1 Definición del proyecto

En este proyecto se busca que los estudiantes diseñen un filtro digital por medio de la optimización sin restricciones.

1.1 Definición matemática del problema

Denotamos por $h_0,\ h_1, \ldots,\ h_n$ los coeficientes de un filtro de respuesta finita (FIR) de longitud $n+1$. Esto significa que la salida del filtro esta dada por [1]

$$y[k] = \sum_{k=0}^{n} h_i u[k-i].$$

La respuesta en frecuencia del filtro es la función

$$H[\omega] = \sum_{k=0}^{n} h_k e^{-jk\omega},\ \text{con}\ \omega \in [0, \pi],$$

donde $j = \sqrt{-1}$ y ω la variable de velocidad angular. La expresión anterior la podemos reescribir como el siguiente producto punto

$$H[\omega_1] = [1, e^{-j(1)\omega_1}, \ldots, e^{-j(n)\omega_1}] \cdot [h_0, h_1, \ldots, h_n],$$

y de forma mas general, como el siguiente producto matricial

$$\mathbf{H} = \begin{bmatrix} 1 & \mathrm{e}^{-j\omega_1} & \mathrm{e}^{-j2\omega_1} & \cdots & \mathrm{e}^{-j(n)\omega_1} \\ 1 & \mathrm{e}^{-j\omega_2} & \mathrm{e}^{-j2\omega_2} & \cdots & \mathrm{e}^{-j(n)\omega_2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \mathrm{e}^{-j\omega_N} & \mathrm{e}^{-j2\omega_N} & \cdots & \mathrm{e}^{-j(n)\omega_N} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_n \end{bmatrix} = \mathbf{A}\mathbf{h}.$$

donde $\mathbf{H}$ es un vector con N elementos, $\mathbf{A} \in \mathbb{R}^{N \times (n+1)}$ y $\mathbf{h} \in \mathbb{R}^{n+1}$ es un vector compuesto por los coeficiente h_k. Buscamos minimizar la diferencia entre la respuesta en frecuencia deseada ($\mathbf{H}^{\text{des}}$) con la generada a partir del filtro que estamos sintonizando

$$\underset{\mathbf{h}}{\text{minimizar}} \left\| \begin{bmatrix} 1 & \mathrm{e}^{-j\omega_1} & \mathrm{e}^{-j2\omega_1} & \cdots & \mathrm{e}^{-j(n)\omega_1} \\ 1 & \mathrm{e}^{-j\omega_2} & \mathrm{e}^{-j2\omega_2} & \cdots & \mathrm{e}^{-j(n)\omega_2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \mathrm{e}^{-j\omega_N} & \mathrm{e}^{-j2\omega_N} & \cdots & \mathrm{e}^{-j(n)\omega_N} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_n \end{bmatrix} - \begin{bmatrix} H_1^{\text{des}} \\ H_2^{\text{des}} \\ \vdots \\ H_N^{\text{des}} \end{bmatrix} \right\|^2,$$

o de manera matricial

$$\underset{\mathbf{h}}{\text{minimizar}} \left\| \mathbf{A}\mathbf{h} - \mathbf{H}^{\text{des}} \right\|^2.$$

Debido que la función objetivo pertenece a los números complejos, se puede reescribir la ecuación anterior como

$$\underset{\mathbf{h}}{\text{minimizar}} \left\| \text{real}(\mathbf{A}\mathbf{h} - \mathbf{H}^{\text{des}}) \right\|^2 + \left\| \text{imag}(\mathbf{A}\mathbf{h} - \mathbf{H}^{\text{des}}) \right\|^2.$$

1.2 Codigo en Python

```
import numpy as np

n = 20 #Cantidad de coeficientes

N = 15*n #Numero de frecuencias

#Vector columna de frecuencias
w = np.linspace(0,np.pi,N).reshape(-1, 1)

# Respuesta en frecuencia del filtro deseado
D = 5.25
Hdes = np.exp(-1j*D*w)

# Construccion de la matriz A
A = np.exp( -1j * np.kron(w, np.arange(n)))

#Extraer parte real e imaginaria
Hdes_R = np.real(Hdes)
Hdes_I = np.imag(Hdes)

A_R = np.real(A)
A_I = np.imag(A)

def ObjFun(h):
   return np.sum( np.square(np.matmul(A_R,h) - Hdes_R) + np.square(np.matmul(A_I,h)
      - Hdes_I))

def gradObjFun(h):
   return 2*A_R.T@(np.matmul(A_R,h) - Hdes_R) + 2*A_I.T@(np.matmul(A_I,h) - Hdes_I)
```

1.3 Procedimiento

1. A partir de la función objetivo y su gradiente, usar el método del gradiente más descendiente (Steepest Descent), para encontrar la solución al problema de optimización.

2. Seleccionar un valor adecuado del paso (α), de tal manera que el proceso de optimización converja a buenas soluciones.

3. Realizar el proceso de optimización 50 veces. En cada realización almacenar el vector solución y la curva de la función objetivo vs iteraciones.

4. Realizar un BoxPlot sobre las 50 soluciones encontradas en cada optimización. Analizar la variabilidad de cada parámetro.

5. Graficar la media de las 50 curvas de función objetivo vs iteraciones calculada en cada realización del proceso de optimización. Determinar para que cantidad de iteraciones es suficiente ejecutar el proceso de optimización.

6. Comparar la respuesta en frecuencia deseada con la obtenida a partir de los mejores parámetros encontrados en las 50 realizaciones.

2 Informe

Desarrollar un notebook en Python, que incluya las siguientes secciones:

1. Introducción al problema.
2. Codigo y desarrollo de la solución.
3. Análisis de resultados.
4. Conclusiones.
5. Bibliografía.

Referencias

[1] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Hervé Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1-3):193–228, November 1998.