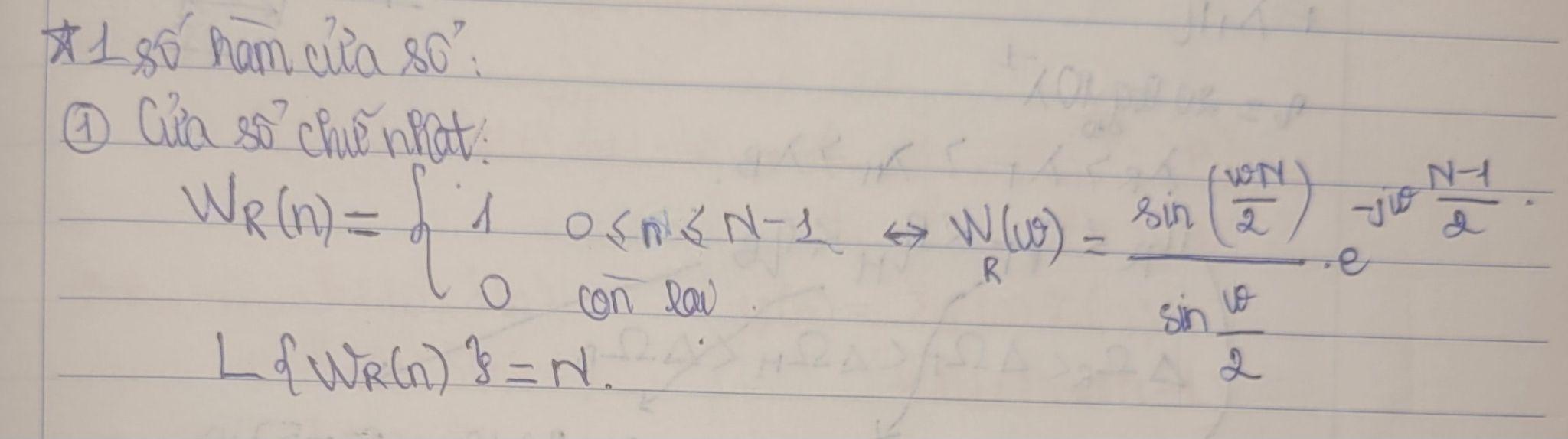
| LẬP TRÌNH ÂM THANH  Spring 2023- D20PT | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Nhóm Bài tập lớn: SNDPROGSP23PRJ09 | | | Bài: Sound enhancement & Basic sound effects | | | |
| Họ và tên: Nguyễn Quốc Đạt (NT) | Mã sinh viên: B20DCPT053 | | | Nhóm lớp: 01 | | 10 |
| Họ và tên : Nguyễn Hồng Nhung | Mã sinh viên: B20DCPT150 | | | Nhóm lớp: 01 | | 10 |
| Họ và tên : Nguyễn Văn Tuấn | Mã sinh viên: B20DCPT182 | | | Nhóm lớp: 01 | | 10 |
| Mức độ hoàn thành: 95% | | | | | | |
| Đánh giá cụ thể | | | | | | |
| Hạng mục | | Mức độ thực hiện | | | Số ví dụ thử nghiệm | |
| Các hàm cửa sổ | | Mô tả thành công các hàm cửa sổ thông dụng như rectangle, triangle, hanning, hamming, blackman | | | 5 | |
| Các bộ lọc LPF, HPF, BSF, BPF | | Thiết kế các bộ lọc lý tưởng theo công thức đã có | | | 4 | |
| Thực hiện sử dụng các bộ lọc xây dựng được theo phương pháp cửa sổ, lọc tín hiệu âm thanh, hiển thị kết quả tín hiệu (dạng sóng thời gian, đồ hình biên độ phổ) trước và sau lọc, nghe cảm nhận tín hiệu trước và sau lọc | | Thực hiện lọc thành công với các bộ lọc cơ bản, nhưng chưa thể kết hợp các bộ lọc lại với nhau để tạo thành Equalizer | | | 0 | |
| Hàm thực hiện các hiệu ứng âm thanh đơn giản như echo, reversal, dangle, .... | | Mô tả thành công các hiệu ứng âm thanh đơn giản | | | 4 | |

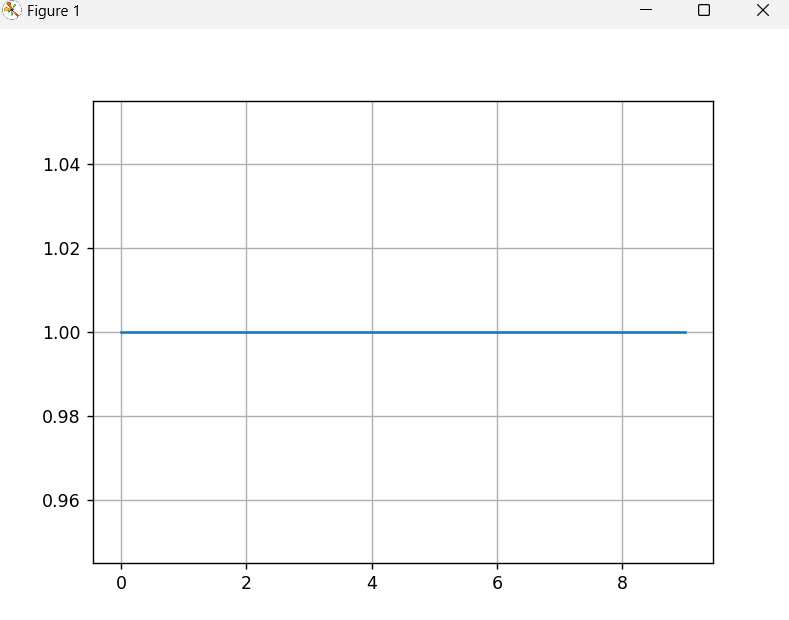
1. **Các hàm cửa sổ:**
2. **Cửa sổ chữ nhật:**



def rectangle(n = 1000):

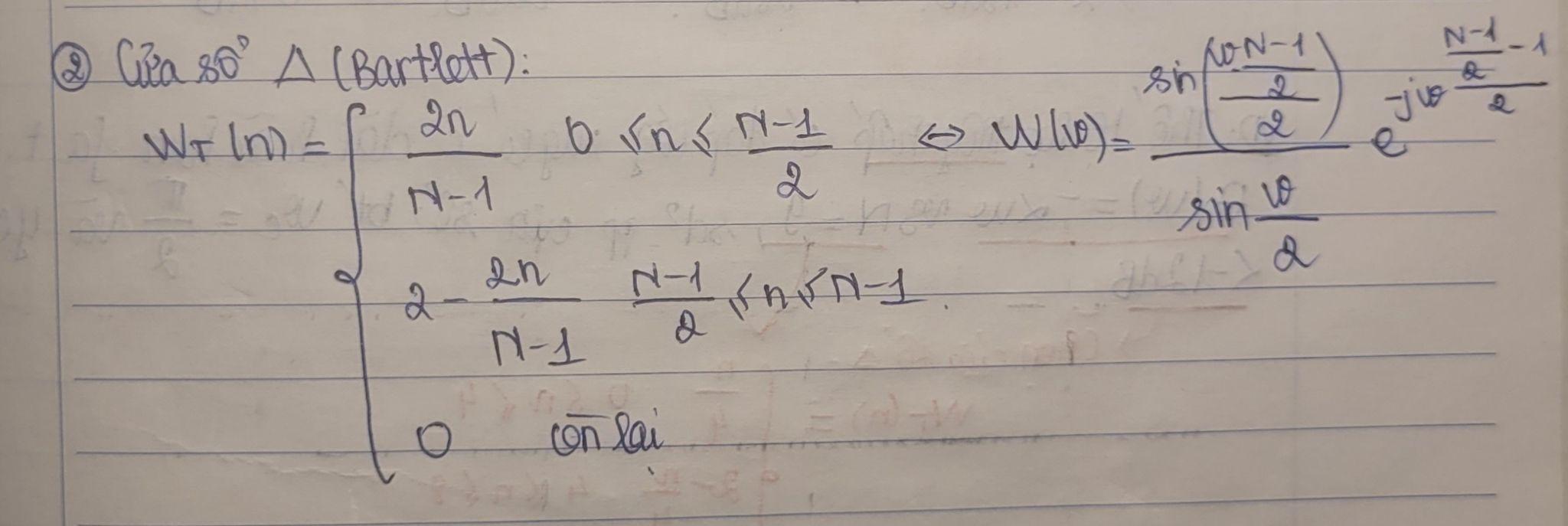
w = np.ones(n)

return np.array(w)



Cửa sổ chữ nhật

1. **Cửa sổ tam giác:**



def triangle(n = 1000):

w = np.zeros(n)

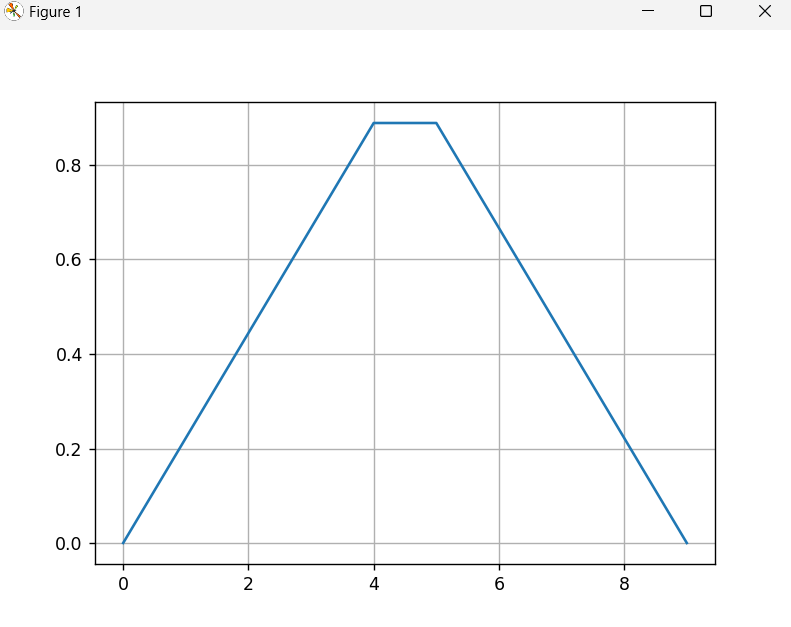
for i in range(0, int((n - 1) / 2) + 1):

w[i] = (2 \* i) / (n - 1)

for i in range(int((n - 1) / 2) + 1, n):

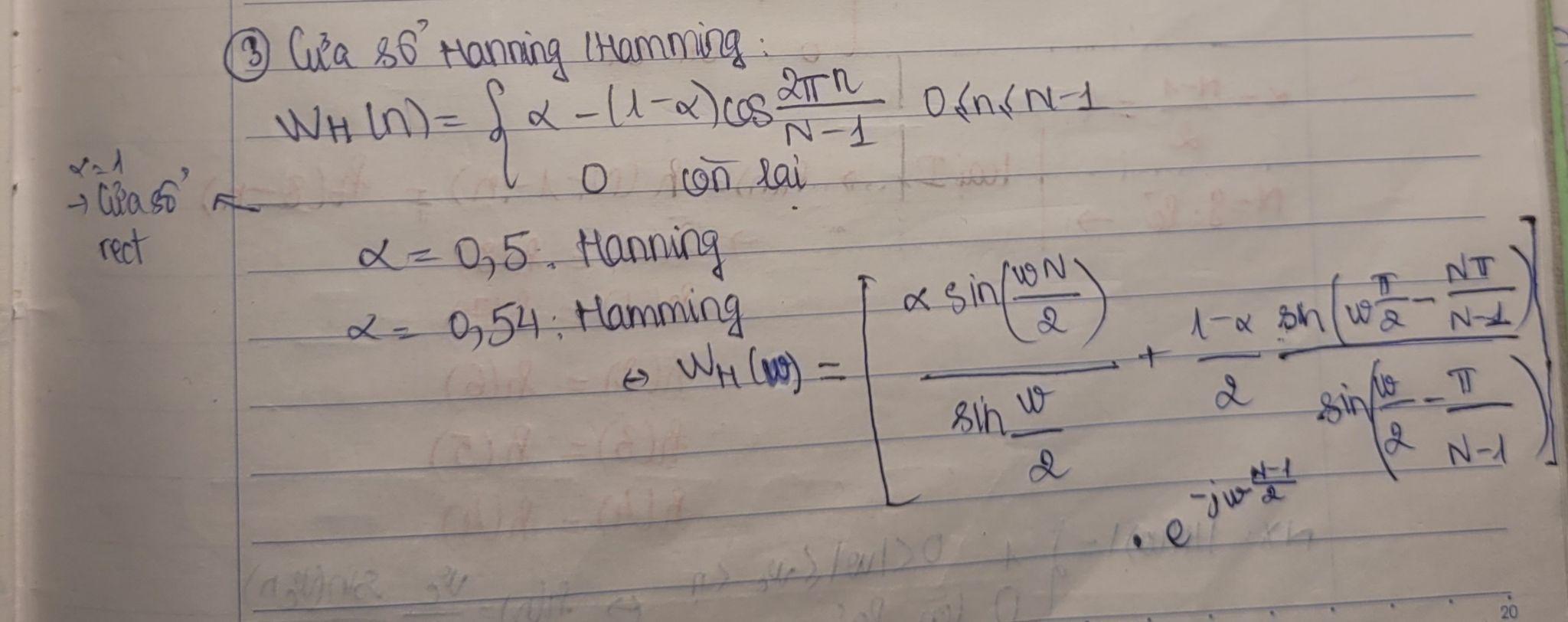
w[i] = 2 - ((2 \* i ) / (n - 1))

return np.array(w)



Cửa sổ tam giác

1. **Cửa sổ Hanning & Hamming:**



def hanning(n = 1000):

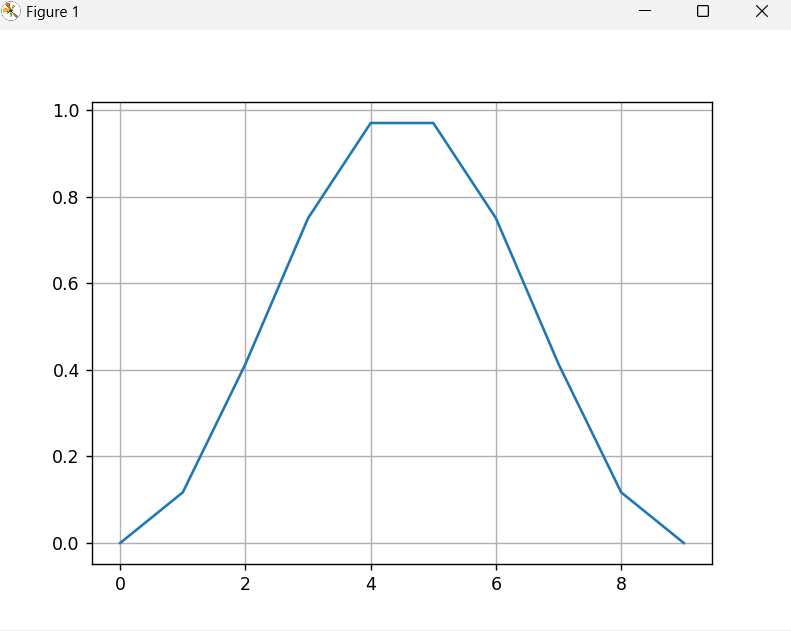
w = 0.5 - 0.5 \* np.cos(2 \* np.pi \* np.arange(n) / (n - 1))

return np.array(w)

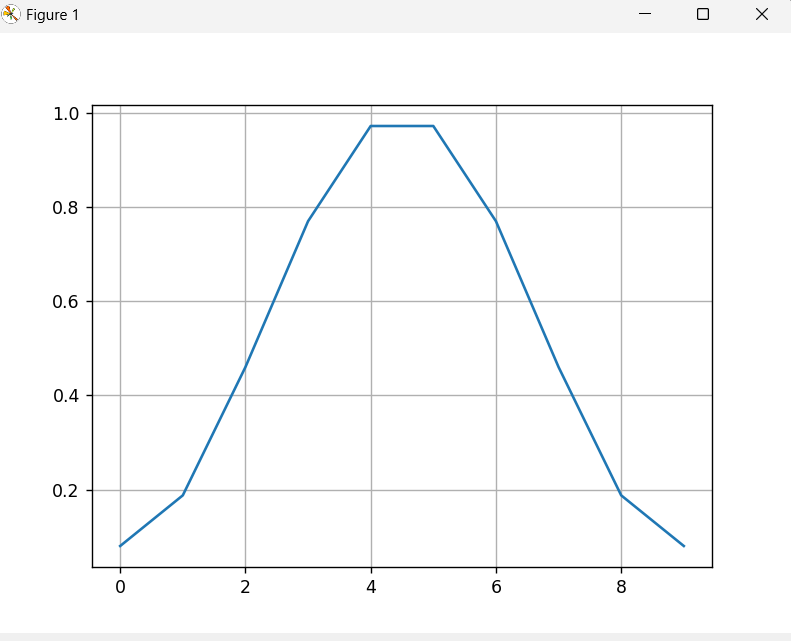
def hamming(n = 1000):

w = 0.54 - 0.46 \* np.cos(2 \* np.pi \* np.arange(n) / (n - 1))

return np.array(w)

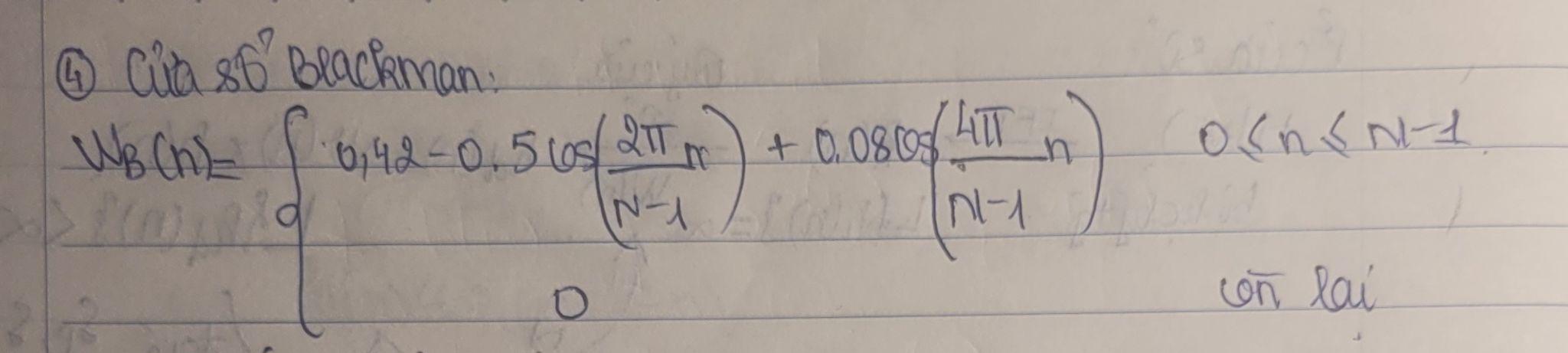


Cửa sổ Hanning



Cửa sổ Hamming

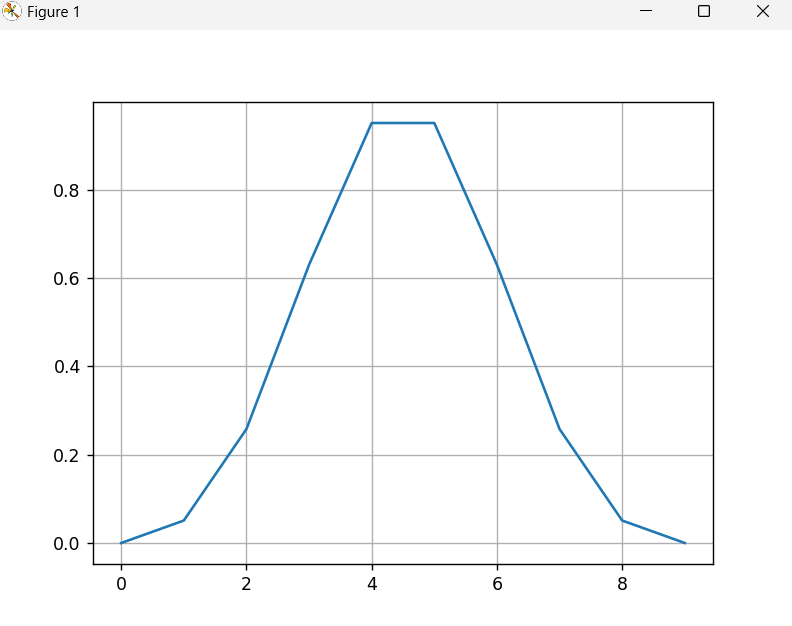
1. **Cửa sổ Blackman:**



def blackman(n = 1000):

w = 0.42 - 0.5 \* np.cos(2 \* np.pi \* np.arange(n) / (n - 1)) + 0.08 \* np.cos(4 \* np.pi \* np.arange(n) / (n - 1))

return np.array(w)



Cửa sổ Blackman

Ngoài ra ta cũng có thể sử dụng các hàm cửa sổ có sẵn trong thư viện Numpy với cú pháp, np.(tên cửa sổ), tham số đầu vào là chiều dài của cửa sổ.

1. **Các bộ lọc cơ bản LPF, HPF, BPF, BSF:**
2. **Bộ lọc LPF:**

* Khái niệm: Bộ lọc thông thấp là một bộ lọc dùng để giảm các tín hiệu có tần số cao hơn một giá trị cắt xác định. Nó được sử dụng để loại bỏ các tín hiệu nhiễu hoặc tín hiệu không mong muốn khác. Bộ lọc thông thấp có thể được áp dụng trong nhiều ứng dụng khác nhau, chẳng hạn như trong xử lý âm thanh, xử lý ảnh và xử lý tín hiệu số.
* Công dụng chính của bộ lọc thông thấp là loại bỏ các tín hiệu có tần số cao hơn một giá trị cắt. Giá trị cắt này thường được chọn dựa trên yêu cầu của ứng dụng cụ thể và được xác định dựa trên tần số cao nhất mà tín hiệu mong muốn được giữ lại. Bộ lọc thông thấp hoạt động bằng cách giảm bớt các tần số cao hơn giá trị cắt, nhưng vẫn giữ lại các tần số thấp hơn giá trị cắt.
* Code minh họa: Các tham số truyền vào bao gồm: N - bậc của bộ lọc, fc - tần số cắt, fs - tần số lấy mẫu, được tính bằng tần số nyquist = tần số lấy mẫu của tín hiệu âm thanh / 2.

def LPF(N, fc = 2000, fs = 8000, lamda = 0):

deltaW = 4 \* np.pi / N

fc\_norm = fc / fs

wc\_norm = 2 \* np.pi \* fc\_norm

alpha = (N - 1) // 2

n = np.arange(N)

h = np.zeros\_like(n, dtype=float)

h[n == 0] = 2 \* fc\_norm

h[n != 0] = (2 \* fc\_norm) \* np.sinc(2 \* fc\_norm \* (n[n != 0] - (N - 1) / 2))

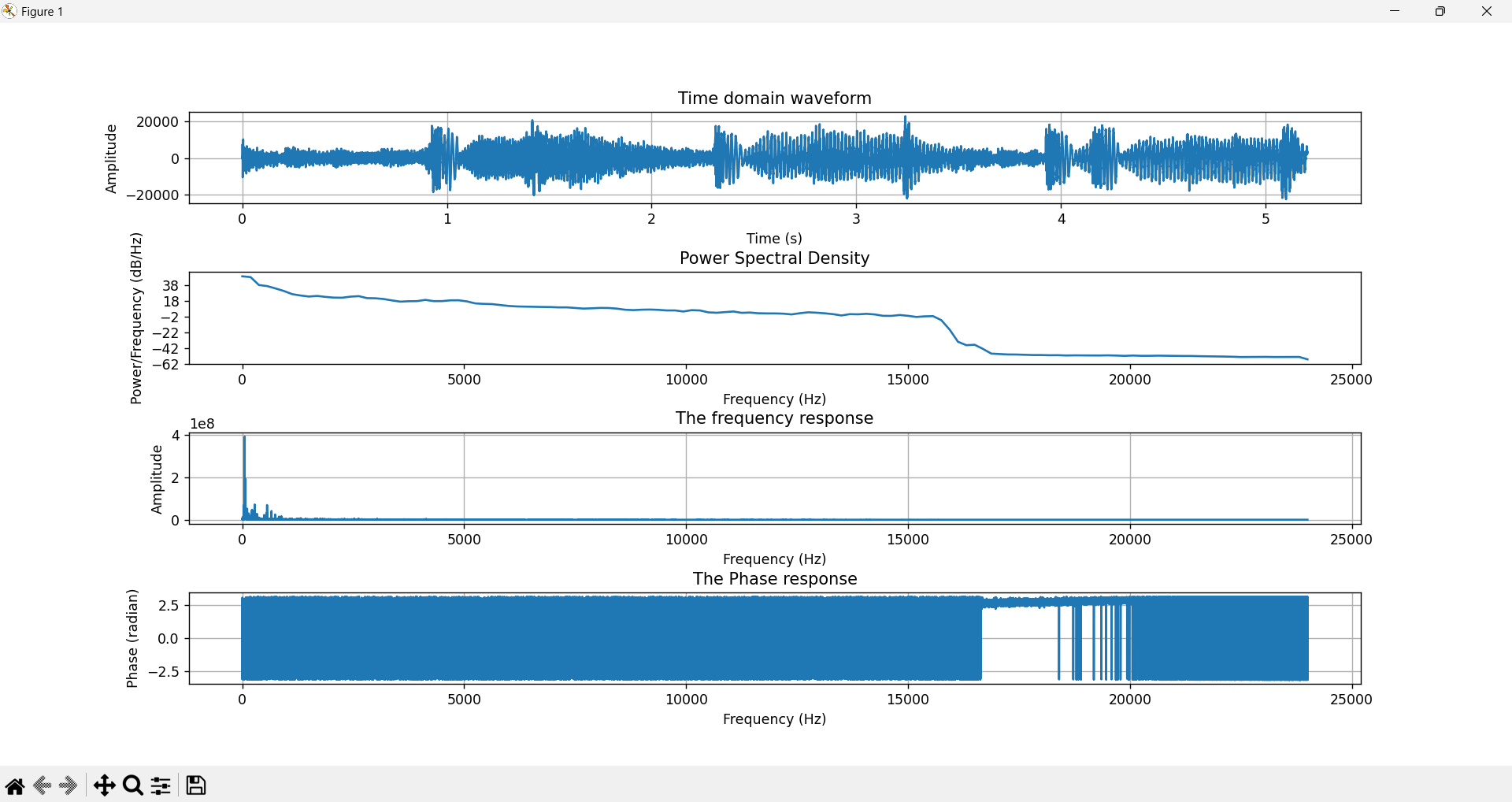
windowx = window\_function.hanning(N)

h\_d = windowx \* h

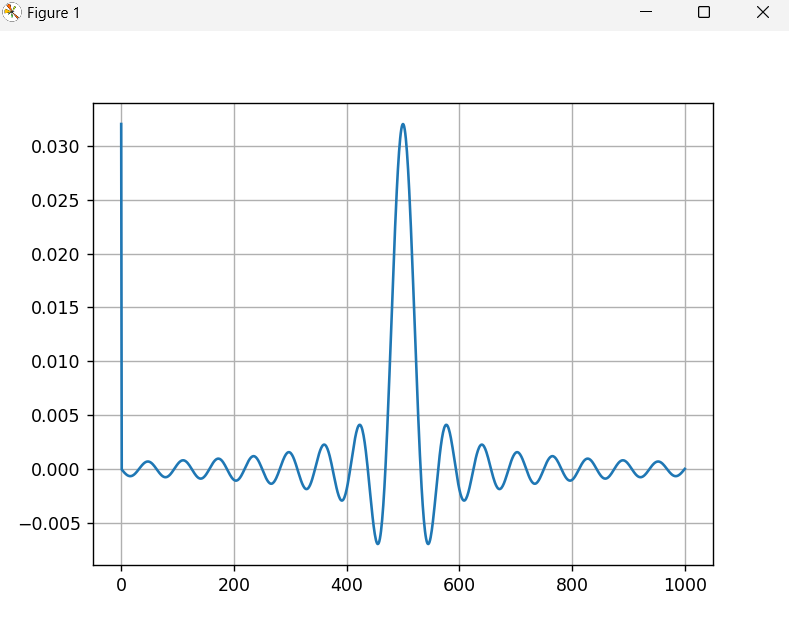
return h\_d

Bộ lọc LPF

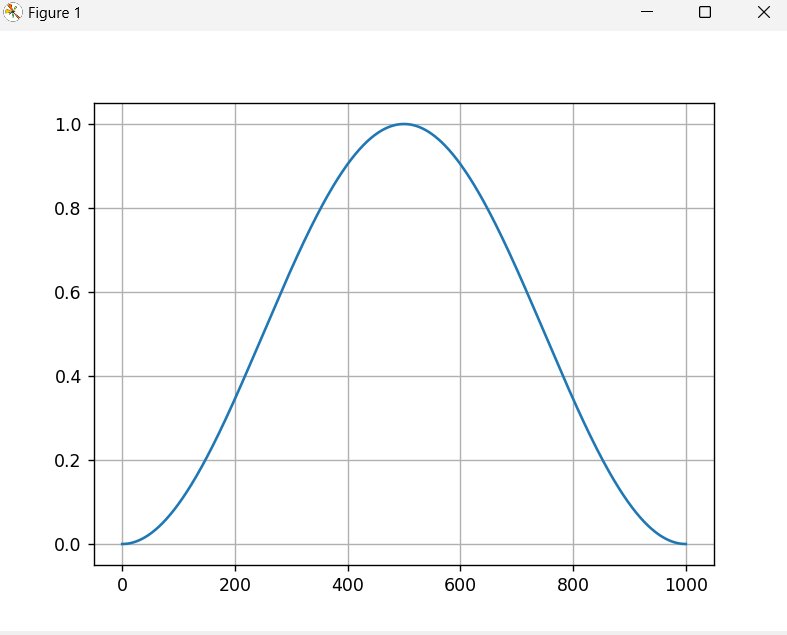
* Test:



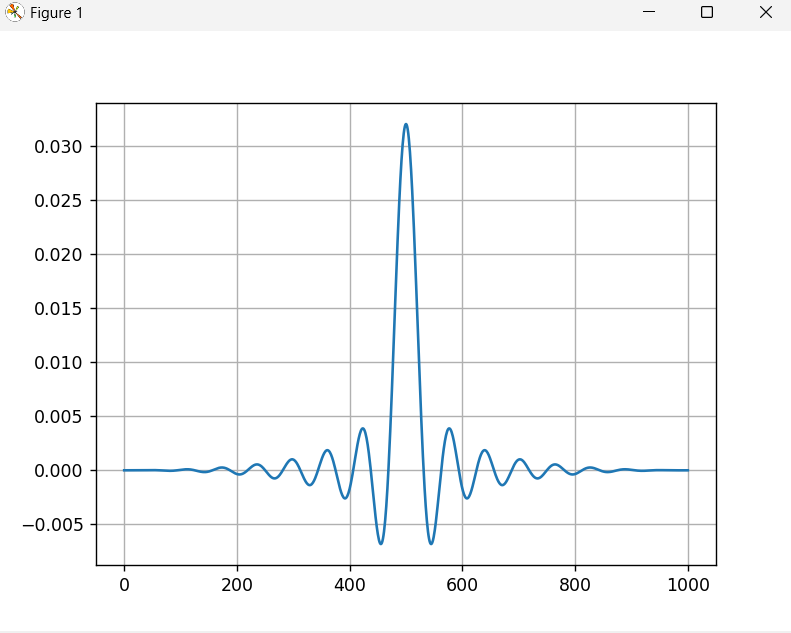
Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha phổ theo miền tần số.



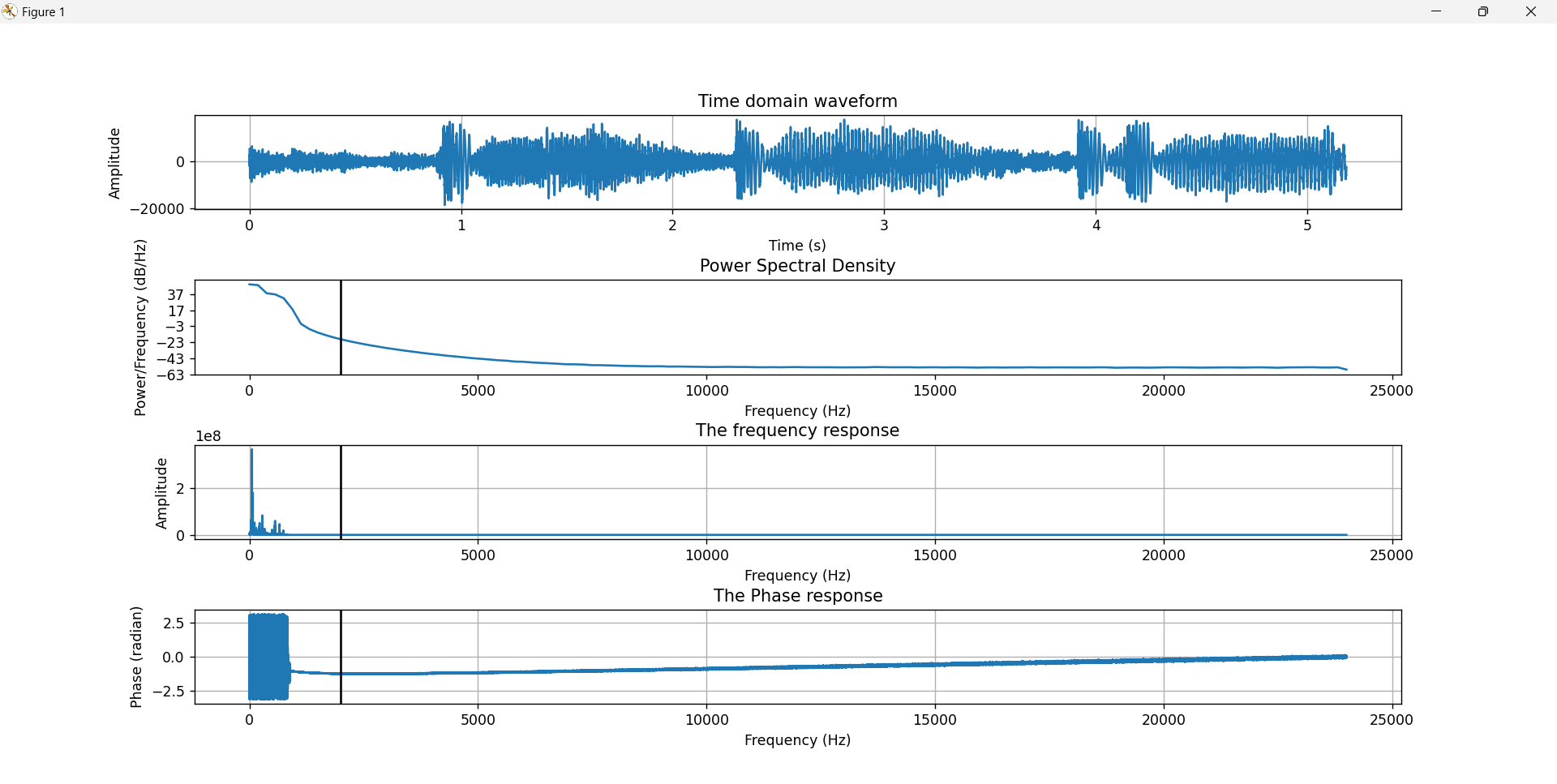
Dãy giá trị bộ lọc



Đồ thị cửa sổ Hanning với bậc N = 1001,,



Dãy đáp ứng xung của bộ lọc



Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha theo miền tần số.

* Nhận xét: Bộ lọc thông thấp cho phép tín hiệu có tần số nhỏ hơn hoặc bằng tần số cắt đi qua, và chặn các tín hiệu có tần số lớn hơn tần số cắt.

1. **Bộ lọc BPF:**

* Code minh họa: Các tham số truyền vào bao gồm: N - bậc của bộ lọc, fc1 và fc2 là các tần số cắt thứ nhất và thứ hai , fs - tần số lấy mẫu, được tính bằng tần số nyquist = tần số lấy mẫu của tín hiệu âm thanh / 2, tham số lamda.

def BPF(N, fc1 = 2000, fc2 = 5000, fs = 8000, lamda = 0):

deltaW = 4 \* np.pi / N

fc\_norm1 = fc1 / fs

fc\_norm2 = fc2 / fs

wc\_norm1 = 2 \* np.pi \* fc\_norm1

wc\_norm2 = 2 \* np.pi \* fc\_norm2

alpha = (N - 1) // 2

n = np.arange(N)

h, h1, h2 = np.zeros\_like(n, dtype=float), np.zeros\_like(n, dtype=float), np.zeros\_like(n, dtype=float)

h1[n == 0] = 2 \* fc\_norm1

h1[n != 0] = (2 \* fc\_norm1) \* np.sinc(2 \* fc\_norm1 \* (n[n != 0] - (N - 1) / 2))

h2[n == 0] = 2 \* fc\_norm2

h2[n != 0] = (2 \* fc\_norm2) \* np.sinc(2 \* fc\_norm2 \* (n[n != 0] - (N - 1) / 2))

h = h2 - h1

windowx = window\_function.hanning(N)

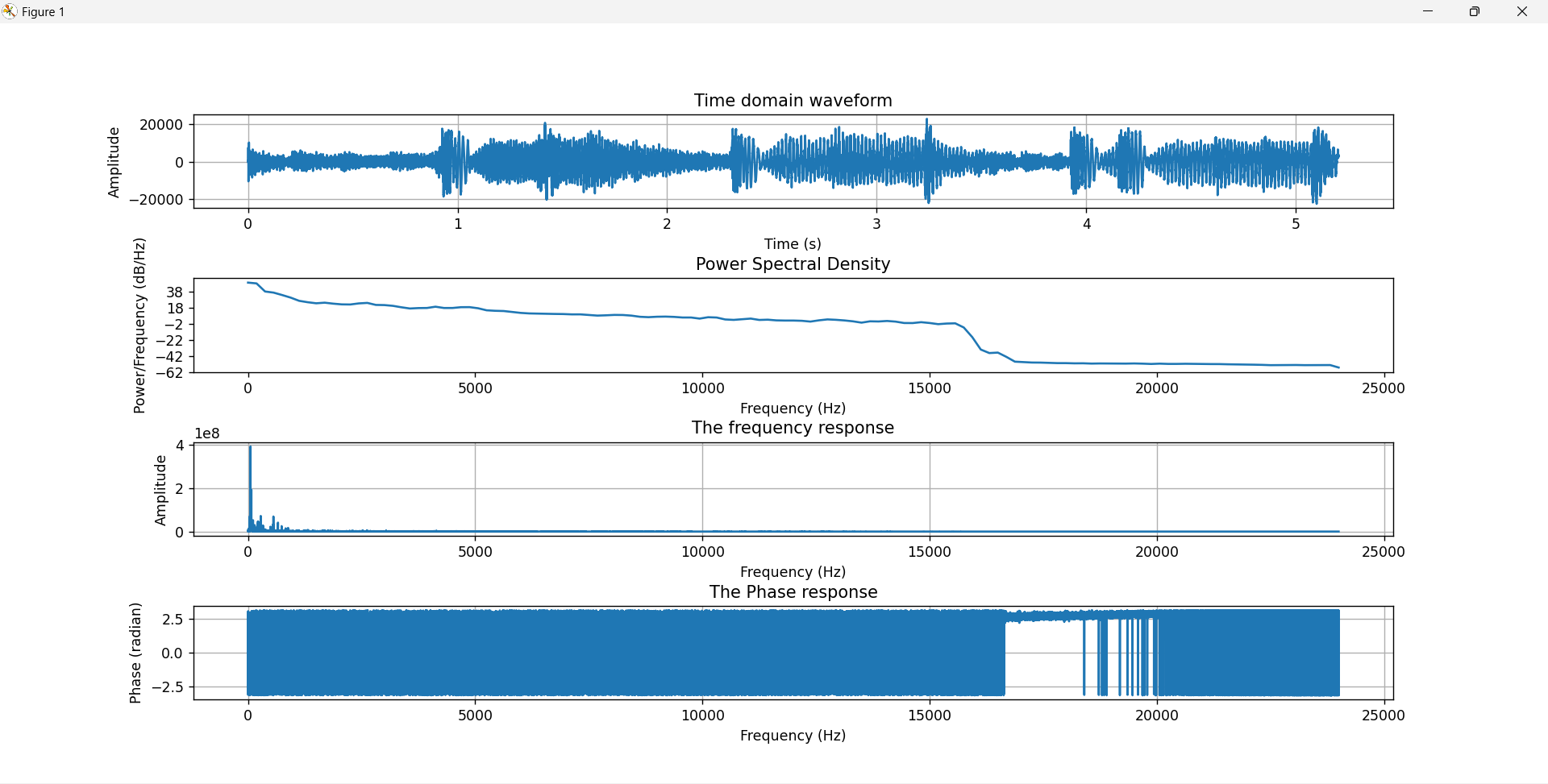
h\_d = windowx \* h

window\_function.plotWindow(h\_d, len(h\_d))

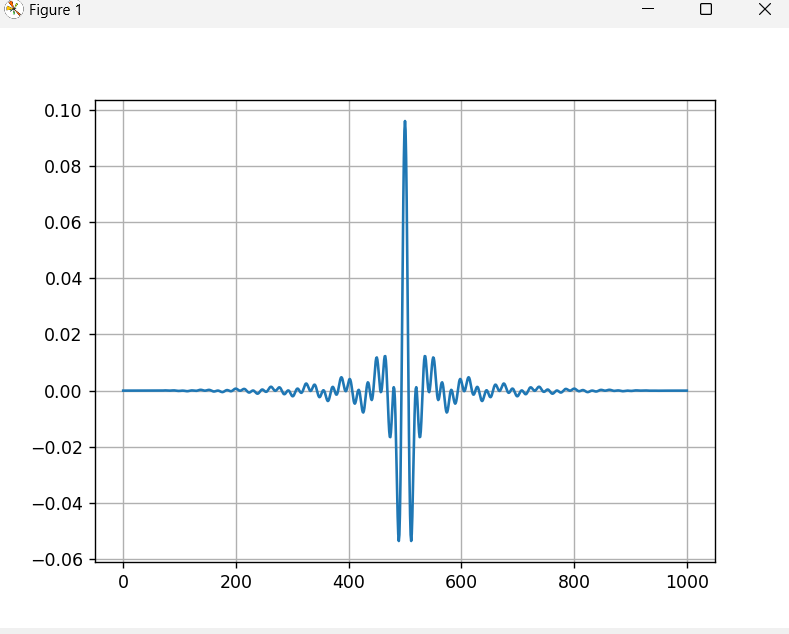
return h\_d

Bộ lọc BPF

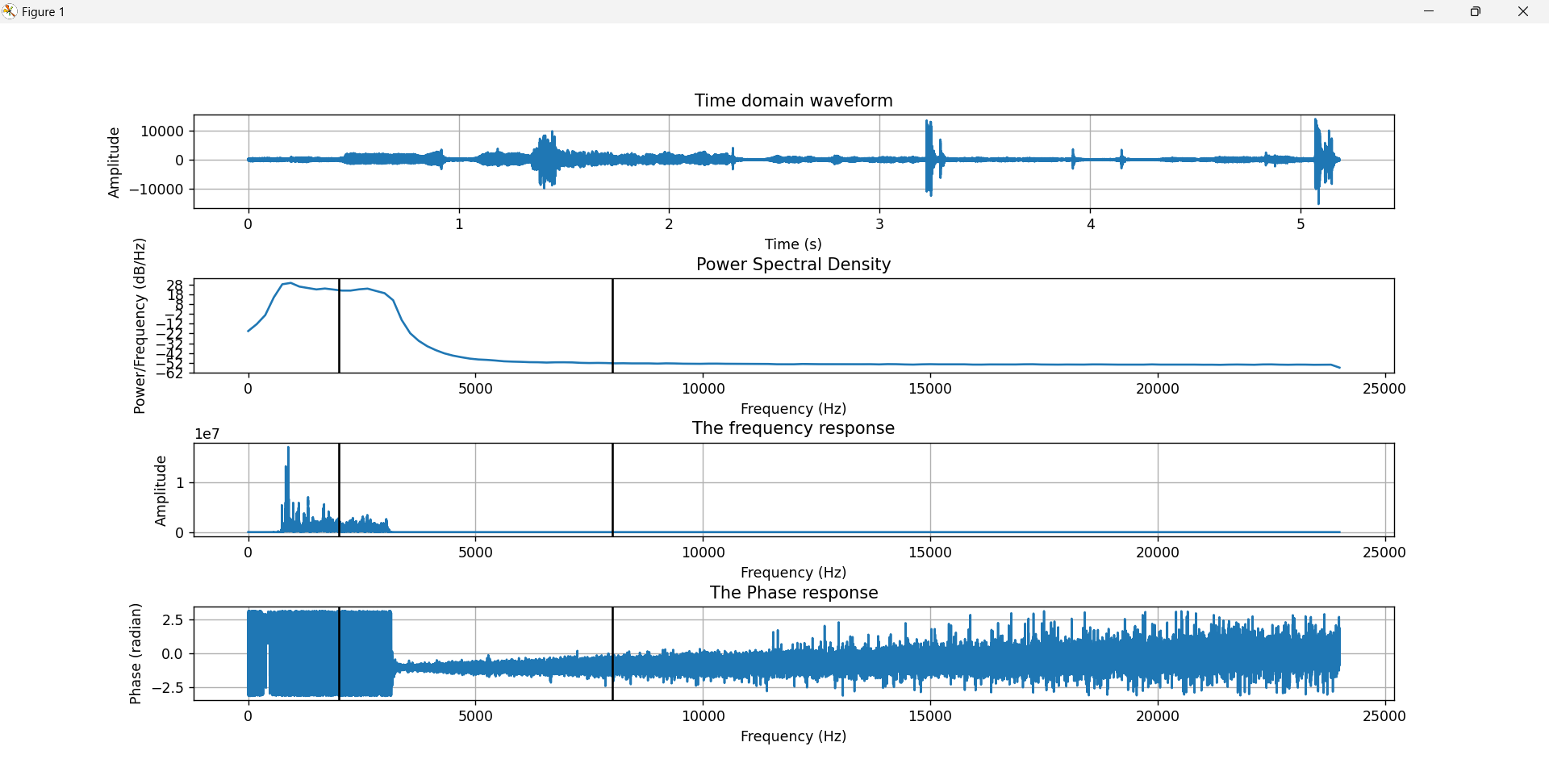
* Test:



Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha phổ theo miền tần số.



Dãy đáp ứng xung bộ lọc



Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha theo miền tần số.

* Nhận xét: Bộ lọc thông dải cho phép tín hiệu có tần số trong khoảng tần số thứ nhất đến tần số thứ 2 đi qua và chặn tín hiệu trong các trường hợp còn lại.

1. **Bộ lọc HPF:**

* Code minh họa: Các tham số truyền vào bao gồm: N - bậc của bộ lọc, fc - tần số cắt, fs - tần số lấy mẫu, được tính bằng tần số nyquist = tần số lấy mẫu của tín hiệu âm thanh / 2, tham số lamda.

def HPF(N, fc = 2000, fs = 8000, lamda = 0):

deltaW = 4 \* np.pi / N

fc\_norm = fc / fs

wc\_norm = 2 \* np.pi \* fc\_norm

alpha = (N - 1) // 2

n = np.arange(N)

h = np.zeros\_like(n, dtype=float)

h[n == 0] = 1 - 2 \* fc\_norm

h[n != 0] = - (2 \* fc\_norm) \* np.sinc(2 \* fc\_norm \* (n[n != 0] - (N - 1) / 2))

windowx = window\_function.hanning(N)

h\_d = windowx \* h

window\_function.plotWindow(h, len(h))

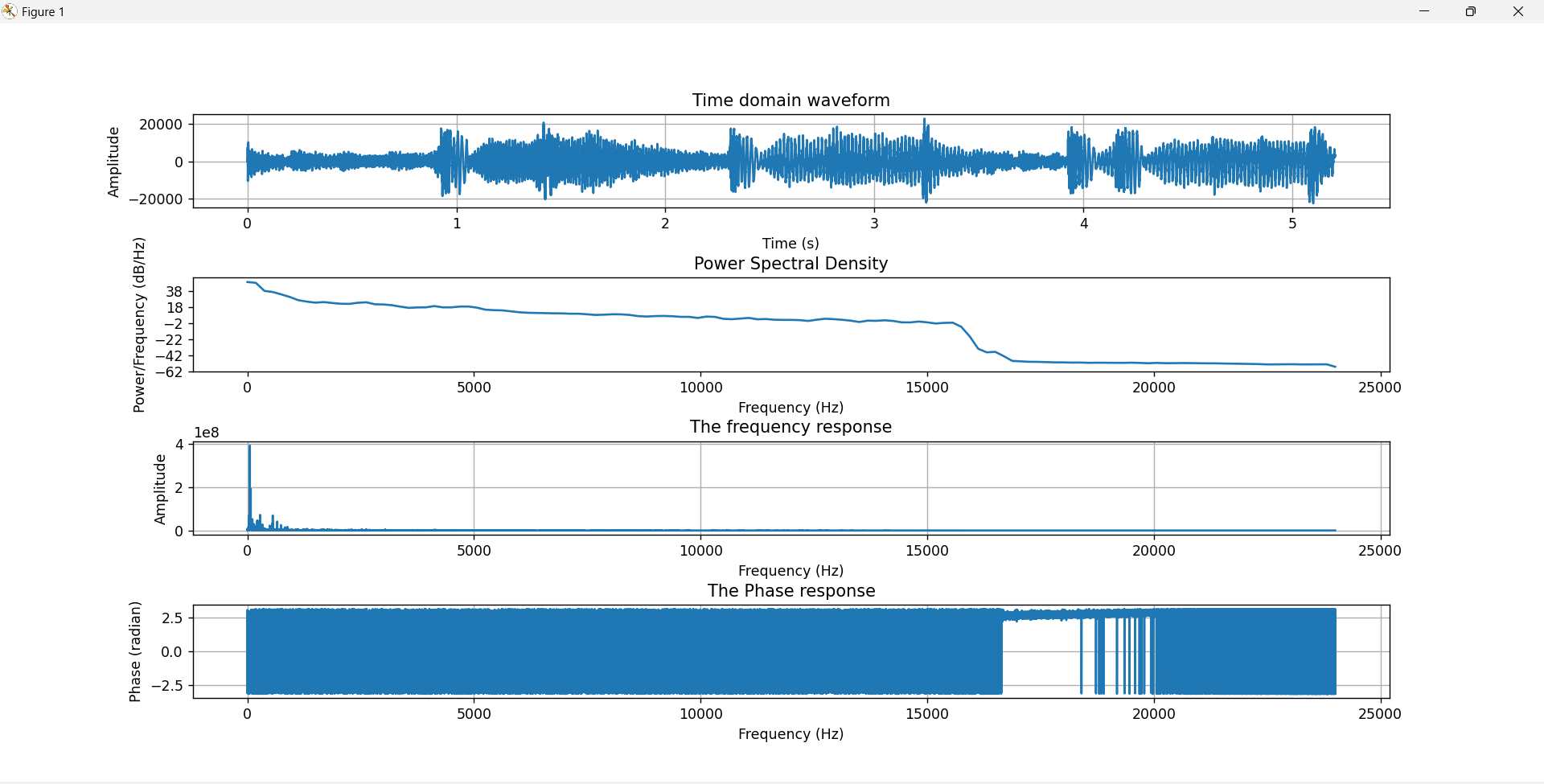
window\_function.plotWindow(windowx, len(windowx))

window\_function.plotWindow(h\_d, len(h\_d))

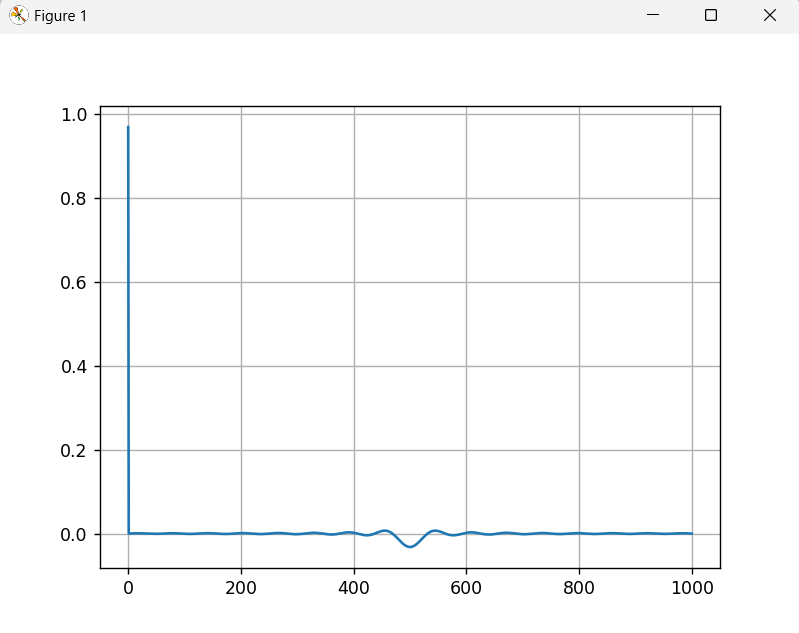
return h\_d

Bộ lọc HPF

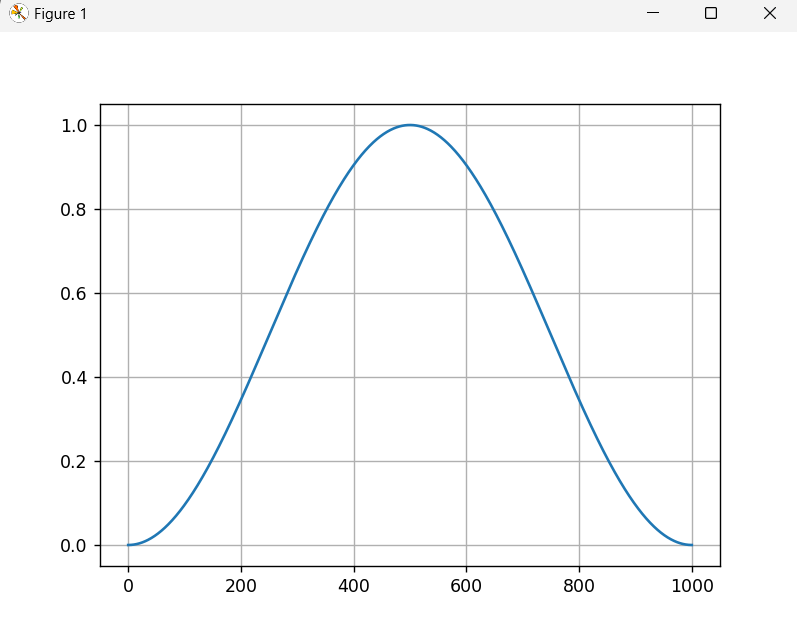
* Test:



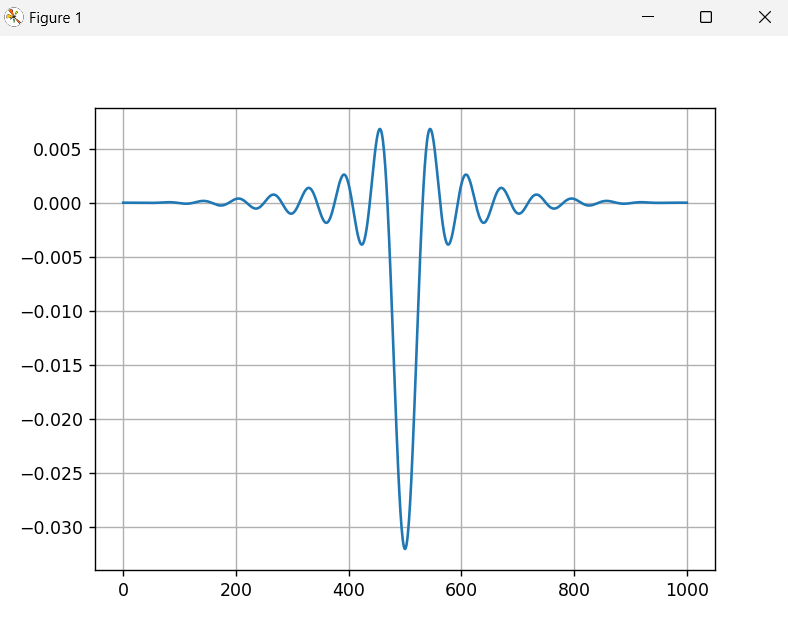
Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha phổ theo miền tần số.



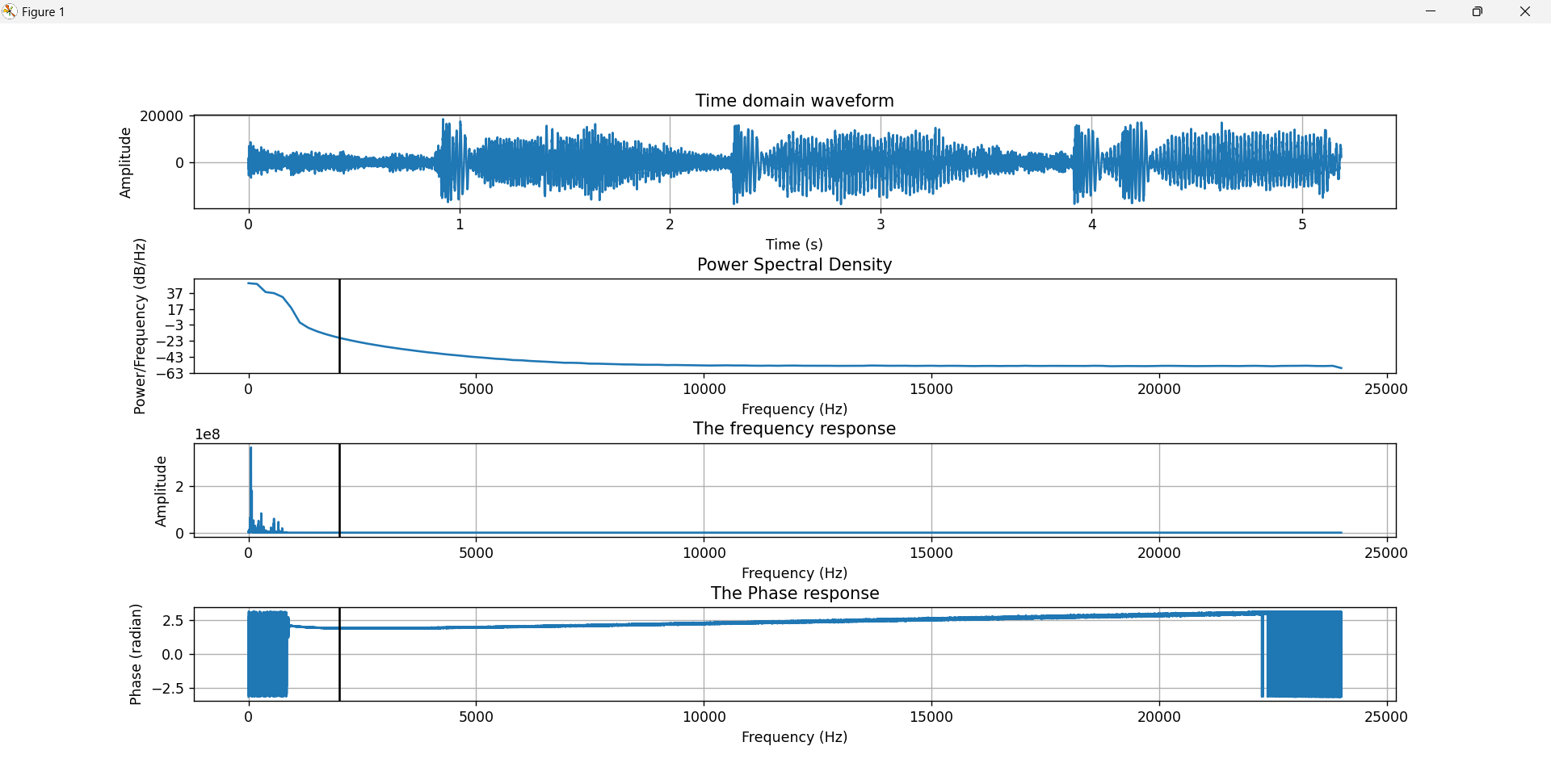
Dãy giá trị bộ lọc



Đồ thị cửa sổ Hanning với bậc N = 1001



Dãy đáp ứng xung bộ lọc



Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha theo miền tần số.

* Nhận xét: Bộ lọc thông thấp cho phép tín hiệu có tần số nhỏ hơn hoặc bằng tần số cắt đi qua, và chặn các tín hiệu có tần số lớn hơn tần số cắt.

1. **Bộ lọc BSF:**

* Code: Các tham số truyền vào bao gồm: N - bậc của bộ lọc, fc1 và fc2 là các tần số cắt thứ nhất và thứ hai , fs - tần số lấy mẫu, được tính bằng tần số nyquist = tần số lấy mẫu của tín hiệu âm thanh / 2, tham số lamda.

def BSF(N, fc1 = 2000, fc2 = 5000, fs = 8000, lamda = 0):

deltaW = 4 \* np.pi / N

fc\_norm1 = fc1 / fs

fc\_norm2 = fc2 / fs

wc\_norm1 = 2 \* np.pi \* fc\_norm1

wc\_norm2 = 2 \* np.pi \* fc\_norm2

alpha = (N - 1) // 2

n = np.arange(N)

h, h1, h2 = np.zeros\_like(n, dtype=float), np.zeros\_like(n, dtype=float), np.zeros\_like(n, dtype=float)

h1[n == 0] = 1 - 2 \* fc\_norm1

h1[n != 0] = - (2 \* fc\_norm1) \* np.sinc(2 \* fc\_norm1 \* (n[n != 0] - (N - 1) / 2))

h2[n == 0] = 1 - 2 \* fc\_norm2

h2[n != 0] = - (2 \* fc\_norm2) \* np.sinc(2 \* fc\_norm2 \* (n[n != 0] - (N - 1) / 2))

h = h2 - h1

windowx = window\_function.hanning(N)

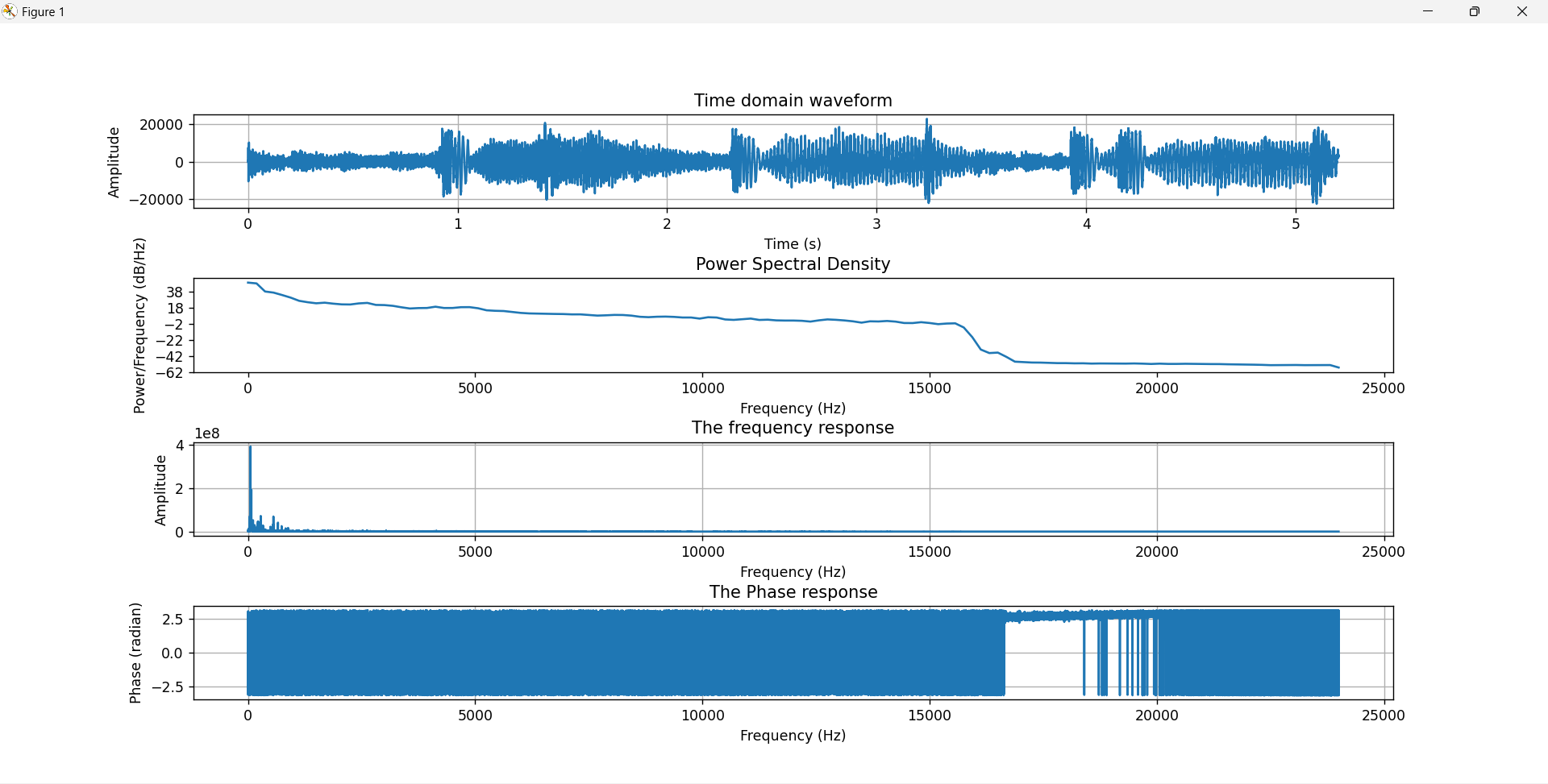
h\_d = windowx \* h

window\_function.plotWindow(h\_d, len(h\_d))

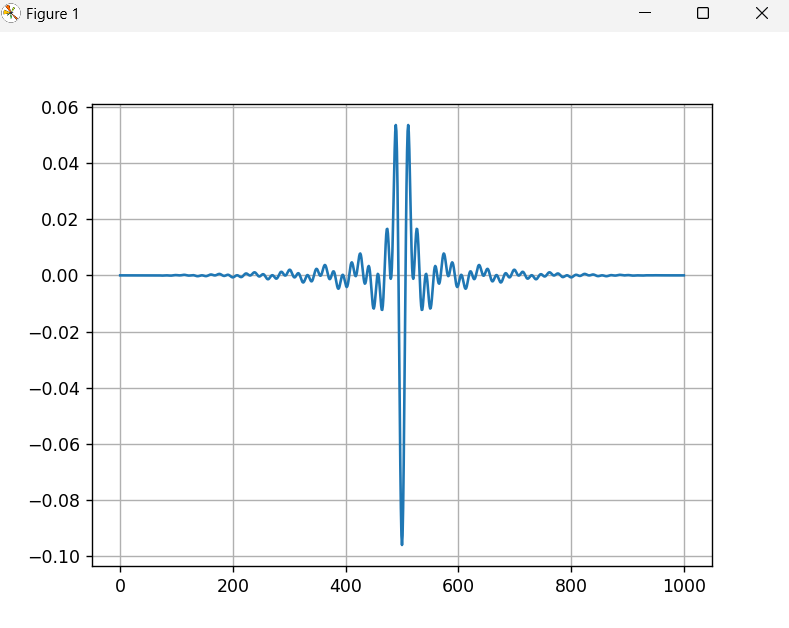
return h\_d

Bộ lọc BSF

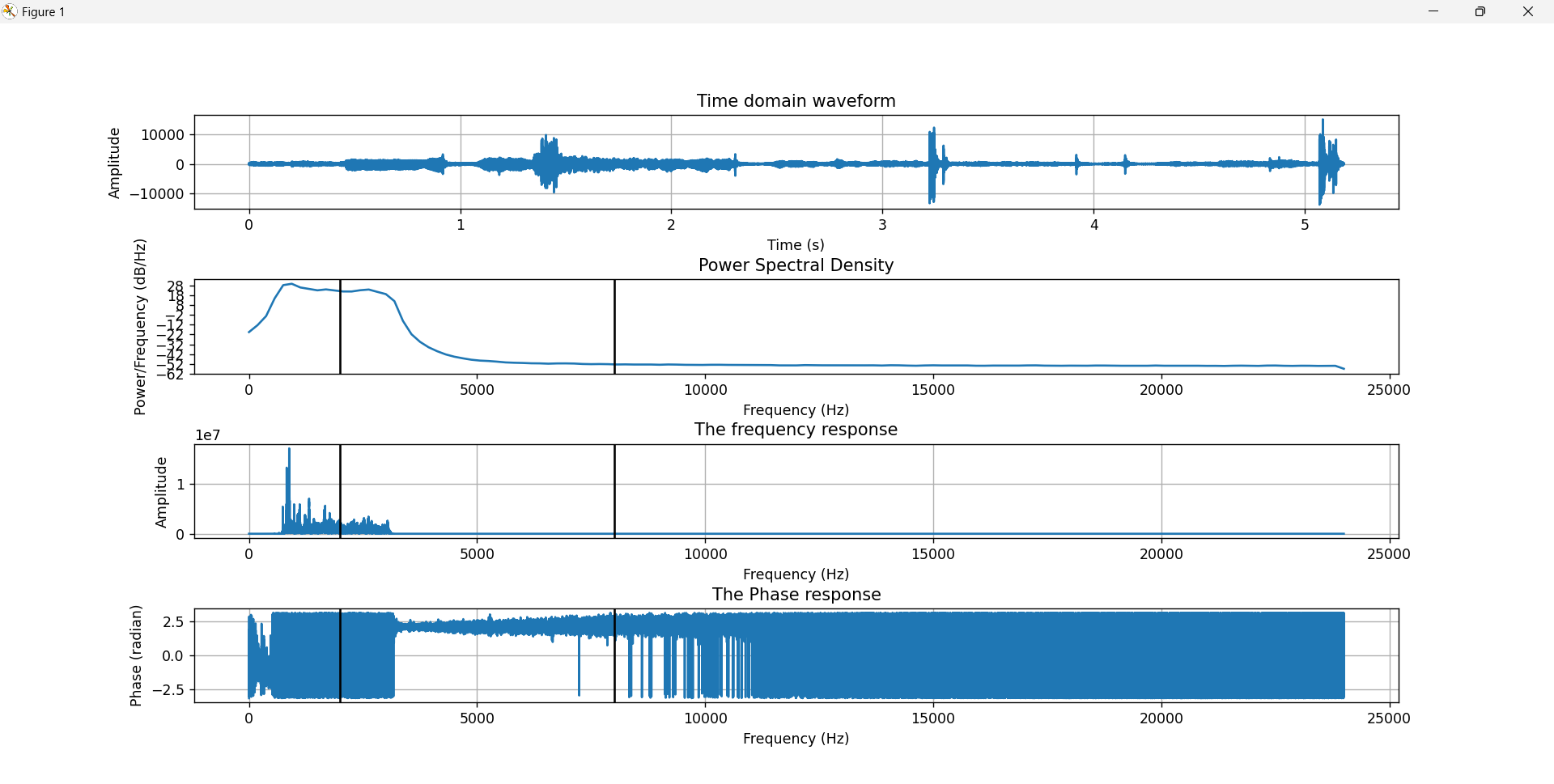
* Test:



Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha phổ theo miền tần số.



Dãy đáp ứng xung bộ lọc



Đồ thị hiển thị biên độ theo thời gian, năng lượng, biên độ phổ theo miền tần số, pha theo miền tần số.

* Nhận xét: Bộ lọc chắn dải cho phép tín hiệu có tần số trong khoảng tần số thứ nhất đến tần số thứ 2 đi qua và chặn tín hiệu trong các trường hợp còn lại.

1. **Lọc âm thanh(EQUALIZER)**
2. **Hiệu ứng âm thanh:** Các hiệu ứng như Echo, Reversing, Modulation, Faded
3. **Echo:**

Echo là hiện tượng âm thanh được phát ra và phản xạ trở lại từ các bề mặt xung quanh, tạo ra một phiên bản bị trễ của âm thanh gốc. Hiện tượng này có thể được tạo ra bằng cách thêm một bản sao trễ của tín hiệu âm thanh vào tín hiệu gốc. Kỹ thuật này được sử dụng trong nhiều ứng dụng âm nhạc, từ tạo hiệu ứng phòng karaoke đến tạo ra những bản nhạc với hiệu ứng echo đặc biệt.

def echo(filename, time\_delay):

wav\_obj = wave.open(str(filename), 'rb')

sample\_freq = wav\_obj.getframerate() # tần số lấy mẫu

n\_samples = wav\_obj.getnframes()

t\_audio = n\_samples / sample\_freq

n\_channels = wav\_obj.getnchannels()

signal\_wave = wav\_obj.readframes(n\_samples)

signal\_array = np.frombuffer(signal\_wave, dtype=np.int16)

output\_audio = np.zeros(len(signal\_array))

output\_delay = time\_delay \* sample\_freq

for count, e in enumerate(signal\_array):

output\_audio[count] = e + signal\_array[count - int(output\_delay)]

signal\_array = output\_audio

filtered\_audio\_frames = signal\_array.astype(np.int16).tobytes()

filtered\_audio\_file = wave.open('echo' + filename, 'wb')

filtered\_audio\_file.setnchannels(n\_channels)

filtered\_audio\_file.setsampwidth(wav\_obj.getsampwidth())

filtered\_audio\_file.setframerate(sample\_freq)

filtered\_audio\_file.setnframes(n\_samples)

filtered\_audio\_file.writeframes(filtered\_audio\_frames)

filtered\_audio\_file.close()

1. **Faded:**

Hiệu ứng faded trong âm thanh là một kỹ thuật âm thanh được sử dụng để giảm dần hoặc tăng dần âm lượng hoặc độ mạnh của âm thanh theo một cách liên tục, thường được sử dụng để tạo ra hiệu ứng mượt mà hoặc thu hút sự chú ý của người nghe. Hiệu ứng faded thường được sử dụng trong nhiều loại âm thanh, bao gồm các bản nhạc, bản ghi âm, các phim truyền hình, video game và các dự án đa phương tiện khác.

def faded(filename, alpha, time, type='in'):

wav\_obj = wave.open(str(filename), 'rb')

sample\_freq = wav\_obj.getframerate() # tần số lấy mẫu

n\_samples = wav\_obj.getnframes()

t\_audio = n\_samples / sample\_freq

n\_channels = wav\_obj.getnchannels()

signal\_wave = wav\_obj.readframes(n\_samples)

signal\_array = np.frombuffer(signal\_wave, dtype=np.int16)

# type bao gồm in & out

if type == 'in':

fade\_samples = time \* sample\_freq

fade\_in\_curve = alpha \* np.linspace(0, 1, fade\_samples)

signa1 = np.array(signal\_array[0:fade\_samples])

signa2 = np.array(signal\_array[fade\_samples:])

one = np.ones(len(signal\_array) - len(fade\_in\_curve))

signa3 = signa2 \* one

signa4 = signa1 \* fade\_in\_curve

signal\_array = np.concatenate([signa4, signa3])

else:

fade\_samples = time \* sample\_freq

fade\_in\_curve = alpha \* np.linspace(1, 0, fade\_samples)

signa1 = np.array(signal\_array[0:len(signal\_array) - fade\_samples])

signa2 = np.array(signal\_array[len(signal\_array) - fade\_samples:])

one = np.ones(len(signal\_array) - len(fade\_in\_curve))

signa3 = signa1 \* one

signa4 = signa2 \* fade\_in\_curve

signal\_array = np.concatenate([signa3, signa4])

filtered\_audio\_frames = signal\_array.astype(np.int16).tobytes()

filtered\_audio\_file = wave.open('faded\_' + type + filename, 'wb')

filtered\_audio\_file.setnchannels(n\_channels)

filtered\_audio\_file.setsampwidth(wav\_obj.getsampwidth())

filtered\_audio\_file.setframerate(sample\_freq)

filtered\_audio\_file.setnframes(n\_samples)

filtered\_audio\_file.writeframes(filtered\_audio\_frames)

filtered\_audio\_file.close()

1. **Reversing:**

Hiệu ứng reversing (hoặc còn gọi là hiệu ứng đảo ngược) trong âm thanh là một kỹ thuật âm thanh được sử dụng để đảo ngược thứ tự của các đoạn âm thanh, từ âm đến âm hoặc từ tiếng đến tiếng. Hiệu ứng reversing thường được sử dụng để tạo ra một hiệu ứng âm thanh độc đáo, gây sự chú ý và tạo ra một không gian âm thanh độc đáo.

def reversing(filename):

wav\_obj = wave.open(str(filename), 'rb')

sample\_freq = wav\_obj.getframerate() # tần số lấy mẫu

n\_samples = wav\_obj.getnframes()

t\_audio = n\_samples / sample\_freq

n\_channels = wav\_obj.getnchannels()

signal\_wave = wav\_obj.readframes(n\_samples)

signal\_array = np.frombuffer(signal\_wave, dtype=np.int16)

signal\_array = signal\_array[::-1]

filtered\_audio\_frames = signal\_array.astype(np.int16).tobytes()

filtered\_audio\_file = wave.open('reversing' + filename, 'wb')

filtered\_audio\_file.setnchannels(n\_channels)

filtered\_audio\_file.setsampwidth(wav\_obj.getsampwidth())

filtered\_audio\_file.setframerate(sample\_freq)

filtered\_audio\_file.setnframes(n\_samples)

filtered\_audio\_file.writeframes(filtered\_audio\_frames)

filtered\_audio\_file.close()

1. **Modulation:**

Hiệu ứng modulation là một kỹ thuật âm thanh được sử dụng để thay đổi các thuộc tính của một tín hiệu âm thanh gốc. Nó có thể thay đổi biên độ, tần số, pha, hay thời gian của tín hiệu âm thanh để tạo ra các hiệu ứng âm thanh độc đáo và phong phú.

def modulation(filename, alpha, wc = 20):

wav\_obj = wave.open(str(filename), 'rb')

sample\_freq = wav\_obj.getframerate() # tần số lấy mẫu

n\_samples = wav\_obj.getnframes()

t\_audio = n\_samples / sample\_freq

n\_channels = wav\_obj.getnchannels()

signal\_wave = wav\_obj.readframes(n\_samples)

signal\_array = np.frombuffer(signal\_wave, dtype=np.int16)

lm = np.array([1 + alpha \* np.cos(wc \* i) for i in range(len(signal\_array))])

signal\_array = signal\_array \* lm

# ghi vào file

filtered\_audio\_frames = signal\_array.astype(np.int16).tobytes()

filtered\_audio\_file = wave.open('modulation' + filename, 'wb')

filtered\_audio\_file.setnchannels(n\_channels)

filtered\_audio\_file.setsampwidth(wav\_obj.getsampwidth())

filtered\_audio\_file.setframerate(sample\_freq)

filtered\_audio\_file.setnframes(n\_samples)

filtered\_audio\_file.writeframes(filtered\_audio\_frames)

filtered\_audio\_file.close()

**V. Nhận xét**

Với bài tập lớn lần này, nhóm đã thực hiện cài đặt thành công các cửa sổ, các bộ lọc cơ bản, các hiệu ứng âm thanh cơ bản

--------HẾT --------