Notas PSTD

August 2, 2020

0.1 Procesamiento de Señales en Tiempo Discreto - Notas de Clase y Ejercicios

Docente: **Dr. Ing. Mario Hueda**

Estudiate: **Ing. Franco Batrouni**

[1]:
```
%%javascript
MathJax.Hub.Config({
    TeX: { equationNumbers: { autoNumber: "AMS" } }
});
```

```
<IPython.core.display.Javascript object>
```

[2]:
```
%%javascript
MathJax.Hub.Queue(
  ["resetEquationNumbers", MathJax.InputJax.TeX],
  ["PreProcess", MathJax.Hub],
  ["Reprocess", MathJax.Hub]
);
```

```
<IPython.core.display.Javascript object>
```

1 Señales en tiempo discreto

Se llama señal en tiempo discreto a una secuencia de números en función de n , en la que $n \in \mathbb{N}$ es un índice discreto.

Por ejemplo:

$$x[n] = x_a[nT_s]$$

Donde T_s es el (sampling period) período de muestreo en el caso de que la secuencia provenga de un muestreo de una señal en tiempo continuo $x_a(t)$.

- Cuando nos referimos a **tiempo-discreto** , se indica que se está muestrando una determinada señal periódicamente con período T_s.
- Cuando nos referimos a que la amplitud está quantizada, o a una **señal digital** nos referimos a que se ha convertido de analógico a digital **ADC** con una cierta resolución.

1.1 Impulso unitario

El impulso unitario se define para señales en tiempo discreto de la siguiente manera:

$$\delta[n] = \begin{cases} 1 \text{ si } n = 0 \\ 0 \text{ si } n \neq 0 \end{cases} \tag{1}$$

Entre otras cosas ésta función nos permite expresar cualquier secuencia como una combinación lineal de impulsos desplazados :

$$x[n] = \sum_k x[k]\delta[n-k]$$

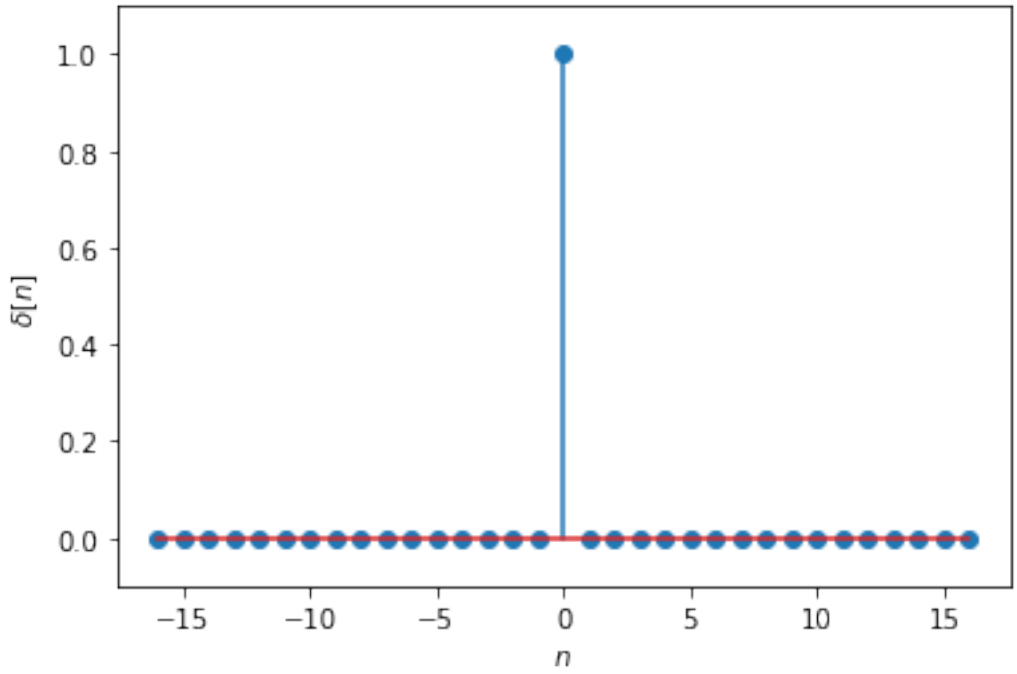
1.1.1 Implementación en Python

En python podemos definirlo de la siguiente manera:

[3]:
```
# No me interesan los warnings (ninguno en este caso)
# Si algo no llegara a funcionar correctamente quitaría este inicio de código␣
 ↪para ver q pasa
import warnings
warnings.filterwarnings("ignore")
```

[4]:
```
# Importamos librerías necesarias
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Definición de la función impulso unitario (delta dirac)
def dirac(k):
    return np.where(k == 0, 1.0, 0.0)


# Creación del indice de muestras desde n_inicial hasta n_final
n_inicial = -16
n_final = 17
n = np.arange(n_inicial, n_final) # Vector cubriendo el intervalo␣
 ↪[n_inicial,nfinal)

# Aplicamos la función impulso sobre el índice
x = dirac(n)

# Plot de los "chupetines"

plt.stem(n, x) ;

# Configuración de las etiquetas para los ejes
plt.xlabel('$n$');
```

```
plt.ylabel('$\delta[n]$');
plt.ylim([-0.1, 1.1]);
```



1.2 Escalón Unitario

La función escalón unitario se define de la siguiente manera:

$$\mu[n] = \begin{cases} 1 \text{ si } n \geq 0 \\ 0 \text{ si } n < 0 \end{cases} \tag{2}$$

Equivalentemente se puede definir usando el impulso unitario usando una sumatoria (una integral se usaría en el caso de tiempo contínuo) :

$$\mu[n] = \sum_{k=-\infty}^{n} \delta[k] \tag{3}$$

- El producto de la ecuación anterior es nulo excepto para $n = k$

Y en sentido inverso se puede usar el escalón unitario para definir el impulso unitario de la siguiente manera:

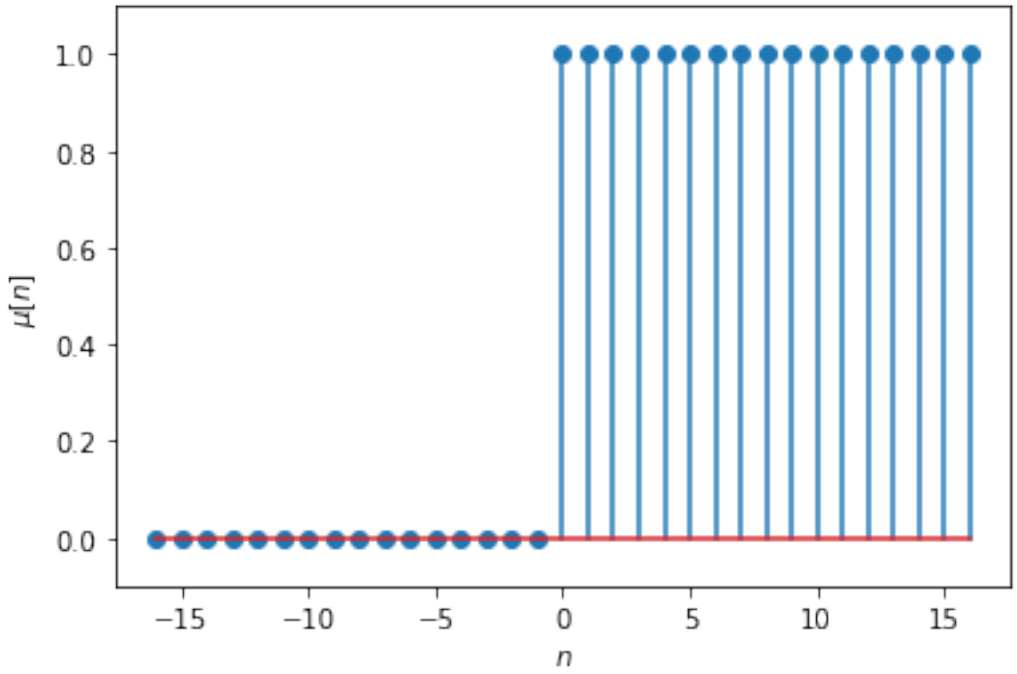
$$\delta[n] = \mu[n] - \mu[n-1]$$

1.2.1 Implementación

En PYTHON podemos implementar el escalón de la siguiente manera:

[5]:
```
# Defino el la función escalón unitario (step)
def step(k):
    return np.where(k >= 0, 1.0, 0.0)

# Aplico la función escalón sobre el índice creado anteriormente

x = step(n)

# Plot de los "chupetines"
plt.stem(n, x)

# Configuración de las etiquetas del plot
plt.xlabel('$n$')
plt.ylabel('$\mu[n]$')
plt.ylim([-0.1, 1.1])
```

[5]: (-0.1, 1.1)



1.3 Señal exponencial compleja

La exponencial compleja se define de la siguiente manera:

$$x[n] = A\alpha^n$$

Para A y $\alpha \in \mathbb{C}$ de modo que :

$$A =\mid A \mid e^{j\phi}$$

$$\alpha =\mid \alpha \mid e^{j\Omega_0}$$

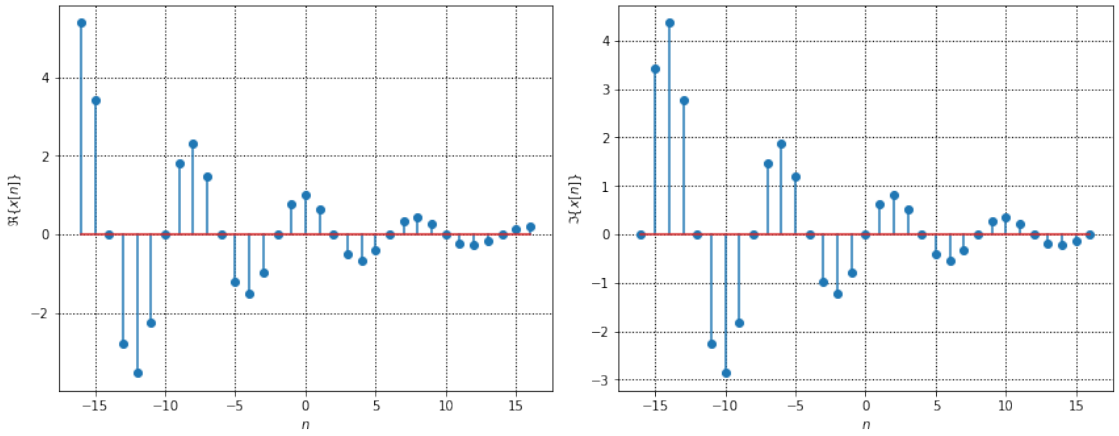
Por lo tanto:

$$x[n] =\mid A \mid\mid \alpha \mid^n [\cos(\Omega_0 n + \phi) + j\sin(\Omega_0 n + \phi)]$$

[6]:
```
# Defino la función exponencial compleja
def exp_cplx(k, norm_A,norm_alpha, Omega, phi):
    return norm_A*norm_alpha**k*np.exp( 1j * ( Omega * k + phi))


#Defino la función que realiza los plots de la señal compleja en su parte REAL
 ↪e IMAG
def plot_cplx_signal(k, x):
    plt.figure(figsize=(12, 4.8)) # Tamaño del plot 12 por 4.8

    plt.subplot(121)
    plt.stem(k, np.real(x))
    plt.xlabel('$n$')
    plt.ylabel(r'$\Re \{ x[n] \}$')
    plt.grid(color='k', linestyle=':', linewidth=1)

    plt.subplot(122)
    plt.stem(k, np.imag(x))
    plt.xlabel('$n$')
    plt.ylabel(r'$\Im \{ x[n] \}$')
    plt.grid(color='k', linestyle=':', linewidth=1)
    plt.tight_layout()
```

[7]:
```
# Defino los parámetros
norm_A = 1
norm_alpha=0.9
m=1
N=8
Omega=2*np.pi*m/N
phi=0

# Aplico la función para los parametros definidos sobre el índice n
```

```
x = exp_cplx(n, norm_A,norm_alpha, Omega, phi)


# Plot de la función sobre los vectores definidos

plot_cplx_signal(n, x)
```



1.4 Periodicidad

Para analizar la periodicidad podemos considerar como ejemplo el caso en que $\mid \alpha \mid = 1$ lo que nos permite escribir :

$$x_0[n] = Ae^{j\Omega_0 n}$$

$$x_1[n] = Ae^{j\Omega_1 n}$$

donde

$$\Omega_1 = \Omega_0 + 2\pi k$$

para $k \in \mathbb{Z}$ se cumple que

$$x_0[n] = x_1[n]$$

Lo que nos deja claro que las señales exponenciales complejas en tiempo discreto son siempre periódicas en Ω_0 con período 2π , por lo cuál sólo necesita considerarse en un intervalo de longitud 2π

De forma más general planteamos la siguiente condicion de periodicidad:

$$x[n+N] = Ae^{j\Omega_0 n}e^{j\Omega_0 N} = x[n]$$

Para lo cual debe cumplirse que :

$$e^{j\Omega_0 N} = 1$$

Lo que significa que :

$$\Omega_0 N = 2\pi m$$

$$\frac{\Omega_0}{2\pi} = \frac{m}{N}$$

De donde se deduce que siendo m , n y $N \in \mathbb{Z}$ **la función** $x[n]$ **es periódica** en n si y sólo si se cumple :

$$\frac{\Omega_0}{2\pi} \in \mathbb{Q}$$

- En clase como ejemplo se considera $ e^{j2n} $ y se pregunta si es periódica en n y como $\frac{\Omega_0}{2\pi}$ no es **racional** se deduce que **no es periódica**

1.5 Rapidez y Frecuencia

En una señal de tiempo continuo al aumentar la pulsación ω_o aumenta la rapidez de la señal en todos los casos.

En cambio en una señal de tiempo discreto la **rapidez** de las señales: * Aumenta cuando Ω_0 va de 0 a π * Disminuye cuando Ω_0 va de π a 2π

Como se aclara en el libro de referencia *página 15* la pulsación Ω_0 en :

- La vecindad de $2k\pi$ se considera de **baja rapidez**
- La vecindad de $\pi(2k+1)$ se considera de **alta rapidez**

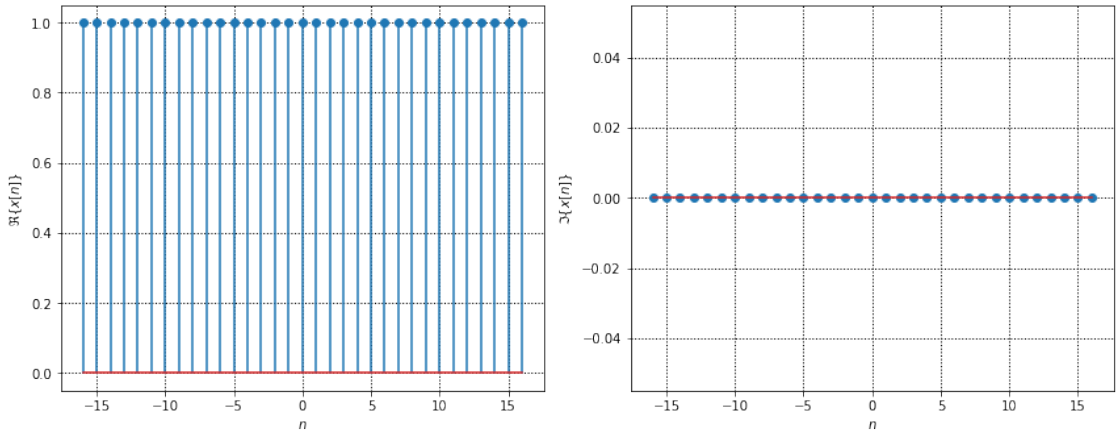
1.5.1 Implementación interactiva de los ejemplos en python

Para los ejemplos de la página 15 del libro .

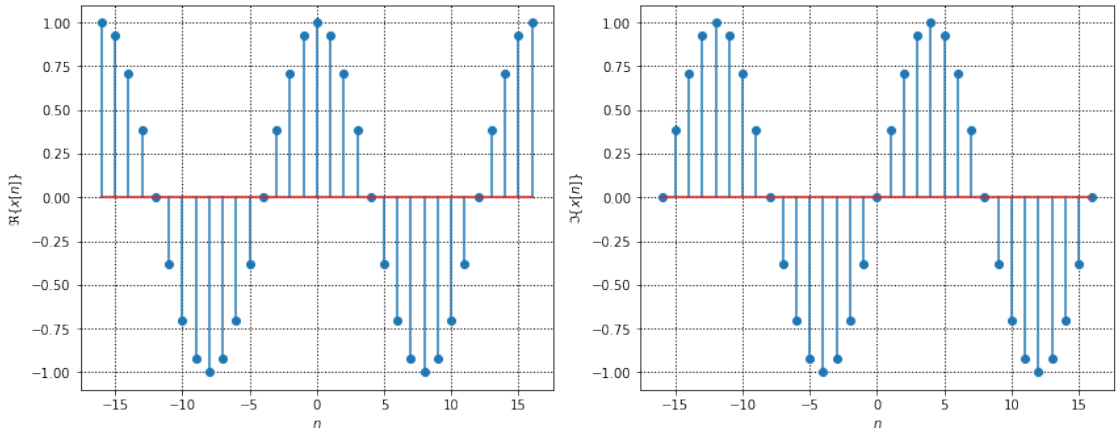
Se puede ver que recorriendo los ejemplos de arriba abajo estamos aumentando la pulsación Ω_0 de 0 a π y recorriendo de abajo hacia arriba estamos aumentando de π a 2π

[8]:
```
# Aquí código mostrando los ejemplos de las filminas
norm_A = 1
norm_alpha=1. # Notar q el . está para denotar q es un float, pq ésta función␣
 ↪no toma int
phi=0
```

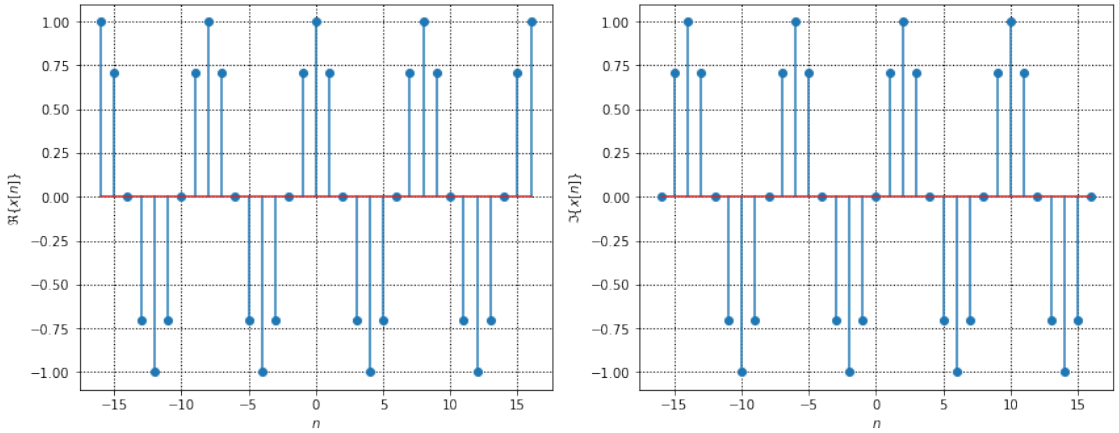
[9]:
```
# Para Omega = 0 o 2*pi
Omega=0

x = exp_cplx(n, norm_A,norm_alpha, Omega, phi)

# Plot de la función sobre los vectores definidos

plot_cplx_signal(n, x)
```



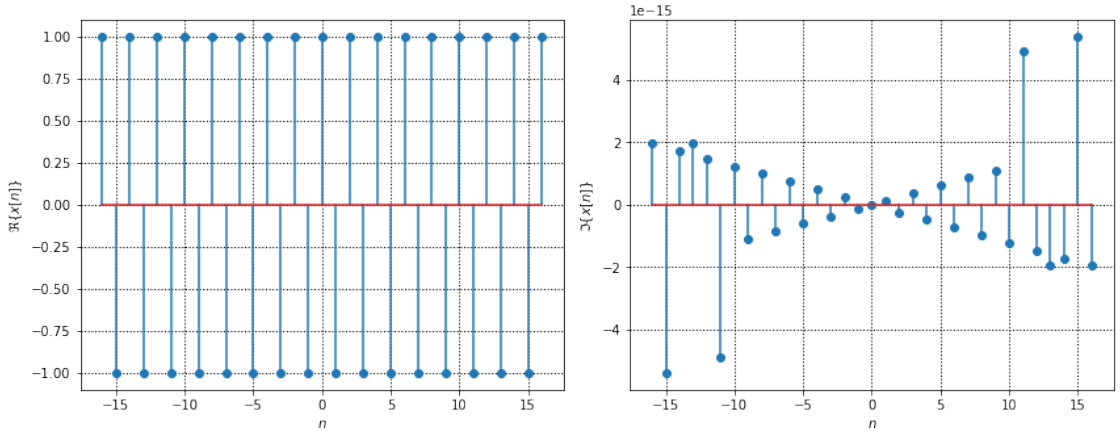
[10]:
```
# Para Omega = (1/8)*pi   o  (15/8)*pi (período 16)
Omega=np.pi/8

x = exp_cplx(n, norm_A,norm_alpha, Omega, phi)

# Plot de la función sobre los vectores definidos

plot_cplx_signal(n, x)
```



[11]:
```
# Para Omega = pi/4 (7/4)*pi (período 8)
Omega=np.pi/4

x = exp_cplx(n, norm_A,norm_alpha, Omega, phi)

# Plot de la función sobre los vectores definidos

plot_cplx_signal(n, x)
```



[12]:
```
# Para Omega = pi  (período 2)
Omega=np.pi

x = exp_cplx(n, norm_A,norm_alpha, Omega, phi)

# Plot de la función sobre los vectores definidos

# Notar el error en la parte imaginaria del orden de 1e-15 por la precisión␣
 ↪para pi en
# la librería numpy  . Lo mismo pasaría en el primer ejemplo si en lugar de␣
 ↪Omega igual a 0
# Hubiesemos usado Omega = 2*pi

plot_cplx_signal(n, x)
```

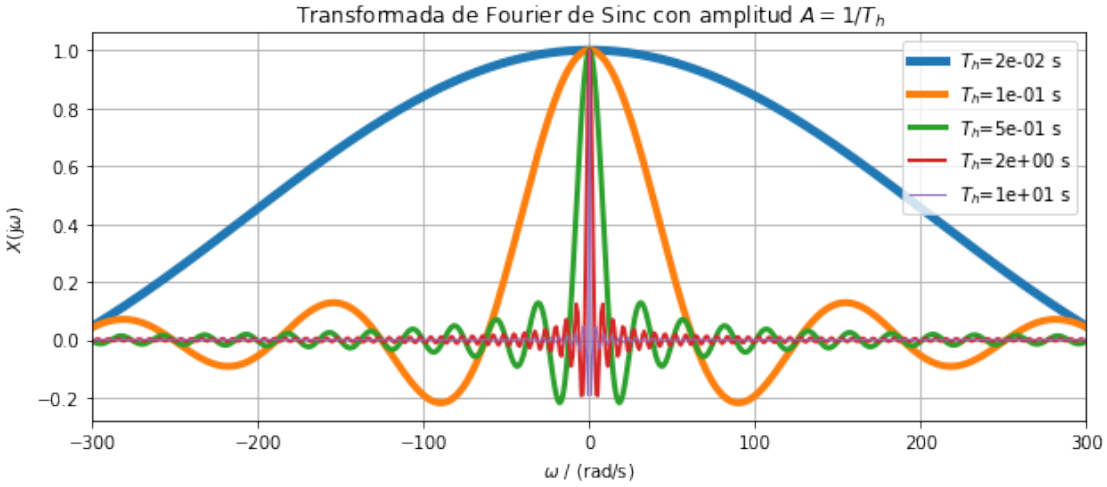


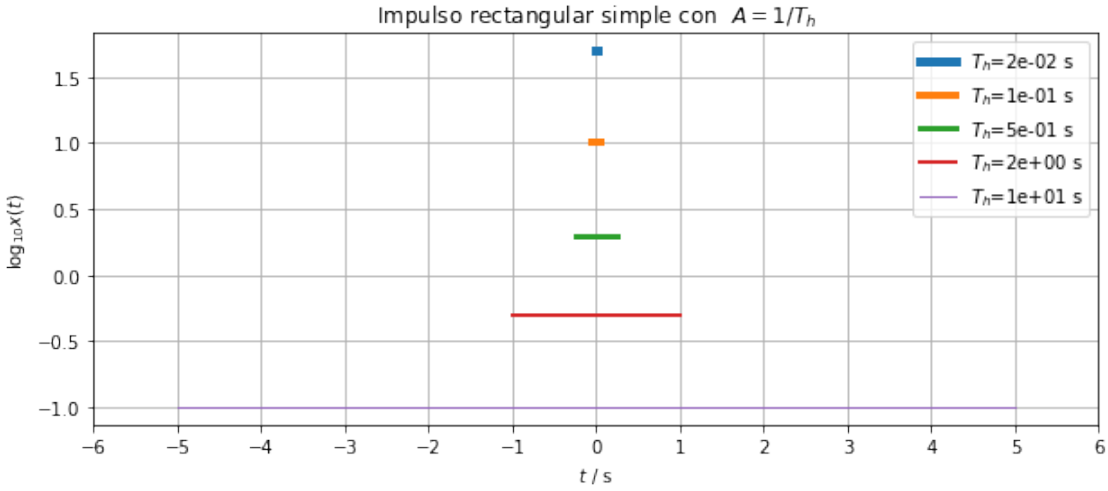
1.6 Señal Sinc

[13]:
```
import numpy as np
import matplotlib.pyplot as plt


def my_sinc(x):  # usamos  sinc(x) = sin(x)/x, por lo q dividimos el argumento␣
 ↪por pi:
    return np.sinc(x/np.pi)
```

[14]:
```
Th_des = [0.02, 0.1, 0.5, 2, 10]

plt.figure(figsize=(10, 4))
om = np.arange(-300, 300+1, 1)
for idx, Th in enumerate(Th_des):
    A = 1/Th  # Para que la amplitud del sinc sea 1
    # Transformada de Fourier pulso sinc
    X = A*Th * my_sinc(om*Th/2)
    plt.plot(om, X, label=r'$T_h$=%1.0e s' % Th, lw=5-idx)
plt.legend()
plt.title(r'Transformada de Fourier de Sinc con amplitud $A=1/T_h$')
plt.xlabel(r'$\omega$ / (rad/s)')
plt.ylabel(r'$X(\mathrm{j}\omega)$')
plt.xlim(om[0], om[-1])
plt.grid(True)
plt.savefig('uno.pdf')

plt.figure(figsize=(10, 4))
for idx, Th in enumerate(Th_des):
    A = 1/Th  # Para que la amplitud del sinc sea 1
    plt.plot([-20, -Th/2, -Th/2, +Th/2, +Th/2, 20],
```

```
            np.log10([0, 0, A, A, 0, 0]),
            label=r'$T_h$=%1.0e s' % Th,
            lw=5-idx)
plt.xlim(-6, +6)
plt.xticks(np.arange(-6, 6+1, 1))
plt.legend()
plt.title(r'Impulso rectangular simple con  $A=1/T_h$')
plt.xlabel(r'$t$ / s')
plt.ylabel(r'$\log_{10} x(t)$')
plt.grid(True)
plt.savefig('dos.pdf')
```





2 Sistemas en Tiempo Discreto

Un sistema en tiempo discreto es una transformación T de una secuencia de entrada en otra secuencia de salida.

La podemos denominar de la siguiente manera:

$$y[n] = T\{x[n]\}$$

Por ejemplo:

- Sistema de retardo ideal $y[n] = x[n - n_d]$
- Sistema sin memoria $y[n] = [x[n]]^2$

2.1 Sistemas Lineales

Si un sistema cumple con las propiedades de **aditividad** y **homogeneidad** se dice que es lineal.

- Aditividad

$$T\{x_1[n] + x_2[n]\} = T\{x_1[n]\} + T\{x_2[n]\}$$

- Homogeneidad

$$T\{ax_1[n]\} = aT\{x_1[n]\}\}$$

Si es lineal ambas propiedades se pueden ver condensadas en el principio de **superposición** que puede ser expresado de la siguiente manera:

Si la entrada del sistema es:

$$x[n] = \sum_k a_k x_k[n]$$

Su salida resulta ser :

$$y[n] = \sum_k a_k y_k[n]$$

Donde $y_k[n] = T\{x_k[n]\}$

2.2 Sistemas Causales

Se denomina causal a un sistema que depende de los valores actuales y pasados de sus entradas. Si consideramos el índice discreto desde un instante n_0 la salida depende sólo de instantes $n \leq n_0$

2.2.1 Ejemplo de sistema causal

Si su salida depende de desplazamientos a la derecha de su señal de entrada y de su señal de entrada actual. Esto es:

$$y[n] = x[n] - x[n-k]$$

con $k > 0$

2.2.2 Ejemplo de sistema no-causal

Si su salida depende de desplazamientos a la izquierda de su señal de entrada y de su señal de entrada actual. Esto es:

$$y[n] = x[n+k] - x[n]$$

con $k > 0$

2.3 Sistemas invariantes en el tiempo

Se dice que un sistema es invariante en el tiempo si su respuesta al impulso no cambia al pasar el tiempo.

Si hay cualquier desplazamiento temporal en la secuencia de entrada, el mismo desplazamiento temporal se refleja en la señal de salida.

Es decir que si:

$$y_0[n] = T\{x[n]\}$$

$$y_1[n] = T\{x[n-n_d]\} = y_0[n-n_d]$$

2.3.1 Ejemplo de canal de comunicaciones

Por ejemplo un canal de comunicaciones cualquiera en la práctica sufre cambios en el tiempo debidos a cambios en temperatura , o cualquier otro parámetro físico.

De todos modos es válido modelar ese canal como si esos cambios no existieran y luego realizar ajustes en el modelo para adaptar el sistema idealizado con la realidad.

2.4 SLIT (LTI) Sistemas Lineales Invariantes en el Tiempo

Teniendo en cuenta que **toda señal en tiempo discreto puede expresarse como una combinación lineal de impulsos desplazados** :

$$x[n] = \sum_k x[k]\delta[n-k]$$

La salida del sistema puede expresarse como :

$$y[n] = T\left\{\sum_k x[k]\delta[n-k]\right\}$$

Por la propiedad de **linearidad** se cumple que :

Siendo la **respuesta al impulso** unitario del sistema

$$h_k[n] = T\left\{\delta[n-k]\right\}$$

Puede expresarse su salida como

$$y[n] = \sum_k x[k]h_k[n]$$

Y al ser también **invariante en el tiempo** se tiene que dados $h_k[n] = T\left\{\delta[n-k]\right\}$ y $h[n] = T\left\{\delta[n]\right\}$ se cumple que:

$$h_k[n] = h[n-k]$$

2.4.1 Suma de Convolución

Dadas las propiedades anteriores de un SLIT tenemos que su salida puede ser representada como la suma de convolución de su entrada con la respuesta al impulso del sistema, esto es:

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

Lo que nos permite modelar, por ejemplo, un canal de comunicaciones como un **SLIT**, utilizando su respuesta al impulso .

También se puede modelar un sistema estimador de series temporales como un **SLIT**, tema que es de interés del estudiante.

2.5 Propiedades de un SLIT

2.5.1 Conmutatividad

$$x[n] * h[n] = h[n] * x[n]$$

2.5.2 Linearidad

$$x[n] * (h_1[n] + h_2[n]) = x[n] * h_1[n] + x[n] * h_2[n]$$

2.6 Estabilidad BIBO de un SLIT

La estabilidad en sentido **BIBO** (BoundedInputBoundedOutput) es propia de un sistema que teniendo una entrada acotada, su salida también es acotada.

Es decir que si para una constante arbitraria B_x, $x[n]$ satisface:

$$|x[n]| < B_x \forall n$$

Es estable si se cumple:

$$|y[n]| \leq \sum_k |h[k]||x[n-k]| < B_x \sum_k |h[k]|$$

Se puede resumir diciendo que la estabilidad en el sentido BIBO de un SLIT se cumple si su respuesta al impulso es **absolutamente sumable** .

$$\sum_k |h[k]| < \infty$$

2.7 Causalidad de un SLIT

La salida de un SLIT causal depende sólo de las muestras de la secuencia de entrada en el instante n y de intstantes anteriores.

Por lo que observando la ecuación:

$$y[n] = \sum_k h[k]x[n-k]$$

Diremos que el sistema $h[n]$ es estable si se cumple que :

$$h[k] = 0 \forall k < 0$$

2.8 Invertibilidad de un SLIT

Diremos que un SLIT caracterizado por $h[n]$ es invertible si existe otro SLIT $h^{-1}[n]$ tal que:

$$h[n] * h^{-1}[n] = \delta[n]$$

2.9 Ejemplos de SLIT con respuesta finita al impulso FIR (Finite Impulse Response)

2.9.1 Retardo Ideal

$$h[n] = \delta[n - n_d] \forall n_d > 0$$

- Causal (Sólo desplazamientos a la derecha)
- Estable ya q su respuesta queda acotada para una entrada acotada

2.9.2 Moving Average (Promediador móvil)

$$h[n] = \frac{1}{M_1 + M_2 + 1} \sum_{k=-M_1}^{M_2} \delta[n-k]$$

- No-Causal ya que su salida depende de desplazamientos a la izquierda
- Sólo es causal si $M_1 \leq 0$
- Estable

2.9.3 Foward Difference

$$h[n] = \delta[n+1] - \delta[n]$$

- No-Causal
- Estable

2.9.4 Backward Difference

$$h[n] = \delta[n] - \delta[n-1]$$

- Causal
- Estable

2.10 Ejemplos de SLIT con respuesta infinita al impulso IIR (Infinite Impulse Response)

2.10.1 Acumulador y su sistema inverso

$$h[n] = \sum_{k=-\infty}^{n} \delta[k] = u[n] \quad \rightarrow \quad h^{-1}[n] = \delta[n] - \delta[n-1]$$

- Causal
- Inestable
- Invertible

2.10.2 Acumulador ponderado

$$h[n] = a^n u[n] |a| < 1$$

- Causal
- Estable

2.11 Suma de la serie geométrica:

Para los ejercicios que se realizarán en apartados siguientes es importante tener presente las siguientes expresiones:

$$\sum_{k=n_1}^{n_2} a^k = \frac{a^{n_1} - a^{n_2+1}}{1-a}$$

Si $|a| < 1$ tenemos el caso particular de la sumatoria de cero a infinito :

$$\sum_{k=0}^{\infty} a^k = \frac{1}{1-a}$$

2.12 Descripción de un SLIT mediante ecuaciones diferencia

Son muy importantes aquellos SLIT en los que la entrada y la salida están relacionadas por ecuaciones diferencia lineales a coeficientes constantes:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]$$

2.12.1 Soluciones a las ecuaciones diferencia

En forma general se cumple que las soluciones se pueden expresar como:

$$y[n] = y_p[n] + y_h[n]$$

- $y_p[n]$ es la solución particular dependiente de las condiciones iniciales
- $y_h[n]$ es la solución homogénea independiente de la secuencia de entrada

Si el sistema está inicialmente en reposo $y[n] = 0 n < n_0$, diremos que es lineal causal e invariante en el tiempo. En éste caso la solución para cada entrada es única.

2.12.2 Ejemplo: Representación recursiva de filtro acumulador

La salida de un filtro acumulador puede expresarse como:

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

Lo que tomando por separado el valor de $k = n$ nos permite escribir de forma equivalente:

$$y[n] = x[n] + \sum_{k=-\infty}^{n-1} x[k]$$

Desplazando a la derecha la primer expresión tenemos:

$$y[n-1] = \sum_{k=-\infty}^{n-1} x[k]$$

Con lo que finalmente llegamos a la expresión recursiva en ecuación de diferencia del filtro:

$$y[n] = x[n] + y[n-1]$$

$$y[n] - y[n-1] = x[n]$$

3 Fourier como lo presenta Prof. Alan V. Oppenheim en MIT

https://ocw.mit.edu/resources/res-6-007-signals-and-systems-spring-2011/video-lectures/

3.1 Serie de Fourier en Tiempo Continuo

3.2 Transformada de Fourier en Tiempo Continuo

3.3 Propiedades de la Transformada de Fourier en Tiempo Continuo

3.4 Serie de Fourier en Tiempo Discreto

3.5 Transformada de Fourier en Tiempo Discreto

3.6 Filtrado

4 Transformada de Fourier en Tiempo Discreto

En el apartado anterior vimos cómo es posible la representación de secuencias y la caracterización de sistemas como combinaciones lineales de impulsos desplazados y respuestas al impulso desplazadas. Dicha representación da lugar a la suma de convolución:

$$x[n] = \sum_k x[k]\delta[n-k] \quad \rightarrow \quad y[n] = \sum_k x[k]h[n-k] \tag{4}$$

En el siguiente apartado veremos cómo usando otras representaciones usando exponenciales complejas llegamos a importantes herramientas de análisis para SLIT

4.1 Autofunciones y Autovalores de un SLIT

En análisis matemático,cuando un operador ofrece como resultado ante una función la misma función multiplicada por un escalar decimos que esa función es una autofunción del operador y el escalar es el autovalor asociado a dicha autofunción. Así, las exponenciales complejas son autofunciones de todos los sistemas lineales e invariantes. Para cada valor distinto del parámetro obtenemos distintas autofunciones, cada una de ellas con su correspondiente autovalor asociado.

Consideramos un SLIT con respuesta al impulso $h[n]$ y en cuya entrada tenemos una exponencial compleja de la forma $x[n] = e^{j\Omega n}$

$$y[n] = x[n] * h[n] = \sum_k h[k]x[n-k]$$

$$y[n] = \sum_k h[k]e^{j\Omega(n-k)}$$

$$y[n] = \sum_k h[k]e^{j\Omega n}e^{-j\Omega k}$$

$$y[n] = e^{j\Omega n} \sum_k h[k] e^{-j\Omega k}$$

Si llamamos **respuesta en frecuencia** $H(e^{j\Omega})$ a :

$$H(e^{j\Omega}) = \sum_k h[k] e^{-j\Omega k}$$

Podemos expresar la salida como:

$$y[n] = e^{j\Omega n} H(e^{j\Omega})$$

Siendo

- $e^{j\Omega n}$ Auto-Función del SLIT
- $H(e^{j\Omega})$ Auto-Valor del SLIT (Respuesta en frecuencia : **Periódica en Ω con período** 2π)

4.2 Ejemplo: Respuesta en Frecuencia filtro promediador

Como vimos en el apartado anterior la respuesta al impulso de un filtro promediador es:

$$h[n] = \frac{1}{M_1 + M_2 + 1} \sum_{k=-M_1}^{M_2} \delta[n-k]$$

Siendo su respuesta en frecuencia:

$$\begin{aligned} H\left(e^{j\Omega}\right) = \sum_k h[k] e^{-j\Omega k} &= \frac{1}{M_1 + M_2 + 1} \sum_k e^{-j\Omega k} \sum_{q=-M_1}^{M_2} \delta[k-q] \\ &= \frac{1}{M_1 + M_2 + 1} \sum_{k=-M_1}^{M_2} e^{-j\Omega k} \end{aligned}$$

Ésta última expresión es la que usaremos para implementar el filtro promediador simbólicamente en el próximo apartado.

Operando sobre la expresión anterior nos queda (Aplicando la suma de la serie geométrica):

$$H\left(e^{j\Omega}\right) = \frac{1}{M_1 + M_2 + 1} \frac{\sin\left[\Omega\left(M_1 + M_2 + 1\right)/2\right]}{\sin(\Omega/2)} e^{-j\Omega(M_2 - M_1)/2}$$

4.3 Implementación simbólica del filtro promediador

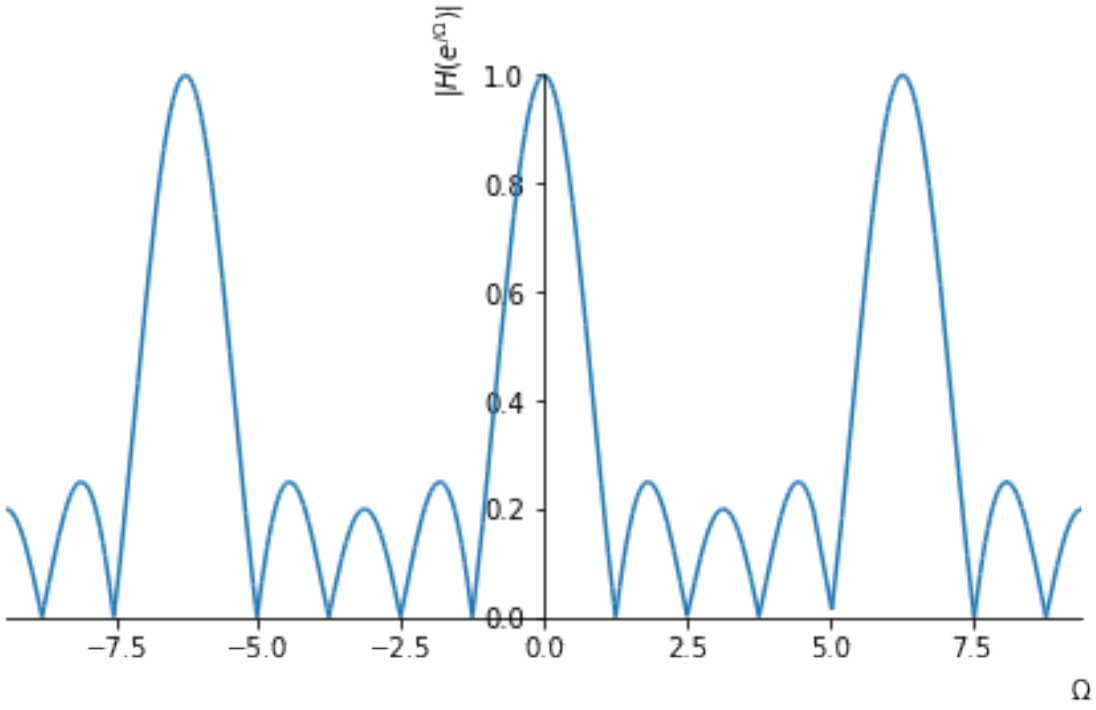
En éste apartado implementamos en PYTHON de forma simbólica el filtro promediador y lo particularizamos para valores de $M_1 = 0$ y $M_2 = 4$

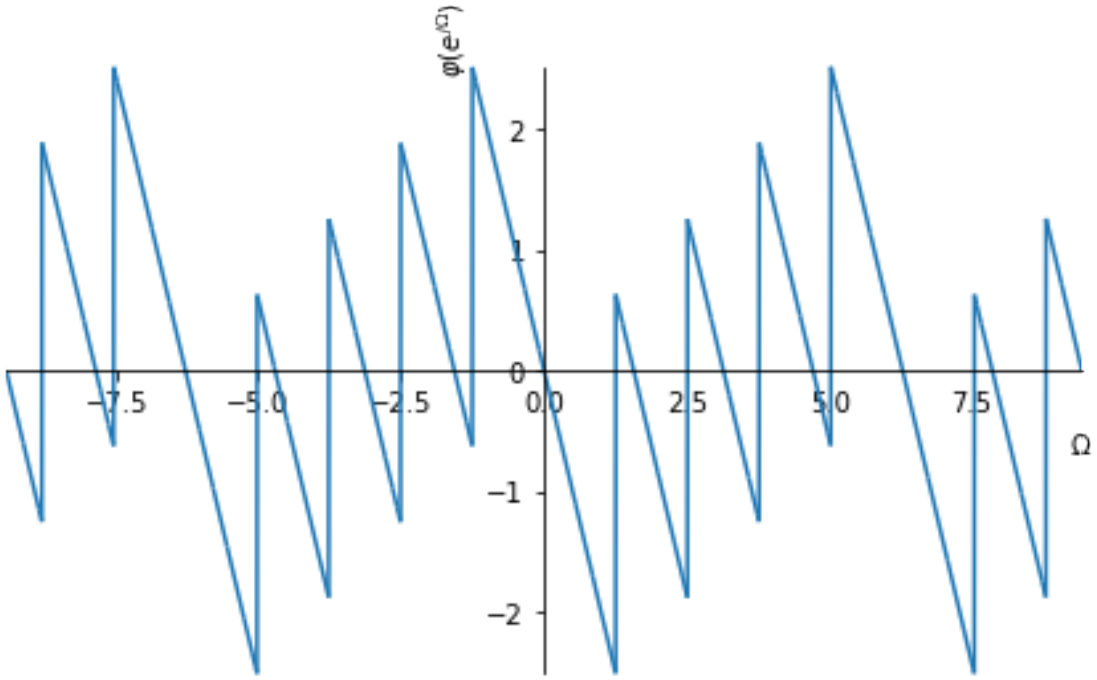
[15]:
```
%matplotlib inline
import sympy as sym
```

```
sym.init_printing()

W = sym.symbols('Omega', real=True)
k = sym.symbols('k', integer=True)
M_1 = 0
M_2 = 4
H = (1/(M_1+M_2+1))*sym.summation(sym.exp(-sym.I*W*k), (k, M_1, M_2))
H
```

[15]:
$$0.2 + 0.2e^{-i\Omega} + 0.2e^{-2i\Omega} + 0.2e^{-3i\Omega} + 0.2e^{-4i\Omega}$$

[16]:
```
sym.plot(sym.Abs(H), (W, -3*sym.pi, 3*sym.pi), xlabel=r'$\Omega$', ylabel=r'$|
↪H(e^{j \Omega}) |$')
sym.plot(sym.arg(H), (W, -3*sym.pi, 3*sym.pi), xlabel=r'$\Omega$',
↪ylabel=r'$\varphi(e^{j \Omega})$');
```





Notamos como el filtro promediador se comporta como un **filtro pasabajos** ya que atenúa las frecuencias cercanas a π y deja pasar a las frecuencias cercanas a 2π

4.4 Transitorios para secuencia-exponencial-causal como entrada a un SLIT

Si considereamos una entrada exponencial multiplicada por un escalón unitario $x[n] = e^{j\Omega n}u[n]$ como entrada a un SLIT tenemos a su salida:

$$y[n] = \sum_k h[k]x[n-k] = e^{j\Omega n}\sum_{k=0}^{n} h[k]e^{-j\Omega k}$$

Para $n \geq 0$ siendo $y[n] = 0$ para $n < 0$

Reescribiendo la expresión anterior tenemos:

$$\begin{aligned} y[n] &= e^{j\Omega n}\sum_{k=0}^{\infty} h[k]e^{-j\Omega k} - e^{j\Omega n}\sum_{k=n+1}^{\infty} h[k]e^{-j\Omega k} \\ &= e^{j\Omega n}H\left(e^{j\Omega n}\right) - e^{j\Omega n}\sum_{k=n+1}^{\infty} h[k]e^{-j\Omega k} \quad n \geq 0 \end{aligned}$$

$$y[n] = y_p[n] + y_t[n]$$

- $y_p[n] = e^{j\Omega n}H\left(e^{j\Omega n}\right)$ Es la respuesta en régimen permanente

- $y_t[n] = -e^{j\Omega n} \sum_{k=n+1}^{\infty} h[k] e^{-j\Omega k}$ Es la respuesta transitoria

Al existir ésta respuesta transitoria una exponencial ``causal'' no es autofunción de los SLIT.

4.4.1 Análisis de la respuesta transitoria

- Si $h[n]$ es de duración finita : $h[n] = 0$ para $0 \leq n \leq M$

Y tenmos en cuenta que se cumple:

$$|y_t[n]| = \left| \sum_{k=n+1}^{\infty} h[k] e^{-j\Omega k} \right| \leq \sum_{k=n+1}^{\infty} | h[k] |$$

se concluye que para $n + 1 > M$ la salida resulta :

$$y[n] = e^{j\Omega n} H\left(e^{j\Omega}\right) \quad n > M - 1$$

- Si la respuesta es de duración infinita y el SLIT es estable, se puede demostrar que $y_t[n]$ decae a cero cuando $n \to \infty$

5 El que mira entiende, el que estudia aprende, pero el que hace sabe (Ladislao Tsé)

Pregunta la transformada de Fourier de delta(n)-delta(n-1)

5.1 Ordenar

	$x[k]$	$X(e^{j\Omega}) = \mathcal{F}_*\{x[k]\}$
Linearity	$A\,x_1[k] + B\,x_2[k]$	$A\,X_1(e^{j\Omega}) + B\,X_2(e^{j\Omega})$
Real-valued signal	$x^*[k]$	$X^*(e^{-j\Omega})$
Convolution	$x[k] * h[k]$	$X(e^{j\Omega}) \cdot H(e^{j\Omega})$
Shift	$x[k - \kappa]$	$e^{-j\Omega\kappa} \cdot X(e^{j\Omega})$
Multiplication	$x[k] \cdot h[k]$	$\frac{1}{2\pi} X(e^{j\Omega}) \circledast_{2\pi} H(e^{j\Omega})$
Modulation	$e^{j\Omega_0 k} \cdot x[k]$	$X(e^{j(\Omega - \Omega_0)})$
Parseval's Theorem	$\sum_{k=-\infty}^{\infty} \lvert x[k] \rvert^2$	$\frac{1}{2\pi} \int_{-\pi}^{\pi} \lvert X(e^{j\Omega}) \rvert^2 \, d\Omega$

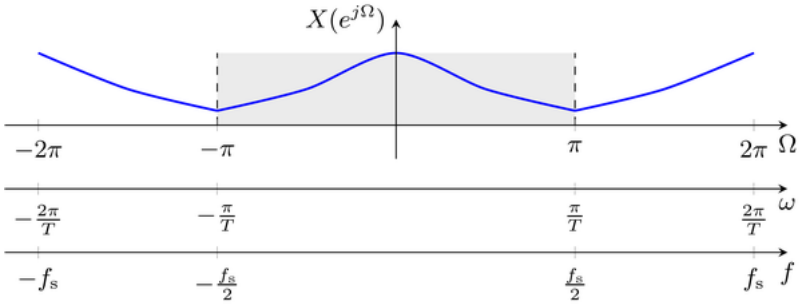
where $A, B \in \mathbb{C}$, $\Omega_0 \in \mathbb{R}$ and $\kappa \in \mathbb{Z}$.

$x[k]$	$X(e^{j\Omega}) = \mathcal{F}_*\{x[k]\}$
$\delta[k]$	1
1	$\text{⊥⊥⊥}\left(\frac{\Omega}{2\pi}\right)$
$\mu[k]$	$\frac{1}{1-e^{-j\Omega}} + \frac{1}{2}\text{⊥⊥⊥}\left(\frac{\Omega}{2\pi}\right)$
$\texttt{rect}_N[k]$	$e^{-j\Omega\frac{N-1}{2}} \cdot \frac{\sin\left(\frac{N\Omega}{2}\right)}{\sin\left(\frac{\Omega}{2}\right)}$
$e^{j\Omega_0 k}$	$\text{⊥⊥⊥}\left(\frac{\Omega - \Omega_0}{2\pi}\right)$

$x[k]$	$X(e^{j\Omega}) = \mathcal{F}_*\{x[k]\}$
$\sin(\Omega_0 t)$	$\frac{1}{2}\left[\text{⊥⊥⊥}\left(\frac{\Omega+\Omega_0}{2\pi}\right) + \text{⊥⊥⊥}\left(\frac{\Omega-\Omega_0}{2\pi}\right)\right]$
$\cos(\Omega_0 t)$	$\frac{j}{2}\left[\text{⊥⊥⊥}\left(\frac{\Omega+\Omega_0}{2\pi}\right) - \text{⊥⊥⊥}\left(\frac{\Omega-\Omega_0}{2\pi}\right)\right]$
$a^k \mu[k]$	$\frac{1}{1-ae^{-j\Omega}}$

where $\Omega_0 \in \mathbb{R}$ and $|a| < 1$. More transforms may be found in the literature or online.

6 Muestreo Transformadas de Fourier



[ ]: