0 INTRODUCCIÓN AL CSOUND

0.1 ¿Qué es?

El Csound es un potente y versátil programa de síntesis. Posee más de 450 módulos de procesamiento de sonido y convierte un ordenador personal en una estación de trabajo musical, capaz de emular cualquier sintetizador o procesador de audio comercial. Csound fue creado en 1985 por Barry Vercoe en el MIT. Hay versiones para Unix, Linux, MSDOS, Windows y Mac.

Csound es un generador y procesador de sonido digital y se manipula como un lenguaje de programación. Aúna en una única herramienta muchos de los conceptos vistos en este curso: síntesis digital de sonido, procesamiento y secuenciación. Funciona compilando unas especificaciones acerca de instrumentos musicales virtuales escritas en un fichero llamado *orquesta*. Junto a las especificaciones de estos instrumentos se compilan unas órdenes para su control referidas a notas, tablas de ondas y parámetros de control escritas en otro fichero llamado *partitura*. El resultado de la compilación es la generación de una señal sonora digital que se almacena en un fichero que puede oírse con cualquier editor / reproductor capaz de ello.

Así pues cualquier trabajo en Csound se basa en la redacción de dos ficheros de texto: una orquesta (de extensión típicamente "`.orc`") que contiene las especificaciones de los instrumentos y una partitura (de extensión típicamente "`.sco`" – de *score* = partitura –) que se compilan juntos con un comando del tipo:

```
csound  ejemplo.orc  ejemplo.sco
```

creando en el disco un fichero de nombre `test` que contendrá el sonido generado.

0.2 ¿Cómo conseguirlo? y ¿dónde documentarse?

Existen multitud de páginas web dedicadas al Csound y a lo que se puede hacer y a lo que se ha hecho con él. A continuación se muestran unas cuantas, seleccionadas:

Csound Front Page	www.csounds.com
Csound Resources for Linux	www.bright.net/~dlphilp/linux_csound.html
Csound para windows	www.dlsi.ua.es/asignaturas/sds/Practicas
Manuales en distintos formatos e idiomas	www.lakewoodsound.com/csound
	web2.airmail.net/dboothe
	home.worldonline.es/mvalero
	ftp://ftp.musique.umontreal.ca/pub/mirrors/dream
Ejemplos sonoros	mitpress.mit.edu/e–books/csound/fpage/univ/ Berklee/berklee.html

No existen muchos libros de referencia sobre Csound, pero el mejor es: R. Boulanger. "The Csound Book", MIT Press. 2000.

Vamos a centrarnos primero en los principales aspectos de la sintaxis de Csound y la estructura general de los ficheros de orquestas y de partituras. Después veremos la sintaxis y funcionamiento de una selección de generadores y modificadores de señal de audio y de control.

0.3 Estructura de los ficheros de orquestas

Un fichero de orquesta de Csound consta de dos partes: la sección de cabecera y la de especificación de instrumentos, constituida por una secuencia de definiciones de instrumentos, a cada uno de los cuales se asignará un número de identificación.

Tanto estos ficheros como los de partitura admiten comentarios usando el carácter '`;`', el cual situado en una línea hace que el resto de esa línea sea ignorado por el compilador.

En Csound cada sentencia debe situarse en una línea. Si, debido a una gran longitud, una sentencia no cupiera en una línea, puede seguirse en la siguiente acabando la primera con el carácter '\'.

0.3.1 Cabecera

La cabecera contiene sentencias de inicialización de variables globales. En ella se definen las frecuencias de muestreo y de control con las que los instrumentos generarán las señales, así como el número de canales de salida. Por ejemplo:

```
sr = 44100
kr =  4410
ksmps = 10
nchnls = 1   ; son todos opcionales puesto que tienen valores por defecto (estos mismos).
```

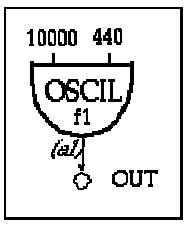
- **sr** : establece la frecuencia de muestreo en muestras por segundo y por canal. El valor por defecto es 44100, que genera sonido con calidad CD.
- **kr** : establece la frecuencia de control en valores por segundo. Por defecto vale 4410.
- **ksmps** : establece el número de muestras que se genera en cada período de control. Este valor debe ser igual a `sr/kr`. El valor por defecto es 10.
- **nchnls** : establece el número de canales de la salida de audio. (1 = mono, 2 = estéreo, 4 = cuadrafónico). El valor por defecto es 1 (mono).

Si hay que dar valor a alguna variable global (que afecta por igual a todas las definiciones de instrumentos), este es momento de hacerlo. Estas variables empiezan todas con la letra "g".

0.3.2 Definición de instrumentos

En Csound, los instrumentos se definen interconectando "módulos" cada uno de los cuales genera o modifica señales. Estas señales se representan mediante variables que conectan unos módulos con otros. Cada uno de los instrumentos recibe un número (o más de uno) y su definición viene parentizada por dos palabras reservadas: "`instr`" seguido del número que se le asigna al instrumento y "`endin`". Entre estas palabras, el instrumento se define, línea a línea, de arriba a abajo. Un fichero de orquesta puede contener prácticamente cualquier número de instrumentos. Un ejemplo de definición muy simple sería (código fuente y esquema gráfico):

```
instr 1
  a1 oscil  10000 , 440 , 1
     out    a1
endin
```



que es un oscilador por tabla de onda que genera una onda (según el ciclo de onda de la tabla número "1" que habrá sido definida en la partitura, no aquí) de 440 Hz y amplitud 10.000. Este instrumento recibe el número 1 y como tal será referenciado a partir de ese momento por las órdenes de la partitura; cada vez que se haga sonar una nota con el instrumento 1 sonará este.

La sintaxis genérica de una sentencia de Csound es:

```
[ etiqueta: ]  salida  opcode  argumentos  [ ; comentarios ]
```

opcode (de *operation code*) es el nombre que reciben los módulos de generación o modificación de señales. Su salida se almacena en una variable que se coloca a la izquierda del *opcode* ("`a1`" en el ejemplo anterior). A la derecha del *opcode* se colocan sus argumentos cuyos rango, orden y cantidad dependen de cada *opcode* concreto. En el ejemplo, en el caso del *opcode* "**`oscil`**", que es un oscilador de los del tipo "tabla de onda" que vimos en el tema de síntesis, espera primero la amplitud de la onda a generar, luego su frecuencia y finalmente el número de la tabla donde encontrar un ciclo de la onda a repetir (que se define en el fichero de partitura).

La orden "**out**" al final del instrumento envía a la salida la onda generada en la variable que le sigue (para estéreo se usa "**outs**" y necesita dos señales como argumentos).

Por otra parte, Csound permite expresiones matemáticas y lógicas, y gestión del flujo de ejecución, como los lenguajes de programación de propósito general. En ambos casos, la sintaxis está inspirada en la de C.

0.3.3 Tipos de variables

Las variables en Csound son de distintos tipos en función de la letra con la que empiezan sus identificadores (*i, k, a* o *g*; por ejemplo `kamp`, `iphs`, `a1`,`gv`). Del tipo depende la frecuencia con la que varían sus valores. Hay cuatro tipos que toman valores con la siguiente frecuencia:

- *tipo g-* : una vez sólo, cuando se configura la orquesta al inicio de la compilación (es, en realidad, una asignación global permanente)
- *tipo i-* : una vez al principio de cada nota (es decir, en su inicialización)
- *tipo k-* : una vez en cada iteración del bucle de control (según la frecuencia de control)
- *tipo a-* : una vez para cada muestra de sonido generada (según la frecuencia de muestreo)

El prefijo *i-* indica un valor escalar válido en el momento de la inicialización de cada nota; los prefijos *k-* o *a-* indican, respectivamente, un valor de control (un escalar) o una muestra de audio (un vector; es decir, un array de valores), modificado y referenciado continuamente a lo largo de la ejecución (es decir, en cada período de muestreo mientras el instrumento esté activo).

Los argumentos cambian de valor a las frecuencias prefijadas por sus tipos. Las salidas de cada *opcode* se calculan de la misma manera y quedan disponibles para su uso como argumentos de entrada para cualquier módulo. La validez de estos argumentos viene definida de la siguiente manera:

1- con el prefijo *g-* son globales a toda la orquesta y válidos desde el principio.
2- con el prefijo *i-* serán válidos en el instante de inicialización de una nota.
3- con el prefijo *k-* serán accesibles tanto como valores de control como de inicialización.
4- con el prefijo *a-* deben ser vectores (arrays de elementos).
5- con el prefijo *x-* pueden ser vectores o escalares (el compilador distinguirá).

Todos los argumentos, a no ser que se especifique de otra manera, pueden ser expresiones cuyos resultados sigan las especificaciones anteriores. La mayoría de los *opcodes* (como por ejemplo **linen** y **oscil**) pueden ser usados en más de una forma, determinada por el prefijo de la variable en la que se almacena la salida. Pueden ser utilizados para generar señales de audio (que tendrán entonces prefijo *a-*) o señales de control (que tendrán prefijo *k-*).

0.4 Estructura de los ficheros de partituras

Un fichero de partitura se divide en secciones separadas por una sentencia de fin de sección, "**s**" (que pone a cero la cuenta de tiempos para la siguiente sección) y finaliza con una sentencia "**e**". Igual que en la orquesta, cada sentencia se especifica en una línea distinta.

En el fichero de partitura básicamente se pueden encontrar dos tipos de eventos: notas y construcción de tablas de ondas. Vamos a ver primero esta clasificación y más adelante describiremos los distintos tipos de sentencias que se pueden encontrar en estos ficheros.

0.4.1 Tablas de ondas

Las tablas de ondas se generan mediante rutinas GEN, que son utilizadas por las sentencias "**f**" para construir y almacenar las tablas. Las tablas están disponibles a lo largo de toda la ejecución de la orquesta y pueden ser invocadas en cualquier instante de la partitura para decirle a un instrumento que las utilice para generar señales. En función de la GEN utilizada se

necesitará distinta cantidad de parámetros que se escriben separados por espacios. Por ejemplo, la sentencia " `f 1 0 16 10 1` " generará una tabla conteniendo 16 muestras de un periodo de una onda sinusoidal. Las rutinas GEN se verán con más detalle en la sección 7.5.

0.4.2 Notas

Las notas son eventos generados mediante instrucciones "**`i`**". Estas notas hacen sonar los instrumentos de la orquesta y les pasan parámetros de control (ver más abajo) como frecuencias, amplitudes, tiempos de ataque, vibrato, etc. Otros parámetros fundamentales de las notas son los que tienen que ver con el tiempo: cuándo empieza a sonar una nota y durante cuánto tiempo.

0.4.3 Campos p (*p-fields*)

Los *campos p* son los parámetros que se escriben a continuación de cada sentencia de la partitura (separados por espacios). A sus valores se accede en la orquesta mediante un nombre formado por una "p" y el número del orden de aparición de izquierda a derecha en la partitura:

```
opcode  p1  p2  p3  p4  p5  ...
```

Cuando un instrumento en el fichero de orquesta reciba una orden desde la partitura recibirá los valores de estos campos mediante estos nombres. Cada *opcode* espera un número determinado de parámetros. Si se escriben más de los necesarios no tienen efecto.

Un punto como valor de un *campo p* indica que para la instrucción de esa línea dicho campo toma el mismo valor del *campo p* correspondiente de la sentencia anterior.

Ejemplo:
```
i  1  0  2  90  200
i  1  2  .   .  300
i  1  4  .   .  400
```

Un signo "<" indica que se interpole entre los valores superior e inferior para dicho campo. El ejemplo anterior se podría haber escrito también de la siguiente manera:

Ejemplo:
```
i  1  0  2  90  200
i  1  2  .   .  <
i  1  4  .   .  400
```

Para el campo p2 cabe la posibilidad de que su valor sea la acumulación de los p2 y p3 de la instrucción anterior mediante un carácter "+". El ejemplo anterior se puede escribir como:

Ejemplo:
```
i  1  0  2  90  200
i  1  +  .   .  <
i  1  +  .   .  400
```

0.5 Sentencias de partitura

0.5.1 Sentencias "`f`"

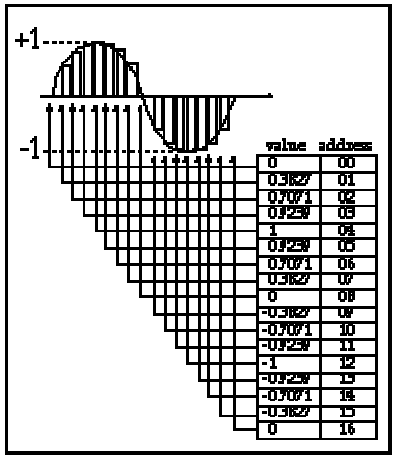
Estas sentencias hacen que una subrutina GEN almacene valores en una tabla, para usarla con los instrumentos de la orquesta. Los *campos p* de esta sentencia son:

- p1 : número que se asigna a la tabla que se crea
- p2 : tiempo a partir del cual la tabla estará disponible (en ese momento se crea)
- p3 : tamaño en número de puntos de la tabla (debe ser potencia de dos, máx: 2^{24})
- p4 : número de rutina GEN de generación.
- p*n*, $n \geq 5$: su uso depende de la rutina GEN utilizada.

Ejemplo:
```
f1  0  16  10  1
```

Con esta sentencia, la tabla 1 se rellena en el instante $t = 0$ con 16 valores, indexados del 0 al 15, usando la rutina GEN10 con un parámetro de valor 1.

La rutina GEN10 genera formas de onda compuestas por la suma de ondas sinusoidales simples cuyas amplitudes relativas figuran de p5 en adelante. Así pues, con la sentencia anterior lo que se hace es generar un período de una onda sinusoidal. La representación gráfica de la tabla y la curva generada se puede observar en la figura de la derecha. Evidentemente, con sólo 16 valores la curva tendrá muy poca resolución, mejor sería utilizar valores como 1024, por ejemplo, pero así sirve para ilustrar el proceso.



0.5.2 Sentencias "`i`"

Estas sentencias llaman a un instrumento de la orquesta para activarlo en un instante específico y durante un tiempo determinado. Los valores de los *campos p* se pasan a ese instrumento justo antes de su inicialización, y permanecen válidos durante toda su activación. Los *campos p* de esta sentencia son:

p1 : número del instrumento invocado en la orquesta (del 1 al 200)
p2 : tiempo en el cual se hace la llamada para empezar a estar activo
p3 : tiempo durante el cual estará activo (se mide en tiempos – notas negras –)
p≥4 : su utilidad depende del uso que se le dé en el instrumento invocado

Ejemplo: `i 1 0 4 ;` Hace sonar el instrumento 1 desde $t = 0$ durante 4 tiempos.

Típicamente, p4 se reserva para la amplitud (en valores o dB) de cada nota y p5 para su frecuencia (en Hz o altura en octava-nota). Esto es sólo un convenio pero es mayoritariamente respetado. Si la amplitud se expresa en dB, habrá que convertirla a valor de amplitud en la orquesta para suministrarla a los *opcodes*. Para ello se usa la función **`ampdb(x)`**. Las equivalencias son:

60 dB ⇒ ampdb(60) = 1000	78 dB ⇒ ampdb(78) = 8000
66 dB ⇒ ampdb(66) = 2000	84 dB ⇒ ampdb(84) = 16000
72 dB ⇒ ampdb(72) = 4000	90 dB ⇒ ampdb(90) = 32000

Los decibelios se refieren en este caso a proporciones respecto al menor valor posible (amplitud 1), por lo que serán menores o iguales a 96 dB (trabajamos a 16-bits).

Para la altura, normalmente se usan frecuencias en Hz. Si se quiere usar la altura de la partitura hay que indicar el número de octava (de 0 a 13) y el de semitono dentro de la misma (de 0 a 11, aunque también admite alturas fraccionarias), separados por un punto. Por ejemplo 8.02 es el RE (8.00 sería el DO, más 2 semitonos, será un RE – ver tabla abajo –) de la octava que normalmente se designa con un tres (hay que restarle 5 para tener la numeración habitual).

.00	.01	.02	.03	.04	.05	.06	.07	.08	.09	.10	.11
DO	DO# REb	RE	RE# MIb	MI	FA	FA# SOLb	SOL	SOL# LAb	LA	LA# SIb	SI

Estos valores de altura habrá que convertirlos a Hz en la orquesta mediante la función **`cpspch(x)`** para obtener hacer sonar la nota a la frecuencia deseada. Algunas de las equivalencias para que sirvan de referencia son las siguientes:

Nota	Hz	8ª + semitono	MIDI	observaciones
DO_{-2}	8,176	3.00	0	la nota más baja posible en MIDI
DO_3	261,626	8.00	60	El DO central del piano
LA_3	440	8.09	69	El LA de referencia
SOL_8	12 543	13.07	127	la nota más alta posible en MIDI

0.5.3 Sentencias "t"

Estas sentencias ajustan el tempo y especifican las aceleraciones y desaceleraciones de la sección en la que aparezcan. La medida del tempo en Csound se hace en tiempos por minuto (*beats per minute, bpm*). Los *campos p* de esta sentencia son:

p1 : pulso en el que empieza a tener efecto la sentencia (normalmente 0)
p2 : tempo inicial en bpm
p3,p5,p7,...: pulsos en los que se da un cambio de tempo (en bpm) en orden creciente
p4,p6,p8,...: tempos a aplicar a partir de los instantes anteriores (respectivamente)

Ejemplo: `t 0 120 4 80 7 220`

Con esta sentencia, el tempo inicial de la sección será de 120 bpm, pero a partir del cuarto pulso será de 80 bpm y desde el séptimo hasta el final de la sección será de 220 bpm.

Si no se especifica ninguna sentencia t dentro de una sección, ésta se desarrollará con un tempo de 60 bpm, por tanto la ausencia de al menos una sentencia **t** en una sección equivale a escribir al principio de ella la sentencia "`t 0 60`" que fija el tempo en un pulso por segundo.

0.6 Rutinas GEN

Las rutinas GEN son métodos de representación de funciones matemáticas que permiten construir tablas de ondas para su uso en los instrumentos de la orquesta. Existen más de 20 funciones GEN distintas. A continuación vamos a ver algunas de las más utilizadas. En todas ellas los tres primeros parámetros tendrán el mismo significado: `num` es el número de la tabla generada, `time` es el tiempo a partir del cual estará disponible y `size` es su tamaño en número de muestras. El cuarto parámetro es el número de GEN. El resto depende de la rutina que se use.

GEN 1

GEN 1 transfiere el contenido de un fichero de sonido a una tabla hasta que se llega al final del fichero o se llena la tabla. Su sintaxis es:

```
f#  time  size  1  filcod  skiptime  format  channel
```

`filcod` es un entero que designa un fichero de nombre "soundin.*filcod*" o una cadena entre dobles comillas con el nombre del fichero. Si no se da toda la ruta, el fichero se busca en el directorio actual. `skiptime` permite indicar los segundos que pueden omitirse del principio del fichero. `format` es para indicar el formato del fichero, el valor 0 indica que se deduzca de la cabecera del fichero. `channel` indica el canal del que leer, un 0 hace leerlos todos.

GEN 3

Dibuja un polinomio. Su sintaxis es:

```
f#  time  size  3  xval1  xval2  c0  c1  c2 . . . cn
```

`xval1` y `xval2` son los valores izquierdo y derecho del intervalo en x en los que se define el polinomio ($xval1 < xval2$). `ci` son los coeficientes del polinomio: $c_0 + c_1x + c_2x^2 + \ldots + c_nx^n$. Pueden ser positivos o negativos.

GEN 5 y GEN 7

Dibujan tablas con segmentos rectilíneos o exponenciales. Su sintaxis es:

```
f#  time  size  5  a  n1  b  n2  c . . .  ;  con 7 para GEN7
```

`a, b, c,` etc. son los valores de las ordenadas de los vértices (deben ser > 0 para GEN5). `n1, n2,` etc. son las longitudes en número de puntos de la tabla. Un 0 sirve para especificar funciones discontinuas. La suma de las n debería ser igual a la longitud de la tabla.

GEN 10

Genera formas de onda mediante una suma ponderada de armónicos sinusoidales.

```
f#  time  size  10  arm1  arm2  arm3  arm4  . . .
```

`armn` son las amplitudes relativas de los distintos armónicos que intervienen en la construcción de la onda. `arm2` será la amplitud del segundo armónico, `arm4` la del cuarto, etc. No hay un límite superior al número de armónicos utilizados. Si alguno intermedio en la lista no se utiliza se pondrá como cero. Lo habitual es que el armónico de la fundamental (el primero) sea la unidad y que los demás sean fracciones de este menores que uno.

0.7 Generadores de señales periódicas

Básicamente los operadores que tienen las orquestas de Csound son bien generadores de señales o bien modificadores de señales. Los generadores utilizan una serie de entradas, normalmente escalares pero no siempre, para producir una señal a tasa de control (tipo *k–*) o a tasa de audio (tipo *a–*). En los modificadores una de las entradas es una señal y el resto parámetros normalmente escalares y dan como resultado una nueva versión de la señal de entrada. Vamos a ver uno de los *opcodes* más utilizados de entre los primeros, como ejemplo:

- **oscil:** sintaxis:
```
kr  oscil  kamp, kcps, ifn [, iphs]
ar  oscil  xamp, xcps, ifn [, iphs]
```
Las unidades **oscil** producen señales periódicas de control (*k–*) o de audio (*a–*) que consisten en el valor devuelto por el muestreo de la tabla *ifn*, realizado a una frecuencia de control (*k–*) o de audio (*a–*), multiplicado por un factor *kamp* (*xamp*). La frecuencia de la señal de salida la fija el argumento *cps* y la fase inicial es *iphs*. Si este parámetro está ausente se considera cero. Mientras que los argumentos de entrada de amplitud y de frecuencia de la unidad **oscil** que trabaja con señales de control (*k–*) son únicamente escalares, los mismos argumentos del **oscil** que trabaja con señales de audio (*a–*) pueden ser tanto escalares como vectores, permitiendo así la modulación de frecuencia o amplitud tanto a frecuencias de control (*k–*) como de audio (*a–*), de ahí el prefijo *x–* de sus dos primeros argumentos.

Ejemplo:
```
k1 oscil 10, 5, 1
a1 oscil 5000, 440 + k1, 1
```
La primera sentencia genera una señal (`k1`) de amplitud 10 a 5 Hz. Los valores se toman desde la tabla número 1 que se inicializará desde la partitura con un sinusoide mediante una orden "`f 1 0 1024 10 1`". Esta señal generada a tasa de control (prefijo *k–*) se utiliza en la segunda sentencia para alterar el valor de la frecuencia que será de 440±10. La salida de este segundo generador será una sinusoide de amplitud 500 y de frecuencia que oscilará en torno a los 440 Hz con una amplitud de 10 Hz y una frecuencia de 5Hz. Es un *vibrato*.

CSound cuenta con decenas de *opcodes* para generación de señales de audio. A continuación se enuncian algunos de los más utilizados o representativos:

- **pluck:** Las unidades **pluck** utilizan un algoritmo de síntesis por modelado virtual (algoritmo de Karplus-Strong) para simular el sonido de una cuerda pulsada por una púa.
- **fof:** Este generador está basado en el programa CHANT[1] del IRCAM (Xavier Rodet y colaboradores). Cada **fof** produce un formante parecido a los de la voz humana y las salidas de estas unidades pueden sumarse para producir una imitación vocal rica. La síntesis con **fof** es una forma especial de síntesis granular.
- **rand:** La salida es una serie controlada de números aleatorios entre ± un rango, generando así un ruido blanco uniforme. Existen otros *opcodes* como **cauchy**, **poisson**, **gauss**, etc. que generan ruido siguiendo las distribuciones estadísticas del mismo nombre.

[1] X. Rodet, Y. Potard, Barrière J-B. "The CHANT project: from the synthesis of the singing voice to synthesis in general". In: The Music Machine. Curtis Roads ed. MIT Press. 1989. pp. 449-466.

- **wgbrass:** La salida de audio que genera es un sonido parecido al de un metal usando un modelo físico desarrollado por Perry Cook[2], pero recodificado para Csound.
- **wgflute:** Similar al anterior pero generando un sonido parecido al de la flauta.
- **wgclar:** Lo mismo pero para clarinete.
- **wgbow:** Lo mismo pero para una cuerda frotada.
- **soundin:** es un generador de audio que deriva su señal de un fichero preexistente en vez de desde una tabla creada con una rutina GEN. **soundin** abre ese fichero cada vez que el instrumento "anfitrión" se inicializa y se cierra de nuevo una vez leído su contenido.

0.8 Generadores de señales aperiódicas

Una de las características de Csound es que uno puede utilizar la salida de casi cualquier generador de señales como entrada de otro *opcode*, enriqueciendo espectacularmente la capacidad de expresividad de las señales generadas. Una de las utilizaciones más comunes de esas posibilidades es la creación de envolventes y modulaciones. Las señales que gobiernan estas características serán de baja frecuencia (modulaciones) o aperiódicas (envolventes). Por tanto, serán típicamente señales del tipo *k*–. Vamos a ver algunas de las más usadas o representativas:

- **line:** sintaxis: `xr line ia, idur, ib`
  Las unidades **line** producen señales aperiódicas que forman una línea recta entre dos puntos dados. Las coordenadas *x* (tiempo) e *y* (magnitud) de estos puntos son: inicial = (0 , *ia*) y final = (*idur* , *ib*). *idur* se expresa en segundos.
- **expon:** sintaxis: `xr expon ia, idur, ib`
  Las unidades **expon** producen señales aperiódicas que forman una curva exponencial entre dos puntos dados. Las coordenadas *x* (tiempo) e *y* (magnitud) de estos puntos son: inicial = (0 , *ia*) y final = (*idur* , *ib*). *idur* se expresa en segundos. *ia* e *ib* no pueden ser 0 ni de signos distintos.
- **linseg:** sintaxis: `xr lineseg ia, idur, ib [, idur2, ic […]]`
  Las unidades **lineseg** producen señales aperiódicas que forman una serie de segmentos lineales entre pares de puntos. Las coordenadas *x* (tiempo) e *y* (magnitud) de estos pares de puntos son para cada segmento: inicial = (0 , *ia*) y final = (*idur* , *ib*). Para el siguiente segmento serán: inicial = (*idur* , *ib*) y final = (*idur*+*idur*2 , *ic*), etc.
- **expseg:** sintaxis: `xr expseg ia, idur, ib [, idur2, ic […]]`
  Las unidades **expseg** producen señales aperiódicas que forman una serie de segmentos exponenciales entre pares de puntos. Las coordenadas *x* (tiempo) e *y* (magnitud) de estos pares de puntos son para cada segmento: inicial = (0 , *ia*) y final = (*idur* , *ib*). Para el siguiente segmento serán: inicial = (*idur* , *ib*) y final = (*idur*+*idur*2 , *ic*), etc.
- **randh** es una variante de **rand** que genera valores aleatorios, pero manteniendo el mismo valor para el período completo de cada ciclo especificado. Otra variante es **randi**, que producirá una interpolación lineal entre cada número aleatorio y el siguiente.
- **linen:** sintaxis: `xr linen xamp, irise, idur, idec`
  Las unidades **linen** producen señales compuestas por unas líneas de ataque y caída de la amplitud de cualquier señal de entrada mediante segmentos lineales. *kamp* es el valor de amplitud de referencia, *irise* es el tiempo en segundos hasta alcanzar dicho valor, *idur* es la duración total en segundos, *idec* es la duración en segundos de la caída (0 significa que no hay curva de caída y un valor *idec* mayor que *idur* causará una caída truncada).

[2] P.R. Cook ."A meta-wind instrument physical model and a meta-controller for real time performance control". In proc. 1992 Int. Comp. Music Conf. San Francisco. Int. Computer Music Assoc. pp.273-276.

- **adsr:** sintaxis: `kr adsr iatt, idec, islev, irel[, idel]`
  Las unidades **adsr** producen señales aperiódicas con el perfil típico de un envolvente ADSR clásico (Ataque, Decaimiento, Sostenimiento, Caída). *iatt* es la duración del ataque, *idec* la duración del decaimiento, *islev* el nivel durante el período de sostenimiento, *irel* la duración de la caída e *idel* un tiempo de retardo opcional del principio de la envolvente.

0.9 Modificadores de señales

Como en cualquier sintetizador, la riqueza de las señales puede incrementarse mucho aplicándole todas las técnicas de procesamiento de señal que vimos en teoría, como mezclar y desafinar osciladores para crear efectos de "chorus", filtros, reverberaciones, ecos, etc. Recuérdese que todas las técnicas de síntesis sustractiva se basan en el uso de filtro sobre señales. Vamos a ver primero hacer un rápido repaso a *opcodes* que implementan filtros.

Csound implementa los filtros pasivos básicos mediante los siguientes opcodes: **tone** — pasa-baja, **atone** — pasa-alta, **reson** — pasa-banda y **areson** — elimina-banda. La sintaxis para los dos primeros es la misma y la vemos con el primero:

- **tone:** sintaxis: `ar tone asig, khp[, istor]`
  La señal de entrada *asig* es modificada de acuerdo con la respuesta de un filtro pasa-baja en el que *khp* (en Hz) determina la frecuencia de corte definida como la que causa que la potencia se reduzca a la mitad (amplitud máxima / $\sqrt{2}$). *istor* tiene que ver con la implementación del algoritmo del filtro y podemos obviarlo en principio. **atone** es lo mismo pero con un pasa-alta. También existen **tonek** y **atonek** que hacen lo mismo pero sobre señales de tipo control (prefijo *k-* en vez de *a-*).

- **reson:** sintaxis: `ar reson asig, kcf, kbw[, iscl, istor]`
  La señal de entrada *asig* es modificada de acuerdo con la respuesta de un filtro pasa-banda en el que *kcf* controla su frecuencia central, o posición (en Hz) del pico de su respuesta, y *kbw* controla su ancho de banda (la diferencia en Hz entre los puntos de potencia media superior e inferior). *istor* tiene que ver con la implementación del algoritmo del filtro y podemos obviarlo en principio. *iscl* es un factor de normalización codificado: un valor 1 implica un factor de respuesta pico de 1, es decir, que todas las frecuencias que no sean *kcf* se atenúan de acuerdo con la curva de respuesta (normalizada); un valor 2 eleva el factor de respuesta hasta que su valor RMS total sea 1; un valor de 0 (es el valor por defecto) implica la no normalización de la señal, dejándola tal cual para un posterior ajuste (ver **balance**). **areson** es lo mismo pero con un elimina-banda. También existen **resonk** y **aresonk** que hacen lo mismo pero sobre señales de tipo control (prefijo *k-* en vez de *a-*).

- **balance:** sintaxis: `ar reson asig, acomp[, ihp, istor]`
  Este *opcode* devuelve una versión de *asig* modificada en amplitud, de manera que su valor de potencia media sea igual al de una señal de comparación *acomp*. Así una señal que ha sufrido pérdida de potencia al pasarla por un banco de filtros puede ser restaurada a su potencia original al compararla, por ejemplo, con su propia fuente sin modificar. *ihp* e *istor* pueden ser obviados inicialmente y que funcionen con sus valores por defecto.

0.10 Efectos

Vamos a describir algunos de los opcodes que permiten "adornar" la señal generada mediante operadores modificadores de señal que permiten diseñar e implementar algunos de los efectos más conocidos como reverbera-ciones, retardos, distorsiones de fase o panorámicos creativos.

- **comb:** sintaxis: `ar comb asig, krvt, ilpt[, istor]`
  Este *opcode* produce como salida una reverberación de una señal de entrada *asig*, durante *krvt* segundos. *ilpt* es la duración del bucle en segundos, que determina la densidad del eco

en la reverberación. La duración del bucle puede ser tan larga como lo permita la memoria disponible. El espacio requerido para un bucle de n segundos es de $4n \times \mathbf{sr}$ bytes.

- **delay:** sintaxis: `ar` **`delay`** `asig, idlt[, istor]`
  Este *opcode* permite que una señal de entrada, *asig*, pueda ser escrita en una línea de retardo y luego leída desde esa misma línea. *idlt* es el tiempo de retardo requerido en segundos. Existe otro opcode relacionado con este que es **vdelay** en el cual el retardo se especifica en número de muestras y puede ser variable, con lo que se logran retardos muy pequeños que pueden combinarse con la señal original para lograr efectos de *chorus*, *phaser* o *flanger*.

- **flanger:** sintaxis: `ar` **`flanger`** `asig, adel, kfeedback[, imaxd ]`
  Este *opcode* aplica a una señal de entrada, *asig*, un efecto de *flanger* donde *adel* es el retardo en segundos, *kfeedback* la cantidad de retroalimentación (en tareas normales no debería exceder un valor de 1) e *imaxd* el retardo máximo en segundos. El retardo debe ser variado a una frecuencia de audio conectando la salida de un oscilador a *adel*. También la retroalimentación puede ser variada, a frecuencia de control.

- **pan:** sintaxis: `a1, a2, a3, a4` **`pan`** `asig, kx, ky, ifn[, imode[, ioffset]]`
  Este *opcode* distribuye una señal de audio de entrada, *asig*, entre cuatro canales, con control sobre la localización exacta. *ifn* es un número de tabla de función que contendrá un patrón que describe el aumento de amplitud en el canal de un altavoz al acercársele el sonido de un altavoz adyacente. *imode* es el modo de los valores de posición *kx* y *ky*. 0 significa modo de indexación normal, 1 indica que las entradas se normalizan en el intervalo de 0 a 1. El valor por defecto es 0. *ioffset* es un desplazamiento del origen para *kx* y *ky*. Un valor 0 deduce que el origen es el canal 3 (altavoz trasero izquierdo); un valor 1 pide un desplazamiento del eje hacia centro cuadrafónico. El valor por defecto es 0.

  pan toma la señal de entrada *asig* y la distribuye por los cuatro canales de salida (esencialmente altavoces cuadrafónicos) de acuerdo con el valor de los controles *kx* y *ky*. Para entradas normalizadas (modo 1 de *imode*) y sin offset, las cuatro posiciones de salida van en el siguiente orden: altavoz izquierdo delantero en (0,1), derecho delantero en (1,1), izquierdo trasero en el origen (0,0) y derecho trasero en (1,0). En la notación (*kx*, *ky)*, las coordenadas *kx* y *ky*, cada una de ellas en el rango de 0 a 1, controlan de esta manera la localización del sonido en cualquier dirección "izquierda-derecha, adelante-atrás".

0.11 Otros *opcodes*

Finalmente, expondremos un par de *opcodes* de uso general pero muy útiles para saber cómo se están generando las señales. Son:

- **display:** sintaxis: **`display`** `xsig, idur [, inprds[, iwtflg]]`
  Este *opcode* mostrará valores de inicialización de la orquesta, o producirá una representación gráfica de señales de control o de audio. *idur* es la duración de la respresentación en segundos. La gráfica será de amplitud frente a tiempo.

- **dispfft:** sintaxis: **`dispfft`** `xsig, idur, iwsiz[, iwtyp[, idbouti[, iwtflg]]`
  Este *opcode* representa gráficamente la transformada de Fourier de una señal de audio o de control (*asig* o *ksig*) cada *idur* segundos usando el método de la transformada rápida de Fourier. *iwsiz* es el tamaño de la ventana de entrada en muestras. Una ventana de *iwsiz* puntos producirá una transformada de Fourier de *iwsiz*/2 puntos, distribuida linealmente a lo largo del espectro desde 0 a sr/2. *iwsiz* debe ser una potencia de 2, con un valor mínimo de 16 y uno máximo de 4096. Las ventanas pueden solaparse.