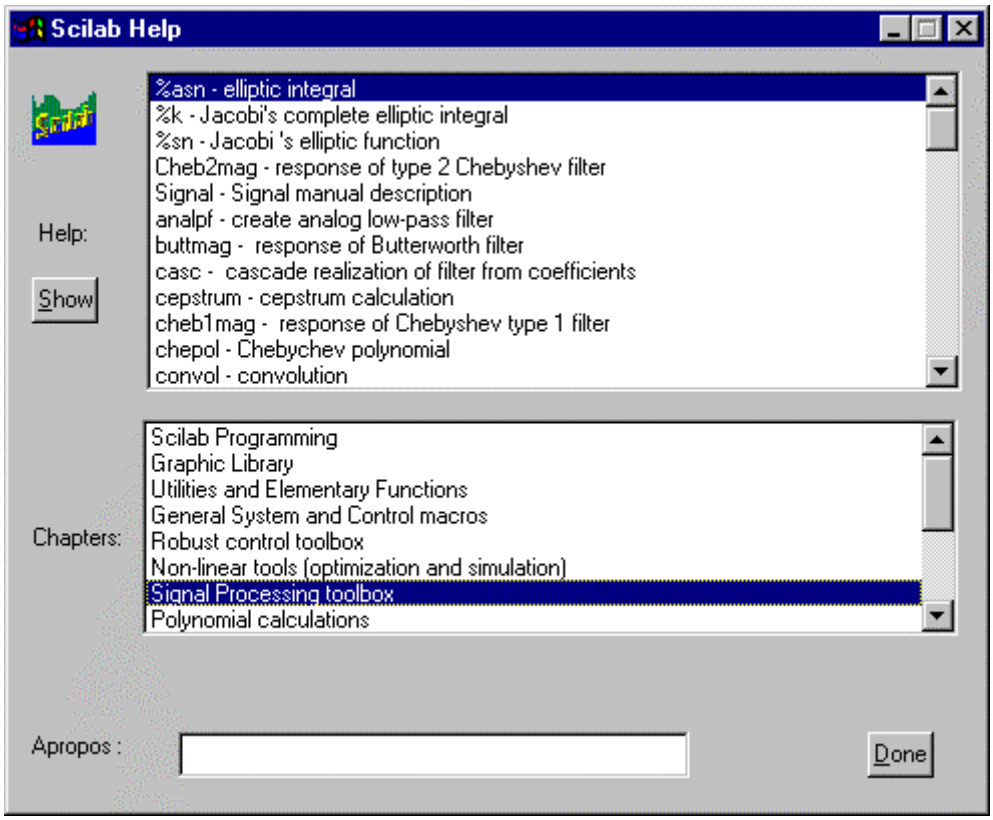
A continuación hago click sobre **Demos** apareciendo el siguiente menú donde me moveré sobre el que me interese y pulsaré el botón **OK** :



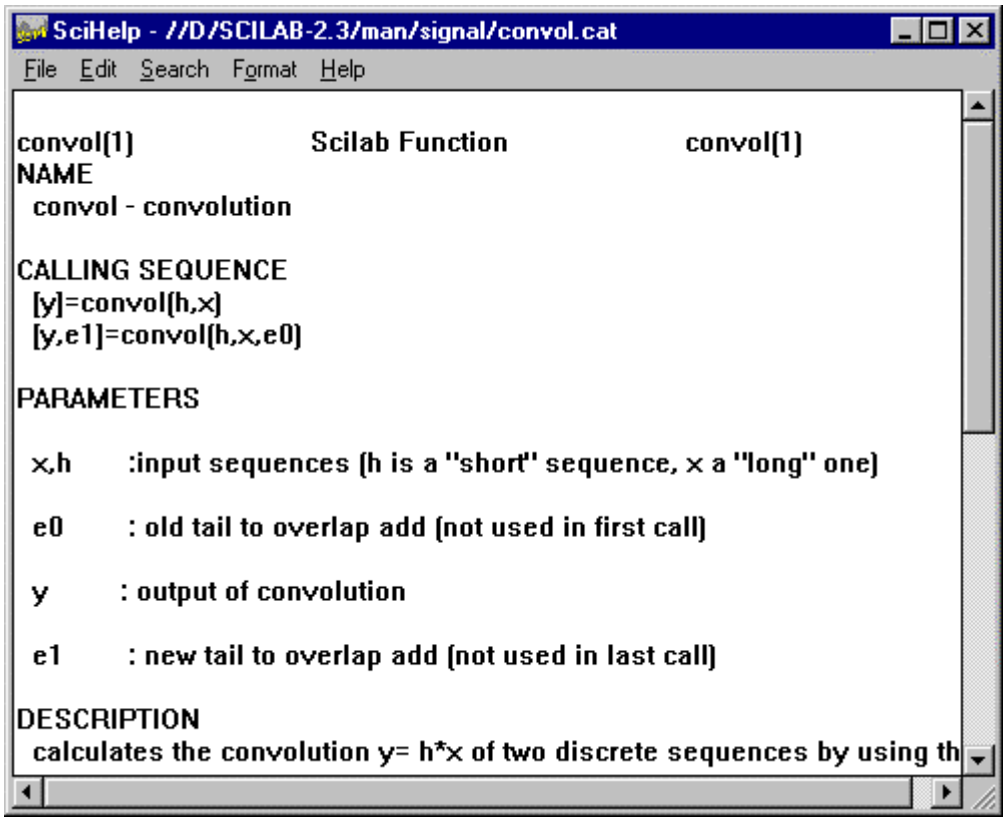
Para consultar la *ayuda on-line* debo despliego el menú **Help** de la ventana de *Scilab* :



Si selecciono **Topic** o **Apropos** podré hacer búsquedas en la ayuda más restrictivas por un término o grupo de términos, mientras que si hago click sobre **Help Dialog**, aparece una ventana de ayuda general dividida en temas :



Como ejemplo se ha seleccionado en la figura anterior el tema **Signal Processing Toolbox**, apareciendo en la parte superior todas los comandos y funciones relacionados con el tema. Hemos seleccionado la función **convol** (convolución), como ejemplo, haciendo click sobre el botón **show**. Apareciendo la siguiente ventana de ayuda en la que se explica la sintaxis, una descripción de lo que hace la función, así como un ejemplo de usarla :



3.b. Comandos básicos.

Vamos a presentar en este apartado algunos comandos básicos de *Scilab*, que al usuario de *Matlab* le van a ser muy familiares. Al lector que no conozca *Matlab*, le llamará la atención la simplicidad del entorno *Scilab*, punto totalmente extrapolable a *Matlab* y estos conocimientos le facilitarán la labor si piensa utilizar este entorno.

Una vez en la ventana *Scilab* debo fijarme en que aparece un *prompt* con la siguiente apariencia **- ->**, y será ahí donde debo teclear los comandos *Scilab*, pulsando después **intro** para hacer efectiva su entrada en el entorno.

Como veremos en los ejemplos siguientes al introducir una nueva variable o constante , si pongo **;** al final , no se nos mostrará el valor de las mismas.

3.b.1. Escalares.

- ***Constante real :***

```
a=1
 a  =

    1.
```

- ***Comparación Booleana :***

```
1==1
 ans  =

  T
```

- ***Cadena de caracteres :***

```
'string'
 ans  =

 string
```

- ***Polinomio con la variable z con raiz en cero :***

```
z=poly(0,'z')
 z  =

    z
```

- ***Polinomio :***

```
p=1+3*z+4.5*z^2
 p  =

                2
  1 + 3z + 4.5z
```

- ***Racional :*** Voy a aprovechar los polinomios definidos anteriormente ,para definir un racional entre polinomios.

```
r=z/p
 r  =

        z
  -------------
                2
  1 + 3z + 4.5z
```

3.b.2. Matrices.

En estos ejemplos se utilizarán constantes definidas en el apartado anterior 3.b.1.

- ***Matrices de constantes :***

```
A=[a+1 2 3
     0 0 atan(1)
     5 9 -1]
 A  =

!   2.    2.    3.         !
!   0.    0.    0.7853982 !
!   5.    9.  - 1.         !
```

- ***Matrices Booleanas :*** Este ejemplo es de 1× 2

```
 b=[%t,%f]
  b   =

 ! T F !
```

- ***Matrices de Cadenas de Caracteres :***

```
Mc=['this','is';
    'a' ,'matrix']
 Mc   =

!this  is      !
!              !
!a     matrix  !
```

- ***Matrices polinómicas :***

```
Mp=[p,1-z;
    1,z*p]

 Mp   =

!                2                    !
!   1 + 3z + 4.5z     1 - z           !
!                                     !
!                           2      3  !
!   1                  z + 3z + 4.5z  !
```

- ***Matrices con racionales :***

```
F=Mp/poly([1+%i 1-%i 1],'z')
 F  =

!                  2                          !
!    1 + 3z + 4.5z          - 1               !
!   ---------------      ---------             !
!                2   3               2        !
! - 2 + 4z - 3z + z      2 - 2z + z            !
!                                              !
!                                    2      3  !
!          1                 z + 3z + 4.5z     !
!   ---------------      ---------------       !
!                2   3                 2   3   !
! - 2 + 4z - 3z + z    - 2 + 4z - 3z + z       !
```

- ***Matrices Sparce :***

```
Sp=sparse([1,2;4,5;3,10],[1,2,3])
 Sp  =

(    4,   10) sparse matrix

(    1,    2)        1.
(    3,   10)        3.
(    4,    5)        2.
```

- ***Matrices Sparce Booleanas :***

```
Sp(1,10)==Sp(1,1)
 ans  =

(    1,    1) sparse matrix

(    1,    1)    T
```

3.b.3. Listado de variables en Scilab.

- ***Variables del entorno inicialmente cargado :*** Se visualizan con **who** y son variables que siempre estarán presentes.

```
-->who
your variables are...

 home       pwd        startup    ierr       scicos_pal            TMPDIR
 soundlib   blockslib              scicoslib             xdesslib   utillib
 tdcslib    siglib     s2flib     roblib     percentlib            optlib
 metalib    elemlib    commlib    polylib    autolib    armalib    alglib
 SCI        %F         %T         %z         %s         %nan       %inf
 old        newstacksize          $          %t         %f         %eps
 %io        %i         %e         %pi
 using    3863 elements  out of 1000000.
          and      41 variables out of    1023
```

- ***Variables definidas por el usuario :*** Una vez haya introducido variables el usuario, puedo controlar que variables existen en mi entorno de trabajo. Se utiliza el mismo comando anterior :

```
-->a=1;

-->B=3;

-->who
your variables are...

 B          a          home       pwd        startup    ierr
 scicos_pal            TMPDIR     soundlib   blockslib
 scicoslib             xdesslib   utillib    tdcslib    siglib     s2flib
 roblib     percentlib            optlib     metalib    elemlib    commlib
 polylib    autolib    armalib    alglib     SCI        %F         %T
 %z         %s         %nan       %inf       old        newstacksize
 $          %t         %f         %eps       %io        %i         %e
 %pi
 using     3869 elements  out of 1000000.
           and      43 variables out of    1023
```

Como vemos he introducido dos constantes nuevas **a** y **B;** pudiéndose ver que aparecen.

3.b.4. Algunas Operaciones básicas.

- ***Multiplicación de matrices de constantes :***

```
v=1:5;W=v'*v

 W  =

!   1.    2.    3.    4.    5.  !
!   2.    4.    6.    8.    10. !
!   3.    6.    9.    12.   15. !
!   4.    8.    12.   16.   20. !
!   5.    10.   15.   20.   25. !
```

Como podemos ver **v** es una matriz de 1× 5**, v'** es la matriz traspuesta 5× 1de **v.** Luego su producto **W** es una matriz de 5 × 5.

- ***Extracción de filas de una matriz :***

```
W(1,:)

ans  =

!   1.    2.    3.    4.    5. !
```

- ***Extracción de columnas de matrices :*** En este ejemplo hemos extraído la última columna.

```
W(:,$)
 ans  =

!   5.  !
!   10. !
!   15. !
!   20. !
!   25. !
```

- ***Matriz inversa :***

```
inv(A)

 ans  =

!   0.5     0.          0.    !
!   0.      0.3333333   0.    !
!   0.      0.          0.25  !
```

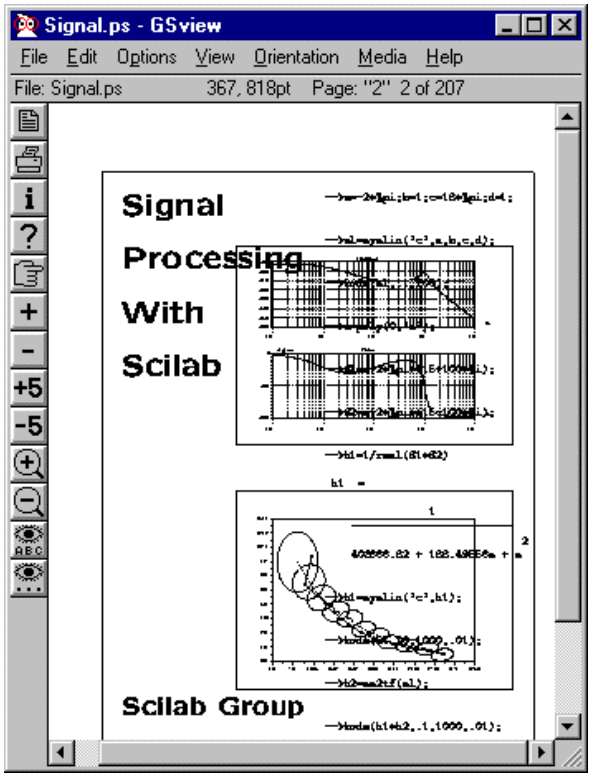
A puede ser una matriz de constantes, racional, polinómica o booleana.

El usuario que quiera seguir profundizando en los comandos más sencillos de *Scilab* les remitimos a la ayuda on-line o al manual **Introduction to Scilab**.

4. TRATAMIENTO DE SEÑALES CON *SCILAB*.

4.a. Documentación sobre Tratamiento de señales con *Scilab*.

Toda la documentación relacionada con el tratamiento de señales se encuentra en el manual **SIGNAL PROCESSING WITH SCILAB**, en el directorio **\doc** en ftp.inria.fr dentro del fichero **Signal.ps.gz**. Este fichero es un **Zip** que al descomprimirlo con **WinZip** obtendré el **Signal.ps** que podré visualizar con **Gsview**:



Puedo encontrar también más información en las **Demos on-line**.

4.b. Comandos *Scilab* para el tratamiento de señales.

Scilab al igual que *Matlab* tiene una gran colección de funciones para el tratamiento de señales, como por ejemplo, la convolución, inversión de matrices, transformada rápida de Fourier (FFT), etc.

Con las herramientas que me encuentro en *Scilab* para el tratamiento de señales podré realizar entre otras cosas :

- Análisis e implementación de filtros digitales, incluyendo respuesta en frecuencia, retardo de grupo, retardo de fase. La implementación de filtros puede ser directa como utilizando técnicas en el dominio de la frecuencia basadas en la FFT.
- Diseño de filtros IIR, incluyendo Butterworth, Chebyschev I, Chebyschev II, Elíptico, optimizado en su diseño con el criterio mínimo L_p.
- Diseño de filtros FIR y su optimizado .
- Procesamiento de la FFT, incluyendo la transformación base-2 y su inversa, y la transformada para potencias que no sean de dos.
- Estimación espectral : espectro de potencia.
- Filtrado óptimo y suavizado : filtro de Kalman, filtro de la raíz cuadrada de Kalman, transformación de Householder, filtro de Wiener,etc.
- Representaciones de señales en tiempo-frecuencia : representación de Wigner-Ville, etc.

4.b.1. Comandos básicos.

El enfoque de los apartados siguientes aunque es aplicable al tratamiento de señales en general, quiere hacer más hincapié en las herramientas existentes para el control automático.

- ***Polinomios y raíces :*** En este ejemplo se define el polinomio con las raíces.

```
  q1=poly([1 2],'x')
q1  =

               2
   2 - 3x + x

  roots(q1)
ans  =

!   1. !
!   2. !
```

En el siguiente ejemplo se define el polinomio por sus coeficientes :

```
  q2=poly([1 2],'x','c')
q2  =

   1 + 2x

  roots(q2)
ans  =

 - 0.5
```

Operaciones con polinomios : *Scilab* puede manipular polinomios de la misma manera que lo hace con los escalares, vectores y matrices . En los siguientes ejemplos se muestran ejemplos de resta, suma, división y

```
  x=poly(0,'x')
x  =

   x

  q1=3*x+1
q1  =

   1 + 3x

  q2=x**2-2*x+4
q2  =

              2
   4 - 2x + x

  q2+q1
ans  =

             2
   5 + x + x
```

```
  q2-q1
 ans  =

              2
   3 - 5x + x

  q2*q1
 ans  =

                 2    3
   4 + 10x - 5x + 3x


   q2/q1
  ans  =

   4 - 2x + x
   ----------
     1 + 3x

  q2./q1
 ans  =

              2
   4 - 2x + x
   ----------
     1 + 3x
```

Racionales polinómicos : Un racional está representado por una **lista** de cuatro elementos, siendo p(1) el valor r que indica que es un racional, p(2) el valor del numerador, p(3) el valor del denominador y el p(4) el último elemento de la lista:

```
  p=poly([1 2],'x')./poly([3 4 5],'x')
 p  =

                 2
      2 - 3x + x
   ---------------
                 2   3
  - 60 + 47x - 12x + x

  p(1)
 ans  =

!r  num  den  dt  !
```

```
  p(2)
ans  =

              2
 2 - 3x + x


  p(3)
ans  =

                    2   3
 - 60 + 47x - 12x + x

  p(4)
ans  =

    []
```

- ***Representación de un Sistema Lineal por su Función de Transferencia :***
  se utilizan las expresiones racionales polinómicas para describir funciones de transferencia, por ejemplo de filtros. Estas funciones de transferencia pueden representar señales en el dominio temporal continuo como discreto. Pero como ya sabemos los sistemas o señales continuos pueden ser transformados a discretos mediante el muestreo.

  Para describir la Función de Transferencia utilizo el comando **syslin** :

```
-->sl=syslin(domain,num,den)
```

- ***Representación el Espacio de Estados de un Sistema Lineal :***

  Existen dos representaciones clásicas en el Espacio de Estados, la continua y la discreta :

  Continua:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \\ x(0) &= x_0 \end{aligned}$$

  Discreta :

$$\begin{aligned} x(n+1) &= Ax(n) + Bu(n) \\ y(n) &= Cx(n) + Du(n) \\ x(0) &= x_0 \end{aligned}$$

Siendo A,B,C y D matrices y x_0 un vector.

Para representar en el Espacio de Estados, *Scilab* vuelve a utilizar la función **syslin**, pero de la siguiente forma:

```
-->sl=syslin(domain,a,b,c [,d[,x0]])
```

Siendo el valor de retorno de **sl** es una **lista** con los siguientes elementos :

```
s=list('lss',a,b,c,d,x0,domain)
```

- ***Cambio de representación : paso de la Función de Transferencia (FT) al Espacio de Estados (EE) y viceversa.***

En los problemas normales del tratamiento de señales me puede interesar una u otra representación, que son totalmente equivalentes.

Paso de FT a EE :

```
-->sl=tf2ss(h)
```

Paso de EE a FT :

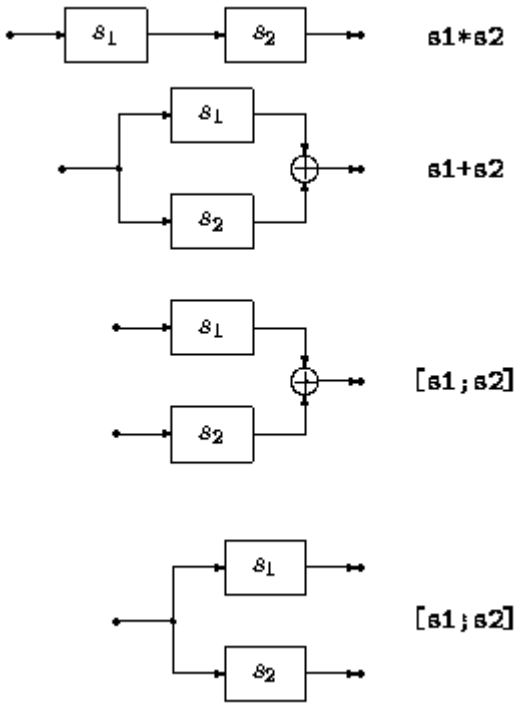
```
-->h=ss2tf(sl)
```

Vamos a ver ***un ejemplo*** en el que se va a utilizar $\mathbf{h_1}$, **filtro de respuesta impulsional infinita**, que todavía no hemos visto en este manual y del que ahora sólo me interesa su **FT** para poder pasar al **EE** :

```
h1=iir(3,'lp','butt',[.3 0],[0 0])
 h1  =

                                        2           3
   0.5138312 + 1.5414936z + 1.5414936z + 0.5138312z
   ------------------------------------------------
                                      2   3
      0.0562972 + 0.4217870z + 0.5772405z + z

h1=syslin('d',h1);
```

```
s1=tf2ss(h1)
 s1  =


       s1(1)    (state-space system:)

 lss

       s1(2) = A matrix =


  !   0.1885381  - 0.5295125    4.163D-16 !
  !   0.5652611  - 0.3696295  - 0.2846464 !
  !   0.0793729    0.4230663  - 0.3961492 !

          s1(3) = B matrix =

  ! - 1.3233236 !
  !   0.7457050 !
  ! - 0.0391397 !


          s1(4) = C matrix =

  ! - 0.9407294    3.331D-16    4.163D-16 !

          s1(5) = D matrix =

     0.5138312

          s1(6) = X0 (initial state) =

  !   0. !
  !   0. !
  !   0. !

          s1(7) = Time domain =

   d
```

- ***Interconexión de Sistemas*** : Los Sistemas Lineales creados en el entorno *Scilab* pueden ser interconectados en serie o en paralelo. Vamos a ver cuatro ejemplos de los sistemas S_1 y S_2 que podrían estar en su representación del EE o FT :



- ***Discretización de Sistemas Continuos:*** En *Scilab* un sistema lineal continuo en el tiempo representado por su EE o su FT, puede ser convertido en discreto dentro del mismo dominio temporal, mediante la función **dscr** :

  Si consideramos el siguiente EE continuo (C) :

$$(C)\begin{cases} \dot{x}(t) &= Ax(t)+Bu(t) \\ y(t) &= Cx(t)+Du(t) \end{cases}$$

  De la formula de variación de constantes podemos calcular el valor del estado x(t) en cualquier instante de tiempo t :

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\sigma)}Bu(\sigma)d\sigma$$

  Siendo h los pasos de tiempo del muestreo de la señal continua. La entrada u se mantiene constante en los intervalos de longitud h, luego el *modelo discreto* (D) obtenido del anterior (C) será :

$$(D)\begin{cases} x(n+1) &= A_h x(n)+B_h u(n) \\ y(n) &= C_h x(n)+D_h u(n) \end{cases}$$

$$A_h = \exp(Ah)$$
$$B_h = \int_0^h e^{A(h-\sigma)}Bd\sigma$$
$$C_h = C$$
$$D_h = D$$

La sintaxis para esta función **dscr** es :

```
-->[f,g[,r]]=dscr(a,b,dt[,m])
```

Siendo a y b matrices asociadas al EE continuo, mientras que f y g las matrices resultantes para el EE discreto

$$x(n+1) = Fx(n) + Gu(n)$$

Si el argumento de entrada en la función **dscr** fuera el EE continuo como una lista sl, sería :

```
-->[sld[,r]]=dscr(sl,dt[,m])
```

siendo **sld** la lista que representa la lista EE discreto.

Si el EE continuo lo tengo representado mediante su FT h, esta será el argumento de entrada de la función **hd** (FT del discreto) :

```
-->[hd]=dscr(h,dt)
```

Un ejemplo de uso de **dscr** puede ser el siguiente :

```
    a=[2 1;0 2]
 a  =

 !   2.    1. !
 !   0.    2. !

    b=[1;1]
 b  =

 !   1. !
 !   1. !

   [sld]=dscr(syslin('c',a,b,eye(2,2)),.1);

   sld(2)
 ans  =

 !   1.2214028    0.1221403 !
 !   0.           1.2214028 !

   sld(3)
 ans  =

 !   0.1164208 !
 !   0.1107014 !
```

- ***Filtrado y representación gráfica de***

El filtrado de señales mediante filtros representados por su EE o por su FT se realiza con la función **flts** la cual tiene dos formatos :

Cuando el sistema lineal viene dado en el EE :

```
-->[y[,x]]=flts(u,sl[,x0])
```

Cuando el sistema lineal viene dado por su FT :

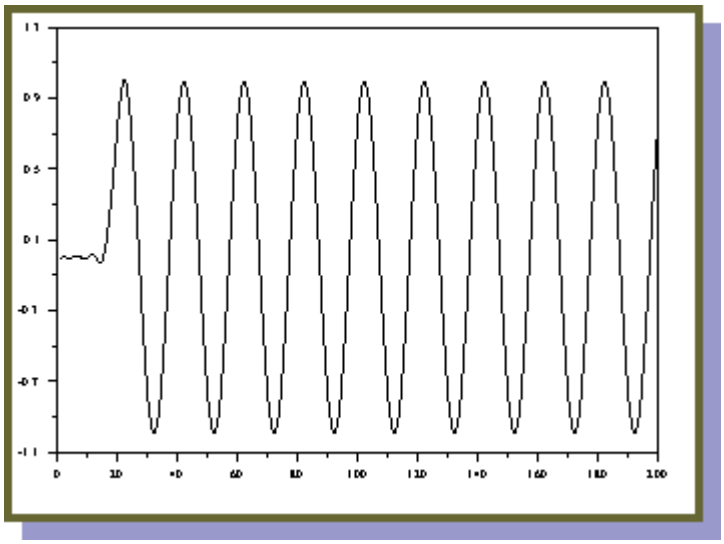
```
-->y=flts(u,h[,past])
```

Con el comando **plot** se representa gráficamente cualquier señal, en este apartado vamos a ver su uso más simple, para otros usos más complejos se recomienda la consulta de la *ayuda on-line*.

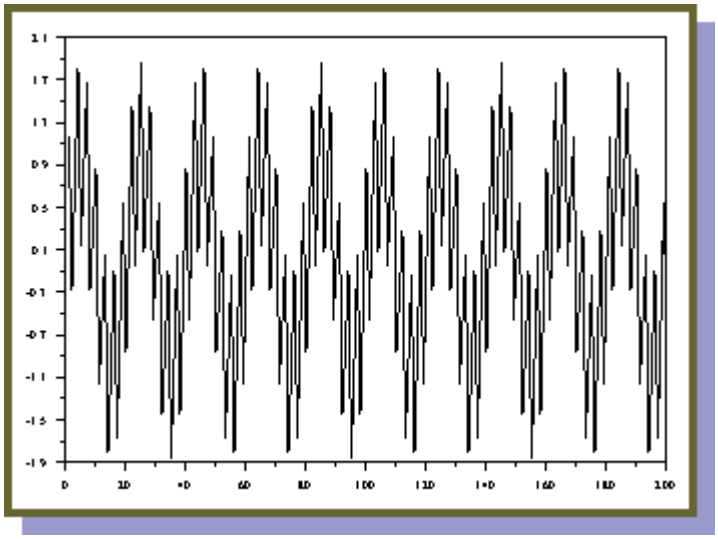
Vamos a ver un ejemplo del uso del comando **plot** y **flts** en el segundo caso , para ello generamos dos señales sinusoidales **x1** y **x2**, definimos un filtro de respuesta impulsional finita **wfir** (que todavía no hemos visto en este manual), sumamos las dos señales de entrada al filtro obteniendo **x**, ponemos el filtro en su FT **hz**, obteniendo **yhz** la señal filtrada y representándola gráficamente :

```
[h,hm,fr]=wfir('lp',33,[.2 0],'hm',[0 0]);

t=1:200;

x1=sin(2*%pi*t/20);

x2=sin(2*%pi*t/3);


x=x1+x2;

z=poly(0,'z');

hz=syslin('d',poly(h,'z','c')./z**33);

yhz=flts(x,hz);

plot(yhz);
```

La señal de salida del filtro yhz :



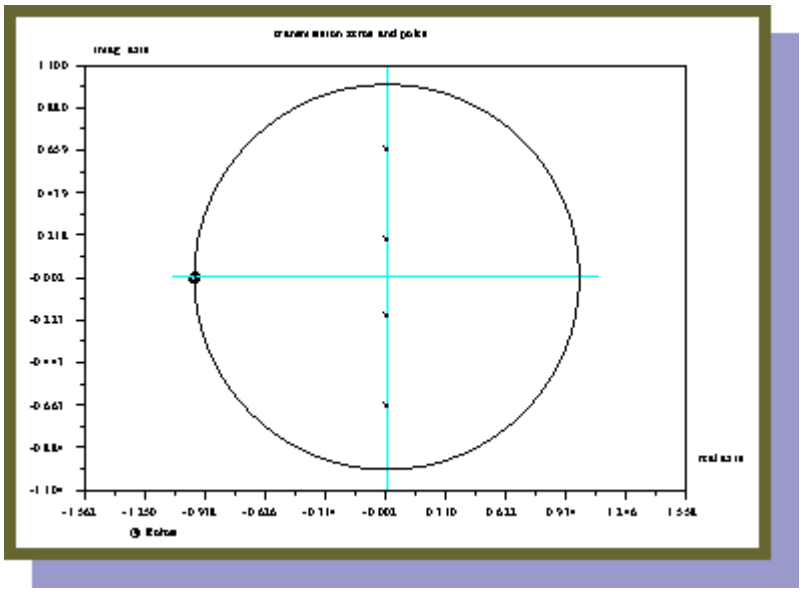
Si quisiera visualizar **la señal x en la entrada del filtro** tendría que poner **plot(x)** obteniendo :



- ***Representación de Polos y Ceros :*** Para representar gráficamente los polos y ceros de un filtro se utiliza **plzr**. En el ejemplo siguiente se muestra un ejemplo de esta función para un *filtro IIR* (respuesta impulsional infinita) :

```
hz=iir(4,'lp','butt',[.25 0],[0 0])
 hz  =

                                        2          3          4
   0.1879617 + 0.7518468z + 1.1277702z + 0.7518468z + 0.1879617z
   ---------------------------------------------------------------
                                      2          3   4
      0.0176648 + 1.928D-17z + 0.4860288z + 4.317D-17z + z

plzr(hz)
```

Los polos y ceros del filtro IIR son :



- ***Respuesta en frecuencia (Diagrama de Bode, Retardo de Grupo y Fase) :***

Diagrama de Bode : En Scilab se utiliza la función **bode**, cuya sintaxis es según parta de EE o FTes:

Si parto de la FT :

```
-->bode(h,fmin,fmax[,step][,comments])
```

Si parto del EE :

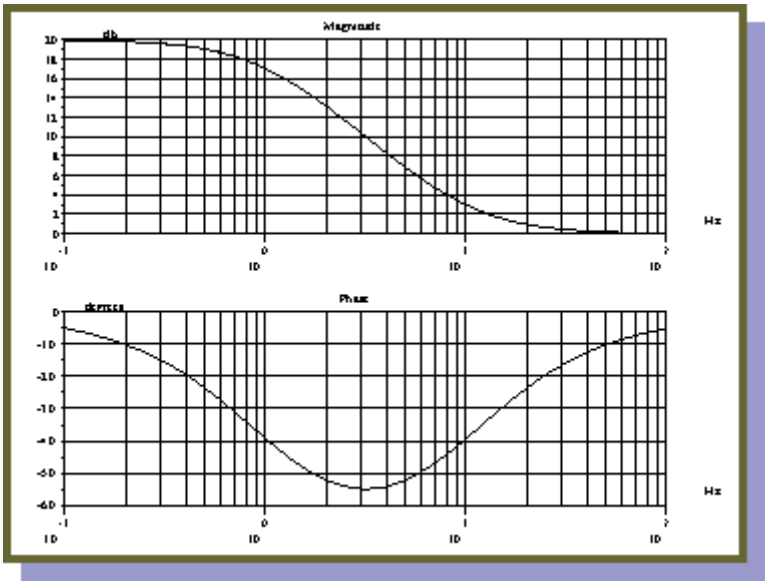
```
-->sl=syslin(domain,a,b,c[,d][,x0])
-->bode(sl,fmin,fmax[,pas][,comments])
```

vamos a aplicar **bode** por ejemplo al siguiente EE :

$$\begin{aligned} \dot{x} &= -2\pi x + u \\ y &= 18\pi x + u. \end{aligned}$$

```
a=-2*%pi;b=1;c=18*%pi;d=1;
sl=syslin('c',a,b,c,d);
bode(sl,.1,100);
```

El diagrama de *Bode* es el siguiente :



- ***Retardo de Grupo (t_g) y Fase (t_p):***

  Recordando como se definen, partiendo de la *transformada de Fourier* de un sistema H(w) :

$$H(\omega) = A(\omega)e^{j\theta(\omega)}$$

$$t_p(\omega) = \theta(\omega)/\omega$$

$$t_g(\omega) = d\theta(\omega)/d\omega.$$

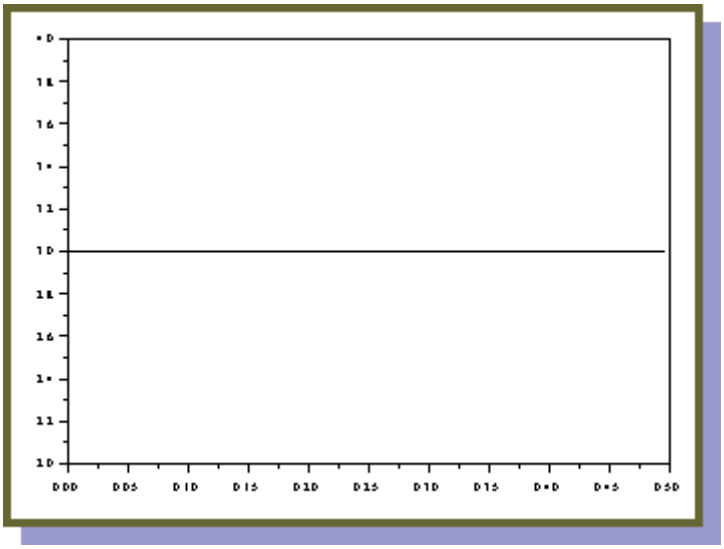
Como es sabido es preferible que el retardo de grupo de un filtro sea constante, ya que si no lo es tiende a causar deformación en la señal. La función **group** cuya sintaxis es :

```
-->[tg,fr]=group(npts,h)
```

Primero vamos a ver ***un ejemplo*** con un **Filtro de Fase lineal** diseñado con la función **wfir** :

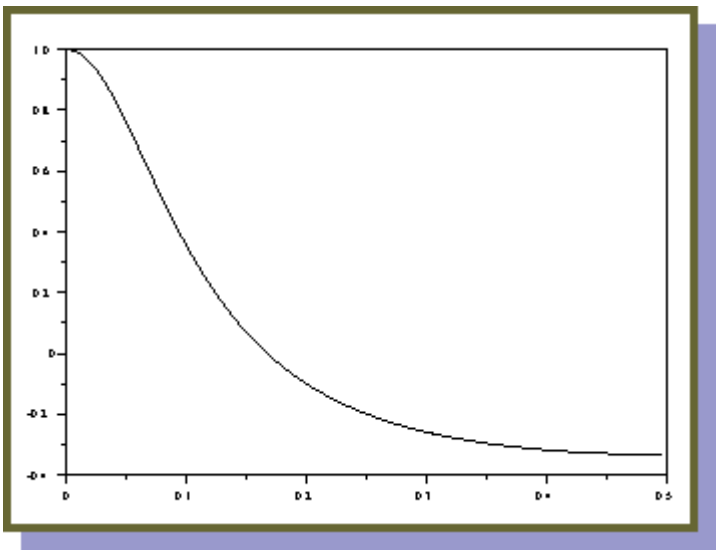
```
[h w]=wfir('lp',7,[.2,0],'hm',[0.01,-1]);

h'
 ans  =

! - 0.0049893 !
!   0.0290002 !
!   0.2331026 !
!   0.4       !
!   0.2331026 !
!   0.0290002 !
! - 0.0049893 !

[tg,fr]=group(100,h);

plot2d(fr',tg',1,'011',' ',[0,2,0.5,4.])
```

El *retardo de grupo* es una constante como cabría esperar para un *filtro de fase lineal* :



Otro ***segundo ejemplo*** lo aplicamos a la **FT de un Filtro**, expresada de **forma racional polinómica** :

```
z=poly(0,'z');

h=z/(z-0.5)
 h  =

      z
   -------
   - 0.5 + z

[tg,fr]=group(100,h);

plot(fr,tg)
```



- ***La FFT (Transformada rápida de Fourier) y la DFT (Transformada de Fourier discreta)*** :

La FFT es un algoritmo de gran eficiencia computacional para el cálculo de la DFT de secuencias de longitud finita discretas en el tiempo.

Partiendo de la definición de la DFT se requieren para su calculo del orden de N^2 multiplicaciones mientras que la FFT requiere del orden de $N \log_2 N$ multiplicaciones.

La definición de la ***DFT DIRECTA*** para una secuencia finita $x(n)$ de longitud N es :

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}nk}.$$

De la definición anterior que $X(k)$, la DFT de $x(n)$, es periódica con periodo N (debido al hecho de que fijado n el término $\exp(-j2\pi nk/N)$ es periódico con periodo N).

La definición de la llamada ***DFT INVERSA*** es :

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}nk}.$$

$n = 0,1,....., N-1$

$k = 0,1,....., N-1$

Vamos a representar de *forma matricial* la definición de *DFT DIRECTA* :

$$\begin{bmatrix} X(1) \\ X(2) \\ \vdots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{-j\frac{2\pi}{N}} & e^{-j\frac{4\pi}{N}} & \cdots & e^{-j\frac{2(N-1)\pi}{N}} \\ 1 & e^{-j\frac{4\pi}{N}} & e^{-j\frac{8\pi}{N}} & \cdots & e^{-j\frac{4(N-1)\pi}{N}} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & e^{-j\frac{2(N-1)\pi}{N}} & e^{-j\frac{4(N-1)\pi}{N}} & \cdots & e^{-j\frac{(N-1)^2\pi}{N}} \end{bmatrix} \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{bmatrix}$$

como podemos ver se requieren del orden de N^2.

El algoritmo FFT es el más efectivo para el calculo de la DFT. Estas ventajas es que la FFT utiliza ciertas simetrías y periodicidades que existen en el calculo de la DFT, la FFT asume que $N=2^{\gamma}$, siendo γ un entero positivo y pudiendo calcular la FFT como :

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}nk}$$

$$= \sum_{n=even} x(n)e^{-j\frac{2\pi}{N}nk} + \sum_{n=odd} x(n)e^{-j\frac{2\pi}{N}nk}$$
$$= \sum_{r=0}^{\frac{N}{2}-1} x(2r)e^{-j\frac{2\pi}{N/2}rk} + e^{j\frac{2\pi}{N}k} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)e^{-j\frac{2\pi}{N/2}rk}$$

La última igualdad se ha obtenido haciendo el cambio de variable $n=2r$. Como vemos la anterior expresión está compuesta de la suma de dos secuencias de $N/2$ puntos de la DFT, una para los puntos desde $x(0)$, $x(2)$,....., $x(N-2)$ que es la secuencia de los $N/2$ pares y por otro $x(1)$, $x(3)$,.....,$x(N-1)$ que son los $N/2$ impares. Como vemos en la expresión anterior el cálculo de la FFT requiere N multiplicaciones adicionales por los N términos $\exp(-j2\pi nk/N)$, con $k = 0,1,.....,\ N-1$.

Por tanto las ventajas de calculo de la FFT radican en la suposición de que $N=2^{\gamma}$, para lo que en algunas aplicaciones se le añaden ceros y obtener una secuencia de longitud que cumpla

Scilab para el calculo de la FFT utiliza la función **fft**, cuya sintaxis es :

[x]=fft(a,-1)
[x]=fft(a,1)
x=fft(a,-1,dim,incr)
x=fft(a,1,dim,incr)

Explicación de los parámetros de entrada :

x : . vector real o complejo. Matriz real o compleja (2-dim fft)

a : vector real o complejo.

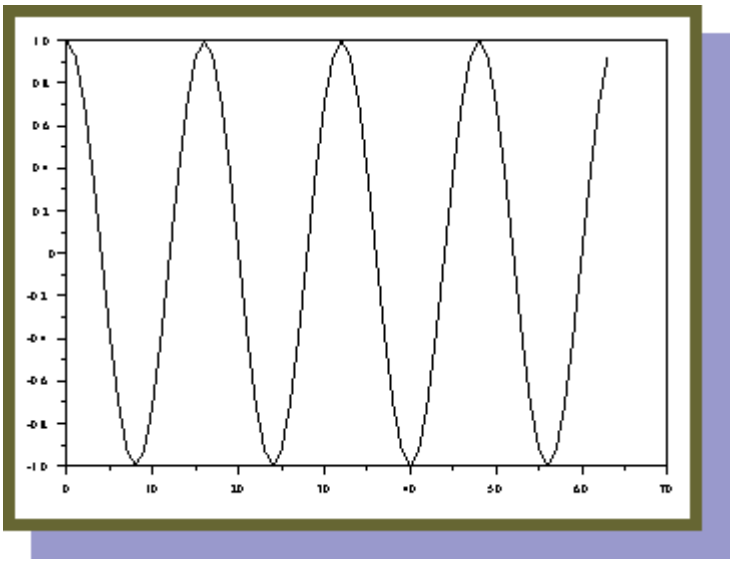
dim : entero

incr : entero

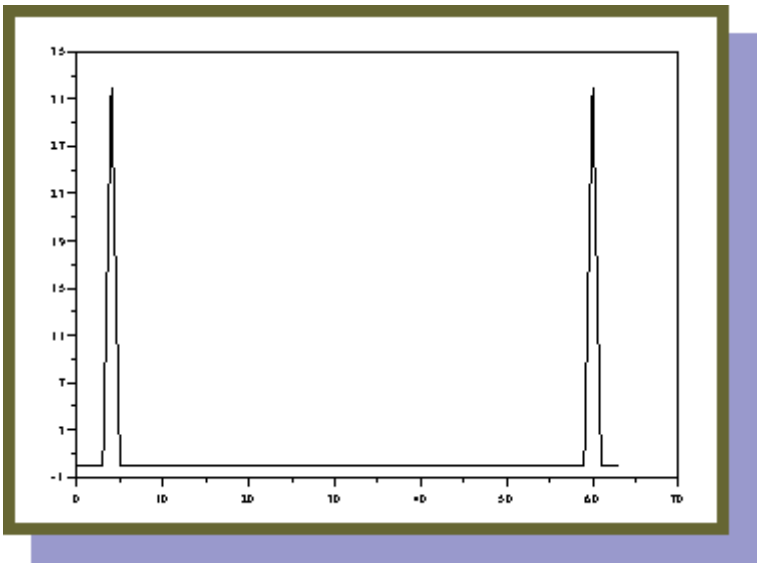
Vamos a ver ***un ejemplo*** **simple** de una **función coseno** que es transformada mediante la **fft** :

```
x=0:63;y=cos(2*%pi*x/16);

yf=fft(y,-1);

plot(x,real(yf)');
```

La *función coseno* sin transformar es la siguiente (si hago **plot(y)**) :



Siendo la *DFT* de la *función coseno* :

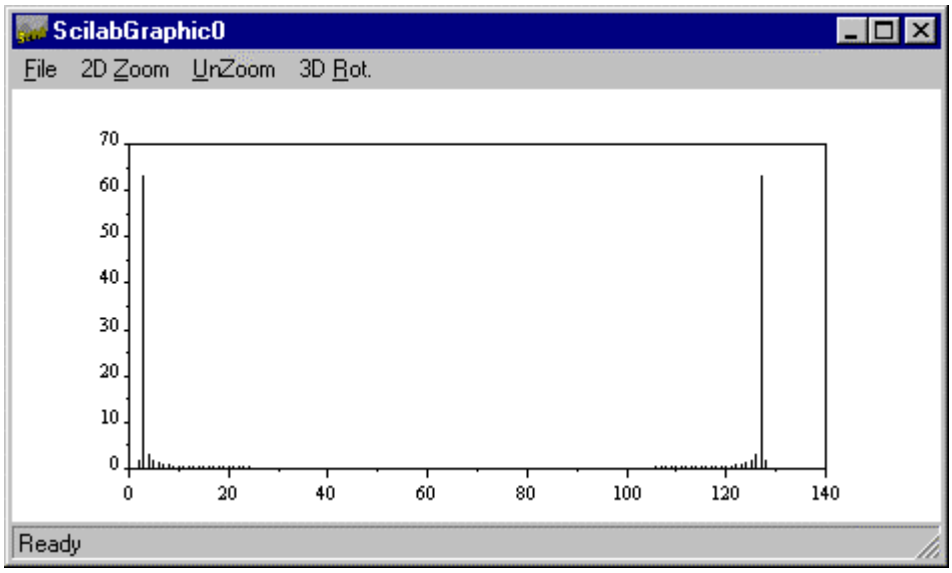


Como es $\pi/16$ de 64 puntos, deberíamos estar esperando dos picos en su DFT, uno en k=4 y otro en k=60. Esto es debido a que el valor k=64 corresponde a una frecuencia de 2π y consecuentemente el valor debe corresponder a la frecuencia $2\pi/16$, mediante una simple regla de tres; y que corresponde a la señal coseno bajo estudio.

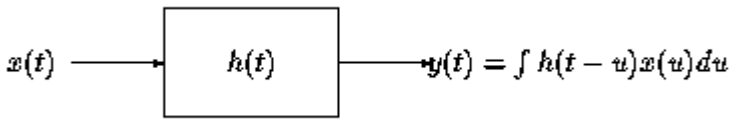
Otro ejemplo para una ***función seno*** es el siguiente :

```
-->s=sin(0:0.1:5*%pi);

-->
-->ss=fft(s(1:128),-1);
```

```
-->xbasc();

-->
-->plot2d3("enn",1,abs(ss)');
```



- ***Convolución :*** *Scilab* utiliza la función **convol** cuya sintaxis es :




```
-->y=convol(h,x)
```

A continuación un ejemplo :

```
x=1:3
 x  =

!   1.    2.    3. !

h=ones(1,4)
 h  =

!   1.    1.    1.    1. !

y=convol(h,x)
 y  =

!   1.    3.    6.    6.    5.    3. !
```

Para ver otros formatos de la función convolución puedo consultar la *ayuda on-line*.

5. DISEÑO DE FILTROS CON *SCILAB* .

5.a. Introducción.

La teoría general del *diseño de Filtros Digitales* nos lleva al agrupamiento de las funciones para el diseño en cuatro métodos:

- Diseño de *filtros IIR* usando *prototipos analógicos*.
- Diseño de *filtros IIR directo*.
- Diseño de *filtros FIR directo*.
- Diseño de *filtro inverso*.

Los tres primeros producen filtros selectivos de frecuencias que son diseñados a partir de las especificaciones de funcionamiento de la respuesta en magnitud. Las técnicas de la última categoría encuentran en los coeficientes del filtro responsables de los datos de la respuesta temporal o de la respuesta en frecuencia dados.

Hay varias formas de especificar el filtro que se necesita. Una especificación amplia puede ser " para un sistema con una velocidad de muestreo de 500 Hz, se necesita un filtro que elimine el ruido por encima de 30 Hz ". Una especificación más rigurosa fuerza a dar especificaciones de rizado en banda pasante, atenuación en la banda de rechazo o anchura en la transición. Una especificación muy precisa pediría conseguir los objetivos de funcionamiento con el filtro de orden mínimo o una forma de magnitud determinada o requerir un filtro FIR.

Si se requieren especificaciones amplias, la clase de filtros Butterworth IIR es suficiente en la mayoría de los casos. Para requerimientos más rigurosos, se pueden utilizar para conseguir este tipo de funcionamiento, los filtros Butterworth , Chebyschev I, Chebyschev II y elípticos. Mientras que para especificaciones muy precisas se usan las técnicas de diseño de filtros FIR directo y IIR directo.

5.b. Diseño de Filtros de Respuesta Impulsional Finita (FIR).

5.b.1. Técnicas de Ventana (Windowing).

En teoría el diseño de filtros FIR es sencillo. Se toma la *transformada inversa de Fourier* de la respuesta en frecuencia deseada y se obtiene la *respuesta impulsional discreta* en el dominio temporal , de acuerdo con:

$$h(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega)e^{j\omega n}d\omega \quad -\infty < n < \infty$$

En la práctica el problema es que muchos de los filtros de interés tienen una *respuesta impulsional infinita* y una relación causa-efecto no casual. Un ejemplo claro de esto, viene dado por este *Filtro Pasa Bajo* con una frecuencia de corte ω_c :

$$H(\omega|\omega_c) = \begin{cases} 1, & |\omega| \le \omega_c \\ 0, & \text{otherwise} \end{cases}$$

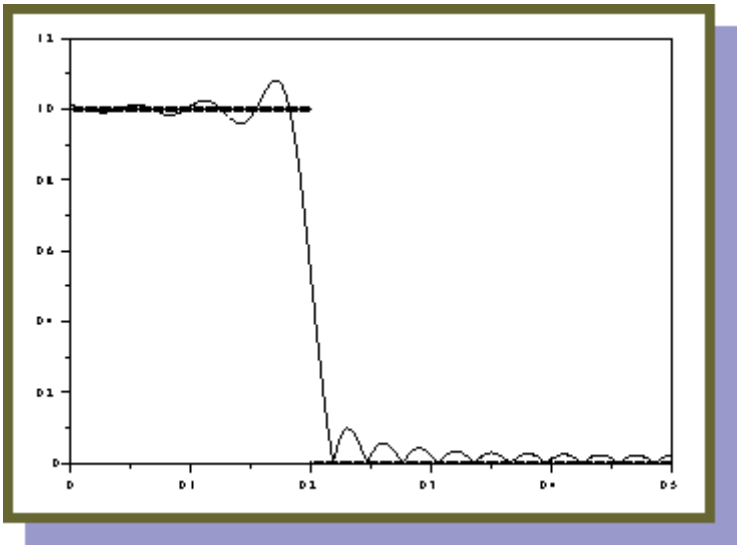
Su *respuesta impulsional* se obtiene introduciendo esta $H(\omega/\omega_c)$ en la primera formula de arriba, obteniendo :

$$h(n|\omega_c) = \frac{1}{\pi n}\sin(\omega_c n) \quad -\infty < n < \infty$$

Para obtener una **longitud finita** es necesario implementar una técnica mediante la cual se pueda tomar N elementos de h(n/ω_c) y que centrado en n=0, eliminando los elementos sobrantes. Esta operación puede ser representada mediante la multiplicación de la secuencia de elementos por una **Ventana Rectangular** de la siguiente forma :

$$R_N(n) = \begin{cases} 1, & 0 \le n \le N-1 \\ 0, & \text{otherwise} \end{cases}$$

Pero ahora **la respuesta en frecuencia del filtro con ventana aplicada**, es la siguiente para **un filtro de longitud N=33 y con ω_c = 0.2 :**



Como puede verse el efecto de la ventana (*Windowing*) con respecto al filtro ideal es que se produce rizado (ripple) tanto en la banda pasante (*pass band*) como en la banda de rechazo (*stop band*) y las mayores desviaciones en las proximidades de ω_c = 0.2. El rizado es debido a la *convolución* de la respuesta ideal del filtro con la respuesta en frecuencia de la ventana rectangular.

Para muchas aplicaciones el rizado en la respuesta en frecuencia es inaceptable, pero es posible disminuir la cantidad de rizado mediante el uso de otro tipo de ventanas. Por este motivo **Scilab** otras ventanas como : la *ventana triangular*, *de Hamming generalizada*, *de Kaiser* y *de Chebyshev*.

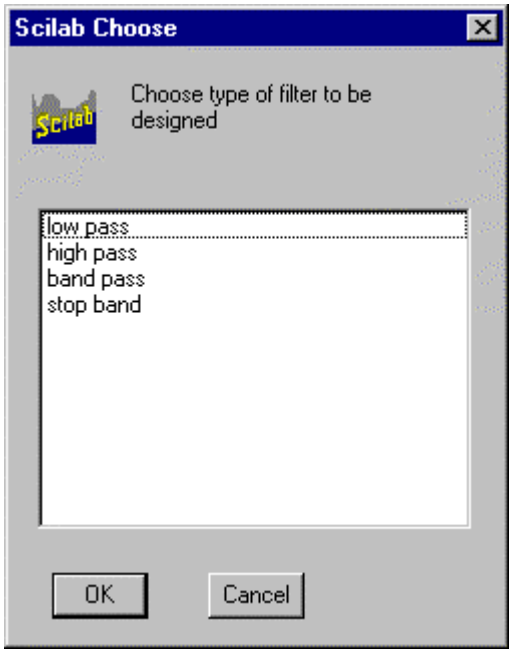
5.b.2. La función **wfir.**

La sintaxis de la función **wfir** tiene dos formatos :

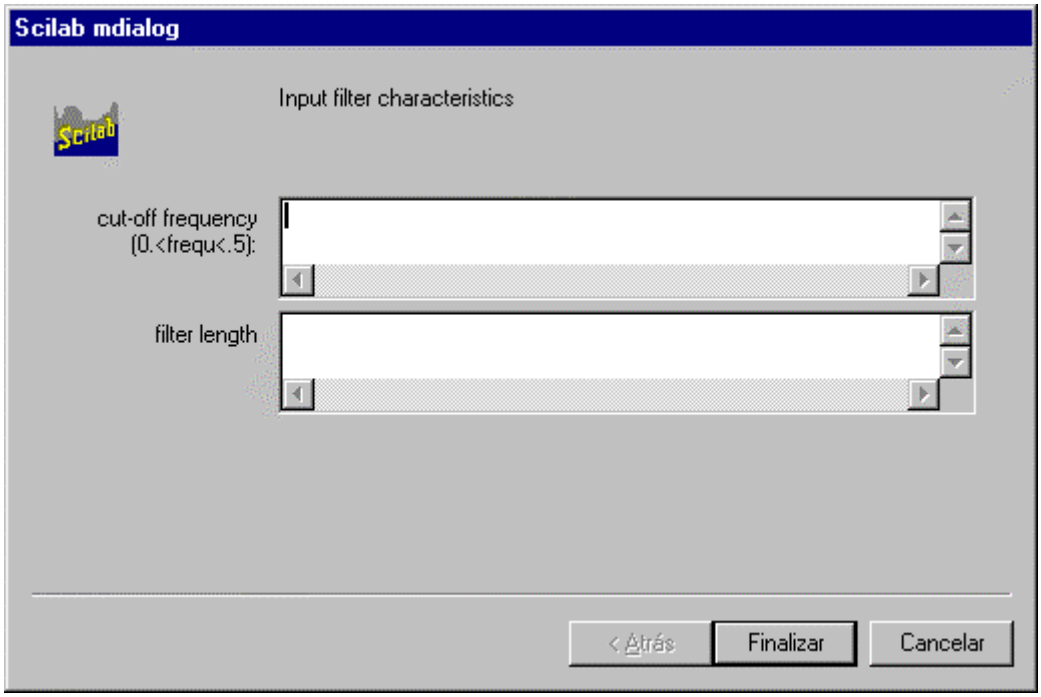
- ***Primer formato :***

```
--> [wft,wfm,fr]=wfir()
```

Los paréntesis son requeridos como parte del nombre, este formato de la función es interactivo con el usuario apareciendo una primera ventana donde elijo el **tipo de filtro** :



A continuación aparecerá una segunda ventana, donde introduzco la **frecuencia de corte** y la **longitud** del tipo de filtro elegido :



Y una tercera ventana para que seleccione el **tipo de ventana** a aplicar al filtro:

Los parámetros de retorno de **wfir** son tres :

wft : Vector que contiene los coeficientes del filtro de longitud n con ventana aplicada.

wfm : Vector de longitud 256 que contiene la respuesta en frecuencia del filtro con ventana.

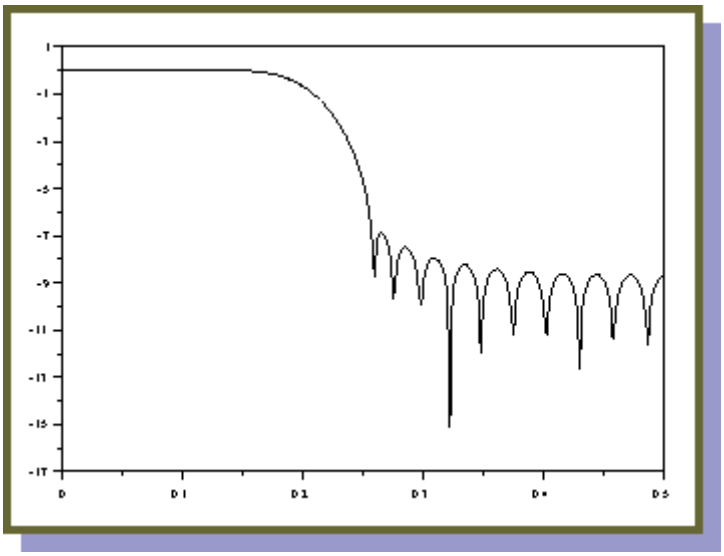
fr : Vector de longitud 256 que contiene los valores del eje de frecuencia ($0 \le fr \le 0.5$) asociados a los valores de wfm.

- ***Segundo formato :***

```
--> [wft,wfm,fr]=wfir(ftype,forder,cfreq,wtype,fpar)
```

Este formato ya no es interactivo para el usuario y por tanto los parámetros de entrada deben ser pasados como argumentos a la función . el primer argumento **ftype** indica el tipo de filtro que se quiere construir pudiendo elegir entre : **lp** : filtro paso bajo, **hp** : filtro paso alto, **bp** : filtro paso banda o **sp**: elimina banda ; el segundo argumento **forder** debe ser un entero positivo que indique el orden del filtro deseado; el tercer argumento **cfreq** es un vector de dos elementos, utilizando en los filtros **lp** o **hp** , **cfreq(1)** como la frecuencia de corte ,y en **bp** o **sp** para indicar la frecuencia inicial y final de la banda; el cuarto argumento **wtype** indica el tipo de ventana : **re** = rectangular, **tr** = triangular, **hm** = hamming, **kr** = kaiser y **ch** = chebyshev ; y por último el **fpar** que es un vector con dos elementos donde se introducen los parámetros α (filtros Hamming) y (filtros Kaiser)

Vamos a hacer ***un ejemplo*** de diseño con el formato interactivo para **un filtro pasa bajo de longitud n=33 usando una ventana Kaiser con parámetro β=5.6** para ello iré introduciendo estos datos en las ventanas interactivas y al final haré un **plot** de los vectores de retorno, obteniendo :



5.b.3. Optimizado en el diseño de Filtros con *Scilab*.

Lo más importante después de las técnicas de aplicación de ventanas en el diseño es la obtención de un filtro el cual esté tan cerca del filtro deseado con un *error cuadrático mínimo*. Para ello *Scilab* tiene implementados los algoritmos matemáticos *Minimax* y de *Remez*.

Scilab incluye la función **remezb** cuya sintaxis es :

```
--> an=remezb(nc,fg,df,wf)
```

Este algoritmo busca uniformemente el *valor minimizado* de **la función de error E(f)** dentro del intervalo [f_0,f_1] que se define como :

$$E(f) = W(f)(D(f) - H(f))$$

Donde D(f) es una función que quiere ser aproximada por H(f), y W(f) es una función de peso positiva, de tal forma que el algoritmo busca de forma iterativa tal que :

$$H^*(f) = \arg \min_{H(f)} \|E(f)\|_\infty$$

$$\|E(f)\|_\infty = \max_{f_0 \le f \le f_1} |E(f)|$$

La norma de E(f) es conocida como la ***norma minimax o Chebyshev***. *Scilab* tiene implementado obligar a la función H(f):

$$H(f) = \sum_{n=0}^{N} a_n \cos(2\pi f n).$$

Dentro de un intervalo de aproximación de [0 , 0.5]. Pero en el diseño de filtros digitales H(f) representa la transformada discreta de Fourier de un filtro FIR de fase lineal, longitud par y simetría par. Lo que me voy a encontrar al intentar resolver es un sistema lineal sobredeterminado con infinitas ecuaciones que cada una contiene N+1 incógnitas a_n ,

Para ver un ejemplo ver **Signal Processing with Scilab**, ya que el profundizar en esta función se deja para el usuario experimentado.

Los *filtros digitales con respuesta a un impulso de duración finita (FIR)* tiene tanto ventajas como desventajas si se comparan con los *filtros de respuesta a un impulso de duración infinita (IIR).* Las principales ventajas de los filtros FIR son:

- Pueden tener fase lineal exacta.
- Son siempre estables.
- Los métodos de diseño son generalmente lineales.
- Pueden ser realizados eficientemente en hardware.
- Los transitorios de arranque del filtro son de duración finita.

La principal desventaja es :

- Requieren un mayor orden del filtro que los filtros IIR para obtener el mismo nivel de funcionamiento.

5.c. Diseño de Filtros de Respuesta Impulsional Infinita (IIR).

5.c.1. Filtros Analógicos.

En esta sección recordaremos algunos de los filtros analógicos (tiempo continuo) clásicos . Estos son definidos en el dominio frecuencial mediante su función de transferencia racional de la siguiente forma:

$$H(s) = \frac{\sum_{i=0}^{m} b_i s^i}{1 + \sum_{i=1}^{n} a_i s^i}$$

El problema radica en determinar los coeficientes a_i y b_i o de forma equivalente los ceros z_i y los polos p_i de H de tal forma que se consigan las especificaciones de la magnitud de respuesta cuadrada definida como :

$$h^2(\omega) = |H(i\omega)|^2 = H(s)H(-s)|_{s=i\omega}$$

Siendo $h(\omega)$ el espectro de salida de un filtro lineal que admite un ruido blanco como entrada. Para los ejemplos de *filtros analógicos* siguientes vamos a considerar sólo **filtros pasa bajo**, de tal forma que lo que deseamos obtener será $h(\omega)=0$ para $\omega>\omega_C$ y $h(\omega)=1$ para $\omega<\omega_C$. Para *filtros pasa alto*, *pasa banda* o *elimina banda* serán construidos fácilmente simplemente con un cambio de variables.

consiste en encontrar una función H_2 de variable compleja s, tal que $H_2(s)=H(s)H(-s)$. Siendo $H_2(i\omega)=h^2(\omega)$ ya que es simétrica con respecto al eje imaginario, así la función H_2 deberá ser elegida de forma racional definida por sus ceros y polos.

5.c.1.a Filtros Butterworth.

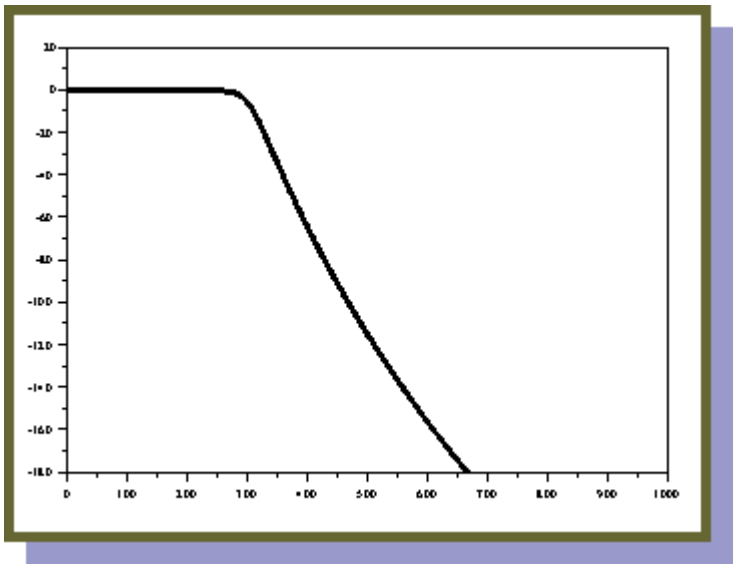
Su *magnitud de respuesta cuadrada* viene dada por :

$$h_n^2(\omega|\omega_c) = \frac{1}{1+(\frac{\omega}{\omega_c})^{2n}}$$

Siendo ω_C la *frecuencia de corte* y n el *orden del filtro*. Su respuesta puede ser representada gráficamente en *Scilab* mediante la función **buttmag**. El siguiente ***ejemplo*** corresponde a un **filtro Butterworth de orden 13 y frecuencia de corte 300**:

```
h=buttmag(13,300,1:1000);

mag=20*log(h)'/log(10);

plot2d((1:1000)',mag,[-1],"011"," ",[0,-180,1000,20]),
```

La *respuesta* es la siguiente , estando su magnitud representada en dB :



Como podemos ver en la figura la función decrece con el aumento de frecuencia. También podemos ver que es independientemente de n :

$$h_n^2(\omega_c|\omega_c) = \frac{1}{2}$$

Esto nos permite definir **la función de transferencia H(s) de un filtro de Butterworh**, de la siguiente manera :

$$H(s) = \frac{k_0}{\prod_{k=1}^{n}(s-p_k)}$$

$$p_k = e^{i\pi[1/2+(2k-1)/2n]} \qquad k = 1,\ldots,n$$

Esta FT la podemos obtener en *Scilab* mediante la función **zpbutt**, la cual calcula automáticamente los polos p_k y la ganacia k_0. ***Un ejemplo*** con n=7 y ω_C= 3, obteniendo la siguiente FT :

```
n=7;

omegac=3;

[pols,gain]=zpbutt(n,omegac);

h=poly(gain,'s','coeff')/real(poly(pols,'s'))
 h  =

                              2187
    -------------------------------------------------------------
                             2            3            4
    2187 + 3276.0963s + 2453.7738s + 1181.9353s + 393.97843s
                 5           6   7
      + 90.880512s + 13.481878s + s
```

La determinación del orden en un filtro de Butterworth en general se calcula dando la *atenuación* deseada 1/A en una *frecuencia normalizada* especifica $f=\omega_r/\omega_c$, mediante la siguiente expresión:

$$n = \frac{\log_{10}(A^2 - 1)}{2\log_{10}(f)}$$

5.c.1.b Filtros Chebyshev.

El polinomio T_n (x) de Chebyshev viene definido por :

$$T_n(x) = \begin{cases} \cos(n\cos^{-1}(x)) & \text{if } |x| < 1 \\ \cosh(n\cosh^{-1}(x)) & \text{otherwise} \end{cases}$$

Estos polinomios son usados para definir analíticamente la magnitud de respuesta cuadrada en frecuencia.

Filtros Chebyshev tipo 1 (rizado en la banda pasante) :

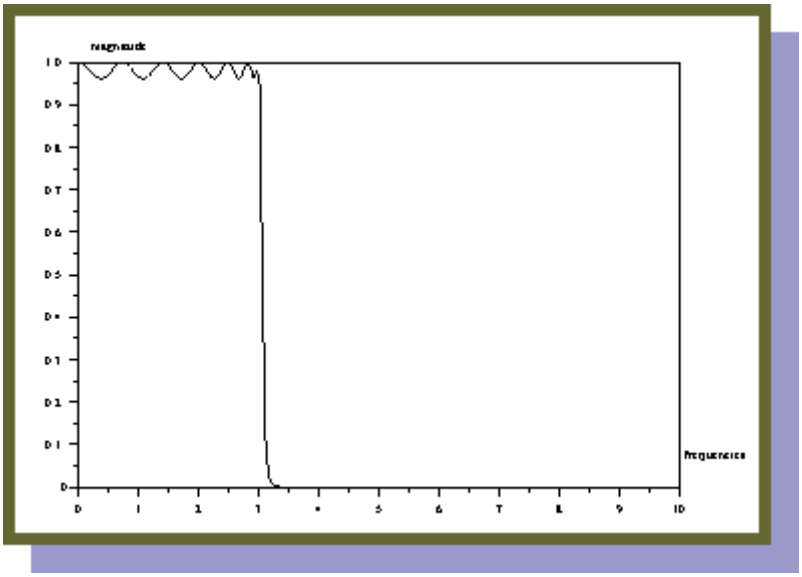
Se define su función de respuesta como :

$$h^2_{1,n}(\omega \mid \omega_c, \epsilon) = \frac{1}{1 + \epsilon^2 T_n^2(\frac{\omega}{\omega_c})}$$

Mediante la función **cheb1mag** calculo la función de respuesta anterior, una vez determinados los parámetros ω_c ,$\in$ **(** determina el rizado) y **n**. ***Un ejemplo*** es el siguiente:

```
n=13;epsilon=0.2;omegac=3;sample=0:0.05:10;

h=cheb1mag(n,omegac,epsilon,sample);

plot(sample,h,'frequencies','magnitude')
```

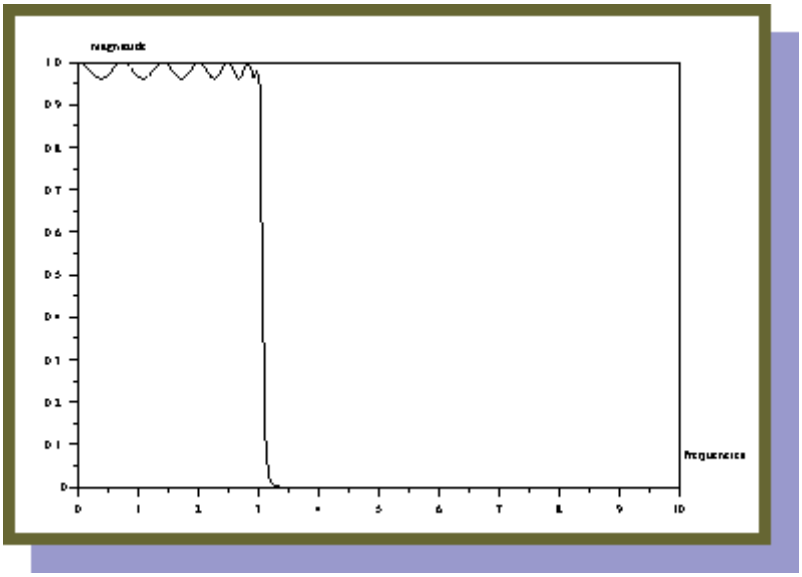
La *magnitud de respuesta* de este ***filtro Chebyshev tipo 1*** es :



También *Scilab* utiliza la función **zpch1** para calcular los polos y la ganancia de este filtro, y por tanto su FT, pudiendo también de esta obtener su magnitud de respuesta en frecuencia:

```
n=13;epsilon=0.2;omegac=3;sample=0:0.05:10;

[p,gain]=zpch1(n,epsilon,omegac);

//Transfer function computation tr_fct(s)=gain/deno(s)

tr_fct=poly(gain,'s','coef')/real(poly(p,'s'))
tr_fct  =

                                1946.1951
  ------------------------------------------------------------------
                                      2           3           4
  1946.1951 + 7652.7444s + 14314.992s + 18875.541s + 17027.684s
                 5           6           7           8
    + 13282.001s + 7398.971s + 3983.2216s + 1452.2192s
                  9           10          11          12  13
    + 574.73496s + 131.30929s + 39.153835s + 4.4505809s + s

//Magnitude of the frequency response computed along the

//imaginary axis for the values %i*sample...

rep=abs(freq(tr_fct(2),tr_fct(3),%i*sample));

plot(sample,rep,'frequencies','magnitude')
```

Como es lógico obtenemos la *misma respuesta* que por el otro camino:



La determinación del *orden del filtro* , viene dado por la expresión :

$$n = \frac{\cosh^{-1}\left(\frac{\sqrt{A^2-1}}{\epsilon}\right)}{\cosh^{-1}(f)} = \frac{\log(g+\sqrt{(g^2-1)})}{\log(f+\sqrt{(f^2-1)})}$$

$$g = \sqrt{\left(\frac{A^2-1}{\epsilon^2}\right)}$$

1/A = atenuación deseada.

Filtros Chebyshev tipo 2 (rizado en la banda de rechazo) :

Se define su *función de respuesta* como :

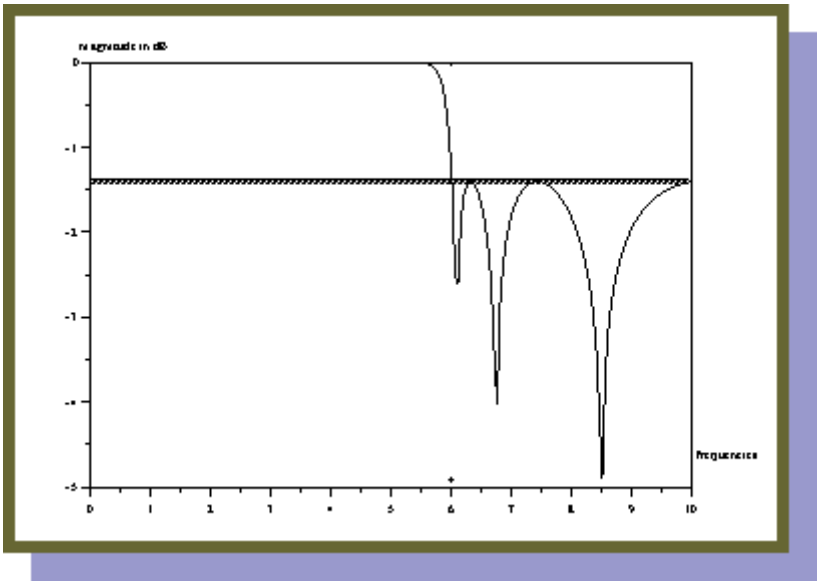
$$h^2_{2,n}(\omega \mid \omega_r, A) = \frac{1}{1+\frac{A^2-1}{T_n^2\left(\frac{\omega_r}{\omega}\right)}}$$

Mediante la función **cheb2mag** calculo la función de respuesta anterior, una vez determinados los ω_r (final de banda pasante**) ,A (**determina la atenuación) y **n**. ***Un ejemplo*** es el siguiente, **con n=10, A=1/0.2 y ω_r = 6 :**

```
n=10;omegar=6;A=1/0.2;sample=0.0001:0.05:10;

h2=cheb2mag(n,omegar,A,sample);

plot(sample,log(h2)/log(10),'frequencies','magnitude in dB')
```

```
//Plotting of frequency edges

minval=(-maxi(-log(h2)))/log(10);

plot2d([omegar;omegar],[minval;0],[-1],"000");

//Computation of the attenuation in dB at the stopband edge

attenuation=-log(A*A)/log(10);

plot2d(sample',attenuation*ones(sample)',[-2],"000")
```

La *magnitud* de respuesta de este *filtro Chebyshev tipo 2* es :



La función **zpch2** calcula los polos y ceros del **filtro de Chebyshev tipo 2**, introduciendo los parámetros ω_r, A y n. El rizado viene dado por la expresión :

$$\gamma = (A + \sqrt{A^2 - 1})^{1/n}$$

Con esta función puedo determinar la FT del filtro, vamos a ver ***un ejemplo*** con ω_r **= 6 , A = 5 y n = 10**.

```
n=10;

omegar=6;

A=1/0.2;

[z,p,gain]=zpch2(n,A,omegar);

num=real(poly(z,'s'));    //Numerator
```

```
den=real(poly(p,'s'));   //Denominator

transf=gain*num./den    //Transfer function
 transf  =

                                2           4         6       8       10
    6.192D+09 + 4.300D+08s + 10450944s + 103680s + 360s + 0.2s
  ------------------------------------------------------------------------
                                          2           3           4
  6.192D+09 + 1.526D+09s + 6.179D+08s + 1.069D+08s + 20878805s
                 5            6            7           8
    + 2494608.7s + 282721.94s + 21546.997s + 1329.062s
                  9  10
    + 50.141041s + s
```

El cálculo del *orden del filtro* coincide con el tipo 1:

$$n = \frac{\cosh^{-1}\left(\frac{\sqrt{A^2-1}}{\epsilon}\right)}{\cosh^{-1}(f)} = \frac{\log(g+\sqrt{(g^2-1)})}{\log(f+\sqrt{(f^2-1)})}$$

5.c.1.c Filtros Elípticos.

Los *filtros elípticos* presentan rizado tanto en la banda pasante como en la de rechazo y están basados en las propiedades de la función elíptica Jacobiana. Scilab utiliza **%asn** para calcular la integral elíptica , **%sn** que calcula la función elíptica de Jacobi y **%k** la integral elíptica completa de Jacobi; para estos filtros nos vamos a limitar a ser usuarios de ellas (para profundizar ver pag 94 de Signal Processing with Scilab).

La posición de los polos y ceros de la función **sn** (Jacobi) nos permitirá definir la magnitud de respuesta en frecuencia de este filtro. Mediante la función **ell1mag** siendo su sintaxis :

[v]=ell1mag(eps,m1,z)

Parámetros :

eps : Rizado de banda de paso.

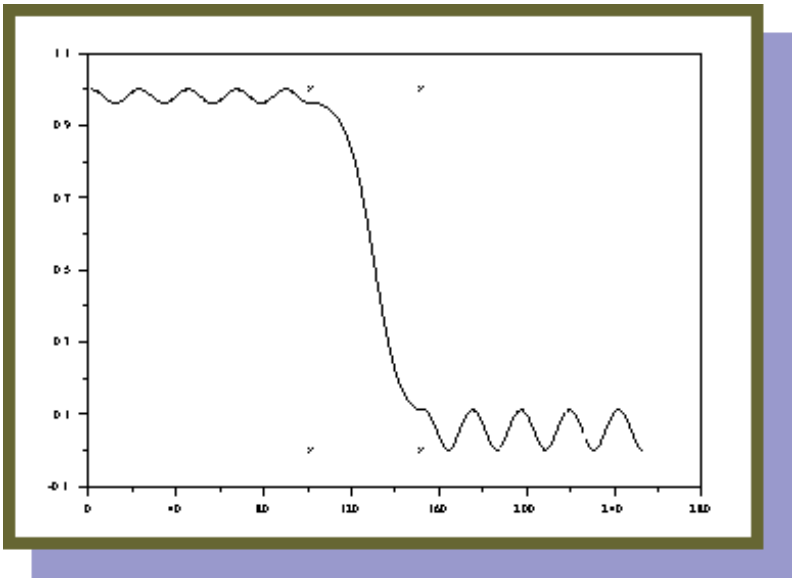
m1 : Rizado de banda de rechazo.

z : Vector con los puntos de muestreo en el plano complejo.

v : Vector con los valores del filtro elíptico.

Vamos a ver ***un ejemplo***, **filtro elíptico con n=9, $\in$ =0.2, A = 3 y m1=$\in^2$/(A^2-1) :**

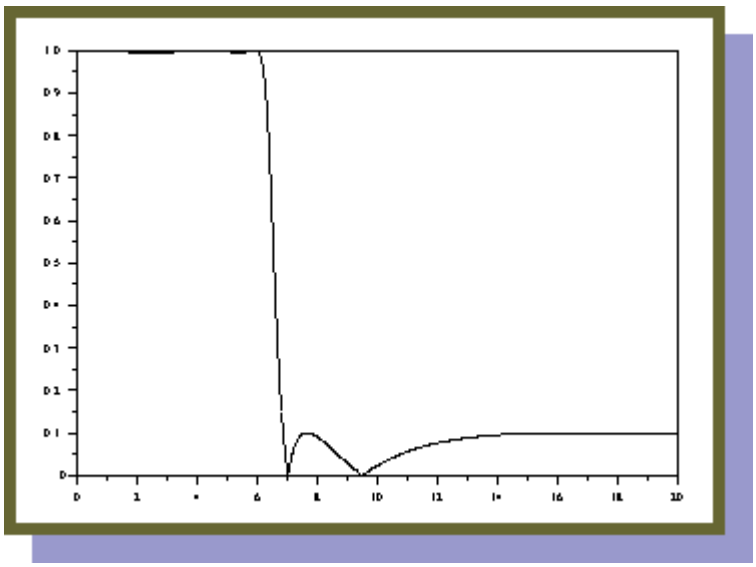
```
n=9;eps=0.2;A=3;m1=eps*eps/(A*A-1);

K1=%k(m1);K1T=%k(1-m1);

z1max=n*K1;z2max=K1T;

z1=0:(z1max/100):z1max;

z2=%i*(0:(z2max/50):z2max);z2=z2+z1max*ones(z2);

z3=z1max:-(z1max/100):0;z3=z3+%i*z2max*ones(z3);

plot(ell1mag(eps,m1,[z1,z2,z3]));

omc=prod(size(z1));

 omr=prod(size([z1,z2]));

 plot2d([omc,omc]',[0,1]',[-2],"000");

 plot2d([omr,omr]',[0,1]',[-2],"000");
```



La función **zpell** calcula los polos y ceros de los filtros elípticos, pudiendo determinar su FT, introduciendo **A,** $\in$**,**ω_c y ω_r**.** Vamos a ver ***un ejemplo*** de utilización de esta función (nota: la función **find_freq** se utiliza para calcular ω_r cuando me dan **A,** $\in$**,**ω_c y **find_ripple** para calcular $\in$ cuando me dan ω_r**,** ω_c y **A**):

```
epsilon=0.1;A=10;  //ripple parameters

m1=(epsilon*epsilon)/(A*A-1);n=5;omegac=6;

m=find_freq(epsilon,A,n);

omegar = omegac/sqrt(m)
 omegar  =

    6.8315017

[z,p,g]=zpell(epsilon,A,omegac,omegar);

//Now computes transfer function

num=real(poly(z,'s'));den=real(poly(p,'s'));

transfer=g*num/den
 transfer  =


                                        2          4
                  10783.501 + 340.56385s + 2.454884s
   ---------------------------------------------------------------
                                      2           3           4   5
   10783.501 + 3123.7307s + 773.85348s + 120.79402s + 11.89508s + s

//Plot of the response

sample=0:0.01:20;

rep=freq(g*num,den,%i*sample);


 plot(sample,abs(rep))
```

La *respuesta* del filtro obtenida con **zpell** :



5.c.2. Diseño de Filtros IIR a partir de Filtros Analógicos.

Una forma de *diseñar filtros IIR* es a partir de aproximaciones discretas de los *filtros analógicos*. Si usamos un método de aproximación este debe mantener las características de diseño del . Partiendo de un filtro analógico estable y casual , el filtro digital obtenido a partir de él mediante la técnica de aproximación debería ser también estable y casual. Por ejemplo, un
de diseño tales como localización de la frecuencia de corte, ancho de la banda de transición y la cantidad de error en la banda pasante y de rechazo. La aproximación debería preservar estas especificaciones de diseño.

y uno ***digital*** bajo esta aproximación es :

$$H(z)|_{z=e^{sT}} = \frac{1}{T}\sum_{k=-\infty}^{\infty} H(s+j\frac{2\pi k}{T}).$$

Como vemos consiste en la superposición de la variaciones de H(s) a lo largo de los ejes jω, pudiendo ocurrir **Aliasing** si el *filtro analógico* no tiene la banda limitada.

Existen técnicas de aproximación de filtros analógicos para evitar el problema del **Aliasing**, que son los conocidos como aproximación de la derivada y aproximación de la integral. En las transformaciones obtenidas en estos métodos para pasar del plano s al z, aparecen a veces una frecuencia que distorsiona el diseño del

En el *método de la aproximación de la integral* se llega a la correspondencia entre el plano s y z mediante la que se conoce como **transformación bilineal** :

$$z = \frac{1+(sT/2)}{1-(sT/2)}$$

En la mayoría de los filtros de interés tienen magnitudes que permanecen constante y que no les afecta la frecuencia distorsionadora. Por lo cual la transformación bilineal mantiene las características de diseño de los

5.c.2.a Diseño de Filtros Pasa Bajo.

La *transformación bilineal* es la mejor para convertir *filtros analógicos* en *digitales*. En los siguientes apartados vamos a ver otras tranformaciones que podemos usar para convertir un *filtro digital pasa bajo* a *pasa alto, pasa banda, elimina banda* o en otro *pasa bajo*.

Si consideramos que :

$$s = \frac{2j}{T}\tan(\omega/2) = \sigma + j\Omega.$$

La **frecuencia** ω para el *filtro digital* le corresponde una para el *filtro analógico* que es :

$$\Omega = \frac{2}{T}\tan(\omega/2).$$

Para el diseño de un *filtro digital pasa bajo* con una **frecuencia de corte** ω_c requiere **una frecuencia de corte de diseño** del *filtro analógico* Ω_c:

$$\Omega_c = 2\tan(\omega_c/2).$$

Cualquiera de los *filtros pasa bajo analógicos* cuyas técnicas de diseño ya se han visto (como los diseños de filtros de Butterworth, Chebyshev y elípticos) pueden ser usadas para obtener el diseño de *un filtro digital pasa bajo.*

Vamos a ver **un ejemplo** de diseño con *Scilab* de **un filtro digital pasa bajo con una** $\omega_c = \pi/2$:

Calculamos la frecuencia analógica $\Omega_c = 2 \tan (\pi/4) = 2$, me voy a basar por ejemplo para el diseño en **un filtro analógico de Butterworth de orden n=3 y un rizado en la banda de paso de 0.05**
zpch1 de *Scilab* en la que debo calcular también el parámetro de entrada $\in$, de la formula rizado=1/(1+ $\in^2$), obteniendose $\in$ = (1/0.95 – 1)$^{1/2}$, Siendo la llamada a la función en *Scilab* es:

```
[pols,gn]=zpch1(3,.22942,2);

gn
 gn  =

    8.7176358

pols'
 ans  =

! - 0.7915862 - 2.2090329i !
! - 1.5831724 - 1.562D-16i !
! - 0.7915862 + 2.2090329i !
```

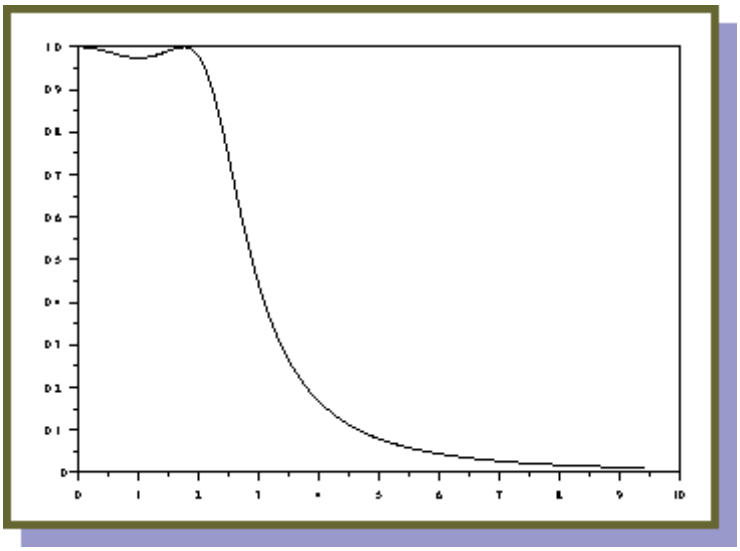
```
hs=gn/real(poly(pols,'s'))
 hs  =

                  8.7176358
 ---------------------------------------
                                      2   3
 8.7176358 + 8.0128698s + 3.1663448s + s
```

Como podemos ver la función de transferencia **hs** se ha calculado a partir de la ganancia y los polos de retorno de la función **zpch1**.

Calculamos La *magnitud de respuesta* de esta FT como sigue:

```
 gn  =

    8.7176358
 ans  =

! - 0.7915862 - 2.2090329i !
! - 1.5831724 - 1.562D-16i !
! - 0.7915862 + 2.2090329i !
 hs  =


                  8.7176358
 ---------------------------------------
                                      2   3
 8.7176358 + 8.0128698s + 3.1663448s + s

fr=0:.05:3*%pi;

hsm=abs(freq(hs(2),hs(3),%i*fr));

plot(fr,hsm)
```

Siendo la *magnitud del filtro analógico* la siguiente (representada gráficamente) :

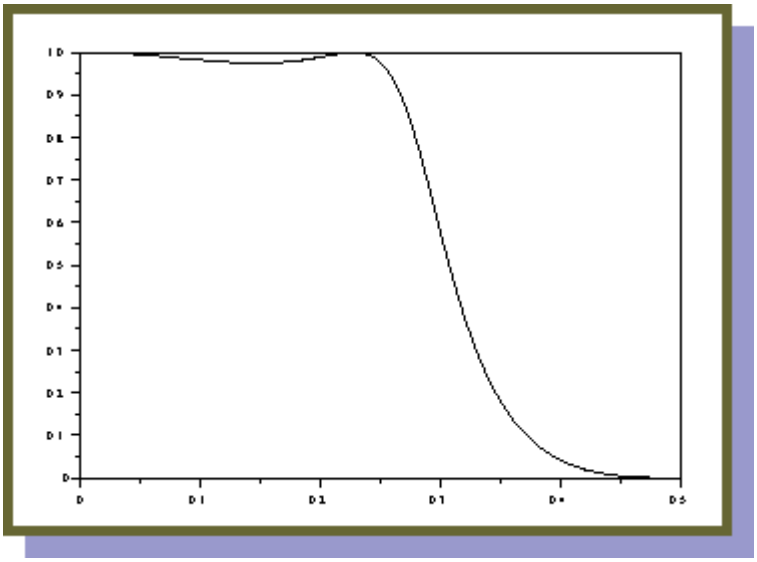


Ahora el diseño de *filtro analógico pasa bajo* puede ser transformado a *digital* usando la transformación bilineal como sigue :

```
 gn  =

    8.7176358
 ans  =

! - 0.7915862 - 2.2090329i !
! - 1.5831724 - 1.562D-16i !
! - 0.7915862 + 2.2090329i !
 hs  =


                  8.7176358
   ---------------------------------------
                                    2   3
   8.7176358 + 8.0128698s + 3.1663448s + s

z=poly(0,'z');

hz=horner(hs,2*(z-1)/(z+1))
 hz  =


                                      2           3
  8.7176358 + 26.152907z + 26.152907z + 8.7176358z
  ----------------------------------------------
                                      2           3
- 2.6427245 + 21.461789z + 5.5132676z + 45.408755z
```

nota: la función **horner** realiza la evaluación de polinomios y racionales polinómicos.

Si representamos la magnitud de respuesta del Filtro Digital, obtenemos :



5.c.2.b Diseño de Filtros Digitales mediante transformación de un Filtro Digital pasa bajo.

En el apartado anterior hemos visto como diseñar con *Scilab* un *filtro IIR pasa bajo* mediante la transformación bilineal de un *filtro analógico pasa bajo*. Ahora vamos a ver como obtener *filtros digitales pasa alto, pasa banda , elimina banda* u otro *pasa bajo* usando transformaciones similares a la bilineal partiendo de un *filtro digital pasa bajo*.

La expresión que relaciona la frecuencia de corte del *filtro pasa bajo original* ω_c y la nueva para un **nuevo filtro digital pasa bajo** ω_u , es la siguiente transformación :

$$z \to \frac{z-\alpha}{1-z\alpha}$$

$$\alpha = \frac{\sin[(\omega_c - \omega_u)/2]}{\sin[(\omega_c + \omega_u)/2]}.$$

Para el **filtro digital pasa alto** la frecuencia de corte ω_u, uso la transformación:

$$z \to -\frac{z+\alpha}{1+z\alpha}$$

$$\alpha = -\frac{\cos[(\omega_c - \omega_u)/2]}{\cos[(\omega_c + \omega_u)/2]}.$$

Para el **filtro digital pasa banda** las frecuencias que delimitan la banda de paso ω_u y ω_l , uso la transformación:

$$z \to -\frac{z^2 - (2\alpha k/(k+1))z + ((k-1)/(k+1))}{1 - (2\alpha k/(k+1))z + ((k-1)/(k+1))z^2}$$

$$\alpha = \frac{\cos[(\omega_u + \omega_l)/2]}{\cos[(\omega_u - \omega_l)/2]}$$

$$k = \cot[(\omega_u - \omega_l)/2]\tan(\omega_c/2).$$

Para el **filtro digital elimina banda** las frecuencias que delimitan la banda de eliminada ω_u y ω_l , uso la transformación:

$$z \to -\frac{z^2 - (2\alpha k/(k+1))z + ((k-1)/(k+1))}{1 - (2\alpha k/(k+1))z + ((k-1)/(k+1))z^2}$$

$$\alpha = \frac{\cos[(\omega_u + \omega_l)/2]}{\cos[(\omega_u - \omega_l)/2]}$$

$$k = \cot[(\omega_u - \omega_l)/2]\tan(\omega_c/2).$$

5.c.3. Uso de la función **iir** de *Scilab*.

La llamada a la función sigue la siguiente sintaxis :

```
--> [hz]=iir(n,ftype,fdesign,frq,delta)
```

Argumentos de entrada :

n Orden del filtro que debe ser un entero positivo.

ftype Indica el tipo de filtro puede tomar los valores : **lp**=pasa bajo, **hp**=pasa alto, **bp**=pasa banda y **sb**=elimina banda.

fdesign Indica el tipo de técnica usada para el diseño del puede tomar los valores: **butt**=Butteworth, **cheb1**=Chebyshev tipo 1, **cheb2**= Chebyshev tipo 2 y **ellip**=elíptico.

frq Vector de dos dimensiones que contiene las frecuencias de corte y en el caso de bandas la inicial y final.

delta Vector de dos dimensiones que contiene los valores de rizado.

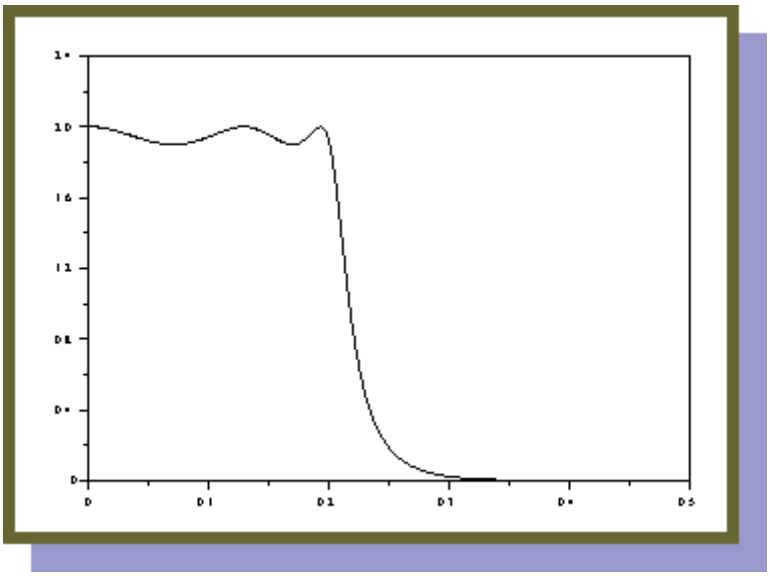
El valor de retorno **hz** es un polinomio racional que nos da los coeficientes del filtro.

5.c.4. Ejemplos de diseño usando la función **iir** .

5.c.4.a **Especificación de diseño : Filtro pasa bajo diseñado a partir de un filtro analógico Chebyshev tipo 1 con ω_c = 0.2, n = 5 y δ = rizado banda paso = 0.05.**

```
hz=iir(5,'lp','cheb1',[.2 0],[.05 0.05])

 hz  =

                                          2         3         4
   0.0207392 + 0.1036960z + 0.2073921z + 0.2073921z + 0.1036960z
                5
     + 0.0207392z
    ------------------------------------------------------------
                                          2         3         4
  - 0.2213294 + 0.9336888z - 1.9526644z + 2.5422088z - 1.9700766z
        5
     + z
```

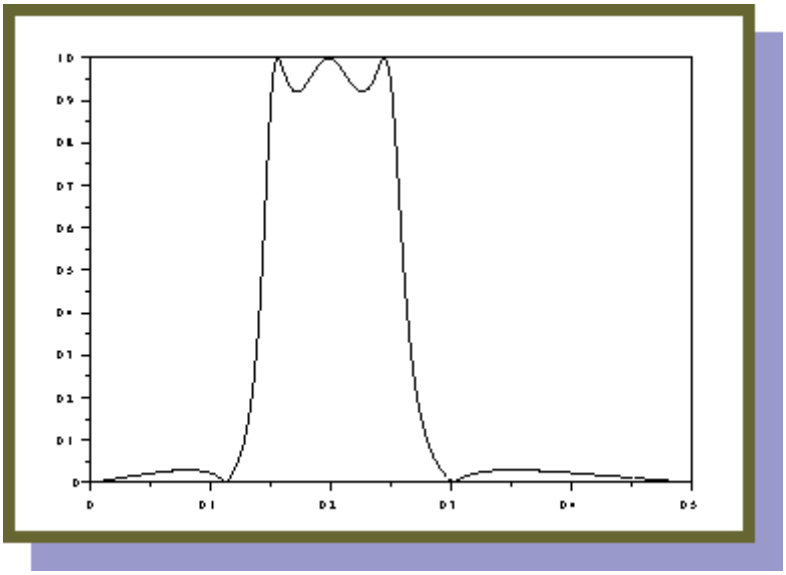
La *respuesta* del *filtro digital pasa bajo* diseñado:



5.c.4.a Especificación de diseño : Filtro pasa banda diseñado a partir de un filtro analógico Elíptico con frecuencias de corte $\omega_l = 0.15$ y $\omega_h = 0.25$, n = 3 , δ_p = rizado banda paso = 0.08 y δ_s = rizado banda rechazo = 0.03.

```
hz=iir(3,'bp','ellip',[.150.25],[.080.03])
 hz  =

                                        2            3            4
0.0944242 - 0.0840374z + 0.0026735z + 2.097D-17z - 0.0026735z
                 5            6
   + 0.0840374z - 0.0944242z
-----------------------------------------------------------------

                                       2           3           4
1.9820273 - 3.2624882z + 5.9409383z - 5.1962315z + 4.8074599z
                5    6
   - 2.0635356z + z
```

La *magnitud de respuesta* resulta la siguiente:



5.c.5. Uso de la función **eqiir** de *Scilab*.

Scilab para el diseño de filtros IIR tiene implementada también la función **eqiir** , además de la función **iir**, que contiene el código de la rutina **syredi** de Fortran. Este algoritmo permite el diseño de los cuatro tipos de filtros lp, hp, pb y sb; estando basado también en la transformación bilineal de filtros analógicos. Pero el filtro resultante se obtiene como producto de elementos de segundo orden, calculando automáticamente el orden del filtro para cumplir con las especificaciones de diseño.

El filtro puede ser dado por sus *polos* y *ceros* o mediante la siguiente *representación equivalente* :

$$H(z) = fact \prod_{i}^{N} n_i(z)/d_i(z)$$

Siendo la fracción : $n_i(z)/d_i(z)$ el i-ésimo elemento de segundo orden.

La sintaxis de llamada a la función es la siguiente :

[cells,fact,zzeros,zpoles]=eqiir(ftype,approx,om,deltap,deltas)

Parámetros de entrada :

ftype : Tipo de filtro ('lp','hp','sb','bp')

approx : Aproximación de diseño ('butt','cheb1','cheb2','ellip')

om : Un vector de 4 elementos que contiene la frecuencias de corte (en radianes)
om=[om1,om2,om3,om4], 0 <= om1 <= om2 <= om3 <= om4 <= pi.
Cuando **ftype**='lp' or 'hp', om3 y om4 no son usados podemos poner cero.

deltap : Rizado en la banda de paso. 0<= deltap <=1

deltas : Rizado en la banda de rechazo. 0<= deltas <=1

Parámetros de retorno :

cells : Realización del filtro por elementos de segundo orden

fact : Constante de normalización

zzeros : Ceros en el plano z.

zpoles : Polos en plano z.

5.c.6. Ejemplos de diseño usando la función **eqiir** .

5.c.6.a **Especificación de diseño de un Filtro IIR elíptico pasa bajo diseñado a partir de un filtro analógico elíptico pasa bajo con rizados máximos δ_p = rizado banda paso = 0.02 y δ_s = rizado banda paso = 0.001, y frecuencias de corte $\omega_1=2\pi/10$ y $\omega_2=4\pi/10$.**

```
[cells,fact,zeros,zpoles]=...
eqiir('lp','ellip',[2*%pi/10,4*%pi/10],0.02,0.001);
Warning :redefining function: zeros


zpoles'
 ans  =

!   0.6906008 - 0.5860065i !
!   0.6906008 + 0.5860065i !
!   0.6373901 - 0.3437403i !
!   0.6373901 + 0.3437403i !
!   0.6247028              !

zeros'
 ans  =

  !   0.6906008 - 0.5860065i !
  !   0.6906008 + 0.5860065i !
  !   0.6373901 - 0.3437403i !
  !   0.6373901 + 0.3437403i !
  !   0.6247028              !

  zeros'
   ans  =

  !   0.2677115 - 0.9634991i !
  !   0.2677115 + 0.9634991i !
  ! - 0.1744820 - 0.9846604i !
  ! - 0.1744820 + 0.9846604i !
  ! - 1.                     !

  cells'
   ans  =
!                          2     !
!       1 - 0.5354229z + z       !
!   ------------------------     !
!                             2  !
!   0.8203331 - 1.3812015z + z   !
!                                !
!                          2     !
!       1 + 0.3489640z + z       !
!   ------------------------     !
!                             2  !
!   0.5244235 - 1.2747803z + z   !
!                                !
!       1 + z                    !
!   ------------                 !
! - 0.6247028 + z                !

transfer=fact*poly(zeros,'z')/poly(zpoles,'z')
```

```
transfer  =

                                      2           3           4
  0.0059796 + 0.0048646z + 0.0097270z + 0.0097270z + 0.0048646z
                 5
    + 0.0059796z
  ---------------------------------------------------------------
                                      2           3           4
- 0.2687484 + 1.5359753z - 3.7100842z + 4.7646843z - 3.2806846z
       5
    + z
```

5.c.6.b **Especificación de diseño de un Filtro IIR Chebyshev tipo 1 pasa bajo diseñado a partir de un filtro analógico Chebyshev tipo 1 pasa bajo con un rizado en la banda de paso de 2 dB y en la banda de rechazo de –30 dB de atenuación y las frecuencias de corte en 37.5 Hz y 75 Hz respectivamente, siendo la frecuencia de muestreo de 3000Hz.**

```
sf=3000;

f1=37.5;

f2=75;

as=30;

ap=2;

om=[f1*(2*%pi)/sf,f2*(2*%pi)/sf];


deltas=10.00**(-0.05*as);

deltap=(1.0-10.0**(-0.05*ap));

[cells,fact,zers,pols]=...
eqiir('lp','ch1',om,deltap,deltas);

cells
 cells  =
```

```
!                          2                                2     !
!       1 - 1.9711824z + z               1 - 1.8376851z + z      !
!   ----------------------------    ----------------------------  !
!                              2                                2 !
!   0.9995251 - 1.9942623z + z      0.9988526 - 1.9979487z + z    !
```

5.c.6.c **Especificación de diseño de un Filtro IIR Butterworh pasa bajo diseñado a partir de un filtro analógico Butterworth pasa bajo con una atenuación en la banda de paso de -5 dB en la frecuencia normalizada 0.25π/10 y en la banda de rechazo de –120 dB de atenuación en la frecuencia normalizada 4π/10.**

```
om=[.25*%pi/10,4*%pi/10];

pdB=5;

sdB=120;

deltap=(1.0-10.0**(-0.05*pdB));

deltas=10.00**(-0.05*sdB);

[cells,fact,zers,pols]=eqiir('lp','butt',om,deltap,deltas);

cells
 cells  =


          column 1 to 2


!                   2                                2         !
!          1 + 2z + z                       1 + 2z + z          !
!   ----------------------------    ----------------------------  !
!                              2                                2 !
!   0.9450100 - 1.9368524z + z      0.8621663 - 1.8543562z + z    !

         column 3

!       1 + z      !
!   -------------  !
! - 0.9123352 + z  !
```

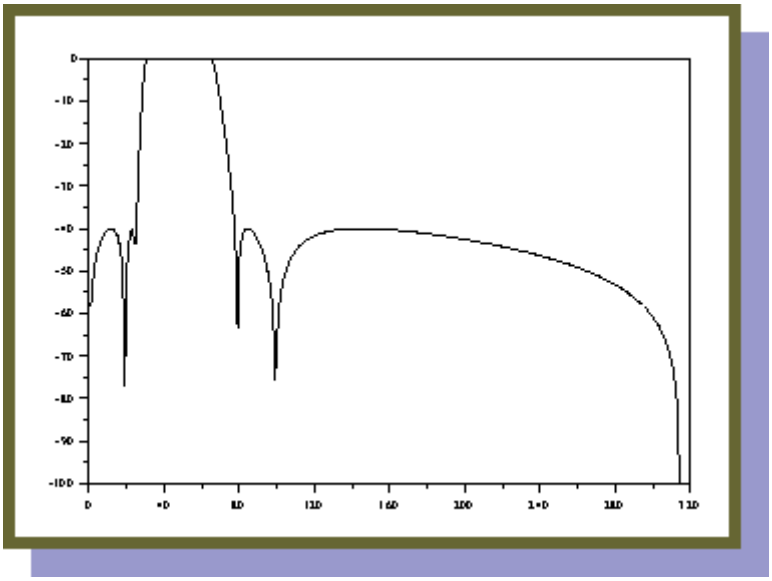
```
n=prod(calls(2));

d=prod(calls(3));

tr=n./d
 tr  =
```

$$\frac{1 + 5z + 10z^2 + 10z^3 + 5z^4 + z^5}{-0.7433304 + 3.937017z - 8.3477808z^2 + 8.8576437z^3 - 4.7035438z^4 + z^5}$$

5.c.6.d **Especificación de diseño de un Filtro IIR elíptico pasa banda simétrico diseñado a partir de un filtro analógico elíptico pasa banda simétrico con rizados máximos δ_p = rizado banda paso = 0.022763 y δ_s = rizado banda paso = 0.01, y frecuencias de corte ω_1= 0.251463 , $\omega_2=\pi/10$, $\omega_3=2\pi/10$ y ω_4=0.773302.**

```
om=[0.251463,1*%pi/10,2*%pi/10,0.773302];

deltap=0.022763;

deltas=0.01;

[calls,fact,zers,pols]=eqiir('bp','el',om,deltap,deltas);

n=prod(calls(2));d=prod(calls(3));



rep=freq(n,d,exp(%i*(0:0.01:%pi)));

rep=fact*abs(rep);

n=prod(size(rep))
 n  =

    315.

plot(20*log(rep(2:n))/log(10))
```

Siendo la *magnitud de respuesta* de este filtro la siguiente :



6. ESTIMACIÓN ESPECTRAL CON SCILAB .

6.1. Estimación del Espectro de Potencia.

El **espectro de potencia** de una señal discreta en el tiempo, de longitud finita y determinista, x(n), puede ser definida la *transformada de Fourier* de la *magnitud cuadrada* de la señal :

$$S_x(\omega) = \frac{1}{N}\left| \sum_{n=0}^{N-1} x(n)e^{-j\omega n}\right|^2.$$

De una forma análoga podemos definir el **espectro cruzado** de dos señales x(n) e y(n) como:

$$S_{xy}(\omega) = \frac{1}{N}\left(\sum_{n=0}^{N-1} x(n)e^{-j\omega n}\right)\left(\sum_{n=0}^{N-1} y(n)e^{-j\omega n}\right)^*.$$

La **función de autocorrelación** R_x y la **función de correlación cruzada** Rxy de x con y tenemos que son por definición :

$$\begin{aligned} R_x(m) &= E\{x(n+m)x^*(n)\} \\ R_{xy}(m) &= E\{x(n+m)y^*(n)\}. \end{aligned}$$

Consecuentemente el **espectro de potencia de x(n)** y el **espectro de potencia cruzada de x(n) con y(n)** se pueden expresar :

$$S_x(\omega) = \sum_{m=-\infty}^{\infty} R_x(m)e^{-j\omega m}$$
$$S_{xy}(\omega) = \sum_{m=-\infty}^{\infty} R_{xy}(m)e^{-j\omega m}.$$

Estas expresiones requieren la estimación de las *funciones de correlación*. Las funciones que estiman las *funciones de autocorrelación* y *correlación cruzada* de señales aleatorias de longitud finita (esto es x(n)≠0 e y≠0 para n=0,1,....,N-1) son :

$$\hat{R}_x(m) = \frac{1}{N}\sum_{n=0}^{N-1-m} x(n+m)x^*(n)$$
$$\hat{R}_{xy}(m) = \frac{1}{N}\sum_{n=0}^{N-1-m} x(n+m)y^*(n).$$

Las estimaciones en las formulas anteriores son imparciales, son *estimadores* consistentes cuando N $\rightarrow \infty$. Pero en la práctica los *espectros de potencia estimados*, en el límite, no son consistentes. Ya que existe una *varianza de error* debido a que no decrece aunque se aumente el número datos. Por lo cual, el *espectro de potencia estimado* obtenido tomando la transformada de Fourier a la magnitud cuadrada de la señal tiene una alta varianza y una baja calidad de estimación.

Para mejorar esta estimación existen dos técnicas o métodos basadas en la media del espectro estimado de una forma clásica. Ambas técnicas utilizan ventanas para disminuir los efectos de que estemos utilizando datos finitos sobre la estimación espectral. Estos efectos son idénticos a los encontrados en el diseño de filtros FIR desde el punto de vista de la elección de la ventana más adecuada.

6.1.a. Estimación del Espectro de Potencia por el Método del Periodograma Modificado.

El **periodograma** de una secuencia finita de datos se define como :

$$I(\omega) = \frac{1}{U}\left|\sum_{n=0}^{N-1} w(n)x(n)e^{-j\omega n}\right|^2.$$

Donde ω(n) es una función ventana la cual tiene una . Por lo que si subdividimos en K segmentos de longitud N, podremos usar todos para calcular **el espectro de la señal** mediante **el espectro del periodograma modificado estimado** , que es justo la media de los K periodogramas :

$$\hat{S}_x(\omega) = \frac{1}{K}\sum_{k=0}^{K-1} I_k$$

En *Scilab* se utiliza la función **pspect** para calcular **una estimación del espectro de potencia** mediante **el método del periodograma modificado**. Vamos a mostrar ***un ejemplo*** de uso de esta función, los datos utilizados son obtenidos al hacer pasar un ruido blanco de media cero a través de **un filtro FIR pasa bajo de longitud 33 generado usando** la función **eqfir**.

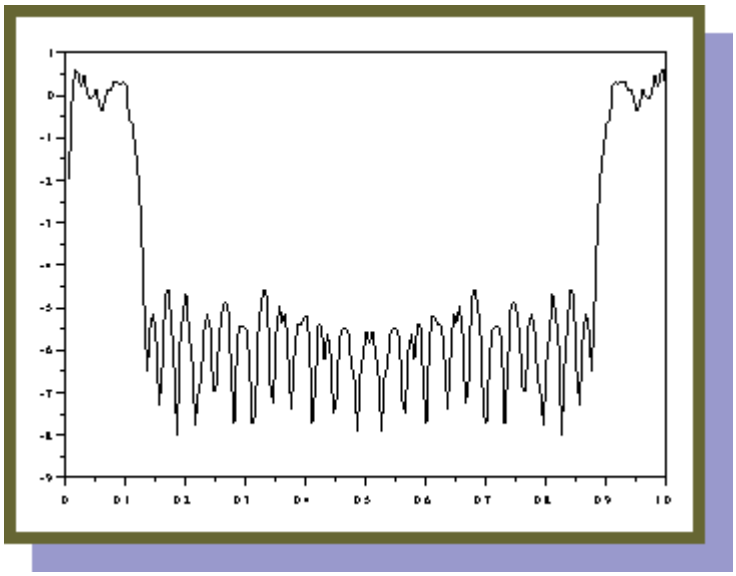
La generación de los 1024 datos para mi ejemplo es la siguiente :

```
rand('normal');

rand('seed',0);

x=rand(1:1024-33+1);

//make low-pass filter with eqfir

nf=33;

bedge=[00.1;.1250.5];

des=[1 0];

wate=[1 1];

hn=eqfir(nf,bedge,des,wate);

//filter white data to obtain colored data

h1=[hn 0*ones(1:maxi(size(x))-1)];

x1=[x 0*ones(1:maxi(size(hn))-1)];

hf=fft(h1,-1);

xf=fft(x1,-1);

yf=hf.*xf;

y=real(fft(yf,1));
```

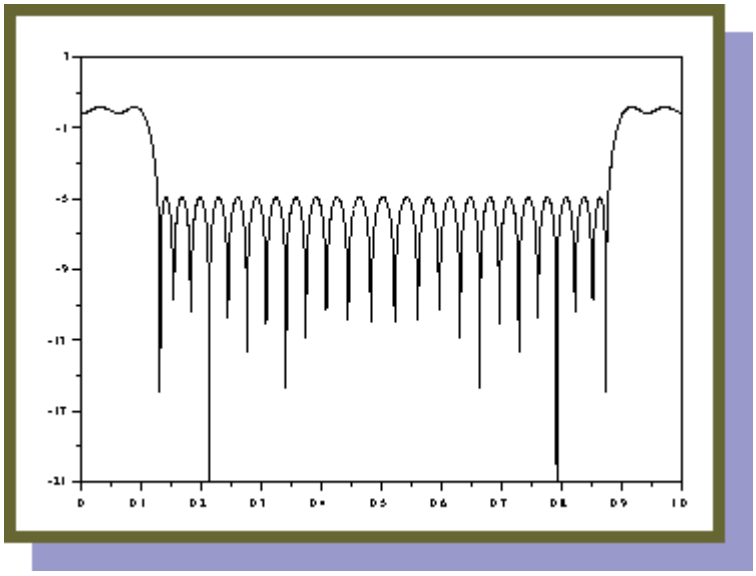
Una vez obtenidos los datos como se expone arriba **estimamos el espectro de potencia** de la siguiente forma :

```
-->[sm]=pspect(100,200,'tr',y);
```

La representación logarítmica del espectro de potencia estimado **sm** (valor de retorno de la función **pspect**) es la siguiente :



El **espectro teórico (magnitud cuadrada) del filtro** se representa a continuación en un gráfico logarítmico :



Como se puede observar el espectro estimado se ajusta muy bien al teórico, sobre todo en los picos.

6.1.b. Estimación del Espectro de Potencia por el Método de la

ectro de potencia mediante **la transformada de Fourier de la estimación modificada de la función de** . La estimación modificada de la consiste en calcular estimaciones de forma repetitiva de *la función de autocorrelación* de segmentos de puntos obtenidos de los datos totales y la media de estas estimaciones me da la estimación modificada.

Si dividimos los datos en N segmentos que contengan cada uno $x_k(n)$ datos , se calcula su *función de autocorrelación* por :

$$\hat{R}_k(m) = \sum_{n=0}^{N-1-m} x(n+m)x^*(n)$$

m = 0,±1,.....,±M . Para K estimaciones calculadas de la *función de autocorrelación* ,**el espectro de potencia estimado** se calcula mediante la transformada de Fourier del producto de una función ventana y la media de las K funciones de autocorrelación calculadas :

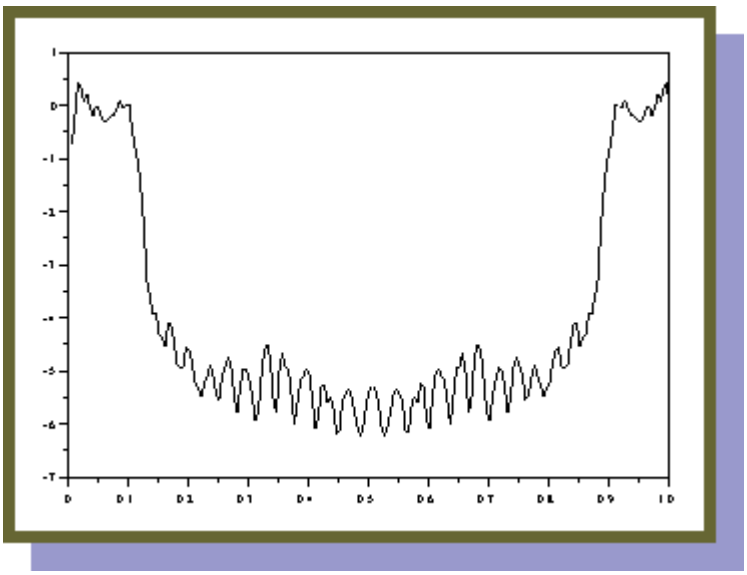
$$\hat{S}_x(\omega) = \mathcal{F}\{\hat{R}_x(m)w(m)\}$$

El cálculo de la *estimación espectral* mediante el método de correlación está basado en la función básica **corr** de *Scilab*, pero la función que cálcula este método es la **cspect**.

Vamos a ver ***un ejemplo*** de uso de esta función tomando los 1024 puntos calculados en el apartado anterior :

```
-->[sm]=cspect(100,200,'tr',y);
```

Si representamos en un gráfico logarítmico **sm**, tenemos :



Como podemos ver existen más discrepancias respecto al teórico que en el anterior método.

6.2. Estimación del Espectro de Potencia mediante el Método de la

El espectro de potencia de una **señal determinista** viene definido por la transformada de Fourier de la magnitud cuadrada de la señal. Esto es, para una señal discreta x(n), **el espectro de potencia Sx(ω)**, viene dado por :

$$S_x(\omega) = |X(\omega)|^2$$

$$X(\omega) = \sum_{n=-\infty}^{\infty} x_n e^{-j\omega n}.$$

En muchas aplicaciones es muy útil conocer el *espectro de potencia* de la señal, pero lo raro es que la señal obtenida pueda ser considerada una señal determinista. A menudo la señal se encuentra presente en un entorno con ruido que me degrada la medida de la señal. Luego necesito un método para estimar **el espectro de potencia de una señal no determinista** y este es el **Método de la Máxima Entropía Espectral Estimada (MESE)**.

Este método consiste a grandes rasgos en que si tenemos una señal estacionaria x(n) de media cero y $r_x(n)$ la *función de autocorrelación* de la señal , entonces el *espectro de potencia estimado* de x(n) :

$$\hat{r}_x(n) = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{m=-N}^{N} x(m)x(m-n).$$

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \hat{S}_x(\omega)e^{j\omega n}d\omega = \hat{r}_x(n), \quad n = 0, 1, \ldots, N-1.$$

$$\hat{S}_x(\omega) = \max_{S(\omega)}\{-\int_{-\pi}^{\pi} S(\omega)\,\mathbf{log}[S(\omega)]d\omega\}$$

Este método se calcula en *Scilab* mediante la función **mese**, cuya sintaxis es:

```
-->[sm,fr]=mese(x)
```

Scilab tiene implementada la función **lev** , que es otra solución a las ecuaciones del método anterior mediante un algoritmo recursivo más eficiente conocido como **algoritmo de Levison.** Su sintaxis es :

```
-->[ar,sigma2,rc]=lev(r)
```

El tratamiento de señales con *Scilab* no se acaba aquí. El usuario que quiera seguir ampliando se **Técnicas de Suavizado y Filtrado Óptimo** (ver funciones **optgain**, **ltitr**, **sskf** – filtro Kalman -- , **kalm**, **srkf**, **wiener**, etc), **Diseño de Filtros Optimizado** (ver funciones **iirlp**, **optiir**, **optim**, **optfir1**, **optfir2**, etc), **Realización Estocástica** (ver funciones **sfact**, **phcsrfaur**, **lindquist**, **levin** , procesos **ARMA**, etc.), **Representación de Señales en tiempo-frecuencia** (ver funciones **wigner**, etc.).

7. RESUMEN DE FUNCIONES PARA EL TRATAMIENTO DE SEÑALES EN *SCILAB* Y *MATLAB*.

7.1. Funciones *Scilab*.

Se ha respetado la clasificación que *hace Scilab* en la ayuda.

FILTROS

analpf : analog low-pass filter

buttmag : squared magnitude response of a Butterworth filter

casc : creates cascade realization of filter

cheb1mag : square magnitude response of a type 1 Chebyshev filter

cheb2mag : square magnitude response of a type 1 Chebyshev filter

chepol : recursive implementation of Chebychev polynomial

convol : convolution of 2 discrete series

ell1 mag : squared magnitude of an elliptic filter

eqfir : minimax multi-band, linear phase, FIR filter

eqiir : design of iir filter

faurre : optimal lqg filter.

lindquis : optimal lqg filter lindquist algorithm

ffilt : FIR low-pass,high-pass, band-pass, or stop-band filter

filter : compute the filter model

find_freq : parameter compatibility for elliptic filter design

findm : for elliptic filter design

frmag : magnitude of the frequency responses of FIR and IIR filters.

fsfirlin : design of FIR, linear phase (frequency sampling technique)

fwiir : optimum design of IIR filters in cascade realization,

iir : designs an iir digital filter using analog filter designs.

iirgroup : group delay of iir filter

iirlp : Lp IIR filters optimization

group : calculate the group delay of a digital filter

optfir : optimal design of linear phase filters using linear programming

remezb : minimax approximation of a frequency domain magnitude response

.

kalm : Kalman update and error variance

lev : resolve the Yule-Walker equations

levin : solve recursively Toeplitz system (normal equations)

srfaur : square-root algorithm for the algebraic Riccati equation.

srkf : square-root Kalman filter algorithm

sskf : steady-state Kalman filter

system : generates the next observation given the old state

trans : transformation of standardized low-pass filter into low- pass, high-pass, band-pass, stop-band.

wfir : linear-phase windowed FIR low-pass, band-pass, high-pass, stop-band

wiener : Wiener estimate (forward-backward Kalman filter formulation)

wigner : time-frequency wigner spectrum of a signal.

window : calculate symmetric window

zpbutt : Butterworth analog filter

zpch1 : poles of a type 1 Chebyshev analog filter

zpch2 : poles and zeros of a type 2 Chebyshev analog filter

zpell : poles and zeros of prototype lowpass elliptic filter

ESTIMACIÓN ESPECTRAL

corr : correlation coefficients

cspect : spectral estimation using the modified periodogram method.

czt : chirp z-transform algorithm

intdec : change the sampling rate of a 1D or 2D signal

mese : calculate the maximum entropy spectral estimate

pspect : auto and cross-spectral estimate

wigner : Wigner-Ville time/frequency spectral estimation

TRANSFORMADAS

dft : discrete Fourier transform

fft : fast flourier transform

hilb : Hilbert transform centred around the origin.

hank : hankel matrix of the covariance sequence of a vector process

mfft : fft for a multi-dimensional signal

IDENTIFICACIÓN

lattn,lattp : recursive solution of normal equations

phc : State space realisation by the principal hankel component approximation method,

rpem : identification by the recursive prediction error method

VARIOS

lgfft : computes p = ceil (log_2(x))

sinc : calculate the function sin(2*pi*fl*t)/(pi*t)

sincd : calculates the function Sin(N*x)/Sin(x)

%k : Jacobi's complete elliptic integral

%asn : .TP the elliptic integral :

%sn : Jacobi 's elliptic function with parameter m

bilt : bilinear transform or biquadratic transform.

jmat : permutes block rows or block columns of a matrix

7.2. Funciones *Matlab*.

Las funciones de *Matlab* mostradas a continuación corresponden a la **Signal Processing Toolbox** Version 2.0b 13-Nov-92, respetándose la clasificación de las funciones que hace *Matlab* en la ayuda.

PROTOTIPOS DE FILTROS ANALÓGICOS PASA BAJO

buttap - Butterworth filter prototype.

cheb1ap - Chebyshev type I filter prototype (passband ripple).

cheb2ap - Chebyshev ty[pe II filter prototype (stopband ripple).

ellipap - Elliptic filter prototype.

DISEÑO DE FILTROS DIGITALES IIR

butter - Butterworth filter design**.**

cheby1 - Chebyshev type I filter design.

cheby2 - Chebyshev type II filter design.

ellip - Elliptic filter design.

yulewalk - Yule-Walker filter design.

DISEÑO DE FILTROS FIR

fir1 - Window based FIR filter design - low, high, band, stop.

fir2 - Window based FIR filter design - arbitrary response.

remez - Parks-McClellan optimal FIR filter design.

SELECCIÓN DEL ORDEN DEL FILTRO

buttord - Butterworth filter order selection.

cheb1ord - Chebyshev type I filter order selection.

cheb2ord - Chebyshev type II filter order selection.

ellipord - Elliptic filter order selection.

TRANSFORMACIONES DE FILTROS

bilinear - Bilinear transformation with optional prewarping.

lp2bp - Lowpass to bandpass analog filter transformation.

lp2bs - Lowpass to bandstop analog filter transformation.

lp2hp - Lowpass to highpass analog filter transformation.

lp2lp - Lowpass to lowpass analog filter transformation.

ss2zp - State-space to zero-pole conversion.

tf2ss - Transfer function to state-space conversion.

zp2ss - Zero-pole to state-space conversion.

ANÁLISIS DE FILTROS

abs - Magnitude.

angle - Phase angle.

filter - Filter implementation.

filtfilt - Zero-phase version of filter.

fftfilt - Overlap-add filter implementation.

filtic - Determine filter initial conditions.

freqs - Laplace transform frequency response.

freqz - Z-transform frequency response.

grpdelay - Group delay.

unwrap - Unwrap phase.

MODELADO PARAMÉTRICO

invfreqs - Analog filter fit to frequency response.

invfreqz - Discrete filter fit to frequency response.

prony - Prony's discrete filter fit to time response.

ident - See also the Identification Toolbox.

ANÁLISIS ESPECTRAL

cceps - Complex cepstrum.

czt - Chirp-z transform.

detrend - Linear trend removal.

fft - Fast Fourier transform.

fftshift - Swap vector halves.

hilbert - Hilbert transform.

ifft - Inverse fast Fourier transform.

czt - Chirp-z transform.

rceps - Real cepstrum and minimum pahse reconstruction.

specplot - Plot output of spectrum function.

specgram - Spectrogram, for speech signals.

spectrum - Power spectrum estimate, Welch method.

CORRELACIÓN Y CONVOLUCIÓN

conv - Convolution.

corrcoef - Correlation coefficients.

cov - Covariance matrix.

deconv - Deconvolution.

xcorr - Cross-correlation function.

xcov - Covariance function.

SEÑALES BIDIMENSIONALES Y PROCESAMIENTO DE IMAGENES

conv2 - 2-D convolution.

fft2 - 2-D fast Fourier transform.

fftshift - Swap quadrants of array.

ifft2 - Inverse 2-D fast Fourier transform.

xcorr2 - 2-D cross-correlation.

VENTANAS

bartlett - Bartlett window.

blackman - Blackman window.

boxcar - Rectangular window.

chebwin - Chebyshev window.

hamming - Hamming window.

hanning - Hanning window.

kaiser - Kaiser window.

triang - Triangular window.

INTERPOLACIÓN Y MUESTREO

decimate - Resample data at a lower sample rate.

interp - Resample data at a higher sample rate.

interp1 - General 1-D interpolation.

OTROS

stem - Plot discrete data sequence.

convmtx - Convolution matrix.

cplxpair - Order vector into complex conjugate pairs.

dftmtx - Discrete Fourier transform matrix.

mpoles - Identify repeated poles and their multiplicities.

polystab - Polynomial stabilization.

residuez - Z-transform partial fraction expansion.

sawtooth - Sawtooth function.

square - Square wave function.

El resumen anterior se ha incluido para que el usuario de *Matlab* pueda buscar funciones equivalentes en *Scilab*. Muchas veces aunque coincida el nombre de la función o sea casi igual, la sintaxis de llamada es diferente.

8. CONVERSIÓN DE FICHEROS PROGRAMADOS CON MATLAB A SCILAB

El usuario que tenga desarrollado mucho código en *Matlab* y quiera empezar a desarrollar cosas en *Scilab*, puede echarse para atrás, al pensar que todo su trabajo anterior no le vale. Pero la versión 2.4 de Scilab incorpora una utilidad que transforma **ficheros m de *Matlab*** a **ficheros sci de *Scilab***, también hay que decir que esta conversión no está del todo conseguida (está en versión beta) y a veces necesitaré de retoques a mano, aunque el gran grueso de código lo realiza sin problemas.

Lo que es imposible es hacer llamadas de *funciones Matlab desde Scilab*, primero hay que traducirlas con la función **mfile2sci** cuya sintaxis de llamada es la siguiente:

mfile2sci (M_file_path [, Result_path [, Imode [, Recmode]]])

M_file_path : Cadena que da el path del directorio del m-file a traducir, incluyendo el nombre de fichero (ej. c:\matlab\uned\ejemplo.m).

Result_path : Cadena que da el directorio donde queremos guardar los ficheros de salida, por defecto los guardará en el directorio actual si no pongo nada.

Imode ; Representa un chequeo Booleano de la tradución. Si es true la función mfile2sci preguntará por el tipo de variable a traducir y su tamaño cuando Scilab no pueda deducirlo. El valor por defecto es %f.

Recmode: Representa un chequeo Booleano que usa la función **translatepaths**. Debe ser %f en el caso que vaya a traducir un único fichero m.

El resultado de la ejecución de esta función producirá la generación de tres ficheros en el directorio elegido de la siguiente forma :

<nombre_función_matlab>.sci : que será el equivalente al fichero m de Matlab en el entorno Scilab.

fichero de ayuda asociado al fichero m para Scilab.

Sci_<nombre_función_matlab>.sci : función Scilab neceraria para traducir las llamadas de este fichero m de Matlab a otros ficheros m. Esta función puede ser mejorada a mano.

Algunas funciones de *Matlab* como **eye, ones, size, sum,** etc... se comportan diferente forma según sea la dimensión de sus argumentos. Cuando la función **mfile2sci** no puede deducir la dimensiones , lo que hace es sustituir la función de llamada correspondiente dentro de el fichero m que estoy traduciendo, por una función **mltb_<nombre_función>**. Para obtener más rendimiento en mi fichero sci obtenido podría sustituir estas funciones de emulación por su instrucción equivalente en *Scilab* si la conozco.

Existen otras funciones como **plot** que no tienen una traducción sencilla en *Scilab* y que también son sustituidas por una función de emulación **mltb_<nombre_función>**.

Algunas de las funciones básicas de Matlab 4 ya han sido traducidas para el usuario por Scilab y se encuentran en el directorio **scilab-2.4\macros\m2sci**. Entre las funciones que todavía no están traducidas se encuentran:

graphics

axes.m	clc.m	contourc.m	drawnow.m	fill.m
patch.m	fill3.m	get.m	getframe.m	image.m
movie.m	plot3.m	rbbox.m	semilogx.m	
semilogy.m	set.m	surface.m	eeset.m	

gui

uicontrol.m	uigetfile.m	uimenu.m	uiputfile.m

debug

dbclear.m	dbcont.m	dbdown.m	dbquit.m	dbstack.m
dbstatus.m	dbstep.m	dbstop.m	dbtype.m	dbup.m

c files

fclose.m	feof.m	ferror.m	fopen.m	fread.m
fseek.m	ftell.m	fwrite.m		

misc

global.m	home.m	isglobal.m	lasterr.m	which.m
sparsfun.m	computer.m	version.m	dos.m	echo.m
flops.m	type.m	what.m		

Existe otra función la **translatepaths()** que traduce un grupo de *ficheros m* a la vez. Se aconseja a los usuarios que estén interesados en la traducción de ficheros m consulten la *ayuda on-line* de *Scilab*.

9. ALGUNAS VALORACIONES DE LOS USUARIOS DE *MATLAB* Y *SCILAB* DEL GRUPO DE NEWS.

A continuación se van a mostrar algunos *e-mails* obtenidos en el foro de opinión **comp.soft-sys.math.scilab** que le pueden ser de utilidad al usuario de *Matlab*. Quiera reseñar que el usuario de *Matlab* que busque en *Scilab* un producto muy profesionalizado, puede que le encuentre deficiencias tales como velocidad de calculo, uso de recursos de mi ordenador, etc con respecto a otros softwares comerciales. Pero le recordamos que *Scilab* es gratuito y el tenerlo le abre las puertas de una herramienta d e gran potencialidad,

pudiendola mejorar entre todos sus usuarios en este foro de opinión.

- **En este e-mail se comenta el tema la velocidad de cálculo de *Scilab* y *Matlab* . Este usuario explica que no cree que *Scilab* sea tan lento como dicen.**

Subject: Re: scilab-matlab-compatibility
From: harmonic@omatrix.com (Beau Paisley)
Date: Thu, 30 Jul 1998 21:57:42 -0700
Newsgroups: comp.soft-sys.math.scilab
Organization: Harmonic Software
References: <6pmjdh$r8s@majestix.uni-muenster.de>

In article <6pmjdh$r8s@majestix.uni-muenster.de>, wuebbel@uni-muenster.de says...First of all, I think scilab is a great piece of software.
*After some problems, I actually succeeded to get the code to work.However, it was *most* disappointing that scilab was slower by a factor of 4 to 5 compared to matlab. I didn't see the reason for that, might be that sparse matrices are handled in an inefficient way.*
If you are concerned about performance you might check out O-Matrix,
It's actually 4-6 times faster than Matlab.

A free version is available at
http://www.omatrix.com

- **El siguiente usuario nos comenta problemas que le surgieron a él (como usuario de *Matlab*) al usar por primera vez *Scilab*, da consejos y comenta código equivalente.**

Subject: scilab-matlab-compatibility
From: "Frank Wuebbeling" <wuebbel@uni-muenster.de>
Date: 29 Jul 1998 07:36:17 GMT
Newsgroups: comp.soft-sys.math.scilab
Organization: Westfaelische Wilhelms-Universitaet Muenster, Germany
Sender: "Frank Wuebbeling" <wuebbel@uni-muenster.de>
User-Agent: tin/pre-1.4-980514 (UNIX) (Linux/2.0.34 (i586))

Hi,

just for the sake of curiousity, I used a FiniteElement-package developped under MATLAB under SCILAB. Maybe someone's interested in my experiences,so I thought I could share them in the newsgroup (which I am not subscribed to and will be able to read only sometimes):

I tried scilab2.4 on a LINUX box with Pentium processor. Installation was without problems, everything worked right out of the tarfile(and on a Windows95-PC as well). I read about the mfile2sci-interface,so I gave it a try on a finite element-solver with about 500 lines of matlab code distributed in various files. I didn't write that one,and I didn't understand it,so this is really a test if automatic conversion works.

First of all, I think scilab is a great piece of software.After some problems, I actually succeeded to get the code to work.

*However, it was *most* disappointing that scilab was slower by a factor of 4 to 5 compared to matlab. I didn't see the reason for that, might be that sparse matrices are handled in an inefficient way.*

Note: I only read the FAQ and browsed the manual, so many things I note here may actually be mentioned somewhere. Take it as an example of typical problems a matlab-user will have when switching to scilab and which might be compiled in a scilab-primer for matlab-users. If you want people to switch from matlab to scilab (and I would like our users to because our licenses are limited), be prepared that users won't put much work into the transition, either it works more or less out of the box,or (our) users just won't try it. In particular, I won't get them to read a manual, as they're already familiar with matlab, unless they are desperately looking for something they can't find there, or getting an advantage (which would most notably be a speed-advantage).

Here are the difficulties I found along the way, in the order they occured:

*-> (this *is* in the FAQ):*
"load" doesn't work on binary files. On automatical translation,mtlb_load and mtlb_save are substituted, but these functions are never declared. I couldn't get the files in contrib/matlab_interface to run correctly, so I had to work around that ("load" the files in matlab and "save" them in ascii format). When saving in ascii,it turned out that scilab had problems with lines longer than some fixed length. I didn't see this mentioned anywhere: I had a 256 by 256-matrix which I saved in matlab and which produced lines of several thousand bytes, this could not be read into scilab (I got "; expected"). Smallmatrices were no problem.

-> It would be nice to have a macro that translates all .m-files in a directory into .sci-files. Without knowledge of the sci-language, I had to translate all mfiles one after the other (using mfile2sci).

-> Also, it would be nice to have a way of simulating the way matlab reads unknown functions (whenever it encounters an unknown function, it looks for a file defining it). If this is not possible, something like "getf all .sci-files in this directory" would be helpful.

-> The precedence of . related to * is different in matlab and scilab. While in matlab the construction*

```
 vol = geometrie(:,1).*a/2*ones(1,size(phi_x,2));
A = phi_x'*vol.*phi_x+phi_y'*vol.*phi_y;
```

worked, I had to add parentheses to have it do the same thing in scilab:

```
vol = (geometrie(:,1).*a/2)*ones(1,size(phi_x,2));
A = phi_x'*(vol.*phi_x)+phi_y'*(vol.*phi_y);
```

This is a bad thing. While this will produce an error if the dimensions don't match, this is never going to be noted by anybody if thedon't match, this is never going to be noted by anybody if the dimensions match and just going to produce wrong results.

-> mtlb_mean delivers a result that sums over rows rather than columns, which is what the matlab-mean does:

```
<MATLAB>
 >> a=[1 2 3;4 5 6];
 >> mean(a)
```

```
ans =
2.5000    3.5000    4.5000

<SCILAB>
-->a=[1 2 3;4 5 6];
-->mtlb_mean(a)
ans  =
!   3.  !
!   7.5 !
```

Hmmmm... Now that I come to think of it, this is not even the mean of the rows, it is garbage.

To correct, just change an "r" to a "c" in the last line of the defintion file of mtlb_mean.

-> This is a bad one, although it seems only cosmetic. I found no way to work around it: fprintf and its friends always add a '\n' to all strings written out. It would be nice to have a compile option for that.
Reason: When carrying out an iteration, it's common practice to spit out a single dot '.' for each iteration step so you can easily see how many iterations have been done. Adding a '\n' the screen willsoon be filled.
Also, the syntax of fprintf in matlab is quite strange.
It actually allows you to leave out the file and write something like

```
fprintf('Hello, world\n')
```

*I know this is *bad code*, but if it works, people tend to use it. I don't want you to actually support this, but you might take note of it in the differences-FAQ.*

Last, in matlab files (units) are numbers with 0,1,2 assigned tostdin, stdout, stderr. in Scilab, fprintf(1,"hello") will go to nirwana. Oops, wrong, the output goes to the window from which scilab wasstarted... Hmmm, maybe that could be changed to output into the scilab-window?

-> This is one that I found most annoying: I found no obvious way of getting rid of the message. Normally, our matlab-codes run for hours unattended, and I just want to have the output scroll over my screen so I can have a glance from time to time. Again, I presume that there is a way to get that beast switched off, but at least it was not in the FAQ or obvious glancing at the manuals.

Maybe something of this can be worked into a future version of scilab.

Thanks for listening,

"Dr. Frank Wuebbeling (frank.wuebbeling@math.uni-muenster.de)"
(http://www.uni-muenster.de/math/users/wuebbel)
Institut fuer Numerische Mathematik der Universität Münster
FON (+49) 251 - 83 33795, FAX (+49) 251 - 83 32714
Einsteinstr. 62, D-48149 Muenster, Germany

- **El siguiente usuario pregunta por las posibilidades de Scilab en el tratamiento de imágenes y de señales, obteniendo la siguiente respuesta:**

Subject: Re: Does SciLab fit my needs ?
From: scilab@inria.fr (Dr Scilab)
Date: 3 Aug 1998 08:20:22 GMT
Distribution: world
Newsgroups: comp.soft-sys.math.scilab
Organization: Inria
References: <u6ogu7mpyb.fsf@HPCAL.crpp.u-bordeaux.fr>
Reply-To: scilab@inria.fr (Dr Scilab)
Sender: scilab@velo.inria.fr (Dr Scilab)

Hi !

I'm working on signal and image analysis. For this I use a software I'm writing interfacing C and Tcl-Tk. This software is not very stable and always evolve in a chaotic way. In particular the user interface can be highly improved. But since the UI and the core are strongly separated, I want to see if I can use the core with an otherUI.

I've heard of SciLab and I wonder if I will be happy with sucha tool.

Here is what I want :

- Good scripting language (like Tcl)
- Easy interfacing with C.
- Good image and signal display + printing capabillities.
- User defined display of data. For exemple I want to add several vectors (gradient) or lines on an image display window. Or I want to display sevaeral graphs in a given window.
- User defined binding on display window. For example I want to execute a computation on the image when I use the key <c> and then display the resulting signal in an other window.
- Background computation with log messages (for very long analysis).
- Good strong coffe making.

What are your opinions on each points ? And what are, for you, the biggest advantages of SciLab ? Why have you choose to work with it ?

Thanks to spend a little of your time for this post.

Nicolas.

I think Scilab can do all this (except may be the last item). But then the nice thing about free software is that you can try it yourself... Note however that the tcl-tk interface is a beta release for now; it has not been extensively tested.

Dr Scilab

- **Este usuario comenta las diferencias que existen en el código *Matlab* y *Scilab*.**

Subject: Re: scilab-matlab-compatibility
From: physrmh@phys.canterbury.ac.nz (Ryurick M. Hristev)
Date: 30 Jul 1998 08:00:33 +1200
Newsgroups: comp.soft-sys.math.scilab
Organization: University of Canterbury
References: <6pmjdh$r8s@majestix.uni-muenster.de>

"Frank Wuebbeling" <wuebbel@uni-muenster.de> writes:

-> The precedence of . related to * is different in matlab and scilab.*
While in matlab the construction

```
vol = geometrie(:,1).*a/2*ones(1,size(phi_x,2));
A = phi_x'*vol.*phi_x+phi_y'*vol.*phi_y;
```

worked, I had to add parentheses to have it do the same thing in scilab:

```
vol = (geometrie(:,1).*a/2)*ones(1,size(phi_x,2));
A = phi_x'*(vol.*phi_x)+phi_y'*(vol.*phi_y);
```

This is a bad thing. While this will produce an error if the dimensions don't match, this is never going to be noted by anybody if the dimensions match and just going to produce wrong results.

AFAIK the precedence of Hadamard product vs. regular product is undefined in mathematical theory, or at least there ain't a widespread mathematical convention.

*So IMHO is a bad thing *not* to use parentheses in either case, if nothing else because it will make your code unreadable by others or later.*

-> This is one that I found most annoying: I found no obvious way of getting rid of the message.

In my code I use something like:

```
// save old values of lines
old_lines = lines();
// ... and change to suit you
lines(3000,99);

// start code
....
// end code

// restore lines to previous values
lines(old_lines(2),old_lines(1));
```

See lines man page for details.

Cheers,

- **Comenta diferencias en el tiempo de calculo para sparce matrix con *Matlab* y *Scilab*.**

To: Frank Wuebbeling <wuebbel@uni-muenster.de>
Subject: Re: scilab-matlab-compatibility
From: Serge Steer <Serge.Steer@inria.fr>
Date: Fri, 21 Aug 1998 18:05:40 +0200
CC: scilab@inria.fr
Newsgroups: comp.soft-sys.math.scilab
Organization: I.N.R.I.A Rocquencourt
References: <6pmjdh$r8s@majestix.uni-muenster.de> 6pn0r4$gfs@news-rocq.inria.fr
<6pna0a$ru4@majestix.uni-muenster.de>

Frank Wuebbeling wrote:

I'm not sure about that one. Look at the following innocent code. It builds a bandlimited Hilbert-matrix (or is it i-k? blush...). This one is bandlimited, but spreading the entries doesn't change the point. My memory is large enough to hold the matrices:

```
clear
n=300
w=10
a=sparse(zeros(n,n));
for i=1:n, for k=max(1,n-w):min(n,n+w), a(i,k)=1/(i+k-1); end; end
```

*Execution time is about two seconds in matlab, but 15 seconds in scilab (on LINUX on a 90 MHz Pentium PC). That's well over a factor of 5.Results on a SOLARIS-machine show a similar (altough smaller) factor. I know that building up matrices is not what is time-consuming, but it might be a hint that something's wrong with the sparse code, or with *my* code. Is there something I should definitely do or not do when handling sparsity in scilab?*

Probably the sparse matrix insertion in scilab can be improved. It seems that our internal storage is not efficient for insertion, but is efficient for arithmetic operations for example mutiplication,transposition,addition of sparse matrices in scilab are faster in scilab than in matlab

```
Scilab:
-->a=sprand(300,300,0.03);
-->nnz(a)
 ans  =

  2742.

-->timer();for k=1:100;a+a';end;timer()
ans  =

1.066624

-->timer();for k=1:10;a*a';end;timer()
ans  =
0.799968
```

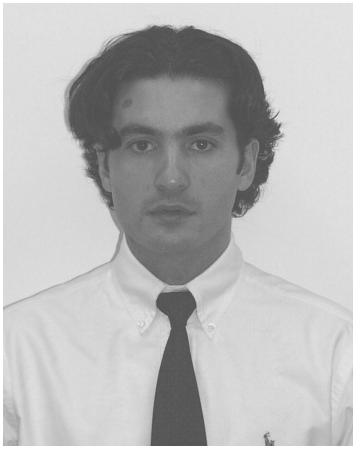
```
Matlab:

>> a=sprandn(300,300,0.03);
>> nnz(a)

ans =

  2659
>> t=cputime;for k=1:100;a+a';end;cputime-t

ans =

2.9000

>>  t=cputime;for k=1:10;a*a';end;cputime-t

ans =

1.8167

Both on a Dec alpha 300
Serge Steer
Inria Rocquencourt
BP 105 78153 Le Chesnay CEDEX
email:Serge.Steer@inria.fr
```

10. BIBLIOGRAFÍA.

- **INTRODUCTION TO SCILAB** (Grupo Scilab – INRIA) **http//www-rocq.inria.fr/scilab**.
- **SIGNAL PROCESSING WITH SCILAB** (Grupo Scilab – INRIA) **http//www-rocq.inria.fr/scilab**.
- **THE MATLAB EXPO** (The Math Works, Inc) .
- **TRATAMIENTO DIGITAL DE SEÑALES – CURSO DE DOCTORADO** (Dr. Sebastián Dormido y Y Dr. José Luis Fdez. Marrón – Dpto. Informática y Automática- Facultad de Ciencias-UNED).
- **SIGNAL PROCESSING TOOLBOOX – MATLAB** (The Math Works, Inc).

11. AGRADECIMIENTOS.

Quiero agradecer especialmente **al Prof. Dr. José Luis Fdez. Marrón del Dpto. de Informática y Automática de la UNED** por haberme descubierto la herramienta *Scilab*, así como al **Dr. Claude Gómez del Institut National de Recherche en Informatique et en Automatique (INRIA)** por haberme respondido a mis e-mails con dudas sobre *Scilab* casi en tiempo real.



Bernardo A. Delicado has a **B.S. DEGREE in AERONAUTICAL ENGINEERING** and a **M.S. DEGREE in Automatic Control.** He currently studies his second year **Ph.D.** in Automatic Control.

Bernardo works as a **Test Engineer** in **The Electromagnetic Compatibility Aircraft Test Group** of **INTA** (the spanish aerospace agency) and a **Professor** at **the Universidad Carlos III de Madri**d in **the Electronics and Automatic Engineering Departament**.

e-mail : delicadob@inta.es

Bernardo A. Delicado
INTA (Instituto Nacional de Técnica Aeroespacial)
Grupo de Ensayos de EMC en Aeronaves
Carretera de Ajalvir, Km 4
28850 TORREJÓN DE ARDOZ
MADRID (ESPAÑA)

Tel. 34 91 5201719
FAX : 34 91 5202021