BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP HỒ CHÍ MINH

Khoa: Công Nghệ Thông Tin

LAB REPORT 03

Student's ID :
Student's name : Hồ Phúc Lâm
Subject : PTHTDPT
Instructor : Nguyễn Thành Thái
Faculty : Công Nghệ Thông Tin
Completed Date : 04/09/2024

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Điểm lab report:

LAB 2.2 AUDIO PROCESSING

1)Mục đích yêu cầu :

+Củng cố kiến thức về SIGNAL, AUDIO, sampling, sampling-rate, bit-rate,…

+Tiếp cận thư viện xử lý tín hiệu Audio, các API hỗ trợ,…

2)Tài liệu tham khảo

https://www.pythonforengineers.com/audio-and-digital-signal-processingdsp-in-python/

{thực hành}

- Trả lời các câu hỏi sau:
  - Tần số là gì?
  - Tốc độ lấy mẫu là gì?
  - Tốc độ truyền là gì?
  - Viết công thức của hàm sin và cho biết các đại lượng tương ứng
  - Giải thích đầy đủ các tham số.
  - Viết code(ghi chú giải thích ý nghĩa dòng code), chạy thử kiểm tra hoàn chỉnh kết quả.
  - Nộp code hoàn chỉnh vào file báo cáo

https://docs.python.org/3/library/mm.html (tham khảo)

https://matplotlib.org/tutorials/introductory/pyplot.html {thư viện tham khảo}

3) Công cụ : Python programming language

a)Python IDE hoặc PYTHON command line (trên LAPTOP)

b)Cài đặt các thư viện hỗ trợ :

Câu lệnh cài đặt : pip install <gói cài đặt>

Vd : pip install numpy

pip install simpleaudio

pip install matplotlib

4)Thực hiện: theo hướng dẫn trong LINK

- Cài đặt thư viện (nếu chưa có): mở CMD trên Anaconda, gõ conda install <thư viện cài đặt>
- Mở Spyder (hoặc VScode) viết chương trình python lưu trong một thư mục{thường là chung với thư mục của file Wav)
- Chạy thử từng dòng lệnh
- Lưu các bài tập trong thư mục, nén và nộp (cuối giờ thực hành)

I. Lý thuyết

1. Tần số là gì?

Tần số (Frequency): là số lần một sự kiện lặp lại trong một khoảng thời gian nhất định. Trong xử lý tín hiệu, tần số thường đề cập đến số lần sóng (hoặc chu kỳ) hoàn thành trong một giây, và đơn vị của nó là Hertz (Hz).

Tần số là số lần sóng sin lặp lại trong một giây. Ở đây tác giả sẽ sử dụng tần số 1 KHz.

2. Tốc độ lấy mẫu là gì?

Tốc độ lấy mẫu (Sampling rate): là số lần tín hiệu được lấy mẫu trong một giây. Nó xác định mức độ chi tiết mà tín hiệu gốc có thể được tái tạo từ các mẫu. Tốc độ lấy mẫu được do bằng Hertz (Hz).

Hầu hết các tín hiệu trong thế giới thực là tín hiệu Analog hay còn gọi là tín hiệu tương tự hay tín hiệu liên tục, trong khi đó máy tính là tín hiệu số (Digital). Vì vậy, chúng ta cần một bộ chuyển đổi tín hiệu số. Tác giả sẽ sử dụng giá trị 48000, đây là giá trị được sử dụng trong thiết bị âm thanh chuyên nghiệp

3. Tốc độ truyền là gì?

Tốc độ truyền (Bit-rate): là lưu lượng được truyền trong một khoảng thời gian cụ thể, thường được đo bằng bit trên giây (bit per second | bps). Trong xử lý tín hiệu số, tốc độ truyền có thể liên qun đến việc truyền tín hiệu qua một kênh truyền thông, và nó ảnh hưởng đến chất lượng và tốc độ truyền tải dữ liệu.

4. Viết công thức của hàm sin và cho biết các đại lượng tương ứng

Công thức sóng sin:

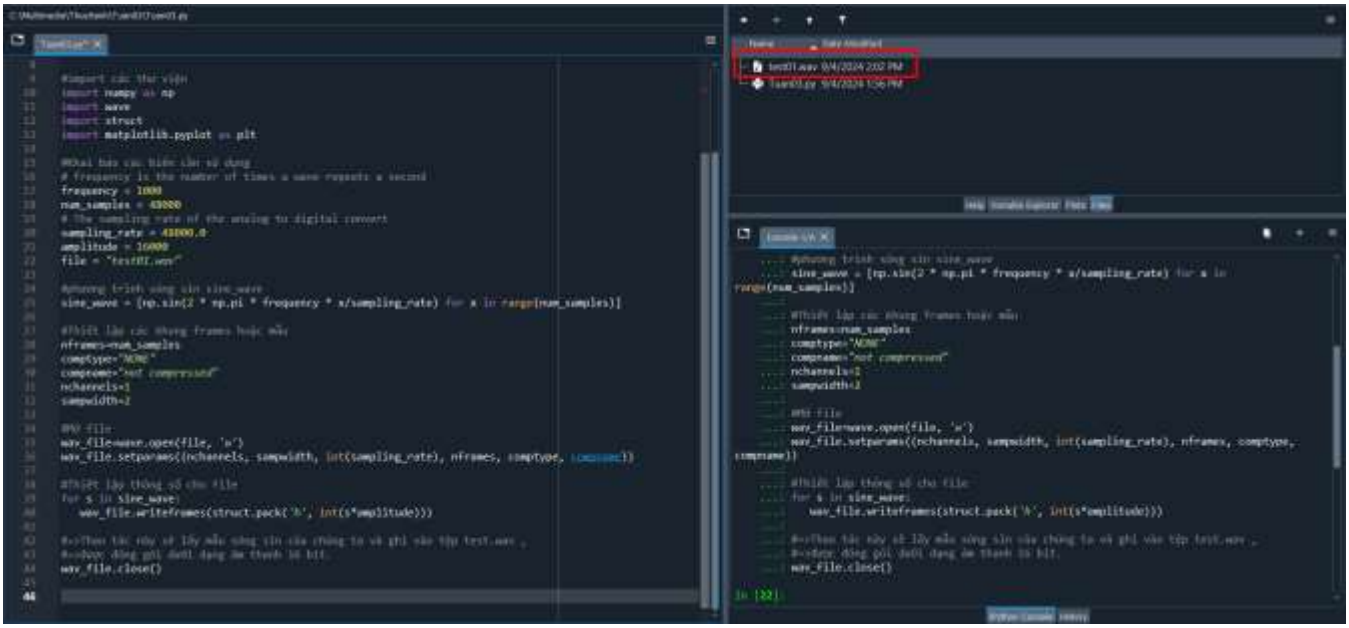
$$y(t) = A * sin(2 * pi * f * t)$$

Trong đó:

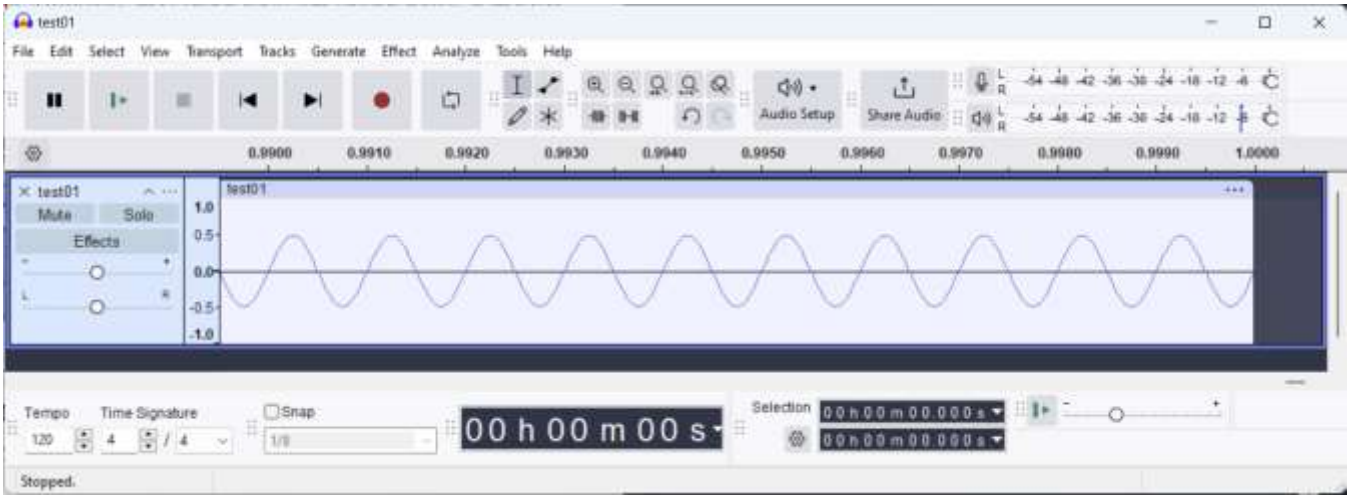
- A là biên độ (amplitude) của sóng, đại diện cho độ cao của sóng.
- f là tần số (frequency) của sóng, số chu kỳ mỗi giây.
- t là thời gian (time).
- pi là $\pi = 3.1415$
- ở đây tác giả sử dụng biên độ là 16000

5. Thực hành

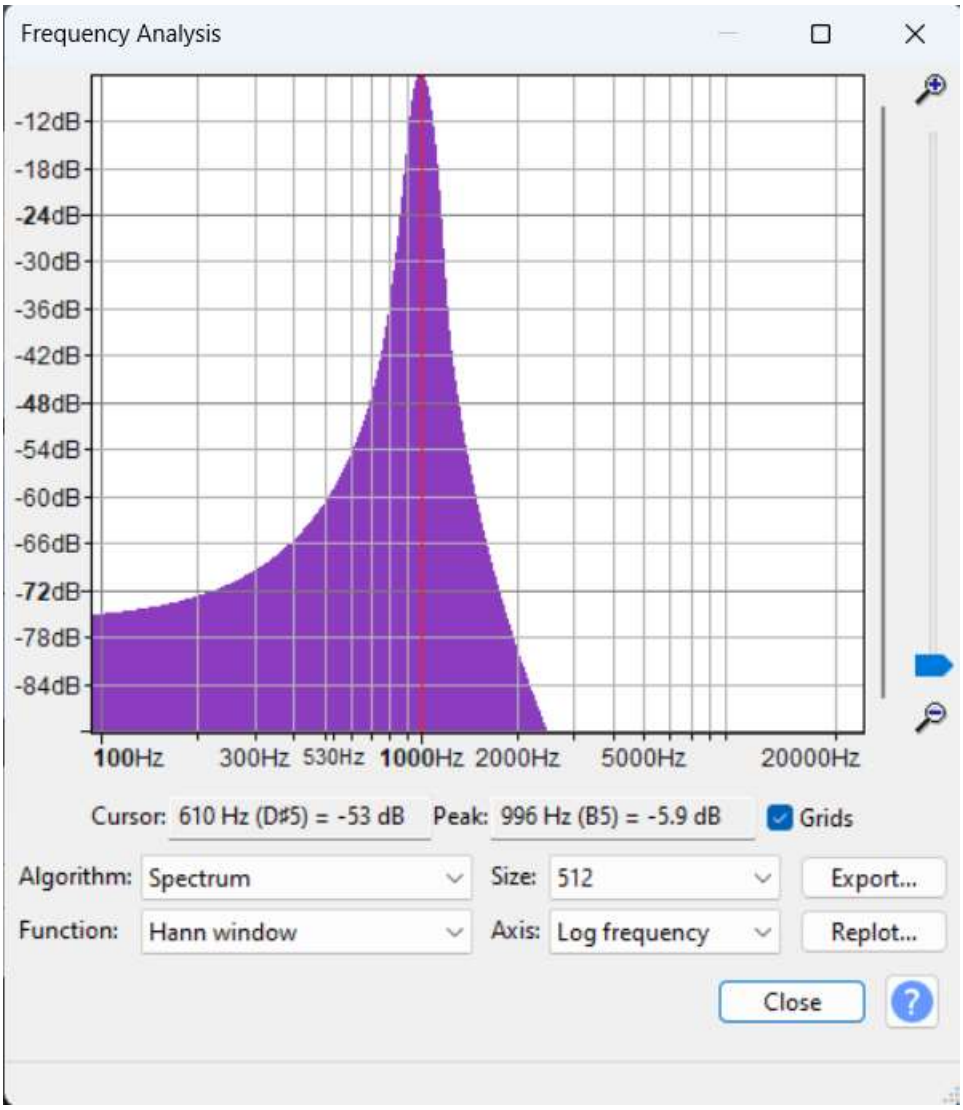
Thao tác này sẽ lấy mẫu sóng sin của chúng ta và ghi vào tệp *test.wav* , được đóng gói dưới dạng âm thanh 16 bit.



Cài đặt **Audacity** tại https://www.audacityteam.org/ và kiểm tra tần số âm thanh có đúng không?



Vào Edit->Select All (hoặc nhấn Ctrl A), sau đó vào Analyse->Plot Spectrum.



Có thể thấy rằng đỉnh ở khoảng 1000 Hz, đó là cách chúng ta tạo ra tệp sóng của mình.

Sử dụng DFT để có được tần số

Hàm numpy abs() sẽ lấy tín hiệu phức tạp của chúng ta và tạo ra phần thực của nó.

Nếu chúng ta muốn tìm phần tử mảng có giá trị cao nhất, chúng ta có thể tìm bằng cách

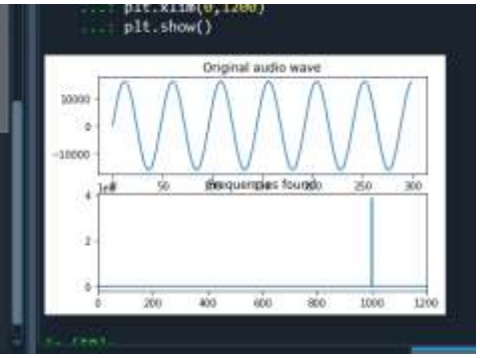
Side Detour – Đường vòng

```
#Side Detour - Đường vòng
data_fft = np.fft.fft(sine_wave)
abs(data_fft[0])
abs(data_fft[1])
abs(data_fft[1000])
```

```
In [51]: data_fft = np.fft.fft(sine_wave)

In [52]: abs(data_fft[0])
Out[52]: 8.55655965377196e-13

In [53]: abs(data_fft[1])
Out[53]: 9.94155621117267e-12

In [54]: abs(data_fft[1000])
Out[54]: 23999.999999999996
```

Vẽ biểu đồ dữ liệu

```
#vẽ biểu đồ dữ liệu
plt.subplot(2,1,1)
plt.plot(data[:300])
plt.title("Original audio wave")
plt.subplot(2,1,2)
plt.plot(frequencies)
plt.title("Frequencies found")
plt.xlim(0,1200)
plt.show()
```

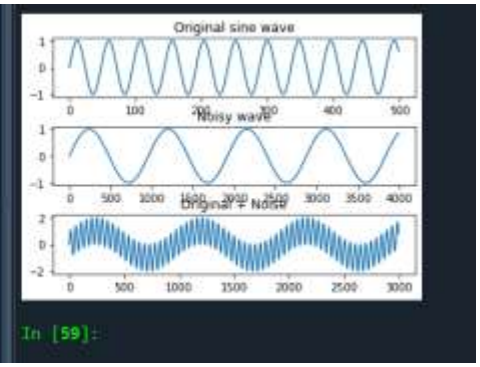


Cleaning a noisy sine wave – Loại bỏ sóng sin nhiễu

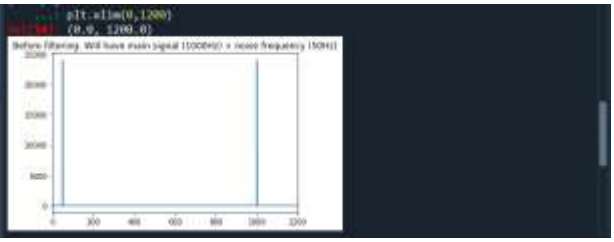
```
# Loại bỏ sóng sin nhiễu - Cleaning a noisy sine wave
# frequency is the number of times a wave repeats a second
frequency = 1000
noisy_freq = 50
num_samples = 48000
# The sampling rate of the analog to digital convert
sampling_rate = 48000.0

#Tần số chính là 1000Hz và chúng ta sẽ thêm vào đó một tiếng ồn 50Hz.
#Create the sine wave and noise
sine_wave = [np.sin(2 * np.pi * frequency * x1 / sampling_rate) for x1 in range(num_samples)]
sine_noise = [np.sin(2 * np.pi * noisy_freq * x1/  sampling_rate) for x1 in range(num_samples)]
#Convert them to numpy arrays
sine_wave = np.array(sine_wave)
sine_noise = np.array(sine_noise)
```

-vẽ biểu đồ dữ liệu

```
#vẽ biểu đồ dữ liệu
plt.subplot(3,1,1)
plt.title("Original sine wave")
# Need to add empty space, else everything looks scrunched up!
plt.subplots_adjust(hspace=.5)
plt.plot(sine_wave[:500])
plt.subplot(3,1,2)
plt.title("Noisy wave")
plt.plot(sine_noise[:4000])
plt.subplot(3,1,3)
plt.title("Original + Noise")
plt.plot(combined_signal[:3000])
plt.show()
```



```
plt.show()


data_fft = np.fft.fft(combined_signal)
freq = (np.abs(data_fft[:len(data_fft)]))
plt.plot(freq)
plt.title("Before filtering: Will have main signal (1000Hz) + noise frequency (50Hz)")
plt.xlim(0,1200)
```

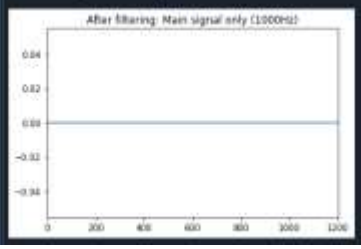


```
filtered_freq = []
index = 0
for f in freq:
    # Filter between lower and upper limits
    # Choosing 950, as closest to 1000. In real world, won't get exact numbers like th
    if index > 950 and index < 1050:
        # Has a real value. I'm choosing >1, as many values are like 0.000000001 etc
        if f > 1:
            filtered_freq.append(f)

        else:
            filtered_freq.append(0)
    else:
        filtered_freq.append(0)
index += 1
```

```
In [62]: filtered_freq = []
....: index = 0
....: for f in freq:
....:     # Filter between lower and upper limits
....:     # Choosing 950, as closest to 1000. In real world, won't get exact numbers like these
....:     if index > 950 and index < 1050:
....:         # Has a real value. I'm choosing >1, as many values are like 0.000000001 etc
....:         if f > 1:
....:             filtered_freq.append(f)
....:
....:         else:
....:             filtered_freq.append(0)
....:     else:
....:         filtered_freq.append(0)
....: index += 1

In [63]:
```

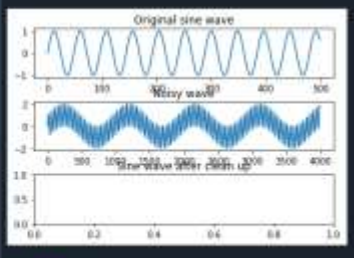
```
filtered_freq = []
index = 0
for f in freq:
    # Filter between lower and upper limits
    # Choosing 950, as closest to 1000. In real world, won't get exact numbers like th
    if index > 950 and index < 1050:
        # Has a real value. I'm choosing >1, as many values are like 0.000000001 etc
        if f > 1:
            filtered_freq.append(f)

        else:
            filtered_freq.append(0)
    else:
        filtered_freq.append(0)
index += 1


plt.plot(filtered_freq)
plt.title("After filtering: Main signal only (1000Hz)")
plt.xlim(0,1200)
plt.show()
plt.close()
```

```
0.000000001 etc
....:         if f > 1:
....:             filtered_freq.append(f)
....:
....:         else:
....:             filtered_freq.append(0)
....:     else:
....:         filtered_freq.append(0)
....: index += 1

In [63]: plt.plot(filtered_freq)
....: plt.title("After filtering: Main signal only (1000
....: plt.xlim(0,1200)
....: plt.show()
....: plt.close()
```




```
plt.subplot(3,1,1)

plt.title("Original sine wave")

# Need to add empty space, else everything looks scrunched up!
plt.subplots_adjust(hspace=.5)

plt.plot(sine_wave[:500])

plt.subplot(3,1,2)

plt.title("Noisy wave")

plt.plot(combined_signal[:4000])

plt.subplot(3,1,3)

plt.title("Sine wave after clean up")

plt.plot((recovered_signal[:500]))
plt.show()
```

```
....: plt.title("Sine wave after clean up")
....:
....: plt.plot((recovered_signal[:500]))
....: plt.show()
Traceback (most recent call last):

  Cell In[64], line 21
    plt.plot((recovered_signal[:500]))

NameError: name 'recovered_signal' is not defined
```




```
In [65]:
```