BEE341 Lab1 Report

10/17/2019

Done by: Biniyam yemane-berhane, David Landry

Fall 2019

1)  **What is the minimum sampling rate required to digitize a speech signal whose spectrum ranges from 0 - 3000 Hz?**

minimum sampling rate is two times the highest frequency. Therefore this spectrum the sampling rate is 6,000 Hz.

**2) What is the standard sampling rate used for CD quality music? (Hint: search online for the information)**

The standard sampling rate used for CDs will be 44.1 KHz.

**3**) **Synthesize the piece appearing in Figure 2 using only information from Sections 2 and 3. Use a pause of 0.0625 seconds.**

* We can hear the music but with a poor quality. As a result, it does have pauses between the notes. In addition, its beginning and end are not smooth because they are too sharp.
* The Martlab code is here:

%--------------------------------------------------------------------------

% main1

% This is version 1 of Lab 1: Elementary Music Synthesis.

% This version takes in a sequence of notes and plays them.

% The notes are separated by a brief pause.

% Team members: David Landry and Biniyam Yemane

%--------------------------------------------------------------------------

% The variable "notes" may be modified to change the notes of the song.

notes = ['A' 'A' 'E' 'E' 'E' 'B' 'C' 'B' 'A'];

% The variable "noteLength" is a parallel array of note lengths. Its length

% must match the variable "notes"

noteLength = [2, 1, 1, 1, 1, 1, 1, 1, 4];

% "samplingFrequency" is the sampling frequency. Default is 8000 samples

% per second.

samplingFrequency = 8000;

% "pause" is the length of pause between notes.

pause = 0.0625\*samplingFrequency;

% The song is constructed below.

song = [];

for i = 1:length(notes)

switch(notes(i))

case 'A'

freq = 220\*2^(0/12);

case 'B'

freq = 220\*2^(2/12);

case 'C'

freq = 220\*2^(3/12);

case 'D'

freq = 220\*2^(5/12);

case 'E'

freq = 220\*2^(7/12);

case 'F'

freq = 220\*2^(8/12);

case 'G'

freq = 220\*2^(10/12);

end

n = 0:(noteLength(i)\*samplingFrequency/2)-1;

song = [song, cos(2\*pi\*freq/samplingFrequency\*n), zeros(1,pause)];

end

% Play the song:

soundsc(song);

4. We implemented the ADSR in our sound synthesis project. This resulted in much smoother sounding notes. The MatLab code is listed below:

%--------------------------------------------------------------------------

% main2

% This is version 2 of Lab 1: Elementary Music Synthesis.

% This version takes in a sequence of notes and plays them.

% It utilizes an ADSR to control the volume, allowing for a smoother

% progression of notes.

% Team members: David Landry and Biniyam Yemane

% File Dependencies:

% This file calls functions found in the following files:

% makeSong.m

%--------------------------------------------------------------------------

% The notes vector is the collection of notes to be played for the

% current song.

notes = ['A' 'A' 'E' 'E' 'E' 'B' 'C' 'B' 'A'];

% noteLength is a parallel vector to notes, indicating the length of each

% of the notes to be played.

noteLength = [2, 1, 1, 1, 1, 1, 1, 1, 4];

% samplingFrequency is set at 8000S/s:

samplingFrequency = 8000;

% Declare the running song vector.

song = [];

% Set up each note:

for i = 1:length(notes)

switch(notes(i))

% The switch statement assigns a frequency to the current note.

case 'A'

freq = 220\*2^(0/12);

case 'B'

freq = 220\*2^(2/12);

case 'C'

freq = 220\*2^(3/12);

case 'D'

freq = 220\*2^(5/12);

case 'E'

freq = 220\*2^(7/12);

case 'F'

freq = 220\*2^(8/12);

case 'G'

freq = 220\*2^(10/12);

end

% The n vector is an index count.

n = 0:(noteLength(i)\*samplingFrequency/2)-1;

% currentNote forms the note vector for the current note.

currentNote = cos(2\*pi\*freq/samplingFrequency\*n);

% ADSR is the magnitude multiplier for currentNote to make it sound

% smoother.

ADSR = [linspace(0,1,.05\*length(n)) linspace(1,.8,.1\*length(n)) ...

linspace(.8,.8,.7\*length(n)) linspace(0.8,0,.15\*length(n))];

% We ran into an issue multiplying currentNote by ADSR since they are

% both row matrices. Therefore, we appended the song vector one index

% at a time for the current note:

for j=1:length(n)

song = [song, currentNote(j)\*ADSR(j)];

end

end

% Play the song once it has been fully assembled.

soundsc(song);

%--------------------------------------------------------------------------

% Function: makeSong

% Description: Forms a song with the input notes and note lengths.

% Input: notes: A character vector of notes: A-G

% noteLength: A vector of note lengths:

% 1: quarter note

% 2: half note

% 4: full note

% samplingFrequency: The desired sampling frequency.

% Output: song: The song formed from the notes & lengths.

% Dependencies: This file is called from main2.m.

% Team members: David Landry and Biniyam Yemane

%--------------------------------------------------------------------------

function [song] = makeSong(notes, noteLength, samplingFrequency)

% Declare the running song vector.

song = [];

% Set up each note:

for i = 1:length(notes)

switch(notes(i))

% The switch statement assigns a frequency to the current note.

case 'A'

freq = 220\*2^(0/12);

case 'B'

freq = 220\*2^(2/12);

case 'C'

freq = 220\*2^(3/12);

case 'D'

freq = 220\*2^(5/12);

case 'E'

freq = 220\*2^(7/12);

case 'F'

freq = 220\*2^(8/12);

case 'G'

freq = 220\*2^(10/12);

end

% The n vector is an index count.

n = 0:(noteLength(i)\*samplingFrequency/2)-1;

% currentNote forms the note vector for the current note.

currentNote = cos(2\*pi\*freq/samplingFrequency\*n);

% ADSR is the magnitude multiplier for currentNote to make it sound

% smoother.

ADSR = [linspace(0,1,.05\*length(n)) ...

linspace(1,.8,.1\*length(n)) ...

linspace(.8,.8,.7\*length(n)) ...

linspace(0.8,0,.15\*length(n))];

% We ran into an issue multiplying currentNote by ADSR since they are

% both row matrices. Therefore, we appended the song vector one index

% at a time for the current note:

thisNote = []

for j=1:length(n)

song = [song, currentNote(j)\*ADSR(j)];

%thisNote = [thisNote, currentNote(j)\*ADSR(j)];

end

%thisNote = [thisNote, zeros(1, 16000-length(n))];

%song = [song;thisNote];

end

5. We added overlapped playing of notes. To implement this, we created a new function called “playSong”, which separates the song to each of its notes and plays them sequentially with 100-sample overlaps. The MatLab code for this version can be found below. We noticed a much smoother transition between notes by doing this.

%--------------------------------------------------------------------------

% main3

% This is version 3 of Lab 1: Elementary Music Synthesis.

% This version takes in a sequence of notes and plays them.

% It utilizes an ADSR to control the volume, allowing for a smoother

% progression of notes.

% It also utilizes a slight overlap between notes, allowing for smoother

% transition between notes.

% Team members: David Landry and Biniyam Yemane

% File Dependencies:

% This file calls functions found in the following files:

% makeSong2.m

% playSong.m

%--------------------------------------------------------------------------

% Modify your notes here. By default, the notes are set up to play the

% first part of the main melody to "Scarborough Fair".

notes = ['A' 'A' 'E' 'E' 'E' 'B' 'C' 'B' 'A'];

% Modify note lengths here.

% 1: Quarter note

% 2: Half note

% 4: Full note

noteLength = [2 1 1 1 1 1 1 1 4];

% Modify the sampling frequency here. By default, it is 8000 samples

% per second.

samplingFrequency = 8000;

% Below is the function call to make the song from the provided data.

song = makeSong2(notes, noteLength, samplingFrequency);

% Finally, the below function plays the constructed melody.

%soundsc(song);

test = playSong(song);

%--------------------------------------------------------------------------

% Function: makeSong2

% Description: Forms a song with the input notes and note lengths.

% Input: notes: A character vector of notes: A-G

% noteLength: A vector of note lengths:

% 1: quarter note

% 2: half note

% 4: full note

% samplingFrequency: The desired sampling frequency.

% Output: song: The song formed from the notes & lengths.

% Dependencies: This file is called from main3.m.

% Team members: David Landry and Biniyam Yemane

%--------------------------------------------------------------------------

function [song] = makeSong2(notes, noteLength, samplingFrequency)

% Declare the running song vector.

song = [];

% Set up each note:

for i = 1:length(notes)

switch(notes(i))

% The switch statement assigns a frequency to the current note.

case 'A'

freq = 220\*2^(0/12);

case 'B'

freq = 220\*2^(2/12);

case 'C'

freq = 220\*2^(3/12);

case 'D'

freq = 220\*2^(5/12);

case 'E'

freq = 220\*2^(7/12);

case 'F'

freq = 220\*2^(8/12);

case 'G'

freq = 220\*2^(10/12);

end

% The n vector is an index count.

n = 0:(noteLength(i)\*samplingFrequency/2)-1;

% currentNote forms the note vector for the current note.

currentNote = cos(2\*pi\*freq/samplingFrequency\*n);

% ADSR is the magnitude multiplier for currentNote to make it sound

% smoother.

ADSR = [linspace(0,1,.05\*length(n)) ...

linspace(1,.8,.1\*length(n)) ...

linspace(.8,.8,.7\*length(n)) ...

linspace(0.8,0,.15\*length(n))];

% We ran into an issue multiplying currentNote by ADSR since they are

% both row matrices. Therefore, we appended the song vector one index

% at a time for the current note:

thisNote = []

for j=1:length(n)

thisNote = [thisNote, currentNote(j)\*ADSR(j)];

end

for j=length(n):15999

thisNote = [thisNote 2];

end

%thisNote = [thisNote, zeros(1, 16000-length(n))];

song = [song;thisNote];

end

%--------------------------------------------------------------------------

% Function: playSong

% Description: Plays the song. Utilizes an algorithm that allows

% note overlap.

% Input: song: The song to be played.

% Output: z: The song modified with overlaps.

% Dependencies: This file is called from main3.m.

% Team members: David Landry and Biniyam Yemane

%--------------------------------------------------------------------------

function z = playSong(song)

%mySong = song;

overlap = 100;

z = [];

for i=1:size(song,1)

note = song(i,:);

if (find(note==2,1))

lastIndex = find(note==2,1);

else

lastIndex = 16001;

end

%lastIndex = find(note==2,1);

% noteTrunc is the current note

noteTrunc = note(1:lastIndex-1);

if (i == 1)

z = noteTrunc;

else

z2 = [z, zeros(1, length(noteTrunc)-overlap)];

noteTrunc = [zeros(1, length(z)-overlap), noteTrunc];

z = z2 + noteTrunc;

end

end

soundsc(z);