Journal of Neurological Disorders
ISSN: 2329-6895

Androulakis, J Neurol Disord 2019, 7:2
DOI: 10.4172/2329-6895.1000404

Research Article Open Access

Encoding and Assessing Sung Melodies in Stroke Patients with Aphasia

Anthony Androulakis*

Center for the Study of Aphasia Recovery (C-STAR), University of South Carolina, Columbia, South Carolina, USA

Abstract

Aphasia is a language and communication disorder caused by damage to the brain, usually occurring after stroke or traumatic brain injury. Two MATLAB computer programs are presented for encoding and assessing the sung melodies of stroke patients. The first MATLAB program, Sung Melody to Matrix (SMM.m), converts a patient's sung melody into a matrix containing the frequency and corresponding duration of each note sung. To find when the patient moves from one note to another, a novel method called Visual Audio Signal Envelope (VASE) is used, which determines an audio signal's envelope through visual cues. Other existing envelopes that were tested did not work as well with the voice of post-stroke patients recorded in a noisy environment. The second MATLAB program, Melodic Fidelity Evaluator (MFE.m), compares this matrix to the matrix of the tune that the patient was trying to imitate. This second program provides a fair assessment of the note interval error and the time interval error of the patient. In addition, these programs are easy-to-use and can be automated for large data sets to correlate with brain lesions in stroke patients.

Keywords: Aphasia; Stroke; MATLAB; Melody; Signal envelope; Sung Melody to Matrix (SMM.m); Visual Audio Signal Envelope (VASE); Melodic Fidelity Evaluator (MFE.m); Euclidean distance

Introduction

Aphasia is a language and communication disorder that can take away one's ability to speak. Ongoing research focuses on the relationship between singing abilities of post-stroke patients and damage in their brain. Moreover, singing has been used as a treatment for aphasic patients as in Melodic Intonation Therapy [1]. To evaluate the abilities of post-stroke patients to repeat melodies, simple melodies were first provided to the patients and they were asked to repeat them. The patients were recorded in a casual (not noiseless) environment as they attempted to repeat the melodies. The assessment of the patients' sung melodies can be a demanding task if many patient recordings need to be analyzed, and therefore a computer program would be more efficient and consistent. However, currently, there is no existing computer program to fairly assess a sung melody of a post- stroke individual in a noisy real-world environment. The fair assessment of the singing voice, that is, finding and assessing the notes that a person most likely sung, has been a subjective matter. This assessment becomes even more subjective with post-stroke patients since many have difficulties speaking and singing. Therefore, the automatic and objective assessment of sung melodies of post-stroke patients is very important. The purpose of this project is to provide this automatic and objective assessment, which can be used to correlate brain imaging findings to melodic repetition impairment.

This project consists of two codes written in MATLAB: Sung Melody to Matrix (SMM.m) and Melodic Fidelity Evaluator (MFE.m). These programs are available in GitHub [2]. The first code, SMM.m, encodes a sung melody into a matrix. The task of encoding a clear singing voice of non-aphasic individual in an ideal noiseless environment has been addressed by many automatic pitch detection algorithms [3]. This task is similar to monophonic singing transcription based on hysteresis of pitch-time curve, intonation, auditory models, and probabilistic methods [4-9]. Although there are speech recognition programs and even machine-learning algorithms to recognize the words of aphasic patients [10,11], these programs do not apply to melodies sung by post-stroke patients. The program presented here is based on a new envelope that I constructed using visual cues. The second code, MFE.m, gives a fair (and consistent) assessment of the attempt of a post- stroke patient to imitate a certain original tune. Although there exists a program that can evaluate the difference between two audio inputs [12], the presented program gives a fair and informative evaluation of the singing ability of the stroke patients. This program is informative since it gives separate error evaluations on note intervals, time durations, and number of notes added or subtracted. Other singing assessment programs which are mainly designed for singers who seek to improve their singing abilities can be found in [13-15], which use dynamic time warp (DTW) and use machine learning based on pitch interval accuracy and vibrato. Our approach uses simple rules that I find to be fair based on abilities of aphasic patients [16,17].

Implementation and Architecture

The outline of how the two programs SMM.m and MFE.m work together along with example inputs and outputs is shown in Figure 1.

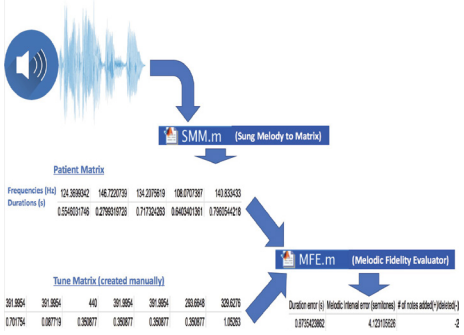

Figure 1: The input of the SMM.m is an audio file and the output is a matrix containing the frequencies and corresponding durations for each note sung. The inputs of MFE.m are the output matrix of the SMM.m as well as a matrix of the frequencies and corresponding durations of the notes of the tune the patient was attempting to repeat. The outputs of MFE.m are duration error, note interval error and number of notes added or deleted.

***Corresponding author:** Anthony Androulakis, Center for the Study of Aphasia Recovery (C-STAR), University of South Carolina, Columbia, South Carolina, USA, Tel: 803-777-7700; E-mail: aandroulakis@zoho.com

Received February 19, 2019; **Accepted** March 15, 2019; **Published** March 22, 2019

Citation: Androulakis A (2019) Encoding and Assessing Sung Melodies in Stroke Patients with Aphasia J Neurol Disord 7: 404. doi:10.4172/2329-6895.1000404

The implementation of the two programs to evaluate the repetition ability of a post- stroke patient is as follows:

1) A simple melody is played to the post-stroke patient. Examples of such melodies are the following (Figure 2):

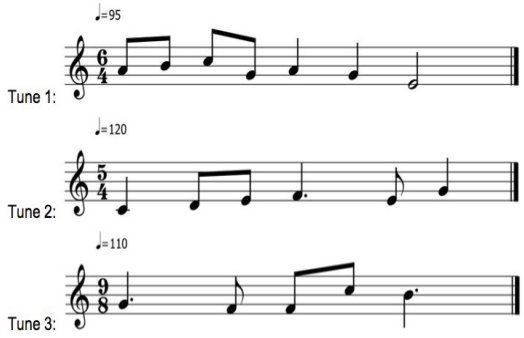

Figure 2: The WAV files for these tunes can be found in GitHub [2], and they are named tune1.wav, tune2.wav and tune3.wav, respectively.

The WAV files for these tunes can be found in GitHub [2], and they are named tune1.wav, tune2.wav and tune3.wav, respectively.

2) While being recorded, the patient attempts to repeat the played tune.
3) Open a 2018 version of MATLAB. You will need to have these two toolboxes:

- Image Processing Toolbox (version 10.2 was used)
- Audio System Toolbox (version 1.4 was used)

4) Clear your workspace variables (using the command clear) and close any figures that may be open (using the command close).
5) Find out the path of the recording of the patient, which will be put into a variable called filename as a string. An example of this in Mac computers would be:

```
filename = ' / Users / Androulakis / Recordings / patient tune1.wav';
```

And likewise, in Windows would be:

```
filename = 'C : \Users \ Androulakis \ Recordings \ patient tune1.wav';
```

Note that the apostrophe used (‘) is straight and not curved as in (’). Else, MATLAB will give you an error. Of course, your path does not have to be identical to the one shown above, so this is just an example. This is how your recording will be Note that the apostrophe used (‘) is straight and not curved as in (’). Else, MATLAB will give you an error. Of course, your path does not have to be identical to the one shown above, so this is just an example. This is how your recording will be inputted into SMM.m.

6) Run SMM.m. The output will be in a variable called PHz (PHz stands for Patient Hertz) containing the frequencies and corresponding durations for each note sung.
7) Manually create the matrix of the frequencies and durations of the melody the patient attempted to repeat. For finding the frequencies of the notes, Wikipedia’s Piano Key Frequencies chart [18] was used. For finding the durations of the notes, use the formula applicable to the tempo of your tune (Table 1).

For example, the matrices of the tunes given above in Step 1 would be inputted as follows (OHz Stands for Original Hertz):

Matrix of Tune 1: OHz=[440 493.8833 523.2511 391.9954 440 391.9954 329.6276;30/95 30/95 30/95 30/95 60/95 60/95 120/95];

Matrix of Tune 2: OHz=[261.6256 293.6648 329.6276 349.2282 329.6276 391.9954;60/120 30/120 30/120 90/120 30/120 60/120];

Matrix of Tune 3: OHz=[391.9954 349.2282 349.2282 523.2511 493.8833;90/110 30/110 30/110 30/110 90/110];

Even though these matrices could be attempted to be computed through the SMM.m, this program works better with the human voice, and thus for better accuracy, manually create the matrices of the original tunes.

8) Now that you have both the variables PHz and OHz in your MATLAB workspace, run MFE.m. This program will output the Time Error (seconds), Note Interval Error (semi- tones), and Number of Notes Added (+)/Deleted (-). The time error and note interval error is saved in the variable BR. The number of notes added or deleted is saved in the variable Notes added or deleted.

Which note gets the beat	Formula
Half Note	120/(beats per minute)
Quarter Note	60/(beats per minute)
Eight Note	30/(beats per minute)
Sixteenth Note	15/(beats per minute)

Table 1: For finding the durations of the notes, use the formula applicable to the tempo of your tune.

```
[y, fs] = audioread(filename);
song = y(:,1);
clear y
```

The SMM.m Code

In this section, the MATLAB code SMM.m which encodes a sung melody into a matrix, is described. In a melody, two notes never overlap. Therefore, the mathematical objects that characterize the melody are the frequencies and time durations of the notes. Thus, a matrix of two rows suffices where one row is reserved for the frequencies of the notes and the other row is reserved for the corresponding time durations. To identify this matrix of two rows, the program SMM.m first cuts the sung melody into intervals where each interval contains a single note and then computes the median frequency and time length in each of those intervals. The input of the SMM.m program is an audio file. Acceptable formats are .wav, .ogg, .flac, .au, .aiff, .aif, .aifc, .mp3, .m4a, .mp4. The program has been tested extensively with 1,269 recordings of human singing, each of length up to 3.6 seconds. Only one channel of the audio recording is considered.

```
[y, fs] = audioread(filename);
song = y(:,1);
clear y
```

The properties of the formations created by the amplitude graph are usually very oscillatory, as can be seen in Figure 3.

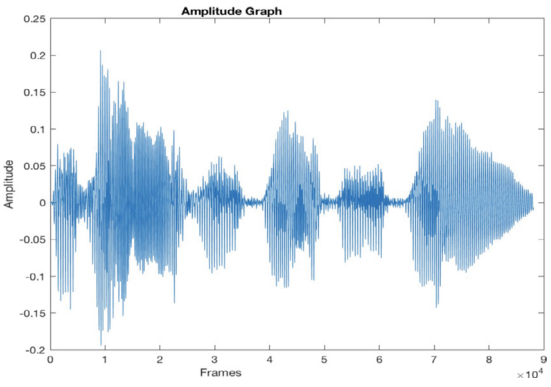

Figure 3: The highly oscillatory graph of the amplitude graph of the sung melody of a post- stroke patient can be seen here.

Therefore, the amplitudes are first converted into decibels for better readability. Figure 4 shows the graph of the sung melody of a post-stroke patient, converted into decibels and shifted above the x- axis for better image processing.

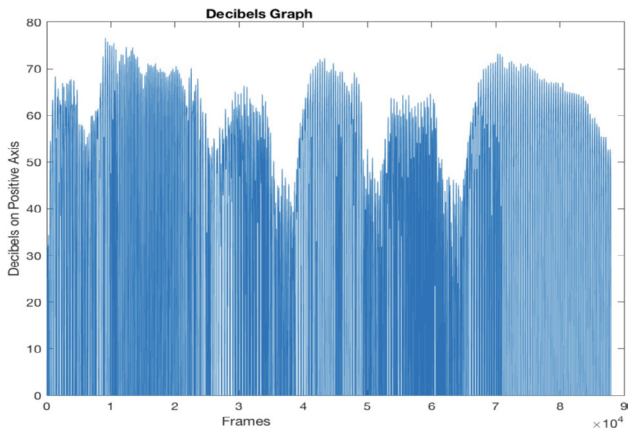

Figure 4: The graph of the sung melody converted into decibels and shifted above the x-axis is shown here. The decibels graph is not as oscillatory as the amplitude graph.

```
decibelsSong = mag2db(song);
decibelsSong(decibelsSong == -Inf) = NaN;
decibelsSong(isnan(decibelsSong)) = min(decibelsSong);
decibelsSong = decibelsSong - min(decibelsSong);
```

During singing, at the point of the passage from one note to the next, the loudness of the human voice usually dips. Unfortunately, the decibels graph is still slightly oscillatory and thus obscures the voice dips which occur when the post-stroke patient changes notes. To reveal and exemplify the voice dips which occur at the note changes, a simple graph is constructed which outlines the graph of the decibels. This simple graph is called an envelope. The envelope should not be prone to err from oscillations created by noise. There exist signal enveloping functions in MATLAB such as peak, Hilbert, and rms (Root-Mean-Square), but none performed well when tested on post-stroke singing. Also, currently existing voice enveloping functions that were tested, erroneously signified sudden unintended sounds such as taps or door slams. Here a novel envelope function is presented called Visual Audio Signal Envelope (VASE) that uses visual cues of an audio signal's graph to identify a rough outline (envelope) of the decibels graph. VASE is described next. First the picture of the decibels graph is extracted with a line thickness of 1 without the axes. The reason for thickening the lines is to fill the spaces that are between the almost vertical lines of the decibels graph.

```
saveas(plot(1:length(decibelsSong),decibelsSong,'lineWidth',1),'MelodyTask.png')
saveas(plot(1:length(decibelsSong),decibelsSong,'W'),'MelodyTaskaxes.png')
imageDecibelsGraph = imread('MelodyTask.png');
imageAxes = imread('MelodyTaskaxes.png');
decibelsGraphNoAxes = imageAxes - imageDecibelsGraph;
```

Then this image is blurred with a motion filter at a 45° to "fill" small oscillatory dips that occur in the graph. Next, a convolution of the blurred picture is taken which further blurs the borders of the already blurred object. Finally, the contour of tis blurred object is drawn and saved in a variable called contour Data.

```
grayImage = rgb2gray(decibelsGraphNoAxis);
motionBlurDecibels = imilter(grayImage, fspecial('motion',30-45));
binarizedMotionBlur = imbinarizeMotionBlurDecibels);
fullyBlurred = conv2(binarizedMotionBlur, one(50) / (50*50),'same');
newImage = fullyBlurred > 0.5
[contourData,~] = imcontour(newimage,1);
close
```

Then, x in frames (which is a time unit) and y in decibels on the positive axis are computed from the variable contour Data. This task is tedious because one must ensure that the x coordinates of the envelope are in increasing order and do not repeat. However, the dimensions of the contour are proportional to the original decibels graph and therefore can be scaled appropriately (Figure 5).

```
xContourCoord = contourData(1,2:contourData(2,1));
ycontourCoord = contourData(2,2:contourData(2,1));
shiftedY = -yContourCoord - min(-yContourCoord);
findX = xContourCoord(find(shiftedY > 30));
findY = shiftedY(find(shiftedY > 30));
shiftedX = findX - min(findX);
[sortedX,SortingIndex] = sort(shiftedX)
sortedY = findY(sortingIndex);
hSong = reshape(song,[],1)';
silence = strfind(num2str(hSong == 0,'%d'),'00')'
removedSilenceSong = song(silence(1):(silence(end)+1));
scale0fx = length(removedSilenceSong) / (max(sortedX) - min(sortedX));
decibelsSong = mag2db(removedSilenceSong);
decibelsSong(decibelsSong == -Inf) = NaN;
decibelsSong(isnan(decibelSong)) = min(decibelsSong)
decibelsSong = decibelsSong - min(decibelsSong);
scale0fy = ((max(decibelssong) - min(decibelsSong)) / (max(sortedY) - min(sortedY)+30));
scaledX = sortedX * scale0fX;
scaledY = sortedY * scale0fy;
nonRedundantX = scaledX(find(sclaedX ~= 0));
nonRedundantY = scaledY(find(scaledX ~= 0));
[~,xIndex,~] = unique(nonRedundantX);
for n = 1: length(xIndex) -1
loc = find(nonRednndantY == max(nonRedndantY(xIndex(n):xIndex(n+1)-1)));
x(1,n) = nonRedundantX(loc(find(loc >= xIndex(n) & loc <= (xIndex(n+1)-1))));
y(1,n) = nonRedundantY(loc(find(loc >= xIndex(n) & loc <= (xIndex(n=1)-1))));
clear LocationsofMaximus
end
clear n
```

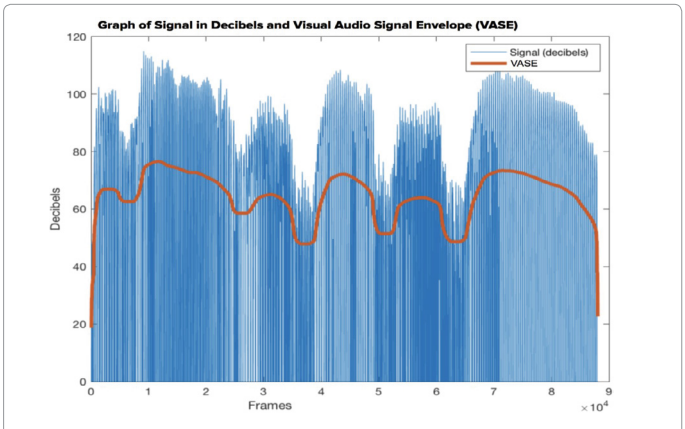

Figure 5: The graph of the decibels above the x-axis and the Visual Audio Signal Envelope (VASE) is shown here. VASE is graphed by plotting the variable y against the variable x.

VASE does not erroneously outline sharp unintended sounds as Figure 6 shows.

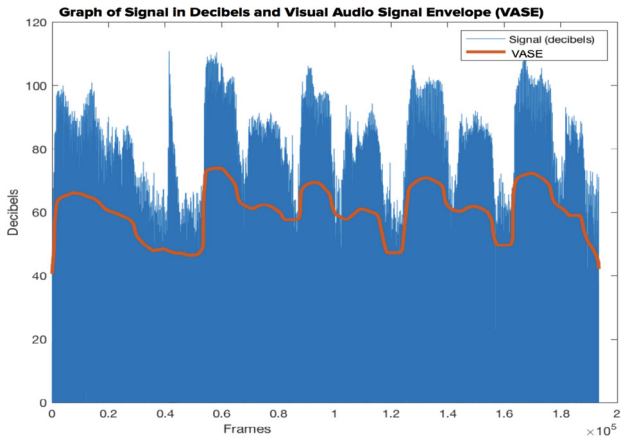

Figure 6: Another signal graph with its corresponding VASE is shown here. The sharp un-intended door slam approximately at frame 0.4 × 10^5 is not effecting VASE.

This completes the description of VASE. Then the local minima of the envelope are found. These are controlled by a min separation parameter of 50 frames and a min prominence parameter of 1. The locations of these minima are the points when the patient changes notes.

Finally, the time matrix is defined, which contains the beginning and ending in frames of each note sung. In each interval, the fundamental frequency (f0) is estimated. Using MATLAB's pitch function from the Audio System Toolbox, this task is done by utilizing the Pitch Estimation Filter (PEF) [19]. The frequencies and time intervals of each note are then saved as the first and second rows of the final matrix, respectively. In the program below, the matrix created from the patient audio is called PHz.

```
TimeMatrix = [1 round(frameWhenNoteChanges)  length(song)];
for  n = 1 : length(TimeMatrix) - 1
    [f0, ~] = pitch(song(TimeMatrix(n) : TimeMatrix(n + 1)), Fs, ...
    'Method', 'PEF', ...
    'Range', [50 800], ...
    'WindowLength', round(Fs * 0.08), ...
    'OverlapLength', round(Fs * 0.05));
    FrequencyMatrix(1, n) = median(f0);
    clear  f0
end
clear  n
PHz(1, :) = FrequencyMatrix;
PHz(2, :) = diff(TimeMatrix) / Fs;
```

The MFE.m Code

In this section, the MATLAB code MFE.m which gives a fair (and consistent) assessment of the attempt of a post-stroke patient to repeat a certain original tune, is described. The inputs of MFE.m are two matrices: the PHz matrix (produced by the SMM.m code described in the previous section) and the OHz matrix (produced manually) containing the frequencies and time durations of the tune and the patient recording respectively. In order to change the subjective matter of the assessment of a sung melody into an objective matter, the following rules are adapted:

1. The patient should not be penalized for not singing on the exact pitch as long as s/he produces the correct semitone intervals.
2. The patient should not be penalized if he/she sings at a faster or slower tempo as long as the patient preserves the correct ratios of note durations.
3. If the patient produces the original tune correctly with added notes, then for computing the error of the patient, the time durations of the original song and the note intervals when the patient was adding notes are both set equal to zero.
4. Likewise, if the patient reproduces only a subsection of the original tune, then it is assumed that the notes the patient failed to produce have time durations and note intervals containing at least one missed note are both equal to zero.

First, the MFE.m program converts the frequencies into notes by corresponding A4 to 0.

```
for i = 1 : size(OHz, 2)
  O(1, i) = floor((24 * (log(OHz(1, i) / 440) / log(2)) + 1) / 2);
  O(2, i) = OHz(2, i);
end
for i = 1 : size(PHz, 2)
  P(1, i) = floor)((24 * (log(PHz(1, i) / 440) / log(2)) + 1) / 2);
  P(2, i) = PHz(2, i);
end
```

There are three cases which are based on the number of notes the patient makes relative to the number of notes of the original tune.

Case 1: If the patient produces the same number of notes as the original tune, first rescale uniformly the durations of the time intervals of the patient's recording to minimize the Euclidean distance between the original durations and the uniformly scaled durations of the patient. Given two time vectors $(T_1,...,T_k)$ and $(t_1,...,t_k)$ of same length and non-negative coordinates, then by the Pythagorean Theorem it can be seen that the uniform rescaling factor x of $(t_1,...,t_k)$ that makes the Euclidean distance $(T_1 - xt_1)^2 + \cdots + (T_k - xt_k)^2$ minimum, is given by $x = (T_1t_1 + \cdots + T_kt_k)/(t^2 + \cdots + t^2)$. This uniform rescaling of the time intervals of the patient ensures that the patient will not be penalized if he/she sings at a different tempo than the original song, as long as the correct ratios of the durations of the individual notes of the original tune are kept. After this time interval rescaling, compute the Euclidean distance of the uniformly rescaled time intervals of the notes of the patient's recording from the time intervals of the original tune. This is the time error of the patient. Also compute the Euclidean distance of the note intervals of the patient's recording from the note intervals of the original tune. This is the note interval error of the patient.

```
if size(p,2) == size(O,2)
disp('notes added (positive) or deleted (negative) : 0')
    numerator = 0
    fori = 1 : size(P,2)
      numerator = numerator + P(2,i) * O(2,i);
    end
    denominator = 0;
    for i = 1 : size(P,2)
      denominator = denominator + P(2,i) ^ 2;
    end
    x = numerator / denominator;
    SCALEDP(1,:) = P(1,:);
    SCALEDP(2,:) = P(2,:);
    TE = 0;
      for i = 1 : size(P,2)
        TE = TE + (O(2,i) - SCALEDP(2,i)) ^ 2;
    end
    TTE = sqrt(TE);
    JE = 0;
    for i = 1 : size(P,2) - 1
      JE = JE + ((O,(1,i+1) - O(1,i)) - (P(1,i+1) - P(1,i))) ^ 2;
    end
  TJE = sqrt(JE);
  BR = [TTE, TJE];
  disp('The results are : '')
  disp('Time error (sec) Note Interval error (semitones)')
  disp(BR)
end
```

Case 2: If the patient produces more notes than the original tune, then first select a submatrix of the matrix of the patient that has the same number of columns (each column corresponds to one note made) as the original and minimizes the Euclidean distance of the note interval error. Second, scale the time intervals of the matrix of the original tune to minimize the Euclidean distance of the time intervals between these two matrices of the same size. Third, enlarge the matrix of the original tune by adding auxiliary columns where the patient added notes. Each such auxiliary column contains a zero for the time interval and contains the same tone as the column on its left. Now the augmented matrix of the original song and the matrix of the patient's recording have the same size, so the Euclidean distance of their top rows gives the note error, and the Euclidean distance of their bottom rows gives the time duration error. To understand the mentioned augmentation in the third step, assume for example that the patient produces the matrix

$$\begin{pmatrix} N1 & N2 & N3 & N4 & N5 \\ T1 & T2 & T3 & T4 & T5 \end{pmatrix}$$

While the matrix of the original song was merely

$$\begin{pmatrix} N1 & N2 & N4 \\ T1 & T2 & T4 \end{pmatrix}$$

(here $N1, N2, \ldots$ stand for notes and $T1, T2, \ldots$ stand for time durations). Then the matrix of the original song is augmented to

$$\begin{pmatrix} N1 & N2 & N2 & N4 & N4 \\ T1 & T2 & 0 & T4 & 0 \end{pmatrix}$$

The note error becomes

$$\sqrt{\left((N2-N1)-(N2-N1)\right)^2+\left((N3-N2)-(N2-N2)\right)^2+\left((N4-N3)-(N4-N2)\right)^2+\left((N5-N4)-(N4-N4)\right)^2}$$

The time error becomes

$$\sqrt{(T1-T2)^2+(T2-T2)^2+(T3-0)^2+(T4-T4)^2+(T5-0)^2}$$

```
if size(p,2) > size(0,2)
disp(['notes added(positive) or (negative) : 'num2str(p,2) - size(0,2))'])
  C = nchoosek(1 : size(p,2), size(0,2));
  for k =: size(C,1)
    Sk = P(:, C(k,:);
    if size(Sk,2) == 1
     SQJE = 0;
  else
    SQJE = 0;
    for L = 1(size(Sk,2) - 1)
      SQJE = SQJE + ((0(1,L+1) - 0(Sk(1,L+1) - Sk(1,L))) ^ 2;
    end
   end
   JEMATRIX(k,1) = SQJE;
end
MINJE = min(JEMATRIX);
```

```
  if size(p,2) > size(0,2)
  disp(['notes added(positive) or (negative) : 'num2str(p,2) - size(0,2))'])
    C = nchoosek(1 : size(p,2), size(0,2));
    for k =: size(C,1)
      Sk = P(:, C(k,:);
      if size(Sk,2) == 1
       SQJE = 0;
    else
      SQJE = 0;
      for L = 1(size(Sk,2) - 1)
        SQJE = SQJE + ((0(1,L+1) - 0(Sk(1,L+1) - Sk(1,L))) ^ 2;
      end
     end
     JEMATRIX(k,1) = SQJE;
  end
  MINJE = min(JEMATRIX);
```

Case 3: If the patient produces fewer notes than the original tune,

then first select a submatrix of the matrix of the original tune that has the same number of columns (each column corresponds to one note made) as the matrix of the patient's recording and minimizes the Euclidean distance of the note interval error. Second, scale the time intervals of the matrix of the patient's recording to minimize the Euclidean distance of the time intervals between these two matrices of the same size. Third, enlarge the matrix of the patient recording by adding auxiliary columns where the patient missed notes. Each such auxiliary column contains a zero for the time interval and contains the same tone as the column on its left. Now the augmented matrix of the patient's recording and the matrix of the original tune have the same size, so the Euclidean distance of their top rows gives the note error, and the Euclidean distance of their bottom rows gives the time duration error.

```
r = 1;
for k = 1 : size(C,1)
  if JEMARTRIX(k,1) == MINJE
   Sk = P(:, C(k,:));
   numerator = 0
   for i = 1; size(O, 2)
    numerator = numerator + Sk(2, i) * O(2, i);
   end
   denominator = 0;
   for i = 1; size(O, 2)
    denominator = denominator + Sk(2, i) ^ 2;
   end
   x = numerator / denominator;
   SCALEDO(1,:) = O(1,:);
   SCALEDO(2,:) = O(2,:) * x;
   TE = 0;
    for i = 1 : size(P, 2)
     if 1 == ismember(i, C(k,:))
      TE = TE + (P(2, i) - SCALEDO(2, find(C(k,:) == i))) ^ 2;
     else TE = TE + P(2, i) ^ 2;
     end
    end
   TTE = sqrt(TE);
  JE = MINJE;
   for i = 1; size(P, 2) - 1
    if 2 ~= ismember(i, C(k,:)) + ismember(i + 1, C(k,:))
     JE = JE + (P(1, i + 1) - P(1, i)) ^ 2;
    end
   end
    TJE = sqrt(JE);
    R(r,1) = TTE;


R(r, 2) = TJE;
    r = r + 1;
  end
 end
if size(R,1) == 1
  BR = R;
  disp('The results are : ')
  disp('Time error (sec)    Note interval error (semitones)')
  disp(BR)
 else BR = min(R);
  disp('The results are')
  disp('Time error (sec)    Note interval error (semitones)')
  disp(BR)
 end
end
```

Quality Control

To check that VASE has faithfully created an envelope of the decibels graph of the patient's sung melody, run the program graphing.m, which can be found in GitHub [2]. The SMM.m code has been tested in 1064 patient wav files and their envelopes have been checked using graphing.m with very satisfactory results.

System requirements

Operating system: macOS: El Capitan (10.11) Windows: Server 2012

Ubuntu: 14.04 LTSx

Debian: 8

Red Hat: Enterprise Linux 6 (minimum 6.7) SUSE: Linux Enterprise Server 12 (minimum SP2)

Programming language: MATLAB 2018a

Additional system requirements: Minimum processor: Any Intel or AMD x86-64 processor

Recommended disk space: 4-6 GB

Minimum RAM: 4 GB

Dependencies: Image Processing Toolbox Audio System Toolbox

Software location

Code repository
Name: GitHub
Location: https://git.io/fx8rp
License: BSD 3-Clause "New" or "Revised" License
Date published: October 23, 2018

Conclusion

The significance of the presented MATLAB codes is that they can be easily implemented in other stroke research labs to evaluate the singing abilities of post-stroke patients. At the time of this writing, the computer programs described here are being used in an ongoing study to examine correlations between brain lesions and melodic repetition errors. Future studies to combine these computer programs with neural networks to investigate the correlation between melodic repetition errors and treatment recovery are of great interest.

Acknowledgements

I would like to thank Professor Fridriksson, head of the Aphasia Lab at the University of South Carolina, and the members of his lab (who can be found here: https://web.asph.sc.edu/aphasia/members/) for their support, help, and stimulating discussions during this research.

References

1. Albert ML, Sparks RW, Helm NA (1973) Melodic intonation therapy for aphasia. Arch Neurol 29: 130-131.
2. Androulakis A (2015) Encoding and assessing sung melodies. GitHub.
3. Hess W (2012) Pitch determination of speech signals: Algorithms and devices. Springer Science & Business Media. Germany.
4. Clarisse LP, Martens JP, Lesaffre M, Baets BD, Meyer HD, et al. (2002) An auditory model based transcriber of singing sequences. Inf Retrieval ISMIR. pp: 116-123.
5. DeMulder T, Martens JP, Lesaffre M, Leman M, De Baets B, et al. (2003) An auditory model based transcriber of vocal queries. Inf Retrieval ISMIR.
6. McNab RJ, Smith LA, Witten IH (1996) Signal processing for melody transcription, In: Proc 19th Australasian Comput Sci Conf 18: 301-307.
7. Haus G, Pollastri E (2001) An audio front end for query-by-humming systems. In: Proceedings of 2nd International Symposium Music Inf Retrieval (ISMIR) p: 6572.

8. Molina E, Tardon LJ, Barbancho AM, Barbancho I (2015) SiPTH: Singing transcription based on hysteresis defined on the pitch-time curve. Trans Audio Speech Language Process 23: 252-263.

9. Krige W, Herbst T, Niesler T (2008) Explicit transition modelling for automatic singing transcription. J New Music Res 37: 311-324.

10. Deller JR, Hansen JHL, Proakis JG (2000) Discrete-time processing of speech signals. Wiley-IEEE Press. USA.

11. Kohlschein C, Schmitt M, Schuller B, Jeschke S, Werner CJ (2017) A machine learning based system for the automatic evaluation of aphasia speech. IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom).

12. Wolfram Language Contributors (2018) Use DTW to compare recordings. In: Wolfram Language.

13. Molina E, Barbancho I, Gomez E, Barbancho AM, Tardon LJ (2013) Fundamental frequency alignment vs. note-based melodic similarity for singing voice assessment. IEEE International Conference on Acoustics, Speech and Signal Processing.

14. Abeßer J, Hasselhorn J, Dittmar C, Lehmann A, Grollmisch S (2013) Automatic quality assessment of vocal and instrumental performances of ninth-grade and tenth-grade pupils. CMMR. pp: 975-988.

15. Schramm R, Nunes HDS, Jung CR (2015) Automatic solfège assessment. In: 16th International Society for Music Information Retrieval Conference (ISMIR) pp: 183-189.

16. Nakano T, Goto M, Hiraga Y (2006) An automatic singing skill evaluation method for unknown melodies using pitch interval accuracy and vibrato features. In: Interspeech pp: 1706-1709.

17. Bozkurt B, Baysal O, Yüret D (2017) A dataset and baseline system for singing voice assessment, Proc. of the 13th International Symposium on CMMR pp: 25-28.

18. Wikipedia Contributors (2010) Piano key frequencies. In Wikipedia, The Free Encyclopedia.

19. Gonzalez S, Brookes M (2014) PEFAC - A pitch estimation algorithm robust to high levels of noise. IEEE/ACM Trans Audio Speech Language Process 22: 518-530.