RhythmTools.pl

Automating measurements to facilitate analysis of speech rhythm

S. Qiouyi Lu

s@qiouyi.lu

LING 415

December 10, 2010

1 Introduction

1.1 Background

This project is based off of the experiment performed in "Quantitative characterizations of speech rhythm: Syllable-timing in Singapore English" by Low Ee Ling, Esther Grabe, and Francis Nolan, published in *Language and Speech*, 2000, 43(4), pp. 377–401.

The rhythm patterns of languages can be broadly classified into two types: syllable-timed languages and stress-timed languages. In syllable-timed languages, each syllable in an utterance is roughly the same length; in stress-timed languages, syllables in utterances vary in length depending on whether or not they are stressed. Additionally, syllable-timed languages are expected to show little, if any, vowel reduction, as vowels are typically reduced in unstressed syllables, which are not as prominent, if they exist at all, in syllable-timed languages.

Certain phonetic measurements provide quantitative information for classifying a language as syllable- or stress-timed. To address the question of syllable length, the syllables themselves can be measured and their lengths compared. For greater consistency in measuring syllable length, Low et al. measure vowel lengths as an indicator of syllable length. Additionally, as a more accurate illustration for syllable length differences, Low et al. calculate the Pairwise Variability Index for vowels in an utterance, yielding an index that compares adjacent vowels, rather than averages all vowel durations together, thereby potentially neutralizing key differences in the syllable length patterns of two languages.

In order to quantitatively evaluate whether or not a vowel is reduced, Low et al. calculate the formant values for each vowel based on a measurement roughly in the temporal center of the vowel, then compute a vector value for the dispersion of that vowel from the center of the speaker's vowel space. If vowel formants are significantly closer to values in the center of the vowel space in unstressed position than when they were uttered full, stressed position, then the vowel is considered to have been reduced.

Low et al. recruited ten speakers (five male and five female) of Received Pronunciation (RP) and ten speakers of Standard Singapore English (SSE) to record ten predetermined sentences in order to make quantitative measurements. For this project, I took a sample of approximately 85 syllables of natural speech from a previously recorded interview of a speaker of SSE.

This experiment, and other experiments that target speech rhythm, produce a large number of files that need to be measured in a consistent manner. Although the script is not equipped to make automatic measurements of utterance and vowel boundaries, it is still able to automate a significant portion of the calculations in a consistent manner, thereby reducing the amount of human labor and human error. In particular, the script automatically calculates syllable duration based on vowel length, formants for each vowel, and the PVI for each utterance. The script works in conjunction with information from Praat and rewrites the information along with the output of each calculation into comma-separated value (CSV) files that can be read into other programs for further analysis.

1.2 Required equipment

This script uses Praat (available online at `http://www.fon.hum.uva.nl/praat/`) and Perl (comes pre-installed on Unix and Mac systems; available online for updates or Windows at `http://www.perl.org/`). Audio files can be in any Praat-readable format; the script only relies on valid TextGrids and valid formant information from Praat.

To save the TextGrid in the appropriate format,

(1) Select the TextGrid in the Praat objects window.

(2) Select Write > Write to text file…

(3) Save the file as (base name).TextGrid.

To save the formant tracker in the appropriate format,

(1) Make sure formants are visible by selecting "Show formants" from the Formant menu in the spectrogram window.

(2) Select Formant > Extract visible formant contour from the spectrogram window.

(3) Select the extracted formant information from the Praat objects window.

(4) Select Write > Write to text file…

(5) Save the file as (base name).Formant.

1.3 Prerequisites

The script is unable to automatically detect vowel boundaries, word boundaries, and utterance boundaries. Therefore, TextGrids must have at least two tiers: one labeled ”Vowels” and the other labeled ”Utterances”. A ”Words” tier is optional; if word boundaries are marked, the script will pull the boundary and duration information for words.

All boundaries must be marked by hand, and all intervals must be labeled. The script will discard any unlabeled intervals. If formant information is being used to calculate vowel dispersion from the center of the vowel space for stressed and unstressed vowels (which the script currently cannot calculate), stressed vowels should be marked by hand within the TextGrid intervals.

The script is currently unable to find and calculate multiple formant measurements for diphthongs. If diphthong measurements are required, a workaround is to divide the diphthong interval into two even vowel intervals and label each with the individual diphthong component (e.g. [aɪ] can be separated into [a] and [ɪ], the formants of which will be computed separately).

2 Running the script

The script can be invoked from the command line by the following command:

```
perl RhythmTools.pl (base names)
```

The script can process multiple file sets. For instance, for the files `file1.TextGrid`, `file1.Formant`, `file2.TextGrid`, and `file2.Formant`, the command would be:

```
perl RhythmTools.pl file1 file2
```

However, the TextGrid and formant information that the script is to process must be in the same directory as the script itself.

2.1 Calculating vowel duration

The script pulls the xmin and xmax of each interval from the text grid and prints those, along with the interval label and a numerical label for the entry, into a CSV file. The script calculates the duration of the vowel from the given xmin and xmax and will also print that to the CSV file.

For instance, for the input in figure 1, the script will print a corresponding, reformatted output (figure 2).

```
intervals [187]:
    xmin = 23.480663780663782
    xmax = 23.63899765423411
    text = "u"
```

Figure 1. TextGrid input for a vowel.

```
"Count","Vowel","Start Point","End Point","Duration"
...
75,u,23.480663780663782,23.63899765423411,0.158333873570328
```

Figure 2. CSV output for the vowel in figure 1.

The script will calculate durations for all labeled vowel intervals, all labeled word intervals, and all labeled utterance intervals in the same TextGrid.

2.2 Calculating PVI

As mentioned in the introduction, the PVI calculation, which compares pairs of adjacent vowels in an utterance, is a more accurate representation of the difference in average syllable length in an utterance than a simple average of the duration of all vowels in the utterance. For instance, Low et al. provide the following two hypothetical languages:

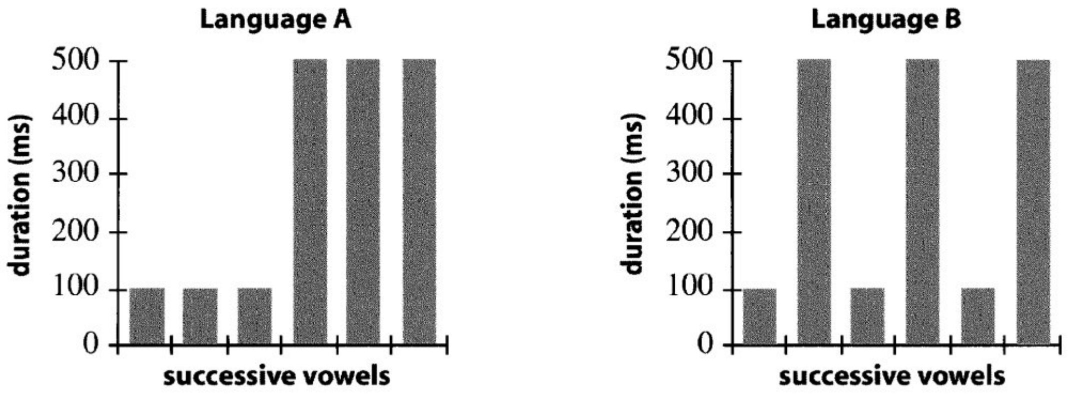

Figure 3. Successive vowel durations of hypothetical languages A and B.

The average vowel duration for both languages would result in the same number, even though both languages display very different patterning in syllable length. An average of the durations would not capture this difference, whereas a PVI value would indeed capture the greater variability in syllable duration for Language B.

The script calculates the PVI for each utterance based on the following equation:

$$\text{PVI} = 100 \times \left[\sum_{k=1}^{m-1} \frac{\left| \frac{d_k - d_{k+1}}{\frac{1}{2}(d_k + d_{k+1})} \right|}{m-1} \right]$$

where $m =$ number of vowels in utterance and $d =$ duration of the k^{th} vowel.

Utterance length is entirely user-defined and can vary in range from word boundaries to sentence boundaries and beyond. Vowel duration information and utterance boundary information are pulled from the CSV created in part 2.1.

The script matches vowels with utterances by lining up the xmin and xmax values of both and finding vowels whose xmin and xmax values fall within the xmin and xmax values of the utterance. Therefore, the only caveat is that frames that do not line up neatly with utterance boundaries (e.g. an utterance boundary occurring in the middle of a vowel) will be discarded for the sake of clarity. The script then takes all the vowels that have been matched with an utterance and calculates the PVI for that utterance.

2.3 Measuring vowel formants

The script pulls vowel information from the previously created CSV file and pulls formant information from Praat's formant tracker. The script then calculates the xmin and xmax of each formant frame, as well as the midpoint of the frame. The script also calculates the midpoint of each vowel frame. Each vowel, then, is represented in the script as the following set of values ($xmin_v, $xmid_v, and $xmax_v):

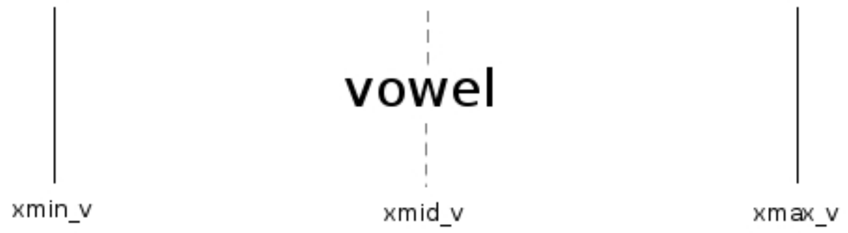

Figure 4. A typical vowel frame.

From this information, the script iterates over each formant frame to find a frame that:

(1) falls within the xmin and xmax of the vowel frame and

(2) has the smallest distance, calculated as the absolute value of the difference between $xmid_v and $xmid_f, between the xmid of the formant frame and the xmid of the vowel frame, therefore best approximating the midpoint of the vowel.

Figure 5 provides a visualization of this iteration process.

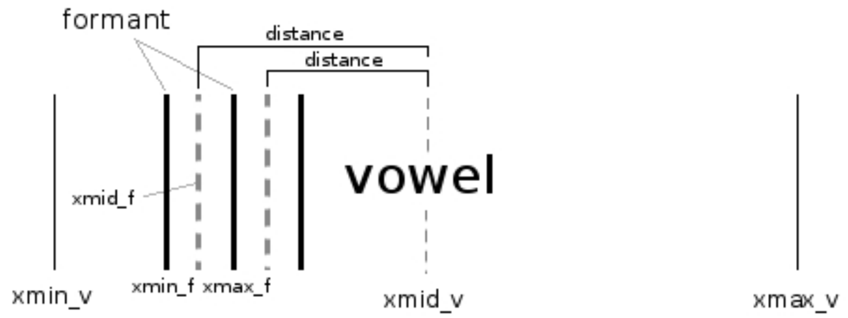

Figure 5. A sample of two formants being compared. The script will continue to examine formant windows until it finds the smallest distance between the xmids.

The script then prints the formants from this point into a separate CSV file.

Depending on the settings for the formant tier, however, formants can misalign with vowels if there are unusually wide formant frames or unusually short vowels frames (or a combination of both). In the event of this misalignment, formants will not be printed in the CSV.

2.4 Log file

The log file (`RhythmTools.log`) is used for debugging purposes. For each file set, it will log the successful completion of each process, or will log warnings if any warnings should occur. The log file also notes which vowels fall into which utterance frame and provides a line-by-line record of formants being calculated for each vowel. Please note that the log can become rather large if any of the individual files (TextGrid, formant tracker, etc.) are particularly large.

3 Discussion and conclusion

This script successfully automates a significant portion of the measurements and calculations. In a test run of the script on a sample of approximately 85 syllables of natural SSE speech, the script calculated a PVI of 47.8194554411902, which is within the range that Low et al. found among their speakers.

Future versions of this script can better address ambiguous boundaries. Currently, if there are overlapping boundaries (i.e. an utterance boundary in the middle of a vowel), the overlapped section will be discarded. Additionally, the script can address the issue of diphthongs and can automate multiple formant calculations for diphthongs in the future.

4 References

Low Ee Ling, Esther Grabe, and Francis Nolan. 2000. "Quantitative characterizations of speech rhythm: Syllable-timing in Singapore English." *Language and Speech*, 43(4), pp. 377–401. Print.

Schwartz, Randal L., Tom Phoenix, and brian d foy. 2008. *Learning Perl*. 5th ed. Sebastopol, CA: O'Reilly Media, Inc. Print.

5 Appendix: Source Code

```
#!/usr/bin/perl
########################################################################
#
# RhythmTools.pl
#
# INVOCATION: perl RhythmTools.pl FILENAME
# -----------
#    e.g. perl RhythmTools.pl file1
#         (for file1.TextGrid and file1.Formant)
#
# INPUT:
# ------
#
# * Praat TextGrids with vowels and utterances annotated.  Word
#   annotation is optional.
#
# * Praat formant tier (.Formant).
#
# OUTPUT:
# -------
#
# * Calculates durations for vowel, word, and utterance tiers on
#   TextGrid.  Outputs durations, as well as start- and end-points,
#   into a separate CSV file for each tier.
#
# * Calculates PVI for utterances.
#
# * Calculates formants for vowels based on formant information closest
#   to the midpoint of the vowel.
#
# * Produces log (RhythmTools.log) with a copy of the processes.
#
# S. Qiouyi Lu | LING 415 | December 10, 2010
#
########################################################################

# Open log.
open LOG, ">RhythmTools.log";

# Open input files.
foreach $filename (@ARGV) {

    print LOG "FILENAME: \U$filename\E\n";
    print LOG "--------------------------------------------------\n\n";
```

```
##################################################
# PROCESS TEXTGRID AND CALCULATE VOWEL DURATIONS #
##################################################

# Check that $filename.TextGrid exists.
if (! open TEXTGRID, "<$filename.TextGrid") {
    # Warn and skip if either file is missing.
    print LOG "Missing TextGrid for $filename.";
    next;

} else {

    print LOG "Extracting TextGrid information to CSVs... ";

    # Open TextGrid and CSVs.
    open TEXTGRID, "<$filename.TextGrid";
    open OUTPUT_CSV_UTTERANCES, ">$filename.utterances.csv";
    open OUTPUT_CSV_WORDS, ">$filename.words.csv";
    open OUTPUT_CSV_VOWELS, ">$filename.vowels.csv";

    # Print CSV headers.
    print OUTPUT_CSV_UTTERANCES "\"Count\",\"Utterance\","
        ."\"Start Point\",\"End Point\",\"Duration\"\n";
    print OUTPUT_CSV_WORDS "\"Count\",\"Word\",\"Start Point\""
        .",\"End Point\",\"Duration\"\n";
    print OUTPUT_CSV_VOWELS "\"Count\",\"Vowel\",\"Start Point"
        ."\",\"End Point\",\"Duration\"\n";

    while (<TEXTGRID>) {

        # Store which tier is being processed; reset token counter.
        $name = $1 and $count = undef if ($_ =~ /name = "(.*)"/);

        # Extract xmin and xmax.
        $xmin = $1 if ($_ =~ /xmin = (\d+\.?\d+)/);
        $xmax = $1 if ($_ =~ /xmax = (\d+\.?\d+)/);

        # Calculate duration.
        $duration = $xmax - $xmin if ($xmin and $xmax);

        # Extract the vowel label.
        my $entry = $1 if ($_ =~ /text = \"(.+?)\"/);

        # Add one to token count if vowel interval is labeled.
        $count += 1 if $entry;
        # Print relevant data only if interval is labeled.
        print OUTPUT_CSV_UTTERANCES "$count,$entry,$xmin,$xmax,".
            "$duration\n" if ($entry and ($name eq "Utterances"));
        print OUTPUT_CSV_WORDS "$count,$entry,$xmin,$xmax,".
            "$duration\n" if ($entry and ($name eq "Words"));
        print OUTPUT_CSV_VOWELS "$count,$entry,$xmin,$xmax,".
            "$duration\n" if ($entry and ($name eq "Vowels"));
    }

# Close files.
close TEXTGRID;
close OUTPUT_CSV_UTTERANCES;
close OUTPUT_CSV_WORDS;
close OUTPUT_CSV_VOWELS;

# Update log.
print LOG "done.\n\n\t$filename.vowels.csv\n\t" .
    "$filename.words.csv\n\t$filename.utterances.csv\n\n";

}

#################
# CALCULATE PVI #
#################

# Update log.
print LOG "Calculating PVIs for $filename... \n\n";

# Check that $filename.words.csv and $filename.vowels.csv exist.
if (! open OUTPUT_CSV_UTTERANCES, "<$filename.utterances.csv" or
   ! open OUTPUT_CSV_VOWELS, "<$filename.vowels.csv") {

    # Warn and skip if either file is missing.
    print LOG "WARNING: Missing one or both utterance/vowel CSV" .
        " files for $filename.";
    next;

} else {

    # Open files.
    open UTTERANCES, "<$filename.utterances.csv";
    open VOWELS, "<$filename.vowels.csv";
    open OUTPUT_CSV_PVI, ">$filename.pvi.csv";

    # Print CSV header.
    print OUTPUT_CSV_PVI "\"Utterance\",\"PVI\"\n";
```

```
# Slurp vowels and remove header.
@vowels = <VOWELS>;
shift @vowels;

# Set initial line count.
$line_n = 1;

# Extract words.
while (<UTTERANCES>) {

    # Skip header.
    next if ($_ !~ /[0-9]/);

    # Extract word.
    ($utterance, $xmin_u, $xmax_u) = (split /,/) [1, 2, 3];

    # Count lines to aid troubleshooting.
    $line_n++;

    # Print word entry to output file and update log.
    print OUTPUT_CSV_PVI "$utterance,";
    print LOG "\n$utterance -- Looking for a value between " .
        "$xmin_u and $xmax_u.\n\n";

    # Examines vowels.
    while (@vowels) {

        # Examine the first line.
        $current = $vowels[0];

        # Extract vowels.
        ($vowel, $xmin_v, $xmax_v, $duration) = (split /,/,
            $current) [1, 2, 3, 4];

        # Move to the next vowel if the vowel occurs before the
        # word range.
        if ($xmax_v < $xmin_u) {
            shift @vowels;
            print LOG "Too small!\n";
            next;
        }

        # Select new word if vowel overlaps word boundary or is
        # too big.
        print LOG "Overlapping boundary!\n" and last if
            ($xmin_v == $xmax_u);
        print LOG "Too big!\n" and last if ($xmin_v > $xmax_u
            or $xmax_v > $xmax_u);

        # If vowel is within the range, push vowel to utterance
        # array for later calculation, then toss vowel.
        push @utterance, $duration;
        print LOG "\t$vowel\t$xmin_v\t$xmax_v\n";
        shift @vowels;

    }

    # Print for words without measured vowels.
    if (! @utterance) {
        # Warn.
        print LOG "No vowels measured for utterance " .
           "\"$utterance\" on line $line_n; cannot" .
           " calculate PVI.\n";

        # Set PVI to be printed as "undef".
        print OUTPUT_CSV_PVI "undef";
    }

    # Calculate PVI.
    foreach $vowel (@utterance) {

        # Set number of syllables in word.
        $m = @utterance;

        # End process if word is monosyllabic---PVI cannot be
        # calculated.
        if ($m == 1) {
            # Warn.
            print LOG "Only one measured vowel for utterance " .
               "\"$utterance\" on line $line_n; cannot" .
               " calculate PVI.\n";

            # Set PVI to be printed as "undef".
            print OUTPUT_CSV_PVI "undef";
        }

        # Set upper range of calculation.
        last if (! $utterance[$k + 1]);

        # Calculate PVI.
        $function = abs (($utterance[$k] - $utterance[$k+1]) /
            (($utterance[$k] + $utterance[$k + 1]) / 2)) /
            ($m - 1);
        $sum += $function;
```

```
                $PVI = 100 * $sum;
                $k++;
            }

            # Empty variables.
            @utterance = ();
            $k = 0;
            $sum = 0;

            # Print result.
            print OUTPUT_CSV_PVI "$PVI\n";
            $PVI = undef;
        }
    }

    # Update log.
    print LOG "\nPVI calculations done.\n\n";

    # Close files.
    close VOWELS;
    close UTTERANCES;
    close OUTPUT_CSV_PVI;

    ########################
    # PRINT FORMANT VALUES #
    ########################

    print LOG "Calculating formants...\n\n";

    # Check that $filename.words.csv and $filename.vowels.csv exist.
    if (! open FORMANTS, "<$filename.Formant") {

        # Warn and skip if either file is missing.
        print LOG "WARNING: Missing formant tier for $filename.";
        next;

    } else {

    # Open relevant files.
    open VOWELS, "<$filename.vowels.csv";
    open FORMANTS, "<$filename.Formant";
    open OUTPUT_FORMANT, ">$filename.formants.csv";

    # Print CSV header.
    print OUTPUT_FORMANT "\"Vowel\",\"Time Point (xmid_v)\"," .
        "\"Formants\"\n";
```

```
# Slurp formants.
@formants = <FORMANTS>;

# Find and extract formant tier parameters.
shift @formants until ($formants[0] =~ /nx = ([^ ]*)/);
$nx = $1;

shift @formants until ($formants[0] =~ /dx = ([^ ]*)/);
$dx = $1;

shift @formants until ($formants[0] =~ /x1 = ([^ ]*)/);
$x1 = $1;

# Pull vowels and vowel information.
while (<VOWELS>) {

    # Skip header.
    next if ($_ !~ /[0-9]/);

    # Extract xmin, xmax, and the vowel.
    ($vowel, $xmin_v, $xmax_v) = (split /,/) [1, 2, 3];

        # Calculate xmid.
        $xmid_v = ($xmin_v + $xmax_v) / 2;

        # Count lines to aid troubleshooting.
        $line_n++;

        # Print vowel and time point.
        print OUTPUT_FORMANT "$vowel,$xmid_v,";
        print LOG "\n$vowel -- Looking for a value between " .
            "$xmin_v and $xmax_v.\n\n";

        # Print log header.
        print LOG "\$n_frame\t\$xmin_f\t\$xmax_f\t\$xmid_f\t" .
            "\$nformants\tformant\n";

        # Set $done to undef for future reference.
        $done = undef;

        # Run through formants.
        while (@formants) {

            # Find frame number and calculate endpoints.
            shift @formants until ($formants[0] =~ /frame
                \[([0-9]+)\]/);
            $n_frame = $1;
```

```
print LOG "$n_frame\t";

$xmin_f = ($n_frame - 1) * $dx + $x1;
$xmax_f = $xmin_f + $dx;
$xmid_f = ($xmin_f + $xmax_f) / 2;

# Determine number of formants in this frame.
shift @formants until $formants[0] =~ /nFormants =
    ([0-9]+)/;
$nformants = $1;

print LOG "$xmin_f\t$xmax_f\t$xmid_f\t$nformants\n";

# Move to the next vowel if the formants occurs before
# the formant range.
print LOG "Too small!\n" and next if ($xmax_f < $xmin_v);

# Select new word if vowel overlaps word boundary or is
# too big.
print LOG "Overlapping boundary!\n" and last if
    ($xmin_f == $xmax_v);
print LOG "Too big!\n" and last if ($xmin_f > $xmax_v);

# Calculate distance between midpoint of formant and
# midpoint of vowel.
$prev_distance = $distance;
print LOG "\$prev_distance = $prev_distance\t";
$distance = abs ($xmid_f - $xmid_v);
print LOG "\$distance = $distance\t";

# Find midpoint closest to the midpoint of the vowel
# and print formants from that frame.
if (($distance and $prev_distance) and ($distance >
  $prev_distance) and ! $done) {
    print LOG "Success!\n";
    print OUTPUT_FORMANT join ",", @frame;
    print OUTPUT_FORMANT "\n";

    # Mark as done to avoid repetition.
    $done = 1;
}

# Save the formant frequencies of the current frame for
# future reference.  Empty information from previous
# frame.
@frame = ();
$n = 1;
```

```
                # Extract and save formants of the current frame.
                while ($n <= $nformants) {
                        $n++;
                        shift @formants until $formants[0] =~
                            /frequency = ([^ ]*)/;
                        print LOG "\t$1\n";
                        push @frame, $1;
                        shift @formants;
                }
            }
        }
    }

    # Close files.
    close VOWELS;
    close FORMANTS;
    close OUTPUT_FORMANT;

    # Update log.
    print LOG "\nFormant calculations done.\n\n"
}

# Close log.
close LOG;

#######################################################################
#
# Acknowledgements
#
#######################################################################

# Thanks to Dr. Elliott Moreton for patiently helping me debug and
# troubleshoot my program.

# Thanks to Roger Que, Ian Kim, Abraham Buditama, and Mary Kohn for
# listening to me whine and complain about how my script wasn't
# working.
```