Appendices

V. Lazzarini, *Computer Music Instruments*,
https://doi.org/10.1007/978-3-319-63504-0

Appendix A
Signal Processing Mathematics

This appendix provides some support for the decoding of the mathematical expressions given in this book. We will look at the specific operations and concepts that are fundamental to the principles of synthesis and digital signal processing that we have developed and implemented in Part II.

A.1 Fundamental Concepts and Operations

In addition to the rules for the usual arithmetic operations $(+, -, \times, /)$, the following apply for *exponentiation*:

$$\begin{aligned} &a^0 = 1 \quad (a \neq 0) \\ &a^1 = a \\ &a^x a^y = a^{x+y} \\ &\frac{a^x}{a^y} = a^{x-y} \\ &(a^x)^y = a^{xy} \\ &a^x b^x = (ab)^x \\ &a^{-x} = \frac{1}{a^x} \end{aligned} \tag{A.1}$$

From these definitions, we can work out the exponentiation of a number. For instance

$$a^5 = a^{1+1+1+1+1} = a \times a \times a \times a \times a \tag{A.2}$$

The logarithm of a number is defined by

$$\log_b a = x \quad \Rightarrow \quad a = b^x \quad (b \neq 0 \text{ and } b \neq 1) \tag{A.3}$$

where b is called the *base* of the logarithm, which is most commonly set to one of 2, 10, or $e = 2.718281828459...$ (the *natural* base of logarithms). In this book, log (with no subscript) implies base e.

From this and the definitions in Eq. A.1, we have the following relations:

$$\begin{aligned} \log_c ab &= \log_c a + \log_c b \\ \log_c \frac{a}{b} &= \log_c a - \log_c b \\ \log_c a^b &= b \log_c a \end{aligned} \tag{A.4}$$

From these we can see that the ratio of two values becomes a difference of their logarithms. While a linear distance between two quantities a and b is defined as the difference $b - a$, their logarithmic distance is $\log b - \log a$. The former applies to our understanding of physical space, whereas the latter relates closely to the way we perceive amplitudes and frequencies.

A.1.1 Vector and matrix operations

The transposition of a vector is the exchange of columns for rows (and vice-versa):

$$\begin{bmatrix} a & b & c & d \end{bmatrix}^T = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{A.5}$$

The product of two square matrices A and B is defined as

$$\begin{aligned} AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \\ \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} \end{aligned} \tag{A.6}$$

Note that the commutative property does not apply here, so, in general, $AB \neq BA$ (although it is obviously possible to engineer cases where this is not true).

Likewise, we can effect the product of a matrix A of dimensions $N \times N$ by a column vector B of size N, producing a column vector also of size N:

$$AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae + bf \\ ce + df \end{bmatrix} \tag{A.7}$$

A.1.2 Sum and product

Shorthand forms for sum and product are defined as follows

$$\sum_{n=s}^{N} a_n = a_s + a_{s+1} + a_{s+2} + \ldots + a_{N-1} + a_N \tag{A.8}$$

and

$$\prod_{n=s}^{N} a_n = a_s \times a_{s+1} \times a_{s+2} \times \ldots \times a_{N-1} \times a_N \tag{A.9}$$

In these expressions, n is called the *index* (of the summation or product), and its range is normally explicitly defined (unless it can be deduced from elsewhere). Summation formulae can have *closed* forms, as discussed extensively in Chap. 4, the simplest of which is just

$$\sum_{n=1}^{N} n = \frac{N(N+1)}{2} \tag{A.10}$$

called an *arithmetic series*.

Some fundamental relations should also be noted:

$$\begin{aligned}
\sum_{n=s}^{N} f(n) \pm \sum_{n=s}^{N} g(n) &= \sum_{n=s}^{N} [f(n) \pm g(n)] \\
\sum_{n=s}^{N} k f(n) &= k \sum_{n=s}^{N} f(n) \\
\sum_{n=s+m}^{N+m} f(n) &= \sum_{n=s}^{N} f(n+m) \\
\sum_{n=0}^{2N+1} f(n) &= \sum_{n=0}^{N} [f(2n) + f(2n+1)]
\end{aligned} \tag{A.11}$$

A.1.3 Polynomials and functions

Polynomials are expressions involving a variable raised to various powers:

$$\sum_{n=0}^{N} a_n x^n = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_N x^N \tag{A.12}$$

The multipliers a_n associated with the various terms are called *coefficients*, and the *order* of a polynomial is given by N, its highest exponent. A function is an expression that maps one or more parameters into a value. A *polynomial* function

$f(x)$ is one that involves an expression such as the one in eq A.12, where an input x is mapped to the result of the polynomial defined by $f(x)$. Such functions can be evaluated for various ranges of x.

A.2 Trigonometry

An angle can be the defined as the space between two intersecting lines. It can be measured in degrees, ranging from 0 to 360°, with 90° marking the angle between two perpendicular lines, called a *right* angle. A triangle with one right angle is called a *right* triangle. Angles can also be measured in relation to the diameter d and circumference c of a circle whose centre is the intersecting point of the two lines. This is the preferred way, and can be formed by this relation:

$$c = \pi d \tag{A.13}$$

So, for a circle or radius 1, called the *unit circle*, angles can be measured linearly on the circumference from 0 to 2π, with $\frac{\pi}{2}$ being equivalent to a right angle. This can be extended to all angle measurements by defining a scale on this basis, whose unit is called a *radian* (Fig. A.1). All angles in the signal-processing formulae in this book are thus defined.

The fundamental trigonometric functions are first defined in relation to a right triangle with sides A, O, and H, where the right angle is placed between the A and O sides (Fig. A.2). Firstly, from Pythagoras, we have

$$H^2 = A^2 + O^2 \tag{A.14}$$

In this arrangement, we have, for an angle θ set between the A and H sides,

$$\begin{aligned} \sin(\theta) &= \frac{O}{H} \\ \cos(\theta) &= \frac{A}{H} \end{aligned} \tag{A.15}$$

To simplify the relationships, we set $H = 1$, making $\sin(\theta) = O$ and $\cos(\theta) = A$. In this case, we can see how the functions $\sin(x)$ and $\cos(x)$ taken together can track the position of a point on the unit circle. For instance, in Fig. A.2 the (x,y) coordinates of point b are $(H\cos(\theta), H\sin(\theta))$, with respect to the centre of the circle. Thus we have $\sin(0) = 0$, $\sin(\pi/2) = 1$, $\sin(\pi) = 0$, and $\sin(3\pi/2) = -1$; similarly, $\cos(0) = 1$, $\cos(\pi/2) = 0$, $\cos(\pi) = -1$, and $\cos(3\pi/2) = 0$. Note that

$$\begin{aligned} \cos(\theta) &= \sin(\theta + \pi/2) \\ \sin(\theta) &= \cos(\theta - \pi/2) \end{aligned} \tag{A.16}$$

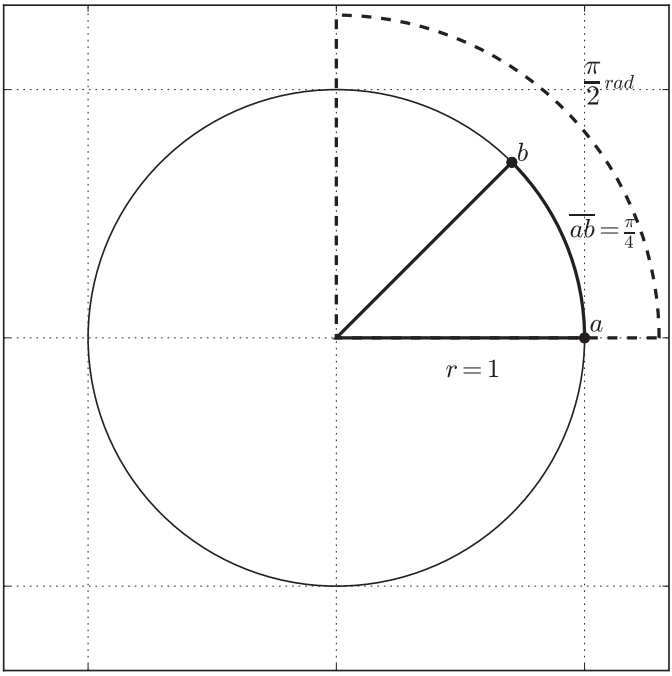

Fig. A.1 Angle definitions: for a circle of radius 1, the total circumference measures 2π, and the segment $\overline{ab}$ in the picture measures $\pi/4$. This is equivalent to an angle of $\pi/4$ radians. A right angle (dashes) measures $2\pi rad$.

and

$$\begin{aligned} \sin(\theta) &= -\sin(-\theta) \\ \cos(\theta) &= \cos(-\theta) \end{aligned} \tag{A.17}$$

In addition to these functions, we can also define the tangent as

$$\tan(\theta) = \frac{O}{A} = \frac{\sin(\theta)}{\cos(\theta)} \tag{A.18}$$

which is equivalent to the length of the line $\overline{de}$ in Fig. A.2. To recover the angle from a given ratio of triangle sides, we have inverse functions:

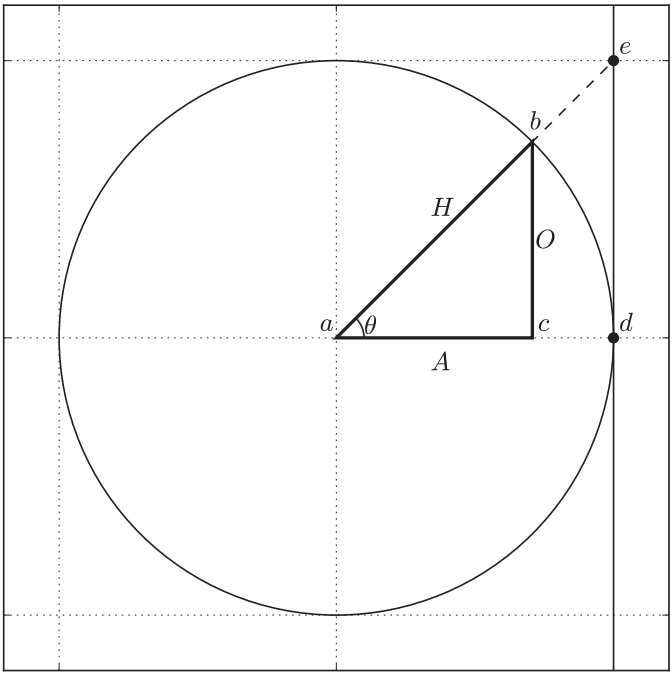

Fig. A.2 Trigonometric circle: a circle of radius H inside which we construct a right triangle *abc*, with sides A, O, and H. An angle θ is defined between H and A. A line that is tangent to the circle at point d is drawn, with which we can define another right triangle *aed*.

$$\begin{aligned} \arcsin\left(\frac{O}{H}\right) &= \cos^{-1}\left(\frac{O}{H}\right) = \theta \\ \arccos\left(\frac{A}{H}\right) &= \sin^{-1}\left(\frac{A}{H}\right) = \theta \\ \arctan\left(\frac{O}{A}\right) &= \tan^{-1}\left(\frac{O}{A}\right) = \theta \end{aligned} \tag{A.19}$$

and $\cos(\arccos(\theta)) = \theta$ etc.

A.2.1 Identities

There are a number of identities between trigonometric functions that can be used to manipulate expressions using them:

$$\begin{aligned}
\cos(\theta)^2 + \sin(\theta)^2 &= 1 \\
\cos(\theta)^2 &= \frac{1+\cos(2\theta)}{2} \\
\sin(\theta)^2 &= \frac{1-\cos(2\theta)}{2} \\
\cos(2\theta) &= \cos(\theta)^2 - \sin(\theta)^2 \\
&= 1 - 2\sin(\theta)^2 = 2\cos(\theta)^2 - 1 \\
\sin(2\theta) &= 2\cos(\theta)\sin(\theta)
\end{aligned} \tag{A.20}$$

$$\begin{aligned}
\cos(\theta \pm \phi) &= \cos(\theta)\cos(\phi) \mp \sin(\theta)\sin(\phi) \\
\sin(\theta \pm \phi) &= \sin(\theta)\cos(\phi) \pm \cos(\theta)\sin(\phi) \\
\cos(\theta)\cos(\phi) &= \frac{\cos(\theta-\phi)+\cos(\theta+\phi)}{2} \\
\sin(\theta)\sin(\phi) &= \frac{\cos(\theta-\phi)-\cos(\theta+\phi)}{2} \\
\cos(\theta)\sin(\phi) &= \frac{\sin(\theta+\phi)-\sin(\theta-\phi)}{2} \\
\sin(\theta)\cos(\phi) &= \frac{\sin(\theta+\phi)+\sin(\theta-\phi)}{2}
\end{aligned} \tag{A.21}$$

$$\begin{aligned}
\cos(\theta)+\cos(\phi) &= 2\cos\left(\frac{\theta+\phi}{2}\right)\cos\left(\frac{\theta-\phi}{2}\right) \\
\cos(\theta)-\cos(\phi) &= -2\sin\left(\frac{\theta+\phi}{2}\right)\sin\left(\frac{\theta-\phi}{2}\right) \\
\sin(\theta)\pm\sin(\phi) &= -2\sin\left(\frac{\theta\pm\phi}{2}\right)\cos\left(\frac{\theta\mp\phi}{2}\right)
\end{aligned} \tag{A.22}$$

A.3 Numeric Systems

In this book, we use three numeric systems that have some important characteristics. The first one of these is the *integers*, which represent whole-number quantities, both positive and negative. These are used mainly when counting, keeping track of summation indices, delay time in samples, etc.

The next system that we employ is the *real* numbers. These represent the one-dimensional continuum, for instance time passing from one instant to another, infinitesimal differences in values, etc. We can think of these numbers as points on a line that extends from $-\infty$ to ∞, where a number a is to the left of b if $a < b$, and to the right of it if $a > b$. Between a and b lie an infinite number of points. The reals also include some special numbers such as π and e, as well as quantities such as $\sqrt{2}$.

This system encompasses all integers and all fractions of integers (which are also called *rationals*). The synthesis and processing theory developed in Part II depends heavily on it.

The third system that we use extends the reals to two dimensions. These are called *complex* numbers. We will outline its key characteristics here. A complex number is defined as a conjunction of two real numbers (we can think of it as a *tuple*) a and b, where one of them carries a special mark, j, to distinguish it from the other:

$$z = a + jb \tag{A.23}$$

This special symbol has an algebraic interpretation, $j^2 = -1$, which we will use when necessary. For the moment, we can think of it as something that marks b as different from a, placing it in a different dimension. So if a lies on one line, b lies on a different one that is perpendicular to it (Fig.A.3)

This is the geometric interpretation of j, placing the quantity b in that direction. The number z then lies at the intersection of a line starting from a at a right angle to the real line and a line from b parallel to the real line. So it is a two-dimensional quantity, defined on a plane, and requiring a and b to locate it. The line on which b lies is called the *imaginary* line and, taken together, this and the real line form a set of axes that divides the plane (the *complex plane*, shown in Fig. A.3) into four quadrants ($+, +j$; $-, +j$; $-,-j$; and $+,-j$).

Complex arithmetic follows the usual rules, so we have the following relationships for two complex numbers $z = a + jb$ and $w = c + jd$:

$$z + w = (a + jb) + (c + jd) = a + c + j(b + d) \tag{A.24}$$

$$z - w = (a + jb) - (c + jd) = a - c + j(b - d) \tag{A.25}$$

$$z \times w = (a + jb)(c + jd) = ac + j^2bd + j(bc + ad) \tag{A.26}$$

Since we have defined earlier that $j^2 = -1$, then

$$z \times w = ac - bd + j(bc + ad) \tag{A.27}$$

$$\frac{z}{w} = \frac{a + jb}{c + jd} = \frac{(a + jb)(c - jd)}{(c + jd)(c - jd)} = \frac{ac + bd + j(bc - ad)}{c^2 + d^2} \tag{A.28}$$

An alternative representation for a complex number is through its *polar* form. To get this, we use a trigonometric interpretation of the complex number, where

$$\Re\{z\} = A\cos(\phi) \tag{A.29}$$

and

$$\Im\{z\} = A\sin(\phi) \tag{A.30}$$

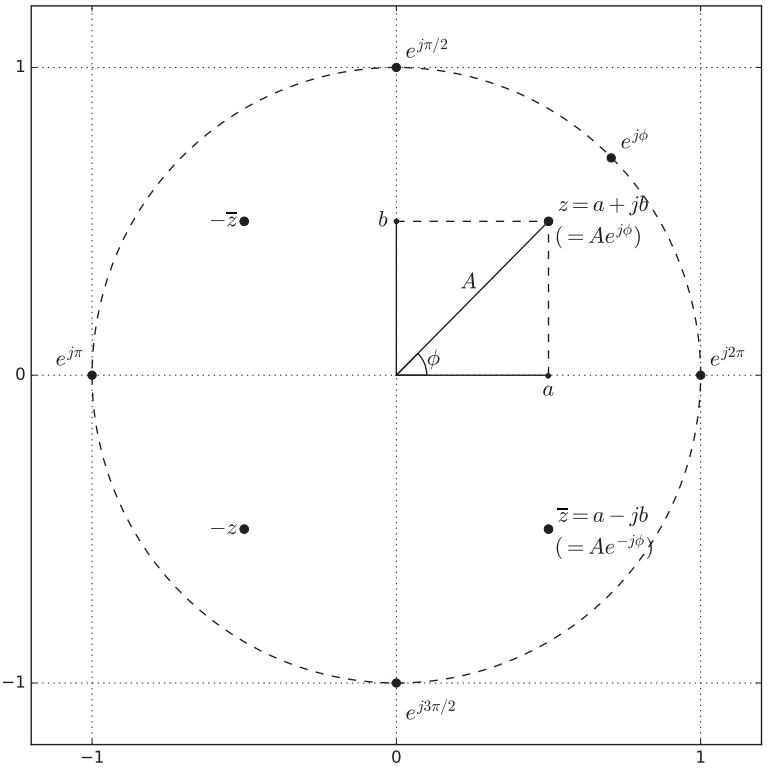

Fig. A.3 The complex plane: a complex number $z = a + jb$ may lie anywhere in this plane. In this particular case, z is 0.5 + 0.5j, and its coordinates are $(a,b) = (\Re\{z\}, \Im\{z\}) = (0.5, 0.5)$. Its complex exponential form is $Ae^{j\phi} = \sqrt{2}e^{j\pi/4}$. Its complex conjugate $\overline{z}$ is also shown, along with $-z$ and $-\overline{z}$. The figure also demonstrates that a number e^{jx} will always lie on the unit circle (dashes). The numbers $e^{j\pi/4}$, $e^{j\pi/2}$, $e^{j\pi}$, $e^{j3\pi/2}$ and $e^{j2\pi}$ are shown to illustrate some examples of this.

where $\Re\{z\} = a$ and $\Im\{z\} = b$ for $z = a + jb$ (compare figs. A.2 and A.3).

This reveals the number's magnitude (absolute value) $|z| = A$ and phase (angle) $\arg(z) = \phi$, which can be extracted by

$$A = |z| = \sqrt{a^2 + b^2} \tag{A.31}$$

$$\phi = \arg(z) = \arctan\left(\frac{b}{a}\right) \tag{A.32}$$

In contrast with this, the $a + jb$ representation is called the *rectangular* form. The algebraic expression that connects the two forms is the complex exponential:

$$z = Ae^{j\phi} = A\cos(\phi) + jA\sin(\phi) \tag{A.33}$$

which follows from Euler's formula,

$$e^{jx} = \cos(x) + j\sin(x) \tag{A.34}$$

This number (e^{jx}) will always lie on the unit circle in the complex plane (Fig. A.3).

The polar form is very useful in signal processing, as it splits complex quantities neatly into amplitudes and phases, which are, in many cases, more meaningful. The complex exponential function $x(t) = e^{j2\pi ft}$ is the fundamental signal for analysis and synthesis, a complex sinusoid that packs a cosine and a sine wave together into a single unit.

The complex exponential form also simplifies some of the arithmetic when it comes to multiplication and division:

$$z \times w = A_z e^{j\phi_z} A_w e^{j\phi_w} = A_z A_w e^{j(\phi_z + \phi_w)} \tag{A.35}$$

$$\frac{z}{w} = \frac{A_z e^{j\phi_z}}{A_w e^{j\phi_w}} = \frac{A_z}{A_w} e^{j(\phi_z - \phi_w)} \tag{A.36}$$

Thus, to multiply we take the product of the magnitudes and the sum of the phases, and to divide we take the quotient of the magnitudes and the difference of the phases. This allows us to reinterpret j as a *rotation* operator that shifts the phase of a complex number by $\pi/2$ radians (90 degrees anticlockwise). Consider the number $z = e^{j\phi} = \cos(\phi) + j\sin(\phi)$, then

$$jz = e^{j(\phi + \pi/2)} = \cos\left(\phi + \frac{\pi}{2}\right) + j\sin\left(\phi + \frac{\pi}{2}\right) = -\sin(\phi) + j\cos(\phi) \tag{A.37}$$

This also shows that $j^2 = -1$, which is equivalent to a shift of π radians.

Finally, we should define the complex conjugate of a number as

$$z = a + jb \quad \Leftrightarrow \quad \overline{z} = a - jb \tag{A.38}$$

that is, the imaginary part changes sign when two numbers are complex conjugates of each other (as shown in Fig. A.3). In polar (complex exponential) form, the magnitudes remain the same, but the phases change sign:

$$z = Ae^{j\phi} \quad \Leftrightarrow \quad \overline{z} = Ae^{-j\phi} \tag{A.39}$$

Some identities apply:

$$\begin{aligned} z \times \overline{z} &= a^2 + b^2 = |z|^2 = A^2 \\ z \times \overline{z}^{-1} &= e^{2j\phi} = \frac{z}{|z|} e^{j\phi} \\ z + \overline{z} &= 2a = 2\Re\{z\} \\ z - \overline{z} &= 2jb = 2j\Im\{z\} \end{aligned} \tag{A.40}$$

A.4 Complex Polynomials

A complex polynomial function is defined as before (Sect. A.1.3), but using a complex variable instead of a real-valued one:

$$f(z) = \sum_{n=0}^{N} a_n z^n = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \ldots + a_N z^N \tag{A.41}$$

One such function is called the *z-transform*, which we can use to obtain the *transfer function* of a digital filter whose impulse response $h(t)$ is given:

$$H(z) = \sum_{t=-\infty}^{\infty} h(t) z^{-t} \tag{A.42}$$

where the coefficients of the polynomial are the samples of the impulse response $h(t)$.

For a finite $h(t)$ of size N, Eq. A.42 can be use to give us the filter *frequency response* sampled at points $z = e^{j2\pi k/N}$ (representing sinusoid frequencies $f(k) = 2\pi k/N$ Hz):

$$H(e^{j2\pi k/N}) = \sum_{t=0}^{N-1} h(t) e^{-2\pi kt/N} = DFT(h(t)) \tag{A.43}$$

and so the amplitude and phase responses of this filter at a point k are $|H(e^{j\omega})|$ and $\arg(H(e^{j\omega}))$, respectively, with $\omega = 2\pi k/N$.

In addition, the z-transform can also be used more generally to derive transfer functions and frequency responses of arbitrary linear time-invariant digital filters. For this, the following relationship applies:

$$Y(z) = H(z)X(z) \tag{A.44}$$

where X(z) and Y(z) are the z-transforms of the input and output, respectively, of a filter. $H(z)$, in this case, plays the part of the transfer function. When this is sampled at the DFT frequencies, as in Eq. A.43, we can use it to multiply another spectrum (X(z) sampled at the same frequencies) to accomplish the filtering process (as discussed in Sect. 7.1.4).

The z-transform can be applied directly to the study of filters. For instance the simple filter $y(t) = 0.5 * x(t) + 0.5 * x(t-1)$ ($h(t) = [0.5, 0.5]$), has the following z-transform:

$$H(z) = \sum_{t=0}^{1} h(t) z^{-t} = 0.5 + 0.5 z^{-1} \tag{A.45}$$

From this we can infer a meaning for z^t as a time shift operator, applying a shift of t samples to a signal. So z^{-1} implies a 1-sample delay, z^{-2} implies a 2-sample delay etc. Associated with each delay is a coefficient, which is the value of the impulse

response at time t. To get the amplitude and phase responses of this filter, we can evaluate $|H(z)|$ and $\arg(H(z))$ directly for any frequency ω by setting $z = e^{j\omega}$

$$\begin{aligned} |H(e^{j\omega})| &= \sqrt{(0.5+0.5\cos(\omega))^2 + (0.5\sin(\omega))^2} = \cos(\omega/2) \\ \arg(H(e^{j\omega})) &= \arctan\left(\frac{\sin(\omega)}{0.5+0.5\cos(\omega)}\right) = \frac{\omega}{2} \end{aligned} \tag{A.46}$$

which makes this a low pass filter (set $\omega = 2\pi f/f_s$ for any f in Hz to check), with a linear phase shift of $\omega/2$ radians across the frequency spectrum (equivalent to a 1/2-sample delay for the whole signal).

The case of Eq. A.45 is that of an finite impulse response (FIR) filter. For an infinite impulse response (IIR) $y(t) = a_0x(t) + a_1x(t-1) + b_1y(t-1)$, we can first rearrange the equation as

$$y(t) - b_1y(t-1) = a_0x(t) + a_1x(t-1) \tag{A.47}$$

where each side now has its own impulse response $g(t) = [1, -b_1]$ and $f(t) = [a_0, b_0]$. If the z-transforms of $g(t)$ and $f(t)$ are $G(z)$ and $F(z)$, and the filter transfer function is $H(z) = Y(z)/X(z)$ (Eq. A.44), we have

$$Y(z)G(z) = X(z)F(z) \tag{A.48}$$

Therefore, the z-transform $H(z)$ of this filter is the ratio of the z-transforms $G(z)$ and $F(z)$ of $g(t)$ and $f(t)$, respectively:

$$H(z) = \frac{F(z)}{G(z)} = \frac{a_0 + a_1z^{-1}}{1 - b_1z^{-1}} \tag{A.49}$$

Thus, for an arbitrary digital filter

$$y(t) = \sum_{n=0}^{N} a_nx(t-n) + \sum_{m=1}^{M} b_my(t-m) \tag{A.50}$$

the z-transform gives us the transfer function:

$$H(z) = \frac{\sum_{n=0}^{N} a_nz^{-n}}{1 - \sum_{m=1}^{M} b_mz^{-m}} = \frac{a_0 + a_1z^{-1} + a_2z^{-2} + \ldots + a_Nz{-N}}{1 - b_1z^{-1} - b_2z^{-2} - \ldots - b_Mz^{-M}} \tag{A.51}$$

Two final remarks: (i) the infinite part of the impulse response is represented by the denominator, and the finite part by the numerator; and (ii) the filter will be stable if the denominator of $|H(z)|$ is not zero, and therefore an FIR filter is always stable.

A.5 Differentiation and Integration

In the real continuum, as the infinitesimal difference between two numbers x and $x+dx$[1] approaches 0, the derivative of a function $f(x)$ can be defined as

$$\frac{df(x)}{dx} \tag{A.52}$$

This relation is used to get the rate of change of a function and the process of obtaining its value is called *differentiation*. For instance, the derivative of the phase of a sinusoidal signal (as a function of time $\phi(t)$) gives us the instantaneous frequency $f(t)$:

$$f(t) = \frac{d\phi(t)}{dt} \tag{A.53}$$

The basic rules of differentiation are:

1. For a constant C

$$\frac{dC}{dx} = 0 \tag{A.54}$$

2. A polynomial term x^n has a derivative

$$\frac{d(x^n)}{dx} = nx^{n-1} \tag{A.55}$$

3. The derivative of the sum or difference of two terms f and g is

$$\frac{d(f \pm g)}{dx} = \frac{df}{dx} \pm \frac{dg}{dx} \tag{A.56}$$

4. The derivative of the product of f and g is

$$\frac{d(f \times g)}{dx} = g\frac{df}{dx} + f\frac{dg}{dx} \tag{A.57}$$

5. The chain rule can be used to work out the derivative of a function of a function (e.g. $df(g(x))/dx$). In this case, we set $y = f(z)$ and $z = g(x)$ in

$$\frac{dy}{dx} = \frac{dy}{dz} \times \frac{dz}{dx}. \tag{A.58}$$

For trigonometric functions, we have well-known derivatives, such as

$$\begin{aligned} \frac{d}{dx}\cos(x) &= -\sin(x) \\ \frac{d}{dx}\sin(x) &= \cos(x) \end{aligned} \tag{A.59}$$

[1] In this book, the symbol d is sometimes replaced by ∂ (normally used in partial differentiation) to avoid ambiguity if a variable d is already being used.

With these principles, we can work out, for example, the instantaneous frequency of an oscillator whose phase is modulated by a cosine wave, $\sin(\omega t + \cos(\omega t))$:

$$\begin{aligned} f(t) &= \frac{d\phi(t)}{dt} = \frac{d(\omega t + \cos(\omega t))}{dt} \\ &= \frac{d\omega t}{dt} + \frac{\cos(\omega t)}{dt} \\ &= \omega + \frac{d\omega t}{dt}\left[\frac{\cos(\omega t)}{d\omega t}\right] \\ &= \omega - \omega \sin(\omega t) \end{aligned} \tag{A.60}$$

The reverse operation to differentiation, in a general sense, is integration. This can take the form of a *definite integral*, over a set interval, or an *indefinite integral*, also known as an *antiderivative*, which is a function $F(x)$ with a derivative $f(x) = dF(x)/dx$. In the former case, the operation gives us a value for the integral, which is a sum over a continuous interval; in the latter case, we have an expression.

In some cases, we can use the rules for differentiation in reverse, as well as some other integration techniques, to work out the antiderivative, which will in turn allow us to calculate a definite integral. However, it is not always possible to guarantee that we will be able to find an antiderivative, and the process is more involved than in the case of differentiation.

The antiderivative of a polynomial term can be worked out using the relation

$$\int x^n dx = \frac{x^{n+1}}{(n+1)} + C \quad (n \neq -1) \tag{A.61}$$

where C is an arbitrary value called the *constant of integration*. Likewise, the antiderivatives of trigonometric functions can be taken from their derivatives:

$$\begin{aligned} \int \cos(ax)dx &= \frac{1}{a}\sin(ax) + C \\ \int \sin(ax)dx &= -\frac{1}{a}\cos(ax) + C \end{aligned} \tag{A.62}$$

For example, to find the phase of an oscillator whose frequency is $\omega + \omega\cos(\omega t)$ (FM), we have

$$\begin{aligned} \phi(t) &= \int \omega + \omega\cos(\omega t)dt \\ &= \int \omega dt + \int \omega\cos(\omega t)dt \\ &= \omega t + \sin(\omega t) + C \end{aligned} \tag{A.63}$$

So when we modulate the frequency with a cos(), the equivalent PM expression employs a sin(). The meaning of the C constant is that there may be an initial phase offset that we are not able to define when calculating the antiderivative.

Finally, definite integrals are used when we need to realise the sum to obtain a numerical result. This is the case, for instance, for the integral in the Fourier transform formula. Thus is a definite integral, even though the interval goes from $-\infty$ to ∞, because we are using it to calculate a complex spectral coefficient for a given frequency:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \tag{A.64}$$

The dt and the $\int$ symbol go together to represent the fact that this sum is over a continuum (rather than over a discrete series of numbers as in the case of the symbol $\sum$). In practical applications, we can do the integration over a finite interval, such as one cycle of a waveform. In that case we can use the formula:

$$\int_a^b f(x)dx = F(b) - F(a) \tag{A.65}$$

where $F(x)$ is the antiderivative of $f(x)$. For this to work, we need to find this function, which is more difficult in some cases than in others. However, there are several techniques that can help us get there.

For example, to get the Fourier series coefficients for a square wave,

$$x(t) = \begin{cases} 1 & 0 \le t < \pi \\ -1 & \pi \le t < 2\pi \end{cases} \tag{A.66}$$

we can use

$$X(k) = a_k + jb_k = \int_0^{2\pi} x(t) e^{-jkt} dt \tag{A.67}$$

The a_k and b_k coefficients are

$$\begin{aligned} a_k &= \int_0^{2\pi} x(t)\cos(kt)dt = \frac{1}{2}\left[\int_0^{\pi} \cos(kt)dt + \int_{\pi}^{2\pi} -\cos(kt)dt\right] \\ &= \frac{1}{2}\left[\frac{1}{k}(\sin(0) - \sin(\pi k)) - \frac{1}{k}(\sin(\pi k) - \sin(2\pi k))\right] = 0 \end{aligned} \tag{A.68}$$

$$\begin{aligned} b_k &= -\int_0^{2\pi} x(t)\sin(kt)dt = -\frac{1}{2}\left[\int_0^{\pi} \sin(kt)dt + \int_{\pi}^{2\pi} -\sin(kt)dt\right] \\ &= -\frac{1}{2}\left[\frac{1}{k}(1 - \cos(\pi k)) - \frac{1}{k}(1(\cos(\pi k) - 1)\right] \\ &= \frac{\cos(\pi k) - 1}{k} \end{aligned} \tag{A.69}$$

Since $\cos(\pi k)$ is -1 for k odd and 1 for k even, the square wave Fourier series (Eq. 7.6) becomes

$$x(t) = 4 \sum_{k=0}^{\infty} \frac{1}{(2k+1)} \sin(2\pi[2k+1]t) \tag{A.70}$$

We can use similar approaches to derive the Fourier series for various waveforms, such as sawtooth (see Eq. 7.10), triangle and pulse waveforms. With some luck, we can find Fourier series for others too.

Appendix B
Application Code

This appendix presents complete applications and other programming examples for the topics studied in this book. Python applications are written for version 3 and depend on the installation of pylab (SciPy, NumPy and matploltlib), as well as Csound and ctcsound.

B.1 Shapes

The full code for the Shapes application discussed in chapter 2 is presented in the listing below:

Listing B.1 Shapes application code.

```
import tkinter
import ctcsound

code = '''
sr=44100
ksmps=64
nchnls=1
0dbfs=1


gar init 0
gifbam2 faustcompile {{
beta = hslider("beta", 0, 0, 2, 0.001);
fbam2(b) = *~(((\(x).(x + x'))*b)+1);
process = fbam2(beta);
}}, "-vec -lv 1"

gar init 0
```

```
instr 1
ival = p4
kp chnget "pitch"
kp tonek kp, 10, 1
kv chnget "volume"
kv tonek kv, 10
ain oscili 1, p5*kp, -1, 0.25
ib, asig faustaudio gifbam2,ain
kb1 expsegr ival,0.01,ival,30,0.001,0.2,0.001
faustctl ib,"beta",kb1
asig balance asig, ain
kenv expsegr 1,20,0.001,0.2,0.001
aenv2 linsegr 0,0.015,0,0.001,p4,0.2,p4
aout = asig*aenv2*kenv*0.4*kv
 out  aout
gar += aout
endin

instr 100

a1,a2 reverbsc gar,gar,0.7,2000
amix = (a1+a2)/2
out amix
ks rms gar+amix
chnset ks, "meter"
gar = 0
endin

schedule(100,0,-1)
'''

class Application(tkinter.Frame):
 def move(self,event):
    canvas = event.widget
    x = canvas.canvasx(event.x)
    y = canvas.canvasy(event.y)
    item = canvas.find_withtag("current")[0]
    canvas.coords(item, x+10, y+10, x-10, y-10)
    self.cs.setControlChannel("pitch",
        0.5+1.5*x/self.size)
    self.cs.setControlChannel("volume",
        2.0*(self.size-y)/self.size)

 def play(self,event):
    note = event.widget.find_withtag("current")[0]
```

```
    self.canvas.itemconfigure(note, fill="red")
    self.perf.inputMessage("i1 0 -1 0.5 440")

 def stop(self,event):
    note = event.widget.find_withtag("current")[0]
    self.canvas.itemconfigure(note, fill="black")
    self.perf.inputMessage("i-1 0 0.5 440")

 def createCanvas(self):
    self.size = 600
    self.canvas = tkinter.Canvas(self,height=self.size,
                          width=self.size, bg="violet")
    self.canvas.pack()

 def createCircle(self):
      circle = self.canvas.create_oval(self.size/2-10,
                                       self.size/2-10,
                                       self.size/2+10,
                                       self.size/2+10,
                                       fill="black")

      self.canvas.tag_bind(circle,"<ButtonPress>",
                           self.play)
      self.canvas.tag_bind(circle,"<B1-Motion>",
                           self.move)
      self.canvas.tag_bind(circle,"<ButtonRelease>",
                           self.stop)

 def createMeter(self):
  iw = 10
  self.vu = []
  for i in range(0, self.size, iw):
   self.vu.append(self.canvas.create_rectangle(i,
        self.size-40,i+iw,self.size,fill="grey"))

 def drawMeter(self):
    level,err = self.cs.controlChannel("meter")
    cnt = 0
    level *= 16000
    red = (self.size/10)*0.8
    yellow = (self.size/10)*0.6
    for i in self.vu:
      if level > cnt*100:
       if cnt > red:
        self.canvas.itemconfigure(i, fill="red")
```

```
        elif cnt > yellow:
         self.canvas.itemconfigure(i, fill="yellow")
        else:
         self.canvas.itemconfigure(i, fill="blue")
       else:
        self.canvas.itemconfigure(i, fill="grey")
       cnt  = cnt + 1
    self.master.after(50,self.drawMeter)

def quit(self):
    self.perf.stop()
    self.perf.join()
    self.master.destroy()

def createEngine(self):
    self.cs = ctcsound.Csound()
    res = self.cs.compileOrc(code)
    self.cs.setOption('-odac')
    if res == 0:
     self.cs.start()
     self.cs.setControlChannel('pitch', 1.0)
     self.cs.setControlChannel('volume', 1.0)
     self.perf =
      ctcsound.CsoundPerformanceThread(self.cs.csound())
     self.perf.play()
     return True
    else:
     return False

def __init__(self,master=None):
    tkinter.Frame.__init__(self,master)
    self.master.title('Csound + Tkinter: '
     'just click and play')
    self.master = master
    self.pack()
    self.createCanvas()
    self.createCircle()
    self.createMeter()
    if self.createEngine() is True:
      self.drawMeter()
      self.master.protocol('WM_DELETE_WINDOW',
                           self.quit)
      self.master.mainloop()
    else: self.master.quit()
```

```
Application(tkinter.Tk())
```

B.2 Vocal Quartet Simulation

Listing B.2 provides a full example of a vocal quartet simulation, based on a variant of the instrument discussed in chapter 3, including the use of a global reverb instrument.

Listing B.2 Csound vocal simulation.

```
gasig init 0
opcode Filter,a,akk
 as,kf,kbw xin
 as = reson(as,kf,kbw,1)
 as = reson(as,kf,kbw,1)
    xout(as)
endop

instr 1
 kf[][] init 5,5
 kb[][] init 5,5
 ka[][] init 5,5

 kf[][] fillarray 800,1150,2800,3700,4950,
                  450,800,2830,3500,4950,
                  400,1600,2700,3500,4950,
                  350,1700,2700,3700,4950,
                  325,700,2530,3500,4950
 kb[][] fillarray 80,90,120,130,140,
                  70,80,100,130,135,
                  60,80,120,150,200,
                  50,100,120,150,200,
                  50,60,170,180,200
 ka[][] fillarray 0,4,20,36,60,
                  0,9,16,28,55,
                  0,24,30,35,60,
                  0,20,30,36,60,
                  0,12,30,40,64

 kv = linseg(p6,p3/2-p7,p6,p7,p8,p3/2,p8)
 itim = p7

 kf0 = port(kf[kv][0],itim,i(kf[p6][0]))
```

```
 kf1 = port(kf[kv][1],itim,i(kf[p6][1]))
 kf2 = port(kf[kv][2],itim,i(kf[p6][2]))
 kf3 = port(kf[kv][3],itim,i(kf[p6][3]))
 kf4 = port(kf[kv][4],itim,i(kf[p6][4]))

 kb0 = port(kb[kv][0],itim,i(kb[p6][0]))
 kb1 = port(kb[kv][1],itim,i(kb[p6][1]))
 kb2 = port(kb[kv][2],itim,i(kb[p6][2]))
 kb3 = port(kb[kv][3],itim,i(kb[p6][3]))
 kb4 = port(kb[kv][4],itim,i(kb[p6][4]))

 ka0 = port(ampdb(-ka[kv][0]),itim,-i(ka[p6][0]))
 ka1 = port(ampdb(-ka[kv][1]),itim,-i(ka[p6][0]))
 ka2 = port(ampdb(-ka[kv][2]),itim,-i(ka[p6][0]))
 ka3 = port(ampdb(-ka[kv][3]),itim,-i(ka[p6][0]))
 ka4 = port(ampdb(-ka[kv][4]),itim,-i(ka[p6][0]))

 kjit = randi(p5*0.01,15,2)
 kvib = oscili(p5*(0.01+
            gauss:k(0.005)),
            4+gauss:k(0.05))

 kfun = p5+kjit+kvib
 a1 = buzz(p4,kfun,sr/(2*kfun),-1,rnd(0.1))
 kf0 = kfun > kf0 ? kfun : kf0
 af1 = Filter(a1*ka0,kf0,kb0)
 af2 = Filter(a1*ka1,kf1,kb1)
 af3 = Filter(a1*ka2,kf2,kb2)
 af4 = Filter(a1*ka3,kf3,kb3)
 af5 = Filter(a1*ka4,kf4,kb4)

 amix = delay(af1+af2+af3+af4+af5,rnd(0.1))
 asig= linenr(amix*5,0.01,0.05,0.01)
   out(asig)
   gasig += asig
endin

instr 2
 a1,a2 reverbsc gasig,gasig,0.8,3000
  out (a1+a2)/3
  gasig = 0
endin

schedule(2,0,-1)
icnt init 0
```

```
while icnt < 4 do
schedule(1,0+icnt*20,4.9,0dbfs,cpspch(7.04),
        3,1+gauss(0.3),4)
schedule(1,5+icnt*20,4.9,0dbfs,cpspch(7.02),
        4,1+gauss(0.3),2)
schedule(1,10+icnt*20,4.9,0dbfs,cpspch(7.00),
        2,1+gauss(0.3),1)
schedule(1,15+icnt*20,4.9,0dbfs,cpspch(6.11),
        1,1+gauss(0.3),3)
if icnt > 0 then
 schedule(1,0+icnt*20,4.9,0dbfs,cpspch(8.07),
        3,1+gauss(0.3),4)
 schedule(1,5+icnt*20,4.9,0dbfs,cpspch(8.06),
        4,1+gauss(0.3),2)
 schedule(1,10+icnt*20,4.9,0dbfs,cpspch(8.04),
        2,1+gauss(0.3),1)
 schedule(1,15+icnt*20,4.9,0dbfs,cpspch(8.06),
        1,1+gauss(0.3),3)
endif
if icnt > 1 then
 schedule(1,0+icnt*20,4.9,0dbfs*1.5,cpspch(9.04),
        3,1+gauss(0.3),4)
 schedule(1,5+icnt*20,4.9,0dbfs*1.2,cpspch(8.11),
        4,1+gauss(0.3),2)
 schedule(1,10+icnt*20,4.9,0dbfs,cpspch(9.00),
        2,1+gauss(0.3),1)
 schedule(1,15+icnt*20,4.9,0dbfs*1.3,cpspch(9.03),
        1,1+gauss(0.3),3)
endif
if icnt > 2 then
 schedule(1,0+icnt*20,4.9,0dbfs,cpspch(7.07),
        3,1+gauss(0.3),4)
 schedule(1,5+icnt*20,4.9,0dbfs,cpspch(7.11),
        4,1+gauss(0.3),2)
 schedule(1,10+icnt*20,4.9,0dbfs,cpspch(7.07),
        2,1+gauss(0.3),1)
 schedule(1,15+icnt*20,4.9,0dbfs,cpspch(7.09),
        1,1+gauss(0.3),3)
 endif
icnt += 1
od

schedule(1,icnt*20,4.9,0dbfs,cpspch(7.04),
        3,1+gauss(0.3),4)
schedule(1,icnt*20,4.9,0dbfs,cpspch(8.07),
```

```
        3,1+gauss(0.3),4)
schedule(1,icnt*20,4.9,0dbfs,cpspch(9.04),
        3,1+gauss(0.3),4)
schedule(1,icnt*20,4.9,0dbfs,cpspch(7.11),
        3,1+gauss(0.3),4)
event_i("e",0,icnt*20+8,0)
```

B.3 Closed-Form Summation Formulae User-Defined Opcodes

The following listings provide ready-to-use implementations of the techniques explored in chapter 4.

Listing B.3 Bandlimited pulse UDO.

```
/*********************************
asig Blp kamp,kfreq
kamp - amplitude
kfreq - fundamental frequency
*********************************/
opcode Blp,a,kki
  setksmps 1
  kamp,kf  xin
  kn = int(sr/(2*kf))
  kph phasor kf/2
  kden tablei kph,-1,1
  if kden != 0 then
      knum tablei kph*(2*kn+1),-1,1,0,1
      asig = (kamp/(2*kn))*(knum/kden - 1)
   else
   asig = kamp
  endif
  xout asig
endop
```

Listing B.4 Generalised summation-formulae UDO.

```
/*********************************
asig Blsum kamp,kfr1,kfr2,ka
kamp - amplitude
kfr1 - frequency 1 (omega)
kfr2 - frequency 2 (theta)
ka - distortion amount
*********************************/
opcode Blsum,a,kkkki
```

```
 kamp,kw,kt,ka xin
 kn = int(((sr/2) - kw)/kt)
 aphw phasor kw
 apht phasor kt
 a1 tablei aphw,-1,1
 a2 tablei aphw - apht,-1,1,0,1
 a3 tablei aphw + (kn+1)*apht,-1,1,0,1
 a4 tablei aphw + kn*apht,-1,1,0,1
 acos tablei apht,-1,1,0.25,1
 kpw pow ka,kn+1
 ksq = ka*ka
 aden = (1 - 2*ka*acos + ksq)
 asig = (a1 - ka*a2 - kpw*(a3 - ka*a4))/aden
 knorm = sqrt((1-ksq)/(1 - kpw*kpw))
 xout asig*kamp*knorm
endop
```

Listing B.5 Non-bandlimited generalised summation-formulae UDO.

```
/*********************************
asig NBlsum kamp,kfr1,kfr2,ka
kamp - amplitude
kfr1 - frequency 1 (omega)
kfr2 - frequency 2 (theta)
ka - distortion amount
*********************************/
opcode NBlsum,a,kkkk
 kamp,kw,kt,ka  xin
 aphw phasor kw
 apht phasor kt
 a1 tablei aphw,-1,1
 a2 tablei aphw - apht,itb,1,0,1
 acos tablei apht,-1,1,0.25,1
 ksq = ka*ka
 asig = (a1 - ka*a2)/(1 - 2*ka*acos + ksq)
 knorm = sqrt(1-ksq)
 xout asig*kamp*knorm
endop
```

Listing B.6 PM synthesis UDO.

```
/*********************************************
asig  PM kamp,kfc,kfm,kndx
kamp - amplitude
kfc - carrier frequency
kfm - modulation frequency
kndx - distortion index
```

```
*********************************************/
opcode PM,a,kkkk
 kamp,kfc,kfm,kndx xin
 acar phasor kfc
 amod oscili kndx/(2*$M_PI),kfm
 apm  tablei acar+amod,-1,1,0.25,1
     xout apm*kamp
endop
```

Listing B.7 PM operator UDO.

```
/*****************************************
asig PMOp kamp,kfr,apm,iatt,idec,isus,irel[,ifn]
kamp - amplitude
kfr - frequency
apm - phase modulation input
iatt - attack
idec - decay
isus - sustain
irel - release
ifn - optional wave function table (defaults to sine)
*****************************************/
opcode PMOp,a,kkaiiiij
 kmp,kfr,apm,
    iatt,idec,
    isus,irel,ifn xin
 aph phasor kfr
 a1  tablei aph+apm/(2*$M_PI),ifn,1,0,1
 a2  madsr iatt,idec,isus,irel
     xout  a2*a1*kmp
endop
```

Listing B.8 Asymmetric FM UDO.

```
f5 0 131072 "exp" 0 -50 1
/*****************************************
asig Asfm kamp,kfc,kfm,kndx,kR,ifn,imax
kamp - amplitude
kfc - carrier frequency
kfm - modulation frequency
kndx - distortion index
ifn - exp func between 0 and -imax
imax - max absolute value of exp function
*****************************************/
opcode Asfm,a,kkkkkii
 kamp,kfc,kfm,knx,kR,ifn,imax
 kndx = knx*(kR+1/kR)*0.5
```

```
 kndx2 = knx*(kR-1/kR)*0.5
 afm oscili kndx/(2*$M_PI),kfm
 aph phasor kfc
 afc tablei aph+afm,ifn,1,0,1
 amod oscili kndx2, kfm, -1, 0.25
 aexp tablei -(amod-abs(kndx2))/imx, ifn, 1
         xout kamp*afc*aexp
endop
```

Listing B.9 PAF UDO.

```
opcode Func,a,a
 asig xin
       xout 1/(1+asig^2)
endop

/*************************************
asig PAF kamp,kfun,kcf,kfshift,kbw
kamp - amplitude
kfun - fundamental freq
kcf - centre freq
kfshift - shift freq
kbw - bandwidth
*************************************/
opcode PAF,a,kkkkki
  kamp,kfo,kfc,kfsh,kbw  xin
  kn = int(kfc/kfo)
  ka = (kfc - kfsh - kn*kfo)/kfo
  kg = exp(-kfo/kbw)
  afsh phasor kfsh
  aphs phasor kfo/2
  a1 tablei 2*aphs*kn+afsh,-1,1,0.25,1
  a2 tablei 2*aphs*(kn+1)+afsh,-1,1,0.25,1
  asin tablei aphs, 1, 1, 0, 1
  amod Func 2*sqrt(kg)*asin/(1-kg)
  kscl = (1+kg)/(1-kg)
  acar = ka*a2+(1-ka)*a1
  asig = kscl*amod*acar
         xout asig*kamp
endop
```

Listing B.10 ModFM UDO.

```
/*****************************************
asig ModFM kamp,kfc,kfm,kndx,ifn,imax
kamp - amplitude
kfc - carrier frequency
```

```
kfm - modulation frequency
kndx - distortion index
ifn - exp func between 0 and -imax
imax - max  absolute value of exp function
*******************************************/
opcode ModFM,a,kkkkiii
 kamp,kfc,kfm,kndx,iexp,imx xin
 acar oscili kamp,kfc,-1,0.25
 acos oscili 1,kfm,-1,0.25
 amod table -kndx*(acos-1)/imx,iexp,1
           xout acar*amod
endop
```

Listing B.11 ModFM formant synthesis UDO.

```
/*****************************************
asig ModForm kamp,kfo,kfc,kbw,ifn,imax
kamp - amplitude
kfo - fundamental frequency
kfc - formant centre frequency
kbw - bandwidth
ifn - exp func between 0 and -imax
imax - max absolute value of exp function
*******************************************/
opcode ModForm,a,kkkkii
 kamp,kfo,kfc,kbw,ifn,itm  xin
 ioff = 0.25
 itab = -1
 icor = 4.*exp(-1)
 ktrig changed kbw
 if ktrig == 1 then
  k2 = exp(-kfo/(.29*kbw*icor))
  kg2 = 2*sqrt(k2)/(1.-k2)
  kndx = kg2*kg2/2.
 endif
 kf = kfc/kfo
 kfin = int(kf)
 ka = kf  - kfin
 aph   phasor kfo
 acos tablei aph, 1, 1, 0.25, 1
 aexp table kndx*(1-acos)/itm,ifn,1
 acos1 tablei aph*kfin, itab, 1, ioff, 1
 acos2 tablei aph*(kfin+1), itab, 1, ioff, 1
 asig = (ka*acos2 + (1-ka)*acos1)*aexp
 xout asig*kamp
endop
```

Listing B.12 Waveshaping UDO.

```
/*********************************************
asig  Waveshape kamp,kfreq,kndx,ifn1,ifn2
kamp - amplitude
kfreq - frequency
kndx - distortion index
ifn1 - transfer function
ifn2 - scaling function
*********************************************/
opcode Waveshape,a,kkkiii
 kamp,kf,kndx,itf,igf xin
 asin oscili 0.5*kndx,kf
 awsh tablei asin,itf,1,0.5
 kscl tablei kndx,igf,1
 xout awsh*kamp*kscl
endop
```

Listing B.13 Sawtooth wave oscillator based on waveshaping.

```
f2 0 16385 "tanh" -157 157
f3 0 8193 4 2 1
/*********************************************
asig  Sawtooth kamp,kfreq,kndx,ifn1,ifn2
kamp - amplitude
kfreq - frequency
kndx - distortion index
ifn1 - transfer function
ifn2 - scaling function
*********************************************/
opcode Sawtooth,a,kkkii
 kamp,kf,kndx,itf,igf xin
 amod oscili 1,kf,-1,0.25
 asq Waveshape kamp*0.5,kf,kndx,-1,itf,igf
        xout asq*(amod + 1)
endop
```

B.4 Pylab Waveform and Spectrum Plots

The code for plotting waveforms and spectra from an input signal is presented in Listing B.14. The function in this example can be dropped into any Python program that produces a signal in a NumPy array.

Listing B.14 Pylab example code for waveform and spectrum plotting.

```
import pylab as pl

def plot_signal(s,sr,start=0,end=440,N=32768):
 '''plots the waveform and spectrum of a signal s
    with sampling rate sr, from a start to an
    end position (waveform), and from a start
    position with a N-point DFT (spectrum)'''
 pl.figure(figsize=(8,5))

 pl.subplot(211)
 sig = s[start:end]
 time = pl.arange(0,len(sig))/sr
 pl.plot(time,sig, 'k-')
 pl.ylim(-1.1,1.1)
 pl.xlabel("time (s)")

 pl.subplot(212)
 N = 32768
 win = pl.hanning(N)
 scal = N*pl.sqrt(pl.mean(win**2))
 sig = s[start:start+N]
 window = pl.rfft(sig*win/max(sig))
 f = pl.arange(0,len(window))
 bins = f*sr/N
 mags = abs(window/scal)
 spec = 20*pl.log10(mags/max(mags))
 pl.plot(bins,spec, 'k-')
 pl.ylim(-60, 1)
 pl.ylabel("amp (dB)", size=16)
 pl.xlabel("freq (Hz)", size=16)
 pl.yticks()
 pl.xticks()
 pl.xlim(0,sr/2)

 pl.tight_layout()
 pl.show()
```

B.5 FOF Vowel Synthesis

The code in Listing B.15 demonstrates FOF synthesis of vowel sounds, together with a transition to a granular texture. The instrument has a large number of parameters, defining the time-varying features of the sound:

- p4: amplitude
- p5, p6: start, end fundamental
- p7 - p11: formant amplitudes
- p12, p13: start, end octave transposition
- p14, p15: start, end grain attack time
- p16, p17: start, end bandwidth
- p18, p19: start, end grain size
- p20, p21: start, end vowel cycle frequency
- p22: vowel jitter

Listing B.15 FOF vowel synthesis and granulation.

```
gif = ftgen(0,0,16384,7,0,16384,1)
gif1 = ftgen(0,0,8,-2,400,604,325,325,
                360,604,238,360)
gif2 = ftgen(0,0,8,-2,1700,1000,700,700,
                750,1000,1741,750)
gif3 = ftgen(0,0,8,-2,2300,2450,2550,2550,
                2400,2450,2450,2400)
gif4 = ftgen(0,0,8,-2,2900,2700,2850,2850,
                2675,2700,2900,2675)
gif5 = ftgen(0,0,8,-2,3400,3240,3100,3100,
                2950,3240,4000,2950)

instr 1
 iscal = p4/(p7+p8+p9+p10+p11)
 iol = 300

 ksl = line(p20,p3,p21)
 kv = rand(p22)
 kff = phasor(ksl+kv)
 kf1 = tablei(kff,gif1,1,0,1)
 kf2 = tablei(kff,gif2,1,0,1)
 kf3 = tablei(kff,gif3,1,0,1)
 kf4 = tablei(kff,gif4,1,0,1)
 kf5 = tablei(kff,gif5,1,0,1)

 koc = linseg(p12,p3*.4,p12,p3*.3,p13,p3*.3,p13)
 kat = linseg(p14,p3*.3,p14,p3*.2,p15,p3*.5,p15)
```

```
 kbw = linseg(p16,p3*.25,p16,p3*.15,p17,p3*.6,p17)
 ks  = linseg(p18,p3*.25,p18,p3*.15,p19,p3*.6,p19)
 idc = .007

 aj1 = randi(.02,2)
 aj2 = randi(.01,.75)
 aj3 = randi(.02,1.2)
 av = oscil(.007,7)
 kf = line(p5, p3, p6)
 af =(cpspch(kf))*(aj1+aj2+aj3+1)*(av+1)

 a1 = fof(p7,af,kf1,koc,kbw,kat,ks,idc,iol,-1,gif,p3)
 a2 = fof(p8,af,kf2,koc,kbw,kat,ks,idc,iol,-1,gif,p3)
 a3 = fof(p9,af,kf3,koc,kbw,kat,ks,idc,iol,-1,gif,p3)
 a4 = fof(p10,af,kf4,koc,kbw,kat,ks,idc,iol,-1,gif,p3)
 a5 = fof(p11,af,kf5,koc,kbw,kat,ks,idc,iol,-1,gif,p3)
 amx = (a1+a2+a3+a4+a5)*iscal
     out(amx)
endin

schedule(1,0,60,
        0dbfs/3,6.09,6.07,
        1,.5,.3,.1,.01,
        0,3,.003,.01,
        100,10,.02,.1,
        0.5,.01,.01)
```

B.6 Convolution Programs

The following program outputs the convolution of two input files, using an FFT overlap-add method. It works with audio files of the RIFF-Wave type, taking three arguments,

```
$ python3 conv.py input1.wav input2.wav output.wav
```

Listing B.16 FFT-based convolution program.

```
import pylab as pl
import scipy.io.wavfile as wf
import sys

def conv(signal,ir):
```

```
    N = len(ir)
    L = len(signal)
    M = 2
    while(M <= 2*N-1): M *= 2

    h = pl.zeros(M)
    x = pl.zeros(M)
    y = pl.zeros(L+N-1)

    h[0:N] = ir
    H = pl.rfft(h)
    n = N
    for p in range(0,L,N):
     if p+n > L:
          n = L-p
          x[n:] = pl.zeros(M-n)
     x[0:n] = signal[p:p+n]
     y[p:p+2*n-1] += pl.irfft(H*pl.rfft(x))[0:2*n-1]
    return y

(sr,x1) = wf.read(sys.argv[1])
(sr,x2) = wf.read(sys.argv[2])
scal = 32768
if len(x1) > len(x2):
 y = conv(x1,x2/scal)
 a = max(x1)
else:
 y = conv(x2, x1/scal)
 a = max(x2)

s = max(y)
out = pl.array(y*a/s,dtype='int16')
wf.write(sys.argv[3],sr,out)
```

The next program applies the partitioned convolution method to split the impulse response into segments of 1024 samples. It works with the same command-line parameters as in the previous case.

Listing B.17 Uniform partitioned convolution program.

```
iimport pylab as pl
import scipy.io.wavfile as wf
import sys

def pconv(input,ir,S=1024):
    N = len(ir)
    L = len(input)
```

```
    M = S*2

    P = int(pl.ceil(N/S))
    H = pl.zeros((P,M//2+1)) + 0j
    X = pl.zeros((P,M//2+1)) + 0j
    x = pl.zeros(M)
    o = pl.zeros(M)
    s = pl.zeros(S)

    for i in range(0,P):
        p = (P-1-i)*S
        ll = len(ir[p:p+S])
        x[:ll] = ir[p:p+S]
        H[i] = pl.rfft(x)

    y = pl.zeros(L+N-1)
    n = S
    i = 0
    for p in range(0,L+N-S,S):
      if p > L:
        x = pl.zeros(M)
      else:
       if n+p > L:
         n = L - p
         x[n:] = pl.zeros(M-n)
       x[:n] = input[p:p+n]
      X[i] = pl.rfft(x)
      i = (i+1)%P
      O = pl.zeros(M//2+1) + 0j
      for j in range(0,P):
        k = (j+i)%P
        O += H[j]*X[k]
      o = pl.irfft(O)
      y[p:p+S] = o[:S] + s
      s = o[S:]

    return y

(sr,x1) = wf.read(sys.argv[1])
(sr,x2) = wf.read(sys.argv[2])

scal = 32768
L1 = len(x1)
L2 = len(x2)
y = pl.zeros(L1+L2-1)
```

```
m = 0
if L1 > L2:
 y = pconv(x1,x2/scal)
 a = max(x1)
else:
 y = pconv(x2,x1/scal)
 a = max(x2)
s = max(y)
scal = a/s
out = pl.array(y*scal,dtype='int16')
wf.write(sys.argv[3],sr,out)
```

The next example is the implementation of non-uniform partitioned convolution in Csound, as a UDO. It uses `dconv` for direct convolution and `ftconv` for fast partitioned convolution.

Listing B.18 Csound UDO for non-uniform partitioned convolution.

```
/*******************************************
asig  Pconv ain,ifn
kamp - input signal
ifn - IR function table
*******************************************/
opcode Pconv,a,ai
 asig,ifn xin
 a1 dconv asig,32,ifn
 a2 ftconv asig,ifn,32,32,96
 a3 ftconv asig,ifn,128,128,896
 a4 ftconv asig,ifn,1024,1024
 xout a1+a2+a3+a4
endop
```

B.7 Spectral Masking

The following program is a skeleton for a spectral masking application. The function `mask(mags,...)` can be replaced by a different user-defined operation on the magnitude spectrum, taking the original values and returning a modified array. The example supplied creates a band-pass filter centred at `fc`. The command-line for this program takes two file names (input and output):

```
$ python3 mask.py input1.wav output.wav
```

Listing B.19 Spectral masking program.

```
import pylab as pl
import scipy.io.wavfile as wf
import sys

def stft(x,w):
    X = pl.rfft(w*x)
    return X

def istft(X,w):
    xn = pl.irfft(X)
    return xn*w

def p2r(mags, phs):
    return mags*pl.cos(phs) + 1j*mags*pl.sin(phs)

def mask(mags,fc):
    m = pl.zeros(N//2+1)
    d = int((fc/2)*N/sr)
    b = int(fc*N/sr)
    m[b-d:b+d] = pl.hanning(2*d)
    mags *= m
    return mags

N = 1024
D = 4
H = N//D
zdbfs = 32768
(sr,signal) = wf.read(sys.argv[1])
signal = signal/zdbfs
L = len(signal)
output = pl.zeros(L)
win = pl.hanning(N)
scal = 1.5*D/4

fc = 1000
fe = 5000
incr = (fe - fc)*(H/L)

for n in range(0,L,H):
    if(L-n < N): break
    frame = stft(signal[n:n+N],win,n)
    mags = abs(frame)
    phs = pl.angle(frame)
    mags = mask(mags,fc)
```

```
    fc += incr
    frame = p2r(mags,phs)
    output[n:n+N] += istft(frame,win,n)

output = pl.array(output*zdbfs/scal,dtype='int16')
wf.write(sys.argv[2],sr,output)
```

B.8 Cross-synthesis

In this section we present a cross-synthesis program example. The process implemented here is the spectral vocoder (see 7.2.2), but this code can be used as a skeleton for other types of cross-synthesis applications. The command-line for this program is:

```
$ python3 cross.py input1.wav input2.wav output.wav
```

Listing B.20 Spectral vocoder program.

```
import pylab as pl
import scipy.io.wavfile as wf
import sys

def stft(x,w):
    X = pl.rfft(w*x)
    return X

def istft(X,w):
    xn = pl.irfft(X)
    return xn*w

def p2r(mags, phs):
    return mags*pl.cos(phs) + 1j*mags*pl.sin(phs)

def spec_env(frame,coefs):
    mags = abs(frame)
    N = len(mags)
    ceps = pl.rfft(pl.log(mags[:N-1]))
    ceps[coefs:] = 0
    mags[:N-1] = pl.exp(pl.irfft(ceps))
    return mags

N = 1024
```

```
D = 4
H = N//D
zdbfs = 32768
(sr,in1) = wf.read(sys.argv[1])
(sr,in2) = wf.read(sys.argv[2])
L1 = len(in1)
L2 = len(in2)
if L2 > L1: L = L2
else: L = L1
signal1 = pl.zeros(L)
signal2 = pl.zeros(L)
signal1[:len(in1)] = in1/zdbfs
signal2[:len(in2)] = in2/zdbfs
output = pl.zeros(L)
win = pl.hanning(N)
scal = 1.5*D/4

for n in range(0,L,H):
    if(L-n < N): break
    frame1 = stft(signal1[n:n+N],win,n)
    frame2 = stft(signal2[n:n+N],win,n)
    mags = abs(frame1)
    env1 = spec_env(frame1,20)
    env2 = spec_env(frame2,20)
    phs = pl.angle(frame1)
    if(min(env1) > 0):
     frame = p2r(mags*env2/env1,phs)
    else:
     frame = p2r(mags*env2,phs)
    output[n:n+N] += istft(frame,win,n)

a = max(signal1)
b = max(output)
c = a/b
output = pl.array(output*zdbfs*c/scal,dtype='int16')
wf.write(sys.argv[3],sr,output)
```

B.9 Pitch Shifting

A pitch shifter with formant preservation is shown in Listing B.21. A version of this program without this feature can be created by removing the spectral envelope scaling factor in the phase vocoder synthesis method call. Its command line takes a

number for the pitch ratio and two files:

```
$ python3 pshift.py <ratio> input.wav output.wav
```

Listing B.21 Pitch shifter program with formant preservation.

```
import pylab as pl
import scipy.io.wavfile as wf
import sys

def stft(x,w,n):
    N = len(w)
    X = pl.rfft(x*w)
    k = pl.arange(0,N/2+1)
    return X*pl.exp(-2*pl.pi*1j*k*n/N)

def istft(X,w,n):
    N = len(w)
    k = pl.arange(0,N/2+1)
    xn = pl.irfft(X*pl.exp(2*pl.pi*1j*k*n/N))
    return xn*w

def modpi(x):
   if x >= pl.pi: x -= 2*pl.pi
   if x < -pl.pi: x += 2*pl.pi
   return x
unwrap = pl.vectorize(modpi)

class Pvoc:
   def __init__(self,w,h,sr):
     N = len(w)
     self.win = w
     self.phs = pl.zeros(N//2+1)
     self.h = h
     self.n = 0
     self.fc = pl.arange(N//2+1)*sr/N
     self.sr = sr

   def analysis(self,x):
     X = stft(x,self.win,self.n)
     delta = pl.angle(X) - self.phs
     self.phs = pl.angle(X)
     delta = unwrap(delta)
     f = self.fc + delta*self.sr/(2*pl.pi*self.h)
     self.n += self.h
     return abs(X),f
```

```
   def synthesis(self,a,f):
      delta = (f - self.fc)*(2*pl.pi*self.h)/self.sr
      self.phs += delta
      X = a*pl.cos(self.phs) + 1j*a*pl.sin(self.phs)
      y = istft(X,self.win,self.n)
      self.n += self.h
      return y

N = 1024
D = 8
H = N//D
zdbfs = 32768
(sr,signal) = wf.read(sys.argv[2])
signal = signal/zdbfs
L = len(signal)
output = pl.zeros(L)
win = pl.hanning(N)

pva = Pvoc(win,H,sr)
pvs = Pvoc(win,H,sr)
trans = float(sys.argv[1])

def scale(amps,freqs,p):
    N = len(freqs)
    a,f = pl.zeros(N),pl.zeros(N)
    for i in range(0, N):
       n = int((i*p))
       if n > 0 and n < N-1:
          f[n] = p*freqs[i]
          a[n] = amps[i]
    return a,f

def true_spec_env(amps,coefs,thresh):
   N = len(amps)
   sm = 10e-15
   form = pl.zeros(N)
   lmags = pl.log(amps[:N-1]+sm)
   mags = lmags
   check = True
   while(check):
     ceps = pl.rfft(lmags)
     ceps[coefs:] = 0
     form[:N-1] = pl.irfft(ceps)
     for i in range(0,N-1):
```

```
      if lmags[i] < form[i]: lmags[i] = form[i]
      diff = mags[i] - form[i]
      if diff > thresh: check = True
      else: check = False
   return pl.exp(form)+sm

for n in range(0,L,H):
    if(L-n < N): break
    amps,freqs = pva.analysis(signal[n:n+N])
    env1 = true_spec_env(amps,40,0.23)
    amps,freqs = scale(amps,freqs,trans)
    env2 = true_spec_env(amps,40,0.23)
    output[n:n+N] += pvs.synthesis(amps*env1/env2,freqs)

scal = max(output)/max(signal)
output = pl.array(output*zdbfs/scal,dtype='int16')
wf.write(sys.argv[3],sr,output)
```

B.10 Time Scaling

The example in this section implements time scaling with phase locking. Its command line is:

```
$ python3 tscale.py <ratio> input.wav output.wav
```

Listing B.22 Time scaling program

```
import pylab as pl
import scipy.io.wavfile as wf
import sys

def stft(x,w):
    X = pl.rfft(x*w)
    return X

def istft(X,w):
    xn = pl.irfft(X)
    return xn*w

N = 1024
D = 8
H = N//D
```

```
zdbfs = 32768
(sr,signal) = wf.read(sys.argv[2])
signal = signal/zdbfs
L = len(signal)
win = pl.hanning(N)
Z = pl.zeros(N//2+1)+0j+10e-20
np = 0
ts = float(sys.argv[1])
output = pl.zeros(int(L/ts)+N)
ti = int(ts*H)
scal = 3*D/4

def plock(x):
    N = len(x)
    y = pl.zeros(N)+0j
    y[0], y[N-1] = x[0], x[N-1]
    y[1:N-1] = x[1:N-1] - (x[0:N-2] + x[2:N])
    return y

Z = pl.zeros(N//2+1)+0j+10e-20
for n in range(0,L-(N+H),ti):
    X1 = stft(signal[n:n+N],win)
    X2 = stft(signal[n+H:n+H+N],win)
    Y = X2*(Z/X1)*abs(X1/Z)
    Z = plock(Y)
    output[np:np+N] += istft(Y,win)
    np += H

output = pl.array(output*zdbfs/scal,dtype='int16')
wf.write(sys.argv[3],sr,output)
```

References

1. Allan, R.: A History of the Personal Computer: The People and the Technology. Allan Pub. (2001)
2. Arfib, D.: Digital synthesis of complex spectra by means of multiplication of non-linear distorted sine waves. In: Audio Engineering Society Convention 59 (1978)
3. Askill, J.: The Physics of Musical Sound. Van Nostrand, New York (1979)
4. Atmel Corp.: Home page Atmel Microprocessors (2016). URL http://www.atmel.com
5. Backus, J.: The Acoustical Foundations of Music. W.W. Norton, New York (1977)
6. Beauchamp, J.: Analysis and synthesis of musical instrument sounds. In: J. Beauchamp (ed.) Analysis, Synthesis, and Perception of Musical Sounds: The Sound of Music, Modern Acoustics and Signal Processing, pp. 1–89. Springer, New York (2007)
7. Beaudouin-Lafon, M.: Instrumental interaction: An interaction model for designing post-WIMP user interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '00, pp. 446–453. ACM, New York, NY, USA (2000)
8. Benade, A.H.: Fundamentals of Musical Acoustics. Dover, New York (1976)
9. Benson, D.: Music, A Mathematical Offering. Cambridge University Press, Cambridge (2006)
10. Bogert, B., Healy, M., Tukey, J.: The quefrency alanysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking. In: Proceedings Symposium on Time Series Analysis, pp. 209–243 (1963)
11. Bracewell, R.: The Fourier Transform and Its Applications. Electrical Engineering Series. McGraw-Hill, New York (2000)
12. Brandtsegg, Ø., Saue, S., Johansen, T.: Particle synthesis, a unified model for granular synthesis. In: Proceedings of the Linux Audio Conference 2011 (2011)
13. Brun, M.L.: A derivation of the spectrum of FM with a complex modulating wave. Computer Music Journal **1**(4), 51–52 (1977)
14. Cherniakov, M.: An Introduction to Parametric Digital Filters and Oscillators. John Wiley & Sons, New York (2003)
15. Chowning, J.: The synthesis of complex spectra by means of frequency modulation. Journal of the Audio Engineering Society **21**(7), 526–534 (1973)
16. Chu, E., George, A.: Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms. Computational Mathematics. Taylor & Francis (1999)
17. Cook, P.: A toolkit of audio synthesis classes and instruments in C++ (1995). URL https://ccrma.stanford.edu/software/stk/
18. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation **19**(90), 297–301 (1965)
19. Damasevicius, R., Stuikys, V.: Separation of concerns in multi-language specifications. Informatica **13**(3), 255–274 (2002)

V. Lazzarini, *Computer Music Instruments*,
https://doi.org/10.1007/978-3-319-63504-0

20. Dattorro, J.: Effect design, part 2: Delay line modulation and chorus. Journal of the Audio Engineering Society **45**(10), 764–788 (1997)
21. Dirichlet, P.G.L.: Sur la convergence des séries trigonometriques qui servent à répresenter une fonction arbitraire entre des limites donneés. Journal für die reine und angewandte Mathematik **4**, 157–169 (1829)
22. Dodge, C., Jerse, T.A.: Computer Music: Synthesis, Composition and Performance, 2nd edn. Schirmer, New York (1997)
23. Dolson, M.: The phase vocoder: A tutorial. Computer Music Journal **10**(4), 14–27 (1986)
24. Dougherty, D.: The Maker movement. Innovations **7**(3), 11–14
25. Flanagan, F., Golden, R.: Phase vocoder. Bell System Technical Journal **45**, 1493–1509 (1966)
26. Fourier, J.B.: Théorie analytique de la chaleur. Chez Firmin Didot, Père et fils, Paris (1822)
27. Gardner, W.G.: Efficient convolution without input-output delay. Journal of the Audio Engineering Society **43**(3), 127–136 (1995)
28. Gentner, D., Nielsen, J.: The Anti-Mac interface. Commun. ACM **39**(8), 70–82 (1996)
29. Grey, J.: An Exploration of Musical Timbre Using Computer-based Techniques. Department of Psychology, Stanford University, Stanford (1975)
30. Harris, F.: On the use of windows for harmonic analysis with the discrete Fourier transform. Proceedings of the IEEE **66**(1), 51–83 (1978)
31. Howard, D., Angus, J.: Acoustics and Psychoacoustics. Focal Press (2009)
32. IEEE: IEEE standard for floating-point arithmetic. IEEE Std 754-2008 pp. 1–70 (2008)
33. Jaffe, D.A.: Spectrum analysis tutorial, part 1: The discrete Fourier transform. Computer Music Journal **11**(2), 9–24 (1987)
34. Jaffe, D.A.: Spectrum analysis tutorial, part 2: Properties and applications of the discrete Fourier transform. Computer Music Journal **11**(3), 17–35 (1987)
35. Jayant, N.S., Noll, P.: Digital Coding of Waveforms: Principles and Applications to Speech and Video. Prentice Hall Professional Technical Reference, Englewood Cliffs, NJ (1990)
36. Keller, D., Lazzarini, V., Pimenta, M.S. (eds.): Ubiquitous Music. Computational Music Science. Springer (2014)
37. Kleimola, J., Lazzarini, V., Timoney, J., Valimaki, V.: Vector phase shaping synthesis. In: Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11), pp. 223–240. Paris, France (2011)
38. Kleimola, J., Lazzarini, V., Vämäki, V., Timoney, J.: Feedback amplitude modulation synthesis. EURASIP Journal Adv. Sig. Proceedings **2011:434378** (2011)
39. Kushner, D.: The making of Arduino. IEEE Spectrum URL http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino/
40. Lazzarini, V.: Introduction to digital audio signals. In: R. Boulanger, V. Lazzarini (eds.) The Audio Programming Book, pp. 431–462. MIT Press, Cambridge, MA (2010)
41. Lazzarini, V.: Programming the phase vocoder. In: R. Boulanger, V. Lazzarini (eds.) The Audio Programming Book, pp. 557–580. MIT Press, Cambridge, MA (2010)
42. Lazzarini, V.: Spectral audio programming basics: The DFT, the FFT, and convolution. In: R. Boulanger, V. Lazzarini (eds.) The Audio Programming Book, pp. 521–538. MIT Press, Cambridge, MA (2010)
43. Lazzarini, V.: The STFT and spectral processing. In: R. Boulanger, V. Lazzarini (eds.) The Audio Programming Book, pp. 539–556. MIT Press, Cambridge, MA (2010)
44. Lazzarini, V.: Time-domain audio programming. In: R. Boulanger, V. Lazzarini (eds.) The Audio Programming Book, pp. 463–520. MIT Press, Cambridge, MA (2010)
45. Lazzarini, V.: The development of computer music programming systems. Journal of New Music Research **42**(1), 97–110 (2013)
46. Lazzarini, V., Costello, E., Yi, S., ffitch, J.: Development tools for ubiquitous music on the World Wide Web. In: Ubiquitous Music, pp. 111–128. Springer (2014)
47. Lazzarini, V., J., T., Pekonen, J., Valimaki, V.: Adaptive phase distortion synthesis. In: Proceedings of the 12th International Conference on Digital Audio Effects, pp. 28–35. Milan Institute of Technology, Como, Italy (2009)

48. Lazzarini, V., Keller, D., Pimenta, M.S., Timoney, J.: Ubiquitous music ecosystems: Faust programs in Csound. In: Ubiquitous Music, pp. 129–150. Springer (2014)
49. Lazzarini, V., Kleimola, J., Timoney, J., Valimaki, V.: Five variations on a feedback theme. In: Proceedings of the 12th International Conference on Digital Audio Effects, pp. 139–145. Milan Institute of Technology, Como, Italy (2009)
50. Lazzarini, V., Kleimola, J., Timoney, J., Valimaki, V.: Aspects of second-order feedback AM synthesis. In: Proceedings of the International Computer Music Conference, pp. 92–98 (2011)
51. Lazzarini, V., Timoney, J.: New methods of formant analysis-synthesis for musical applications. In: Proceedings International. Computer Music Conference, pp. 239–242. Montreal, Canada (2009)
52. Lazzarini, V., Timoney, J.: New perspectives on distortion synthesis for virtual analog oscillators. Computer Music Journal **34**(1), 28–40 (2010)
53. Lazzarini, V., Timoney, J.: Theory and practice of modified frequency modulation synthesis. Journal of the Audio Engineering Society **58**(6), 459–471 (2010)
54. Lazzarini, V., Timoney, J.: Synthesis of resonance by nonlinear distortion methods. Computer Music Journal **37**(1), 35–43 (2013)
55. Lazzarini, V., Timoney, J., Lysaght, T.: Spectral processing in Csound 5. In: Proceedings of International Computer Music Conference, pp. 102–105. New Orleans, USA (2006)
56. Lazzarini, V., Timoney, J., Lysaght, T.: Asymmetric-spectra methods for adaptive FM synthesis. In: Proceedings of the 11th International Conference on Digital Audio Effects (DAFx-11), pp. 42–49. Espoo, Finland (2008)
57. Lazzarini, V., Timoney, J., Lysaght, T.: The generation of natural-synthetic spectra by means of adaptive frequency modulation. Computer Music Journal **32**(2), 9–22 (2008)
58. Lazzarini, V., Timoney, J., Lysaght, T.: Nonlinear distortion synthesis using the split-sideband method, with applications to adaptive signal processing. Journal of the Audio Engineering Society **56**(9), 684–695 (2008)
59. Lazzarini, V., Yi, S., ffitch, J., Heintz, J., Brandtsegg, Ø., McCurdy, I.: Csound: A Sound and Music Computing System. Springer, Berlin (2016)
60. Lazzarini, V., Yi, S., Timoney, J.: Web audio: Some critical considerations. In: Sixth Workshop on Ubiquitous Music, pp. 1–12. Linnaeus University, Växjö, Sweden (2015)
61. Le Brun, M.: Digital waveshaping synthesis. Journal of the Audio Engineering Society **27**(4), 250–266 (1979)
62. Loy, G.: The CARL system: Premises, history and fate. Computer Music Journal **26**(4), 23–54 (2002)
63. Mathews, M.: An acoustical compiler for music and psychological stimuli. Bell System Technical Journal **40**(3), 553–557 (1961)
64. Mathews, M., Miller, J.E.: MUSIC IV Programmer's Manual. Bell Telephone Lab, Murray Hill, N.J. (1964)
65. Mathews, M., Miller, J.E., Moore, F.R., Pierce, J.R.: The Technology of Computer Music. MIT Press, Cambridge, MA (1969)
66. McAulay, R., Quatieri, T.: Speech analysis/synthesis based on a sinusoidal representation. IEEE Transactions on Acoustics, Speech and Signal Processing **34**(4), 744–754 (1986)
67. MIDI Manufacturers Association: Midi 1.0 specification (1983). URL http://www.midi.org
68. Moore, F.R.: Elements of Computer Music. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1990)
69. Moorer, J.: Signal processing aspects of computer music: A survey. Proceedings of the IEEE **65**(8), 1108–1137 (1977)
70. Moorer, J.A.: The synthesis of complex audio spectra by means of discrete summation formulas. Journal of the Audio Engineering Soc **24**(9), 717–727 (1976)
71. Moorer, J.A.: The use of the phase vocoder in computer music applications. Journal of the Audio Engineering Society **26**(1/2), 42–45 (1978)
72. Mulgrew, B., Grant, P., Thompson, J.: Digital Signal Processing: Concepts and Applications. Macmillan Press, London (1999)

73. Nielsen, J.: Noncommand user interfaces. Communications ACM **36**(4), 83–99 (1993)
74. Noble, J., Joshua, N.: Programming Interactivity: A Designer's Guide to Processing, Arduino, and Openframeworks, 1st edn. O'Reilly Media, Inc. (2009)
75. Nyquist, H.: Certain topics in telegraph transmission theory. Transactions of the AIEE **47**, 617–644 (1928)
76. Oppenheim, A.V., Schafer, R.W., Buck, J.R.: Discrete-time Signal Processing (2nd Ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1999)
77. Orlarey, Y., Fober, D., Letz., S.: Faust: An efficient functional approach to DSP programming. In: G. Assayag, A. Gerszo (eds.) New Computational Paradigms for Computer Music. Edition Delatour (2009)
78. Ousterhout, J.: Scripting: higher-level programming for the 21st century. IEEE Computer **31**(3), 23–30 (1998)
79. Palamin, J.P., Palamin, P., Ronveaux, A.: A method of generating and controlling musical asymmetrical spectra. Journal of the Audio Engineering Society **36**(9), 671–685 (1988)
80. Pampin, J.: ATS: A system for sound analysis transformation and synthesis based on a sinusoidal plus crtitical-band noise model and psychoacoustics. In: Proceedings of the International Computer Music Conference, pp. 402–405. Miami, FL (2004)
81. Parallax Inc.: The original BASIC stamp microcontroller (2016). URL https://www.parallax.com/microcontrollers/basic-stamp
82. Phillips, D.: Linux Music & Sound. No Starch Press, San Francisco, CA (2000)
83. Poepel, C., Dannenberg, R.: Audio signal driven sound synthesis. In: Proceedings of the 2005 International Computer Music Conference, pp. 391–394. Barcelona, Spain
84. Puckette, M.: Formant-based audio synthesis using nonlinear distortion. Journal of the Audio Engineering Society **43**(1/2), 40–47 (1995)
85. Puckette, M.: Phase-locked vocoder. In: IEEE ASSP Workshop onApplications of Signal Processing to Audio and Acoustics, pp. 222–225 (1995)
86. Risset, J.C.: An Introductory Catalogue of Computer Synthesized Sounds. Bell Telephone Lab, Murray Hill, N.J. (1969)
87. Roads, C.: Microsound. MIT Press, Cambridge, MA (2001)
88. Roads, C., Mathews, M.: Interview with Tongues. Computer Music Journal **4**(4), pp. 15–22 (1980)
89. Rocher, M.: Introduction to the theory of Fourier's series. Annals of Mathematics **7**(3), 81–152 (1906)
90. Rodet, X.: Time domain formant-wave-function synthesis. Computer Music Journal **8**(3), 9–14 (1984)
91. Roebel, A.: Efficient spectral envelope estimation and its application to pitch shifting and envelope preservation. In: Proceedings of the 8th International Conference on Digital Audio Effects, pp. 30–35 (2005)
92. Schottstaedt, B.: The simulation of natural instrument tones using frequency modulation with a complex modulating wave. Computer Music Journal **1**(4), 46–50
93. Serra, X., Smith, J.: Spectral modeling synthesis: A sound analysis/synthesis based on a deterministic plus stochastic decomposition. Computer Music Journal **14**, 12–24 (1990)
94. Shannon, C.E.: Communication in the presence of noise. Proceedings Institute of Radio Engineers **37**(1), 10–21 (1949)
95. Shneiderman, B.: Designing the User Interface: Strategies for Effective Human-computer Interaction. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1986)
96. Smalley, D.: Spectro-morphology and structuring processes. In: S. Emmerson (ed.) The Language of Electroacoustic Music, pp. 61–96. Macmillan Press, London (1986)
97. Smith, J., Serra, X.: PARSHL: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation. In: International Computer Music Conference, pp. 290–297. Urbana, Illinois, USA (1987)
98. Steiglitz, K.: A Digital Signal Processing Primer, with Applications to Digital Audio and Computer Music. Addison-Wesley Longman, Redwood City (1996)

99. Stockham Jr., T.G.: High-speed convolution and correlation. In: Proceedings of the April 26-28, 1966, Spring Joint Computer Conference, AFIPS '66, pp. 229–233. ACM, New York (1966)
100. Tenney, J.: Sound generation by means of a digital computer. Journal of Music Theory **7**(1), 24–70 (1963)
101. Timoney, J., Pekonen, J., Lazzarini, V., Välimäki, V.: Dynamic signal phase distortion using coefficient-modulated allpass filters. Journal of the Audio Engineering Society **62**(9), 596–610 (2014)
102. Tomisawa, N.: Tone production method for an electronic musical instrument (1981). US Patent 4,249,447
103. Torger, A., Farina, A.: Real-time partitioned convolution for ambiophonics surround sound. In: 2001 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. New Paltz, New York (2001)
104. Widrow, B., Kollár, I.: Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control, and Communications. Cambridge University Press, Cambridge, UK (2008)
105. Windham, G., Steiglitz, K.: Input generators for digital sound synthesis. Journal of the Acoustic Society of America **47**(2), 665–6
106. Wishart, T.: Audible Design. Orpheus the Pantomime, York (1994)
107. Wood, A.: The Physics of Music. Chapman and Hall, London (1975)
108. Wright, M., Freed, A.: Open Sound Control: A new protocol for communicating with sound synthesizers. In: Proceedings of the ICMC, pp. 101–104. Thessaloniki, Greece (1997)
109. Wright, M., Freed, A., Momeni, A.: Open Sound Control, state of the art 2003. In: Proceedings of the 2003 Conference on New Interfaces for Musical Expression (NIME-03), pp. 153–159. Montreal, Canada (2003)

Index

V. Lazzarini, *Computer Music Instruments*,
https://doi.org/10.1007/978-3-319-63504-0