**TÌM TẦN SỐ CƠ BẢN CỦA TÍN HIỆU TRÊN MIỀN THỜI GIAN DÙNG HÀM TỰ TƯƠNG QUAN**

**Châu Trường Long, Hồ Nguyên Vũ**

**Nhóm 3, Lớp HP: 18N14A**

**Giảng viên hướng dẫn: TS. Ninh Khánh Duy**

|  |  |  |  |
| --- | --- | --- | --- |
| **Điểm** | **Bảng phân công nhiệm vụ** | | **Chữ ký của SV** |
|  | Châu Trường Long (nhóm trưởng) | * Đọc và tìm hiểu tài liệu * Lập trình giao diện * Cài đặt hàm tự tương quan, thuật toán tìm đỉnh, thuật toán xác định tần số cơ bản |  |
|  | Hồ Nguyên Vũ | * Đọc và tìm hiểu tài liệu * Viết báo cáo * Lập trình giao diện, cài đặt hàm lọc trung vị |  |

Lời cam đoan: Chúng tôi, gồm các sinh viên có chữ ký ở trên, cam đoan rằng báo cáo này là do chúng tôi tự viết dựa trên các tài liệu tham khảo liệt kê ở cuối báo cáo. Các số liệu thực nghiệm và mã nguồn chương trình nếu không chỉ dẫn nguồn tham khảo đều do chúng tôi tự làm. Nếu vi phạm thì chúng tôi xin chịu trách nhiệm và tuân theo xử lý của giáo viên hướng dẫn.

Tóm tắt: Tìm tần số cơ bản của tín hiệu là bài toán cơ bản nhưng rất cần thiết trong việc xử lý tín hệu, đặc biệt là tín hiệu tiếng nói. Trong báo cáo này bài toán đặt ra là đi thực hiện việc xác định tần số cơ bản của một tín hiệu âm thanh trên miền thời gian. Sử dụng ngôn ngữ lập trình python kèm các thư viện hỗ trợ xử lý tín hiệu. Từ một file âm thanh có định dạng \*.wav, chia tín hiệu âm thanh thành các khung cửa sổ có độ dài khoảng 20 – 60 ms, lần lượt cho các khung tín hiệu đi qua hàm tự tương quan, từ đó xác định được tần số cơ bản của khung tín hiệu khoảng 75 – 350 Hz. Kết quả cũng cho thấy có thể xác định được tần số cơ bản của tín hiệu tiếng nói trên miền thời gian.

Từ khóa:

“auto correlation”,

“fundamental frequency on time domain”,

“pitch contour example, image”,

“median filter algorithm”,

“fast fourier transform”,

“fft and ifft function”.

Mục lục

[I. ĐẶT VẤN ĐỀ 3](#_Toc57929360)

[II. CƠ SỞ LÝ THUYẾT 3](#_Toc57929361)

[A. Tần số cơ bản F0 là gì? 3](#_Toc57929362)

[B. Các loại tín hiệu tiếng nói 3](#_Toc57929363)

[*1.* Âm hữu thanh 3](#_Toc57929364)

[*2.* Âm Vô thanh 3](#_Toc57929365)

[C. Tự tương quan 4](#_Toc57929366)

[D. Lọc trung vị 5](#_Toc57929367)

[E. Chia cửa sổ và Hamming 6](#_Toc57929368)

[III. MÃ CHƯƠNG TRÌNH CÀI ĐẶT CÁC THUẬT TOÁN 6](#_Toc57929369)

[A. Giao diện và chức năng của chương trình 6](#_Toc57929370)

[B. Thuật toán xác định tần số cơ bản F0 7](#_Toc57929371)

[C. Hàm tự tương quan 9](#_Toc57929372)

[D. Hàm tìm các đỉnh cực đại cục bộ 10](#_Toc57929373)

[E. Hàm lọc trung vị 11](#_Toc57929374)

[IV. KẾT QUẢ THỰC NGHIỆM 12](#_Toc57929375)

[A. Thực nghiệm với file lab\_female.wav [7] 12](#_Toc57929376)

[B. Bảng số liệu 13](#_Toc57929377)

[V. KẾT LUẬN 14](#_Toc57929378)

[A. Kết quả đạt được 14](#_Toc57929379)

[B. Hướng phát triển 14](#_Toc57929380)

[VI. TÀI LIỆU THAM KHẢO 14](#_Toc57929381)

# ĐẶT VẤN ĐỀ

Trong thời đại ngày nay, cùng với sự phát triển mạnh mẽ của khoa học kỹ thuật thì vấn đề trao đổi thông tin đa phương tiện ngày càng trở nên cần thiết, từ lúc đầu chỉ giao tiếp, tương tác người máy trở nên cấp thiết hơn. Vì vậy mà quá trình xử lý tiếng nói, đặc biệt là các thuật toán xử lý nâng cao chất lượng tiếng nói được ra đời. Tuy không thể bảo toàn được y nguyên tín hiệu ban đầu nhưng sử sụng các thuật toán này ta có thể tăng cường được chất lượng tiếng nói và giảm bớt nhiễu nền để tín hiệu sau khi xử lý đến người nghe vẫn mang đầy đủ nội dung thông tin và không gây khó chịu bởi nhiễu đối với người nghe.

Bài báo gồm các phần như sau:

* Phần I: Nêu ra vấn đề
* Phần II: Trình bày cơ sở lý thuyết xử lý tín hiệu tiếng nói và các thuật toán tính F0.
* Phần III: Trình bày mã python và phương pháp cài đặt thuật toán.
* Phần IV: Trình bày kết quả thực nghiệm thu được.
* Phần V: Đúc kết những gì đã làm được và hướng phát triển

# CƠ SỞ LÝ THUYẾT

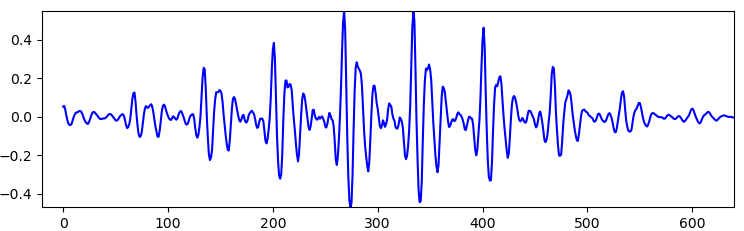
## Tần số cơ bản F0 là gì?

Tần số cơ bản là tốc độ rung của dây thanh trong quá trình phát âm, gọi là F0. Người nói có thể điều khiển mức độ căng của hai dây thanh để khoảng giữa hai dây thanh đó đóng lại hoàn toàn, tạo thành khe hẹp hay mở rộng ra. Khoảng không ở giữa này được gọi là thanh môn. Khi thanh môn hẹp, không khí đi qua nó sẽ tạo ra một âm thanh điều hòa. Thuật ngữ “cao độ” (pitch) dùng để chỉ tần số cơ bản mà người nghe có thể cảm nhận được. Bằng cách thay đổi độ căng của dây thanh, người nói có thể điều chỉnh tần số cơ bản. Thông thường, F0 của giọng nam nằm trong khoảng từ 70 Hz đến 250 Hz, trong khi đó giọng nữ có F0 từ 150 Hz đến 400 Hz [1].

## Các loại tín hiệu tiếng nói

### Âm hữu thanh

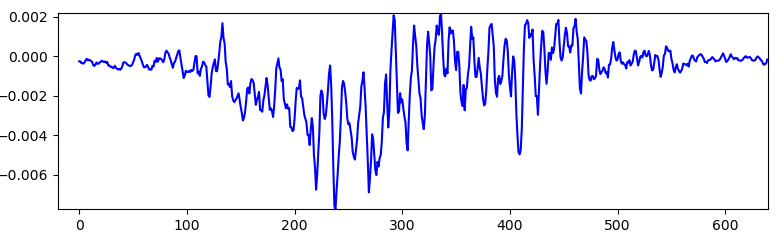
* Âm hữu thanh (voiced speech) là âm phát ra có thanh, ví dụ như các nguyên âm /a/, /e/, /i/, /o/, /u/ hoặc các phụ âm như /m/, /n/, /l/. Thực ra âm hữu thanh được tạo ra là do việc không khí qua thanh môn (thanh môn tạo ra sự khép mở của dây thanh dưới sự điều khiển của hai sụn chóp) với một độ căng của dây thanh sao cho chúng tạo nên dao động [1].
* Vì âm hữu thanh là tín hiệu tuần hoàn nên có thể xác định được tần số cơ bản F0.



1. Một khung tín hiệu tuần hoàn của file lab\_female.wav [7]

### Âm Vô thanh

* Âm vô thanh (voiced speech) là âm khi tạo ra tiếng thì dây thanh không rung hoặc rung đôi chút tạo ra giọng như giọng thở, ví dụ như /t/, /p/ hay /k/ [1].
* Vì âm vô thanh là tín hiệu không tuần hoàn nên không thể xác định được tần số cơ bản F0.



1. Một khung tín hiệu không tuần hoàn của file lab\_female.wav [7]

## Tự tương quan

Trong xử lý tín hiệu số nói chung và xử lý tín hiệu tiếng nói nói riêng, hàm tự tương quan dùng để biến đổi tín hiệu tuần hoàn thành một tín hiệu tuần hoàn khác có các điểm cực đại có thể xác định được dễ dàng, nhờ đó ứng dụng để xác định chu kỳ cơ bản T0 và tần số cơ bản F0 [1].

Hàm tự tương quan được xác định bởi công thức [1]:

**rxx(l) =**

Trong đó:

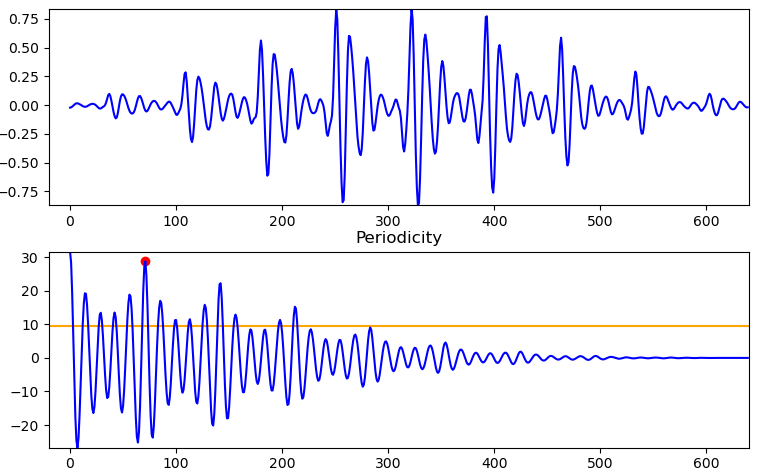
* *rxx(l)* là giá trị hàm tự tương quan theo độ trễ *l*.
* (2*N+1)* là độ dài khung tín hiệu.
* *x(n)* là biên độ tín hiệu tại thời điểm *n*.

Hàm tự tương quan có các tính chất sau:

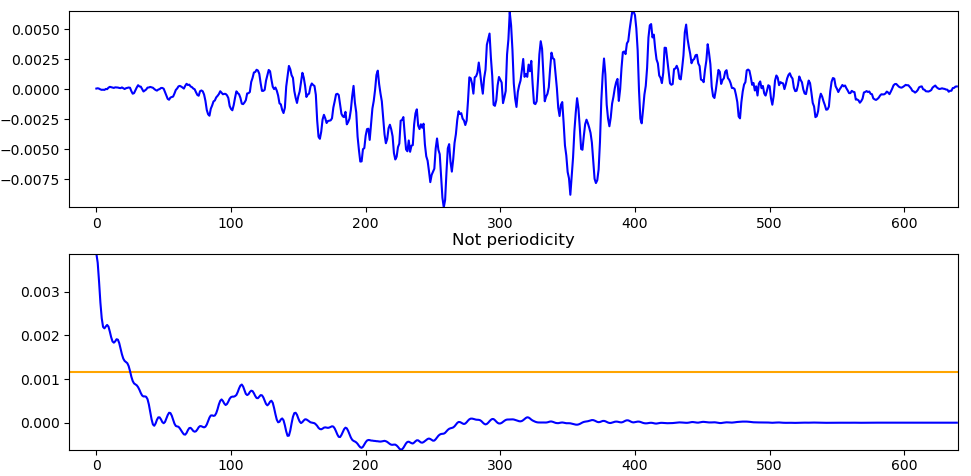
* Là một hàm chẵn: rxx(l)=rxx(-l);
* Đạt giá trị cực đại tại *l=0: |rxx(l)| ≤ rxx(0)* với mọi *l*;
* Đại lượng *rxx(0)* bằng năng lượng của tín hiệu tiếng nói.

Khi xử lý tín hiệu dùng kỹ thuật xử lý ngắn hạn, ta chia tín hiệu tiếng nói thành các khung tín hiệu có độ dài hữu hạn và công thức tự tương quan trở thành:

**Rt(τ) =**



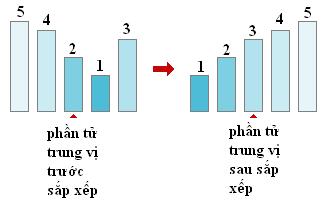
1. Hình trên là một khung tín hiệu tuần hoàn, hình dưới là hàm tương quan của file lab\_female.wav [7]



1. Hình trên là một khung tín hiệu không tuần hoàn, hình dưới là hàm tương quan của file lab\_female.wav [7]

## Lọc trung vị

* Làm trơn trung vị (median smoothing) là kỹ thuật lọc phi tuyến được sử dụng phổ biến trong xử lý tín hiệu. Nó có ưu điểm là loại bỏ được giá trị nhảy vọt so với các giá trị lân cận mà vẫn bảo toàn các điểm gián đoạn trong tín hiệu. Giá trị đầu ra của bộ lọc trung vị ứng với giá trị đầu vào *x(n) [1].*
* *Kích thước của bộ lọc luôn là một số lẻ thường là (3, 5, 7, 9), để dễ dàng xác định được điểm ở giữa.*
* *Nếu các phần tử bên trái hoặc bên phải không đủ phần tử để đưa vào khung thì sẽ bù vào đó các phần tử 0.*



1. Minh họa về lọc trung vị

## Chia cửa sổ và Hamming

* Việc chia tín hiệu tiếng nói thành các khung tín hiệu giúp ta xác định và xử lý được các tín hiệu tiếng nói có đặc tính hầu như không thay đổi, độc lập [1].



1. Minh họa về cách chia khung tín hiệu [1]

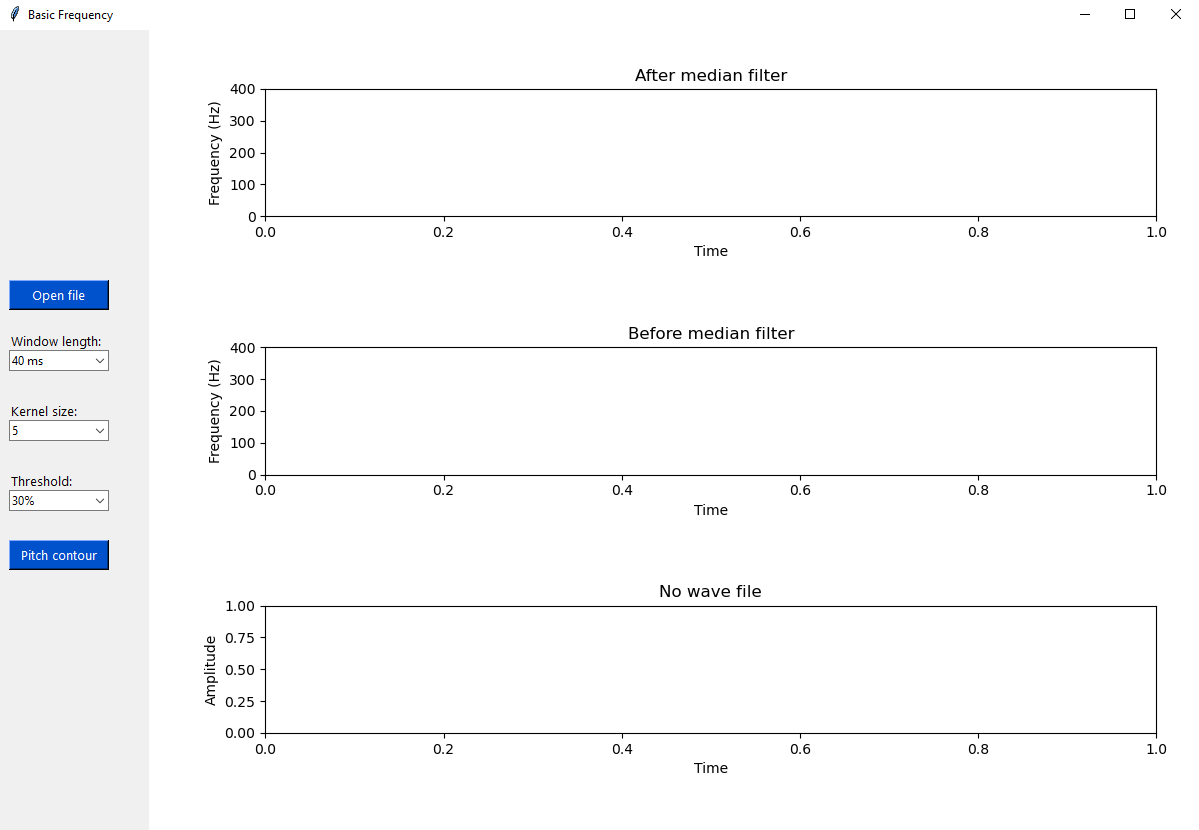
* Có nhiều hàm cửa sổ phổ biến như: Hamming, Blackman, tam giác, chữ nhật. Các hàm cửa sổ này đều có công dụng là làm trơn các khung tín hiệu, qua đó dể xác định được tần số cơ bản F0.
* Hàm cửa sổ Hamming được xác định theo công thức:

**Hamming(n) = 0.54 – 0.46cos(2π)**

# MÃ CHƯƠNG TRÌNH CÀI ĐẶT CÁC THUẬT TOÁN

## Giao diện và chức năng của chương trình

* Toàn bộ source code có thể tham khảo tại link [7].



1. Giao diện chương trình

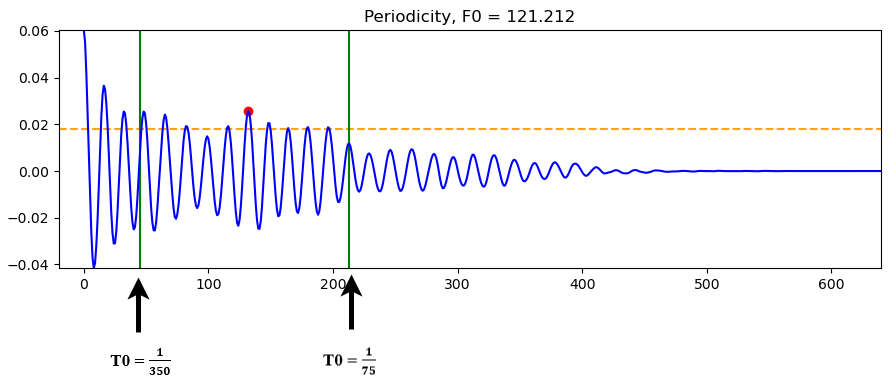
* Các subplots trong giao diện:
  + Subplot thứ 3 để hiện thị plot(đồ thị) của file âm thanh.
  + Subplot thứ 2 để hiện thị scatter(dấu chấm) của các tần số cơ bản F0 khi chưa lọc trung vị.
  + Subplot thứ 1 để hiện thị scatter(dấu chấm) của các tần số cơ bản F0 sau khi lọc trung vị.
* Các chức năng:
* Cho phép chọn các tham số đầu vào khác nhau như: độ dài khung cửa sổ(window length), kích thước bộ lọc(kernel size), và ngưỡng(threshold).
* Button “Open file” để chọn các file âm thanh và plot nó.
* Button “Pitch contour” để hiện thị tần số cơ bản của file âm thanh.
* Ngoài ra sau khi đã hiện thị plot của file âm thanh, có thể di chuột vào từng đoạn của file âm thanh để xem khung cửa sổ và hàm tự tương quan của tín hiệu tại con trỏ chuột.

## Thuật toán xác định tần số cơ bản F0

* Để xác định ngưỡng của khung tín hiệu, ta tìm cực đại toàn cục của khung tín hiệu tại vị trí có dộ trễ bằng 0. Sau đó chỉ lấy khoảng 30% giá trị biên độ của điểm cực đại toàn cục đó:

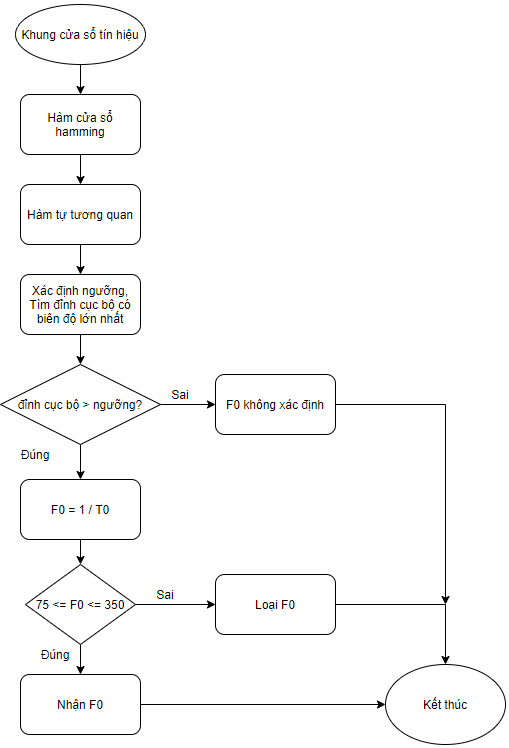
**Ngưỡng = 30% (Biên độ cực đại toàn cục)**

* Để tránh việc tín hiệu âm thanh tuần hoàn có tần số cơ bản F0, nhưng giá trị lại nằm ngoài vùng giá trị từ [75, 350] ta giới hạn việc tìm các đỉnh cực đại cục bộ của nó trong đoạn thời gian [, ] giây

****

1. Giới hạn khoảng thời gian trong việc xác định tần số cơ bản F0

* Cho lần lượt các khung tín hiệu đã chia đi qua thuật toán sau:



1. Sơ đồ khối xác định tần số cơ bản F0

* Code:

def pitch\_contour(*data*, *win\_len*, *window*, *ham*, *ratio*, *min\_frame*, *max\_frame*):

    F0s = []  *# for save valid F0*

    indexes = []  *# for save index of window have valid F0*

    index = 0  *# for counter*

*while* index + window <= len(data):

        w = data[index:index + window]  *# get current window*

        w = w \* ham  *# smoothing window*

        index += window // 2  *# next window*

        a\_corr = fftautocorr(w)  *# calc auto correlation*

        threshold = a\_corr[0] \* ratio  *# calc threshold = n% maximum global*

*# find top peaks, bottom peaks and check it*

        max\_indexes = find\_peaks(a\_corr, min\_frame, max\_frame)  *# find top peaks*

        min\_indexes = find\_peaks(-1 \* a\_corr, 0, len(a\_corr) - 1)  *# find bottom peaks*

*if* len(max\_indexes) == 0 or len(min\_indexes) == 0:

*continue*

*# calc max\_local, min\_local and check it*

        max\_index = get\_index\_of\_max\_local(a\_corr, max\_indexes)  *# get index of max\_indexes*

        max\_local = a\_corr[max\_indexes[max\_index]]

        min\_local = a\_corr[min\_indexes[max\_index]]

*if* max\_local < threshold or max\_local - min\_local < 0.02:

*continue*

*# calc basic frequency and check it*

        T0 = max\_indexes[max\_index] \* win\_len / window

        F0 = 1000 / T0

*# append F0 and index*

        F0s.append(F0)

        indexes.append(index - window)

*# return tuple*

*return* F0s, indexes

## Hàm tự tương quan

* Code:

def autocorr(*x*):  *# O(n^2)*

    n = len(x)  *# Lấy độ dài của tín hiệu x*

    auto\_corr = [0] \* n  *# Khởi tạo mảng để lưu kết quả và trả về*

    delay = range(0, n)  *# Sinh mảng delay*

*for* i in delay:

*for* j in range(n - delay[i]):

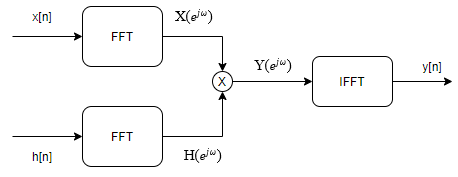
            auto\_corr[i] += x[j] \* x[j + delay[i]]  *# Tự tương quan tại độ trễ delay[i]*

*return* round(auto\_corr, 8)  *# Trả về kết quả và làm tròn 8 chữ số thập phân*

* **Đánh giá độ phức tạp thuật toán:** 
  + Theo đoạn code trên ta thấy việc sử dụng hai vòng lặp lồng nhau làm tăng độ phức tạp tính toán của hàm này lên rất nhiều. **Big O(n2)**
  + Kết quả chạy thực nghiệm cho thấy phải mất tầm 1 – 2 phút chương trình mới trả về kết quả. Vậy kết luận không thể thực hiện tự tương quan theo thuật toán này.
* **Giải pháp:**
* Bản chất của công thức tính tự quan được đề cập ở trên và trong code bản chất cũng là tích chập của hai vector đối nhau.
* Do đó có thể viết lại công thức như sau:

**rxx[n] = x[n] \* x[-n]**

* Áp dụng hệ quả số 2, tính chất “Tính tổng chập trên miền thời gian” của biến đổi Fourier:

****

1. Sơ đồ khối của hàm tự tương quan theo biến đổi fft và ifft

* Code:

def fftautocorr(*x*):  *# O(n \* log(n))*

*# Độ dài của y[n] = x[n] \* h[n] theo công thức N \* M - 1*

*# Trong đó N, M lần lượt là độ dài của x và h*

    n = 2 \* len(x) - 1

    a = fft(x, n)  *# Fast fourier tranform x[n]*

    b = fft(x[::-1], n)  *# Fast fourier tranform x[-n]*

    c = ifft(a \* b)  *# Inverse fast fourier tranform*

*return* real(c[n // 2:])  *# Trả về nữa cuối của mảng, chỉ lấy phần thực, bỏ đi phần ảo*

* **Đánh giá độ phức tạp thuật toán:**
* Kết quả chạy thực nghiệm cho thấy chương trình cho ra kết quả nhanh hơn rất nhiều so với hàm autocorr(x). Bởi vì độ phức tạp tính toán, ở đây là phép nhân đã giảm đi rất nhiều
* **Big O(nlog(n))**

## Hàm tìm các đỉnh cực đại cục bộ

* Code:

def find\_peaks(*arr*, *min\_frame*, *max\_frame*):

    index\_peaks = []  *# Lưu các index của các đỉnh*

    index\_tmp = 0  *# Biến tạm để kiểm tra trường hợp đỉnh nằm ngang(có các giá trị liên tiếp bằng nhau) [1, 2, 5, 5, 5, 1]*

    is\_tmp = False  *# Biến để kiểm tra trường hợp này [2, 2, 2, 1]*

*# Ý tưởng: duyệt mảng từ min\_frame đến max\_frame*

*# Tại phần tử đang xét so sánh với hai phần tử bên cạnh*

*for* i in range(min\_frame, max\_frame):

*if* arr[i] > arr[i - 1] and arr[i] > arr[i + 1]:

            index\_peaks.append(i)

*elif* arr[i] > arr[i - 1] and arr[i] == arr[i + 1]:

            index\_tmp = i

            is\_tmp = True

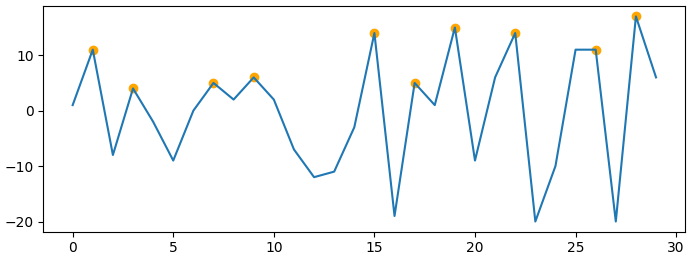
*elif* arr[i] == arr[i - 1] and arr[i] > arr[i + 1] and is\_tmp is True:

            index\_peaks.append(i)

            is\_tmp = False

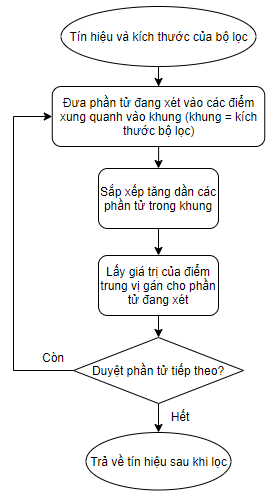
*return* index\_peaks  *# Trả về index của các đỉnh trong mảng đầu vào*

* Kết quả chạy thử:



1. Thuật toán tìm đỉnh

## Hàm lọc trung vị



1. Sơ đồ khối của hàm lọc trung vị

* Code:

def median\_filter(*arr*, *kernel\_size*):

*if* type(arr) is not list:

        arr = arr.tolist()  *# Kiểm tra có phải kiểu dữ liệu "list" không*

    length = len(arr)  *# Lấy độ dài của mảng đầu vào*

    part = (kernel\_size - 1) // 2  *# Tính số phần tử ở mỗi bên (trái, phải)*

    med\_arr = []  *# Khởi tạo mảng mới để trả về kết quả*

*for* i in range(length):

        left = i - part

        right = i + part

*if* left < 0:  *# Trường hợp bên trái không đủ phần tử*

            tmp = [0] \* (0 - left) + arr[0:right + 1]  *# Thêm phần tử 0 vào bên trái*

*elif* right >= length:  *# Trường hợp bên phải không đủ phần tử*

            tmp = arr[left:length] + [0] \* (right - length + 1)  *# Thêm phần tử 0 vào bên phải*

*else*:  *# Trường hợp hai bên đều đủ*

            tmp = arr[left: right + 1]

        tmp.sort()  *# Sắp xếp tăng dần*

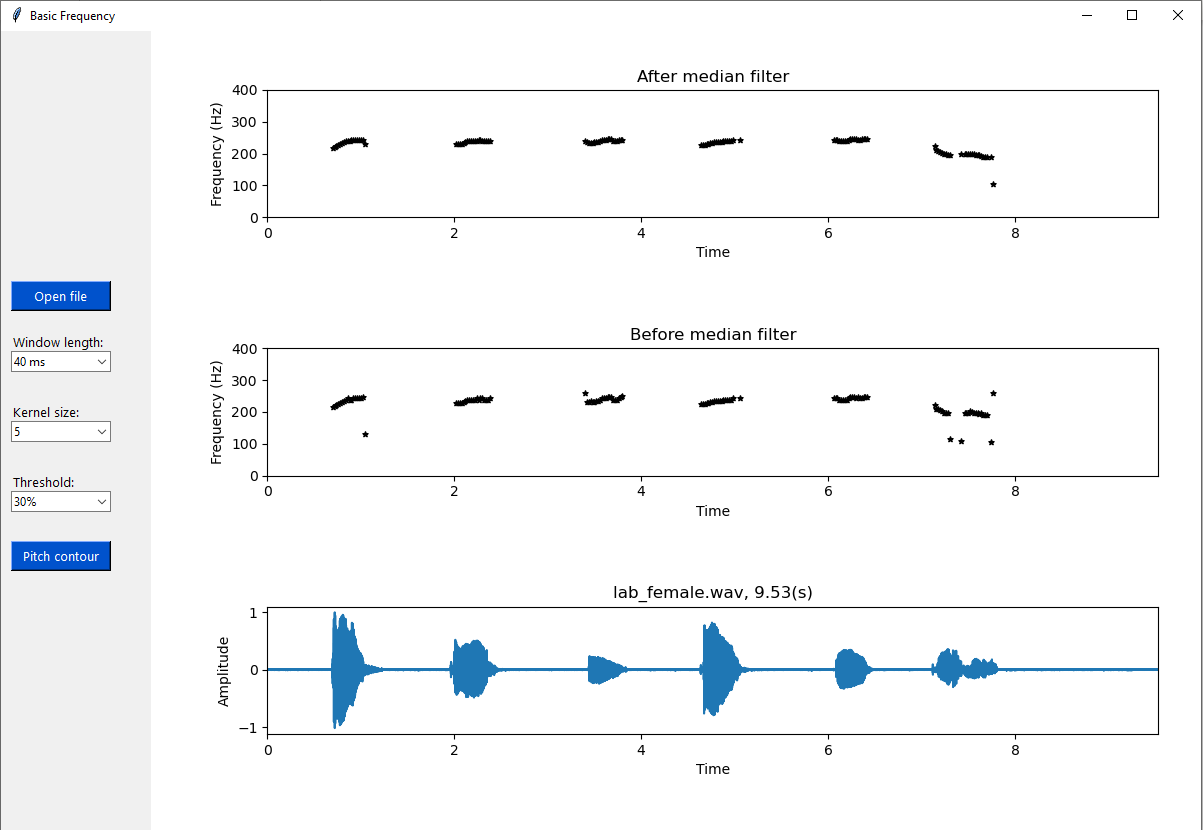
        med\_arr.append(tmp[part])  *# Thêm vào mảng*

*return* array(med\_arr)  *# Trả về kết quả sau khi lọc*

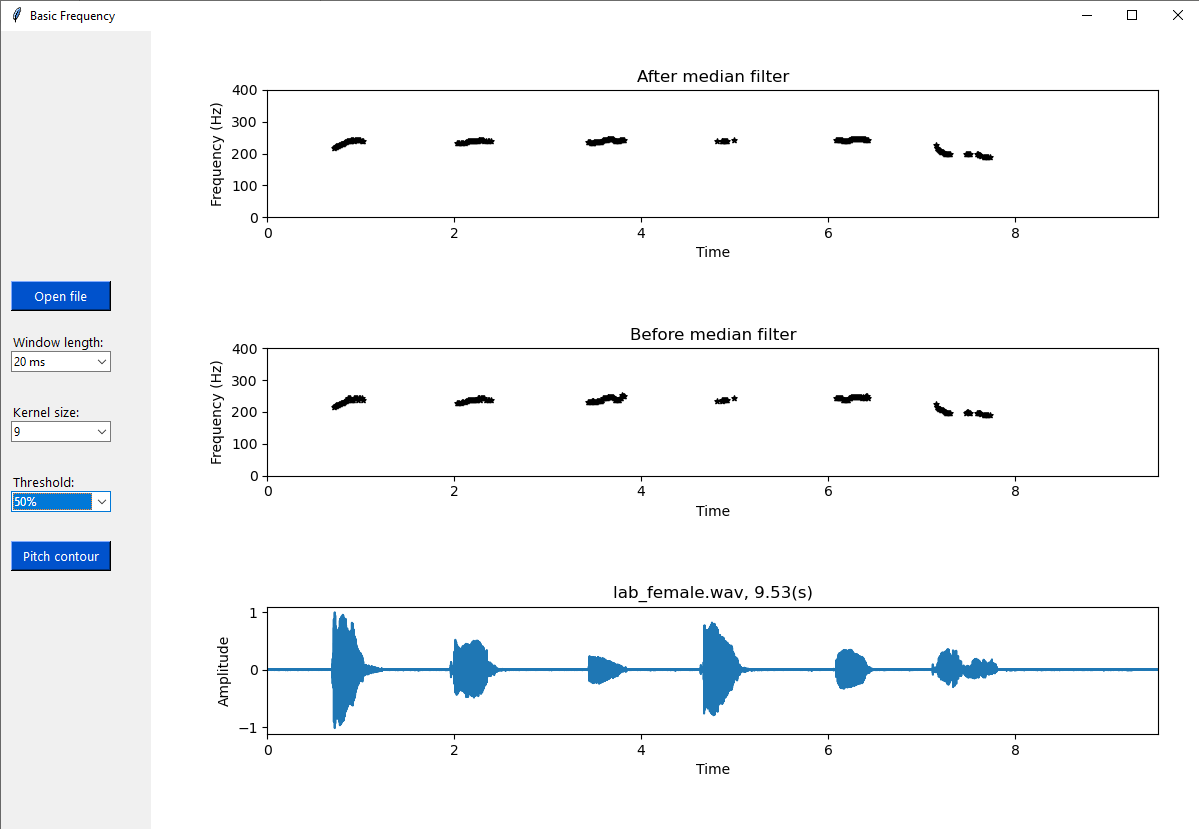
# KẾT QUẢ THỰC NGHIỆM

## Thực nghiệm với file lab\_female.wav [7]

* Chạy chương trình với các tham số khác nhau

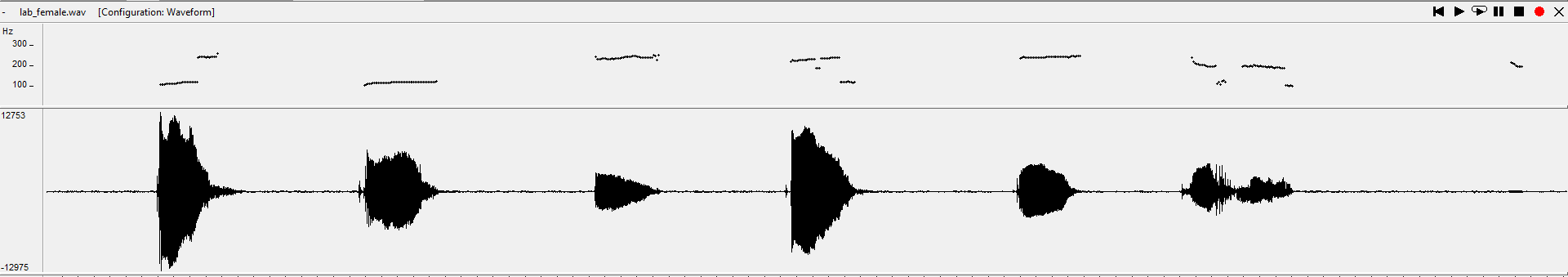


1. Khung = 40 ms, Bộ lọc = 5, Ngưỡng = 30%



1. Khung = 20 ms, Bộ lọc = 9, Ngưỡng = 50%

* So sánh kết quả với phần mềm wave suffer

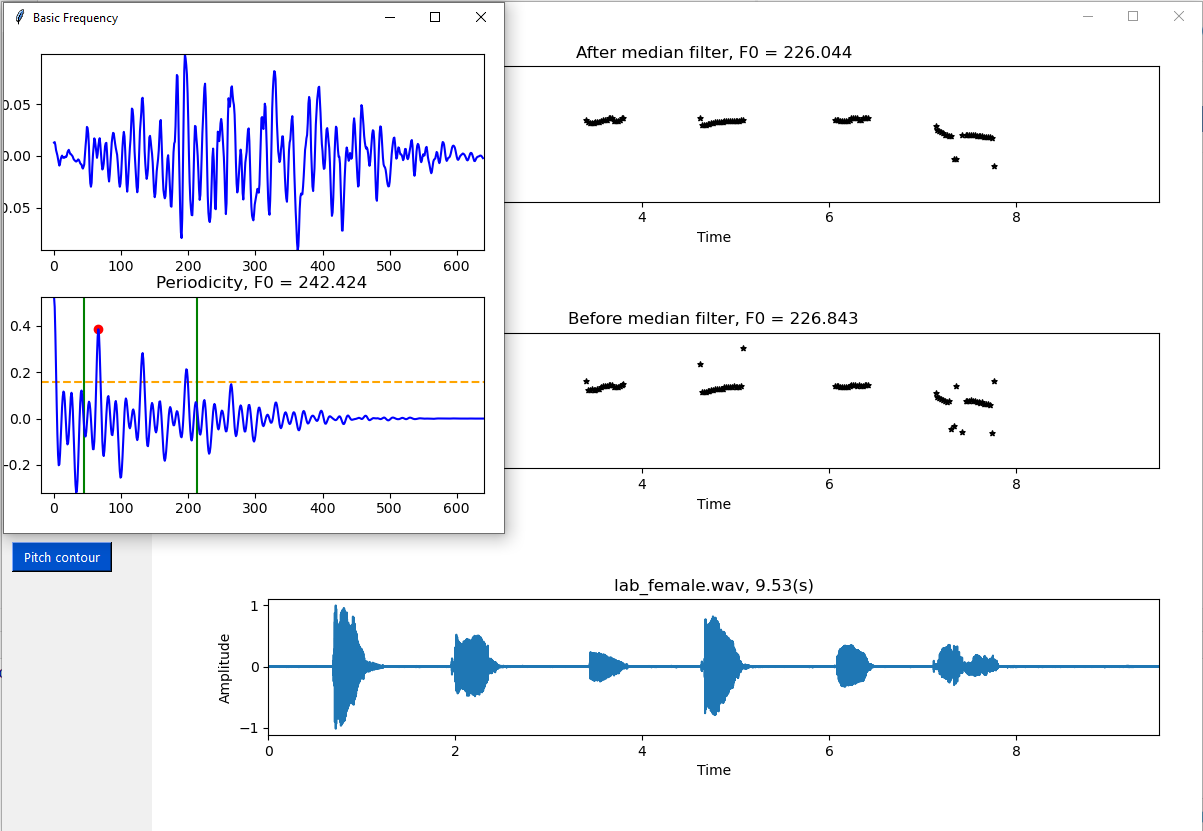


1. Kết quả của phần mềm wave suffer

* Di chuột vào file âm thanh để xem khung cửa sổ

Subplot thứ 2 của khung tín hiệu có trả về kết quả:

* + Periodicity: Tín hiệu tuần hoàn.
  + Not Periodicity: Tín hiệu không tuần hoàn.



1. Xem khung tín hiệu tại con trỏ

## Bảng số liệu

* Công thức có việc tính toán kết quả và sai số:
* F0 thủ công được tính toán bằng phần mềm wave suffer

F0 tự động được tính toán bằng code của báo cáo này

1. Bảng số liệu sai số

|  |  |  |  |
| --- | --- | --- | --- |
| File | F0 thủ công | F0 tự động |  |
| lab\_female.wav | 205.675 | 228.667 | -22.992 |
| lab\_male.wav | 117.192 | 114.913 | 2.279 |
| studio\_female.wav | 232.784 | 245.801 | -13.017 |
| studio\_male.wav | 121.415 | 121.103 | 0.312 |
| 1.wav | 116.912 | 117.968 | -1.056 |
| 2.wav | 190.242 | 191.153 | -0.911 |
| LA001.wav | 228.874 | 232.615 | -3.741 |
| LA025.wav | 121.498 | 121.176 | 0.322 |

# KẾT LUẬN

## Kết quả đạt được

* Xây dựng được chương trình khá hoàn thiện, đẹp mắt.
* Học được cách làm việc nhóm, viết báo cáo.
* Hiểu được ý nghĩa của việc xác định tần số cơ bản F0.

## Hướng phát triển

* Trong phạm vi báo cáo này chỉ dừng lại ở việc khảo sát trên miền thời gian. Trong tương lai sẽ tiếp tục nghiên cứu và khảo sát trên cả miền tần số.
* Cần phải tiếp tục tối ưu hàm tự tương quan.

# TÀI LIỆU THAM KHẢO

1. Trần Văn Tâm,“Luận văn thạc sĩ ký thuật: Xác định tần số cơ bản của tín hiệu tiếng nói dùng hàm tự tương quan”, Đà Nẵng – Năm 2019.
2. Nguyễn Bình Thiên, Ninh Khánh Duy, “Cải tiến thuật toán tự tương quan tìm cao độ của tín hiệu đàn ghi-ta trên vi xử lý ARM CORTEX-M4”, Khoa công nghệ thông tin, Trường đại học Bách Khoa, Đại học Đà Nẵng.
3. Nguyễn Xuân Lưu, “Slides tích chập và tương quan chéo”,

Link: <https://sites.google.com/site/nguyenxuanluubkhn/4th/tin-hieu-he-thong/on-tap-cuoi-ki/on-tap-cuoi-ki-tich-chap>

1. Biến đổi fourier nhanh, Link: <https://vi.wikipedia.org/wiki/Bi%E1%BA%BFn_%C4%91%E1%BB%95i_Fourier_nhanh>
2. Pitch contour example, Link: <https://languagelog.ldc.upenn.edu/nll/?p=34251>
3. Lọc trung vị, Link: <https://globlib4u.wordpress.com/2013/02/05/bo-loc-trung-vi-median-filter/>
4. Source code và các file âm thanh trong bác cáo, Link: <https://github.com/chautruonglong/Fundamental-Frequency>