LỜI CẢM ƠN

Sau hơn 4 tuần học tập, nghiên cứu và làm việc cùng nhau, nhóm 1 chúng em đã học được rất nhiều điều bổ ích, cần thiết cho một sinh viên chuyên ngành Vật lý Tin học. Khoảng thời gian tuy không quá dài, nhưng đủ để chúng em hiểu như thế nào là làm việc nhóm, tự học, tự tìm hiểu và học hỏi thêm được nhiều kỹ năng cần thiết cho một con người năng động để dễ dàng hòa vào lối sống hội nhập, hiện đại. Đó không chỉ là những gì trên giấy mực, mà cả là những gì thiết thực nhất trong cuộc sống như việc làm thế nào sắp xếp thời gian hợp lý, kỹ năng nói, thuyết phục người khác, kỹ năng tự học, tự tìm tòi, … Đây chắc chắn sẽ là hành trang quý báu để chúng em có khả năng học tập tốt hơn, có thêm kinh nghiệm sống, làm việc khi vào đời.

Qua quãng thời gian đó, chúng em rất biết ơn và gửi lời cảm ơn chân thành đến Bộ môn Vật lý Tin học, luôn là nơi sẵn sàng mở cửa, tạo điều kiện thuận lợi cho chúng em được học tập, nghiên cứu dễ dàng hơn. Chúng em xin gửi lời cảm ơn đến Cô Hứa Thị Hoàng Yến, giảng viên môn học "Xử lý tín hiệu số", là người trang bị cho chúng em đầy đủ kiến thức, tận tâm trên những giờ trên lớp và sẵn lòng giúp đỡ khi chúng em gặp khó khăn.

Ngoài ra, chúng em cũng rất biết ơn bạn bè cùng lớp, các anh chị khóa K15 luôn kề vai, sát cánh để giúp đỡ khi chúng em gặp khó khăn, luôn đưa ra những lời khuyên quý báu, những chia sẻ, động viên chân thành để chúng em hoàn thành được mục tiêu của đề tài này.

Chúng em xin chân thành cảm ơn!

TP. Hồ Chí Minh, tháng 06 năm 2019

Nhóm 1

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MỤC LỤC

Trang

DANH MỤC CÁC HÌNH ẢNH

DANH MỤC CÁC BẢNG

DANH MỤC CÁC SƠ ĐỒ

LỜI MỞ ĐẦU

Ngày nay, với sự phát triển không ngừng của khoa học, công nghệ và kỹ thuật; con người đã và đang từng bước hòa nhập, phát triển trên đà Công nghiệp hóa – Hiện đại hóa. Đi đôi với sự phát triển về kinh tế, chính trị là sự phát triển không ngừng về khoa học – kỹ thuật. Xã hội 4.0 ngày nay, chúng ta không còn quá xa lạ với việc bắt gặp con người điều khiển các đồ vật, thiết bị trong gia đình, ở nhà máy hay bất cứ đâu qua thiết bị nghe – nhìn từ xa. Có thể điều khiển bằng tay, cũng có thể bằng chính giọng nói của bản thân. Nắm bắt được chính xu thế đó, đồng thời, cũng là cơ hội tìm hiểu, nghiên cứu xem cơ chế của việc các thiết bị xử lý câu lệnh lời nói như thế nào đã khiến nhóm chúng em quyết định chọn đề tài tìm hiểu việc xử lý câu lệnh âm thanh bất kỳ.

Với mục tiêu được tự tìm hiểu, trau dồi khả năng làm việc nhóm, tự học, sáng tạo đồng thời được sử dụng chính những kiến thức cụ thể của môn "Xử lý tín hiệu số" và cách dùng phần mềm Matlab để tạo ra một ứng dụng cụ thể, mang ý nghĩa thực tiến cho việc tìm tòi, nghiên cứu, học tập. Qua thời gian làm việc và học tập cùng nhau, nhóm chúng em đã hoàn thành trong việc tìm hiểu và xây dựng thành công chương trình *"**Ứng dụng xử lý câu lệnh dạng âm thanh bằng Matlab**"*, đảm bảo được các yêu cầu và mục tiêu đề tài môn học mà nhóm hướng đến.

NHÓM SINH VIÊN THỰC HIỆN (Nhóm 1)

Bùi Ngô Tôn Bách	1613013
Tống Hải Đăng	1613026
Nguyễn Hồng Sỹ Nguyên	1613124
Phan Thanh Tùng	1613240

CHƯƠNG I
GIỚI THIỆU TỔNG QUAN VỀ ĐỒ ÁN

1.1. Tổng quan về ý tưởng đề tài:

Hình 1 Trợ lý ảo Siri trên thiết bị di động iOS – một ví dụ về việc xử lý và phản hồi giọng nói

Lấy ý tưởng chính từ việc phát triển của xã hội hiện đại, con người giao tiếp với máy móc bằng chính ngôn ngữ (giọng nói). Đề tài giúp chúng em tiếp cận việc nghiên cứu tại sao một câu lệnh khi được nói ra, thiết bị vô tri vô giác có thể hiểu và thi hành được mệnh lệnh, cho dù lời nói có được trong suốt, hay trong điều kiện nhiễu (có nhiều tạp âm). Bằng việc áp dụng các kiến thức môn học "Xử lý tín hiệu số" cùng với những công cụ hỗ trợ đắc lực từ phần mềm Matlab, nhóm đã tạo ra một ứng dụng được viết bằng ngôn ngữ Matlab có các chức năng hỗ trợ trong việc xử lý câu lệnh (mệnh lệnh) dưới dạng âm thanh (lời nói) một cách tương quan nhất. Đồng thời, sau khi xử lý tín hiệu nguồn được mã hóa (lượng tử hóa) thành file nhị phân 8 bit, phù hợp cho việc lưu trữ, phát triển và xử lý bộ dữ liệu câu lệnh gốc sau này.

1.2. Nhiệm vụ – mục tiêu đề tài:

- Tìm hiểu cách xử lý tín hiệu âm thanh thu âm từ máy tính – laptop cá nhân, hoặc từ một file có sẵn qua phần mềm Matlab.
- Áp dụng được kiến thức "Xử lý tín hiệu số" về thuật toán FFT và các bộ lọc để xử lý file âm thanh nguồn.
- Biểu thị được sự tương quan của quá trình xử lý qua hệ thống các sơ đồ.
- Tìm hiểu cách xử lý nén – lượng tử hóa file đã được xử lý nhằm mục đích lưu trữ và phát triển sau này.
- Xây dựng được ứng dụng từ Matlab với giao diện người dùng cụ thể, kết hợp các chức năng vào ứng dụng.

1.3. Các thành phần – chức năng cơ bản của ứng dụng:

Ứng dụng, sản phẩm sau khi hoàn thành có một số chức năng cơ bản như sau:

- Thu âm trực tiếp – Tải lên trực tiếp file nguồn âm thanh cần xử lý.
- Vùng vẽ biểu đồ theo miền âm thanh và tần số trước và sau khi âm thanh được phân tích.
- Biến đổi FFT tín hiệu nguồn và xử lý tạp âm, nhiễu.
- Xuất ra được file âm thanh sau khi xử lý.
- Xuất ra được file nhị phân (mã hóa – lượng tử hóa) âm thanh cho mục đích lưu trữ, hình thành phát triển các chức năng sau này.

1.4. Phân chia nhiệm vụ trong nhóm:

- Điều hành chung: ***Bùi Ngô Tôn Bách***, ***Phan Thanh Tùng***.
- Chịu trách nhiệm chính về lý thuyết, tài liệu: ***Phan Thanh Tùng***, ***Tống Hải Đăng***.
- Chịu trách nhiệm chính về lập trình ứng dụng: ***Nguyễn Hồng Sỹ Nguyên***, ***Bùi Ngô Tôn Bách***.
- Văn bản, báo cáo: ***Phan Thanh Tùng***, ***Nguyễn Hồng Sỹ Nguyên***, ***Bùi Ngô Tôn Bách***, ***Tống Hải Đăng***.

CHƯƠNG II

LÝ THUYẾT

2.1. Giới thiệu chung về biến đổi Fourier nhanh (Fast Fourier Transform – FFT):

FFT được coi là một thuật toán có nhiều ứng dụng quan trọng trong nhiều lĩnh vực. Thuật toán này được sử dụng rộng rãi trong các lĩnh vực khoa học và kỹ thuật. Được đặt tên theo nhà toán – vật lý học người Pháp Jean - Baptiste Joseph Fourier cuối thế kỷ 18, biến đổi Fourier là một phép toán biến đổi tín hiệu từ miền thời gian (không gian) sang miền tần số. Vào thời điểm này, các máy tính đầu tiên đã mất hàng trăm giờ để thực hiện một phép biến đổi đơn giản theo FFT khi số lượng phép tính lớn.

Hình 2.1 Jean – Baptiste Joseph Fourier (1768 – 1830)

Do đó, kể từ năm 1805 đã có những nỗ lực để nâng cao hiệu quả của thuật toán. Cùng năm đó, Carl Friedrich Gauss đã phát minh ra một phương pháp biến đổi Fourier hiệu quả. Tuy nhiên, mãi cho đến 160 năm sau, năm 1965, James Cooley của IBM và John Tukey của Princeton đã khám phá ra thuật toán này và phổ biến nó. Việc làm của họ làm giảm đáng kể số lượng phép tính. Thuật toán được gọi là FFT ngay sau khi được giới thiệu. Nó lập tức tạo nên sự cách mạng trong biến đổi Fourier của các tín hiệu. Với 64000 điểm FFT, thuật toán này nhanh gấp 4000 lần so với phương pháp ban đầu. Nó đã có một số sửa đổi kể từ khi phát hiện ra. Và vào tháng 1 năm 2000, nó được đưa vào Top 10 Thuật toán của thế kỷ 20.

Với sự ra đời của việc xử lý tín hiệu số, FFT được nâng lên một tầm quan trọng mới. Một số công ty như Texas Instruments và Analog Devices giới thiệu bộ xử lý FFT riêng biệt. Một khi tín hiệu thoát ra khỏi miền tương tự thông qua một bộ chuyển đổi tương tự sang số, các khả năng là vô tận.

Nhiều ứng dụng được hưởng lợi từ FFT, đặc biệt là lĩnh vực truyền thông. Radar, điện thoại tế bào, xử lý tiếng nói, và SETI (tìm kiếm trí tuệ ngoài trái đất) là một vài ứng dụng sử dụng rộng rãi FFT. Các ứng dụng của FFT bao quanh mọi lĩnh vực của cuộc sống. Đó có thể là các thiết bị giám sát, một bộ tổng hợp hoặc một bộ phân tích tín hiệu để phép biến đổi Fourier có thể được ứng dụng. Ngày nay thì hầu hết các tín hiệu đều không thể được ứng dụng nếu không có phép biến đổi Fourier.

Hình 2.2 Đài quan sát Arecibo ở Puerto Rico để phân tích tín hiệu tìm kiếm thông tin của người ngoài hành tinh.

2.2. Lý thuyết về FFT:

Theo định lý Fourier, một tín hiệu là sự hợp thành của một số hàm điều hòa với biên độ, tần số và pha cho trước. Dạng chuỗi Fourier cho tín hiệu tuần hoàn, x(t):

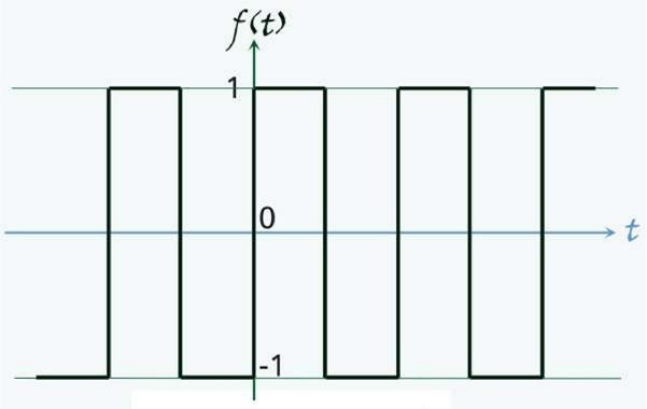
$$X(t) = \frac{A_0}{2} + \sum_{n=1}^{N} (A_n \sin\left(\frac{2\pi nt}{P} + \emptyset_n\right)) = \sum_{n=-N}^{N} (C_n e^{\frac{2\pi jnt}{P}})$$

trong đó P là chu kỳ tín hiệu và

C_n được gọi là là hệ số Fourier của tín hiệu của X(t) hệ số này được tính như sau:

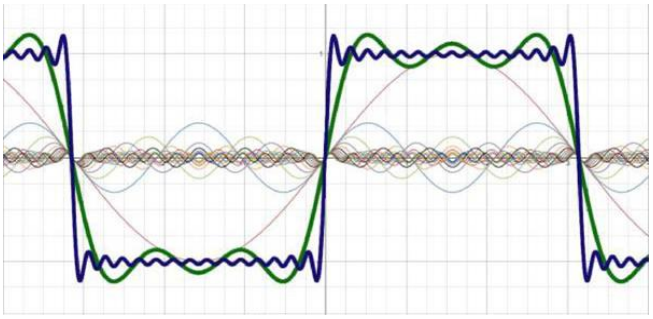
$$C_n = \frac{1}{P}\int_0^P e^{-\frac{2\pi nt}{P}} X(t)dt$$

Với sóng sin, chỉ có một hệ số. Tuy nhiên, nếu tín hiệu chứa một số lượng vô hạn các tần số, chẳng hạn như trong trường hợp sóng vuông lý tưởng, một kết quả hữu hạn N sẽ dẫn đến lỗi. Hình 2.3 và 2.4 là tín hiệu gốc được biểu diễn theo chuỗi Fourier với N = 3 (tín hiệu màu xanh lục) và N = 15 (tín hiệu màu xanh lam). Như chúng ta thấy, độ chính xác của phép biến đổi phụ thuộc vào bản chất của tín hiệu và số số hạng trong chuỗi.



$$f(t)\begin{cases} +1 : 0 \le t < \frac{1}{2} \\ -1 : \frac{1}{2} \le t < 1 \end{cases}$$

Hình 2.3 Hàm sóng vuông lý tưởng



$$f(t) = \frac{4}{\Pi}\sum_{0}^{\infty}\frac{1}{2k+1}\sin(2\Pi(2k+1)t)$$

Hình 2.4 Dạng chuỗi Fourier của sóng vuông với N=3 và N=15

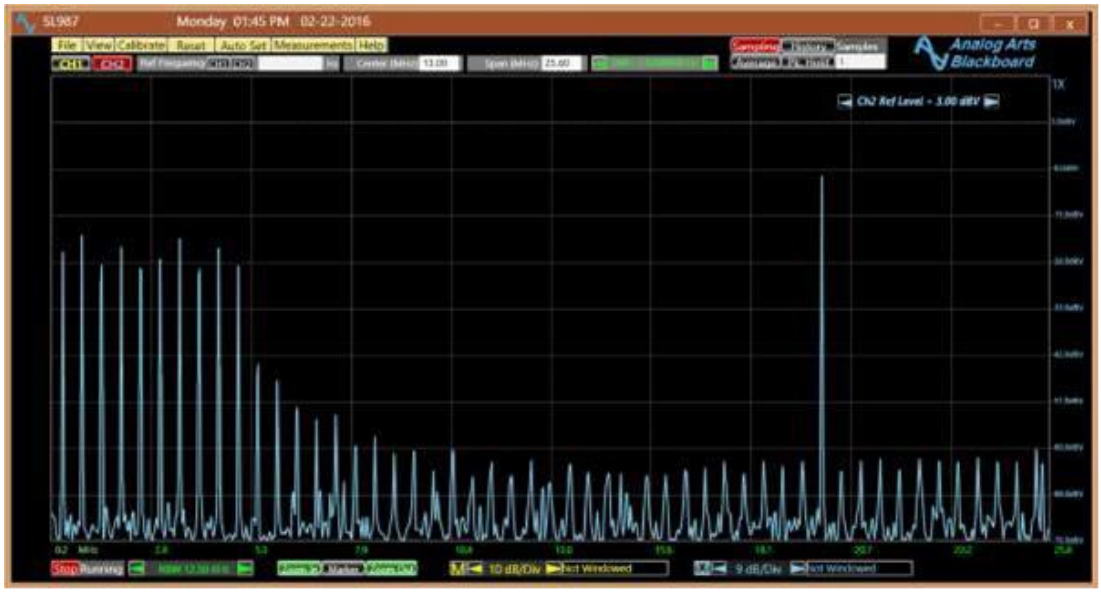
Tuy nhiên, hầu hết các tín hiệu không tuần hoàn. Các tín hiệu không tuần hoàn không có chuỗi Fourier. Thực tế này đã làm Fourier thất vọng, ông phải mất gần 20 năm để phát triển một công thức chung có thể hoạt động cho bất kỳ hàm nào. Bây giờ, mọi tín hiệu đều có thể được chuyển từ miền thời gian sang miền tần số theo phương trình sau đây:

$$X(\Omega) = \sum_{n=-\infty}^{\infty} X[k]e^{-j\Omega nk}$$

Phương trình này không chỉ đơn thuần là toán học. Nó mô tả các khối cấu thành các tín hiệu, và từng khối riêng biệt. Tức là, cho dù các thành phần của tín hiệu dù nhỏ hay lớn, chúng đều sẽ xuất hiện trong danh sách thành phần được cung cấp bởi phép biến đổi. Xét một tín hiệu phức tạp bao gồm một sóng sin và một hàm sinc.

Hình 2.5 Biểu diễn một tín hiệu phức tạp bao gồm một sóng sin và một hàm sinc trên miền thời gian

Biểu diễn tín hiệu trên miền thời gian không có gì hữu ích. Phổ tần số biên độ cho thấy rõ ràng thành phần của nó. Lưu ý rằng những thành phần mong muốn có thể được phân tích một cách riêng biệt. Đó là khả năng của phép biến đổi.



Hình 2.6 Phổ tần của một tín hiệu phức tạp bao gồm một sóng sin và một hàm sin

2.3. Bộ lọc nhiễu tín hiệu – Bộ lọc Butterworth:

2.3.1. Bộ lọc FIR (Finite Impulse Response Filter):

a) Giới thiệu:

Bộ lọc FIR được đặc trưng bởi đáp ứng xung có chiều dài hữu hạn, tức là h(n) chỉ khác không trong một khoảng có chiều dài hữu hạn N (từ 0 đến N-1). Bộ lọc số có đáp ứng xung có chiều dài hữu hạn được đặc trưng bởi hàm truyền đạt sau đây:

$$H(z) = \sum_{n=0}^{N-1} h(n) z^{-n}$$ (Tức là: L[h(n)] = [0, N-1] = N)

Với bộ lọc FIR, H(z) chỉ có các điểm cực tại gốc tọa độ của mặt phẳng z, vậy các điểm cực này luôn nằm trong vòng tròn đơn vị cho nên hệ thống luôn ổn định. Một thuận lợi khác đối với bộ lọc FIR là do chiều dài của h(n) hữu hạn nên nếu h(n) là không nhân quả: $h(n) \neq 0$ (với $n<0$), thì ta có thể đưa nó về nhân quả bằng cách chuyển về gốc tọa độ giá trị đầu tiên khác không của h(n) mà vẫn đảm bảo không thay đổi.

Cái lợi cơ bản nhất của bộ lọc FIR là khi tính toán theo bộ lọc pha tuyến tính. Tức là chúng ta có thể gia công bộ lọc FIR bằng cách coi đáp ứng tần số của nó có pha tuyến tính. Cũng vậy, tín hiệu qua dải thông của bộ lọc sẽ xuất hiện chính xác ở đầu ra với độ trễ đã cho, bởi vì chúng ta đã biết chính xác đáp ứng pha của nó.

Khi áp đặt thêm điều kiện pha tuyến tính vào bộ lọc FIR, dãy đáp ứng xung của bộ lọc chỉ có thể đối xứng hoặc phản đối xứng. Dựa trên tính chất đối xứng hay phản đối xứng của dãy đáp ứng xung và chiều dài N của dãy đáp ứng xung, người ta phân loại bộ lọc FIR làm 4 loại sau đây:

- Bộ lọc loại 1: h(n) đối xứng, N lẻ.
- Bộ lọc loại 2: h(n) đối xứng, N chẵn.
- Bộ lọc loại 3: h(n) phản đối xứng, N lẻ.
- Bộ lọc loại 4: h(n) phản đối xứng, N chẵn.

Các phương pháp thiết kế bộ lọc FIR:

- Phương pháp thiết kế bằng biến đổi Fourier
- Phương pháp cửa sổ
- Phương pháp lấy mẫu tần số

b) Thiết kế bộ lọc FIR sử dụng MATLAB:

Để vẽ đáp ứng tần số khi biết hàm truyền đạt của hệ thống chúng ta sử dụng hàm ***freqz()*** trong Matlab.

Hàm ***freqz()***: trả về đáp ứng tần số của một hệ thống tại một số hữu hạn các điểm rời rạc trên vòng tròn đơn vị khi biết hàm truyền đạt của nó.

Cú pháp của nó như sau:

```
[h,w] = freqz(b,a,n)

[h,f] = freqz(b,a,n,Fs)

[h,w] = freqz(b,a,n,'whole')
```

[h,f] = freqz(b,a,n,'whole',Fs)

h = freqz(b,a,w)

h = freqz(b,a,f,Fs)

freqz(b,a)

Giải thích:

Hàm ***freqz(b,a)*** tìm đáp ứng tần số $H(e^{j\omega T})$ của bộ lọc số từ các hệ số tử số và mẫu số trong vector b và a.

[h,w] = freqz(b,a,n) tìm đáp ứng tần số của bộ lọc số với n điểm.

Từ các hệ số trong vector b và a hàm ***freqz()*** tạo ra vector đáp ứng tần số hồi tiếp và vector w chứa n điểm tần số. Hàm ***freqz()*** xác định đáp ứng tần số tại n điểm nằm đều nhau quanh nửa vòng tròn đơn vị, vì vậy w chứa n điểm giữa 0 và π.

h: Vector đầu ra chứa đáp ứng tần số.

w: Vector đầu ra chứa các giá trị tần số phân phối trong khoảng từ 0 đến radians.

b: Vector đầu vào cho hệ số của tử số.

a: Vector đầu vào cho hệ số của mẫu số.

n: Số các điểm tần số được chuẩn hóa để tính toán đáp ứng tần số.

[h,f] = freqz(b,a,n,Fs) chỉ ra tần số lấy mẫu dương Fs (tính bằng Hz). Nó tạo ra vector f chứa các điểm tần số thực giữa 0 và Fs/2 mà tại đó lệnh sẽ tính đáp ứng tần số.

[h,w] = freqz(b,a,n,'whole') và ***[h,f] = freqz(b,a,n,'whole',Fs)*** sử dụng n điểm quanh vòng tròn đơn vị (từ 0 tới 2π hoặc từ 0 tới Fs).

h = freqz(b,a,w) tạo ra đáp ứng tần số tại các điểm tần số được chỉ trong vector ***w***. Các điểm tần số này phải nằm trong khoảng (0 – 2π).

h = freqz(b,a,f,Fs) tạo ra đáp ứng tần số tại các điểm tần số được chỉ trong vector ***f***. Các điểm tần số này phải nằm trong khoảng (0 – Fs).

N: Số lượng hệ số của bộ lọc (N phải là số lẻ).

$\boldsymbol{H_k}$: Đáp ứng tần số lấy mẫu với k = 0, 1, 2, ... và M = (N - 1) /2.

B: Hệ số của bộ lọc.

2.3.2. Bộ lọc IIR (Infinite Impulse Response Filter):

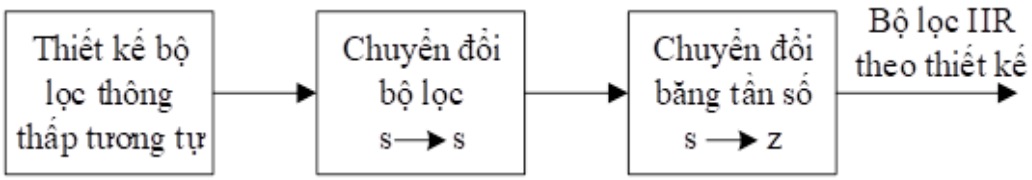
Bộ lọc số FIR có ưu điểm nổi bật là pha tuyến tính. Nói một cách khác, bộ lọc FIR pha tuyến tính đảm bảo được cùng một độ trễ với các nhóm tần số, mỗi nhóm là một tập hợp các tần số lân cận nào đó. Thực nghiệm cho thấy tai người về phần nào đó có khả năng nhận biết được trễ nhóm của tín hiệu âm thanh. Bộ lọc có đáp ứng xung chiều dài vô hạn, hay bộ lọc số IIR, không đảm bảo được tính chất này.

Trong những trường hợp pha tuyến tính không phải là yêu cầu bắt buộc trong thiết kế thì việc lựa chọn bộ lọc FIR hay bộ lọc IIR đều được chấp nhận. Tuy nhiên, bộ lọc IIR thường được lựa chọn hơn vì một số lý do. Thứ nhất, nếu có cùng yêu cầu về độ suy giảm thì bộ lọc IIR đơn giản hơn nhiều so với bộ lọc FIR dẫn đến số phép tính để thực hiện trong bộ lọc IIR ít hơn bộ lọc FIR và các phần tử nhớ trong kết cấu của bộ lọc IIR sẽ ít hơn bộ lọc FIR. Thứ hai, bộ lọc IIR được thiết kế thông qua việc chuyển đổi các thiết kế của bộ lọc tương tự sang bộ lọc số và rất may mắn là các bảng thông số trong thiết kế bộ lọc số có thể tra được trong rất nhiều các tài liệu.

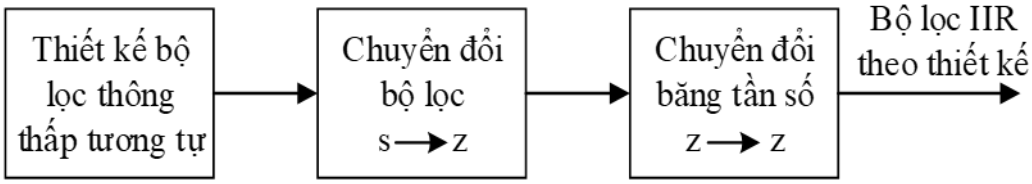
Bộ lọc số IIR, trên nguyên tắc là chuyển đổi từ thiết kế của bộ lọc tương tự thông qua một trong một số phương pháp chuyển đổi bộ lọc. Các phương pháp chuyển đổi sẽ được trình bày tóm tắt trong phần này. Một mặt, các định dạng có sẵn

và các bảng tra cho chúng chỉ áp dụng đối với bộ lọc thông thấp. Do đó, để có được kết quả cuối cùng là bộ lọc số với các loại khác, ví dụ bộ lọc thông dải, quá trình thiết kế cần có một bước để thực hiện việc chuyển đổi băng tần số theo một trong hai cách tiếp cận: hoặc chuyển đổi băng tần số tương tự, hoặc chuyển đổi băng tần số. Nhìn chung, con đường để đi đến một thiết kế bộ lọc số IIR có 2 cách thức được mô tả như sau:

Sơ đồ 2.1 Cách thức lọc số IIR thứ nhất:

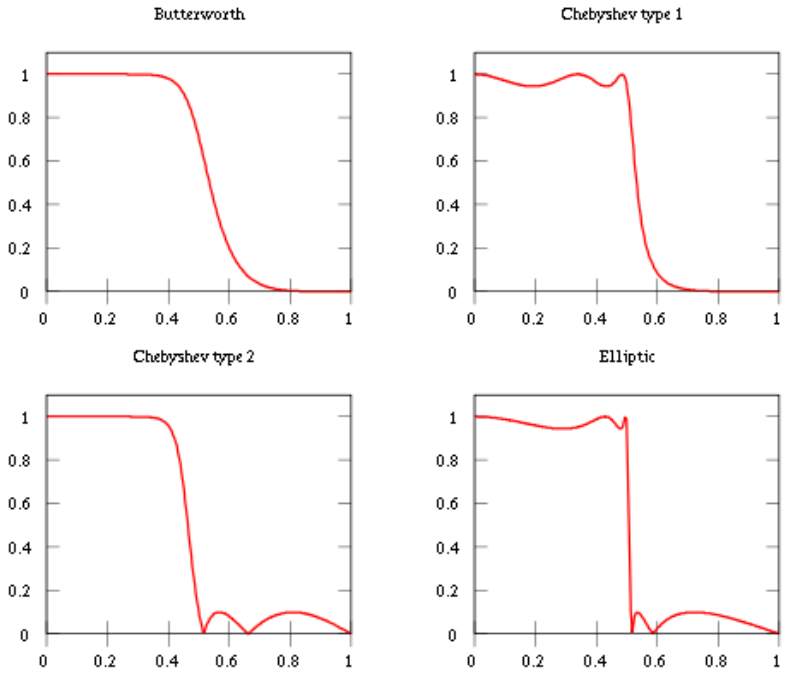


Sơ đồ 2.2 Cách thức lọc số IIR thứ hai:



Tất cả các định dạng bộ lọc đều dựa trên nguyên tắc lựa chọn hàm đáp ứng tần số của bộ lọc thực tế xấp xỉ với đáp ứng tần số của bộ lọc lý tưởng và điểm cực của hàm đáp ứng tần số của bộ lọc thực tế được phân bố sao cho hệ thống là nhân quả và ổn định. Các hàm số bình phương biên độ thường được lựa chọn có dạng gợn sóng vừa phải trong khoảng từ 0 đến tần số cắt và giảm mạnh khi vượt ra ngoài tần số cắt đồng thời có xu hướng giảm về 0. Điều này tương đương với hàm gợn sóng bị chặn trong khoảng từ 0 đến 1 (với 1 là tần số đã được chuẩn hoá bởi tần số cắt Ω_c) và tăng nhanh khi vượt ra ngoài 1.

Có 4 định dạng cơ bản thường được vận dụng trong quá trình thiết kế bộ lọc tương tự là: bộ lọc Butterworth, bộ lọc Chebyshev-1, bộ lọc Chebyshev-2 và bộ lọc Elliptic.



Hình 2.7 Đồ thị biểu diễn tín hiệu qua 4 bộ lọc thuộc loại IIR

2.3.3. Bộ lọc Butterworth:

a) Giới thiệu:

Butterworth là một bộ lọc thông thấp thuộc loại lọc IIR. Hàm bình phương biên độ của đáp ứng tần số bộ lọc Butterworth bậc N được cho bởi phương trình:

$$|H_a(j\omega)|^2 = \frac{1}{1 + (\frac{\Omega}{\Omega_c})^{2N}} \qquad \text{(với } \Omega_c \text{ là tần số cắt)}$$

Các điểm cực của hàm bình phương biên độ của hàm truyền đạt $|H_a(s)|^2$ là :

$$S_{pk} = \Omega_c e^{j\frac{\Pi}{2N}(2k+1+N)}$$

Dẫn đến các điểm cực của hàm truyền đạt $H_a(s)$ là N điểm nằm trên nửa đường tròn tâm O bán kính Ω_c ở nửa bên trái mặt phẳng S và N điểm này đối xứng qua trục thực. Giá trị thích hợp của bậc bộ lọc thông thấp Butterworth được tính theo công thức sau:

$$N = \left[\frac{log[(10^{\frac{R_p}{10}} - 1).(10^{\frac{A_S}{10}} - 1)]}{2log(\frac{\Omega_S}{\Omega_p})} \right]$$

(Giá trị nguyên nhỏ nhất lớn hơn hoặc bằng biểu thức trong dấu)

b) Thiết kế bộ lọc Butterworth sử dụng MATLAB:

Các hàm được sử dụng để thiết kế bộ lọc Butterworth (loại IIR) trong Matlab gồm hàm ***buttord()*** và hàm ***butter()***.

- ***Hàm buttord():***

Cú pháp: ***[N, Wn] = buttord(Wp, Ws, Rp, Rs)***

Chức năng xác định bậc ***N*** của bộ lọc được chia trên thang tần số ***Wn***.

Đối với các mạch lọc thông thấp thì ***Wp*** và ***Ws*** là các tần số ở mép dải thông và dải chặn với ***Wp*** < ***Ws***. Các tần số này phải nằm giữa 0 và 1, với 1 tương ứng với *radians/sample*. Nếu tần số lấy mẫu f, tần số của dải thông fp và của dải chặn fs, quy định bằng Hz, thì khi đó Wp = 2fp/f và Ws = 2fs/f.

Rp là độ mấp mô của dải thông.

Rs là độ suy giảm của dải chặn.

Đối với các mạch lọc thông cao ***Wp*** > ***Ws***. Đối với mạch lọc thông dải và chắn dải thì ***Wp*** và ***Ws*** là những vector có chiều dài bằng 2 quy định các mép của dải chuyển tiếp, tần số ở mép thấp hơn là phần tử đầu tiên của vector. Trong hai trường hợp sau, thì Wn cũng là vector có độ dài bằng 2, còn N là nửa bậc của mạch lọc cần thiết kế.

- ***Hàm butter():***

Cú pháp: ***[B,A] = butter(N,Wn)***

[B,A] = butter(N,Wn,'low')

Hàm này xác định hàm truyền của mạch lọc thông thấp với bậc ***N*** và với thừa số chia thang tần số ***Wn*** nằm giữa 0 và 1 ứng với ½ tốc độ lấy mẫu. Hai thông số ***N*** và ***Wn*** đã được tìm ra nhờ hàm ***buttord()***.

- Với bộ lọc thông cao:

[B,A] = butter(N,Wn,'high')

- Với bộ lọc thông dải:

[B,A] = butter(N,Wn,'bandpass')

(Wn là vector có chiều dài bằng 2 với *Wn=[W1, W2] ,W1<W<W2*)

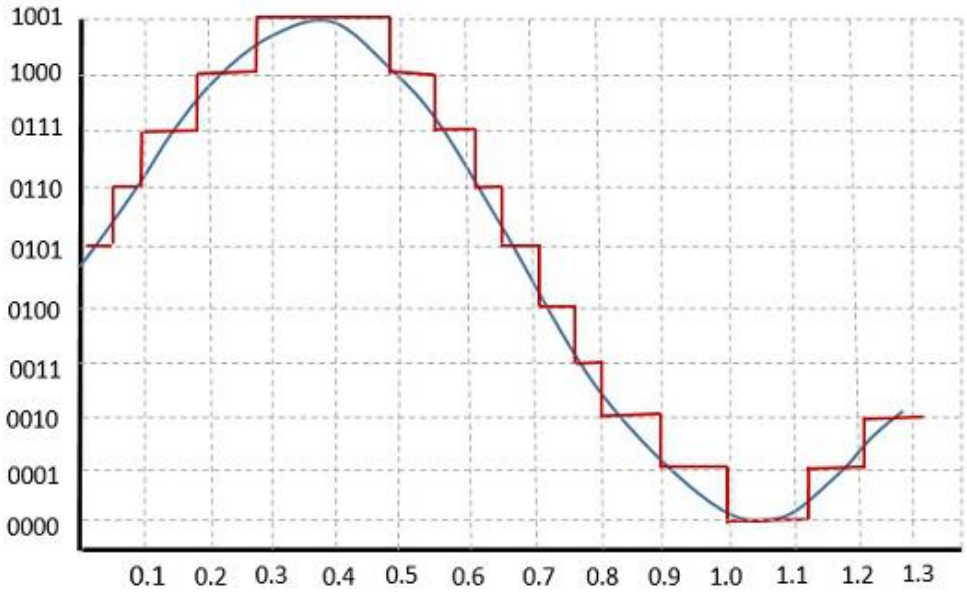
- Với bộ lọc chắn dải:

[B,A] = butter(N,Wn,'stop')

(Wn là vector có chiều dài bằng 2 với *Wn=[W1, W2] ,W1<W<W2*)

2.4. Lượng tử hóa tín hiệu âm thanh:

Lượng tử hóa là quá trình rời rạc hóa tín hiệu về mặt biên độ, cụ thể là thay thế tất cả các giá trị của tín hiệu nằm trong một khoảng xác định nào đó thành một giá trị duy nhất. Miền giá trị của tín hiệu được chia thành một số hữu hạn các khoảng chia. Như vậy, độ lớn của tín hiệu sau khi lượng tử hóa chỉ có thể nhận một trong số hữu hạn các giá trị cho trước.



Hình 2.8 Một mẫu tín hiệu được thực hiện lượng tử hóa nhị phân 4 bit

Tuy nhiên, việc thực hiện lượng tử hóa đối với các tín hiệu thoại (speech processing) sẽ khó khăn hơn, vì trước đó mẫu tín hiệu khi lấy mẫu ổn định lên đến khoảng 44000 mẫu. Việc lượng tử hóa với số lượng mẫu quá lớn như vậy là không thể. Vì thế, trước khi xử lý lượng tử hóa, cần thông qua bước ***nén tín hiệu*** cần lượng tử. Như vậy, việc lượng tử hóa tín hiệu cơ bản cần đi qua 2 bước chính:

1. ***Nén tín hiệu (compress):***

   Nén tín hiệu theo hàm logarithm, mục đích là để tín hiệu ở mức biên độ nhỏ sẽ thay đổi nhiều mức hơn so với ở các giá trị biên độ lớn. Do đó, sai số lượng tử tương đối ở các mức biên độ nhỏ và lớn sẽ không chênh lệch nhau nhiều như đối với trường hợp không nén. Để khôi phục lại đúng tín hiệu ban đầu thì sau khi giải mã, ta phải đưa qua một bộ giãn tín hiệu (expander) có đặc điểm tuyến truyền đạt là nghịch đảo của đặc tuyến của bộ nén (compressor).

   Sự kết hợp của bộ nén và bộ giãn tín hiệu gọi chung là bộ nén giãn tín hiệu (compander).

Hai luật nén giãn thường được sử dụng trong xử lý tín hiệu thoại là luật μ sử dụng ở Bắc Mỹ và luật A sử dụng ở châu Âu:

- **Luật μ:**

$$y = y_{max} \frac{ln[1+\mu(\frac{|x|}{x_{max}})]}{ln(1+\mu)} sgn(x)$$

Theo các chuẩn ở Bắc Mỹ, giá trị của μ là 255. x_{max} và y_{max} lần lượt là các giá trị dương lớn nhất của x và y.

- **Luật A:**

$$y = \begin{cases} y_{max} \frac{A(\frac{|x|}{x_{max}})}{1+lnA} sgn(x) \\ y_{max} \frac{1+ln[A(\frac{|x|}{x_{max}})]}{1+lnA} sgn(x) \end{cases} \quad \text{nếu} \quad \begin{cases} 0 < \frac{|x|}{x_{max}} \leq \frac{1}{A} \\ \frac{1}{A} < \frac{|x|}{x_{max}} \leq 1 \end{cases} \quad \text{với A là hằng số}$$

Giá trị chuẩn của A là 87,6.

2. *Lượng tử hóa (quantization):*

Sau khi tín hiệu cần lượng tử hóa đã được nén lại qua bộ nén – giãn tín hiệu, thực hiện quá trình lượng hóa. Tập hợp các khoảng chia gọi là sự phân hoạch của tín hiệu (partition). Tập các giá trị thay thế cho mỗi khoảng chia gọi là bộ mã (codebook).

MATLAB biểu diễn phân hoạch của tín hiệu bằng một vector mà các phần tử của nó là các điểm ranh giới giữa hai khoảng chia liên tiếp. Ví dụ, nếu tín hiệu có miền xác định là R, được phân hoạch thành các khoảng (-∞, 0], (0,2], (2,4] và (4,+∞) thì có thể biểu diễn sự phân hoạch này bằng vector:

```
>> partition = [0,2,4];
```

Tương ứng với vector phân hoạch tín hiệu là vector biểu diễn bộ mã tín hiệu. Các phần tử của nó là các giá trị thay thế trong mỗi khoảng chia tương ứng của phân hoạch. Nếu ta thay thế các giá trị trong khoảng (-∞, 0] bằng -1, các giá

trị trong khoảng (0,2] bằng 1, các giá trị trong khoảng (2,4] bằng 3 và các giá trị trong khoảng (4, +∞] bằng 5 thì vector biểu diễn bộ mã sẽ là:

>> codebook = [-1,1,3,5];

Như trên đã đề cập, thông thường người ta sẽ phân hoạch miền xác định của tín hiệu thành 2^v khoảng, sau đó mỗi khoảng tín hiệu sẽ được lượng tử hoá, sau đó mã hoá bằng một từ mã nhị phân có chiều dài ***v*** bit. Phương pháp lượng tử hoá này được gọi là phương pháp điều mã xung (Pulse Code Modulation). Phương pháp này không cần đòi hỏi bất kỳ thông tin nào về tín hiệu ở các thời điểm trước đó. Trong thực tế, vì tín hiệu thường thay đổi chậm từ thời điểm lấy mẫu này sang thời điểm lấy mẫu kế tiếp nên nếu ta thực hiện lượng tử và mã hoá các giá trị sai biệt giữa thời điểm hiện tại với thời điểm trước đó thì sẽ tốn ít giá trị hơn so với mã hoá đầy đủ độ lớn của tín hiệu. Trên cơ sở này, ta có một phương pháp lượng tử hoá mới, gọi là lượng tử hoá tiên đoán, trong đó giá trị của tín hiệu ở thời điểm hiện tại sẽ được tính thông qua một số các giá trị của tín hiệu ở các thời điểm quá khứ. Tiêu biểu cho loại lượng tử hoá này là kỹ thuật điều mã xung vi sai (*DPCM – Differential Pulse Code Modulation*).

Để thực hiện mã hoá DPCM, ta không những phải xác định sự phân hoạch và bộ mã lượng tử mà còn phải xác định hàm dự đoán, để dự đoán giá trị của tín hiệu ở thời điểm hiện tại. Thông thường, người ta sử dụng hàm dự đoán tuyến tính:

$$y(k) = p(1)x(k-1) + p(2)x(k-2) + ... + p(m-1)x(k-m+1) + p(m)x(k-m)$$

Trong đó x là tín hiệu gốc còn y(k) là giá trịdự đoán của x(k); p là một vector gồm các hằng số thực.

Thay vì lượng tử hoá tín hiệu ***x***, ta sẽ thực hiện lượng tử hoá tín hiệu sai số dự đoán ***(y – x)***. ***m*** được gọi là bậc dự đoán. Trường hợp đặc biệt m = 1 được gọi là điều chế delta.

CHƯƠNG III

TẠO GIAO DIỆN ỨNG DỤNG CƠ BẢN BẰNG MATLAB

3.1. Giới thiệu chung về Matlab:

MATLAB là một bộ chương trình phần mềm lớn dành cho tính toán kỹ thuật. ta có thể dùng MATLAB để:

- Tính toán.
- Phát triển thuật toán.
- Thu thập dữ liệu.
- Mô hình và mô phỏng.
- Phân tích dữ liệu.
- Vẽ đồ thị.
- Giao diện đồ họa.

Hình 3.1 Logo phần mềm Matlab

MATLAB là tên viết tắt từ **"*MAT*rix *LAB*oratory"**. Như tên của phần mềm cho thấy, phần cốt lõi của phần mềm là dữ liệu được lưu dưới dạng array (ma trận) và các phép tính toán ma trận, giúp việc tính toán trong MATLAB nhanh và thuận tiện hơn so với lập trình trong C hay FORTRAN. Đặc biệt, khả năng tính toán của MATLAB có thể dễ dàng được mở rộng thông qua các bộ toolbox (toolbox là tập hợp các hàm MATLAB (M-file) giúp giải quyết một bài toán cụ thể).

MATLAB gồm 5 phần chính:

1. **Development Environment:** là một bộ các công cụ giúp ta sử dụng các hàm và tập tin của MATLAB. Nó bao gồm: MATLAB Desktop, Command Window, A Command History, Editor Space, Debugger, Browsers for Viewing Help, The Workspace, Files, The Search Path.
2. **MATLAB Mathematical Function Library:** tập hợp các hàm toán học như sum, sine, số học, …
3. **MATLAB Language (Script):** ngôn ngữ lập trình bậc cao.

4. **Graphics:** các công cụ giúp hiển thị dữ liệu dưới dạng đồ thị. Ngoài ra, nó còn cho phép xây dựng giao diện đồ họa.

5. **MATLAB Application Program Interface (API):** bộ thư viện cho phép ta sử dụng các chức năng tính toán của MATLAB trong chương trình C hay FORTRAN.

Một số tính năng của Matlab:

- MATLAB là ngôn ngữ lập trình cao cấp, cho phép tính toán các con số, hình dung và phát triển ứng dụng.
- Cung cấp môi trường tương tác để khảo sát, thiết kế và giải quyết các vấn đề.
- Cung cấp thư viện lớn các hàm toán học cho đại số tuyến tính, thống kê, phân tích Fourier, bộ lọc, tối ưu hóa, tích phân và giải các phương trình vi phân bình thường.
- MATLAB cung cấp các đồ thị được tích hợp sẵn để hiển thị hình ảnh dữ liệu và các công cụ để tạo đồ thị tùy chỉnh.
- Giao diện lập trình của MATLAB cung cấp các công cụ phát triển để nâng cao khả năng bảo trì chất lượng mã và tối đa hóa hiệu suất.
- Cung cấp các công cụ để xây dựng các ứng dụng với các giao diện đồ họa tùy chỉnh.
- Cung cấp các hàm để tích hợp các thuật toán dựa trên MATLAB với các ứng dụng bên ngoài và các ngôn ngữ khác như C, Java, NET và Microsoft Excel.

3.2. Cách tạo giao diện ứng dụng cơ bản bằng Matlab:

3.2.1. GUI là gì?

GUI (Graphical User Interface) là giao diện đồ họa có điều khiển bởi nhiều thanh công cụ được người lập trình tạo sẵn, cho tương tác giữa người dùng và giao diện chương trình, Mỗi chương trình được người lập trình tạo sẵn giao diện thực

hiện một vài chức năng được lập trình và giao tiếp với người sử dụng. Ứng dụng của Matlab lập trình giao diện rất mạnh và dễ thực hiện, nó có thể tạo ra giao diện người dùng tương tự VBB, C++…

Matlab GUI bao gồm đầy đủ các chương trình hỗ trợ như thực hiện phép toán logic, mô phỏng không gian 2D, 3D, đọc hiển thị dữ liệu, liên kết đa phương tiện, giao tiếp với người dùng thông qua hình ảnh, các nút nhấn thực thi ...

Hình 3.2 Một GUI cơ bản được tạo bằng MATLAB

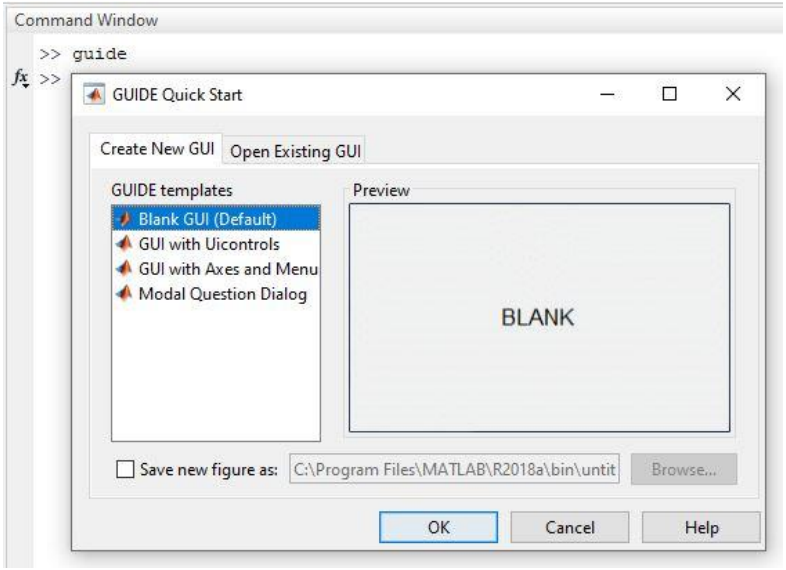
Có 2 phương pháp để lập trình GUI:

- Cách đơn giản nhất là sử dụng công cụ có sẵn trong GUI Matlab để lập trình. Ưu điểm của cách này là dễ thực hiện và các hàm (Function) được GUI tự tạo sẵn.
- Cách thứ hai là lập trình từ siêu tệp file .m bằng các hàm do người lập trình tự viết, nó có ưu điểm là tùy biến cao. Tuy nhiên cách này khó hơn và đòi hỏi người lập trình phải có hiểu biết sâu và trình độ.

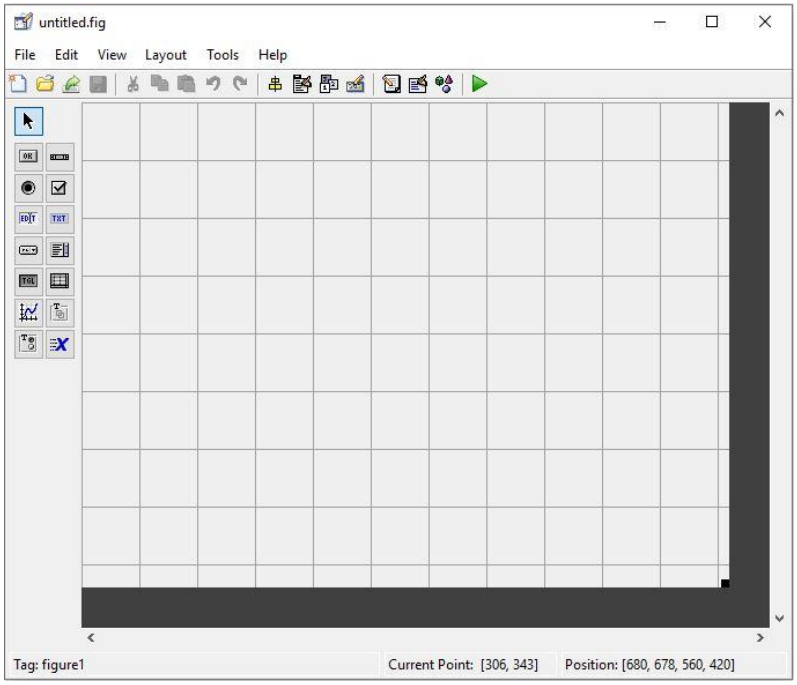
Ở đây, ta thực hiện cách thứ nhất để lập trình giao diện với những công cụ được hỗ trợ sẵn trong Matlab.

3.2.2. Khởi động GUI:

- Trong cửa sổ Command Window gõ lệnh "guide" và enter.



- Chọn: "Blank GUI (Default)" để tạo một giao diện GUI trống. Các dòng còn lại để khởi động GUI với một giao diện được tạo sẵn.



(Nhóm hình 3.3 Hướng dẫn khởi động GUI)

- Trước khi tạo giao diện ta lưu file lại, Matlab sẽ tự động lưu hai file, một file đuôi .m và một file đuôi .fig hoặc ta có thể nhấn F5, Matlab sẽ chuyển đường dẫn đến thư mục lưu file, chọn nơi cần lưu và nhấn Save.

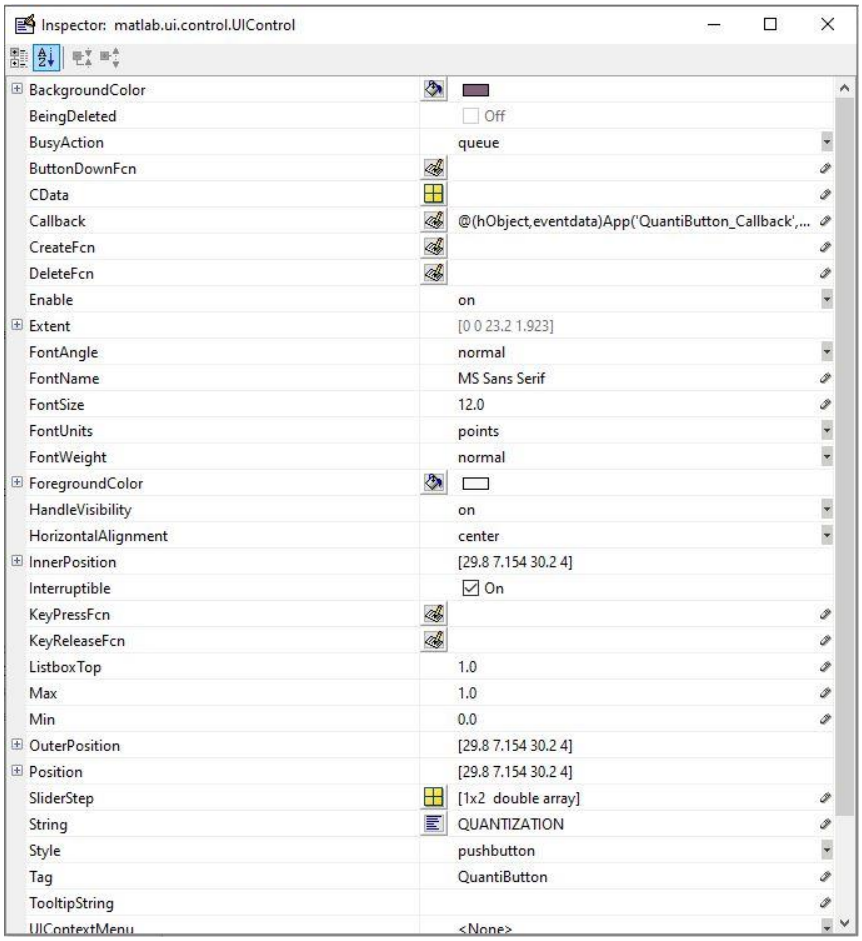
3.2.3. Các công cụ trong GUI:

- ***Push Button:*** là nút nhấn, khi nhấn vào sẽ thực thi lệnh trong cấu trúc hàm callback của nó.
- ***Slider:*** là thanh trượt cho phép người dùng di chuyển thanh trượt để thục thi lệnh.
- ***Radio Button:*** nó giống như Check Box nhưng thường được sử dụng để tạo sự lựa chọn duy nhất, tức là 1 lần chỉ được chọn 1 trong số các nhóm nhiều nút. Khi một ô được chọn thì các ô còn lại trong nhóm bị bỏ chọn.
- ***Check box:*** sử dụng để đánh dấu tích (thực thi) vào và có thể check nhiều ô để thực thi.
- ***Edit Text:*** là nơi các kí tự được nhập vào từ người dùng, người dùng có thể thay đổi được.
- ***Static Text:*** là các kí tự được hiển thị thông qua các callback, hoặc thông thường để viết nhãn cho các biểu tượng, người dùng không thể thay đổi nội dung.
- ***Pop-up Menu:*** mở ra danh sách các lực chọn khi người dùng nhấp chuột vào. Chỉ chọn được 1 mục trong danh sách các mục.
- ***List Box:*** hộp thoại danh sách cách mục, cho phép người dùng chọn một hoặc nhiều mục.
- ***Toggle Button:*** là nút nhấn có 2 điều khiển, khi nhấp chuột và nhả ra, nút nhấn được giữ và lệnh thực thi, khi nhấp chuột vào lần thứ 2, nút nhấn nhả ra, hủy bỏ lệnh vừa thực thi.
- ***Table:*** tạo ra một bảng tương tự trong Excel.
- ***Axes:*** đây là giao diện đồ họa hiển thị hình ảnh, nó có nhiều thuộc tính bao gồm: không gian 2D (theo trục đứng và trục ngang), 3D (hiển thị không gian 3 chiều)

- ***Panel:*** tạo ra một mảng nhóm các biểu tượng lại với nhau giúp ta dễ kiểm soát và thao tác khi di chuyển
- ***Button Group:*** quản lí sự lựa chọn của nút Radio Button.
- ***Active Control:*** quản lí một nhóm các bút hoặc các chương trình liên quan với nhau trong Active.

3.2.4. Giới thiệu hộp thoại Inspector:

Tất cả các công cụ vừa nêu trên để sử dụng, ta nhấp chọn và kéo thả vào vùng cần thiết kế. Mỗi hộp thoại có các thông số riêng, để điều chỉnh các thông số liên quan, ta nhấp đôi vào hộp thoại đó (hoặc click chuột phải chọn Property Inspector). Hộp thoại Inspector sẽ hiện ra cho phép ta thay đổi các thông số:



Hình 3.4 Hộp thoại chức năng Inspector

- Phía bên trái của hộp thoại Inspector là tên thuộc tính, có thể gọi thực thi các thuộc tính này bằng các lệnh.
- Phía bên trái là giá trị của thuộc tính, giá trị này do người dùng đặt, có thể thay đổi thông qua các lệnh gọi (callback), hoặc được thiết lập trước.
- Một vài thuộc tính cơ bản:
  - ***String:*** là trường để hiển thị tên nào đó mà ta muốn ở GUI. Nó không ảnh hưởng đến chương trình mà ta thực hiện.
  - ***Style:*** cho phép thay đổi đối tượng này thành một đối tượng khác mà ta muốn ví dụ như thay đổi Static Text thành Push Button. Nhưng thường thì bạn cũng không nên thay đổi.
  - ***Tag:*** có chức năng là định danh cho đối tượng để khi vào lập trình có thể lấy tên này để gọi xử lý một chức năng nào đó. Ta có thể thay đổi thành bất cứ tên nào nhưng không được đặt trùng với tên mà các đối tượng khác đã đặt trước đó.

3.3. Tìm hiểu về file .m :

- Khi ***Run***, Matlab sẽ tự chạy file .m và hiển thị giao diện đồ họa lên màn hình.
- Trong file .m, tất cả các hàm function đều được Matlab hỗ trợ tạo sẵn các hàm chức năng có liên quan, ta chỉ việc thao tác trên đó.
- Mỗi sau một hàm bất kì đều có các chú thích bên dưới (sau dấu %), ta có thể xóa toàn bộ chúng đi để dễ nhìn cũng không ảnh hưởng đến các hàm.
- Trong file .m có sẵn các hàm callback, ta có thể tìm trong file .m các hàm liên quan để viết câu lệnh.

Một số hàm trong file .m:

```
function App_OpeningFcn(hObject, eventdata, handles, varargin)
```

+ ***App:*** là tên file .m mà ta đã lưu trước đó.

+ ***OpeningFcn:*** Function này đảm nhận nhiệm vụ là phần để khởi tạo các biến, các yếu tố mà khi ta muốn chạy trương trình thì lập tức các biến này sẽ được gọi nên đầu tiên.

+ ***hObject:*** hàm truy cập nội bộ của mỗi function riêng lẻ
+ ***evendata:*** hàm xác đinh thuộc tính của function
+ ***handles***: hàm truy cập liên kết giữa các function, nó bao gồm tất cả các cấu trúc của người dùng, được sử dụng để truy xuất qua các điều khiển khác.

```
function StartButton_Callback(hObject, eventdata, handles)
```

+ ***StartButton:*** giá trị của Tag mà ta đã đặt khi tạo giao diện.
+ ***Callback***: Function khi ta thực hiện thao tác gì đó trên GUI. Giả sử khi nhấn chuột vào một ***PushButton*** nào đó thì Function sẽ thực hiện các câu lệnh bên trong mà ta đã viết.
+ Hàm callback được lập trình cho các nút button, checkbox, edit text… nhưng static text và axes… thì không có hàm callback.

- Hàm **Get** cho phép ta gọi thuộc tính của đối tượng.

- Hàm **Set** cho phép ta đặt giá trị cho thuộc tính của đối tượng.

App.m

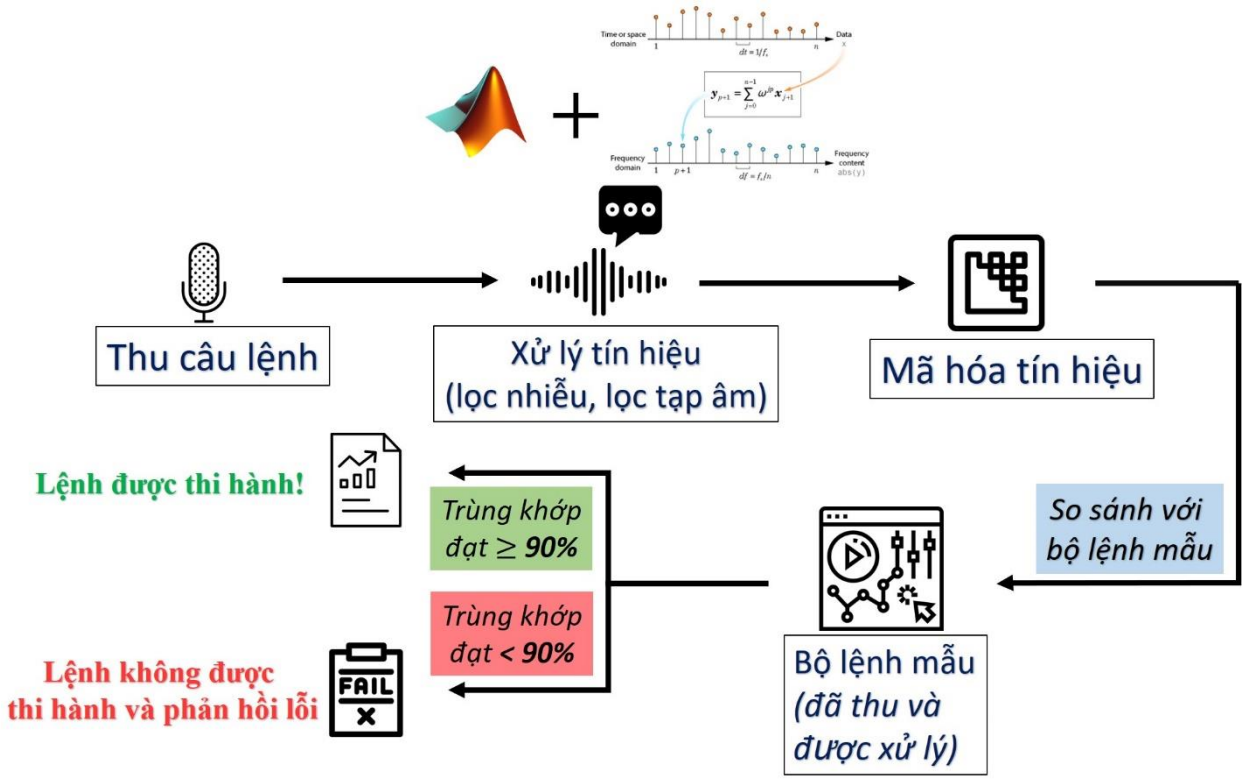
```
function varargout = App(varargin)
    % Begin initialization code - DO NOT EDIT
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @App_OpeningFcn, ...
                       'gui_OutputFcn',  @App_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   []);
    if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
    end

    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end
    % End initialization code - DO NOT EDIT
end
    % --- Executes just before App is made visible.
function App_OpeningFcn(hObject, eventdata, handles, varargin)
    % Choose default command line output for App
    handles.output = hObject;
    % Update handles structure
    guidata(hObject, handles);
end

  % --- Outputs from this function are returned to the command line.
function varargout = App_OutputFcn(hObject, eventdata, handles)
    % varargout  cell array for returning output args (see VARARGOUT);
    % Get default command line output from handles structure
    varargout{1} = handles.output;
end

  % --- Executes on button press in StartButton.
function StartButton_Callback(hObject, eventdata, handles)
    rec = audiorecorder(44100,16,1);
    disp('Start speaking');
    record(rec);
    handles.rec = rec;
    handles.fullFile = '';
    guidata(hObject,handles);
```

Hình 3.5 Hình ảnh thực tế chương trình trong file .m được cắt ra từ chương trình của ứng dụng

CHƯƠNG IV

TRIỂN KHAI VÀ THỰC HIỆN

4.1. Sơ đồ chung của quá trình xử lý:



Sơ đồ 4.1 Sơ đồ mô tả chung quá trình xử lý của ứng dụng

Quá trình xử lý được diễn ra theo thứ tự như sau:

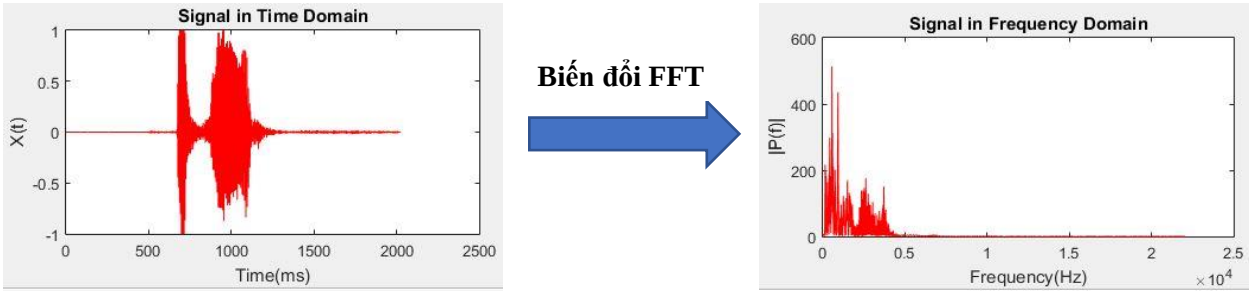
1. Quá trình lấy dữ liệu nguồn cần xử lý: Có thể thực hiện bằng 2 cách, thu âm trực tiếp từ ứng dụng hoặc upload file thu âm (*.wav*) có sẵn từ máy tính.
2. Xử lý tín hiệu:
   - Vẽ biểu đồ thể hiện tín hiệu trên miền thời gian.
   - Biến đổi tín hiệu qua miền tần số bằng FFT và vẽ biểu đồ.
   - Lọc nhiễu, tạp âm tín hiệu qua bộ lọc thông dải Butterworth.
   - Xuất, phát file sau khi xử lý và vẽ biểu đồ tín hiệu để so sánh sự tương quan.

3. Mã hóa tín hiệu (lượng tử hóa):
   - Nén file tín hiệu sau khi xử lý thành 1024 mẫu để thực hiện lượng tử hóa.
   - Lượng tử hóa tín hiệu đã nén thành mã nhị phân 8 bit.
   - Xuất file .txt với bộ dữ liệu 8x1024 mã nhị phân của tín hiệu sau xử lý.
4. Lưu trữ và phát triển sau này: Sau khi thu đủ một lượng bộ lệnh mẫu được lượng tử hóa, ta có tiếp tục phát triển hệ thống với việc thu âm câu lệnh đồng thời so sánh với bộ lệnh mẫu có sẵn trong cơ sở dữ liệu để máy thực hiện đúng mệnh lệnh mà con người nói ra.

4.2. Áp dụng lý thuyết về FFT trong việc triển khai ứng dụng:

Đối với việc áp dụng lý thuyết FFT cho ứng dụng của, thuật toán FFT có chức năng biến đổi tín hiệu câu lệnh âm thanh nguồn từ miền thời gian sang miền tần số. Vì hầu hết việc xử lý tín hiệu âm thanh đều được thực hiện trên miền tần số, nên việc áp dụng FFT rất quan trọng. MATLAB hỗ trợ các hàm cho phép ta thực hiện biến đổi một tín hiệu với FFT bằng các lệnh code như sau:

```
L = length(data); %lấy ra chiều dài của tín hiệu
NFFT = 2^nextpow2(L); %hàm nextpow2(len) cho ra giá trị p nhỏ nhất thỏa
%công thức (2^p>=len) và gán NFFT=2^p
f = fs / 2 * linspace(0, 1, NFFT/2+1); %hàm linspace(0,1,NFFT/2+1) tạo
%(NFFT/2+1) với các giá trị cách đều nhau từ 0 đến 1
xf = abs(fft(data, NFFT));
%vẽ tín hiệu:
plot(f, xf(1:NFFT/2+1), 'r');
title('Signal in Frequency Domain');
xlabel('Frequency(Hz)');
ylabel('|P(f)|');
```

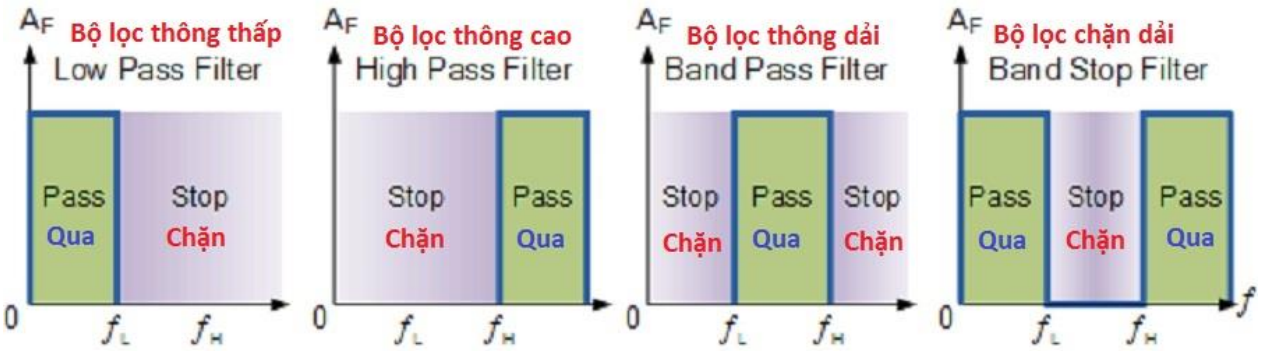


Tính hiệu trong miền thời gian *Tính hiệu trong miền tần số*

Nhóm hình 4.1 Kết quả của quá trình biến đổi FFT tín hiệu thực tế

4.3. Ứng dụng của bộ lọc Butterworth trong việc triển khai ứng dụng:

MATLAB hỗ trợ trong việc tạo ra bộ lọc Butterworth cho việc lọc một loại tín hiệu bất kì với 4 kiểu lọc khác nhau: thông thấp, thông cao, thông dải và chặn dải.



Nhóm hình 4.2 Đồ thị làm việc của 4 bộ lọc lý tưởng

Bộ lọc tần số là một bộ phận có chức năng cho qua hoặc chặn lại một dải tần số nào đó. Nói cách khác, bộ lọc tần số cho phép tách được các âm thanh có tần số khác nhau ra khỏi tín hiệu đầy đủ ban đầu. Dựa vào đáp ứng tần số, có thể chia bộ lọc ra làm 4 loại như hình trên:

- ***Bộ lọc thông thấp LPF (Low Pass Filter):*** Bộ lọc thông thấp chỉ cho các tần số thấp hơn tần số cắt fL đi qua, các tần số cao hơn sẽ bị chặn lại.
- ***Bộ lọc thông cao HPF (High Pass Filter):*** Bộ lọc thông cao chỉ cho các tần số cao hơn tần số cắt fH đi qua, các tần số thấp hơn sẽ bị chặn lại.
- ***Bộ lọc thông dải BPF (Band Pass Filter):*** Bộ lọc thông dải chỉ cho các tần số nằm trong dải thông fL - fH đi qua, các tần số cao hơn và thấp hơn sẽ bị chặn lại. Bộ lọc thông dải bản chất là sự kết hợp của hai bộ lọc độc lập, bao gồm một bộ lọc thông cao có tần số cắt fL kết hợp với một bộ lọc thông thấp có tần số cắt fH.
- ***Bộ lọc chặn dải BSF (Band Stop Filter):*** Bộ lọc chặn dải fL - fH chỉ cho các tần số nằm ngoài dải chặn đi qua, các tần số nằm trong dải sẽ bị chặn lại. Bộ lọc chặn dải bản chất là sự kết hợp của hai bộ lọc độc lập, bao gồm một bộ lọc thông thấp có tần số cắt fL kết hợp với một bộ lọc thông cao có tần số cắt fH.

Như vậy, để đáp ứng đúng nhu cầu phù hợp cho việc phát triển ứng dụng, nhóm chọn sử dụng bộ lọc loại thông dải với khoảng tần số cắt phù hợp với bảng phân tích tần số được nghiên cứu và chỉ ra như sau:

Tần số (Hz)	Mô tả
16 - 32	Ngưỡng dưới của khả năng nghe của con người, và nốt thấp nhấp của đàn đại phong cầm.
32 - 512	Tần số nhịp điệu, nơi có các nốt thấp và cao của giọng nam trầm.
512 - 2048	Độ nghe rõ tiếng nói con người, có tiếng kim.
2048 - 8192	Âm thanh lời nói, nơi có âm môi và âm xát.
8192 - 16384	Chói, tiếng chuông và cái chũm chọe và âm xuýt
16384 - 32768	Trên chói, đạt tới âm thanh âm u và hơi quá ngưỡng nghe của con người

Bảng 4.1 Bảng tần số âm thanh

Như vậy, với việc chọn lựa bộ lọc Butterworth dạng lọc thông dải cùng với khoảng tần số cắt (*512 – 2048Hz: cho qua, loại bỏ âm tần còn lại*), kết hợp với các hàm hỗ trợ của MATLAB, ta có được bộ chương trình lọc chức năng sau:

```
n = 7; %bộ lọc bậc 7
lowFreq = 512 / (fs/2); %tần số mép của dải thông là 512Hz
highFreq = 2048 / (fs/2); %tần số mép của dải chặn là 2048Hz
% Thực hiện lọc nhiễu bằng bộ lọc Butterworth và tín hiệu ra trả về
% biến output:
[b, a] = butter(n, [lowFreq, highFreq], 'bandpass');
out = filter(b, a, data);
```

4.4. Xử lý lượng tử hóa tín hiệu sau khi lọc:

Đầu tiên, để lượng tử hóa được tín hiệu âm thanh sau khi xử lý, đầu tiên ta cần thực hiện quá trình nén tín hiệu. MATLAB cung cấp hàm ***compand*** để thực hiện nén giãn tín hiệu. Hàm này hỗ trợ hai luật nén giãn A và μ đã được đề cập đến trong phần ***"Lý thuyết 2.4"*** trên với cú pháp:

>> out = compand(in, param, v, method)

với:

in là tín hiệu vào còn ***out*** là tín hiệu ra

v là biên độ đỉnh của tín hiệu vào

param là thông số của luật nén giãn (hằng số A hoặc μ)

method có thể nhận một trong các giá trị sau**:**

method	**Chức năng**
'mu/compressor'	Nén theo luật μ
'mu/expander'	Nén theo luật μ
'A/compressor'	Nén theo luật A, μ
'A/expander'	Giãn theo luật A

Bảng 4.2 Bảng tra cứu các hàm chức năng

Như vậy, chúng ta cần cài đặt 1 file hàm thực hiện chức năng nén tín hiệu như sau:

```
%nén tín hiệu trước khi lượng tử hóa
%sig_t là tín hiệu đưa vào
function sig_comp = compressor(sig_t)
A = 87.6;    %sử dụng nén theo luật A với A = 86.7
V = max(sig_t);
sig_comp = compand(sig_t,A,V,'A/compressor');
end
```

Sau khi tín hiệu đã được nén đúng tiêu chuẩn, ta thực hiện lượng tử hóa tín hiệu. Trước tiên, ta cần thực hiện lượng tử hóa tín hiệu sai số dự đoán (y – x) (như đã được đề cập tại phần ***"Lý thuyết 2.4"*** trên). MATLAB Communications Toolbox, hàm dự đoán được sử dụng là hàm dự đoán tuyến tính như trên và được biểu diễn bằng vector:

>> predictor = [0, p(1), p(2), p(3), … , p(m-1), p(m)]

Sau đó, sử dụng hàm ***dpcmenco*** thực hiện quá trình mã hóa với ***quant*** là vector chứa các giá trị lượng tử còn ***index*** là vector chứa các chỉ số tương ứng trong

bộ mã. Như vậy, ta tiếp tục xây dựng được 1 file hàm thực hiện chức năng lượng tử hóa tín hiệu như sau:

```
function [index,sig_quants] = DPCM(sig_comp,codebook,partition)
% lượng tử hóa DPCM với tín hiệu sig_comp sau khi nén
% bits là số bit biểu diễn một mẫu
% sử dụng hàm dự đoán là: y(k) = x(k-1)
predictor = [0 1];% hàm dự đoán với delta bậc 1
[index,sig_quants] = dpcmenco(sig_comp,codebook,partition,predictor);
end
```

Hoàn thành việc xử lý lượng tử hóa, ta tiếp tục tìm cách biến đổi các giá trị index từ 0 – 255 (2^8 giá trị) thành các số nhị phân 8 bit bằng một hàm chuyển Encoder như sau:

```
%biến đổi từ mã thập phân sang nhị phân k bit
function sig_enc = Encoder(index)
sig_enc= de2bi (index,'left-msb'); %biến đổi chuỗi thập phân
%thành chuỗi bit nhị phân
end
```

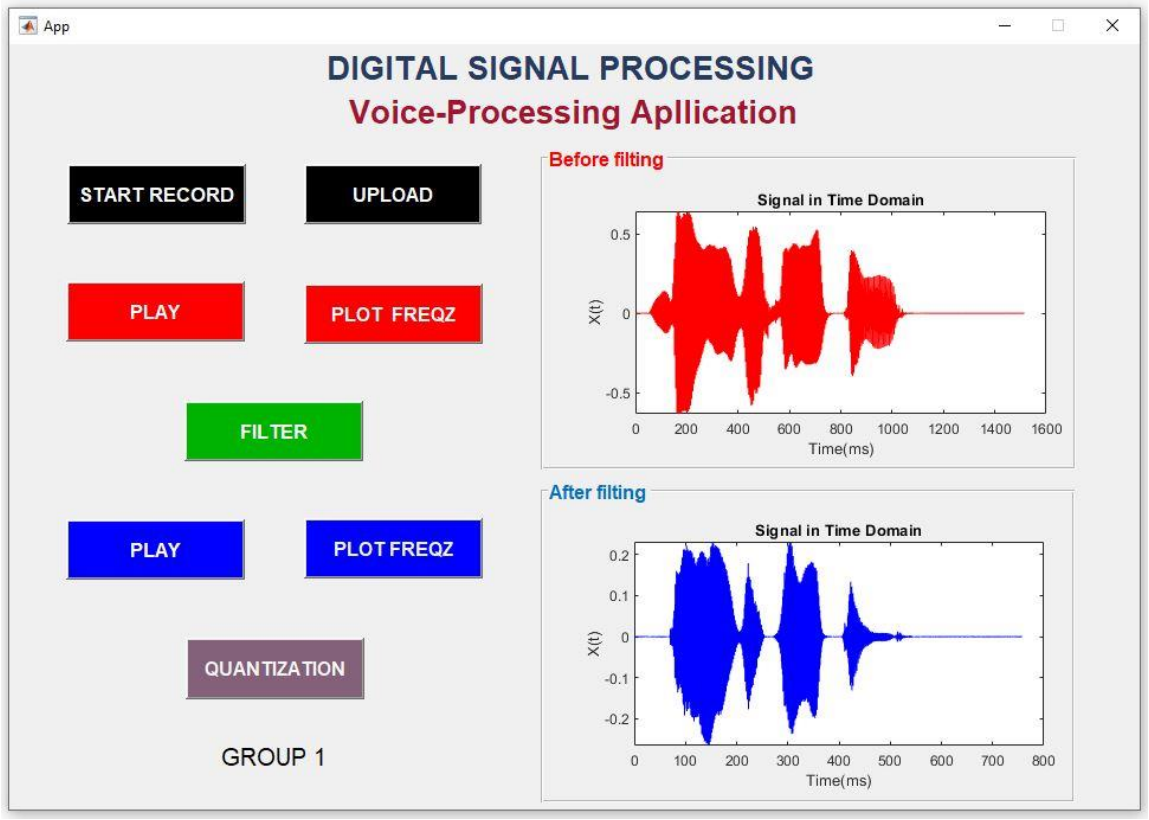
Sau khi có được 3 hàm cần thiết cho quá trình lượng tử hóa, ta bắt đầu thực hiện việc kết hợp chúng và viết vào chương trình chính với đoạn code chương trình đầy đủ cho phần xử lý lượng tử hóa như sau:

```
% Khảo sát tín hiệu cần lượng tử hóa signal:
    l = length(signal);
    k = mod(l,1024);            %đảm bảo độ dài của signal là độ bội của 1024
    signal = signal([1:l-k],1); %loại bỏ k phần tử dư
    Mp = max(signal);
    bits = 8;                   %dùng 8 bit để thực hiện mã hóa cho 1 mẫu
    levels = 2^bits;            %biến lưu lại số mẫu 8 bit mã hóa được
    step_size = (2*Mp)/levels;  %bước nhảy mỗi khoảng
    codebook = [-Mp+step_size:step_size:Mp]; %số đại diện mỗi khoảng, trong
%phần >> partition
    partition = [-Mp+2*step_size:step_size:Mp]; %chia tín hiệu thành 256
%khoảng

% Chia tín hiệu thành các frame, mỗi frame có chứa 1024 mẫu:
    frame = reshape(signal,1024,length(signal)/1024);
    %frame là một ma trận có 1024 hàng và length(signal)/1024 cột
    k = length(signal)/1024;

    for i = 1:k
        sig_t = frame(:,i);%chọn frame thứ i
        sig_comp = compressor(sig_t);%thực hiện nén
        [index,sig_quant] = DPCM(sig_comp,codebook,partition);
        sig_enc = Encoder(index);%chuyển lần lượt các chỉ số index sang mã nhị
%phân
    end
```

Như vậy, ta đã có được bộ mã hóa nhị phân 8 bit của tín hiệu, được lưu lại tại biến sig_enc thành một ma trận kích thước 8x1024. Để thuận tiện cho việc lưu trữ lại bộ dữ liệu tín hiệu câu lệnh âm thanh đã được mã hóa, ta sử dụng phần ghi ra file .txt của MATLAB để lưu ma trận chứa tín hiệu đã được lượng tử ra file riêng biệt với cú pháp như sau:

```
fid = fopen('BinaryCode.txt','wt'); %tạo ra file lưu dạng .txt để ghi dữ
%liệu lên
    for i=1:size(sig_enc,1) %duyệt từng dòng kích thước của ma trận dữ
liệu
        fprintf(fid,'%g\t',sig_enc(i,:)); %ghi lên file từng dòng bit từ
bộ dữ
    %liệu
        fprintf(fid,'\n'); %xuống dòng để ghi các bit tiếp theo
    end
    fclose(fid);    %đóng file kết thúc việc ghi dự liệu
```

4.5. Thực thi chương trình – Hoàn thành ứng dụng:



Hình 4.3 Giao diện ứng dụng và các chức năng hoàn thiện

4.5.1. Ghi âm và tải lên file có sẵn:

- Nút **'START RECORD'** cho phép người sử dụng ghi âm một đoạn âm thanh để phục vụ cho việc xử lý lọc nhiễu.

  + Tạo 1 đối tượng ghi âm recorder, để ghi âm một kênh đơn với tần số lấy mẫu là 44.1 kHz và 16 bit/mẫu:

```
recorder = audiorecorder(44100, 16, 1);
```

  + Ghi âm và ngừng ghi âm:

```
record(recorder);
stop(recorder);
```

- Nút **'UPLOAD'** cho phép người dùng tải lên một file âm thanh có sẵn để thực hiện xử ký lọc nhiễu.

  + Thực hiện chọn file âm thanh .wav hoặc .mp3 và trả về hai biến *name* (tên file) và *path* (đường dẫn file):

```
[name, path] = uigetfile('*.wav; *.mp3', 'Select the record file');
```

  + Kết hợp tên file và đường dẫn tạo thành một đường dẫn xuất file hoàn chỉnh:

```
file = fullfile(path, name);
```

4.5.2. Phát file âm thanh và vẽ đồ thị:

- Nút **'PLAY'** có chức năng phát file âm thanh vừa được ghi âm hoặc file vừa được tải lên từ máy tính.

  + Thực hiện đọc file và trả về 2 biến số *data* (mảng chứa biên độ của mẫu) và *fs* (tần số lấy mẫu):

```
[data, fs] = audioread(file);
```

  + Phát âm thanh từ 2 biến số vừa đọc:

```
sound(data,fs);
```

- Nút **'PLOT TIME'** có chức năng vẽ đồ thị từ file trên theo miền thời gian.

```
plot(data);
title('Signal in Frequency Domain');
xlabel('Time (ms)');
ylabel('X(t)');
```

4.5.3. Biến đổi Fourier nhanh (Fast Fourier Transform – FFT):

- Nút **'PLOT FREQZ'** thực hiện chức năng biến đổi tín hiệu âm thanh vừa đọc ở trên từ miền thời gian sang miền tần số.

+ Hàm ***nextpow2(L)*** tìm giá trị p nhỏ nhất thỏa công thức $2^p \geq L$ và gán $n = 2^p$. Điều này sẽ đệm tín hiệu với các số 0 ở cuối để cải thiện hiệu suất của ***fft***:

```
L = length(data);
n = 2^nextpow2(L);
```

+ Chuyển tín hiệu sang miền tần số:

```
Y = fft(data, n);
```

+ Xác định miền tần số và vẽ lên đồ thị:

```
f = fs*(0:(n/2))/n;
P = abs(Y/n);
plot(f, P(1:n/2+1), 'b');
title('Signal in Frequency Domain');
xlabel('Frequency (Hz)');
ylabel('|P(f)|');
```

4.5.4. Lọc nhiễu:

- Nút **'FILTER'** thực hiện chức năng loại bỏ các tín hiệu nhiễu (tín hiệu có tần số nằm ngoài tần số cắt).

+ Xác định bậc của bộ lọc n = 7, tần số mép của dải thông là 512 Hz, mép của dải chặn là 2048 Hz:

```
n = 7;
Wp = 512 / (fs/2);
```

```
Ws = 2048 / (fs/2);
```

+ Thực hiện lọc nhiễu bằng bộ lọc *Butterworth* và tín hiệu được trả về biến output:

```
[b, a] = butter(n, [Wp, Ws], 'bandpass');
output = filter(b, a, data);
```

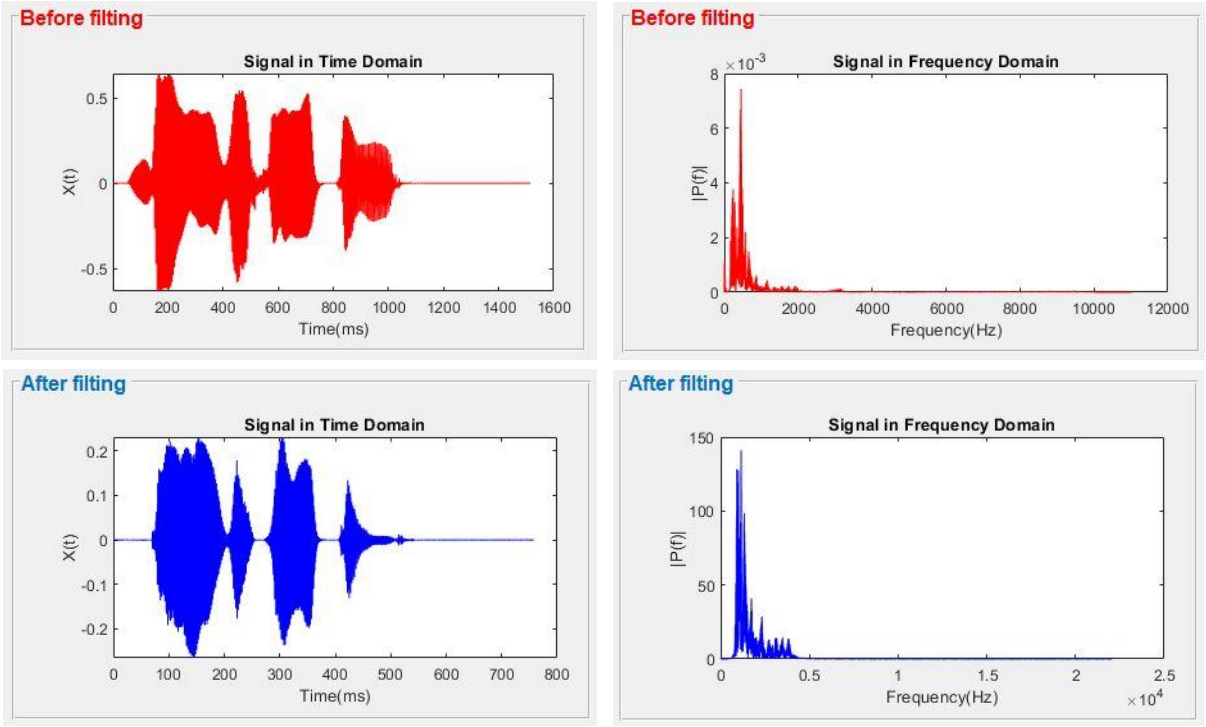
4.5.5. Lượng tử hóa (mã hóa nhị phân):

- Nút **'QUANTIZATION'** thực hiện chức năng lượng tử hóa tín hiệu đã được xử lý và xuất ra file .txt bộ nhị phân 8 bit. Sau đó, mở file đã xuất ra màn hình bằng lệnh:

```
open('BinaryCode.txt');
```

4.5.6. Hệ thống đồ thị biểu diễn:

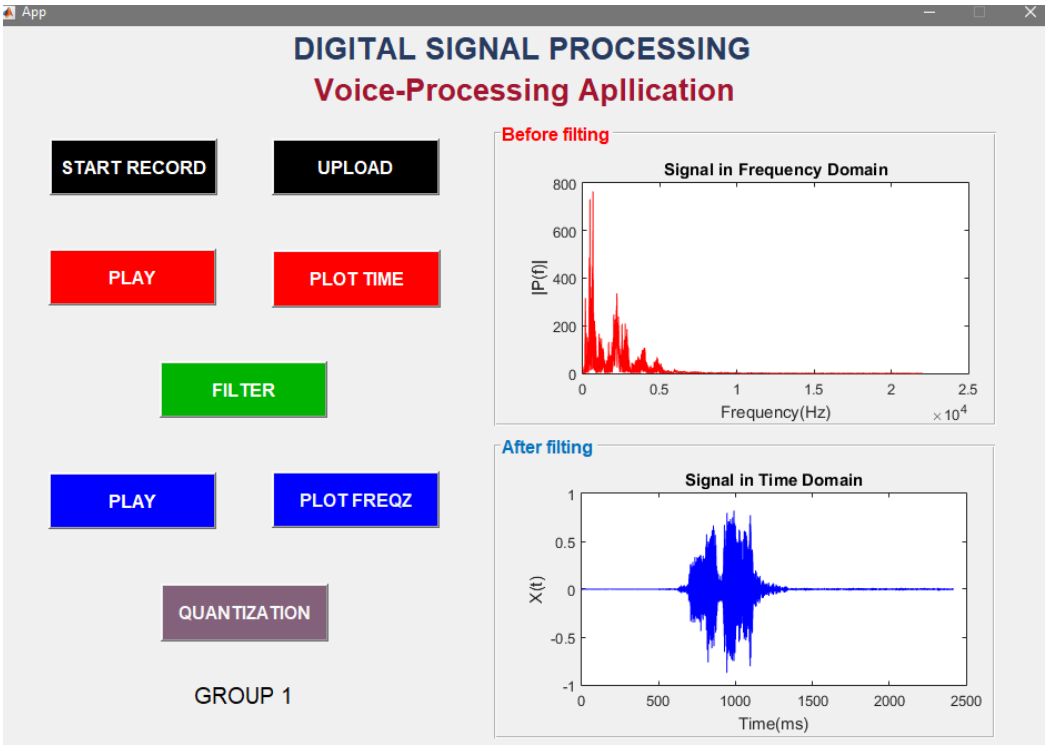
Sử dụng các đối tượng ***Panel*** và ***Axes*** từ GUI Matlab để tạo nên không gian hiển thị các độ thị trước và sau khi lọc nhiễu.



Nhóm hình 4.4 Hình ảnh các đồ thị thực tế được vẽ ra từ ứng dụng

CHƯƠNG V
KẾT QUẢ THỰC HIỆN

- **Kết quả thực hiện:** Hoàn thành được việc xây dựng thành công ứng dụng với các chức năng và mục tiêu mà nhóm đã đề ra từ trước. Hoàn thành được báo cáo tổng kết chi tiết cùng với tập tin trình chiếu báo cáo đúng thời gian.



Hình 5 Cửa sổ ứng dụng sau khi hoàn thành

- **Đánh giá kết quả làm việc nhóm:**
  - Các thành viên nhóm tích cực tham gia hoạt động chung.
  - Chủ động tìm tòi, học hỏi và sắp xếp thời gian vì công việc chung.
  - Biết cách phân công, chia việc hợp lý.
  - Hoàn thành tốt công việc được phân công.

CHƯƠNG VI
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết luận:

Sau thời gian học tập, làm việc nhóm cùng nhau, chúng em rút ra được rất nhiều kinh nghiệm trong việc tự học, tự tìm tòi và hiểu rõ hơn việc ứng dụng các kiến thức cơ bản môn học ***"Xử lý tín hiệu số"*** kết hợp với việc sử dụng tốt các chức năng, thư viện mà phần mềm MATLAB hỗ trợ để tạo nên được một sản phẩm có ứng dụng thiết thực.

Ngoài ra, qua quá trình học tập và làm việc cùng nhau trong quãng thời gian tuy không quá dài, nhưng cũng đã giúp chúng em học cách gắn kết các thành viên trong một nhóm học tập, làm việc sao cho hiểu quả. Trau dồi các kỹ năng như tìm tài liệu, kỹ năng giao tiếp và đọc các tài liệu bằng tiếng Anh, quản lý thời gian, quản lý kế hoạch cũng như việc phấn đấu và cố gắng để biến các mong muốn thành hiện thực.

6.2. Ưu điểm – Hạn chế:

Ưu điểm: ***"Ứng dụng xử lý câu lệnh dạng âm thanh bằng Matlab"*** không chỉ là công cụ hỗ trợ trong việc biến đổi và mã hóa tín hiệu câu lệnh, mà nó còn giúp cho người dùng hiểu thêm về quá trình biến đổi từng giai đoạn, cùng với việc nắm bắt được các chức năng, ứng dụng của lý thuyết vào cụ thể một cách trực quan, sinh động nhất.

Hạn chế:

- Giao diện còn đơn giản, chưa được đẹp mắt.
- Khả năng lọc nhiễu, tạp âm chưa được tối ưu.

6.3. Hướng phát triển:

Sản phẩm ***"Ứng dụng xử lý câu lệnh dạng âm thanh bằng Matlab"*** không chỉ dừng lại ở việc xử lý âm thanh câu lệnh và mã hóa đơn giản. Mà nó còn có cơ hội phát triển sâu hơn trong việc ứng dụng các thuật toán máy học như Machine Learning

(máy học) hoặc Artificial Intelligence (trí tuệ nhân tạo) trong việc tự động so sánh các bộ tín hiệu âm thanh được lưu trữ sẵn trong một cơ sở dữ liệu, sau đó phản hồi lại người dùng bằng việc nhận dạng đúng mệnh lệnh và thi hành đúng chức năng, thay con người làm mọi việc chỉ với việc ra lệnh bằng giọng nói.

TÀI LIỆU THAM KHẢO

[1] ThS Đặng Hoài Bắc, Sách hướng dẫn học tập "XỬ LÝ TÍN HIỆU SỐ".

[2] Nguyễn Quốc Trung, "Xử lý tín hiệu và lọc số".

[3] Quách Tuấn Ngọc, "Xử lý tín hiệu số".

[4] Wikipedia, https://en.wikipedia.org/wiki

[5] MathWorks – Fast Fourier Transform, https://www.mathworks.com/help/Matlab/ref/fft.html?s_tid=srchtitle

[6] MathWorks – Audio Recording, https://www.mathworks.com/help/Matlab/ref/audiorecorder.record.html?

[7] MathWorks – HDL Butterworth Filter, https://www.mathworks.com/help/hdlfilter/examples/hdl-butterworth-filter.html?

[8] MathWorks – Integer to Bit Converter, https://www.mathworks.com/help/comm/ref/integertobitconverter.html?