

Universidad Tecnológica de Panamá
Facultad de Ingeniería Eléctrica

Señales y Sistemas

con Octave



Carlos A. Medina C.
Héctor Poveda



Medina, Carlos y Poveda, Héctor. 2021

© 2021, Material didáctico para el Curso de Señales y Sistemas por Medina, Carlos y Poveda, Héctor. Universidad Tecnológica de Panamá (UTP).

Obra bajo Licencia Creative Commons Atribución-NoComercial-CompartirIgual 4.0 Internacional.
Para ver esta licencia: <https://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>

Fuente del documento Repositorios Institucional UTP-Ridda2:
[http://ridda2.utp.ac.pa/...](http://ridda2.utp.ac.pa/)

Prefacio

Este **documento** ha sido creado por Carlos A. Medina C. y Héctor Poveda como material didáctico de apoyo para ser utilizado, principalmente, por estudiantes de ingeniería en el proceso de aprendizaje del curso de "Señales y Sistemas". Es un documento útil como referencia rápida y de apoyo sobre los conceptos más relevantes y el uso de una herramienta de programación - Octave - para modelar, simular y analizar estos conceptos. Sin embargo, *este documento no es un tutorial ni mucho menos un sustituto de un libro de texto* – el cual es necesario para la comprensión y aprendizaje completo de la materia.

El conocimiento en señales y sistemas es crucial para los estudiantes que se especializan en muchas áreas de la ingeniería, pero principalmente en eléctrica y electrónica, ya que los prepara para estudiar materias avanzadas sobre procesamiento de señales, comunicación, control y sistemas de potencia, entre otros.

Se supone que los estudiantes que toman este curso conocen los conceptos básicos sobre álgebra lineal, cálculo (en números complejos, diferenciación e integración) y ecuaciones diferenciales.

El objetivo principal de este **material didáctico** es facilitar a los estudiantes el aprendizaje de los conceptos básicos sobre señales y sistemas con el uso de Octave como herramienta para profundizar y comprender estos conceptos. En cada tema, integramos ampliamente el uso del software con códigos de ejemplo y resultados, así como el análisis de los mismos. Esto, con el propósito de ayudar a los estudiantes a comprender el significado físico, la interpretación y / o la aplicación de conceptos, tales como: convolución, correlación, respuesta de tiempo / frecuencia, análisis de Fourier, filtros, y otros.

Esperamos que este **material didáctico** los incentive, queridos estudiantes, a interesarse por el curso de Señales y Sistemas y el uso de Octave para el análisis y aprendizaje, de forma que les permita en un futuro ser capaces de desarrollar sus propios códigos para resolver otros problemas.

Signals and systems is one of the toughest classes you'll take as an engineering student. But struggling to figure out this material doesn't necessarily mean you need to sprout early-onset gray hairs and resign yourself to frown lines in your college years. And you definitely don't want to give up on engineering over this stuff because becoming an engineer is, in my opinion, one of the best career choices you can make. See, you're no dummy!

Mark Wickert, PhD
Professor of Electrical and Computer
Engineering, University of Colorado,
Colorado Springs

**Signals
& Systems**
FOR DUMMIES



Contenido

Introducción a Octave

- Ambiente de trabajo
- Ventana de comandos
- Operadores y caracteres especiales
- Matrices y vectores
- Operaciones y funciones
- Números complejos
- Operadores relacionales y lógicos
- Archivos m
- Gráficos
- Programación

1. Señales y sistemas

- Señales y sistemas – definiciones
- Operaciones con señales
- Clasificación de señales
- Señales de interés

2. Sistemas lineales invariantes en el tiempo

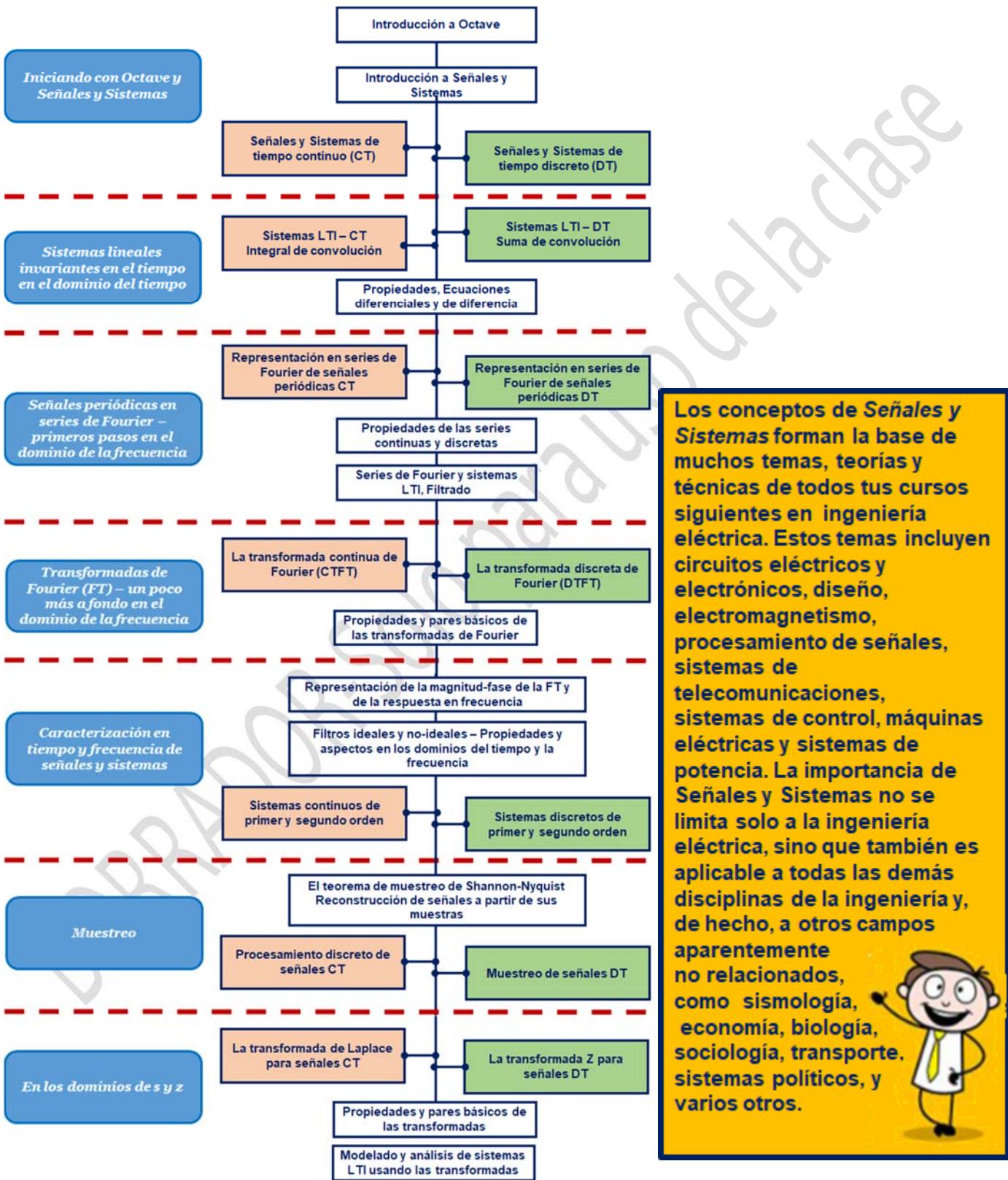
- Sistemas LTI
- La suma de convolución
- La integral de convolución
- La correlación



BORRADOR. Solo para uso de la clase



¿Qué vamos a estudiar?



Introducción a Octave



Octave es un programa para realizar cálculos numéricos y simbólicos, muy utilizado en ingeniería y ciencias, así como en matemáticas.

Esta es una breve introducción diseñada para brindarte una manera rápida de familiarizarte y trabajar con el software. Veremos sus características, entornos, y algunos comandos y funciones básicas.

Características básicas:

Proporciona muchos de los atributos de las hojas de cálculo y los lenguajes de programación.

En el modo interactivo, los scripts son independientes de la plataforma (proporciona portabilidad).

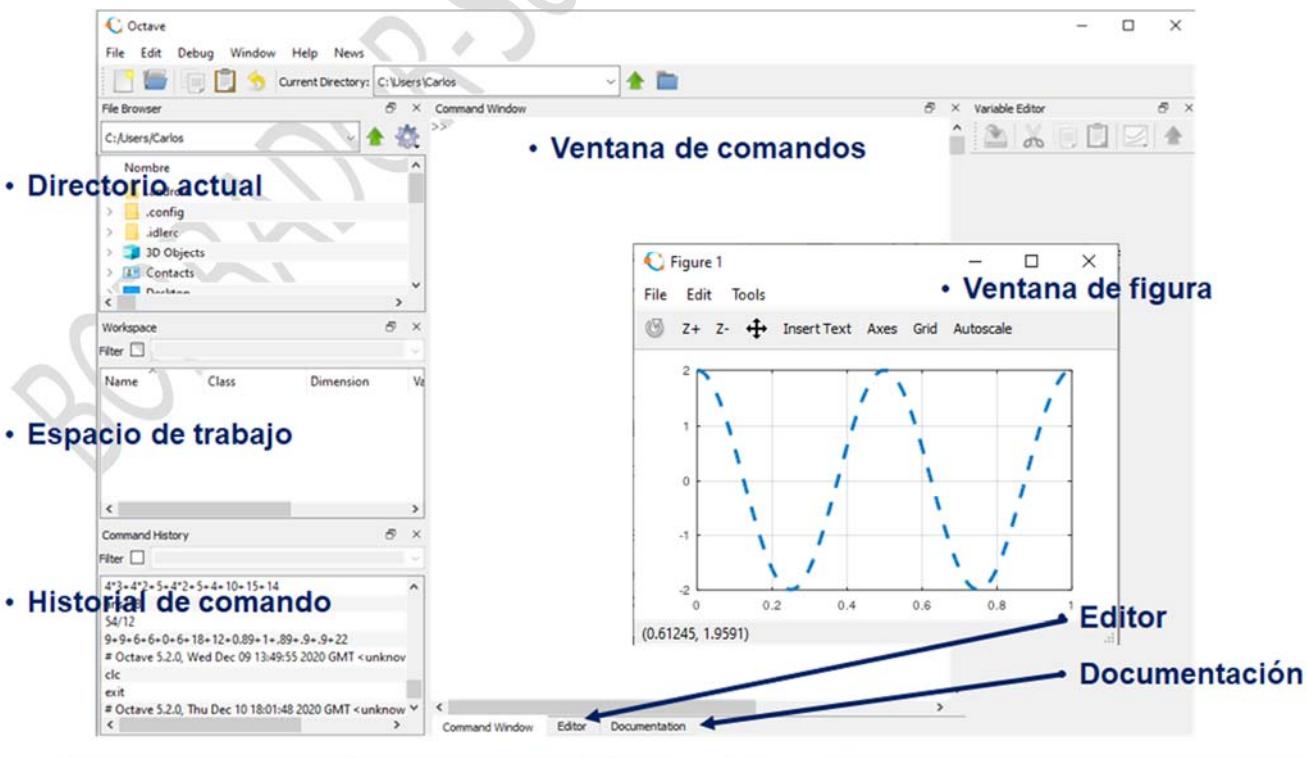
Funciona con matrices: todo es una matriz (desde texto hasta matrices de celdas grandes y matrices de estructura).

Es un lenguaje sensible a mayúsculas y minúsculas.

Usos típicos de Octave:

- Cálculos matemáticos
- Desarrollo algorítmico
- Creación de prototipos de modelos (antes del desarrollo de modelos complejos)
- Análisis de datos y exploración de datos (visualización)
- Gráficos científicos y de ingeniería para presentación
- Análisis complejo usando cajas de herramientas (e.g., estadística, redes neuronales, lógica difusa, control, economía, comunicaciones, procesamiento de señales, etc.).

Ambiente de trabajo



Ventana de comandos

The screenshot shows the Octave graphical user interface. The Command Window at the bottom contains the following text:

```
>> 2+4.2
ans = 6.2000
```

A red circle highlights the value `6.2000`. A callout box points to it with the text: "Todos los comandos se ejecutan desde este indicador. Si no se asigna una variable, el último resultado se guarda en la variable `ans`. Por defecto, Octave devuelve expresiones numéricas como decimales con 5 dígitos."

```
>> format rat
>> 5.1-3.3
ans = 9/5
>> format compact
>> 5*7
ans = 35
>> format short e
>> 2*1000
ans = 2.0000e+03
```

A callout box points to the command `format rat` with the text: "La función `format` sirve para cambiar el formato de la salida. Por ejemplo, `format rat` devuelva expresiones racionales. Otros son, por ejemplo, `format compact`, `format short e`, ...".

```
>> help abs
'abs' is a built-in function from the file libinterp/corefcn/mappers.cc
-- abs (Z)
    Compute the magnitude of Z.

    The magnitude is defined as |Z| = 'sqrt (x^2 + y^2)'.

    For example:
        abs (3 + 4i)
        => 5
    See also: arg.
```

A callout box points to the command `help abs` with the text: "La función `help` permite acceder a la documentación de una función particular de interés."

```
>> % 5+6
>>
```

A callout box points to the percentage sign (%) with the text: "Un signo de porcentaje (%) es un comentario y se ignora."

```
>> x = 25
x = 25
>> x^2+3*x-(6*x+4*sqrt(x))
ans = 530
>>
```

Los valores de las variables se asignan directamente con el signo de igual.

Las operaciones matemáticas siguen el orden de precedencia estándar:

potenciación - multiplicación y división - suma y resta.

Las expresiones se evalúan de izquierda a derecha.

Los paréntesis funcionan de adentro hacia afuera.

```
>> b = 1+2+3+...
4+5+6
b = 21
```

Si la expresión no cabe en una línea, utilice puntos suspensivos (tres puntos al final de la línea) y continúe en la siguiente línea.



Hay varias *variables predefinidas* que se pueden utilizar en cualquier momento, de la misma manera que las variables definidas por el usuario.

| | |
|----|-----------|
| i | sqrt(-1) |
| j | sqrt(-1) |
| pi | 3.1416... |



```
>> y = pi*(1+2*j)
y = 3.1416 + 6.2832i
```

```
Command Window
>> x=5.5;
>> y=x+2;
>> who
Variables in the current scope:
```

El ";" suprime la impresión en pantalla.

```
x y
```

```
>> whos
Variables in the current scope:
```

| Attr | Name | Size | Bytes | Class |
|-------|-------|-------|-------|--------|
| ===== | ===== | ===== | ===== | ===== |
| | x | 1x1 | 8 | double |
| | y | 1x1 | 8 | double |

```
Total is 2 elements using 16 bytes
```

```
>> clear
>> who
>> x
error: 'x' undefined near line 1 column 1
>>
```

El comando **who** devuelve una lista de todas las variables en el espacio de trabajo actual.

El comando **whos** devuelve la misma lista con información más detallada sobre cada variable.

El comando **clear** borra todas las variables del espacio de trabajo.

Operadores y caracteres especiales

| | |
|-----|--|
| + | Plus; addition operator. |
| - | Minus; subtraction operator. |
| * | Scalar and matrix multiplication operator. |
| / | division operator |
| .* | Array multiplication operator. |
| .^ | Scalar and matrix exponentiation operator. |
| .^ | Array exponentiation operator. |
| : | Colon; generates regularly spaced elements and represents an entire row or column. |
| () | Parentheses; encloses function arguments and array indices; overrides precedence. |
| [] | Brackets; enclosures array elements. |
| . | Decimal point. |
| ... | Ellipsis; line-continuation operator. |
| , | Comma; separates statements and elements in a row. |
| ; | Semicolon; separates columns and suppresses display. |
| = | Assignment (replacement) operator. |



También existe una multiplicidad de funciones predefinidas que se pueden usar directamente.



| | |
|-----------------------|---|
| <code>abs</code> | magnitude of a number (absolute value for real numbers) |
| <code>angle</code> | angle of a complex number, in radians |
| <code>cos</code> | cosine function, assumes argument is in radians |
| <code>sin</code> | sine function, assumes argument is in radians |
| <code>exp</code> | exponential function |
| <code>ceil(x)</code> | Round to smallest integer not less than x |
| <code>floor(x)</code> | Round to largest integer not greater than x |
| <code>round(x)</code> | Round towards nearest integer |
| <code>fix(x)</code> | Round towards zero |

Matrices y Vectores

```
>> A=[2 1 -1 8; 1 0 8 -3; 7 1 2 4]
A =
2 1 -1 8
1 0 8 -3
7 1 2 4
```

Las matrices se definen entre paréntesis []

```
>> A=[2,1,-1,8;1,0,8,-3;7,1,2,4]
A =
2 1 -1 8
1 0 8 -3
7 1 2 4
```

Los elementos de cada columna se separan por espacio o por coma, y las filas se separan por punto y coma o con un "enter" para cada fila.

```
>> A=[2 1 -1 8
1 0 8 -3
7 1 2 4]
A =
2 1 -1 8
1 0 8 -3
7 1 2 4
```

`>> A=[5 -2 9; 11 7 8]`

```
A =
5 -2 9
11 7 8
>> A'
ans =
5 11
-2 7
9 8
```

► `A'` representa la transpuesta de la matriz `A`.

Se pueden definir varias matrices especiales

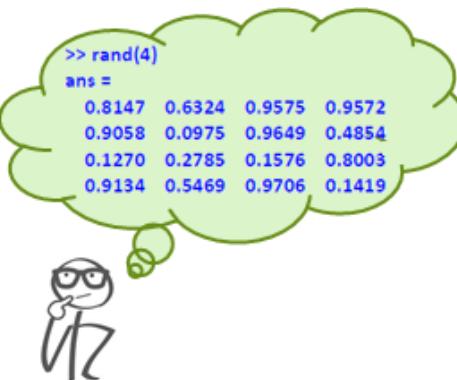
null matrix: `M = []`

`n x m` matrix of zeros: `M = zeros(n,m)`

`n x m` matrix of ones: `M = ones(n,m)`

`n x m` matrix of random numbers: `M = rand(n,m)`

`n x n` identity matrix: `M = eye(n)`



```
>> A=[3 -2 7 8; 4 3 2 1; 10 15 -2 9]
```

A =

| | | | |
|----|----|----|---|
| 3 | -2 | 7 | 8 |
| 4 | 3 | 2 | 1 |
| 10 | 15 | -2 | 9 |

```
>> A(3,2)
```

ans =
15

► A(i,j) representa al elemento de la fila i y la columna j de la matriz A

```
>> A(3,[2 4])
```

ans =

15 9

► Es posible extraer múltiples elementos de cualquier fila o columna.

Por ejemplo, el segundo y cuarto elementos de la tercera fila.

► También se puede extraer toda una fila i

o toda una columna j



```
>> v=[ 2; 3; -4]
```

v =
2
3
-4

► Los vectores (columna) son simplemente matrices con una sola columna

```
>> w=[3 -2 5 11]
```

w =
3 -2 5 11

► Los vectores (fila) son matrices con una sola fila.

```
>> 2:5
```

ans =
2 3 4 5
>> 3:2:-9

ans =
3 5 7 9

► Cuando se necesita un vector con elementos igualmente espaciados se usan los dos puntos (:) para generar rápidamente estos vectores

- aij = A(i,j) Get an element
- r = A(i,:) Get a row
- c = A(:,j) Get a column
- B = A(i:k,j:l) Get a submatrix

■ **Useful indexing command** end :

```
> data = [4 -1 35 9 11 -2];
> v = data(3:end)
v =
35 9 11 -2
```

Observe que si solo se dan dos números el incremento es 1, y que si se incluye un tercer número, el número intermedio define el incremento o decremento.

```
>> V = [1 3 5 7];
```

```
>> (V(5)=10;
```

```
>> V
```

V =

1 3 5 7 10

Se pueden incluir elementos adicionales a una matriz o vector.

```
>> a=[9 11];
```

```
>> b=[V a]
```

b =

1 3 5 7 10 9 11

Vectores definidos previamente se pueden usar para definir un nuevo vector o matriz.

| | | | | |
|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 |
| 2 | 2 | 2 | 2 | 2 |
| -3 | -3 | -3 | -3 | -3 |
| 4 | 4 | 4 | 4 | 4 |



```
> A(2,:)=[]
```

Deleting a Row/Column

• Assigning an empty matrix [] deletes the referenced rows or columns.

| | | | | |
|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 |
| -3 | -3 | -3 | -3 | -3 |
| 4 | 4 | 4 | 4 | 4 |

```
>> A=[1 2 3; 4 5 6]
```

A =

1 2 3

4 5 6

```
>> B=[7 8; 9 10]
```

B =

7 8

9 10

```
>> [A B]
```

ans =

1 2 3 7 8

4 5 6 9 10

```
>> C=[7 8 9]
```

C =

7 8 9

```
>> [A;C]
```

ans =

1 2 3

4 5 6

7 8 9

Matrices con el mismo número de filas se pueden concatenar horizontalmente

Matrices con el mismo número de columnas se pueden concatenar verticalmente.

El comando **length(x)** retorna la longitud del vector x.
El comando **size(x)** retorna la dimensión de la matriz x.



Operaciones y funciones

Las operaciones y funciones definidas para escalares, también se pueden usar con vectores y matrices.

Las funciones se aplican elemento a elemento.

$$\begin{array}{l} a = [1 \ 2 \ 3]; \\ b = [4 \ 5 \ 6]; \\ c = a + b \\ c = [5 \ 7 \ 9] \end{array}$$

Vector de tiempo – soporte de la función x.
Vector x con elementos iguales a la función evaluada para cada $t = 0, 1, 2, \dots, 10$

$$\begin{array}{l} t = 0:10; \\ x = \cos(2*t); \end{array}$$

Las operaciones de multiplicación, división, radicación y potenciación, si quieren implementarse elemento-a-elemento requieren usar un punto (.) antes del símbolo de la operación.

Por ejemplo, para obtener un vector x que contenga los elementos de $x(t) = t \cos(t)$, no se puede simplemente multiplicar el vector t con el vector $\cos(t)$. Es necesario multiplicar cada elemento correspondiente.

$$\begin{array}{l} t = 0:10; \\ x = t.*\cos(t); \end{array}$$

| t | $\cos(t)$ | $x=t.*\cos(t)$ |
|----------|-----------|----------------|
| 0.00000 | 1.00000 | 0.00000 |
| 1.00000 | 0.54030 | 0.54030 |
| 2.00000 | -0.41615 | -0.83229 |
| 3.00000 | -0.98999 | -2.96998 |
| 4.00000 | -0.65364 | -2.61457 |
| 5.00000 | 0.28366 | 1.41831 |
| 6.00000 | 0.96017 | 5.76102 |
| 7.00000 | 0.75390 | 5.27732 |
| 8.00000 | -0.14550 | -1.16400 |
| 9.00000 | -0.91113 | -8.20017 |
| 10.00000 | -0.83907 | -8.39072 |

```
>> x=t.*cos(t)
error: operator *: nonconformant arguments
```

Algunas funciones útiles para vectores y matrices

| | |
|----------|---------------------------------------|
| find | Finds indices of nonzero elements. |
| length | Computes number of elements. |
| linspace | Creates regularly spaced vector. |
| logspace | Creates logarithmically spaced vector |
| max | Returns largest element. |
| min | Returns smallest element. |
| prod | Product of each column. |
| reshape | Change size |
| size | Computes array size. |
| sort | Sorts each column. |
| sum | Sums each column. |

- cumsum(v) Compute cumulative sum of elements of v
- cumprod(v) Compute cumulative product of elements of v
- diff(v) Compute difference of subsequent elements [v(2)-v(1) v(3)-v(2) ...]
- mean(v) Mean value of elements in v
- std(v) Standard deviation of elements

```
>> X=[1 0 4 -3 0 0 0 8 6];
indices = find(X)
indices =
1 3 4 8 9
>> indices = find(X>2)
indices =
3 8 9
```



```
Command Window
>> x=[1 2 4 7]
x =
1 2 4 7
>> sum(x)
ans = 14
>> prod(x)
ans = 56
>> cumsum(x)
ans =
1 3 7 14
>> cumprod(x)
ans =
1 2 8 56
>> diff(x)
ans =
1 2 3
>> mean(x)
ans = 7/2
>> size(x)
ans =
1 4
>> max(x)
ans = 7
```



Números complejos

Los números imaginarios son de hecho reales, pero tienen un factor de escala multiplicativo especial los hace imaginarios. La necesidad de números imaginarios surge cuando intentas resolver una ecuación que involucra la raíz cuadrada de un número negativo. El factor de escala representa la raíz cuadrada de -1 y se denota con las letras i y j .



- El símbolo "i" o "j" identifica la parte imaginaria y debe escribirse inmediatamente después del valor numérico de la parte imaginaria, por ejemplo, $2 + 3i$.
 - Si se inserta un espacio, por ejemplo, $2 + 3 i$ – parece la misma expression pero será procesada como un número ($2 + 3$) y un carácter (i), y no como el número complejo ($2 + 3i$).
 - Es importante que sepas que la terminación con el carácter i solamente funciona con número simples, no con expresiones.
 - Por ejemplo, la expression $(1 - 2i)i$ no tiene significado. Si quieres multiplicar la expression por i, debes usar el símbolo de multiplicación (*).

```
>> 2+3i
ans = 2 + 3i
>> 2+3j
ans = 2 + 3i
>> 2+i^3
ans = 2 + 3i
>> 2+j^3
ans = 2 + 3i
```

Algunas funciones para números complejos

| | |
|------------------------------|---|
| <code>Complex(x,y)</code> | Return a complex number $x+yi$ |
| <code>real(x)</code> | real part of a complex number |
| <code>imag(x)</code> | imaginary part of a complex number |
| <code>Abs(x)</code> | magnitude of the complex number |
| <code>angle(x) arg(x)</code> | angle of a complex number x |
| <code>conj(x)</code> | complex conjugate of the complex number x |
| <code>Cart2pol</code> | Convert Cartesian to Polar form of complex number |
| <code>Pol2cart</code> | Convert Polar to Cartesian form of complex number |

```
>> x = complex(5,3)
x = 5 + 3i
>> real(x)
ans = 5
>> imag(x)
ans = 3
>> abs(x)
ans = 5.8310
>> angle(x)
ans = 0.54042
>> conj(x)
ans = 5 - 3i
>> cart2pol(real(x),imag(x))
ans =
    0.54042    5.83095
```



Operadores relacionales y lógicos

Relational Operators

- $x < y$ True if x is less than y
 - $x \leq y$ True if x is less than or equal to y
 - $x == y$ True if x is equal to y
 - $x \geq y$ True if x is greater than or equal to y
 - $x > y$ True if x is greater than y
 - $x \sim= y$ True if x is not equal to y

Boolean Expressions

- $x \& y$ Element-wise logical **and**
 - $x \mid y$ Element-wise logical **or**
 - $\sim x$ Element-wise logical **not**

```

Command Window
>> x = [1 2 3 4 5];
>> y = [1 3 5 7 5];
>> x < y
ans =
    0 1 1 1 0
>> x <= y
ans =
    1 1 1 1 1
>> x == y
ans =
    1 0 0 0 1
>> x > y
ans =
    0 0 0 0 0
>> x >= y
ans =
    1 0 0 0 1
>> x ~= y
ans =

```

```
Command Window

>>> x = [0 1 2 3 0 4]
>>> y = [0 0 1 2 0 0]
>>> x & y
ans =
    0   0   1   1   0   0
>>> x | y
ans =
    0   1   1   1   0   1
>>> ~x
ans =
    1   0   0   0   1   0
```

Archivos m (*scripts y funciones*)

Desarrollo de un script en el editor

Editor

File Edit View Debug Run Help

operv.m script2.m script.m

```
1 % script o programa para realizar algunas
2 % operaciones con vectores
3 clc; clear;
4 x = 1:2:10;
5 y = 1:5;
6 A = x+y;
7 B = x.*y;
8 C = 2*x-y;
9 D = x./y;
10
11 disp('Operaciones con vectores');
12 disp('Vector x'); disp(x);
13 disp('Vector y'); disp(y);
14 disp('x+y'); disp(A);
15 disp('x*y'); disp(B);
16 disp('2x-y'); disp(C);
17 format rat
18 disp('x/y'); disp(D);
19
```

nombre del archivo

El comando disp sirve para imprimir variables y mensajes en pantalla

Los archivos m son macros de comandos que se almacenan como texto ordinario con la extensión ".m", esto es, nombredearchivo.m

Un archivo m puede ser

- un **script**, esto es, una lista de comandos – programa.
- una **función**, esto es, un conjunto de comandos con variables de entrada y salida con operaciones particulares.



Resultados en la ventana de comandos

Command Window

Operaciones con vectores

| Vector x | 1 | 3 | 5 | 7 | 9 |
|----------|---|-----|-----|-----|-----|
| Vector y | 1 | 2 | 3 | 4 | 5 |
| x+y | 2 | 5 | 8 | 11 | 14 |
| x*y | 1 | 6 | 15 | 28 | 45 |
| 2x-y | 1 | 4 | 7 | 10 | 13 |
| x/y | 1 | 3/2 | 5/3 | 7/4 | 9/5 |

>> |

Desarrollo de una función en el editor

Editor

File Edit View Debug Run Help

operv.m script2.m

```
1 function [M] = operv(a,b)
2 % función para realizar algunas operaciones
3 % con vectores
4 A = a+b;
5 B = a.*b;
6 C = 2*a-b;
7 D = a./b;
8 M = [A;B;C;D];
9
```

El nombre del archivo m y de la función tienen que ser el mismo.

Note como los resultados de las operaciones se almacenan en las filas de una matriz M

Todas las variables definidas dentro de una función, son variables locales.

Script para obtener los mismos resultados del ejemplo anterior pero usando la función oper

Editor

File Edit View Debug Run Help

operv.m script2.m script.m

```
1 % script para realizar algunas
2 % operaciones con vectores
3 clc; clear;
4 x = 1:2:10;
5 y = 1:5;
6 Z = operv(x,y);
7
8 disp('Operaciones con vectores');
9 disp('Vector x'); disp(x);
10 disp('Vector y'); disp(y);
11 disp('x+y'); disp(Z(1,:));
12 disp('x*y'); disp(Z(2,:));
13 disp('2x-y'); disp(Z(3,:));
14 format rat
15 disp('x/y'); disp(Z(4,:));
```

Resultados en la ventanade commandos

Command Window

Operaciones con vectores

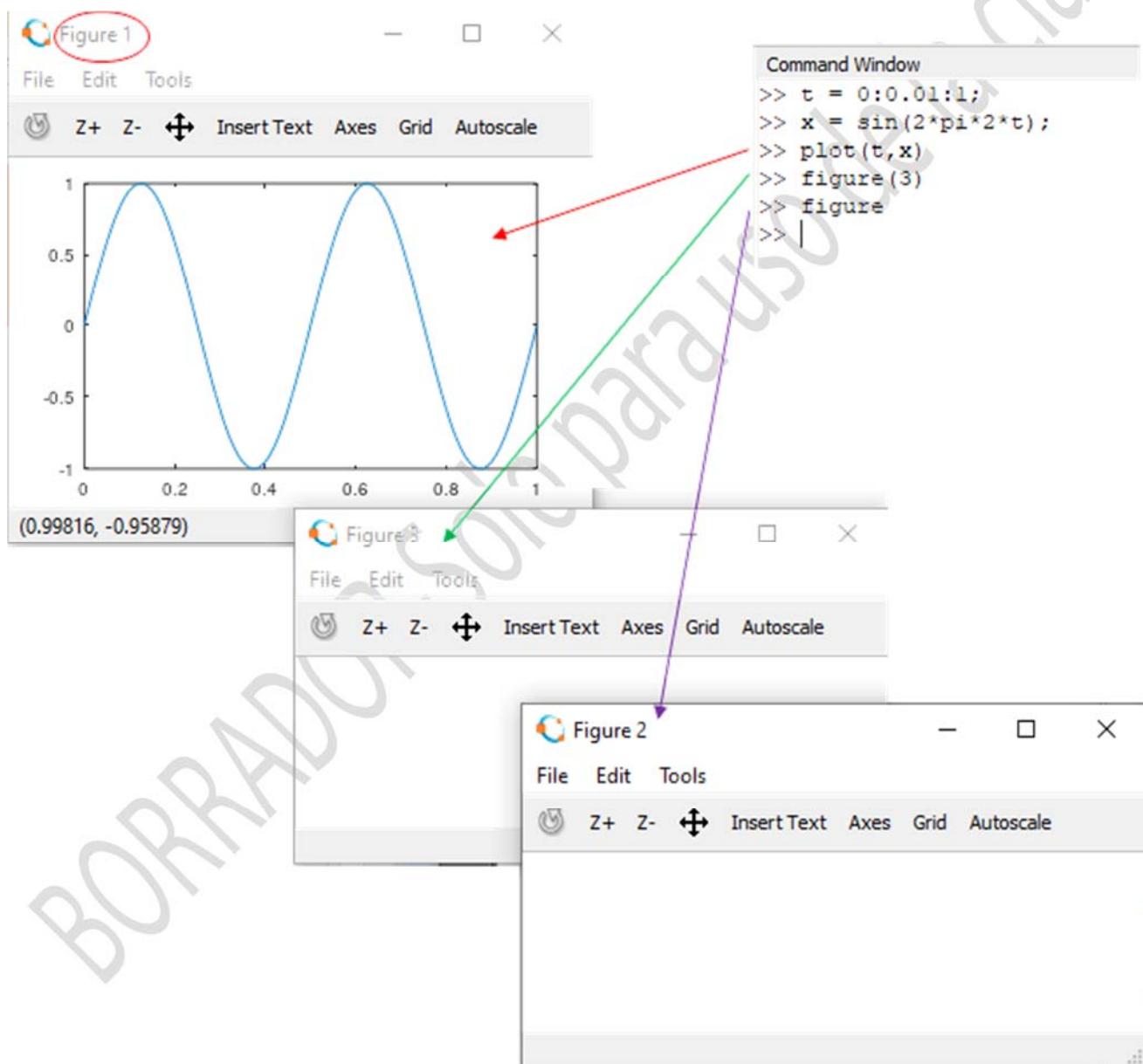
| Vector x | 1 | 3 | 5 | 7 | 9 |
|----------|---|-----|-----|-----|-----|
| Vector y | 1 | 2 | 3 | 4 | 5 |
| x+y | 2 | 5 | 8 | 11 | 14 |
| x*y | 1 | 6 | 15 | 28 | 45 |
| 2x-y | 1 | 4 | 7 | 10 | 13 |
| x/y | 1 | 3/2 | 5/3 | 7/4 | 9/5 |

>>

Gráficas

Gráficas en 2-dimensiones

- `plot(x,cos(x))` Crea automáticamente una ventana de figura y muestra una gráfica x-y de trazos continuos.
- `figure(n)` Crea una ventana de figura 'n' y la hace la figura activa
- `figure` Crea una nueva ventana de figura con el identificador incrementado en 1.



Varias gráficas

- series de patrones x-y `plot(x1,y1,x2,y2,...)`
e.g. `plot(x,cos(x),x,sin(x),x,cos(x).*sin(x))`

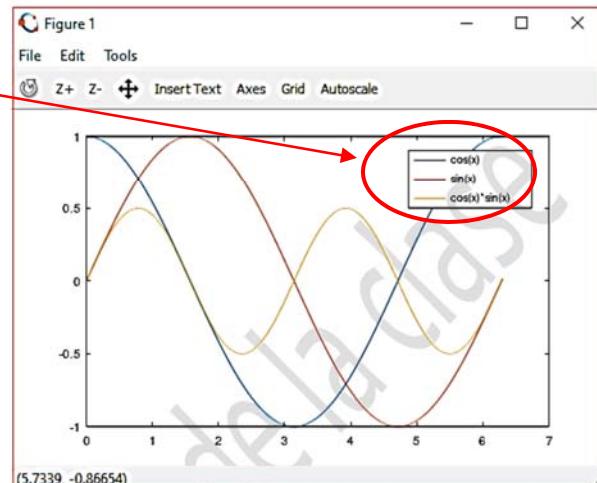
```
Command Window
>> x = 0:0.01:6.3;
>> plot(x,cos(x),x,sin(x),x,cos(x).*sin(x))
>> legend('cos(x)', 'sin(x)', 'cos(x)*sin(x)')
>>
```

- Incluye una leyenda a las gráficas

```
legend('cos(x)', 'sin(x)', 'cos(x)*sin(x)')
```

- Puede hacerse lo mismo con el comando **hold on**

```
> hold on; plot(x,cos(x));
> plot(x,sin(x));
> plot(x,cos(x).*sin(x));
```

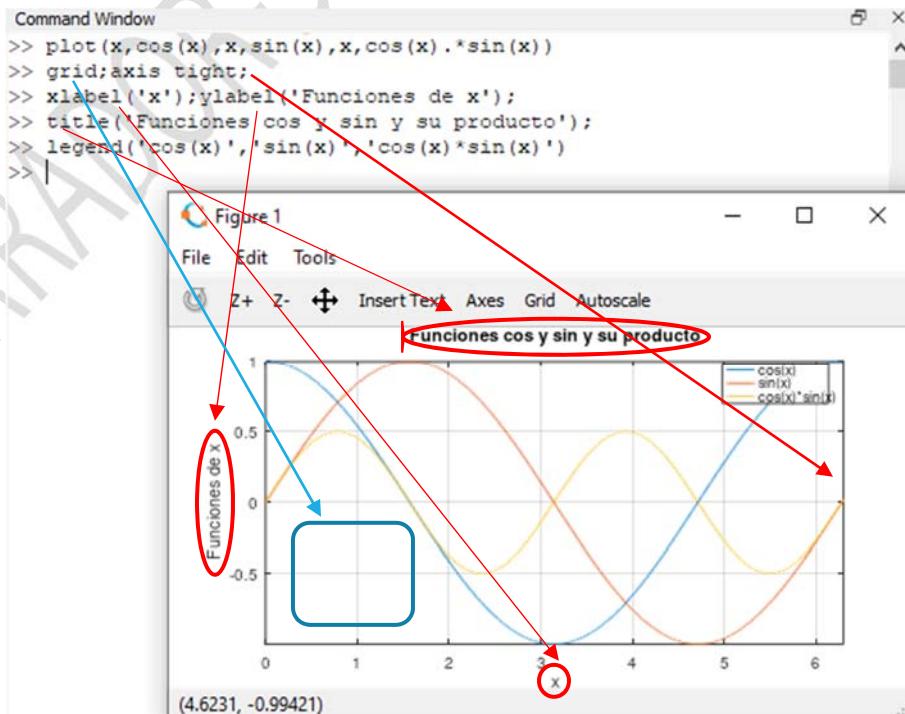


Comandos frecuentes para dar formato a las gráficas

| | |
|-----------------------------|--|
| <code>clf</code> | Clear figure |
| <code>hold on</code> | Hold axes. Don't replace plot with new plot, superimpose plots |
| <code>grid on</code> | Add grid lines |
| <code>grid off</code> | Remove grid lines |
| <code>title('Expl1')</code> | Set title of figure window |
| <code>xlabel('time')</code> | Set label of x-axis |
| <code>ylabel('prob')</code> | Set label of y-axis |
| <code>subplot</code> | Put several plot axes into figure |

Control de los ejes

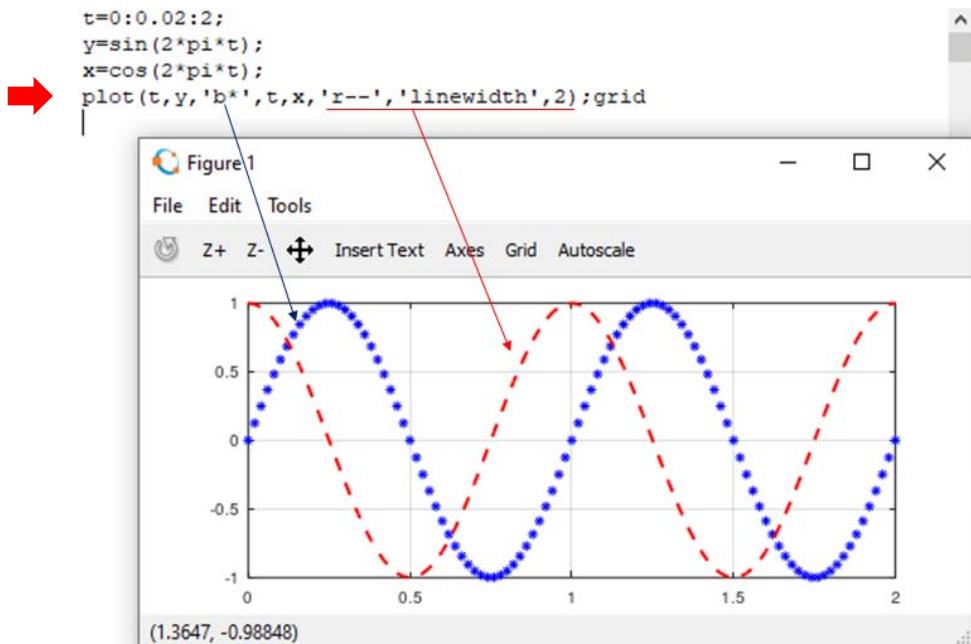
| | |
|-------------------------------|---|
| <code>axis equal</code> | Set equal scales for x-/y-axes |
| <code>axis square</code> | Force a square aspect ratio |
| <code>axis tight</code> | Set axes to the limits of the data |
| <code>a = axis</code> | Return current axis limits [xmin xmax ymin ymax] |
| <code>axis([-1 1 2 5])</code> | Set axis limits (freeze axes) |
| <code>axis off</code> | Turn off tic marks |
| <code>box on</code> | Adds a box to the current axes |
| <code>box off</code> | Removes box |



Colores, tipos de línea, marcadores,

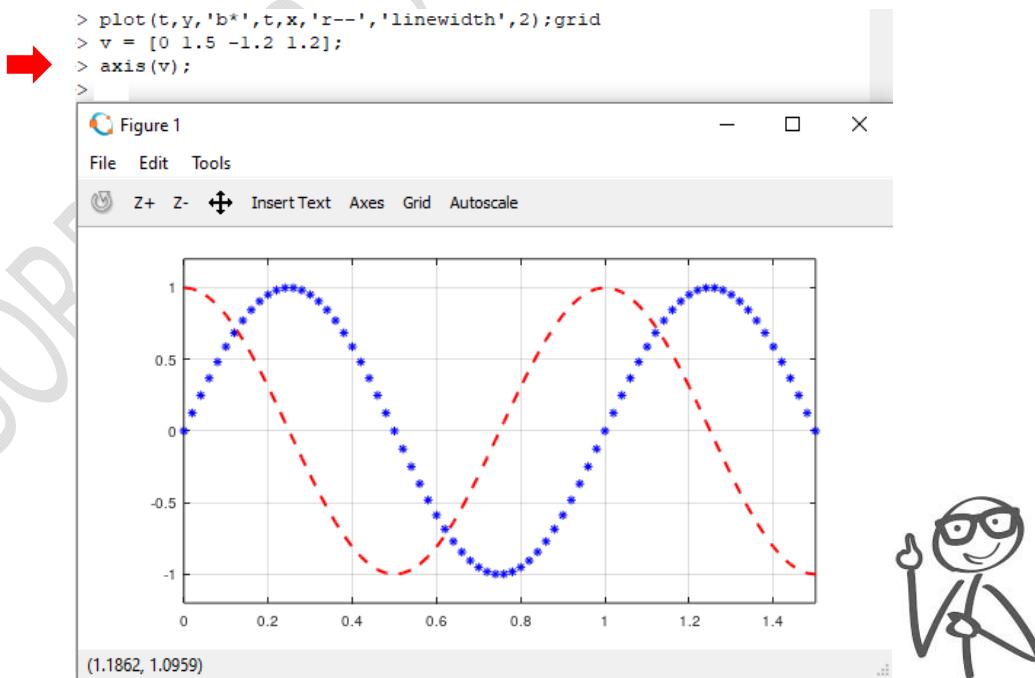
Se pueden seleccionar diferentes símbolos, colores, estilos de línea y también controlar el grosor de los trazos – ver **help plot**.

En la línea del plot del ejemplo, la expresión de formato ‘b*’ significa *línea azul con marcadores de tipo **, y la expresión ‘r--’ significa *línea roja con trazos segmentados*, y ‘linewidth’, 2 significa que el *grosor del trazo es de 2 puntos*.



Se puede ajustar la escala de los ejes

Por ejemplo, para la gráfica anterior se pueden modificar los límites de los ejes con el comando **axis ([xmin xmax ymin ymax])**.



Múltiples gráficas en una sola ventana de figura.

Se pueden tener arreglos de gráficas $m \times n$ (m filas y n columnas) en una misma ventana de figura usando el comando **subplot(m,n,i)** donde m indica el número de filas, n el número de columnas del arreglo y i es el índice de la figura gráfica activa que se quiere controlar. Los índices corresponden a valores $i = 1, 2, 3, \dots$ numerados de izquierda a derecha iniciando en la primera fila de la matriz $m \times n$ especificada.

Gráficas de impulsos – sirve para señales discretas y digitales

Para estas gráficas se usa el comando **stem**, similar a plot pero grafica impulsos para los valores correspondientes del par x-y.

Se puede agregar texto con el **comando text(coord x, coord. y,'texto')**

¡Para el texto se puede usar la sintaxis de **Latex!**

El siguiente ejemplo muestra el uso de los tres comandos anteriores.

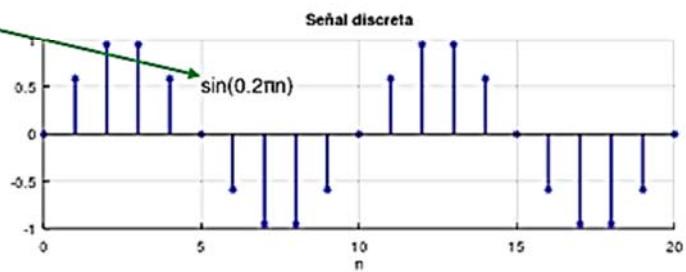
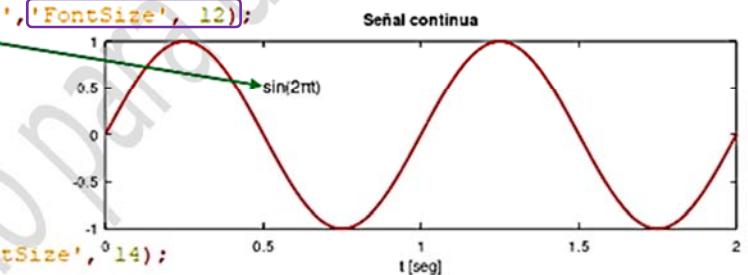
```
t=0:0.02:2;
x=sin(2*pi*t);

subplot(2,1,1)
plot(t,x,'r','LineWidth',2);
xlabel('t [seg]');text(0.5,0.5,'sin(2\pit)', 'FontSize', 12);
title('Señal continua');

n=0:20;
y=sin(0.2*pi*n);

subplot(2,1,2)
stem(n,y,'b','LineWidth',2);
xlabel('n');text(5,0.5,'sin(0.2\pin)', 'FontSize', 14);
title('Señal discreta');
grid
```

Permite controlar el tamaño del texto.
Se puede usar en cualquier comando de texto, e.g., xlabel, title, etc.



Además de las gráficas de línea o impulsos, hay gran cantidad de otros tipos de gráficas. Simplemente usar el comando **help** con los siguientes comandos:

```
hist, bar, pie, area, fill, contour, quiver,
scatter, compass, rose, semilogx, loglog,
stem, stairs, image, imagesc
```

Además, se tienen **gráficas en 3-dimensiones**, e.g., **plot3, mesh, surf, bar3, ...**



Programación



!Programar
en Octave es
muy fácil!

Debes recordar que
los índices comienzan con 1 !!!

```
> v = 1:10
> v(0)
error: subscript indices must be either positive
integers or logicals.
```

Octave is case-sensitive!

Estructuras de control

▪ if Statement

```
if condition,
then-body;
elseif condition,
elseif-body;
else
else-body;
end
```

The else and elseif clauses are optional.
Any number of elseif clauses may exist.

▪ switch Statement

```
switch expression
case label
command-list;
case label
command-list;
...
otherwise
command-list;
end
```

Any number of case labels are possible.

▪ while Statement

```
while condition,
body;
end
```

▪ for statement

```
for var = expression,
body;
end
```

Interrupting and Continuing Loops

▪ break

Jumps out of the innermost for or while loop that encloses it.

▪ continue

Used only inside for or while loops. It skips over the rest of the loop body, causing the next cycle to begin. Use with care.

Setting Paths

- path Print search path list
- addpath('dir') Prepend the specified directory to the path list
- rmpath('dir') Remove the specified directory from the path list
- savepath Save the current path list



Archivos I/O

Save Variables

- `save my_vars.mat`
Saves all current variables into file `my_vars.mat`
- `save results.mat resultdata X Y`
Saves variables `resultdata`, `X` and `Y` in file `results.mat`
- `save ... -ascii`
Saves variables in ASCII format
- `save ... -mat`
Saves variables in binary MAT format

Load Variables

The corresponding command is `load`.

- `load my_vars.mat`
Retrieves all variables from the file `my_vars.mat`
- `load results.mat X Y`
Retrieves only `X` and `Y` from the file `results.mat`

An ASCII file that contains **numbers in a matrix format** (columns separated by spaces, rows separated by new lines), can be simply read in by

- `A = load('data.txt')`

Open, Write, Close Files

- `fopen` Open or create file for writing/reading
- `fclose` Close file
- `fprintf` Write formatted data to file. C/C++ format syntax.

Example:

```
v = randn(1000,1);
fid = fopen('gauss.txt','w');
for i = 1:length(v),
    fprintf(fid,'%7.4f\n',v(i));
end
fclose(fid);
```

Reading Files (more advanced stuff)

- `textread` Read formatted data from text file
- `fscanf` Read formatted data from text file
- `fgetl` Read line from file
- `fread` Read binary data file

Read/write images

- `imread` Read image from file (many formats)
- `imwrite` Write image to file (many formats)

Otros commandos

Cleaning Up

- `clear A` Clear variable `A`
- `clear frame*` Clear all variables whose names start with `frame...`
- `clear` Clear **all** variables
- `clear all` Clear everything: variables, globals, functions, links, etc.
- `close` Close foreground figure window
- `close all` Close all open figure windows
- `clc` Clear command window (shell)

Random Seeds

- `rand` and `randn` obtain their initial seeds from the system clock.
- To generate identical/repeatable sequences, set the random generator seeds manually.

To set the random seeds:

- `rand('seed',value)` Set seed to scalar integer value `value`.
- `randn('seed',value)` Set seed to scalar integer value `value`.



Algunos consejos para el uso práctico

Speed!

- The **lack of speed** of Octave/Matlab programs is widely recognized to be their biggest drawback.
- Mostly it's **your program that is slow**, not the built-in functions!
- This brings us to the following **guidelines**:
 - **For-loops are evil**
 - **Vectorization is good**
 - **Preallocation is good**
 - Prefer **struct of arrays** over arrays of struct

Speed: Preallocation

- If a for- or while-loop cannot be avoided, do not grow data structures in the loop, **preallocate them** if you can. Instead of, e.g.,

```
for i = 1:100,  
    A(i,:) = rand(1,50);  
end;
```

Write:

```
A = zeros(100,50); % preallocate matrix  
for i = 1:100,  
    A(i,:) = rand(1,50);  
end;
```

Speed: Vectorization

- Given `phi = linspace(0,2*pi,100000);`
The code

```
for i = 1:length(phi),  
    sinphi(i) = sin(phi(i));  
end;
```

is significantly slower than simply
`sinphi = sin(phi);`
- Nearly **all built-in commands** are **vectorized**. Think vectorized!



1. Señales y sistemas

En el estudio de *Señales y Sistemas*, aunque la naturaleza de los problemas puede ser muy variada y diferente de una aplicación a otra, todas contienen dos elementos comunes: **señales** (de entrada y de salida) y un **sistema** descrito por una relación de entrada-salida.



Una **señal** se define como una función de una o más variables independientes que contiene o representa información acerca del comportamiento o la naturaleza de algún fenómeno.

Un **sistema** es una entidad que manipula una o más señales para realizar una operación y producir nuevas señales o tener un comportamiento deseado. Los sistemas se describen por una relación entrada-salida.

Señal unidimensional – la función depende de una sola variable, e.g., señal de voz (la amplitud varía con el tiempo).

Señal multi-dimensional – la función depende de dos o más variables, e.g., una imagen (tiene elementos – pixeles - distribuidos vertical y horizontalmente en el espacio).

En términos matemáticos, un sistema es una función u operador $T\{ \cdot \}$, que asigna la señal de entrada $x(t)$ a la señal de salida $y(t) = T\{x(t)\}$.

Un ejemplo de un sistema son los circuitos de un amplificador electrónico de audio.

Nos enfocaremos solamente en las señales como funciones de *una sola variable independiente – el tiempo*.

El valor de la función (señal) puede ser una cantidad escalar de valor real, una cantidad de valor complejo o tal vez un vector.

$$\begin{aligned} \text{Señal de valor real: } x(t) &= 5 \sin(10\pi t) \\ \text{Señal de valor complejo: } x(t) &= 5e^{-j10\pi t} \\ &= 5 \cos(10\pi t) - j \sin(10\pi t) \end{aligned}$$

Trataremos con dos tipos básicos de señales: **continuas y discretas**.

- Señales continuas (CT) son aquellas que son funciones continuas del tiempo t .
- Señales discretas (DT) son aquellas que son funciones discretas del índice de tiempo n (donde n es un entero).



A diferencia de una señal de tiempo continuo, una señal de tiempo discreto solo está activa en períodos de tiempo específicos. Así, las señales de tiempo discreto se pueden almacenar en la memoria de un computador porque el número de valores de la señal que deben almacenarse para representar un intervalo de tiempo finito es finito.



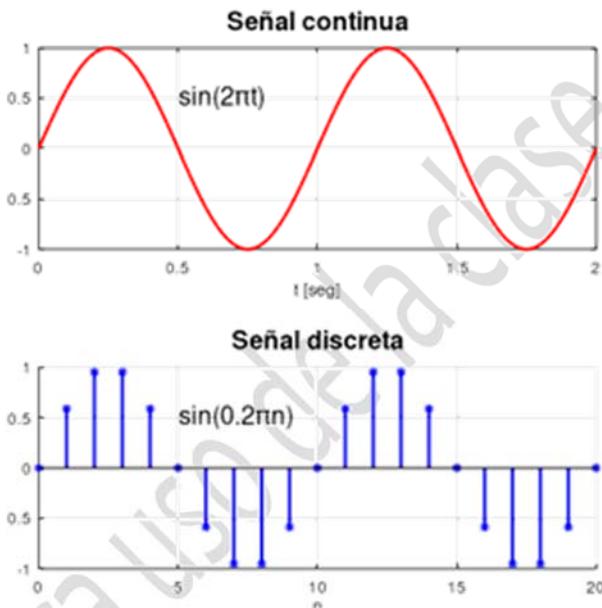
En un computador *todas las señales son discretas*, pero podemos simular una señal continua en Octave si usamos un incremento pequeño de la variable tiempo y graficamos la señal con el comando **plot**.

Observe como la misma señal se puede simular como una *señal continua* y una *señal discreta*.

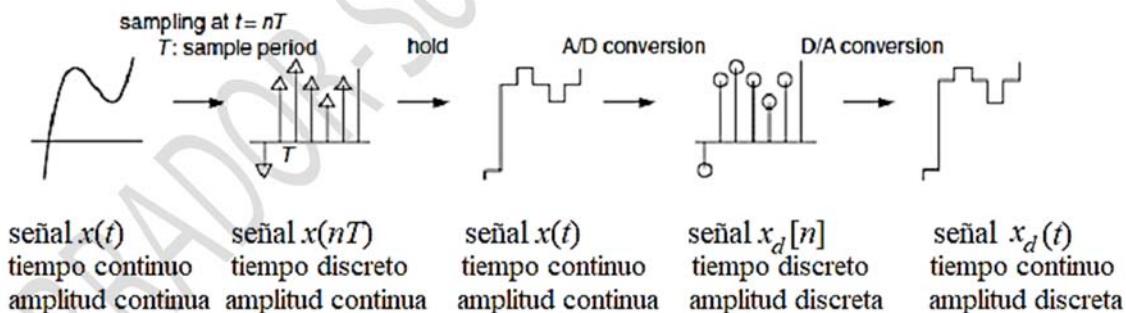
```
t=0:0.02:2;
x=sin(2*pi*t);
subplot(2,1,1)
plot(t,x,'r','LineWidth',2);grid;
xlabel('t [seg]');
text(0.5,0.5,'sin(2\pit)', 'FontSize', 16);
title('Señal continua', 'FontSize', 16);

n=0:20;
y=sin(0.2*pi*n);

subplot(2,1,2)
stem(n,y,'b*','LineWidth',2);grid;
xlabel('n');
text(5,0.5,'sin(0.2\pin)', 'FontSize', 16);
title('Señal discreta', 'FontSize', 16);
```



La amplitud de una señal también puede ser continua o discreta, y dependiendo de ella y del tiempo, una señal puede ser de varios tipos y sufrir transformaciones de estos dos parámetros – como estudiaremos más adelante. En la figura se ilustran estas transformaciones y los tipos de señales.



Las señales en función del tiempo son cómo la mayoría de las personas experimentan el mundo real, pero la información completa de una señal no siempre es visible en ese dominio. Por eso, la transformación a otros dominios, específicamente, el dominio de la frecuencia (*f, s, z*) y viceversa, es necesario en muchas situaciones para analizar, comprender y diseñar señales y sistemas. Más adelante estudiaremos estas transformaciones.



Operaciones con señales

A continuación se describen varias operaciones con señales que debes aprender y se ilustra la forma como podrían implementarse en Octave.

Adición

Para sumar dos o más señales continuas (CT) o discretas (DT), se suman los valores de cada señal en cada instante de tiempo t o cada índice n , respectivamente. En otras palabras, la señal resultante en un instante dado, corresponde a la suma de los valores de cada señal en ese instante t o n .

Por lo tanto, para sumar las dos señales, ambas deben estar definidas para los mismos valores de t o n .

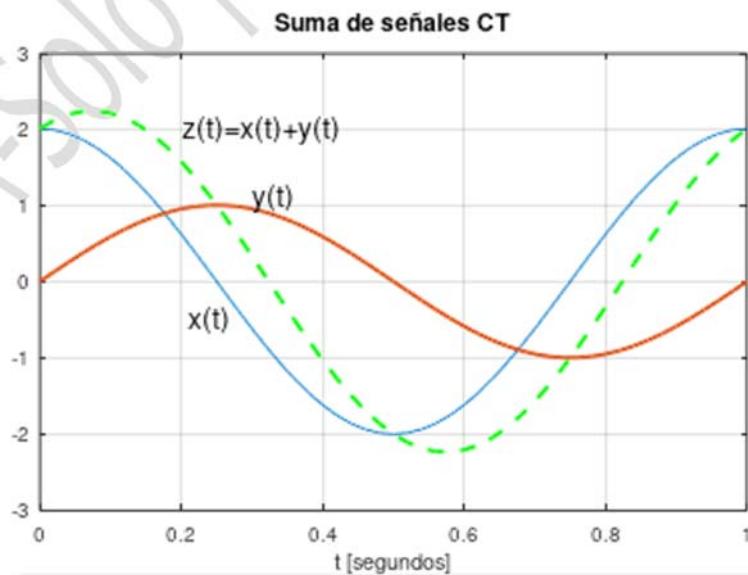
Para implementar la suma en Octave, recuerde que todas las señales son DT y por lo tanto los vectores correspondientes a las señales que se van a sumar, deben tener el mismo vector de soporte y ser de la misma longitud.

Ejemplo, sean las señales CT $x(t) = 2 \cos(2\pi t)$ y $y(t) = \sin(2\pi t)$, definidas en el intervalo $0 \leq t \leq 1$.

Determine la suma $z(t) = x(t) + y(t)$ y grafique las tres señales.

```
t = 0:0.02:1;
x = 2*cos(2*pi*t);
y = sin(2*pi*t);
z = x + y;

plot(t,x,t,y,'LineWidth',2);
hold on
plot(t,z,'g--','LineWidth',2);
title('Suma de señales CT','FontSize', 12);
text(0.21,-0.5,'x(t)','FontSize', 14);
text(0.3,1.1,'y(t)','FontSize', 14);
text(0.2,2,'z(t)=x(t)+y(t)','FontSize', 14);
xlabel('t [segundos]');grid;
```

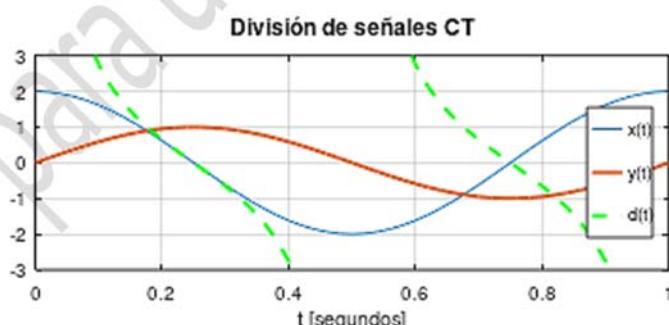
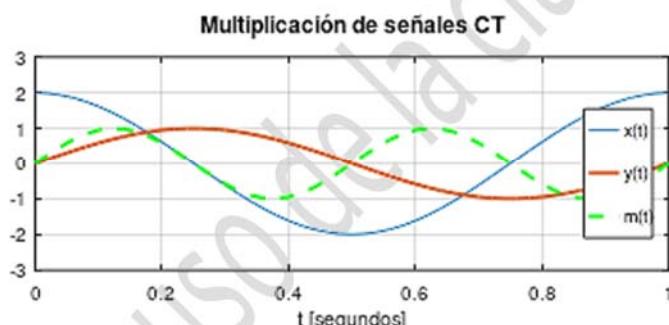
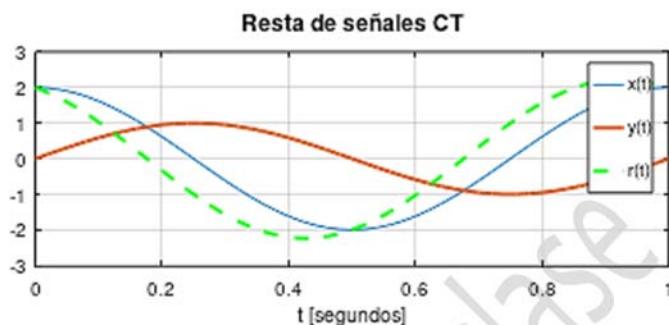


La **sustracción, multiplicación** y **división** se realizan de forma similar a la adición, es decir, el valor resultante de la operación en un instante dado es igual a la operación de los valores de cada señal en ese instante.



```
t = 0:0.02:1;
x = 2*cos(2*pi*t);
y = sin(2*pi*t);
r = x - y;
m = x.*y; ← *
d = x./y; ← *
```

Recuerde que en Octave para multiplicar y dividir vectores – elemento a elemento – debe usar los operadores `.*` y `.`



Se ajustó la escala de los ejes.

Como hay divisiones por valores muy pequeños, incluyendo cero, la función resultante tiene valores muy grandes, por lo tanto $d(t) = x(t) / y(t)$ no se muestra completa.

Cambio de escala de amplitud

Para cambiar la amplitud de una señal CT o DT, simplemente se multiplica la señal por un factor de escala A, tal que

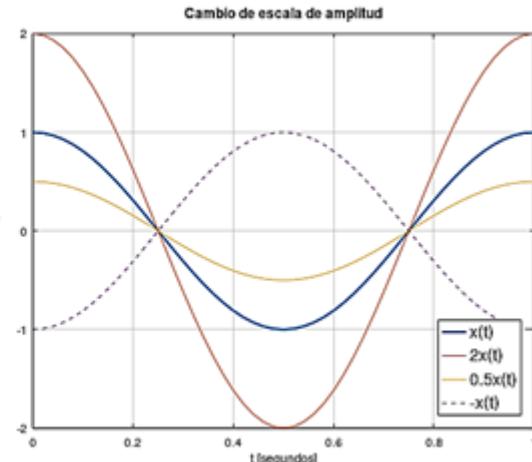
- si $A > 1$, se amplifica la señal
- si $A < 1$, se atenúa la señal
- si $A = -1$, hay un cambio de fase de 180°

```

t = 0:0.02:1;
x = cos(2*pi*t);
A1 = 2;
A2 = 0.5;
A3 = -1;

plot(t,x,'linewidth',2,t,A1*x,t,A2*x,t,A3*x,'--')
title('Cambio de escala de amplitud')
h=legend('x(t)', '2x(t)', '0.5x(t)', '-x(t)');
legend(h, "location", "southeast");
set(h, "fontsize", 14);
xlabel('t [segundos]');grid;

```



Corrimiento en el tiempo

Para cualquier t_0 o n_0 , el corrimiento en el tiempo es una operación definida como

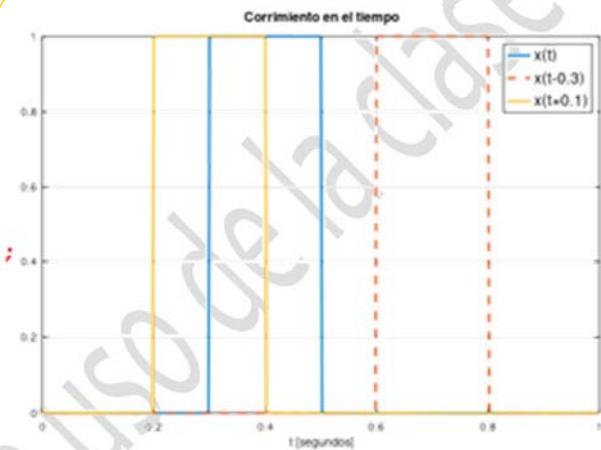
$$x(t) \rightarrow x(t - t_0)$$

$$x[n] \rightarrow x[n - n_0]$$

- Si $t_0 > 0$, se conoce como **retraso**
- Si $t_0 < 0$, se conoce como **adelanto**

```
t = 0:0.002:1;
x = (0.3<=t & t<=0.5); % pulso cuadrado
y = (0.3<=t-0.3 & t-0.3<=0.5);; % x(t-0.3)
z = (0.3<=t+0.1 & t+0.1<=0.5); % x(t+0.1)
```

```
p = plot(t,x,t,y,'--',t,z);
set(p,'linewidth',2);
title('Corrimiento en el tiempo','FontSize', 12);
h=legend('x(t)', 'x(t-0.3)', 'x(t+0.1)');
legend (h, "location", "southeastoutside");
set (h, "fontsize", 14);
xlabel('t [segundos]');grid;
```



Cambio de escala en el tiempo

El cambio de escala en el tiempo se ve de forma diferente para señales CT y DT.

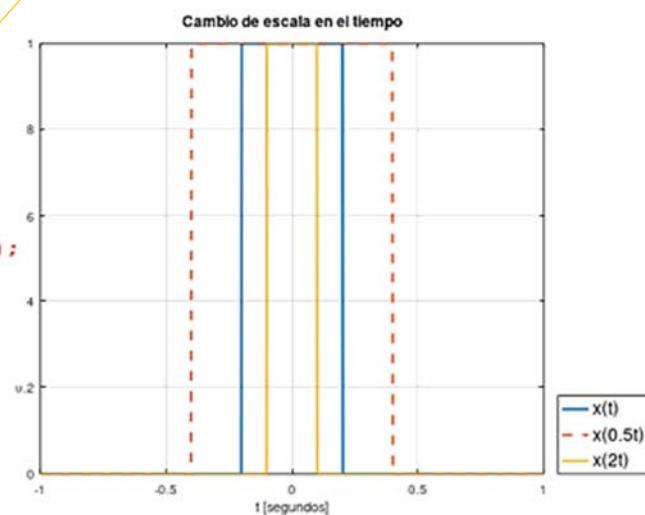
Para señales CT, es la operación donde la variable independiente t se multiplica por una constante, a:

$$x(t) \rightarrow x(at)$$

- Si $a < 1$, la señal se **expande** en el tiempo
- Si $a > 1$, la señal se **comprime** en el tiempo

```
t = -1:0.002:1;
x = (-0.2<=t & t<=0.2); % pulso cuadrado
y = (-0.2<=0.5*t & 0.5*t<=0.2);; % x(0.5t)
z = (-0.2<=2*t & 2*t<=0.2); % x(2t)
```

```
p = plot(t,x,t,y,'--',t,z);
set(p,'linewidth',2);
title("Cambio de escala en el tiempo",'FontSize', 12);
h=legend('x(t)', 'x(0.5t)', 'x(2t)');
legend (h, "location", "southeastoutside");
set (h, "fontsize", 14);
xlabel('t [segundos]');grid;
```



$$\text{Algo} = \frac{\sin(\omega_0 t) + \sin(\omega_0(t-\Delta t))}{2}$$



Para señales discretas, el cambio de escala en el tiempo corresponde a

- una **expansión (up sampling)** para

$x[n] \rightarrow x_E \left[\frac{n}{L} \right]$ donde L es el factor de expansión y n es un múltiplo entero de L .

- una **decimación (down sampling)** para

$x[n] \rightarrow x_D[Mn]$ donde M es un entero.

```

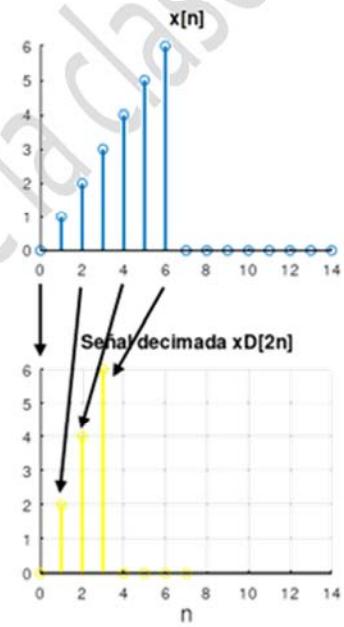
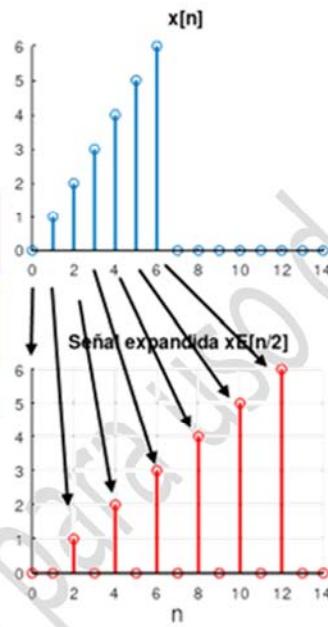
n = 0:15;
x = (0<=n & n<=6).*n; % señal DT
xE = upsample(x,2); % xE(n/2)
nE = 0:length(xE)-1;
xD = x(1:2:end); % xD(2n)
nD = 0:length(xD)-1;

```

Para la expansión se usa el comando **upsample(x,L)** que inserta $L-1$ ceros entre cada elemento.

Para la decimación podemos simplemente extraer los elementos del vector cada M términos.

⚠ Es importante corregir los vectores de soporte de las señales modificadas.



Inversión en el tiempo

La inversión en el tiempo se define como

$$x(t) \rightarrow x(-t)$$

$$x[n] \rightarrow x[-n]$$

y se puede interpretar como un *giro sobre el eje-y*.

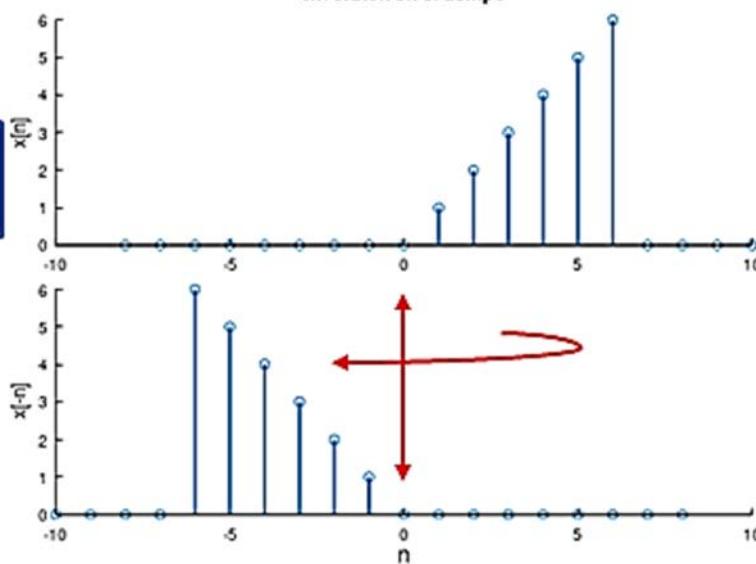
```

nx = -8:10;
x = (0<=nx & nx<=6).*nx;
xi=fliplr(x);
ni=-fliplr(nx);

```

⚠ Comando fliplr
Recuerde generar el vector de soporte correspondiente

Inversión en el tiempo



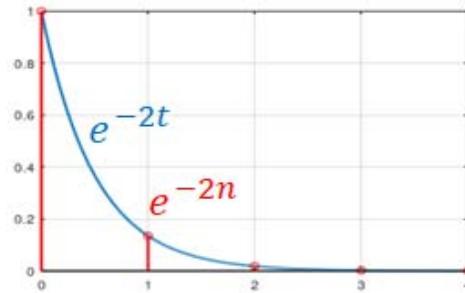
Clasificación de señales

Las señales se clasifican según diferentes criterios en diversos tipos.

Señal continua (de tiempo continuo) o discreta (de tiempo discreto)

Señal continua $x(t)$ es aquella definida para todo instante de tiempo t , donde t es una variable de valor real.

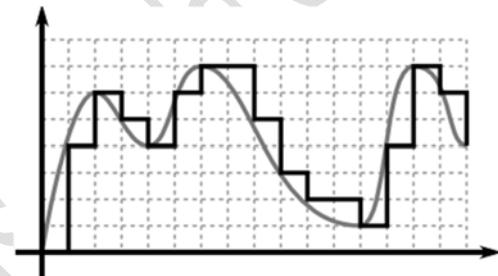
Señal discreta $x[n]$ es aquella definida solo para instantes discretos – específicos de tiempo n , donde n es una variable de valor real entero.



Señal analógica o digital

Señal analógica (continua o discreta) es aquella cuya amplitud es continua, i.e., toma cualquier valor en un intervalo dado.

Señal digital (continua o discreta) es aquella cuya amplitud es discreta, i.e., solo toma valores específicos en un intervalo dado.



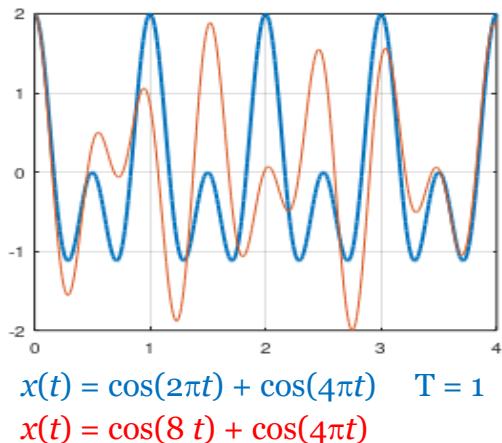
Señal periódica o aperiódica

Señal periódica (continua o discreta) es aquella que satisface la condición de periodicidad,

$$x(t) = x(t + T) \quad \text{o} \quad x[n] = x[n + N]$$

esto es, la señal se repite para un valor constante $T > 0$ ($N > 0$), donde el menor valor de T que satisface esta condición se llama periodo.

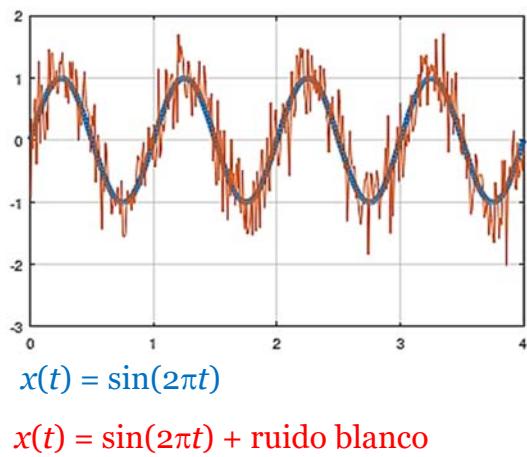
Señal aperiódica o no-periódica (continua o discreta) es aquella que no satisface la condición de periodicidad.



Señal determinista o probabilística

Señal determinista (continua o discreta) es aquella cuya descripción física es conocida completamente, ya sea en una forma matemática o en forma gráfica.

Señal probabilística o aleatoria (continua o discreta) es aquella cuya descripción física es conocida solamente en términos de su descripción probabilística, tal como el valor cuadrático medio, valor medio, etc.



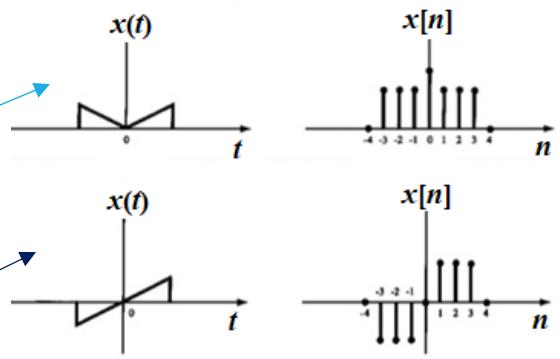
Señal par o impar

Señal par (continua o discreta) es aquella que satisface la condición

$$x(-t) = x(t) \quad \text{o} \quad x[-n] = x[n]$$

Señal impar (continua o discreta) es aquella que satisface la condición

$$x(-t) = -x(t) \quad \text{o} \quad x[-n] = -x[n]$$



Cualquier señal se puede separar en la suma de dos señales, una de las cuales es par y la otra impar.



$$x(t) = x_e(t) + x_o(t)$$

$$x[n] = x_e[n] + x_o[n]$$

parte par

$$x_e(t) = \frac{1}{2} \{x(t) + x(-t)\}$$

$$x_e[n] = \frac{1}{2} \{x[n] + x[-n]\}$$

parte impar

$$x_o(t) = \frac{1}{2} \{x(t) - x(-t)\}$$

$$x_o[n] = \frac{1}{2} \{x[n] - x[-n]\}$$

```
n = -10:10;
x = [0 0 0 0 0 1 2 3 4 5 5 5 5 5 5 5 5 0 0 0 0 0]; % Señal x[n]
xi=flipr(x); % Señal invertida
xe=0.5*(x+xi); % Componente par
xo=0.5*(x-xi); % Componente impar
```

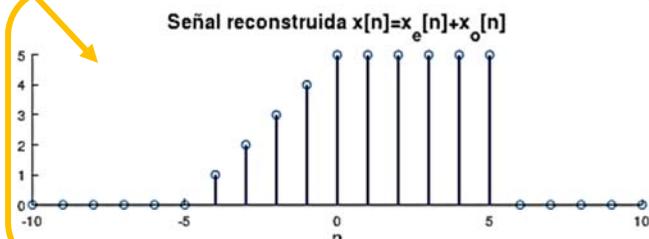
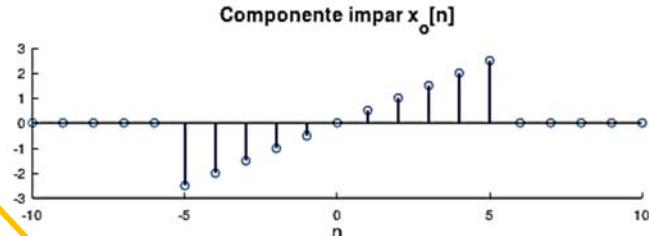
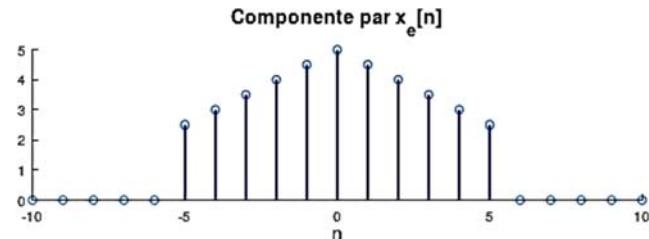
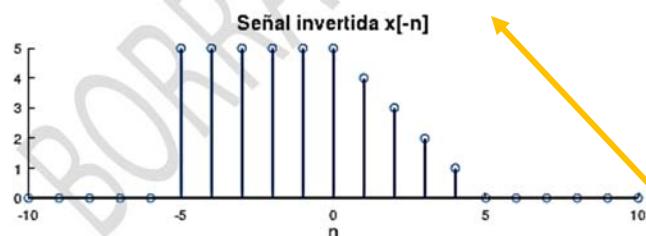
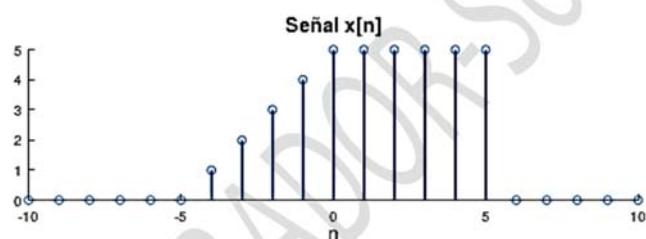
```
figure
subplot(3,2,1);stem(n,x,'linewidth',2);
title('Señal x[n]', 'fontsize', 14); xlabel('n');

subplot(3,2,3);stem(n,xi,'linewidth',2);
title('Señal invertida x[-n]', 'fontsize', 14);

subplot(3,2,2);stem(n,xe,'linewidth',2);
title('Componente par x_e[n]', 'fontsize', 14); xlabel('n');

subplot(3,2,4);stem(n,xo,'linewidth',2);
title('Componente impar x_o[n]', 'fontsize', 14); xlabel('n');

subplot(3,2,6);stem(n,xe+xo,'linewidth',2);
title('Señal reconstruida x[n]=x_e[n]+x_o[n]', 'fontsize', 14); xlabel('n');
```



Señal de energía o de potencia

Una señal (continua o discreta) de energía es aquella que tiene energía finita.

Una señal (continua o discreta) de potencia es aquella que tiene potencia finita y distinta de cero.

Para clasificar una señal como señal de energía o de potencia, la energía y la potencia se deben calcular sobre un intervalo de tiempo infinito.

Los términos “energía” y “potencia” se usan independientemente de si las cantidades estén en verdad relacionadas con la energía física.

En el caso de señales de voltaje o corriente se denominan “**energía normalizada**” y “**potencia normalizada**” de una señal $x(t)$ o $x[n]$.



$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt \quad E = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

$$P \triangleq \lim_{T \rightarrow \infty} \frac{1}{2T} \int_{-T}^T |x(t)|^2 dt \quad P = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{n=-N}^N |x[n]|^2$$



- Si una señal es de energía, su potencia es cero.
- Si una señal es de potencia, su energía es infinita.
- ¡Una señal no puede ser de potencia y energía al mismo tiempo!
- Hay señales que no son ni de energía ni de potencia.

También se puede calcular la energía y la potencia de cualquier señal en un intervalo específico $t_1 \leq t \leq t_2$ ($n_1 \leq n \leq n_2$)

$$E = \int_{t_1}^{t_2} |x(t)|^2 dt$$

$$P = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} |x(t)|^2 dt$$

$$E = \sum_{n=n_1}^{n_2} |x[n]|^2$$

$$P = \frac{1}{n_2 - n_1 + 1} \sum_{n=n_1}^{n_2} |x[n]|^2$$



Sea la señal

$$x(t) = \cos(2\pi 8t)$$

Vamos a clasificar esta señal –

$x(t)$ es una señal *determinista, continua, real, analógica, impar, periódica* con $T = 1/8$ s.

Su energía es $E = \int_{-\infty}^{\infty} |\cos(16\pi t)|^2 dt = \infty$

y su potencia es $P \triangleq \lim_{T \rightarrow \infty} \frac{1}{2T} \int_{-T}^T |\cos(16\pi t)|^2 dt = \frac{1}{2T} \int_{-T}^T [0.5 + 0.5\cos(16\pi t)^2] dt = 0.5$

Entonces, $x(t)$ es una señal de *potencia*.



¿Cómo puedo simular esta señal y calcular su energía y potencia en Octave?

Recuerde que en Octave todas las señales son discretas, por lo que esta señal continua debe modelarse para tal naturaleza.

```
dt=0.001; // Observe que el tiempo es discreto!
t=0:dt:3600; // No puedo tener una duración infinita, por lo que escojo un intervalo
si=cos(2*8*pi*t);
E = sum(abs(si).^2)*dt; // Observe que se usa la ecuación para señales discretas
P = sum(abs(si).^2)/length(si); // pero se incluye un factor dt en el cálculo – ya que
disp('Energía y potencia - intervalo "infinito"'); // corresponde a una señal “continua”.
disp(E)
disp(P) // Note que es la ecuación para señales discretas.
```

```
Energía y potencia - intervalo "infinito"
1800.0
0.50000
>>
```

Observe que la energía no es infinita como en el cálculo analítico porque es una simulación de una onda continua en un intervalo grande (pero finito). Sin embargo, note que el valor es significativo con respecto a la potencia (0.5) que corresponde efectivamente al resultado analítico de esta señal.



Si aumenta el tiempo de la señal, verá que aumenta el valor de E pero permanece igual el de P. De lo que podemos concluir que es una señal de potencia con potencia promedio normalizada de 0.5.

Señales de interés

Las señales que más vamos a utilizar en el estudio de Señales y Sistemas son las siguientes:

- *Impulso unitario $\delta(t)$, $\delta[n]$*
- *Escalón unitario $u(t)$, $u[n]$*
- *Pulso rectangular unitario $rect(t/\tau)$, $\delta[n/D]$*
- *Pulso triangular unitario $\Delta(t/\tau)$, $\Delta[n/D]$*
- *Exponencial $e^{at} u(t)$, $a^n u[n]$*
- *Sinusoidal real $\cos(\omega_o t + \phi)$, $\cos(\omega_o n + \phi)$*
- *Función de interpolación $sinc(x)$*

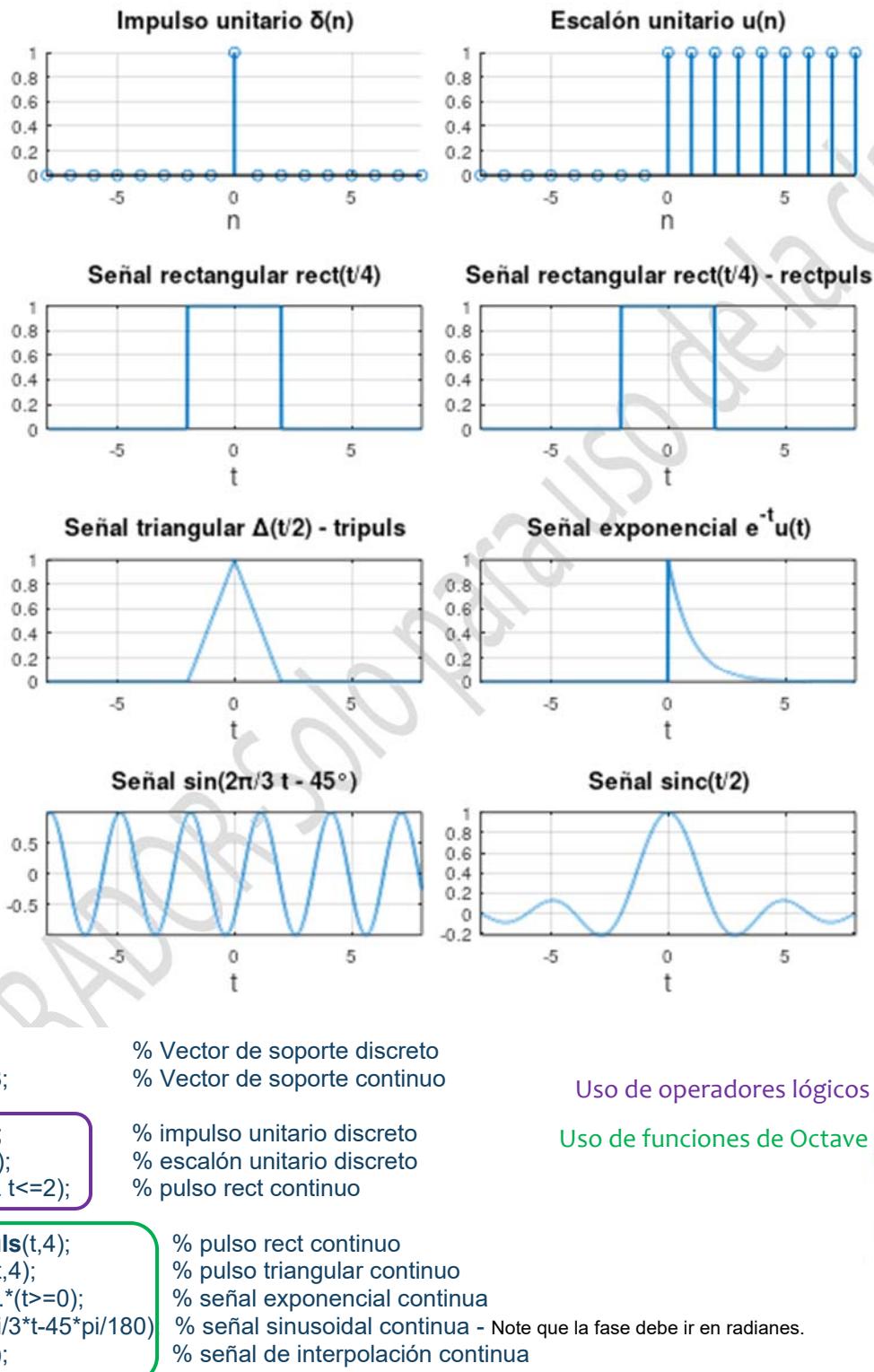
Recuerde que en Octave todas las señales son discretas, por lo que para simular las señales continuas debemos usar un vector de soporte de tiempo con un incremento pequeño y al momento de graficar, usar el comando plot. Mientras que para las señales discretas, el vector de soporte n son enteros y usamos el comando stem para graficar.

Octave incluye ya varias funciones como *exp*, *sin*, *cos*, *rectpuls*, *tripuls*, *sinc* que se pueden usar para modelar nuestras funciones de interés. Pero también, muchas de ellas son fáciles de implementar con operadores lógicos como hemos visto en ejemplos anteriores.



A continuación se incluye un script con algunos ejemplos de cómo generar estas señales de interés – algunas se modelan como señales discretas y otras continuas.

En el caso del pulso rectangular, éste se genera usando operadores lógicos y también con la función rectpuls de Octave.



Uso de operadores lógicos

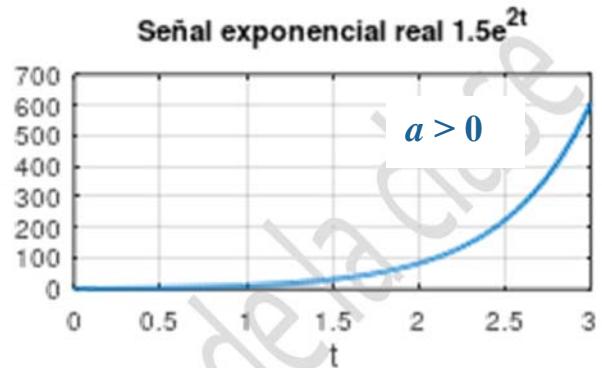
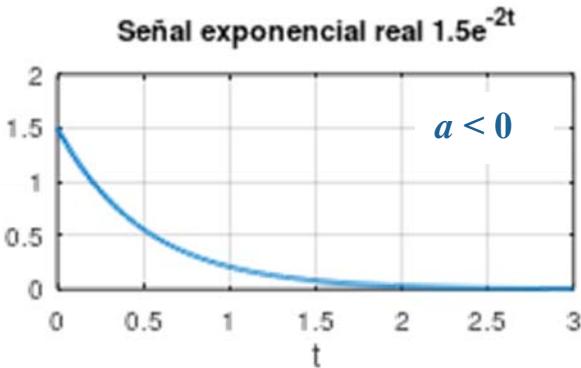
Uso de funciones de Octave



Señal exponencial compleja continua

$x(t) = Ce^{at}$ donde C y a son, en general números complejos

Exponencial real: C y a son reales



Exponencial compleja periódica: a es imaginaria, $a = j\omega_0$

$$x(t) = e^{j\omega_0 t}$$

Periodicidad: $x(t) = e^{j\omega_0 t} = e^{j\omega_0(t+T)} = e^{j\omega_0 t}e^{j\omega_0 T} \rightarrow$ para ser periódica, $e^{j\omega_0 T} = 1$, entonces el periodo fundamental T_o de $x(t)$ (es decir, el valor más pequeño de T) es

$$T_0 = \frac{2\pi}{|\omega_0|}$$

Función sinusoidal real

$x(t) = A \cos(\omega_0 t + \phi)$ donde $\omega_0 = 2\pi f_0$ en rad/s

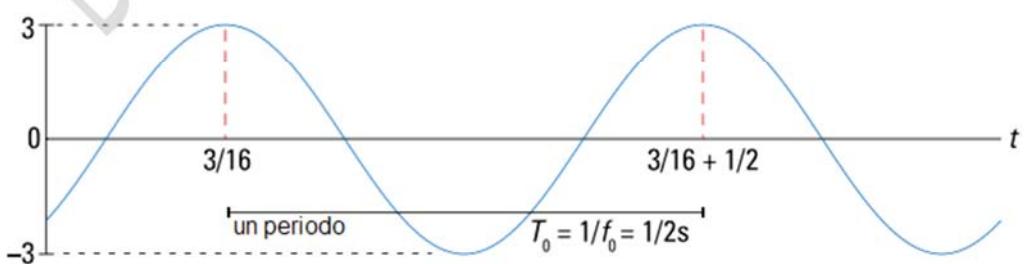
$$x(t) = 3 \cos(2\pi \cdot 2 \cdot t - 3\pi/4)$$

amplitud

frecuencia f_0 en Hz

fase ϕ_0 en rad

Generalmente usamos la función \cos como referencia, pero es lo mismo para la función \sin .



Identidad de Euler – relación entre la señal exponencial compleja y la señal sinusoidal



La identidad o fórmula de Euler, llamada así por Leonhard Euler, es una fórmula matemática en análisis complejo que establece la relación fundamental entre las funciones trigonométricas y la función exponencial compleja. La fórmula de Euler establece que para cualquier número real x :

$$e^{jx} = \cos(x) + j\sin(x)$$

donde e es la base del logaritmo natural, j es la unidad imaginaria y \cos y \sin son las funciones trigonométricas coseno y seno, respectivamente.

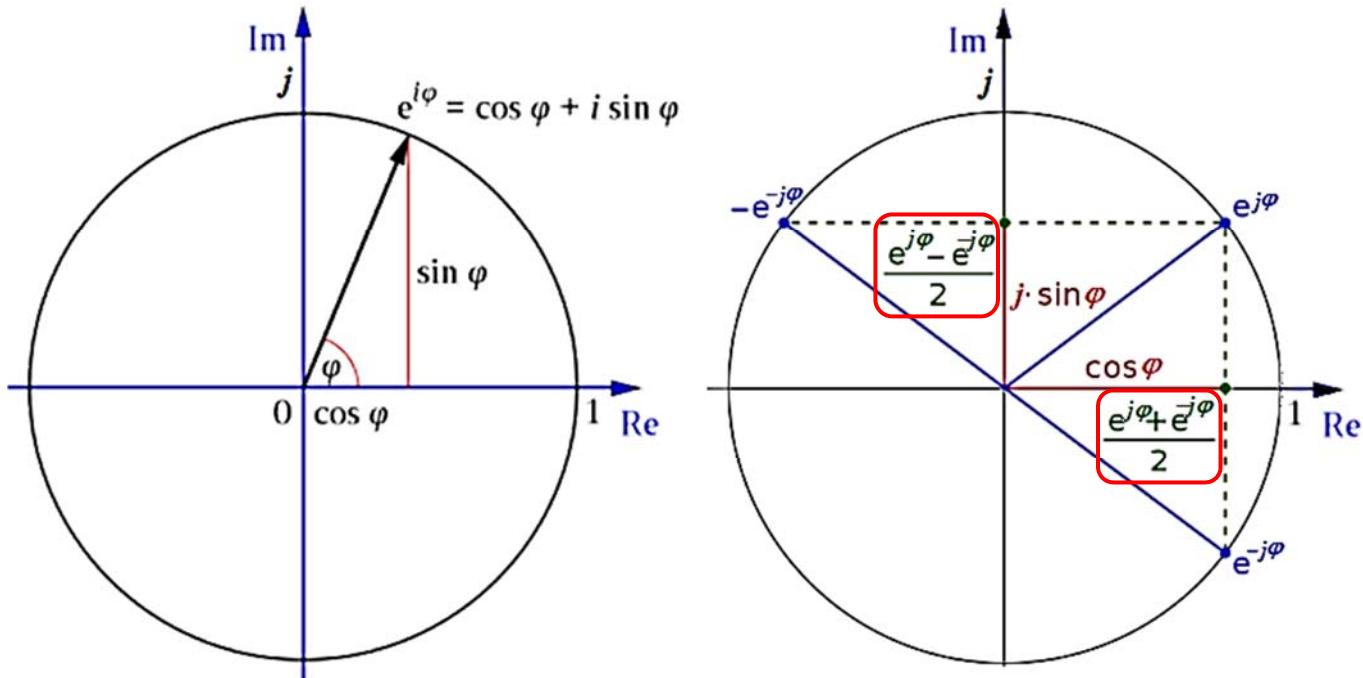
La fórmula sigue siendo válida si x es un número complejo, por lo que algunos autores se refieren a la versión compleja más general como fórmula de Euler.

La fórmula de Euler es omnipresente en matemáticas, física e ingeniería. El físico Richard Feynman llamó a la ecuación "nuestra joya" y "la fórmula más notable en matemáticas".

https://www.wikiwand.com/en/Euler%27s_formula



También permite construir las funciones seno y coseno a partir de la función exponencial.



La fórmula de Euler permite, entre otras cosas, representar números complejos (en términos de su magnitud y ángulo), definir el logaritmo de un número complejo, definir las funciones \sin y \cos en el plano complejo (ver figura superior derecha), simplificar las operaciones trigonométricas, simplificar soluciones en ecuaciones diferenciales, describir señales que varían periódicamente en el tiempo, y representar las impedancias en el análisis de circuitos.



Volviendo a nuestra exponencial compleja periódica $x(t) = e^{j\omega_0 t}$, se tiene

$$e^{j\omega_0 t} = \cos \omega_0 t + j \sin \omega_0 t$$

y con las relaciones anteriores, en general, para una señal sinusoidal se puede escribir

$$\begin{aligned} A \cos(\omega_0 t + \phi) &= \frac{A}{2} e^{j\phi} e^{j\omega_0 t} + \frac{A}{2} e^{-j\phi} e^{-j\omega_0 t} \\ A \sin(\omega_0 t + \phi) &= \frac{A}{2j} e^{j\phi} e^{j\omega_0 t} - \frac{A}{2j} e^{-j\phi} e^{-j\omega_0 t} \end{aligned}$$

iObserve que las amplitudes de las exponenciales son complejas!

Además, se tiene

$$A \cos(\omega_0 t + \phi) = A \operatorname{Re}\{e^{j(\omega_0 t + \phi)}\} \quad A \sin(\omega_0 t + \phi) = A \operatorname{Im}\{e^{j(\omega_0 t + \phi)}\}$$

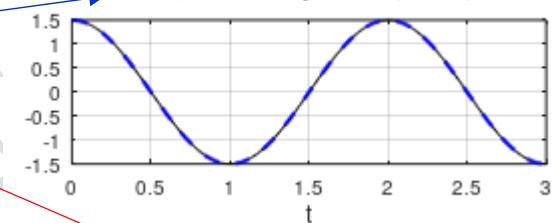
```
t=0:0.01:3;
x = 1.5*exp(j*(2*pi*0.5*t)); Exponencial compleja
```

Parte real = cos

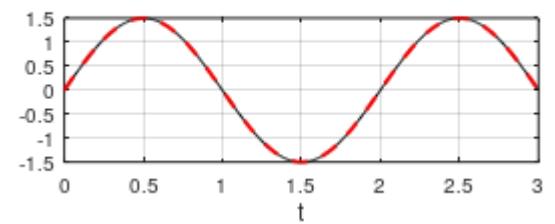
Parte imaginaria = sin

```
subplot(2,1,1);
plot(t,xr,t,xc,'linewidth',2,'b--');grid;
xlabel('t','fontsize',12);
title('Re\{1.5 e^{j2\pi 0.5t}\} y 1.5 \cos(2\pi 0.5t)');
subplot(2,1,2);
plot(t,xi,t,xs,'linewidth',2,'r--');grid;
xlabel('t','fontsize',12);
title('Im\{1.5 e^{j2\pi 0.5t}\} y 1.5 \sin(2\pi 0.5t)');
```

Re{1.5 e^{j2π0.5t}} y 1.5 cos(2π0.5t)

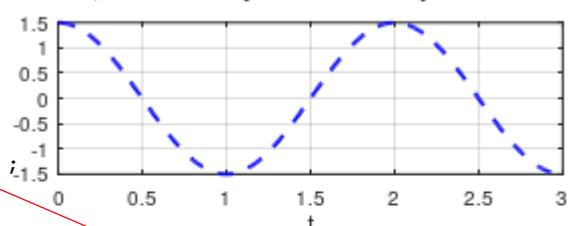


Im{1.5 e^{j2π0.5t}} y 1.5 sin(2π0.5t)

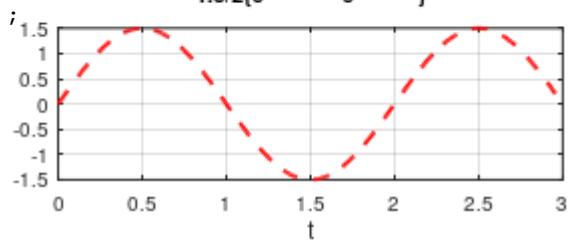


```
xc2 = 0.5*(x+conj(x));
xs2 = (0.5/j)*(x-conj(x));
subplot(2,2,1);
plot(t,xc2,'linewidth',2,'b--');grid;
xlabel('t','fontsize',12);
title('1.5/2\{e^{j2\pi 0.5t} + e^{-j2\pi 0.5t}\}');
subplot(2,2,3);
plot(t, xs2,'linewidth',2,'r--');grid;
xlabel('t','fontsize',12);
title('1.5/2\{e^{j2\pi 0.5t} - e^{-j2\pi 0.5t}\}');
```

1.5/2{e^{j2π0.5t} + e^{-j2π0.5t}}



1.5/2{e^{j2π0.5t} - e^{-j2π0.5t}}



Conjunto de exponentiales complejas relacionadas armónicamente

Es un conjunto de señales exponenciales periódicas con frecuencias fundamentales que son todas múltiplos enteros de una sola frecuencia positiva ω_0 .

$$\varphi_k(t) = e^{jk\omega_0 t}, \quad k = 0, \pm 1, \pm 2, \dots$$

para $k = 0$, $\varphi_k(t)$ es una constante

Para $k \neq 0$, $\varphi_k(t)$ es periódica con frecuencia fundamental $|k|\omega_0$, y periodo fundamental

$$\frac{2\pi}{|k|\omega_0} = \frac{T_0}{|k|}$$

La k -ésima armónica $\varphi_k(t)$ es también periódica con periodo T_0 ; es decir que en el intervalo T_0 tiene exactamente $|k|$ de sus periodos fundamentales.

En este ejemplo, podemos ver las propiedades anteriores usando la parte real (funciones coseno) de las exponentiales complejas.

Sea

$$\varphi_k(t) = e^{jk\omega_0 t}, \quad k = 0, 1, 2, 3$$

con periodo fundamental $T_0 = 1$ segundo. Entonces, $\omega_0 = 2\pi$ y se tienen las siguientes exponentiales complejas armónicamente relacionadas

$\varphi_0(t) = 1$ componente dc – valor promedio

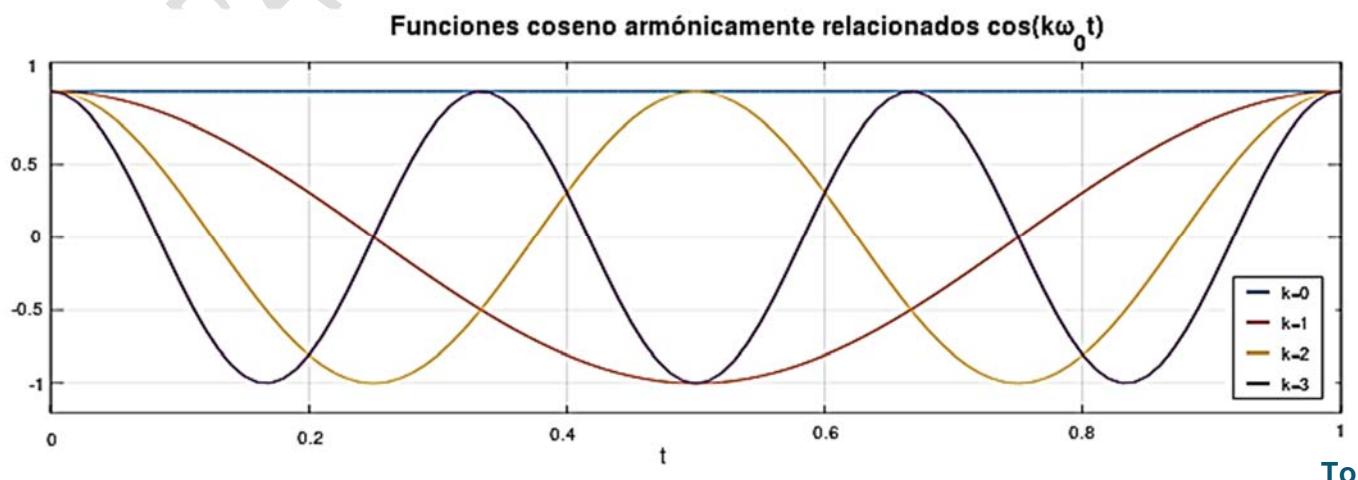
$\varphi_1(t) = e^{j\omega_0 t}$ fundamental

$\varphi_2(t) = e^{j2\omega_0 t}$ segundo armónico

$\varphi_3(t) = e^{j3\omega_0 t}$ tercer armónico

Vamos a graficar solo la parte real, es decir, las funciones cos asociadas a estas funciones exponentiales complejas, pero que mantienen la propiedad de que están armónicamente relacionadas también.

Observe en las gráficas el número de períodos (o la frecuencia) de cada componente armónica.



Si tuviese $k = [-1000, 1000]$, por ejemplo, sería muy complicado, demorado y extenso escribir cada señal para cada valor de k en una línea de código. ¿Qué puedo hacer?

Puedo usar un ciclo **for** para generar todas las señales exponenciales armónicamente relacionadas en un sola matriz. Así, cada fila corresponde a una señal exponencial con un valor de k particular. ¡Eso es una solución eficiente e inteligente!

De igual forma, puedo usar un ciclo **for** para graficar cada fila de la matriz (señal de interés).



```
t=0:0.01:1; % To = 1 periodo de la fundamental
x = zeros(4,length(t)); % Matriz con funciones armónicas - inicialización

for k=0:3
    x(k+1,:)=exp(j*k*2*pi*t); % exponenciales complejas armónicamente relacionadas
end

for k=0:3
    plot(t,real(x(k+1,:)), 'linewidth',2); % cosenos armónicamente relacionados
    hold on;
end

title('Funciones coseno armónicamente relacionados cos(k\omega_0t)', 'fontsize',14);
xlabel('t', 'fontsize',12); grid;
legend('k=0','k=1','k=2','k=3', 'location', 'southeastoutside');
axis([0 1 -1.2 1.2]);
```

Señales exponenciales complejas generales

$$x(t) = Ce^{at}$$

donde $C = |C|e^{j\theta}$ (en forma polar)

$a = r + j\omega_0$ (en forma rectangular)

Así

$$x(t) = |C|e^{rt}e^{j(\omega_0t+\theta)}$$

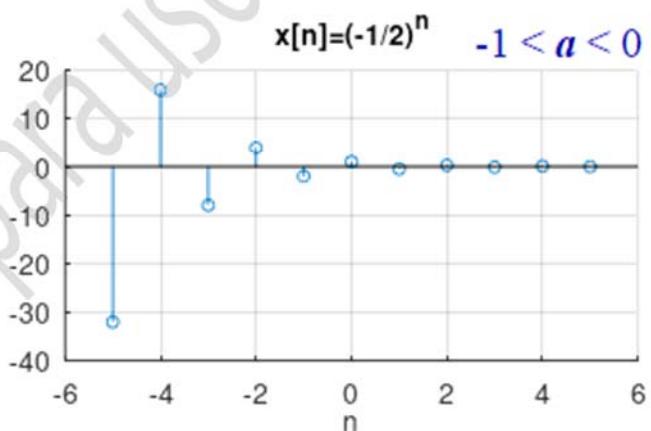
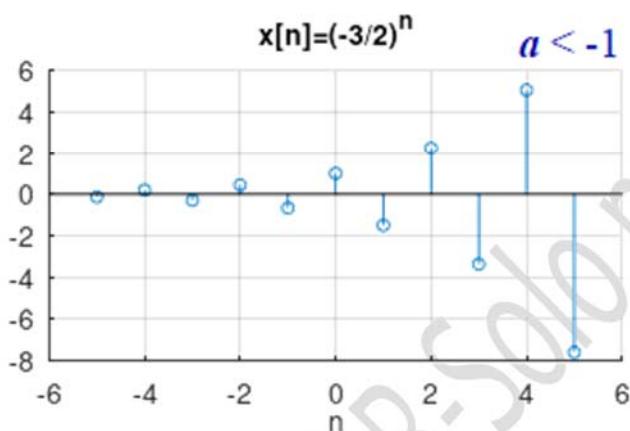
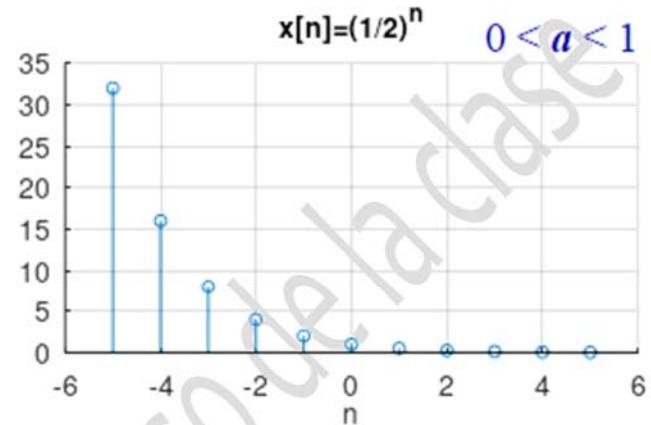
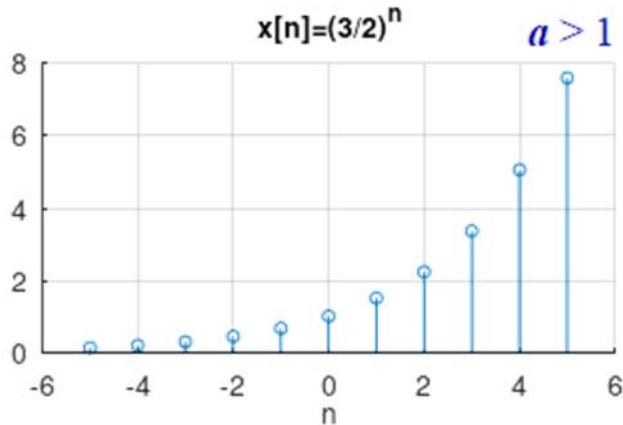
$$x(t) = |C|e^{rt}[\cos(\omega_0t + \theta) + j\sin(\omega_0t + \theta)]$$



Señal exponencial compleja discreta

$x[n] = Ca^n$ donde C y a son, en general, números complejos.

Exponencial real discreta: C y a son reales



Recuerda que para señales discretas n toma valores enteros y debes usar **stem** para graficar estas señales.



La señal exponencial compleja discreta y las señales sinusoidales discretas



$$x[n] = e^{j\omega_0 n} = \cos(\omega_0 n) + j\sin(\omega_0 n)$$

$$A\cos(\omega_0 n + \phi) = \frac{A}{2}e^{j\phi}e^{j\omega_0 n} + \frac{A}{2}e^{-j\phi}e^{-j\omega_0 n}$$



La periodicidad de exponenciales discretas no es igual a la periodicidad de exponenciales continuas.

Considere una señal exponencial compleja discreta con frecuencia $\omega_0 + 2\pi$,

$$e^{j(\omega_0 + 2\pi)n} = e^{j2\pi n}e^{j\omega_0 n} = e^{j\omega_0 n}$$

Observa que la señal con frecuencia ω_0 es idéntica a las señales con frecuencias $\omega_0 + 2\pi$, $\omega_0 + 4\pi$, ...



Así, para señales discretas exponenciales complejas (y las sinusoidales) solamente se necesita tomar en cuenta un intervalo de frecuencia de longitud 2π dentro del cual se escoge ω_0 . Generalmente se usan los intervalos

$$0 < \omega_0 < 2\pi \quad \text{o} \quad -\pi < \omega_0 < \pi$$

Para que la señal $e^{j\omega_0 n}$ sea periódica con periodo $N > 0$, se debe tener

$$e^{j\omega_0(n+N)} = e^{j\omega_0 n}$$

es decir que, $e^{j\omega_0 N} = 1$, por lo cual

$$\omega_0 N = m \cdot 2\pi \quad \text{o en forma equivalente} \quad \frac{\omega_0}{2\pi} = f_0 = \frac{m}{N} \quad (\text{es un número racional})$$



Lo anterior también es válido para señales sinusoidales discretas.



Conjunto de exponentiales periódicas relacionadas armónicamente

$$\varphi_k[n] = e^{jk(\frac{2\pi}{N})n}, \quad k = 0, \pm 1, \pm 2, \dots$$

Observa que solamente hay N exponenciales periódicas distintas en el conjunto dado.



$$\begin{aligned}\varphi_{k+N}[n] &= e^{j(k+N)\left(\frac{2\pi}{N}\right)n} \\ &= e^{jk\left(\frac{2\pi}{N}\right)n} e^{j2\pi n} \\ &= e^{jk\left(\frac{2\pi}{N}\right)n}\end{aligned}$$

En este ejemplo, podemos ver las propiedades anteriores usando la parte real (funciones coseno) de las exponenciales complejas.

Sea

$$\varphi_k[n] = e^{jk\omega_o n}$$

con periodo fundamental $N = 16$. Entonces, $\omega_o = 2\pi/16$ y **solo se tienen 16** exponentiales complejas armónicamente relacionadas ($k = 0$ a 15).

$\varphi_0[n] = 1$ componente dc – valor promedio

$$\varphi_1[n] = e^{j\frac{2\pi}{16}n} \text{ fundamental}$$

$$\varphi_2[n] = e^{j2\frac{2\pi}{16}n} \text{ segundo armónico}$$

$$\varphi_3[n] = e^{j3\frac{2\pi}{16}n} \text{ tercer armónico}$$

1

Note que

$$\varphi_{1e}[n] = e^{j16\frac{2\pi}{4}n} = e^{j8\pi n} = 1 = \varphi_0[n]$$

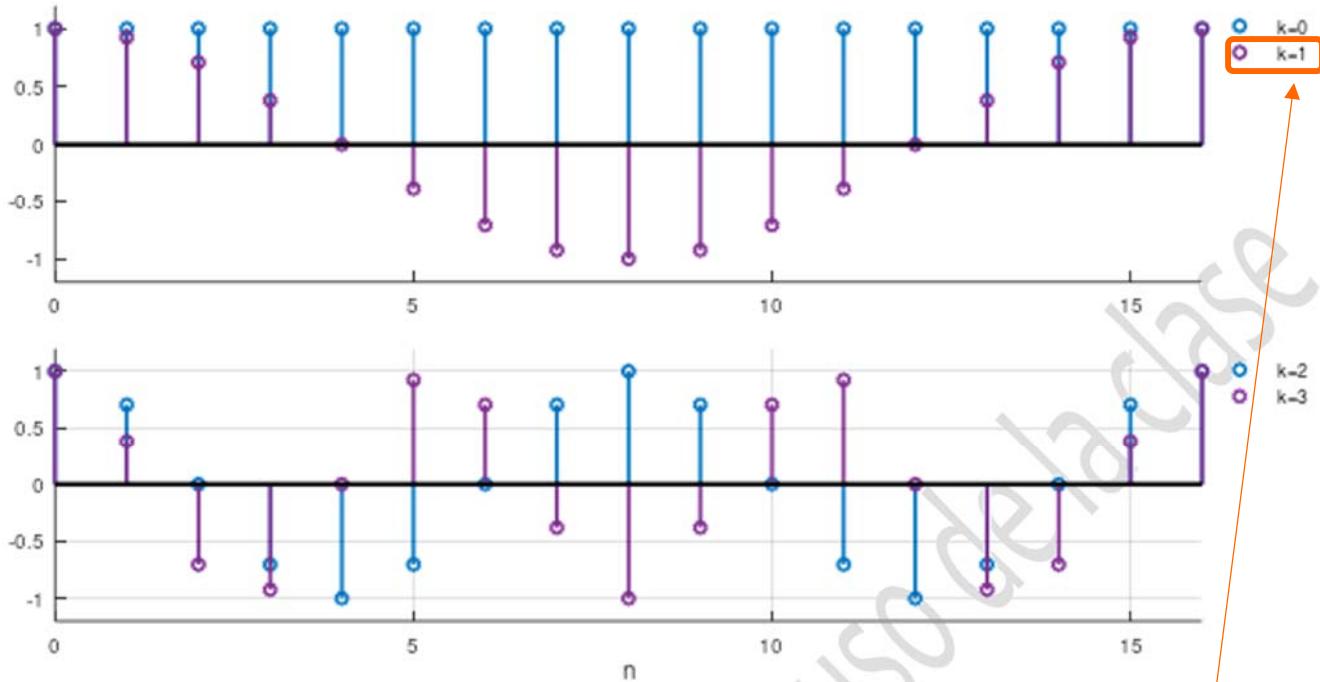
$$\varrho_{17}[n] \equiv \varrho_1[n]$$

Vamos a graficar solo la parte real, es decir, las funciones **cos** asociadas a estas funciones exponenciales complejas, pero que mantienen la propiedad de que están armónicamente relacionadas también. Solo graficamos las señales para los valores de $k = 0, 1, 2$ y 3 .

Observe en las gráficas el número de períodos (o la frecuencia) de cada componente armónica.



Funciones coseno armónicamente relacionados $\cos(k2\pi/16n)$



```

N = 16; % N = 16 periodo de la fundamental
n=0:N; % vector de soporte para un periodo de la señal fundamental - ver k=1
x = zeros(4,length(n)); % Matriz con funciones armónicas - inicialización

for k=0:3
    x(k+1,:)=exp(j*k*2*pi/N*n); % exponenciales complejas armónicamente relacionadas
end

subplot(2,1,1)
for k=0:1
    stem(n,real(x(k+1,:)), 'linewidth',2); % cosenos armónicamente relacionados
    hold on;
end
title('Funciones coseno armónicamente relacionados cos(k2\pi/16n)', 'fontsize',14);
legend('k=0', 'k=1', 'location', 'northeastoutside');
axis([0 16 -1.2 1.2]);

subplot(2,1,2)
for k=2:3
    stem(n,real(x(k+1,:)), 'linewidth',2); % cosenos armónicamente relacionados
    hold on;
end
grid
xlabel('n', 'fontsize',12);
legend('k=2', 'k=3', 'location', 'northeastoutside');
axis([0 16 -1.2 1.2]);

```



2. Sistemas lineales invariantes en el tiempo

Los sistemas lineales invariantes en el tiempo (LTI – linear time invariant) son sistemas que producen una señal de salida a partir de cualquier señal de entrada sujeto a las propiedades de **linealidad e invarianza en el tiempo**.



Linealidad

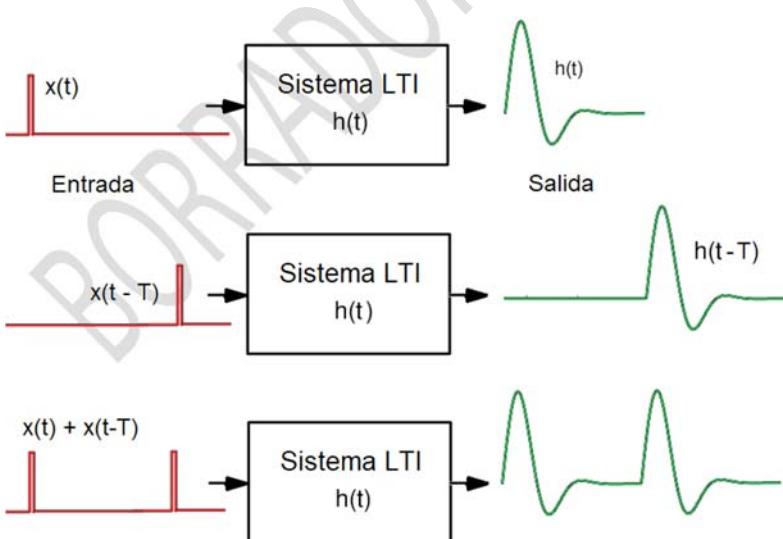
significa que la relación entre la entrada y la salida es el resultado de ecuaciones diferenciales lineales, es decir, ecuaciones diferenciales que emplean solo operadores lineales.

Un sistema lineal que mapea una entrada $x(t)$ a una salida $y(t)$ mapeará una entrada escalada $ax(t)$ a una salida $ay(t)$ igualmente escalada por el mismo factor a .

El **principio de superposición** se aplica a un sistema lineal: si el sistema mapea las entradas $x_1(t)$ y $x_2(t)$ a las salidas $y_1(t)$ y $y_2(t)$, respectivamente, entonces mapeará $x_3(t) = x_1(t) + x_2(t)$ a la salida $y_3(t)$ donde $y_3(t) = y_1(t) + y_2(t)$.

Invarianza en el tiempo

significa que si aplicamos una entrada al sistema en este instante o T segundos a partir de este instante, la salida será idéntica excepto por un retraso de tiempo de T segundos. Es decir, si la salida debida a la entrada $x(t)$ es $y(t)$, entonces la salida debida a la entrada $x(t - T)$ es $y(t - T)$. Por lo tanto, el sistema es invariante en el tiempo porque la salida no depende del momento particular en que se aplica la entrada.



La teoría de sistemas LTI es un área de matemáticas aplicadas que tiene aplicaciones directas en el análisis y diseño de circuitos eléctricos, filtros, procesamiento de señales y de imágenes, teoría de control, ingeniería mecánica, diseño de instrumentos de medición, espectroscopía de Resonancia Magnética Nuclear y muchas otras áreas técnicas donde se presentan sistemas de ecuaciones diferenciales ordinarias.

https://en.wikipedia.org/wiki/Linear_time-invariant_system

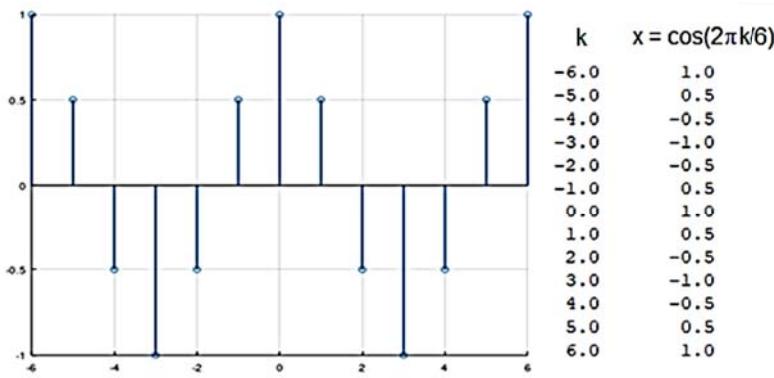


La suma de convolución

Cualquier señal discreta se puede representar como una secuencia de impulsos individuales (desplazados en el tiempo y escalados en la amplitud).

$$x[n] = \sum_{k=-\infty}^{+\infty} x[k]\delta[n - k]$$

Eso es exactamente lo que hacemos en Octave cuando generamos una señal x a partir de un vector de soporte k . Si observamos, el resultado es un vector que tiene los valores de la señal para cada valor de k . Cada uno de estos valores corresponde a un impulso retrasado (valor de k) y escalado en su amplitud (valor de la señal).



$$x[n] = 1.0 \delta[n+6] + 0.5 \delta[n+5] - 0.5 \delta[n+4] - 1.0 \delta[n+3] - 0.5 \delta[n+2] + 0.5 \delta[n+1] + 1.0 \delta[n] + 0.5 \delta[n-1] - 0.5 \delta[n-2] - 1.0 \delta[n-3] - 0.5 \delta[n+4] + 0.5 \delta[n-5] + 1.0 \delta[n-6]$$



```
k=-6:6;
x=cos(2*pi*k/6);
stem(k,x);grid;
```

Entonces, si consideramos un sistema LTI y conocemos su respuesta al impulso $h[n]$, podemos calcular la respuesta $y[n]$ del sistema a una entrada $x[n]$, ya que la salida es la superposición de las respuestas escaladas del sistema a cada uno de los impulsos desplazados (que conforman la señal de entrada).

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n - k]$$



En el siguiente ejemplo se implementa la ecuación anterior usando Octave de una forma muy básica, solo para observar el proceso. Luego veremos formas más eficientes de hacer lo mismo.

Sea la respuesta al impulso de un sistema LTI $h[n] = -\delta[n + 1] + 2\delta[n] + \delta[n - 1] + \delta[n - 3]$ y la señal de entrada $x[n] = -\delta[n + 1] + \delta[n] + \delta[n - 1]$. Determine la señal de salida.

La señal de salida la podemos calcular como la superposición de la respuesta al impulso ponderada y corrida como indica la ecuación anterior.



Para ello, vamos primero a sustituir la variable n de la señal $x[n]$ por k , i.e., $x[k]$. Así, tenemos los valores para $x[-1] = -1$, $x[0] = 1$, $x[1] = 1$.

En Octave, vamos a utilizar la función **circshift** para lograr el corrimiento de la respuesta $h[n-k]$, y obtener $h[n+1]$, $h[n]$, $h[n-1]$.

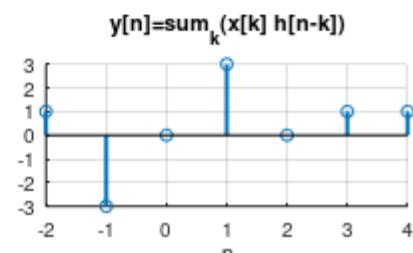
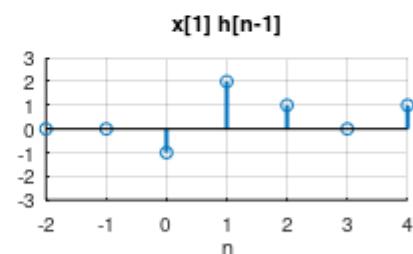
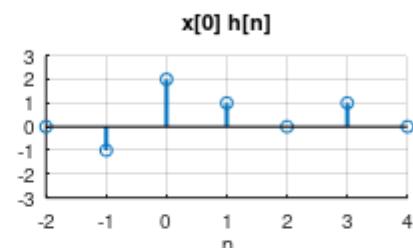
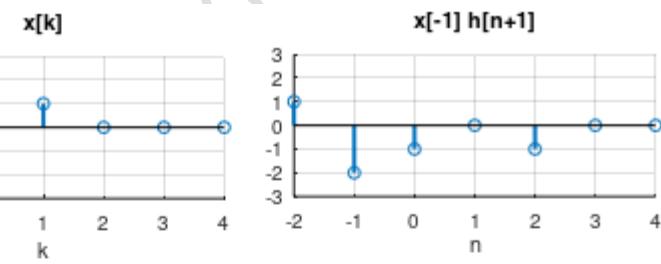
En este sentido, hay dos puntos importantes que vamos a tomar en consideración.

- (i) El resultado es la suma de las respuestas al impulso ponderadas y corridas. Para sumar vectores en Octave, éstos tienen que tener la misma longitud. Por lo tanto es necesario ajustar la longitud de todos los vectores, para lo cual vamos a extender el vector $h[n]$ original, agregando ceros a izquierda y derecha. La cantidad de ceros la hemos calculado para que podamos realizar los corrimientos necesarios de $h[n-k]$ y que todos tengan la misma longitud.
- (ii) Agregar ceros también es necesario porque la función circshift realiza un corrimiento circular a izquierda o derecha de los elementos del vector. Por lo tanto, para obtener estos corrimientos sin afectar el vector original debemos tener ceros a izquierda y derecha.

```

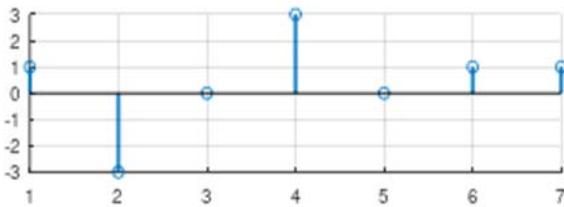
n=-2:4; % soporte de la señal de salida
h=[0 -1 2 1 0 1 0]; % respuesta al impulso h[n]
x=[0 -1 1 1 0 0 0]; % señal de entrada
y=zeros(size(h));
for k=-1:1
    yk=x(k+3)*circshift(h,[0,k]);
    y=y+yk
end
v=[-2 4 -3 3];
subplot(4,2,1)
stem(n,x,'linewidth',2);grid; axis(v);
title('x[k]'); xlabel('k');
subplot(4,2,2)
stem(n,x(2)*circshift(h,[0,-1]),'linewidth',2);grid;
title('x[-1] h[n+1]'); xlabel('n'); axis(v);
subplot(4,2,4)
stem(n,x(3)*circshift(h,[0,0]),'linewidth',2);grid
title('x[0] h[n]'); xlabel('n'); axis(v);
subplot(4,2,6)
stem(n,x(4)*circshift(h,[0,1]),'linewidth',2);grid
title('x[1] h[n-1]'); xlabel('n'); axis(v);
subplot(4,2,8)
stem(n,y,'linewidth',2);grid;axis(v);
title('y[n]=sum_k(x[k] h[n-k]))');
xlabel('n');

```



Octave tiene la función **conv** que realiza la convolución de dos vectores. En el siguiente ejemplo vemos que fácilmente se puede obtener el resultado anterior con esta función.

conv (a, b) convoluciona los vectores a y b. El vector de salida tiene una longitud igual a longitud (*a*) + longitud (*b*) - 1.



```
x=[-1 1 1]; % señal de entrada  
h=[-1 2 1 0 1]; % respuesta la impulso  
y=conv(x,h);  
stem(y,'linewidth',2);grid;
```



Pero observa que esta función no involucra los vectores de soportes de los vectores **a** y **b** en el caso de que éstos correspondan a los valores de una señal.

Es importante en el análisis de señales y sistemas no solo conocer la señal de salida, sino también el intervalo de tiempo en que existen las señales y la respuesta.

El vector de salida es el mismo que en el caso anterior, pero no tiene soporte. Se ha graficado vs el índice de los elementos del vector.

¿Puedo crear una función que realice la convolución de dos señales usando la función **conv** y que incluya sus soportes?



Vamos a crear una función **conv_n** que realice la convolución de dos señales **x** y **y**, pero que también considere los vectores de soporte de las mismas. De tal forma que se pueda obtener la señal de salida completa – vector con valores de la señal y valores del tiempo correspondientes.

```
function [z,nz] = conv_n (x, nx, y, ny)  
nz = (nx(1)+ ny(1)):(nx(end)+ny(end));  
z = conv(x,y);  
endfunction
```



```

x=[-1 1 1];           % señal de entrada
nx=[-1 0 1];          % soporte de la señal x
h=[-1 2 1 0 1];       % respuesta la impulso
nh=[-1 0 1 2 3];      % soporte de la señal h

[y,n]=conv_n(x,nx,h,nh);
stem(n,y,'linewidth',2);grid;

```



Observa que obtenemos el mismo resultado del vector de salida y ahora sí tenemos el vector de soporte correspondiente. Es la misma respuesta obtenida en el primer ejemplo, pero de una manera muy sencilla, usando las señales originales (sin agregar ceros) y sus vectores de soporte, y la función conv.

La integral de convolución

La integral de convolución es el equivalente de la suma de convolución para señales y sistemas de tiempo continuo. Así, la salida de un sistema LTI basado en una señal arbitraria, $x(t)$, y la respuesta al impulso, $h(t)$, está dada por

$$y(t) \equiv \int_{-\infty}^{+\infty} x(\tau)h(t - \tau)d\tau$$

$$y(t) = x(t) * h(t)$$

Recuerda que en un computador *todas las señales son discretas*, y que solo podemos simular una señal continua. Para ello usamos un incremento pequeño de la variable tiempo y graficamos la señal con el comando **plot**.

Entonces puedo modificar la función conv_n para señales continuas.



Sí, pero tienes que considerar dos cosas importantes:

- (i) es necesario que el intervalo de tiempo (diferencial dt) de los vectores de soporte de las señales que se van a convolucionar sea el mismo, y
- (ii) debes multiplicar la convolución por el diferencial de tiempo dt para obtener la amplitud correcta del resultado.

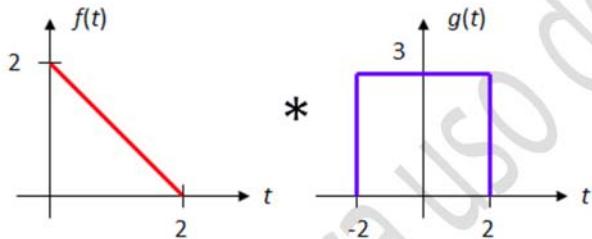


Modificación de la función **conv_n** para funciones de “**tiempo continuo**” → **conv_t**.

```
function [z,tz] = conv_t (x, tx, y, ty)
dtx = tx(2)-tx(1); dty = ty(2)-ty(1);
if dty ~= dtx
    disp('ERROR: Vectores con intervalos de tiempo diferentes')
    disp(dtx); disp(dty)
else
    tzi = tx(1)+ ty(1); tzf =(tx(end)+ty(end));
    z = dtx*conv(x,y);
    tz=linspace(tzi,tzf,length(z));
end
endfunction
```

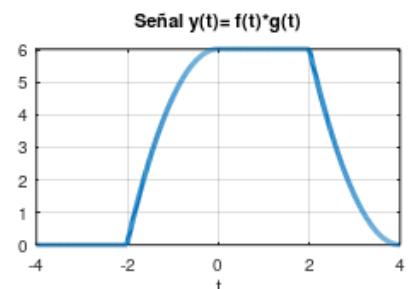
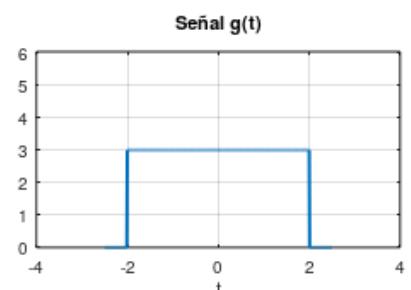
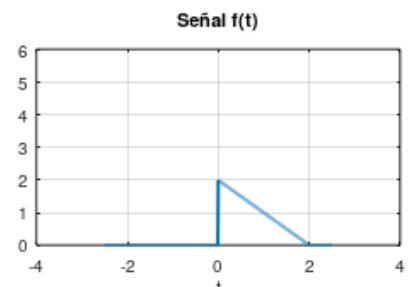


Calcula la convolución de las señales continuas $f(t)$ y $g(t)$ que se muestran en la siguiente figura.



```
t=-2.5:0.01:2.5;
f=(t>=0 & t<=2).*(-t+2);
g=3*(t>=-2 & t<=2);
[y,ty]=conv_t(f,t,g,t);

v=[-4 4 0 max(y)];
subplot(3,1,1)
plot(t,f,'linewidth',2);grid;axis(v);
xlabel('t');title('Señal f(t)');
subplot(3,1,2)
plot(t,g,'linewidth',2);grid;axis(v);
xlabel('t');title('Señal g(t)');
subplot(3,1,3)
plot(ty,y,'linewidth',3);grid;axis(v);
xlabel('t');title('Señal y(t)= f(t)*g(t)');
```



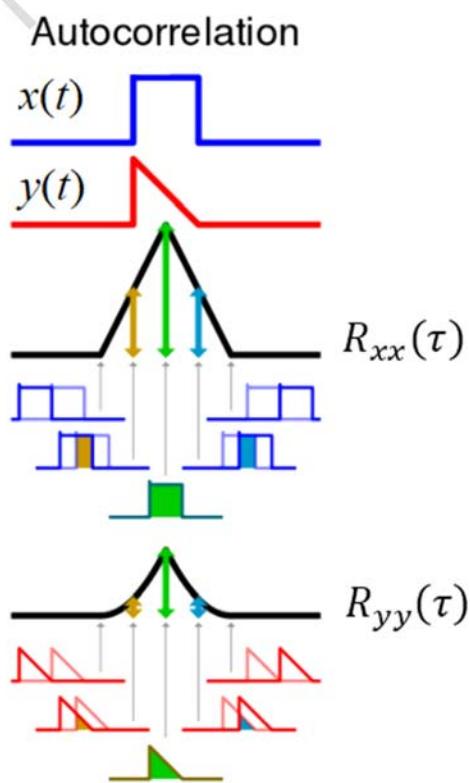
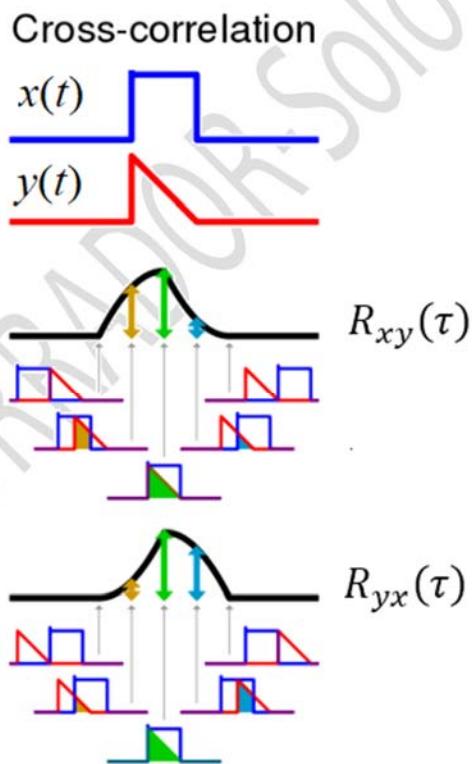
La correlación

Es una función de gran importancia asociada a la convolución y definida como

$$R_{xy}(\tau) = \int_{-\infty}^{+\infty} x(t)y(t + \tau)dt$$

- La correlación calcula una medida de similitud de dos señales en **función del desplazamiento (o retardo)** de una con respecto a la otra. El resultado alcanza un máximo en el instante cuando las dos señales mejor coinciden.
- Se usa comúnmente para buscar una señal extensa para una característica conocida más corta.
- Cuando se tienen dos secuencias distintas, se denomina correlación cruzada, y cuando es una secuencia consigo misma, se denomina autocorrelación.
- En la autocorrelación siempre habrá un pico, y su tamaño será la energía de la señal.

Tiene aplicaciones en reconocimiento de patrones, análisis de partículas individuales, tomografía electrónica, promediado, criptoanálisis y neurofisiología, entre muchas otras.



Implementación en Octave

Si observamos con detenimiento, ¡la correlación de dos señales se puede implementar como la convolución de la primera señal con el inverso de la segunda señal!

Entonces, podría usar las funciones anteriores de convolución y modificarlas para implementar funciones de correlación para señales discretas y continuas.



```
function [z,nz] = corr_n(x,nx,y,ny)
ny= -fliplr(ny); % inversión de y[n]
y = fliplr(y);
z = conv(x,y);
nz = (nx(1)+ny(1)):(nx(end)+ny(end));
end
endfunction
```

```
function [z,tz] = corr_t(x,tx,y,ty)
% Es necesario que los soportes de ambas señales tengan igual
% intervalo de muestreo dt
dtx = tx(2)-tx(1); dty = ty(2)-ty(1);
if dtx ~= dty
    disp('ERROR: Vectores con intervalos de tiempo diferentes')
    disp(dtx); disp(dty)
else
dt = tx(2)-tx(1);
y = fliplr(y); % inversión de y(t)
ty= -fliplr(ty);
tzi= tx(1)+ty(1);
tzf= tx(end)+ty(end);
z = dt*conv(x,y);
tz=linspace(tzi,tzf,length(z));
end
endfunction
```

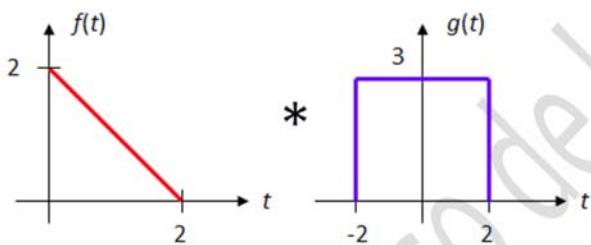




Hagamos un ejemplo.

Calcula las siguientes operaciones de correlación con las señales continuas $f(t)$ y $g(t)$ que se muestran en la figura.

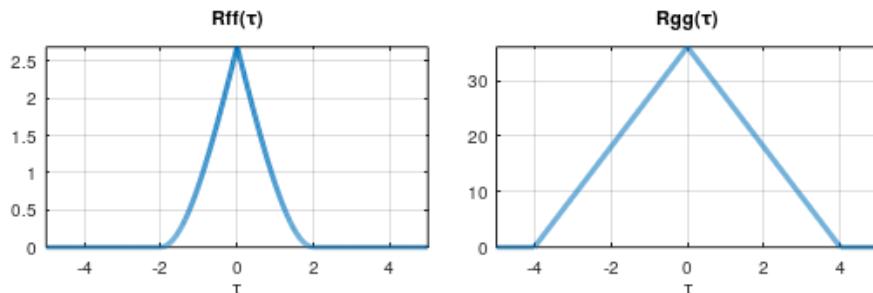
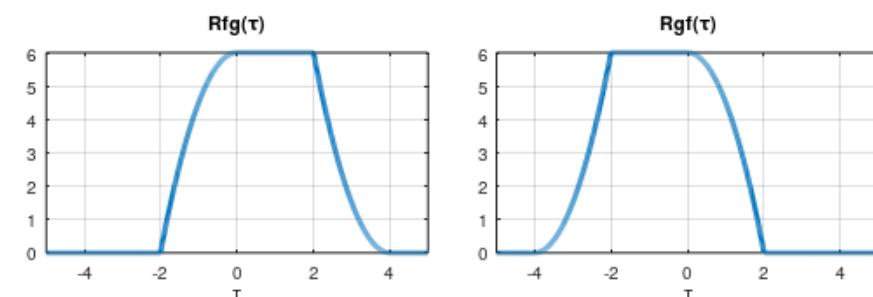
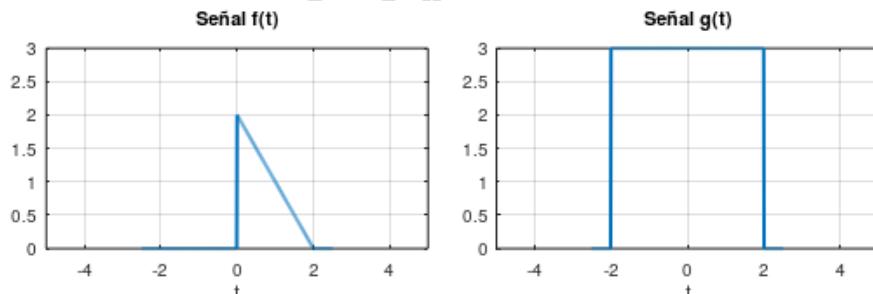
- a. $R_{fg}(\tau)$
- b. $R_{gf}(\tau)$
- c. $R_{ff}(\tau)$
- d. $R_{gg}(\tau)$



```
t=-2.5:0.01:2.5;  
f=(t>=0 & t<=2).*(-t+2);  
g=3*(t>=-2 & t<=2);  
[ya,tya]=corr_t(f,t,g,t);  
[yb,tyb]=corr_t(g,t,f,t);  
[yc,tyc]=corr_t(f,t,f,t);  
[yd,tyd]=corr_t(g,t,g,t);
```

```
subplot(3,2,1)  
plot(t,f);  
subplot(3,2,2)  
plot(t,g);  
subplot(3,2,3)  
plot(tya,ya);  
subplot(3,2,4)  
plot(tyb,yb);  
subplot(3,2,5)  
plot(tyc,yc);  
subplot(3,2,6)  
plot(tyd,yd);
```

El código no incluye el formato de las gráficas.



BORRADOR. Solo para uso de la clase

