PROYECTO FINAL
Procesamiento Digital de Señales

DETECCIÓN Y CORRECCIÓN DE ERRORES EN TRANSMISIONES DE DATOS INALÁMBRICOS

David alexander Sánchez García

2023-1

1. Introducción.

En las transmisiones inalámbricas, las señales pueden sufrir diferentes tipos de errores debido a la interferencia del canal, el ruido, la atenuación, la dispersión y otros fenómenos. Para garantizar una transmisión confiable y precisa de los datos, es necesario utilizar técnicas de detección y corrección de errores.

Los filtros FIR e IIR son técnicas comunes utilizadas en el procesamiento de señales para modificar o extraer información de una señal. En el contexto de las transmisiones inalámbricas, se utilizan para filtrar las señales recibidas y reducir el impacto de los errores de transmisión.
En este laboratorio lo que se va a realizar es la simulación de un canal de comunicaciones, con señales sintéticas generadas cumpliendo ciertos parámetros y mirando como es su comportamiento en el dominio de la frecuencia, para luego hacerlas pasar por filtros FIR e IIR y ver como es su comportamiento; Para terminar, se miraran el rendimiento de los filtros mediante técnicas de validación, como la tasa de error de bits, la relación señal a ruido y la tasa de detección de errores.

Imagen1: Sistema de transmisión de comunicaciones

2. Generación de señales.

2.1 ¿Qué sucede al generar dos señales senoidales sintéticas con diferentes frecuencias de muestreo y cómo se relaciona esto con el teorema de muestreo de Nyquist? Explique según las gráficas obtenidas.

2.2 Compare las dos señales senoidales sintéticas con diferentes frecuencias de muestreo en un solo gráfico y explique las diferencias observadas

Nota: Consulte la generación de señales sintéticas y en que son más utilizadas, para más información puede consultar: **https://github.com/amalvarezme/ProcesoDigitalSen/tree/master/DSP_Notebooks/2_spectral_analysis_deterministic_signals**

3. Generación de un canal de comunicación inalámbrico.

Para la generación de las señales de transmisión puede hacer uso de las siguientes líneas:

```
atenuación = x #lo que simula una reducción de la amplitud de las señales
interferencia = x * np.sin(2 * np.pi * x * tiempo)
señal_transmitida = señal * atenuación + canal _ruido + interferencia
```

3.1 Al generar un canal inalámbrico con errores de transmisión controlados, como ruido, atenuación e interferencias, ¿qué observaciones se pueden hacer en relación con la calidad de la señal transmitida?

3.2 ¿Cómo se calcula y qué representa la relación señal-ruido (SNR) en una señal?

```
SNR = 10 * log10(P_señal / P_ruido)
```

3.3 Al analizar las señales transmitidas en el dominio de la frecuencia, ¿qué observaciones destacan o qué patrones se pueden identificar?

3.4 Aplicar una transformada diferente a la Transformada de Fourier para comparar los resultados obtenidos. ¿Qué características o diferencias se

pueden observar en la nueva transformada en comparación con la Transformada de Fourier?

Nota: Una transformada alternativa que se puede aplicar para comparar los resultados con la Transformada de Fourier es la Transformada de Ondaleta (Wavelet Transform). La Transformada de Ondaleta es una técnica que permite analizar las señales en el dominio tiempo-frecuencia y ofrece una mayor resolución tanto en tiempo como en frecuencia en comparación con la Transformada de Fourier.
Puede tener más información aquí:
https://www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-07642012000100008

Su aplicación en Python es sencilla:

```
!pip install PyWavelets
import pywt
# Aplicar la Transformada de Ondaleta
wavelet_señal = pywt.wavedec(señal_transmitida, 'db4')
# Obtener los coeficientes de detalle y el coeficiente de aproximación
wavelet_coef_señal1 = np.concatenate(wavelet_señal [1:]) # Unir los
coeficientes de detalle
```

Instalar primero

4. Aplicación de Filtros FIIR e IIR.

Respuesta al impulso finita (FIR)

$$y[n] = (x * h)[n]$$

$$y[n] = \sum_{k} x[k]h[n-k]$$

Respuesta al impulso infinita (IIR)

$$y[n] = a_1 y[n-1] + a_2 y[n-2] + b_0 x[n] + b_1 x[n-1] + b_2 x[n-2]$$

12

Filtros FIR

Un filtro FIR es un tipo de filtro digital cuya respuesta al impulso es de duración finita. Esto significa que su respuesta al impulso tiene una longitud finita y luego se vuelve cero. La respuesta al impulso determina cómo el filtro responde a diferentes frecuencias en la señal de entrada.

La respuesta en frecuencia de un filtro FIR se puede diseñar utilizando diversas técnicas, como el método de enventanado o el método de muestreo de frecuencia.

La ventaja de los filtros FIR es que tienen una respuesta al impulso de fase lineal, lo que significa que no introducen distorsiones de fase en la señal filtrada. Esto es importante en aplicaciones de detección y corrección de errores, donde preservar la forma de onda original de la señal es crucial

Filtros IIR

A diferencia de los filtros FIR, los filtros IIR tienen una respuesta al impulso de duración infinita. Esto se debe a que la salida de un filtro IIR depende tanto de las entradas actuales como de las salidas anteriores, lo que crea una retroalimentación en el sistema.

Estos filtros IIR son más flexibles en términos de diseño y pueden lograr una respuesta en frecuencia más ajustada utilizando menos coeficientes que los filtros FIR. Sin embargo, su principal desventaja es la introducción de distorsiones, ya que este es de fase no lineal.

Para detectar y corregir los errores en las señales recibidas, se utilizarían filtros FIR e IIR. Estos filtros se aplicarían a las señales recibidas para reducir el impacto de los errores y mejorar la calidad de la señal.

4.1 Diseñe un filtro FIR pasa-bajas con una banda de transición de 200 Hz, una frecuencia de corte de 600 Hz, Frecuencia de muestreo de 16khz y un ripple de 60 dB.
Nota: Puede apoyarse del siguiente script que corresponde al diseño de un filtro pasa-altas

```
from scipy.signal import kaiserord, lfilter, firwin, freqz

nyq_rate = fs / 2.0
roll_off = 200.0
cutoff_hz = 5000.0
width = roll_off/nyq_rate
ripple_db = 60.0 #The desired attenuation in the stop band, in dB.

N, _ = kaiserord(ripple_db, width) # Compute the order and Kaiser parameter for the FIR filter.

taps = firwin(N, cutoff_hz/nyq_rate, pass_zero=False)

w, h = signal.freqz(taps, [1], worN=2000)
plt.plot(nyq_rate*w/np.pi, np.abs(h))
```

4.2 Al pasar las señales por el filtro FIR diseñado, ¿qué se puede observar en términos de los cambios en las señales y cómo se ven afectadas por el filtro?

4.3 Al realizar un filtro IIR de su elección, ¿cuáles son las diferencias y similitudes entre este filtro IIR y el filtro FIR previamente diseñado? ¿Cómo se comparan en términos de rendimiento, características de filtrado y efecto en las señales filtradas?

Nota: Haciendo uso de script en Python, un filtro IIR pasa bajás se puede diseñar como:

```
import scipy.signal as sp

b, a = sp.butter(6, Wn, btype='low', analog=False, output='ba', fs=None)
wb, Hb = sp.freqz(b, a);
plt.plot((wb*fs)/(2*np.pi), np.abs(Hb))
```

Además, se puede filtrar una señal x haciendo uso de la siguiente función:

```
y = sp.lfilter(b, a, x)
```

Nota: Consulte qué son y cómo variar los parámetros de las funciones compartidas para conseguir los respectivos filtros aquí: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.butter.html

5. Detección Errores

5.1 Realizar una evaluación del rendimiento de los filtros mediante técnicas de validación, como la tasa de error de bits, la relación señal-ruido y la tasa de detección de errores. ¿Concluir si estos errores se pueden arreglar?
Nota: puede apoyarse de las funciones de la tasa de error de bits, la relación señal-ruido y la tasa de detección de errores.

```
def calcular_ber(señal_transmitida, señal_filtrada):
    total_bits = len(señal_transmitida)
    bits_erroneos = np.sum(señal_filtrada != señal_transmitida)
    ber = bits_erroneos / total_bits
    return ber

def calcular_snr(señal_original, señal_filtrada):
    potencia_señal_original = np.mean(señal_original**2)
    potencia_ruido = np.mean((señal_filtrada - señal_original)**2)
    snr = potencia_señal_original / potencia_ruido
    return snr

def calcular_tasa_deteccion(señal_original, señal_filtrada):
    errores_originales = np.sum(señal_original != señal_filtrada)
    errores_detectados = np.sum(señal_filtrada != señal_original)
    tasa_deteccion = 1 - (errores_detectados / errores_originales)
    return tasa_deteccion
```

5.2 Calcule el porcentaje de valores que son iguales entre las señales filtradas y la señal transmitida.

6. Conclusiones

Realice conclusiones generales sobre la práctica