UNIVERSIDAD DEL NORTE
DEPARTAMENTO DE INGENIERÍAS ELÉCTRICA Y ELECTRÓNICA
Asignatura: Señales y Sistemas. Docente: Carlos Cárdenas Pérez.

Laboratorio 2: Modelado de Sistemas

Grupo : Sebastián Fernández, Sergio Rodriguez, Carlos Rodriguez
{sfernandeza,rodriguezas,rodriguezmendoza} @uninorte.edu.co

15 de Octubre de 2022

Resumen. En el siguiente informe se presenta la propuesta de laboratorio en la que se orienta al estudiante en la generación y operación de convolución para los diferentes tipos de señales teniendo como base principal el lenguaje de programación Python, además de la interfaz web Streamlit acorde a los criterios solicitados en la guía de laboratorio.

1. INTRODUCCIÓN

Es de tener en cuenta que una señal se puede definir como función de una o más variables que representan una cantidad física. Además, estas señales suelen contener información sobre algo en específico, por ejemplo las mediciones de dispositivos eléctricos o ya sea los ciclos cardíacos del ser humano. Ahora bien, esta práctica de laboratorio fue realizada con el fin de analizar e implementar el proceso de generación y convolución de los diferentes tipos de señales (clasificadas en periódicas, deterministas, pares e impares), ya sea en el dominio de tiempo discreto o en el dominio de tiempo continuo.

Teniendo en cuenta lo anteriormente mencionado, el desenvolvimiento de esta práctica se consiguió a partir de las bases teóricas en la generacion, transformacion y convolución de señales que previamente fueron estudiadas, una comprensión en el lenguaje de programación Python, y finalmente haciendo uso de una interfaz gráfica de usuario (Streamlit), en la que de manera interactiva y amigable, el usuario fuese capaz de escoger que tipo de señal con la que se desea trabajar, ya sea como señal de entrada y señal de respuesta al impulso. Este laboratorio fue realizado en Python, ya que de antemano se acordó en utilizar la herramienta de trabajo contraria a la utilizada en el laboratorio previo a este (MATLAB "App Designer").

2. OBJETIVO

- Desarrollar e implementar algoritmos para generar y convolucionar señales discretas y continuas en Python.

3. PROCEDIMIENTOS

3.1 Herramientas

Para la realización de la aplicación en Python se tuvieron en cuenta diversas librerías que nos ayudarían en la significativa reducción de líneas de código para hacerlo más sencillo y eficiente posible los cuales fueron:

- Numpy: Para la ejecución de funciones como la sinusoidal, la exponencial y la cuadrada. También incluye la función convolve la cual es la función que nos ayudará en el cálculo de la convolución. Además ayuda con la inclusión de algunos símbolos como **π.**
- Matplotlib.pyplot : Como su nombre indica se utiliza más que todo para la graficación de las funciones que se requieran implementar, tanto de manera continua como de manera discreta.
- Streamlit: Esta librería se encargaría de todo lo que es nuestra interfaz gráfica, es decir, lo que ve el usuario una vez se ejecuta el código.
- scipy: Incluye la función "Sawtooth" para nuestra señal triangular.

```
import numpy as np
import matplotlib.pyplot as p
import streamlit as st
from scipy import signal
```

Figura .1 sintaxis de las librerías implementadas en Python

3.2 Generación de señales:

Luego se procedió a implementar en el código ciertas señales específicas para luego poder hacer el proceso de convolución. Estas señales fueron:

- Señal sinusoidal.

  Continua:

```
if X_tn=="Sinusoidal":
    A1=st.sidebar.number_input("Ingrese el valor de la amplitud X(k)")
    F1=st.sidebar.number_input("Ingrese el valor de la frecuencia [Hz] X(k)")
    puntoinicio1=st.sidebar.number_input("Ingrese el valor del punto de inicio X(k)")
    puntofinal1=st.sidebar.number_input("Ingrese el valor del punto final X(k)")
    t1=np.arange(puntoinicio1,puntofinal1,0.001)
    x1= A1*np.sin(2*np.pi*F1*t1)
```

Figura 1. Código de la señal Sinusoidal Continua.

  Discreta:

```
if X_tn=="Sinusoidal":
    A1=st.sidebar.number_input("Ingrese el valor de la amplitud 1 sin")
    W1=st.sidebar.number_input("Ingrese el valor de la frecuencia [rad/s] 1 sin")
    puntoinicio1=int(st.sidebar.number_input("Ingrese el valor del punto de inicio (numero entero) 1 sin"))
    puntofinal1=st.sidebar.number_input("Ingrese el valor del punto final (numero entero) 1 sin")
    vx1=np.arange((puntofinal1-puntoinicio1)/0.999999)
    n1=np.arange(puntoinicio1,puntofinal1,0.999999)
    c1=int(puntofinal1-puntoinicio1+1)
    i1=puntoinicio1
    for i in range(puntoinicio1,c1,1):
        vx1[i1]= A1*np.sin(W1*i1)
        i1=i1+1
```

Figura 5. Código de la señal sinusoidal Discreta.

- Señal exponencial.

  Continua:

```
if X_tn=="Exponencial":
    A1=st.sidebar.number_input("Ingrese el valor del intercepto de la función X(k) ")
    B1=st.sidebar.number_input("Ingrese el factor de decrecimiento X(k)")
    puntoinicio1=st.sidebar.number_input("Ingrese el valor del punto de inicio X(k)")
    puntofinal1=st.sidebar.number_input("Ingrese el valor del punto final X(k)")
    t1=np.arange(puntoinicio1,puntofinal1,0.001)
    x1=A1*(np.exp(-B1*t1))
```

Figura 3. Código de la señal exponencial Continua.

  Discreta:

```
if X_tn=="Exponencial":
    b1=st.sidebar.number_input("Ingree el factor de decrecimiento b 1 exp")
    A1=st.sidebar.number_input("Ingree el valor del intercepto de la función A 1 exp")
    puntoinicio1=int(st.sidebar.number_input("Ingree el valor del punto de inicio 1 exp"))
    puntofinal1=st.sidebar.number_input("Ingree el valor del punto final 1 exp")
    vx1=np.arange((puntofinal1-puntoinicio1)/0.999999)
    n1=np.arange(puntoinicio1,puntofinal1,0.999999)
    c1=int(puntofinal1-puntoinicio1+1)
    i1=puntoinicio1
    for i in range(puntoinicio1,c1,1):

        vx1[i1]= A1*(np.exp(-b1*i1))
        i1=i1+1
```

Figura 4. Código de la señal exponencial Discreta.

- Señal triangular.

  Continua:

```
if X_tn=="Triangular":
    A1=st.sidebar.number_input("Ingrese el valor de la amplitud X(k)")
    F1=st.sidebar.number_input("Ingrese el valor de la frecuencia [Hz] X(k)")
    puntoinicio1=st.sidebar.number_input("Ingrese el valor del punto de inicio X(k)")
    puntofinal1=st.sidebar.number_input("Ingrese el valor del punto final X(k)")
    t1=np.arange(puntoinicio1,puntofinal1,0.001)
    x1=A1*signal.sawtooth(2*np.pi*F1*t1,0.5)
```

Figura 5. Código de la señal triangular Continua.

  Discreta:

```
if X_tn=="Triangular":
    A1=st.sidebar.number_input("Ingree el valor de la amplitud 1 tri")
    W1=st.sidebar.number_input("Ingree el valor de la frecuencia [Hz] 1 tri ")
    puntoinicio1=int(st.sidebar.number_input("Ingree el valor del punto de inicio 1 tri"))
    puntofinal1=st.sidebar.number_input("Ingree el valor del punto final 1 tri")
    vx1=np.arange((puntofinal1-puntoinicio1)/0.999999)
    n1=np.arange(puntoinicio1,puntofinal1,0.999999)
    c1=int(puntofinal1-puntoinicio1+1)
    i1=puntoinicio1
    for i in range(puntoinicio1,c1,1):

        vx1[i1]= A1*signal.sawtooth(W1*i1,0.5)
        i1=i1+1
```

Figura 6. Código de la señal triangular Discreta.

- Señal rectangular.

  Continua:

```
if X_tn=="Rectangular":
    A1=st.sidebar.number_input("Ingrese el valor de la amplitud X(k)")
    F1=st.sidebar.number_input("Ingrese el valor de la frecuencia [Hz] X(k)")
    puntoinicio1=st.sidebar.number_input("Ingrese el valor del punto de inicio X(k)")
    puntofinal1=st.sidebar.number_input("Ingrese el valor del punto final X(k)")
    t1=np.arange(puntoinicio1,puntofinal1,0.001)
    x1=A1*signal.square(2*np.pi*F1*t1)
```

Figura 7. Código de la señal rectangular Continua.

  Discreta:

```
if X_tn=="Rectangular":
    A1=st.sidebar.number_input("Ingree el valor de la amplitud 1 rec")
    W1=st.sidebar.number_input("Ingree el valor de la frecuencia [Hz] 1 rec")
    puntoinicio1=int(st.sidebar.number_input("Ingree el valor del punto de inicio 1 rec"))
    puntofinal1=st.sidebar.number_input("Ingree el valor del punto final 1 rec")
    vx1=np.arange((puntofinal1-puntoinicio1)/0.999999)
    n1=np.arange(puntoinicio1,puntofinal1,0.999999)
    c1=int(puntofinal1-puntoinicio1+1)
    i1=puntoinicio1
    for i in range(puntoinicio1,c1,1):

        vx1[i1]= A1*signal.square(W1*i1)
        i1=i1+1
```

Figura 8. Código de la señal rectangular Discreta.

- Rampa_1.

  Continua:

```
if X_tn=="Rampa_1":
    rampa=1
    ui=st.number_input("Digite el intervalo de tiempo X(k)")
    di=ui
    ai=0
    bi=(di/3)*1
    ci=(di/3)*2

    def trameo1 (xi):
        return 0

    def trameo2 (xi):
        return xi-bi

    def trameo3 (xi):
        return bi

    xi=np.arange(ai,di,0.01)
    yi=np.piecewise(xi,[(ai<=xi) & (xi<bi), (bi<=xi) & (xi<=ci), (ci<xi) & (xi<=di)],[lambda xi:trameo1(xi), lambda xi:trameo2(xi), lambda xi:trameo3(xi)])
    trameo1=np.vectorize(trameo1)
    trameo2=np.vectorize(trameo2)
    trameo3=np.vectorize(trameo3)
```

Figura 9. Código de la señal Rampa_1 Continua.

Discreta:

Figura 10. Código de la señal Rampa_1 Discreta.

- Rampa_2.

Continua:

Figura 11. Código de la señal Rampa_2 Continua.

Discreta:

Figura 12. Código de la señal Rampa_2 Discreta.

- Rampa_3.

Continua:

Figura 13. Código de la señal Rampa_3 Continua.

Discreta:

Figura 14. Código de la señal Rampa_3 Discreta.

Recordemos que necesitamos crear una GUI para que el usuario pueda elegir libremente los parámetros que quiera. Esto también incluye los valores que asumen estas señales. Con base a esto, entonces debemos implementar una forma en la que el usuario pueda ingresar los valores de "punto de inicio" y "punto final" para cualquier función que este escoja; Adicionalmente otros parámetros como la amplitud y la frecuencia para las señales periódicas (Sinusoidal, Triangular, Rectangular); el factor de de-crecimiento para la función exponencial; Y el intervalo de tiempo para las señales rampa.

Para el caso de las señales con denominación rampa, no es necesario que a estas se les pidan la amplitud, pero se debe ingresar el intervalo de tiempo. Esto es debido a que estas señales rampa están constituidas por tres tramos las cuales están divididas proporcionalmente y esta no se debe modificar. Estas divisiones se pueden ver reflejadas con distintos colores, como se pueden ver en las Figuras 1, 2 y 3 respectivamente. Otro detalle a tener en cuenta es que el valor del exponente de la señal exponencial debe ser el parámetro a modificar.

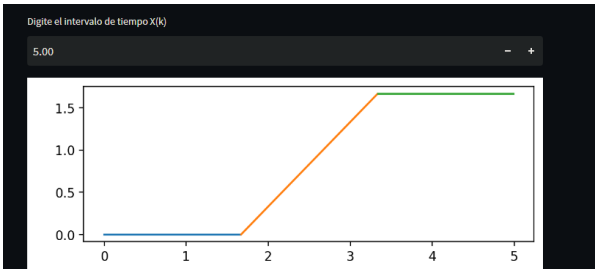

Figura 15. Señal Rampa_1.

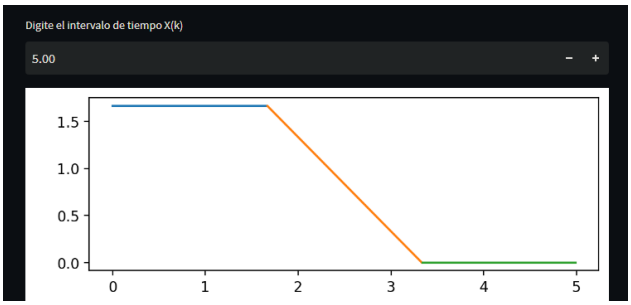

Figura 16. Señal Rampa_2.

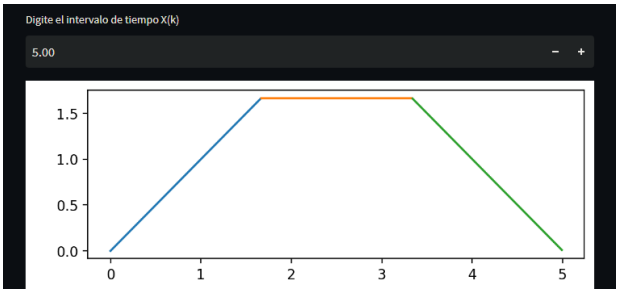

Figura 17. Señal Rampa_3.

Luego se hace uso de la librería antes mencionada "Streamlit". Streamlit lo que hace es convertir distintas lineas de código en una pagina web. Conjunto a esto el programa para escribir el código es "Visual Studio Code" dentro de "Anaconda

Navigator". Utilizando el "Anaconda Prompt", fácilmente se puede correr el código para activar la pagina web y de esa forma editar en tiempo real y ver las modificaciones del codigo.

Para la generación de las señales, se realizó utilizando el condicional IF. Ya previamente definido las distintas opciones para el cuadro de selecciones, estás siendo de selección de tipo de señal (Continua o Discreta) y la señal de entrada y salida, se usa el condicional IF para la selección del tipo de señal. Entonces, si la señal es continua, de igual manera se usa el mismo condicional para distinguir si las señales son de entrada o de salida. Es decir, que para cada señal existen dos condicionales para esta. Lo cual significa que en realidad existen cuatro condiciones para una sola señal, ya que este mismo procedimiento se repite cuando la señal es de tipo discreta. Se hizo de esta manera ya que para los ojos de los diseñadores, esta resulta menos confusa de entender. La diferencia más notable entre las condicionales de señal continua y discreta es el uso de la función "stem" para la generar las señales discretas y la función ""plot" para la generación de funciones continuas.

Ya teniendo esto configurado correctamente se procede a escribir la convolución.

3.3 Convolución

Para la generación de la señal convolucionada se requiere que anteriormente el usuario haya ingresado dos funciones X[n] y H[n] ó X[t] y H[t] con todos sus parámetros. Luego de haber establecido ambas señales se procede con el proceso de convolución, para ello nos apoyamos de una de las funciones de la librería "numpy" antes mencionada, la cual sería la función "convolve". A Pesar de no haber podido lograr con mucho éxito el proceso de nuestra convolución al momento de la sustentación, posteriormente y de manera independiente se logró la realización de un código funcional el cual calcula el proceso de convolución, la gráfica y realiza una animación, tanto como para señales continuas como discretas. El código implementado fue el siguiente:

```
if tiposeñal=="Discreta":

    y_conv=np.convolve()

    y_conv = np.convolve(vx,vh, mode='full')*r

    y_conv=np.resize(y_conv, np.shape(ty))

    hs = vh[::-1]
    tm = np.arange(xi1-(xf2-xi2),xi1,r)
    aniy = np.zeros(len(y_conv))
    frames = max(np.size(t1),np.size(t2))
    factor = len(ty)/frames
    factor = math.floor(factor)
    factor_t = (max(ty)-min(ty))/frames

    vx =np.resize(vx, np.shape(t1))
    hs =np.resize(hs,np.shape(tm))
    for i in range(frames):
        aniy[:i*factor] = y_conv[:i*factor]
        grafica3.clear()
        grafica3.stem(tm,hs,linefmt='g',basefmt="purple")
        grafica3.stem(t1,vx)
        grafica3.axis(xmin=xi1-(xf2-xi2),xmax=xf1+(xf2-xi2))
        grafica4.clear()
        grafica4.stem(ty,aniy)
        tm = tm + factor_t
        time.sleep(0.01)
        grafica3.legend(['h(t)','x(t)'])
        st_a.pyplot(plt)
```

Figura 18. Código de la convolución Discreta.

Para señales Discretas.

```
if tiposeñal == "continua":
    y_conv = np.convolve(x1,x2, mode='full')*0.001
    grafica3.plot(t1,y_conv)
    y_conv = np.resize(y_conv, np.shape(ty))
    hs =x2[::-1]
    tm = np.arange(puntoinicio1-(puntofinal2-puntoinicio2),puntoinicio1+0.001,0.001)
    aniy = np.zeros(len(y_conv))
    frames = 30
    factor = len(ty)/frames
    factor = math.floor(factor)
    factor_t = (max(ty)-min(ty))/frames
    x1 =np.resize(x1, np.shape(t1))
    hs =np.resize(hs,np.shape(tm))
    for i in range(frames+2):
        aniy[:i*factor] = y_conv[:i*factor]
        grafica3.clear()
        grafica3.plot(tm,hs,t1,x1)
        grafica3.axis(xmin=puntoinicio1-(puntofinal2-puntoinicio2),xmax=puntofinal1+(puntofinal2-puntoinicio2))
        grafica4.clear()
        grafica4.plot(ty,aniy)
        tm = tm + factor_t
        time.sleep(0.01)
        grafica3.legend(['h(t)','x(t)'])
    st_a.pyplot(plt)
```

Figura 19. Código de la convolución Continua.

4. ANÁLISIS DE RESULTADOS

En este apartado se analizan las pruebas realizadas de manera óptima en el algoritmo de Python.

En primera instancia se inició el código realizado en Python para verificar el buen funcionamiento de las señales en el dominio de tiempo discreto y en el dominio de tiempo continuo. Se puede observar en las siguientes figuras.

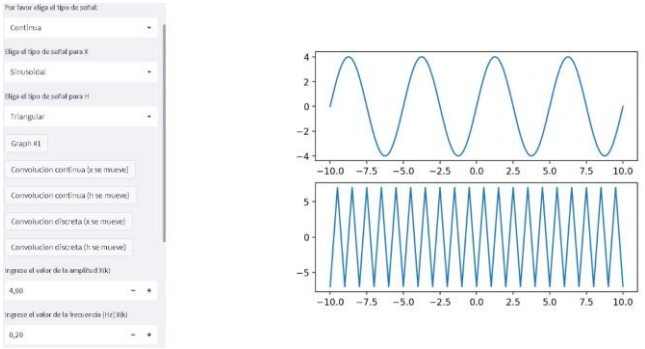

Figura 20. Ejemplo de una señal sinusoidal y una señal triangular en el dominio continuo.

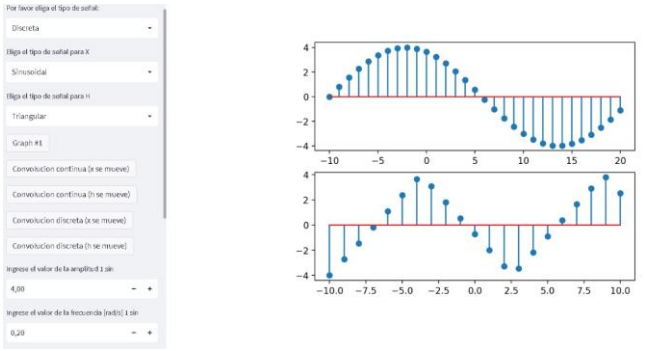

Figura 21. Ejemplo de una señal sinusoidal y una señal triangular en el dominio discreto.

Como se puede observar en las figuras 20 y 21 se logró ejecutar con éxito las señales solicitadas por el usuario con sus respectivos parámetros.

Una vez fueron generadas las señales, se pasó a realizar la convolución de estas en los diferentes dominios de tiempo. Observar figuras 22 y 23.

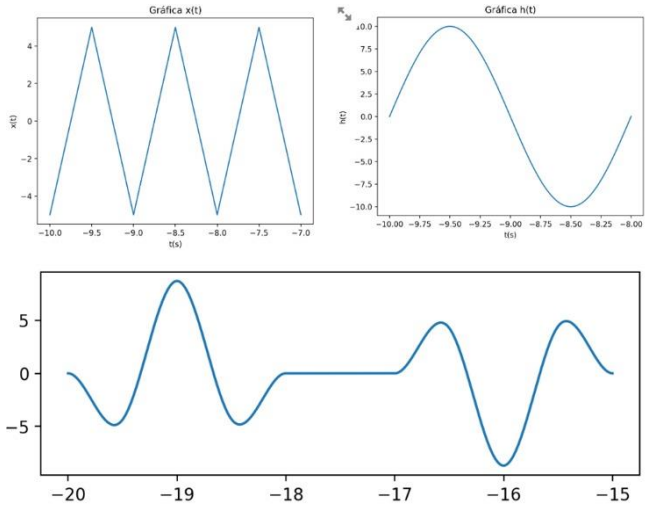

Figura 22. Ejemplo de una convolución entre una señal triangular y una señal sinusoidal continua.

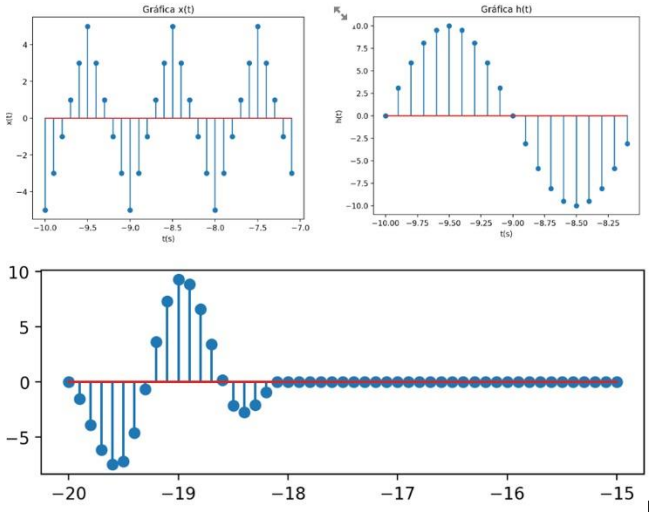

Figura 23. Ejemplo de una convolución entre una señal triangular y una señal sinusoidal discreta.

Nuevamente como se puede observar, se consiguió graficar con éxito la convolución de dichas señales, obteniendo como resultado una señal resultante que puede ser analizada para el caso que sea necesario.

5. CONCLUSIONES

Para concluir se puede decir que a pesar de no haber podido terminar el código por completo, de manera autónoma, se pudo realizar con código que funciona y lograr finalmente el cometido de este laboratorio el cual fue la comprensión de la convolución de dos señales tanto de manera matemática como de manera visual, todo esto gracias a la GUI desarrollada en Python. Adicionalmente podemos destacar el gran aprendizaje que se tuvo con respecto a lo que programar respecta, debido a la cantidad de funciones, librerías, sintaxis nueva que agrega Python a la programación. Finalmente podemos decir que este error solo fue un peldaño para poder lograr nuestro cometido y que a pesar de no haber obtenido buenos resultados en su momento, se logró superar este obstáculo para poder llegar al nivel esperado.

6. REFERENCIAS

[1] Pablo, T. P. J. (2017). Introducción a Las señales y sistemas. Universidad del Norte.