



PENTATrainer2

User Manual

Version 1.1

University College London, UK
King Mongkut's University of Technology Thonburi, Thailand

8 July 2016

Santitham Prom-on
Yi Xu

Contents

CHAPTER 1 INTRODUCTION	3
1.1 FEATURES	3
1.2 SYSTEM REQUIREMENTS	4
1.3 INSTALLATION	4
1.4 LIMITATIONS	4
CHAPTER 2 DATA PREPARATION AND ANNOTATION	6
2.1 UNDERSTANDING THE PENTA FRAMEWORK	6
2.2 PREPARING SPEECH DATASET	7
2.3 ANNOTATION	9
2.4 AUXILIARY FILES IN ANNOTATION PROCESS	17
CHAPTER 3 LEARNING PARAMETERS	20
3.1 LEARNING PARAMETERS OF COMMUNICATIVE FUNCTIONS	20
3.2 UNDERSTANDING MODEL PARAMETERS	26
3.3 STOCHASTIC OPTIMIZATION	28
3.4 FINE TUNING OPTIMIZATION PROCESS	30
3.5 OUTPUT FILES	31
CHAPTER 4 SYNTHESIS	33
4.1 REQUIRED INPUT FILES	33
4.2 SYNTHESIS WITH LEARNED PARAMETERS	33
4.3 SYNTHESIS WITH MODIFIED PARAMETERS FOR PERCEPTION EXPERIMENT	35
REFERENCE	37

CHAPTER 1

Introduction

This manual describes PENTATrainer2 (*pen-ta-train-two*), a generalized semi-automatic tool for modeling speech prosody based on communicative functions and articulatory dynamics. Based on the Parallel Encoding and Target Approximation (PENTA) framework (Xu, 2005), PENTATrainer is a program that integrates a Praat script and a Java program to achieve computational modeling of any language. It encapsulates the quantitative Target Approximation (qTA) model (Prom-on et al., 2009), which represents the dynamic F_0 control, the encoding schemes, which realizes multiple how communicative functions in parallel, and simulated annealing, a stochastic learning algorithm (Kirkpatrick et al., 1983) that extracts the functional parameters through global optimization. The extracted functional parameters are then used in PENTATrainer2 to synthesize continuous F_0 contours that can be directly compared to those of natural speech.

1.1 Features

Quantitative Modeling

PENTATrainer allows users to quantitatively model speech prosody based on the PENTA framework. Such a quantitative representation allows user to objectively test postulated hypotheses or theories.

Articulatory-Functional Approach

The PENTA framework bridges the gap between the physical layer and the information layer by representing F_0 as a response of the articulatory process to multiple communicative functions which are parallel to each other.

Flexible Annotation System

PENTATrainer implements the parallel encoding schemes by allowing users to create tiers of factors. This enables users to flexibly create annotation labels.

Accurate Optimization

PENTATrainer implements the simulated annealing optimization for parameter learning. The optimized parameters are thus ensured to be close to the globally optimum.

1.2 System requirements

- Operating Systems: Mac OS X, Windows, or Linux
- Praat version 5.3 or newer
- Java Runtime Environment version 1.6 or newer
- (Optional) Audio Editing Program (e.g. Audacity)

1.3 Installation

Users only need to copy the Praat scripts and Java programs to the designated working directory. The working directory contains sound and annotation files and other auxiliary files generated by the scripts. The Praat scripts can then be run using Praat.

1.4 Limitations

- PENTATrainer models the F_0 variation based on the core articulatory process of target approximation. The current version of the program does not yet model prosodic effects that result from additional articulatory factors, such as consonantal perturbation, anticipatory dissimilation, and post-low bouncing. The modeling of these factors will be added in future versions.
- Each annotation layers may have different number of intervals. A layer with the smallest interval size will have the largest number of intervals. Boundaries of other annotation layers must be aligned to the boundaries of the layer with the shortest intervals.

- The simulated annealing algorithm requires a large number of iterations for the solution to be converged and stabilized. The simulation time is thus directly proportional to the size of the corpus and the number of annotation layers.

CHAPTER 2

Data Preparation and Annotation

This chapter provides a step-by-step guide to understanding the preparation and annotation process in PENTATrainer2. Users unfamiliar with the PENTA framework are advised to first read section 2.1 before proceeding to the step-by-step tutorial.

2.1 Understanding the PENTA framework

The basic concepts of the PENTA framework are based on the assumption that speech conveys communicative information and is produced by an articulatory system. In PENTA as shown in Figure 2.1, the production of speech prosody is described as having two main systems: (1) information encoding system and (2) articulatory system.

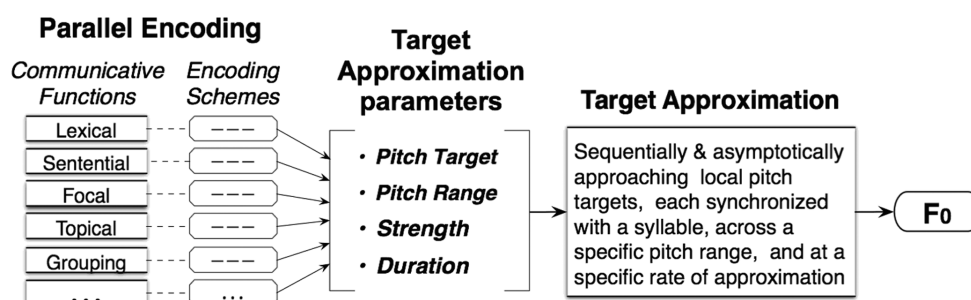


Figure 2.1 PENTA framework

In PENTA, a communicative function represents the intended communicative meanings, which can be, e.g., lexical, focal, sentential, phrasal, emotional, etc. Different communicative function must have distinct encoding schemes which may span across different temporal intervals and may have different function-internal interval divisions. For example, lexical tones may each span a syllable while lexical focus may require at least two temporal intervals consisting of one or more

syllables. With the different encoding schemes, communicative functions are transmitted in parallel in the sense that they each add to the specification of the target in a unique way. The combined effects are then implemented through a purely sequential target approximation process, resulting in the dynamic changes in surface F_0 values and timings of various F_0 events such as peak, valleys, plateaus and their elbows.

The target approximation process in the PENTA framework simulates the behavior of the articulatory system in producing the acoustic features that convey the intended communicative meanings. We can simplify the notion of the target approximation as a goal-oriented model of F_0 movements. Figure 2.2 illustrates the realization of F_0 contours in a tone language as a response of the target approximation process. For each syllable, there is a pitch target that underlies the changes in F_0 dynamic state. F_0 asymptotically approaches the target and, at the end of the syllable, its momentum was carried on to the next syllable as a carryover effect. In this framework, F_0 is therefore not a direct representation of prosodic meaning but an observable output resulting from an articulatory implementation of multiple communicative functions.

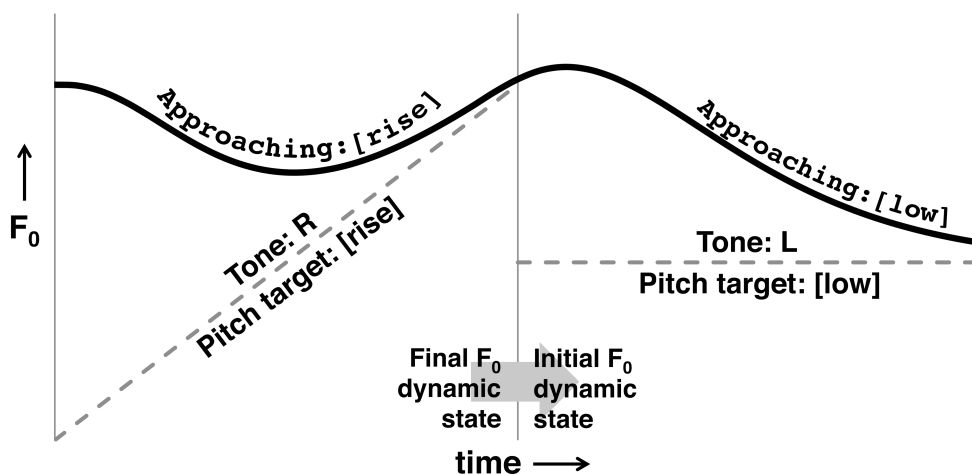


Figure 2.2 Target Approximation Process

2.2 Preparing speech dataset

1. If the speech data consist of a number of utterances in a long sound file, it is best to separate each utterance as an individual file using some type of audio editing software. This step minimizes the

memory usage of the learning program and simplifies the annotation step. All sound files must be saved in “.wav” format.

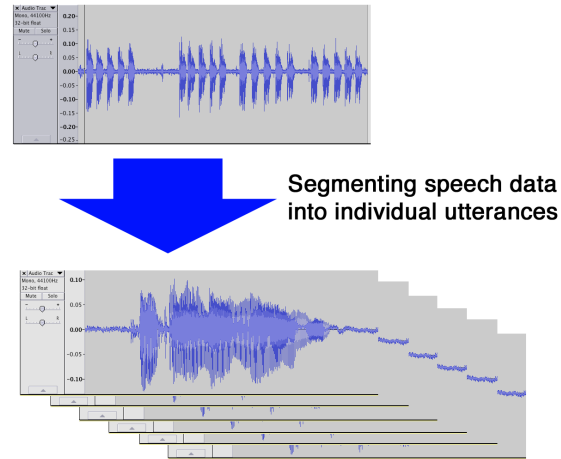


Figure 2.3 Segmenting speech data

2. Copy speech data that will be used in the learning step to a separate folder.

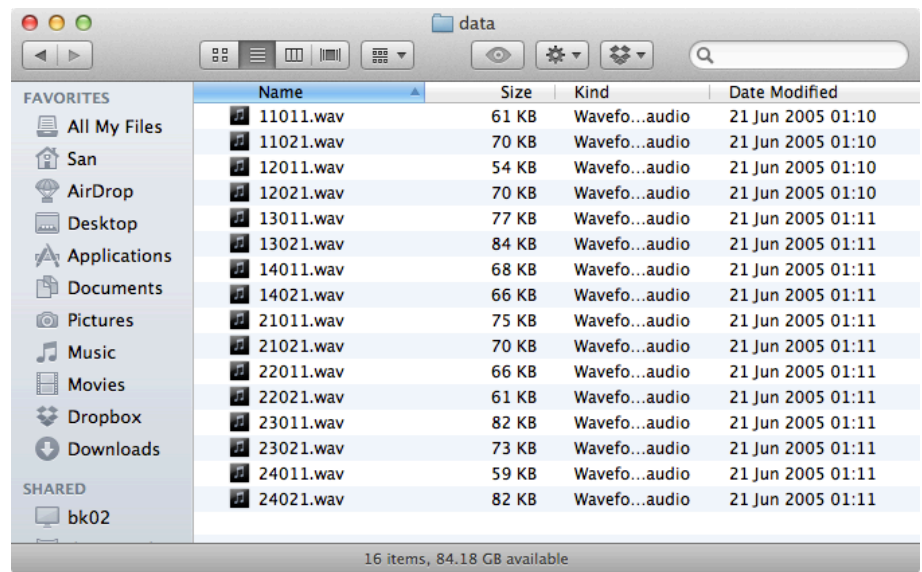


Figure 2.4 Speech data folder

3. Copy PENTATrainer2.praat and the Java .jar programs into the working folder. These files are designed to perform four different tasks: importing existing annotation into PENTATrainer2, annotating a speech dataset, learning target parameters, and synthesizing F_0 contours based on the learned parameters (_synthesize.praat &

Synthesize.jar) and inspecting their goodness of fit to natural contours.

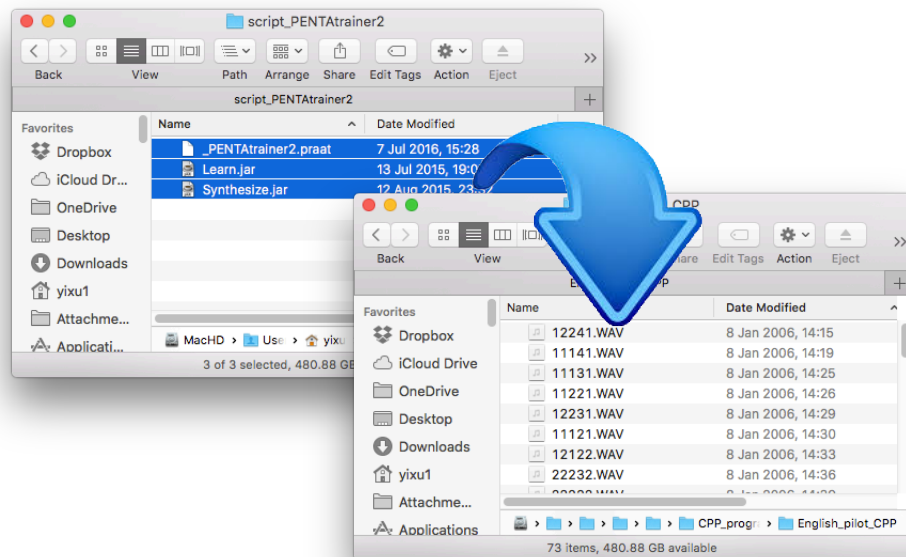


Figure 2.5 Copying PENTATrainer2 to the working folder

4. (Optional) If users already have existing pulse marking files (.pulse) or annotation files (.annotation & .annotation_short) generated from _annotation.praat, make sure that they are in the same working folder.

2.3 Annotation

The Praat annotation script can be used to either generate new annotations or edit existing ones. Once completed, the script will generate a number of auxiliary files for each sound file. Users can refer to section 2.4 for more detail on each auxiliary file.

Users have multiple options to use _annotate.praat. Users can create new annotation files, edit the content of existing annotations, add a layer to the annotation, or remove a layer from the annotation. This option can be selected from the “Task:” drop down menu. The following paragraphs provide step-by-step guides for performing each annotation task.

Create new annotation files

In PENTATrainer2, “Edit annotation” is the default task. The other tasks can be selected through the drop down menu. While performing annotation, users can also edit the F_0 analysis parameters used in the F_0 extraction process by checking the “Edit F_0 analysis option” box. For accurate F_0 estimation, pulse marks can be manually inspected by checking the “Inspect pulse” box. The accuracy of estimated F_0 depends on whether the pulses were marked correctly.

1. To create new annotation files, select “Create new annotation file”. Also, check both the “Edit F_0 analysis option” and “Inspect pulse” boxes. Click Start to continue.

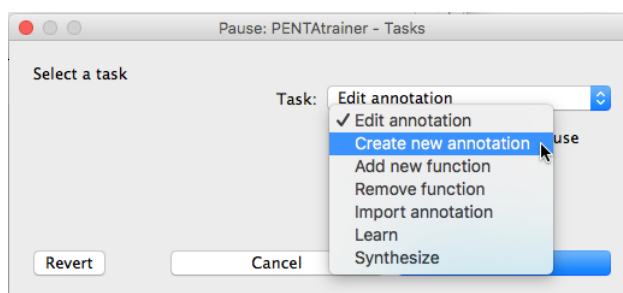


Figure 2.6 Create new annotation.

2. If the “Edit F_0 analysis option” box is checked, PENTATrainer will show the “Pause: Edit F_0 analysis option” window. In this window, users can fine-tune the F_0 estimation process. Left and right F_0 ranges indicate the possible minimum and maximum F_0 values that will be searched. Number of point per interval specifies the resulting number of F_0 points in each annotated interval in the .timenormf0 files (which is useful only for optional graphic displays and has no impact on parameter learning). F_0 sample rate indicates the sampling rate of F_0 estimation. “Perturbation length” & “Final offset” and the check box “Set initial time for normf0 to 0” are used only for generating F_0 measurements and so do not need to be changed.

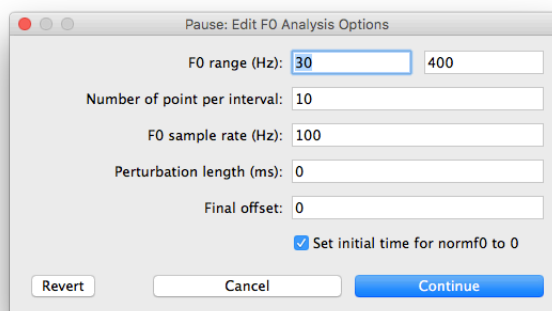


Figure 2.7 F0 analysis options.

3. In the window "Pause: Create new annotation", users can specify the functions to be modeled in the "List of communicative functions" text box. The text there is space-delimited, so that each spaceless name in the list will be used as the name of a layer in the annotation.

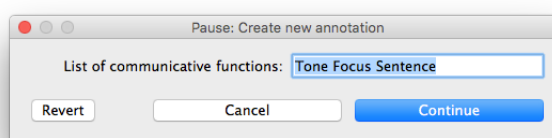


Figure 2.8 Create new annotations

4. For each .wav file, two windows will prompt for user input. The first window in Figure 2.9 (a) shows the annotation window in which users need to put in boundaries of each prosodic event. The second window in Figure 2.9 (b) shows the pulse marking window in which users can manually inspect and change the pulse mark locations. Note that the pulse marking window will only be shown if the "Inspect pulse" box in the "Annotation - Option" is checked,

The annotation window shows three main panels: (1) speech signal (waveform), (2) spectrogram and pitch contour (estimated by Praat but not used by PENTATrainer), and (3) functional encoding layers. Panels (1) and (2) help users put in the boundaries of prosodic events in each layer in panel (3). Users can insert boundaries by using the "Boundary" menu on the top of the window or using the shortcuts as listed below:

- Add on all tiers (Mac: ⌘F9)
- Add on selected tier (Mac: ⌘↵)
- Add on tier 1 (Mac: ⌘F1) and so on for tier 2, 3, ...
- Remove (Mac: ⌘⌘)

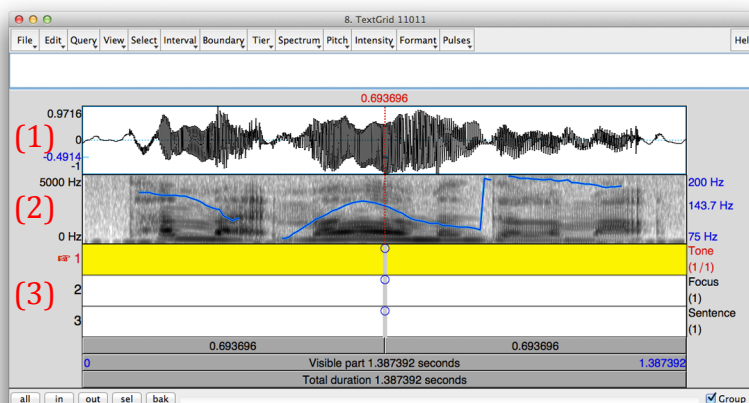


Figure 2.9 (a) The annotation window.

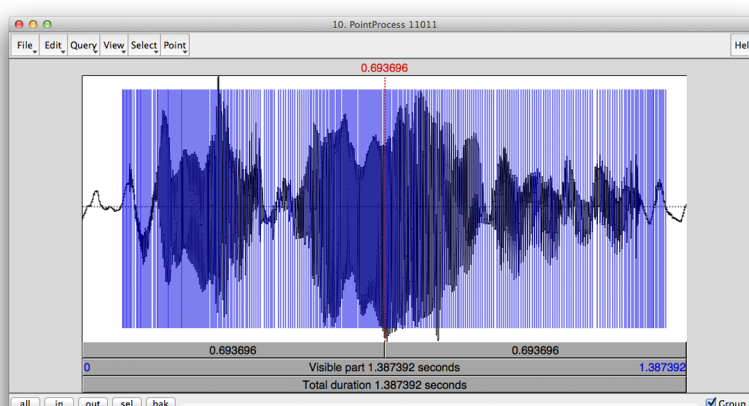


Figure 2.9 (b) The pulse marking window.

The pulse marking window contains only one panel which displays the speech signal in time domain (waveform). The vertical blue lines are pulse marks which specify the locations where the signal appears to repeat itself. The distance between each pulse mark thus indicates the pitch period which is inversely proportional to the fundamental frequency. The pitch estimation algorithm in PENTATrainer uses this pulse mark information to calculate F_0 rather than using Praat's pitch tracks. The pulse marking window allows users to rectify the pulse mark in cases where the initial markings are incorrect. The pulse mark data are stored in the pulse marking

files (.pulse). If users prefer to use existing pulse marking data, they need to copy the pulse marking files to the working folder before running the annotation script.

It should be noted that in this version of PENTATrainer, boundaries in all layers must be synchronous with the boundaries in the layer with the highest number of intervals.

5. Add boundaries to each layer in the annotation window. Users should add boundaries to the layer with the highest number of interval first, as shown in Figure 2.10, because boundaries in other layers must be synchronous with it.

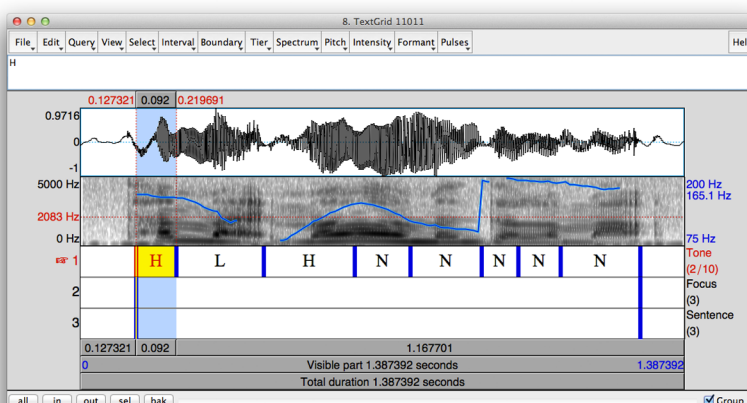


Figure 2.10 Annotate the layer with higher number of intervals first.

Users can then use the boundaries in the marked layer as shown in Figure 2.10 as the guide to other layers.

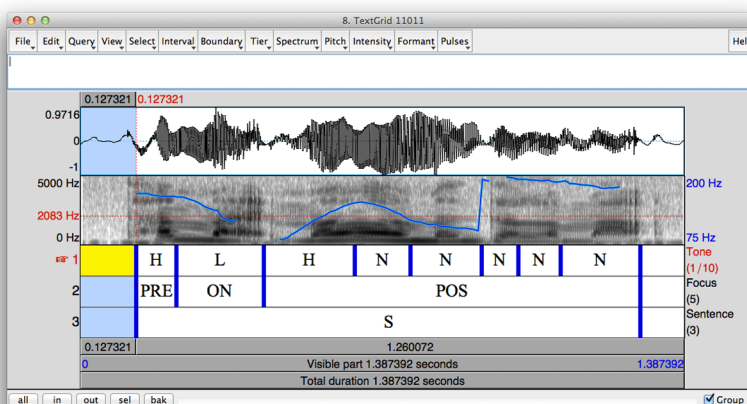


Figure 2.11 Annotated data

6. (optional) Users can rectify the pulse marks in the pulse marking window using the following keyboard shortcuts:
- Add point at cursor (Mac: ⌘P)
 - Remove point(s) (Mac: ⌘P)

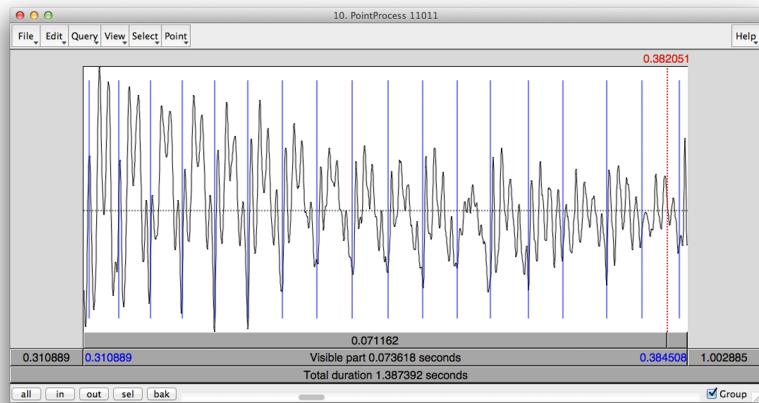


Figure 2.12 Pulse marks can be rectified to guarantee accurate F_0 estimation.

7. After finishing annotation and rectification of pulse marks, click “Continue” on the “Pause: stop or continue” control window to save data and continue to the next sound file.

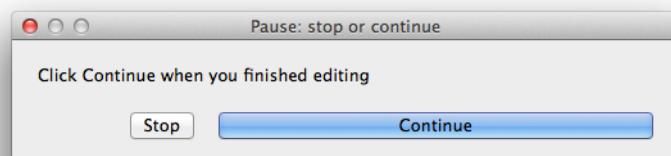


Figure 2.13 Finishing the annotation of one file.

8. Repeat steps 5-8 until all the sound files are annotated.
9. Users can stop the annotation task at anytime by closing all windows. To resume, only steps 1-4 need to be repeated. The program will show a prompt asking whether users want to resume the current annotation. Click “Yes” to resume the incomplete annotation task or click “No” to start the annotation from the first sound file in the working folder.

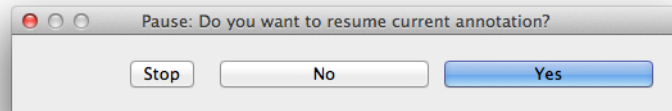


Figure 2.14 resuming the incomplete annotation.

Edit existing annotation

1. To edit or view existing annotation, select “Edit annotation”, which is the default task. Click Start to continue.

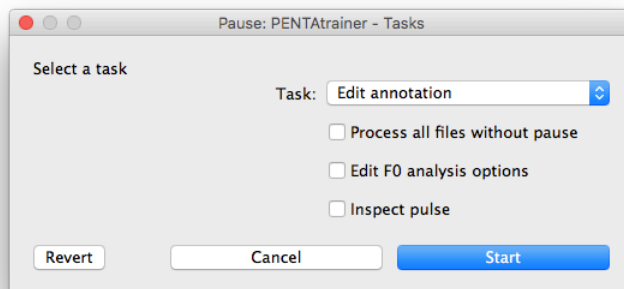


Figure 2.15 Select “Edit existing annotation”.

2. The program will prompt with the window “Pause: Edit existing annotation” to ask for the starting file number. Users can find the number of the input file from the row number in the generated “FileList.txt” (Tip: users can use a spreadsheet program to open “FileList.txt” to find out the input file number”). Using number 1 will make the program go through every annotation files. Click finish to continue.

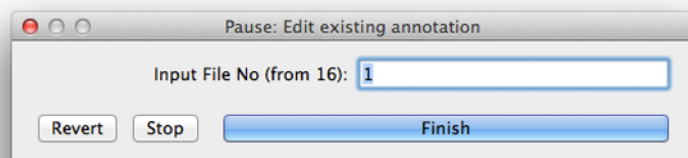


Figure 2.16 Fill in the Input File No.

3. The program will subsequently display the annotation window that already has annotation data from the existing annotation files.

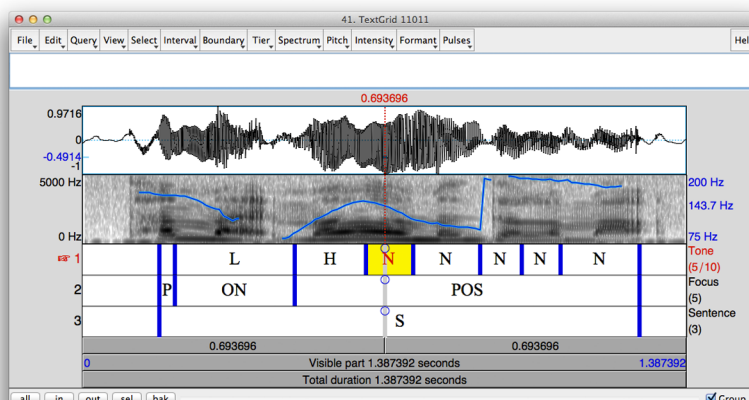


Figure 2.17 Annotation data from existing files

4. After finishing editing/viewing, click Continue to proceed to the next file.

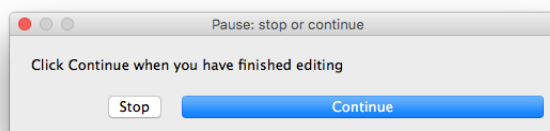


Figure 2.18 Annotation data from existing files.

5. Repeat steps 4-5 until all annotation file are edited/viewed. Users can quit the editing/viewing anytime by closing all PENTATrainer windows.

Tip: If users quit the annotation process before going through all annotation files, some of the Praat objects will not be properly removed from the “Praat Objects” window. User should make sure that these objects are removed before running other scripts to ensure the correct object reference.

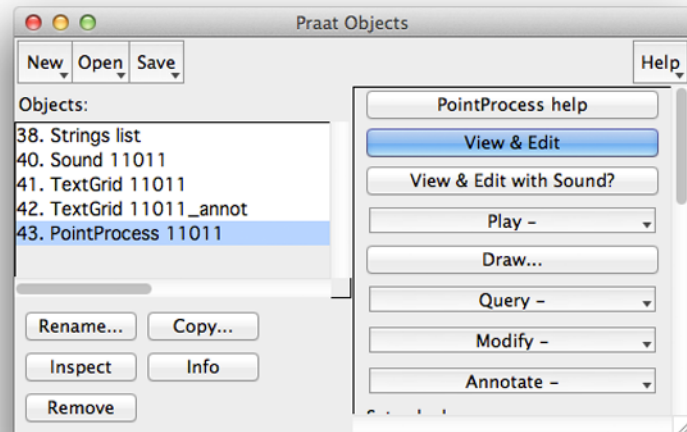


Figure 2.19 Unused Praat objects should be removed.

2.4 Auxiliary files generated during annotation

1. **FileList.txt** contains a list of input sound files in the annotation process. It contains a one column headerless table with each row correspond to each input filename. This file is generated at the beginning of the annotation process.
2. **config.txt** (required for parameter learning) contains a list of parameters in the annotation process that will pass along to the learning process. It contains a two-column table with header (parameter, value). Each row corresponds to a parameter to be passed. This file is generated at the end of the annotation process. Users must run through the annotation process at least once to generate this file.
3. **function.txt** (required for parameter learning) contains a list of communicative functions. It contains a one-column table with header with each row corresponding to each function. This file is generated at the end of the annotation process. Users must run through the annotation process at least once to generate this file.
4. **[sound_file_name].annotation** (required for parameter learning) contains annotation data in Praat TextGrid format. This file is

derived from the “.annotation_short” file which contains a more user-friendly version. This file is regenerated in each annotation process.

5. **[sound_file_name].annotation_short** contains a more user-friendly annotation in Praat TextGrid format. This file is regenerated in each annotation process.
6. **[sound_file_name].pulse** contains pulse marking for F_0 estimation. It is a Praat PointProcess object. This file is regenerated in each annotation process.
7. **[sound_file_name].rawf0** contains F_0 data directly converted from their corresponding “.pulse” files. This file is regenerated in each annotation process.
8. **[sound_file_name].PitchTier** contains trimmed F_0 data in Praat PitchTier format. This file is regenerated in each annotation process.
9. **[sound_file_name].f0** contains trimmed F_0 data directly converted from their corresponding “.PitchTier” files. This file is regenerated in each annotation process.
10. **[sound_file_name].samplef0** contains F_0 data sampled only for non-empty intervals. This file is regenerated in each annotation process.
11. **[sound_file_name].f0velocity** contains F_0 velocity data sampled only for non-empty intervals. This file is regenerated in each annotation process.
12. **[sound_file_name].timenormf0** contains timeless F_0 data normalized to have equal number of samples per interval. This file is regenerated in each annotation process.
13. **[sound_file_name].actutimenormf0** (required for parameter learning) contains F_0 and actual time values. They were sampled and normalized to the specified number of samples per interval. It is a three-column table with header with each row corresponding to each F_0 data point. The first column corresponds to the annotation

intervals of the first layer. The second and third columns are the time and F0 data. PENTATrainer learning program uses F0 and time data in this file to estimate the parameters. This file is regenerated in each annotation process.

CHAPTER 3

Learning Parameters

This chapter provides a step-by-step guide for using PENTATrainer learning tool to estimate model parameters of annotated communicative functions. It then discusses the roles of model parameters of the target approximation process and the parameter learning algorithm used in PENTATrainer.

3.1 Learning Parameters of Communicative Functions

1. Select “Learn” task in the starting window of PENTAtainer2.praat

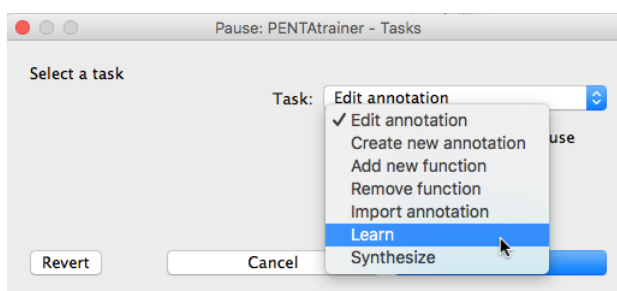


Figure 3.1 Selecting the Learn task.

Users also have the option to inspect Praat manipulation objects that contain the synthesized F_0 contour resulting from the optimized parameters together with the original F_0 contour. This allows users to visually inspect and listen to the synthesized sound. The post-low bouncing simulation can also be optionally selected (Prom-on et al., 2011b). This will include the post-low bouncing rule into the target approximation process. To use it, check the “Inspect Manipulation” box

2. The optimal values of Maximum Iteration and Learning Rate need to be empirically determined. The general idea is that the Maximum Iteration should be large enough to allow the parameters to converge and the Learning Rate should be small enough so that the change will not miss the right solution. The default values are based on previous experience and so can be used as a good starting point.

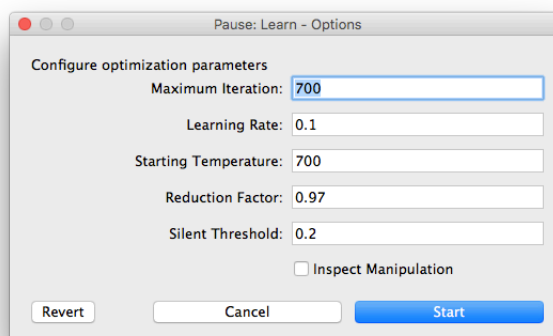


Figure 3.2 Default optimization options.

3. Press "Start" to begin the learning process, and a progress window will pop up. Wait until the progress bar reaches 100%.

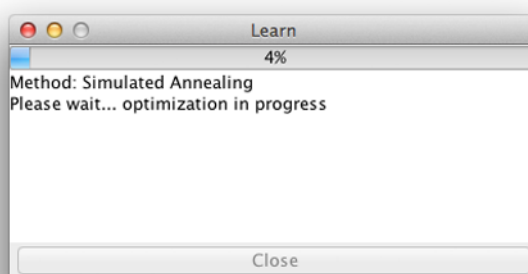


Figure 3.3 Optimization progress window.

4. Once the progress bar reaches 100%, the optimization progress window will display the resulting average per-utterance RMSE and Pearson's correlation between original and synthesized F_0 contours.

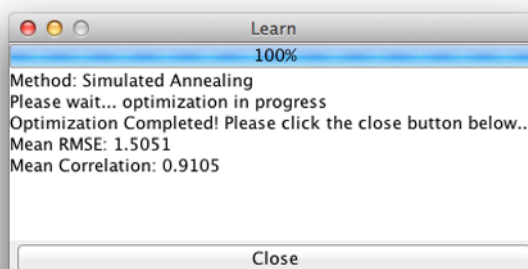


Figure 3.4 Optimization progress window when optimization process is finished.

5. Click “Close” to close the optimization progress window. Afterward, if the “Inspect Manipulation” box is checked, PENTATrainer will generate Praat manipulation object of each sound file with the synthesized F0 contour embedded in it.

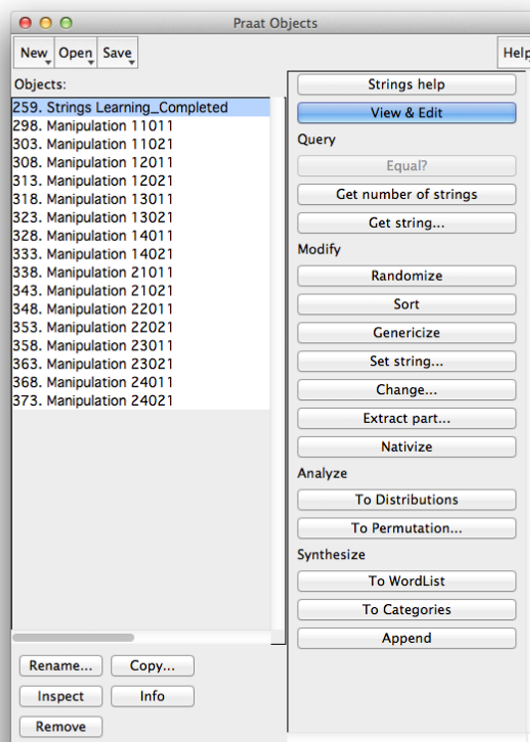


Figure 3.5 Manipulation objects generated from the optimized parameters.

6. Users can inspect the fitness of the synthesized F_0 contour by clicking the manipulation object and select “View & Edit”.

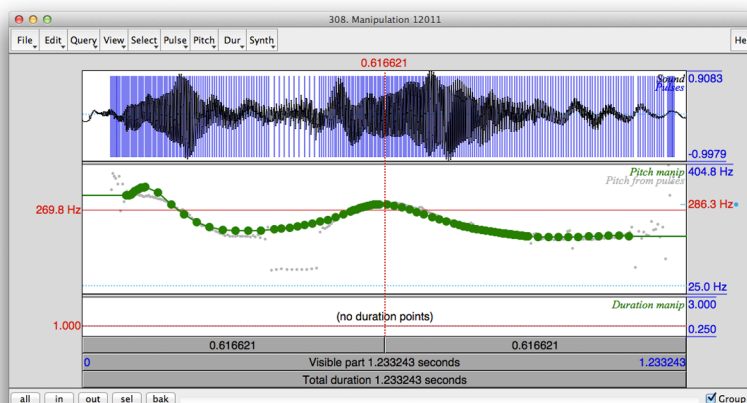


Figure 3.6 Praat manipulation object with the synthesized F_0

In the lower panel, the gray dots indicate the original F_0 while the green dots indicate the synthesized F_0 .

7. The optimized multi-functional parameters are stored in the “parameters.txt” file, which can be opened with a text editor or a spreadsheet program.

	A	B	C	D	E	F
1	Tone	Focus	Sentence	m	b	r
2	H	PRE	S	-70.33	1.25	94.45
3	L	ON	S	15.99	-8.39	26.54
4	H	POS	S	-100	18.22	11.94
5	N	POS	S	-20.05	-7.29	16.22
6	H	PRE	Q	100	-2.57	60.38
7	L	ON	Q	94.3	-14.12	20.72
8	H	POS	Q	-25.71	1.15	22.21
9	N	POS	Q	-2	-7.03	14.26
10	R	POS	S	92.37	3.86	2.95
11	R	POS	Q	100	-5.39	90.48
12	LS	ON	S	78.31	-5.14	23.02
13	L	POS	S	-10.15	-4.17	94.99
14	LS	ON	Q	28.76	-2.2	42.34
15	L	POS	Q	77.76	-12.43	23.32
16	F	POS	S	-49.11	16.48	0
17	F	POS	Q	-4.48	0.28	42.99
18	L	PRE	S	-35.32	-9.34	20.77
19	H	ON	S	-7.9	6.59	23.73
20	L	PRE	Q	33.82	-8.13	28.97
21	H	ON	Q	-3.75	2.21	3.59
22	R	ON	S	49.96	4.15	13.68
23	R	ON	Q	68.57	-4.19	64.86
24	LS	PRE	S	56.28	-4.48	25.85
25	LS	PRE	Q	88.21	-0.39	35.19
26	F	ON	S	-38.05	-0.05	72.5
27	F	ON	Q	-100	-14.95	2.9

Figure 3.7 Optimized multi-functional parameters.

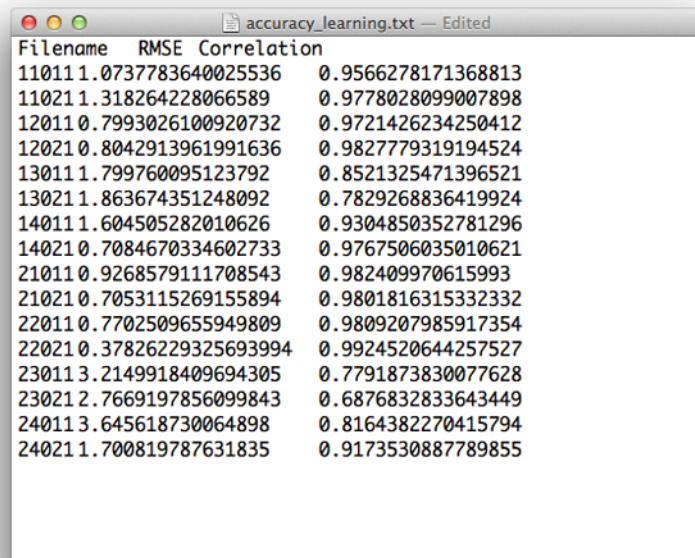
8. Using a spreadsheet program (e.g., Excel), users can sort the parameters to make them more comprehensible. In this example, the parameters are sorted by “Focus”, “Tone” and then “Sentence”.

	A	B	C	D	E	F
1	Tone	Focus	Sentence	m	b	r
2	F	ON	Q	-100	-14.95	2.9
3	F	ON	S	-38.05	-0.05	72.5
4	H	ON	Q	-3.75	2.21	3.59
5	H	ON	S	-7.9	6.59	23.73
6	L	ON	Q	94.3	-14.12	20.72
7	L	ON	S	15.99	-8.39	26.54
8	LS	ON	Q	28.76	-2.2	42.34
9	LS	ON	S	78.31	-5.14	23.02
10	R	ON	Q	68.57	-4.19	64.86
11	R	ON	S	49.96	4.15	13.68

Figure 3.8 Sorted optimal parameters

Figure 3.8 shows part of the sorted parameters. A number of interesting observations can be made from this data. It shows that F (Falling) tone has negative slope while R (rising) tone has positive slope. H has almost static slope with a relatively high target height while L has much lower target height. Pitch targets of LS (sandhi L tone) are similar to R.

9. Utterance specific RMSE and correlation of optimized parameters are stored in the “accuracy_learning.txt” file.



Filename	RMSE	Correlation
110111.0737783640025536	0.9566278171368813	
110211.318264228066589	0.9778028099007898	
120110.7993026100920732	0.9721426234250412	
120210.8042913961991636	0.9827779319194524	
130111.799760095123792	0.8521325471396521	
130211.863674351248092	0.7829268836419924	
140111.604505282010626	0.9304850352781296	
140210.7084670334602733	0.9767506035010621	
210110.9268579111708543	0.982409970615993	
210210.7053115269155894	0.9801816315332332	
220110.7702509655949809	0.9809207985917354	
220210.37826229325693994	0.9924520644257527	
230113.2149918409694305	0.7791873830077628	
230212.7669197856099843	0.6876832833643449	
240113.645618730064898	0.8164382270415794	
240211.700819787631835	0.9173530887789855	

Figure 3.9 Synthesis accuracy with the optimized parameters.

10. The closeness of fit between the original and synthesized F_0 contours can also be visually inspected by opening the file [filename].synf0.

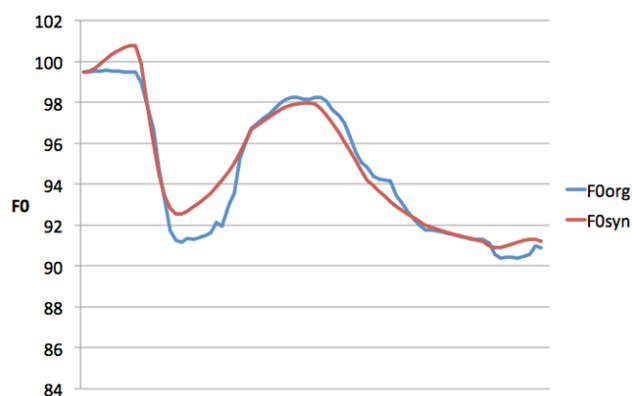


Figure 3.10 Visual inspection of synthesized F_0 contour

11. The changes in the learning errors over iterations can be found in the “total_error.txt” file. Using a spreadsheet program, users can plot the changes in learning errors over iterations.

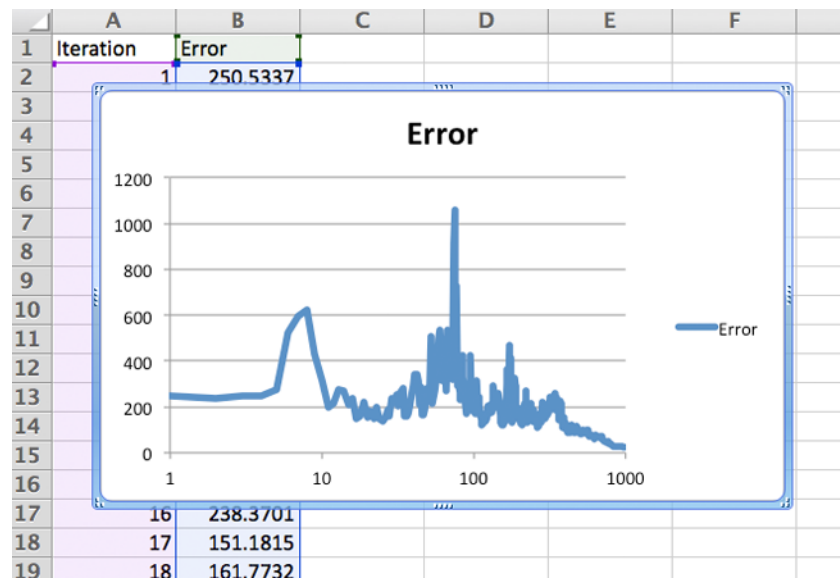


Figure 3.11 Optimization errors (y-axis) over iterations (x-axis, logarithmic scale).

3.2 Understanding model Parameters

This section discusses the role of model parameters of the target approximation process and the parameter learning algorithm used in PENTATrainer2.

As discussed in section 2.1, PENTA assumes that F_0 is a response of the underlying target approximation process that implements designated communicative functions encoded in the form of target approximation parameters. The quantitative version of the target approximation model shown in Figure 2.2 has been proposed, namely the quantitative Target Approximation (qTA) model to explain the F_0 contour changes due to tones and intonations (Prom-on *et al.*, 2009). The qTA model represents F_0 as a surface response of the target approximation process which is driven by pitch targets. A pitch target is a forcing function representing the joint force of the laryngeal muscles that control vocal fold tension. F_0 in qTA is represented by a simple linear equation,

$$x(t) = mt + b \quad (3.1)$$

where m and b denote the slope and height of the pitch target respectively. t is the time relative to the initial time of the interval.

The F_0 control is implemented through a third-order critically damped linear system, in which the total response is

$$f_0(t) = x(t) + (c_1 t + c_2 t + c_3 t^2) e^{-\lambda t} \quad (3.2)$$

where the first term $x(t)$ is the pitch target and the second term is the natural response of the system. The transient coefficient c_1 , c_2 , and c_3 are calculated based on the initial F_0 dynamic state and the pitch target of the specified interval. The parameter λ represents the strength of the target approximation movement. In qTA, the initial F_0 dynamic state consists of initial F_0 level $f_0(0)$, velocity $f'_0(0)$, and acceleration $f''_0(0)$. The dynamic state is transferred from one syllable to the next at the interval boundary to ensure the continuity of F_0 movement. The three transient coefficients are computed with the following formulae.

$$c_1 = f_0(0) - b \quad (3.3)$$

$$c_2 = f'_0(0) + c_1 \lambda - m \quad (3.4)$$

$$c_3 = (f''_0(0) + 2c_2 \lambda - c_1 \lambda^2) / 2 \quad (3.5)$$

In PENTATrainer, three parameters are required for each interval to control the F_0 trajectory of each interval: pitch target slope (m), pitch target height (b), and strength of target approximation (λ). m and b specify the form of the pitch target. For example, the Mandarin rising and falling tones have positive and negative m values, respectively (Prom-on et al., 2009, 2011a). λ indicates how rapidly a pitch target is approached. The higher the value of λ the faster F_0 approaches the target. For example, λ of the Mandarin neutral tone has been found to be smaller than in other tones (Prom-on et al., 2011a).

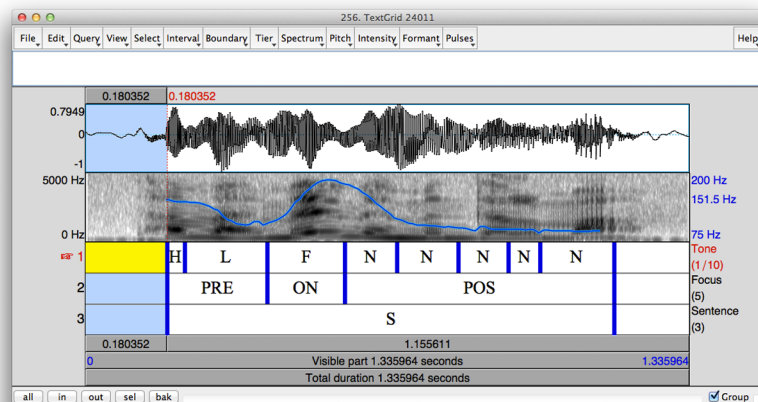


Figure 3.12 Annotated functional combinations.

As done in the annotation of the speech data in chapter 2, communicative functions in different layers are treated as parallel to each other. The temporal spans of intervals in different layers are not necessarily equal, as shown in Figure 3.12. In this example, the temporal spans of tone, focus, and sentence functions are different. However, if we view functional combination at certain intervals, we can summarize the unique combinations as follows.

- Combination 1: H, PRE, S
- Combination 2: L, PRE, S
- Combination 3: F, ON, S
- Combination 4: N, POS, S

PENTATrainer assumes that the combined effect of the functions influence the target approximation parameters. Thus, for each functional combination, a set of target approximation parameters (m , b , λ) is estimated. Note that the functional combination may repeat a number of times in the dataset. PENTATrainer estimates the parameters based on the compiled functional combination from the whole dataset.

3.3 Stochastic Optimization

The previous version of PENTATrainer (PENTATrainer1) models F_0 contours of individual utterances. Despite its success, there are difficulties when it comes to summarizing parameters of all syllables into functional categories. Because of the trade-off between model parameters, there are nonlinear interplays in the model and the differences in the optimum conditions of parameter learning process, a simple averaging procedure could sometimes result in representations that do not reflect globally optimal solutions. This issue is addressed in PENTATrainer 2.

In PENTATrainer2, instead of modeling F_0 contours of each individual utterances and summarizing the parameters afterwards, the parameters of all functional categories are optimized simultaneously, using the simulated annealing algorithm (Kirkpatrick et al., 1983). The algorithm can be summarized as follows.

- Generate parameters (randomly) of all functional combinations
- Repeat until the designated number of iterations is reached
 - For each functional combination
 - For each parameter
 - Modify the parameters (randomly)
 - Calculate the rejection probability threshold
 - Generate a random probability

- Test (probabilistically) to decide whether to accept or reject the proposed changes
- Reduce the temperature

Parameter modification of each functional combination is done in a random manner. In other words, the change of each parameter is scaled with the specified learning rate, the random number between 1 and -1, and the parameter range.

The probability of acceptance/rejection depends on the temperature parameter of the algorithm. At the initial iteration, the temperature is set to a high value to allow the parameters to evolve and converge to the global optimum over the iterations. The rejection probability threshold (p_{th}) is calculated using the following equation.

$$p_{th} = e^{-(E_{current} - E_{previous})/T} \quad (3.6)$$

where $E_{current}$ and $E_{previous}$ are the total errors between the original and synthesized F_0 calculated from the whole dataset after and before parameter modification, respectively. T is the annealing temperature. After calculating this threshold, a random probability is generated. If this random probability is higher than the threshold, the proposed parameter change is rejected. Otherwise, the change is accepted.

The total error is calculated by summing for all utterances the root mean square error (RMSE) between original and synthesized F_0 contours. Synthesized F_0 contours are generated based on the current parameters. RMSE is calculated as follows.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N \left(f_0(t_i)_{original} - f_0(t_i)_{synthesized} \right)^2} \quad (3.7)$$

where N is the number of samples of that utterance.

Thus, there are 4 parameters controlling the optimization process. Users are required to input them (or use the default values) in order to use PENTATrainer learn tool. They are

1. Maximum Iteration, indicating the number of rounds that the learning algorithm will modify and test the parameters. The larger the Maximum Iteration the longer the optimization time.

2. Learning Rate, indicating the scaling factor for parameter modification. The faster the Learning Rate the larger proposed changes in the parameter modification step.
3. Starting Temperature, indicating the starting annealing temperature as shown in Equation (3.6). The higher the Starting Temperature the greater the chance of accepting the parameters that result in high errors in the early iterations. Once the temperature started to cool down (through a number of iterations), the optimization process will become stricter and select only the parameters that result in lower error.
4. Reduction Factor indicates the percentage reduction of the temperature in each iteration. The larger the Reduction Factor the faster the convergence of parameters, but also the greater the chance of remaining in a local optimum.

3.4 Fine Tuning Optimization Process

As mentioned earlier, the four parameters controlling the optimization process need to be determined empirically. Adjusting and evaluating the optimization process are key to maximizing its effectiveness. In the previous section, we have discussed some of the measurements of effectiveness, such as optimization errors as shown in Figure 3.13, visual and perceptual inspections in Figure 3.8 and 3.12.

Error Convergence Rate

The convergence of optimization error indicates the stability of the learned parameters. This is because, as the annealing temperature reduces, the optimization process becomes greedier, thus accepting only the parameters that result in a lower total error. When plotting the optimization error over iterations, it should show a near flat line in the later part of the process. The flat line indicates that the changes in the parameters at the later stage are not significant. On the other hand, if the steady state (flat line) is reached too early, it is possible that the optimized parameters may not be the global optimum.

If the optimization error plot indicates that the convergence rate was too slow, users have 3 options to ensure that the optimized parameters reach the steady state: (1) increasing Maximum Iteration, (2) reducing Starting Temperature, or (3) reducing Reduction Factor. Increasing Maximum Iteration would give the optimization process more time for the parameter to become stable. Reducing Starting Temperature would make the whole optimization process greedier and less random.

Reducing Reduction Factor would increase the convergence rate. Care must be taken with adjusting either Starting Temperature or Reduction Factor, since changing them directly affects the characteristics of the optimization process.

On the other hand, if the convergence rate is too fast, users can either (1) increase Starting Temperature or (2) increase Reduction Factor.

Functional Category Assignment

Visual and perceptual inspections help users to identify the mismatch between original and synthesized F_0 contours. Systematic deviations of the synthesized F_0 contour from the original contour indicate that there are problems in the optimization process. The causes of these problems can be (1) the annotated functional categories are incorrect, (2) the optimization parameters are not optimal, and (3) there are effects from articulatory mechanisms not yet included in the current version of the PENTATrainer.

In the first case, if the observed deviation occurs consistently in a specific functional combination and the nature of the deviation suggests that the combination may have various pitch targets, it is possible that the initial functional category assignment is incorrect. In the second case, if the parameters are not optimal, the optimization process can be tuned by adjusting the optimization parameters as discussed earlier. In the third case, users need to wait until the additional articulatory mechanisms are incorporated into the updated PENTATrainer.

3.5 Output Files

1. **parameters.txt** contains learned target approximation parameters of all functional combinations. This file is generated at the end of the optimization process.
2. **[sound_file_name].synf0** contains original and synthesized F_0 contours with time data. This file is regenerated in each learning process.
3. **[sound_file_name].intervals** contains a Praat Table converted from “.annotation” file. This file is used in the optimization process by Java program. This file is regenerated in each learning process.

4. **accuracy_learning.txt** contains the synthesis accuracy of each utterance (sound file). The accuracy measurements are Root Mean Square Error (RMSE) and Pearson's correlation coefficient.
5. **total_error.txt** contains total RMSE summarized from all sound files in each iteration. This file is used for fine-tuning the optimization process.
6. **log.txt** contains the first occurrence of each functional combination. This file is used for troubleshooting. Users can use this file to check if there are any errors in annotation. If there is an error such as a typo, log.txt will show the typo version as another category. Users can then correct the typo in the file indicated by log.txt.

CHAPTER 4

Synthesis

This chapter explains how to use PENTATrainer2 to synthesize speech prosody with the parameters learned during optimization. This is done with the selection of the “Synthesize” task in the starting window.

4.1 Required Input Files

1. **parameters.txt** contains target approximation parameters of all functional combinations that have been trained, together with the annotation data, to synthesize F_0 contour.
2. **[sound_file_name].wav** contains original speech utterance. PENTATrainer synthesis tool uses the Praat manipulation object from the original sound file as a host and embeds in it the synthesized F_0 contour.
3. **[sound_file_name].annotation** contains annotation data in Praat TextGrid format. This file is used by Synthesis.jar.
4. **[sound_file_name].intervals** contains annotation data in Praat Table format. This file is used by Synthesis.jar.
5. **config.txt** contains dataset configuration parameters. This file is used by Synthesis.jar.
6. **function.txt** contains a list of communicative functions. This file is used by Synthesis.jar.

4.2 Synthesis with Learned Parameters

The Synthesize task uses parameters from parameters.txt and related annotation files to synthesize F_0 contour and embed it in the Praat manipulation object.

1. Run PENTAtainer2.praat and select the Synthesize task.

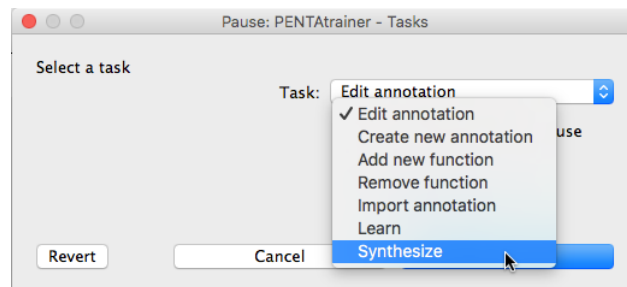


Figure 4.1 Selecting Synthesize task.

2. After clicking Start, the program will show an interactive Synthesis window. In this window, you can inspect synthesized F0 contours both visually and perceptually. You can move through the corpus by using the control panel on the top (Previous, Next). You can play original and synthesized sounds by clicking the buttons on the control panel (Play original, Play resynthesis). You can also zoom into the contours by using the zoom and arrow buttons. The list of communicative functions are displayed below the F0 contours.

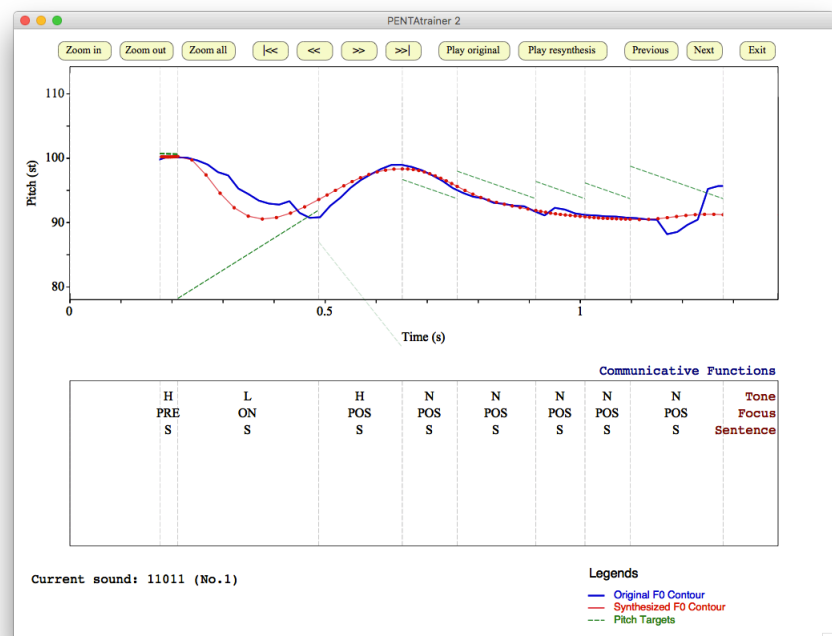
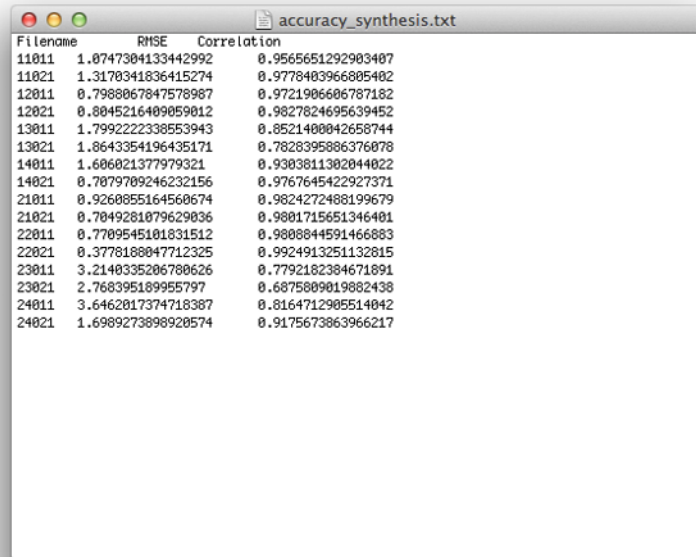


Figure 4.3 Inspecting synthesized F_0 contours.

3. Utterance specific RMSE and correlation of synthesized contours are stored in the “accuracy_synthesis.txt” file.



Filename	RMSE	Correlation
11011	1.0747304133442992	0.9565651292903407
11021	1.3170341836415274	0.9778403966805402
12011	0.7988067847578987	0.9721906606787182
12021	0.8045216409059012	0.9827824695639452
13011	1.7992222338553943	0.8521400042650744
13021	1.8643354196435171	0.7828395886376078
14011	1.606021377979321	0.9303811302044022
14021	0.7879709246232156	0.9767645422927371
21011	0.9260855164560674	0.9824272488199679
21021	0.7049281079629036	0.9801715651346401
22011	0.7709545101831512	0.9808844591466883
22021	0.3778188047712325	0.9924913251132815
23011	3.2140335206780626	0.7792182384671891
23021	2.768395189955797	0.6875809019882438
24011	3.6462017374718387	0.8164712905514042
24021	1.6989273898920574	0.9175673863966217

Figure 4.4 accuracy_synthesis.txt

4. Synthesized F_0 contour of each input sound file is stored in the file [sound_file_name].synthesizedf0.

4.2 Synthesis with Modified Parameters for Perception Experiment

The synthesis tool in PENTATrainer2 can also be used to generate stimuli for intonation studies. That is, the parameters learned through the optimization process can be modified to generate perception stimuli for testing various hypotheses. The advantage of creating stimuli this way is that they sound more natural than those generated through direct acoustic manipulations, thanks to the implementation of the basic articulatory mechanisms in the PENTA model.

For example, The 5 F_0 contours in Figure 4.5 are generated by just changing the value of m in the third syllable in a 8-syllable Mandarin sentence. This is done by changing the highlighted cell in the parameter.txt file, as shown in Figure 4.6. The effect of this change is seen not only on the third syllable, but also on the following syllable.

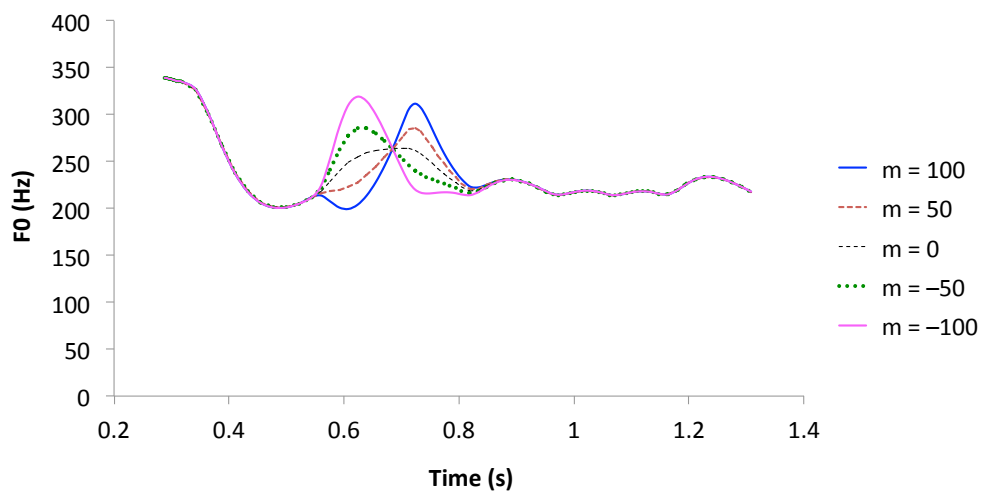


Figure 4.5 A stimulus continuum generated by changing m of the third syllable in an 8-syllable Mandarin utterance.

Tone	Focus	Sentence	m	b	r
H	PRE	S	-2.82	0.01	64.39
L	ON	S	49.7	-8.68	28.56
H	POS	S	-100	-30	4.08
N	POS	S	-27.71	-6.9	48.67
H	PRE	Q	51.35	-1.55	68.81
L	ON	Q	32.09	-7.99	31.23
H	POS	Q	-16.32	-0.39	30.74
N	POS	Q	-1.25	-7.16	15.96
R	POS	S	-55.61	-28.19	4.08
R	POS	Q	-100	-19.1	1.2
LS	ON	S	11.28	-3.95	100
L	POS	S	9.29	-16.72	8.23
LS	ON	Q	60.06	-0.74	32.27
L	POS	Q	90.77	-10.89	25.62
F	POS	S	-33.01	-0.67	35.28
F	POS	Q	-38.85	24.31	7.91
L	PRE	S	47.39	-9.78	27.94
H	ON	S	-100	-30	1.9
L	PRE	Q	29.35	-9	28.68
H	ON	Q	-100	-17.43	0.98
R	ON	S	100	-4.04	52.2
R	ON	Q	-9.61	-30	1.86
LS	PRE	S	-100	11.17	14
LS	PRE	Q	100	-3.87	23.05
F	ON	S	-100	2.01	39.89
F	ON	Q	-70.59	4.32	30.7

Figure 4.6 The parameters.txt file used to generate the contours in Figure 4.5. The value of the highlighted cell was changed to 50, 0, -50, and -100, respectively.

Reference

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). "Optimization by simulated annealing", *Science*, vol. 220, no. 4598, pp. 671-680.

Prom-on, S., Xu, Y., and Thipakorn, B. (2009). "Modeling tone and intonation in Mandarin and English as a process of target approximation", *Journal of the Acoustical Society of America*, vol. 125, no. 1, pp. 405-424.

Prom-on, S., Liu, F. and Xu, Y., (2011a). "Functional modeling of tone, focus, and sentence type in Mandarin Chinese", in proceedings of the 17th International Conference of Phonetic Sciences, 1638-1641, Hong Kong, China.

Prom-on, S., Xu, Y., and Liu, F., (2011b). "Simulating post-L bouncing by modeling articulatory dynamics", in proceedings of INTERSPEECH 2011, 289-292.

Xu, Y. (2005). "Speech melody as articulatorily implemented communicative functions", *Speech Communication*, vol. 46, no. 3-4, 220-251.

Xu, Y. and Prom-on, S., (2014). Toward invariant functional representations of variable surface fundamental frequency contours: Synthesizing speech melody via model-based stochastic learning. *Speech Communication* 57, 181-208.