Pràctica 5: Síntesi musical

Anna Falceto Piñol

L'objectiu d'aquesta pràctica era el de sintetitzar diferents instruments virtuals a partir diverses tècniques de síntesi musical. En l'enunciat de la pràctica ens presentaven la síntesi per taula i la síntesi FM.

Com a ampliació i amb la finalitat d'explorar altres mètodes s'ha implementat instruments amb síntesi additiva i la síntesi per taula externa.

Tots els fitxers ***.wav*** *sintetitzats que s'aniran anomenant a la pràctica es poden trobar ordenats en la carpeta "àudios_memòria".*

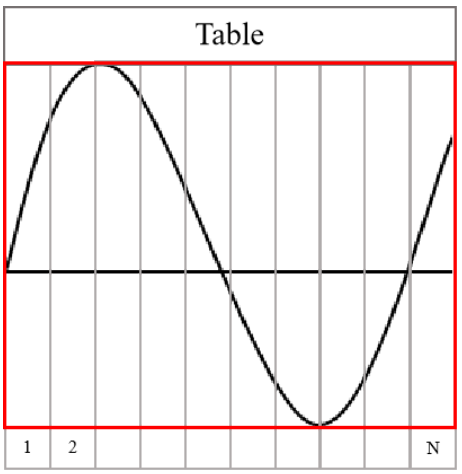
1) Síntesi per taula

El primer instrument que hem implementat s'ha creat mitjançant síntesi per taula. Aquest tipus de síntesi consisteix a crear una taula que contingui els valors d'un sol període d'una ona sinusoidal i en reproduir-la a diferents velocitats per tal de sintetitzar les diferents notes.

S'ha implementat l'instrument "seno". Per fer-ho s'ha generat, en el **constructor** de l'instrument, una taula amb l'ona sinusoidal:

```
seno::seno(const std::string
&param):adsr(SamplingRate, param) {
  bActive = false;
  x.resize(BSIZE);
  KeyValue kv(param);
  int N;
  [...]
  tbl.resize(N);
  float phase = 0, step = 2 *
  M_PI /(float) N;
  index = 0;
  for (int i=0; i < N ; ++i) {
    tbl[i] = sin(phase);
    phase += step;
  }
```

on l'argument del sinus és el valor 2π normalitzat pel nombre d'elements de la taula. D'aquesta manera ens podem imaginar la taula com un seguit de valors que representen un període de l'ona.



Imatge 1: Representació de la taula amb l'ona sinusoidal

Per tal de sintetitzar les notes de la partitura és necessari recórrer la taula a una velocitat proporcional a la nota requerida.

Les notes de la nostra partitura prenen valors en el rang [0, 127] i segueixen la següent expressió:

$$Note = 69 + 12 * log_2(\frac{f0}{440})$$

La freqüència fonamental de la nota es pot expressar de la següent manera:

$$f0 = 440 * 2^{\frac{(Note-69)}{12}}$$

El concepte "recórrer la taula a la velocitat requerida" es pot entendre com el resultat d'accedir a determinats índexs de la taula. Per tant, a partir de la freqüència fonamental de la nota hem de definir un pas que ens permeti recórrer la taula de la forma adequada.
Aquests pas el definirem al mètode **command**, en el que prèviament haurem transformat de freqüència MIDI en freqüència fonamental.

```
void seno::command(long cmd, long note,
long vel) {
  if (cmd == 9) {
    bActive = true;
    adsr.start();
    index = 0;
        A = vel / 127.;
    [...]
   }
  float F0 =
(pow(2,(note-69)/12.)*440)/SamplingRate
   step_recorregut_aux=F0*tbl.size();
}
```

Finament, en el mètode **synthetise** recorrem la taul amb el pas definit anteriorment.

```
const vector<float> & seno::synthesize()
{
[...]
for (unsigned int i=0; i<x.size(); ++i){
    index_recorregut +=
step_recorregut_aux;
 if(index_recorregut>=tbl.size()){ (1)
      index_recorregut =
index_recorregut - tbl.size();
    }
    x[i] = A *
tbl[(int)index_recorregut];
    if (index == tbl.size())
      index = 0;
  }
  adsr(x);

  return x;
}
```

Cal tenir en compte que amb l'*step* definit poder acabar de recórrer la taula abans que finalitzi l síntesi de la nota. És per això que cal reiniciar l'índe en el cas que aquest superi la dimensió de la taula. (1

Per afegir aquest instrument al sistema cal modifica tres fitxers:

a) Incorporar l'instrument a **src/meson.build**.
b) Crear el fitxer **seno.h** associat al **seno.cpp**.
c) Modificar **instrument.cpp** per afegir la invocació a l'instrument **seno.** [1]

Finalment, creem un fitxer **seno.orc** per tal de sintetitzar el senyal.

```
1       seno    ADSR_A=0.02; ADSR_D=0.1;
ADSR_S=0.4; ADSR_R=0.1; N=40;
```

El resultat de la síntesi es troba en el fitxer ***doremi_seno.wav***

2) Síntesi per taula externa

En aquest segon tipus de síntesi extrapolarem la síntesi implementada anteriorment i utilitzarem com a taula un període de senyal d'un instrument ja enregistrat.

Explicarem la implementació amb l'instrument **"guitar".**

Per tal de produir la síntesi és necessari disposar d'un senyal enregistrat d'una guitarra real. Per fer-ho hem fet una cerca a la pàgina web freesound.org i ens hem descarregat el fitxer **guitar.wav** que conté la nota do d'una guitarra.

Imatge 2: portal web freesound.org

Una vegada obtingut el senyal és necessari que sigui mono. Si no, s'haurà de convertir. Per fer-ho hem fet ús de l'eina online fconvert.com.

A l'enunciat de la pràctica se'ns proposava fer ús de la funció **readwav_mono** per carregar el senyal i emmagatzemar-lo en la taula. Tot i això, la implementació amb aquesta funció donava

[1] Veure en Annex el resultat dels fitxers.

problemes, ja que no retornava una taula amb e contingut correcte del senyal.

És per això que s'ha optat per fer ús de **MATLAB** el programa **conversio_wav.m** extreu els camps d la capçalera **.wav** que no s'han de guardar en l taula i guarda el valor del senyal en un fitxer **.txt**.

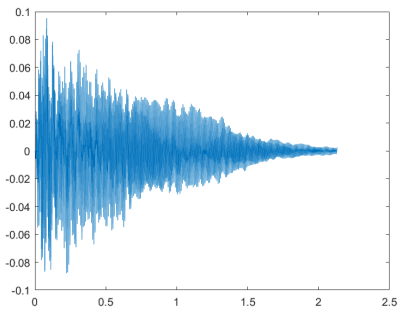
Un fitxer **.txt** és molt més fàcil d'introduir al cod **cpp** i MATLAB és una eina que agrada molt a l implementadora del projecte.

```
instruments = dir('*.wav');
for i=1:length(instruments)
    cd('[...]');
    [y,Fs] =
audioread(instruments(i).name); (1)
    sound(y,Fs)
    name =
strsplit(instruments(i).name,'.');
    nom = string(name(1));
    cd('[...]');
writematrix(y,strcat(nom,'.txt')); 
(2)
end
```

(1) La funció `audioread` retorna els valor del senyal sense la capçalera.
(2) La funció `writematrix` permeti guarda els valors del senyal en un fitxer **.txt**

Si visualitzem el senyal **guitar.wav** observem l següent forma d'ona:

```
t=(1:1:length(y));
plot(t, y.');
```



Imatge 3: ***guitar.wav***

Una vegada obtingut el fitxer **guitar.txt** qu caracteritza la guitarra l'hem carregat en e constructor de l'instrument.

```
float aux;
ifstream myReadFile("guitar.txt");
if (!myReadFile) {
 cerr << "Unable to open file
 datafile.txt";
 exit(1);   // call system to stop
}
while (!myReadFile.eof()) {
    myReadFile >> aux;
    tbl.push_back(aux);
 }
 myReadFile.close();
```

També s'ha modificat per poder introduir com a paràmetres les variables ADSR.

Una vegada tenim la taula carregada cal pensar amb quina velocitat (amb quin pas) s'ha de recórrer.

Sabem que la nostra ona sinusoidal fonamental (sintetitzada en l'apartat 1) té una freqüència de $f0 = \frac{2\pi}{40}$. En el nostre cas, en tractar-se de la nota "C4", $f0 = 261.626\,Hz$. Perquè el sistema funcioni amb el mateix sistema de notes amb el que funcionava a l'apartat anterior necessitem que la freqüència fonamental de la guitarra sigui $\frac{2\pi}{40}$.

$$f0 = 261.626 * x = \frac{2\pi}{40} \rightarrow x = \frac{2\pi}{40 * 261.626}$$

Per tenir en compte aquest factor s'ajusta el mètode **command**:

```
void guitar::command(long cmd, long
note, long vel) {
    [...]
float F0 =
(2*M_PI/40*261.626))*(pow(2,(note-69)/1
2.)*440)/SamplingRate; (1)
step_guitar=F0;
}
```

Com que la taula s'ha creat sense transformar la freqüència fonamental amb la freqüència de mostreig no cal tenir-la en compte en l'step.

(1) El factor 261.626 pot veure's modificat pel seu valor doble (523.251 → C5) o mig (130.831 → C4) en el cas que s'hagi volgut

transportar l'instrument a una octav superior o inferior.

El mètode **synthetise** no es modifica respecte l'instrument **seno.cpp.**

A part de l'instrument **guitarra.cpp** s'h implementat amb el mateix mètode l'instrumen **bass.cpp** i **piano.cpp.**

Els instruments queden registrats als fitxers **.orc.**

```
1 Piano ADSR_A=0; ADSR_D=0;
ADSR_S=0; ADSR_R=0;
```

```
1 Bass ADSR_A=0.1; ADSR_D=4;
ADSR_S=6; ADSR_R=0.2;
```

```
1 Guitar ADSR_A=0.1; ADSR_D=4;
ADSR_S=6; ADSR_R=0.2;
```

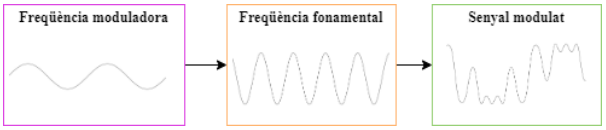
En aquest cas la corba ADSR no té gaire efecte e el resultat, ja que els sons agafats com a base per le taules ja contenen les característiques d'atac manteniment i caiguda. Això es veu clarament en e cas del piano, on no cal configurar l'atac pe sentir-lo.

Podem escoltar els resultats en els fitxers:

- ***doremi_bass.wav.***
- ***doremi_guitar.wav.***
- ***doremi_piano.wav.***

3) Síntesi FM

El tercer tipus d'instrument s'ha implementa mitjançant síntesi FM. Aquesta es basa en l modulació de l'ona de freqüència fonamenta mitjançant una ona de baixa freqüència.



Imatge 4: diagrama de modulació FM

En aquest instrument deixem enrere la construcció del senyal mitjançant una taula i implementem l'ona seguint l'expressió:

$$x(t) = A\,sin(2\pi f_c + I\,sin(2\pi f_m t))$$

La freqüència moduladora es pot expressar en funció de la freqüència fonamental mitjançant l'expressió:

$$f_0 = f_m * \frac{N1}{N2} \rightarrow f_m = f_0 * \frac{N1}{N2}$$

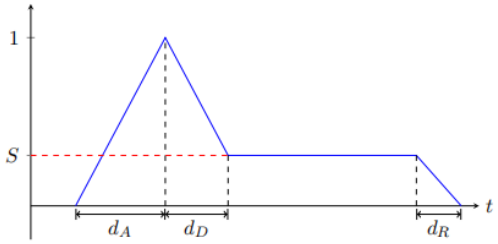
Per implementar aquesta ona en el nostre instrument modifiquem els mètodes **command** i **synthetise**.

```
void instrumentFM::command(long cmd,
long note, long vel) {
[...]
F0 =
(pow(2,(note-69)/12.)*440)/SamplingRate
  fm = N2*F0/N1;
  alpha_c =  2*M_PI*F0;
  alpha_m =  2*M_PI*fm;
}
```

```
const vector<float> &
instrumentFM::synthesize() {
[...]
for (unsigned int i=0; i<x.size(); ++i)
{
x[i] = A*sin(phase_c + I*sin(phase_m));
phase_c = phase_c + alpha_c;
phase_m = phase_m + alpha_m;

while(phase_c > M_PI) phase_c = phase_c
- 2*M_PI;
while(phase_m > M_PI) phase_m = phase_m
- 2*M_PI;
}
}
```

Amb l'objectiu de tenir fer els instruments més realistes hem tingut en compte la corba ADSR.



Imatge 5: Corba ADSR (enunciat pràctica)

Per fer-ho hem modificat el constructor d **l'instrumentFM**:

```
instrumentFM::instrumentFM(const
std::string param)
: adsr(SamplingRate, param) {
  bActive = false;
  x.resize(BSIZE);
  fase = 0; //modificat
 KeyValue kv(param);
 float a,d,s,r;
 if (!kv.to_float("N1", N1))
   N1 = 10;
 if (!kv.to_float("N2", N2))
   N2 = 200;
 if (!kv.to_float("I", I))
    I = 1;
 if (!kv.to_float("fm", fm))
    fm = 10; //default value

 //Paràmetres ADSR
 if (!kv.to_float("ADSR_A", a))
   a = 1.0; //default value
 if (!kv.to_float("ADSR_D", d))
  d = 2.0; //default value
 if (!kv.to_float("ADSR_S", s))
  s = 4.0; //default value
 if (!kv.to_float("ADSR_R", r))
  r = 0.0; //default value
}
```

Com es pot observar s'ha incorporat els paràmetre **N1** i **N2** com a arguments a configurar al fitxer **.or** de l'instrument.
S'han provat diferents configuracions dels anterior paràmetres s'ha sintetitzat un instrument *similar* una flauta.

```
1  Fm  N1=13;  N2=80; I=0.2;  ADSR_A=0.7;
ADSR_D=3; ADSR_S=6; ADSR_R=0.01;
```

Altres combinacions dónen lloc a sons molt més exotèrics. Per exemple:

```
1  Fm  N1=33;  N2=80; I=1;  ADSR_A=0.7;
ADSR_D=3; ADSR_S=6; ADSR_R=0.01;
```

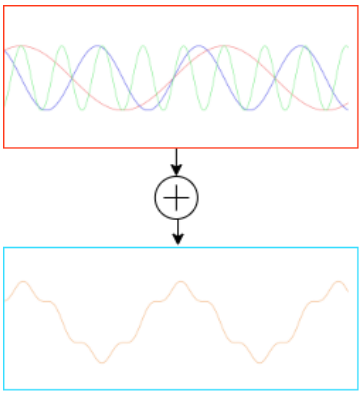
Podem escoltar els resultats en els fitxers:

- ***doremi_flauta.wav***
- ***doremi_exoteric.wav***

4) Síntesi additiva

Finalment, s'ha volgut explorar un últim mètode de síntesi que no es presentava a l'enunciat de la pràctica; la síntesi additiva.

Aquest tipus de síntesi consisteix a crear una ona a base de la suma de diferents sinus a diferents freqüències.



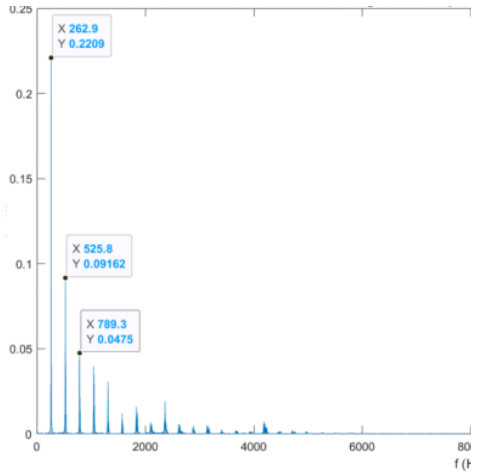
Imatge 6: Síntesi additiva

El nostre propòsit és crear un violí. Per fer-ho necessitem saber a quines freqüències es troben els seus harmònics i quina amplitud tenen. Per trobar-ho hem analitzat, mitjançant MATLAB la Transformada de Fourier d'un àudio que correspon a la nota do tocada per un violí[2]:

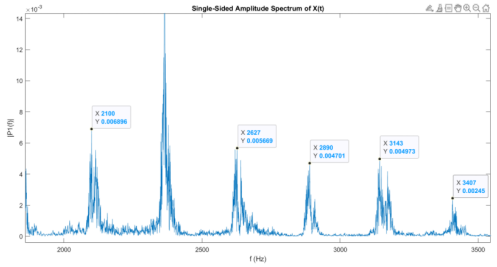
[2] Totes les mostres d'instruments reals utilitzades es troben a la carpeta Samples/C_instruments

violin_analisi.m:

```
[y,Fs] = audioread('Violin.wav');
L=length(y);
Y = fft(y);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;
plot(f,P1)
```



*Imatge 7: Transformada de Fourier de **violí.wav***



*Imatge 8: Zoom de la transformada de Fourier de **violí.wav** a freqüències altes.*

Trobem els següents harmònics amb les següent amplituds:

	Frequency [Hz]	Amplitude
Fonamental	262.9	0.2209
2nd	525.8	0.09162
3rd	788.2	0.04479
4th	1047	0.03961
…		
15th	4192	0.007382

Crearem l'ona seguint la següent expressió:

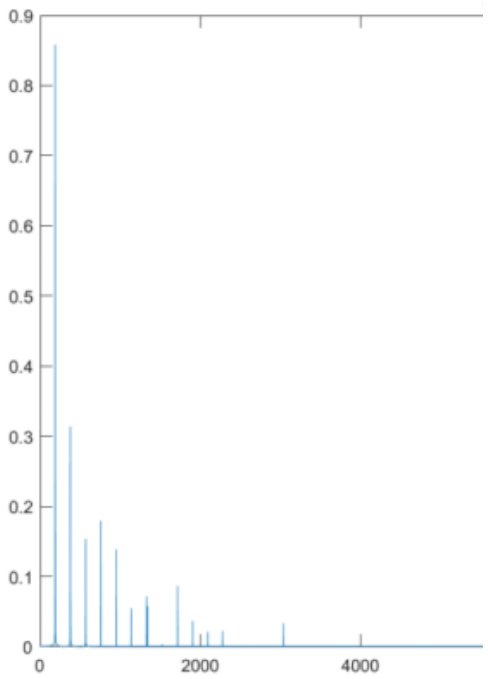
$$X(t) = \sum_{i=1}^{num\ harmònics} \frac{amplitud\ (i)}{amplitud\ (1)} * sin(f_i)$$

Per comprovar que la síntesi és correcta hem implementat l'ona al MATLAB.

```
sum = 0;
for i=1:15
  [sine,t] =
   sinus(harmonics(i),3); (1)
   sum = sum +
  (amplitudes(i)/amplitudes(1))*
   sine;
end
writematrix(sum.','violin_synth.txt'
);
audiowrite('violin_synth.wav', sum,
Fs);
```

(1) La funció **sinus** és una funció auxiliar que ens retorna un sinus a la freqüència i la duració temporal especificada. Veure el codi en el fitxer **sinus.m**.

Si representem la transformada de l'ona creada, obtenim el següent:



Imatge 9: Transformada de Fourier del violí sintètic.

Podem veure el resultat a ***violin_synth.wav***

Veiem que el so, tot i assemblar-se a un violí, qued bastant sintètic. Això és a causa del fet que falte els alguns harmònics aguts, que són els que done "brillantor" a l'instrument.

Una vegada tenim l'anàlisi dels harmònics i l'on creada introduïm aquesta ona sintètica al program **violin.cpp**, com havíem fet en el cas de la síntes per taula externa.

```
violin::violin(const std::string&param)
  : adsr(SamplingRate, param) {
[...]
float aux;
ifstream
myReadFile("violin_synth.txt");
if (!myReadFile) {
  cerr << "Unable to open file
datafile.txt";
  exit(1);   // call system to stop
}
while (!myReadFile.eof()) {
  myReadFile >> aux;
  tbl.push_back(aux);
}
  myReadFile.close();
}
```

En aquest últim instrument i per tal de millorar l qualitat del so hem implementat el recorregut de l taula interpolant[3]. Fins ara accedíem al valor ente de l'índex corresponent:

```
tbl[(int)index_recorregut];
```

Ara tindrem en compte el valor decimal de l'índe per tal de fer ús del següent índex de maner proporcional. Per exemple, si index = 1.3 :

- index_violin = 1
- index_next = index_violin + 1 = 2;

x[i] = 1* tbl[index_violin] + 0.3* tbl[index_next].

[3] Aquesta modificació es podria incloure en els instruments sintetitzats per taula externa, però no ha donat temps.

El mètode **synthesize** es modifica de la següent manera:

```
const vector<float> &
violin::synthesize() {
[...]
for (unsigned int i=0; i<x.size(); ++i)
{
if(index_violin>=tbl.size()){
  index_violin = index_violin -
tbl.size();
 }
index_violin += step_violin;
float coef_inter = index_violin -
(int)index_violin;
int index_violin_next = 0;
if((int)index_violin +1 == tbl.size()){
  index_violin_next = 0;
} else{
  index_violin_next = (int)index_violin
+1;
}
x[i] = coef_inter * A *
tbl[(int)index_violin] +
(1-coef_inter)* A *
tbl[index_violin_next];

if (index_violin == tbl.size())
        index_violin = 0;
[...]
}
```

Finalment, per tal de modelar l'envolvent ADRS hem tornat a observar mitjançant **wavesurfer** el fitxer **violin.wav.** *(Vegeu captura en annex).*

```
1        Violin   ADSR_A=0.15;
ADSR_D=0.45; ADSR_S=1; ADSR_R=0.05;
```

Podem escoltar els resultats al fitxer:

- ***doremi_violin.wav***

5) Reproducció d'una partitura

La millor forma de provar tots els instruments sintetitzats és reproduint una partitura.

Per fer-ho hem buscat una partitura de la cançó popular francesa “Frère Jaques” al portal web MuseScore[4]

La partitura descarregada del lloc web només teni la línia principal de melodia tocada amb un so instrument. Per fer participar tota la nostra orquestr l’hem editada i hem creat un cànon.

Imatge 10: Cànon “Frère Jaques”; fitxer ***Frere_Jaques.mscz***

Exportem la partitura a mid (**work/frere_jaques_tutti.mid**) i amb la funci mid2sco la transformem a un **.sc** (**work/frere_jaques.sco**)

```
annafalpi@ASUS-ANNA:~/PAV/P5$ midi2sco work/frere_jaques_tutti.m
id work/frere_jaques_tutti.sco
Usando bpm=120 pulsos por minuto y tpb=120 ticks por pulso. Se r
ecomienda
usar estos mismos valores al reproducir work/frere_jaques_tutti.
sco con el programa 'synth'.

El fichero work/frere_jaques_tutti.mid incluye 6 instrumento(s):
        0
        1
        2
        3
        4
        5
Consulte el fichero work/frere_jaques_tutti.sco para ver si prop
orciona información de los mismos.
```

Veiem que el cànon es pot tocar amb 6 instruments.

Creem el fitxer .orc que definirà la nostra orquestra

```
5   seno    ADSR_A=0.02; ADSR_D=0.1; ADSR_S=0.4; ADSR_R=0.1; N=40;
4   Bass   ADSR_A=0.1; ADSR_D=4; ADSR_S=6; ADSR_R=0.0;
3   Guitar   ADSR_A=0.1; ADSR_D=4; ADSR_S=6; ADSR_R=0.0;
0   Piano   ADSR_A=0.1; ADSR_D=0; ADSR_S=0; ADSR_R=0;
1   Fm  N1=13;  N2=80; I=0.2;  ADSR_A=0.7; ADSR_D=3; ADSR_S=6; ADSR_R=0.01;
6   Fm  N1=33;  N2=80; I=1;  ADSR_A=0.7; ADSR_D=3; ADSR_S=6; ADSR_R=0.01;
2   Violin  ADSR_A=0.15; ADSR_D=0.45; ADSR_S=1; ADSR_R=0.05;
```

i sintetitzem el fitxer.**wav** resultant amb les ordres

```
annafalpi@ASUS-ANNA:~/PAV/P5$ synth work/orc/orquestra.orc work/
frere_jaques_tutti.sco work/wav_files/frere_tutti.wav
Register instrument: 5 seno
Register instrument: 4 Bass
Register instrument: 3 Guitar
Register instrument: 0 Piano
Register instrument: 1 Fm
Register instrument: 6 Fm
Register instrument: 2 Violin
```

Podem escoltar el resultat a ***frere_tutti.wav***

6) Conclusions

En aquest projecte ens hem centrat en l’exploració i cerca de mètodes de síntesi musical més o menys funcionals. Hem vist que alguns funcionen molt bé, com és el cas de síntesi per taula, tant interna com externa. Cal remarcar, però que la síntesi per taula externa no és ben bé síntesi absoluta, ja que es parteix d’una base ja enregistrada. És per això que s’ha volgut explorar l'opció de la síntesi additiva. Aquesta ha donat un resultat millor de l’esperat tot i que, com ja s’ha comentat, és millorable.

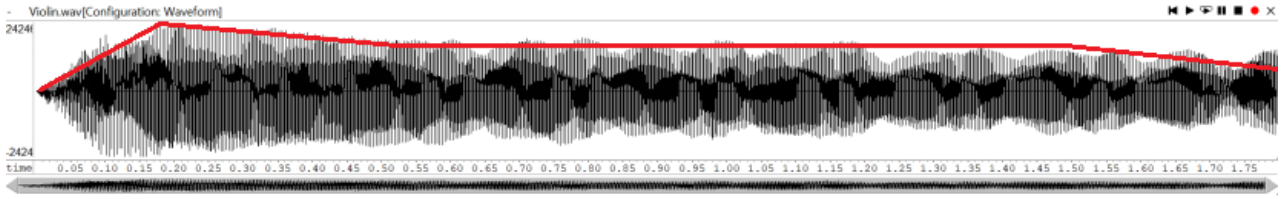
Una possible millora, de la síntesi additiva, passaria per crear l’ona en el mateix instrument programat en **cpp,** però no hi ha hagut temps. A més, es voldria destacar que l’anàlisi del violí s’ha fet amb una sola mostra, per tant, l’estudi dels harmònics i la relació d’amplituds és pobra. Es podria millorar la síntesi de l’ona estudiant diferents senyals i combinant les dades de les diferents anàlisis.

Pel que fa a la síntesi FM, tot i que sabem que és un mètode molt potent, no s’ha pogut experimentar gaire amb ell també per falta de temps. No s’ha tingut temps de profunditzar en la síntesi d’altres efectes.

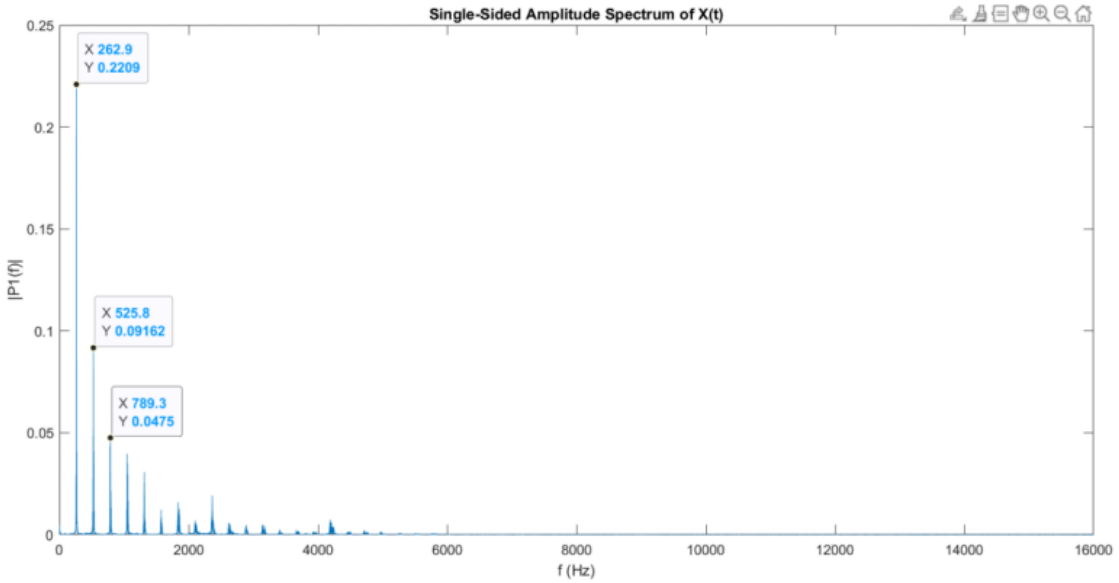
[4] Muscore és un programari lliure de creació i edició de partitures.

ANNEX

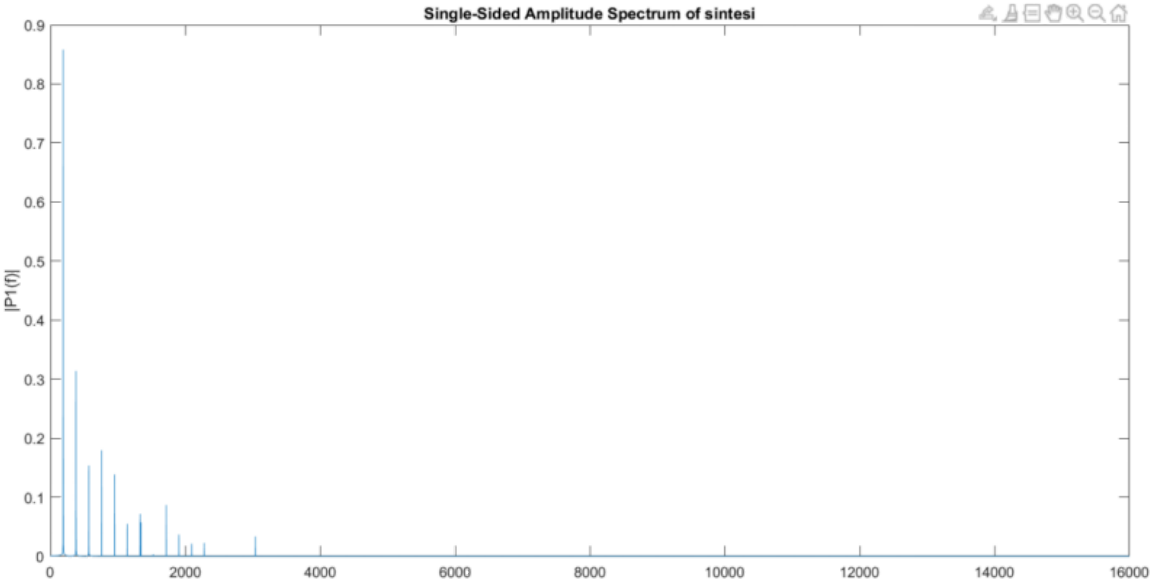
Envolvent ADSR violí:



Transformada de violí.wav



Transformada de violi_sinth.wav



meson.build

```
# Programas del proyecto

sources = [
       'instruments/instrument.cpp',
       'instruments/instrument_dumb.cpp',
       'instruments/seno.cpp',
       'instruments/instrument_FM.cpp',
       'instruments/piano.cpp',
       'instruments/guitar.cpp',
       'instruments/bass.cpp',
       'instruments/violin.cpp',
       'synth/envelope_adsr.cpp',
       'synth/midi_score.cpp',
       'synth/multinote_instr.cpp',
       'synth/orchest.cpp',
       'synth/synthesizer.cpp',
       'effects/effect.cpp',
       'effects/tremolo.cpp',
       'effects/vibrato.cpp',
]

executable(
    'synth',
    sources: sources,
    include_directories: inc,
    link_args: ['-lm', '-lsndfile'],
    link_with: lib_pav,
    install: true,
```

instrument.cpp

```
#include "bass.h"
#include "violin.h"

/*
  For each new instrument:
  - Add the header in this file
  - Add the call to the constructor in get_instrument() (also in this file)
  - Add the source file to src/meson.build
*/

using namespace std;

namespace upc {
  Instrument * get_instrument(const string &name,
                              const string ¶meters) {
    Instrument * pInst = 0;
    //    cout << name << ": " << parameters << endl;
```

```
    if (name == "InstrumentDumb") {
      pInst = (Instrument *) new InstrumentDumb(parameters);
    }
    if (name == "seno") {
        pInst = (Instrument *) new seno(parameters);
      }
    if (name == "Fm") {
      pInst = (Instrument *) new instrumentFM(parameters);
    }
    if (name == "Piano") {
      pInst = (Instrument *) new piano(parameters);
    }
    if (name == "Guitar") {
      pInst = (Instrument *) new guitar(parameters);
    }
    if (name == "Bass") {
      pInst = (Instrument *) new bass(parameters);
    }
    if (name == "Violin") {
      pInst = (Instrument *) new violin(parameters);
    }
  return pInst;
```