

**Contenido**

[**1.Introducción** 3](#_Toc37353707)

[**2. Nociones básicas** 4](#_Toc37353708)

[2.1 Física del sonido 5](#_Toc37353709)

[2.2 Propiedades acústicas de la música 6](#_Toc37353710)

[**3. Audio digital** 9](#_Toc37353711)

[3.1 Muestreo y cuantificación 9](#_Toc37353712)

[3.2 Formatos más importantes 10](#_Toc37353713)

[**4. MIDI** 12](#_Toc37353714)

[**5. Supercollider** 14](#_Toc37353715)

[5.1 UGens 14](#_Toc37353716)

[5.1.1 Osciladores 15](#_Toc37353717)

[5.1.2 Generadores de ruido 16](#_Toc37353718)

[5.1.3 Operaciones con UGens 18](#_Toc37353720)

[5.2 Filtros 19](#_Toc37353721)

[5.3 Envolvente 22](#_Toc37353722)

[5.4 Canales 24](#_Toc37353723)

[5.5 Operaciones con MIDI 27](#_Toc37353724)

[5.6 Arrays 28](#_Toc37353725)

[5.7 Variables 30](#_Toc37353726)

[5.8 SynthDef 31](#_Toc37353727)

[5.9 Uso de UGens para gestionar UGens 33](#_Toc37353728)

# **1.Introducción**

Este documento corresponde a la memoria del Trabajo de Fin de Grado para la titulación de Ingeniería Informática – Ingeniería de Software.

A lo largo de este trabajo estudiaremos el lenguaje de programación musical “SuperCollider”; desarrollado en 1996 por James McCartney. En 2002 este lenguaje fue lanzado como software gratuito bajo la licencia pública general de GNU, y actualmente es mantenido por los propios usuarios, tratándose así de un proyecto completamente opensource.

Para la realización de este trabajo, estaremos utilizando la versión 3.10.00.

La plataforma ofrece tres componentes principales, los cuales se estudiarán en profundidad más adelante:

* **Scsynth:** Un servidor para audio en tiempo real. Aunque se suele usar desde SuperCollider, se puede acceder a este acceder a él de forma independiente. Incluye una gran cantidad de “UGens” o generadores unitarios, además de poder importar nuevos UGens programados en C++, facilitando la creación de plugins potentes para el lenguaje.
* **Sclang:** Un lenguaje de programación interpretado. Está enfocado, pero no limitado, al sonido. Controla scsynth mediante Open Sound Control. Puede usarse para composición algorítmica y secuenciación, conectar a hardware externo como controladores MIDI, puedes crear aplicaciones visuales o interfaces gráficas para este lenguaje…

Es similar a Ruby o a Smalltalk, y su sintáxis recuerda a Javascript o C.

Las extensiones para SuperCollider programadas por los usuarios se denominan “Quarks”.

* **Scide:** Un editor para sclang con un sistema integrado de ayuda.

A diferencia de otros lenguajes de programación musical, en Supercollider nos encontramos frente a un lenguaje de programación orientado a objetos, cuyo dinamismo y expresividad permite que cada vez más músicos lo utilicen como instrumento principal en sus conciertos o shows, que en este caso se denominan “sesiones de live-coding”, junto con científicos que ha encontrado en él una herramienta para desarrollar y experimentar en el campo de la investigación acústica.

Mi motivación principal a la hora de realizar este trabajo ha sido el poder estudiar por primera vez un lenguaje de programación musical, concepto que dista mucho de las materias impartidas en la titulación y que me resulta de especial interés ya que todas mis aficiones giran en torno a la música en directo y la grabación y edición musical. Un lenguaje de programación de estas características supone una herramienta más, bastante útil, con el fin de desarrollar mis conocimientos y poder experimentar en el campo del audio digital.

Para realizar este trabajo hemos tomado como fuentes de información principal la propia documentación de Supercollider, disponible en su página web, así como videotutoriales alojados en Youtube.

# **2. Nociones básicas**

En este capítulo trataremos varios conceptos básicos sobre el sonido desde los puntos de vista físicos, musicales y digitales, a fin de sentar una base que nos ayude a estudiar el lenguaje de programación en cuestión en mayor profundidad.

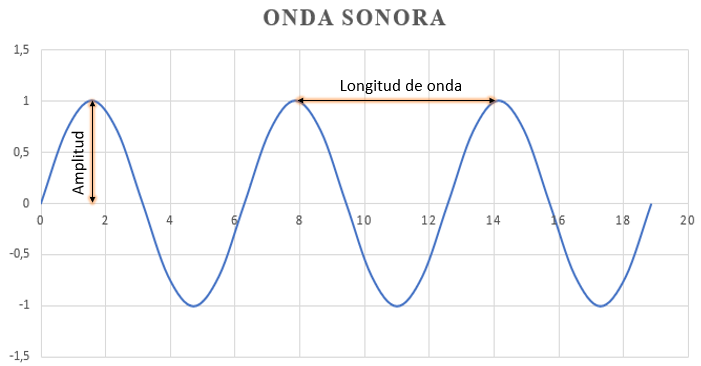
## 2.1 Física del sonido

Desde un punto de vista físico podemos definir el sonido como la propagación de ondas que se originan por la vibración de un cuerpo a través de un fluido o un medio elástico, generalmente el aire. Estas ondas comparten las características de las ondas mecánicas:

• **Frecuencia (f):** Medida en Hercios (Hz), describe la cantidad de ciclos o perturbaciones completadas por unidad de tiempo, normalmente medida en segundos. Representa la altura del sonido, ya que en base a su frecuencia distinguimos sonidos graves y agudos. Entre los 20 y los 20.000 Hz consideramos frecuencias audibles, puesto que por encima y por debajo de esa franja, los sonidos no son perceptibles por el oído humano.

• **Amplitud:** Es la distancia entre el punto más alto y el más bajo de una onda. Representa la intensidad del sonido, lo que llamaríamos comúnmente “volumen”.

• **Longitud de onda:** Mide la distancia que recorre una onda en un periodo concreto de tiempo. Se aplica en el caso de ondas periódicas. También es conocida como periodo espacial, que es el inverso de la frecuencia y representa el tiempo que tarda una onda en completar un ciclo.

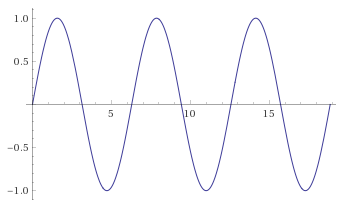


## 2.2 Propiedades acústicas de la música

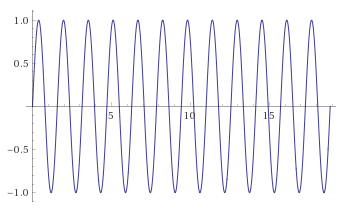
Dentro de un sonido podemos encontrar cuatro propiedades que percibimos de forma subjetiva y que vienen derivadas de las características físicas del sonido. Estas propiedades son: altura, duración, intensidad y timbre.

Gracias a estas propiedades podemos distinguir un sonido agradable, producido por una vibración armónica y regular, de un ruido.

**• Altura o tono:** Es una propiedad que percibimos de forma subjetiva y que deriva de la frecuencia. Mientras mayor frecuencia decimos que un sonido tiene un tono más “alto” y viceversa. Nos referimos a estos sonidos respectivamente como “agudos” y “graves”. A diferencia de la frecuencia, la altura es subjetiva y por tanto no es cuantificable, lo que hace que dependiendo del receptor y la situación se perciba de forma distinta.

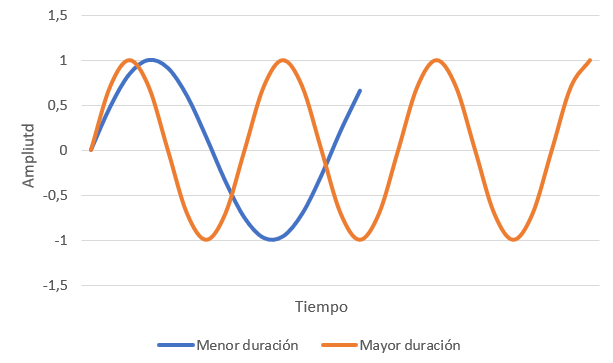


Sonido grave

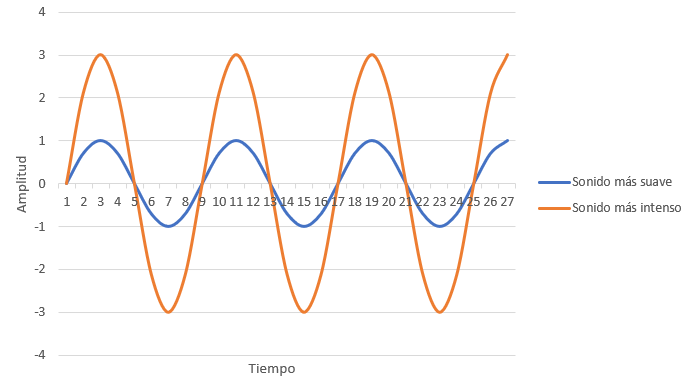


Sonido agudo

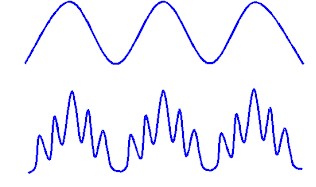
• **Duración:** Representa el tiempo que se extiende un sonido desde su inicio hasta su extinción. En función de la duración existen sonidos largos, medios, cortos, muy cortos… Estos sonidos de duración variable combinados originan ritmos.



**• Intensidad:** Es la cantidad de energía contenida en un sonido; hace que podamos escucharlo desde una distancia mayor o menor. La intensidad deriva de la amplitud y la potencia acústica de un sonido, y se mide en decibelios (dB). Un sonido es audible a partir de los 0 dB y comienza a causar dolor y malestar al oído humano a partir de los 130 dB. En la propagación real, cambios físicos en el aire como la humedad, la presión o la temperatura hacen que el sonido se amortigüe o se disperse.



**• Timbre:** Permite que distingamos una misma nota tocada por instrumentos musicales distintos. Representa la forma de onda, que nos ayuda a distinguir la fuente de sonidos con la misma frecuencia e intensidad. Esta propiedad agrupa las tres anteriores y solo se puede describir, no medir.



Ondas igual frecuencia (nota) pero distinto timbre

# **3. Audio digital**

Con las características principales del sonido ya definidas, es el momento de plantearnos cómo lograr convertirlas en información procesable por un ordenador y poder manipularlas.

Las técnicas que se emplean se basan en imitar el funcionamiento del oído humano y convertir esas ondas mecánicas en impulsos eléctricos que podamos almacenar como tal o convertir en señales digitales.

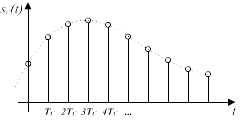
Dicha codificación consiste en una secuenciación de valores enteros que se obtienen mediante el muestreo y la cuantificación digital.

## 3.1 Muestreo y cuantificación

El **muestreo** es el proceso por el cual se captura la amplitud de la señal eléctrica en intervalos regulares de tiempo, denominados “tasa de muestreo”.

Para abarcar todo el espectro audible por el oído humano suele ser suficiente con una tasa de 40kHz, lo que implica 40000 muestras por segundo de audio capturado.

Por otro lado, la **cuantificación** se refiere al proceso de transformar las muestras fijadas durante el muestreo, que suele ser un valor de tensión, en un valor entero dentro de un rango determinado. Con una cuantificación lineal de 8 bits discriminamos 256 niveles equidistantes de señal.



Muestreo digital de una señal de audio

## 3.2 Formatos más importantes

Una vez definida la forma de transformar ondas acústicas analógicas en señales digitales que pueden ser procesadas, cabe destacar los formatos más importantes de estas.

• **Formatos PCM:** Denominados PCM por sus siglas (Pulse Coded Modulation), albergan en su totalidad la información que se obtuvo del convertidor analógico a digital sin omitir nada, otorgándole una calidad mejor al resto de formatos.

Dentro de los PCM tenemos los formatos WAV, AIFF, SU, AU y RAW, cuyo encabezado posee unos 1000 bytes al comienzo del archivo.

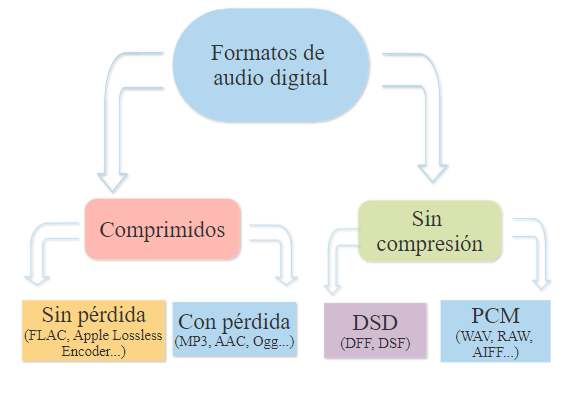
• **Formatos DSD:** Los formatos Direct-Stream Digital (marca registrada de Sony Corp. Y Philips) se basan en tecnologías de registro y reconstrucción de señales de audiofrecuencia, que se usaron por primera vez en el formato Super Audio CD y actualmente en los formatos DSF y DFF.

Esto se hace mediante el método de Pulse Density Modulation (PDM), el cual se diferencia del PCM en poseer una profundidad de bits bajísima, de tan solo 1 bit, en contraste con una gigantesca frecuencia de muestreo de 2,8224 Mhz.

**• Formatos comprimidos:** Para evitar usar tanta memoria como los formatos mencionados anteriormente, existen formatos que comprimen la información, como el archiconocido MP3, AAC, Ogg…

Estos formatos están basados en algoritmos que eliminan de las pistas de audio aquella información que no es perceptible por nuestro oído, llegando a reducir el espacio en memoria de un archivo hasta en más de una decena de veces en comparación con el mismo archivo en formato PCM.

Esta pérdida de información hace que a estos formatos se les considere formatos comprimidos “con perdida”, aunque también existen formatos comprimidos sin pérdida o “lossless” como pueden ser FLAC o el Apple Lossless Encoder, cuyo tamaño ronda la mitad de un archivo PCM.



• **Formatos descriptivos:** Más conocido como archivos “MIDI”, no pertenecen técnicamente al audio digital, pero sí a la informática musical.

Un archivo MIDI no almacena sonido capturado por una grabadora de ningún tipo, sino que está compuesto por indicaciones para que cualquier dispositivo MIDI como podría ser un sintetizador, un launchpad o una guitarra MIDI interpreten una serie de notas y acciones; haciendo que sea un equivalente moderno a las partituras, con los nombres de los instrumentos, las notas, tiempos y más indicaciones.

# **4. MIDI**

En este capítulo trataremos más a fondo qué es el estándar MIDI y cómo funciona.

El estándar MIDI (Musical Instrument Digital Interface) surge en 1983 como un convenio del que resultó la MMA (MIDI Manufacturers Association).

Dicho estándar tecnológico describe un protocolo, una interfaz digital y conectores con el objetivo de hacer que ciertos instrumentos musicales electrónicos y computadores puedan relacionarse y comunicarse entre ellos.

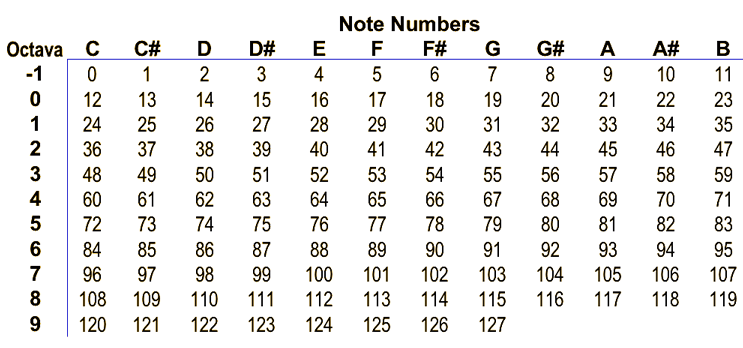
Una sola conexión de este tipo tiene la capacidad de transmitir hasta 16 canales de información que se pueden conectar a distintos equipos.

Los mensajes de evento descritos en el protocolo MIDI especifican notación musical, tono, velocidad, señales de control de parámetros musicales (dinámica, vibrato, tempo…).

A continuación se muestra una tabla que describe los comandos admitidos en estos mensajes:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Mensaje*** | ***Código*** | ***Parámetro 1*** | ***Parámetro*** |
| Note Off | 0x8 | Número de nota | Velocidad |
| Note On | 0x9 | Número de nota | Velocidad |
| Note Aftertouch | 0xA | Número de nota | Presión |
| Controller | 0xB | Número de controlador | Controller Value |
| Program Change | 0xC | Número de programa | - |
| Channel Pressure | 0xD | Presión | - |
| Pitch Bend | 0xE | Pitch Value (LSB) | Pitch Value (MSB) |

Los mensajes Note On y Note Off están codificados en un rango que admite las 88 teclas de un piano más algunas notas extra que no existen en instrumentos analógicos:



Estos mensajes se envían a través de un cable MIDI a los demás equipos conectados, pero también pueden ser grabados en secuenciadores, tanto software como hardware, con el fin de poder editar esta información a posteriori.

La mayor ventaja de este formato es poder codificar composiciones completas en un espacio de tan reducido como un par de kilobytes, así como poder manipular y editar las distintas pistas asignadas a cada instrumento.

Además, cabe recalcar que a raíz de este protocolo han ido apareciendo una serie de extensiones que permiten desde controlar el transporte de dispositivos hardware de grabación (MIDI Transport Control) hasta poder sincronizar máquinas, sonidos y pirotecnia (MIDI Show Control) para exhibiciones de museo, escenarios de rodaje…

# **5. Supercollider**

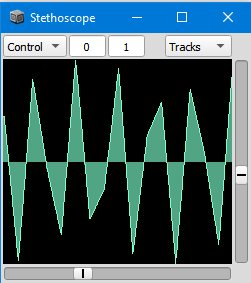
## 5.1 UGens

En Supercollider, el concepto de UGen ó Unit Generator define a los objetos que producen señales; sus nombres siempre comienzan por mayúscula y a su conjunto nos referiremos como *patch.*

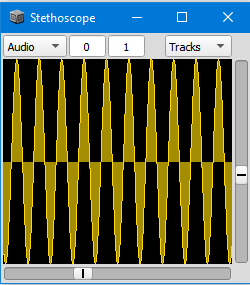
Dentro de los UGens encontramos dos tipos:

* **Audio Rate (.ar):** Cuando un UGen recibe un mensaje ar. lo ejecuta a una velocidad de 44k muestras por segundo. Si el UGen va a ser parte de una cadena de audio que vaya a ser escuchada se le debe enviar un mensaje de este tipo.
* **Control Rate (.kr):** En este caso, el UGen corre a velocidad de control. Producen una muestra por cada 64 producidas por el UGen a velocidad de audio. Se usan como moduladores para dar forma a la señal de audio, y son más baratos computacionalmente.

A continuación, tenemos una representación gráfica, producto de visualizar mediante la función *scope* un oscilador sinusoidal a 440 Hz.



SinOsc.kr(440)



SinOsc.ar(440)

)

### 5.1.1 Osciladores

Los osciladores en SuperCollider se rigen por una serie de argumentos que actúan como los parámetros que definen el comportamiento de estos.

Los más comunes son:

* **Frecuencia:** Su valor por defecto es de 440Hz, que representa la nota La índice 5.
* **Fase: Por defecto** vale 0, representando al inicio del ciclo. Su valor puede oscilar de 0 a 2pi.
* **Amplitud:** Su valor por defecto es 1, no alterando la señal.
* **Suma:** Es un número que se le suma a la señal. Se aplica justo después de multiplicar la señal por el valor del parámetro amplitud. Su valor por defecto es 0.

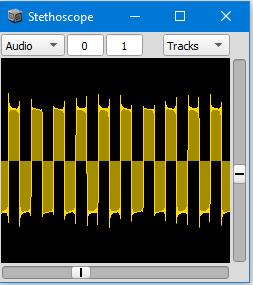
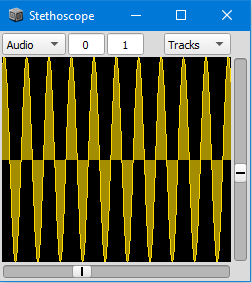
Aunque existen UGens que poseen argumentos propios, estos son cuatro son generales para casi todos.

A continuación, diseccionaremos los osciladores más comunes que podemos encontrar en SuperCollider.

* **SinOsc:** Es un oscilador de onda senoidal. Funciona tanto con .ar como con .kr y recibe como atributos (en este orden): frecuencia, fase, amplitud, suma.
* **Pulse:** Oscilador de onda cuadrada. Solo funciona con .ar. Recibe como argumentos: frecuencia, ancho de banda, amplitud y suma.

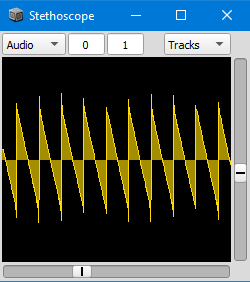
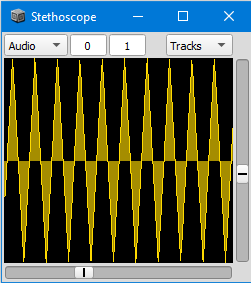
El ancho de banda está en el intervalo abierto (0, 1) y gestiona el timbre de la señal. Por defecto su valor es 0,5.

* **LFTri:** Oscilador de onda triangular. Funciona tanto con .kr como con .ar. Recibe como parámetros: frecuencia, fase, amplitud y suma
* **Saw:** Es un oscilador de dientes de sierra. Solo funciona con .ar y recibe como parámetros: frecuencia, amplitud y suma.



Pulse

SinOsc



LFTri

Saw

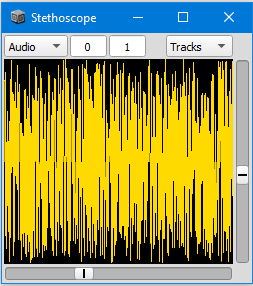
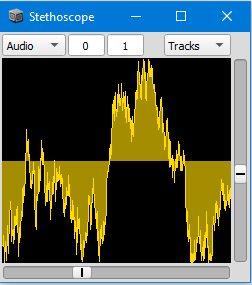
### 5.1.2 Generadores de ruido

Por último. hablaremos de los generadores de ruido más utilizados dentro de SuperCollider.

* **WhiteNoise:** Es un generador de señales aleatorias, que produce señales en todas las frecuencias y con la misma potencia.
* **PinkNoise:** En este caso, las señales aleatorias generadas decrecen en un factor de 3dB por cada octava, misma proporción en la que aumenta el ancho de banda.
* **BrownNoise:** Es similar al PinkNoise, solo que en este caso decrece a un nivel de 6dB por cada octava.
* **Dust:** Este UGen genera impulsos aleatorios.

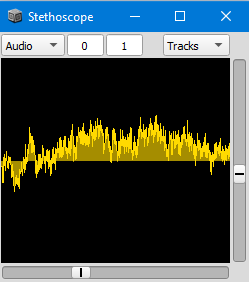
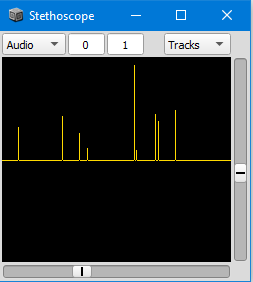
WhiteNoise

BrownNoise



BrownNoise

WhiteNoise



### 

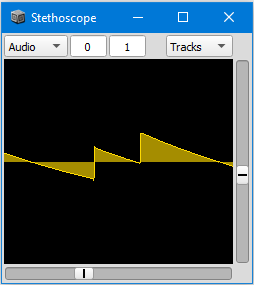
Dust

PinkNoise

### 5.1.3 Operaciones con UGens

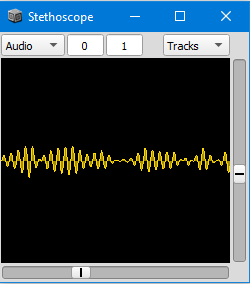
Una vez comprendido el concepto de UGen que maneja SuperCollider nos encontramos con que podemos crear nuevos timbres a partir de estos mediante sumas y multiplicaciones.

Las sumas dan como resultado un nuevo timbre mediante síntesis aditiva, en el que se siguen percibiendo los dos sonidos previos. Las amplitudes de los dos UGens se suman, no debiendo superar el valor 1.0.



{Saw.ar(30,0.3)+Saw.ar(70,0.3)}

Por otro lado, al multiplicar dos UGens obtenemos un timbre de mayor complejidad.



{SinOsc.ar(1360,0,0.6)\*WhiteNoise.kr(0.3)}

## 5.2 Filtros

En electrónica, un filtro es un elemento que en función de uno parámetros discrimina una señal de entrada, realizando cambios en su salida.

Mientras nos refiramos a los filtros en SuperCollider trataremos como primer parámetro la señal a filtrar y tras esta la frecuencia de corte.

A continuación, estudiaremos los tres filtros principales que encontramos en este lenguaje de programación. Los tres funcionan tanto en .ar como en kr., teniendo como único requisito que tanto la señal de entrada como el filtro a aplicar tenga el mismo rate.

* **HPF (High Pass Filter):** El filtro paso alto atenúa todas las frecuencias inferiores a la de corte, solo permitiendo pasar las que estén por encima de esta.

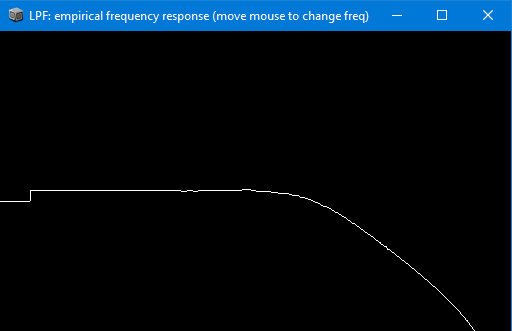
Recibe como argumentos una señal de entrada, una frecuencia de corte, una amplitud y la suma.

* **LPF (Low Pass Filter):** Los filtros paso bajo funcionan al contrario que los filtros HPF; solo permiten pasar las frecuencias inferiores a la frecuencia de corte que recibe como argumento. Estos filtros también reciben los mismos argumentos que su contraparte.
* **BPS (Band Pass Filter):** Los filtros paso banda solo permiten el paso de frecuencias que se encuentren dentro de una banda concreta y limitada por una frecuencia suelo y una frecuencia techo. Entre ambas se encuentra la frecuencia de corte.

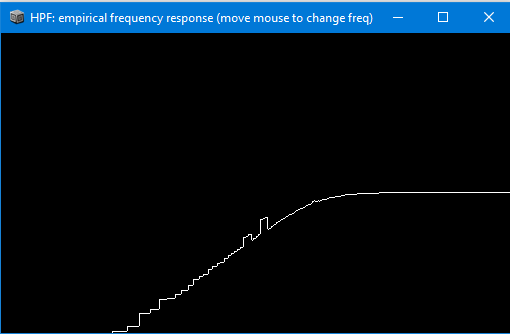
Para calcular las cotas inferiores y superiores aplicamos las siguientes fórmulas:

De esta forma, si conocemos las cotas que buscamos es tan sencillo como aplicar las siguientes fórmulas para hallar la frecuencia de corte y el ancho de banda buscados:

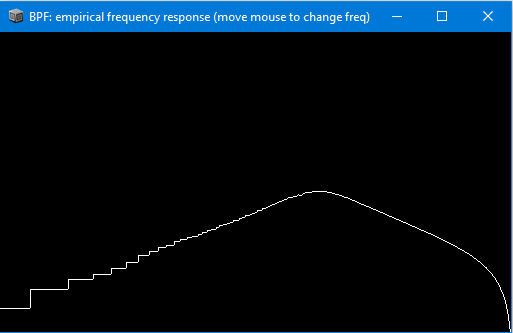
Este filtro recibe como parámetros una señal de entrada, una frecuencia de corte, qr (que representa al ancho de banda partido de la frecuencia de corte), una amplitud y una suma.



Filtro paso bajo



Filtro paso alto



Filtro paso banda

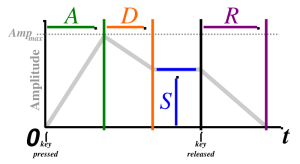
## 5.3 Envolvente

La envolvente representa la persistencia en el tiempo de un sonido frente a su amplitud, es decir, con qué intensidad se escucha el timbre a lo largo del ciclo de vida que dura desde que se produce dicho sonido hasta que se extingue.

Por poner un ejemplo más sencillo de comprender podríamos pensar en el sonido que produce un bombo de batería al golpearlo con una baqueta; este sonido nace siendo muy intenso, pero rápidamente se diluye hasta reducir su amplitud de forma casi total.

Una vez hemos definido el concepto de envolvente, procederemos a estudiar sus cuatro componentes más comunes, popularmente denominados “ADSR”:

* **Attack:** El ataque representa cuánto tarda el sonido en alcanzar su amplitud máxima una vez se ha producido. En los instrumentos mecánicos este punto es prácticamente instantáneo.
* **Decay:** El decaimento es la cantidad de tiempo que tarda el sonido en estabilizarse una vez pasado su punto de amplitud más alto.
* **Sustain:** El sostenimiento mide cuánto tiempo dura esa estabilización que ocurre tras el decaimiento hasta que la nota se libera (deja de sonar).
* **Release:** La liberación representa cómo de rápido se desvanece una nota una vez termina (dejamos de pulsar una tecla de un piano, por ejemplo). Este tiempo puede ser más largo si utilizamos un sintetizador con algún efecto de sustain.



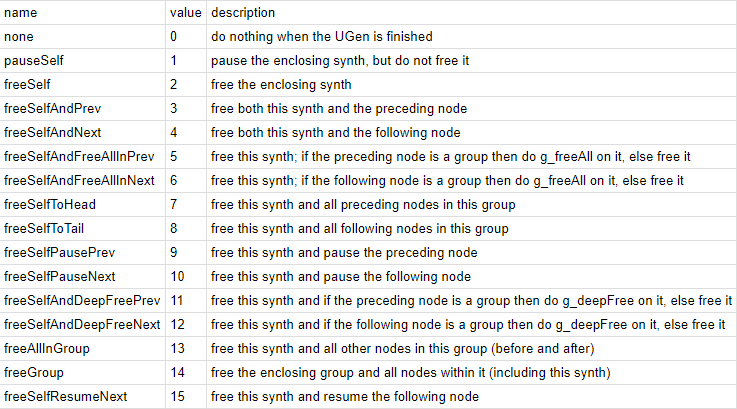
Representación gráfica de ADSR

Cabe destacar que no todas las envolventes necesitan de estas cuatro propiedades; puede haber envolventes que solo dependan de dos de ellas.

En SuperCollider manipulamos las envolventes mediante la clase *EnvGen,* la cual estudiaremos en modo *kr* (control rate) ya que la usaremos para modificar señales.

A esta clase se la llama mediante EnvGen.kr(envolvente, gate, doneAction). Estos tres parámetros funcionan de la siguiente forma:

* **Envolvente:** Podemos utilizar alguna de las muchas envolventes incluidas en SuperCollider. Por ejemplo; *Env.adsr(attack, decay, volumen, release)*, se usa para sonidos sostenidos y recibe como tercer argumento volumen en vez de tiempo. También tenemos *Env.perc(attack, release)* que usamos como envolvente para percusión y por último tenemos el ejemplo de *Env.asr(attack, volumen, release)* en el cual la nota se mantiene hasta que nosotros lo indiquemos y ya procede con su liberación.
* **Gate:** Activa la envolvente y la mantiene mientras su valor sea superior a 0. Si la envolvente tiene una duración fija, como es en el caso de Env.perc; donde su duración es igual a la suma del attack y el release, sirve simplemente como un activador. Por otro lado, si usamos una envolvente con sustain, esta se aplica hasta que el valor sea 0, en cuyo momento comienza la fase de release.
* **doneAction:** Es un entero que representa la acción que se realizará una vez se cierre la envolvente. Las acciones posibles van del 0 al 15 y se detallan en la siguiente tabla:



## 5.4 Canales

Tanto el sonido analógico como el digital pueden ser reproducidos en uno o más canales, siendo las formas más comunes el mono (un solo canal) y el estéreo (dos canales; uno izquierdo y uno derecho).

Las tarjetas de sonido nos dan la posibilidad de reproducir el sonido en más canales (2, 4, 8… incluso 9, como es el caso del audio 9.1, que se reproduce mediante tres canales frontales, dos laterales, tres traseros y uno en el techo).

En SuperCollider tenemos una serie de clases que nos ayudan a poder trabajar de forma multicanal:

* **Out.ar(canal, señal):** Reproduce el sonido a través de un canal concreto que sirve como offset para la distribución del sonido.

Donde el canal es un número entero que representa el canal de salida del sonido; siendo 0 = canal izquierdo, 1 = canal derecho y 2,3,4,5… multicanal.

La señal que recibe es cualquier oscilador que creemos, pudiendo este tener una envolvente aplicada.

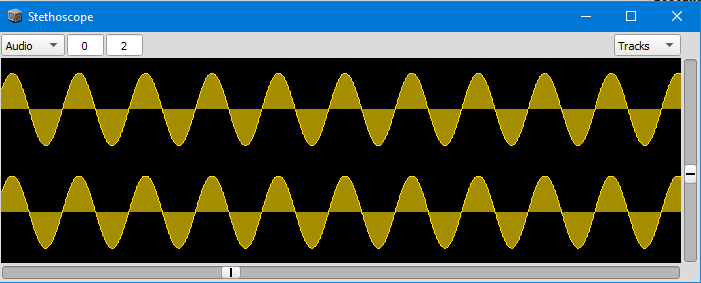
* **Pan2.ar(señal, posición):** En este caso el sonido es distribuido por los canales de salida, haciendo que el sonido no tenga mayor volumen si se reproduce en mono ni menor volumen al reproducirlo en multicanal.

Al igual que Out, recibe una señal que puede ser cualquier oscilador, pero en vez de recibir un canal de salir recibe una posición, siendo -1 = izquierda, 1 = derecha y el resto del intervalo siendo una extrapolación del sonido entre los dos canales.

A continuación, se adjuntan algunos ejemplos interesantes del uso de estas clases:

En la siguiente imagen se muestra el resultado de ejecutar: **{Out.ar(0,Pan2.ar(SinOsc.ar(440),0))}.scope**

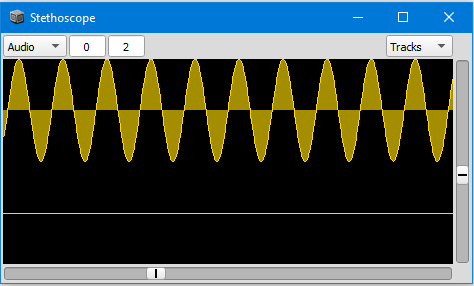
Esto genera un oscilador senoidal a 440Hz y lo reproduce por el canal izquierdo mediante la clase *Out*, a la vez que en vez de usar el oscilador como tal como argumento, lo utiliza dentro de la clase *Pan2* con posición 0, haciendo que se reproduzca por los dos canales con posición central y distribuyendo por ellos su sonido de forma uniforme.



{Out.ar(0,Pan2.ar(SinOsc.ar(440),0))}.scope

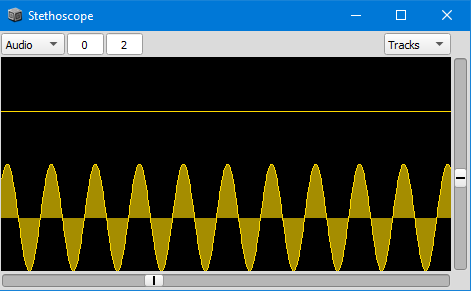
También podemos ejecutar el siguiente trozo de código:

**{Pan2.ar(SinOsc.ar(440),-1)}.scope** para reproducir el mismo oscilador senoidal solo por el canal izquierdo.



{Pan2.ar(SinOsc.ar(440),-1)}.scope

Y por último podemos ejecutar **{Pan2.ar(SinOsc.ar(440),1)}.scope** para reproducirlo por el canal derecho.



{Pan2.ar(SinOsc.ar(440),1)}.scope

## 5.5 Operaciones con MIDI

A la hora de trabajar con notas musicales en SuperCollider podemos hacerlo mediante interacciones con la representación numérica de estas en el protocolo MIDI de forma sencilla.

Para ello utilizaremos dos métodos:

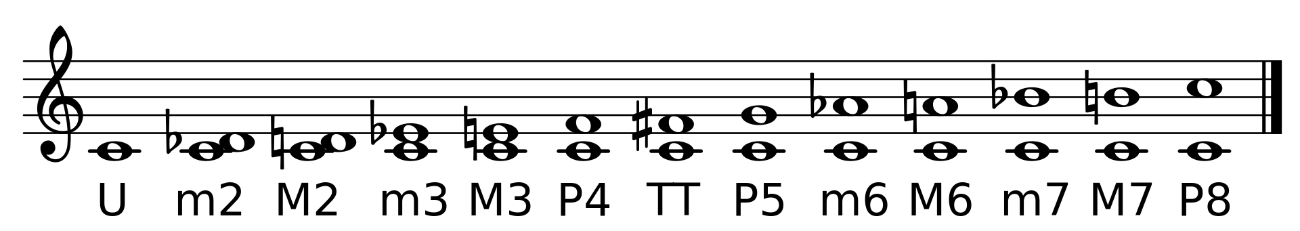
* **.midicps:** Este método convierte cualquier nota en notación MIDI a su equivalente en Hercios para su cómoda manipulación en SuperCollider.

Por ejemplo; la nota Re sería representada en notación MIDI por el número 62, y para utilizarla en el entorno de SuperCollider la llamaríamos mediante *62.midicps*

* ***.cpsmidi:***Por otra parte, este método es la contraparte de *.midicps*, y convierte una frecuencia dada en Hercios por su equivalente en notación MIDI.

De esta forma, la nota La, cuya frecuencia es de 440 Hercios, podría ser convertida a notación MIDI mediante la llamada *440.cpsmidi*, que daría como resultado 69 en notación MIDI.

SuperCollider también nos ofrece la posibilidad de utilizar el método *.midiratio* para trabajar con intervalos expresados en semitonos.



Intervalos harmónicos a partir de la nota do

De esta forma podemos aplicar dicho método sobre un número entero y multiplicado por una nota con los siguientes resultados:

|  |  |  |
| --- | --- | --- |
| **Número en *.midiratio*** | **Nombre del intervalo** | **Distancia entre tonos y semitonos** |
| 0 | Unísono | Mismo sonido |
| 1 | Segunda menor | 1 semitono |
| 2 | Segunda mayor | 1 tono |
| 3 | Tercera menor | 1 ½ tonos |
| 4 | Tercera mayor | 2 tonos |
| 5 | Cuarta justa | 2 ½ tonos |
| 6 | Cuarta aumentada (Tritono) | 3 tonos |
| 7 | Quinta justa | 3 ½ tonos |
| 8 | Sexta menor | 4 tonos |
| 9 | Sexta mayor | 4 ½ tonos |
| 10 | Séptima menor | 5 tonos |
| 11 | Séptima mayor | 5 ½ tonos |
| 12 | Octava justa | 6 tonos |

Los valores numéricos sobre los que usamos *.midiratio* también se pueden aplicar en forma de números negativos, haciendo de esta forma que en vez de aumentar un semitono por cada uno, se disminuya.

Así tendríamos que si la nota La (69 en notación MIDI) la multiplicamos por *8.midiratio* obtenemos su sexta menor:

***69.midicps \* 8.midiratio***

Por otro lado, si la multiplicamos por *-5.midiratio* obtenemos su cuarta descendente:

***69.midicps \* -5.midiratio***

## 5.6 Arrays

Al igual que en la mayoría de los lenguajes de programación modernos, SuperCollider nos permite trabajar con *arrays*, que representan un conjunto de elementos ordenados, los cuales se declaran dentro de corchetes y separados por comas.

Los *arrays* pueden ser utilizados para operar entre ellos, aunque con ciertas particularidades. Si operamos con dos *arrays* del mismo tamaño, la operación se aplica entre sus miembros uno a uno:

Sin embargo, si los *arrays* poseen un distinto número de elementos, la operación se ejecuta miembro por miembro hasta que el de menor tamaño llega a su fin, momento en el que se continúa operando a partir del principio de este:

A continuación, explicaremos algunos de los métodos más útiles que nos ofrece SuperCollider para trabajar con estos *arrays*, algunos de los cuales no facilitarán mucho el trabajo dependiendo de lo que necesitemos:

* **.size:** Devuelve como output el tamaño del *array.*
* **.scramble:** Altera de forma aleatoria el contenido del *array* sobre el que se ejecuta.
* **.mirror:** Genera un *array* que contiene un espejo del primero, es decir, coge el punto medio del array y a continuación añade los mismos elementos en sentido inverso.

Por ejemplo *[1,2,3].mirror* nos devuelve el *array* [1,2,3,2,1]

* **.choose:** Escoge un elemento al azar del *array.*
* **.wchoose:** Es similar a *.choose*, solo que recibe como parámetro un *array* de pesos probabilísticos con una entrada por cada elemento del array, en el cual se base para asignar la probabilidad de devolver cada uno de estos elementos. Debemos tener en cuenta de que la suma de los pesos no puede ser mayor que 1.
* **.sum:** Devuelve el valor resultante al sumar todos los elementos del *array*.
* **.find:** Recibe como argumento un objeto, el cual busca dentro del *array*. Si dicho objeto está contenido, devuelve su posición dentro de este; por otro lado, si no se encuentra el objeto en cuestión dentro del *array*, devuelve *nil*.
* **Acceder a datos de un array:** Si asignamos un array a una variable (por ejemplo, arr = [1,2,3]) podemos acceder al dato que se encuentre en su posición i mediante la llamada arr[i].

Para añadir un nuevo valor a un *array*, emplearemos la función *add*. Por ejemplo, *arr.add(3)* añadiría el valor 3 al final de array.

* **Métodos comunes:** Cabe destacar que cualquier método aplicable a u número es aplicable a un *array*. Por ejemplo, podemos ejecutar los antes mencionados *.midiratio, .midicps y .cpsmidi* sobre cualquier *array* numérico.

La clase array también posee constructores para crear *arrays* de datos con estructuras concretas. A continuación. veremos los más importantes:

* **Array.fill(i,”x”):** Crea un *array* que contiene el valor *“x”* repetido *i* veces.
* **Array.rand(i,x,z):** Genera un *array* que contiene *i* valores numéricos dentro, los cuales se generan aleatoriamente dentro del rango delimitado entre *x* y *z*.
* **Array.series(i,x,z):** Genera un *array* de *i* elementos. Este *array* comienza en *x* y se incrementa sumando este valor a *z* en cada iteración.
* **Array.geom(i,x,z):** Funciona de forma similiar a *array.series()*, a diferencia de que en este caso, en cada iteración el valor se multiplica por *z* en vez de sumarlo.



Array con las notas de la escala menor natural

## 5.7 Variables

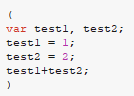
El concepto de variable es común a todos los lenguajes de programación. Representan espacios de memoria en los que se almacenan datos bajo un identificador, con el objetivo de poder acceder a ellos y manipularlos de forma cómoda.

Normalmente se usan para guardar información, asignar valores entre variables, representar información para imprimirla por pantalla…

Las variables no tienen un número máximo ni un nombre predefinido; podemos definir todas las que queramos o necesitemos y asignarles los nombres que nos resulten más claros a la hora de interactuar con ellas.

La forma de declararlas en SuperCollider es la siguiente:

1. Escribimos *var* seguido de un espacio y el nombre que deseemos asignar.
2. Las separamos entre sí mediante comas.
3. Una vez hayamos asignado un valor a cada variable la cerramos mediante punto y coma (;).
4. Si el nombre de nuestra variable solo consiste en una letra no es necesario declararla. Aunque debemos tener en cuenta que la variable s está reservada para referirse al servidor e interactuar con él.
5. El nombre de una variable siempre debe comenzar por minúsculas.



Ejemplo de declaración de variables. El resultado mostrado por consola sería '3'

También podemos guardar osciladores concretos dentro de variables, y a su vez estas variables dentro de arrays.

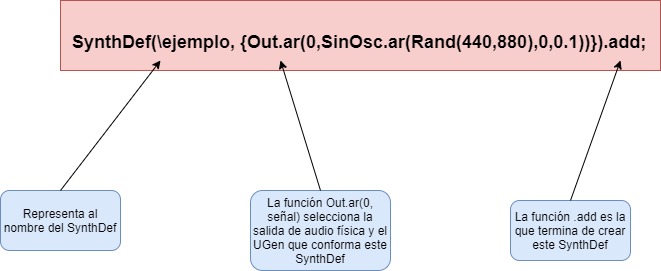
## 5.8 SynthDef

SuperCollider utiliza la definición de SynthDef como plantillas para hacer nuevos synth. Los SynthDef son la forma que utilizaremos en SuperCollider para crear sonidos.

Un SynthDef es la representación en el lado del cliente de un synth, el cual podremos reproducir, guardar en memoria…

Una vez definidos, estos synth se pueden manipular independientemente unos de otros, y son muy útiles para utilizarlos como “recetas” que podemos usar para diseñar y manipular varios synth a partir de un solo SynthDef.

La sintaxis de estos es la siguiente:



Para hacer sonar este SynthDef simplemente tendríamos que ejecutar la siguiente línea de código:

**Synth.new(\ejemplo);**

Una vez definido el SynthDef y visto como reproducirlo podríamos encapsular este Synth dentro de una variable, y sobre ella aplicar el método .set para alterar los argumentos propios del SynthDef del que parte, provocando alteraciones en el sonido en tiempo en real.

Cuando asignamos un Synth a una variable y lo reproducimos, recibimos por consola el nombre de este seguido de un número separado por dos puntos. Este número representa la identificación del nodo en el que se encuentra el Synth (NodeID).

Este identificador se adquiere al generar el Synth, que busca un identificador disponible y se lo autoasigna.

El servidor de síntesis maneja un árbol de nodos, que son objetos con los que se puede mantener una comunicación. Este árbol define el orden en el que los synth son ejecutados.

Los SynthDef son la estructura más compleja de manejar dentro de SuperCollider, pero al mismo tiempo son donde realmente reside la capacidad creativa de este lenguaje. Son la base fundamental para crear composiciones y poder reproducirlas, pero también son muy útiles para poder realizar interpretaciones en vivo, también conocidas como sesiones de livecoding, en las que se van alterando las propiedades de los SynthDef para conseguir patrones rítmicos y tonales variantes, que dan lugar a melodías complejas.



Sesión de live-coding en SuperCollider por Juan Romero y Patrick Borgeat

## 5.9 Uso de UGens para gestionar UGens