Procesamiento Digital de Señales usando GNU Octave

Dr. Alfonso Alba Cadena
fac@galia.fc.uaslp.mx

Facultad de Ciencias
UASLP

GNU Octave...

- Es un lenguaje de alto nivel orientado al cómputo numérico
- Trabaja nativamente con vectores y matrices
- Es altamente compatible con Matlab
- Puede extenderse mediante funciones escritas en C/C++
- Es de distribución gratuita

Octave puede descargarse de `http://www.octave.org`

Operaciones con matrices y vectores

- En **Octave** se pueden definir matrices escribiendo sus elementos entre corchetes.

- La coma separa los elementos en columnas, y el punto y coma los separa en renglones.

Ejemplo: `m = [1, 2, 3; 4, 5, 6]` asigna a la variable `m` la matriz

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}.$$

Acceso a los elementos de una matriz

Dada una matriz `m` se puede tener acceso a cualquier elemento, renglón, columna, o sub-matriz de `m`:

- `m(i,j)` hace referencia a un elemento.
- `m(i,:)` hace referencia al `i`-ésimo renglón.
- `m(:,j)` hace referencia a la `j`-ésima columna.
- `m(i1:i2, j1:j2)` hace referencia a una sub-matriz.

Aritmética de matrices

- Suma y resta: `a + b`, `a - b`
- Producto matricial: `a * b`
- Producto elemento por elemento: `a .* b`
- Transpuesta conjugada: `a'`

Matrices especiales

Las siguientes funciones de **Octave** devuelven matrices de utilidad general.

- Identidad: `eye(n, m)`
- Unos: `ones(n, m)`
- Ceros: `zeros(n, m)`
- Ruido uniforme: `rand(n, m)`
- Ruido normal: `randn(n, m)`
- Vector de n valores equiespaciados: `linspace(base, limit, n)`
- Rango de índices: `a:b` (devuelve el vector `[a, a+1, ..., b]`)

Ejercicios

Defina las siguientes matrices en **Octave** :

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 2 & 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} -1 & 1 \\ 2 & 0 \\ 1 & -2 \end{bmatrix}, \quad C = 3 \cdot I_{3\times 3}.$$

Usando las matrices anteriores, calcule las siguientes expresiones:

a) $A \times B$

b) $B \times A - C$

c) $A + \lambda N$, donde $\lambda = 0.1$ y N es ruido uniforme

Funciones

Una función de **Octave** toma cero o mas parámetros (escritos entre paréntesis), realiza algún procedimiento, y puede o no devolver algún resultado.

Ejemplos de funciones:

```
> cos(1)
ans = 0.54030
> ones(1, 5)
ans =
  1  1  1  1  1
> floor(mean([1, 2, 3, 4, 5]))
ans = 2
```

Definición de funciones

Uno puede escribir sus propias funciones de **Octave** usando la siguiente sintaxis:

`function` *resultado* = *nombre* (*parametros*)
    cuerpo de la función
`end`

Ejemplo:

```
> function y = cuad(x)
> y = x * x;
> end
> cuad(5)
ans = 25
```

Librerías de funciones

Para poder utilizar una función en el futuro, sin tener que escribirla nuevamente, es necesario guardarla en un archivo de texto con el mismo nombre que la función y extensión `.m`

```
# Archivo:       cuad.m
# Descripcion:   cuad(x) devuelve el cuadrado de x

function y = cuad(x)
    y = x * x;
end
```

Buenas prácticas de programación

- Escribir funciones de librería para realizar tareas comunes.
- Dividir un problema grande en sub-problemas mas pequeños cuya implementación sea sencilla.
- Agregar comentarios a las funciones de librería que describan la tarea que realizan y los parámetros que toman.
- Limitar las funciones a las tareas que deben realizar de manera que sean claras y reutilizables.

Ejercicios:

1. Escribir una función `fact(x)` que calcule y devuelva el factorial de `x`. Para esto, investigar cómo realizar ciclos (for, while, etc.) en **Octave** .

2. Escribir una función que tome como entrada dos vectores y devuelva el producto punto. La función debe verificar primero que los vectores tengan la misma longitud. Usar las estructuras `if`, `for`, y la función `length()`.

3. Utilizando la función anterior, escriba una función que calcule la norma de un vector x (dada por $\sqrt{x \cdot x}$).

Señales discretas en Octave

- Una señal discreta y finita $x[n]$ se representa en **Octave** como un vector. En ocasiones es útil obtener un vector adicional que contenga el rango de n para el cual $x[n]$ está definida.

- Ejemplos:

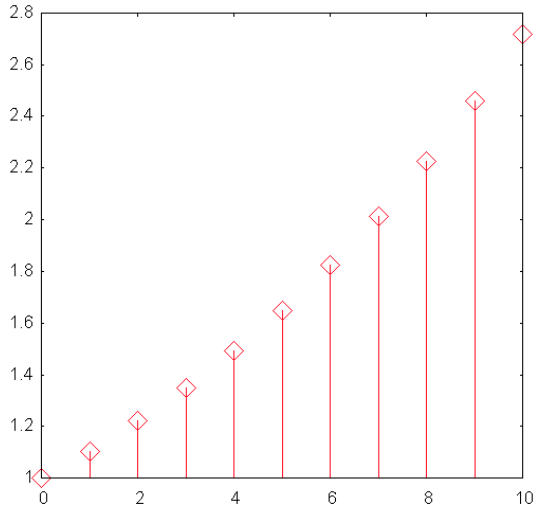
a)
```
x = [2, 1, 0, -1, 1]
```

b)
```
x = exp(0.1 * (1:10))
```

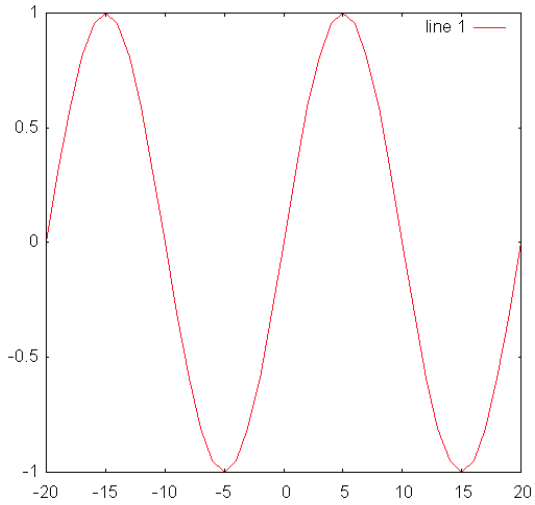
c)
```
n = -10:10
x = sin(n * pi / 5)
```

Graficación de señales

Para graficar una señal en **Octave** podemos usar las funciones `plot()` o `stem()`. Ejemplos:




```
n = 0:10;
x = exp(0.1 * n);
stem(n, x);
```

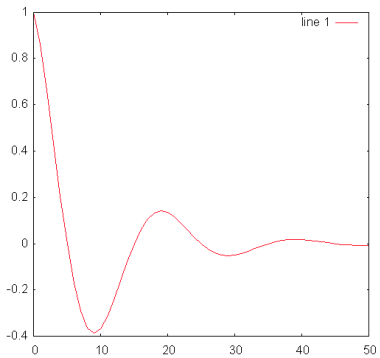



```
n = -10:10;
x = sin(n * pi / 5);
plot(n, x);
```

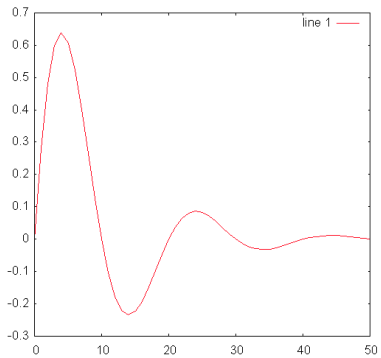
Señales complejas

Para una señal compleja podemos graficar por separado las partes real e imaginaria, o bien la magnitud y la fase.

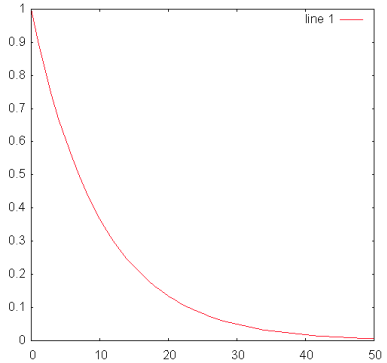
```
n = 0:50;
x = exp(j * n * pi / 10) * exp(-n / 10)
```



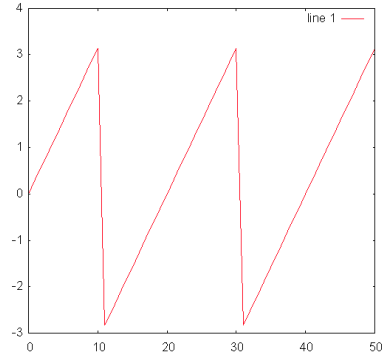
`plot(n, real(x))`



`plot(n, imag(x))`



`plot(n, abs(x))`



`plot(n, angle(x))`

Ejercicios:

Graficar las siguientes señales en **Octave** :

a) $x[n] = 3\cos\left(\frac{\pi}{5}t + \frac{\pi}{4}\right),\ -10 \leq n \leq 10$

b) $x[n] = [3, -4, 5, 1, -2, 0, 4, 2]$

c) $x[n] = 20e^{j\pi n/8} - n^2,\ 0 \leq n \leq 20$

d) $x[n]$ es una señal de ruido uniforme de 50 muestras

Solución de sistemas lineales e inavariantes en el tiempo

Para encontrar la salida de un sistema LIT se puede utilizar:

- Convolución (sistemas no-recursivos):
  `conv(x, h)` devuelve la convolución de `x` y `h`.

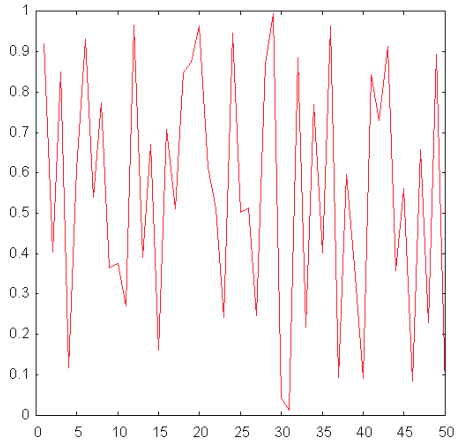
- Ecuaciones de diferencias (sistemas recursivos):
  `filter(b, a, x)` devuelve la salida $y[n]$ del sistema

$$\sum_{k=0}^{N} a_{k+1} y[n-k] = \sum_{k=0}^{M} b_{k+1} x[n-k].$$

Convolución

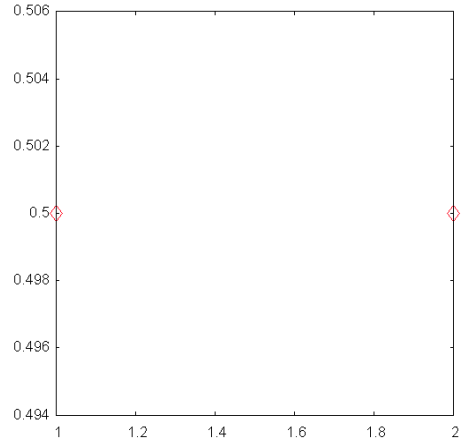
Ejemplo: procesar una señal de ruido mediante un sistema de promedio móvil dado por

$$T\{x[n]\} = \frac{1}{2}(x[n] + x[n-1]).$$



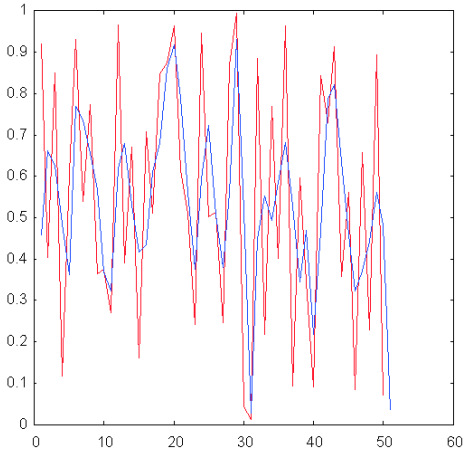
Señal de ruido

`x = rand(1,50)`



Respuesta al impulso

`h = [0.5, 0.5]`



Señal procesada

`y = conv(x, h)`

Convolución

La longitud de la convolución será igual a la suma de las longitudes de las señales de entrada menos uno:

- `length(x)` devuelve 50
- `length(h)` devuelve 2
- `length(conv(x, h))` devuelve 51

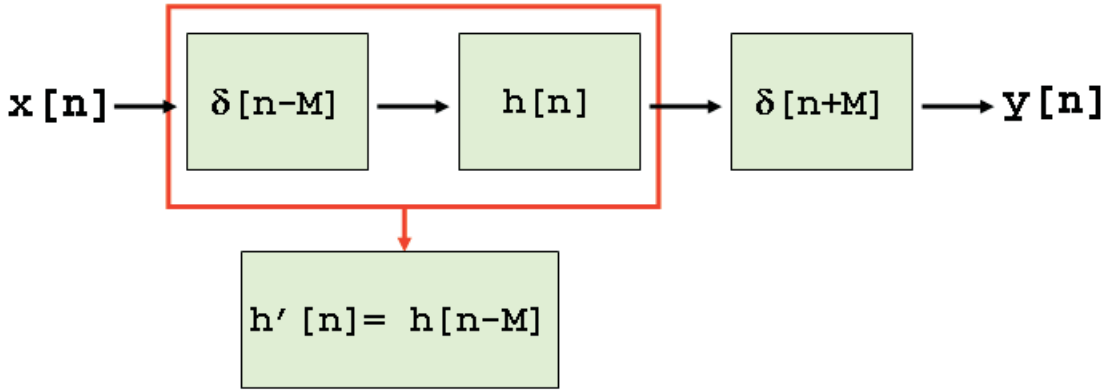
En ocasiones es necesario que la longitud de la señal de salida sea igual que la de entrada:

```
y = conv(x, h);
y = y(1:length(x));
```

Implementación de sistemas no-causales

Para un sistema no-causal, la respuesta al impulso $h[n]$ es distinta de cero para algunos $n < 0$; sin embargo, en **Octave** los elementos de los vectores no pueden tener índices negativos.

Para resolver este problema, podemos usar retardos:



Donde M es tal que $h[n] = 0$ para $n < -M$.

Ejercicios:

- Escribir una función `lp_ideal` que tome parámetros ω_c (dado en radianes por muestra) y N, y que devuelva la señal:

$$h_{lp}[n] = \frac{\sin \omega_c n}{\pi n}, \ -N \leq n \leq N$$

como un vector de $2N + 1$ elementos (donde el elemento $N + 1$ corresponde a $h_{lp}[0]$). *Sugerencia: utilice un vector de subíndices* `n = -N:N`.

- La señal del ejercicio anterior es la respuesta al impulso de un filtro ideal pasa-bajas con frecuencia de corte ω_c. Escribir una función llamada `lp_filt` que tome parámetros x y ω_c, y que devuelva la salida y del sistema $y[n] = x[n] * h_{lp}[n]$. Pruebe la función filtrando un impulso y una señal de ruido.

Ecuaciones de diferencias

Para resolver un sistema de ecuaciones de diferencias del tipo:

$$\sum_{k=0}^{N} a_{k+1} y[n-k] = \sum_{k=0}^{M} b_{k+1} x[n-k],$$

usamos la instrucción

```
y = filter(b, a, x);
```

donde `a` y `b` contienen los coeficientes a_{k+1} y b_{k+1}, respectivamente.

Ecuaciones de diferencias

Considerar como ejemplo el sistema:

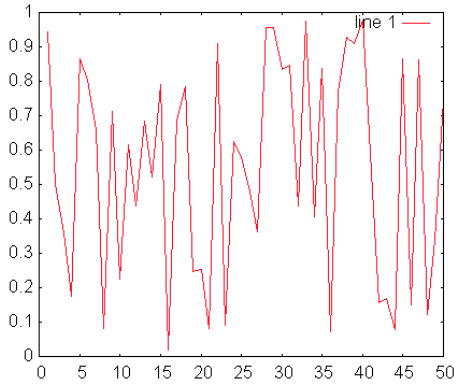
$$y[n] = py[n-1] + (1-p)x[n],$$

que corresponde a un filtro pasa-bajas recursivo de un polo con frecuencia de corte f_c, $0 < f_c < 1$, donde $p \approx \exp(-2\pi f_c)^*$. Este filtro puede implementarse en **Octave** mediante la siguiente instrucción:

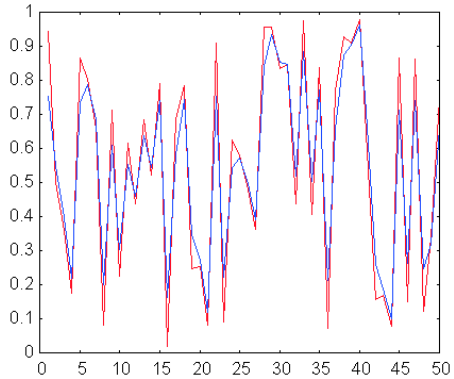
```
y = filter([1 - p], [1, -p], x);
```

*The Scientist and Engineer's Guide to Digital Signal Processing. Steven W. Smith, Ch. 19, pag. 324. www.dspguide.com

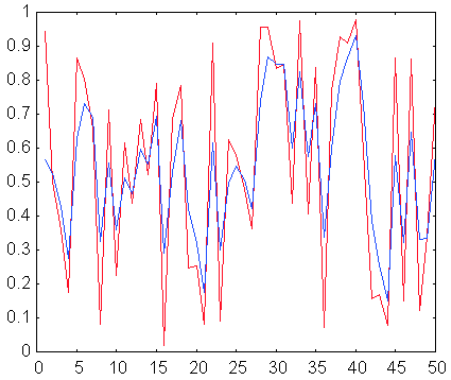
Filtro pasa-bajas recursivo de un polo



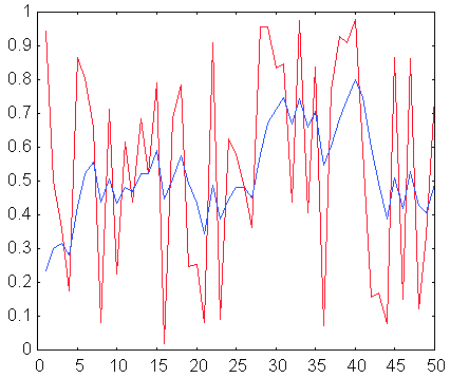
x



y con $p = 0.2$



y con $p = 0.4$



y con $p = 0.75$

Ejercicios:

- Implementar una función de **Octave** llamada `lp_1pole(x, fc)` que devuelva la salida de un filtro pasa-bajas de un polo con frecuencia de corte f_c y señal de entrada x.

- Implementar una función llamada `lp_npole(x, fc, n)` que aplique n veces la función `lp_1pole` a una señal de entrada x con frecuencia de corte f_c.

Transformada de Fourier

Octave incluye la función `fft` que devuelve la transformada discreta de Fourier (TDF) de una señal de entrada:

```
X = fft(x);
```

donde

$$X[k] = \sum_{n=0}^{N-1} x[n+1] e^{j\frac{2\pi k}{N}n},\ 0 \leq k < N,$$

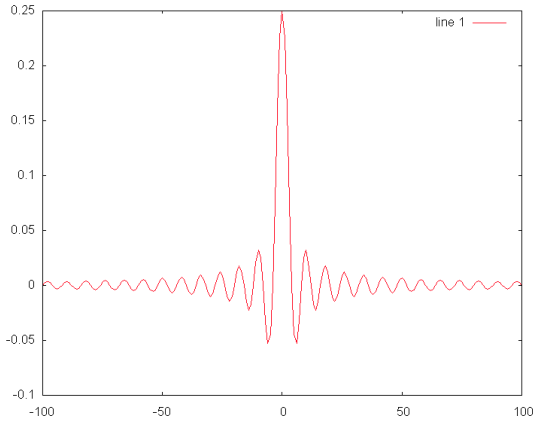
y N es la longitud de x y X.

Podemos ver la TDF como una versión discretizada de la transformada contínua de Fourier $X(\omega)$ evaluada en las frecuencias $\omega = 2\pi k/N$ para $k = 0, \ldots, N-1$.

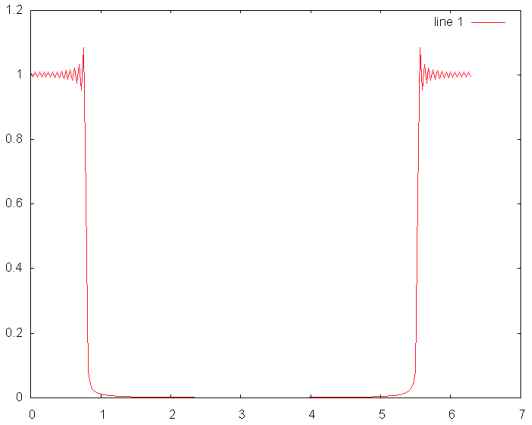
Respuesta en frecuencia de un sistema LIT

Si conocemos la respuesta al impulso h de una transformación LIT, la TDF nos dará una representación de la respuesta en frecuencia (compleja) del sistema.

Ejemplo: Respuesta al impulso de un filtro ideal pasa-bajas dada por $h[n] = \sin(\omega_c n)/(\pi n)$.



`h`

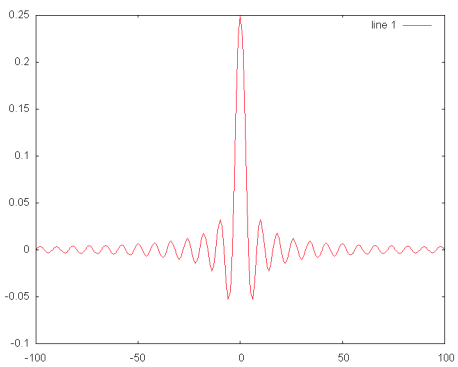


`abs(fft(h))`

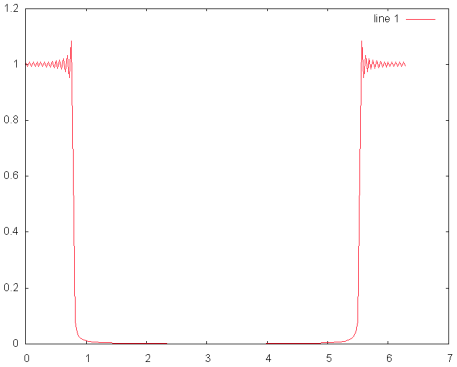
Frecuencias negativas

La función `fft(x)` nos da la representación discreta de Fourier $X[k] = X(\omega_k)$ de una señal $x[n]$ para $\omega_k = 2\pi k/N$, $k = 0, \ldots, N-1$. Por lo tanto, las frecuencias ω_k corren desde 0 hasta 2π.

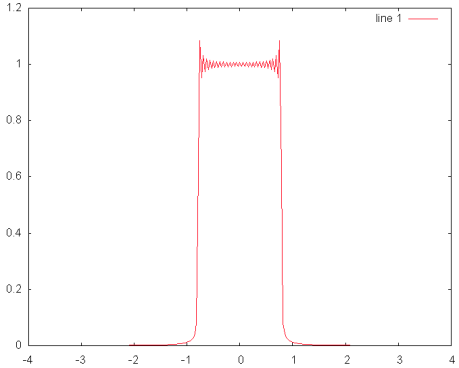
En ocasiones, es preferible trabajar con frecuencias entre $-\pi$ y π. Esto puede lograrse simplemente mediante un desplazamiento circular $X[k] \longleftarrow X[k - \lceil N/2 \rceil]$.



`h`



`fft(h)`



`fftshift(fft(h))`

Transformada inversa de Fourier

La función `ifft(X)` devuelve la transformada inversa de Fourier (IFT) de $X[k]$; es decir, recupera $x[n]$.

Esto puede verificarse calculando el error cuadrado medio (MSE) entre una señal `x` y la señal `ifft(fft(x))`:

```
> x = rand(1, 100);
> x0 = ifft(fft(x));
> sum(abs(x - x0).^2) / length(x)
ans =  7.5800e-33
```

Como puede observarse, el MSE es muy pequeño, pero distinto de cero a causa de los errores de redondeo.

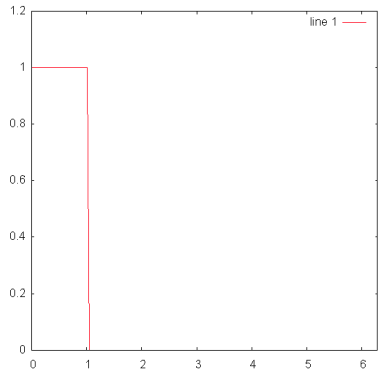
Diseño de filtros mediante IFT

Ejemplo: Implementar un filtro ideal pasa-bajas con frecuencia de corte $\omega_c = \pi/3$ y respuesta nula a las frecuencias negativas.

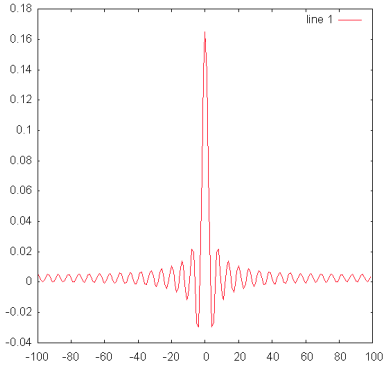
Respuesta en frecuencia: $H[k] = \begin{cases} 1 & \text{si } 0 \leq k < \frac{\omega_c N}{2\pi}, \\ 0 & \text{para otros valores de } k. \end{cases}$

Para obtener el kernel de convolución en **Octave** :

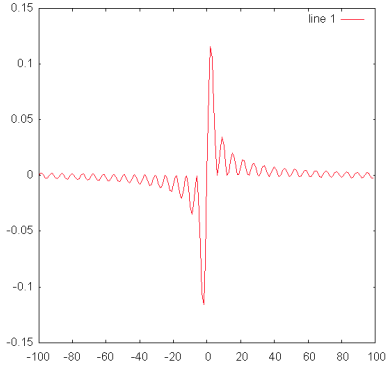
```
N = 200;
wc = pi / 3;
kc = wc * N / (2 * pi);
H = zeros(1, N);
H(1:kc) = 1;
h = fftshift(ifft(H));
```



H



real(h)



imag(h)

Ejercicios

- Implementar una función `rfplot(h)` que grafique la respuesta en frecuencia $|H(\omega)|$ y la respuesta en fase $\angle H(\omega)$, para $-\pi \leq \omega \leq \pi$, de un sistema cuya respuesta al impulso está dada por el parámetro `h`. Utilice la función `subplot()` para mostrar ambas gráficas en una sola ventana, y la función `unwrap()` para desenvolver las fases.

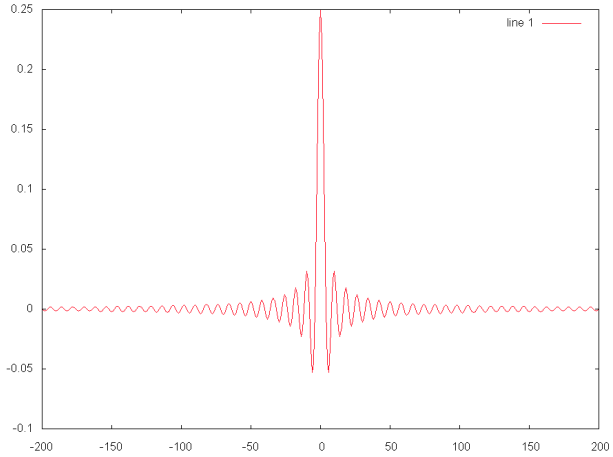
- Modifique la función anterior para que acepte como parámetro opcional la frecuencia de muestreo`fm` (e.g., `rfplot(h, fm)`). En caso de especificar este parámetro, la escala de frecuencias deberá ser de `-fm/2` a `fm/2`. Utilice la función `nargin()` para determinar el número de parámetros pasados a `rfplot()`.

Filtro ideal pasa-bajas

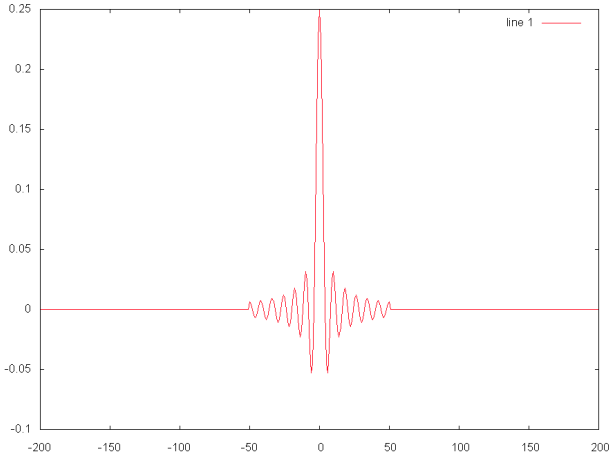
El filtro ideal pasa-bajas tiene una respuesta al impulso infinita dada por la función:

$$h[n] = \frac{\sin \omega_c n}{\pi n}, \ -\infty < n < \infty, \ h[0] = \omega_c/\pi.$$

Para poder representarla en **Octave** truncamos $h[n]$ y solo calculamos para $-N < n < N$:



Filtro ideal ($\omega_c = \pi/4$)

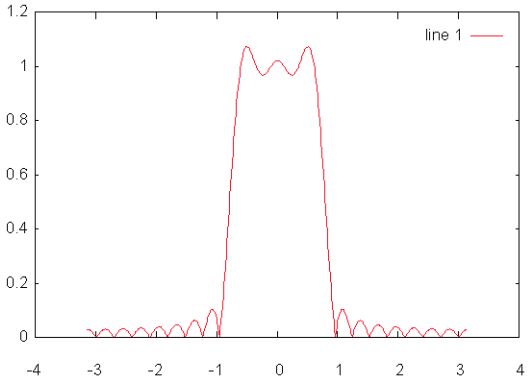


Filtro ideal (truncado)

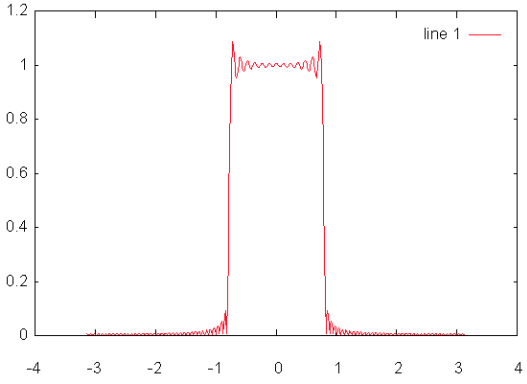
Respuesta en frecuencia del filtro ideal

Debido al truncamiento, la respuesta en frecuencia del filtro será una mera aproximación de la respuesta de un filtro ideal:

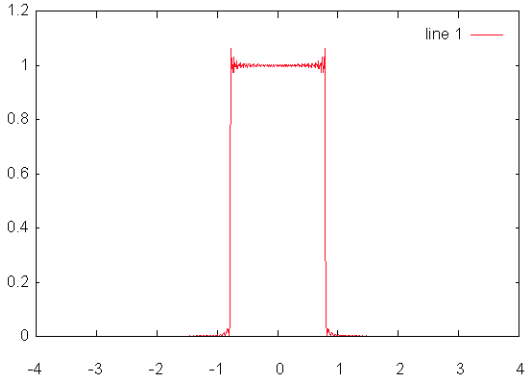
```
n = -N:N;
h = sin(wc * n) ./ (pi * n);
h(N + 1) = wc / pi;
H = fftshift(fft(h, 512));
plot(linspace(-pi, pi, length(H)), H);
```



$N = 10$



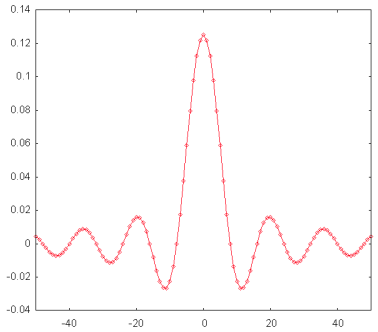
$N = 50$



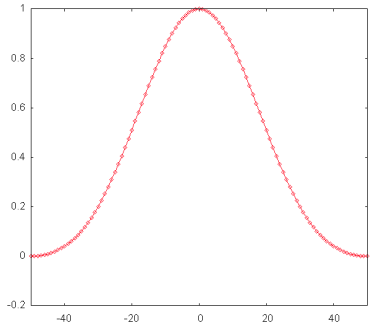
$N = 200$

Filtros sinc con ventana

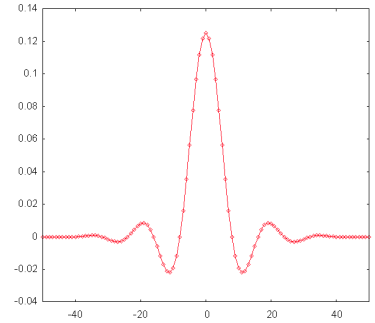
Podemos multiplicar la respuesta al impulso del filtro ideal por una ventana para suavizar la respuesta en frecuencia:



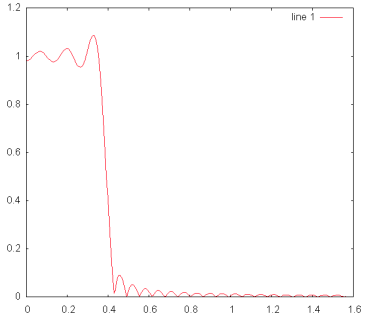
`h[n]`



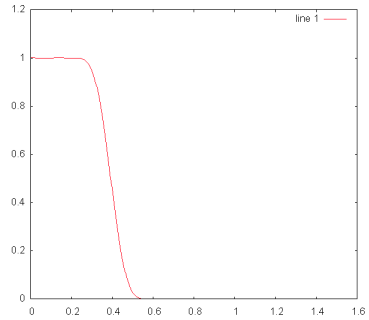
`blackman(length(h))`



`h.*blackman(length(h))`



Respuesta en frecuencia

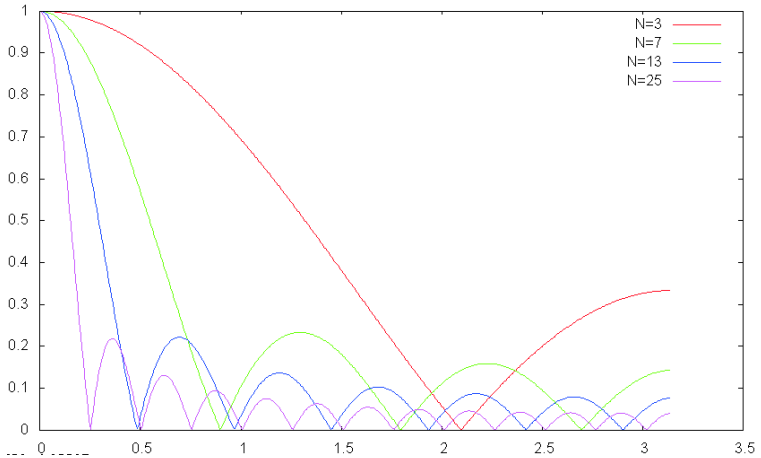


Promedios móviles

Un promedio móvil es un filtro de suavizamiento cuya respuesta al impulso puede expresarse como:

$$h[n] = \begin{cases} 1/N & \text{si } 1 < n < N \\ 0 & \text{para otros valores de } n \end{cases}$$

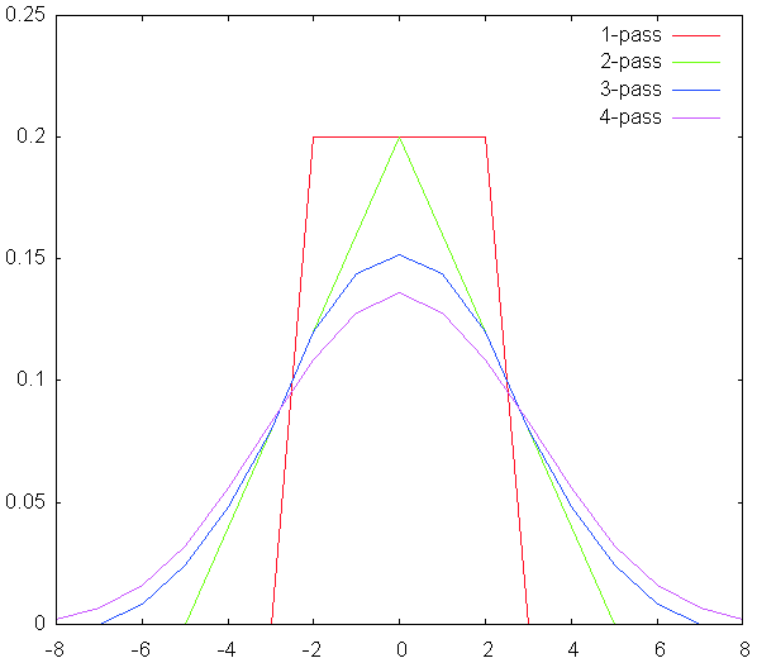
En **Octave** : `h = ones(1, N) / N;`



Respuesta en frecuencia del promedio móvil

Filtro Gaussiano

Si aplicamos varias veces un promedio móvil, el kernel de la transformación resultante se aproxima a una Gaussiana.

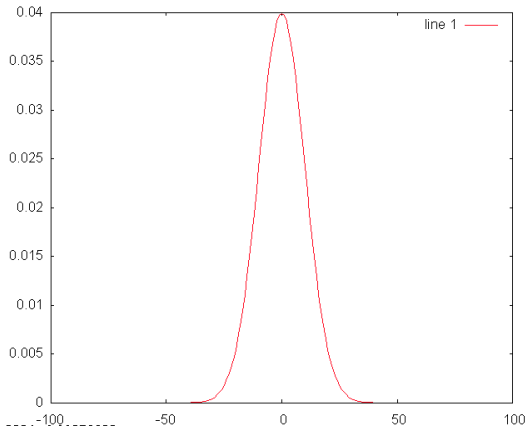


$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$

Función Gaussiana

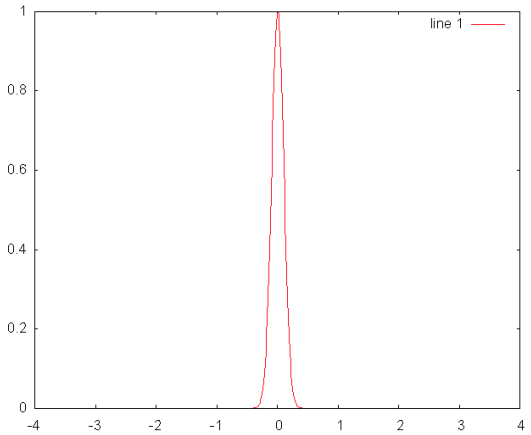
Filtro Gaussiano

Se puede demostrar que la TF de una Gaussiana con ancho σ_n (en muestras) es otra Gaussiana con ancho $\sigma_\omega = 1/\sigma_n$ (en radianes por muestra).



$$h[n] = \frac{1}{\sigma_n\sqrt{2\pi}} e^{-\frac{n^2}{2\sigma_n^2}}$$

$\sigma_n = 10$ muestras



$$H(e^{j\omega}) = e^{-\frac{\omega^2}{2\sigma_\omega^2}}$$

$\sigma_\omega = \frac{1}{\sigma_n} = 0.1$ rad/muestra

Filtros pasa-banda

Propiedad de desplazamiento en frecuencia de la Transformada de Fourier: si la TF de una señal $x[n]$ es $X(e^{j\omega})$, entonces

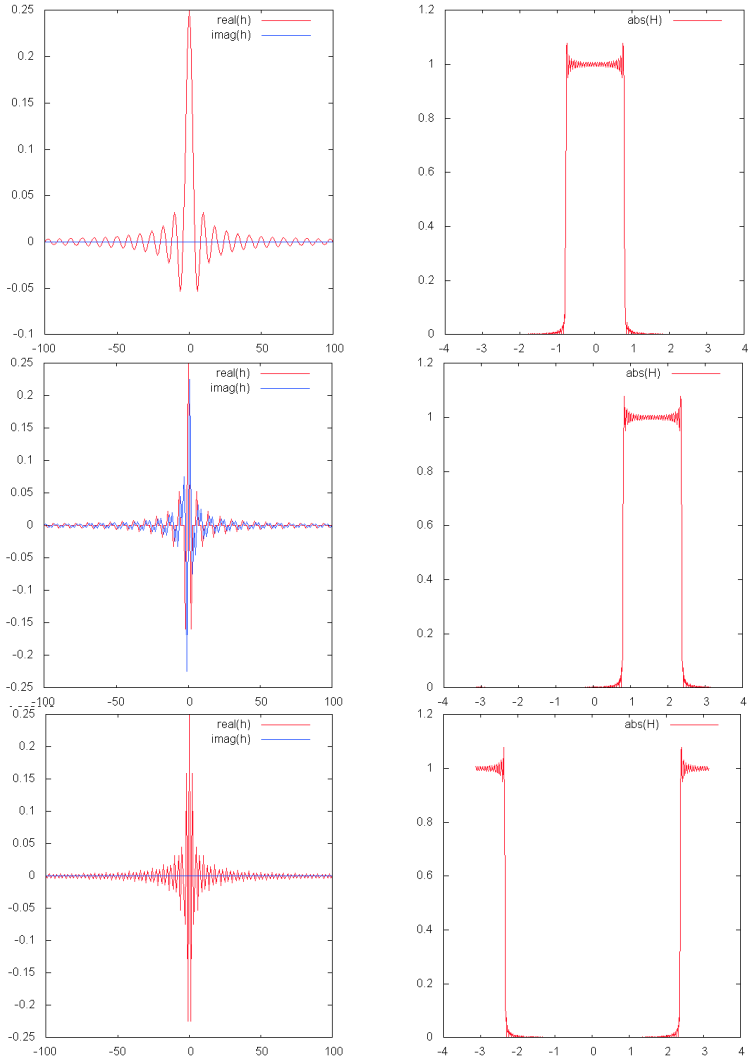
$$\mathcal{F}\left\{x[n] * e^{j\omega_0 n}\right\} \;=\; X\left(e^{j(\omega-\omega_0)}\right).$$

Una aplicación de esta propiedad es el diseño de filtros pasa-banda. Si multiplicamos el kernel de un filtro pasa-bajas por $\exp(j\omega_0 n)$, el kernel resultante corresponderá a un filtro pasa-banda centrado en ω_0 con un ancho dado por la frecuencia de corte del filtro pasa-bajas.

Filtro pasa-bajas con kernel $h[n]$ y frecuencia de corte $\omega_c = \pi/4$

Pasa-banda con kernel $h[n]\exp(j\omega_0 n)$ entonado en $\omega_0 = \pi/2$ y ancho de banda $\Delta\omega = \pi/4$

Pasa-altas con kernel $h[n](-1)^n$ y frecuencia de corte $\omega_c = \pi - \pi/4$



Filtros de Gabor

Un filtro de Gabor es un filtro pasa-banda con respuesta Gaussiana, cuya respuesta al impulso está dada por:

$$h[n] = \frac{1}{\sigma_n \sqrt{2\pi}} \exp\left[-\frac{n^2}{2\sigma_n^2}\right] \exp\left[j\omega_0 n\right],$$

donde ω_0 es la frecuencia de entonamiento, y σ_n es el ancho del kernel (en muestras). El ancho de banda del filtro está dado por

$$\sigma_\omega = \frac{1}{\sigma_n} \text{ rad/muestra.}$$

Estos filtros se utilizan ampliamente para descomposición tiempo-frecuencia de señales y procesamiento de imágenes.

Ejercicios:

Escribir funciones de **Octave** que calculen y devuelvan la respuesta al impulso (kernel) para diversos filtros pasa-banda. Estas funciones deben tomar como parámetros la frecuencia de entonamiento (ω_0), el ancho de banda (Δ_ω ó σ_ω), y el tamaño del kernel en muestras (N). Calcular el kernel para $-N/2 \leq n \leq N/2$. Los filtros a implementar son los siguientes:

- Filtro ideal (sinc)
- Filtro sinc con ventana (usar ventana de Blackman)
- Filtro de Gabor