# Composition And IIR Filtering Of a Song in MatLab And In C Language
# LAB-1
# ECE161B

Samuel Liimatainen

March 29, 2022

| | |
|---|---|
| Date Performed: | 1/25/22 to 2/3/22 |
| Lab TA/'s: | Yiqian Wang |
| | Prasad Kamath |
| Instructor: | Professor Truong Nguyen |

## 1 Introduction

The purpose of this lab is to compose a song in MatLab and in C language. The song is then passed through an IIR high pass filter where lower frequency components are attenuated. We can visualize these results using the spectrogram function in MatLab which allows us to see each frequency component of a signal. It can be seen the at the IIR filter attenuates parts of the song.

## 2 Task 1

Write a MATLAB function generate melody() which generates a time domain signal given the sampling rate Fs, a list of notes ziNi=1 (in Hz), and the duration of each note (in seconds) t.
The melody that is generated goes up in pitch as it is played.

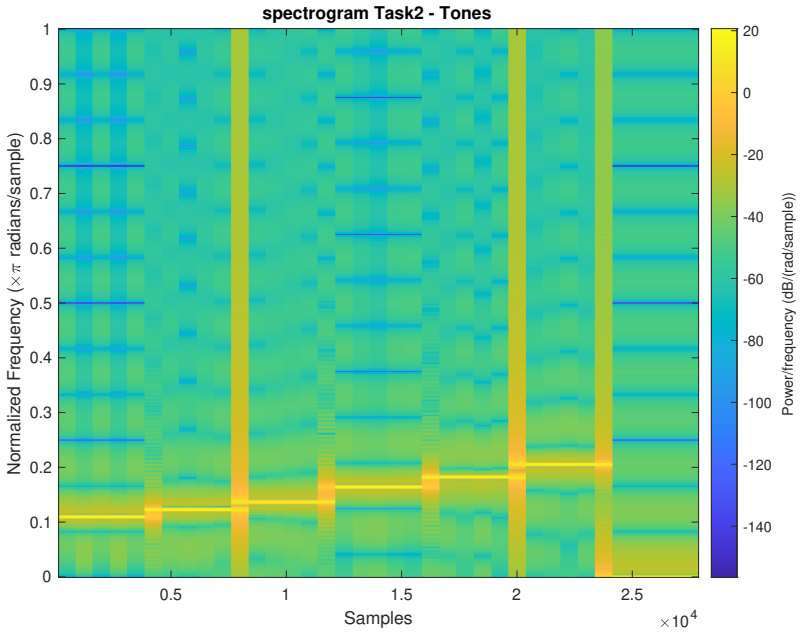
```
------------------------------MATLAB CODE:----------------------------------------
listOfNotes = [440,495,550,660,733,825,0];
Fs = 8e3;
durationOfEachNote= (Fs * 0.5);
resultingTimeDomainSignal = generate_melody(durationOfEachNote,Fs,listOfNotes);
----------------------------------------------------------------------------------
------------------------------FUNCTION CODE:--------------------------------------
function newSignal = generate_melody(duration,Fs,ListOfNotes)
    b = [];
    for v = ListOfNotes
        r = generate_sigs(v,Fs,duration);
        r = transpose(r);
        b=[b;r];
    end
    newSignal = b;
end
----------------------------------------------------------------------------------
------------------------------FUNCTION CODE:--------------------------------------
function x = generate_sigs(f,Fs,L)
    t = 0:1/Fs:L/Fs - 1/Fs ;
    x = cos(2*pi*f*t);
end
----------------------------------------------------------------------------------
```

# 3 Task 2

The spectrogram shows a time-frequency representation of the signal generated in Task 1. We can see the frequency of notes in the melody in the spectrogram. The x-axis represents time and the y-axis represents frequency. This validates the implementation because the resulting time domain signal generated in Task one shows the correct frequency of notes in the spectrogram.

```
------------------------------MATLAB CODE:------------------------------
window = 1000;
y = resultingTimeDomainSignal;
spectrogram(y,window,(0.25*window),'yaxis')
title("spectrogram Task2 - Tones")

print -depsc spectrogramTask2
------------------------------------------------------------------------
```

### Spectrogram

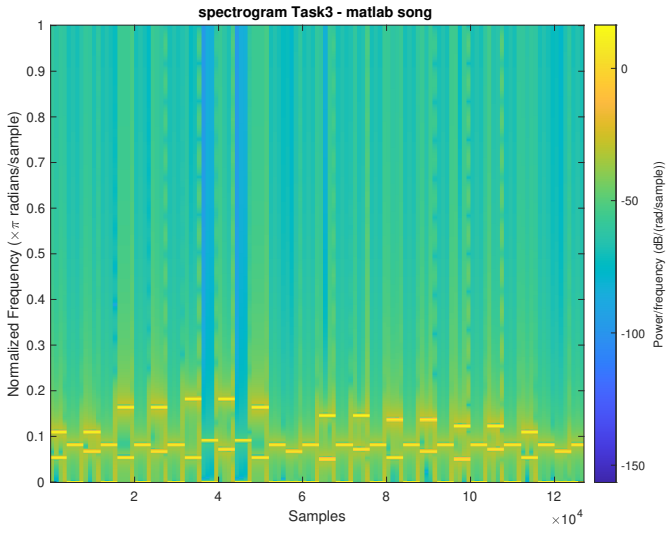


# 4 Task 3

The spectrogram x-axis represents time and y-axis represents frequency. The spectrogram confirms the implementation because we can see the frequency components of the chorus and melody in the song signal stored in variable x. The resulting signal is written to a WAV file and we can clearly hear Twinkle Twinkle Little Star. The chorus and melody notes can be seen in the spectrogram.

```
-------------------------------------------------MATLAB CODE:---------------------------
tableTwo = [440,0,440,0,660,0,660,0,733,0,733,0,660,0,0,0,587,0,587,0,550,0,550,0,495,0,495,
tableThree = [220,330,275,330,220,330,275,330,220,367,293,367,220,330,275,330,206,330,293,33

Xmelody = generate_melody(durationOfEachNote,Fs,tableTwo);
Xchorus = generate_melody(durationOfEachNote,Fs,tableThree);

xm = 0.6*(Xmelody)+0.4*(Xchorus);

spectrogram(xm,1024,24,'yaxis')
title("spectrogram Task3 - matlab song")
print -depsc spectrogramTask3

fileName1 = "Task3AudioFile.wav";
audiowrite(fileName1,xm,Fs);

[y, Fs] = audioread('Task3AudioFile.wav');
sound(y,Fs);
-------------------------------------------------------------------------------------------
```

### Spectrogram

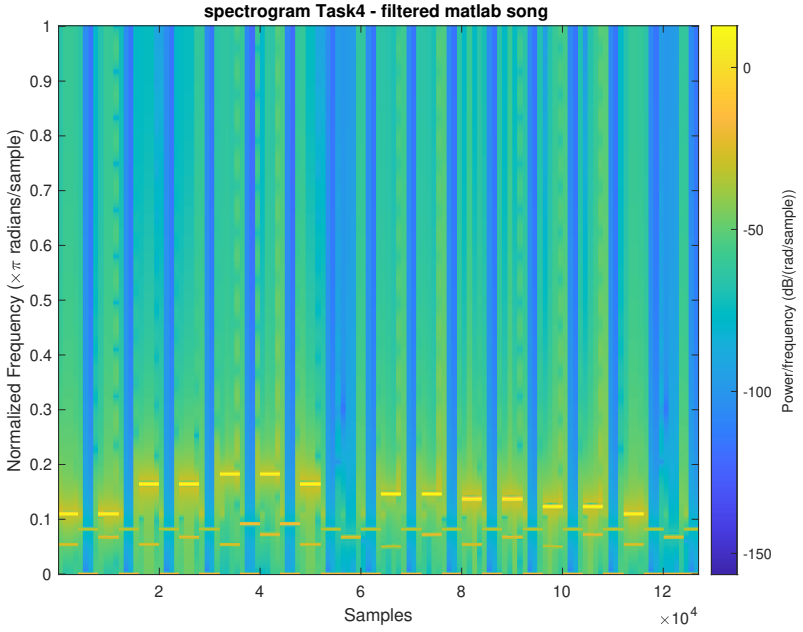


# 5 Task 4

The results of the filtering process are stored in variable xf. When we write xf to a WAV file we can hear Twinkle Twinkle Little Star but the chorus notes are attenuated because the song was passed through a high pass IIR filter. We can see the results of the filtering in the spectrogram. The lower frequency components are attenuated out.

```
------------------------------MATLAB CODE-------------------------------
a = [0.62477732,-2.444978,3.64114,-2.444978,0.62477732];
b = [1,-3.1820023,3.9741082,-2.293354,0.52460587];

xf = iir_filter(x,a,b);
spectrogram(y,1024,24,'yaxis');
title("spectrogram Task4 - filtered matlab song")
print -depsc spectrogramTask4

fileName2 = "Task4AudioFile.wav";
audiowrite(fileName2,xf,Fs);

[b, Fs] = audioread('Task4AudioFile.wav');
sound(b,Fs);
------------------------------------------------------------------------
```

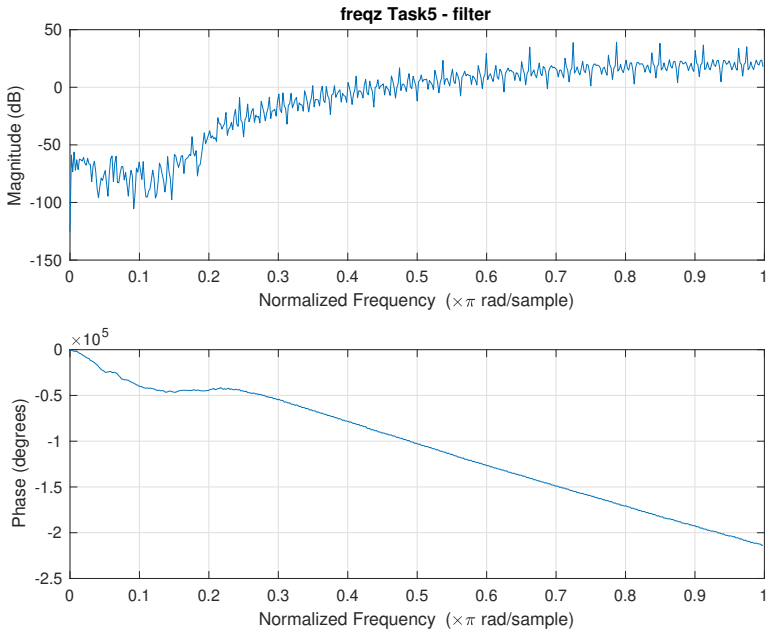
### Spectrogram



# 6 Task 5

Use MATLAB's freqz() function to generate a magnitude response plot of the IIR filter from Task 4. We can see that this is a high pass filter. This allows high frequency components of the song to pass but attenuates lower frequency components. This directly affects the chorus notes which are all 400Hz and below by making the amplitude smaller, thus making the notes sound weaker.

```
------------------------------MATLAB CODE-----------------------------
freqz(a,b);
title("freqz Task5 - filter")
print -depsc freqzTask5
----------------------------------------------------------------------------
```

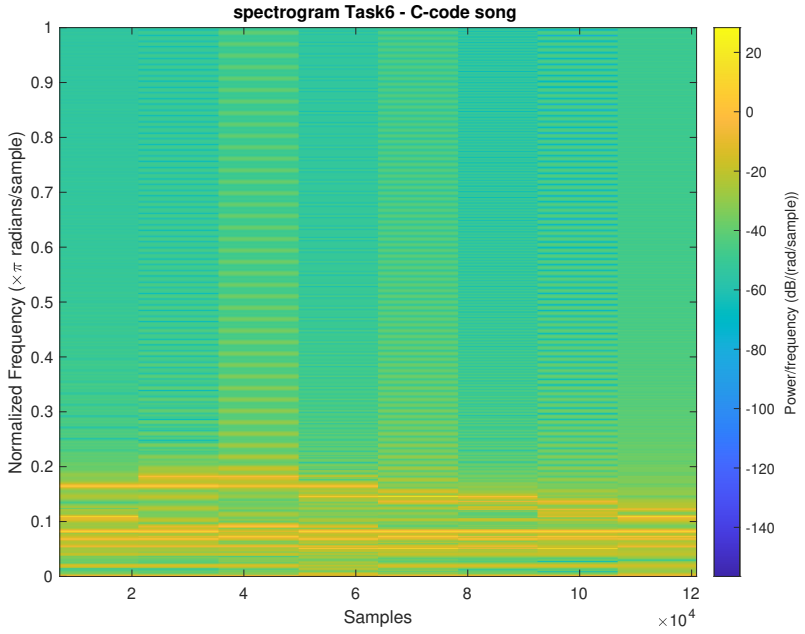


# 7 Task 6

In Task 6 C code language is used to replicate Task 3 MatLab code. This C code produces a song with the same notes used in Task 3. The song produced in C is written to a WAV file and examined in MatLab using spectrogram. The spectrogram shows the notes that make up the chorus and melody of the Twinkle Twinkle Little Star song.
l2 error is 3.6201e-05.

```
------------------------------MATLAB CODE------------------------------
[xc,Fs] = audioread("song.wav");
sound(xc,Fs)
spectrogram(xc,'yaxis')
title("spectrogram Task6 - C-code song")
print -depsc spectrogramTask6
l2 = norm(xm - xc)/norm(xm);
l2
-----------------------------------------------------------------------
```

See Appendix for C-Code:
**Spectrogram**

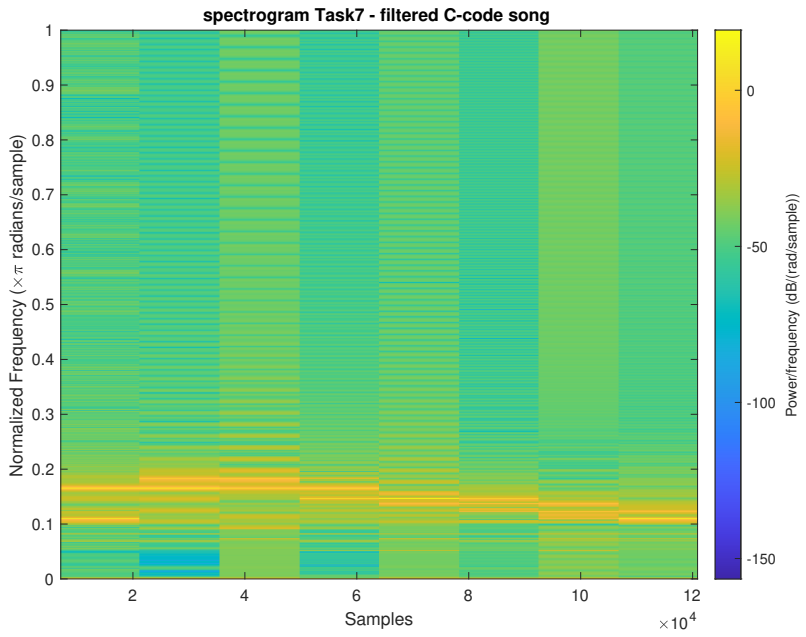


# 8 Task 7

The IIR filtering is implemented in C using the same coefficients from Task 4.
The results of the filtered song is written to a WAV file.
The WAV file is read into MATLAB and compared with your MATLAB result from Task 4 using normalized l2f error as the distance measure.
The spectrogram shows that song after being filtered with the IIR C-code filter, and we can see the lower frequency components are attenuated.
l2f error is unknown because my system could not handle the size of the number.

```
------------------------------MATLAB CODE------------------------------
[xcf,Fs] = audioread("filtered.wav");
sound(xcf,Fs)
spectrogram(xcf,'yaxis')
title("spectrogram Task7 - filtered C-code song")
print -depsc spectrogramTask7
l2f = norm(xf - xcf)/norm(xf);
l2f
-------------------------------------------------------------------------
```

Please see Appendix for C-code and IIR filter function.
**Spectrogram**



# 9 Appendix

```
-----------------------------------------C CODE-----------------------------------------
#include <stdlib.h>
#include <stdio.h>
#include <float.h>
#include <sndfile.h>
#include <math.h>

#define PI 3.14159265
#define Fs 8000

// Declare function
void generate_note(float f, float L, float* x);
void iir_filter(float* x,float* a, float* b,float* y);

int main(int argc, char *argv[])
{

//Require 2 arguments: input file and output file
if(argc < 2)
{
printf("Not enough arguments \n");
return -1;
}

SF_INFO sndInfoOut;;
sndInfoOut.format = SF_FORMAT_WAV | SF_FORMAT_PCM_16;
sndInfoOut.channels = 1;
sndInfoOut.samplerate = Fs;
 // file for complete song x
SNDFILE *sndFileOut = sf_open(argv[1], SFM_WRITE, &sndInfoOut);
// file for iir filtered song
    SNDFILE *sndFileOut2 = sf_open(argv[2], SFM_WRITE, &sndInfoOut);

    // start here

// load parameters and coefficients
    float tableTwo[] = {440,0,440,0,660,0,660,0,733,0,733,0,660,0,0,0,587,0,...
    587,0,550,0,550,0,495,0,495,0,440,0,0,0};
    float tableThree[] = {220,330,275,330,220,330,275,330,220,367,293,367,...
    220,330,275,330,206,330,293,330,220,330,275,330,206,330,293,330,220,330,275,330};
    float a[] = {0.62477732,-2.444978,3.64114,-2.444978,0.62477732}; //coef for iir filter
    float b[] = {1,-3.1820023,3.9741082,-2.293354,0.52460587}; //coef for iit filter
float t = 0.5;
float L = t * Fs; //4000
```

```
//Melody array = x_m[32*4000] this will initialize an empty array with 128000 spaces
float x_m[128000];
    //Chorus array = x_c[32*4000] this will initialize an empty array with 128000 spaces
    float x_c[128000];
    // Complete song = x[32*4000]
    float x[128000];
    //iir filtered song = y[32*4000]
    float y[128000];


for(int i = 0;i < 32;i++)
    {
    //this will build x_m
        generate_note(tableTwo[i],L,&(x_m[(int) L*i]));
        //this will build x_c
        generate_note(tableThree[i],L,&(x_c[(int) L*i]));
    }

//write the composed song to a WAV file
for(int j=0;j < 128000;j++)
    {
        // build the complete song x[] array now that you have x_m and x_c
    x[j] = (0.6*x_m[j]) + (0.4*x_c[j]);
        //write the complete song x[] to a file now
        sf_writef_float(sndFileOut, &(x[j]), 1);
    }

//call the iir filter to filter the full song x
    iir_filter(x,a,b,y);

//write the filtered song to a WAV file
    for(int k=0;k < 128000;k++)
    {
        //write the filtered song y[] to a file now
        //this will write to a diff WAV file than the one x[] went to
        sf_writef_float(sndFileOut2, &(y[k]), 1);
    }

sf_write_sync(sndFileOut);
sf_close(sndFileOut);

    sf_write_sync(sndFileOut2);
    sf_close(sndFileOut2);

// free(x); not needed bc i used static variable
```

```
// free(x_m);
// free(x_c);

return 1;
}

// Implement function
// This fucntion is like the generate_sigs(f,Fs,L) function in our MatLab code
void generate_note(float f, float L, float* x) {
    for (int i = 0; i < L; i++)
    {
        x[i] = cos(2*PI*f*((float)i/Fs));
    }
}


void iir_filter(float* x,float* a, float* b,float* y) {
    y[0] = (a[0]*x[1])/b[0];
    y[1] = (a[0]*x[2] + a[1]*x[2-1] - b[1]*y[2-1])/b[0];
    y[2] = (a[0]*x[3] + a[1]*x[3-1] + a[2]*x[3-2] - b[1]*y[3-1] - b[2]*y[3-2])/b[0];
    y[3] = (a[0]*x[4] + a[1]*x[4-1] + a[2]*x[4-2] + a[3]*x[4-3] - b[1]*y[4-1] ...
    - b[2]*y[4-2] - b[3]*y[4-3])/b[0];
    for(int n = 4; n < 128000; n++)
    {
        y[n] = (a[0]*x[n] + a[1]*x[n-1] + a[2]*x[n-2] + a[3]*x[n-3] + a[4]*x[n-4]...
        - b[1]*y[n-1] - b[2]*y[n-2] - b[3]*y[n-3] - b[4]*y[n-4]) / b[0];
    }

}
```

----------END C-CODE----------

```
------------------FUNCTION CODE: iir_filter(x,a,b)---------------------
function filtered = iir_filter(x,a,b)
    b0 = b(1);
    b1 = b(2);
    b2 = b(3);
    b3 = b(4);
    b4 = b(5);
    a0 = a(1);
    a1 = a(2);
    a2 = a(3);
    a3 = a(4);
    a4 = a(5);
    y = [];
    %calc the first 4 terms
    y(1) = (a0*x(1))/b0;
    y(2) = (a0*x(2) + a1*x(2-1) - b1*y(2-1))/b0;
    y(3) = (a0*x(3) + a1*x(3-1) + a2*x(3-2) - b1*y(3-1) - b2*y(3-2))/b0;
    y(4) = (a0*x(4) + a1*x(4-1) + a2*x(4-2) + a3*x(4-3) - b1*y(4-1)...
    - b2*y(4-2) - b3*y(4-3))/b0;

    for n = (1+4):(length(x))
        y(n) = (a0*x(n) + a1*x(n-1) + a2*x(n-2) + a3*x(n-3) + a4*x(n-4)...
        - b1*y(n-1) - b2*y(n-2) - b3*y(n-3) - b4*y(n-4)) / b0;
    end
    filtered = y;
end
------------------------------------------------------------------------

-------FUNCTION CODE: generate_melody(duration,Fs,ListOfNotes)------------
function newSignal = generate_melody(duration,Fs,ListOfNotes)
    b = [];
    for v = ListOfNotes
        r = generate_sigs(v,Fs,duration);
        r = transpose(r);
        b=[b;r];
    end
    newSignal = b;
end
------------------------------------------------------------------------
--------------FUNCTION CODE: generate_sigs(f,Fs,L)-----------------------
function x = generate_sigs(f,Fs,L)
    t = 0:1/Fs:L/Fs - 1/Fs ;
    x = cos(2*pi*f*t);
end
------------------------------------------------------------------------
```