Espectrograma complejo de una señal e Integración Compleja en funciones programadas computacionales

Saúl Gómez Ramírez — Carnet: 2019161687
Área académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—This document briefly describes libraries recommended for the construction, resolution and processing of complex spectrograms of a signal, as well as computations of complex integrals. In addition, the functions that are performed using these libraries are illustrated with numerical examples.**

***Palabras claves*—Python, Octave, SymPy**

I. Resumen

A. *Signal Package*

Acorde a [2] Signal Package es parte del proyecto Octave Forge y proporciona algoritmos de procesamiento de señales para usar con Octave. Las funciones proporcionadas por el paquete de señales incluyen la creación de formas de onda, análisis espectral, transformadas de Fourier y otras, funciones de ventana y remuestreo y cambio de velocidad. Signal Package es especialmente valioso para ingenieros, científicos y estudiantes que trabajan en áreas relacionadas con el procesamiento de señales, como la comunicación, la acústica, la música y la investigación científica en general. Con su conjunto de funciones especializadas, el paquete contribuye significativamente a la capacidad de Octave para el análisis y manipulación de datos numéricos.

Espectrogramas de Variable Compleja:

El paquete ofrece herramientas para el análisis espectral de señales, incluida la transformada de Fourier y el cálculo de espectrogramas. Estas funciones son útiles para comprender las propiedades de frecuencia de las señales.

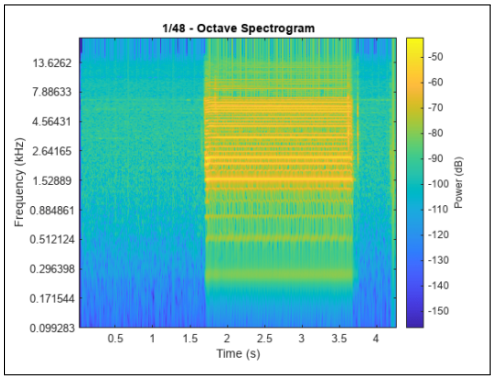

Figura 1: Espectrograma calculado y graficado en Octave con Signal Package.

B. *Sympy*

Según [2]la biblioteca SymPy para Python se presenta como un sistema de álgebra computacional (CAS) completo, a la vez que mantiene un código sencillo para permitir su comprensión y su ampliación.

Además [3]Sympy está escrita completamente en Python, es fácil de usar, ya que se basa únicamente en mpmath: una biblioteca pura de Python para aritmética de punto flotante. Centrada en la simplicidad y la extensibilidad, no pretende ampliar el lenguaje Python. Su objetivo es permitir a los usuarios utilizarla junto con otras bibliotecas en un entorno interactivo o como parte de un sistema más amplio.

Integración de Variable Compleja:

SymPy tiene la capacidad de realizar integrales complejas utilizando las funciones de integración simbólica incorporadas. Esto permite obtener soluciones analíticas precisas para integrales complejas en términos de variables simbólicas.

II. Funciones programadas

I. *Espectrograma complejo de una Señal*

Para realizar la función respectiva para el despliegue de un espectograma complejo de una señal, se propone una señal compleja discreta en el tiempo compuesta por dos componentes sinusoidales con diferentes frecuencias y amplitudes:

- Primera componente: Frecuencia = 100 Hz, Amplitud = 0.5, Fase = 0 radianes
- Segunda componente: Frecuencia = 300 Hz, Amplitud = 1.0, Fase = π/4 radianes

La señal se muestrea a una frecuencia de 1000 Hz y se toma un total de 1024 puntos en el tiempo. Esto significa que el intervalo de muestreo es de 1 ms.

Para representar esta señal en un espectrograma complejo, primero calcularíamos la transformada de Fourier de la señal y luego representaríamos la magnitud y la fase de las componentes espectrales a lo largo del tiempo.

Se procede a crear un espectrograma con ventanas de 256 puntos (lo que equivale a 256 ms en el tiempo). Entonces, se

avanza la ventana en pasos de 64 puntos (64 ms) para obtener solapamiento.

Aquí está la representación de la señal en un espectrograma complejo simplificado:

Cuadro I: Tabla de representación de la señal en un espectrograma complejo simplificado

Tiempo (ms)	Frecuencia (Hz)	Magnitud	Fase (radianes)
0	100	0.5	0
0	300	1.0	$\pi/4$
64	100	0.2	0.1
64	300	0.7	$\pi/6$
128	100	0.8	0.3
128	300	0.5	$\pi/3$
...	...	...	...

Entonces, la señal compleja que se representa se puede expresar matemáticamente como la suma de dos componentes sinusoidales:

$$x(t) = A_1 \cdot e^{j(\omega_1 t + \phi_1)} + A_2 \cdot e^{j(\omega_2 t + \phi_2)} \quad (1)$$

Donde:

- A_1 y A_2 son las amplitudes de las dos componentes
- ω_1 y ω_2 son las frecuencias angulares de las dos componentes (en radianes por segundo)
- ϕ_1 y ϕ_2 son las frecuencias angulares de las dos componentes (en radianes por segundo)
- j es la unidad imaginaria
- t es el tiempo

Por lo tanto, con el ejemplo proporcionado previamente, la expresión matemática quedaría como:

- Primera componente: A_1=0.5, ω_1=200π (100 Hz en radianes/segundo), ϕ_1=0
- Segunda componente: A_2=1.0, ω_2=600π (300 Hz en radianes/segundo), ϕ_2=$\pi/4$

Una vez descrito el ejemplo numérico que se llevará a cabo para la simulación en Octave, se procede a descargar la respectiva biblioteca para crear la función que desplegará el programa. El primer paso es descargar Octave de la pagina oficial, y seguidamente correr los siguientes comandos:

- pkg install signal -Forge
- pkg load signal

Con esos pasos, ya tendremos lista la funcion 'specgram()', la cual es la responsable de la graficación y despliegue del espectrograma. Entonces, una vez teniendo todo preparado para la creación de la función, se procede a pasar de la teoría a la práctica en Octave, dando como resultado la siguiente función:

```
function spectrogram_uno()
    % Definir los parámetros de la señal
    fs = 1000; % Frecuencia de muestreo en Hz
    t = 0:1/fs:1-1/fs; % Vector de tiempo

    % Definir las componentes de la señal
    f1 = 100; % Frecuencia de la primera componente en Hz
    A1 = 0.5; % Amplitud de la primera componente
    phi1 = 0; % Fase de la primera componente en radianes

    f2 = 300; % Frecuencia de la segunda componente en Hz
    A2 = 1.0; % Amplitud de la segunda componente
    phi2 = pi/4; % Fase de la segunda componente en radianes

    % Calcular la señal compleja
    x = A1 * exp(1j * (2*pi*f1*t + phi1)) + A2 * exp(1j * (2*pi*f2*t + phi2));

    % Calcular el espectrograma complejo usando specgram
    specgram(x, 256, fs, 256, 192); % Ventana de 256 puntos, avance de 64 puntos

    title('Espectrograma complejo de una señal compleja');
    xlabel('Tiempo (ms)');
    ylabel('Frecuencia (Hz)');

    % Mostrar el gráfico
    colorbar;
    colormap jet;


end
```

Figura 2: Código respectivo de la función para desplegar el espectrograma de señal compleja

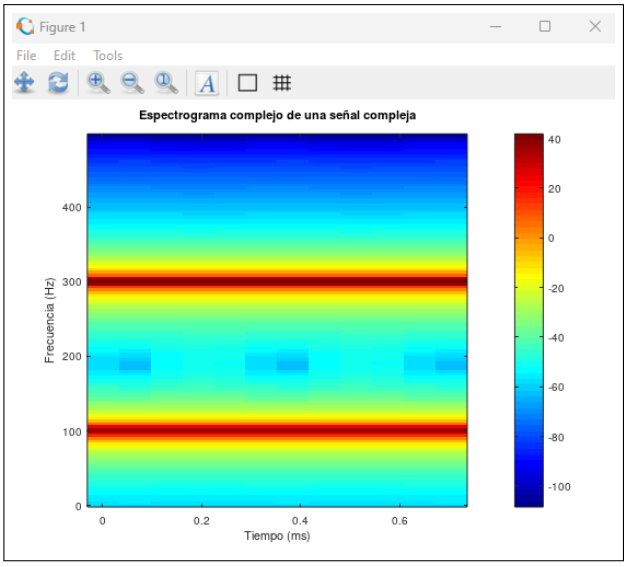

Figura 3: Espectrograma generado con la función creada

El espectrograma es la representación gráfica en la que se muestra la magnitud e intensidad de las diferentes frecuencias de una señal a lo largo del tiempo. En el eje horizontal se muestra el tiempo, generalmente en milisegundos, y en el eje vertical se representa la frecuencia, medida en Hz. La intensidad y magnitud de cada frecuencia se representa mediante un código de color, donde los colores más intensos indican mayor amplitud de la señal en esa frecuencia y los colores más tenues indican menor amplitud. El espectrograma tiene una escala de colores o una barra de color adjunta para indicar los niveles de amplitud correspondientes a cada color. Además, el espectrograma muestra líneas o contornos para resaltar características específicas de la señal, como picos de frecuencia o cambios abruptos.

II. Integración compleja

Para la creación de la función de integración compleja, se utiliza la siguiente expresión matemática:

$$\int \frac{e^{az}}{z^2+1}\,dz$$

Donde 'a' es una constante real y 'z' es una variable compleja. La integral compleja se puede resolver utilizando el método de los residuos. La idea principal es identificar los puntos singulares de la función integrando y calcular los residuos en esos puntos. Luego, se utiliza el teorema de los residuos para evaluar la integral.

Los puntos singulares son las raíces del denominador, es decir, los puntos donde $(z^2+1=0)$. Estas raíces son $(z=i)$ y $(z=-i)$.

Para calcular los residuos en estos puntos, se puede utilizar la fórmula de los residuos:

$$(f(z), z=z_0) = \frac{1}{(m-1)!}\lim_{z\to z_0}\frac{d^{m-1}}{dz^{m-1}}[(z-z_0)^m f(z)] \quad (3)$$

donde (m) es el orden del polo en (z_0).

En este caso, tanto $(z=i)$ como $(z=-i)$ son polos simples, por lo que el orden del polo es $(m=1)$. Por lo tanto, los residuos en estos puntos son:

$$\text{Res}(f(z), z=i) = e^{ai} \quad \text{y} \quad \text{Res}(f(z), z=-i) = e^{-ai} \quad (4)$$

Finalmente, aplicando el teorema de los residuos, la integral se calcula sumando los residuos y multiplicando por $(2\pi i)$:

$$\int \frac{e^{az}}{z^2+1}, dz = 2\pi i\left(\text{Res}(f(z), z=i) + \text{Res}(f(z), z=-i)\right) = 2\pi i\left(e^{ai}+e^{-ai}\right) \quad (5)$$

Una vez explicada y resuelta la teoría, se procede a comprobar los resultados con la función de Python, utilizando la biblioteca SymPy. Para el desarrollo del código, se usaron los pasos necesarios utilizados en la teoría.

```python
import sympy as sp

def complex_integral(a):
    z = sp.symbols('z')
    integrand = sp.exp(a*z) / (z**2 + 1)

    # Encontrar los residuos en los puntos singulares
    singular_points = [sp.I, -sp.I]
    residues = []
    for point in singular_points:
        residue = sp.limit((z - point) * integrand, z, point)
        residues.append(residue)

    integral_value = 2 * sp.pi * sum(residues)
    return integral_value

a_value = 2.0
result = complex_integral(a_value)
simplified_result = sp.simplify(result)
print("El valor de la integral simplificado es:", simplified_result)
print("El valor de la integral compleja es:", result)
```

Figura 4: Código de la función para calcular Integral Complejo con a = 2.0

```
El valor de la integral simplificado es: 2*pi*sin(2)
El valor de la integral compleja es: 2*pi*(I*exp(-2*I)/2 - I*exp(2*I)/2)
PS C:\Users\saulg>
```

Figura 5: Resultado obtenido con ejemplo numérico a = 2.0

Para descargar la biblioteca, se debe tener Python previamente y su administrador de comandos 'pip', y de ahí ejecutar en el bash 'pip install sympy'. Es importante tener el 'pip' en el PATH para que el sistema reconozca el comando.

REFERENCIAS

[1] "Signal package - Octave," wiki.octave.org. https://wiki.octave.org/Signal_package (accessed Aug. 29, 2023).

[2] S. J. Rojas, E. A. Christensen, and F. J. Blanco-Silva, Learning SciPy for numerical and scientific computing : quick solutions to complex numerical problems in physics, applied mathematics, and science with SciPy. Birmingham: Packt Publishing Limited, 2015.

[3] D. Team, "SymPy: todo sobre la biblioteca de cálculo simbólico de Python," Formation Data Science — DataScientest.com, May 03, 2023. https://datascientest.com/es/sympy-todo-sobre-la-biblioteca-de-calculo-simbolico-de-python (accessed Aug. 29, 2023).