Laboratorio 1

PROCESAMIENTO DE SEÑALES E IMÁGENES

Profesores:

- Violeta Chang C.
- Leonel E. Medina

Ayudante: Luis Corral

Introducción a MATLAB

En MATALB representamos las variables numéricas como vectores y matrices. Cuando omitimos el punto y coma al final de una lÍnea de código, su valor puede ser visto en el panel lateral. Podemos utilizar los operadores de suma, resta multiplicación, división y exponenciación (+,-,*,/,^). Un punto antes del operador indica operación elemento a elemento.

```
clearvars         % Borra todas las variables de la memoria.

a = [1 2 3 4 5 6] % Vector fila 1x6.
```

```
a = 1×6
     1     2     3     4     5     6
```

```
b = a'            % La transpuesta genera un vector
```

```
b = 6×1
     1
     2
     3
     4
     5
     6
```

```
                  % columna 6x1.
a_2 = [1 0; 0 1]  % Matrix identidad de 2x2.
```

```
a_2 = 2×2
     1     0
     0     1
```

```
a_3 = (((a+3)*2)/4).^2 % Operaciones matemáticas.
```

```
a_3 = 1×6
    4.0000    6.2500    9.0000   12.2500   16.0000   20.2500
```

Las funciones generadoras permiten crear datos enteros y de punto flotante de manera simple. Existen variadas formas de crear matrices y vectores, las cuales pueden ser consultadas en la documentación MATLAB Help.

```
c_1 = 0:1:5           % Vectores espaciados linealmente
```

```
c_1 = 1×6
     0     1     2     3     4     5
```

```
                      % inicio:distancia:fin o,
c_2 = linspace(0,5,6) % linspace(inicio,fin,largo).
```

```
c_2 = 1×6
     0     1     2     3     4     5
```

```
d = zeros(2,3)        % Matriz de zeros 2x3
```

```
d = 2×3
     0     0     0
     0     0     0
```

Para cambiar las dimensiones y valores, utilizamos concatenaciones, slicing, reshape, o permute (más la ya mencionada transpuesta) según corresponda:

```
d_2 = permute(d,[2,1]) % Matriz de zeros 3x2.
```

```
d_2 = 3×2
     0     0
     0     0
     0     0
```

```
c_1(1,3:end) = 0       % Se cambian los valores desde la
```

```
c_1 = 1×6
     0     1     0     0     0     0
```

```
                      % posición 3 hasta el final por 0.
c_3 = [[c_1;c_2] d]   % Matriz de 2x9
```

```
c_3 = 2×9
     0     1     0     0     0     0     0     0     0
     0     1     2     3     4     5     0     0     0
```

```
c_4 = [1 2;3 4]       % Matriz de 2x2
```

```
c_4 = 2×2
     1     2
     3     4
```

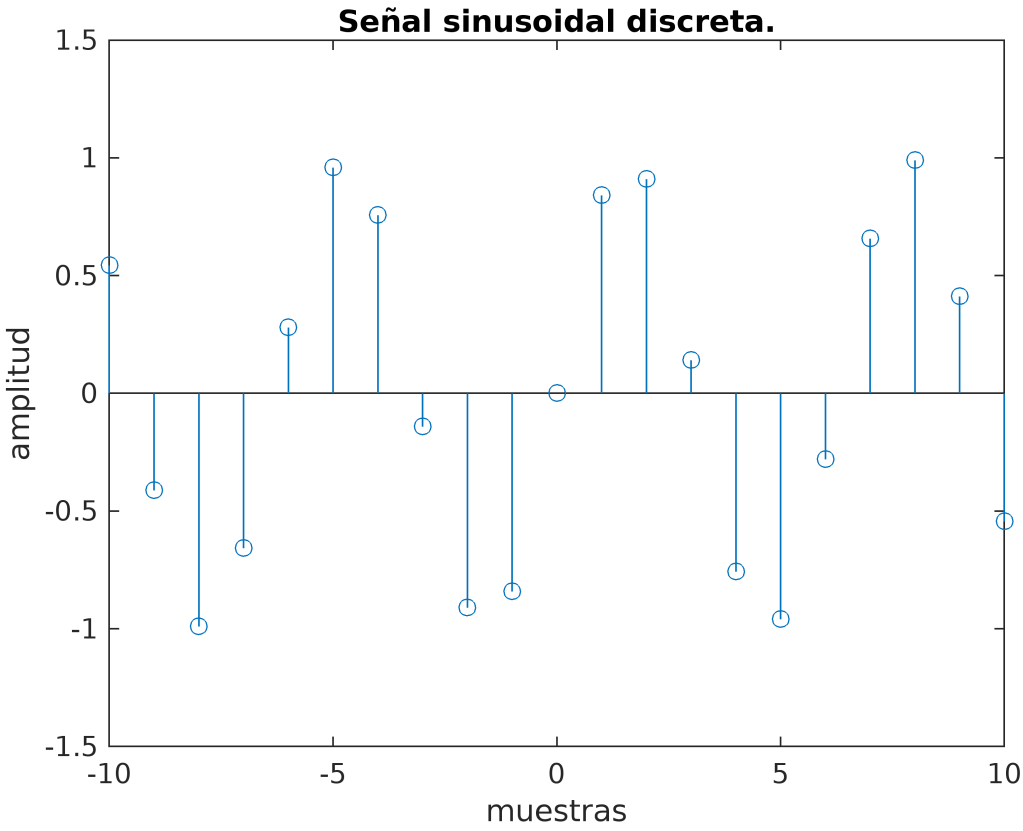
```
d = reshape(d,[1,6])   % Vector de zeros 1x6
```

```
d = 1×6
     0     0     0     0     0     0
```

Señales discretas

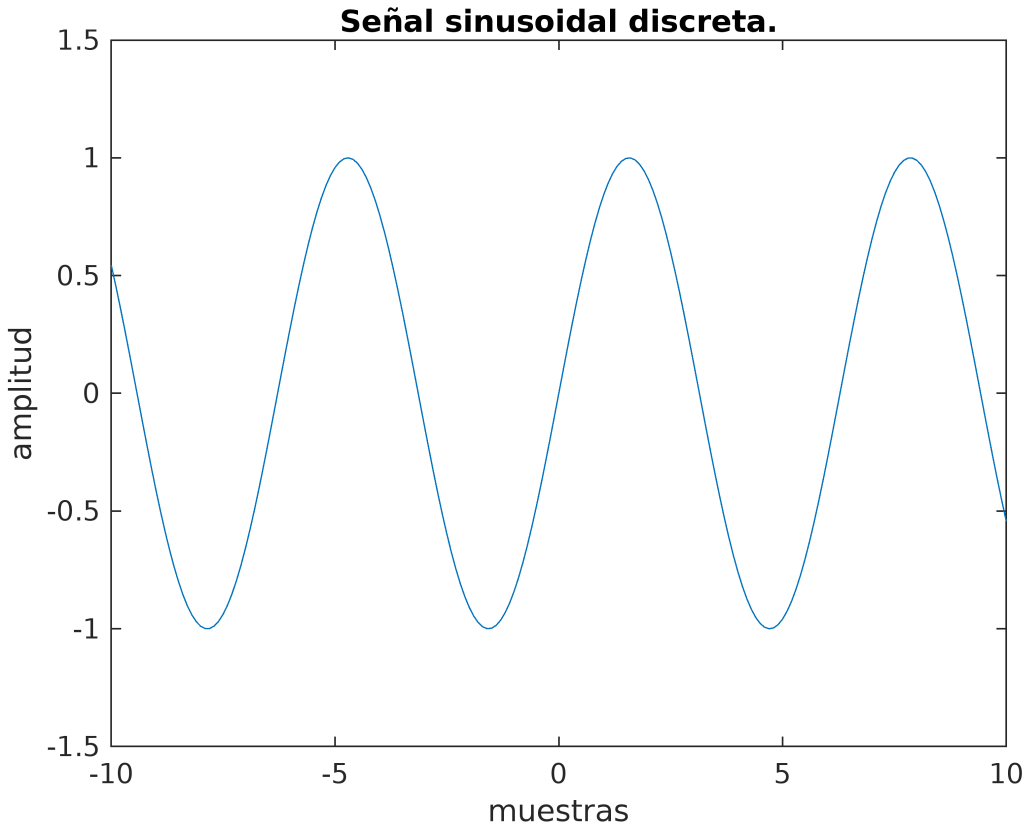
La representación de señales discretas se puede realizar utilizando la función stem(eje_x, eje_y) para generar gráficos. Las funciones title, xlabel, ylabel e ylim nos permiten dar formato al gráfico. Las funciones trigonométricas funcionan sobre el arreglo directamente.

```
clearvars

n = -10:1:10;        % Utilizamos n para señales discretas
x_n = sin(n);        % (21 muestras).
figure               % Crea una figura para el gráfico.
stem(n,x_n)
title('Señal sinusoidal discreta.')
xlabel('muestras')   % Etiqueta del eje x.
ylabel('amplitud')   % Etiqueta del eje y.
ylim([-1.5 1.5])     % Límites del eje y.
```



Para señales de mayor cantidad de muestras, la función plot(eje_x, eje_y) nos entrega un gráfico de líneas entre cada muestra. Ambas señales son discretas, pero utilizamos plot para mejorar la visualización.

```
n = -10:0.1:10;      % 201 muestras.
x_n = sin(n);
figure
plot(n,x_n)
title('Señal sinusoidal discreta.')
xlabel('muestras')
ylabel('amplitud')
ylim([-1.5 1.5])
```

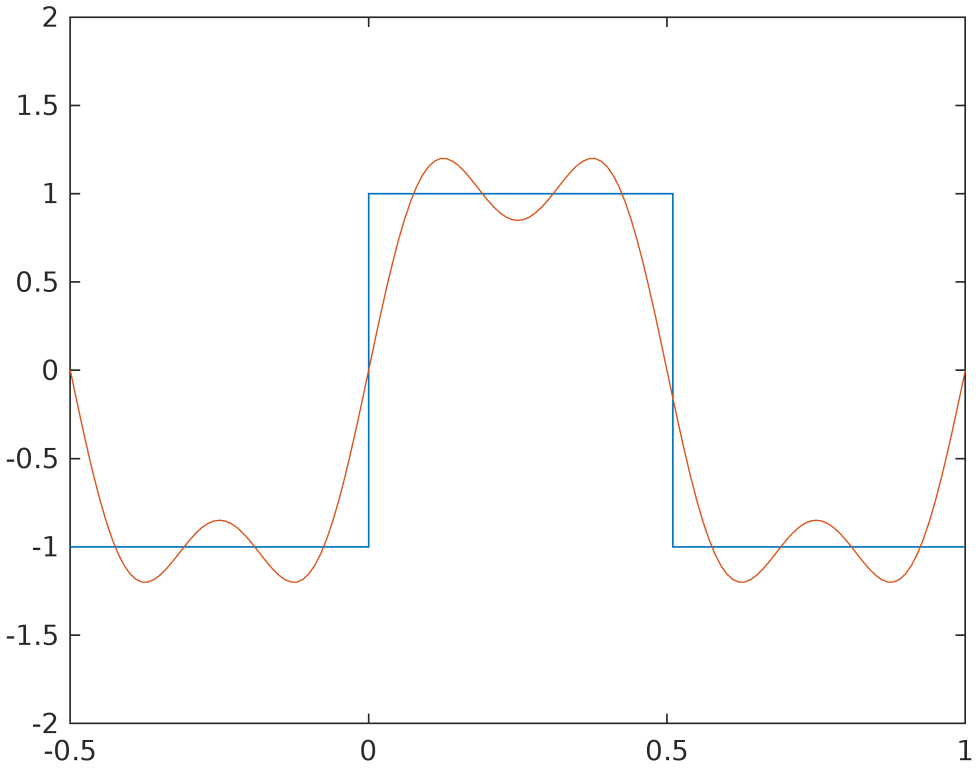


Representación de señales como suma de sinusoides

Una señal se puede aproximar como la suma de señales sinusoidales. Podemos generar esta señal a partir de un ciclo for (floor redondea hacia -infinito):

```
clearvars

t = -0.5:0.01:1;                    % Equivalente a 1 1/2 ciclos.
N = length(t);
d = -ones(1,N);                     % Onda cuadrada.
d(1,floor(N/3)+1:2*floor(N/3)+1) = 1;
y_t = sin(2*pi*t);                  % Sinusoidal del igual periodo.
fin = 3;                            % Cantidad de sinusoides.
for i = 3:2:fin                     % Ciclo for.
    y_t = y_t + sin(i*2*pi*t)/i;    % i se utiliza como coeficientes
                                    % de Fourier.
end                                 % El último valor de i = fin.
figure
stairs(t,d)                         % Gráfico de escaleras para
hold on                             % visualizar funciones discontinuas.
plot(t,(4/pi)*y_t)
hold off
```

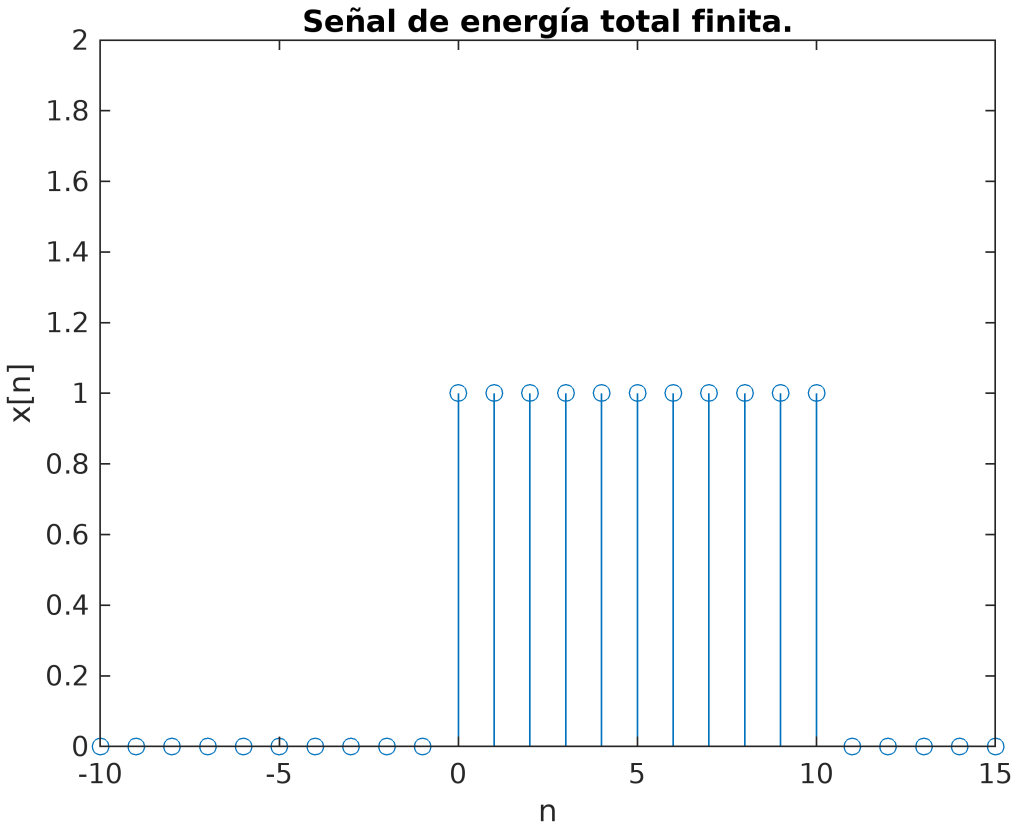
```
ylim([-2 2])
```



Energía y potencia

Señal de energía total finita (en el intervalo $0 \leq n \leq 10$) x[n] = 1 para $0 \leq n \leq 10$ x[n] = 0 en el resto:

```
clearvars

n_T = 0:10;                                  % Al omitir :paso: es = 1.
T = length(n_T);                             % Tamaño del intervalo.
n = [-10:-1 n_T 11:15];
x_n = [zeros(1,10) ones(1,T) zeros(1,5)];  % Se añaden zeros (total 26
                                             % muestras).
figure
stem(n,x_n)
title('Señal de energía total finita.')
xlabel('n')
ylabel('x[n]')
ylim([0 2])
```

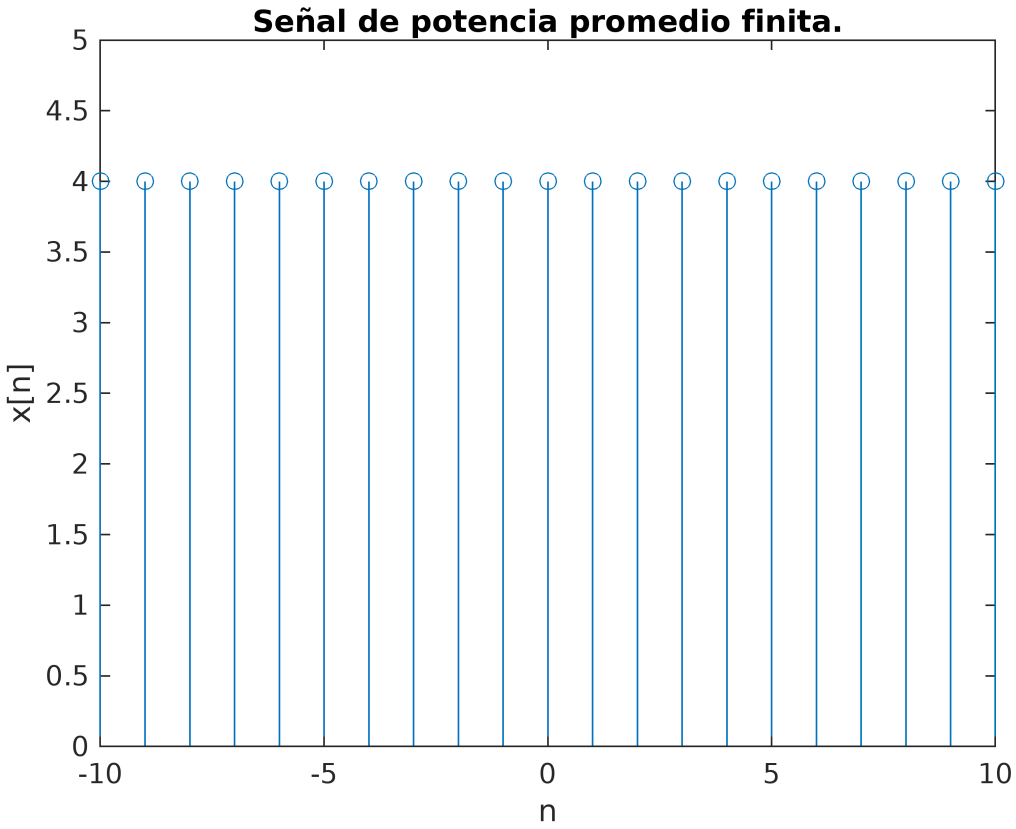



```
E_inf_rect = sum(abs(x_n).^2)/T            % Energía total.
```

```
E_inf_rect = 1
```

Señal de potencia promedio finita x[n] = 4 para todo n (sum suma todos los valores):

```
N_v = 10;
n = -10:1:10;
N = length(n);          % N es el número finito de muestras.
x_n = ones(1,N)*4;      % x[n] = 4.
figure
stem(n,x_n)
title('Señal de potencia promedio finita.')
xlabel('n')
ylabel('x[n]')
ylim([0 5])
```




```
P_prom_const = (1/(2*N_v+1))*sum(x_n.^2) % Potencia promedio.
```

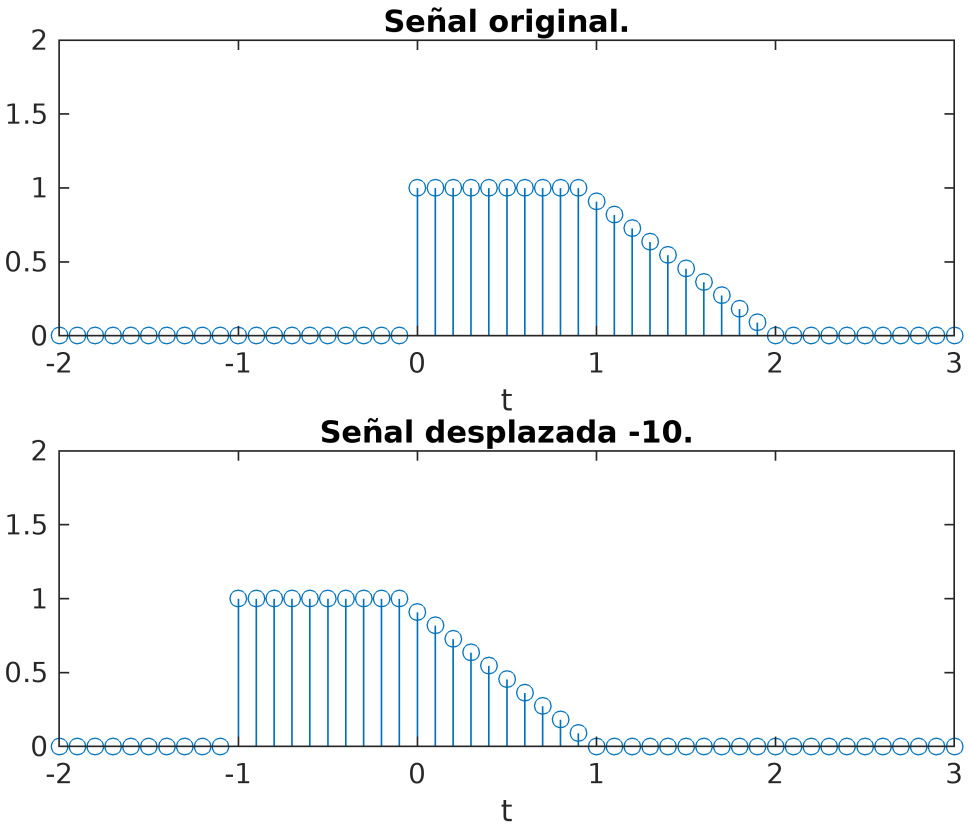
```
P_prom_const = 16
```

Transformaciones en variable independiente

Desplazamiento de tiempo (shifting):

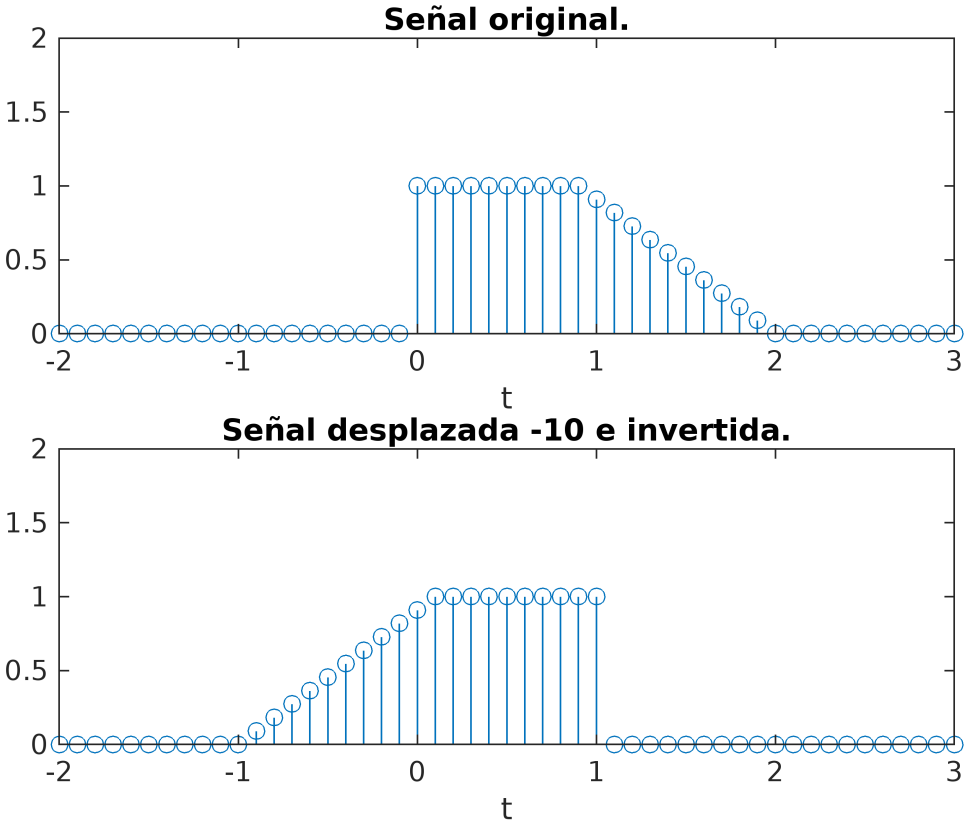
```
clearvars

t = 0:0.1:5;
T = 10;                          % 10 muestras serán equivalente a 1.
y = zeros(1,length(t));
y(1,1:2*T+1) = 1;                % Señal Figura 1.13 Oppenheim.
y(1,T:2*T+1) = y(1,T:2*T+1)-linspace(0,1,T+2);
t = [-5:0.1:-0.1 t];
y = [zeros(1,50) y];
y_shift = circshift(y,-10);      % Desplaza la señal de manera circular
                                 % manteniendo el número de muestras.
figure
subplot(2,1,1)                   % Subgráficos de 2 filas 1 columna.
stem(t,y)
xlim([-2 3])
```

```
ylim([0 2])
title('Señal original.')
xlabel('t')
subplot(2,1,2)
stem(t,y_shift)
xlim([-2 3])
ylim([0 2])
title('Señal desplazada -10.')
xlabel('t')
```



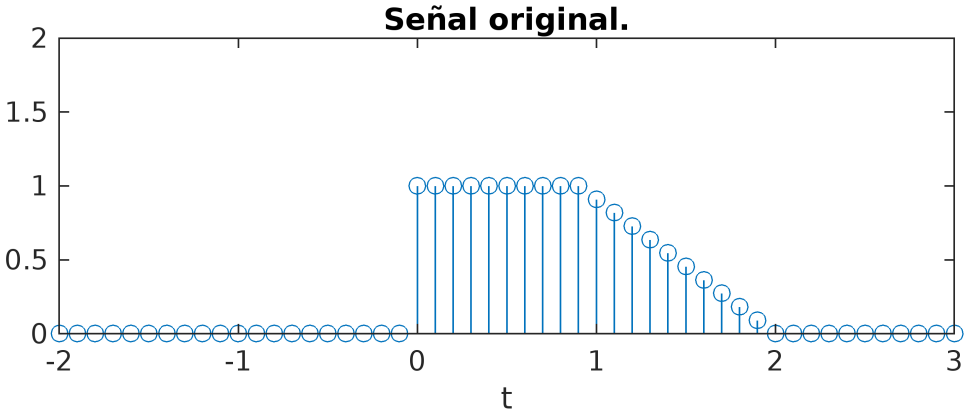
Inversión:

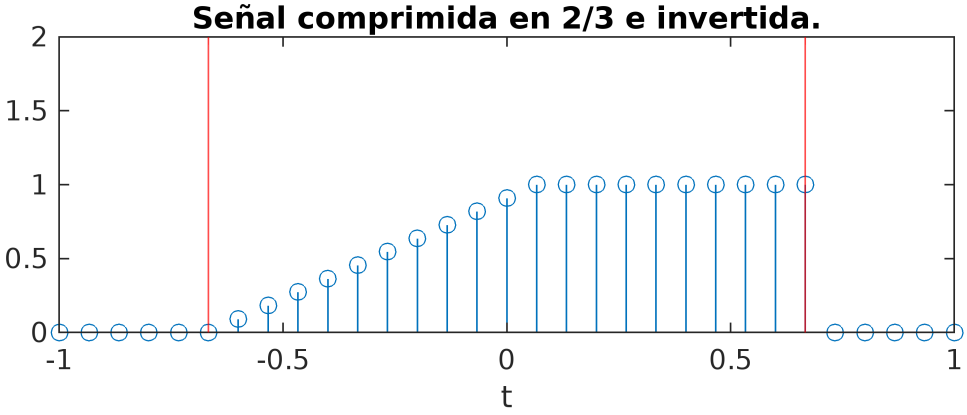
```
figure
subplot(2,1,1)
stem(t,y)
title('Señal original.')
xlabel('t')
xlim([-2 3])
ylim([0 2])
subplot(2,1,2)
stem(-t,y_shift)                          % -t invierte el gráfico.
title('Señal desplazada -10 e invertida.') % El valor de t no cambia.
xlabel('t')
xlim([-2 3])
ylim([0 2])
```



Contracción:

```
figure
subplot(2,1,1)
stem(t,y)
xlim([-2 3])
ylim([0 2])
title('Señal original.')
xlabel('t')
subplot(2,1,2)
stem(-t*2/3,y_shift)   % Se comprime los valores de t para el gráfico.
xlim([-1 1])           % Los valores de t no cambian.
hold on
xline(-2/3,'-r')       % Línea vertical roja en x = -2/3.
hold on
xline(2/3,'-r')        % Línea vertical roja en x = 2/3.
title('Señal comprimida en 2/3 e invertida.')
xlabel('t')
ylim([0 2])
```

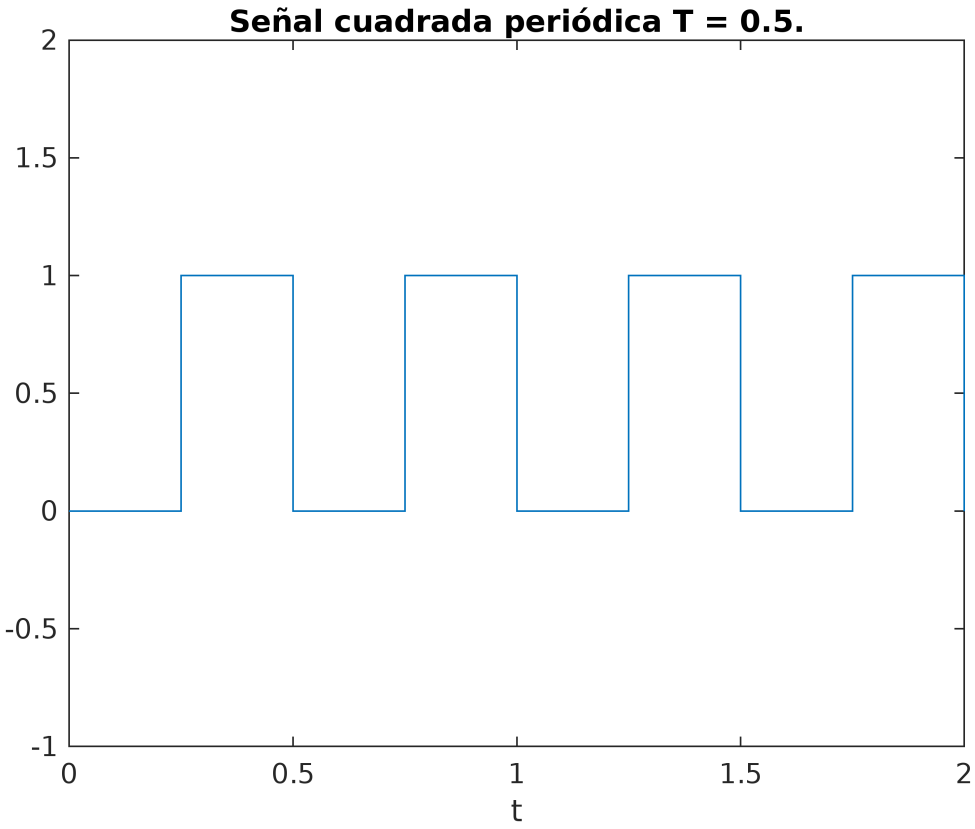




Señales periódicas

En Matlab podemos utilizar la función gensig para generar señales periódicas.

```
T = 0.5;                            % Periodo = T.
Tf = 2;                             % Tf/T periodos en total.
[u0,t] = gensig("square",T,Tf);
figure
stairs(t,u0)
ylim([-1 2])
title('Señal cuadrada periódica T = 0.5.')
xlabel('t')
```

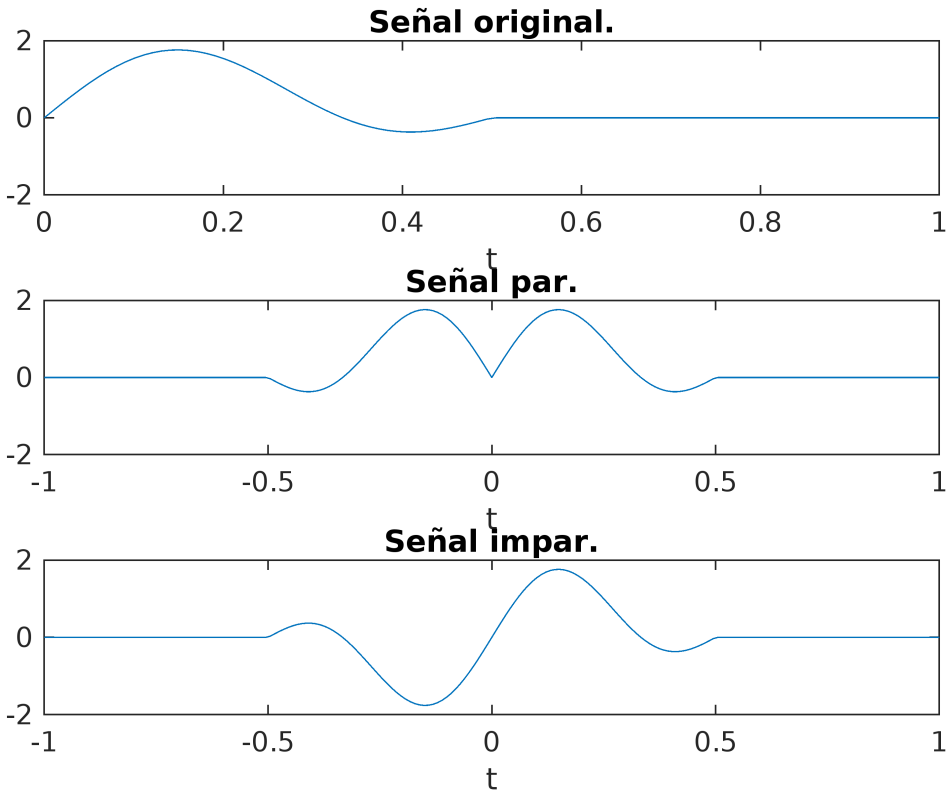


Señales par e impar

Podemos generar señales pares es impares a partir de una señal de largo finito.

```
N = 100;
t = linspace(0,1,N);
y = sin(2*pi*t)+sin(4*pi*t);
y(N/2+1:end) = 0;               % Señal. Para vectores podemos omitir
figure                          % el primer numero en la indexación.
subplot(3,1,1)
plot(t,y)
ylim([-2 2])
title('Señal original.')
xlabel('t')

t_s = [-fliplr(t(2:end)) t];    % fliplr invierte la señal de izquierda
y_s = [fliplr(y(2:end)) y];     % a derecha.
subplot(3,1,2)
plot(t_s,y_s)
ylim([-2 2])
title('Señal par.')
xlabel('t')

y_as = [-fliplr(y(2:end)) y];   % Negando el valor de la señal se vuelve
subplot(3,1,3)                  % impar.
```

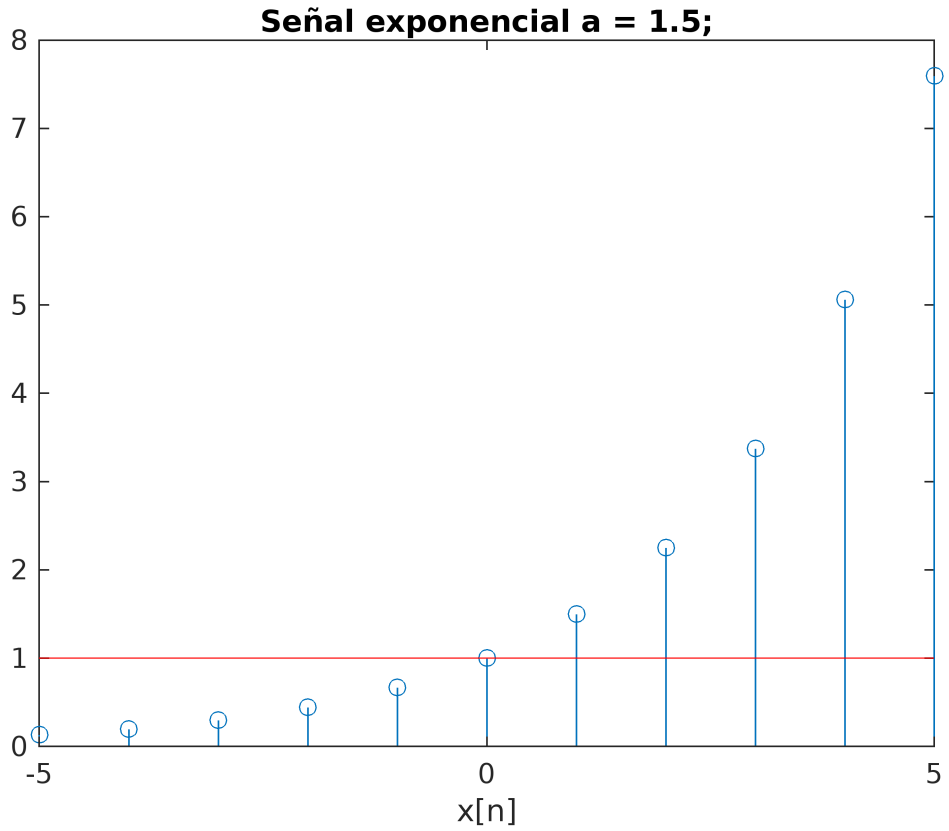
```
plot(t_s,y_as)
ylim([-2 2])
title('Señal impar.')
xlabel('t')
```



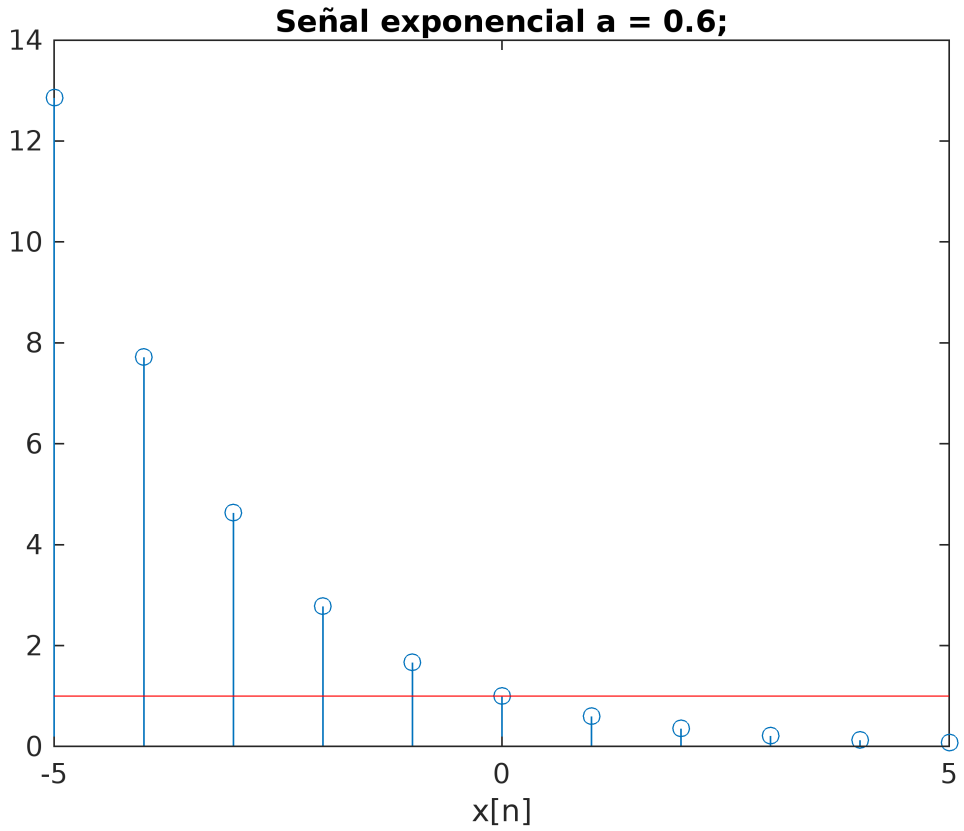
Señales exponenciales y sinusoidales discretas

Exponenciales:

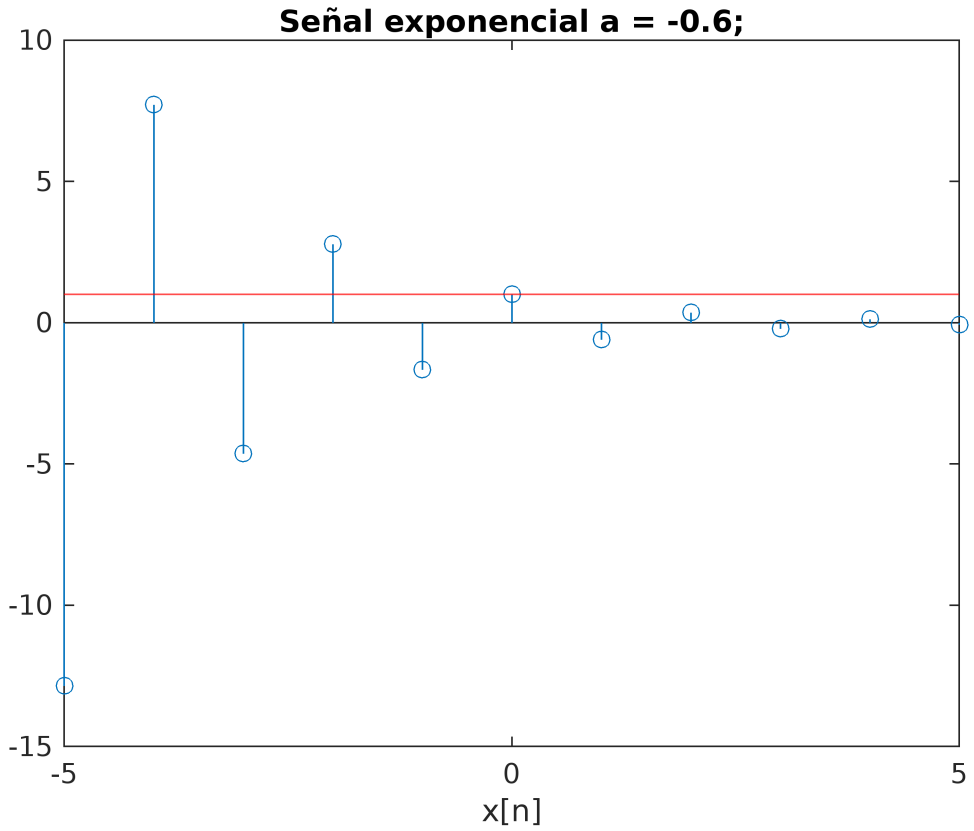
```
n = -5:5;
C = 1;
a = 1.5;
y = C*a.^n;
figure
stem(n,y)
hold on
yline(1,'-r')                          % Línea horizontal en y = 1
title('Señal exponencial a = 1.5;')    % de color rojo.
xlabel('x[n]')
```



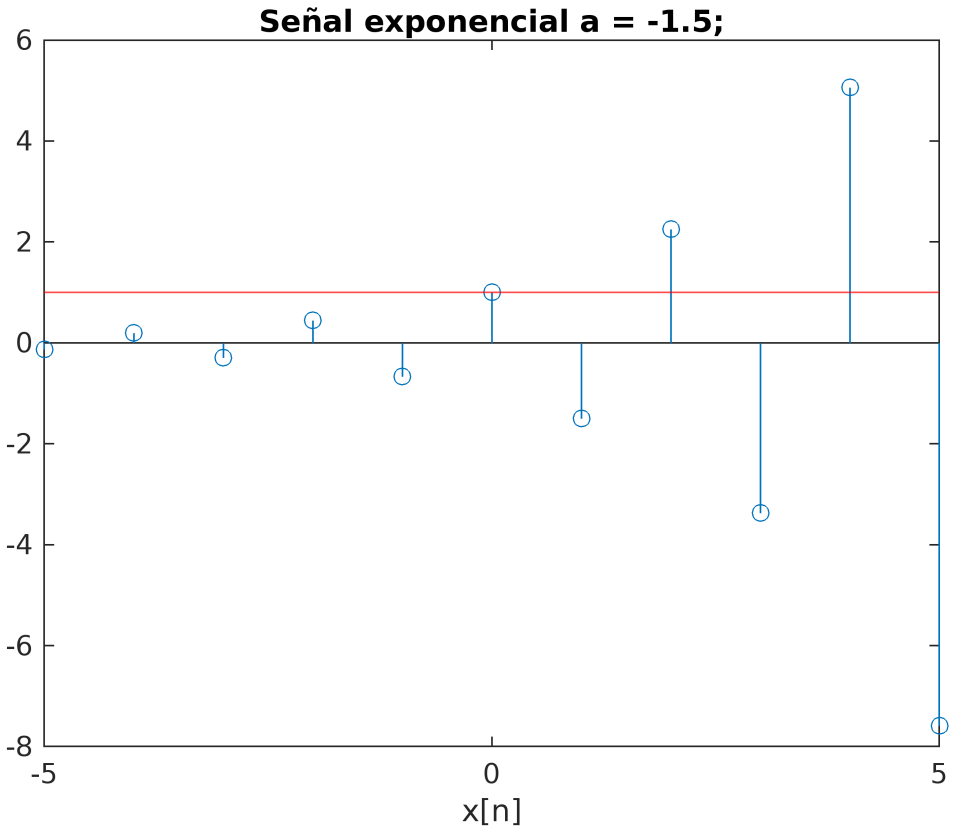

```
a = 0.6;
y = C*a.^n;
figure
stem(n,y)
hold on
yline(1,'-r')
title('Señal exponencial a = 0.6;')
xlabel('x[n]')
```




```
a = -0.6;
y = C*a.^n;
figure
stem(n,y)
hold on
yline(1,'-r')
title('Señal exponencial a = -0.6;')
xlabel('x[n]')
```

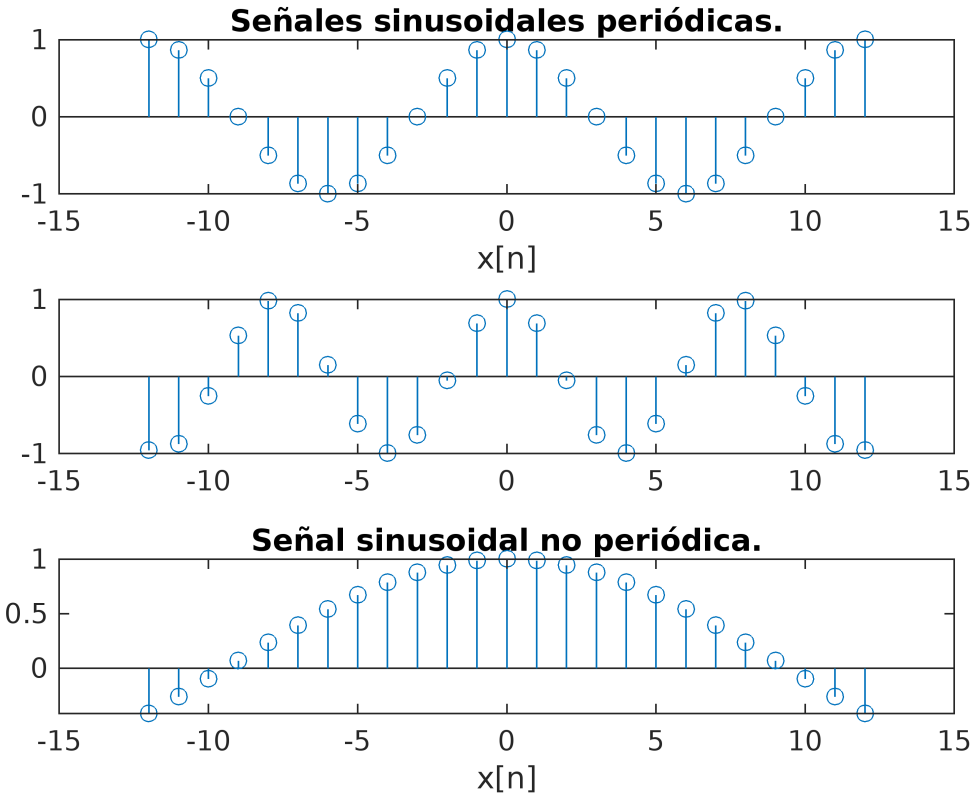



```
a = -1.5;
y = C*a.^n;
figure
stem(n,y)
hold on
yline(1,'-r')
title('Señal exponencial a = -1.5;')
xlabel('x[n]')
```



Sinusoidales:

```
n = -12:12;
y = cos(2*pi*n/12);
figure
subplot(3,1,1)
stem(n,y)
title('Señales sinusoidales periódicas.')
xlabel('x[n]')
y = cos(8*pi*n/31);
subplot(3,1,2)
stem(n,y)

y = cos(n/6);                % No periódica.
subplot(3,1,3)
stem(n,y)
title('Señal sinusoidal no periódica.')
xlabel('x[n]')
```



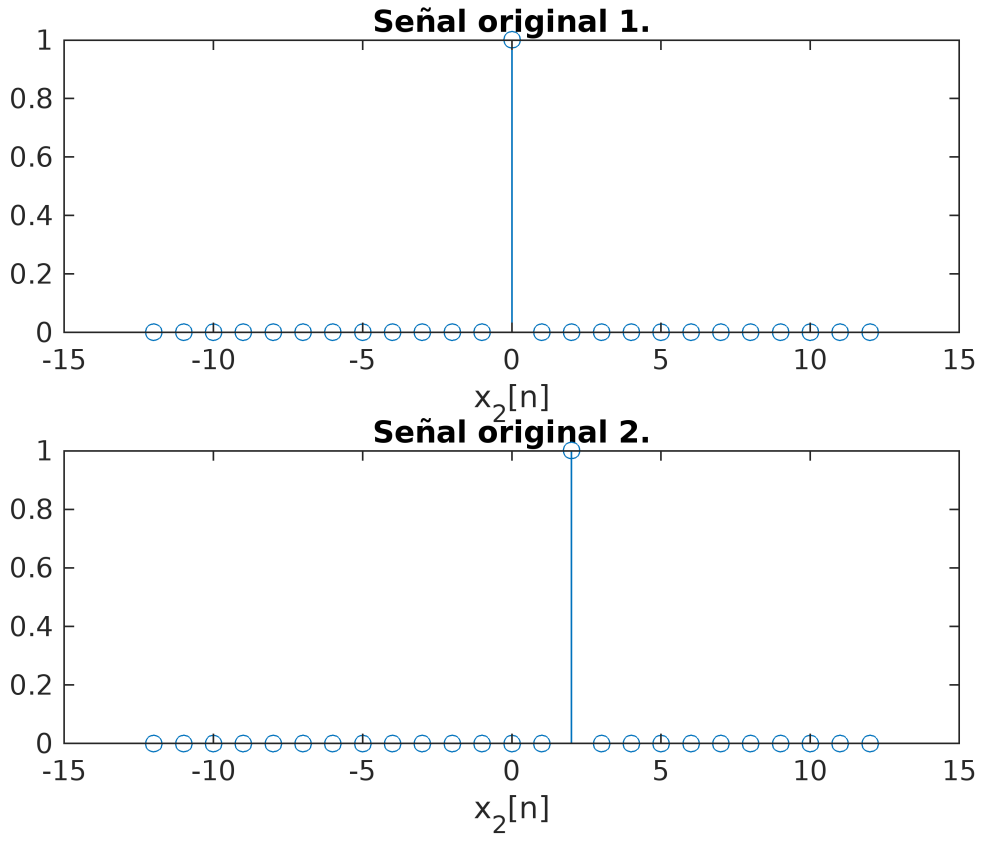
Sistema discreto

Podemos probar la linealidad de un sistema discreto que desplaza una señal en 3 muestras y la escala en 0.8 con dos señales distintas.

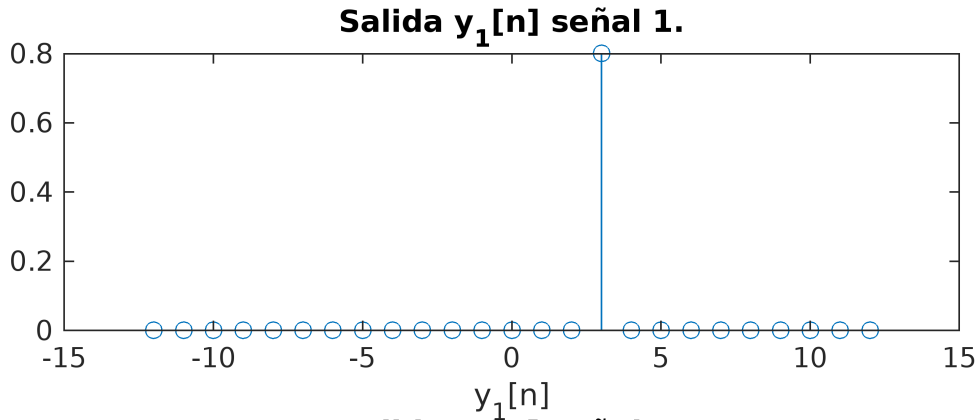
$y[n] = 0.8x[n-3]$

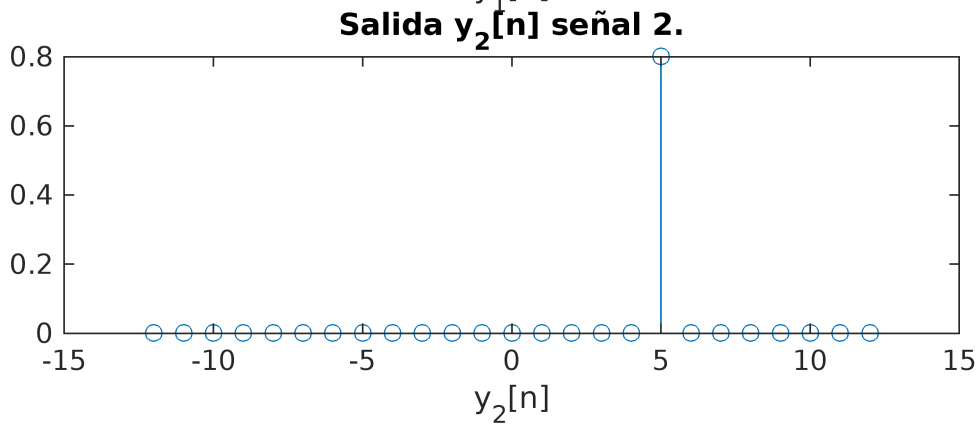
```
zpv = 12;
n = -zpv:zpv;
x_n_1 = zeros(1,length(n));
x_n_1(zpv+1) = 1;               % Señal impulso unitario.
x_n_2 = circshift(x_n_1,2);     % Señal impulso unitario desplazada en
                                % 2 muestras.

figure
subplot(2,1,1)
stem(n,x_n_1)
title('Señal original 1.')
xlabel('x_2[n]')
subplot(2,1,2)
stem(n,x_n_2)
title('Señal original 2.')
```

```
xlabel('x_2[n]')
```

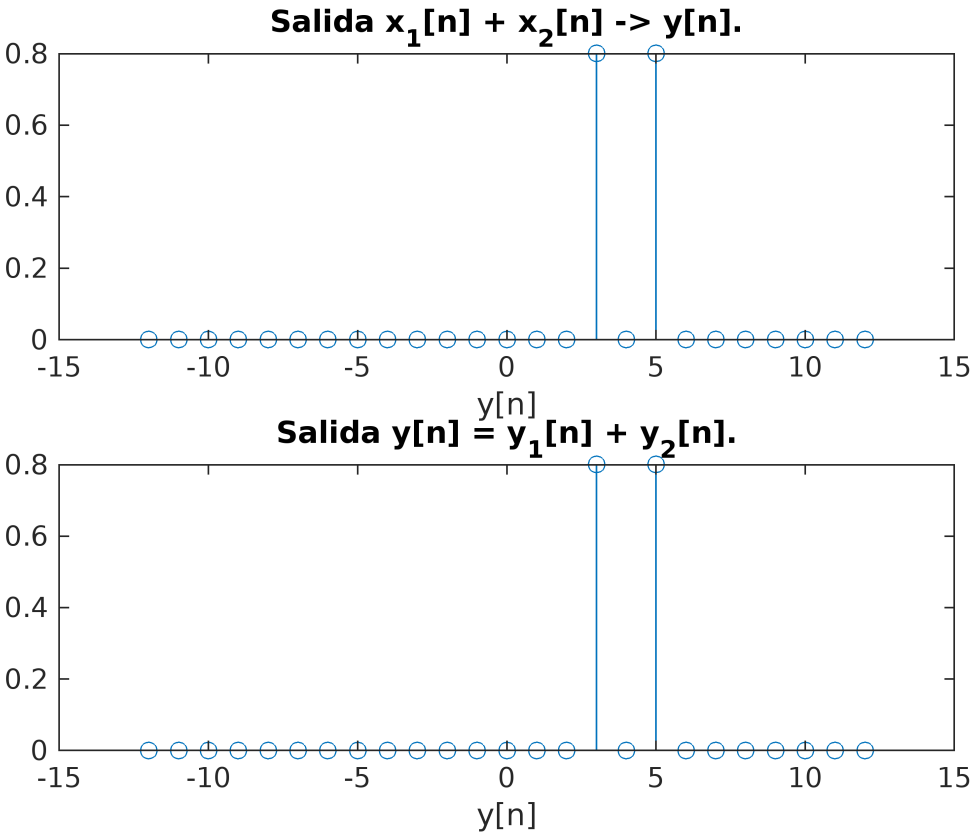



```
y_1 = 0.8*circshift(x_n_1,3); % Dos salidas a partir de ingresar ambas
y_2 = 0.8*circshift(x_n_2,3); % señales al sistema por separado.
figure
subplot(2,1,1)
stem(n,y_1)
title('Salida y_1[n] señal 1.')
xlabel('y_1[n]')
subplot(2,1,2)
stem(n,y_2)
title('Salida y_2[n] señal 2.')
xlabel('y_2[n]')
```






```
y_12_1 = 0.8*circshift(x_n_1+x_n_2,3); % Salida al ingresar una suma de
y_12_2 = y_1 + y_2;                    % las señales.
figure
subplot(2,1,1)
stem(n,y_12_1)
title('Salida x_1[n] + x_2[n] -> y[n].')
xlabel('y[n]')
subplot(2,1,2)
stem(n,y_12_2)
title('Salida y[n] = y_1[n] + y_2[n].')
xlabel('y[n]')
```



Ejercicio

Cree una señal **discreta** a partir de la función impulso unitario $\delta[n]$ definida como:

$$\delta[n] = \begin{cases} 0, & n \neq 0 \\ 1, & n = 0 \end{cases}$$

para el intervalo $-10 \leq n \leq 10$. A partir de esta señal $\delta[n]$, cree la señal escalón unitario $u[n]$ **discreta** para el mismo intervalo utilizando la transformación de corrimiento (shifting) vista anteriormente y una iteración con ciclo for. Se evalúa los conceptos en formato de texto, los comentarios dentro del código, la exactitud del algoritmo y la calidad de los gráficos generados. Muestre solo los valores más importantes.

Referencias

[1] Oppenheim, A.V. & Willsky, A.S. & Nawab, S.H. (1997). Señales y sistemas (2nd ed.). Prentice Hall.