CAPÍTULO I

Fundamentos de Señales Análogas y Señales Continuas

En este capítulo encontrarás una breve introducción a los conceptos de señales continuas, discretas y digitales. Adicionalmente, se explicarán las transformaciones que podemos hacerles a las señales en el dominio del tiempo, y las propiedades de los sistemas.

Al finalizar el capítulo, deberás estar en capacidad de:

1. Diferenciar señales análogas, continuas, discretas y digitales.
2. Escribir código en Python para dibujar señales continuas y discretas.
3. Aplicar transformaciones (desplazamiento, inversión, escalamiento) a señales continuas y discretas.
4. Escribir código en Python para realizar desplazamientos, inversión, y/o escalamiento a señales continuas y discretas.
5. Identificar si un sistema cumple con las propiedades de: memoria, causalidad, estabilidad, invarianza, linealidad e invertibilidad.

En este capítulo introductorio del curso, vamos a conocer las generalidades de las señales continuas y discretas, y las transformaciones de la variable independiente. Adicionalmente, identificaremos en qué casos un sistema continuo o discreto cumple con las siguientes propiedades: linealidad, invarianza, estabilidad, memoria, causalidad e invertibilidad.

1.1. SEÑALES CONTINUAS Y SEÑALES DISCRETAS Y SU VISUALIZACIÓN EN PYTHON

1.1.1. Señales continuas vs. señales discretas

Una señal de tiempo continuo es aquella cuya variable independiente puede tomar cualquier valor real dentro de un intervalo determinado. Matemáticamente, este tipo de señales se representan como:

$$x(t), \quad t \in R$$

donde t representa el tiempo continuo. En la naturaleza, la mayoría de las señales físicas son continuas, por ejemplo: la voz humana, una señal eléctrica analógica o la temperatura ambiente medida en función del tiempo. Desde un punto de vista teórico, una señal continua posee un número infinito de valores en el tiempo. Sin embargo, esta característica impide su almacenamiento y procesamiento directo en un computador.

Por otro lado, una señal de tiempo discreto es aquella definida únicamente para instantes de tiempo separados, usualmente equiespaciados. Se representa matemáticamente como:

$$x[n], \quad n \in \mathbb{Z}$$

donde n es un índice entero que identifica cada muestra de la señal. Este tipo de señales surge naturalmente cuando una señal continua es muestreada para su almacenamiento o procesamiento digital. A diferencia de las señales continuas, las señales discretas contienen un número finito o infinito de muestras, pero definidas en instantes discretos, lo que las hace adecuadas para ser manipuladas mediante algoritmos computacionales.

En este libro, utilizaremos la siguiente notación estándar:

Tipo de señal	Notación	Descripción
Tiempo continuo	$x(t)$	La variable independiente es el tiempo continuo, representado por la letra ***t***. Se utilizan paréntesis redondos.
Tiempo discreto	$x[n]$	La variable independiente es el tiempo discreto, representado por la letra ***n***. Se utilizan paréntesis cuadrados.

1.1.2. Visualización de señales

Aunque el análisis matemático distingue claramente entre señales continuas y discretas, en la práctica todas las señales representadas en un computador son discretas, ya que solo se pueden almacenar y procesar un número finito de muestras.

Nota importante

Cuando se utiliza Python para graficar señales, la diferencia entre señal continua y discreta no depende únicamente de la gráfica, sino del modelo matemático asumido. La instrucción *plt.plot* interpola visualmente entre muestras, generando una apariencia continua, mientras que plt.stem representa explícitamente cada muestra discreta.

Ejemplo 1: señal senoidal

Consideremos una señal senoidal de frecuencia $f = 10\ Hz$

y duración de $0.1\ s$. A continuación, se muestra su generación y visualización en Python.

```python
import numpy as np
import matplotlib.pyplot as plt

f = 10 # frecuencia [Hz]
fs = 1000 # frecuencia de muestreo [Hz]
t = np.linspace(0, 0.1, int(0.1*fs))

x = np.sin(2*np.pi*f*t)

plt.plot(t, x)
plt.xlabel("Tiempo [s]")
plt.ylabel("Amplitud")
plt.title("Señal senoidal visualizada como continua")
plt.show()
```

Figura 1. Código en Python del Ejemplo 1 para crear y visualizar una señal senoidal.

Y obtenemos la siguiente figura:

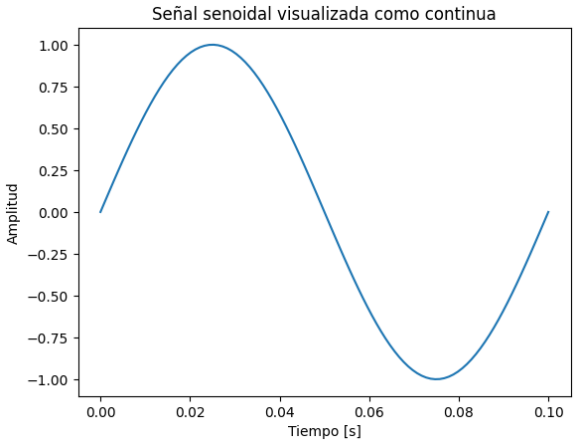


Figura 2. Gráfica señal senoidal del Ejemplo 1, visualizada como continua.

Aunque la señal se muestra como continua, en realidad está compuesta por un conjunto finito de muestras. Si se desea enfatizar su naturaleza discreta, se utiliza la función stem:

```
plt.stem(t, x)
plt.xlabel("Tiempo [s]")
plt.ylabel("Amplitud")
plt.title("Señal senoidal muestreada")
plt.show()
```

Figura 3. Código en Python del Ejemplo 1 para visualizar una señal como discreta.

Obteniendo la siguiente gráfica:

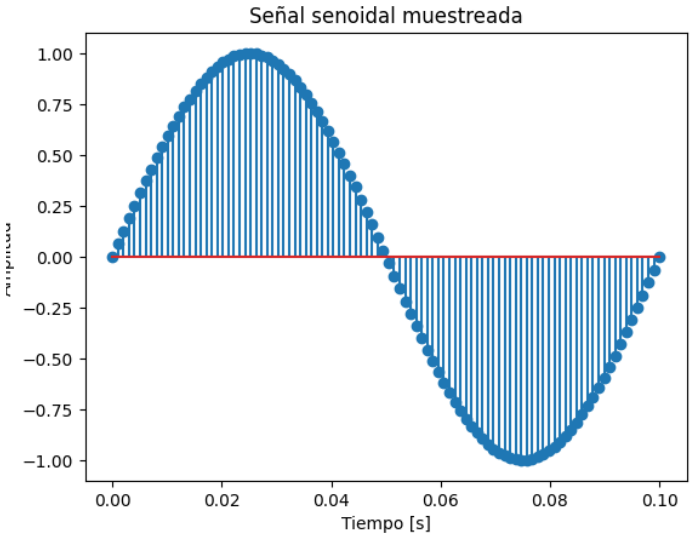


Figura 4. Gráfica señal senoidal del Ejemplo 1, visualizada como discreta.

Ejemplo 2: señales discretas con índice entero

Para representar una señal estrictamente discreta, el eje horizontal debe expresarse mediante un índice entero n. Por ejemplo, una señal senoidal discreta con m muestras y n_c ciclos se puede escribir como:

```
m = 10 # número de muestras
nc = 1 # número de ciclos
n = np.arange(m)

x = np.sin(2*np.pi*nc*n/m)

plt.stem(n, x)
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Señal senoidal discreta x[n]")
plt.show()
```

Figura 5. Código en Python del Ejemplo 2 para visualizar señal senoidal discreta con índice entero.

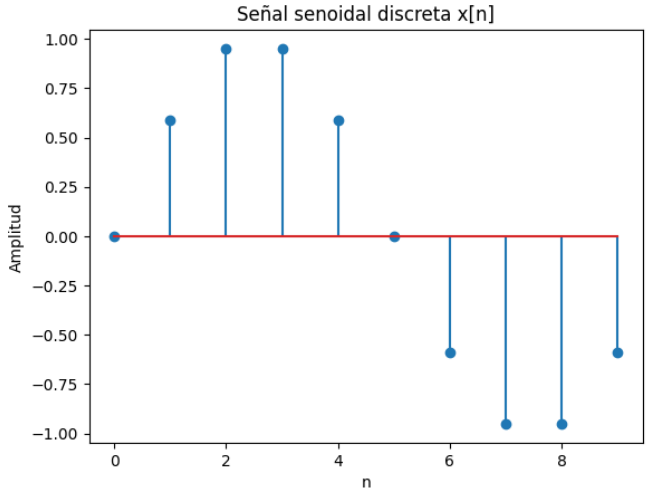


Figura 6. Gráfica señal senoidal del Ejemplo 2, visualizada como discreta con índice entero.

En este caso, la señal está definida únicamente para los valores enteros de n, lo que refleja correctamente el concepto de señal de tiempo discreto.

Al comprender esta distinción entre señales continuas y discretas, se establecen las bases necesarias para el estudio de las transformaciones en el dominio del tiempo y el análisis de sistemas, temas que abordaremos en las siguientes secciones.

1.2. TRANSFORMACIÓN DE LA VARIABLE INDEPENDIENTE

En el análisis de señales, es común modificar la variable independiente (tiempo continuo o índice discreto) con el fin de analizar cómo cambia la forma de una señal. Estas modificaciones reciben el nombre de transformaciones en el dominio del tiempo.

En esta sección estudiaremos tres transformaciones:

- Desplazamiento temporal
- Inversión temporal
- Escalamiento temporal

Cada transformación se analizará tanto para señales de tiempo continuo como de tiempo discreto, resaltando sus similitudes y diferencias.

1.2.1. Desplazamiento de señales continuas y discretas

El desplazamiento temporal consiste en adelantar o atrasar una señal respecto al eje del tiempo. Matemáticamente, esta transformación se expresa como:

	Señal original	Señal desplazada
Tiempo continuo	$x(t)$	$x(t - t_o)$
Tiempo discreto	$x[n]$	$x[n - k]$

En señales en tiempo continuo, el parámetro t_0 es un número real (positivo o negativo). En señales discretas, el desplazamiento k es un número entero.

- Si el valor de desplazamiento es positivo, la señal se desplaza hacia la derecha.
- Si el valor de desplazamiento es negativo, la señal se desplaza hacia la izquierda.

Ejemplo 3: desplazamiento en tiempo continuo

Consideremos una señal con $f = 1\,Hz$, definida en el intervalo $t \in [0,\ 1]s$. Para su representación gráfica se utiliza una frecuencia de

muestreo $f_s = 50\ Hz$. En este caso, al utilizar $t_0 = 0.5\ s$, la señal se desplaza hacia la derecha, quedando en el intervalo $t\ \epsilon\ [0.5,\ 1.5]s$.

Si se utilizara un valor negativo de t_0, el desplazamiento ocurriría hacia la izquierda.

```
import numpy as np
import matplotlib.pyplot as plt

f = 1      # frecuencia [Hz]
fs = 50    # frecuencia de muestreo [Hz]
ti = 0
tf = 1
t0 = 0.5   # desplazamiento temporal [s] (retardo)

# Tiempo de la señal original (duración finita)
t = np.linspace(ti, tf, int((tf - ti)*fs), endpoint=False)
x = np.sin(2*np.pi*f*t)

# Tiempo de la señal desplazada (misma forma, pero empieza después)
t_shift = t + t0
x_shift = np.sin(2*np.pi*f*(t_shift - t0))  # equivalente a sin(2π f t)

plt.plot(t, x, label="x(t), soporte [0,1]")
plt.plot(t_shift, x_shift, label="x(t - t₀), soporte [0.5,1.5]")

plt.xlabel("Tiempo [s]")
plt.ylabel("Amplitud")
plt.title("Señal de duración finita y su desplazamiento temporal")
plt.legend()
plt.grid(True)
plt.show()
```

Figura 7. Gráfica señal senoidal del Ejemplo 3.

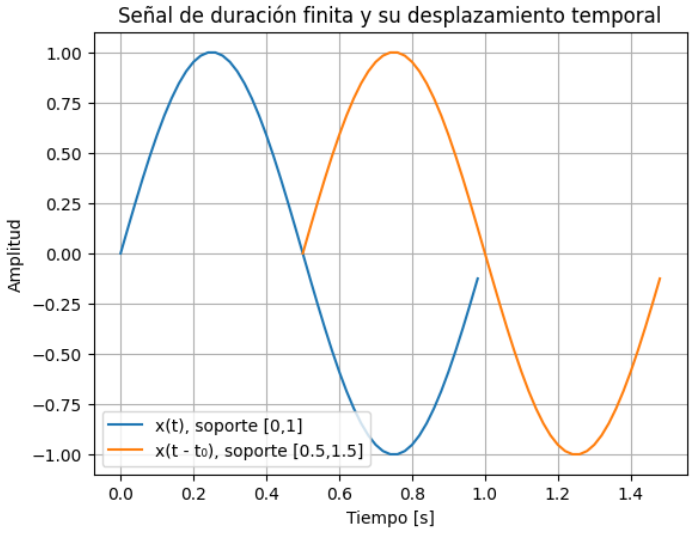


Figura 8. Gráficas del Ejemplo 3: señal original (azul) y señal desplazada (naranja).

Ejemplo 4: desplazamiento en tiempo discreto

Para una señal discreta con $m = 40$ muestras y $n_c = 2$ ciclos, el desplazamiento se realiza modificando el índice n, a partir del valor de k. Este desplazamiento no altera las amplitudes de la señal, únicamente su ubicación temporal.

```python
import numpy as np
import matplotlib.pyplot as plt

# Parámetros de la señal
m = 40
nc = 2
k = 5

# Eje temporal discreto
n = np.arange(m)

# Señal original
x = np.sin(2 * np.pi * nc * n / m)

# Señal desplazada: x[n-k]
n_shift = n + k
x_shift = x

# Gráficas (cada plt.stem en una sola línea)
plt.stem(n, x, linefmt='C0-', markerfmt='C0o', basefmt=" ", label='x[n], soporte [0, 39]')
plt.stem(n_shift, x_shift, linefmt='C1-', markerfmt='C1o', basefmt=" ", label='x[n-k], soporte [5, 44]')

# Formato
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Desplazamiento temporal de una señal discreta (soporte finito)")
plt.legend()
plt.grid(True)
plt.show()
```

Figura 9. Código en Python del Ejemplo 4.

Obteniendo la siguiente gráfica, en la cual la señal original definida entre $n = 0$ y $n = 39$ se desplaza 5 posiciones hacia la derecha, quedando definida entre $n = 5$ y $n = 44$.

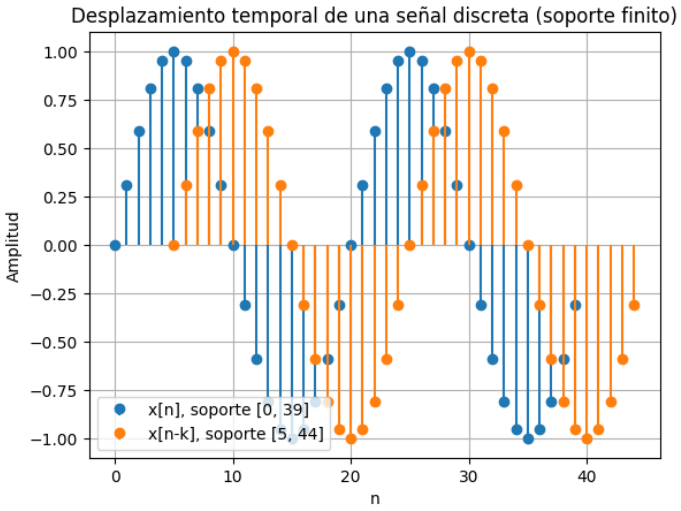


Figura 10. Gráficas del Ejemplo 4: señal original (azul) y señal desplazada (naranja).

1.2.2. Inversión temporal en señales de tiempo continuo y discreto

La inversión temporal produce un efecto de espejo de la señal con respecto al origen del eje temporal y se define matemáticamente como:

	Señal original	Señal invertida
Tiempo continuo	$x(t)$	$x(-t)$
Tiempo discreto	$x[n]$	$x[-n]$

Donde el origen temporal se fija en $t = 0$, ó $n = 0$. Esta transformación intercambia la parte de la señal ubicada a la derecha del origen temporal con la que se encuentra a la izquierda, manteniendo fija la muestra definida en el origen.

Ejemplo 5. Inversión temporal de señales de soporte finito en tiempo continuo

En este ejemplo se ilustra el efecto de la inversión temporal sobre una señal de soporte finito, en tiempo continuo. Se compara la señal original con su versión invertida, resaltando el cambio en el soporte temporal y el efecto de espejo con respecto al origen del eje del tiempo.

Supongamos que tenemos una señal con $f = 1\,Hz$, definida en el intervalo $t \ \epsilon \ [0, \ 1]s$. Para su representación gráfica se utiliza una frecuencia de muestreo $f_s = 200\ Hz$. Al aplicar la inversión temporal, la señal resultante queda definida en el intervalo $t \ \epsilon \ [-1, \ 0]s$, como un reflejo de la señal original respecto al origen temporal. Se observa que el valor de la señal en $t = 0$ se mantiene invariante en ambas representaciones.

```python
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
f = 1       # frecuencia de la señal [Hz]
fs = 200    # frecuencia de muestreo para visualización [Hz]
ti = 0
tf = 1

# Señal original (soporte finito en [ti, tf))
t = np.linspace(ti, tf, int((tf - ti) * fs), endpoint=False)
x = np.sin(2 * np.pi * f * t)

# Señal invertida temporalmente
# La inversión x(-t) se implementa invirtiendo el eje temporal
# y reordenando las muestras para mantener un eje creciente
t_inv = -t[::-1]     # eje temporal invertido
x_inv = x[::-1]      # muestras invertidas

# Gráficas
plt.plot(t, x, label="x(t), soporte [0, 1]")
plt.plot(t_inv, x_inv, label="x(-t), soporte [-1, 0]")

plt.xlabel("Tiempo [s]")
plt.ylabel("Amplitud")
plt.title("Inversión temporal en tiempo continuo: x(t) vs x(-t)")
plt.legend()
plt.grid(True)
plt.show()
```

Figura 11. Código en Python del Ejemplo 5: inversión temporal en tiempo continuo.

A partir de la señal original $x(t)$, se implementa la inversión temporal construyendo la señal $x(-t)$. Esta operación se realiza invirtiendo el eje temporal y reordenando las muestras del arreglo. Adicionalmente, las amplitudes también se recorren en orden inverso del arreglo.

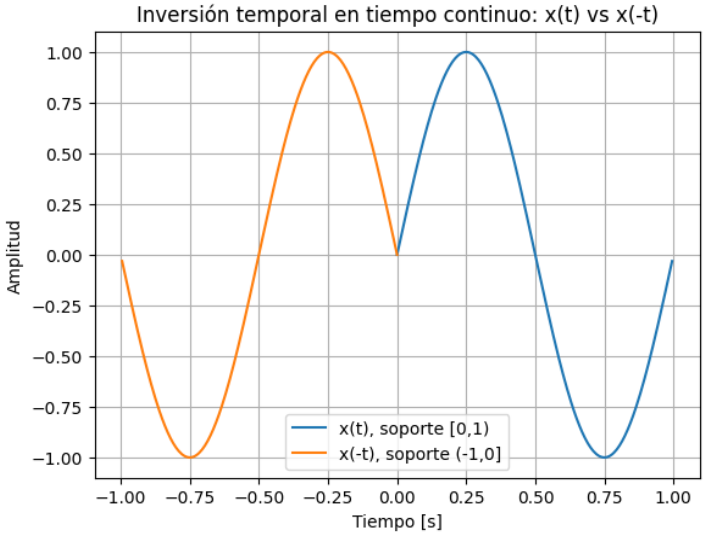


Figura 12. Representación gráfica del Ejemplo 5: señal original x(t) (azul) y señal invertida x(-t) (naranja).

Ejemplo 6. Inversión temporal de señales de soporte finito en tiempo continuo y discreto

Ahora, supongamos que tenemos una señal $x[n]$ de soporte finito definida para $n \in [0 \quad 19]$, es decir, una señal discreta con 20 muestras que inicia en el origen. Al aplicar la inversión temporal, se obtiene una señal $x[-n]$, cuyo soporte queda definido para $n \in [-19 \quad 0]$, lo cual implica que el orden temporal de las muestras se invierte.

En este caso, las amplitudes de la señal se asignan en orden inverso con respecto a la señal original. De manera análoga, las posiciones temporales discretas se invierten previamente y se reflejan con respecto al

origen, garantizando la correcta correspondencia entre cada muestra y su nuevo instante temporal.

```python
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
m = 20      # número de muestras
nc = 1      # número de ciclos en la ventana de m muestras

# Señal original (tiempo discreto) con soporte finito en [0, m-1]
n = np.arange(m)
x = np.sin(2 * np.pi * nc * n / m)

# Inversión temporal: x[-n]
# Para graficar de izquierda a derecha, invertimos el eje y reordenamos las muestras
n_inv = -n[::-1]   # soporte [- (m-1), 0]
x_inv = x[::-1]    # mismas muestras, en orden inverso

# Gráfica
plt.stem(n, x, linefmt='C0-', markerfmt='C0o', basefmt=" ",
         label=f"x[n], soporte [0, {m-1}]")
plt.stem(n_inv, x_inv, linefmt='C1-', markerfmt='C1o', basefmt=" ",
         label=f"x[-n], soporte [-{m-1}, 0]")

plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Inversión temporal en tiempo discreto: x[n] vs x[-n]")
plt.legend()
plt.grid(True)
plt.show()
```

Figura 13. Código en Python del Ejemplo 6: inversión temporal en tiempo discreto.

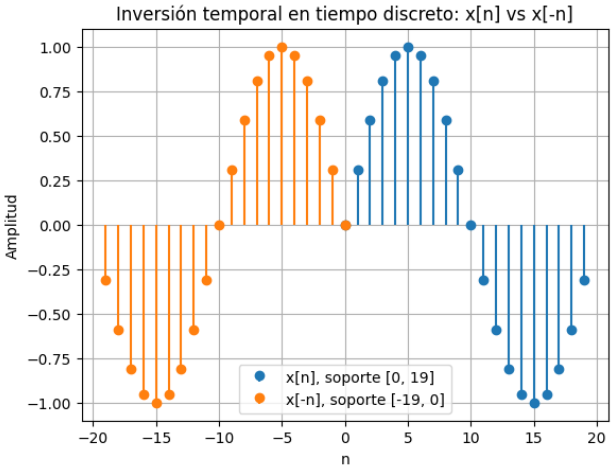


Figura 14. Representación gráfica del Ejemplo 6: señal original x[n] (azul) y señal invertida x[-n] (naranja).

1.2.3. Escalamiento en el dominio del tiempo continuo

La tercera transformación aplicada a la variable independiente corresponde al escalamiento temporal. A diferencia de las transformaciones de desplazamiento e inversión, el escalamiento presenta diferencias fundamentales cuando se aplica a señales de tiempo continuo y de tiempo discreto.

En el caso del tiempo continuo, el escalamiento modifica la duración temporal de la señal, produciendo un efecto de compresión o dilatación del eje temporal, según el valor del factor de escalamiento α:

Señal original	Compresión de la señal (menor duración)	Dilatación de la señal (mayor duración)
$x(t)$	$x(\alpha t)$ para α>1	$x(\alpha t)$ para 0<α<1

Ejemplo 7. Compresión de una señal en tiempo continuo

Supongamos que tenemos una señal de frecuencia $f = 1\,Hz$, definida en el intervalo temporal $t \in [1, 2]s$, es decir, con una duración de un segundo.

Al aplicar un escalamiento temporal con $\alpha > 1$, la señal se comprime, reduciendo su duración.

El siguiente código ilustra el efecto de compresión, para diferentes valores de α.

```python
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
f = 1      # frecuencia de la señal [Hz]
fs = 50    # frecuencia de muestreo para visualización [Hz]
ti = 1     # tiempo inicial [s]
tf = 2     # tiempo final [s]

# Vector temporal original (α = 1)
t = np.linspace(ti, tf - 1/fs, int((tf - ti) * fs))
x = np.sin(2 * np.pi * f * t)

# Compresión temporal
alpha2 = 2
alpha4 = 4

# Nuevos ejes temporales creados a partir de t
t2 = t / alpha2
t4 = t / alpha4

# Señales comprimidas
x2 = np.sin(2 * np.pi * f * (alpha2 * t2))
x4 = np.sin(2 * np.pi * f * (alpha4 * t4))

# Gráfica
plt.plot(t,  x,  label='x(t), α = 1', linewidth=2)
plt.plot(t2, x2, label='x(αt), α = 2', linewidth=2)
plt.plot(t4, x4, label='x(αt), α = 4', linewidth=2)

plt.xlabel("Tiempo [s]")
plt.ylabel("Amplitud")
plt.title("Compresión temporal en tiempo continuo")
plt.legend()
plt.grid(True)
plt.show()
```

Figura 15. Código en Python del Ejemplo 7: compresión temporal en tiempo continuo, para α > 1.

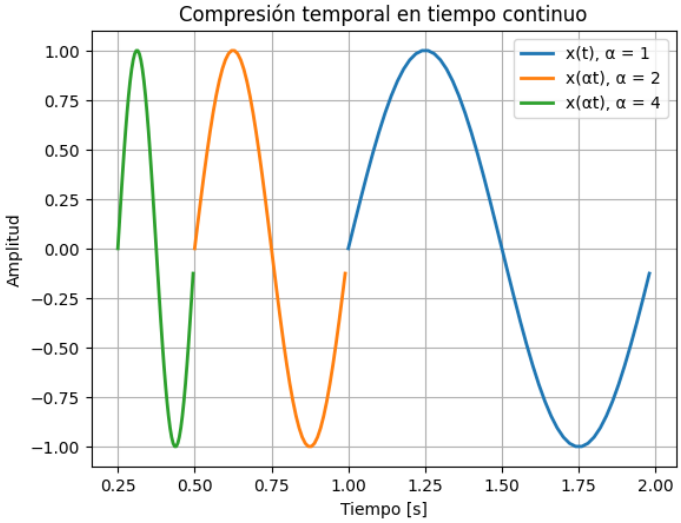


Figura 16. Representación gráfica del Ejemplo 7: señal original x(t) (azul), señal comprimida con α=2 (naranja) y α=4 (verde).

Al comprimir la señal se pueden apreciar dos efectos:

- La duración de la señal comprimida es igual a la duración de la señal inicial dividida en α.
- El tiempo inicial de la señal comprimida es igual al tiempo inicial de la señal inicial, dividido en α.
- El tiempo final de la señal comprimida es igual al tiempo final de la señal inicial, dividido en α.

Ejemplo 8. Dilatación de una señal en tiempo continuo

Vamos a utilizar los mismos datos del ejemplo 7, pero ahora, en este caso $\mathbf{0} < \boldsymbol{\alpha} < \mathbf{1}$.

```python
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
f = 1      # frecuencia de la señal [Hz]
fs = 50    # frecuencia de muestreo para visualización [Hz]
ti = 1     # tiempo inicial [s]
tf = 2     # tiempo final [s]

# Vector temporal original (α = 1)
t = np.linspace(ti, tf - 1/fs, int((tf - ti) * fs))
x = np.sin(2 * np.pi * f * t)

# Dilatación temporal
alpha2 = 0.5
alpha4 = 0.25

# Nuevos ejes temporales creados a partir de t
t2 = t / alpha2
t4 = t / alpha4

# Señales dilatadas
x2 = np.sin(2 * np.pi * f * (alpha2 * t2))
x4 = np.sin(2 * np.pi * f * (alpha4 * t4))

# Gráfica
plt.plot(t,  x,  label='x(t), α = 1', linewidth=2)
plt.plot(t2, x2, label='x(αt), α = 0.5', linewidth=2)
plt.plot(t4, x4, label='x(αt), α = 0.25', linewidth=2)

plt.xlabel("Tiempo [s]")
plt.ylabel("Amplitud")
plt.title("Compresión temporal en tiempo continuo")
plt.legend()
plt.grid(True)
plt.show()
```

Figura 17. Código en Python del Ejemplo 8: dilatación en tiempo continuo, para $0<\alpha<1$.

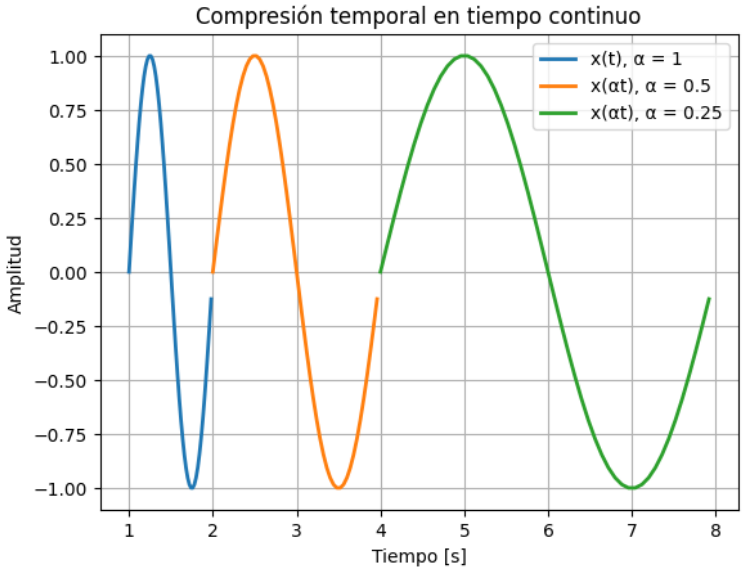


Figura 18. Representación gráfica del Ejemplo 8: señal original x(t) (azul), señal dilatada con α=0.5 (naranja) y α=0.25 (verde).

En este caso, a medida que α se va haciendo más pequeño, la duración de la señal se va haciendo mayor. El vector de tiempos se divide entre α.

1.2.4. Escalamiento en el dominio del tiempo discreto

A diferencia del caso continuo, estas operaciones (compresión/ dilatación) no son equivalentes exactos, sino aproximaciones prácticas para su implementación digital.

En particular, cuando se realiza una compresión temporal, se descartan muestras de la señal original que no pueden recuperarse posteriormente. Por el contrario, al efectuar una dilatación temporal, se introducen muestras adicionales con valor cero en posiciones específicas del eje temporal.

Ejemplo 9: Compresión señal discreta

Supongamos que tenemos un ciclo de una señal senoidal con 40 muestras, empezando en la muestra $n = 0$ y terminando en $n = 39$. Esta señal se comprime con un $\alpha = 3$, como se presenta con el siguiente código:

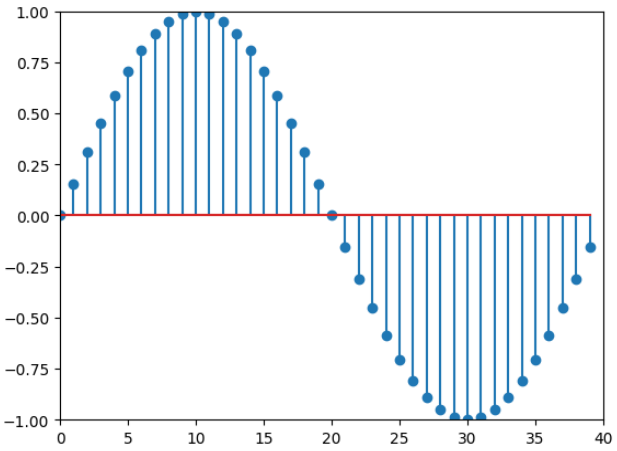
```
import numpy as np
import matplotlib.pyplot as plt

m = 40
nc = 1
α = 3  # prueba con 2, 3, 4...

n = np.arange(m)  # entero
x = np.sin(2 * np.pi * nc * n / m)

x_new = x[::α]
n_new = np.arange(len(x_new))  # 0..len(x_new)-1

plt.stem(n, x)
plt.axis([n.min(), n.max()+1, x.min(), x.max()])
plt.show()

plt.stem(n_new, x_new)
plt.axis([n_new.min(), n_new.max()+1, x_new.min(), x_new.max()])
plt.xticks(n_new)  # ticks enteros
plt.show()
```

Figura 19. Código en Python del Ejemplo 9: compresión en tiempo discreto, para α=3.

Las amplitudes las tomamos cada α posiciones, empezando desde la posición n=0. Así, para un vector de 40 posiciones que empieza en 0 y termina en 39, tendremos 14 muestras, empezando en 0 y terminando en 13 (que se obtiene de dividir 39 entre 3).

Obtenemos las siguientes gráficas:

A)



B)

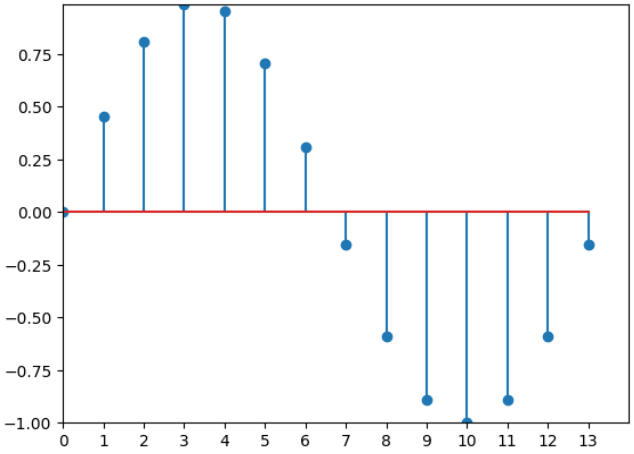


Figura 20. Representación gráfica del Ejemplo 9: a) señal x[n], b) señal x[3n].

Ejemplo 10: Dilatación señal discreta, caso α=1/q.

Vamos ahora a realizar un ejemplo para el caso discreto, cuando hacemos una transformación de dilatación. En este caso $\alpha = 1/q$, donde $q \in Z$.

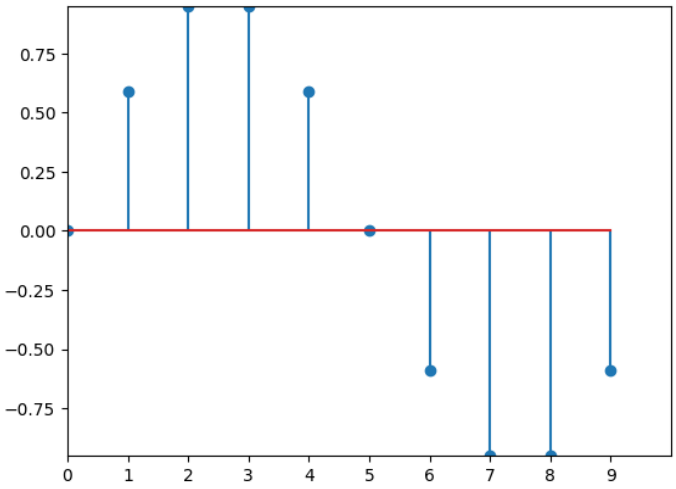
```
import numpy as np
import matplotlib.pyplot as plt

m = 10      # número de muestras
nc = 1      # número de ciclos

q = 3       #  q es entero
α = 1/ q

# Señal original (tiempo discreto entero)
n = np.arange(m)
x = np.sin(2 * np.pi * nc * n / m)

# Dilatación
L = int(round(1 / α))       # factor entero
n_new = np.arange(m * L)    # eje discreto entero
x_new = np.zeros(m * L)
x_new[::L] = x              # insertar ceros

# Señal original
plt.stem(n, x)
plt.axis([n.min(), n.max()+1, x.min(), x.max()])
plt.xticks(n)
plt.show()

# Señal dilatada
plt.stem(n_new, x_new)
plt.axis([n_new.min(), n_new.max()+1, x_new.min(), x_new.max()])
plt.xticks(n_new[::L])      # 👈 ticks cada L
plt.show()
```

Figura 21. Código en Python del Ejemplo 10: dilatación en tiempo discreto, para α=1/3.

En este código es importante utilizar el entero del redondeo de 1/ α, para conocer las posiciones nuevas de la señal dilatada, y de esta manera se insertan ceros por medio de x_new[::L] = x.

Y se obtiene:

A



B

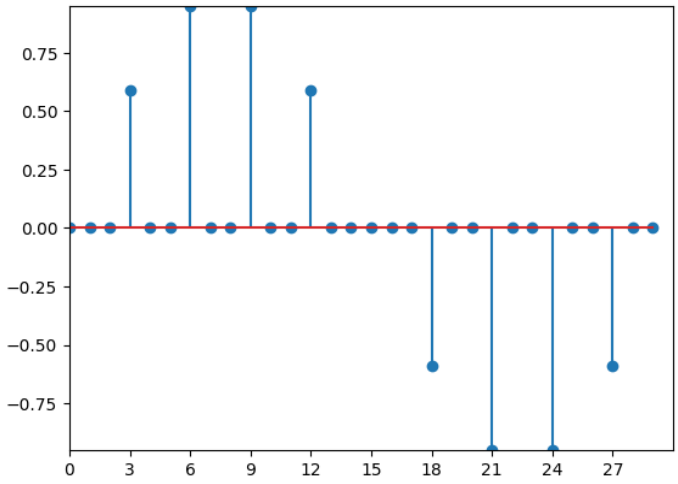


Figura 22. Representación gráfica del Ejemplo 10: a) señal x[n], b) señal x[n/3].

La señal x[n] está comprendida en el intervalo $[0,\ 9]$ (último valor no nulo en n=9); mientras que, la señal x[n/3] está comprendida en el intervalo $[0,\ 29]$, pero con valores no nulos hasta n=27.

Ejemplo 11: Dilatación señal discreta, caso α=p/q.

Vamos ahora a analizar un caso “especial” de dilatación, que ocurre cuando el valor de **p** no es igual a 1. En esta situación, no solo se insertan ceros dentro de la señal dilatada, sino que, adicionalmente, algunas muestras de la señal original se pierden. Por lo tanto, este es un claro ejemplo de que el escalamiento temporal en tiempo discreto no es una operación invertible en general: la compresión nunca es reversible y la dilatación solo lo es cuando α = 1/ q, para q entero.

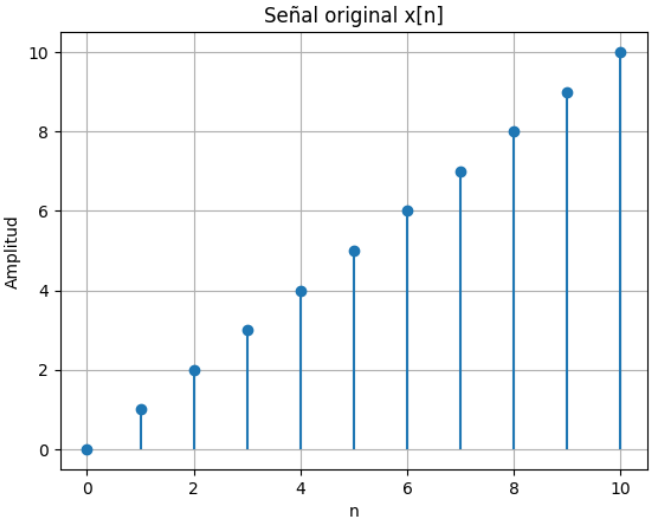
Trabajaremos con el siguiente código:

```python
import numpy as np
import matplotlib.pyplot as plt

# Señal original
n = np.arange(0, 11)      # 0..10
x = np.arange(len(n))     # ejemplo: x[n] = n

# Factor de expansión
alpha = 2/3
ratio = 1/alpha           # 3/2

# Mapeo de índices
n_targets = ratio * n
n_int_mask = np.isclose(n_targets, np.round(n_targets))
n_targets_int = np.round(n_targets[n_int_mask]).astype(int)

# Nueva señal expandida
N_new = n_targets_int.max() + 1
x_new = np.zeros(N_new)
x_new[n_targets_int] = x[n_int_mask]

# Gráfica señal original
plt.figure()
plt.stem(n, x, basefmt=" ")
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Señal original x[n]")
plt.grid(True)
plt.show()

# Gráfica señal expandida
plt.figure()
plt.stem(np.arange(N_new), x_new, basefmt=" ")
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Señal expandida x[n/α],  α = 2/3")
plt.grid(True)
plt.show()
```

Figura 23. Código en Python del Ejemplo 11: dilatación en tiempo discreto, para α=2/3.

Y obtenemos la señal de salida:

A



B

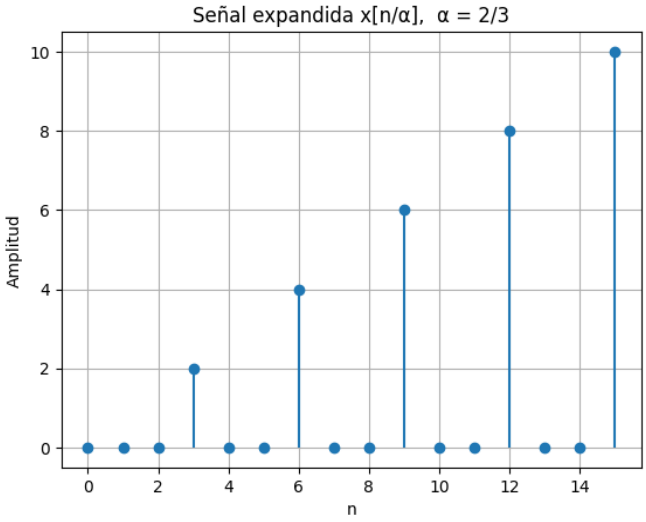


Figura 24. Representación gráfica del Ejemplo 11: a) señal x[n], b) señal x[n/3].

¿Cuáles son los datos que se conservan después de la dilatación? Si al realizar n/(α) el resultado es entero, entonces se conserva; en caso contrario, se pierde. Por ejemplo, el dato ubicado en n=1 se pierde, dado que, al realizar 1/(2/3), el resultado es 1.5; mientras que, el dado ubicado en n=2 se conserva, porque al realizar 2/(2/3) obtenemos como resultado 3.

1.2.5. Combinación de transformaciones

En esta última sección de transformaciones, se aborda la aplicación simultánea de dos o más transformaciones sobre la variable independiente de una señal. Este tipo de operaciones es frecuente en el análisis de señales y, aunque existen distintos procedimientos para realizarlas, a continuación, se propone un método sistemático que facilita su correcta interpretación y evita errores comunes.

El procedimiento recomendado es el siguiente:

1. Factorizar la variable independiente, de manera que el coeficiente que acompaña a la variable tenga magnitud uno y signo positivo. Esto permite identificar de forma clara el tipo de transformaciones involucradas.
2. Aplicar las transformaciones siguiendo el orden de los paréntesis, de afuera hacia adentro. En la práctica, esto implica realizar primero la inversión temporal o el escalamiento (dilatación o compresión), y aplicar el desplazamiento siempre como la última transformación.

Este método garantiza una lectura correcta de la señal transformada y resulta especialmente útil cuando se combinan inversión, escalamiento y desplazamiento en una misma expresión.

Ejemplo 12: Combinación de transformaciones en señal discreta

Vamos a ilustrar el procedimiento mediante un ejemplo. Supongamos que tenemos la siguiente señal discreta x[n]:

```python
import numpy as np
import matplotlib.pyplot as plt

m = 30      # número de muestras de la señal
nc = 2      # número de ciclos de la señal
n = np.arange(0, m)              # eje temporal discreto
x = np.sin(2 * np.pi * nc * n/m) # señal discreta

plt.stem(n, x)
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Señal x[n]")
plt.grid(True)
```

Figura 25. Código en Python del Ejemplo 12: generación de la señal discreta.

La señal obtenida se presenta a continuación:

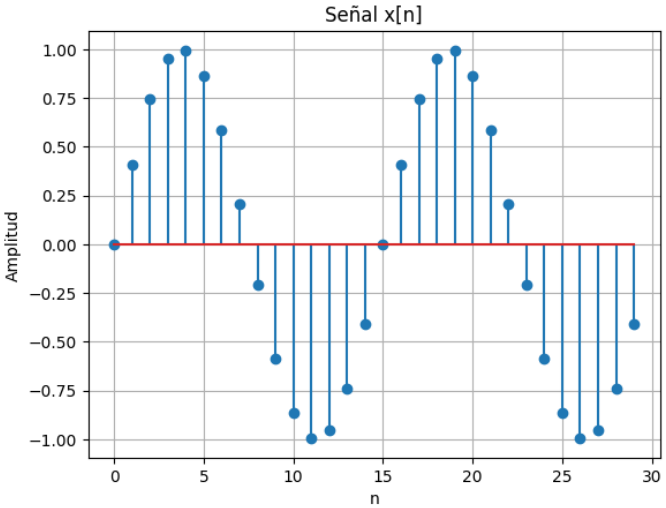


Figura 26. Representación gráfica del Ejemplo 12: señal x[n].

Esta señal es de soporte finito y está comprendida en el intervalo $[0, \ 29]$.

A esta señal se le aplicará la siguiente transformación:

$$x[-2n+8]$$

Siguiendo el procedimiento descrito anteriormente, se realizan los siguientes pasos.

Paso 1: Factorización de la variable independiente

$$x[-2n+8] = x[-2(n-4)]$$

Esta forma permite identificar claramente las transformaciones involucradas.

Paso 2: Escalamiento temporal (compresión, α=2)

Se realiza primero la compresión temporal, tomando una de cada α muestras de la señal original:

```
alpha = 2

n_new = np.arange(0, int(np.ceil(m/alpha)))
x_new = x[::alpha]

plt.stem(n_new, x_new)
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Señal comprimida (α = 2)")
plt.grid(True)
```

Figura 27. Código en Python del Ejemplo 12: compresión, α=2.

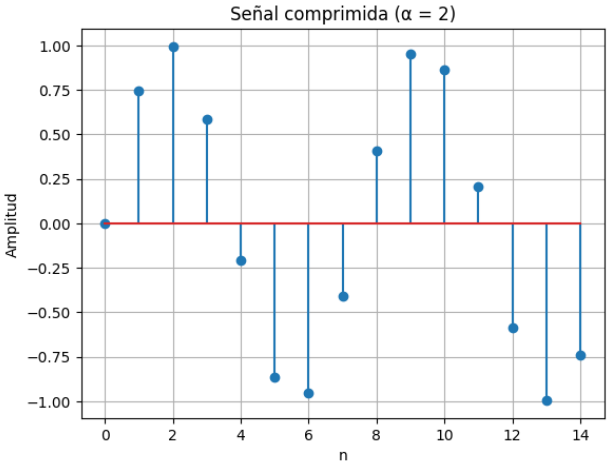


Figura 28. Representación gráfica del Ejemplo 12: compresión, α=2.

En este primer resultado es importante resaltar que la duración de la señal cambió a la mitad, es decir, por un factor de 1/α. De las 30 muestras iniciales, ahora quedaron 15, ubicadas en el intervalo $[0,\ \ 14]$.

Paso 3: Inversión temporal

A continuación, se realiza la inversión temporal reflejando el eje discreto con respecto al origen:

```
n_inv = -n_new

plt.stem(n_inv, x_new)
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Señal invertida")
plt.grid(True)
```

Figura 29. Código en Python del Ejemplo 12: inversión de la señal.

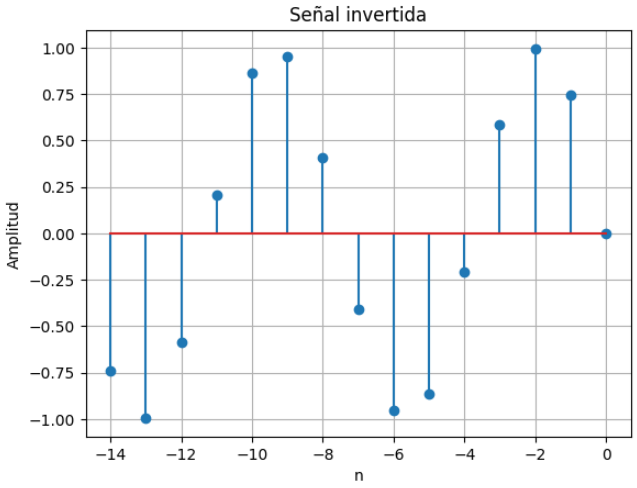


Figura 30. Representación gráfica del Ejemplo 12: inversión de la señal.

La señal resultante tiene la misma cantidad de muestras de la obtenida en el paso anterior, pero con efecto espejo respecto al eje vertical.

Paso 4: Desplazamiento temporal, k=4

Como paso final, vamos a realizar el desplazamiento temporal, con k= 4.

```
k = 4
n_disp = n_inv + k

plt.stem(n_disp, x_new)
plt.xlabel("n")
plt.ylabel("Amplitud")
plt.title("Señal transformada final")
plt.grid(True)
plt.show()
```

Figura 31. Código en Python del Ejemplo 12: desplazamiento temporal, k=4.

Obteniendo:

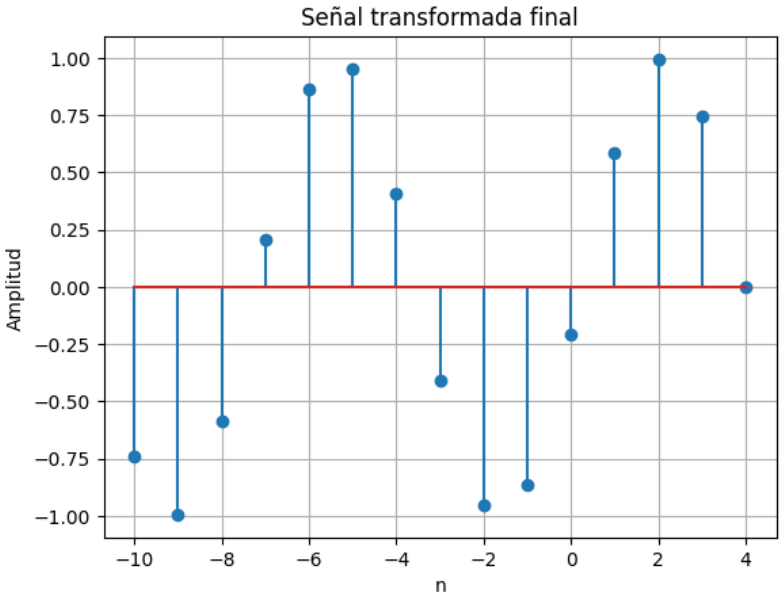


Figura 32. Representación gráfica del Ejemplo 12: resultado final de la transformación.

Finalmente, la señal queda en in intervalo $[-10, \quad 4]$.

Verificación del soporte temporal

Para verificar el soporte de la señal transformada, se evala la expresión −2n+8-2n en los extremos del soporte original:

$$-2n + 8 = 0 \quad \Rightarrow \quad n = 4$$

$$-2n + 8 = 29 \quad \Rightarrow \quad n = -10.5$$ muestra no válida en tiempo discreto

$$-2n + 8 = 28 \quad \Rightarrow \quad n = -10$$

Por lo que la señal transformada queda definida en el intervalo:

$$n \ \epsilon \ [-10, \quad 4]$$

La amplitud de la señal transformada en $n = -10$ corresponde al valor original de $x(28)$, mientras que, la amplitud en $n = 4$ corresponde al valor original de $x(0)$.

Este resultado confirma que la transformación ha sido aplicada correctamente y evidencia que, en tiempo discreto, algunas muestras pueden perderse cuando el argumento de la señal no toma valores enteros.

1.3. PROPIEDADES DE LOS SISTEMAS

En esta última sección del primer capítulo se introducen las propiedades fundamentales de los sistemas. Un sistema puede ser un dispositivo físico, un circuito, un algoritmo o cualquier mecanismo que permita transformar una señal de entrada en una señal de salida.

Existen:

- Sistemas de tiempo continuo, en los cuales tanto la señal de entrada como la señal de salida son continuas.
- Sistemas de tiempo discreto, en los que tanto la entrada como la salida corresponden a señales discretas.
- Sistemas híbridos, donde la entrada es una señal continua y la salida es una señal discreta, como ocurre, por ejemplo, en un conversor analógico–digital (A/D).

A continuación, se describen algunas de las propiedades más importantes de los sistemas.

1.3.1. Memoria

Se dice que un sistema es sin memoria si el valor de la señal de salida en un instante determinado depende únicamente del valor de la señal de entrada en ese mismo instante.

Un ejemplo de un sistema sin memoria en tiempo continuo es:

$$y(t) = 2x(t)$$

Donde $y(t)$ es la salida del sistema y $x(t)$ es la señal de entrada. En este caso, el sistema amplifica la señal por un factor de 2, es decir, por ejemplo, si ingresa una señal constante de 10 [V], la salida será otra señal constante de 20 [V].

En el caso discreto, un ejemplo de sistema sin memoria es:

$$y[n] = 2x[n]$$

Donde $y[n]$ es la salida del sistema y $x[n]$ es la señal de entrada discreta.

Por el contrario, un sistema con memoria es aquel en el que la salida depende de valores pasados o futuros de la señal de entrada. Por ejemplo:

$$y(t) = x(t-2)$$

En ambos casos, para calcular la salida en el instante actual necesito conocer la señal de entrada en un instante pasado. Por ejemplo, para calcular $y(0)$ es necesario conocer $x(-2)$.

1.3.2. Causalidad

Se dice que un sistema es causal, si la salida depende únicamente de valores presentes y/o pasados de la señal de entrada.

Un ejemplo de un sistema causal, es:

$$y(t) = x(t) + x(t-2)$$

Por otro lado, un sistema es no causal si la salida depende de valores futuros de la señal de entrada, como en el caso de:

$$y(t) = x(t+2)$$

Los sistemas no causales no pueden implementarse en tiempo real, aunque pueden utilizarse en aplicaciones de procesamiento fuera de línea.

1.3.3. Estabilidad

Un sistema es estable, si para una entrada acotada (de amplitud finita), la salida también es acotada. Esta propiedad se conoce como estabilidad BIBO (*Bounded Input*, *Bounded Output*).

La verificación de la estabilidad es fundamental, ya que un sistema inestable puede producir salidas no controladas o no ser físicamente realizable.

Por ejemplo, consideremos el siguiente sistema:

$$y(t) = \frac{x(t)}{t}$$

En este caso, para t=0 la salida no está definida y puede crecer sin límite, incluso si la entrada es acotada. Por lo tanto, ese sistema no es estable.

En contraste, el sistema:

$$y(t) = x^2(t)$$

Sí es estable, dado que, para una entrada de amplitud finita, la salida permanece acotada.

1.3.4. Invertibilidad

Se dice que un sistema es invertible si existe otro sistema (llamado sistema inverso) tal que, al conectarlos en serie, se recupera exactamente la señal de entrada. Es decir, si la salida del primer sistema es w(t) o w[n], entonces el sistema inverso permite volver a obtener x(t) o x[n]:

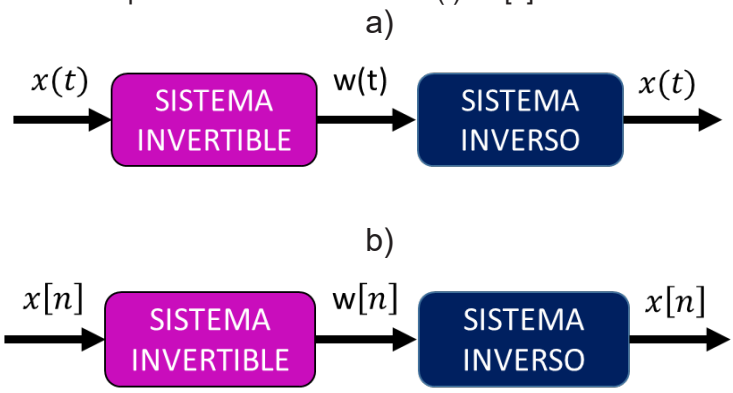


Figura 33. Interconexión sistema invertible-inverso: a) continuo, b) discreto.

Equivalentemente, un sistema es invertible si existe una correspondencia uno a uno entre entrada y salida: entradas diferentes no pueden producir la misma salida.

Ejemplo 13: Sistema invertible en tiempo continuo

Considere el sistema:

$$S_1: \qquad y(t) = x(2t)$$

Cuyo sistema inverso, es:

$$S_2: \qquad y(t) = x(t/2)$$

El primer sistema (S_1) comprime la señal de entrada, y el segundo sistema (S_2) la dilata, de manera que al aplicar ambos en cascada se recupera nuevamente la señal original x(t).

Ejemplo 14: Sistema discreto que no es invertible

Ahora analicemos el sistema discreto:

$$S_1: \qquad y[n] = x[2n]$$

Aquí también ocurre una compresión en el eje temporal. Sin embargo, a diferencia del caso continuo, en tiempo discreto esta operación descarta muestras: la salida solo conserva los valores de x[n] en índices pares (0, 2, 4, …), por lo que se pierde la información correspondiente a las muestras de índices impares (1, 3, 5, …).

En consecuencia, no es posible recuperar x[n] únicamente a partir de y[n], ya que múltiples señales de entrada distintas pueden producir exactamente la misma salida (por ejemplo, cualquier modificación en las muestras impares de x[n] no afecta a y[n]). Por lo tanto, este sistema no es invertible.

1.3.5. Linealidad

Un sistema se dice lineal si cumple el principio de superposición, el cual incluye dos propiedades fundamentales: aditividad y homogeneidad.

Esto significa que, si al sistema se le aplican dos señales de entrada y se combinan linealmente, la salida resultante es la misma combinación lineal de las salidas individuales.

Ejemplo 15: sistema lineal

Considere el sistema:

$$y(t) = 2x(t)$$

Definamos dos señales de entrada constantes:

$$x_1(t) = 2, \quad \text{y} \quad x_2(t) = 3$$

Al evaluar cada señal en el sistema, se obtiene:

$$y_1(t) = 2x_1(t) = 4 \quad \text{y} \quad y_2(t) = 2x_2(t) = 6$$

Ahora definimos una nueva señal como la suma de las dos señales de entrada:

$$x_3(t) = x_1(t) + x_2(t) = 5$$

Y al evaluar la señal en el sistema, se obtiene:

$$y_3(t) = 2x_3(t) = 10$$

Finalmente, comparamos:

$$y_3(t) = y_1(t) + y_2(t) = 10$$

Dado que se cumple el principio de superposición, se concluye que el sistema es lineal.

Ejemplo 16: sistema no lineal

Consideremos ahora el sistema

$$y(t) = 2x(t) + 1$$

Utilizando las mismas señales de entrada

$$x_1(t) = 2, \quad \text{y} \quad x_2(t) = 3$$

Obtenemos:

$$y_1(t) = 5 \quad \text{y} \quad y_2(t) = 7$$

Definiendo nuevamente

$$x_3(t) = x_1(t) + x_2(t) = 5$$

Y al evaluar la señal en el sistema:

$$y_3(t) = 11$$

Sin embargo,

$$y_1(t) + y_2(t) = 12 \neq y_3(t),$$

Por lo tanto, este sistema no cumple el principio de superposición y se clasifica como no lineal.

1.3.6. Invarianza

Un sistema es invariante en el tiempo si al desplazar la señal de entrada y luego evaluarla en el sistema, se obtiene el mismo resultado que al evaluar primero la señal en el sistema y después desplazar la señal de salida en la misma cantidad.

Formalmente, si una señal de entrada x[n] produce una salida y[n], el sistema es invariante si para cualquier desplazamiento n_0 se cumple:

$$x[n - n_0] \xrightarrow{sistema} y[n - n_0]$$

Es decir, el sistema no depende del instante específico en el que se aplica la señal, sino únicamente de su forma.

Ejemplo 17: sistema invariante

Supongamos que tenemos una señal x[n] la cual se encuentra en el intervalo $n \in [0, \ 3]$, con las amplitudes $x[n] = \{1, \ 2, \ 1, \ 2\}$.

Esta señal ingresa al sistema

$$y[n] = x[2n]$$

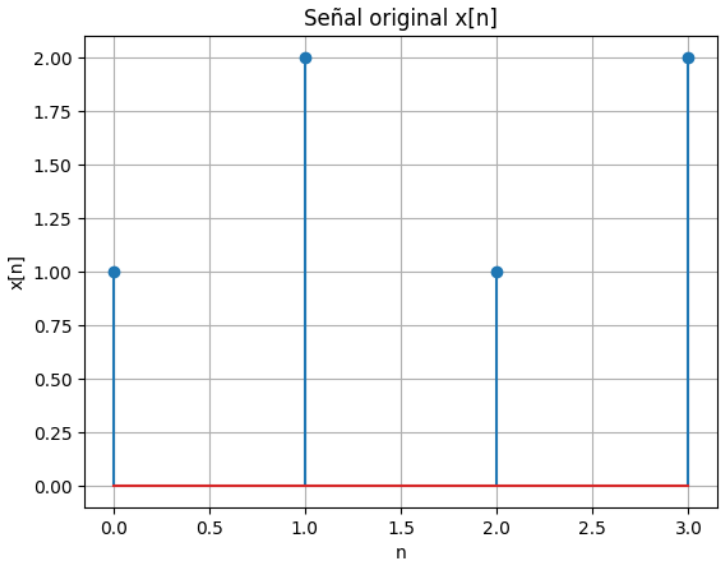
El cual realiza una compresión de la señal de entrada, por factor de α=2, así:

```python
import numpy as np
import matplotlib.pyplot as plt

m = 4
n = np.linspace(0, m-1, m)
x = np.array([1, 2, 1, 2])

α = 2
n_new = np.linspace(0, int((m-1)/α), int(m/α))
x_new = x[::α]


# Señal original
plt.figure()
plt.stem(n, x)
plt.xlabel("n")
plt.ylabel("x[n]")
plt.title("Señal original x[n]")
plt.grid(True)

# Señal comprimida
plt.figure()
plt.stem(n_new, x_new)
plt.xlabel("n")
plt.ylabel("y[n]")
plt.title("Señal comprimida y[n] = x[2n]")
plt.grid(True)
```

Figura 34. Código en Python señal del Ejemplo 17: señal x[n] y y[n].

Obteniendo esta señal de salida del sistema, para la señal de entrada x[n]:

A)



B)

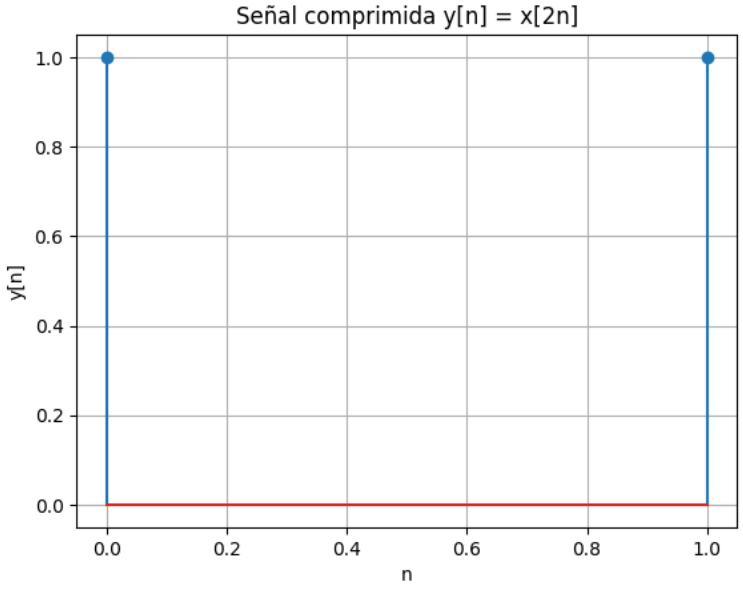


Figura 35. Representación gráfica del Ejemplo 17: a) señal x[n], b) señal y[n].

Ahora, desplacemos la señal de entrada una posición a la derecha, es decir:

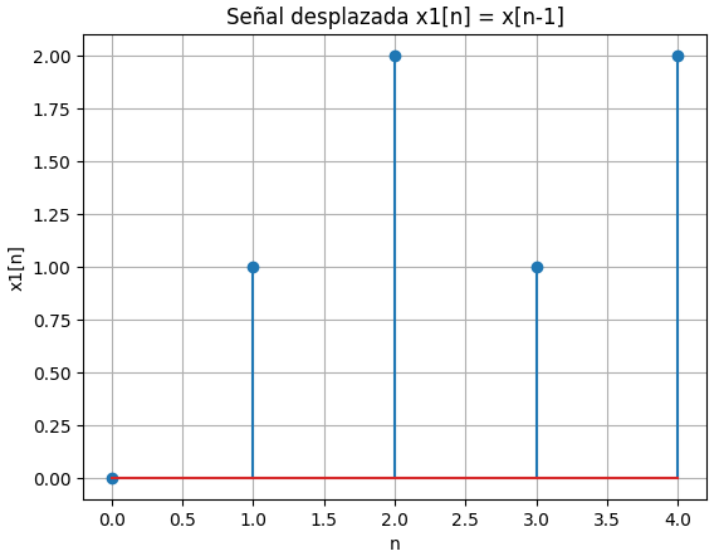
$$x_1[n] = x[n-1]$$

La cual ingresaremos al mismo sistema, obteniendo:

```
# Señal desplazada x1[n] = x[n-1]
m = 4
n = np.arange(0, m+1)
x1 = np.array([0, 1, 2, 1, 2])

# Sistema: y[n] = x[2n]
α = 2
n_new = np.arange(0, int(m/α) + 1)
y1 = x1[::α]

# Señal desplazada
plt.figure()
plt.stem(n, x1)
plt.xlabel("n")
plt.ylabel("x1[n]")
plt.title("Señal desplazada x1[n] = x[n-1]")
plt.grid(True)

# Salida del sistema para la señal desplazada
plt.figure()
plt.stem(n_new, y1)
plt.xlabel("n")
plt.ylabel("y1[n]")
plt.title("Salida del sistema: y1[n] = x1[2n]")
plt.grid(True)
```

Figura 36. Código en Python señal del Ejemplo 17: señal x1[n] y y1[n].

A)



B)

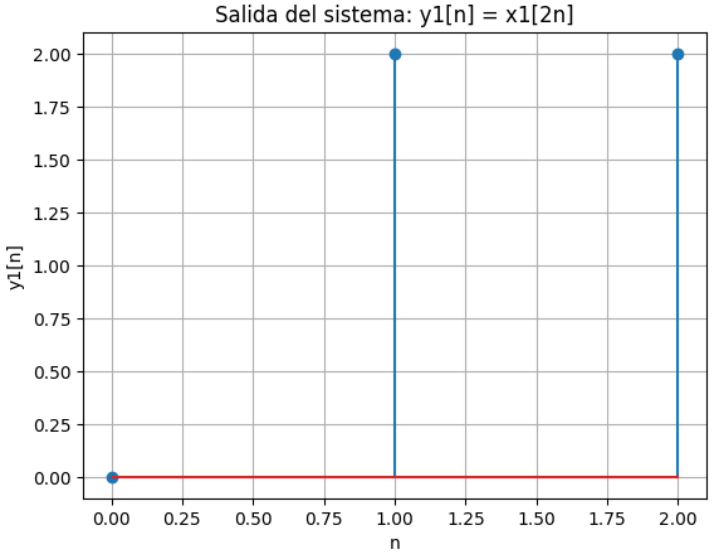


Figura 37. Representación gráfica del Ejemplo 17: a) señal x[n], b) señal y[n].

Si aplicamos desplazamiento a la derecha de la señal y[n] obtenida en la Figura 35b, nos damos cuenta que NO es la misma $y_1[n]$. Entonces, el sistema NO es invariante.

Nota conceptual

Un error común al analizar la invarianza en el tiempo es asumir que toda operación sobre el eje temporal conserva los desplazamientos. Sin embargo, los sistemas que realizan escalamiento temporal (compresión o dilatación) modifican la relación entre las muestras, de modo que un desplazamiento en la señal de entrada no se traduce en el mismo desplazamiento en la señal de salida. Por esta razón, los sistemas del tipo $y[n]=x[\alpha n]$, con $\alpha \neq 1$, no son invariantes en el tiempo.