Síntesis musical polifónica

Antonio Bonafonte – profesores de la asignatura

diciembre de 2019

Resumen:

En esta práctica:

- Se estudiará un sistema sencillo de síntesis musical.
- Se entenderá qué son los mensajes MIDI mediante una notación simplificada.
- Se implementarán distintos instrumentos y efectos.
- Se introducirán los sintetizadores basados en Frecuencia Modulada.

Índice

1. Sintetizadores de audio para aplicaciones musicales.

Una definición habitual de sintetizador es: *instrumento musical electrónico que genera señales de audio*, aunque esta definición resulta, en muchos casos, bastante vaga. Los orígenes de la síntesis musical se encuentran en los primeros años del siglo XX, con la generalización de la electricidad y los circuitos electrónicos. Entre los primeros instrumentos que podemos definir como sintetizadores figuran inventos como el *Thelarmonium*, el *Trautonium* o, muy particularmente, el *Theremin*, instrumento diseñado por el gran inventor soviético Léon Theremin, de enorme complejidad para el intérprete, y que encontró amplia difusión en el mundo del cine, particularmente las películas de misterio, ciencia ficción y terror de los años 40 y 50 del siglo XX[1].

El concepto moderno de sintetizador, no obstante, surge en los años 60 con la comercialización del sintetizador modular Moog, diseñado por el ingeniero estadounidense Robert Moog. Este instrumento se basa en módulos independientes que realizan distintas funciones (generación de sinusoides, ruido blanco y otras formas básicas de onda; modulación de amplitud y de frecuencia; generación de envolventes; etc.), y que se enlazan mediante conectores tipo *jack*.

Los primeros sintetizadores resultaban extremadamente complicados de manejar, ya que cada sonido requería una cierta configuración de sus módulos y un modo concreto de interconectarlos. Además, todo el aparato, que podía ser muy voluminoso, sólo era capaz de producir una nota cada vez (*síntesis monofónica* o de una sola voz). En los años siguientes se desarrollaron toda una serie de tecnologías que permiten una mayor versatilidad y facilidad en la generación de los sonidos, y, además, permiten la síntesis polifónica.

Entre los muchos tipos de sintetizador disponibles hoy en día, se pueden destacar los siguientes (mencionados en orden no cronológico):

Síntesis aditiva: Basada en el análisis de Fourier, cada sonido (periódico) es construido sumando las sinusoides correspondientes a sus armónicos.

Síntesis sustractiva: Se parte de una forma de onda básica rica en armónicos (como las ondas cuadrada, triangular o en diente de sierra), y se filtra con un filtro paso bajo de frecuencia variable.

Síntesis por tabla: Consiste en almacenar un periodo de la señal. Al reproducir el sonido, la tabla se recorre a la velocidad necesaria para producir el pitch deseado, y tantas veces como sea necesario para completar la duración de la nota.

Síntesis por muestreo: Semejante a la síntesis por tabla, en ella la tabla no almacena un único periodo de la señal, sino la nota completa. Generalmente, se usan sonidos grabados a partir de instrumentos acústicos reales.

Síntesis FM: Basada en modular en frecuencia una señal base, habitualmente una sinusoide. Cuando la velocidad de modulación es baja, el efecto es la fluctuación de la frecuencia fundamental (*vibrato*); pero, aumentando la velocidad a valores comparables a la frecuencia fundamental es posible obtener sonidos completamente distintos.

Síntesis por modelado: Consistente en implementar algorítmicamente un proceso de la naturaleza a partir de sus ecuaciones fundamentales, habitualmente diferenciales.

Los sintetizadores comerciales a menudo usan varias de estas tecnologías, o las combinan entre sí, para construir sus paletas de instrumentos. En esta práctica nos centraremos en la síntesis por tabla (con posible ampliación a la síntesis por muestreo) y la síntesis FM.

[1] También son destacables los repetidos intentos de Jimmy Page con el theremin en temas como "Whole Lotta Love" o "No Quarter", con Led Zeppelin, o en la banda sonora de la infumable película protagonizada por Charles Bronson "Yo soy la justicia".

1.1. Envolvente temporal de la nota, ADSR.

Los distintos esquemas de síntesis mencionados en el apartado anterior sirven para caracterizar acústicamente el sonido generado, esto es, su *timbre*. El timbre de un cierto sonido viene determinado por su estructura frecuencial, que, en el caso de las señales periódicas, es igual a la amplitud relativa de sus armónicos (múltiplos enteros de la frecuencia fundamental).

Pero el timbre no basta para caracterizar completamente el sonido de un instrumento; también es necesario dotar al sonido de una envolvente temporal adecuada. Por ejemplo, al tocar una nota en un piano, el sonido producido presenta un pico acusado de la intensidad, seguido de una disminución paulatina hasta su completa extinción. Por el contrario, en el caso de un órgano, la intensidad permanece más o menos constante desde el momento en que se pulsa la tecla y hasta el momento de soltarla, en que disminuye a cero de manera abrupta.

En esta práctica vamos a implementar la envolvente siguiendo un esquema clásico: el ADSR, en el cual se representa la envolvente usando cuatro parámetros, a cuyas iniciales responde su nombre (*Attack*, *Decay*, *Sustain* y *Release*):

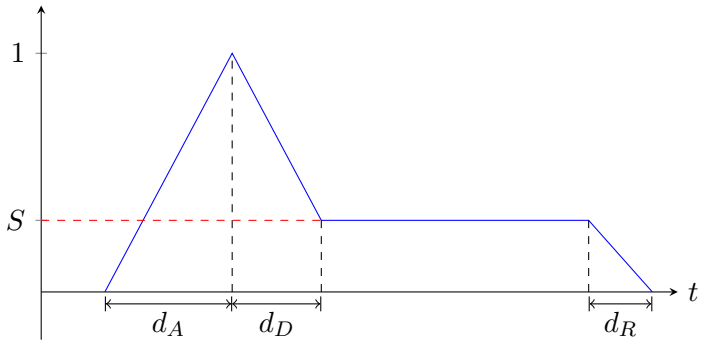
Ataque: Tiempo transcurrido desde que se inicia la nota hasta que alcanza su valor máximo, que se normaliza a uno. El inicio de la nota vendrá marcado por la ejecución del intérprete (por ejemplo, pulsando una tecla en un teclado) o por una orden enviada por un secuenciador.

Caída: Tiempo transcurrido desde que se alcanza el valor máximo hasta que disminuye a su valor de mantenimiento.

Mantenimiento: Valor de la amplitud en que se mantiene la nota después del ataque y caída, y que se mantendrá en tanto en cuanto no se dé la orden de finalizarla (por ejemplo, porque el intérprete deja de pulsar la tecla correspondiente).

Liberación: Tiempo transcurrido desde que se recibe la orden de finalizar la nota hasta que el sonido se apaga completamente.

Nótese que todos los parámetros, salvo el nivel de mantenimiento, son tiempos, que se expresan en segundos. El nivel de mantenimiento es un valor real, habitualmente menor o igual al valor máximo, que es uno.



Hay que indicar que estas envolventes no sólo se utilizan para la amplitud de la señal; también pueden usarse para controlar la evolución temporal de otros parámetros. Por ejemplo, en un sistema sustractivo, el resultado de la envolvente puede ser la frecuencia de corte de un filtro paso bajo, lográndose relaciones entre los armónicos que cambian con el tiempo.

1.2. Control de instrumentos electrónicos: estándar MIDI.

Los instrumentos electrónicos suelen manejarse usando el estándar MIDI (de *Musical Instrument Digital Interface*). Éste describe el protocolo de comunicaciones, el interfaz digital y las conexiones eléctricas

para conectar entre sí todo tipo de instrumentos electrónicos, así como ordenadores, secuenciadores y demás aparatos usados en la grabación, edición y reproducción de música.

El MIDI se basa en el envío de *mensajes* que especifican las instrucciones que deben realizar los distintos elementos de la cadena. El formato de los mensajes es binario, con un primer byte que indica el tipo de mensaje, en sus cuatro primeros bits, y a quién va dirigido, en los cuatro bits restantes; seguido de hasta dos bytes que contienen sus parámetros.

Por ejemplo, los mensajes que controlan cada una de las notas producidas por un sintetizador tienen la forma:

MIDI 1.0 Channel Voice messages

Event	Status byte		Byte 2	Byte 3
Note Off	0x8		Note Number	Velocity
Note On	0x9		Note Number	Velocity
Key Pressure	0xA		Note Number	Pressure
Control Change	0xB	Channel	Controller Number	*not used*
Program Change	0xC		Program Number	*not used*
Channel Pressure	0xD		Pressure	*not used*
Pitch Wheel	0xE		Wheel Value (2 Bytes)	

Así, por ejemplo, el comando `0x8C 0x40 0x7F` implica iniciar un sonido (`0x8`) en el canal 12 (`0xC`), con la nota 64 (`0x40`, que se corresponde con un Mi de la octava central) y con la máxima velocidad 127 (`0x7F`). Si, por ejemplo, el canal 12 se corresponde con una guitarra, el resultado es tañer con la máxima fuerza posible su primera cuerda al aire.

Por motivos de simplicidad de manejo, en esta práctica no se usará el estándar MIDI directamente, sino una adaptación del mismo usando ficheros de texto, cuya descripción puede encontrarse en la Sección 2.1.

2. Funcionamiento del sintetizador polifónico.

2.1. Ficheros `instruments`, `effects` y `score`.

En la práctica se va a usar una adaptación del estándar MIDI cuya característica fundamental es el uso de ficheros de texto, y no binarios, para la configuración de los instrumentos (`instruments`) y efectos (`effects`), así como para especificar la partitura a ejecutar (`score`).

Los tres ficheros pueden incluir líneas en blanco, que se descartan, y comentarios. Estos últimos comienzan con el carácter *almohadilla* (#) y abarcan hasta el final de la línea.

2.1.1. Fichero `instruments`.

Usaremos este fichero para configurar los instrumentos que van a intervenir en la partitura. Cada instrumento aparece como una línea de texto con tres campos separados por tabulador:

Índice: Número entero que identificará al instrumento en el ficheros de efectos y en la partitura.

Nombre: Nombre del instrumento. Debe coincidir con el de una clase C++ que implemente los algoritmos relacionados con el instrumento (constructor, sintetizador, destructor, etc.).

Parámetros: Cadena de los parámetros del instrumento, en la forma `nombre=valor;`. Los parámetros dependerán del instrumento. Por ejemplo, pueden incluir los valores de la envolvente ADSR, la frecuencia de modulación en síntesis FM, la frecuencia de corte en síntesis sustractiva, etc.

Inicialmente, sólo está implementado el instrumento `InstrumentDumb`. Este instrumento pretende generar un sonido con forma sinusoidal mediante una tabla en la que almacenamos los valores de un periodo de sinusoide. En el fichero de instrumentos, deberemos indicar el tamaño de la tabla (parámetro `N`) y los parámetros de la envolvente ADSR (`ADSR_A`, `ADSR_D`, `ADSR_S` y `ADSR_R`):

instruments.orc

```
1   InstrumentDumb  ADSR_A=0.01; ADSR_D=0.5; ADSR_S=0.4; ADSR_R=0.1; N=40;
```

Con esto indicamos que el índice del instrumento es 1; su nombre es `InstrumentDumb`; tiene un ataque de 10 ms, una caída de 0.5 s, un nivel de mantenimiento de 0.4 y un tiempo de liberación de 0.1 s; y el número de muestras de la tabla en la que almacenaremos la forma de un periodo es 40.

2.1.2. Fichero `effects`.

El fichero `effects` es similar al `instruments` salvo que, en lugar de servir para incorporar y configurar los instrumentos que participan en la partitura, hace estas misiones para los efectos que se aplicarán a cada instrumento o a la orquesta completa.

En la versión inicial sólo está implementado un trémolo sencillo. El trémolo es un efecto musical consistente en modificar periódicamente la amplitud de la señal. Si la forma de esta modificación es sinusoidal, el trémolo es equivalente a una modulación sinusoidal de amplitud, gobernada por dos parámetros: la frecuencia de la modulación en hercios (f_m) y su amplitud (A).

$$x_r[n] = x_i[n]\frac{1 + A\cos(2\pi F_m n)}{1 + A} \tag{1}$$

Donde F_m es la frecuencia discreta de modulación, $F_m = f_m/f_s$, con la frecuencia de muestreo f_s.

Por ejemplo, la línea siguiente establece como efecto 13 un trémolo con una amplitud de la modulación igual al 50 % y una frecuencia de modulación de 10 Hz:

effects.orc

```
13   Tremolo  A=0.5; fm=10;
```

2.1.3. Fichero `score`.

El fichero `score` contiene la secuencia de comandos a ejecutar por el sintetizador. Existen dos tipos de comandos: los que indican las notas a interpretar y los que gestionan los efectos a usar. En ambos casos, la línea comienza con el tiempo en *ticks* que tiene que pasar antes de que el comando se ejecute. De este modo, *incremental* e indirecto, el fichero permite ajustar la duración de las notas interpretadas y la velocidad de interpretación (vease el apartado 2.2).

El formato es de una línea de texto por cada comando MIDI a ejecutar. Se reconocen cuatro comandos:

9: Comando MIDI `NoteOn`, indica que debe iniciarse la generación de una nota. Equivale a pulsar una tecla en un teclado.

El comando `NoteOn` toma como argumentos:

Time: Tiempo, en ticks, hasta el inicio de la nota. Es la primera columna de la línea.

Channel: Canal (instrumento) a quien va dirigido el comando. En el estándar MIDI es un número entre 0 y 15, con lo que sólo puede haber 16 instrumentos sonando simultáneamente. Este número se corresponde con el índice del instrumento en el fichero `instruments`.

Note: Valor de la nota (su frecuencia fundamental) en semitonos (subdivisión de la octava en 12 partes iguales). Puede tomar cualquier valor entre 0 y 127, donde 0 corresponde a 8.1758 Hz (Do-1), y 127 a 12544 Hz (Sol9). Se fija de manera que al La central (La4, 440 Hz) le corresponde el valor 69.

Velocity: Velocidad en el pulsado de la nota, utilizado, sobre todo, para fijar su volumen. También puede usarse para determinar el timbre u otros parámetros de la nota. Su valor está comprendido entre 0, equivalente a `NoteOff` y 127.

8: Comando MIDI `NoteOff`, indica que debe iniciarse la finalización de la nota. Equivale a soltar la tecla de un teclado. En general, la nota no finalizará al ejecutarse el comando, sino que en ese momento se inicia la fase *release* de la curva ADSR y la nota finalizará al acabar ésta.

Mismos argumentos que `NoteOn`.

0: Comando no presente en MIDI, indica que la nota debe finalizar, con independencia de en que fase de la curva ADSR se encuentre.

Mismos argumentos que `NoteOn`.

12: Comando MIDI `Effect Control 1`, permite controlar efectos como el trémolo, vibrato, etc.

Admite los siguientes argumentos:

Time: Tiempo, en ticks, hasta que la puesta en marcha del efecto. Primera columna de la línea.

Channel: Canal (instrumento) con el mismo significado en en `NoteOn`.

Effect: Índice del efecto a utilizar, que debe corresponderse con su índice en el fichero `effects`.

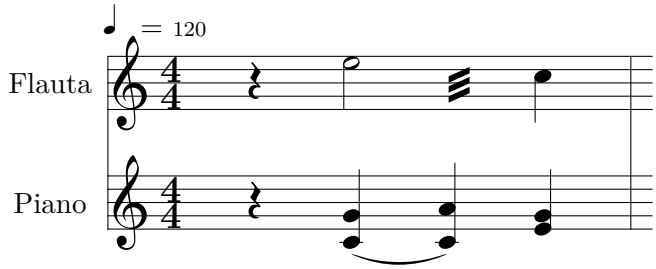
On/Off: Interruptor del efecto para su puesta en marcha (1) o parada (0). Al ponerse en marcha, el efecto se aplicará a todas las notas que se estén ejecutando del instrumento, así como a las que empiecen a partir de ese momento, y hasta que no se reciba la orden de pararlo.

Por ejemplo, considérese el fichero siguiente, en el que se supone que en el fichero `instruments` el índice 4 corresponde a un piano y el 5 a una flauta, y que en el fichero `effects` el índice 7 corresponde a un vibrato; también suponemos que se ha convertido usando 120 ticks por pulso:

```
# Time   On/Off   Channel  Note     Velocity
# Time   Control  Channel  Effect   On/Off
120      9        4        60       100
0        9        4        67       100
0        9        5        76       100
120      8        4        67       100
0        9        4        69       100
0        12       5        7        1
120      8        4        60       100
0        8        4        69       100
0        9        4        64       100
0        9        4        67       100
0        8        5        76       100
0        9        5        72       100
0        12       5        7        0
120      8        4        64       100
0        8        4        67       100
0        8        5        72       100
```

Después de una pausa inicial de 120 ticks, se inician tres notas: un Do (60) y un Sol (67), a cargo del piano, y un Mi (76), a cargo de la flauta. Al cabo de 120 ticks, el piano para el Sol e inicia un La (69); además, se pone en marcha un trémolo que afectará a la nota de la flauta, pero no a las del piano. Al cabo de otros 120 ticks, el piano finaliza las dos notas que estaba interpretando (Do y La) e inicia un Mi (64) y un Sol (67); además, la flauta finaliza su nota e inicia un Do (72), y el trémolo se para. Finalmente, al cabo de otros 120 ticks, el piano y la flauta finalizan sus respectivas notas.

Suponiendo que usamos 120 ticks por pulso, y que el pulso se corresponde con la figura *negra*, el score anterior se corresponde con la partitura siguiente:



El script `midi2sco` permite convertir ficheros MIDI a este formato de fichero. El programa sólo interpreta los comandos MIDI `NoteOn` (9), `NoteOff` (8) y `tempo`. El resto los almacena en el fichero de salida como comentarios. A menudo es útil visualizar este fichero para determinar a qué instrumento corresponde cada canal, aunque esa información no siempre está disponible en el fichero MIDI.

Los comandos MIDI `tempo` también se almacenan en el fichero `score` como comentarios. Pero, si la opción `-bpm` del programa se deja en su valor por defecto (`'auto'`), la partitura se generará usado el valor del primer comando `tempo` encontrado.

2.2. Gestión del tiempo en el sintetizador.

Un *tick* es la unidad básica de duración de la interpretación. La señal sintetizada se genera encadenando segmentos de señal de un tick de duración. La duración de un tick depende de dos factores: la velocidad de ejecución, medido en tiempos o pulsos por minuto (**bpm**: *beats per minute*), y la resolución temporal, medida en ticks por pulso (**tpb**: *ticks per beat*).

Los pulsos por minuto están asociados al concepto musical de *tempo*. Es la velocidad marcada por los metrónomos y es habitual indicarla en las partituras. Por ejemplo, la notación ♩=80 le indica al intérprete que debe ejecutar la composición a una velocidad tal que haya 80 negras en un minuto (tempo de *andante*). Habitualmente, la figura de referencia al indicar el tempo es la de una nota *negra* (♩), aunque también es posible usar la *blanca* (𝅗𝅥) o la negra con puntillo (♩.), y, menos frecuentemente, otras figuras como la *redonda* (𝅝) o la corchea (♪).

Los ticks por pulso fijan la resolución temporal de la interpretación. En principio, debe fijarse de manera que sea posible ejecutar todas las notas presentes la partitura. Por ejemplo, si fijáramos tpb=5, no podríamos interpretar correctamente corcheas (de duración la mitad de una negra) o semicorcheas (de duración un cuarto). Suele tomarse un múltiplo de 12 de manera que las subdivisiones del pulso más comunes (mitad, tercio y cuarto) estén contempladas. En interpretaciones en directo suele ser conveniente tomar un valor elevado, ya que, a menudo, la duración del tick está relacionada con el tiempo de respuesta desde que el intérprete ejecuta la nota hasta que ésta se produce, el denominado *tiempo de latencia*.

Todo junto, la duración de un tick queda:

$$\text{tick} = \frac{60}{\text{bpm} \cdot \text{tpb}} \quad [s] \tag{2}$$

En los ficheros de partitura (*score*) que usaremos en esta práctica el tiempo se expresa en ticks de manera incremental. Por ejemplo, considérese la siguiente línea del fichero `doremi.sco`:

```
40  9   1   64  100
```

En esta línea se está indicando que, pasados 40 ticks, debe iniciarse la nota indicada por el resto de parámetros (`NoteOn`: 9). De alguna manera, y así está implementado el algoritmo en el fichero `synth/synthesizer.cpp`, le estamos diciendo al sintetizador que durante los próximos 40 ticks no pasa nada, y debe continuar produciendo las notas que estuvieran en marcha en ese momento. Al cabo de esos 40 ticks, el sintetizador deberá iniciar la nota, cuya duración no está indicada en la línea, sino que vendrá determinada por la aparición del comando `NoteOff` (8).

Tareas:

- Compile e instale los programas de la práctica, con la orden `make release` , y compruebe que se ejecutan sin problemas.
- Ejecute el script `midi2sco` con la opción `-help` para ver su modo de empleo, y úselo para convertir al formato `score` el fichero MIDI `samples/Hawaii5-0.mid`. Visualice el fichero para ver qué instrumentos participan en él. Analice el inicio de la partitura y determine cómo comienza la canción.

  NOTA: en `Hawaii5-0.mid` cada elemento de percusión ocupa un canal diferente. En otros casos (*General MIDI 1 Sound Set*), el canal 10 se reserva para la batería completa, y cada nota corresponde a un elemento distinto. El score `The_Christmas_Song_Lennon.sco` funciona de este modo, con los distintos elementos de la batería en la pista 2. Puede consultar la organización de estos elementos en las especificaciones de General MIDI.

3. Síntesis por tabla (y algo de muestreo): `InstrumentDumb`.

El modo más sencillo de generar un sonido periódico de frecuencia variable es almacenar un periodo del mismo en una tabla y recorrer ésta a la velocidad adecuada para conseguir el pitch deseado. Es lo que denominamos *síntesis por tabla*.

En los ficheros `instruments/instrument_dumb.cpp` y `instruments/instrument_dumb.h` se define un instrumento *tonto* de este tipo (`InstrumentDumb`), en el que la señal generada tiene forma sinusoidal. Utilizaremos este instrumento para implementar la síntesis por tabla y, además, nos servirá como ejemplo para aprender a gestionar otros nuevos, como los basados en síntesis por muestreo o FM.

El instrumento se inicializa al arrancar el programa `synth` invocando al constructor de su clase asociada, `InstrumentDumb()`, que toma por argumento la cadena de texto con los parámetros del instrumento tal y como aparecen en el fichero `instruments`.

3.1. Constructor de la clase, `InstrumentDumb::InstrumentDumb`.

Lo primero que hace el constructor es declarar el instrumento como inactivo y ubicar el espacio para almacenar un tick de señal:

instruments/instrument_dumb.cpp

```
11 InstrumentDumb::InstrumentDumb(const std::string &param)
12   : adsr(SamplingRate, param) {
13   bActive = false;
14   x.resize(BSIZE);
```

A continuación, y usando las funciones de la librería definidas en `keyvalue.h`, analizamos la cadena de parámetros localizando los de interés para el instrumento. En este caso, el único parámetro de interés es la longitud de la tabla (`N`), ya que los parámetros que gestionan la curva ADSR se procesan en otra parte del programa:

instruments/instrument_dumb.cpp

```
20 KeyValue kv(param);
21 int N;
22
23 if (!kv.to_int("N",N))
24   N = 40; //default value
```

Finalmente, el constructor inicializa la tabla con un período de sinusoide:

instruments/instrument_dumb.cpp

```
27 tbl.resize(N);
28 float phase = 0, step = 2 * M_PI /(float) N;
29 index = 0;
30 for (int i=0; i < N ; ++i) {
31   tbl[i] = sin(phase);
32   phase += step;
33 }
```

3.2. Gestión de comandos, método `InstrumentDumb::command()`

Cada vez que el programa encuentra un comando MIDI en el fichero `score`, invoca al método `command()` de la clase, que toma por argumentos los campos del fichero `score`, esto es: el comando, la nota y la velocidad.

Si el comando es `NoteOn` (9), el método declara activo al instrumento, inicializa la curva ADSR y un contador, `index`, que usaremos para recorrer la tabla, y fija el valor de la amplitud, `A` usando como valor máximo 127 (aunque no se comprueba si la velocidad es mayor):

instruments/instrument_dumb.cpp

```
38 if (cmd == 9) {                //'Key' pressed: attack begins
39   bActive = true;
40   adsr.start();
41   index = 0;
42       A = vel / 127.;
43 }
```

Si el comando es `NoteOff` (8) o `EndNote` (0), el método inicia la fase *release* de la curva ADSR o finaliza el sonido, respectivamente:

instruments/instrument_dumb.cpp

```
44 else if (cmd == 8) {  //'Key' released: sustain ends, release begins
45   adsr.stop();
46 }
47 else if (cmd == 0) {  //Sound extinguished without waiting for release to end
48   adsr.end();
49 }
```

3.3. Síntesis de la señal, método `InstrumentDumb::synthesize()`.

La síntesis en sí se realiza en el método `InstrumentDumb::synthesize()`. La nota puede encontrarse en tres situaciones:

Finalizada: Si la curva ADSR ya ha llegado a su final, asignamos a la señal sintetizada (vector `x`) el valor 0 y marcamos la nota como inactiva. El programa se encargará de destruir todos los objetos asociados a notas marcadas como inactivas.

instruments/instrument_dumb.cpp

```
54 if (not adsr.active()) {
55   x.assign(x.size(), 0);
56   bActive = false;
57   return x;
58 }
```

Inactiva: Si ya ha sido marcada como inactiva, el método devuelve el vector `x`, que, previamente, habremos puesto a cero. De hecho, nunca se debería pasar por este estado, ya que, si la nota está inactiva, es el programa el que se encarga de destruirla.

instruments/instrument_dumb.cpp
```
59  else if (not bActive)
60    return x;
```

Activa: Si la nota está activa, el método realiza la síntesis y aplica la envolvente ADSR:

instruments/instrument_dumb.cpp
```
62  for (unsigned int i=0; i<x.size(); ++i) {
63    x[i] = A * tbl[index++];
64    if (index == tbl.size())
65      index = 0;
66  }
67  adsr(x); //apply envelope to x and update internal status of ADSR
68
69  return x;
```

La envolvente ADSR es un objeto, definido para cada nota individual, que se gestiona desde el programa principal y tiene, de alguna manera *vida propia*. Por ahora no nos tendremos que preocupar mucho de ella.

Fijémonos en el uso de la variable `index` en este método: utilizamos su valor para recorrer, de uno en uno, los valores almacenados en la tabla `tbl`. El valor de `index` sólo se inicializa al construir el objeto; a partir de ese momento, cada vez que se entra en el método `synthesize()` se parte del último valor usado en la invocación anterior. De este modo se garantiza la continuidad de fase en la construcción de la nota completa.

Por otro lado, no hay ningún control de la velocidad con la que se recorre la tabla. Independientemente del pitch de la nota, la tabla se recorre de uno en uno, con lo que el sonido generado será siempre de la misma frecuencia.

3.4. Empleo del programa `synth` con el instrumento `InstrumentDumb`.

Podemos ver las opciones del programa `synth` usando la opción `-help`:

```
usuario:~/PAV/P5/work$ synth --help
synth - Polyphonic Musical Synthesizer

Usage:
    synth [options] <instruments-file> <score-file> <output-wav>
    synth (-h | --help)
    synth --version

Options:
    -b FLOAT, --bpm=FLOAT      Beats per minute: tempo [default: 120]
    -t FLOAT, --tpb=FLOAT      Ticks per beat: must match input score  [default: 120]
    -g FLOAT, --gain=FLOAT     Gain applied to the output waveform [default: 0.5]
    -e STR, --effect-file=STR  Text file with the definition of the effects
    -v, --verbose              Show the score on screen as it is played
    -h, --help                 Show this screen
    --version                  Show the version of the project

Arguments:
    instruments-file           Text file with the definition of the instruments
    score-file                 Text file with the score to be played
    output-wav                 Wave file with synthesized score
```

Como ficheros de prueba, podemos usar un score sencillo, `doremi.sco` que, simplemente, realiza una escala de una octava empezando en el Do central (60), usando corcheas con doble puntillo seguidas de silencio de fusa:

doremi.sco

```
0       9       1       60      100
120     8       1       60      100
40      9       1       62      100
120     8       1       62      100
40      9       1       64      100
120     8       1       64      100
40      9       1       65      100
120     8       1       65      100
40      9       1       67      100
120     8       1       67      100
40      9       1       69      100
120     8       1       69      100
40      9       1       71      100
120     8       1       71      100
40      9       1       72      100
120     8       1       72      100
0       0       1       0       0
```

Usaremos como fichero de instrumentos el fichero `dumb.orc`, que sólo define el instrumento `InstrumentDumb` con los parámetros de la curva ADSR y la longitud de la tabla:

dumb.orc

```
1       InstrumentDumb  ADSR_A=0.02; ADSR_D=0.5; ADSR_S=0.4; ADSR_R=0.1; N=40;
```

Finalmente, generamos el fichero de audio y lo escuchamos[2]:

```
usuario:~/PAV/P5/work$ synth dumb.orc doremi.sco doremi.wav
Register instrument: 1 InstrumentDumb
usuario:~/PAV/P5/work$ pley doremi.wav
doremi.wav:

 File Size: 456k      Bit Rate: 706k
  Encoding: Signed PCM
  Channels: 1 @ 16-bit
Samplerate: 44100Hz
Replaygain: off
  Duration: 00:00:05.17

In:100%  00:00:05.17 [00:00:00.00] Out:228k  [      |      ]        Clip:0
Done.
```

El resultado dista mucho de sonar como una escala de una octava. Ello es debido a que el instrumento es realmente *tonto*, y no ajusta la velocidad con la que recorre la tabla a la frecuencia de pitch deseada.

3.5. Implementación del instrumento `Seno`.

Tareas

- Use el programa `synth` para generar el fichero de audio `doremi.wav` usando el score `doremi.sco` y el fichero de instrumentos `dumb.orc`.
  - Visualice el fichero generado con `wavesurfer` y verifique que se cumplen los parámetros de la envolvente ADSR fijados en `dumb.orc`.

[2] El comando usado para reproducir el archivo es *pley*. No se trata de un error: en WSL no se tiene acceso a la tarjeta de sonido del PC; así pues, el autor se ha visto obligado a definir un comando que invoque al programa `sox` de Windows con los argumentos adecuados. Para ello, ha incluido en su fichero `~/.bshrc` la línea `pley() { echo $1; /mnt/c/Program\ Files\ \(x86\)/sox-14-4-2/sox.exe $1 -t waveaudio; }`.

    - Oiga la señal y compruebe que el instrumento `InstrumentDumb` es realmente tonto.

- Copie los ficheros `InstrumentDumb.cpp` y `InstrumentDumb.h` del directorio `instruments`, a los ficheros `seno.cpp` y `seno.h` del mismo directorio, y modifique el código necesario para construir el instrumento `Seno` que produzca una sinusoide con la frecuencia fundamental correcta.

    - Deberá modificar los ficheros `src/instruments/instrument.cpp` y `src/meson.build` para añadir el nuevo instrumento al programa.
    - Recuerde que el valor de la nota `Note` viene expresado en semitonos y está fijado de manera que el La central (La4, 440 Hz) se corresponde con Note = 69:

$$\text{Note} = 69 + 12 \cdot \log_2 \frac{f_0}{440} \tag{3}$$

    - Tenga presente que es necesario mantener la continuidad de fase.
    - En general, al recorrer la tabla con los saltos adecuados para producir una cierta frecuencia fundamental, será necesario acceder a índices no enteros de la tabla. Es decir, el valor deseado no se corresponde con ninguno de los que están almacenados en ella, sino a uno intermedio entre dos que sí lo están (que pueden ser el último y el primero...).
        - En primera aproximación, puede redondear el índice requerido a entero y usar para la muestra uno de los valores almacenados en la tabla.
            - Pero esta solución introduce una distorsión que es claramente audible.
        - Como trabajo de ampliación, se propone calcular el valor de la muestra como interpolación lineal entre los valores inmediatamente anterior y posterior al índice deseado (pero recuerde que el siguiente del último es el primero...).

3.6. Uso de tablas externas.

Evidentemente, la generación de sonidos a partir de tabla puede extenderse fácilmente a otras formas de onda, como la triangular, la cuadrada o el diente de sierra. Esto permite generar una gran cantidad de instrumentos diferentes, aunque poco realistas y/o interesantes.

Una opción mucho más potente es emplear un fichero externo con un ciclo de señal de una fuente real. El resultado no es tan espectacular como pueda parecer a simple vista por toda una serie de factores:

- Aunque se consigue reproducir el timbre en un instante determinado, en un instrumento real el timbre varía durante la generación de cada nota. Habitualmente, el sonido presenta mayor contenido armónico en el ataque, disminuyendo paulatinamente en las fases de mantenimiento y liberación.
- Al reproducir ciclo tras ciclo los mismos valores, se obtiene una señal perfectamente periódica. En instrumentos reales, esa periodicidad es rota por distintos motivos: la ya comentada variación tímbrica a lo largo de la ejecución de la nota; la aparición de componentes no armónicas (es decir: que no son múltiplos enteros de la frecuencia fundamental), especialmente remarcable en instrumentos de metal; el propio carácter estocástico de la ejecución musical; etc.
- La necesidad de ajustar con sumo cuidado los parámetros de la curva ADSR, ya que, en algunos instrumentos, esta envolvente es casi tan importante como el timbre *estacionario*.

El resultado de todo esto es que, a menudo, el sonido sintetizado suena más a un organillo de baja calidad que al instrumento del cual se han extraído las muestras. Puede mejorarse algo con distintas técnicas. Por ejemplo, añadir aleatoriedad en el recorrido de la tabla, o aplicar un filtrado paso bajo controlado por la ADSR de manera que el contenido armónico disminuya con el tiempo.

Partiendo del instrumento `Seno`, es relativamente sencillo generar un instrumento que, en lugar de generar por sí mismo un ciclo de señal, lo lee de un fichero WAVE externo. Básicamente, sólo cambia el constructor del instrumento, que, en lugar de usar como parámetro el tamaño de la tabla, toma el nombre del fichero con el ciclo:

```
std::string file_name;
static string kv_null;
if ((file_name = kv("file")) == kv_null) {
  cerr << "Error: no se ha encontrado el campo con el fichero de la señal para un ...
  ↪  instrumento FicTabla" << endl;
  throw -1;
}

unsigned int fm;
if (readwav_mono(file_name, fm, tbl) < 0) {
  cerr << "Error: no se puede leer el fichero " << file_name << " para un instrumento ...
  ↪  FicTabla" << endl;
  throw -1;
}
```

3.7. Uso de notas completas, los *samplers*.

Un resultado mucho más natural se obtiene cuando se utilizan grabaciones de notas completas. Son los denominados *samplers*. En los samplers más sencillos se almacena una nota a un pitch determinado, y luego se reproduce a la velocidad necesaria para obtener el pitch deseado de la nota. Este procedimiento tiene el inconveniente de que no sólo se ve afectado el pitch, sino también la duración de la nota y la envolvente del contenido espectral. En los sistemas de más alta calidad se almacenan varias grabaciones para cada nota concreta, y se selecciona la grabación a usar en función de parámetros de la interpretación como la velocidad de pulsado de la tecla o la presión ejercida sobre ella. Así, algunas librerías para piano incluye más de 1000 grabaciones para un total de 88 notas.

Un caso especialmente interesante, y sencillo, del uso de samplers es para la síntesis de sonidos de percusión (tambores, platillos, panderetas, etc.). En este tipo de instrumento no tiene sentido hablar de pitch; ni tampoco lo tiene el grabar un segmento de duración limitada y repetirlo periódicamente. La mejor solución para obtener sonidos percusivos de calidad es, simplemente, reproducirlos cada vez que se solicitan, sin que se vean afectados por el pitch o la envolvente ADSR.

Partiendo de la síntesis mediante sonidos externos almacenados en tabla del apartado anterior, es sencillo construir un sampler para sonidos percusivos. Todo lo que hay que hacer es dos pequeñas modificaciones los métodos `command` y `synthesize`:

`command()`: Fijamos el incremento de fase al valor uno para recorrer la tabla de muestra en muestra (con independencia del pitch). También dejamos sin efecto los comandos `NoteOff` (8) y `SoundOff` (0); el final de la nota se producirá cuando alcancemos el final de la tabla.

`synthesize()`: Modificamos el código para que, si se alcanza el final de la tabla, se complete la señal del tick con ceros y se dé la orden de finalizar la ADSR (llamando a `adsr.end()`).

3.8. Ejemplo sencillo del uso de tablas: Hawaii5-0.

Usando sonidos extraídos de Sonidos mp3 gratis y de FindSounds, los profesores de la asignatura han sintetizado el tema principal de la serie de TV Hawaii5-0. Para los instrumentos melódicos se ha usado síntesis por tabla (editando el fichero original para extraer un único ciclo de señal), y para los percusivos, sampler (almacenando la señal completa). En el directorio `samples` puede encontrar el fichero MIDI original, su conversión a `score` y el resultado de la síntesis (fichero `.wav`).

Tarea de ampliación:

Implemente las síntesis por tabla de fichero externo y sampler. Consiga ficheros de sonidos o genérelos por su cuenta, y realice la síntesis de algún tema musical. Puede usar alguno de los proporcionados en el directorio `samples` o buscar otros de su agrado.

4. Efectos sonoros.

Una parte importante en música electrónica (y en la electrónica aplicada a la música) es la de los efectos sonoros, como la reverberación, la distorsión de amplitud y/o fase, el trémolo, el vibrato, etc. Un ejemplo típico son los clásicos pedales usados por los músicos de rock, los guitarristas especialmente: *overdrive*, *wah*, *chorus*, *flanger*, etc.

En el sintetizador de la práctica los efectos se gestionan de manera similar a los instrumentos y se aplican a la señal generada por instrumentos concretos. Así, si un instrumento produce más de una nota al mismo tiempo, el efecto afecta a la suma de todas ellas, en la línea de cómo actúan los pedales de los guitarristas.

En el programa `synth`, los efectos se almacenan en el directorio `effects` como clases herederas de la clase `Effect`, de un modo similar a lo que se hace con los instrumentos y la clase `Instrument`. Así, para añadir un efecto nuevo deben realizarse los pasos siguientes:

- Implementar el efecto en los ficheros *`efecto`*`.cpp` y *`efecto`*`.h` (donde *`efecto`* es el nombre del efecto, *tremolo*, por ejemplo).

  Constructor: En el constructor debemos utilizar los parámetros pasados como tercer argumento del fichero `effects` para configurar el efecto.

  `command()`**:** Al contrario que en el caso de los instrumentos, para los efectos es el programa principal el que se encarga de gestionar la puesta en marcha y apagado del efecto. Esto se realiza usando el quinto (y último) argumento del comando MIDI del fichero `score`. Si éste vale cero, el comando se apaga; si tiene cualquier otro valor, el comando se pone en marcha o continua su ejecución. Además, su valor se pasa a `command()`. De este modo, podemos modificar el comportamiento de un efecto ya en marcha.

  `operator()()`**:** Sobrecarga del operador `()` que se invocará con un un tick de la señal producida por el instrumento como argumento de entrada. Es el método que, efectivamente, aplica el efecto a la señal.

- Incorporar la clase *`Efecto`* al código fuente del programa `synth`:
  - Incluir el fichero *`efecto`*`.h` en el fichero `effects/effect.cpp`.
  - Añadir la llamada al constructor del efecto en el fichero `effects/effect.cpp`.
  - Añadir el fichero *`efecto`*`.cpp` a la lista de programas en `src/meson.build`.

4.1. Generación de *trémolo*.

El efecto más sencillo de construir, y que servirá de ejemplo y modelo para construir otros, es el *trémolo*. Éste consiste en la modificación rápida y oscilante del volumen del sonido, y es un efecto empleado desde los tiempos del Canto Gregoriano, como poco.

Desde un punto de vista matemático, y suponiendo un variación sinusoidal de la amplitud, el trémolo viene dado por la expresión siguiente:

$$x_r[n] = x_i[n]\,\frac{1 + A\cos(2\pi\, F_m n)}{1 + A} \tag{4}$$

Donde F_m es la frecuencia discreta de modulación: $F_m = f_m/f_s$, con f_m la frecuencia de modulación en hercios y f_s la frecuencia de muestreo. A es la profundidad de la modulación; en principio, $0 \leq |A| < 1$, aunque se pueden conseguir efectos curiosos usando $|A| > 1$.

Implementamos el trémolo en el fichero `effects/tremolo.cpp`. El código es lo suficientemente sencillo para considerarse autoexplicado:

src/effects/tremolo.cpp

```
Tremolo::Tremolo(const std::string &param) {
  fase = 0;

  KeyValue kv(param);

  if (!kv.to_float("A", A))
    A = 0.5; //default value

  if (!kv.to_float("fm", fm))
    fm = 10; //default value

  inc_fase = 2 * M_PI * fm / SamplingRate;
}

void Tremolo::command(unsigned int comm) {
  if (comm == 1) fase = 0;
}

void Tremolo::operator()(std::vector<float> &x){
  for (unsigned int i = 0; i < x.size(); i++) {
    x[i] *= (1 - A) + A * sin(fase);
    fase += inc_fase;

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

En la página 4 se vio un ejemplo de configuración de un trémolo de frecuencia 10 Hz y amplitud 0.5:

```
13   Tremolo  A=0.5; fm=10;
```

Introducimos el trémolo en el fichero `score` como un comando más. En el ejemplo siguiente, el trémolo se añade en la segunda nota de la escala `doremi.sco` y se saca justo al empezar la penúltima:

doremi.sco

```
0       9       1       60      100
120     8       1       60      100
0       12      1       13      1
40      9       1       62      100
120     8       1       62      100
40      9       1       64      100
120     8       1       64      100
40      9       1       65      100
120     8       1       65      100
40      9       1       67      100
120     8       1       67      100
40      9       1       69      100
120     8       1       69      100
40      9       1       71      100
0       12      1       13      0
120     8       1       71      100
40      9       1       72      100
```

```
120     8       1       72      100
40      0       1       0       0
```

4.2. Generación de *vibrato*.

El vibrato es el efecto de variar, alternada y rápidamente, la afinación de la nota. Viene a ser el equivalente en frecuencia del tremolo, y es muy frecuente en música. Así, es habitual ver a los violinistas de una orquesta mover rítmicamente los dedos de la mano izquierda, incluso la mano entera y el antebrazo. Con este movimiento consiguen modificar ligeramente la tensión de la cuerda, haciendo que su afinación fluctúe en torno al pitch de referencia. Se trata de un efecto casi tan popular como el tremolo, aunque sujeto a críticas que éste no recibe (se ha llegado a hablar de las *guerras del vibrato*). Por ejemplo, los cantantes de ópera españoles e italianos del cambio del siglo XIX al XX eran despreciados por el público anglosajón por abusar de este efecto ramplón y empalagoso.

La implementación de un vibrato que tome una señal de duración N muestras y la devuelva con el pitch modificado de manera periódica tiene una parte fácil y otra complicada:

- Realizar el vibrato en sí es sencillo: basta con recorrer las muestras de la señal con una velocidad variable. Obtenemos la señal con el pitch inalterado recorriendo el vector de muestra en muestra; si queremos multiplicar el pitch por una cantidad K, deberemos recorrerlo cada K muestras (como, en general, K no es entero, será necesario realizar algún tipo de interpolación).
- El problema es que, al variar la velocidad con la que se recorre el vector con las muestras, el resultado no tiene por qué tener la misma longitud que la entrada. Es más, según cómo se implemente el vibrato, es posible tener un sistema no causal, lo cual, algorítmicamente hablando, equivale a acceder a muestras más allá del final del vector (`System error: access violation (core dumped)`).
  - Para compatibilizar la longitud de los vectores de entrada y salida, la solución es usar un `buffer` en el que almacenamos las muestras del vector de entrada que *no caben* en el vector de salida. Al crear un vector de salida nuevo, se empieza por las muestras almacenadas en el buffer y se completa el vector con muestras del vector nuevo.
  - Se puede evitar la eventual no causalidad usando un buffer inicial de ceros y longitud suficiente; pero esto implica un retardo entre la entrada y la salida que, en música no es deseable. Otra alternativa es usar una función moduladora tal que su integral desde $t = 0$ sea siempre no positiva; por ejemplo, $x_m(t) = -\sin(2\pi f_m t)$.

Los ficheros `effects/vibrato.cpp` y `effects/vibrato.h` implementan un vibrato siguiendo estas premisas.

El vibrato tiene dos parámetros: f_m, que indica la frecuencia de modulación en hercios; e I, que indica su extensión. I se expresa en semitonos (doceava parte de una octava, $2^{1/12}$) y representa la máxima disminución en el pitch producida por el efecto. Como la función moduladora es simétrica, este valor de la disminución se corresponde linealmente con el de aumento máximo; y, como la disminución máxima posible es (casi) igual a la propia frecuencia original, el aumento máximo queda limitado a una octava.

Indicamos ambos valores en el fichero `effects` correspondiente. Por ejemplo, la siguiente línea en `effects` genera un vibrato de 8 Hz y medio semitono de extensión:

```
4   Vibrato  I=0.5; fm=8;
```

Tareas:

Inserte comandos de control de trémolo y vibrato en el fichero `doremi.sco` y escuche el resultado obtenido.

Use inicialmente valores estándar en música interpretada por humanos:

- En el caso del trémolo, valores estándar serían $0 < A < 0.25$ y $4\,\text{Hz} < f_m < 10\,\text{Hz}$.
- Para el vibrato, mismo rango de frecuencias de muestreo, y $0 < I < 1.5$.

A continuación, pruebe otros valores más propios de otro tipo de criatura, por ejemplo: un vibrato con $fm = 2000$ y/o $I = 24$. El efecto resultante es la denominada *síntesis FM*, aunque, probablemente, sería más adecuada llamarla PM ya que se trata de una modulación de fase[a].

[a] Modulación de fase y de frecuencia son dos caras de una misma moneda: la *modulación angular*. La señal moduladora de una es la integral de la de la otra; pero, si se usa como moduladora una sinusoide, la diferencia se reduce a un factor de escala y otro de fase.

Tarea de ampliación:

Puede seguir el esquema empleado en la implementación del trémolo para implementar otros tipos de efecto sonoro, como la distorsión, el *wah*, etc.

Si se atreve, puede usar el esquema del vibrato para otros efectos que requieren el uso de buffers de longitud variable, como el *flanger* o el *chorus* (mono).

También puede abordar otros efectos, como la reverberación, pero, para ello, es probable que tenga que, o sea conveniente, implementar un esquema completamente nuevo.

5. Síntesis FM.

En esta parte de la práctica, se implementará un *instrumento virtual* basado en modulación de frecuencia. La *síntesis FM* es un método propuesto por John M. Chowning en un artículo publicado hace casi 50 años y que permite generar sonidos muy distintos con un coste computacional muy limitado.

> *The Synthesis of Complex Audio Spectra by Means of Frequency Modulation*, Journal of the Audio Engineering Society, vol 21, no 7, 1973.

En este trabajo, Chowning ilustra distintas configuraciones para lograr distintos instrumentos, tales como de viento de madera o metal, etc.

En general, el sonido asociado a una nota está formada por la fundamental y sus parciales (armónicos).

$$x(t) = E(t) \cdot \sum_{k=1}^{\infty} A_k \sin(2\pi k f_o t + \phi_k)$$

Prescindiendo de la envolvente temporal ($E(t)$), el valor de f_o es la frecuencia de la nota y las relaciones entre las amplitudes de los *parciales*, A_k, define el *timbre*.

Vamos a analizar el sistema más básico del artículo:

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

siendo f_c la frecuencia *portadora*, f_m, la frecuencia moduladora y I el índice de la modulación.

La frecuencia instantánea de esta señal es:

$$f(t) = \frac{1}{2\pi}\frac{d\phi(t)}{dt} = \frac{1}{2\pi}\frac{d}{dt}\Big(2\pi f_c t + I\sin(2\pi f_m t)\Big) = f_c + I f_m \cos(2\pi f_m t) \tag{5}$$

La idea inicial de Chowning era producir *vibrato*, con frecuencia instantánea oscilando entre $f_c \pm I f_m$. Valores típicos de f_m son unos pocos Hz ($\approx 5\,\text{Hz}$), y de la extensión de vibrato (ν), expresada en semitonos, entre 0.1 y 1. La relación entre el índice de modulación I y los parámetros del vibrato es:

$$\nu = 12\log_2\frac{f_c + I f_m}{f_c - I f_m} \quad \Rightarrow \quad I = \frac{f_c}{f_m}\cdot\frac{2^{\nu/12}-1}{2^{\nu/12}+1}$$

Por ejemplo, si $\nu = 1$ semitono, $f_m = 5\text{Hz}$, $f_c = 440\text{Hz}$, el valor de I es $3.28\,10^{-4}$.

Chowning observó que al incrementar el valor de I, la percepción ya no es de un *vibrato*, sino otro tipo de sonido. En particular, veremos que tomando $f_m = M f_c$ se pueden generar sonidos armónicos con distintos timbres.

Para calcular el espectro de $x(t)$ la expresamos como $x(t) = \text{Im}\{x_c(t)\}$, siendo

$$x_c(t) = A\,e^{j\left(2\pi f_c t + I\sin(2\pi f_m t)\right)} = A\,e^{j2\pi f_c t}\,e^{jI\sin(2\pi f_m t)}$$

Como la señal $e^{jI\sin(2\pi f_m t)}$ es periódica, $T_m = \frac{1}{f_m}$, se puede expresar como una serie de Fourier:

$$e^{jI\sin(2\pi f_m t)} = \sum_{k=-\infty}^{\infty} c_k\,e^{j2\pi k f_m t}$$

siendo $c_k = J_k(I)$, la función de Bessel de orden k, evaluada en I:

$$\begin{aligned} c_k &= \frac{1}{T_m}\int_{-\frac{T_m}{2}}^{\frac{T_m}{2}} e^{jI\sin(2\pi f_m t)}\,e^{-j2\pi k f_m t}\text{dt} \\ &= \frac{1}{2\pi}\int_{-\pi}^{\pi} e^{j(I\sin(\tau)-k\tau)}\text{d}\tau \\ &= \frac{1}{\pi}\int_0^{\pi}\cos(k\tau - I\sin\tau)\text{d}\tau \equiv J_k(I) \end{aligned}$$

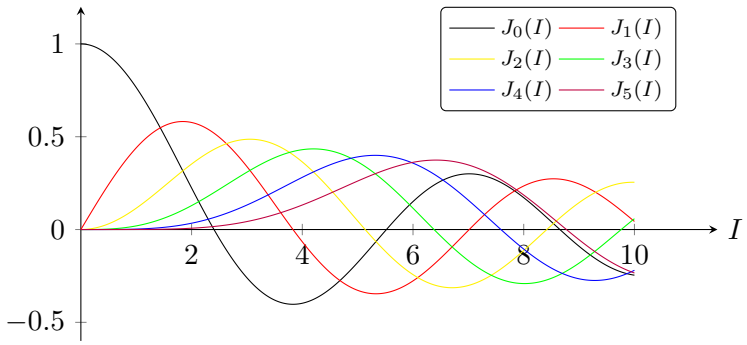
por tanto,

$$x_c(t) = A\,e^{j2\pi f_c t}\cdot\sum_{k=-\infty}^{\infty} J_k(I)\,e^{j2\pi k f_m t} = A\sum_{k=-\infty}^{\infty} J_k(I)\,e^{j2\pi(f_c + k\,f_m)t}$$

$$x(t) = \text{Im}\{x_c(t)\} = A\sum_{k=-\infty}^{\infty} J_k(I)\,\sin 2\pi(f_c + k\,f_m)t$$

Como vemos, aparecen componentes frecuenciales en $f_c + k\,f_m$. Por ejemplo, si seleccionamos $f_m = f_c$, en $f_c(1+k) \quad \forall k$.

La siguiente figura muestra las 6 primeras funciones de Bessel (correspondiente a los 6 primeros componentes frecuenciales) en función del índice de modulación I. Se muestra la parte positiva pues las funciones de Bessel de orden par son pares, e impares las de orden impar.



Si $I = 0$, lógicamente todos los componentes frecuenciales salvo la fundamental son nulos. Pero según crece I, los otros componentes son mayores, el sonido es más *brillante*. La gráficas 1 del artículo citado muestran como es el espectro para distintos valores de I. Un resultado que se indica: el ancho de banda (efectivo) de la señal puede aproximarse por $2\,f_m\,(1 + I)$

En el artículo de Chowning, para simular instrumentos de viento de madera se utilizan relaciones

$$\frac{f_c}{f_m} = \frac{N_1}{N_2}$$

¿Cuál sería la relación de f_c y f_m con la frecuencia fundamental?

Una opción que se contempla es que I cambie con el tiempo, con lo que el ancho de banda del sonido, cambia según avanza la nota (típicamente, mayor en el ataque, menor al disminuir la potencia). Para ello se proponen esquemas (como el de la figura 10) así como valores concretos de los parámetros. para producir distintos instrumentos.

Tareas

- Cree un nuevo instrumento que incorpore el vibrato como parámetro, en vez de como se hizo en el apartado 4.2, donde se implementó como efecto.
  - Los parámetros del vibrato, f_m (en hercios) e I , han de ser accesibles desde el fichero `instruments`.
  - Puede partir del instrumento sinusoidal implementado en el apartado 3.5 o, si los ha implementado, un instrumento por tabla externa para conseguir efectos aún más espectaculares.
  - Compruebe el correcto funcionamiento del vibrato. Puede comparar su resultado con el obtenido usando el efecto proporcionado.
- Cree otro instrumento para síntesis FM, en el que el valor de I sea fijo (pero configurable desde el fichero `instruments`) y f_m sea dependiente de f_o (por ejemplo, proporcional, con constante de proporcionalidad configurable).
  - Actúe sobre estos parámetros para obtener notas con distintos timbres.
  - Si puede profundizar, revise el trabajo de Chowning: esquema 9 y 10, figuras 11 y 12, etc. para preparar instrumentos efectivos.
- Seleccione los parámetros/instrumentos adecuados para algunas de las partituras proporcionadas o use otras de su agrado o composición.