

# Text-to-Speech based on pyttsx3

廖子涵 12010615 耿沛言 12012941 张旭东 12011923

## 1. Background

Text-to-speech (TTS) is a technology that converts written text into spoken words. It enables computers, devices, and applications to generate human-like speech from textual content, allowing users to listen to the information rather than reading it.

The goal of text-to-speech (TTS) synthesis is to **convert an arbitrary input text into intelligible and natural sounding speech**. TTS is not a “cut-and-paste” approach that strings together isolated words. Instead, TTS employs linguistic analysis to infer correct pronunciation and prosody (i.e., NLP) and acoustic representations of speech to generate waveforms (i.e., DSP). These two areas delineate the two main components of a TTS system: **the front-end**, the part of the system closer to the text input, and **the back-end**, the part of the system that is closer to the speech output.

This technology combines **linguistic analysis, phonetics, speech synthesis models, voice databases, and natural language processing** to convert written text into natural-sounding speech. This technology finds applications in various fields, including assistive technologies, accessibility features, voice assistants, and multimedia content production.

## 2. Implementation

### 2.1 Theoretical introduction

Text to speech consists of two parts, including signal processing and speech synthesis. Speech signal processing includes acoustic feature extraction, phoneme alignment, speech synthesis parameter generation and so on. And speech synthesis is the process of converting text into speech. Its goal is to generate natural, smooth speech output that sounds like real human speech.

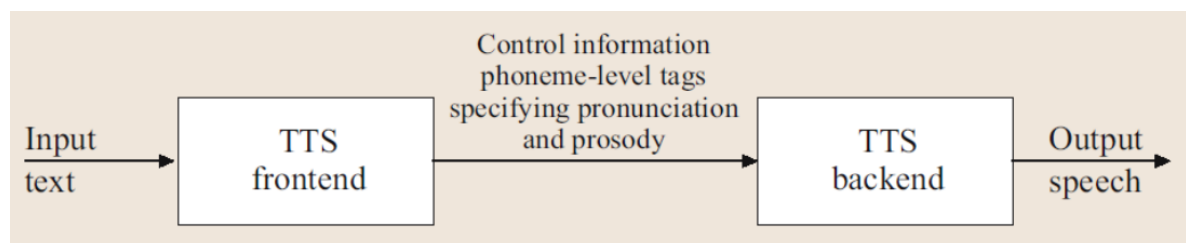


Figure 1. TTS system

#### 2.1.1 Signal processing (TTS front-end)

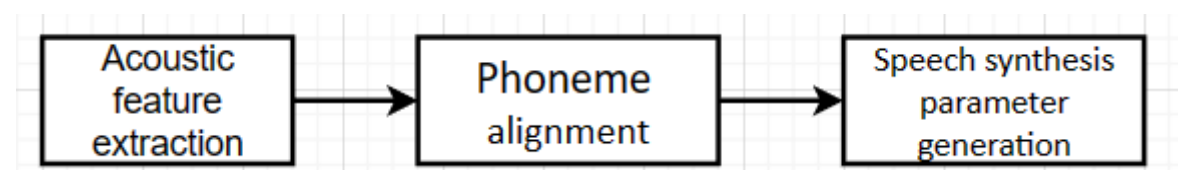


Figure 2. steps of signal processing

The steps of signal processing include acoustic feature extraction, phoneme alignment and speech synthesis parameter generation.

- **Acoustic feature extraction**

Acoustic feature extraction is the process of extracting sound features from speech signals. These features are used to describe the time-domain and frequency-domain characteristics of sound. Common acoustic features include pitch, formant frequencies, and Mel-frequency Cepstral Coefficients (MFCCs). These features play an important role in speech synthesis because they capture key features of speech signals.

- **Phoneme alignment**

Phoneme alignment is the process of aligning input text with phonemes. In speech synthesis, text is usually converted into a sequence of phonemes, each representing a basic unit in speech. The purpose of phoneme alignment is to determine the duration and position of each phoneme in order to accurately control the synthesis of phonemes during the stage of speech synthesis parameter generation.

- **Speech synthesis parameter generation**

In the stage of speech synthesis parameter generation, the parameter sequence for synthesizing speech is generated based on the input text and aligned phonemes. These parameters describe characteristics, such as the length, frequency and intensity of the sound. The generated parameter sequence can include fundamental frequency, formant frequency, volume, tone, and so on. These parameters are typically used by speech synthesis engines to generate a final speech output from the input text.

## 2.1.2 Speech synthesis (TTS back-end)

Speech synthesis can be implemented using rule-based, statistics-based or deep learning-based methods.

- **Rule-based method**

Rule-based speech synthesis methods synthesize speech using a library of predefined speech rules and speech units. These rules are created based on phonetics knowledge and acoustic models. For example, a vocal tract model can describe how sound travels and deforms in the throat, mouth, and nasal cavity. Rule-based approaches require a large number of artificial rules and acoustic models, so there may be limitations in terms of flexibility and naturalness.

- **Statistic-based method**

Statistically based speech synthesis methods use large-scale speech data sets to train models and use these models to generate speech. Common statistical methods include Hidden Markov Models (HMM) and Gaussian Mixture Models (GMM). These models capture the statistical characteristics and conversion rules of speech. For example, in HMM, text is converted into a series of phonemes, and the acoustic characteristics of each phoneme are then predicted by a model to generate a corresponding speech output.

- **deep-learning-based methods**

Deep learning methods have made remarkable progress in speech synthesis. They use deep neural networks to learn the representation and generation patterns of speech. Recurrent Neural Networks (RNNs) and their variants (such as Long Term Memory networks, Long Term memory, and LSTM) are commonly used deep learning architectures for modeling mapping relationships between text and speech. These models are capable of learning complex speech patterns and relationships between phonemes to produce a more natural, fluid speech output.

## 2.2 Common text-to-speech library

There are five common text-to-speech libraries, including `pyttsx3`, `Google Text-to-Speech`, `pyttsx`, `espeak` and `Festival`.

- **pyttsx3**

**Advantage:** Not only is it cross-platform and can run on multiple operating systems, it is also simple to use and provides an intuitive interface. At the same time, it supports the setting of multiple languages and voice properties, and also provides some advanced functions, such as setting voice, obtaining voice list and saving voice as audio files.

**Disadvantage:** Calling it on some operating systems may require additional configuration and dependencies. Also, support for some specific languages and accents may be limited.

- **Google Text-to-Speech**

**Advantage:** Based on the Google Text-to-Speech API, it provides high-quality speech synthesis, supports customization of multiple languages and speech properties, and also allows speech to be saved directly as audio files.

**Disadvantage:** It requires networking to use and relies heavily on the Google Text-to-Speech API. Also, it may be slow to convert when dealing with large amounts of text.

- **pyttsx**

**Advantage:** It is cross-platform, can run on multiple operating systems, and also provides similar interfaces and features to `pyttsx3`.

**Disadvantage:** The library is no longer actively maintained and may not have the latest features and improvements.

- **espeak**

**Advantage:** This library is open source and cross-platform, and can run on multiple operating systems. It supports multiple languages and phoneme libraries, and also provides command-line tools and API interfaces.

**Disadvantage:** It may have limited pronunciation accuracy and naturalness for some languages, and does not guarantee the quality of the text conversion.

- **Festival**

**Advantage:** This library is not only open source and cross-platform and can run on multiple operating systems, but also provides a variety of speech synthesis methods and algorithms to choose from.

**Disadvantage:** It requires additional configuration and setup and is relatively complex to use. Support for some languages and accents may be limited.

Each library has its own advantages and disadvantages. Choosing the right library should be balanced according to specific needs, platform requirements, and speech synthesis quality.

The library used in this project is `pyttsx3`.

### Introduction to pyttsx3

Pyttsx3 is a Python library for text-to-speech conversion that provides a simple yet powerful interface for speech synthesis on multiple platforms. Here are some of the features and functions of `pyttsx3`.

- **Cross-platform support:** Pyttsx3 can run on multiple operating systems, including Windows, Mac, and Linux. This makes it a universal text-to-speech solution.

- **Multilanguage support:** Pytttsx3 supports speech synthesis in multiple languages, including English, Spanish, French, German, Italian, Portuguese, and more. The appropriate language can be selected according to the requirement for speech synthesis.
- **Customizable voice properties:** Pytttsx3 allows to customize the synthesized speech output by setting the properties of the speech. The desired speech effect can be gotten by adjusting the speed, pitch, volume, and other attributes of speech.
- **Event-driven architecture:** Pytttsx3 uses an event-driven architecture that can handle various events of speech synthesis, such as starting synthesis, updating synthesis progress, and completion of synthesis. These events can be listened by registering event handlers and take appropriate actions as needed.
- **Multi-engine support:** Pytttsx3 supports multiple speech synthesis engines. It uses either `SAPI5` or `NSSpeechSynthesizer` by default, but can also choose other available engines such as `espeak`, Microsoft Speech Platform, or other third-party engines on demand.
- **Advanced function:** Pytttsx3 provides a number of advanced features to enhance the flexibility and functionality of speech synthesis. This includes getting a list of available voices, setting up specific voices, saving the voice as an audio file, and more.

## 2.3 Implementation detail

To implement Pytttsx3 into TTS tasks, several steps are necessary to ensure the implementation functions well.

1. Initialization: Before the conversion process starts, initialize the speech engine with `pytttsx3.init()`, which creates an instance of the speech engine that you can use to set properties and perform actions. This function could determine `pytttsx3.driver` according to the current platform, including:
  - sapi5 - SAPI5 on Windows
  - nsss - NSSpeechSynthesizer on Mac OS X
  - espeak - eSpeak on every other platform
2. Setting Properties: `pytttsx3` allows various properties input, like the speech rate, volume, and voice. In this project, we apply `setProperty()` function to adjust the input characteristics. For example, `engine.setProperty('rate', 125)` sets the speech rate to 125 words per minute. And note that the instance we created in step 1 could be reset by repeatedly call the `setProperty()` function, and use `getProperty()` to retrieve current settings.
3. Voice Metadata: class `pytttsx3.voice.voice` contains information about a speech synthesizer voice, including:
  - Age: integer age of the voice in years.
  - Gender: string gender of the voice: male, female, or neutral
  - Id: string identifier of the voice. Used to set the active voice via `pytttsx3.engine.Engine.setPropertyValue()`.

The voice object would be returned when attempting to get the property of the engine.

4. Conversion: Text is converted to speech using the `say()` function. The text you want to convert is passed as an argument to this function. Note that the `say()` function is non-blocking, which means the function returns immediately and the speaking continues in the background.
5. Stopping the Speech: The speech can be stopped using `stop()` function. This function stops the current and all pending speech commands.

## Model detail (python implementation)

- Encoder (Melencoder):

```
def get_torch_mel_spectrogram_class(self, audio_config):
    return torch.nn.Sequential(
        PreEmphasis(audio_config["preemphasis"]),
        # TorchSTFT(
        #     n_fft=audio_config["fft_size"],
        #     hop_length=audio_config["hop_length"],
        #     win_length=audio_config["win_length"],
        #     sample_rate=audio_config["sample_rate"],
        #     window="hamming_window",
        #     mel_fmin=0.0,
        #     mel_fmax=None,
        #     use_htk=True,
        #     do_amp_to_db=False,
        #     n_mels=audio_config["num_mels"],
        #     power=2.0,
        #     use_mel=True,
        #     mel_norm=None,
        # )
        torchaudio.transforms.MelSpectrogram(
            sample_rate=audio_config["sample_rate"],
            n_fft=audio_config["fft_size"],
            win_length=audio_config["win_length"],
            hop_length=audio_config["hop_length"],
            window_fn=torch.hamming_window,
            n_mels=audio_config["num_mels"],
        ),
    )
```

- Vocoder (MelResNet):

```
if self.args.use_upsample_net:
    assert (
        np.cumproduct(self.args.upsample_factors)[-1] ==
        config.audio.hop_length
    ), "[!] upsample scales needs to be equal to hop_length"
    self.upsample = UpsampleNetwork(
        self.args.feats_dims,
        self.args.upsample_factors,
        self.args.compute_dims,
        self.args.num_res_blocks,
        self.args.res_out_dims,
        self.args.pad,
        self.args.use_aux_net,
    )
else:
    self.upsample = Upsample(
        config.audio.hop_length,
        self.args.pad,
        self.args.num_res_blocks,
        self.args.feats_dims,
        self.args.compute_dims,
        self.args.res_out_dims,
        self.args.use_aux_net,
```

```

    )
    if self.args.use_aux_net:
        self.I = nn.Linear(self.args.feat_dims + self.aux_dims + 1,
self.args.rnn_dims)
        self.rnn1 = nn.GRU(self.args.rnn_dims, self.args.rnn_dims,
batch_first=True)
        self.rnn2 = nn.GRU(self.args.rnn_dims + self.aux_dims,
self.args.rnn_dims, batch_first=True)
        self.fc1 = nn.Linear(self.args.rnn_dims + self.aux_dims,
self.args.fc_dims)
        self.fc2 = nn.Linear(self.args.fc_dims + self.aux_dims,
self.args.fc_dims)
        self.fc3 = nn.Linear(self.args.fc_dims, self.n_classes)
    else:
        self.I = nn.Linear(self.args.feat_dims + 1, self.args.rnn_dims)
        self.rnn1 = nn.GRU(self.args.rnn_dims, self.args.rnn_dims,
batch_first=True)
        self.rnn2 = nn.GRU(self.args.rnn_dims, self.args.rnn_dims,
batch_first=True)
        self.fc1 = nn.Linear(self.args.rnn_dims, self.args.fc_dims)
        self.fc2 = nn.Linear(self.args.fc_dims, self.args.fc_dims)
        self.fc3 = nn.Linear(self.args.fc_dims, self.n_classes)

```

- Test code:

```

import pyttsx3
engine = pyttsx3.init() # object creation

""" RATE"""
rate = engine.getProperty('rate') # getting details of current speaking rate
print (rate) #printing current voice rate
engine.setProperty('rate', 125) # setting up new voice rate

"""VOLUME"""
volume = engine.getProperty('volume') #getting to know current volume level
(min=0 and max=1)
print (volume) #printing current volume level
engine.setProperty('volume',1.0) # setting up volume level between 0 and 1

"""VOICE"""
voices = engine.getProperty('voices') #getting details of current voice
#engine.setProperty('voice', voices[0].id) #changing index, changes voices. 0
for male
engine.setProperty('voice', voices[1].id) #changing index, changes voices. 1
for female



engine.say("Hello world!")
engine.say('My current speaking rate is ' + str(rate))
engine.runAndWait()
engine.stop()

"""Saving Voice to a file"""
engine.save_to_file('Hello world', 'test.mp3')
engine.runAndWait()

```

## 2.4 Results and analysis

**Input text:** "Hello, this is the test speech for speech signal processing final project."

**Output speech signal:**  male.wav  female.wav

### 2.4.1 pitch analysis

We use praat to detect the pitch of the two speech signal, in order to verify the accuracy of TTS model with different gender setting.

- male.wav:

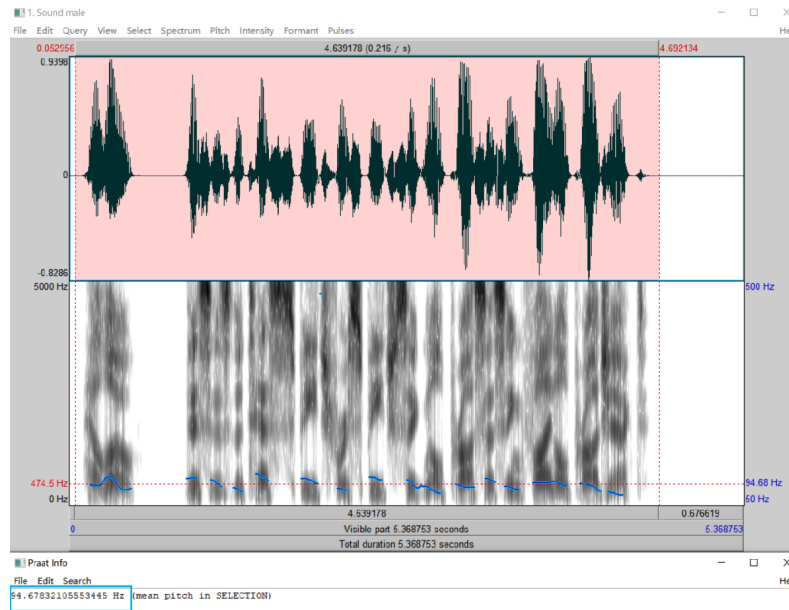


Figure 3. waveform, spectrum and pitch of male speech signal

average pitch = 95 Hz

- female.wav:

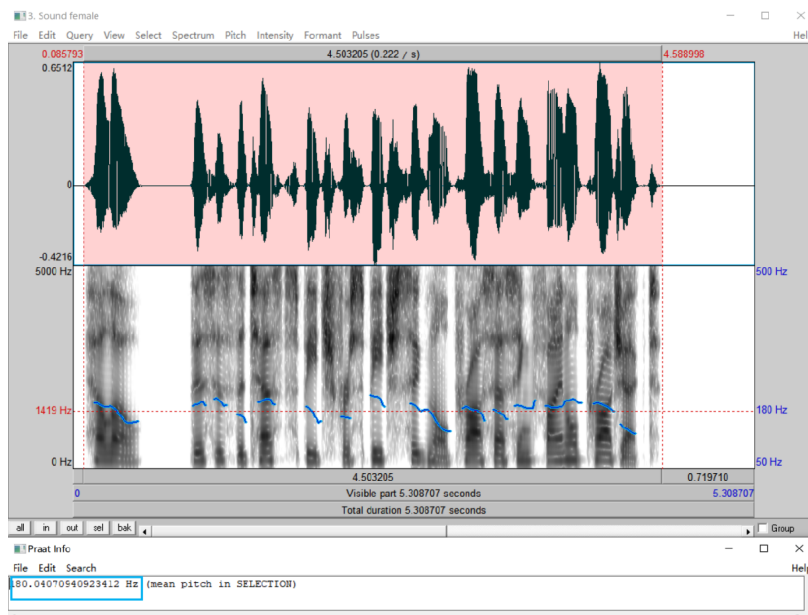


Figure 4. waveform, spectrum and pitch of female speech signal

average pitch = 180 Hz

From the detected pitch results, it shows that the pitch of male is around 95 Hz and that of female is around 180 Hz. This is consistent with the our setting that they are from different gender.

## 2.4.2 vocal tract analysis

we use LPC analysis (autocorrelation-based method) to analyze the frequency response of the two vocal tracts:

$$H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1 - \sum_{k=1}^p a_k z^{-k}} = \frac{1}{A(z)}$$

Figure 5. transfer function of LPC model

**MATLAB code:**

```
[speech,fs]=audioread('male.wav');
frame=speech(6000:7000-1);
%% LPC spectrum
[a,err]=LPC_frame(frame,12); #LPC coefficients (order = 12)
A=fft([1,-a],length(frame));
H=1./A;
LPC_mag=20*log10(abs(H));
```

LPC coefficient computation:

$$\sum_{k=1}^p \alpha_k \phi_{\hat{n}}(i,k) = \phi_{\hat{n}}(i,0), \quad 1 \leq i \leq p$$

$$\sum_{k=1}^p \alpha_k R_{\hat{n}}(|i-k|) = R_{\hat{n}}(i), \quad 1 \leq i \leq p$$

$$\begin{bmatrix} R_{\hat{n}}(0) & R_{\hat{n}}(1) & \dots & R_{\hat{n}}(p-1) \\ R_{\hat{n}}(1) & R_{\hat{n}}(0) & \dots & R_{\hat{n}}(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_{\hat{n}}(p-1) & R_{\hat{n}}(p-2) & \dots & R_{\hat{n}}(0) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} R_{\hat{n}}(1) \\ R_{\hat{n}}(2) \\ \vdots \\ R_{\hat{n}}(p) \end{bmatrix}$$

$$\Re \alpha = r \text{ or } \alpha = \Re^{-1} r$$

Figure 6. computation formula of LPC coefficients

```
function [a,err] = LPC_frame(x,order)
%% hamming window
win = hamming(length(x));
s = x.*win;
fft_size = length(s);
X = fft(s,fft_size);
R = real(ifft((abs(X)).^2));
a = levinson(R,order);
a = -a(2:end);
s_hat = filter([0 a],1,s);
err = s - s_hat;
```



## frequency response:

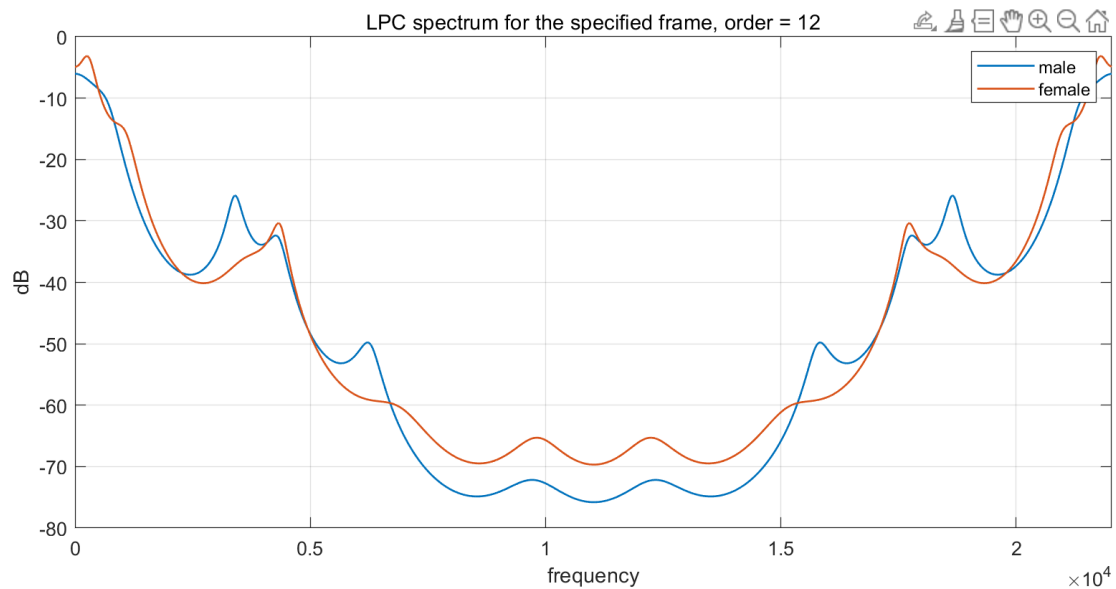


Figure 7. frequencnt response of LPC model (vocal tract)

The figure 7 above shows that the two speech signal has different vocal tracts, which is rational since they are from different person.

### 2.4.3 formant analysis

Again, we use praat to analyze the formants (F1, F2, F3, F4) of the two generated signals:

male.wav:

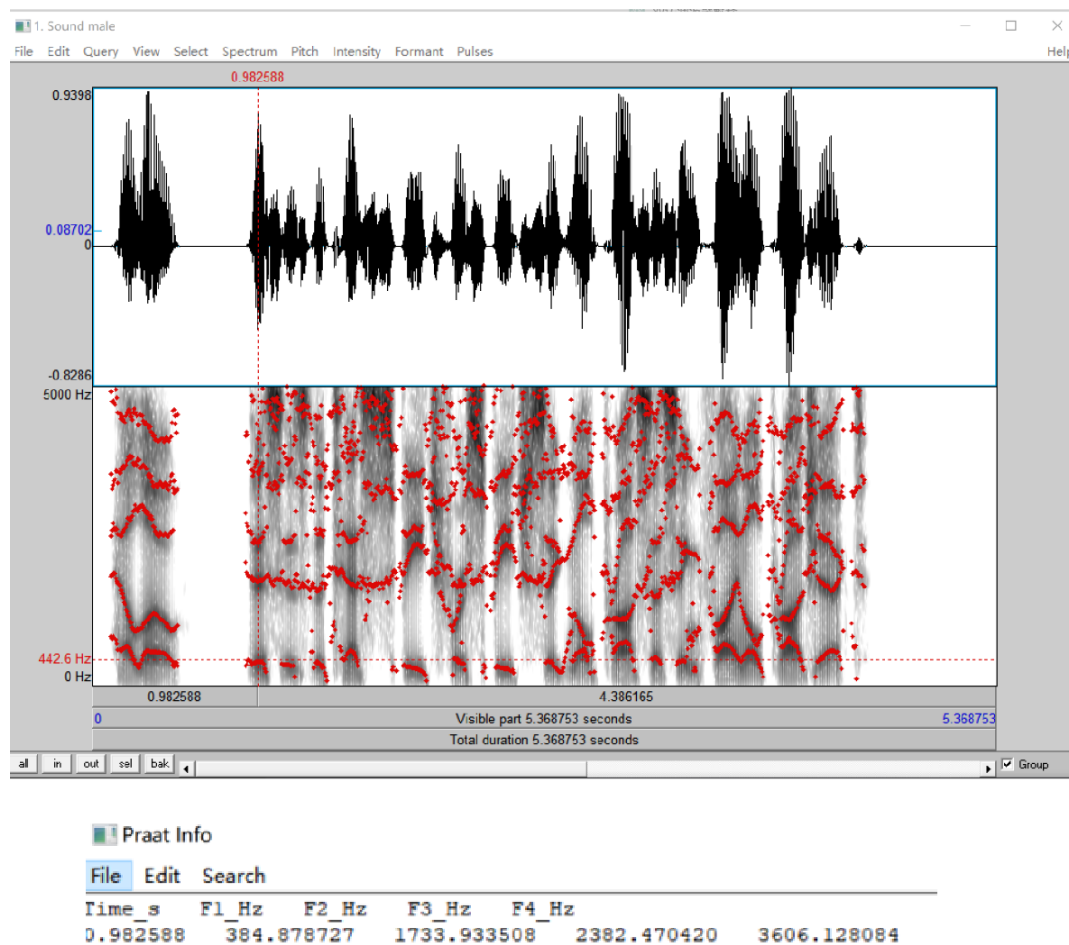


Figure 8. formants of male speech signal

female.wav:

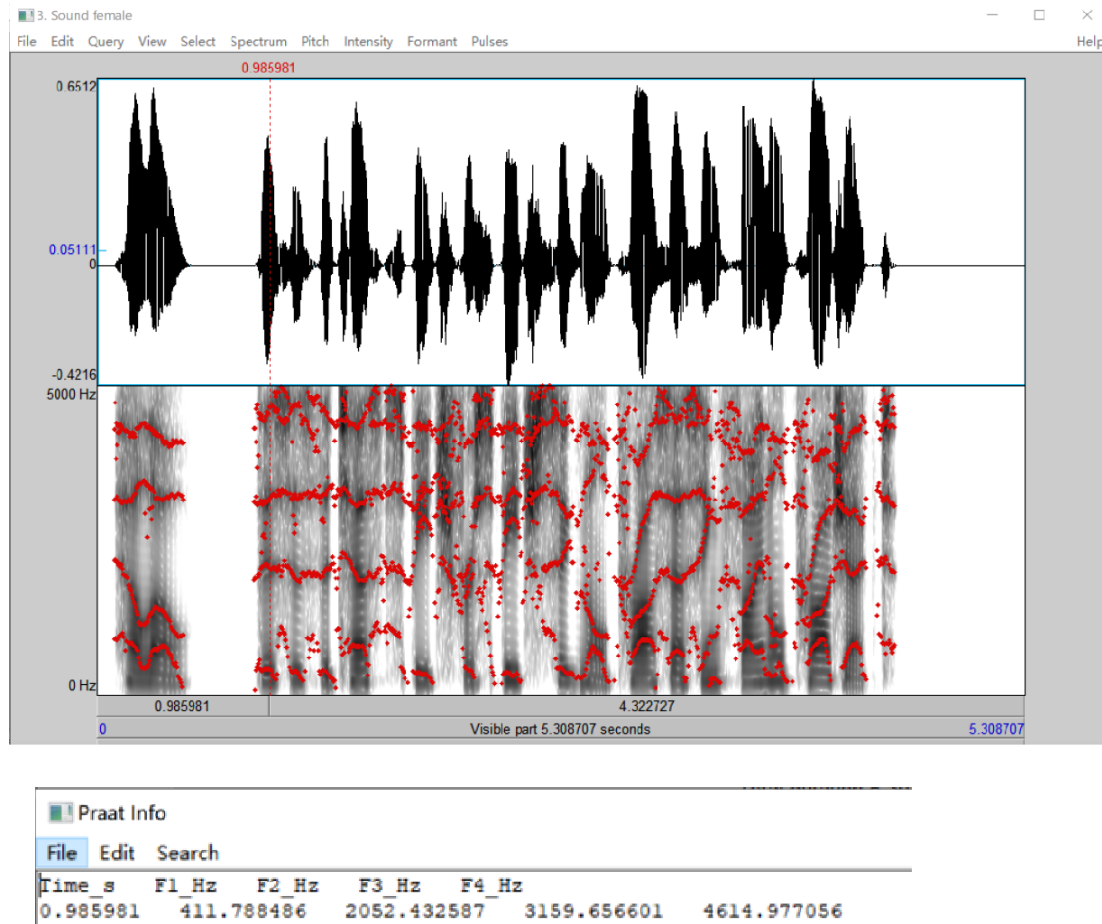


Figure 9. formants of female speech signal

It can be observed from figure 8 and 9 that, the two signals have different formants, thus articulating sound with different tone.

## 3. Future improvement and new application

### 3.1 Improvements

When text-to-speech is performed, the following can help improve the quality and naturalness of speech synthesis.

#### 1. Control speed and pitch:

By adjusting speed and pitch, the speech sound more natural. Appropriate speech speed and pitch can increase the intelligibility and fluency of speech. The best combination of speed and pitch for a particular application can be found by experimenting with different settings.

#### 2. Expression of emphasis and tone:

Where appropriate, emphasize specific words or express specific emotions by using different tones, volumes, and tones. This can make the speech sound more vivid and expressive.

#### 3. The accuracy and coherence of phonemes:

Ensure that the speech synthesis engine can accurately recognize and synthesize different phonemes. The accuracy of phonemes is very important for correct pronunciation and intelligibility of speech. In addition, make sure that transitions and connections between phonemes are smooth and natural to avoid intermittent or disconnected speech output.

#### **4. Focus on specific languages and accents:**

If an application scenario involves a specific language or accent, it is advised to select a speech synthesis engine that supports the language or accent. Different languages and accents have their own unique pronunciation rules and acoustic characteristics, and choosing the right engine can provide more accurate and natural speech output.

#### **5. Introducing the latest technology in the field of speech synthesis:**

Keep abreast of the latest technology and research advances in speech synthesis, such as the application of deep learning and neural networks, as well as methods for training using large data sets. These technologies can provide higher quality, more natural speech synthesis effects.

#### **6. Feedback and evaluation of user:**

Through feedback and evaluation of user, the performance of the speech synthesis system in different application scenarios can be understood. User's feedback can help identify problems, improve the system, and meet user's requirements.

Different application scenarios may have different requirements for speech synthesis. Therefore, according to the specific requirements, the appropriate parameters are set and adjusted to achieve the best text-to-speech experience.

### **3.2 New applications**

In the future, text-to-speech technology has a wide range of application prospects. Here are some possible future applications.

#### **1. Personalized assistants and avatars:**

Text-to-speech gives voice and voice interaction to personalized assistants and avatars. This makes the assistants and characters more vivid and intimate, providing a more natural conversational experience. For example, avatars in smart speakers, smartphones, and virtual reality applications can communicate and interact with users through text-to-speech technology.

#### **2. Voice-assisted technology:**

Text-to-speech plays an important role in assisting the visually impaired and the speech impaired. Future applications could help these populations better interact with their digital and physical environments by converting text to speech, providing real-time screen reading, voice prompts, and voice navigation.

#### **3. Multilingual and interlingual communication:**

Text-to-speech technology has potential in multilingual and cross-lingual communication. It can convert text to speech in real time and provide speech translation capabilities that support different languages. This can promote communication and understanding between different languages and broaden communication channels in the era of globalization.

#### **4. Media and entertainment experiences:**

Text-to-speech can improve the media and entertainment experience. For example, in the realm of e-book readers and audiobooks, text-to-speech can convert book content into audio playback. In addition, in games and virtual reality, text-to-speech technology can empower characters and virtual worlds with voice and interaction capabilities, providing a more immersive and immersive experience.

#### **5. Customize broadcast and media content:**

Text-to-speech can be used for customized broadcast and media content generation. By converting text to speech, users can be provided with personalized content such as news, podcasts, audio books, and radio shows.

With the continuous development and innovation of technology, text-to-speech technology will further expand the application field and provide more personalized, immersive and convenient voice interaction experience.

## Reference

---

- [1] Klatt, D. H. (1987). Review of text-to-speech conversion for English. *The Journal of the Acoustical Society of America*, 82(3), 737-793.
- [2] Ren, Y., Ruan, Y., Tan, X., Qin, T., Zhao, S., Zhao, Z., & Liu, T. Y. (2019). FastSpeech: Fast, robust and controllable text to speech. *Advances in neural information processing systems*, 32.
- [3] Zen, H., Dang, V., Clark, R., Zhang, Y., Weiss, R. J., Jia, Y., ... & Wu, Y. (2019). Libritts: A corpus derived from librispeech for text-to-speech. *arXiv preprint arXiv:1904.02882*.
- [4] Arik, S. Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., ... & Shoeybi, M. (2017, July). Deep voice: Real-time neural text-to-speech. In *International conference on machine learning* (pp. 195-204). PMLR.
- [5] King, S. (2014). Measuring a decade of progress in text-to-speech. *Loquens*, 1(1), e006-e006.