# Analizador de espectros utilizando radio definido por software

Jorge Alberto Chavez Ponce, Neyra Torres Luis Kenny, Javier Cana Remache, Rosangela Yllachura Arapa
*Universidad Nacional de San Agustín*
*Escuela Profesional de Ingenieria de Telecomunicaciones*

En este laboratorio utilizaremos la Transformada Discreta de Fourier (DFT), trabajaremos con hardware de radio real para grabar señales de radio y comprender algunas de las peculiaridades de este dispositivo. Por último, implementaremos un espectrógrafo sencillo para visualizar señales tanto en frecuencia como en tiempo. El objetivo es desarrollar una comprensión de la capa física en la comunicación inalámbrica.

La DFT de una señal $x$ de longitud $N$ se expresa como:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi kn}{N}}$$

Donde $X[k]$ es el componente de frecuencia de $x$ a la frecuencia normalizada $\frac{2\pi k}{N}$ radianes por muestra. La DFT es invertible, y para recuperar $x$, basta con aplicar la transformada discreta de Fourier inversa (IDFT).

***Index Terms*—Analizador de Espectros, Demodulación, Frecuencias de radio, Transformada de Fourier, SDR, Ruido de fondo, Procesamiento de señales.**

## I. Introducción

La Transformada Discreta de Fourier (DFT) es una herramienta esencial en el procesamiento digital de señales, ya que permite descomponer una señal en sus componentes de frecuencia. En el dominio digital, cada componente de frecuencia se asocia a un índice específico en la señal transformada.

La señal transformada $X$, obtenida mediante la DFT, tiene una longitud $N$, y el valor en el índice $k$ representa la componente de frecuencia de $x$ a una frecuencia normalizada de $\frac{2\pi k}{N}$ radianes por muestra. Esto implica que cada elemento de $X$ es el resultado de multiplicar la señal original por un tono correspondiente a esa frecuencia, y luego sumar los productos obtenidos.

## II. Transformada Discreta de Fourier (DFT)

### A. Desarrollo del algoritmo

a) Desarrolle un algoritmo que permita calcular la transformada discreta de Fourier de cualquier señal $x$. La DFT mide los componentes de frecuencia de una señal en el dominio del tiempo en frecuencias igualmente espaciadas entre 0 y $2\pi$ radianes por muestra.

```
def dft(signal):
    N = len(signal)
    dft_output = np.zeros(N, dtype=complex)
    for k in range(N):
        for n in range(N):
            dft_output[k] += signal[n] * np.exp
                (-2j * np.pi * k * n / N)

    return dft_output
```

En el algoritmo presentado, se desarrolla la implementación de la Transformada Discreta de Fourier (DFT) utilizando un enfoque directo. Este algoritmo calcula los componentes de frecuencia de una señal $x$ en el dominio del tiempo, analizando frecuencias igualmente espaciadas entre 0 y $2\pi$ radianes por muestra. Se utilizan dos bucles anidados: el externo recorre las frecuencias $k$ y el interno recorre las muestras $n$ de la señal, acumulando las contribuciones de cada muestra para la frecuencia correspondiente. Aunque este método es computacionalmente intensivo con una complejidad temporal de $O(N^2)$, es adecuado para señales de longitud corta y proporciona una implementación didáctica y directa del concepto de la DFT.

## III. Entendiendo la Frecuencia

Considere la siguiente señal que consiste solo en una frecuencia.

$$\exp(1j \times 2 \times \pi \times \left(\frac{2.5}{128}\right) \times t), \quad t \in [0:128]$$

Si se toma la DFT de esta señal, la frecuencia estará entre dos intervalos.

- **a) Grafique la DFT de esta señal (La DFT es compleja, solo grafique la magnitud de esta señal):**
  - **Código:** Se calculó la DFT de la señal utilizando la fórmula de la DFT discreta y se graficó la magnitud. El código empleado fue el siguiente:

```
dft_signal = dft(signal)
plt.stem(np.abs(dft_signal))
plt.title("DFT de la se al")
plt.xlabel("Frecuencia")
plt.ylabel("Magnitud")
plt.show()
```

– **Interpretación:** La señal original es una exponencial compleja con una frecuencia de $2.5/128$. Al calcular su DFT y graficar la magnitud, observamos un pico prominente en la frecuencia correspondiente (alrededor de la posición $k = 2.5$). Esto indica que la señal está compuesta principalmente por una única frecuencia dominante. El resto de los componentes de la DFT son cercanos a cero, lo que refleja que no hay otras frecuencias presentes en la señal. En la figura 1 podemos ver el resultado del codigo implementado.

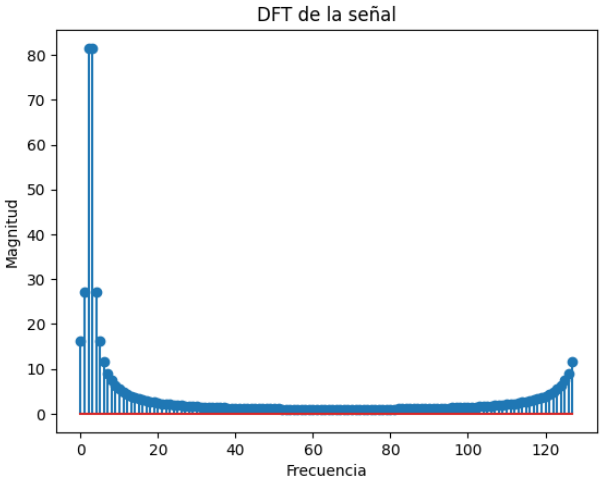


Fig. 1. DFT de la señal

- **b) Doble el tamaño de la señal colocando zeros al final y grafique su DFT:**
  – **Código:** Se duplicó el tamaño de la señal añadiendo ceros al final y se calculó nuevamente la DFT:

```
zero_padded_signal = np.
    concatenate((signal, np.
    zeros(128)))
dft_zero_padded_signal = dft(
    zero_padded_signal)
plt.stem(np.abs(
    dft_zero_padded_signal))
plt.title("DFT de la se al con
    ceros")
plt.xlabel("Frecuencia")
plt.ylabel("Magnitud")
plt.show()
```

  – **Interpretación:** El gráfico de la DFT de la señal con padding muestra un mayor número de puntos en el eje de frecuencia, pero los picos principales permanecen en las mismas frecuencias que en la señal original, particularmente en $k = 2.5$. El relleno de ceros no introduce nuevas frecuencias, pero incrementa la resolución de la DFT. Esto permite una mayor precisión en la visualización de las frecuencias, aunque no añada nueva información frecuencial. En la figura 2 podemos ver el resultado del codigo implementado.

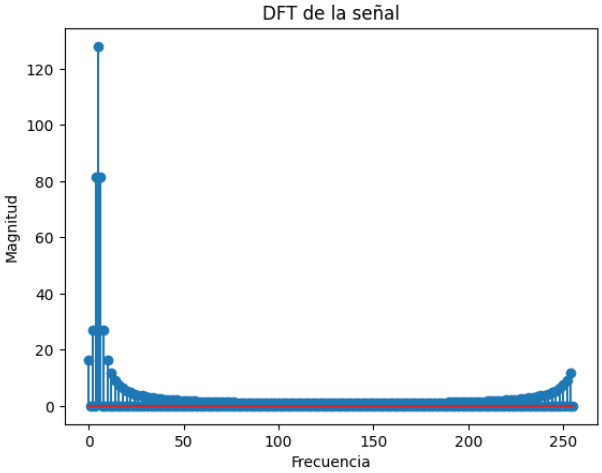


Fig. 2. DFT de la señal con ceros

- **c) Comparación entre la DFT original y la DFT con ceros:**
  – **Código:** Se comparan las dos DFT graficándolas en el mismo gráfico:

```
plt.stem(np.abs(dft_signal),
    linefmt='b-', markerfmt='bo
    ', basefmt='r-', label='DFT
    original')
plt.stem(np.abs(
    dft_zero_padded_signal),
    linefmt='g--', markerfmt='
    go', basefmt='b-', label='
    DFT con ceros')
plt.title("Comparaci n entre
    la DFT original y la DFT
    con ceros")
plt.xlabel("Frecuencia")
plt.ylabel("Magnitud")
plt.legend()
plt.show()
```

  – **Interpretación:** Las frecuencias que coinciden entre ambas DFT son aquellas correspondientes a los primeros $k$ puntos (hasta $k = 127$). Esto se debe a que el relleno con ceros no afecta las frecuencias de la señal original. Las frecuencias adicionales de la DFT con ceros solo incrementan la resolución en el dominio de la frecuencia, pero no aportan nuevas componentes frecuenciales. En la figura 3 podemos ver el resultado del codigo implementado.

## IV. Ajustes Finos de Frecuencia

- **a) Escriba una función que calcule el "bin" (intervalo) de frecuencia para una frecuencia arbitraria en una señal.**

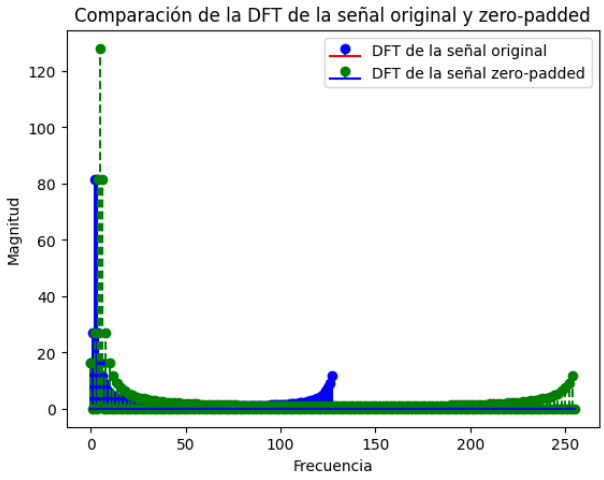


Fig. 3. DFT de la señal con ceros

Para resolver esta pregunta, hemos escrito la siguiente función en Python que calcula el componente de frecuencia para un índice arbitrario $k$ en una señal discreta:

```
def get_freq_component(signal, k):
N = len(signal)
result = 0
for n in range(N):
    result += signal[n] * np.exp(-2
        j * np.pi * k * n / N)
return result
```

La función toma dos entradas:

- `signal`: la señal discreta en el dominio del tiempo,
- `k`: el índice de frecuencia (puede ser fraccionario).

El código realiza una suma de los valores de la señal multiplicados por un término exponencial complejo, que depende del índice $k$.

Para ilustrar su funcionamiento, hemos generado una señal discreta definida como una exponencial compleja. Esta señal tiene una frecuencia normalizada de $2.5/128$. Calculamos los componentes de frecuencia para tres valores arbitrarios de $k$: 0.123, 0.555 y 0.999. Los resultados obtenidos fueron los siguientes:

- Para $k = 0.123$, el resultado es:

$$6.834732696442442 + 14.340484976955768i$$

- Para $k = 0.555$, el resultado es:

$$-3.515668926161199 + 0.7881103904122 62i$$

- Para $k = 0.999$, el resultado es:

$$0.9147527912601011 + 27.1349400407594 57i$$

Estos resultados corresponden a los componentes de frecuencia de la señal en los índices $k$ seleccionados, calculados mediante la transformada discreta de Fourier (DFT) generalizada para valores fraccionarios de $k$.

## V. Filtrado Paso Bajo con la DFT

A continuación, implementaremos un filtro pasa-bajas utilizando la DFT. El siguiente código implementa el filtro pasa-bajas:

```
def low_pass_filter(signal, cutoff):
signal_dft = np.fft.fft(signal)
N = len(signal_dft)
# Aplicar el filtro low-pass: frecuencias
    mayores al cutoff se reemplazan por
    cero
for k in range(cutoff, N - cutoff):
    signal_dft[k] = 0
# Reconstruir la se al filtrada
filtered_signal = np.fft.ifft(signal_dft)
return filtered_signal

# Se al de prueba: se al con frecuencia
    de 5 Hz + ruido de 20 Hz
test_signal = np.sin(2 * np.pi * 5 * np.
    linspace(0, 1, 128)) + 0.5 * np.sin(2 *
     np.pi * 20 * np.linspace(0, 1, 128))
cutoff = 10  # Frecuencia de corte
filtered_signal = low_pass_filter(
    test_signal, cutoff)

# Gr ficos
plt.figure()
plt.subplot(2, 1, 1)
plt.plot(test_signal)
plt.title("Se al␣Original")

plt.subplot(2, 1, 2)
plt.plot(filtered_signal.real)
plt.title("Se al␣Filtrada")
plt.tight_layout()
plt.show()
```

En el código, utilizamos la transformada de Fourier rápida (FFT) para transformar la señal original al dominio de la frecuencia, aplicamos un filtro pasa-bajas eliminando las frecuencias superiores a un valor de corte (`cutoff`), y luego volvemos al dominio temporal utilizando la transformada inversa de Fourier (IFFT). El filtro fue aplicado a una señal compuesta por una componente sinusoidal de 5 Hz y otra de 20 Hz, con una frecuencia de corte de 10 Hz. Como resultado, el filtro eliminó las frecuencias superiores a 10 Hz, incluyendo el ruido a 20 Hz.

A continuación en la figura 4 se muestra la señal original y la señal filtrada

Como se observa en la gráfica, la señal filtrada retiene principalmente la componente de baja frecuencia a 5 Hz, que corresponde a la oscilación lenta de la señal original.

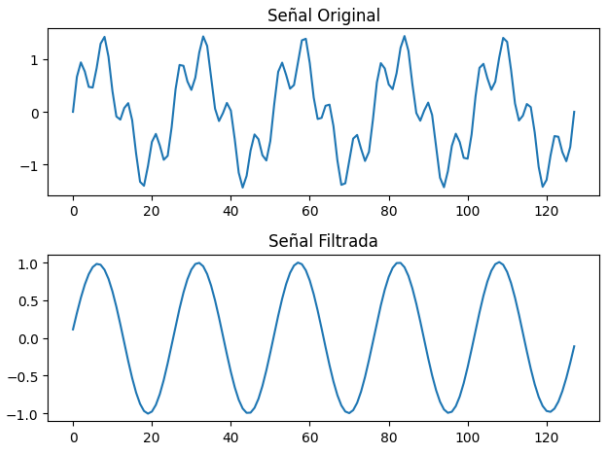


Fig. 4. Señal original y señal filtrada.

Antes de aplicar el filtro, la señal contenía una mezcla de dos componentes sinusoidales: una de 5 Hz y otra de 20 Hz, siendo esta última una oscilación más rápida y que puede considerarse como ruido en muchos contextos. Al aplicar el filtro pasa-bajas, todas las frecuencias por encima de la frecuencia de corte (10 Hz) se eliminan. Esto incluye la componente de 20 Hz, que es visible en la señal original como una oscilación de alta frecuencia superpuesta a la componente de 5 Hz. Después del filtrado, esta oscilación rápida ya no está presente, y lo que queda es una señal más suave y limpia, que preserva únicamente la

## VI. Interfaz con un SDR (bladeRF)

- a) Sintonice su bladeRF a una frecuencia central de 100MHz con un ancho de banda de 5MHz y capture 1M de muestras. En este caso, se ha utilizado un software-defined radio (SDR) bladeRF para recibir señales de radiofrecuencia (RF). El objetivo es sintonizar el dispositivo a una frecuencia central de 100 MHz con un ancho de banda de 5 MHz y capturar 1 millón de muestras. El código siguiente define una clase que configura y controla la recepción de estas muestras:

```
class BladeRFReceiver:
    def __init__(self, center_freq,
        sample_rate, bandwidth, gain,
        num_samples):
        self.center_freq = center_freq
        self.sample_rate = sample_rate
        self.bandwidth = bandwidth
        self.gain = gain
        self.num_samples = num_samples
        self.sdr = _bladerf.BladeRF()
        self.rx_ch = None
        self.x = np.zeros(num_samples,
            dtype=np.complex64)

        self._setup_device()

def _setup_device(self):
    print("Device info:", _bladerf.
        get_device_list()[0])
    print("libbladeRF version:",
        _bladerf.version())
    print("Firmware version:", self
        .sdr.get_fw_version())
    print("FPGA version:", self.sdr
        .get_fpga_version())

    self.rx_ch = self.sdr.Channel(
        _bladerf.CHANNEL_RX(0))
    self.rx_ch.frequency = self.
        center_freq
    self.rx_ch.sample_rate = self.
        sample_rate
    self.rx_ch.bandwidth = self.
        bandwidth
    self.rx_ch.gain_mode = _bladerf
        .GainMode.Manual
    self.rx_ch.gain = self.gain

    self.sdr.sync_config(
        layout=_bladerf.
            ChannelLayout.RX_X1,
        fmt=_bladerf.Format.
            SC16_Q11,
        num_buffers=16,
        buffer_size=8192,
        num_transfers=8,
        stream_timeout=3500
    )

def receive_samples(self):
    bytes_per_sample = 4
    buf = bytearray(1024 *
        bytes_per_sample)

    print("Starting receive")
    self.rx_ch.enable = True

    num_samples_read = 0
    while num_samples_read < self.
        num_samples:
        num = min(len(buf) //
            bytes_per_sample, self.
            num_samples -
            num_samples_read)
        self.sdr.sync_rx(buf, num)
        samples = np.frombuffer(buf
            , dtype=np.int16)
        iq_samples = samples[0::2]
            + 1j * samples[1::2]
        iq_samples /= 2048.0
        self.x[num_samples_read:
            num_samples_read+num] =
             iq_samples[:num]
        num_samples_read += num

    print("Stopping")
    self.rx_ch.enable = False
```

El código anterior configura la tarjeta SDR bladeRF con los siguientes parámetros:

- **Frecuencia central:** 100 MHz.
- **Tasa de muestreo:** 10 MHz.
- **Ancho de banda:** 5 MHz.
- **Ganancia:** 50 dB.

Luego, captura 1 millón de muestras y almacena

los datos de la señal recibida en formato IQ, donde I es la componente real y Q es la componente imaginaria.

- b) Dibuje las componentes real e imaginaria de la señal temporal por separado, y su magnitud del espectro en escala logarítmica.
  Una vez recibidas las muestras, el siguiente código genera dos gráficos:
  1) Las componentes I y Q de la señal en el dominio del tiempo.
  2) El espectro de frecuencia de la señal en escala logarítmica (dB).

```
def plot_time_domain(self):
        plt.figure(figsize=(10, 6))
        plt.plot(np.real(self.x
            [:1000]), label="Real␣(
            I)")
        plt.plot(np.imag(self.x
            [:1000]), label="
            Imaginario␣(Q)")
        plt.title("Time-Domain␣IQ␣
            Muestras␣(Primeras␣1000
            ␣Muestras)")
        plt.xlabel("Index␣de␣
            Muestra")
        plt.ylabel("Amplitud")
        plt.legend()
        plt.grid()
        plt.show()

def plot_frequency_spectrum(self):
        spectrum = np.fft.fftshift(
            np.fft.fft(self.x))
        freqs = np.fft.fftshift(np.
            fft.fftfreq(self.
            num_samples, 1 / self.
            sample_rate))

        plt.figure(figsize=(10, 6))
        plt.plot(freqs / 1e3, 20 *
            np.log10(np.abs(
            spectrum)))
        plt.title("Espectro␣de␣
            frecuencia␣(Magnitud␣en
            ␣dB)")
        plt.xlabel("Frecuencia␣(MHz
            )")
        plt.ylabel("Magnitud␣(dB)")
        plt.grid()
        plt.show()
```

*1) Gráfico en el dominio del tiempo:* El gráfico 5 nos muestra el dominio del tiempo de muestra de las primeras 1000 muestras de las componentes I y Q de la señal recibida. La componente I, que representa la parte real de la señal, captura la variación de la amplitud de la señal a lo largo del tiempo, mientras que la componente Q, que representa la parte imaginaria, captura la fase de la señal en relación con la componente real. En este gráfico, las oscilaciones visibles corresponden a las muestras de la señal en tiempo discreto, y cada punto representa un instante específico en el tiempo. Estas variaciones son características de la forma de onda que fue capturada por el dispositivo SDR y están relacionadas con la modulación de la señal de radiofrecuencia.

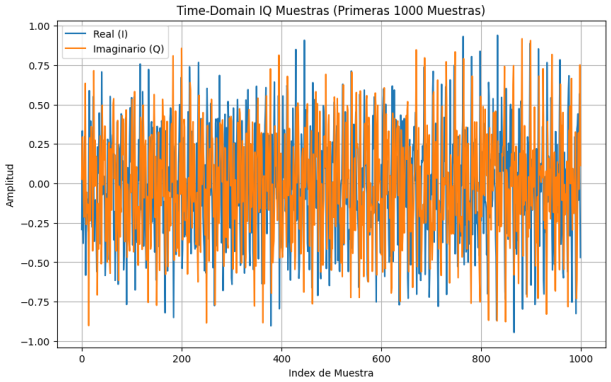


Fig. 5. Gráfico de las componentes real e imaginaria de la señal temporal (IQ).

*2) Gráfico del espectro de frecuencia:* En el grafico 6 muestra el espectro de la señal recibida en el dominio de la frecuencia. Se aplica una transformada de Fourier a las muestras para transformar la señal al dominio de la frecuencia. Luego, se grafica la magnitud del espectro en escala logarítmica (dB). Las frecuencias están centradas alrededor de la frecuencia central de 100 MHz, con un ancho de banda de 5 MHz. Las frecuencias se muestran en la escala de MHz, y la magnitud de cada componente de frecuencia está representada en dB. Este tipo de visualización es crucial para analizar la distribución espectral de la señal y detectar picos o interferencias que podrían estar presentes en el rango de frecuencia de interés.

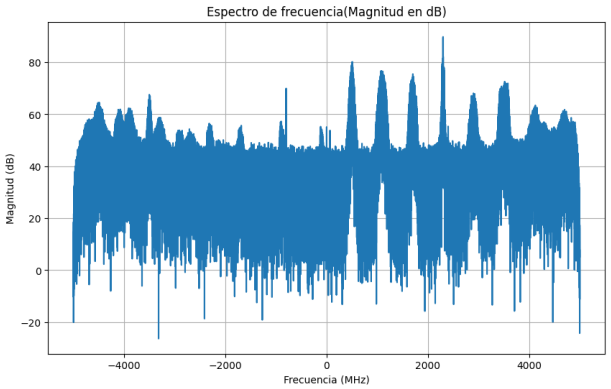


Fig. 6. Espectro de frecuencia de la señal en escala logarítmica (dB).

## VII. Afinando la Recepción

- a) Dibuje la magnitud de la señal I/Q y su espectro considerando la máxima ganancia del bladeRF.

Para afinar la recepción de la señal, se ha configurado el bladeRF con una ganancia máxima de 60 dB. A continuación se presenta el código utilizado para capturar y visualizar las muestras de la señal:

```
# Definir la configuraci n de la
    se al
sample_rate = 10e6    # 10 MHz de
    tasa de muestreo
center_freq = 100e6  # 100 MHz
    frecuencia central
bandwidth = 5e6       # 5 MHz ancho
     de banda
gain = 60              # Ganancia
num_samples = int(1e6)  # 1 mill n
     de muestras
receiver = BladeRFReceiver(
    center_freq, sample_rate,
    bandwidth, gain, num_samples)
receiver.receive_samples()
receiver.plot_time_domain()
receiver.plot_frequency_spectrum()
```

Después de recibir las muestras con la ganancia configurada a 60 dB, se obtienen dos gráficas: una que muestra la magnitud de la señal I/Q en el dominio del tiempo y otra que presenta su espectro de frecuencia.

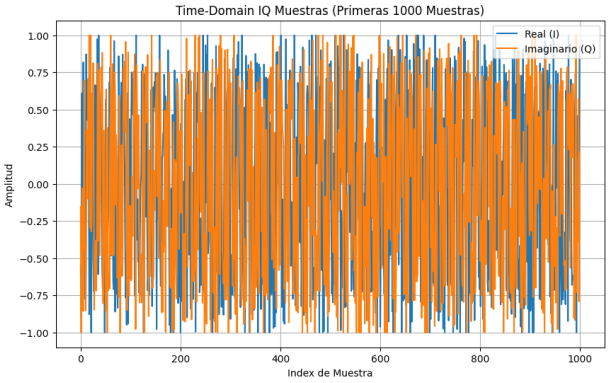


Fig. 7. Gráfico de la señal I/Q con máxima ganancia (60 dB).

En el gráfico 7 del dominio del tiempo, se puede observar que la amplitud de las señales `I` y `Q` ha aumentado significativamente debido a la alta ganancia aplicada. Esto puede resultar en un fenómeno conocido como "clipping", donde la señal supera el rango máximo que el sistema puede manejar, lo que provoca que la parte superior e inferior de la forma de onda se recorten.

En el grafico 8 observamos el espectro de frecuencia, se espera observar un aumento en la magnitud de las componentes a la frecuencia central de 100 MHz, pero también pueden aparecer distorsiones y picos adicionales en frecuencias no deseadas, debido al clipping. La energía que se pierde en la recorte de la señal puede manifestarse como un espectro ensanchado, donde se presentan artefactos

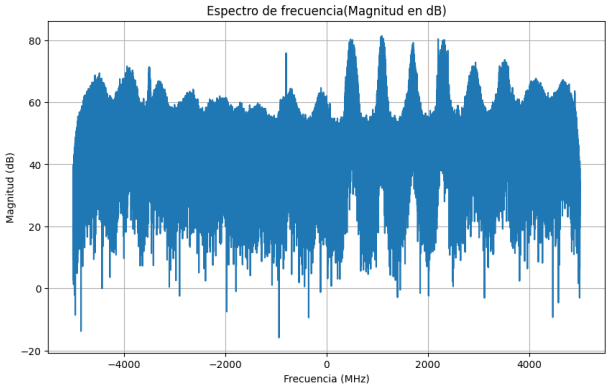


Fig. 8. Espectro de frecuencia de la señal con máxima ganancia (60 dB).

en frecuencias adyacentes, lo que puede interferir con la señal deseada y reducir la calidad de la recepción.

- b) Describa qué ocurre en el espectro cuando se produce un "clipping". Cuando se produce un "clipping" en la señal, significa que la amplitud de la señal excede el rango que el sistema puede manejar. Este fenómeno tiene implicaciones significativas en el espectro de frecuencia. Durante el clipping, las ondas sonoras que deberían ser suaves se convierten en formas de onda más cuadradas, lo que genera componentes armónicas adicionales.
  Esto se traduce en que:
  - **Aparición de armónicos**: Se generan frecuencias adicionales en el espectro que no estaban presentes en la señal original, lo que puede interferir con otras señales en el rango de frecuencia.
  - **Distorsión**: La calidad de la señal se deteriora, y las formas de onda no son representativas de la señal transmitida. Esto puede dificultar la demodulación y el análisis de la señal recibida.
  - **Reducción de la relación señal-ruido (SNR)**: Las componentes no deseadas pueden aumentar el ruido en el sistema, lo que empeora la relación SNR y, por ende, la calidad de la señal recibida.

  Por lo tanto, es crucial ajustar la ganancia del sistema de manera adecuada para evitar el clipping y garantizar la calidad de la señal y la eficacia del procesamiento posterior.

## VIII. Analizador de espectros

En esta sección, implementamos un analizador de espectro utilizando el dispositivo BladeRF. A continuación, se explican las partes clave del código y se presentan los resultados espectrales para varias frecuencias de FM.

### A. Clase `BladeRFSpectrumAnalyzer`

La clase `BladeRFSpectrumAnalyzer` se encarga de configurar el receptor BladeRF y recibir muestras IQ. A continuación se muestra el código relevante:

```
class BladeRFSpectrumAnalyzer:
    def __init__(self, center_freq, sample_rate
        , bandwidth, gain, num_samples):
        self.center_freq = center_freq
        self.sample_rate = sample_rate
        self.bandwidth = bandwidth
        self.gain = gain
        self.num_samples = num_samples
        self.sdr = _bladerf.BladeRF()
        self.rx_ch = self.sdr.Channel(_bladerf.
            CHANNEL_RX(0))  # Canal RX
        self.configure_rx_channel()
```

La inicialización de la clase define los parámetros necesarios para la recepción de la señal y configura el canal de recepción del BladeRF.

### B. Configuración del Canal

El método `configure_rx_channel` establece los parámetros del canal de recepción, incluyendo la frecuencia, la tasa de muestreo, el ancho de banda y la ganancia:

```
def configure_rx_channel(self):
    self.rx_ch.frequency = self.center_freq
    self.rx_ch.sample_rate = self.sample_rate
    self.rx_ch.bandwidth = self.bandwidth
    self.rx_ch.gain_mode = _bladerf.GainMode.
        Manual
    self.rx_ch.gain = self.gain
```

Esto asegura que el receptor esté configurado para captar las señales en la frecuencia deseada.

### C. Recepción de Muestras

El método `receive_samples` recibe las muestras IQ del BladeRF y las devuelve como un array de números complejos:

```
def receive_samples(self):
    buf = bytearray(1024 * 4)  # Buffer para
        datos IQ
    x = np.zeros(self.num_samples, dtype=np.
        complex64)
    num_samples_read = 0

    while num_samples_read < self.num_samples:
        num = min(len(buf) // 4, self.
            num_samples - num_samples_read)
        self.sdr.sync_rx(buf, num)
        samples = np.frombuffer(buf, dtype=np.
            int16)
        iq_samples = samples[0::2] + 1j *
            samples[1::2]  # Combinar I y Q
        iq_samples /= 2048.0
        x[num_samples_read:num_samples_read +
            num] = iq_samples[:num]
        num_samples_read += num

    return x
```

Este método normaliza las muestras recibidas y las almacena en un array de números complejos, permitiendo la posterior manipulación de la señal.

### D. Cálculo del Espectro

Para calcular el espectro de las muestras IQ, utilizamos el método `calculate_spectrum`:

```
def calculate_spectrum(self, samples):
    spectrum = fftshift(fft(samples))  #
        Aplicar FFT
    freqs = fftshift(fftfreq(self.num_samples,
        1 / self.sample_rate))
    magnitude = 20 * np.log10(np.abs(spectrum)
        + 1e-12)
    return freqs, magnitude
```

Este método aplica la Transformada Rápida de Fourier (FFT) a las muestras y devuelve las frecuencias y magnitudes en decibelios (dB).

### E. Resultados del Espectro

A continuación se presentan los resultados espectrales obtenidos para diferentes frecuencias de FM. Para obtener los espectros, se ha ejecutado el código en las frecuencias de 90.3 MHz, 100.5 MHz, 102.9 MHz y 107.7 MHz:

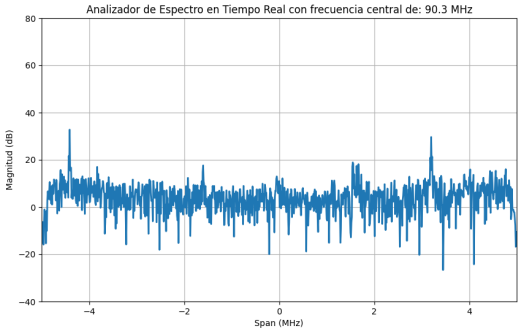


Fig. 9. Espectro para la frecuencia de 90.3 MHz

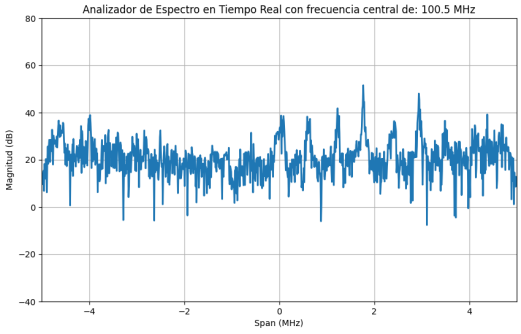


Fig. 10. Espectro para la frecuencia de 100.5 MHz

Los espectros obtenidos muestran la magnitud de las señales recibidas en decibelios (dB), lo que permite analizar en profundidad las características de las transmisiones de radio en las frecuencias seleccionadas. Al

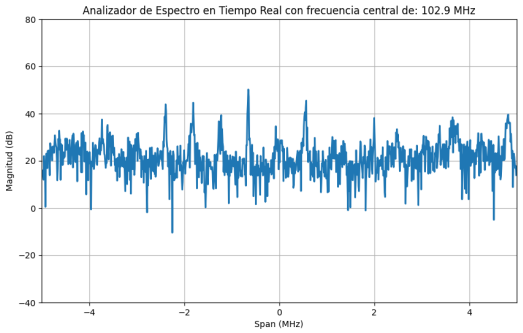


Fig. 11. Espectro para la frecuencia de 102.9 MHz

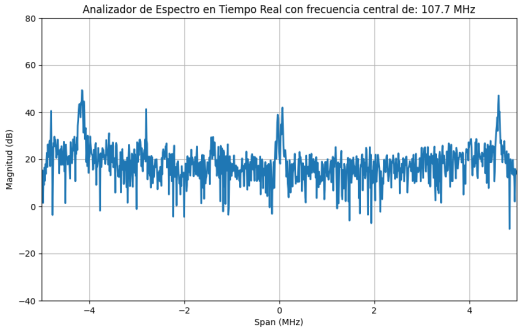


Fig. 12. Espectro para la frecuencia de 107.7 MHz

observar los espectros, se pueden identificar picos que representan las transmisiones de radio activas en esas frecuencias. Estos picos indican la presencia de señales de radiofusión, que pueden ser tanto estaciones de FM como otros tipos de transmisiones. La altura de cada pico en el gráfico refleja la intensidad de la señal, lo que permite discernir qué estaciones son más fuertes y cuáles pueden estar experimentando interferencias o debilitamiento.

## IX. Conclusiones

- La implementación de la Transformada Discreta de Fourier (DFT) es fundamental para el análisis de señales en dominios tanto de tiempo como de frecuencia, lo que facilita la identificación de componentes frecuenciales dominantes en una señal.
- Eficacia de los filtros paso-bajo: La implementación de un filtro pasa-bajas mediante la DFT ha permitido reducir significativamente el ruido de alta frecuencia en las señales de prueba, conservando las componentes de baja frecuencia deseadas. Esto es crucial en aplicaciones como la radio definida por software (SDR), donde es necesario filtrar interferencias no deseadas.
- El uso de radios definidos por software (SDR) permite la captura y procesamiento eficiente de señales de radio, lo que brinda flexibilidad y control sobre los parámetros de recepción como la frecuencia y el ancho de bandas presentes sin alterar la información original.
- El filtrado digital basado en la DFT, como el uso de filtros pasa-bajas, es una herramienta poderosa para eliminar el ruido de alta frecuencia, mejorando la claridad de las señales de interés.
- Mejoras en la resolución frecuencial con zero-padding: El uso de relleno con ceros al final de las señales ha mostrado una notable mejora en la resolución frecuencial de la DFT, permitiendo identificar componentes de frecuencia con mayor precisión, aunque sin alterar las frecuencias originales.

## X. Referencias


- Proakis, J. G., Manolakis, D. G. (2007). Digital Signal Processing: Principles, Algorithms, and Applications. Prentice-Hall.
  Esta obra aborda en profundidad los fundamentos de procesamiento digital de señales, incluida la Transformada Discreta de Fourier (DFT) y sus aplicaciones.
- Oppenheim, A. V., Schafer, R. W., Buck, J. R. (1999). Discrete-Time Signal Processing. Prentice-Hall.
  Referencia clave para comprender la DFT y las técnicas de análisis espectral en tiempo discreto.
- Mitola, J. (1999). Cognitive Radio for Flexible Mobile Multimedia Communications. IEEE Journal on Selected Areas in Communications, 17(5), 191–202.
  Introduce los conceptos de radio definido por software (SDR), sus fundamentos y aplicaciones actuales en comunicaciones inalámbricas.
- Xu, Z., Xu, W., Li, X. (2021). Software Defined Radio (SDR): Architecture, Challenges and Applications. IEEE Communications Surveys Tutorials, 23(1), 30-56.
  Este artículo proporciona una revisión exhaustiva sobre el desarrollo de la tecnología SDR y su implementación en sistemas modernos de comunicación.
- Mahmoud, M. O., Attiya, A. (2018). Low-Pass Filter Design Using Fast Fourier Transform (FFT). Journal of Electrical Engineering, 9(3), 57-62.
  Enfoque práctico sobre el diseño e implementación de filtros en el dominio de la frecuencia utilizando la FFT.
- Jorge Alberto Chavez Ponce, Neyra Torres Luis Kenny, Javier Cana Remache, Rosangela Yllachura Arapa (2024). https://github.com/jorgechvz/redes-inalambricas/tree/main/practica-1

Respositorio de la practica de Analizador de Espectro en tiempo real
- Wiki BladeRF Documentation. https://github.com/Nuand/bladeRF/wiki