UNIVERSIDAD ARGENTINA DE LA EMPRESA
Facultad de Ingeniería y Ciencias Exactas
Departamento de Tecnología Informática

SEÑALES Y SISTEMAS

Trabajo Practico 3

CONVOLUCIÓN

ALUMNO: LU:

CARRERA: FECHA:

NOTA: ESIGUIENTE TRABAJO TIENE CARÁCTER DE EXAMEN ESCRITO Y ES UN DOCUMENTO PÚBLICO DE GRAN IMPORTANCIA PARA LA EVALUACIÓN DE LOS CONOCIMIENTOS ADQUIRIDOS, POR LO TANTO, SE SOLICITA LEER ATENTAMENTE LO SIGUIENTE:

- Responda claramente las consignas, detallando con la mayor precisión posible lo solicitado en cada ítem.
- Sea prolijo y ordenado en el desarrollo de los temas.
- Sea cuidadoso con la ortografía.
- Lea completamente este enunciado antes de comenzar a trabajar.

NOTA EN NÚMEROS	NOTA EN LETRAS	SELLO	FIRMA DEL DOCENTE

1. Objetivos

- Desarrollar capacidades de búsqueda de información
- Entender el comportamiento de las simulaciones
- Descubrir y compartir las diversas herramientas de software del mercado

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso. La entrega será de carácter individual.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con este trabajo expuesto como carátula que contenga los datos completos del alumno, seguido de un informe impreso de acuerdo con lo que mencionaremos en la sección 5, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Trabajo

Considerando las siguientes consignas, resuelva analíticamente y luego realice un programa en Python que sea capaz de resolver cada problema. Represente gráficamente el resultado describiendo las gráficas que considere necesarias.

> 4.1 Desarrolle una serie de scripts en el lenguaje de programación PYTHON, el cual describa y grafique el producto de convolución de dos señales elegidas. Realice en el informe el contraste matemático y compare los resultados.
>
> 4.2 Desarrolle una serie de scripts en el lenguaje de programación PYTHON, el cual describa y grafique la suma de convolución de dos señales elegidas. Realice en el informe el contraste matemático y compare los resultados.

5. Informe

El informe deberá incluir:

- Documentación a cada análisis matemático realizado
- Documentación relevante al diseño e implementación de cada algoritmo
- Documentación relevante al diseño e implementación del programa.
- El código fuente
- Capturas de pantallas y corridas
- Este enunciado junto con la caratula.

4.1 – Producto de Convolución

- *Código:*

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

"""
Basado en el ejemplo que se muestra en:
https://es.wikipedia.org/wiki/Convoluci%C3%B3n

Definimos:
f(t) = square(t) = u(t+1/2) - u(t-1/2)
g(t) = e ^ -(t-1) * u(t)
"""

def square(t):
    return np.heaviside(t+1/2, 1) - np.heaviside(t-1/2, 1)

# Defino las funciones a convolucionar
f = lambda t: square(t)
g = lambda t: np.exp(-t+1) * np.heaviside(t, 1)

# Defino el tiempo
m = 500 # muestras
dt = 1/m

inicio = -5
fin = 5

t = np.arange(inicio, fin + dt, dt)

y  = np.convolve(f(t), g(t), 'same') * dt

# Grafico
fig, ax1 = plt.subplots(figsize=(8, 8))
ax2 = ax1.twinx()

ax1.axvline(0, color='gray')
ax1.axhline(0, color='gray')

ax1.plot(t, f(t), label='f(t)', color='red')
ax1.plot(t, g(t), label='g(t)', color='blue')
ax2.plot(t, y, label='y(t) = {f*g}(t)', color='green')

ax1.set_xlabel("t")
ax1.set_ylabel("f,g")
ax1.tick_params(axis="y")
ax2.set_ylabel("y")
ax2.tick_params(axis="y")

plt.tight_layout()

plt.axvline(x=0.5, ls='--')

fig.legend()
plt.show()
```

Bloque 1

Bloque 2

Bloque 3

Bloque 4

- *Breve explicación:*
    - Bloque 1:
      Se importan las librerías necesarias para la operación del programa desarrollado, y se definen aliases con el objetivo de simplificar las llamadas a dichas librerías

    - Bloque 2:
      En primera instancia, se define una función que genera un pulso rectangular de amplitud 1, que abarca el rango de -0.5 a 0.5 en el eje t, para simplificar la llamada, ya que esta está compuesta por una resta de dos escalones de Heaviside, lo cual la convierte en algo muy aparatoso como para llamarlo constantemente en el código.

      En segunda instancia, se definen las variables necesarias para invocar a las funciones utilizadas, utilizando la variable temporal t como parámetro.

      La función *f* es, como se ha expuesto anteriormente, un pulso rectangular de amplitud 1 comprendido entre -0.5 y 0.5 en el eje t.

      La función *g* es una función exponencial con argumento -t+1, multiplicada por un escalón, tal que su valor sea 0 siempre que t sea menor a 0.

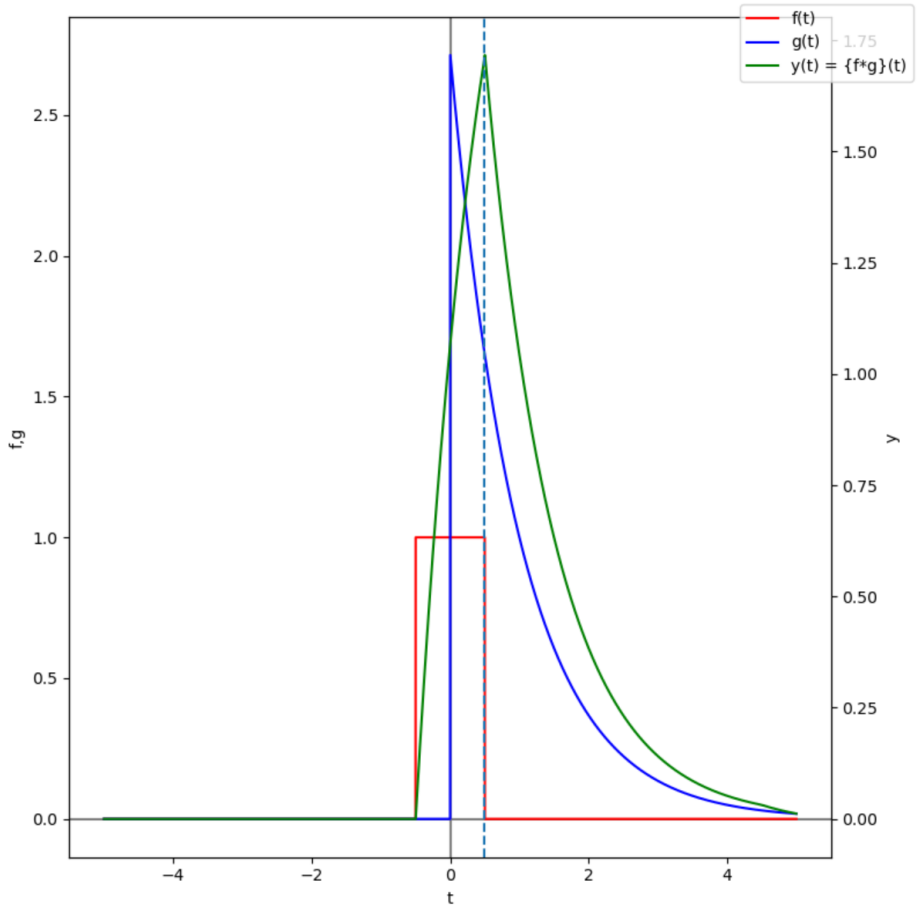
    - Bloque 3:
      Utilizando la función *linspace* de la librería numpy (función que devuelve un array de números en el intervalo especificado, con una distancia entre pasos calculada con la cantidad de pasos que debe haber en el rango especificada), se define para cada función (*t* para *f*, *t1* para *g*) una línea de tiempo sobre la cual se va a muestrear dicha función (dado que las computadoras no son capaces de procesar intervalos contínuos, dada su naturaleza física).

      Luego, se llama a la función *convolve*, de la librería numpy (que devuelve una convolución lineal discreta de un array, que en este caso son las muestras de nuestras funciones), y se escalan los valores de esa convolución por un diferencial de tiempo *dt* (espacio entre la muestra actual y la anterior), a su vez multiplicados por un pequeño factor de corrección necesario para que los valores coincidan con los de la verificación matemática manual.

    - Bloque 4:
      Este bloque utiliza el módulo *pyplot* de la librería *matplotlib* para graficar tanto la convolución resultante como las funciones utilizadas para realizarla. Por una cuestión estilística, se decidió integrar todo en un mismo gráfico, haciendo que el eje izquierdo represente el eje vertical de las funciones *f* y *g*, mientras que el derecho represente el eje vertical de la convolución *y*.

- *Resultado de la ejecución:*

```
In [34]: %run "producto-convolucion.py" 'no-print'
```



- *Análisis matemático manual:*

Para:

$$x(t) = \begin{cases} 1 \; si - 0.5 < t < 0.5 \\ 0 \;\; \forall \; otro \; t \end{cases}$$

$$h(t) = \begin{cases} e^{-t+1} \; si \; t \geq 0 \\ 0 \; \forall \; otro \; t \end{cases}$$

Con:

$$y(t) = \int_{-\infty}^{\infty} x(\tau) \cdot h(t-\tau) \, d\tau$$

Debemos separar la integral en tres intervalos.

- Para t < -0.5:

$$y_1(t) = \int_{-\infty}^{-0.5} 0 \cdot e^{-(t-\tau)+1} \, d\tau$$

$$y_1(t) = 0$$

- Para -0.5 < t < 0.5:

$$y_2(t) = \int_{-0.5}^{t} 1 \cdot e^{-(t-\tau)+1}\, d\tau$$

$$y_2(t) = \int_{-0.5}^{t} e^{-t+\tau+1}\, d\tau$$

$$y_2(t) = e^{-t+1}\left[\int_{-0.5}^{t} e^{\tau}\, d\tau\right]$$

$$y_2(t) = e^{-t+1}\left[e^{\tau} \Big|_{-0.5}^{t}\right]$$

$$y_2(t) = e^{-t+1}[e^{t} - e^{-0.5}]$$

$$y_2(t) = e - e^{-t+0.5}$$

- Para t > 0.5

$$y_3(t) = \int_{-0.5}^{0.5} 1 \cdot e^{-(t-\tau)+1}\, d\tau$$

$$y_3(t) = \int_{-0.5}^{0.5} e^{-t+\tau+1}\, d\tau$$

$$y_3(t) = e^{-t+1}\left[\int_{-0.5}^{0.5} e^{\tau}\, d\tau\right]$$

$$y_3(t) = e^{-t+1}\left[e^{\tau} \Big|_{-0.5}^{0.5}\right]$$

$$y_3(t) = e^{-t+1}[e^{0.5} - e^{-0.5}]$$

Entonces:

$$y(t) = \begin{cases} 0 \; si \; t < 0 \\ e - e^{-t+0.5} \; si \; -0.5 < t < 0.5 \\ e^{-t+1}[e^{0.5} - e^{-0.5}] \; si \; t > 0.5 \end{cases}$$

Gráfico:

4.2 – Suma de Convolución

- *Código:*
  - *Versión 1 (gran número de muestras):*

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

"""
Basado en el ejemplo que se muestra en:
https://es.wikipedia.org/wiki/Convoluci%C3%B3n

Definimos:
f(t) = square(t) = u(t+1/2) - u(t-1/2)
g(t) = e ^ -(t-1) * u(t)
"""

def square(t):
    return np.heaviside(t+1/2, 1) - np.heaviside(t-1/2, 1)

# Defino las funciones a convolucionar
f = lambda t: square(t)
g = lambda t: np.exp(-t+1) * np.heaviside(t, 1)

# Defino el tiempo
m = 25 # muestras por división
dn = 1/(m-1)

inicio = -5
fin = 5

n = np.arange(inicio, fin + dn, dn)

y  = np.convolve(f(n), g(n), 'same') * dn

# Imprimo valores
if len(sys.argv) > 1:
    if sys.argv[1] != 'no-print':
        print('y(t)')
        print(y)

fig, ax1 = plt.subplots(figsize=(8, 8))
ax2 = ax1.twinx()

ax1.axvline(0, color='gray')
ax1.axhline(0, color='gray')

stem1 = ax1.stem(n, f(n), label='f[n]')
stem1.markerline.set_color('blue')
stem1.stemlines.set_color('blue')

stem2 = ax1.stem(n, g(n), label='g[n]')
stem2.markerline.set_color('green')
stem2.stemlines.set_color('green')

stem3 = ax2.stem(n, y, label='y[n] = {f*g}[n]')
stem3.markerline.set_color('red')
stem3.stemlines.set_color('red')

ax1.set_xlabel("n")
ax1.set_ylabel("f,g")
ax1.tick_params(axis="y")
ax2.set_ylabel("y")
ax2.tick_params(axis="y")

fig.legend()
plt.show()
```

Bloque 1

Bloque 2

Bloque 3

Bloque 4

Bloque 5

- *Versión 2 (pequeña cantidad de muestras):*

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

"""
Basado en el ejemplo que se muestra en:
https://es.wikipedia.org/wiki/Convoluci%C3%B3n

Definimos:
f(n) = square(n) = u(n+1/2) - u(n-1/2)
g(n) = e ^ -(n-1) * u(n)
"""

def square(n):
    return np.heaviside(n+1/2, 1) - np.heaviside(n-1/2, 1)

# Defino las funciones a convolucionar
f = lambda n: square(n)
g = lambda n: np.exp(-n+1) * np.heaviside(n, 1)

# Defino el tiempo
m = 20 # muestras
n = np.linspace(-1, 3, m) # m muestras entre -2 y 2

# Convolucion
y = np.convolve(f(n), g(n),'full')

# Defino el tiempo de convolución
n1 = np.linspace(-1, 3, 2*m-1) # 2m-1 muestras entre -2 y 2

# Imprimo valores
#if len(sys.argv) > 1:
#    if sys.argv[1] != 'no-print':
#        print('y[n]')
#        print(y)

# Grafico
fig, ax1 = plt.subplots(figsize=(8, 8))
ax2 = ax1.twinx()

ax1.axvline(0, color='gray')
ax1.axhline(0, color='gray')

stem1 = ax1.stem(n, f(n), label='f[n]')
stem1.markerline.set_color('blue')
stem1.stemlines.set_color('blue')

stem2 = ax1.stem(n, g(n), label='g[n]')
stem2.markerline.set_color('green')
stem2.stemlines.set_color('green')

stem3 = ax2.stem(n1, y, label='y[n] = {f*g}[n]')
stem3.markerline.set_color('red')
stem3.stemlines.set_color('red')

ax1.set_xlabel("n")
ax1.set_ylabel("f,g")
ax1.tick_params(axis="y")
ax2.set_ylabel("y")
ax2.tick_params(axis="y")

fig.legend()
plt.show()
```

Bloque 1

Bloque 2

Bloque 3

Bloque 4

Bloque 5

- *Breve explicación*
  - *Versión 1:*
    - Bloque 1:
      Se importan las librerías necesarias para la operación del programa desarrollado, y se definen aliases con el objetivo de simplificar las llamadas a dichas librerías.

    - Bloque 2:
      En primera instancia, se define una función que genera un pulso rectangular de amplitud 1, que abarca el rango de -0.5 a 0.5 en el eje n, para simplificar la llamada, ya que esta está compuesta por una resta de dos escalones de Heaviside, lo cual la convierte en algo muy aparatoso como para llamarlo constantemente en el código.

      En segunda instancia, se definen las variables necesarias para invocar a las funciones utilizadas, utilizando la variable discreta n como parámetro.

      La función *f* es, como se ha expuesto anteriormente, un pulso rectangular de amplitud 1 comprendido entre -0.5 y 0.5 en el eje n.

      La función *g* es una función exponencial con argumento -n+1, multiplicada por un escalón, tal que su valor sea 0 siempre que n sea menor a 0.

    - Bloque 3:
      Utilizando la función *arange* de la librería *numpy* (función que devuelve un array de números en el intervalo especificado, con una distancia especificada entre valores), se define una única línea de tiempo sobre la cual se va a muestrear ambas funciones, para obtener una lista de valores.

      Luego, se llama a la función *convolve*, de la misma librería (que devuelve una convolución lineal discreta de un array, que en este caso es nuestra lista de valores) y se escalan los valores de esa convolución por un diferencial de tiempo *dn* (espacio entre el valor actual y el anterior).

    - Bloque 4:
      Código que imprime, uno por uno, los valores del array de Convolución devueltos por la función *convolve* de *numpy*, por cuestiones de depuración.

    - Bloque 5:
      Este bloque utiliza el módulo *pyplot* de la librería *matplotlib* para graficar tanto la convolución resultante como las funciones utilizadas para realizarla. Por una cuestión estilística, se decidió integrar todo en un mismo gráfico, haciendo que el eje izquierdo represente el eje vertical de las funciones *f* y *g*, mientras que el derecho represente el eje vertical de la convolución *y*. Cada muestra se encuentra representada por una Delta de Dirac escalada por el valor del punto correspondiente.

- *Versión 2:*
  - Bloque 1:
    Se importan las librerías necesarias para la operación del programa desarrollado, y se definen aliases con el objetivo de simplificar las llamadas a dichas librerías

  - Bloque 2:
    En primera instancia, se define una función que genera un pulso rectangular de amplitud 1, que abarca el rango de -0.5 a 0.5 en el eje n, para simplificar la llamada, ya que esta está compuesta por una resta de dos escalones de Heaviside, lo cual la convierte en algo muy aparatoso como para llamarlo constantemente en el código.

    En segunda instancia, se definen las variables necesarias para invocar a las funciones utilizadas, utilizando la variable discreta n como parámetro.

    La función *f* es, como se ha expuesto anteriormente, un pulso rectangular de amplitud 1 comprendido entre -0.5 y 0.5 en el eje n.

    La función *g* es una función exponencial con argumento -n+1, multiplicada por un escalón, tal que su valor sea 0 siempre que n sea menor a 0.

  - Bloque 3:
    Utilizando la función *linspace* de la librería numpy (función que devuelve un array de números en el intervalo especificado, con una distancia especificada entre valores), se define para cada función (n para *f*, *n1* para *g*) una línea de tiempo sobre la cual se va a muestrear dicha función, con el fin de obtener un array de valores.

    Luego, se llama a la función *convolve*, de la librería numpy (que devuelve una convolución lineal discreta de un array, que en este caso es nuestro array de valores.

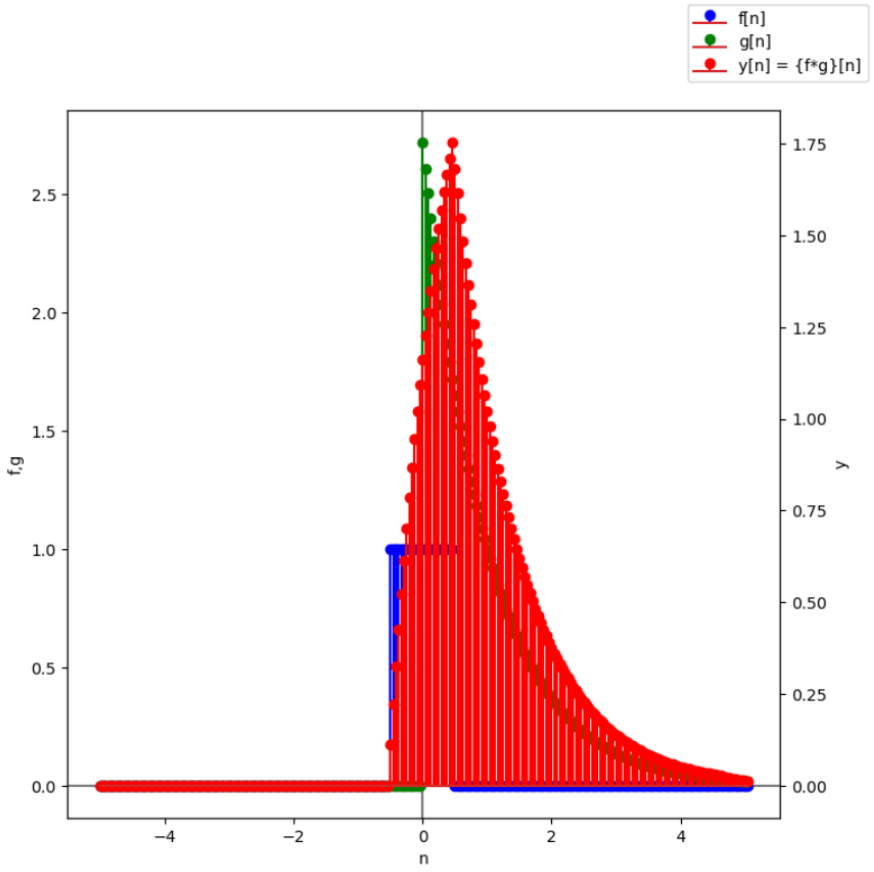
- Bloque 4:
  Código que imprime, uno por uno, los valores del array de Convolución devueltos por la función *convolve* de *numpy*, por cuestiones de depuración.

  - Bloque 5:
    Este bloque utiliza el módulo *pyplot* de la librería *matplotlib* para graficar tanto la convolución resultante como las funciones utilizadas para realizarla. Por una cuestión estilística, se decidió integrar todo en un mismo gráfico, haciendo que el eje izquierdo represente el eje vertical de las funciones *f* y *g*, mientras que el derecho represente el eje vertical de la convolución *y*. Cada muestra se encuentra representada por una Delta de Dirac escalada por el valor del punto correspondiente.

- *Resultado de la ejecución*
  - *Versión 1:*

```
In [5]: %run "suma-convolucion-v1.py" 'no-print'
```

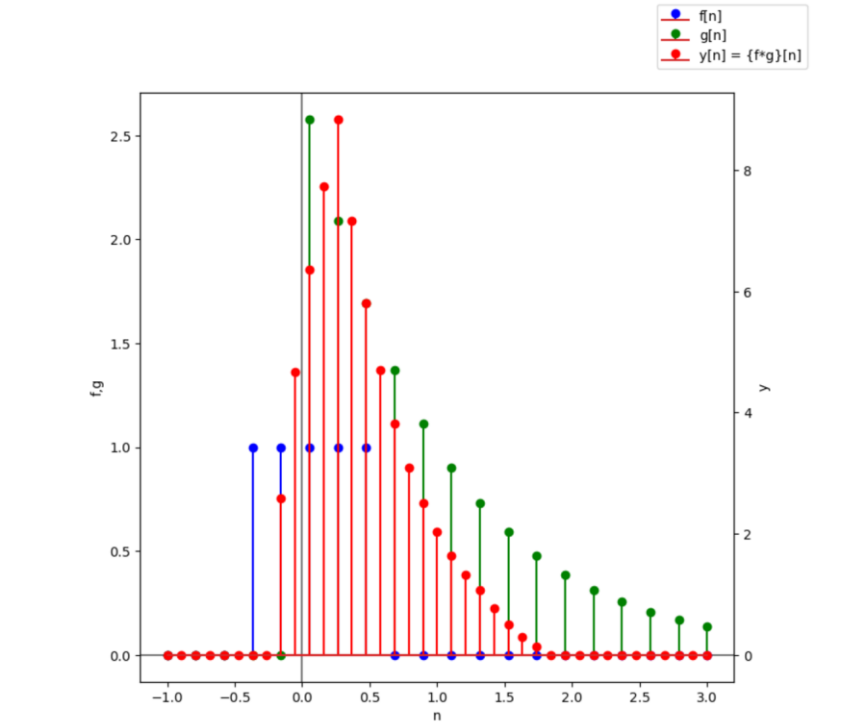
```
y(t)
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
0.11326174 0.22190121 0.32610705 0.42606018 0.52193418 0.6138955
0.70210384 0.78671234 0.86786793 0.94571151 1.02037825 1.0919978
1.16069452 1.22658769 1.28979173 1.35041637 1.40856689 1.46434425
1.51784531 1.56916296 1.61838631 1.66560083 1.7108885  1.75432795
1.68273288 1.61405963 1.54818898 1.48500655 1.42440263 1.36627198
1.31051368 1.25703091 1.20573079 1.15652427 1.10932588 1.06405369
1.02062908 0.97897666 0.93902409 0.900702   0.86394387 0.82868585
0.79486673 0.76242779 0.73131269 0.70146743 0.67284016 0.64538119
0.61904283 0.59377936 0.5695469  0.54630338 0.52400844 0.50262337
0.48211104 0.46243583 0.44356357 0.4254615  0.40809819 0.39144348
0.37546846 0.36014538 0.34544766 0.33134975 0.31782719 0.30485649
0.29241513 0.28048151 0.26903491 0.25805544 0.24752406 0.23742247
0.22773313 0.21843922 0.20952459 0.20097378 0.19277193 0.18490481
0.17735874 0.17012063 0.16317792 0.15651854 0.15013093 0.14400401
0.13812713 0.13249008 0.12708309 0.12189676 0.11692209 0.11215043
0.10757351 0.10318338 0.09897241 0.09493329 0.09105901 0.08734285
0.08377834 0.0803593  0.07707979 0.07393412 0.07091683 0.06802268
0.06524663 0.06258388 0.0600298  0.05757995 0.05523008 0.05297611
0.05081413 0.04874038 0.04675126 0.04484331 0.04301323 0.04125784
0.03957408 0.03795904 0.03640992 0.03492401 0.03349874 0.03213164
0.03082033 0.02956253 0.02765394 0.02582323 0.02406724 0.02238291
0.02076732 0.01921766 0.01773124 0.01630549 0.01493792 0.01362616
0.01236793 0.01116106]
```



Versión 2:

```
In [3]: %run "suma-convolucion-v2.py" 'no-print'

y[n]
[0.         0.         0.         0.         0.         0.
 0.         0.         2.57891411 4.66824132 6.36092592 7.73226744
 8.84327038 7.1644439  5.80432964 4.70242256 3.80970401 3.08646117
 2.50052039 2.02581593 1.64123045 1.32965554 1.07723072 0.76308388
 0.50857538 0.30238335 0.13533528 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.        ]
```



- *Análisis matemático manual:*

  Para:

$$x[n] = \begin{cases} 1 \; si \; -0.5 < n < 0.5 \\ 0 \;\; \forall \; otro \; n \end{cases}$$

$$h[n] = \begin{cases} e^{-n+1} \; si \; n \geq 0 \\ 0 \; \forall \; otro \; n \end{cases}$$

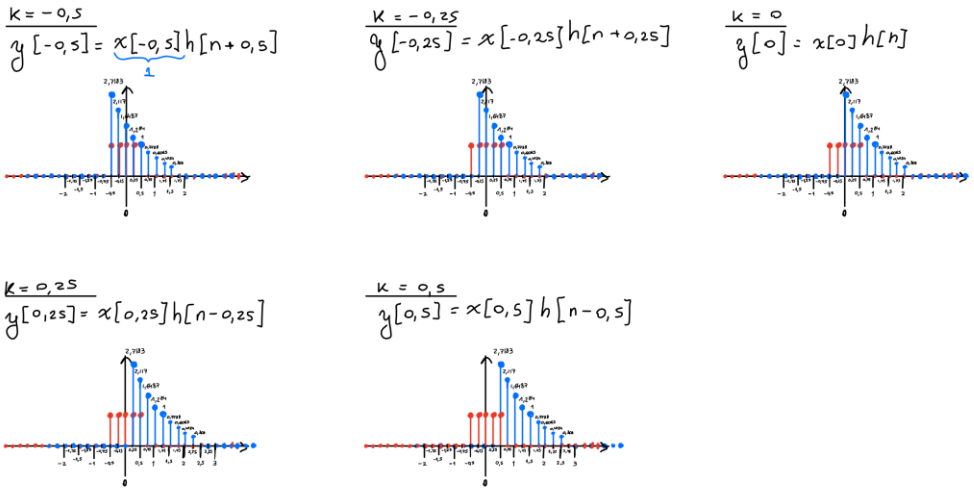
  Con valores de h[n]:

n	h[n]
0	2,71828183
0,25	2,11700002
0,5	1,64872127
0,75	1,28402542
1	1
1,25	0,77880078
1,5	0,60653066
1,75	0,47236655
2	0,36787944

Con:

$$y\{n] = \sum_{k=0}^{N} x[k]\, h[n-k]$$

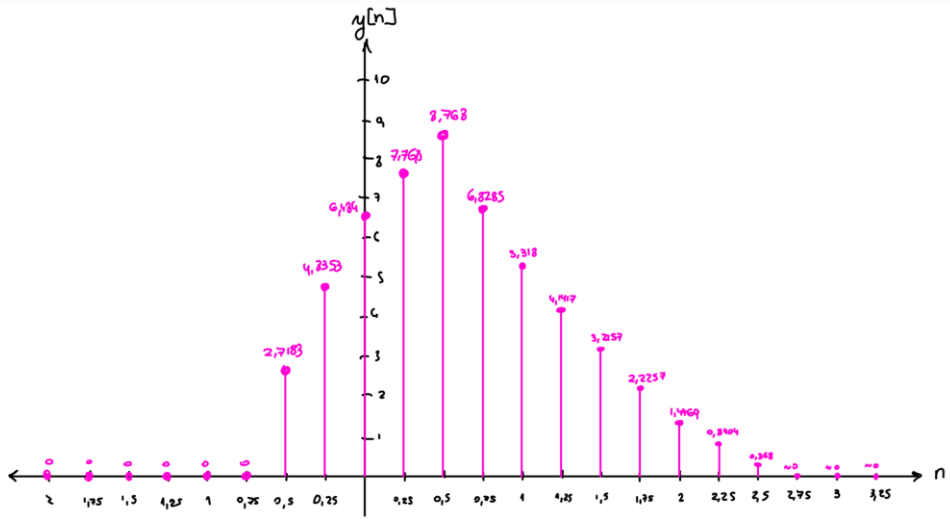
Realizamos el deslizamiento de la función h[n]:



Sumando algorítmicamente, tenemos que:

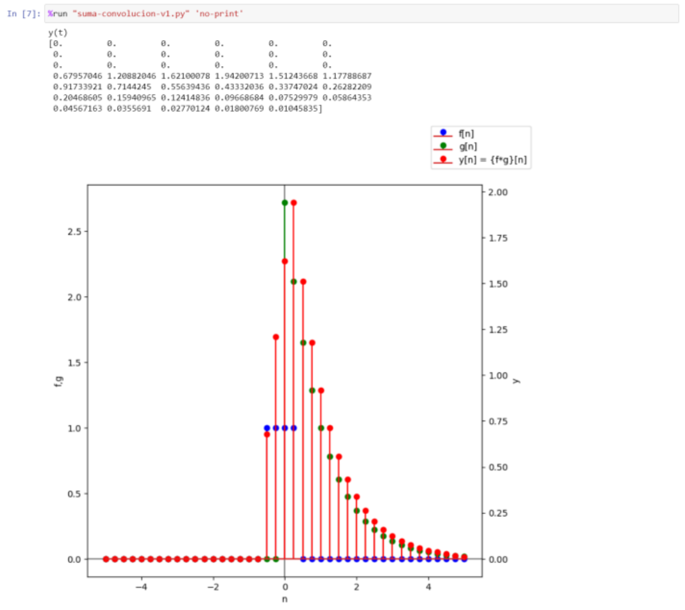
Posición	2,5	2,25	2	1,75	1,5	1,25	1	0,75	0,5	0,25	0	-0,25	-0,5
					0,368	0,4724	0,6065	0,7788	1	1,284	1,6487	2,117	2,7183
							x		1	1	1	1	1
					0,368	0,4724	0,6065	0,7788	1	1,284	1,6487	2,117	2,7183
				0,368	0,4724	0,6065	0,7788	1	1,284	1,6487	2,117	2,7183	-
			0,368	0,4724	0,6065	0,7788	1	1,284	1,6487	2,117	2,7183	-	-
		0,368	0,4724	0,6065	0,7788	1	1,284	1,6487	2,117	2,7183	-	-	-
	0,368	0,4724	0,6065	0,7788	1	1,284	1,6487	2,117	2,7183	-	-	-	-
	0,368	0,8404	1,4469	2,2257	3,2257	4,1417	5,318	6,8285	8,768	7,768	6,484	4,8353	2,7183

Por lo que la convolución resultante será:

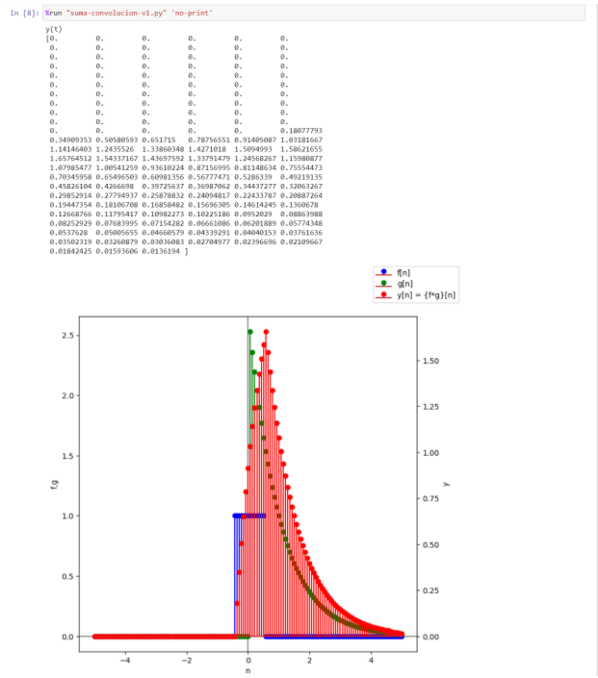


Conclusiones y aclaraciones:

- Se ha limitado el análisis matemático manual a la Versión 2 (la original, en realidad) del código, debido a que, en el otro caso, las muestras son tan abundantes y con intervalos intermedios tan pequeños que es imposible calcularlo manualmente. La Versión 1 fue realizada como una prueba de concepto, para medir los efector del número de muestras sobre los resultados.
- La diferencia entre las versiones 1 y 2 radica en dos nociones principales:
    - En la versión 1, los resultados obtenidos poseen un alto grado de dependencia en la cantidad de muestras tomadas de la función original, y se van acercando progresivamente a los resultados reales a medida que se incrementa el número de muestras por división de escala. 25 muestras parece ser el mejor balance entre tiempo de computación, legibilidad y precisión de los resultados.
        - 5 muestras:



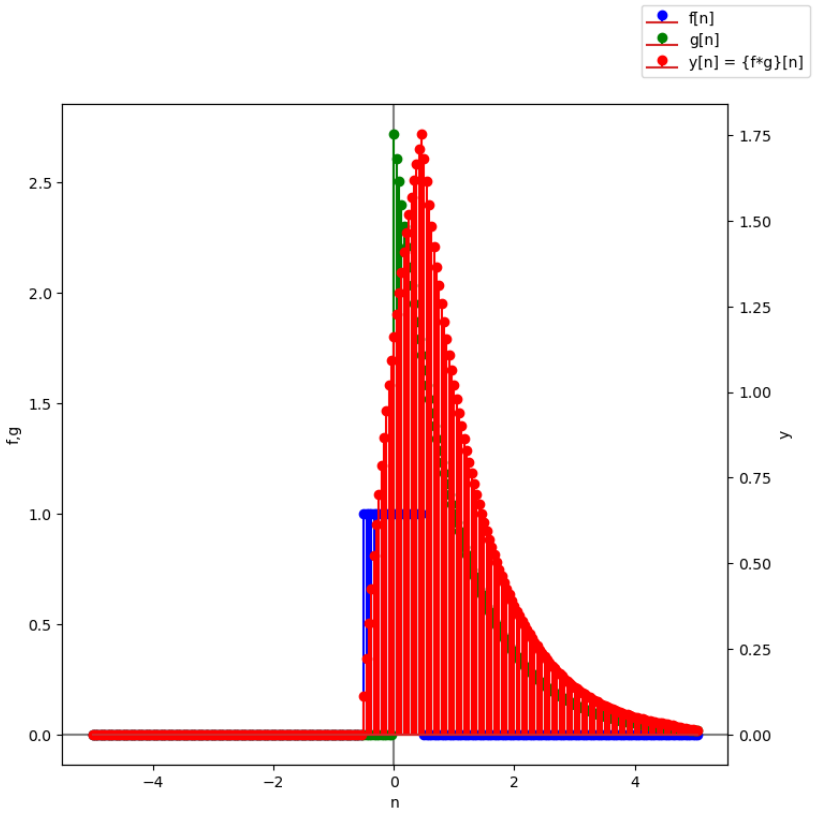
        - 15 muestras:



- 25 muestras

```
In [9]: %run "suma-convolucion-v1.py" 'no-print'
```

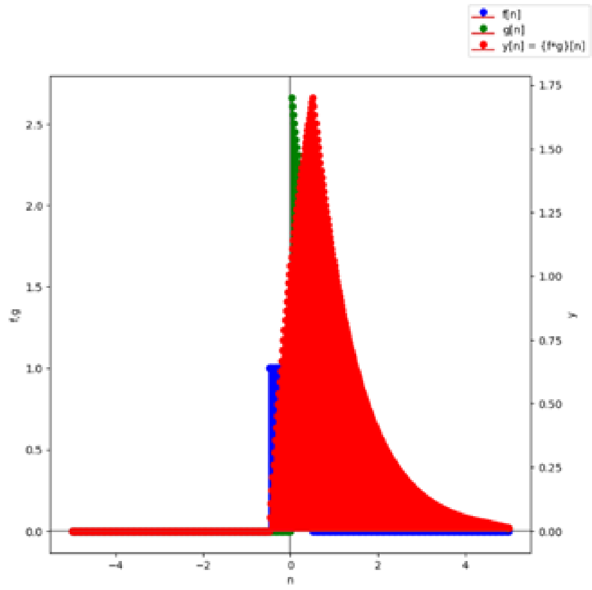
```
y(t)
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
0.11326174 0.22190121 0.32610705 0.42606018 0.52193418 0.6138955
0.70210384 0.78671234 0.86786793 0.94571151 1.02037825 1.0919978
1.16069452 1.22658769 1.28979173 1.35041637 1.40856689 1.46434425
1.51784531 1.56916296 1.61838631 1.66560083 1.7108885  1.75432795
1.68273288 1.61405963 1.54818898 1.48500655 1.42440263 1.36627198
1.31051368 1.25703091 1.20573079 1.15652427 1.10932588 1.06405369
1.02062908 0.97897666 0.93902409 0.900702   0.86394387 0.82868585
0.79486673 0.76242779 0.73131269 0.70146743 0.67284016 0.64538119
0.61904283 0.59377936 0.5695469  0.54630338 0.52400844 0.50262337
0.48211104 0.46243583 0.44356357 0.4254615  0.40809819 0.39144348
0.37546846 0.36014538 0.34544766 0.33134975 0.31782719 0.30485649
0.29241513 0.28048151 0.26903491 0.25805544 0.24752406 0.23742247
0.22773313 0.21843922 0.20952459 0.20097378 0.19277193 0.18490481
0.17735874 0.17012063 0.16317792 0.15651854 0.15013093 0.14400401
0.13812713 0.13249008 0.12708309 0.12189676 0.11692209 0.11215043
0.10757351 0.10318338 0.09897241 0.09493329 0.09105901 0.08734285
0.08377834 0.0803593  0.07707979 0.07393412 0.07091683 0.06802268
0.06524663 0.06258388 0.0600298  0.05757995 0.05523008 0.05297611
0.05081413 0.04874038 0.04675126 0.04484331 0.04301323 0.04125784
0.03957408 0.03795904 0.03640992 0.03492401 0.03349874 0.03213164
0.03082033 0.02956253 0.02765394 0.02582323 0.02406724 0.02238291
0.02076732 0.01921766 0.01773124 0.01630549 0.01493792 0.01362616
0.01236793 0.01116106]
```



- <u>50 muestras</u>

```
In [10]: %run "suma-convolucion-v1.py" 'no-print'

y(t)
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.05435447 0.1076109  0.15979149 0.21091796 0.2610116  0.31009329
 0.35818346 0.40530215 0.45146898 0.49670317 0.54102358 0.58444865
 0.62699648 0.66868479 0.70953094 0.74955194 0.78876446 0.82718484
 0.86482908 0.90171286 0.93785153 0.97326015 1.00795347 1.04194594
 1.07525172 1.10788468 1.13985841 1.17118622 1.20188118 1.23195605
 1.26142337 1.29029542 1.31858421 1.34630152 1.37345892 1.40006769
 1.42613894 1.45168351 1.47671204 1.50123497 1.5252625  1.54880464
 1.5718712  1.59447179 1.61661581 1.6383125  1.65957088 1.68039981
 1.70080798 1.6664494  1.63278491 1.59980049 1.5674824  1.53581718
 1.50479163 1.47439285 1.44460816 1.41542516 1.38683169 1.35881585
 1.33136597 1.30447061 1.27811858 1.25229888 1.22700079 1.20221374
 1.17792743 1.15413174 1.13081674 1.10797275 1.08559023 1.06365986
 1.04217253 1.02111926 1.00049129 0.98028004 0.96047709 0.94107418
 0.92206323 0.90343633 0.88518572 0.86730379 0.84978311 0.83261636
 0.81579641 0.79931624 0.78316899 0.76734794 0.75184649 0.73665819
 0.72177672 0.70719587 0.69290958 0.67891189 0.66519697 0.65175911
 0.63859271 0.62569229 0.61305247 0.600668   0.58853371 0.57664455
 0.56499557 0.55358191 0.54239882 0.53144164 0.52070582 0.51018687
 0.49988042 0.48978217 0.47988792 0.47019355 0.46069501 0.45138836
 0.44226972 0.43333529 0.42458134 0.41600423 0.40760039 0.39936633
 0.3912986  0.38339385 0.37564878 0.36806018 0.36062487 0.35333977
 0.34620184 0.33920811 0.33235565 0.32564163 0.31906323 0.31261773
 0.30630244 0.30011473 0.29405201 0.28811177 0.28229153 0.27658887
 0.2710014  0.26552682 0.26016282 0.25490719 0.24975773 0.24471229
 0.23976878 0.23492513 0.23017933 0.2255294  0.22097341 0.21650945
 0.21213567 0.20785025 0.2036514  0.19953737 0.19550645 0.19155696
 0.18768726 0.18389573 0.18018079 0.1765409  0.17297454 0.16948022
 0.1660565  0.16270194 0.15941514 0.15619475 0.1530394  0.14994781
 0.14691866 0.14395071 0.14104271 0.13819346 0.13540177 0.13266648
 0.12998644 0.12736054 0.12478769 0.12226681 0.11979686 0.1173768
 0.11500564 0.11268237 0.11040604 0.10817569 0.1059904  0.10384925
 0.10175136 0.09969585 0.09768186 0.09570855 0.09377511 0.09188073
 0.09002462 0.088206   0.08642413 0.08467824 0.08296763 0.08129157
 0.07964938 0.07804035 0.07646383 0.07491916 0.0734057  0.07192281
 0.07046987 0.06904628 0.06765146 0.06628481 0.06494577 0.06363378
 0.06234829 0.06108877 0.0598547  0.05864555 0.05746084 0.05630005
 0.05516271 0.05404835 0.05295651 0.05188671 0.05083853 0.04981153
 0.04880527 0.04781934 0.04685332 0.04590683 0.04497945 0.0440708
 0.04318051 0.04230821 0.04145353 0.04061611 0.03979561 0.03899169
 0.038204   0.03743223 0.03667605 0.03593515 0.03520921 0.03449794
 0.03380103 0.03311821 0.03244918 0.03179366 0.03115138 0.03015585
 0.02918042 0.0282247  0.02728829 0.02637079 0.02547183 0.02459103
 0.02372802 0.02288244 0.02205395 0.02124219 0.02044683 0.01966754
 0.018904   0.01815587 0.01742286 0.01670466 0.01600097 0.01531149
 0.01463594 0.01397403 0.0133255  0.01269007 0.01206748]
```



- En la Versión 1, el array de valores que describen el resultado de la convolución se encuentra, al igual que en el programa de Producto de Convolución, escalados por un diferencial que corresponde a la distancia entre cada valor de la lista, lo cual hace que coincidan con la integral realizada en el análisis manual; mientras que el código de la Versión 2 realiza los cálculos estrictamente como están definidos en la fórmula de la suma de convolución.

  Se observa que, mientras que los resultados de la Versión 1 del código coinciden con los del producto de convolución (más allá de algunos errores eventuales de precisión), los resultados de la Version 2 no lo hacen, limitándose simplemente a adoptar el valor de la suma recursiva de los valores de la función *g* al pasar por cada posición discreta