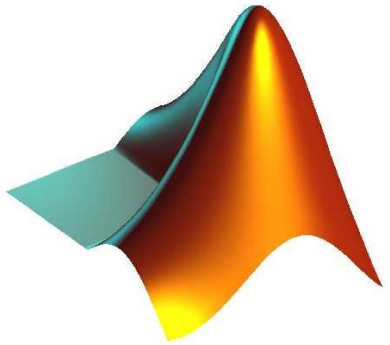
Complementos de Procesado de Señal y Comunicaciones
Máster en Sistemas Multimedia

Introducción a Matlab y Simulink



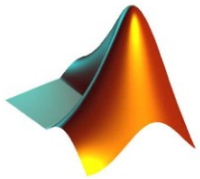
Javier Ramírez Pérez de Inestrosa
Dpto. Teoría de la Señal, Telemática y Comunicaciones
Universidad de Granada
Email: javierrp@ugr.es

Este tutorial se puede obtener en:
http://www.ugr.es/~javierrp

¿Qué es Matlab?

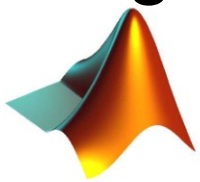
- MATLAB es un lenguaje de alto nivel para realizar cálculos cientifico-técnicos.

- Integra las herramientas de cálculo necesarias con otras de visualización así como, un entorno de programación de fácil uso.



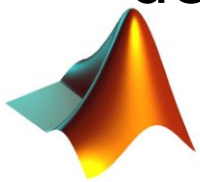
Aplicaciones típicas

- Cálculo matemático
- Desarrollo de algoritmos
- Adquisición de datos
- Modelado, simulación y prototipado
- Análisis de datos y visualización
- Gráficos
- Desarrollo de aplicaciones e interfaces gráficas de usuario (GUI)

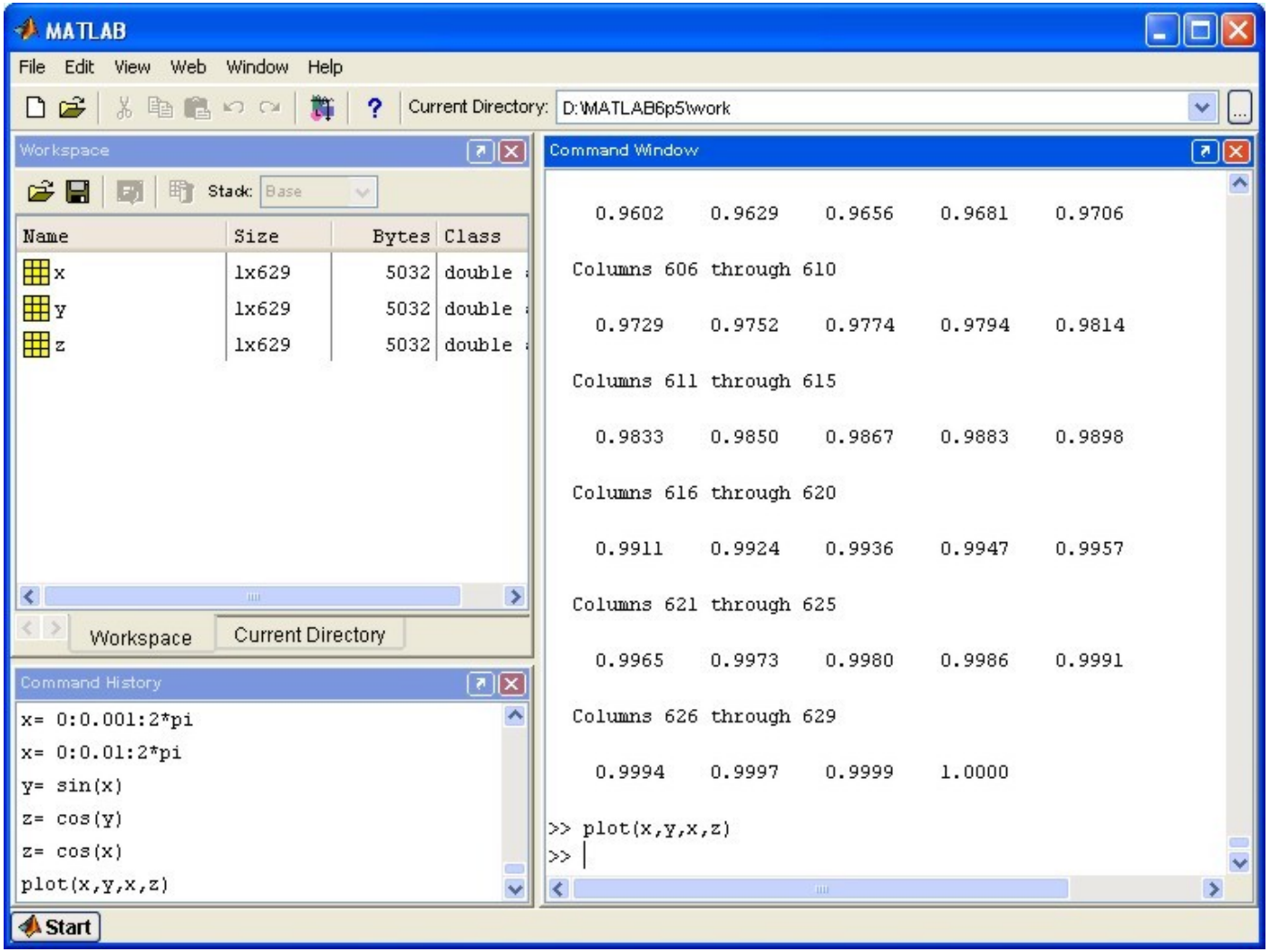


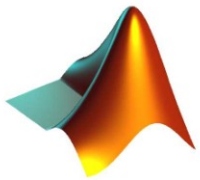
Más sobre MatLab

- MatLab significa “MATrix LABoratory”
- El tipo básico de datos es el vector que no requiere ser dimensionado.
- Proporciona unos paquetes de extensión (“toolboxes”) para aplicaciones específicas
- Estos paquetes incluyen librerías de funciones MatLab (M-files) que extienden las posibilidades de MatLab para resolver problemas específicos



El entorno de Matlab

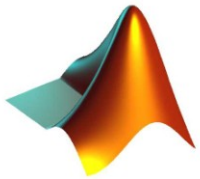




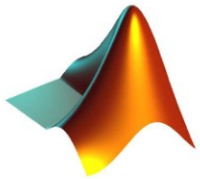
Sintaxis

- Algunos ejemplos sencillos

Entrada	Salida	Comentarios
2 + 3 7-5 34*212 1234/5786 2^5	ans = 5 ans = 2 ans = 7208 ans = 0.2173 ans = 32	Los resultados son los esperados. Nótese que al resultado se le da el nombre *ans*.
a = sqrt(2)	a = 1.4142	Se puede escoger el nombre de la variable.



b = a, pi, 2 + 3i	b = 1.4142 ans = 3.1416 ans = 2.0000 + 3.0000i	Se pueden introducir varios comandos en una sola línea. Pi, i, y j son constantes.
c = sin(pi) eps	c = 1.2246e-016 ans = 2.2204e-016	"eps" es el limite actual de precisión. No se puede operar con números inferiores a eps.
d = [1 2 3 4 5 6 7 8 9] e = [1:9] f = 1:9	d = 1 2 3 4 5 6 7 8 9 e = 1 2 3 4 5 6 7 8 9 f = 1 2 3 4 5 6 7 8 9	Definición de vectores. "d", "e", son "f" vectores. Son iguales. El operador ":" se utiliza para formar vectores; cuenta desde el número inicial al final de uno en uno.
g = 0:2:10 f(3) f(2:7) f(:)	g = 0 2 4 6 8 10 ans = 3 ans = 2 3 4 5 6 7 1 2 3 4 5 6 7 8 9	Otros usos de ":". Se utiliza para acceder a parte o la totalidad de los datos de un vector o matriz.

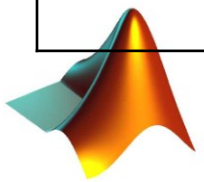


`h = [1 2 3];`
`h'`
	`(nada)`
	`ans = 1`
	`2`
	`3`
		Un punto y coma ";" evita que se visualice la salida.
		Una coma simple " ' " calcula la traspuesta de una matriz, o en el caso de vectores, intercambia entre vectores fila y columna.

`h * h'`
`h .* h`
`h + h`
	`ans = 14`
	`ans = 1 4 9`
	`ans = 2 4 6`
		Operaciones con vectores. * es la multiplicación matricial. Las dimensiones deben ser las apropiadas.
		" .* " es la multiplicación componente a componente.

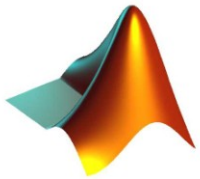
`g = [ 1 2 3;`
`4 5 6; 7 8 9]`
	`g = 1 2 3`
	`4 5 6`
	`7 8 9`
		Construcción de matrices.

`g(2,3)`
`g(3,:)`
`g(2,3) = 4`
	`ans = 6`
	`ans = 7 8 9`
	`g = 1 2 3`
	`4 5 4`
	`7 8 9`
		Accediendo a los elementos de la matriz.
		":" se utiliza para acceder a una fila completa.



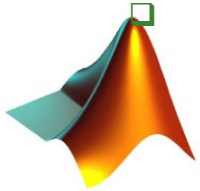
Entrada	Salida	Comentarios

`g^2`
 	
`g .^ 2`
	`ans =  30  36  42`
	`66  81  96`
	`102 126 150`
	`ans =  1  4  9`
	`16 25 36`
	`49 64 81`
		Multiplica la matriz por ella misma.
		
		Eleva al cuadrado cada elemento de la matriz.



Control de la salida

- El comando `format`
  - `format compact`
    - Controla el espaciado de líneas.
  - `format long`
    - Muestra los 15 dígitos que se utilizan en el cálculo.
  - `format short`
    - Muestra únicamente cinco dígitos.
  - “;” al final del comando.
    - No visualizar salida:
- `help format`
  - Más información.



Más sobre matrices

- Funciones incluidas en MatLab

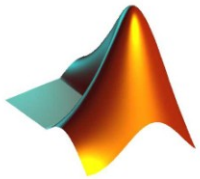
Entrada	Salida	Comentarios

`rand(2)`
 	
`rand(2,3)`
	`ans = 0.9501 0.6068`
	`0.2311 0.4860`
	`ans = 0.8913 0.4565 0.8214`
	`0.7621 0.0185 0.4447`
		Genera una matriz de números aleatorios entre 0 y 1

`zeros(2)`
 	
`ones(2)`
	`ans = 0 0`
	`0 0`
	`ans = 1 1`
	`1 1`
		Genera una matriz 2x2 de ceros o unos.

`eye(2)`	`ans = 1 0` `0 1`	Matriz identidad I.

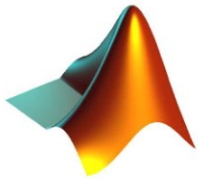
`hilb(3)`	`ans = 1.0000 0.5000 0.3333`
	`0.5000 0.3333 0.2500`
	`0.3333 0.2500 0.2000`
		Matriz de Hilbert 3x3.



Más sobre matrices

- Concatenación
  - Generar nuevas matrices a partir de otras creadas previamente
  - Por ejemplo:
    - Sea la matriz a:

```
>> a = [1 2; 3 4]
a =
     1  2
     3  4
```

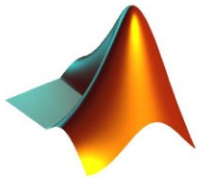


Más sobre matrices - concatenación

Entrada	**Salida**
`[a, a, a]`	`ans = 1 2 1 2 1 2` `3 4 3 4 3 4`

`[a; a; a]`	`ans = 1 2`
	`3 4`
	`1 2`
	`3 4`
	`1 2`
	`3 4`

`[a, zeros(2); zeros(2), a']`	`ans = 1 2 0 0`
	`3 4 0 0`
	`0 0 1 3`
	`0 0 2 4`

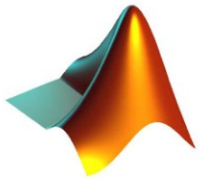


Más sobre matrices

- Programación
  - Se pueden construir matrices mediante programación

```
for i=1:10,
    for j=1:10,
        t(i,j) = i/j;
    end
end
```

  - No se produciría salida puesto que la única línea que podría generar salida `(t(i,j) =i/j;)` termina en ";"
  - Sin el ";", Matlab escribiría la matriz t 100 veces!!



Operaciones con matrices

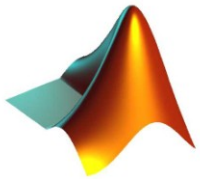
- +, -, *, y /
  - Definen operaciones con matrices.

- Debemos distinguir:
  - ".*":

    Multiplicación componente a componente.

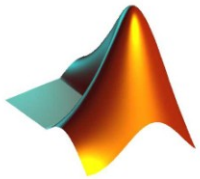
  - "*"

    Multiplicación matricial.



Escalares

- Un escalar es un número.
- Matlab los almacena como matrices 1x1
- Todas las operaciones entre escalares y matrices se realizan componente a componente salvo:
  - La potencia (“^”).

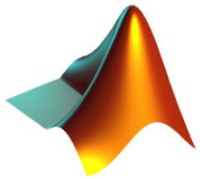


Escalares

```
a = 1   2
    3   4
```

- Ejemplos

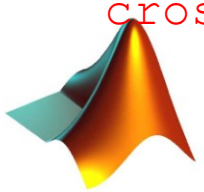
Entrada	Salida	Comentarios
`b=2`	`b=2`	Define b como un escalar.
`a + b`	`ans = 3 4` `5 6`	La suma se hace componente a componente.
`a * b`	`ans = 2 4` `6 8`	Igual que la multiplicación.
`a ^ b`	`ans = 7 10` `15 22`	Potencia matricial - a*a
`a .^ b`	`ans = 1 4` `9 16`	Potencia componente a componente.



Vectores

- Un vector es una matriz de una sola fila o columna

Entrada	Salida	Comentarios
`v = [1 2 3]` `u = [3 2 1]`	`v = 1 2 3` `u = 3 2 1`	Define 2 vectores.
`v * u`	`Error`	Las dimensiones no coinciden.
`v * u'`	`ans = 10`	Al tomar la traspuesta se corrige el error.
`dot(v,u)`	`ans = 10`	Producto escalar (idéntico al anterior).
`cross(v,u)`	`ans = -4 8 -4`	El producto vectorial sólo se emplea con vectores en 3 dimensiones.



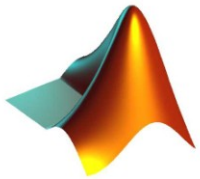
Matrices

- Matlab tiene numerosas funciones predefinidas (`help matfun`).

Entrada	Salida	Comentarios

`k = [16  2  3;`
`5 11 10;`
`9  7  6]`
	`k = 16  2  3`
	`5 11 10`
	`9  7  6`
		Define una matriz.

`trace(k)`	`ans = 33`	Traza de una matriz
`rank(k)`	`ans = 3`	Rango de una matriz.
`det(k)`	`ans = -136`	Determinante de una matriz

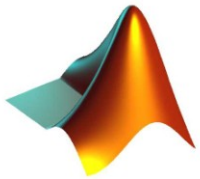


Matrices

Entrada	Salida	Comentarios

`inv(k)`	`ans =  0.0294 -0.0662  0.0956`
	`-0.4412 -0.5074  1.0662`
	`0.4706  0.6912 -1.2206`
		Inversa de una matriz

`[vec,val] = eig(k)`	`vec = -0.4712 -0.4975 -0.0621`
	`-0.6884  0.8282 -0.6379`
	`-0.5514  0.2581  0.7676`
	
	`val = 22.4319  0  0`
	`0  11.1136  0`
	`0  0  -0.5455`
		Vectores propios y autovalores de una matriz.
		Las columnas de "vec" contienen los vectores propios;
		las entradas de la diagonal de "val" son los autovalores.

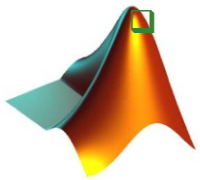


Variables en el espacio de trabajo

- `whos`
  - Lista las variables definidas en el entorno.

```
>> whos
  Name        Size                    Bytes  Class

  a         100x1                       800  double array
  b         100x100                   80000  double array
  c           1x1                         8  double array

Grand total is 10101 elements using 80808 bytes
```

- `clear`
  - Borra variables del entorno.

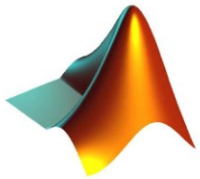


Resolución de sistemas de ecuaciones

- Una de las principales aplicaciones de las matrices es la representación de sistemas de ecuaciones lineales.
- Si **a** es una matriz de coeficientes, **x** es un vector columna que contiene las incógnitas y **b** los términos constantes, la ecuación

$$\boldsymbol{a\,x = b}$$

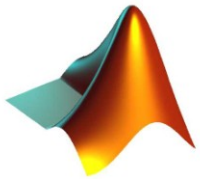
representa el correspondiente sistema de ecuaciones.



Resolviendo ecuaciones

- Para resolver el sistema en MatLab
  - `x = a \ b`
  - **x** es igual a la inversa de **a** por **b**
- Ejemplo
  - `a = [1 2 3; 4 5 6; 7 8 10]; b = [1 1 1]';`
  - Solución:

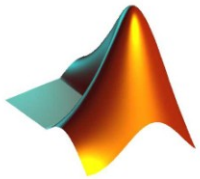
```
x =
     -1
      1
      0
```



Salvar y recuperar datos

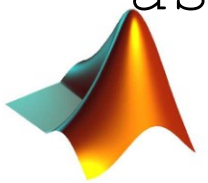
- Los datos de la sesión se pierden al salir de MatLab.

- Para salvar la sesión (entrada y salida)
  - `Diary('session.txt');`
    - `Guarda los comandos introducidos en la sesión.`
  - `Diary <ON/OFF> ;`
- Para salvar una o varias matrices
  - `save datos.mat (guarda todas las variables)`
  - `save datos.mat x  (sólo guarda x)`



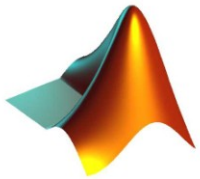
Salvar y recuperar matrices

- `save sesion`
  - Salva todas las variables en el archivo binario “sesion.mat”.
- `save fichero X`
  - Salva sólo la variable X
- `load sesion`
  - Recupera los datos previamente salvados
- Si los ficheros se pueden salvar en formato texto (`-ascii`). Pueden verse con un editor de textos.



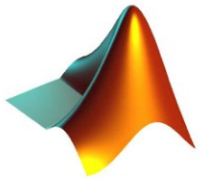
Gráficos

- El comando básico es: `plot`
- `plot(y);`
- `plot(x,y);`
- `plot(x,y,'b+',x,z,'gx');`
  - `color (b,g) blue,green`
  - `Marcador (+,x)`
- Personalización del gráfico:
  - `title, xlabel, ylabel, legend, grid.`



Ejemplo

```
Mes = 1:12;
T_Gr = [-2 0 2 4 8 12 14 14 12 8 4 0];
T_Ma = [-4 -2 0 2 6 14 18 18 16 8 2 -2];

plot(Mes, T_Gr, 'bo', Mes, T_Ma, 'rv');

xlabel('Mes');
ylabel('Temperatura (°C)');
title('Temperaturas minimas en Granada y Madrid');
legend('Granada','Madrid');
grid;
```



Resultado:

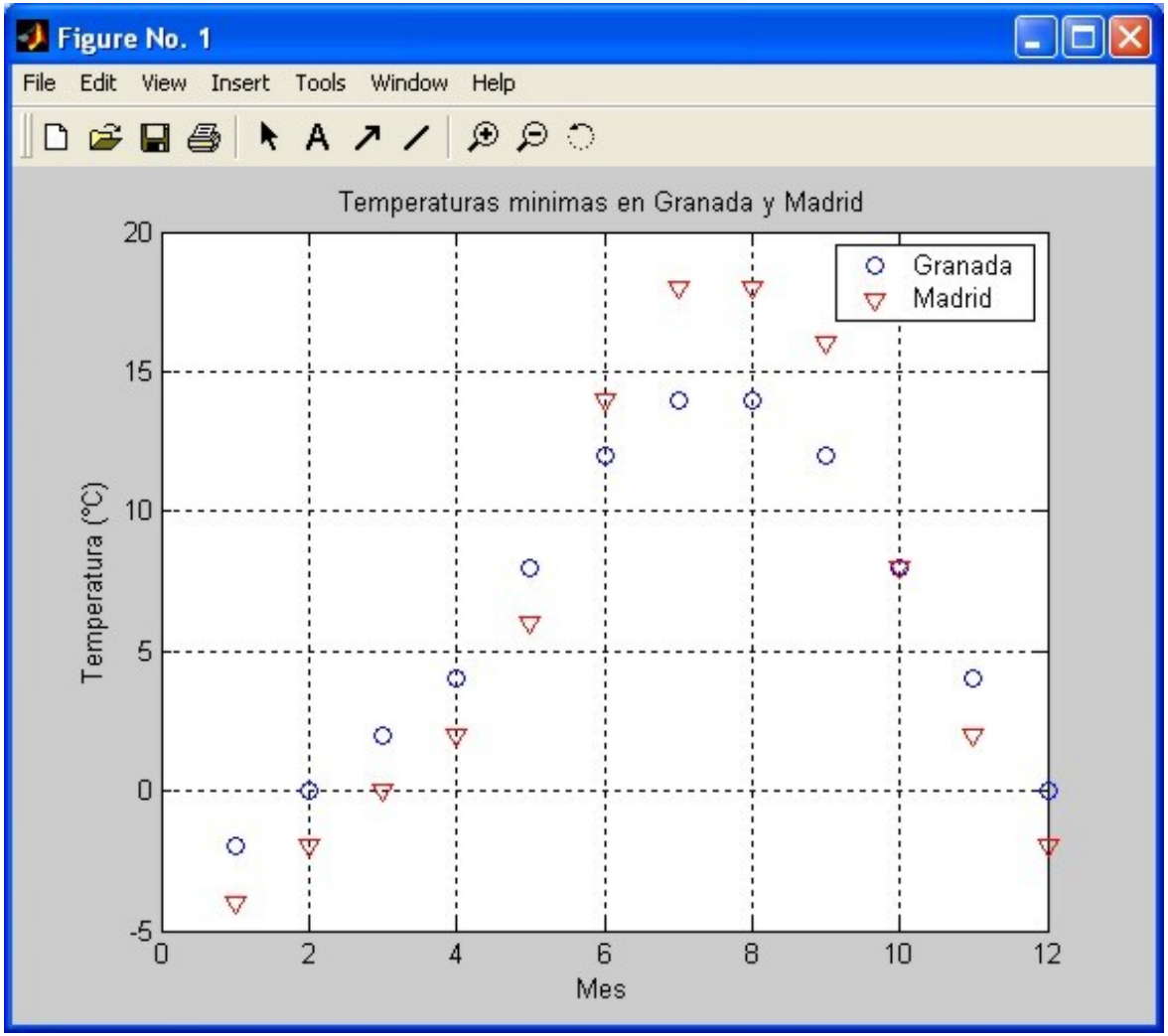
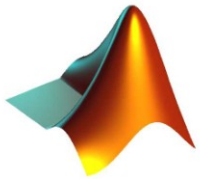
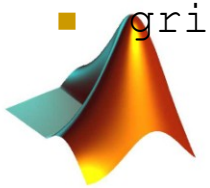


Figure No. 1
File Edit View Insert Tools Window Help
Temperaturas minimas en Granada y Madrid
Granada
Madrid
Temperatura (°C)
Mes
20
15
10
5
0
-5
0
2
4
6
8
10
12



subplot

```
Mes = 1:12
T_Gr = [-2 0 2 4 8 12 14 14 12 8 4 0];
T_Ma = [-4 -2 0 2 6 14 18 18 16 8 2 -2];

subplot(2,1,1);
plot(Mes, T_Gr, 'bo-');
xlabel('Mes');
ylabel('Temperatura (ºC)');
title('Temperaturas minimas en Granada');
grid;

subplot(2,1,2);
plot(Mes, T_Ma, 'rv-');
xlabel('Mes');
ylabel('Temperatura (ºC)');
title('Temperaturas minimas en Madrid');
grid;
```



Resultado

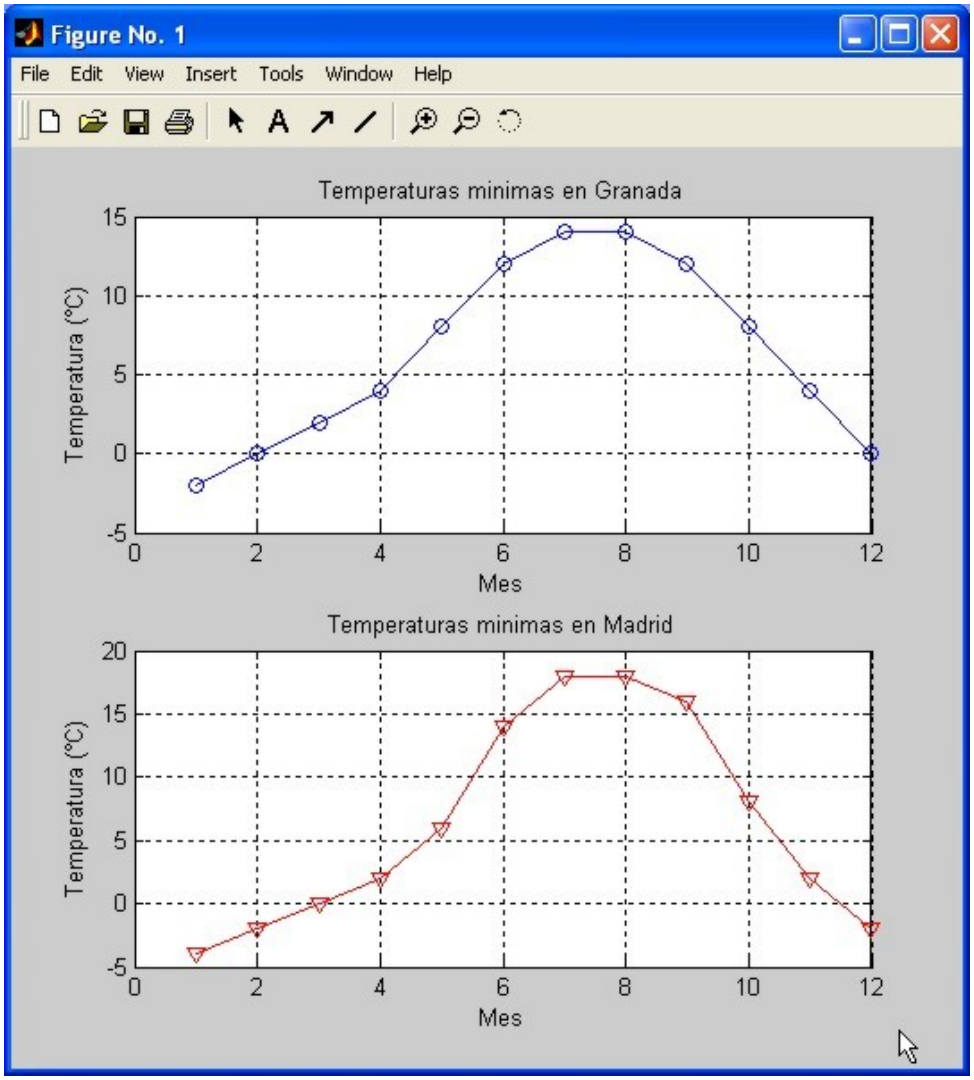
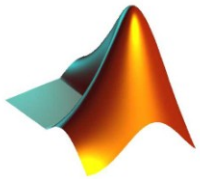
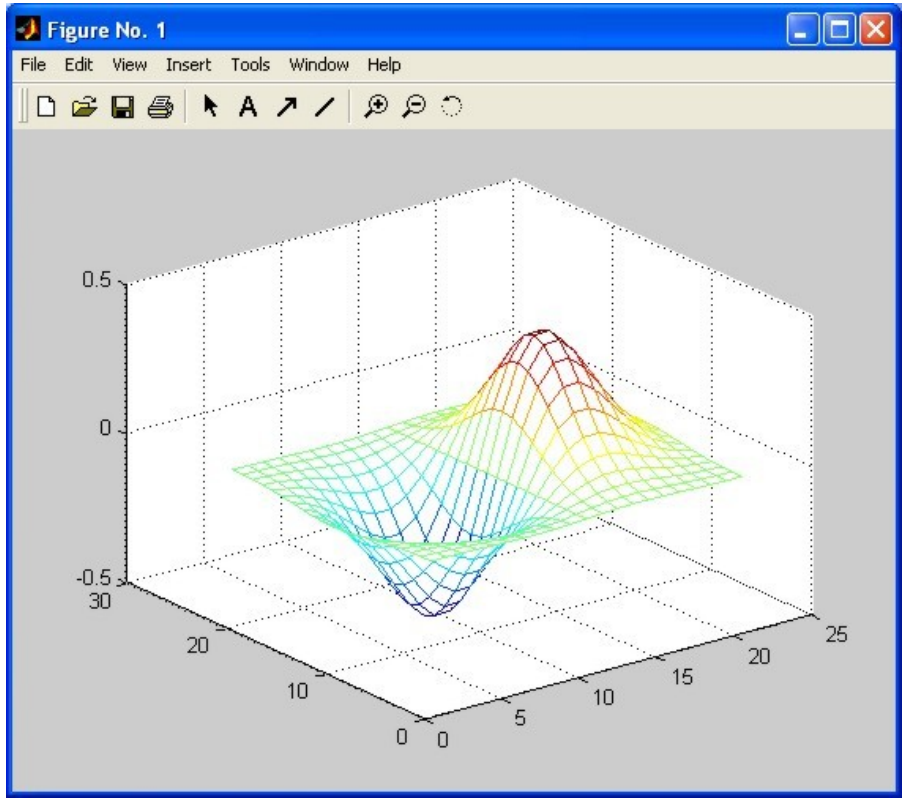


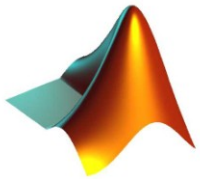
Figure No. 1
File Edit View Insert Tools Window Help
Temperaturas minimas en Granada
Temperatura (°C)
Mes
Temperaturas minimas en Madrid
Temperatura (°C)
Mes



Gráficos tridimensionales

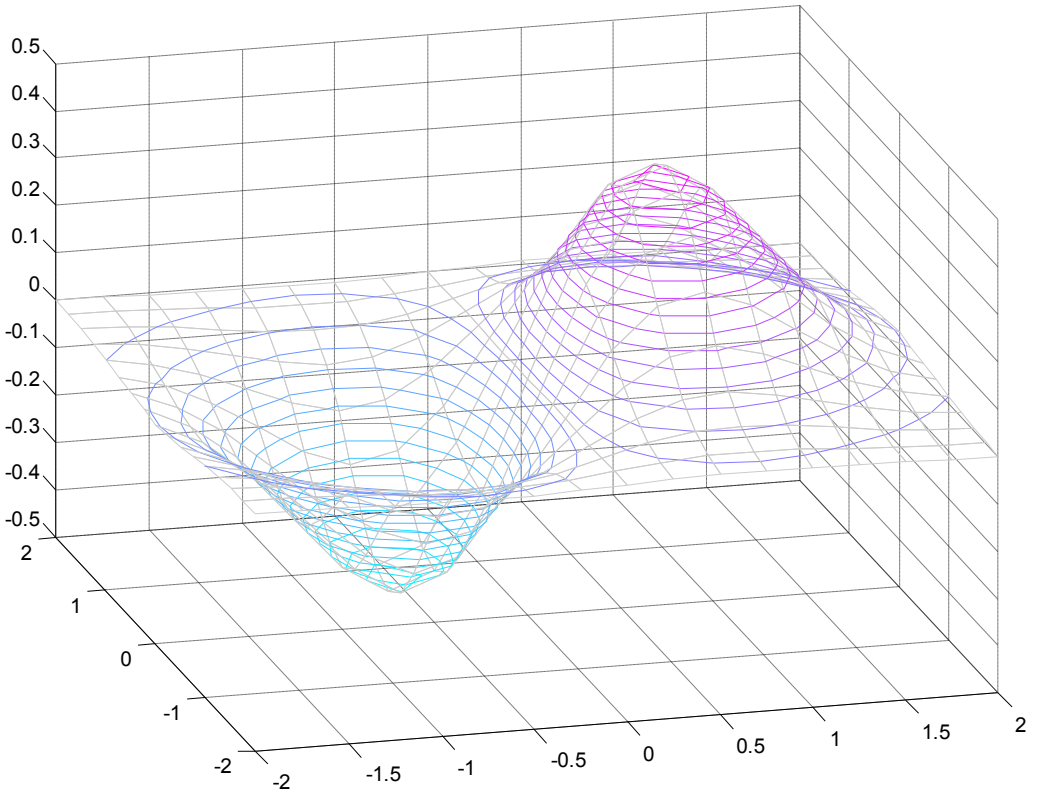
- `[x,y] = meshgrid(-2:.2:2, -2:.2:2);`
- `z = x .* exp(-x.^2 - y.^2);`
- `mesh(z);`

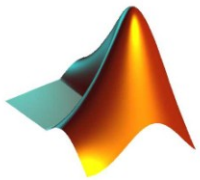




Gráficos tridimensionales

```
Z = X.*exp(-X.^2-Y.^2);
contour3(X,Y,Z,30)
surface(X,Y,Z,'EdgeColor',[.8 .8.8],'FaceColor','none')
grid off
view(-15,25)
colormap cool
```

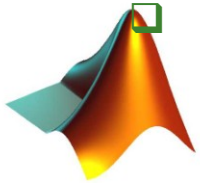




Programación

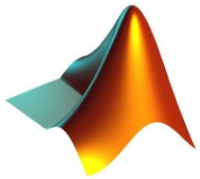
- Ficheros de comandos (*scripts*)
  - Secuencias de comandos.
  - Al invocarlos se ejecutan en el entorno.
  - Las variables creadas son globales.

- Ficheros de función
  - Permiten definir funciones propias.
  - Variables locales.
  - La información se pasa como parámetros.
  - Se pueden definir subfunciones.



Un ejemplo de función

```
function y = media (x)
% Valor medio de x.
% Para vectores, media(x) devuelve el valor medio.
% Para matrices, media(x) es un vector fila
% que contiene el valor medio de cada columna.
[m,n] = size(x);
if m == 1
  m = n;
end
  y = sum(x)/m;
```

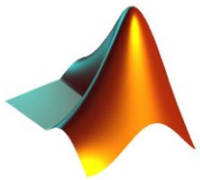


Argumentos de funciones

- `nargin y nargout`
  - Número de argumentos de entrada y salida con los que se llama a la función.

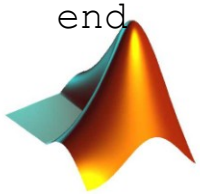
- Ejemplo:

```
function c = testarg1(a,b)
if (nargin == 1)
    c = a.^2;
elseif (nargin == 2)
    c = a + b;
end
```

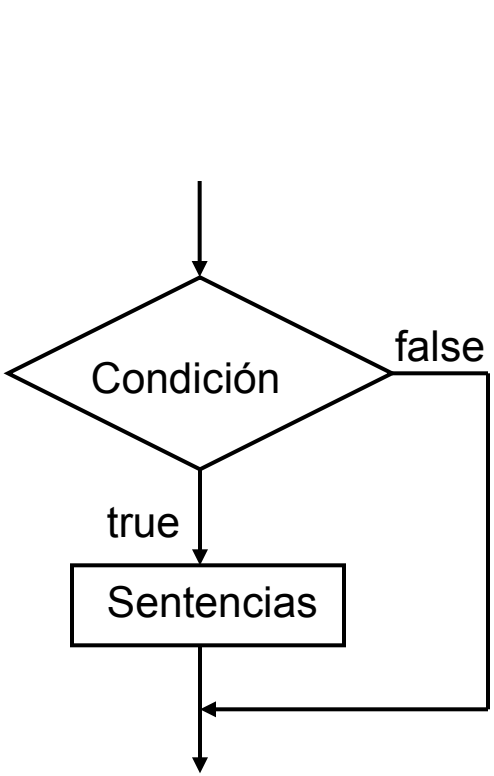


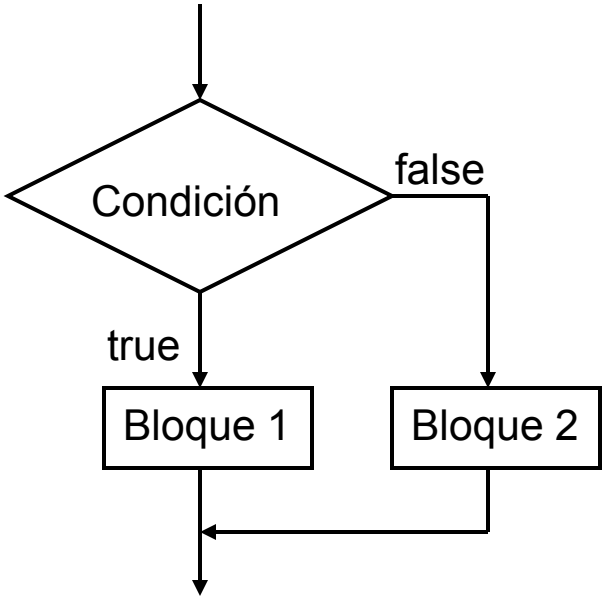
Subfunciones

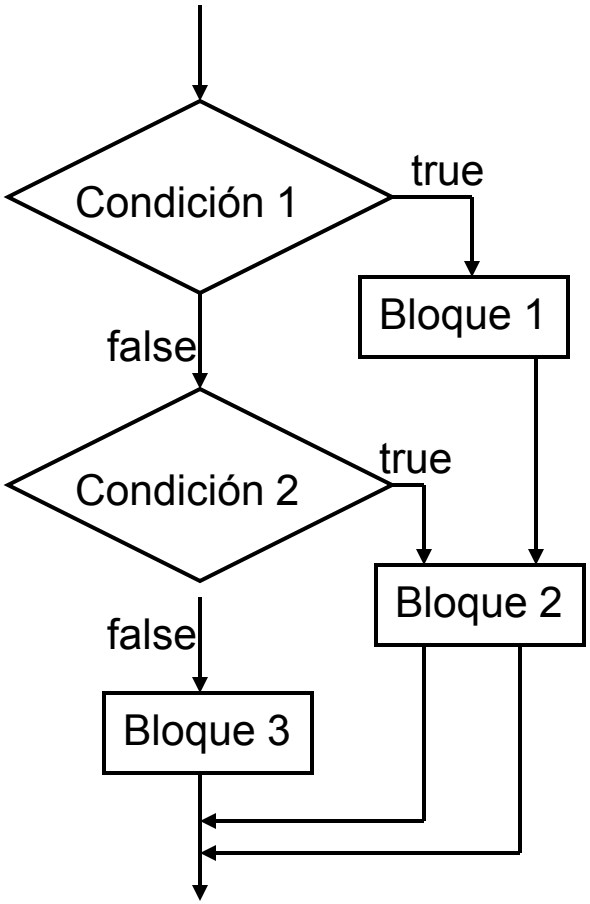
```
function [media,mediana] = estadistica(u) % Función principal
% ESTADISTICA Calcula la media y la
% mediana utilizando funciones internas.
n = length(u);
media = mean(u,n);
mediana = median(u,n);

function a = mean(v,n)              % Subfunción
% Calcula la media.
a = sum(v)/n;

function m = median(v,n)            % Subfunción
% Calcula la mediana.
w = sort(v);
if rem(n,2) == 1
    m = w((n+1)/2);
else
    m = (w(n/2)+w(n/2+1))/2;
end
```

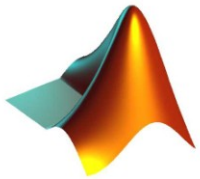


Bifurcaciones









if

Sentencia if

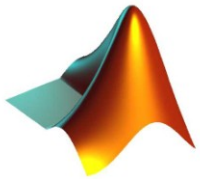
```
if condición
   sentencias
end
```

Ejemplo

```
if rem(a,2) == 0
    disp('a is par')
    b = a/2;
end
```

Bifurcación multiple

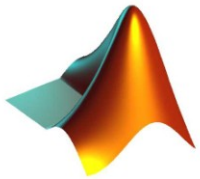
```
if condición1
   bloque1
elseif condición2
    bloque2
elseif condición3
    bloque3
else
    bloque4
end
```



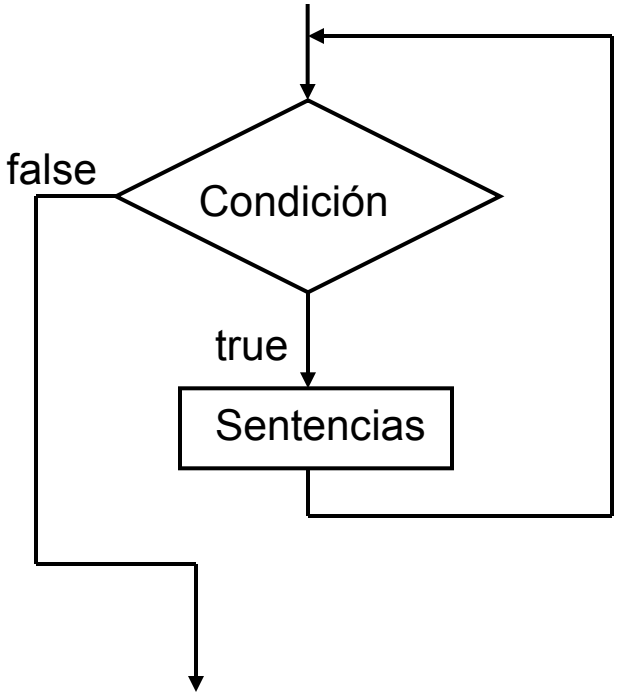
switch

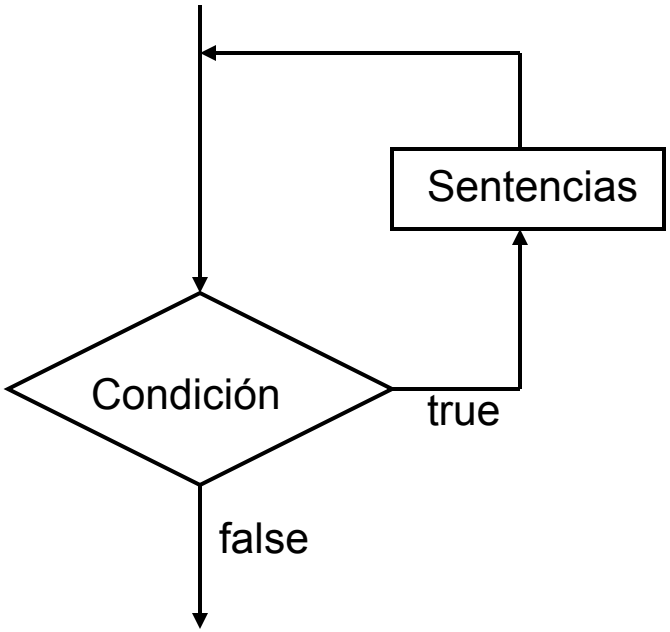
```
switch expresion
       case case1,
              bloque1
       case {case2, case3, ...}
              bloque2
       ...
       otherwise,
              bloque3
end
```

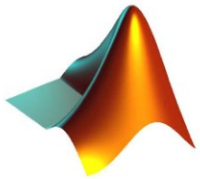
```
switch valor
    case -1
        disp('negativo');
    case 0
        disp('cero');
    case 1
        disp('positivo');
    otherwise
         disp('otro');
end
```



Bucles


false
Condición
true
Sentencias


Sentencias
Condición
true
false



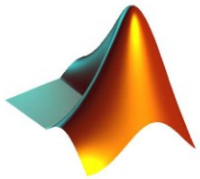
for

```
for i = 1:n
     sentencias
end
```

```
for i = vector
     sentencias
end
```

```
for i = n:-0.2:1
     sentencias
end
```

```
for i = 1:m
     for j = 1:n
          sentencias
     end
end
```



while {continue, break}

```
while condición
      sentencias
end
```

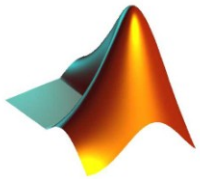
Sentencia **break**

Hace que termine la ejecución

Sentencia **continue**

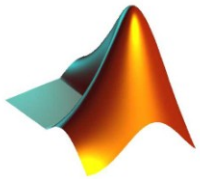
Hace que se pase inmediatamente a la siguiente iteración del bucle for o while

```
n = 1;
while prod(1:n) < 1e100
    n = n + 1;
end
```



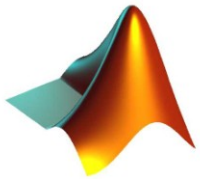
Toolboxes

- Toolboxes para DSP y comunicaciones:
  - Communications Toolbox
  - Filter Design Toolbox
  - Image Processing Toolbox
  - Signal Processing Toolbox
  - Statistics Toolbox
  - System Identification Toolbox
  - Wavelet Toolbox



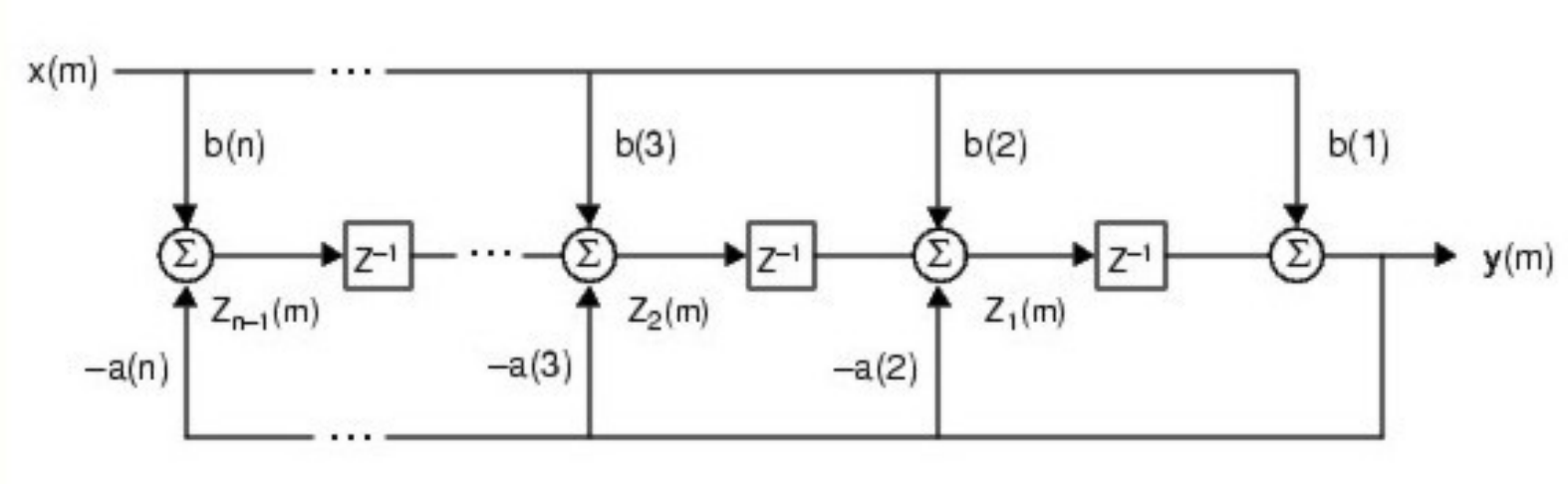
Procesamiento de señales y comunicaciones

- Matlab dispone de unas librerías para tratamiento digital de señales.
  - Signal Processing Toolbox
  - Communications Toolbox

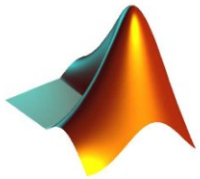


Filtrado de señales

- `y= filter(b,a,x);`
  - Filtra la secuencia x con el filtro descrito por b y a.



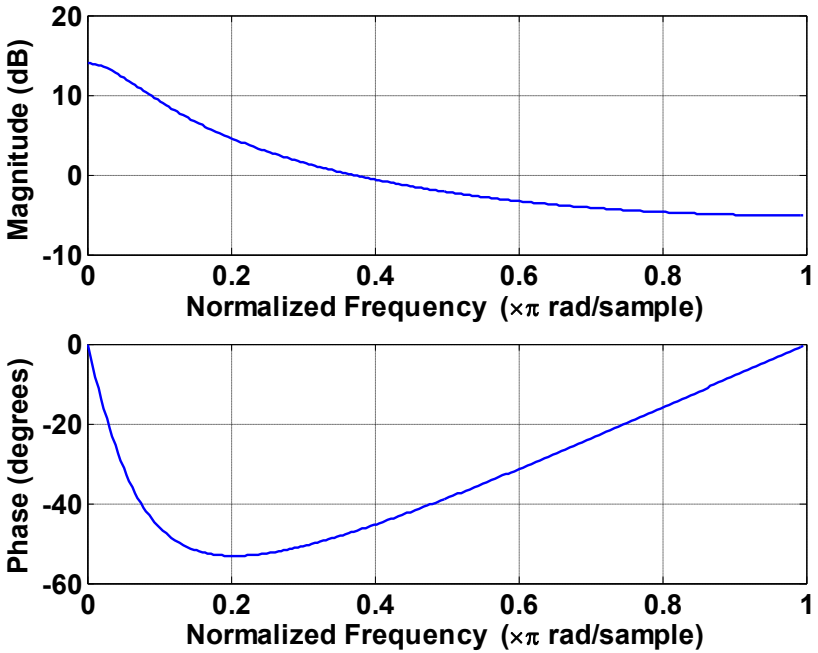
$$Y(z) = \frac{b(1) + b(2)z^{-1} + \ldots + b(nb+1)z^{-nb}}{1 + a(2)z^{-1} + \ldots + a(na+1)z^{-na}} X(z)$$

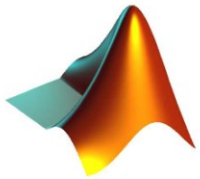


Respuesta en frecuencia

- `[H,w] = freqz(b,a,N);`
  - Calcula N puntos de la respuesta en frecuencia del filtro definido por b y a.
  - Para el filtro:

    $y(n) = 0.8 \cdot y(n-1) + x(n)$
  - `freqz(1,[1 -0.8],256);`





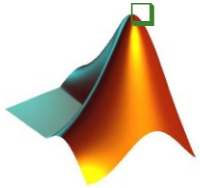
Diseño de filtros

- `B = FIR1(N,Wn);`
  - Filtro FIR paso baja de orden N.
  - Wn es la frecuencia de corte normalizada (0<Wn<1).

- `B = FIR1(N,Wn,'high');`
  - Filtro paso alta.

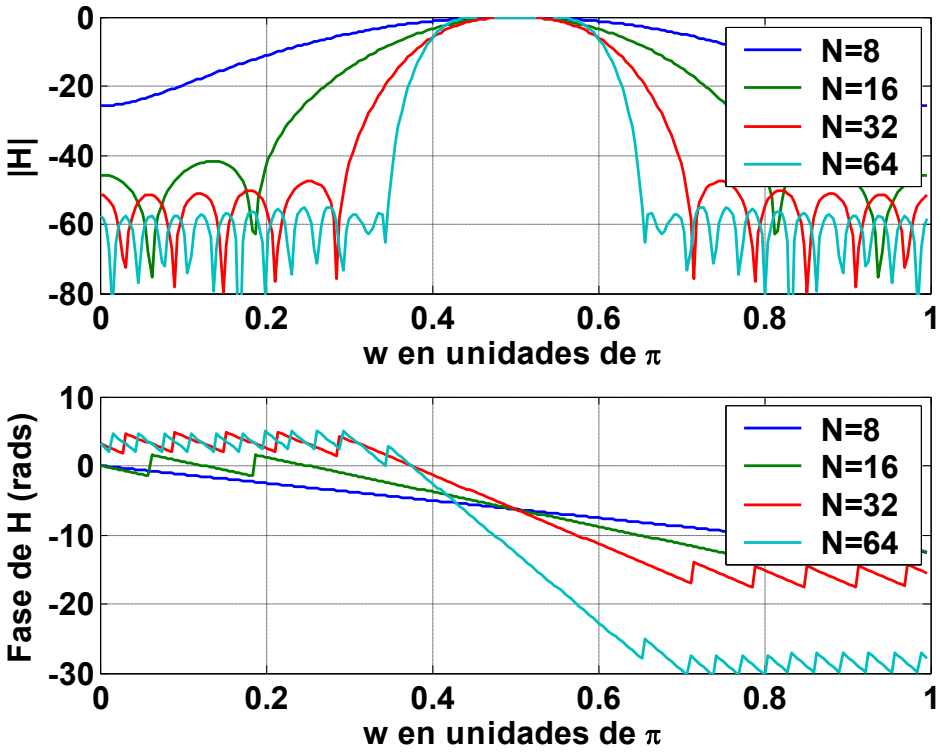
- `Wn = [W1 W2]; B = FIR1(N,Wn,'bandpass');`
  - Filtro paso banda.

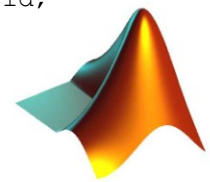
- `Wn = [W1 W2]; B = FIR1(N,Wn,'stop');`
  - Filtro rechaza banda (notch).



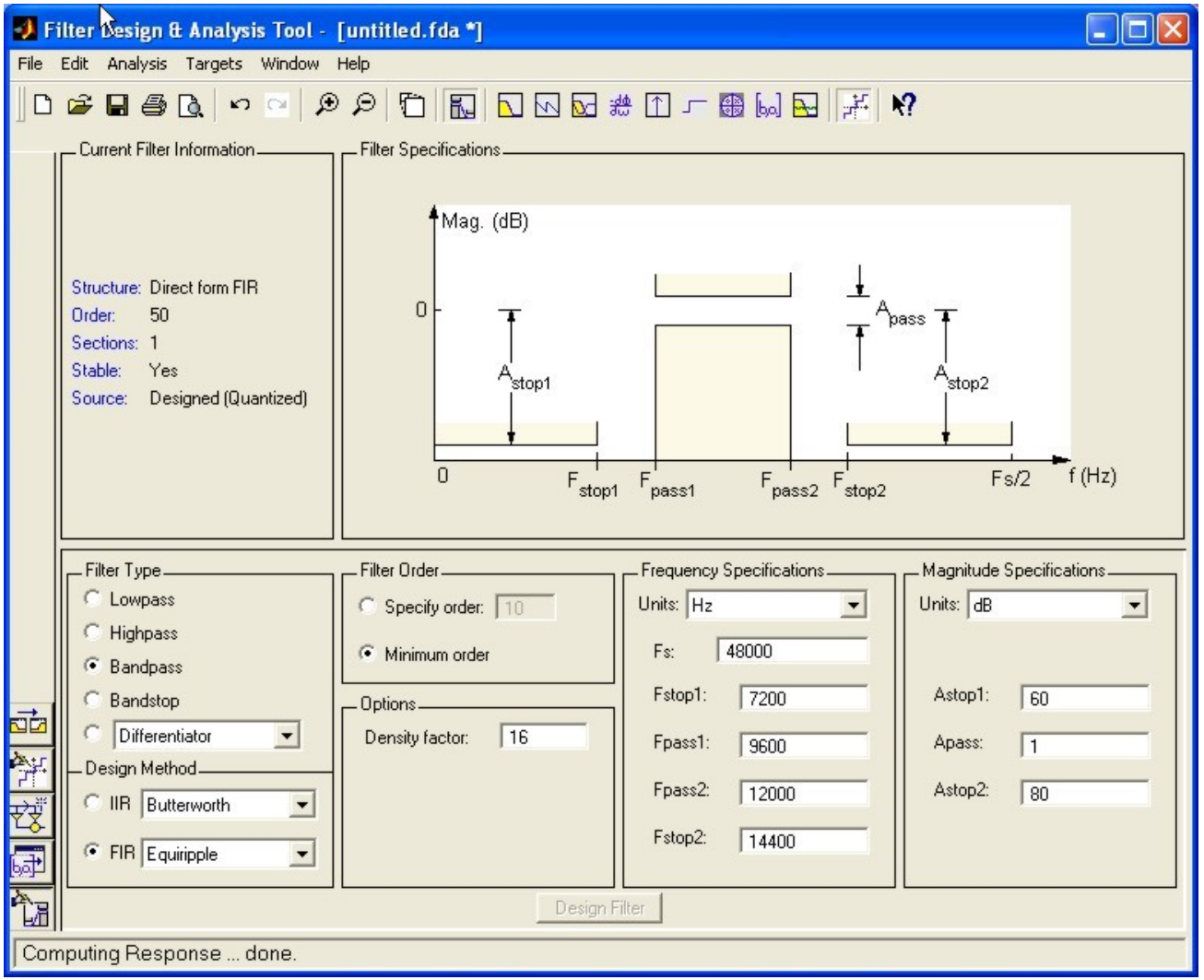
Ejemplo: Filtros FIR paso banda

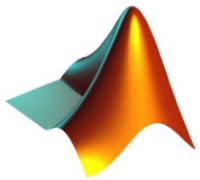
```
% Parametros.
N= [8 16 32 64];              % Coeficientes del filtro
Wn = [0.4 0.6];               % Frecuencias de corte.

NFFT= 256;                    % Respuesta en frecuencia
L    = length(N);
H    = zeros(NFFT,L);

for i=1:L
     B = FIR1(N(i),Wn,'bandpass');  % Diseño.
     [H1,W]= freqz(B,1,NFFT);       % Respuesta.
     H(:,i)= H1;
end

% Visualizacion de la respuesta en frecuencia.
subplot(2,1,1);
plot(W/pi,20*log10(abs(H)));
xlabel('w en unidades de \pi');
ylabel('|H|');
legend('N=8','N=16','N=32','N=64');
grid;

subplot(2,1,2);
plot(W/pi,unwrap(angle(H)));
xlabel('w en unidades de \pi');
ylabel('Fase de H (rads)');
legend('N=8','N=16','N=32','N=64');
grid;
```



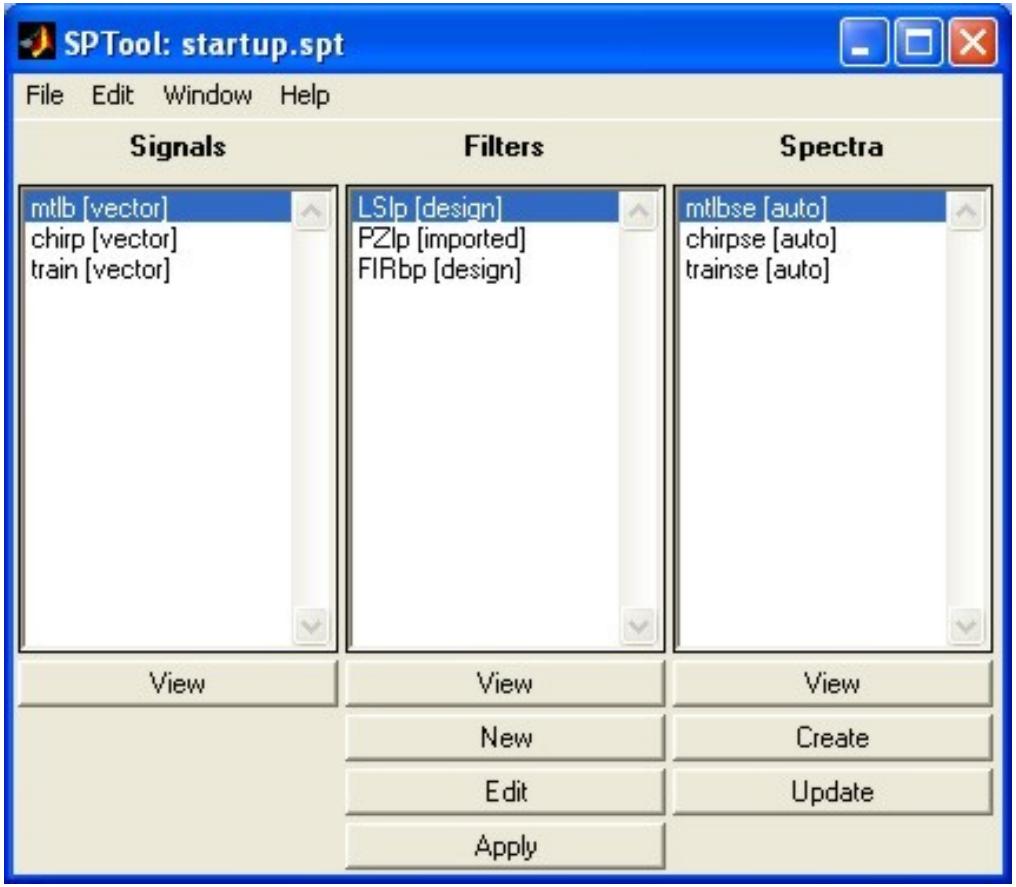


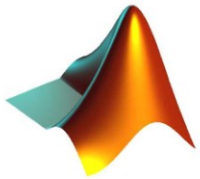
Diseño y análisis de filtros (fdatool)



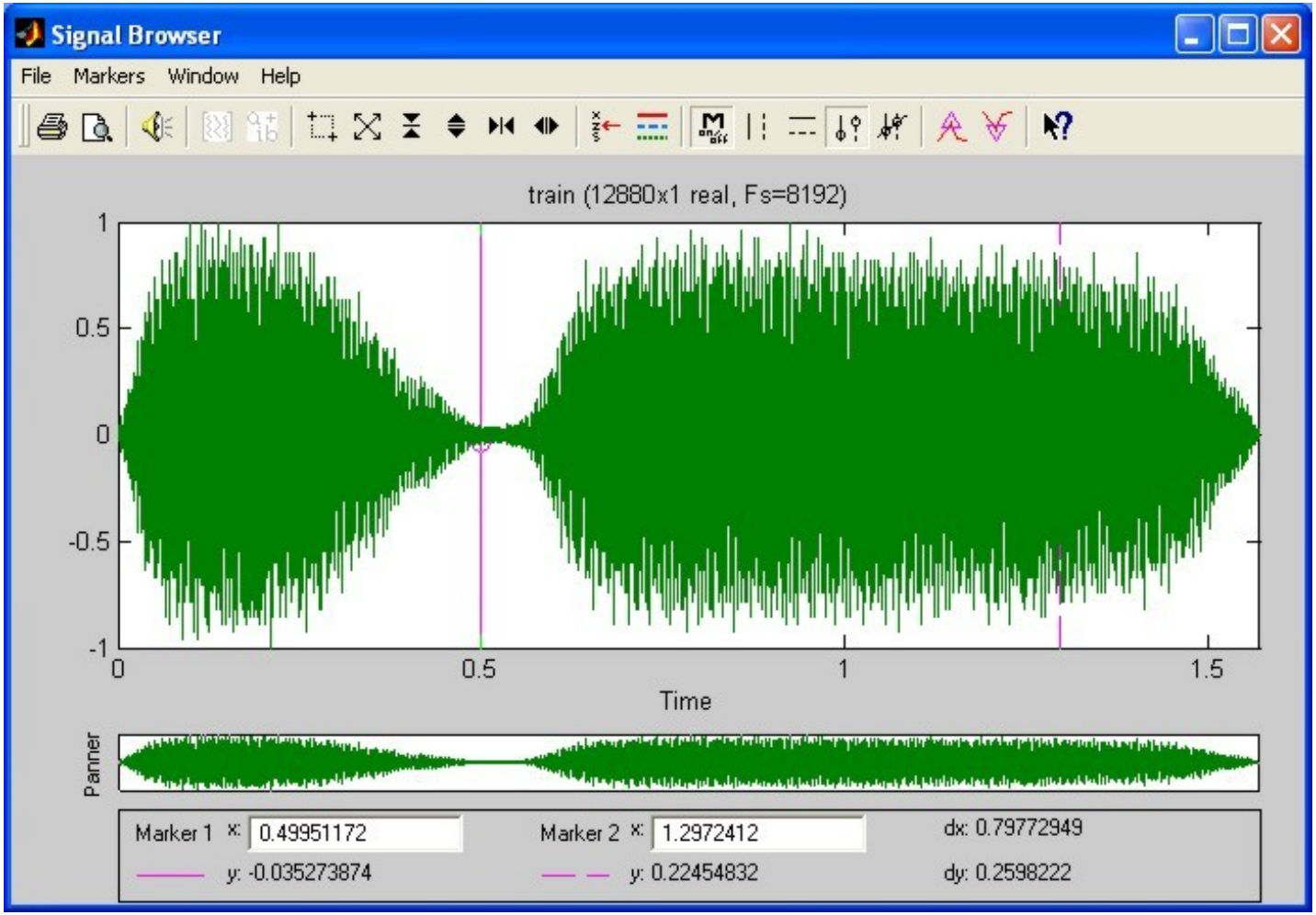


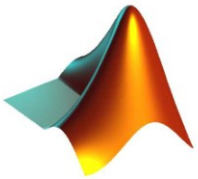
Signal Processing Tool (SPTool)



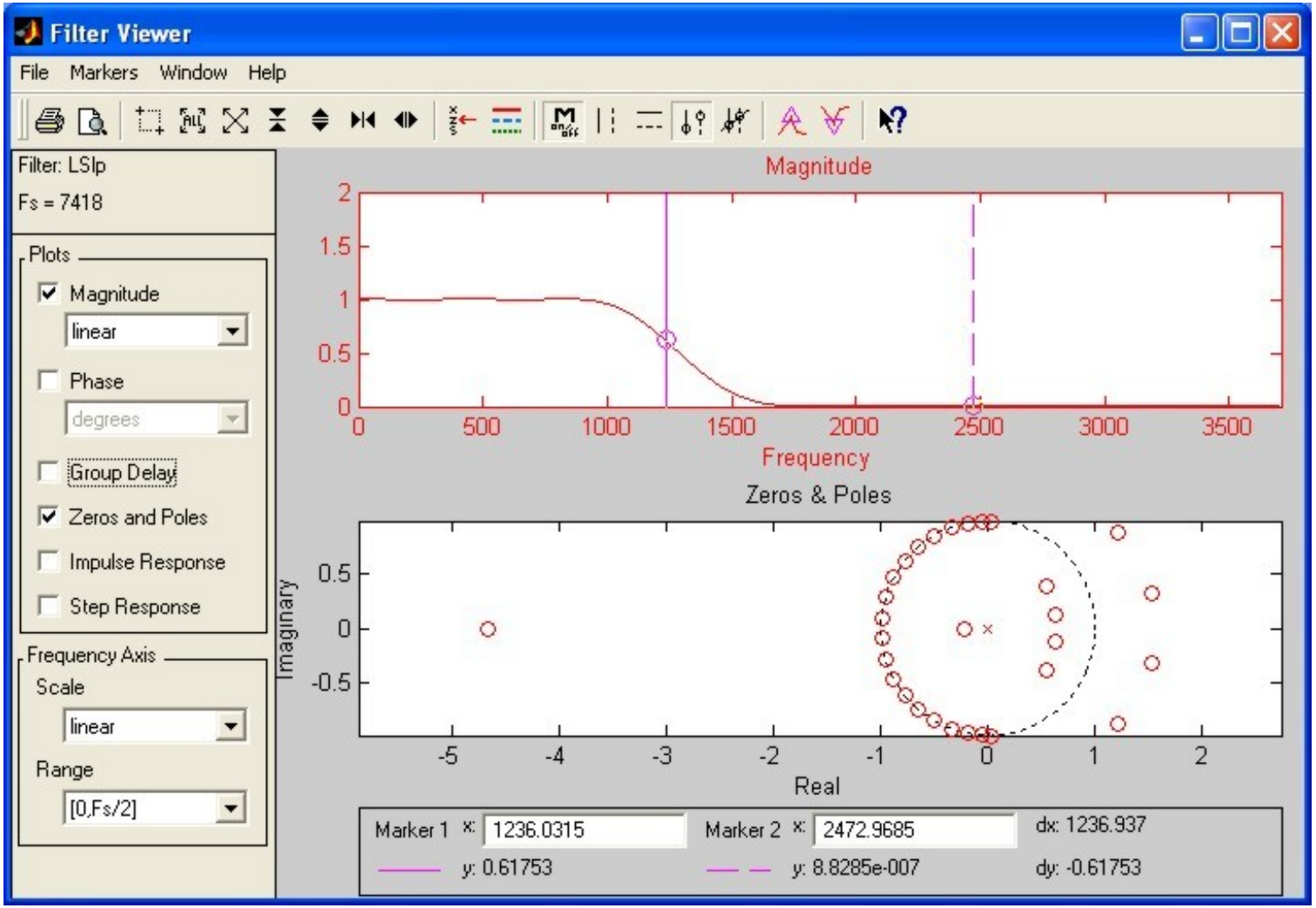


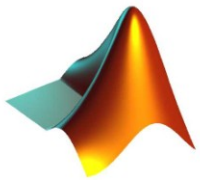
Signal Processing Tool (SPTool)



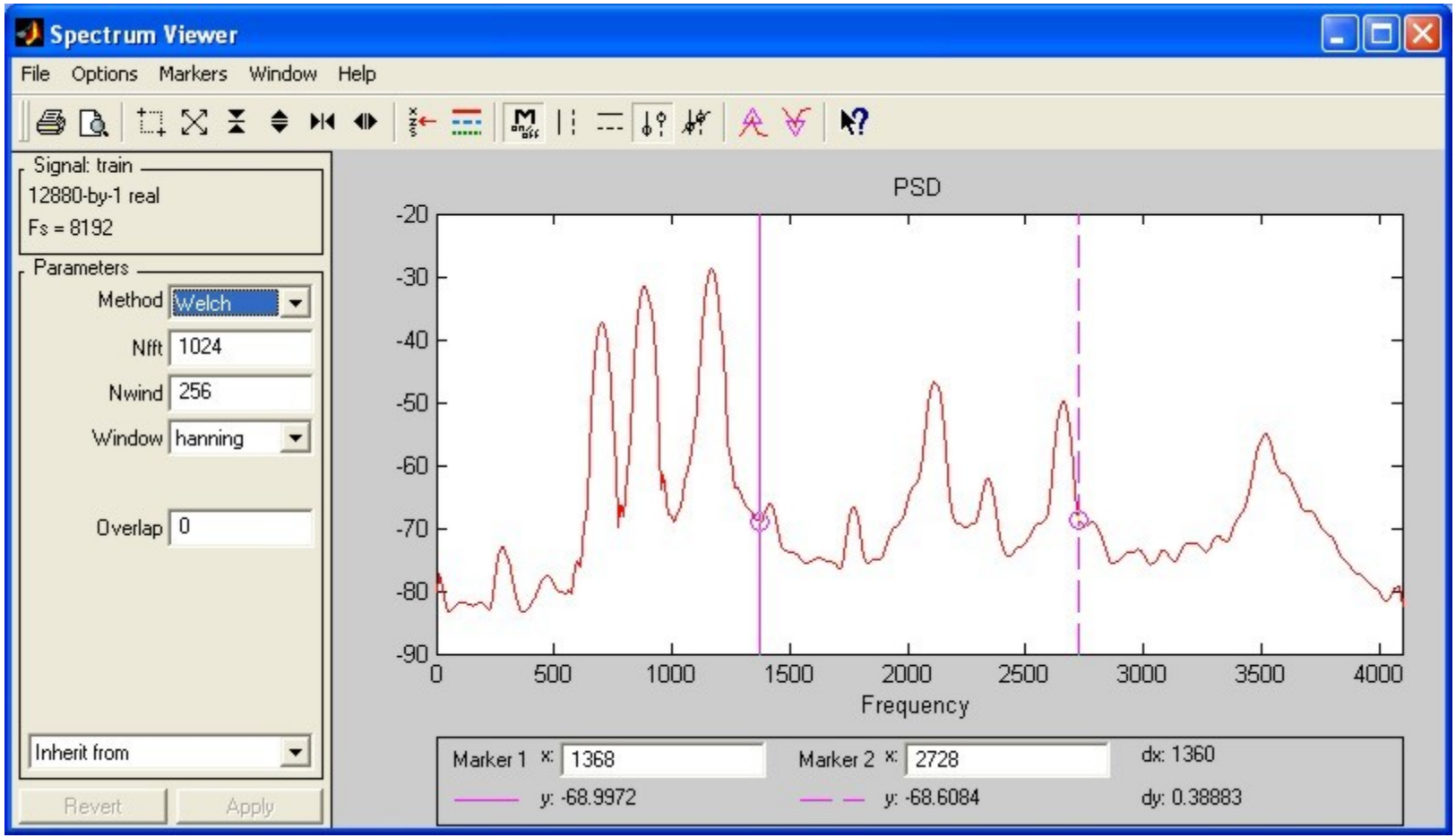


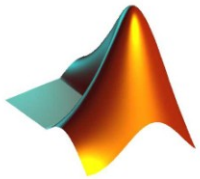
Signal Processing Tool (SPTool)





Signal Processing Tool (SPTool)

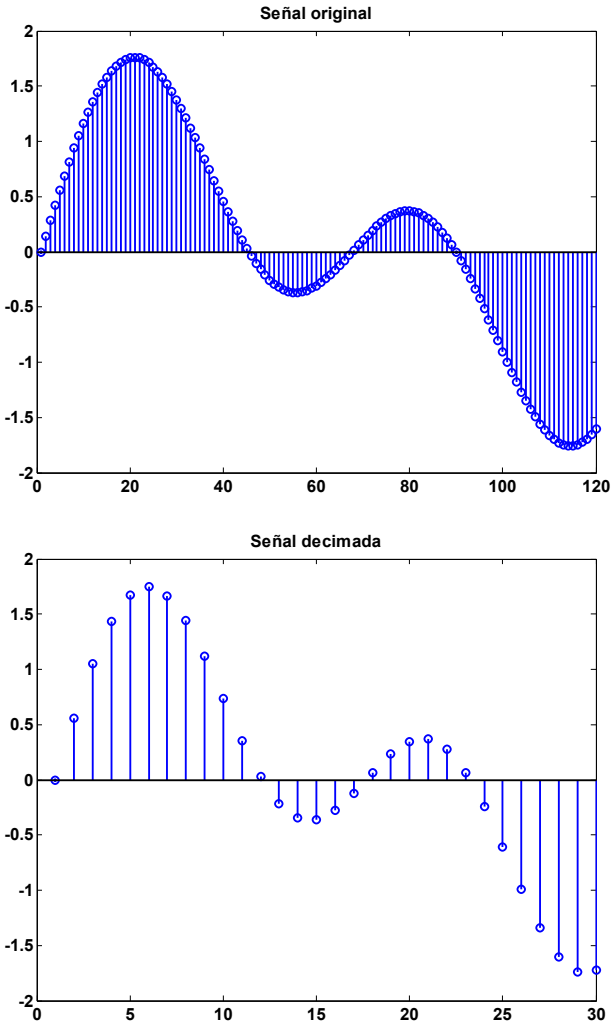


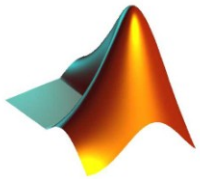


Diezmado e interpolación

- `Y = DECIMATE(X,R)`

```
Ejemplo: Decimación en un factor 4.
t = 0:.00025:1;          % Vector de tiempos
x = sin(2*pi*30*t) + sin(2*pi*60*t);
y = decimate(x,4);

stem(x(1:120));          %Original
axis([0 120 -2 2])
title('Señal original')
figure
stem(y(1:30));           %Decimada
title('Señal decimada')
```

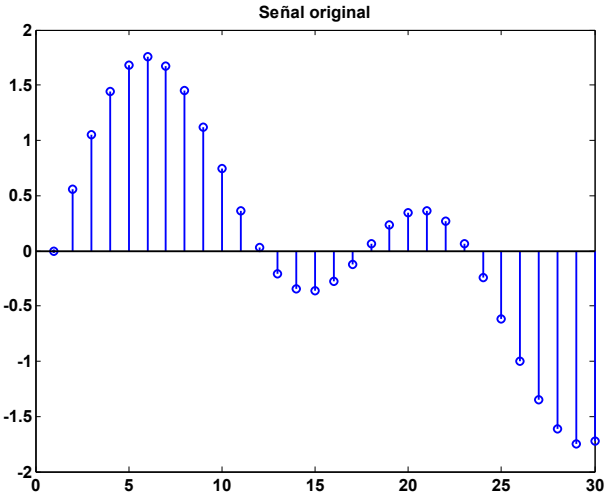


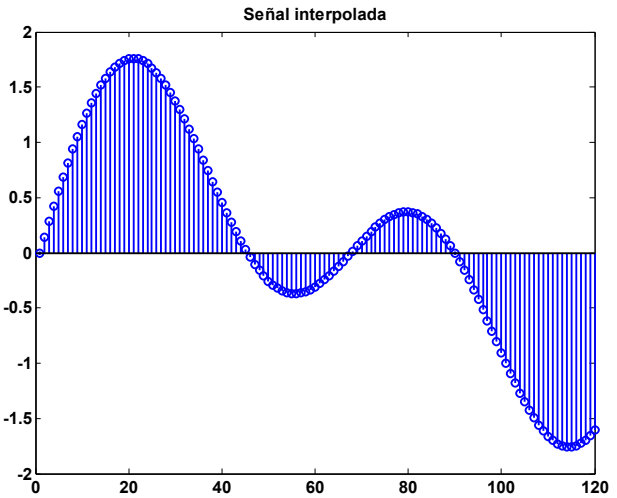


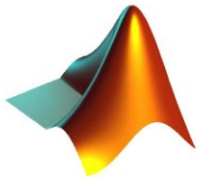
Diezmado e interpolación

- `Y = INTERP(X,R)`

```
t = 0:0.001:1; % Time vector
x = sin(2*pi*30*t) + sin(2*pi*60*t);
y = interp(x,4);
stem(x(1:30));
title('Señal original');
figure
stem(y(1:120));
title('Señal interpolada');
```







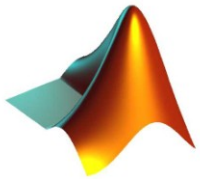
Densidad de potencia espectral

- **periodogram**

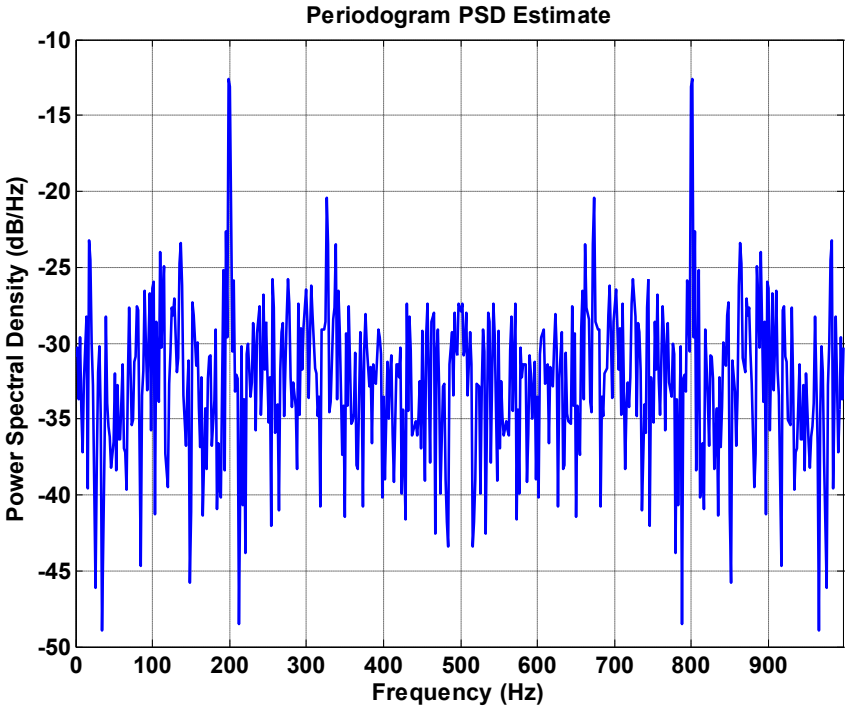
```
Fs = 1000;    t = 0:1/Fs:.3;
% Una señal coseno de 200Hz más ruido
x = cos(2*pi*t*200)+randn(size(t));
periodogram(x,[],'twosided',512,Fs);
% Se usa la ventana por defecto
```

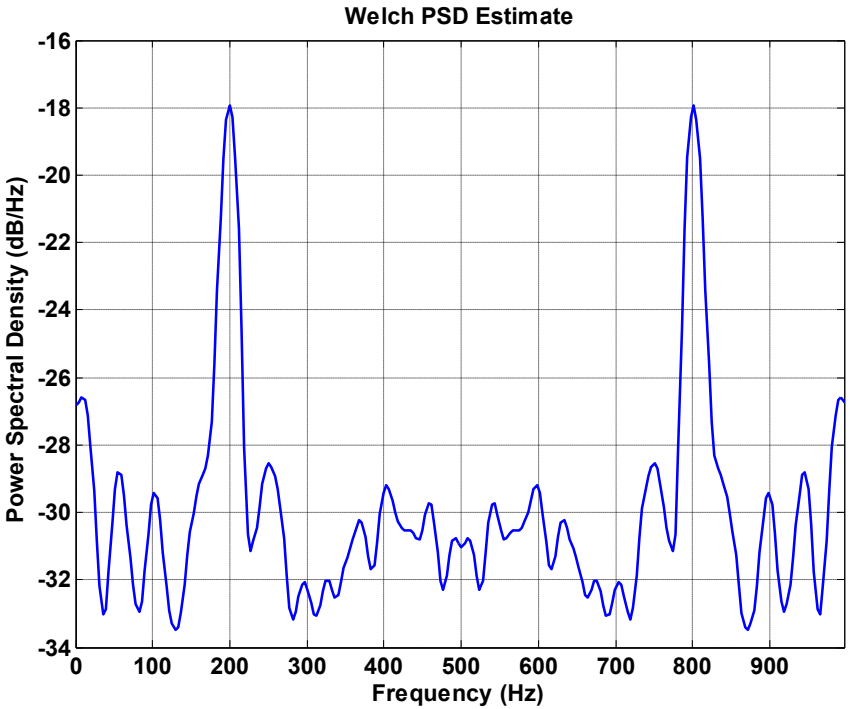
- **pwelch**

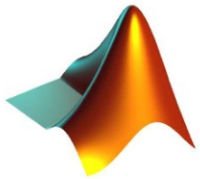
```
Fs = 1000;    t = 0:1/Fs:.296;
% Una señal coseno de 200 Hz más ruido
x = cos(2*pi*t*200)+randn(size(t));
pwelch(x,[],[],[],Fs,'twosided');
% Ventana por defecto, solapamiento y NFFT.
```



Ejemplos


Periodogram PSD Estimate
Power Spectral Density (dB/Hz)
Frequency (Hz)
-10
-15
-20
-25
-30
-35
-40
-45
-50
0
100
200
300
400
500
600
700
800
900


Welch PSD Estimate
Power Spectral Density (dB/Hz)
Frequency (Hz)
-16
-18
-20
-22
-24
-26
-28
-30
-32
-34
0
100
200
300
400
500
600
700
800
900



Estimación espectral paramétrica

- Método de covarianza

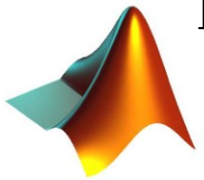
```
pcov(X,ORDER,NFFT,Fs)
```

```
randn('state',1);
x = randn(100,1);
y = filter(1,[1 1/2 1/3 1/4 1/5],x);
pcov(y,4,[],1000);
```

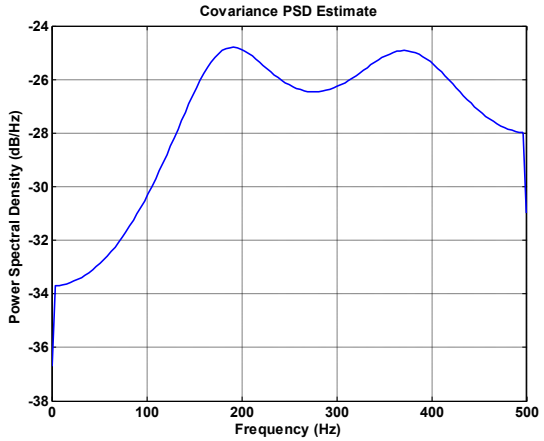
- Método de covarianza modificado

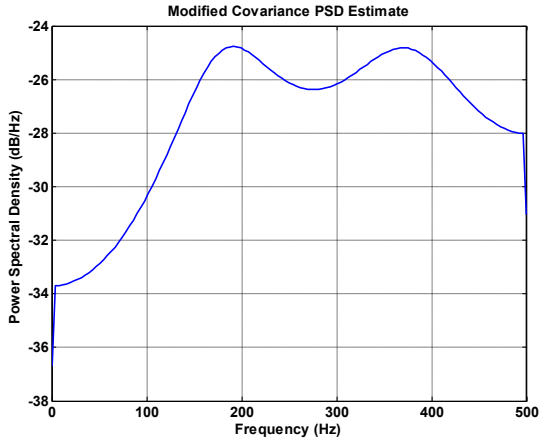
```
pmcov(X,ORDER,NFFT,Fs)
```

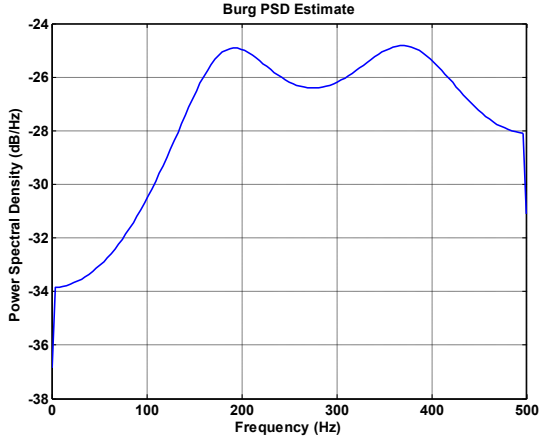
```
randn('state',1);
x = randn(100,1);
y = filter(1,[1 1/2 1/3 1/4 1/5],x);
pmcov(y,4,[],1000);
```

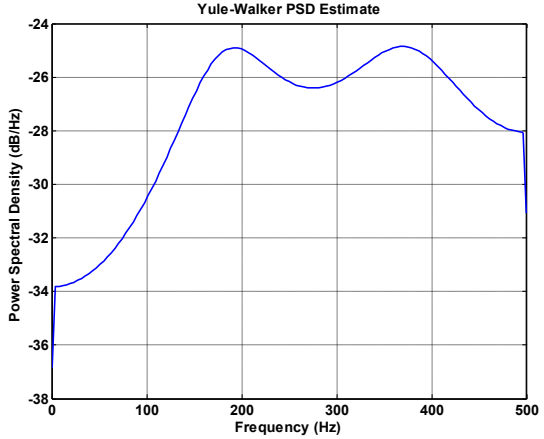


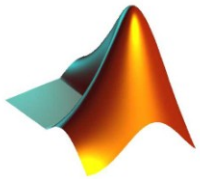
Ejemplos


Covariance PSD Estimate
Power Spectral Density (dB/Hz)
Frequency (Hz)
-24
-26
-28
-30
-32
-34
-36
-38
0
100
200
300
400
500


Modified Covariance PSD Estimate
Power Spectral Density (dB/Hz)
Frequency (Hz)
-24
-26
-28
-30
-32
-34
-36
-38
0
100
200
300
400
500

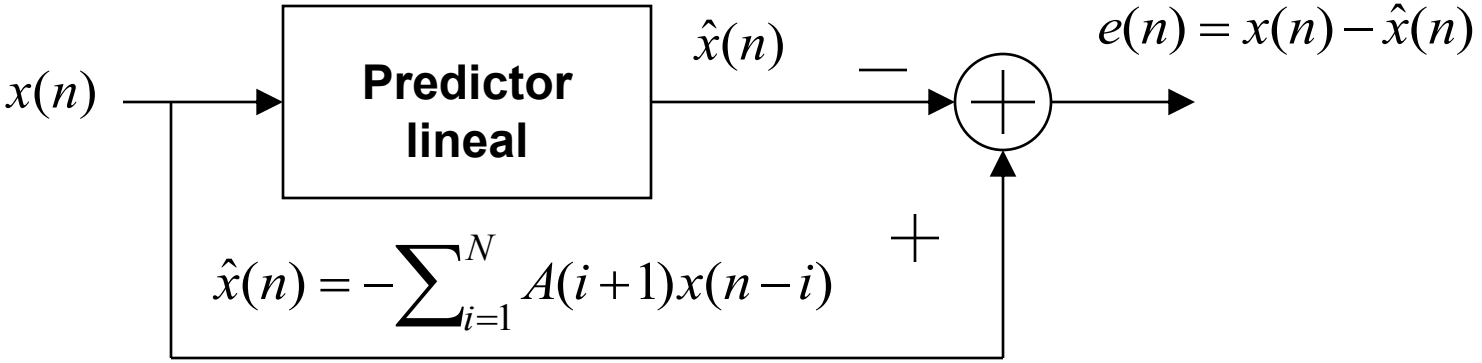

Burg PSD Estimate
Power Spectral Density (dB/Hz)
Frequency (Hz)
-24
-26
-28
-30
-32
-34
-36
-38
0
100
200
300
400
500


Yule-Walker PSD Estimate
Power Spectral Density (dB/Hz)
Frequency (Hz)
-24
-26
-28
-30
-32
-34
-36
-38
0
100
200
300
400
500



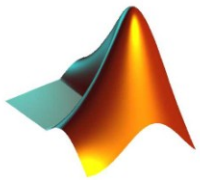
Modelado. Predicción lineal

- Predictor lineal:
  - Estima la muestra siguiente utilizando *N* muestras anteriores.



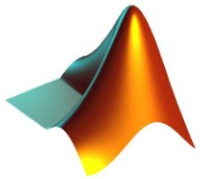
  - `[A,E] = LPC(X,N)`
    - Calcula:
      - Los coeficientes del predictor, A.
      - La varianza del error, E.

Minimizando $J = E\{|e(n)|^2\}$

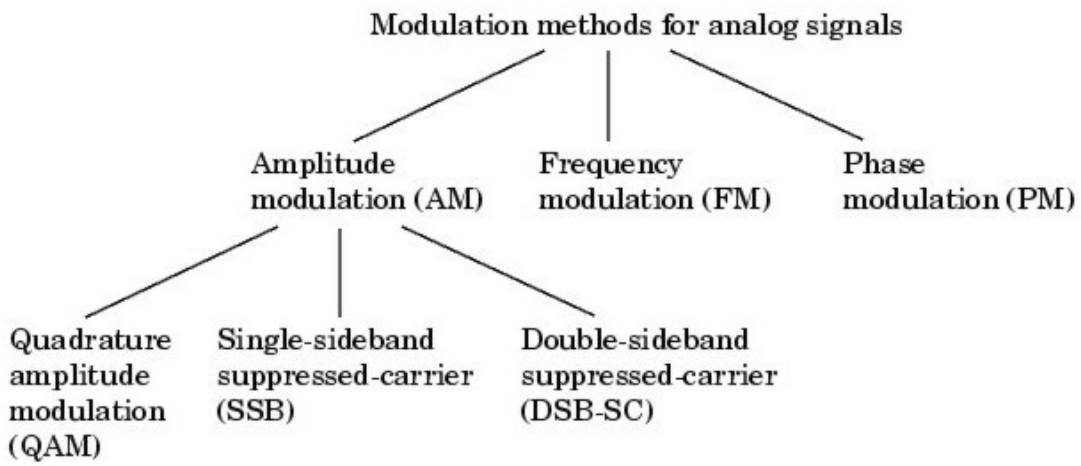


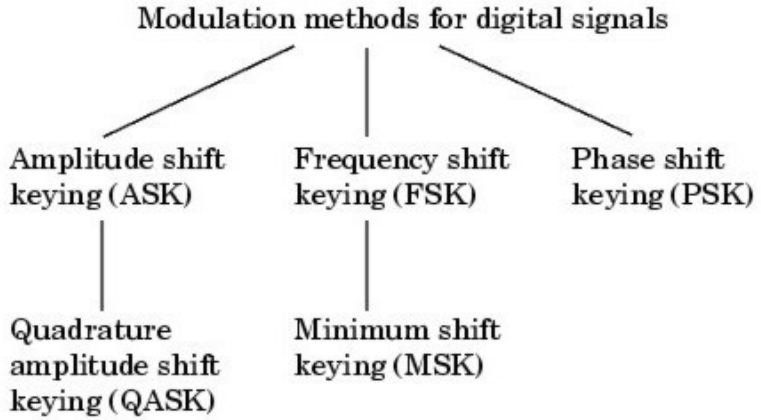
Comunicaciones

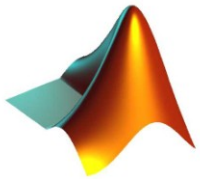
- Funciones principales:
  - Generación de señales aleatorias
  - Análisis de errores
  - Codificación de la fuente (escalar, diferencial)
  - Codificación para el control de errores (convolucional, codificación lineal de bloques)
  - Modulación y demodulación (analógica y digital)
  - Filtrado mediante filtros especiales
  - Aritmética en cuerpos de Galois



Modulación analógica y digital

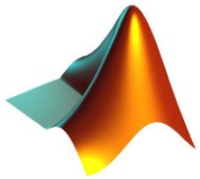




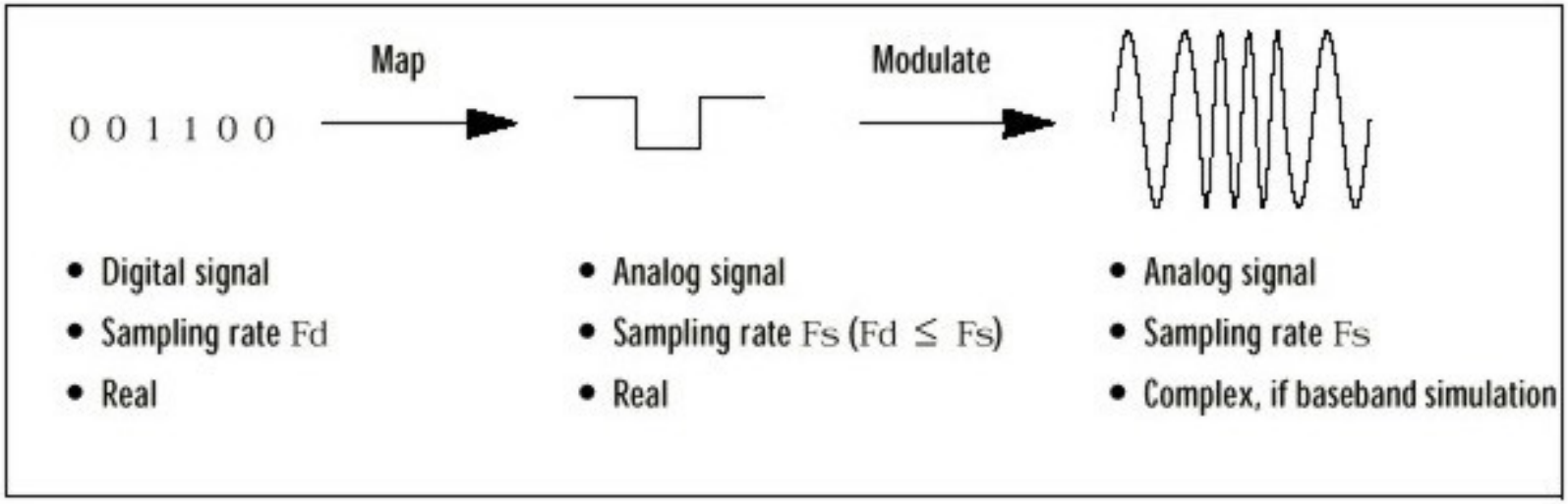


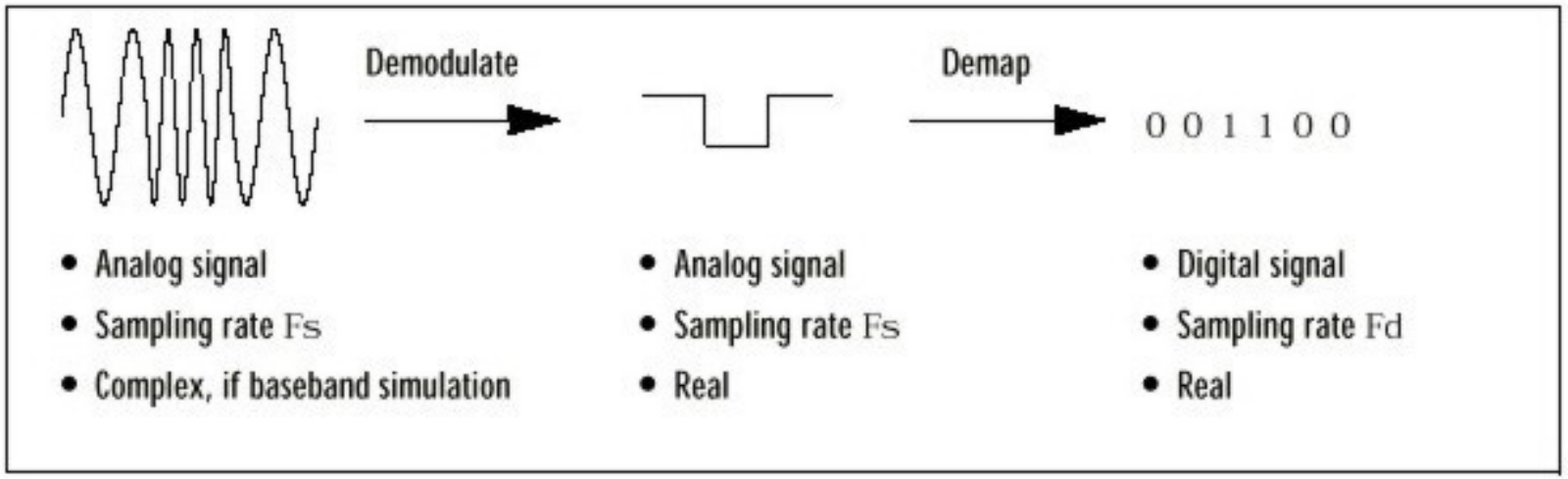
Funciones utilizadas frecuentemente

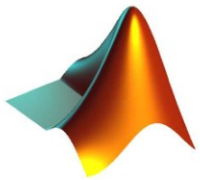
- `randint`
  - Genera una matriz de números aleatorios enteros uniformemente distribuidos.
    - `out = randint;` `% Un solo valor [0,1]`
    - `out = randint(N);` `% Matriz cuadrada N×N`
    - `out = randint(M,N);` `% Matriz M×N`
    - `out = randint(M,N,rg);` `% Se especifica el rango.`



Modulación/demodulación digital







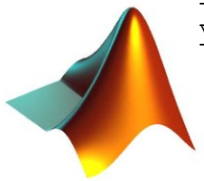
Modulación/demodulación digital

- "Mapping"+modulación/demodulación
  - dmodce/ddemodce

```
y = dmodce(x,Fd,Fs,'method/nomap'...);
y = dmodce(x,Fd,Fs,'ask',M);
y = dmodce(x,Fd,Fs,'fsk',M,tone);
y = dmodce(x,Fd,Fs,'msk');
y = dmodce(x,Fd,Fs,'psk',M);
y = dmodce(x,Fd,Fs,'qask',M);
y = dmodce(x,Fd,Fs,'qask/arb',inphase,quadr);
y = dmodce(x,Fd,Fs,'qask/cir',numsig,amp,phs);
y = dmodce(x,Fd,[Fs initphase],...);
```

- Sólo "mapping"
  - modmap/demodmap

```
modmap('method',...);
y = modmap(x,Fd,Fs,'ask',M);
y = modmap(x,Fd,Fs,'fsk',M,tone);
y = modmap(x,Fd,Fs,'msk');
y = modmap(x,Fd,Fs,'psk',M);
y = modmap(x,Fd,Fs,'qask',M);
y = modmap(x,Fd,Fs,'qask/arb',inphase,quadr);
y = modmap(x,Fd,Fs,'qask/cir',numsig,amp,phs);
```



Ejemplo

```
M = 16;                    % Modulación M-aria.
Fd = 1;                    % Se muestrea el mensaje original
                           % a una muestra por segundo.
Fs = 3;                    % La señal modulada se muestrea
                           % a una frecuencia de 3 muestras por segundo.
x = randint(100,1,M);      % Mensaje digital aleatorio.

% Modulación M-ary PSK
y = dmodce(x,Fd,Fs,'psk',M);

% Adición de ruido Gaussiano.
ynoisy = y + .04*randn(300,1) + .04*j*randn(300,1);

% Diagrama de dispersión a partir de las observaciones ruidosas.
scatterplot(ynoisy,1,0,'b.');

% Demodulación para recuperar el mensaje
z = ddemodce(ynoisy,Fd,Fs,'psk',M);
s = symerr(x,z)            % Comprobar la tasa de error de los símbolos.
```

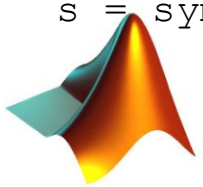
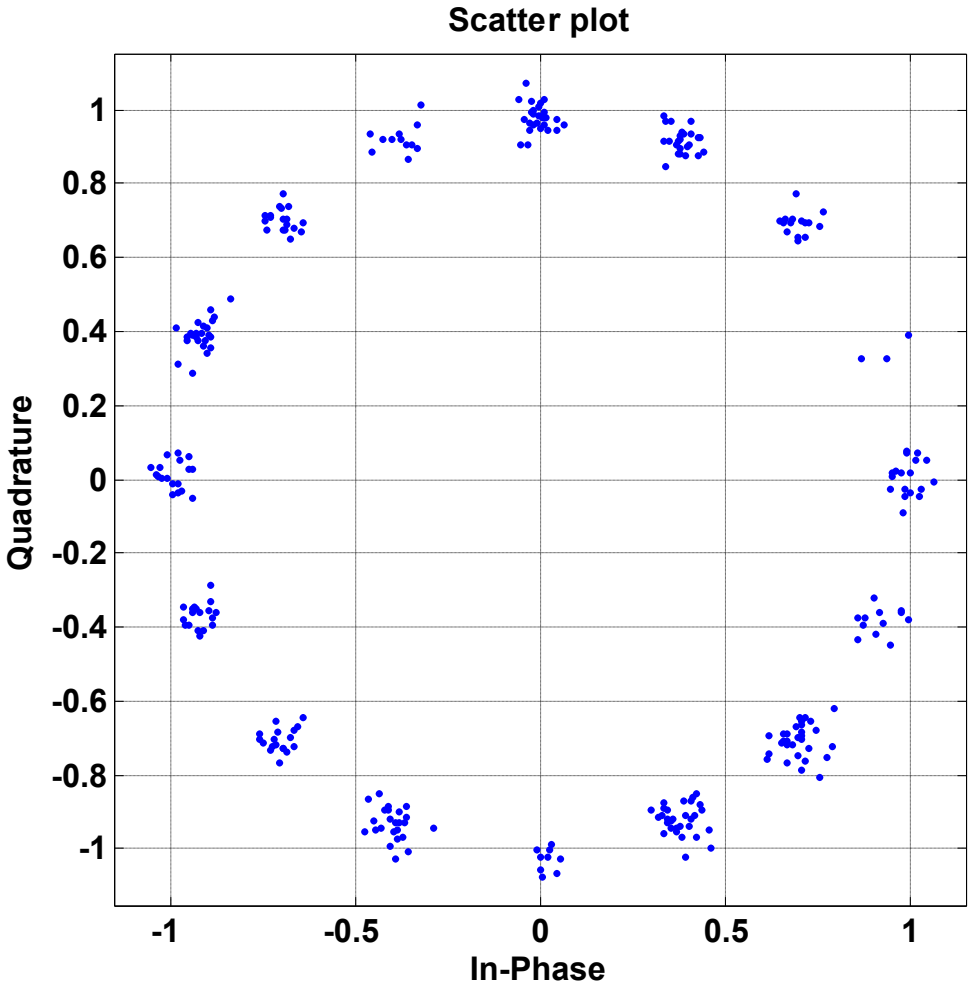
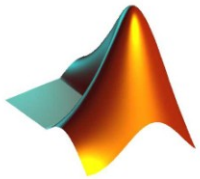


Gráfico de dispersión (ruido)

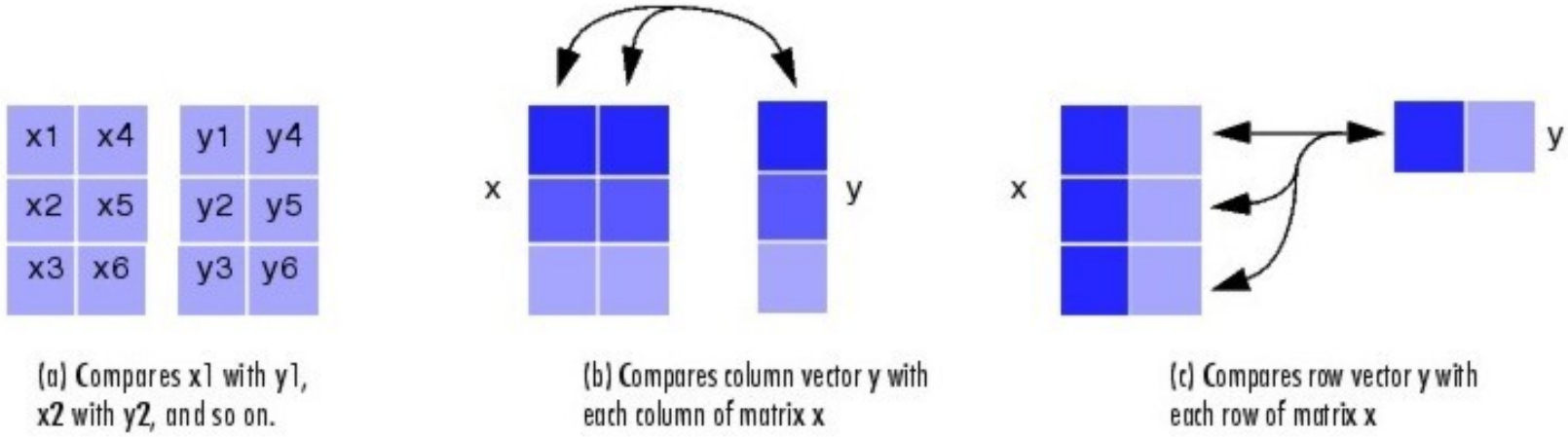




Análisis de errores

- `biterr`
  - Calcula el número de bits erróneos y la tasa de error.

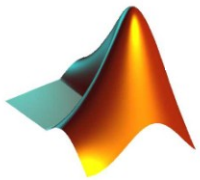
```
[number,ratio] = biterr(x,y);
[number,ratio] = biterr(x,y,k);
[number,ratio] = biterr(...,flg);
[number,ratio,individual] = biterr(...)
```



(a) Compares x1 with y1, x2 with y2, and so on.

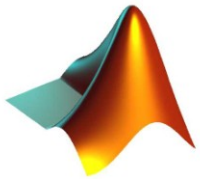
(b) Compares column vector y with each column of matrix x

(c) Compares row vector y with each row of matrix x



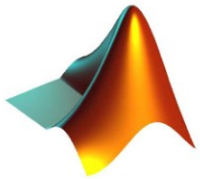
Ejemplo

```
x = randint(100,100,4);    % Señal original

% Introducción de errores.
% Los errores pueden ser 1, 2, o 3 (no 0).
% Colocación de los errores
errorplace = (rand(100,100) > .9);
errorvalue = randint(100,100,[1,3]); % Valor error
errors = errorplace.*errorvalue;
y = rem(x+errors,4); % Señal y error sumadas mod 4

% Análisis de errores
format short
[num_bit,ratio_bit] = biterr(x,y,2)
[num_sym,ratio_sym] = symerr(x,y)
```



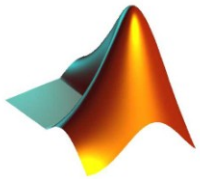
Ejercicio:

- Estudiar mediante Matlab/Simulink el efecto del ruido en los sistemas de comunicación digital.
  - Construir el diagrama de bloques de simulación.
  - Simular el sistema:
    - Diferentes esquemas de modulación (ASK, PSK, FSK, MSK).
    - Obtener las curvas de error en función de la SNR.
    - Adicionalmente, considérese PSK con *M*= 4 y 8 símbolos y compárese las curvas de error frente a la SNR.

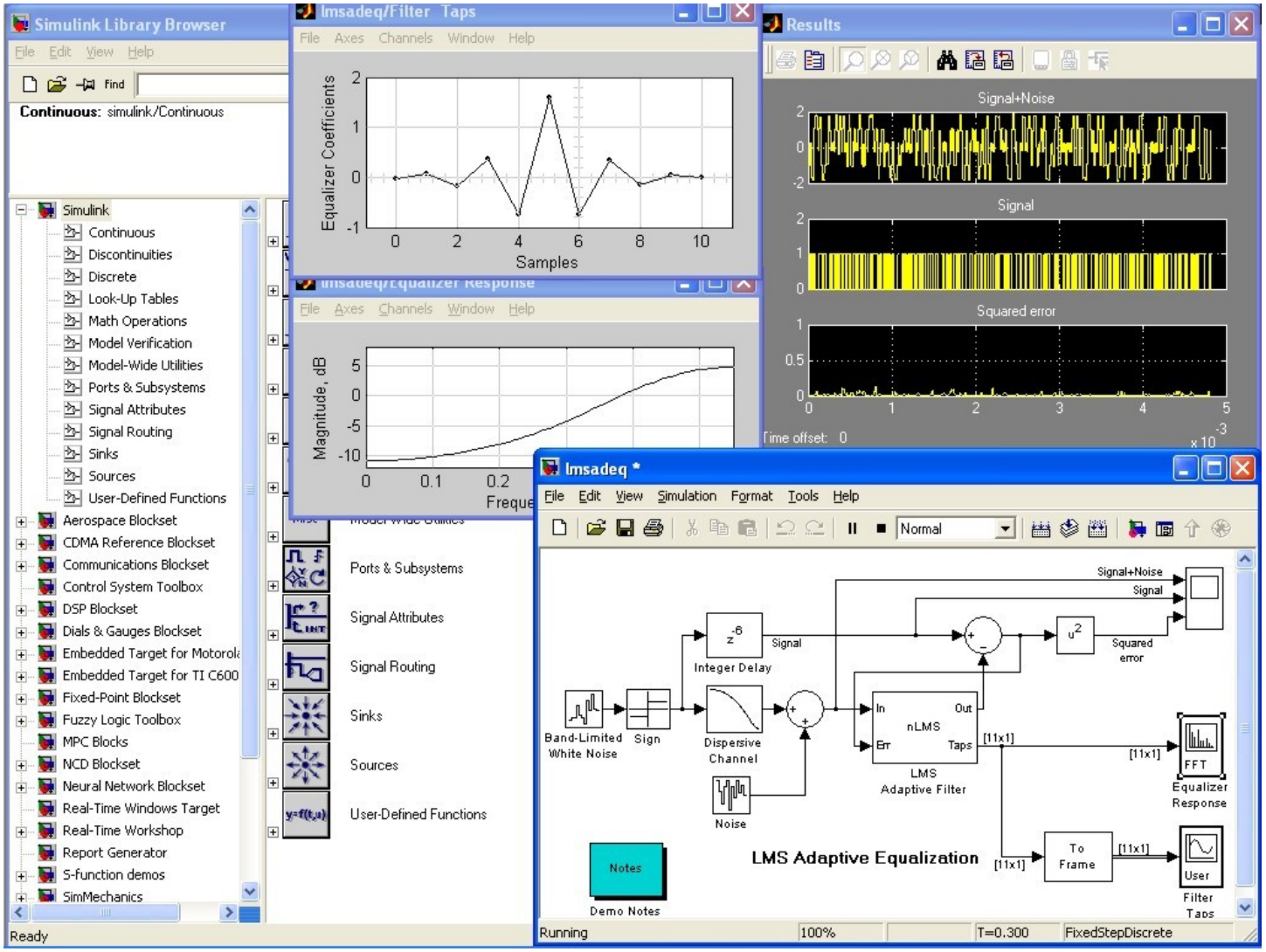


Introducción a Simulink

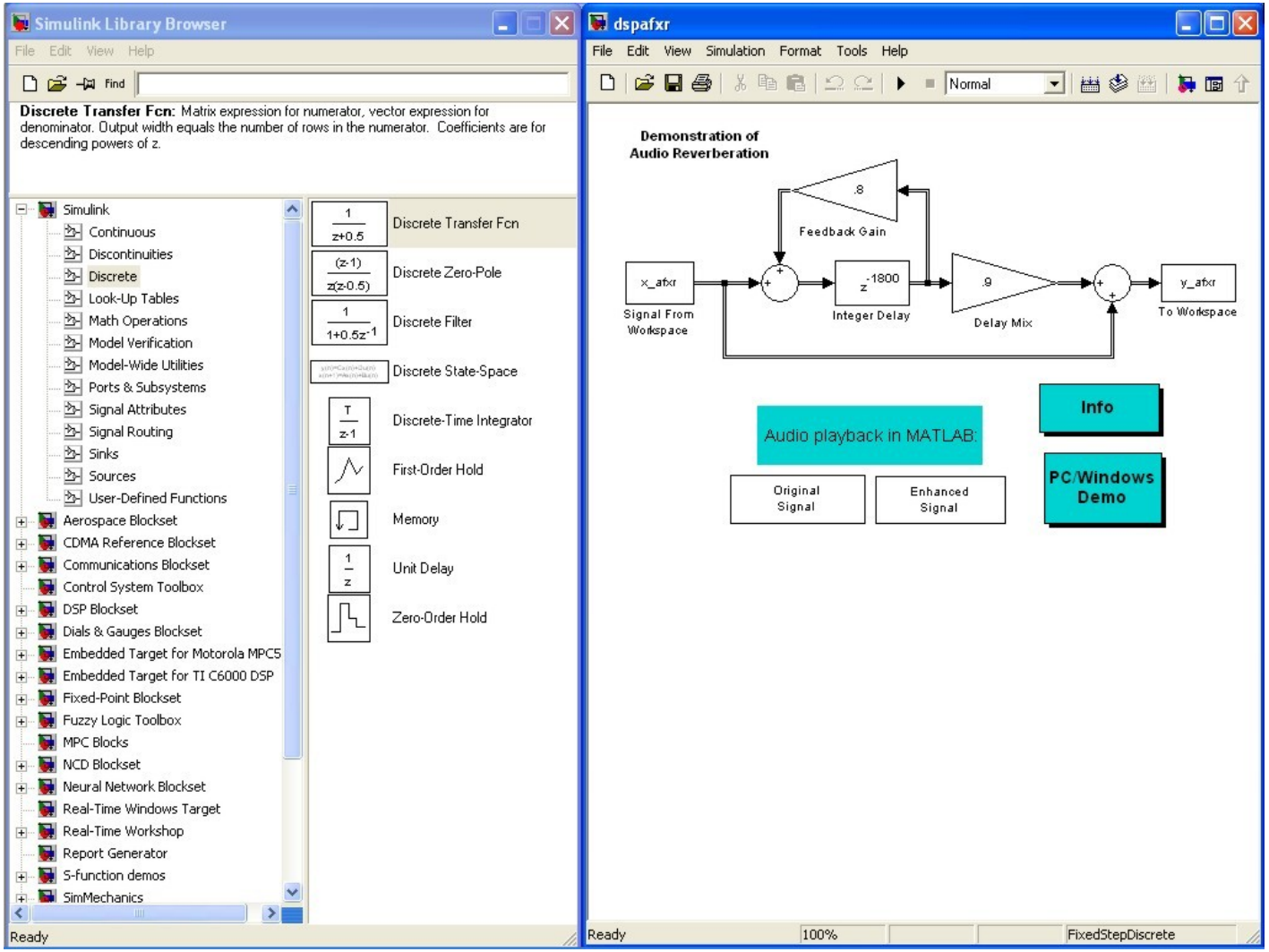
- Simulink es una herramienta para modelado, simulación y análisis de sistemas dinámicos.
- Soporta tanto sistemas lineales como no lineales:
  - en tiempo continuo,
  - muestreados,
  - híbridos y
  - sistemas multifrecuencia (contienen sistemas muestreados a diferente frecuencia).

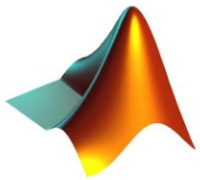


Entorno de trabajo

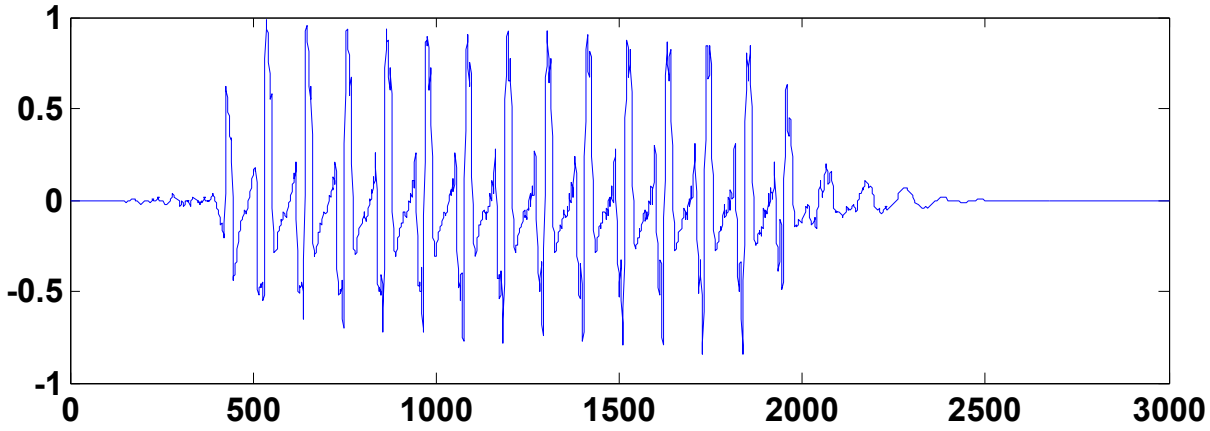


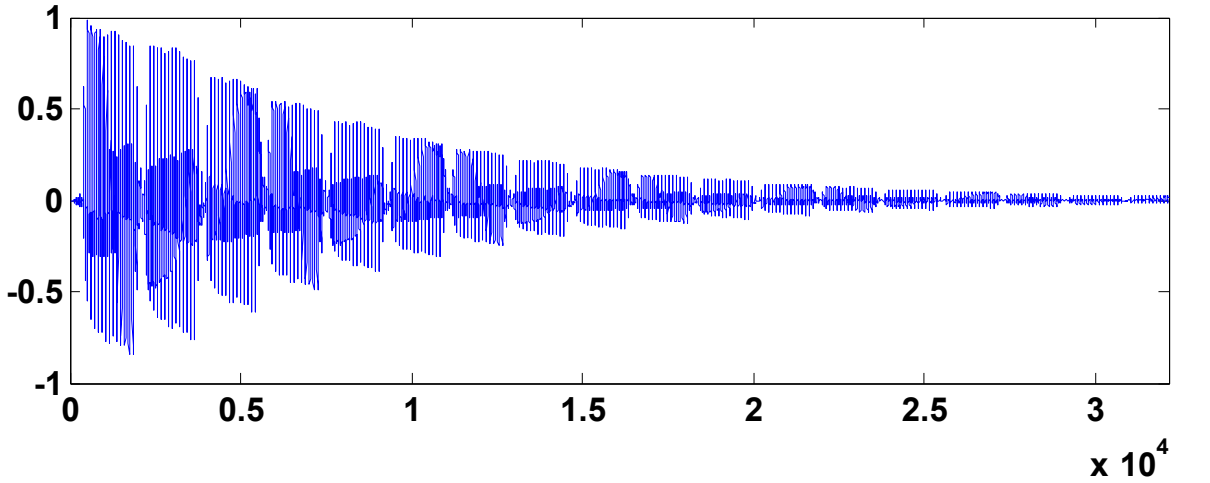
Construcción del diagrama de bloques

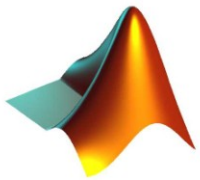




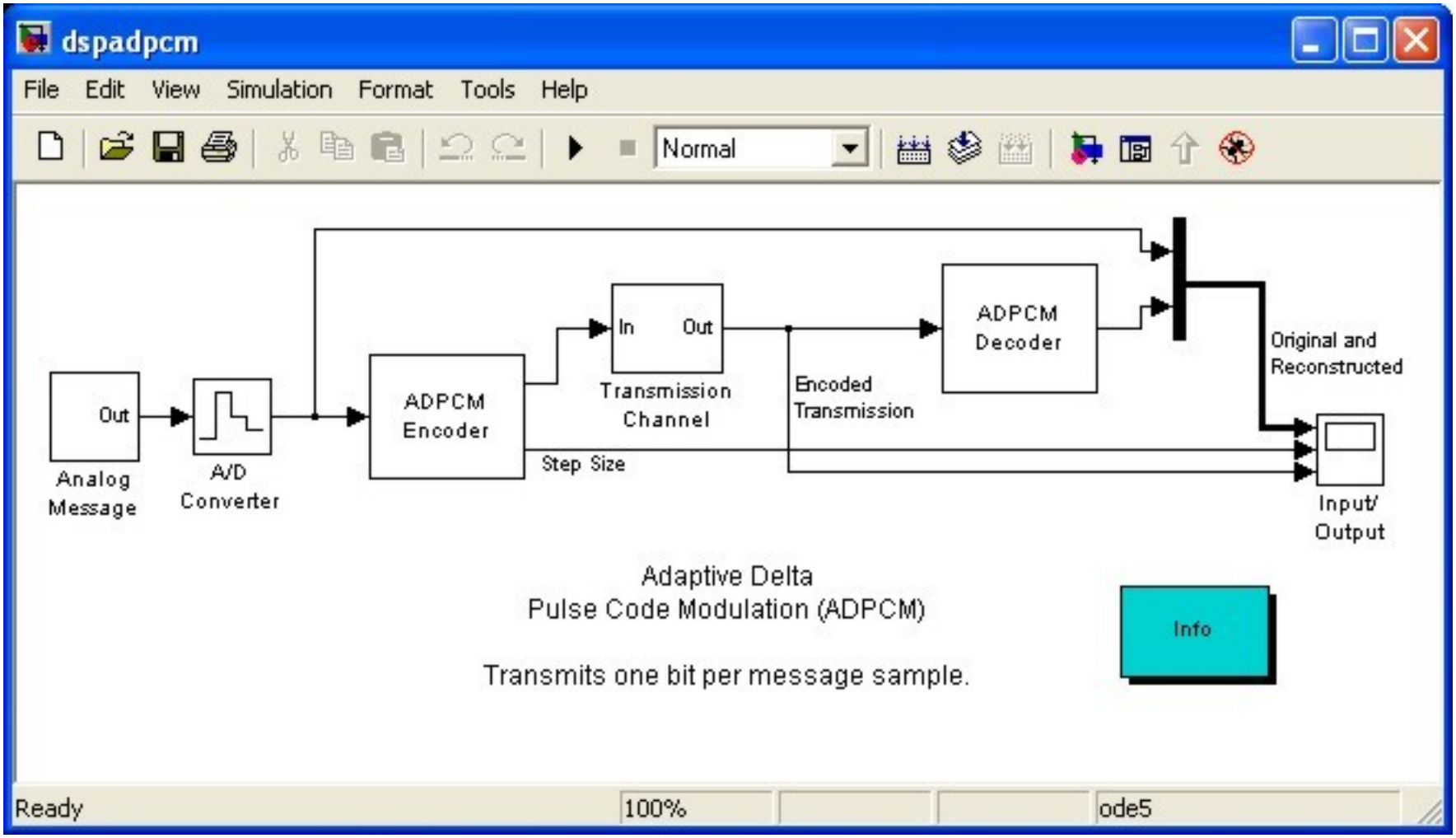
Resultado de la simulación

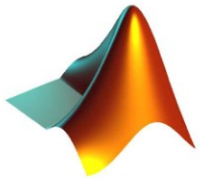




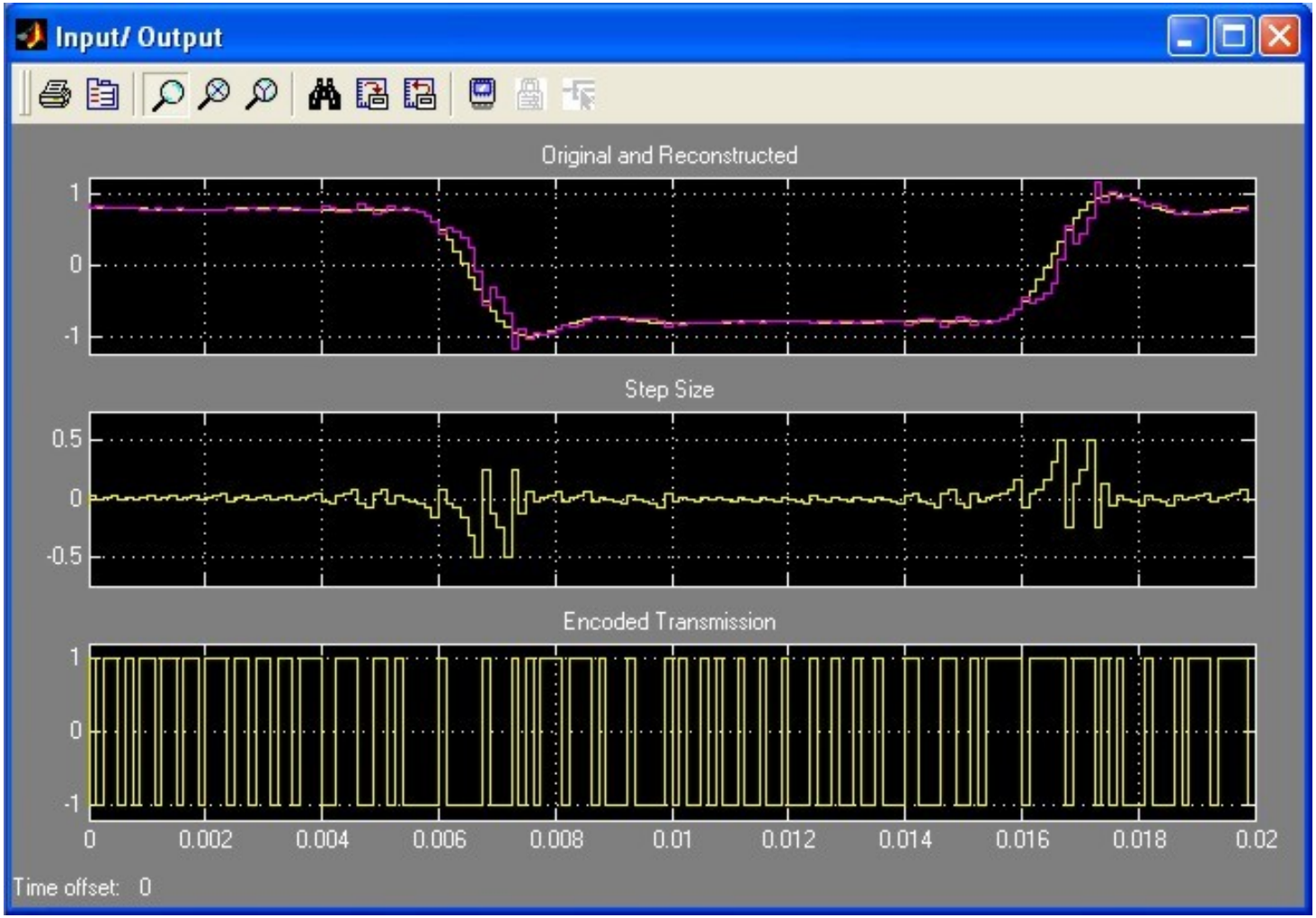


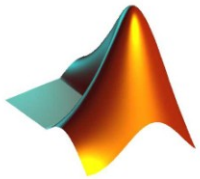
Ejemplo


dspadpcm
File Edit View Simulation Format Tools Help
Normal
Out
Analog Message
A/D Converter
ADPCM Encoder
In Out
Transmission Channel
Step Size
Encoded Transmission
ADPCM Decoder
Original and Reconstructed
Input/ Output
Adaptive Delta
Pulse Code Modulation (ADPCM)
Transmits one bit per message sample.
Info
Ready
100%
ode5

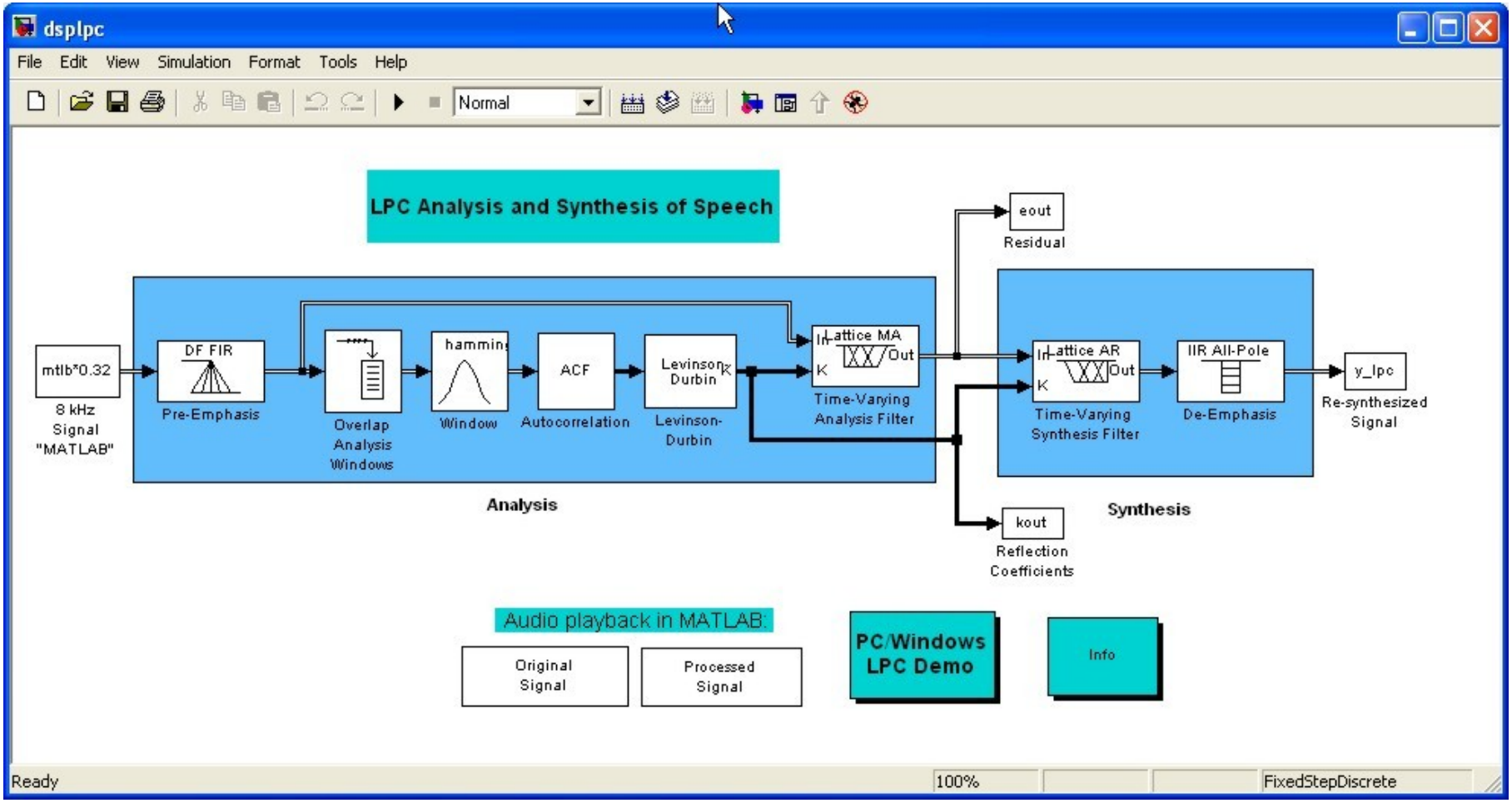


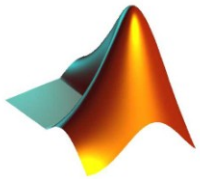
Simulación


Input/ Output
Original and Reconstructed
1
0
-1
Step Size
0.5
0
-0.5
Encoded Transmission
1
0
-1
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02
Time offset: 0

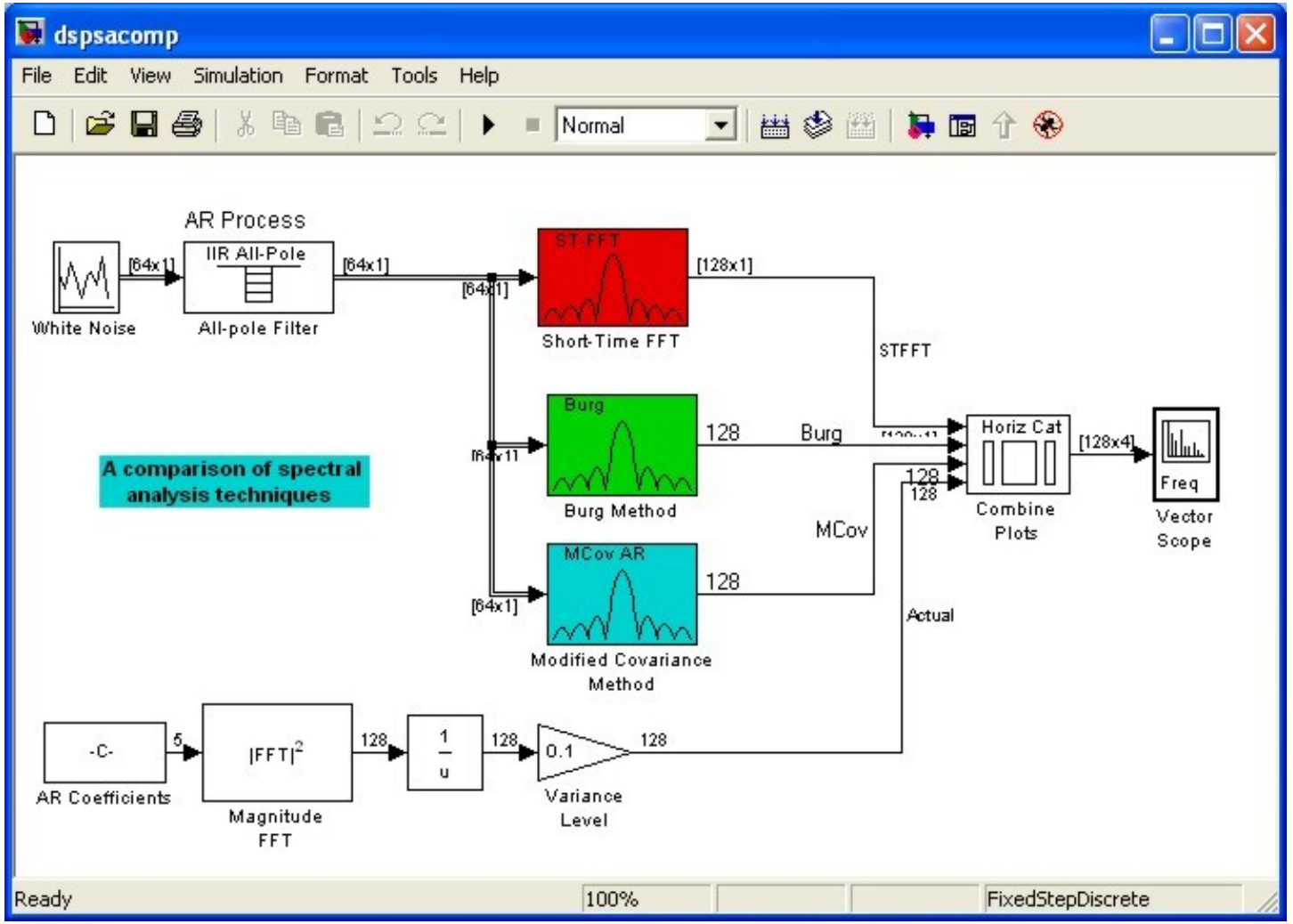


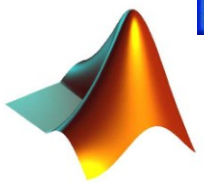
Análisis/síntesis LPC de la señal de voz



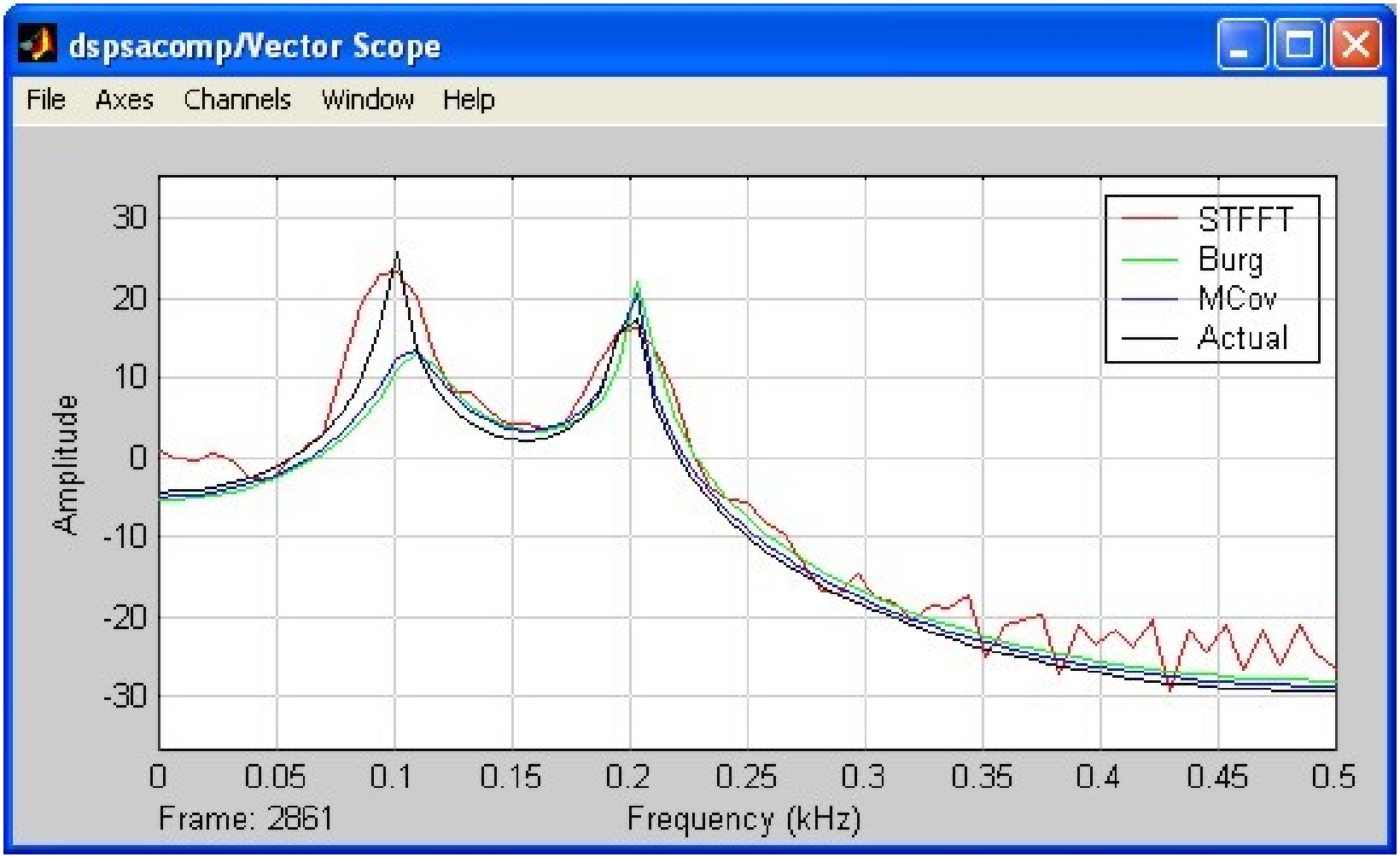


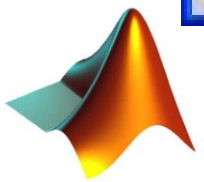
Estimación espectral





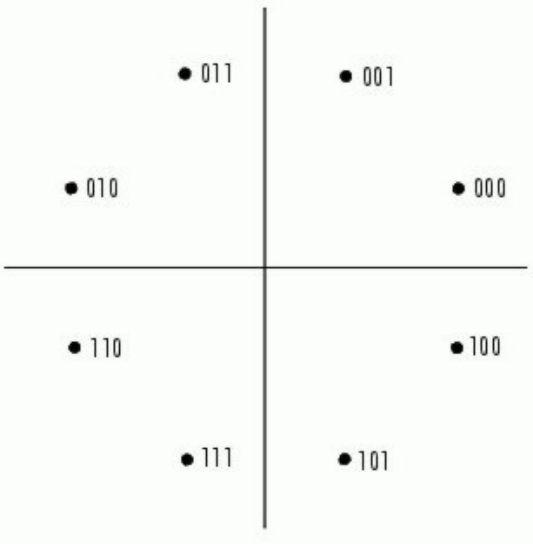
Resultado de la simulación

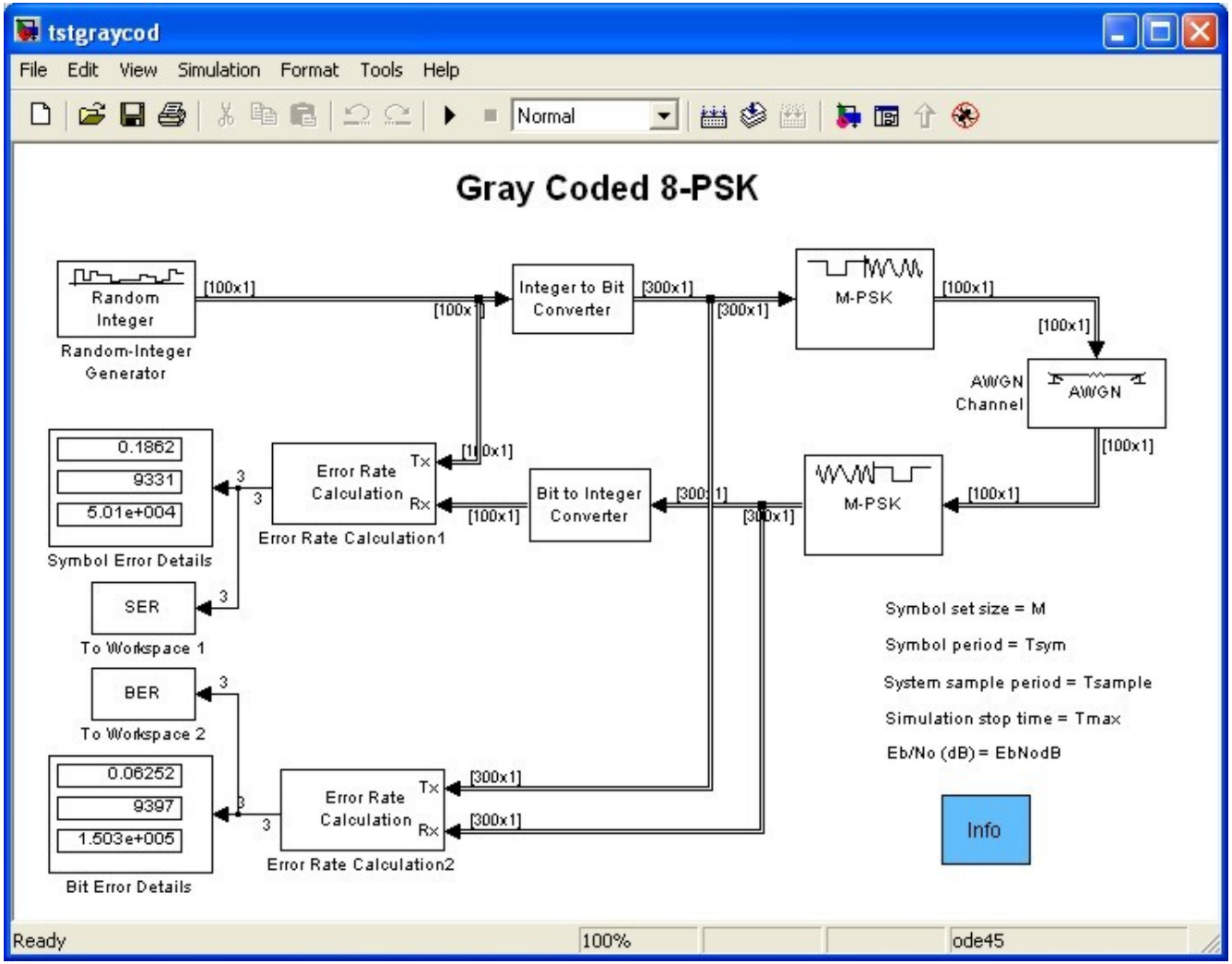


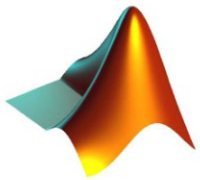


Sistema de comunicación digital

8-PSK (Gray)



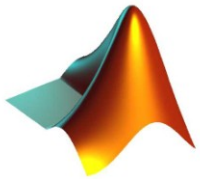




Procesado de imágenes

- Lectura de imágenes:
  - `A = imread(filename, fmt)`
    - Lee una imagen en escala de grises o en color
    - `fmt` especifica el formato de imagen (BMP, JPEG, PNG, TIFF, etc)
  - `[X, map] = imread(...)`
    - `map` Mapa de color

- Visualización de la imagen `imshow`
  - `imshow(I)`
  - `imshow(I,[low high])`
  - `imshow(RGB)`
  - `imshow(BW)`
  - `imshow(X,map)`



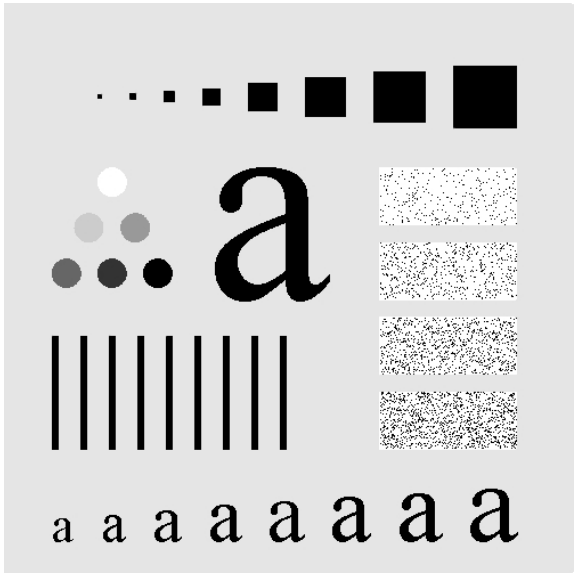
Lectura de imágenes

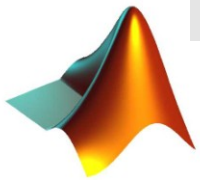
- `A= imread('texto.tif');`
- `imshow(A);`
- `size(A); size(A);`

  `500×500`

- `[B map]= imread('flor.tif');`
- `imshow(B); size(A);`

  `500×500×3`

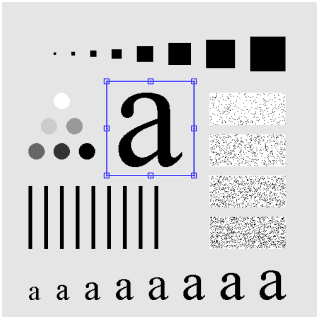


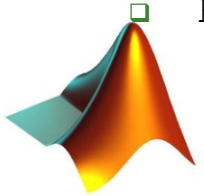


Transformaciones espaciales

- Modificación del tamaño
  - `B = imresize(A, scale)`
  - `B = imresize(A, [mrows ncols])`
- Rotar una imagen
  - `B = imrotate(A,angle)`
  - `B = imrotate(A,angle,method)`
    - `method -> Interpolación {'nearest'}, 'bilinear', 'bicubic'`
  - `B = imrotate(A,angle,method,bbox)`
    - `bbox -> Bounding box {'crop'} 'loose'`
- Recortar una imagen
  - `B= imcrop(A);`
  - `B= imcrop(A,rect);`

Herramienta interactiva

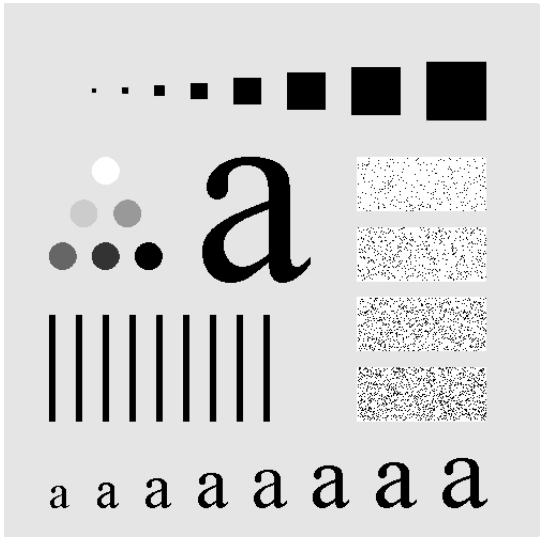




Filtrado 2D

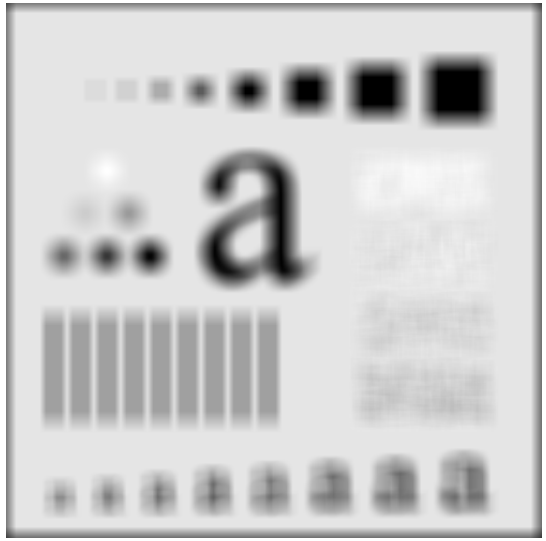
- `B= imfilter(A,h)`

- `N= 5;`
- `h = ones(N,N) / (N*N);`
- `Af = imfilter(A,h);`

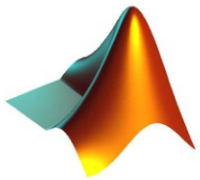


Original

Filtrada N= 5

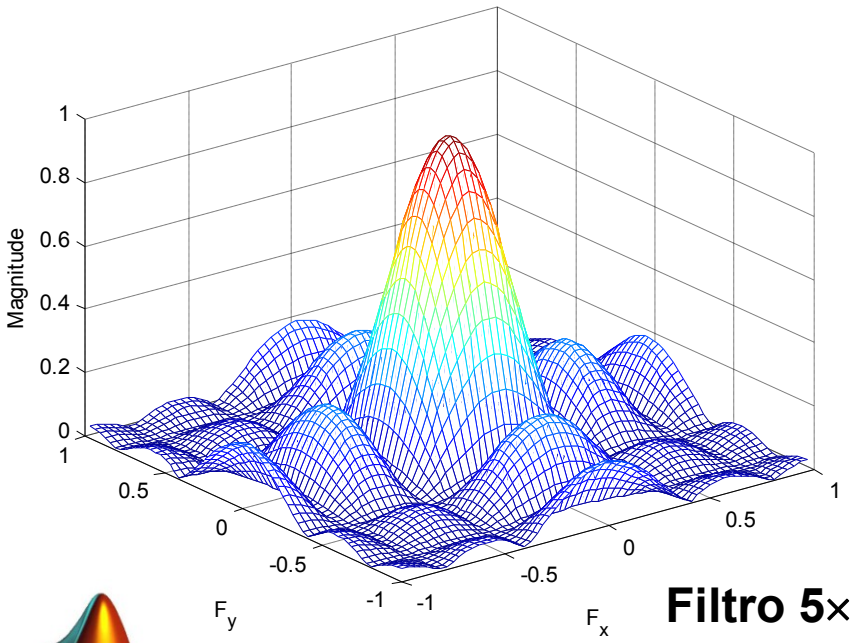


Filtrada N= 20

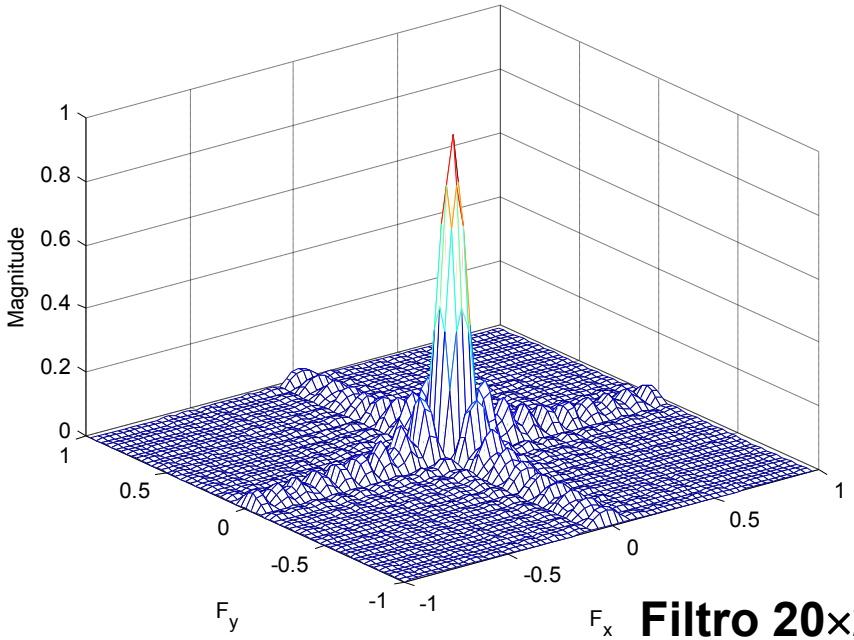


Respuesta en frecuencia de filtros 2D

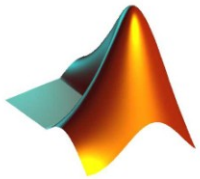
- `[H, f1, f2] = freqz2(h, n1, n2)`
- `[H, f1, f2] = freqz2(h, [n2 n1])`
- `[H, f1, f2] = freqz2(h)`
- `[H, f1, f2] = freqz2(h, f1, f2)`



Filtro 5×5



Filtro 20×20

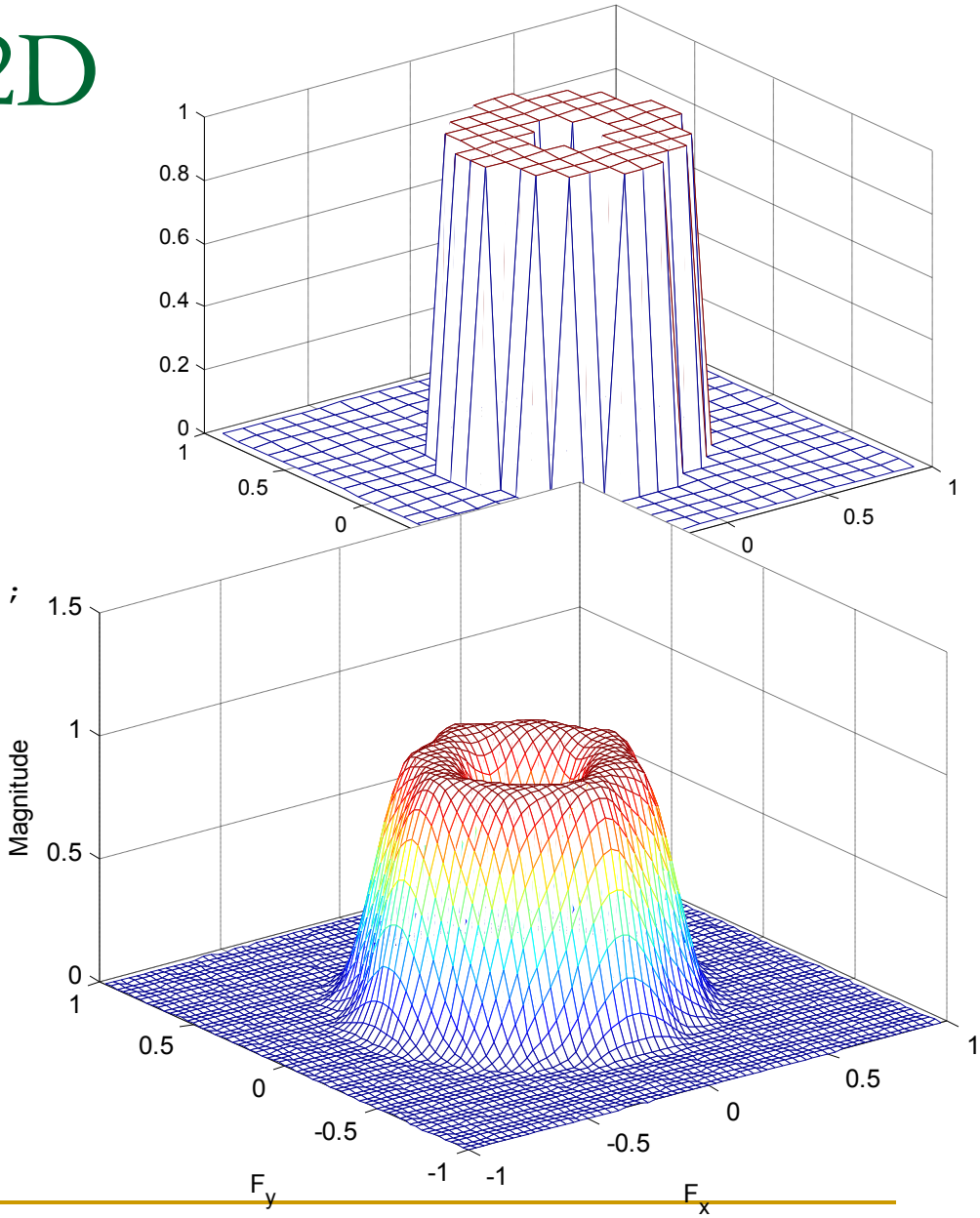


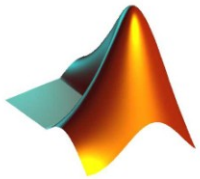
Diseño de filtros 2D

- `h = fwind1(Hd, win)`
- `h = fwind1(Hd, win1, win2)`
- `h = fwind1(f1, f2, Hd,...)`

- Ejemplo:
  - `[f1,f2] = freqspace(21,'meshgrid');`
  - `Hd = ones(21);`
  - `r = sqrt(f1.^2 + f2.^2);`
  - `Hd((r<0.1)|(r>0.5)) = 0;`
  - `colormap(jet(64));`
  - `mesh(f1,f2,Hd);`

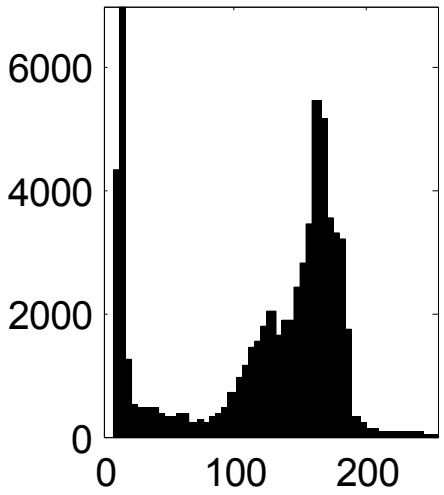
  - `h = fwind1(Hd,hamming(21));`
  - `freqz2(h);`

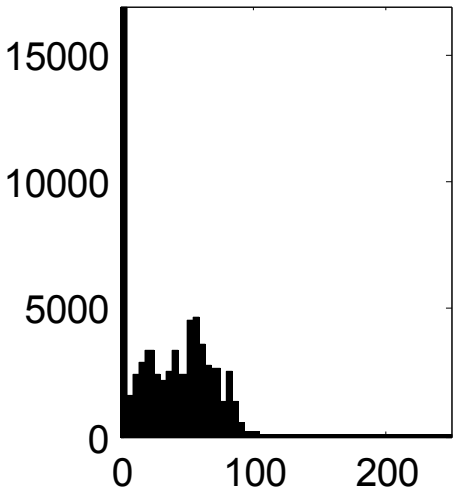


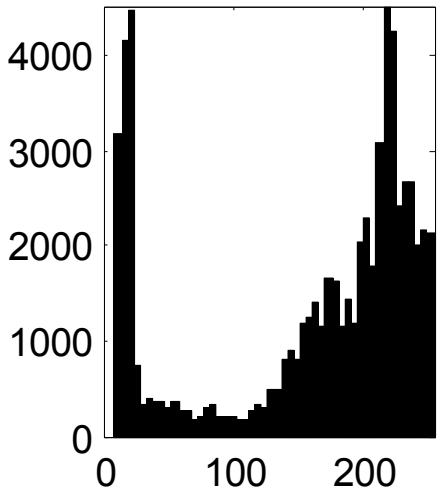


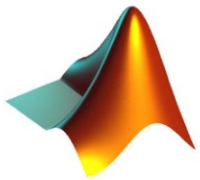
Ajuste del nivel de intensidad





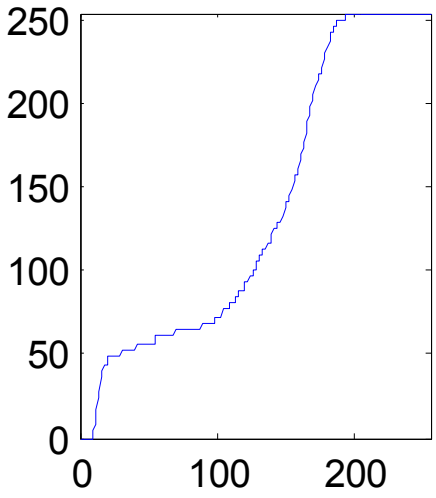


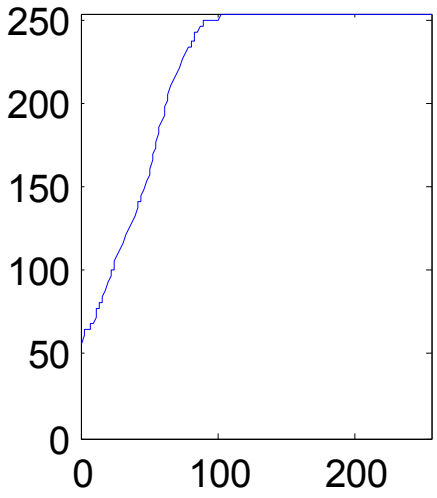


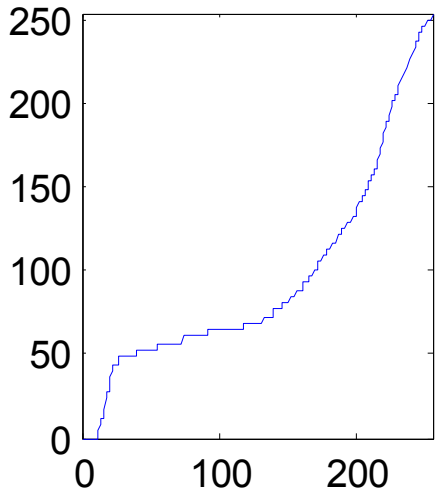


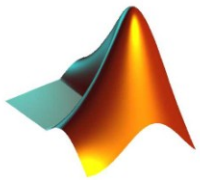
Ecualización del histograma





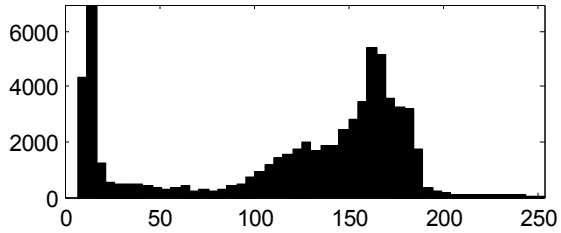


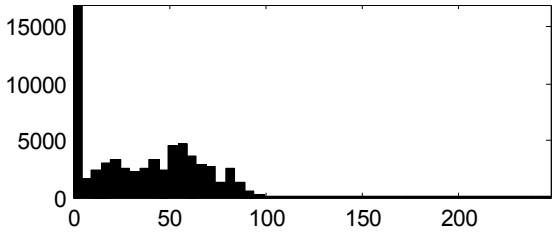


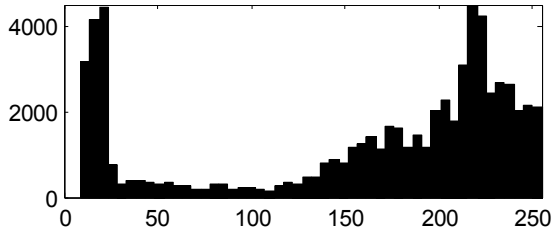


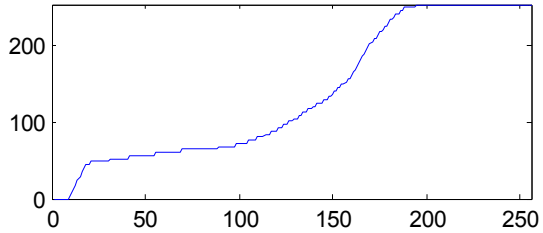
Ecualización del histograma

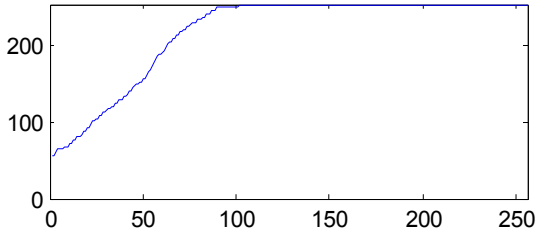


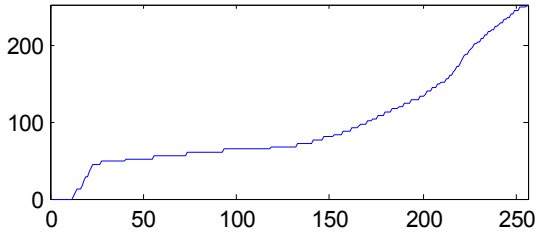


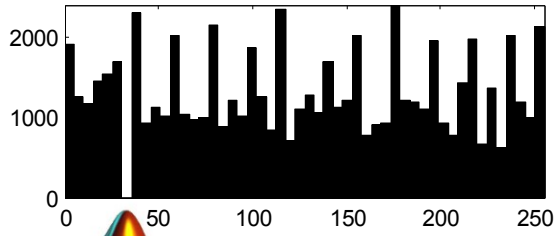


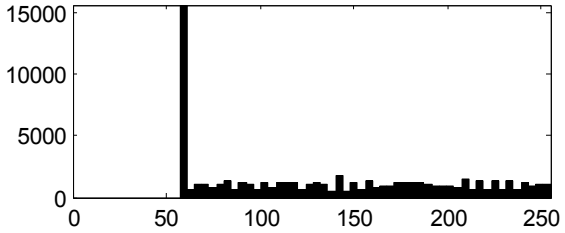


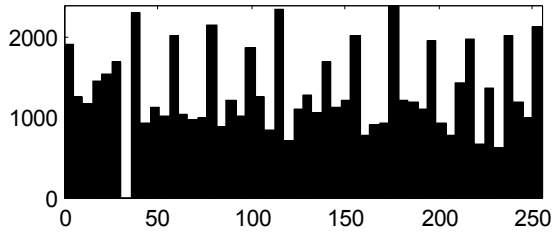


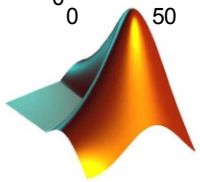












Filtrado de ruido (Filtro de Wiener)

- `J = wiener2(I, [m n], noise)`
- `[J, noise] = wiener2(I, [m n])`
  - Wiener2 estima la media y la varianza entorno a cada pixel

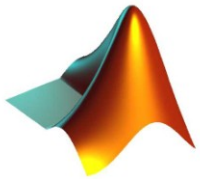
$$\mu = \frac{1}{NM} \sum_{n_1, n_2 \in \eta} a(n_1, n_2)$$

$$\sigma^2 = \frac{1}{NM} \sum_{n_1, n_2 \in \eta} a^2(n_1, n_2) - \mu^2$$

  - A continuación crea un filtro pixel a pixel basado en estas estimaciones

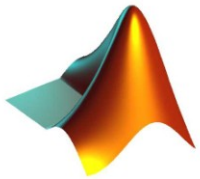
v_2 es la varianza del ruido

$$b(n_1, n_2) = \mu + \frac{\sigma^2 - v^2}{\sigma^2}(a(n_1, n_2) - \mu)$$



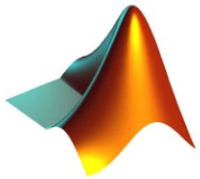
Matlab para reconocimiento de patrones

- Asignación de una clase a un vector de características **x** del objeto a clasificar:
  - Ejemplo: Fisher Iris dataset:
    - http://en.wikipedia.org/wiki/Iris_flower_data_set
    - 50×3 = 150 muestras de flores Iris de tres especies
      - Iris setosa, Iris virginica, Iris versicolor
      - 4 características de cada ejemplo:
        - Longitud y anchura de los pétalos y sépalos



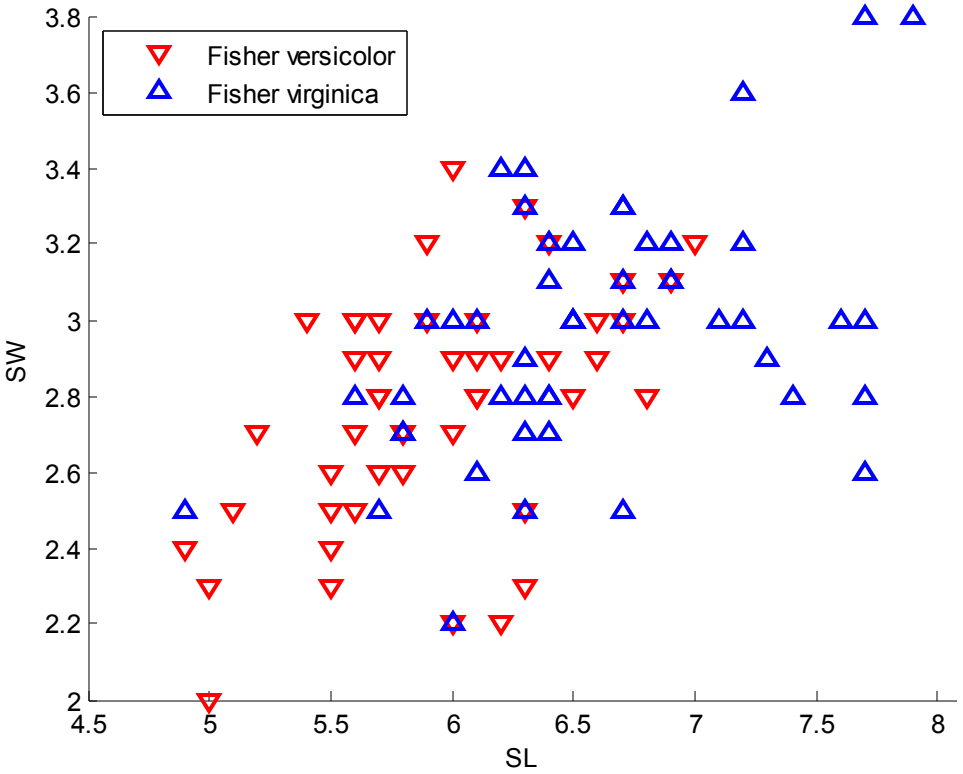
Matlab para reconocimiento de patrones

- Análisis discriminante:
  - `class = classify(sample,training,group)`
  - `class = classify(sample,training,group,type)`
  - `class = classify(sample,training,group,type,prior)`
- knn:
  - `Class = knnclassify(Sample, Training, Group)`
  - `Class = knnclassify(Sample, Training, Group, k)`
  - `Class = knnclassify(Sample, Training, Group, k, distance)`
  - `Class = knnclassify(Sample, Training, Group, k, distance, rule)`
- Máquinas de vectores de soporte:
  - `Group = svmclassify(SVMStruct, Sample)`
  - `Group = svmclassify(SVMStruct, Sample, 'Showplot',ShowplotValue)`
  - `SVMStruct = svmtrain(Training, Group)`
  - `SVMStruct =svmtrain(..., 'Kernel_Function', Kernel_FunctionValue)`

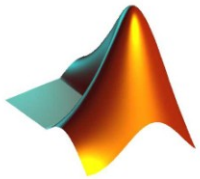


Análisis discriminante: Ejemplo

- `load fisheriris`
- `SL = meas(51:end,1);`
- `SW = meas(51:end,2);`
- `group = species(51:end);`



- `h1 = gscatter(SL,SW,group,'rb','v^',[],'off');`
- `set(h1,'LineWidth',2)`
- `legend('Fisher versicolor','Fisher virginica','Location','NW')`



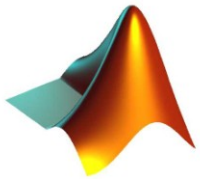
Análisis discriminante: Ejemplo

- Clasificamos una matriz de datos 2-D:

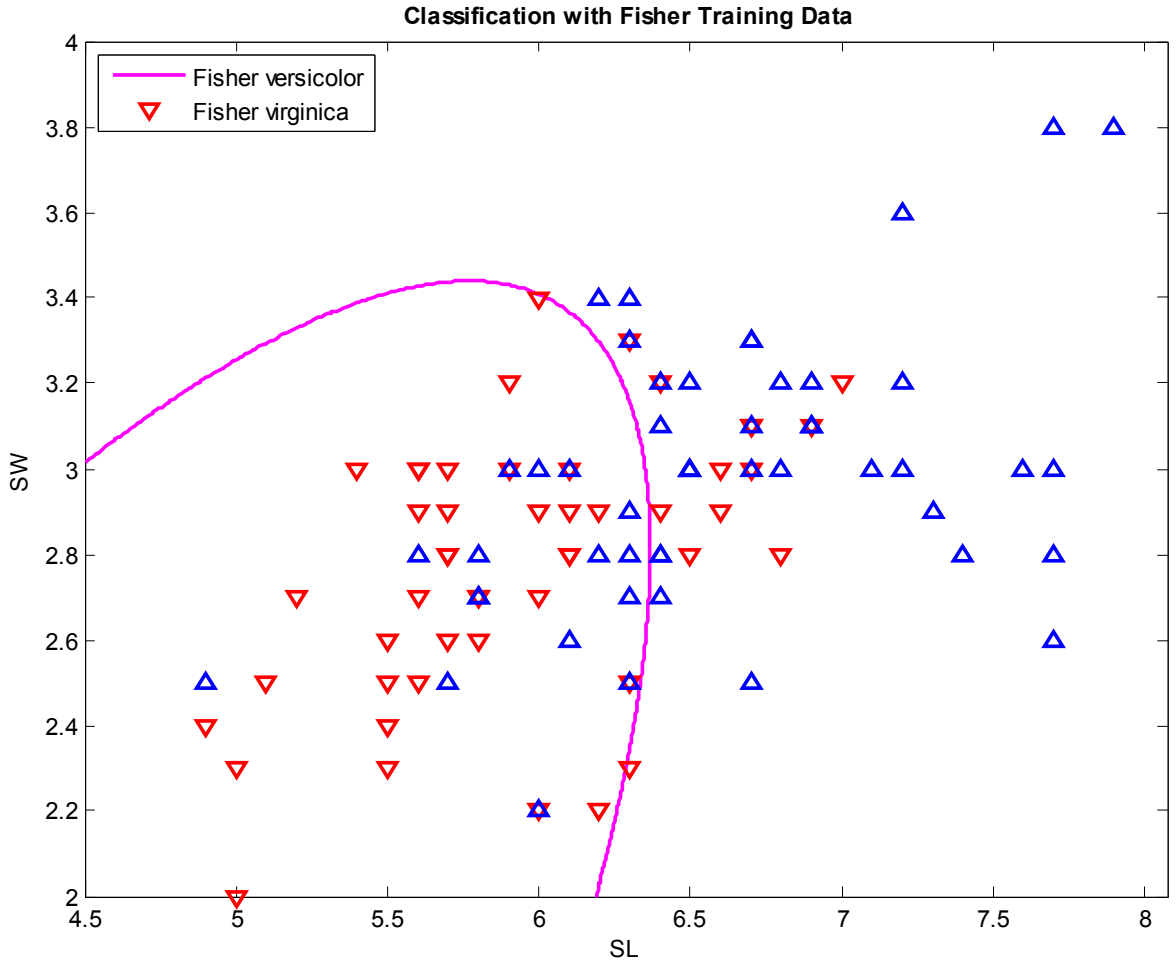
```
[X,Y] =
   meshgrid(linspace(4.5,8)
   ,linspace(2,4));
X = X(:); Y = Y(:);
[C,err,P,logp,coeff] =
   classify([X Y],[SL SW],
   group,'quadratic');
```

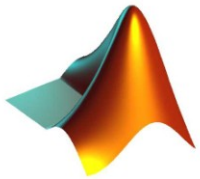
- Visualizar la clasificación:

```
hold on;
gscatter(X,Y,C,'rb','.',1,'off');
K = coeff(1,2).const;
L = coeff(1,2).linear;
Q = coeff(1,2).quadratic;
f = sprintf('0 =
    %g+%g*x+%g*y+%g*x^2+%g*x.*y+%g*y.^2',
    ...
K,L,Q(1,1),Q(1,2)+Q(2,1),Q(2,2));
h2 = ezplot(f,[4.5 8 2 4]);
set(h2,'Color','m','LineWidth',2)
axis([4.5 8 2 4])
xlabel('Sepal Length')
ylabel('Sepal Width')
title('{\bf Classification with Fisher
    Training Data}')
```



Análisis discriminante: Ejemplo

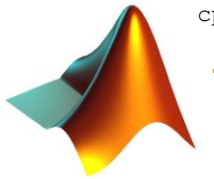




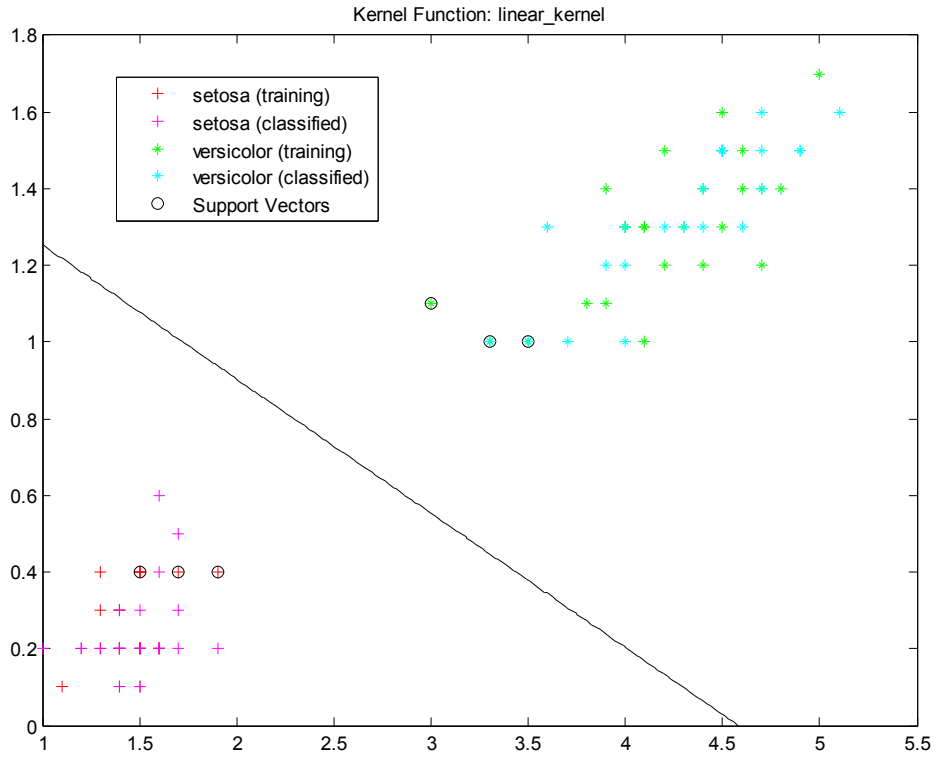
Selección de características

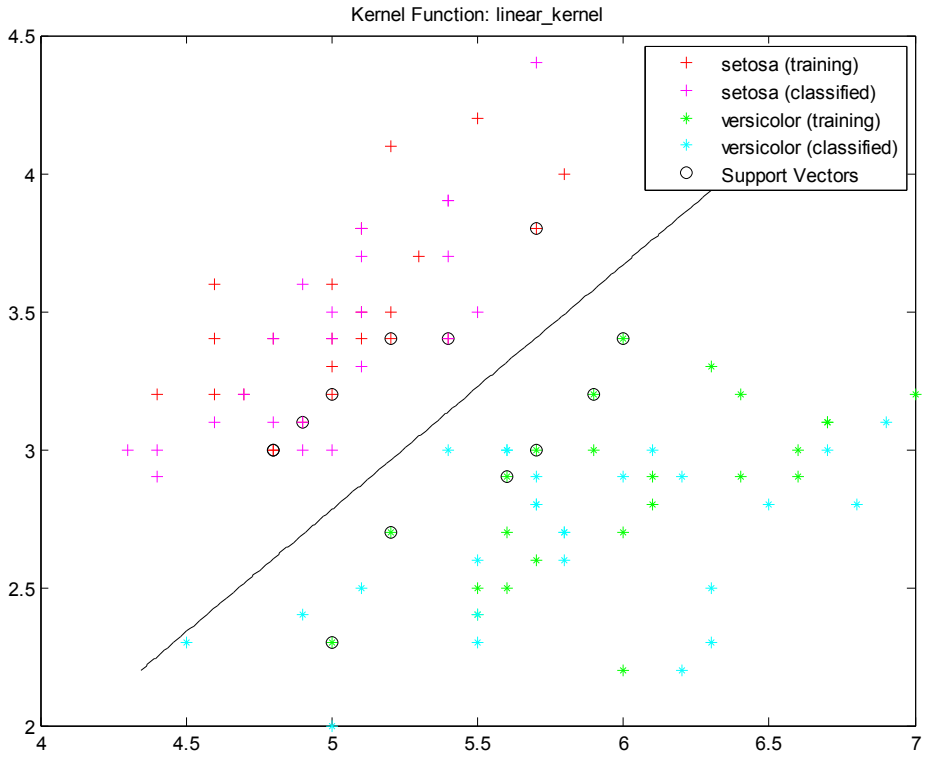
- `[IDX, Z]= rankfeatures(X, Group)`
- `[IDX, Z]= rankfeatures(X, Group,'Criterion', CriterionValue)`

```
load fisheriris;
X= meas(1:100,:);
Group= species(1:100);
[IDX, Z] = rankfeatures(X',Group);

% Selecionamos las variables más discriminativas
data= X(:,[IDX(1) IDX(2)]);

% Selección aleatoria de subconjuntos de entrenamiento y test
[train, test] = crossvalind('holdOut',Group);
cp = classperf(Group);
% Entrenamiento de una máquina de vectores de soporte
svmStruct = svmtrain(data(train,:),Group(train),'showplot',true);
% Añadimos título.
title(sprintf('Kernel Function: %s',...
      func2str(svmStruct.KernelFunction)),...
      'interpreter','none');
% Clasificación del conjunto de test
classes = svmclassify(svmStruct,data(test,:),'showplot',true);
% Evaluación a partir de la tasa de correctas.
classperf(cp,classes,test);
cp.CorrectRate
```



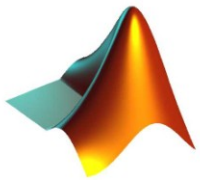
Selección de características






```
data= X(:,[IDX(1) IDX(2)]);
```

```
data= X(:,[IDX(3) IDX(4)]);
```



Estas transparencias se pueden obtener en:

http://www.ugr.es/~javierrp

Para cualquier consulta:

Javier Ramírez (javierrp@ugr.es)
Dpto. Teoría de la Señal, Telemática y Comunicaciones
Despacho 22
ETSII